Università degli Studi di Udine
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Dottorato di Ricerca in Informatica e Scienze Matematiche e Fisiche

Ph.D. Thesis

Temporal Information in Data Science: An Integrated Framework and its Applications

Candidate:
Andrea Brunello

Supervisor:
Prof. Angelo Montanari

Co-Supervisors:
Enrico Marzano
Prof. Guido Sciavicco

Year 2020

Author's e-mail: andrea.brunello@uniud.it

Author's address:

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine
Italia

And you may ask yourself, well,
"How did I get here?"
And you may ask yourself,
"How do I work this?"
And you may ask yourself,
"Am I right? Am I wrong?"
And you may say yourself,
"My God! What have I done?"

– *Talking Heads*

Abstract

Data science is a well-known buzzword, that is in fact composed of two distinct keywords, i.e., *data* and *science*. *Data* itself is of great importance: each analysis task begins from a set of examples. Based on such a consideration, the present work starts with the analysis of a real case scenario, by considering the development of a data warehouse-based decision support system for an Italian contact center company. Then, relying on the information collected in the developed system, a set of machine learning-based analysis tasks have been developed to answer specific business questions, such as employee work anomaly detection and automatic call classification. Although such initial applications rely on already available algorithms, as we shall see, some clever analysis workflows had also to be developed. Afterwards, continuously driven by real data and real world applications, we turned ourselves to the question of how to handle temporal information within classical decision tree models. Our research brought us the development of J48SS, a decision tree induction algorithm based on Quinlan's C4.5 learner, which is capable of dealing with temporal (e.g., sequential and time series) as well as atemporal (such as numerical and categorical) data during the same execution cycle. The decision tree has been applied into some real world analysis tasks, proving its worthiness. A key characteristic of J48SS is its interpretability, an aspect that we specifically addressed through the study of an evolutionary-based decision tree pruning technique. Next, since a lot of work concerning the management of temporal information has already been done in automated reasoning and formal verification fields, a natural direction in which to proceed was that of investigating how such solutions may be combined with machine learning, following two main tracks. First, we show, through the development of an enriched decision tree capable of encoding temporal information by means of interval temporal logic formulas, how a machine learning algorithm can successfully exploit temporal logic to perform data analysis. Then, we focus on the opposite direction, i.e., that of employing machine learning techniques to generate temporal logic formulas, considering a natural language processing scenario. Finally, as a conclusive development, the architecture of a system is proposed, in which formal methods and machine learning techniques are seamlessly combined to perform anomaly detection and predictive maintenance tasks. Such an integration represents an original, thrilling research direction that may open up new ways of dealing with complex, real-world problems.

Acknowledgments

Undertaking this PhD has been a truly life-changing and enriching experience for me and it would not have been possible to do without the support that I received from many people.

First and foremost, I would like to thank my two academic supervisors, Prof. Angelo Montanari from the University of Udine and Prof. Guido Sciavicco from the University of Ferrara. Their passionate participation and input helped me a great deal in this project and during the writing of this thesis.

In the same way, I would also like to thank my "business" supervisor, M.Sc. Enrico Marzano, from Gap Srlu. His strategic vision, expertise and constant feedback provided a fundamental contribution to the success of my PhD.

Many thanks to M.Sc. Andrea Della Pietà Casagrande, also from Gap Srlu, who helped me with the experiments and some of the more technical stuff at the company.

I gratefully acknowledge the hospitality given to me by Prof. Mark Reynolds, from the University of Western Australia, during my four month abroad research period in Perth.

I will always fondly remember the two years spent at the University with my office-mate, Dr. Andrea Viel, who has also been my flatmate for a year. We had a great time both in the office and during our academic trips.

In the end, I am indebted to all my family, for they gave me enough moral support and motivation to accomplish my goals, and to my friends, for their advice and encouragement.

Contents

List of Figures

List of Tables

Introduction

Data science is a broad term. In [95], it is defined as a multi-disciplinary field that relies on scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data. Another way of looking to it is to consider that *data science* has in fact two distinct keywords, that are, *data* and *science* [10].

Data is of the uttermost importance. Either if supervised or unsupervised learning is used, or a simple exploration is performed, all starts from a set of examples. Based on this consideration, the thesis starts by considering a real business scenario, in which an enterprise-wide, event-based data warehouse is built for an Italian multi-skill, multi-channel contact center company [61]. As we shall see, the warehouse has become the fulcrum of the enterprise's whole IT infrastructure, also thanks to the development of a *business intelligence* layer capable of supporting OLTP (On-Line Transaction Processing) as well as OLAP (On-Line Analytical Processing) queries; the latter, in turn, allowed for the development of utilities such as daily reports, real-time call flow tracking and employee performance monitoring, giving rise to a fully-fledged *decision support system*.

Then, there is *science*: data science is only useful when the data are used to answer a question [10]. Empowered by the business intelligence layer, a set of machine learning-based analysis tasks have been developed over the warehouse, including employee work anomaly detection and call classification [68]. Such initial applications relied on already available algorithms, even if some clever analysis workflows had to be developed, as in the case of the feature selection experiment described in Section 3.1, and published as a standalone paper in [63]. Observe that feature selection plays a major role in those applications where data and model readability are of paramount importance, which can be the case, for example, of a business setting, where a domain expert may want to read and validate a model before putting it into production. We specifically addressed such an interpretability requirement by investigating on the possibility of employing multi-objective evolutionary optimization techniques to the task of decision tree pruning [65], so to achieve a better trade-off between model size and classification performance.

The data modelling and preparation steps performed through the data warehouse development, together with these first research work helped us to understand the importance of relying on real data, and working on real world applications. On the basis of this wisdom, and always taking such considerations into account, we then focused on the main topic of this thesis, i.e., how to handle temporal information in data mining tasks. In fact, temporal data plays an important role in the extraction of information in many applications, and it comes in at least two flavours:

it can be represented either by a discrete sequence of finite-domain values, e.g., a sequence of purchases, as well as by a real-valued time series (for instance, think of a stock price history). Sometimes, temporal information is complemented by other, "static" kinds of data, which can be numerical or categorical. As an example, this is the case with the medical history of a patient, which may include: (*i*) categorical attributes, with information about the gender or the smoking habits; (*ii*) numerical attributes, with information about the age and weight; (*iii*) time series data, tracking the blood pressure over several days, and (*iv*) discrete sequences, describing which symptoms the patient has experienced and which medications have been administered to her/him. Such a heterogeneous set of information pieces may be useful, among others, for classification purposes, such as trying to determine the disease affecting the patient. Another use case is that of phone call classification in contact centers [61]: a conversation may be characterized by sequential data (for instance, textual data obtained from the call recordings), one or more time series (keeping track of the volume over time), and a set of categorical or numerical attributes (reporting, e.g., information pertaining to the speakers, or the kind of call). Unfortunately, different kinds of data typically require different kinds of preprocessing techniques and classification algorithms to be managed properly, which usually means that the heterogeneity of data and the complexity of the related analysis tasks are directly proportional. Moreover, since multiple algorithms have to be combined to produce a final classification, the final model may lack in interpretability. Again, this is a fundamental problem in domains where understanding and validating the classification process is as important as the accuracy of the classification itself, e.g., production business systems and life critical medical applications.

Thus, we turned ourselves to the development of J48S (later further extended into J48SS) [66, 67, 69] which, as we shall see, is a decision tree induction algorithm based on WEKA's J48 (a Java implementation of C4.5 [274]). The algorithm is capable of naturally exploiting categorical and numerical attributes as well as sequential and time series data during the same execution cycle. Moreover, the resulting decision tree models are intuitively interpretable, meaning that a domain expert may easily read and validate them. J48SS has been applied to two distinct real world tasks, proving its worthiness within a phone call analysis framework (given that a phrase can be considered as a sequence of words) [66], and in a pollution analysis task [64].

Since a lot of work on the management of temporal information has been done in automated reasoning and formal verification, a natural direction in which to proceed with the research was then that of investigating how such solutions can be combined with machine learning algorithms. In the last part of the thesis, we start such an exploration following two main directions. First, we show how a machine learning algorithm can successfully exploit temporal logic to perform data analysis. More precisely, we present an enrichment of decision tree models with temporal information, encoded by interval temporal logic formulas [64]. Then, we focus on the opposite direction, i.e., that of employing machine learning techniques to generate temporal logic formulas. We consider the task of natural language English utter-

ance formalization: we give an overview of the state of the art, we make a critical experimental evaluation of existing solutions, and we outline some promising directions for future research [70]. Finally, as a conclusive development, we propose the architecture of a system, in which formal methods and machine learning techniques are seamlessly combined to perform anomaly detection and predictive maintenance tasks [60].

The latter topic represents, at the same time, the culmination of the entire research path, and the starting point for a new journey. We started from some basic data modelling and analysis tasks, and this led us to consider temporal information under different aspects, and in ever increasing complexity. The final outcome of our work is an integrated framework that allows us to deal with both temporal as well as different kinds of atemporal information. We believe that, as it is emerging from the most recent literature in the field, a new, thrilling research direction is that of combining machine and statistical learning solutions with logics and formal methods techniques, to deal with complex real-world problems [172], that can be characterized by several kinds of sequential and temporal information.

The thesis is structured as follows. Chapter 1 provides some background knowledge on which the main topics discussed in the thesis are based, including a focus on learning with sequential and time series data, dimensionality reduction via feature selection, decision tree models and their ensembles, and multi-objective evolutionary computation. Chapter 2 presents the development of a data warehouse-based decision support system for Gap Srlu company, an Italian business process outsourcer focused on contact center services. Chapter 3 discusses an advanced analysis task developed over the warehouse, that deals with classical (categorical and numerical) data: a methodology is presented, that makes use of feature selection and fuzzy classification to predict whether an inbound call will or will not be served on time by the contact center staff. Chapter 4 describes an approach to decision tree pruning based on multi-objective evolutionary computation, that allows the user to easily choose the most appropriate trade-off between accuracy and readability of the generated models. Chapter 5 presents J48S, a first extension to J48 which is capable of handling sequential data; the algorithm is then applied within a phone call analysis framework inside Gap Srlu company. Chapter 6 focuses on time series data, and presents J48SS, a further enhancement of J48S which is also capable of dealing with time series attributes for classification purposes; the model is profitably applied to a pollution analysis task on a heterogeneous dataset collected in the city of Wrocław, Poland. Chapter 7 deals with the combination of machine learning and temporal logic: Temporal ID3, a decision tree capable of extracting interval temporal logic formulas from raw temporal data is presented; then, a review of the state of the art concerning the formalization of English utterances by means of machine learning and temporal logic is discussed, along with some possible research paths. Also, a system is proposed, that combines formal methods and machine learning techniques to perform failure detection and predictive maintenance tasks. Finally, we provide an overall assessment of the work done, together with future research directions.

1
Background

This chapter gives an overview of the main topics discussed in the thesis. Specifically, Section 1.1 is devoted to data mining and machine learning, and includes a focus on learning with sequential and time series data, and dimensionality reduction through feature selection. Then, decision trees are presented, both with respect to single (Section 1.2) and ensembles (Section 1.3) of models. Finally, in Section 1.4 evolutionary algorithms are discussed, with a special attention to NSGA-II and ENORA, two multi-objective optimization algorithms that we relied on in our work (see, e.g., Sections 3.1 and 6.1).

1.1 Data mining and machine learning

If we consider *data* to be a raw collection of known facts, then we may define *information* as a set of concepts, regularities, and patterns that are buried in them. *Data mining* is the process of extracting relevant and previously unknown information from (big) quantities of data [331]. The discovered information may then be encoded in suitable *models* (e.g., decision trees, see Section 1.2), so that data mining can also be referred to as a process of *abstraction*. Such models are typically built by means of *machine learning* algorithms, that automatically or semi-automatically delve into data, providing a "technical" foundation to the whole process. Learning may be divided into two broad categories, i.e., *supervised* and *usupervised* learning.

In a supervised learning task, an algorithm is given a training dataset in which each instance is characterized by one or more input variables x (predictors, or features) and an output variable Y (label). The goal is that of inferring a function f, capable of mapping the input into the output: $Y = f(x)$. Then, based on the kind of label, a supervised learning task may be considered to be a *regression* (in which Y is numerical) or a *classification* (in which Y is categorical) task.

On the other hand, unsupervised learning does not make use of labels. An algorithm is given a dataset of instances, each characterized by one or more predictors. The goal is that of modelling the underlying structure or identify general patterns in the data, so as to discover useful information. Examples of unsupervised learning are given by *clustering* and *association rule learning* tasks.

Most machine learning applications deal with static, atemporal data, some examples of which are presented in Section 2.6 and Chapter 3. Nevertheless, the temporal aspect plays an important role in the extraction of information in many domains, and it comes in at least two flavours: it can be represented either by a discrete sequence of finite-domain values (e.g., a sequence of purchases), as well as by a real-valued time series (think for example about a stock price history). Thus, Section 1.1.1 introduces the task of learning on sequential data, whereas Section 1.1.2 presents an overview on the analysis of time series, specifically focused on the task of classification by means of a particular kind of patterns, called *shapelets*.

Regardless of the type of learning, the data mining process always begins with the data collection phase, followed by its preparation, that typically includes tasks such as data cleaning and normalization, missing values imputation, feature engineering and dimensionality reduction. The latter aspect is particularly important when dealing with high-dimensional datasets, as it allows to reduce the number of features, which may lead to smaller and more accurate models being trained. Among the different dimensionality reduction techniques, this thesis presents some original feature selection approaches (see Chapter 3). For this reason, the present background chapter also includes an overview of feature selection, presented in Section 1.1.3.

1.1.1 Learning on sequential data: sequential pattern mining

Sequences arise naturally in many domains: let us think for example about tracking all the purchases made by a customer on an online shopping website, or even consider natural language textual data, which is essentially a list of words; also, they may be the result of a time series discretization process (see Section 1.1.2).

Sequential pattern mining is a popular data mining task aimed at discovering "interesting" patterns in sequences [123, 224], that is, patterns that can prove to be useful for supervised or unsupervised learning tasks.

Following the original definition given by [26], the problem of sequential pattern mining can be described as follows. Consider a database D of customer transactions, where each entry is characterized by the fields *Customer-id*, *Transaction-time*, and the kinds of *items* that the customer bought in the specific transaction. Then, an *itemset* is a non-empty set of items, and a *sequence* can be considered as an ordered list of itemsets. Given the information contained in database D, we may easily construct the sequence of purchases for each customer, as shown in Tables 1.1 and 1.2. Then, a possible pattern that may be extracted from the first sequence in Table 1.2 is, for instance, $\{A\}, \{C\}, \{A, B\}$.

Finally, we may intuitively define the problem of sequential pattern mining as the one of finding interesting patterns made by lists of (sub)itemsets, i.e., concatenations of symbols taken from the original sequences that are useful for the specific task at

Table 1.1: An exemplary database of customer transactions.

Customer-id	Transaction-time	Items bought
1	2018-01-01	{A, B, C}
1	2018-01-15	{A, C}
1	2018-02-20	{A, B}
1	2018-02-22	{C, D}
2	2018-01-23	{A, C}
2	2018-02-18	{A, C, D}
2	2018-03-07	{D}

Table 1.2: Sequences corresponding to the customer transactions.

Customer-id	Associated sequence
1	{A, B, C}, {A, C}, {A, B}, {C, D}
2	{A, C}, {A, C, D}, {D}

hand. Such patterns may also be simple concatenations of items, as in the case of patterns extracted from textual data, that are made of words.

The work presented in Chapter 5 deals with the extraction of *frequent patterns*, which are concatenations of symbols that frequently appear in a set of sequences, with a frequency that is no less than a user-specified *minimum support threshold* [26]. Several sequential pattern mining algorithms have been proposed over the years, such as *PrefixSpan* [264], *SPADE* [338], and *SPAM* [40]. However, a drawback of such algorithms is that they typically generate a large amount of patterns, which may be redundant. This is problematic for two reasons: *(i)* it makes it difficult for a user to gain any insight based on the generated output, and also to exploit the patterns for further data mining tasks, and *(ii)* very large sets of patterns may imply that many resources are consumed and an extremely high computation time is required in the generation process. In order to reduce the computational burden of the mining tasks, and also to present a reduced set of patterns to the user, some concise representations of frequent sequential patterns have been designed. *A frequent closed sequential pattern* is a frequent sequential pattern such that it is not included in any other sequential pattern having exactly the same frequency, that is, it is "maximal". Algorithms that have been developed to extract such patterns include *CloSpan* [336], *BIDE* [325], *ClaSP* [143], and *CM-ClaSP* [121]. Another possible solution is given by *sequential generator patterns*. Opposite to the concept of closed patterns, generators are "minimal", meaning that a pattern is considered to be a generator if it does not exist any pattern which is contained in it and has the same frequency. According to the *minimum description length* (MDL) principle [285], generators are preferable over closed patterns for model selection and classification, since they may be less

prone to overfitting effects [221]. There also exists a very recent solution capable of extracting patterns according to different concise representations during the same execution cycle [101].

In this thesis, we make use of the algorithm VGEN [122] (see Section 5.1), which, at the best of the our knowledge, represents the state-of-the-art in the extraction of general sequential generator patterns. VGEN operates in a top-down fashion, starting from the most general patterns down to more specific ones. A pattern is considered as a sequence of itemsets, which are in turn unordered sets of distinct items. An item is an element that may appear in a sequence belonging the considered dataset. Intuitively, items that belong to the same itemset are considered to occur at the same time. The algorithm starts by listing all single-itemset, single-item frequent patterns, and then it grows them by means of the so-called *s-extension* operations (in which a new itemset is enqueued to the pattern) and *i-extension* operations (in which a new item is added to the last itemset), ensuring also that only generator patterns are produced. Of course, as a pattern grows, its support (number of instances in the dataset that contain the pattern) decreases, and the growing phase continues until a minimum support threshold is reached, which may be specified by the user. The algorithm is also capable of extracting non strictly contiguous patterns, by specifying a maximum gap tolerance between the itemsets.

1.1.2 Learning on time series data: shapelet based time series classification

As we have already discussed, temporal data plays a major role in many fields, ranging from healthcare [234], to weather prediction [232], and financial data analysis [313]. Temporal data mining is a specific instance of data mining that deals with the extraction of useful information from temporal data, which is often represented by time series, that in turn basically consist of a sequence of real values sampled at regular time intervals [238].

Typical time series-based tasks include clustering [212] and forecasting [239], although this thesis mainly focuses on time series-based classification, which is recognized as a difficult problem in the literature [164].

A commonly adopted solution to classification involves summarizing the content of a time series (or parts of it, for instance through *windowing* [238]) by means of statistical measures such as mean, variance, maximum, minimum, and so on. The resulting dataset may then be processed by classical learning algorithms, such as decision trees or support vector machines. Despite the simplicity of the approach, it has a clear downside, i.e., the loss of any explicitly represented temporal information.

Another possible strategy is that of performing a preliminary data discretization step by means of approaches such as *Equal Width* or *Equal Frequency* binning [238], or more advanced strategies like *Symbolic Aggregate Approximation* (SAX) [214] or *Persist* [240]. This step allows one to build a symbolic representation of the

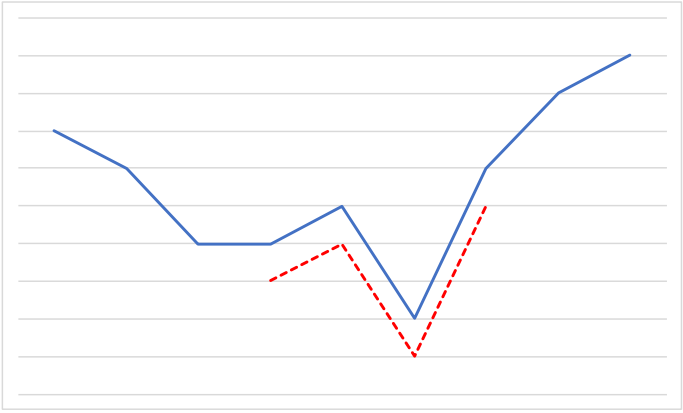


Figure 1.1: A time series (blue solid line) and a possible shapelet (red dotted line). The two lines are intentionally shifted for the ease of comprehension.

time series, that can be then analyzed with traditional pattern mining algorithms [224]. Such an approach has been followed in many studies, e.g., in [240, 244, 340]. However, also the discretization phase typically involves a loss in information, and may reduce the interpretability of the results.

Alternative approaches rely on patterns that can be generated directly from the numerical data, such as *time series shapelets*. Shapelets have been originally presented in [337] as contiguous time series subsequences which are in some sense maximally representative of a class. The overall idea is that shapelets are meant to capture local features of the time series, which should be more resistant to noise and distortions than global characteristics. Figure 1.1 shows an example of a shapelet, together with a time series. Since a shapelet is defined as an arbitrary-length contiguous subsequence of a training set time series, we may determine the total number of shapelets that can be obtained from a dataset. Let us consider n instances, and define as l_i the length of the time series of the ith instance. Then, the total number of shapelets that may be generated is $\sum_{i=1}^{n}(l_i!)$. Given such an enormous space, an exhaustive search for the best shapelets is unfeasible and, indeed, several contributions have concentrated on the problem of speeding up the shapelet extraction process [148, 149, 162, 192, 280, 284, 330], following, for instance, heuristic search, or random sampling of the shapelet candidates. In the present thesis, an evolutionary algorithm-based solution to the problem of shapelet generation is presented in Section 6.1.

Typically, shapelets are used in the following way. Given a training dataset made of time series, k shapelets are extracted through a suitable method. As a result, each of the instances is represented by k predictor attributes: each one of them encodes the distance between the corresponding shapelet and the instance's time series. For its simplicity, the *Euclidean distance* is the most used distance metric: the final value corresponds to the minimum distance between the time series and the

shapelet, calculated with a sliding window approach. Measures based on *Dynamic Time Warping* [295] or *Mahalanobis distance* [35] have also been investigated in the literature.

1.1.3 The feature selection process

Feature selection is the process of removing attributes from the data set that are irrelevant to task to be performed [216]. Its main aim is to facilitate data understanding, and to reduce storage and computation time requirements for model learning, while retaining a suitably high accuracy in representing the original information. Feature selection algorithms may be classified into several categories, depending on the specific criterion under consideration. According to whether the training set is composed of labelled instances or not, the selection may be, respectively, *supervised* or *unsupervised*. Methods in the former category seek for correlations between attribute and class label values, whereas those in the latter employ (usually, descriptive) statistical tests over attributes, such as, for example, a near-zero-variance test. Feature selection methods consist of four steps, namely *subset generation*, *subset evaluation*, *stopping criterion*, and *result validation*. The design of such steps entails the selection of: *(i)* a target to which to apply the procedure; *(ii)* a search strategy, to guide the incremental generation of the feature set; *(iii)* an evaluation strategy, which depends on the target type and, in the case of supervised methodologies, may imply choosing an actual classifier; *(iv)* an evaluation metric used to score the candidates.

Subset generation. Subset generation methods (also referred to as *search strategies*) are used to guide the iterative generation of the feature set, in the space of all the possible combinations of features. They can be categorized into *deterministic* and *probabilistic* methodologies, the former giving back the same set of attributes if repeatedly executed (such as WEKA's *BestFirst* [263]), and the latter taking non-deterministic choices during execution (as in the case of WEKA's *GeneticSearch* [141]). Moreover, in the former category it is possible to distinguish strategies according to their search direction: *forward* search strategies start with an empty attribute set, and then grow it; *backward* search strategies begin with an initial set consisting of all attributes, and proceed by discarding elements; *bidirectional* search strategies consider an initial point in the subset space, and then proceed in both directions; on the contrary, probabilistic (or *random*) strategies do not follow a predefined search direction, for example in optimization through genetic algorithms.

Subset evaluation. According to the target of the selection procedure, it is possible to classify evaluation strategies into *univariate* and *multivariate*. Strategies that belong to the former category evaluate attributes independently; as a result, they are computationally less demanding than those that belong to the latter, which consider subsets of attributes as a whole. Moreover, multivariate approaches can also

take into account complex relationships between features, such as redundancy. In Chapter 3 we take into consideration supervised methods, and, in particular, *filter* and *wrapper* models. Filter models are independent from the successive classifier learning phase, and are based only on general measures such as the correlation or consistency with the variable to predict. Filter techniques scale well with the size of data sets; however, since they ignore the classification performance, they might not always provide the best results [99, 265]. Wrapper models, on the other hand, evaluate the predictive accuracy of the attribute set with a selected classifier. These techniques typically offer better results than filters, at the cost of being computationally more demanding, and more prone to overfitting [217].

1.2 Decision trees

As it is commonly recognized, decision trees have still a very important position among classification models [331]. This is mainly due to the facts that (*i*) they can be trained and applied efficiently even on big datasets, and (*ii*) they are easily interpretable. Thanks to the latter feature, they turn out to be useful not only for prediction, but also for highlighting relevant patterns or regularities in the data. This is extremely beneficial in those application domains where understanding the classification process is at least as important as the accuracy of the prediction itself.

A typical decision tree is constructed recursively, starting from the root, following the traditional *top down induction of decision rees* (TDIDT) approach: at each node the attribute that best partitions the training data, according to a predefined score, is chosen as a test to guide the partitioning of instances into child nodes, and the process continues until a sufficiently high degree of purity (with respect to the target class), or a minimum cardinality constraint (with respect to the number of instances reaching the node), is achieved in the generated partitions. A decision tree induced by the TDIDT approach tends to *overgrow*, and this leads to a loss in interpretability as well as to a risk of *overfitting* training data, that results in capturing unwanted noise. As a direct consequence, such trees typically do not perform well on new, independent instances, since they fit the training data "too perfectly".

In order to simplify the tree structure, thus making the trees more general, *pruning* methods are typically applied. According to the time when such an operation is performed, we may distinguish among: (*i*) *pre-pruning*, consisting of interrupting the construction of the decision tree according to a stopping criterion such as minimum node cardinality, or when none of the attributes leads to a sufficiently high splitting score, and (*ii*) *post-pruning*, that is, building the entire tree first, and then removing or condensing some parts of it. While pre-pruning has the advantage of not requiring the construction of the whole tree, it usually leads to worse accuracy results than post-pruning [274]. There are two main strategies for evaluating the error rate in a post-pruning setting. The first one consists of keeping part of the

training data as an independent *hold out set* (and, thus, working on three, independent, datasets: training, hold out, and test), and deciding whether to prune a section of the tree or not on the basis of the resulting classification error on it. Examples of such techniques include a variant of CART's *Cost-Complexity Pruning* [56] and the so-called *Reduced-Error Pruning* [273]. It should be noticed that splitting data in three different partitions reduces the amount of labelled instances available for training, which, in some cases, are already scarce. The second strategy, on the contrary, focuses on estimating the *apparent* classification error due to pruning on the basis of the training data only, as in, for instance, *Pessimistic Error Pruning* [273] and C4.5's *Error-Based Pruning* [274].

From a computational point of view, it is known that the problem of constructing an optimal binary decision tree is NP-Complete [165]. The result is that all practical implementations of TDIDT algorithm and pruning methodologies are based on heuristics, that typically have a polynomial complexity in the number of instances and features in the training set.

The present thesis focuses on one of the most used and well known decision tree induction algorithms, i.e., C4.5 [274][1]. For this reason, the next section is devoted to a thorough description of such an algorithm, while Section 1.2.2 gives a short account of C5.0, an updated and commercial version of C4.5.

1.2.1 C4.5 decision tree learner

Construction of the tree To guide the splitting process in the decision tree building phase, J48/C4.5 adopts *information gain* (an entropy-based criterion) as the scoring strategy [273]. Given a set of observable values $V = (v_1, \ldots, v_n)$, with associated probabilities $P = (p_1, \ldots, p_n)$, the *information conveyed by* P, or the *entropy of* P, is defined as

$$E(P) = -\sum_{i=1}^{n} p_i * log_2(p_i) \ . \tag{1.1}$$

If a dataset T of instances is partitioned into disjoint exhaustive subsets $C_1, \ldots, C_k$ on the basis of the class value, then the information needed to identify the class of an element of T is

$$Info(T) = E(P), \text{ where } P = (|C_1|/|T|, |C_2|/|T|, \ldots, |C_k|/|T|) \ .$$

Intuitively, the purer T is with respect to the class values, the lower the entropy, and the easier to identify the class of an instance. Given T and a categorical attribute X of the dataset (non class), we may partition the instances into subsets $T_1, T_2, \ldots, T_j$

[1] Specifically, we make use of J48, the WEKA's Java implementation of C4.5 [125]. The terms J48 and C4.5 will be used interchangeably in the remainder of the thesis.

on the basis of the value they take on X. Then, the information needed to identify the class of an element of T becomes the weighted average of the information needed to identify the class of an element of T_i for each $i = 1, \ldots, j$, that is,

$$Info(X,T) = \sum_{i=1}^{j} (|T_i|/|T|) * Info(T_i) \ . \tag{1.2}$$

We may now define *information gain* as the difference between the information needed to identify an element of T and the information needed to identify such an element after the value it takes on X has been obtained:

$$Gain(X,T) = Info(T) - Info(X,T) \ . \tag{1.3}$$

Thus, the information gain represents precisely the gain in information due to attribute X. Starting from the root, at each node of the tree C4.5 chooses to branch on the attribute of the data which it considers best, namely the one with the highest *information gain ratio*, defined as

$$GainRatio(X,T) = \frac{Gain(X,T)}{SplitInfo(X,T)} \ , \tag{1.4}$$

where

$$SplitInfo(X,T) = E(|T_1|/|T|, \ldots, |T_j|/|T|) \tag{1.5}$$

is the *split information* metric, that takes into account the number and size of branches that would be generated if the split on attribute X is chosen. Introducing $SplitInfo$ corrects the bias that information gain has towards highly branching attributes (as an example, consider a dataset of people containing the attribute *social security number*, having unique values; a split on that attribute would produce single-instance subsets, leading to the highest possible *information gain*).

C4.5 also supports numerical attributes. In order to branch on a numerical attribute, the algorithm identifies a threshold and then splits the instances into those whose attribute value is *above* the threshold and those that are *less than or equal to* it [275]. This means that numerical splits are always binary, contrary to the categorical ones, which generate a number of children equal to the number of distinct values. Let us now consider the selection of a threshold for a numerical attribute A, belonging to a dataset D. If there are N distinct values of A in D, then there are $(N-1)$ thresholds that could be used *differently* for a binary test on A during the tree construction phase, since any threshold between two values will have the same effect in dividing the instances. Each threshold gives unique subsets D_1 and D_2, so in this case the value of the gain ratio is not only a function of the attribute, but also of the threshold. Thus, a straightforward extension to the categorical approach is to evaluate the ratio for each of the $(N-1)$ split points,

choosing the one that leads to the highest value. The gain ratio value for that particular split point is considered to be the gain ratio value for the attribute. To overcome the overfitting problems of this approach, an *MDL-based adjustment* [285] to the information gain for splits on numerical attributes was included in [275]: if there are K candidate splits on a certain numerical attribute at the node currently considered for splitting, $log_2(K)/M$ is subtracted from the information gain, where M is the number of instances at the node. Observe that the resulting value may be negative after the subtraction operation, and tree growing will stop if there are no attributes left with a positive information gain.

Pruning of the tree After the tree growing phase, two default, independent, methodologies are typically employed together to reduce the size of the generated model, that are, *Collapsing* and *Error-Based Pruning* (EBP).

Collapsing a tree can be seen as a special case of pruning, in which parts of the tree that do not improve the classification error on the training data are discarded. For example, given a node N that roots a subtree in which all leaves predict the same class C, the entire subtree can be collapsed into the node N that becomes a leaf predicting C.

EBP is based on a different idea. Since the decision tree error rate on the training set is biased by the learning phase, it does not provide a suitable estimate for the classification performance on future cases. Intuitively, EBP consists of systematically evaluating each node N and deciding, using statistical confidence estimates, whether to prune the subtree rooted in N or not. Since pruning always worsens the accuracy of the tree on the training dataset (or leaves it unchanged), EBP evaluates the effect of a pruning by trying to correct the bias, that is, estimating the true error that would be observed on independent data.

In essence, given a node covering n training instances, e of which misclassified, the *observed error rate* $f = e/n$ is calculated; then, the method tries to estimate the *true error rate* p that would be observed over the entire population of instances ever reaching that node, based on several additional hypothesis, among which assuming a *binomial distribution* for the error.

This solution gives rise to a simple method to control the pruning aggressiveness consisting in suitably varying the binomial *confidence intervals* [152], but, at the same time, it has been criticized for the lack of a proper statistical foundation. Also, it has been observed to have a tendency for under-pruning, especially on large datasets [110].

1.2.2 C5.0 decision tree learner

C5.0 (also known as *See5*) is an updated, commercial version of C4.5, reported to be much more efficient than its predecessor in terms of memory usage and computation time [287]. Moreover, the resulting trees tend to be smaller and more

accurate than those generated by C4.5 [7]. The learning algorithm follows a similar TDIDT strategy as its predecessor, relying on information gain and gain ratio scores to partition the training instances. The pruning is based on an EBP-like strategy, complemented by an optional *global pruning* step. Other important characteristics of C5.0 include the possibility of generating an ensemble of trees through *boosting*, the integration of an attribute selection strategy, called *winnowing*, the support for *asymmetric costs* for different kinds of error, *soft-thresholds* for numerical attributes, splitting on *value subsets* for discrete attributes, and a *multi-threaded* architecture [3]. A single-threaded version of C5.0 is available under the GNU GPL (`https://www.rulequest.com/download.html`).

1.3 Decision tree ensembles

Section 1.2 discussed decision trees, remarking upon the fact that they are easily interpretable. However, a drawback of decision trees is that they are usually less accurate in their predictions than other methodologies, and have a tendency to overfit training data [157]. A possible way to improve their accuracy, at the expense of a loss in the interpretability of the model and of a higher complexity in the training and prediction phases, is to build a set of different trees (ensemble), and then combine the single predictions in order to output the final result. Various methodologies can be used to build such an ensemble; one of the most popular is *bootstrap aggregating*, or *bagging*.

Consider a labelled dataset D, made by n instances, and a decision tree learning algorithm L. Decision tree learning algorithms have the characteristic of being *unstable*, meaning that little changes in the input may produce very different trees as output. In order to train an ensemble of k tree models, *bagging* exploits such an instability in the following way: for $i = 1, \dots k$, a dataset D^i is generated by randomly drawing $|D|$ instances from D with replacement (that is, the same instance may occur multiple times in D^i). Then, algorithm L is applied on each of the datasets, with the result of obtaining k different trees. In the prediction phase, the single tree outcomes are combined by voting (in a classification setting) or averaging (in case of regression).

The popular *Random Forest* (RF) algorithm [55] can be viewed as an evolution of the bagging methodology. Let k be the number of trees to generate in the ensemble. In the learning phase, the WEKA's implementation [125] of the Random Forest algorithm operates as follows: as in bagging, the dataset is repeatedly sampled with replacement for k times, obtaining k different datasets having the same cardinality as the original one. Then, a so-called *random tree* is built from each dataset, according to the traditional recursive TDIDT approach: starting from the root, at each node the algorithm randomly determines a (sub)set of the predictor attributes, from which it then chooses the one (categorical or numerical) that most effectively splits the set of samples into subsets with respect to the class labels. The

reason for considering only a subset of all attributes at each split is the correlation of the trees in an ordinary bagging approach: if one or a few attributes are very strong predictors for the response variable, these features will be selected in many of the trees in the ensemble, thus preventing one from achieving a high degree of variability among the models. In the regression setting, the choice of the best splitting attribute is made according to a numerical variance criterion: the attribute that maximizes the difference between the variance of the original node (calculated over the labels) and the sum of the variances of the child nodes is chosen. The splitting process continues until a predefined stopping condition is met, such as a constraint on the minimum number of instances in a node, a minimum reduction in variance, or when the tree has reached its maximum allowed height. In such cases, the corresponding node becomes a leaf of the tree, predicting the average label value of the training instances that belong to it. Notice that, unlike other decision tree learning algorithms, random trees do not perform any kind of pruning. Finally, the overall prediction of the forest is given by averaging the single tree predictions. Empirically, the RF algorithm exhibited a very good performance in terms of prediction accuracy in many application domains [331].

Among the different use cases in which the Random Forest algorithm has been profitably applied in the literature, we collaborated to the development of a solution to the problem of outdoor positioning based on cellular fingerprints, presenting a novel fingerprint comparison method capable of adapting to dynamic and large scale contexts. Specifically, test carried out on real business data, with different tracking devices and environmental settings, confirm that the novel approach is capable of providing consistently better estimations than all the other strategies previously considered in the literature. Since the work is out of scope with respect to the present thesis, we refer the interested reader to the original publications [322, 323].

1.4 Evolutionary algorithms

Evolutionary Algorithms (EAs) are population-based metaheuristics which rely on mechanisms inspired by the process of biological evolution and genetics in order to solve (typically discrete) optimization problems [105], that indicate the process of selecting a best element with respect to some criteria [185]. Unlike blind random search algorithms, EAs are capable of exploiting historical information to direct the search into the most promising regions of the search space, relying on methods designed to imitate the processes that in natural systems lead to adaptive evolution.

In nature, a population of individuals tends to evolve, in order to adapt to the environment in which they live; in the same way, EAs are characterized by a population, where each individual represents a possible solution to the optimization problem. Every solution is *evaluated* with respect to its degree of "adaptation" to the problem through a single- or multi-objective *fitness* function.

During the computation of the algorithm, the population iteratively goes through

a series of *generations*. At each generation step, some of the individuals are picked by a *selection strategy*, and go through a process of reproduction (i.e., new candidate solutions are generated), by the application of suitable *crossover and mutation operators*. In short, crossover is the EA equivalent of natural reproduction, by which the characteristics of two individuals are combined to generate one or more offspring, while mutation is used to maintain the genetic diversity in the elements of the population, through applying random changes in the encoding of the selected solution. Typically, a high crossover probability tends to pull the population towards a local minimum or maximum, while a high degree of mutation allows to explore the search space more broadly.

The selection strategy is perhaps the factor that mainly distinguishes between evolutionary-based meta-heuristics, although typically individuals with high degree of adaptation are more likely to be chosen (elitism): in this way, the elements of the population iteratively evolve toward better solutions. Finally, the algorithm terminates when a predefined criteria is satisfied, such as an upper bound on the number of generations, or a minimum fitness increment that must be achieved between subsequent evolution steps of the population.

Generalization is the ability of a model to perform well on new cases, not belonging to the training set. On the contrary, *overfitting* is a phenomenon which occurs when a model is too closely fit to a specific and limited set of examples, and thus fails in applying its knowledge to new data. Evolutionary techniques tend to produce solutions that are as good as possible for a given set of instances, against which the fitness function is evaluated, without considering the performances on possible new cases. This may be a problem when they are used within a broader machine learning process. Thus, in the recent years, generalization in evolutionary computation has been recognized as an important open issue [146], and several efforts are being made to solve such a problem [88]. For example, to reduce overfitting and improve generalization, in [146], the authors propose to use, through the generations, a small and frequently changing random subset of the training data. Another approach is given by *bootstrapping*, i.e., the repeated re-sampling of training data with replacement, with the goal of simulating the effect of unseen data. Then, the errors of each solution with respect to the bootstrapped datasets are evaluated, speculating that the lower their variance, the higher the generalization capability of the solution [119]. Also, an independent validation set may be used to separately assess the fitness of each individual in the population, which is nevertheless still evolved with respect to the training set [129]. Other strategies are based on the concept of reducing the *functional complexity* of the solutions. This is somewhat different than reducing the *bloat* (i.e., an increase in mean program size without a corresponding improvement in fitness), as it has already been observed in the past that bloat and overfitting are not necessarily correlated [318].

The previously discussed generalization techniques may also be implemented by means of two or more objective functions. *Multi-objective* EAs are designed

to solve a set of minimization/maximization problems for a tuple of n functions $f_1(\vec{x}), \ldots, f_n(\vec{x})$, where $\vec{x}$ is a vector of parameters belonging to a given domain. A set $\mathcal{F}$ of solutions for a multi-objective problem is said to be *non dominated* (or *Pareto optimal*) if and only if for each $\vec{x} \in \mathcal{F}$, there exists no $\vec{y} \in \mathcal{F}$ such that (i) $f_i(\vec{y})$ improves $f_i(\vec{x})$ for some i, with $1 \leq i \leq n$, and (ii) for all j, with $1 \leq j \leq n$ and $j \neq i$, $f_j(\vec{x})$ does not improve $f_j(\vec{y})$. The set of non-dominated solutions from $\mathcal{F}$ is called *Pareto front*. Multi-objective approaches are particularly suitable for multi-objective optimization, as they search for multiple optimal solutions in parallel. Such algorithms are able to find a set of optimal solutions in the final population in a single run, and once such a set is available, the most satisfactory one can be chosen by applying a preference criterion.

The next section briefly describes the selection strategy of NSGA-II [91], which is one of the most used and well-known multi-objective evolutionary algorithms to date. In the present work, it has been used to develop novel solutions for the tasks of decision tree pruning (see Chapter 4) and time series classification (see Section 6.1). Another original application which we worked on, but that is not discussed in the present thesis, is in the context of *information retrieval* systems evaluation [286]. Then, Section 1.4.2 briefly discusses ENORA, another intensively studied evolutionary algorithm, which we applied within the context of feature selection and fuzzy classification (see Chapter 3).

1.4.1 NSGA-II multi-objective evolutionary algorithm

NSGA-II (Nondominated Sorting Genetic Algorithm) [91] is perhaps the most known and used multi-objective evolutionary algorithm to date. As previously discussed, the most important aspect regarding the algorithm is its selection strategy, which is based on the concepts of *ranking* and *crowding distance*, and it aims to ensure *elitism* (meaning that the best solutions are given the opportunity to be carried to the following generations), as well as *diversity* in the population: the entire population is sorted into fronts, according to non-domination. A first front is made by the individuals which are non-dominated. A second one is composed of the individuals which are dominated by the elements in the first front only, and so on for the remaining fronts. Then, pairs of individuals are randomly selected from the population; finally, a *Binary Tournament* selection is carried out for each pair, considering a better-function based on the concepts of *rank* (which considers the front which the instance belongs to) and *crowding distance* (intuitively, it measures how close an individual is to its neighbours, and it is used to enhance the population diversity). Observe that, after the offspring generation phase, the population has doubled in size. In order to select the individuals to pass to the next generation, the entire population is sorted again based on non-domination, and the best ones, according to the same function as before, are selected (elitist criterion).

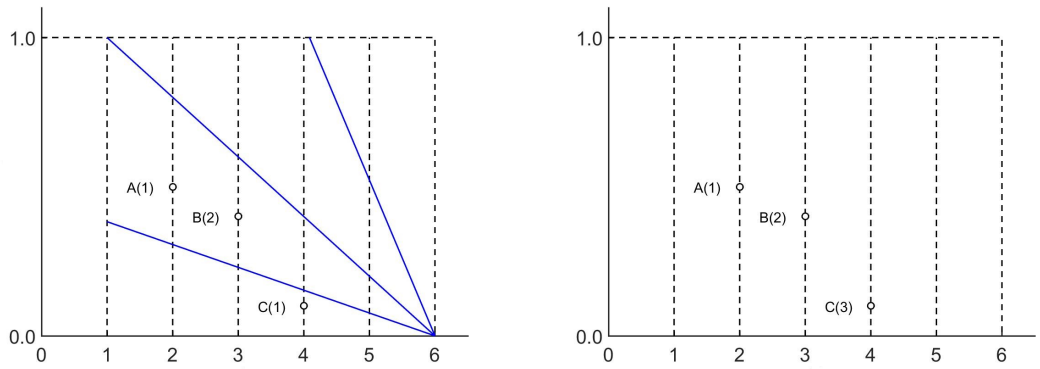


Figure 1.2: Individuals rank assignment with ENORA (left) vs. NSGA-II (right).

1.4.2 ENORA multi-objective evolutionary algorithm

ENORA (Evolutionary NOn-dominated Radial slot- based Algorithm) is a multi-objective evolutionary algorithm that has been intensively studied during the last decade: it has been applied to constrained real-parameter optimization [178], fuzzy optimization [182], fuzzy classification [181] and feature selection for regression [180]. The fundamental difference between ENORA and NSGA-II is how the calculation of the ranking of the individuals in the population is performed. In ENORA each individual belongs to a slot (as established in [180]) of the objective search space, and the rank of an individual in a population is the non-domination level of the individual in its slot. Conversely, in NSGA-II, the rank of an individual in a population is the non-domination level of the individual with respect to the whole population. The main reason that makes ENORA and NSGA-II behave differently is as follows. NSGA-II never selects the individual which is dominated by another one in the binary tournament, while in ENORA, the dominated individual may still be the winner of the tournament. Figure 1.2 shows this behaviour graphically for a multi-objective optimization problem. For example, if individuals B and C are selected for binary tournament with NSGA-II, individual B wins C because B dominates C. Conversely, individual C wins B with ENORA because individual C has a better rank in his slot than individual B. In this way, ENORA allows the individuals in each slot to evolve towards the Pareto front encouraging diversity. Even though in ENORA the individuals of each slot, when compared, may not be the best, this approach generates a better hypervolume than that of NSGA-II throughout the evolution process. This superiority of ENORA over NSGA-II has been verified in some feature selection problems for regression and classification tasks and for fuzzy classification in [179, 180] and [181] respectively.

2

An Event-based Data Warehouse to Support Decisions in Contact Centers

Nowadays, more and more companies deal with very large amounts of heterogeneous information related to their fields of interest, which are potentially extremely useful in developing current and future business strategies. *Business Intelligence* (BI) is a set of tools and techniques for the transformation of raw data into meaningful and useful pieces of information for business analysis purposes[1]. BI entails the management of (often huge) amounts of unstructured data to help in identifying, improving, and possibly defining new strategic business opportunities. In particular, BI aims at providing historical, current, and predictive views of business operations.

Unfortunately, apart from being unstructured, data of interest are often stored in several databases, possibly provided by different vendors, and may adopt different naming conventions or storage formats. Possible reasons for this situation are the gradual, step-by-step extension of company's information infrastructure, or the use of heterogeneous, independent software modules to satisfy different business needs. This makes the exploitation of information very hard. In many cases, the first step toward a profitable use of data is the creation of a single and unified repository, collecting and organizing all the needed pieces of information, namely, a *data warehouse*.

In most enterprises, a data warehouse is the core constituent of the company's *Decision Support System* (DSS) [227]. A DSS allows the company to manage all relevant pieces of information by leveraging a set of *Information Technology* (IT) tools and techniques, including traditional business intelligence and more sophisticated approaches to data analysis, such as data mining.

Decision Management Systems (DMSs) [311] are a further development of a company's infrastructure, whereby an *Artificial Intelligence* (AI) layer complements a decision support system, allowing preventive automated actions based on input information, possibly affecting the production systems. Thus, one may think of the entire process as a sense-decide-act cycle. Various authors have proposed the adoption of such systems in different domains, ranging from infrastructure management

[1]Business Intelligence on Forrester website: `https://www.forrester.com/report/Topic+Overview+Business+Intelligence/-/E-RES39218`

[262] to medicine [268] and education [299].

This chapter presents the development of a DSS for a multi-channel and multi-service contact center for front office business process outsourcing (BPO), along with its prospective extension to a DMS. At the best of the authors' knowledge, such a domain, which offers various interesting insights, has not been thoroughly investigated so far. The case project has been chosen on the basis of an ongoing research collaboration between the university and the R&D office of Gap Srlu contact center company, that provided many interesting insights on domain problems and inspired several meaningful research topics, part of which are explored in the present thesis, including an end-to-end natural language processing workflow for anomaly detection in call conversations (Section 5.2).

Since a BPO contact center typically deals with many different clients, the DSS must handle a huge amount of heterogeneous data, continuously originating from different sources, thus dramatically benefitting from a centralized data repository that allows many advanced data analysis tasks, e.g., enterprise-wide activity tracking of employees. To cope with data heterogeneity and fragmentation, an event-based data abstraction has been devised, which plays a fundamental role in the developed system.

The chapter is organized as follows. Section 2.1 provides some background knowledge, giving a short account of DSS, DMS, data warehousing, and data integration concepts. Then, Section 2.2 introduces the application domain, which is set in the context of BPO and contact centers, followed in Section 2.3 by a description of the infrastructure of a typical BPO information system, inspired by a real company. Afterwards, Section 2.4 outlines a new system infrastructure based on an original DSS. In doing that, special attention is devoted to the development of the new enterprise-wide data warehouse, based on the general concept of event, which is the core of the infrastructure (Section 2.5). Next, Section 2.6 presents some simple analysis tasks supported by the system. Then, the impact of the new system on the company operational processes is discussed. The remaining part of the work outlines system's extensions, that include the evolution of the DSS into a DMS, and the integration of more advanced analysis tasks, based on natural language processing.

2.1 Decision support systems and decision management systems

This section gives a short account of the fundamental notions of decision support system, decision management system, data warehouse, data loading, and data integration.

2.1.1 Decision support systems (DSSs)

A decision support system consists of a set of interactive and expandable IT techniques and tools for data processing and analysis, that supports managers in decision making [142]. To store, manage, and review strategic information, decision support systems make use of data warehouses [227], which explicitly support strategic reasoning. While traditional (operational) information systems are the typical sources of business data, they are not well-suited for dealing with information in DSSs.

A *data warehouse* is a large collection of data, useful for decision-making processes. Operationally, it aims at supporting *Online Analytical Processing* (OLAP) queries rather than *Online Transaction Processing* (OLTP) ones. Its distinctive features are [166]: *(i)* it is subject-oriented, rather than application-oriented, that is, unlike operational databases, which hang on enterprise-specific applications, and thus contain data which is typically organized by business processes belonging to the workflow of the company, it relies on enterprise-specific concepts, such as customers, products, sales, and orders; *(ii)* it is integrated and consistent, i.e., a warehouse is loaded from different sources that store information in various formats according to different conventions; during the migration process, specific tasks are used to check, clean, and transform data into a unified representation to make it possible to easily and efficiently access it; *(iii)* it is append-only, that is, once the data is inserted in the warehouse, it is neither changed nor removed; *(iv)* it evolves over time (time-variant), that is, since data is not volatile, a data warehouse includes time as a variant, that allows one, for instance, to distinguish current valid data from older "rewritten" entries (historical data).

There are two main approaches to the design of a data warehouse: *top-down* and *bottom-up*. The former starts with the design and development of an enterprise-wide database, making use of the Entity-Relationships and relational modelling techniques. Then, it proceeds with the creation of data marts, which store information according to the dimensional model: they source information from the central repository, filtering and aggregating it on the basis of the needs of particular enterprise departments or categories of users [166]. The latter first builds several multi-dimensional data marts that serve the different analytical needs of enterprise departments, starting by creating conformed dimensions that are universally shared across facts; then, the data warehouse results from the conglomerate of all of them [194].

Together with the design of the data warehouse, the location, acquisition, and integration of information generated by data sources play a major role in determining the success or failure of an overall DSS project, since they are among the most time-consuming, tedious, and error-prone tasks of the entire development process [277]. In the literature, the whole process is commonly referred to as *Extract, Transform and Load* (ETL for short) process. The ETL process starts with *schema matching*, which establishes semantic correspondences between elements in the source databases and in the data warehouse (tables and attributes). In the literature, it is possible to find

several approaches to (semi)automating schema matching [47, 278]. Nevertheless, it is still an open problem, and most currently-available solutions extensively resort to manual interventions by domain experts. Once all the correspondences have been established, the next step of the ETL process is *schema mapping*: given two input schemas and a set of correspondences between them, enrich those correspondences with semantics, i.e., rules that specify how to translate instances of the elements in the first schema into instances of elements in the second one. Like automatic schema matching, also automatic schema mapping is still an open problem, although [53] provide some insights regarding such a process. Even though schema matching and mapping are far from being completely automated [245], there is a variety of tools for defining and scheduling ETL workflows. One of the most well-known among them is the open source tool *Pentaho Data Integration*, a comprehensive data integration and business analytics platform, which is part of the larger *Pentaho Suite*[2]. Alternatives are *Open Studio* by *Talend*[3], a Java-based open source data integration software, and the *Tamr* software[4] (previously known as *Data Tamer* [304]), which employs machine learning algorithms to support ETL processes.

2.1.2 Decision management systems (DMSs)

One may think of decision management systems as an "action-oriented" evolution of decision support systems [311]. Generally speaking, a decision is the selection of a course of action from a set of alternatives, and it results in an action being taken, not just knowledge being added to what is known. DSSs aim at recommending such an action by offering managers information upon which to come up with an idea and ultimately to make a choice. DMSs make one step more and take decisions without human intervention on the basis of known information and a set of coded business rules. Of course, not all judgments may be automated. For instance, strategic decisions always need human consideration. However, in many companies there is a large set of recurring operational decisions, which involve a selection among a set of predefined actions. The logic behind them is usually simple and based on well-known data and operation patterns.

2.2 The application domain

Business process outsourcing (BPO) is the contracting of a specific business task, to a third-party service provider. Usually, a company implements it as a cost-saving measure for tasks that it requires, but do not constitute its actual core business [44]. As an example, a utility company may outsource its entire customer-care service, consisting of two interconnected processes: trouble ticketing and support

[2]Pentaho website: `http://www.pentaho.com`
[3]Talend Open Studio website: `https://www.talend.com/products/talend-open-studio/`
[4]Tamr website: `http://www.tamr.com/product/`

team field service management. In the considered scenario, the third party provider is a contact center that plays a key role as a multi-channel and multi-service hub, capable of coordinating such elaborated tasks.

2.2.1 The call center domain

This work focuses on contact centers for front office business process outsourcing. Let us introduce the call center domain.

Telephone call centers, as an integral part of many businesses, are an important part of today's business world. They act as a primary customer-facing channel for firms in many different industries, and they employ millions of operators across the globe. At its core, a call center consists of a set of resources, typically personnel, computers, and telecommunication equipment, which enable the delivery of services via the telephone. The common work environment is a very large room, with numerous open-space workstations, in which employees, called *operators*[5], equipped with headphones sit in front of computer terminals, providing teleservices to customers on the phone. A current trend, made possible by IT advancements, is the extension of call centers into multi-channel *contact centers*, where employees complement their "phone operator" role by services offered via other media, such as email, chat, or web pages [132]. As it operates, a large call center generates vast amounts of data, which one may split in two classes: *operational* and *service data*. The former concerns the technical information of a call (phone number, the operator who has served the call, possible call transfers, timestamps, ...), by which it is in principle possible to reconstruct a detailed history of each call that enters the system. The latter is related to the service for which the call has been made, e.g., in case of an outbound survey service, data would include all the answers given by the interviewed person.

2.2.2 Inbound and outbound call centers

It is possible to categorize call centers according to different dimensions. As an example, one may consider the functions they provide, such as customer services, telemarketing, emergency response, help-desk, and order taking. This notwithstanding, a primary distinction is between *inbound* and *outbound* activities.

Inbound call centers handle incoming traffic, which means that they answer to calls received from the customers, as in the case of help-desks. Note that inbound operations may also generate outgoing calls, as in the case of callback services, i.e., outbound calls made to high-value customers who have abandoned their (inbound) calls before being served, for example because of long waiting times.

Outbound call centers handle outgoing calls, that originate from the call center. Such calls may involve telemarketing initiatives or surveys. Typically, operators

[5]Observe that, in the context of this thesis, we may refer to operators also as *internal agents*, or simply *agents*.

dealing with this kind of operations follow a pre-defined *script*, which tells them precisely how to manage each call, e.g., how to announce themselves to the called person, how to carry on the call, how to answer to possible questions, and so on. In both inbound and outbound operations, the call center agency establishes with its clients a *Service Level Agreement* (SLA), that is, a quality level to guarantee for incoming calls, or a set of goals to reach during an outbound campaign.

2.3 The infrastructure of a typical system

This section presents an instance of a typical information system infrastructure, inspired by a concrete multi-channel and multi-service BPO contact center located in northern Italy.

As shown in Figure 2.1, the company makes use of two different *Customer Relationship Management* (CRM) systems to manage the (phone) interactions: one for the inbound activities and one for the outbound ones.

Data generated during the operations is stored into two distinct operational databases, one for each system, based on Microsoft SQL Server, and on a series of different MySQL databases, one for each offered service.

Each call generates two data sets: a *technical* and a *service* data set. The former is automatically generated by the two CRM systems, and it includes the time and the date of the beginning and ending of the calls, the caller and the called telephone numbers, and so on. The loading into the operational data servers is automatic. The latter is customer-related and usually manually generated by the call center operators. It includes, for instance, the answers given by the interviewed people to the interviewer and the annotations made upon a call by an operator. Once inserted, information is stored in service-dedicated databases.

All historical data coming from the operational data sources is then recorded into two distinct PostgreSQL databases, one for each CRM system, by means of an ETL process. This acquisition process does not execute any transformation: it simply copies new data from the operational to the storage databases. Then, other ETL scripts synthesize relevant information from data stored in the two archives and loads it into an Oracle database. Data in this latter system is ready to be displayed or used for SLA maintenance. Finally, an *Enterprise Resource Planning* (ERP) system has access to such data for invoicing and other project-support activities. The presence of many different databases (and Database Management Systems, DBMS), which may appear (and actually is) a source of problems, is quite common in companies which have been active for a long time, and thus have experienced successive upgrades of their information system infrastructure.

The number and variety of problems induced by the above-described architecture is large. First, the fragmentation of the data into different repositories makes it difficult to formulate complex statistical queries as well as to monitor the behavior of operators and customers by means of queries like *'what is the entire work history*

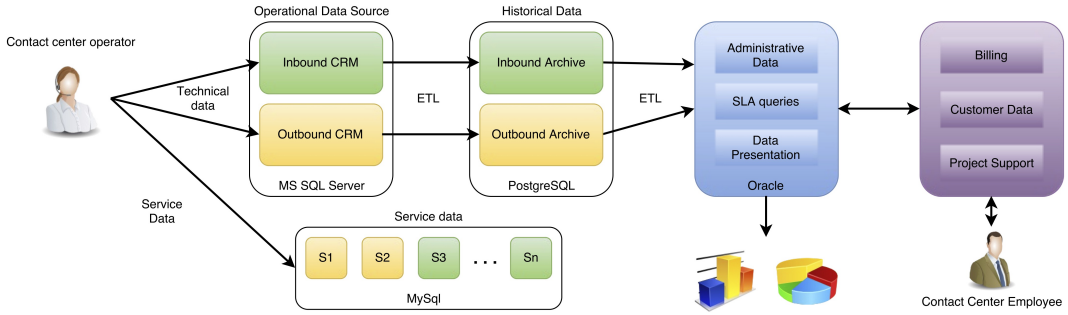


Figure 2.1: The infrastructure of a typical contact center information system.

of operator X?' or *'which is the complete set of interactions that person Y has had with the contact center up to the current date?'*.

Moreover, the two operational databases have been automatically deployed by a commercial CRM software, and they are poorly designed. First, tables and attributes have been tailored to the working logic of the systems, and their intended meaning is often unclear, making it difficult to understand their exact role. As an example, the queue waiting time for an inbound call may be obtained in two ways: directly, by reading the value stored in a dedicated attribute, or by subtracting the timestamps associated with the beginning and ending of the queue period. Due to internal rounding policies of the inbound CRM system, the two values may differ. Second, the two databases fail to enforce many typical relational constraints, such as foreign keys or unique attributes; this inevitably leads to the introduction of large quantities of truncated, replicated, and possibly conflicting entries. Third, there is a lot of redundancy in the data, which may easily lead to inconsistencies. Fourth, software bugs may exacerbate the problem of data inconsistency. As an example, the inbound CRM relies on different clocks for registering queue waiting times and call conversation durations. Fifth, even within each CRM different codes and conventions are used for storing information. As an example, it is possible to store a phone number as a single string, or by dividing it into a country code, an area code and a number; in turn, a country code may use a "+" or a "00" prefix. This makes data comparisons quite hard. The very same problems affect the two historical databases, since new data is simply copied into them. Last but not least, the whole system architecture is strongly coupled to the specific CRM systems used, and thus it turns out to be extremely rigid.

2.4 A new system infrastructure

The proposed decision support system architecture, depicted in Figure 2.2, easily maps on the general infrastructure for decision support systems defined by [300], and later extended by [227] and [308].

Data is collected from several sources: the process does not only consider CRMs

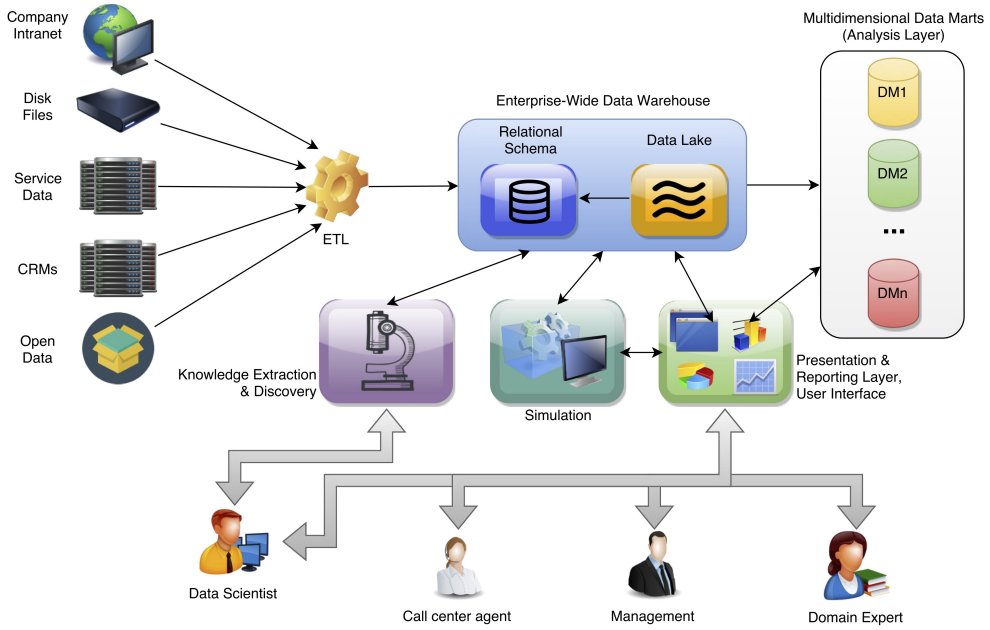


Figure 2.2: The architecture of the proposed DSS.

and service databases, but also automatically acquires raw files and company intranet data, as they provide useful information and insights on business rules, work specifications, and contracts with client companies. Also, it imports open data, including public telephones, address directories, and demographic statistics.

Dedicated ETL tasks load all information into an enterprise-wide data warehouse. The warehouse includes a relational schema for the memorization of structured data, such as those automatically generated by CRMs, and a data lake for storing semi-structured and unstructured information, including contents originating from parsed disk files containing answers to specific opinion surveys. Data may also pass from the *data lake* to the relational schema, as its structure evolves over time to accommodate new kinds of information.

Integrated, relationally stored information feeds a set of data-marts, which contain data required for specific business processes or of interest for specific categories of users, and adopt a multidimensional model: they pre-aggregate data, in order to speed up statistical queries needed by the enterprise.

Various categories of employees, including contact center operators, managers, operational staff (e.g., room supervisors and team/project leaders), and other domain experts, may access the system through a dedicated presentation and reporting layer, which provides relevant information, such as SLA maintenance reports, *Key Performance Indicator* (KPI) tracking, and present and past work statistics.

Besides supporting enterprise-wide, past-oriented statistical queries, data integration sets the basis for more complex, prediction- and action-oriented analytical tasks. Using the presentation interface, indeed, domain experts may set up statis-

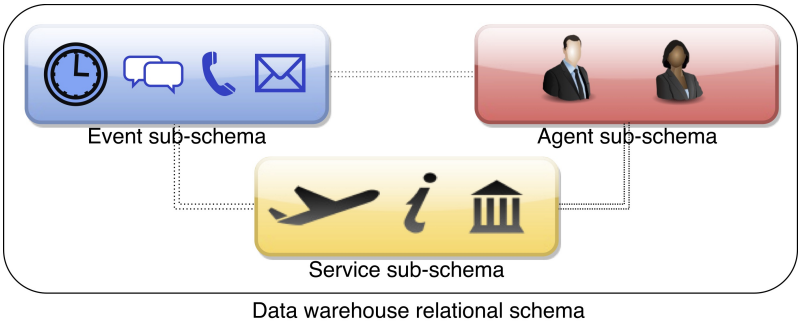


Figure 2.3: The relational sub-schemas of the data warehouse.

tical simulations, based on previously-observed traffic patterns, to test "what if" scenarios. As an example, they can formulate exploratory queries like: *'How long should I expect a particular inbound service queue to be if I am to add three operators to the workforce?'*.

Finally, data unification and reconciliation make exploitation of data mining techniques easier. By means of them, domain experts may be able to discover hidden patterns and regularities in data, and to predict future trends [331]. Some advanced applications are also described in this work, in Section 2.6 and Chapter 3.

2.5 The development of the data warehouse

The schema of the data warehouse at the core of the proposed infrastructure follows a normalized data model. Then, upon it, a multidimensional analysis layer has been defined as a *constellation schema* [176] including several *star schemas*, each of them providing a different insight on the data.

The main schema consists of three sub-schemas, namely, the *service*, the *event*, and the *agent* sub-schema, each of them focused on a distinct part of the domain. They are depicted in Figure 2.3 where double lines represent interactions between elements of the different sub-schemas (the representation of the corresponding relationships has been omitted; however, an intuitive account of them is given in the following). As a matter of fact, the proposed schema is general enough to allow one to adapt it, with a little effort, to various application domains different from the one of contact center operation.

A series of working sessions with the company's R&D and Operation offices brought us towards the definition of the final *Entity-Relationship* (E-R) schema.

2.5.1 The Service sub-schema

This sub-schema stores information about the inbound, outbound, and back-office services offered by the contact center. As an example, a service may be the toll-free

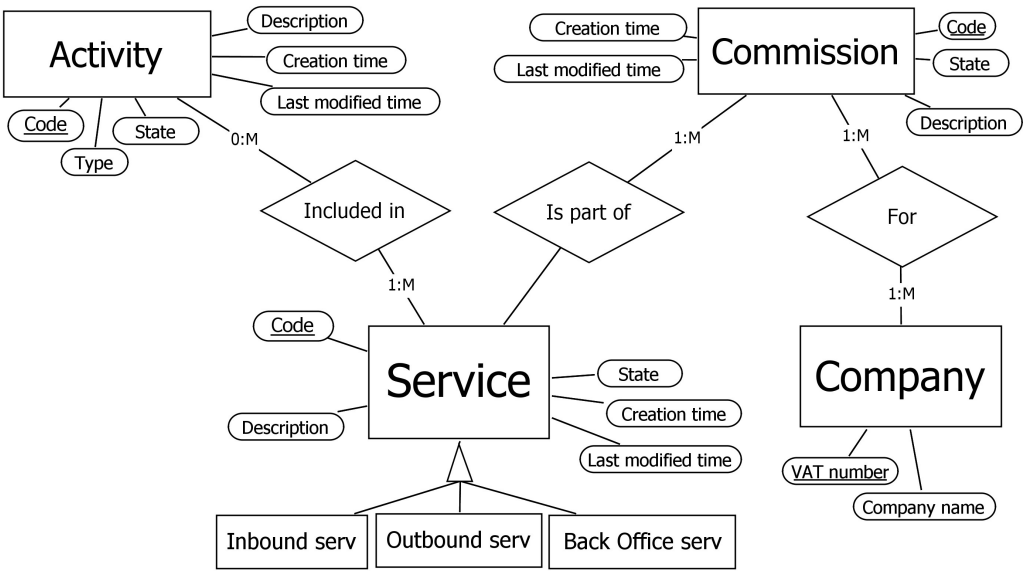


Figure 2.4: The service sub-schema.

number of an airline company. Each service consists of one or more *activities*. The airline company toll-free number service, for instance, may include ticket reservation, car rental, and so on. Related services are grouped into a *commission*, which is agreed upon with and offered on behalf of one or more contact center client *companies* (a consortium in the latter case). A graphical account is given in Figure 2.4.

2.5.2 The Event sub-schema

This sub-schema, depicted in Figure 2.5, is the most important component of the whole schema, as it is in charge of registering *events*, which are the fundamental "building blocks" of the *sessions*. Events are typically bound to the execution of a particular *activity*; a session, in turn, refers to a *service* (see Figure 2.4), and it may involve one or more customer files (see Figure 2.6) which are being carried out. Each closed session has an outcome that depends on the events that happened during the session. Conceptually, one may think of an event as something that has happened at a certain time (possibly for an extended period of time), and which is of importance to the company. Through events, it is possible to describe the operations carried out by an operator when doing back-office work, or to track the history of a particular phone call. Thanks to the general concept of event, it is also possible, through suitable specializations, to support to multi-channel communication, as events may describe various kinds of activity, such as chat conversations, email exchanges, and back-office work. As can be seen in the diagram (double lines), *event* is a weak entity that depends on the *session*.

Let us consider the case of phone calls. Each phone call initiates a new session, which is characterized by the people who are talking, their phone numbers (*iden-*

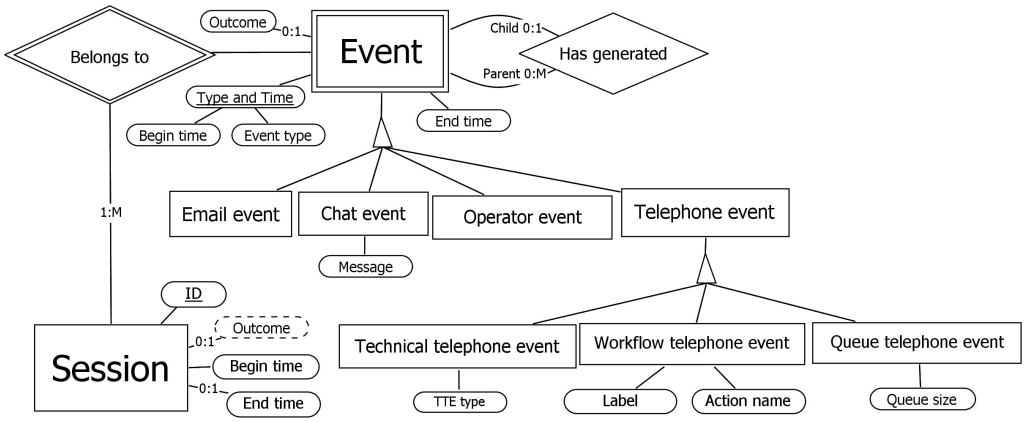


Figure 2.5: The event sub-schema.

tifiers, see Figure 2.6), the state of the involved phones (connecting, connected, on hold, etc.), the specific activity carried on (or activities, if there has been any internal call transfer), and so on.

Events are used to keep track of these characteristics and of their evolution over time. The recursive relationship on the entity Event deals with the case of an event generating a set of other events. As an example, consider a person who has called the contact center and is now selecting, through the phone keypad, the appropriate service to be linked to (the so called "Interactive Voice Response, IVR, choice"). The workflow event tracking the customer choice may start a new *Workflow event*, together with a *Queue telephone event*, as soon as the caller ends up in the appropriate waiting queue. Finally, observe that Event is a weak entity: within a given session, each event is uniquely identified by its type and start instant.

2.5.3 The Agent sub-schema

This sub-schema represents information about *agents*, that is, subjects who can take part to events. As depicted in Figure 2.6, each Agent is uniquely identified by its *Social Security Number* (SSN) or *Value Added Tax* (VAT) number, which is the minimum amount of information required to instantiate the entity (if there is no such a number, it is still possible to store residual data in the data lake).

Every instance of Agent is (disjointly) specialized into either a *Contact* or an *Internal agent*. The first one corresponds to subjects external to the company, e.g., customers of inbound services, while the latter represents the ones working for the contact center, like, for instance, the phone operators[6]. In a similar way, Contact is specialized into Individual or Legal entity, e.g., an enterprise.

Each contact may have one or more *identifiers* (not to be confused with the

[6]Observe that, in the context of this thesis, we may refer to internal agents as simply *agents* or *operators*, while external subjects are always referred to as *contacts*.

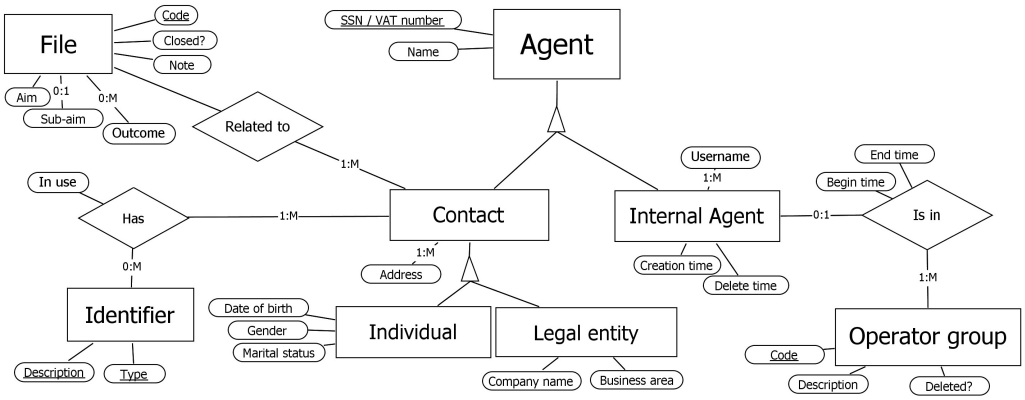


Figure 2.6: The agent sub-schema.

entity identifiers of the E-R model), which are values that characterize the external subject in the contact center domain. Identifiers are, for instance, telephone numbers or email addresses. As a matter of fact, they provide a semi-characterization of contacts, as, in general, they may be shared by different contacts (think of two persons who are living together and share the same home phone number). One or more *files* can be associated with a contact to describe the history of the contact's relationships with the contact center in the context of different services. As an example, a file may consist of all the calls (*sessions*) made to a particular person in order to complete a survey. Each file, once closed, has an outcome, which is in general independent of the outcomes of the sessions associated with it, and is strictly related to the specific service. Finally, an *Internal agent*, which is either a human or an *Interactive Voice Response* (IVR) system, may belong to an *Operator group*, i.e., a set of contact center operators who work on particular services.

2.5.4 The analysis layer

A set of multidimensional data-marts, which together form the so-called *analysis layer*, is continuously populated with data coming from the main database, whose schema has been outlined above. Such a layer is used to pre-aggregate data in order to speed up common OLAP queries needed by the company, such as knowing the current status of outbound survey campaigns, or accessing SLA maintenance statistics for inbound services. The schema of the analysis layer is arranged according to a *constellation pattern*, that is, it consists of several facts and several dimensions, which are possibly shared by facts [176]. The result can be seen as a "composition" of *star schemas*. The star schema related to inbound queue information is illustrated in Figure 2.7 (other schemas, which consider workflow aspects or outbound telephone conversations, follow a similar pattern). There is a fact table, namely, *inbound_queue_event*, surrounded by four dimensions: *time_interval*, *outcome*, *activity*, and *service*. Data in the fact table is aggregated by time-slot granularity,

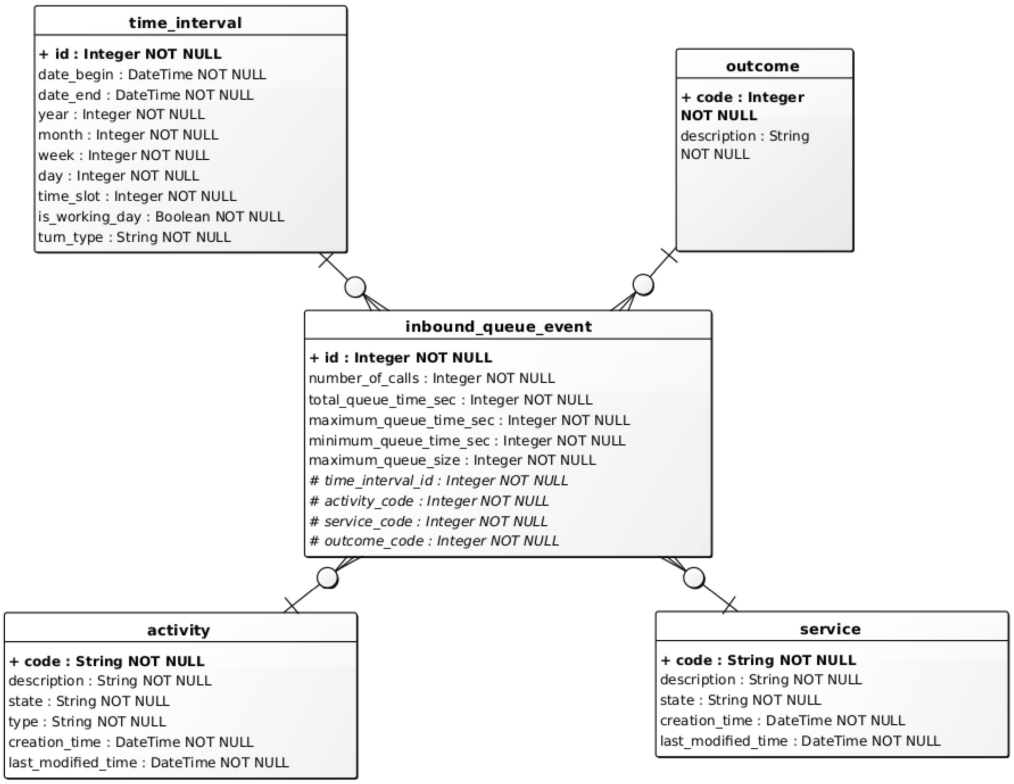


Figure 2.7: The star schema related to inbound queue information.

recording the number of enqueued calls for a particular service or activity, the total, maximum, and minimum queue times, and the maximum queue length.

2.6 Some analysis tasks

This section describes a representative set of analysis tasks made possible by the new, integrated DSS, ranging from patterns of global operator performance assessment to call-flow simulation and more advanced data mining analysis tasks. Note that all agent-related data have been anonymized in order to protect their privacy.

The first functionality that has been added to the new system is a method to formally compute operator performance by means of a single, integrated function, using information gathered from the data warehouse and a statistical model tailored to our specific requirements. Consider a set of attributes typically used in key performance indicators, such as, to mention a few: average call duration, amount of post-call work, percentage of successful outbound surveys, and metrics about

Table 2.1: Possible operator performance indicators.

Inbound	Average conversation time
	Average post-call time
	Generic call notes compiled per session
	Percentage of correctly filled script fields
	Purpose of the call
	Outcome of the call
Outbound	Average conversation time
	Average post-call time
	Amount of surveys over calls
	Number of answered calls per hour
General	Number of different kinds of services managed by an operator
	Degree of interleaving between services
	Respect of work schedule
	Turn flexibility

how operators record written information. For each service, within a suitable time frame, reliable average values for such attributes can be derived. Then, given a single operator working on a service over a certain period of time, it is possible to calculate averages over the same attributes and compare the results. Finally, a proper aggregation of the single performances synthesizes a global score. The overall result of the process is the building of general performance patterns, which allow the operational staff to have a high-level overview of service workflows.

Table 2.1 lists a set of possible performance indicators, while Figure 2.8 illustrates an interface for assessing the score evolution over time.

The curved line represents the global operator performance, which is periodically calculated. The straight, horizontal dashed line shows the operator average trend for the service. By selecting a single data point, it is possible to get additional information on the values for each skill used to compute the score. The performance dashboard is a valuable help for supervisors, as it allows them to examine the scores of each operator, compare different results, and discover possible criticalities that need to be addressed. Moreover, the performance function is a fundamental step in the development of the prospective DMS, for it may allow automated actions such as the assignment of a service to the best qualified operators or the issuing of operator-tailored training tasks.

A simulator capable of statistically predicting contact center inbound call-flow

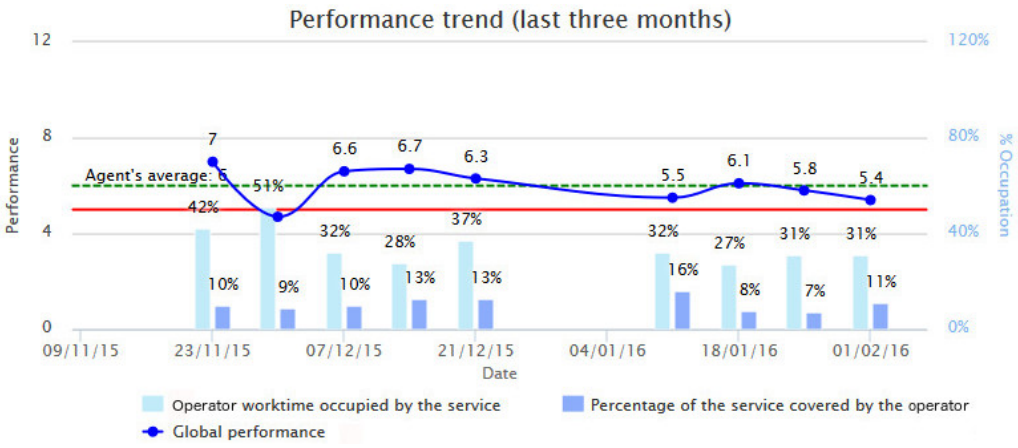


Figure 2.8: Operator performance assessment interface.

has also been developed. It is based on past-observed behaviors, and it is extremely useful for testing "what if" scenarios, such as the effect on waiting queues of the relocation of an operator from a service to another. Moreover, simulation also allows the operational staff not only to anticipate what will happen, but it could shed a light on the reasons why a particular situation has emerged, thus on the correct actions to take. This moves us from a predictive to a more prescriptive analysis.

The simulator interface, shown in Figure 2.9, presents much key information. The central, large graph keeps track of the global average durations of the different inbound call phases, such as the arrival time (during which the caller is making his/her IVR choice), the queue time (the caller has made his/her IVR choice and is now waiting to be served), the conversation time (the caller has been engaged by a call center agent), and the post-call time (the call has finished, and the agent is completing his/her last tasks). Right below, a second graph shows the proportion between the closed (served) and abandoned (renounced by the caller before being served) calls. On the last row, from the left to the right, one may find: the cumulative conversation time, the average response time, the cumulative number of incoming calls, and the queue size evolution over time. At the end of its execution, the simulator also presents a summary of call traffic statistics, such as the average conversation time, the average ratio of closed calls, and the average queue size.

Finally, also some more advanced solutions, based on data mining, have been developed. Several data mining tools are available. Among them, there is the well-known language *R* that can be used for the task. Another popular choice is the open source WEKA suite [125], developed at the University of Waikato. In the context of DMSs, one may rely on data mining to develop models capable of predicting future trends, thus allowing the proper proactive actions to be taken.

The first application is concerned with the analysis of the set of written notes, recorded by call center operators during their calls. Such a task allowed the company's supervising staff to better understand the different profiles of the operators.

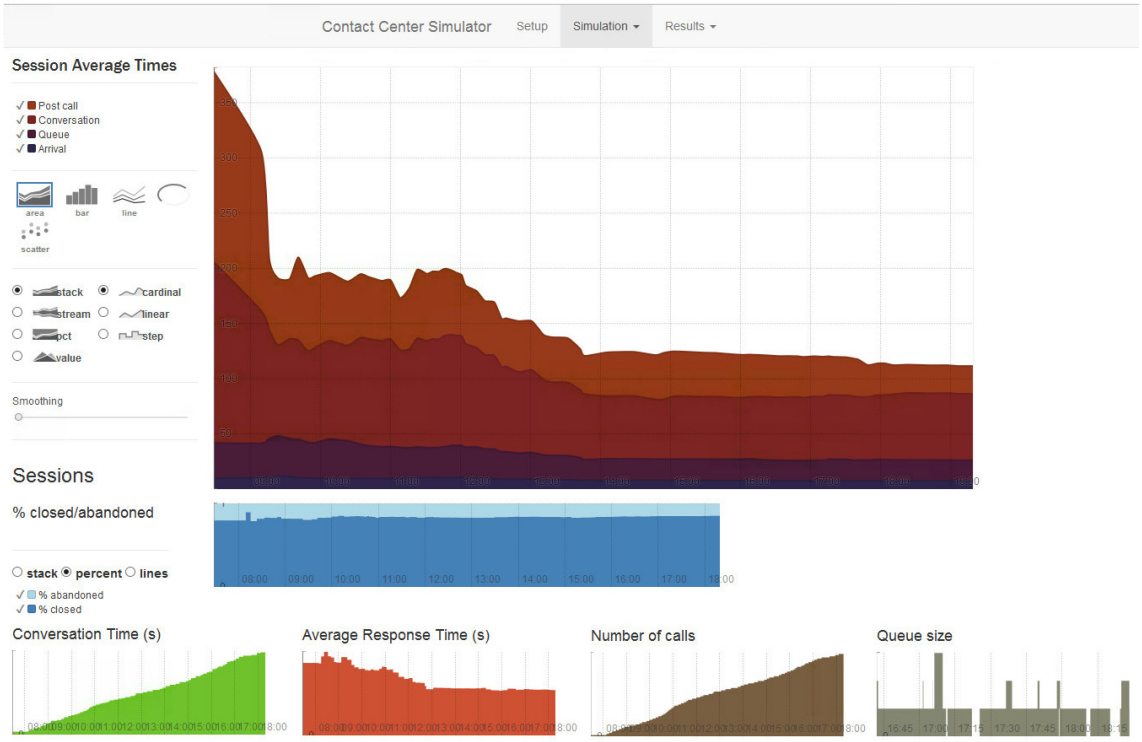


Figure 2.9: Call-flow simulator interface.

We followed the bag-of-words approach, according to which a text may be represented as the bag (multiset) of its words, disregarding their order. From the bag of each available written note, we then derived a set of quantitative attributes, such as, for instance, the number of adjectives, verbs, articles, dictionary-recognized terms, and dominion terms.

Then, a representative subset of all the notes has been selected, spanning all operators and services. Finally, unsupervised learning techniques have been employed to search for generic patterns in the dataset, specifically making use of *Expectation-Maximization* (EM), which is a distribution-based clustering algorithm [183] available on WEKA. One of the reasons that led to such a choice is the fact that in EM there is no need to manually specify the number of clusters to build, unlike other approaches like *k-means*. As a result, 6 clusters have clearly emerged, respectively representing: *(i)* notes mainly composed of abbreviated terms, *(ii)* well written, but not very well articulated, notes, i.e., notes that do not make extensive use of articles, connectives, and prepositions, *(iii)* well written and articulated notes, *(iv)* notes with a large degree of service-specific domain terms, *(v)* notes with a large portion of unrecognized words, and *(vi)* hybrid notes, mixing characteristics of the other clusters. In order to be able to classify all other (and also future) notes, a custom-made procedure enriched instances in the dataset with their cluster label; then, on

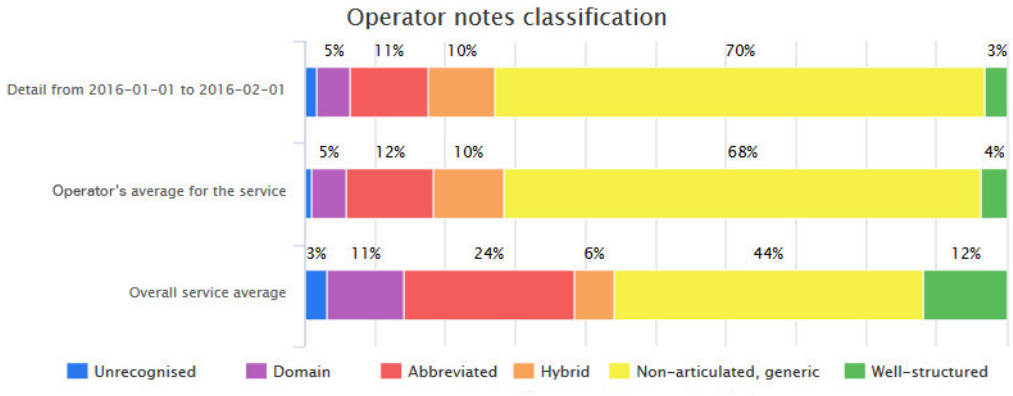


Figure 2.10: Operator notes interface.

the obtained training set, WEKA's J48 classification algorithm has been trained (an implementation of C4.5 [274], see Section 1.2.1), with the goal of predicting the cluster label. In this particular case, tests made in cross-validation mode reported an accuracy above 95%. Figure 2.10 shows the resulting operator notes interface. In the first bar, there is the proportion of notes of each kind written by the operator in a specific period of time for the service. The second and third bar show the general writing behavior of the operator and the overall distribution of service notes, respectively. Using the graphical interface, the personnel in charge of supervising the operators may quickly detect anomalies as they occur, e.g., a high degree of unrecognized notes compared to the average value for the service may indicate the occurrence of a pattern typical of a hurried operator; on the contrary, an operator writing too many well-structured notes may suggest a behavior in which the script fields are not used correctly.

The second data mining application is in the context of outbound services. Outbound calls follow a pre-defined script, which allows one to predict, to a certain extent, their outcome based just on dialling, conversation, and post-call times. This is extremely useful in outbound survey campaigns, as it may allow supervisors to detect contact center operators who systematically annotate wrong call outcomes, either by mistake or to simulate surveys which did not take place. The main idea is thus to build a classification model trained separately for each survey campaign, once again relying on the J48 classification algorithm. The first step is the extraction of the training set, which consists of sessions managed by extremely reliable operators, thus having their outcomes correctly set. Then, after an additional data cleansing phase, the tree-inducing algorithm is run on the training set. A tree model generated by such a procedure looks like the one depicted in Figure 2.11. In this example, the tree classifies as non-answered all phone sessions having a dialling time over 30 seconds, and in which no conversation took place. Moreover, calls with a conversation time between 76 and 87 seconds are classified as successful or unsuccessful based on post-call duration. This is reasonable, as an operator may

```
conversation_time <= 7
|  conversation_time <= 0
|  |  dialling_time <= 30
|  |  |  dialling_time <= 11: busy_or_nonexistent
|  |  |  dialling_time > 11
|  |  |  |  dialling_time <= 14: busy_or_nonexistent
|  |  |  |  dialling_time > 14: no_answer
|  |  dialling_time > 30: no_answer
|  conversation_time > 0
|  |  postcall_time <= 1
|  |  |  dialling_time <= 29: fax_or_answermachine
|  |  |  dialling_time > 29
|  |  |  |  conversation_time <= 1: no_answer
|  |  |  |  conversation_time > 1: fax_or_answermachine
|  |  postcall_time > 1
|  |  |  conversation_time <= 4: fax_or_answermachine
|  |  |  conversation_time > 4: spoken_no_survey
conversation_time > 7
|  conversation_time <= 76
|  |  conversation_time <= 11
|  |  |  postcall_time <= 1
|  |  |  |  conversation_time <= 9
|  |  |  |  |  dialling_time <= 22
|  |  |  |  |  |  conversation_time <= 8: fax_or_answermachine
|  |  |  |  |  |  conversation_time > 8: spoken_no_survey
|  |  |  |  |  dialling_time > 22: fax_or_answermachine
|  |  |  |  conversation_time > 9: spoken_no_survey
|  |  |  postcall_time > 1: spoken_no_survey
|  |  conversation_time > 11: spoken_no_survey
|  conversation_time > 76
|  |  conversation_time <= 87
|  |  |  postcall_time <= 0: spoken_no_survey
|  |  |  postcall_time > 0: survey_made
|  |  conversation_time > 87: survey_made
```

Figure 2.11: Decision tree for operator's fraud detection.

try to save conversation time by writing down only a trace of given answers on the fly, extending and completing them after the call. In such cases, successful phone conversations may show a below average conversation time, complemented by an active post-call phase. For this particular model, tests made in cross-validation mode reported an accuracy above 93%. Given the tree, it is straightforward to detect misclassified phone sessions, which may indicate wrongly reported outcomes. In the context of DMSs, a further development may allow for automatic call tagging. This is particularly useful for inbound activities: phone calls which are not believed to be ended with a positive outcome may be inserted in a recall queue by the system.

Observe that the above presented applications can be considered as simple and immediate consequences of the decision support system development. A more complex and advanced case study that makes use of contact center data is discussed in

Chapter 3, where a methodology that relies on feature selection and fuzzy classification techniques to predict whether an inbound call will or will not be abandoned by the caller before being served is presented [63].

2.7 On the impact on company operational processes

As confirmed by the initial findings, the development of the new infrastructure, and in particular of the enterprise-wide data warehouse, had a profound impact on the company in several respects. First of all, it allowed the operational staff to get complex historical information that was previously difficult or simply impossible to obtain, given the fragmentation of data. Moreover, it led to a more rational use of company resources, as it allowed to uniform all the analytical tools, effectively decoupling the sources of information, e.g., the CRM systems, from the data itself. This is extremely beneficial, since it permits to seamlessly change parts of the IT infrastructure, having only to redefine the mappings established between them and the warehouse. In fact, as a further proof supporting these claims, the entire inbound contact center system has been swiftly upgraded, preserving all the previously existing analytical processes. Finally, it opened for the contact center the possibility to manage data supplied by third-party companies. This represents a whole new business opportunity for the enterprise, which is about providing an outsourced data analysis service to third-party companies. As a matter of fact, the company has already started the deployment of the first analytics-on-demand service for an Italian airline company.

Let us now consider in detail the impact of the new analysis tasks. A typical decision making process in a contact center is mostly reactive rather than predictive. The introduction of predictive analytics and simulation tools supported by a data warehouse deeply changed the way decisions are taken in contact center management. In particular, initial findings provide some compelling examples showing that the decision support system can bolster several tasks, such as: *(i)* operator training, i.e., the assessment of typical operator's weakness and the administration of specific learning solutions; *(ii)* resource planning, e.g., a skill-based assignment that seeks to establish the best operator-service pairings; *(iii)* dynamic resource allocation, to determine the optimal number of operators for each service; *(iv)* detection of typical error and fraud patterns.

The proposed DSS tackles all these decision processes by providing a set of tools that help the operational staff at different levels of decision making, sometimes highlighting elements not usually considered. As an example, unified operator performance pattern evaluation, possibly complemented by hand-written call notes analysis, gives the staff a new overall perspective on contact center operators. This influences at least two of the previously-described decision processes, namely, oper-

ator training and skill-based assignment.

Another key application is the automatic call outcome classification, which supports one of the most important decision-making processes in contact center room management: the prompt detection of patterns related to errors and frauds. Given the expected high turnover and the possibly huge volume of managed contacts, this is a key factor. Predictive analysis exposes now to the operating staff the opportunity of detecting the so-called "operator anomalous behaviours", allowing them to take various actions, including: *(i)* issuing specific warnings to the operational staff; *(ii)* removing and reprocessing the fraudulent call; *(iii)* checking audio for further investigation.

Finally, it is worth stressing that the above-described operational decisions may be taken in a fully automatic way by defining a suitable rule set, thus driving the extension of the DSS into a proper DMS, as explained in the next section. Overall, the work done provides concrete evidence for the following general implication: it is possible to move the way of thinking from "supporting" to "(automatically) managing" decisions.

2.8 System's extensions

This section illustrates some extensions to the project, including the evolution of the DSS into a DMS, and the addition of other advanced analysis tasks, involving natural language processing.

2.8.1 Prospective DMS infrastructure

As already observed, one may extend the decision support system into a decision management system in order to automate routine contact center decisions, leaving managers free to dedicate themselves to the most important tasks, still supported by the DSS.

For instance, based on SLA maintenance requests and key performance indicators, a DMS may decide to: *(i)* allocate an additional operator to a service which is about to exceed SLA, searching for the best match between the profiles of the available operators and the characteristics of the service; *(ii)* assign the best available operator to each inbound call, considering the best match between customer and operator profiles; *(iii)* set a higher priority to a service which is experiencing long-queuing times; *(iv)* schedule outbound campaigns differently, according to their goal, e.g., in the management of enterprise-oriented calls, avoid typical break/congestion periods; *(v)* assign additional training tasks according to the emergent behavioral patterns in a service.

Such decisions impact directly on production systems and on the data they will generate, so one may think of the entire process as an automatic "sense-decide-act

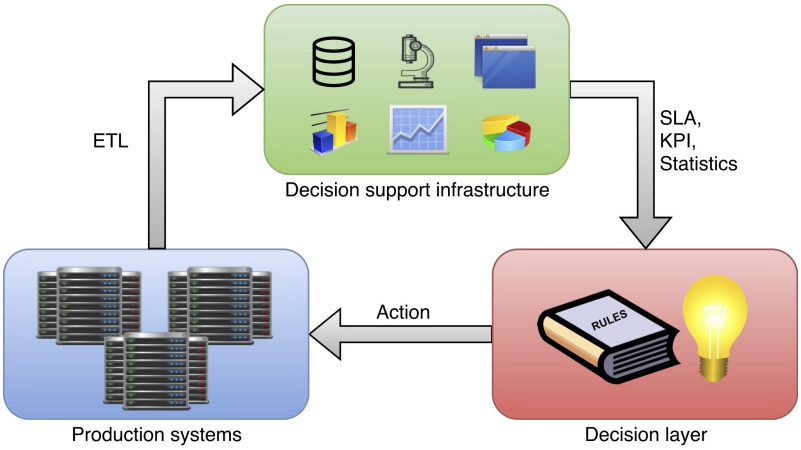


Figure 2.12: DMS infrastructure.

cycle" (see Figure 2.12). Based on advanced analysis tasks, such as data mining, indeed, the system may also act in a proactive way.

2.8.2 Analysis of call recordings

An enhancement to operator performance assessment can be based on the analysis of call recordings. The most straightforward approach is that of directly processing the raw audio files, evaluating indicators built on characteristics such as volume, pitch, energy, conversation turns, and pause durations. As an example, long conversation pauses might indicate deficiencies in training or "difficult" calls. Moreover, new indicators based on raw audio traces might permit to classify phone sessions according to the overall sentiment, as, for instance, in [30, 81, 89].

Also, automatic transcription of phone conversations may allow for more complex analysis, as discussed in Section 5.2, where a system capable of tracking and analysing anomalous call conversations is presented, that fully integrates in the already developed agent performance assessment applications.

3

Working with Numerical and Categorical Data

As discussed in Section 2.6, this chapter presents a case study based on the data collected in Gap's data warehouse, consisting of a methodology, based on feature selection and fuzzy classification, to predict whether an inbound call will or will not be served on time by the contact center staff. The study has been published as a standalone paper in [63].

3.1 Evolutionary feature selection and fuzzy classification of contact center data

As already mentioned in Section 2.2, when a contact center inbound phone session starts, after a pre-queue period during which the caller may be engaged via IVR (*Interactive Voice Response*) or simple vocal messages, the session is inserted into a waiting queue, and, in absence of more elaborate techniques, waiting sessions are scheduled to be managed by an agent via a simple First-In-First-Served protocol. Given the high number of sessions in a medium-sized contact center, and the limited number of resources, some sessions may never be managed by an agent (since the caller may abandon the call). It is important to notice that a simple statistical analysis of this problem, limited to the set of abandoned calls, presents a number of well-known issues. In particular, such an analysis would ignore the potential information carried by the set of non-abandoned calls and, therefore, it would be limited to an estimation of the number, frequency, and/or probability of abandoned calls, without highlighting the driving factors that lead to abandonment. Instead, the main purpose of this work is to answer the following question: *which factors most probably lead to an abandoned call?* In other words, we want to establish which data of an inbound session may influence its probability to be managed and, secondly, to build an interpretable classifier based on this analysis. In terms of practical usability, contact centers are interested in understanding which factors can help to minimize the number of abandoned calls and, in particular, to highlight the non-trivial ones. Such a knowledge becomes essential in this domain, where one of the main goals is

to optimize the engagement of the customer.

Specifically, in this work, we consider a real dataset again provided by Gap Srlu, consisting of more than thirty-thousand *inbound phone-based* sessions over a period of three months, for which the values of twenty features, including the outcome (a binary class, indicating whether the incoming call has been eventually managed or not), has been recorded. As we shall see, compared to previous data mining experiments on contact center datasets, the quality of the information at our disposal is considerably higher. Previous experiments, such as, for instance, those reported in [289], focus on service tree optimization, which is part of the design (as opposed to the evaluation) of a contact center. Other work on contact center datasets include [260], where the classification problem is approached via a hybrid model that uses several different techniques, however neglecting feature selection. In addition, the authors rely on a very restricted set of attributes, thus limiting the significance of the results. Instead, we focus on the problem of determining the call abandoning factors, modelling it as a feature selection task relying, as we shall see, on a *wrapper* methodology based on multi-objective evolutionary feature selection. Then, over the selected features, a set of interpretable rules is built by means of multi-objective evolutionary fuzzy classification techniques.

The use of evolutionary strategies for the selection of features has been initially proposed in [298]. Since then, it has been regarded as a powerful tool for feature selection in machine learning [316] and proposed by numerous authors as a search strategy (see, e.g., [32, 107]). In the first evolutionary approach involving multi-objective feature selection [167], three criteria (accuracy, number of features, and number of instances) are aggregated and then a single-objective optimization algorithm is applied. A formulation of feature selection as a multi-objective optimisation problem using a neuro-fuzzy based wrapper has been proposed in [108]. Other approaches, such as [106, 133, 173, 184, 321], propose to exploit NSGA-II [91] in combination with wrapper methods that use decision trees (such as C4.5 [173]), support vector machines [133, 184], maximal entropy-based models [106], or a filter method [321] that includes measures of consistency, dependency, and distance information. Finally, in [204] the evolutionary algorithm GENESIS has been adapted to work as a filter, using an inconsistency rate to evaluate individuals. As for our research, we have already relied on multi-objective evolutionary algorithms for the task of establishing the factors that in a contact center play a predominant role in the performance assessment of employees (see the work in [62]).

Evolutionary computation applied to fuzzy rule learning can be found, among others, in [90, 344], and it is advocated in recent surveys such as [160]. The first approaches in this direction appeared in the late nineties [144, 169, 310]. In [169], a non-Pareto multi-objective algorithm is used to minimize the classification error and the number of rules. A three-objective evolutionary algorithm for linguistic rule extraction is proposed in [170], which extends [169] with a new objective to minimize the length of the rules. In [177], Pareto-based evolutionary fuzzy rule

learners are used for function approximation and dynamic modelling in standard test problems studied in the literature, while in [310], a single objective genetic algorithm is considered to minimize the approximation error, complexity, sensitivity to noise, and continuity of rules by means of a weighted approach. Moreover, multi-objective evolutionary algorithms are used in [85] to perform feature selection and fuzzy set granularity learning in order to obtain compact and comprehensible fuzzy rule systems with high classification capacity, and in [171] to select fuzzy rules after extracting a large number of candidate rules by a heuristic approach. Both in [145] and in [171], sets of fuzzy rules, each one represented as an integer vector, are generated with different trade-offs between accuracy and complexity/interpretability. An effective multi-objective evolutionary algorithm has been proposed in [27], that makes a fast learning of simple and accurate linguistic fuzzy models possible. In [98], *PAES-RCS* method is used to maximize accuracy and minimize the total rule length for internet traffic classification. In [34], *IT2-PAES-RCS* extends *PAES-RCS* to employ Type-2 fuzzy sets, where sensitivity, specificity and total rule length are optimized for financial data classification. Finally, a distributed version of PAES-RCS by using Apache Spark as the data processing framework is proposed in [116], and in [117] Apache Spark is used as framework of a distributed multi-objective evolutionary algorithm to learn concurrently the rule and datasets of fuzzy rule-based classifiers by maximizing accuracy and minimizing complexity.

3.1.1 Objectives of the work

As already mentioned, the main goal of the work is to determine *the innermost factors that lead to a call to be abandoned*. To this end, we propose to exploit a multi-objective evolutionary algorithm as a strategy for feature selection in a supervised classification context, as a component of a full methodology that integrates pre-processing, feature selection for classification, model evaluation (and statistical test), and decision making, in order to choose the most satisfactory model according to an a-posteriori elaboration, driven by an external process, in a multi-objective context. We compare two state-of-the-art multi-objective evolutionary algorithms for this task, namely, ENORA (see Section 1.4.2) and NSGA-II (see Section 1.4.1), and we do so by designing two essentially different feature selection schemata: the first based on a wrapper with the decision tree learning system J48/C4.5 (Section 1.2.1), and the second based on the WEKA's Correlation-based Feature Selector multivariate filter *CfsSubsetEval* [153]. These four selection methodologies are, in turn, compared with two, more classical, strategies, that are: a single-objective wrapper based on WEKA's *BestFirst* [263] search strategy and J48/C4.5 as the evaluator, and WEKA's univariate filter *InfoGain* Ranker [96]. We perform an extensive test to evaluate the performances of the selected features for classification to establish which methodology behaves better. Figure 3.1 (upper part) recaps the considered feature selection strategies.

Of course, the number of possibilities for the comparative analysis is enormous, given the large number of search strategies, classifiers for wrapper methods and statistical measures for filter methods that can be considered. Our selection has been made based on the following criteria: *(i)* ENORA is a multi-objective evolutionary algorithm that has been intensively studied during the last decade: it has been applied to constrained real-parameter optimization [178], fuzzy optimization [182], fuzzy classification [181] and feature selection for regression [180]; *(ii)* NSGA-II, introduced by Deb, has proven itself very powerful and fast in multi-objective optimization contexts of all kinds, and most researchers in multi-objective evolutionary computation use NSGA-II as a baseline for comparison against new algorithms; although NSGA-II was born in 2002, to this day it remains a state-of-the-art algorithm, and its improvement can still be seens as a challenge; *(iii)* J48/C4.5 is a classifier that gathers different excellent characteristics for the usage in wrapper methods being fast, precise and interpretable; *(iv)* the correlation method used in CfsSubsetEval is a standard measure, widely used and documented in the literature, where subsets are evaluated considering the individual predictive ability of each feature along with the degree of redundancy among them; it is typically used today for comparison with multivariate methods [333]; *(v)* BestFirst is a hill climbing method that we consider in the comparison due to its simplicity, to establish a baseline; finally, *(vi)* InfoGain is often used together with a Ranker method in univariate feature selection tasks [51].

On the obtained reduced dataset, we then apply fuzzy classification. Again, we compare the performances of ENORA and NSGA-II adapted for such a task, against each other and against the WEKA's classification systems J48/C4.5, RandomForest, Logistic and FURIA. Note that RandomForest (and, to a certain extent Logistic, if we consider non-IT domain-experts) is a non-interpretable classifier that is typically more accurate than interpretable classifiers such as C4.5: if we compare interpretable classifiers with non-interpretable classifiers in terms of accuracy only, clearly the former are at a disadvantage thus, in this work, we do not only consider how accurate a classifier is, but also its degree of interpretability for a non-computer scientist domain expert. The comparison with J48/C4.5 and FURIA has also another objective. In this case both learners are, potentially, easily interpretable (the first uses decision trees while the second relies on fuzzy rules). We want to determine which classifier has a better trade-off between interpretability and accuracy, and to measure the former we consider the model complexities (tree size and number of leaves in C4.5, and number of rules and fuzzy sets in the rule-based classifiers).

Finally, this work extends the one reported in [179], that focuses on the application of attribute selection and supervised classification to phone session outcome prediction. A first, notable difference is given by the training dataset: in [179], only technical information is considered as the basis for the prediction of outcomes, which are manually set by human operators; even though this led to some interesting insights on the domain, the results indicated the presence of a semantic gap

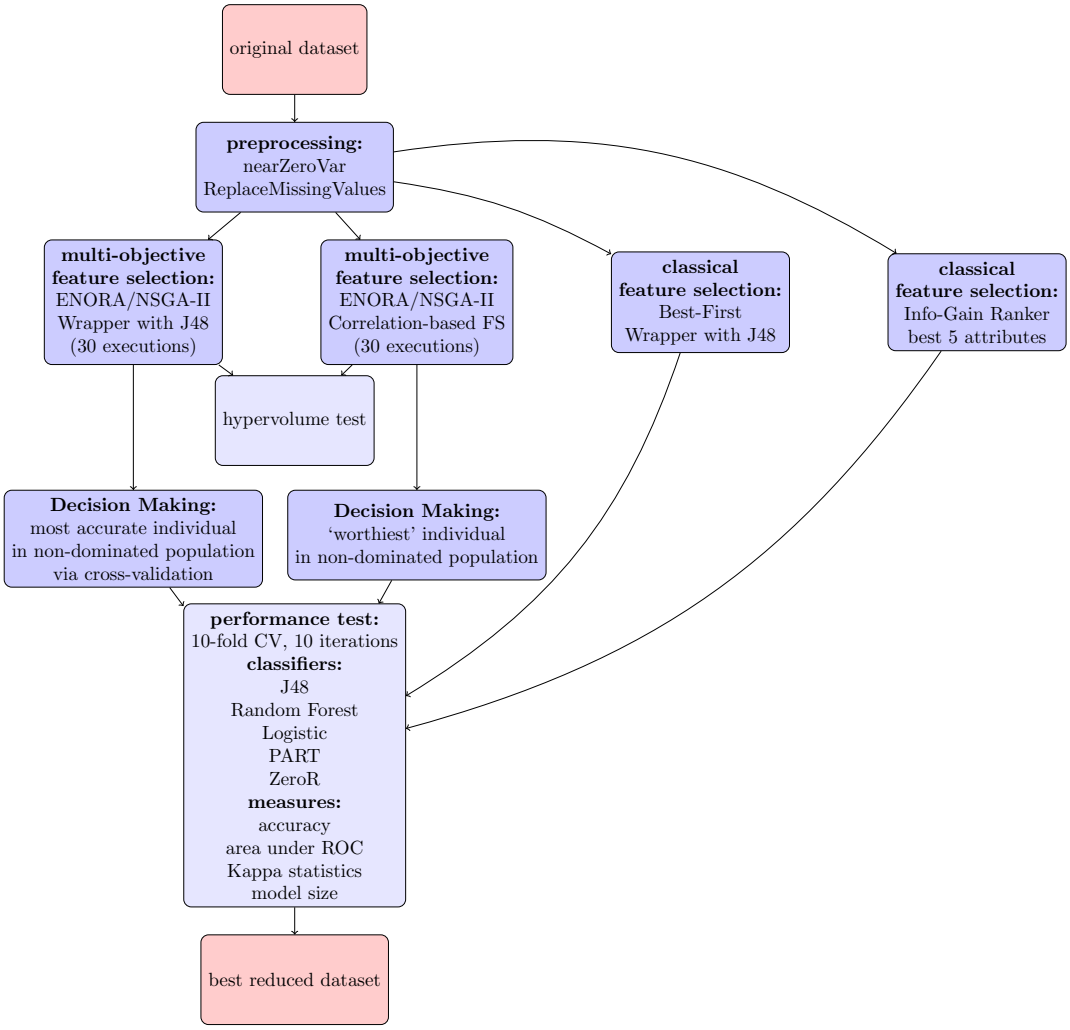


Figure 3.1: Proposed methodology for feature selection.

between the predictors and the predicted variable; on the contrary, here we look for relationships between the technical information of a call and the fact of it being managed or not, which is also an automatically recorded value.

3.1.2 A multi-objective evolutionary model for feature selection and fuzzy classification

In this section, we describe how the tasks of feature selection and fuzzy classification can be modelled as multi-objective optimization problems. As for feature selection, two methodologies are presented: a wrapper-based system with multi-objective evolutionary search that relies on the standard decision tree J48/C4.5 as the classification algorithm; and a multi-objective evolutionary search applied to the

standard multivariate filter CfsSubsetEval. As for the task of fuzzy rules extraction, a multi-objective constrained optimization model is proposed.

Supervised feature selection via wrapper and via filter

In adapting ENORA and NSGA-II to the task of feature selection, a fixed-length representation is used, where each individual consists of a list of M bits, each one representing a selected (1) or non selected (0) feature, being M the total number of features in the dataset.

As far as the wrapper-based methodology is concerned, an individual I is evaluated by means of two fitness functions, $f_1(I)$ and $f_2(I)$, corresponding to the two objectives of the multi-objective optimization model:

$$\begin{cases} f_1(I) = \mathcal{ACC}(I) \\ f_2(I) = \mathcal{C}(I) \end{cases} \tag{3.1}$$

Let $N_c(I)$ and $N_t(I)$ be the number of correctly classified instances and the number of total instances, respectively, of the J48/C4.5 classifier over the dataset with the attributes selected by the the individual I. $\mathcal{ACC}(I)$ is the *accuracy* of the classifier, which is defined as:

$$\mathcal{ACC}(I) = \frac{N_c(I)}{N_t(I)} \tag{3.2}$$

Hence, the function $f_1(I)$ must be maximized. $\mathcal{C}(I)$ is the cardinality of the subset of the selected features represented by individual I, and thus the function $f_2(I)$ has to be minimized. The intervals of f_1 and f_2 are obviously $[0, 1]$ and $[1, M]$, respectively. As for the wrapper approach based on the multivariate filter CfsSubsetEval, an individual I is evaluated according to:

$$\begin{cases} f_1(I) = \mathcal{COR}(I) \\ f_2(I) = \mathcal{C}(I) \end{cases} \tag{3.3}$$

where f_2 is the same as before, and f_1 is now based on the measure:

$$\mathcal{COR}(I) = \frac{\mathcal{C}(I) \cdot R_{cf}(I)}{\sqrt{\mathcal{C}(I) + \mathcal{C}(I) \cdot (\mathcal{C}(I) - 1) \cdot R_{ff}(I)}} \tag{3.4}$$

where $R_{cf}(I)$ is the mean class-feature correlation (for each feature selected by I) and $R_{ff}(I)$ is the average feature-feature correlation (again, features selected by I), by means of which a subset is evaluated considering the individual predictive ability of each feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class while having low inter-correlation are preferred [153]. Clearly, $\mathcal{COR}(I)$ must be maximized. Notice that while this metric can be naturally applied to nominal attributes, numerical attributes may only be considered after a discretization step.

As for the remaining adaptations, both ENORA and NSGA-II have been configured with *uniform random initialization*, *binary tournament selection*, ranking based on *non-domination level* with *crowding distance*, and *self-adaptive uniform crossover* and *one flip mutation* operators (see, e.g., [105, 180]).

An optimization model for fuzzy classification

In [181], a multi-objective constrained optimization model is proposed that allows to identify both accurate and interpretable fuzzy classifiers, as well as an evolutionary learning system to search for multiple Pareto-optimal solutions (classifiers) simultaneously, taking into account accuracy and interpretability according to the optimization model. Also, a detailed account of its main components is given, namely, solution representation, constraint handling, initial population, and evolutionary operators. The proposed optimization model can be described as follows:

$$\begin{cases} f_1(I) = \mathcal{ACC}(I) \\ f_2(I) = \mathcal{NR}(I) \end{cases} \tag{3.5}$$

subject to:

$$\begin{cases} \mathcal{NR}(I) \geq M_{min} \\ \mathcal{NR}(I) \leq M_{max} \\ \mathcal{NL}(I) \leq L_{max} \\ \mathcal{S}(I) \leq g_s \end{cases} \tag{3.6}$$

where f_1 takes into account, as before, the accuracy of the model, and thus must be maximized, and f_2 reflects the simplicity of the model established by counting the number of rules, and hence must be minimized. An individual I is a fuzzy rule-based classifier composed by $\mathcal{NR}(I)$ fuzzy rules, where each fuzzy rule R_j^I, $j = 1, \ldots, \mathcal{NR}(I)$ has the following structure:

$$R_j^I: \ \ if\ x_1\ is\ A_{1j}^I \wedge \ldots \wedge x_p\ is\ A_{pj}^I \ \ \rightarrow z\ is\ C_j^I \tag{3.7}$$

where $x_i \in [l_i, u_i] \subset \mathbb{R}$, $i = 1, \ldots, p$, $p \geq 0$, are real input attributes, and $z \in \{1, \ldots, w\}$, $w > 1$ is a categorical output attribute. Each fuzzy set A_{ij}^I, $i = 1, \ldots, p$, $j = 1, \ldots, \mathcal{NR}(I)$ is defined with a *gaussian membership function* [199]. The function $\mathcal{ACC}(I)$ is the accuracy of the classifier represented by the individual I over the dataset. The function $\mathcal{NR}(I)$ is minimized, and the constraints $\mathcal{NR}(I) \geq M_{min}$ and $\mathcal{NR}(I) \leq M_{max}$ limit the number of rules of the classifier represented by I to the interval $[M_{min}, M_{max}]$ (M_{min} is fixed to the number of classes of the output attribute, while M_{max} is given by user). The constraint $\mathcal{NL}(I) \leq L_{max}$ limits the number of linguistic labels[1] of the real input variables to L_{max}. Finally, the constraint $\mathcal{S}(I) \leq g_s$ ensures a maximum similarity g_s ($0 < gs \leq 1$) between the fuzzy sets; the similarity value of a classifier I represents the maximum value of

[1] As we shall see, each fuzzy set is associated to a linguistic label that is easily interpretable. For instance, a fever below 38 degrees may be labelled as *low*.

overlapping among their fuzzy sets for any input variable. The constraint $\mathcal{S}(I) \leq g_s$ is handled by the multi-objective evolutionary algorithm by means of a repair algorithm[2], which is applied after the initialization of the solutions, and after the crossing and mutation operations. As for the *reasoning method* (which can be intended as the firing policy in the fuzzy context), we rely on the *maximum matching* where the *compatibility degree* (i.e., the amount of applicability) of the rule R_j^I for the example $\mathbf{x}$ is calculated as:

$$\varphi_j^I(\mathbf{x}) = \prod_{i=1}^{p} \mu_{A_{ij}^I}(x_i) \ , \tag{3.8}$$

where $\mu_{A_{ij}^I}(x_i)$ is the membership degree of the i^{th}-component (attribute) of $\mathbf{x}$ to the fuzzy set A_{ij}^I. The compatibility degree is obtained by applying a t-norm product to the degree of satisfaction of the clauses $x_i \ \ is \ A_{ij}^I$. The *association degree* of the example $\mathbf{x}$ with the class C, is calculated by summing the compatibility degrees of each rule R_j^I whose value for the categorical output attribute C_j^I is equal to C, that is:

$$\lambda_C^I(\mathbf{x}) = \sum_{\substack{j=1,\ldots,\mathcal{NR}(I) \\ C_j^I = C}} \varphi_j^I(\mathbf{x}) \tag{3.9}$$

The *classification* for the example $\mathbf{x}$ or output of the classifier I, corresponds to the class C whose association degree is maximum, that is:

$$f_I(\mathbf{x}) = \arg_C \max_{C=1}^{w} \lambda_C^I(\mathbf{x}) \tag{3.10}$$

ENORA and NSGA-II have been implemented for this task using *variable-length representation* with floating-point input variables with a *Pittsburgh approach*, *uniform random initialization*, *binary tournament selection*, *handling constraints* using a *repair algorithm*, ranking based on *non-domination level* with *crowding distance*, and *self-adaptive variation operators* which work on different levels of the fuzzy classifier: *fuzzy set crossover*, *rule crossover*, *rule incremental crossover*, *gaussian set center mutation*, *gaussian set variance mutation*, *fuzzy set mutation*, *rule incremental mutation*, and *integer mutation* (for categorical data) [181]. Once the fuzzy rule set has been extracted, a linguistic label is assigned to each fuzzy set. The assignment of linguistic tags to fuzzy sets is performed at the end of the process by means of an algorithm (Linguistic Labeling Algorithm) whose details are shown in [181]. Algorithm 1 briefly reproduces its basic steps. Basically, the procedure consists of, first, calculating the number of linguistic labels from the distance between

[2]Intuitively, a repair algorithm can be thought of as a procedure that takes as input a model which violates some constraints, and applies some minimal modifications to it in order to make it satisfy such constraints. For example, in an evolutionary setting, the individuals which do not adhere to some given constraints may be given a penalty, while still being included in the population [324].

the centers of the fuzzy sets for each variable (steps 1 to 6). Next, the names of the linguistic labels are established according to the number of fuzzy sets previously calculated (steps 7 to 15; L:Low, M:Medium, H:High, VH:Very-High). The names used for the linguistic labels are standard names commonly used in fuzzy logic. Finally, the best approximating linguistic label is assigned to each fuzzy set (step 16 to 20).

Algorithm 1 Linguistic labeling algorithm

Require: $A_{ij} = (a_{ij}, \sigma_{ij})$, $i = 1, \ldots, p$, $j = 1, \ldots, M$ ▷ M is the number of rules of the classifier

Require: l_i,u_i ($l_i < u_i$), $i = 1, \ldots, p$ ▷ l_i and u_i are the lower and upper limits for real input variable i

1: **for** $i = 1$ to p **do**
2: **if** there is only one different fuzzy set for i real input variable **then**
3: $N_i \leftarrow 1$
4: **else**
5: $S_i \leftarrow \min_{\substack{j=1,\ldots,M \\ k=1,\ldots,M \\ A_{ij} \neq A_{ik}}} |a_{ij} - a_{ik}|$
6: $N_i \leftarrow \max\left(2, \left\lceil \frac{u_i - l_i}{S_i} - 0.5 \right\rceil\right)$
7: **switch**(N_i)
8: **case 1:** $L_i \leftarrow \{DCC\}$ ▷ Don't care condition
9: **case 2:** $L_i \leftarrow \{L, H\}$
10: **case 3:** $L_i \leftarrow \{L, M, H\}$
11: **case 4:** $L_i \leftarrow \{L, ML, MH, H\}$
12: **case 5:** $L_i \leftarrow \{L, ML, M, MH, H\}$
13: **case 6:** $L_i \leftarrow \{VL, L, ML, M, MH, H\}$
14: **case 7:** $L_i \leftarrow \{VL, L, ML, M, MH, H, VH\}$
15: **end switch**
16: **for** $k = 1$ to N_i **do**
17: $C_{ik} \leftarrow l_i + (k - 0.5)\frac{u_i - l_i}{N_i}$
18: **for** $j = 1$ to M **do**
19: $l \leftarrow \arg_k \min_{k=1}^{N_i} |a_{ij} - C_{ik}|$
20: $LABEL_{ij} \leftarrow L_{il}$
21: **return** $LABEL_{ij}$, $i = 1, \ldots, p$, $j = 1, \ldots, M$

3.1.3 Justification of the components of the multi-objective evolutionary algorithms

We chose not to report all the implementation details in this work because they are already described in [180] for feature selection and in [181] for fuzzy classification.

Regarding the choice of the different components of the evolutionary algorithms, the justification is as follows: NSGA-II is a powerful multi-objective evolutionary algorithm which we have compared ENORA with, and that uses $(\mu + \lambda)$-strategy and stochastic binary tournament. When two algorithms are compared, it is well known that they must have exactly the same components, and be executed with the same evaluations of the objective function, so that the differences in performance fall exclusively on the mechanism of the rank assignation of the individuals. Both ENORA and NSGA-II algorithms have been implemented with self-adaptive variation operators, with exactly the same crossover and mutation operators. Flip mutation and uniform crossover are typical mutation operators used for binary representations in feature selection. The variation operators used in fuzzy classification were designed for exploitation and exploration at all levels of the individual. As the individuals represent fuzzy set classifiers, the variation operators can operate at rule level or at fuzzy set level.

3.1.4 Feature selection: experiment design and results

This section illustrates the results of the experiment with the Gap Srlu dataset, obtained by a methodology which includes pre-processing of the data, feature selection, optimizers' performances comparison (based on hypervolume metrics), classifier learning construction, and test, as shown in Figure 3.1.

The Gap dataset

By a complex integration of various data systems [61], Gap Srlu it keeps a rich and detailed record of each agent-customer communication, that is, of each *session*, including both operational and service data. For inbound phone-based communications, recorded information includes operational data generated automatically such as the time slot (hour of the day, day of the week, day of the month, month of the year) at which the session has taken place, queue information (queue length, waiting time), the total duration of the session, the line type (mobile or home phone), and location of the caller; in addition, it includes service data related to the specific service for which the call has taken place, including cumulative workload/service up to the current time, service/activity category, and service priority. Overall, the raw data for the experiments include 19 features (numerical and categorical), and a boolean class that distinguishes the calls that have been managed by an operator from those that have been instead abandoned. A detailed account of the features can be found in Table 3.1. The purpose of the experiment is to identify which features, among

Table 3.1: A short account of the dataset attributes.

Attribute	Description
day_num	Day of month on which the call has arrived
week_day	Day of week on which the call has arrived
month	Month in which the call has arrived
queue_size	Length of the inbound waiting queue on call arrival
arrival_period	Time spent before entering a queue (e.g., making an IVR choice)
queue_period	Time spent in the waiting queue
tot_duration	Total duration of the call
calls_serv_mon	Number of calls of the specific service managed over the current month
sess_serv_inc	Number of incoming calls for the specific service, until the present call
sess_serv_day	Number of incoming calls for the specific service, over the current day
sess_serv_mon	Number of incoming calls for the specific service, over the current month
active_sess_ect	Number of active calls in the system at the end of the present call
queue_size_ect	Number of queued calls in the system at the end of the present call
phone_type	Whether the call is on a land line or a mobile line
caller_area_region	Region of origin of the call
service_type	Type of the service called
activity_type	Type of the activity selected via IVR
service_priority	Priority of the called service
session_time_slot	Hour slot at which the call has arrived
managed_or_not	Whether the call has been managed by an operator or not (class)

these, influence the fact that a session will eventually be managed by an operator. In the second phase, the identified (optimal) subset of features will be used to perform a fuzzy classification. The outcome can be particularly useful, as analyzing the fuzzy classification model Gap personnel may highlight factors influencing the fact that a session is never managed, thus possibly improving the service.

Data pre-processing

The initial dataset consisting of 19 features has been pre-processed as follows. First, all the missing values for categorical and numerical attributes have been replaced by, respectively, the modes and means from the training data using the procedure *ReplaceMissingValues* from the *weka.filters.unsupervised.attribute* package. Such a technique has the benefit of not changing the sample mean for the considered variable, however, mean imputation can be observed to attenuate any correlations involving the variable(s) that are imputed. Next, all features with too small variation have been removed, using the procedure *NearZeroVar* from Caret R [276]. As a matter of fact, no feature has been eliminated via this process, indicating that, potentially, all of them might influence the fact that the current session will or will not be managed by an agent.

Feature selection: experiment design and results

Both search strategies ENORA and NSGA-II have been integrated into a C4.5-based wrapper feature selection method, using the two objective functions described in Section 3.1.2, namely, accuracy maximization and cardinality minimization. After 30 runs, on each non-dominated individual of the last population of each strategy, a 10-folds cross-validation, by using the $\mathcal{ACC}$ metrics, has been performed, identifying the solution with the best cross-validation value (one for ENORA and one for NSGA-II). Specifically, for each run, the following evaluator has been considered:

weka.attributeSelection.WrapperSubsetEval -B weka.classifiers.trees.J48 -F 5 -T 0.01 -R 1 -E acc – -C 0.25 -M 2 .

Alongside, both ENORA and NSGA-II have been integrated as search strategies in the multivariate filter Correlation-based Feature Selector (CFS), using, as already explained, the following objective functions: subset worth maximization and cardinality minimization. After 30 runs, the best non-dominated individual, according to the $\mathcal{COR}$ metrics, that is, the one which performs best regarding non-redundancy and predictive degree with respect to the class (again, one for ENORA and one for NSGA-II), has been extracted. In this case, for each run, the following evaluator has been used:

weka.attributeSelection.CfsSubsetEval -P 1 -E 1 .

As a result, each of the 4 solutions determined a reduced dataset based on the selected features.

The two search strategies (ENORA and NGSA-II) have been implemented with dynamically adapted parameters [301], and written in Java by relying on the WEKA package [331]. Both have been run with the number of generations set to 100 and the number of individuals in each population set to 100, for a total of 10000 evaluations. The search strategy ENORA is incorporated into WEKA as an official package, namely *MultiObjetiveEvolutionarySearch* package [228].

Since the search for an optimal subset of features is performed by two different evolutionary algorithms, it makes sense to compare the *hypervolume* of the two executions, that is, the volume of the search space dominated by a population P [90]. The statistical comparison of the results obtained by the two optimizers, using a confidence interval of 90% for the mean obtained with a pairwise t-test [257], showed no statistically significant differences between the two search strategies, as witnessed also by the respective evolution diagrams (Figure 3.2). It is therefore possible to conclude that ENORA and NSGA-II show a similar behaviour for both problems (3.1) and (3.3), without statistically significant differences. In fact, the two search strategies ENORA and NSGA-II returned precisely the same results (Figure 3.3) with respect to both the J48/C4.5 and CFS based methologies. Therefore,

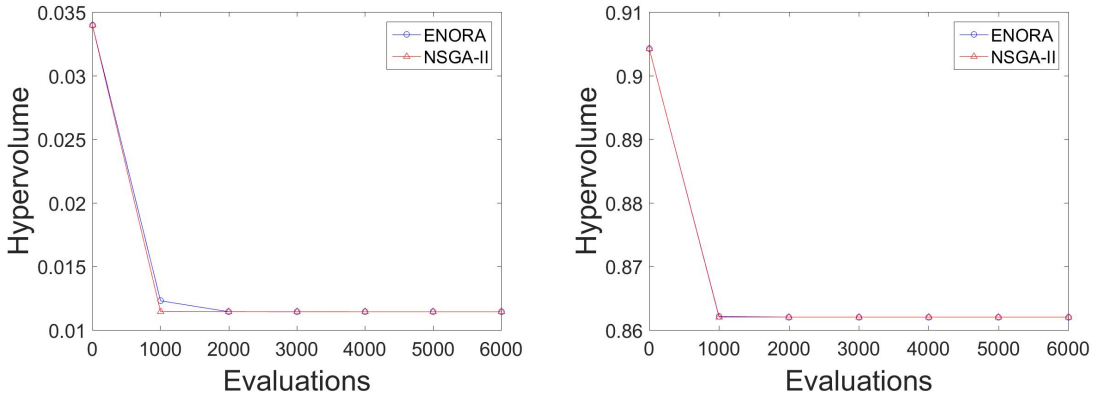


Figure 3.2: Mean hypervolume evolution (left: problem eq. (3.1), right: problem eq. (3.3)) – feature selection phase.

Table 3.2: Selected features for classification, for each method.

Attribute	WRAPPER-MO-DS	CFS-DS	WRAPPER-BF-DS	INFOGAIN-DS
arrival_period	X		X	
queue_period	X		X	X
tot_duration	X	X	X	X
queue_size		X	X	X
queue_size_ect		X	X	X
sess_serv_mon				X

2 (instead of 4) reduced datasets have finally been obtained: WRAPPER-MO-DS and CFS-DS, both with 3 out of 19 features (see Table 3.2). The *classical* feature selection methods, namely, BestFirst (with the same J48/C4.5 evaluator) and InfoGain Ranker, have produced two more datasets, hereby called WRAPPER-BF-DS and INFOGAIN-DS, with 5 attributes each.

All selections have been tested and compared configuring the *Experimenter* tool available in WEKA with the two datasets to perform a 10-fold cross-validation using the following classifiers: J48, PART (which produces a decision list from a partial C4.5 decision tree by transforming the best leaf into a rule [126]), ZeroR (which is a simple classifier to predict the mode for a nominal class [126]), Logistic and RandomForest (the latter being a non-interpretable classifier), all run with the default parameters set by WEKA. The result of the experiment has been analyzed through a paired t-test *corrected*, with 0.05 significance (being WRAPPER-MO-DS the test base). The following measures have been compared: *(i)* the percent of correct classifications; *(ii)* the (weighted) area under the ROC curve; *(iii)* the Kappa statistics; *(iv)* the serialized model size.

The outcomes of the test are shown in Table 3.3, where ORIGINAL-DS denotes the original dataset after the pre-processing. In terms of the performances of the classification models, all feature selection methods were able to reduce the number of

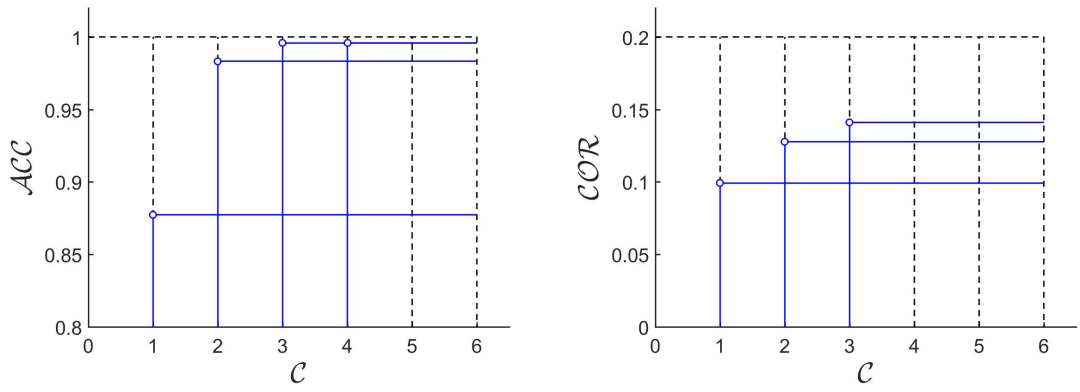


Figure 3.3: Pareto front of the best execution with ENORA and NSGA-II, last population (left: problem eq. (3.1), right: problem eq. (3.3)) – feature selection phase.

features in a very significant way (selections have a cardinality ranging from 3 to 5), allowing for an easier interpretation of the results; the performances of all classifiers obtained by means of the wrapper-based methodology in terms of all parameters stand out in most cases.

From the above analysis, WRAPPER-MO-DS, composed of 3 attributes (see Table 3.2), emerges as the best selection of the 19 original features. This is justified by, first, generally better performances (see Table 3.3) and, second, expert evaluation. As a matter of fact, the selection WRAPPER-MO-DS gives more interesting insights on a fundamental aspect: the potential relationship between the *arrival_period* (that is, the amount of time that the caller spends while engaged with the IVR) and the *queue_period* (that is, the amount of time that the caller spends while simply waiting). On the other hand, the selection CFS-DS shows features that only take into account aspects related to the queue dimension, which might be less interesting. Finally, note that the set of variables selected by WRAPPER-MO-DS is included in the set of variables selected by WRAPPER-BF-DS. That means that the variables common to both sets are very important to the problem at hand. Additionally, WRAPPER-MO-DS has discarded 2 variables that are instead selected by WRAPPER-BF-DS, which means that the multi-objective optimization is being effective. As for INFOGAIN-DS, it somewhat hybridizes the results given by WRAPPER-MO-DS and CFS-DS, and it is also the only one that considers the attribute *sess_serv_month*.

In conclusion, the feature selection phase confirmed that the probability of a session being eventually managed by an agent (therefore not being abandoned) is influenced by the quantity of time that the customer spends in the waiting phase (which is partially active, when engaged by an IVR); while this might not be surprising, the relationship that emerges among the selected features via the fuzzy classification, explained in the next section, may lead to some interesting observations.

Table 3.3: Comparative results of the test.

	WRAPPER-MO-DS	CFS-DS	WRAPPER-BF-DS	INFOGAIN-DS	ORIGINAL-DS
			Percent correct		
J48	99.6072	91.4558	**99.6085**	99.2661	99.5422
PART	**99.5890**	91.3280	99.5556	99.2389	99.3555
Random-Forest	**99.8128**	90.5964	99.7731	99.3442	98.7015
Logistic	**99.9964**	86.7527	99.9947	98.6166	89.3069
ZeroR	83.4948	83.4948	83.4948	83.4948	83.4948
			Weighted avg. area under ROC		
J48	0.9944	0.9288	**0.9946**	0.9901	0.9943
PART	0.9962	0.9424	**0.9967**	0.9958	0.9923
Random-Forest	**0.9997**	0.9110	0.9997	0.9982	0.9983
Logistic	**0.9999**	0.8863	0.9999	0.9980	0.9411
ZeroR	0.5000	0.5000	0.5000	0.5000	0.5000
			Kappa statistics		
J48	**0.9858**	0.6541	0.9858	0.9733	0.9834
PART	**0.9851**	0.6512	0.9839	0.9724	0.9767
Random-Forest	**0.9932**	0.6355	0.9918	0.9762	0.9523
Logistic	**0.9999**	0.3975	0.9998	0.9496	0.6062
ZeroR	0.0000	0.0000	0.0000	0.0000	0.0000
			Serialized model size		
J48	52979.4000	**20617.1600**	50661.3200	65121.4400	55717.8800
PART	129450.0500	**100447.7500**	124713.5700	176653.8300	356636.7400
Random-Forest	767094.9400	**3439187.3100**	764803.3400	1047677.7700	2187567.8300
Logistic	**7649.0000**	7662.0000	8671.0000	8883.0000	17942.0000
ZeroR	907.0000	907.0000	907.0000	907.0000	907.0000

Fuzzy classification: experiment design and results

In order to improve the interpretability of the results, let us now focus on the reduced dataset WRAPPER-MO-DS and search for a fuzzy classification model, as explained in Section 3.1.2. Data need not to be pre-processed again; therefore, the remainder of the present section is going first to describe how the experiment has been designed and, then, the results that have been obtained.

Once again, the search strategies ENORA and NSGA-II have been adapted following the methodology outlined in Section 3.1.2. They are both available in the WEKA class

weka.classifiers.rules.MultiObjectiveEvolutionaryFuzzyClassifier .

The goal is that of searching for fuzzy sets with a minimum of 2 and a maximum of 12 rules, with a maximum similarity (g_s) of 0.4. The maximum number of linguistic labels has been limited to 7. Both search strategies have been run with 100 individuals in each population and 100000 evaluations. The purpose of this phase is threefold: *(i)* to establish, via hypervolume test, the best search strategy for fuzzy classification among ENORA and NGSA-II for this particular dataset; *(ii)* to obtain an interpretable classification model, and compare it with other interpretable classifiers based on decision trees (C4/J48) and fuzzy rules (FURIA[3] [163]); and *(iii)* to

[3]FURIA (Fuzzy Unordered Rule Induction Algorithm) extends the well-known RIPPER rule learner, while preserving its advantages, such as the generation of simple and comprehensible rule sets. In addition, it includes a number of modifications and extensions. In particular, FURIA learns fuzzy rules instead of conventional rules and unordered rule sets instead of rule lists. Moreover, to deal with uncovered examples, it makes use of an efficient rule stretching method.

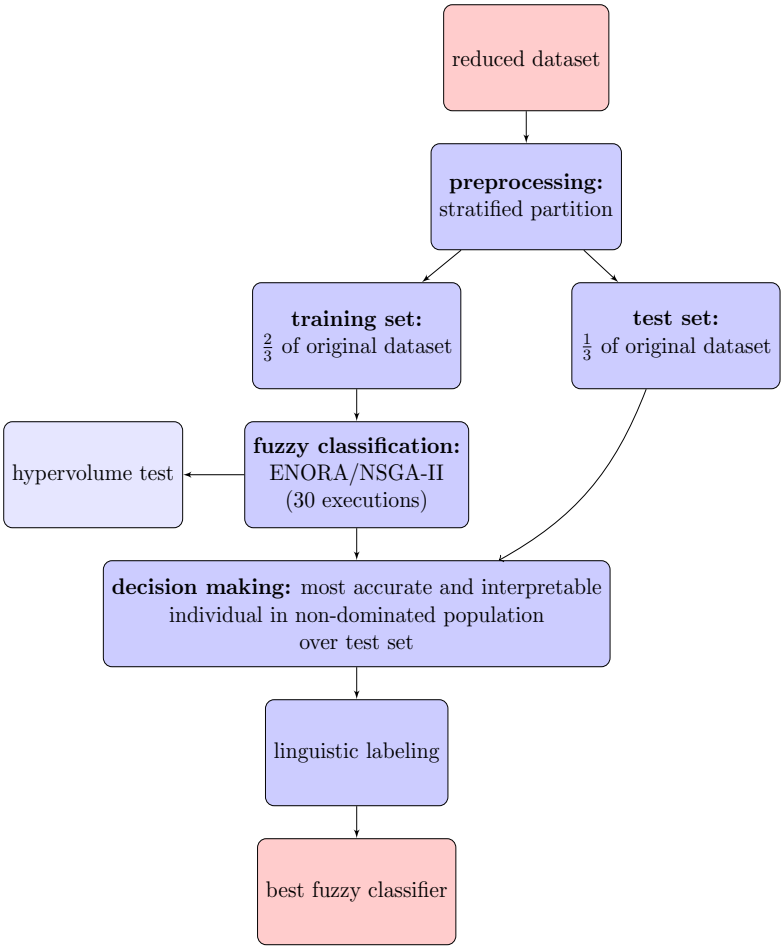


Figure 3.4: The proposed methodology for fuzzy classification.

analyze how accurate our fuzzy rule-based classifier can be by comparing it with highly accurate non-interpretable classifiers, such as RandomForest.

A *stratified* partition, that is, a partition whose elements preserve the class distribution, is obtained by means of the WEKA's algorithm Stratified Remove Folds [331]. The two generated subsets include 66% and 33% of the instances, respectively, and are denoted here as WRAPPER-DS2T and WRAPPER-DS1T. Two fuzzy classifiers (one with ENORA and one with NSGA-II) have been trained over WRAPPER-DS2T. Then, the best two individuals after 30 runs have been chosen based on their performances over WRAPPER-DS1T, and compared against each other and against a decision tree learned with J48 over WRAPPER-DS2T and tested over WRAPPER-DS1T (see Figure 3.4).

In terms of hypervolume statistics, the result of the comparison can be found in Table 3.4, together with the relative evolution graph in Figure 3.5. Most notably, unlike the feature selection phase, in the fuzzy classification phase ENORA outper-

Table 3.4: Mean hypervolume statistics over 30 runs – fuzzy classification phase.

	ENORA	**NSGA-II**
Minimum	0.0049	0.0428
Maximum	0.0975	0.1546
Mean	0.0285	0.1122
S.D.	0.0231	0.0443
C.I. Low	0.0198	0.0956
C.I. High	0.0371	0.1287

S.D = Standard Deviation
C.I. = Confidence Interval (95%)

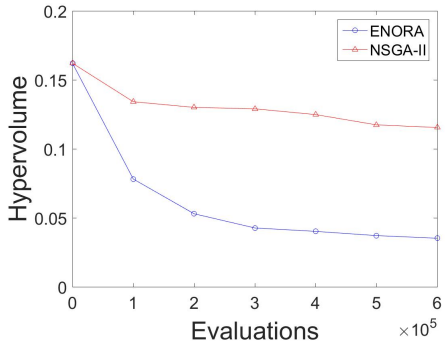


Figure 3.5: Mean hypervolume evolution – fuzzy classification phase.

formed NSGA-II. As can be seen in Figure 3.6, the Pareto front of ENORA contains 8 classifiers, with 2, 3, 4, 5, 6, 7, 8 and 10 rules respectively, while the Pareto front of NSGA-II contains 4 classifiers with 2 , 3, 4 and 5 rules respectively. This means that ENORA shows more diversity than NSGA-II. Although the classifier with 10 rules is the most accurate in ENORA, we have finally chosen (*a posteriori*) the classifier with 4 rules, given that the difference in accuracy between such classifiers is very small (approximately 0.001) but, nevertheless, the 4-rule classifier is much more interpretable than the 10-rule classifier. For NSGA-II, the final decision was the 5-rule classifier.

Let us now analyze both fuzzy models. They are referred to as ENORA-RULES and NSGA-II-RULES, and have been obtained, as explained, in full-training mode over the dataset WRAPPER-DS2T, which contains a stratified partition of two thirds of the instances of the dataset WRAPPER-MO-DS. ENORA-RULES contains 4 rules and 8 labels, while NSGA-II-RULES contains 5 rules and 8 labels. In terms of performance, not only ENORA-RULES is more interpretable than NSGA-II-RULES but, also, ENORA-RULES is more accurate than NSGA-II-RULES. In other words, it is a more interpretable and precise classifier, that can be used to reveal correlations

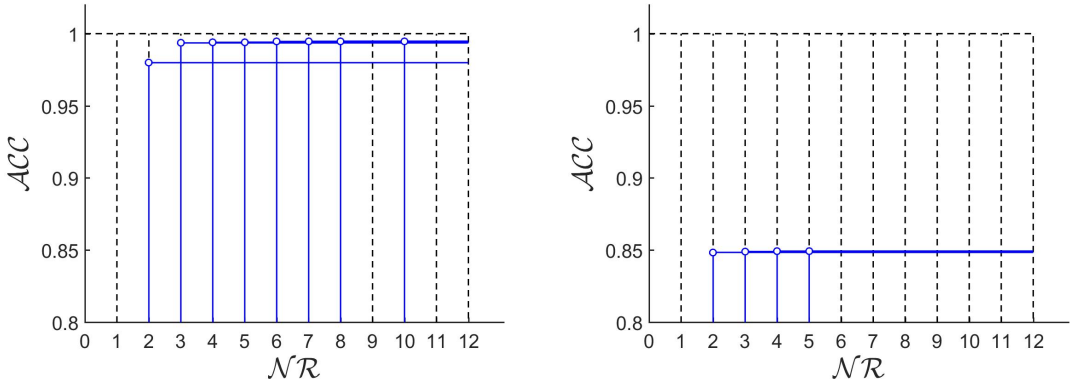


Figure 3.6: Pareto front of the best execution, last population (left: ENORA, right: NSGA-II) – fuzzy classification phase.

Table 3.5: Fuzzy rule-based classifier learned with ENORA.

Rules				
#Rules	**arrival_period**	**queue_period**	**tot_duration**	**Class**
r1	M	M	ML	managed
r2	L	VH	VH	managed
r3	VH	VH	MH	managed
r4	M	VH	MH	not_managed

between the different waiting times and the probability of a session being managed or abandoned. In particular, the rules show that the time spent during the IVR phase is important, and, in particular, that calls are more likely to be abandoned when the IVR phase is not too short nor too long. This can be interpreted as follows: if the IVR phase is short, then the call is managed quickly (which turns out to be in fact a characteristic of the Gap services considered in this dataset), while if it is particularly long, the caller has already "invested" so much time that is not willing to give up. Concerning the number of rules and the different fuzzy sets for each variable, as already reported the model is compact. The similarity of fuzzy sets is minimal, even if a maximum value of 0.4 has been imposed, and the linguistic labels that have been found are acceptable in number. Therefore, the chosen model entirely fulfils the interpretability and compactness criteria that have been imposed. In Figure 3.7, the fuzzy sets are shown in graphical form. ENORA-RULES is shown in Table 3.5, and the relative fuzzy sets are shown in Table 3.6.

Finally, both ENORA-RULES and NSGA-II-RULES have been contrasted to J48/C4.5, RandomForest, Logistic and FURIA learning algorithms, trained over WRAPPER-DS2T, and the results of such a comparison over the test dataset WRAPPER-DS1T are shown in Table 3.7. Table 3.7 also shows that, while ENORA

Table 3.6: Fuzzy sets for the classifier learned with ENORA.

Gaussian Fuzzy Sets			
Attribute	**Mean**	**S.D.**	**Linguistic Labels**
arrival-period	19.0568	12.9271	Low (L)
	52.6136	19.9698	Medium (M)
	93.4797	11.1821	Very High (VH)
queue-period	1629.2688	222.6852	Medium (M)
	3163.1231	298.3534	Very High (VH)
tot-duration	22070.0134	2243.1530	Moderately Low (ML)
	42962.2793	7010.2555	Moderately High (MH)
	62834.9403	9233.0987	Very High (VH)

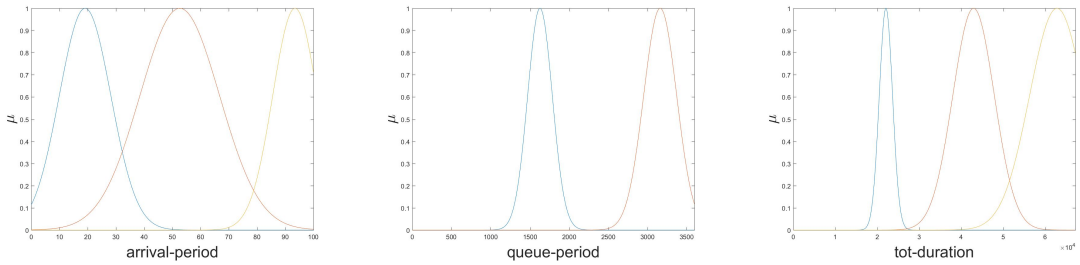


Figure 3.7: Gaussian fuzzy sets with ENORA.

has behaved better than NSGA-II to find an interpretable classifier, the best classifier is the Logistic one[4]. This is reasonable, as the problem of establishing the probability of a call being managed or not can intuitively be seen a logistic classification task. RandomForest also obtained a very high accuracy, better than that of ENORA-RULES, which was also to be expected, since usually black-box classifiers are more accurate than interpretable ones. Regarding the comparison with the other classifiers we can observe the following: J48/C4.5 obtained a precision of 0.995738 with a tree size of 255 nodes (128 leaves), while FURIA obtained a precision of 0.997104 with 106 fuzzy rules. Clearly both models are hardly interpretable since they convey an overwhelming amount of information. In conclusion, following our proposal we obtained a fuzzy rule-based classifier that, although reporting a slightly lower accuracy, has a low number of rules and linguistic labels, which makes it highly interpretable.

[4]The results are rounded to the 3rd decimal digit, and it appears to be 1.000 even if one instance is misclassified; the actual accuracy given by logistic regression is 0.999945.

Table 3.7: Performances of the fuzzy classifiers.

Classifier	Accuracy	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
ENORA-RULES	0.995	0.995	0.008	0.995	0.995	0.995	0.984	0.994	0.994
NSGA-II-RULES	0.848	0.848	0.722	0.830	0.848	0.803	0.267	0.563	0.752
J48	0.996	0.996	0.009	0.996	0.996	0.996	0.985	0.995	0.996
RandomForest	0.998	0.998	0.003	0.998	0.998	0.998	0.993	1.000	1.000
Logistic	1.000	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000
FURIA	0.997	0.997	0.006	0.997	0.997	0.997	0.990	0.998	0.998

3.1.5 Discussion

Wrapping up, in this work, we have tried to establish the main factors that determine whether an inbound call will be managed before being abandoned in a contact center, considering a real world dataset provided by Gap Srlu.

This has been done by comparing two different feature selection techniques (a wrapper technique based on the J48/C4.5 decision tree learner and a Correlation-based Feature Selector filter) driven by two different search strategies (the evolutionary algorithms ENORA and NSGA-II), along with two classical feature selection methods. The outcome of the experiment is that, in the considered domain, the wrapper methodology behaves better than the filter (both univariate and multivariate), and that the two search strategies behave in a comparable way. Then, after selecting the optimal subset of features (considering also domain expert validation), the behaviours of ENORA and NSGA-II in a fuzzy classifier learning setting have been compared. As a result, ENORA extracted a more interpretable, more accurate, and more useful (from the domain point of view) set of fuzzy rules. Such results have then been confronted with those provided by some classical classification algorithms. Specifically, RandomForest and Logistic classifiers obtained a slightly better accuracy than ENORA, although they are less interpretable by a domain expert, if at all. Moreover, the classifiers J48/C4.5 and FURIA have also generated decision trees and fuzzy rules respectively with too much information to be easily interpreted by a human, so we have considered them non-interpretable in this study.

We conclude that the classifier obtained following our proposal is highly interpretable and sufficiently accurate, offering a better trade-off between interpretability and accuracy than the rest of the classifiers analyzed in this study. From the semantical point of view, the rules show that the time spent during the IVR phase is important, and, in particular, that calls are more likely to be abandoned when the IVR phase is not too short nor too long. This can be interpreted as follows: if the IVR phase is short, then the call is managed quickly (which turns out to be in fact a characteristic of the Gap services considered in this dataset), while if it is particularly long, the caller has already "invested" so much time that is not willing to give up.

4

Working on Model Interpretability

This chapter focuses on model interpretability. Specifically, an approach to the post-pruning of decision trees based on *Evolutionary Algorithms* (EAs) [65] is presented, that can be used as a possible way of finding the best trade-off between the readability and accuracy of a model.

4.1 Evolutionary-based decision tree pruning

As it is commonly recognized, decision trees have a predominant position among classification models [331]. This is mainly due to the facts that *(i)* they can be trained and applied efficiently even on big datasets and *(ii)* they are easily interpretable. Thanks to the latter feature, they turn out to be useful not only for prediction, but also for highlighting relevant patterns or regularities in the data. This is extremely beneficial in those application domains where understanding the classification process is at least as important as the accuracy of the prediction itself.

A typical decision tree is constructed recursively, starting from the root, following the traditional *Top Down Induction of Decision Trees* (TDIDT) approach. A decision tree induced by the TDIDT approach tends to *overgrow*, and this leads to a loss in interpretability as well as to a risk of *overfitting* training data. In order to simplify the tree structure, thus making the trees more general, *pruning* methods are typically applied (for further details, see Section 1.2).

EAs have already been successfully applied to the various phases of the decision tree induction process (see, for instance, [42]). As for the decision tree pruning, the problem can actually be seen as a search problem in the space of possible subtrees [109], and EAs seem to be a natural solution to solve such a problem. Despite of that, to the best of our knowledge, the only proposed EA for classical, that is, not *oblique* [159], decision tree post-pruning is Chen and al.'s single-objective algorithm [77]. In that work, the fitness function is given by a weighted sum of the number of nodes in the tree and the error rate. Oddly enough, the latter is estimated directly on the same test dataset also used to evaluate the accuracy of the final solution. As pointed out in [42], this constitutes a serious methodological mistake, since the test set should only be used to assess the validity of the finally generated

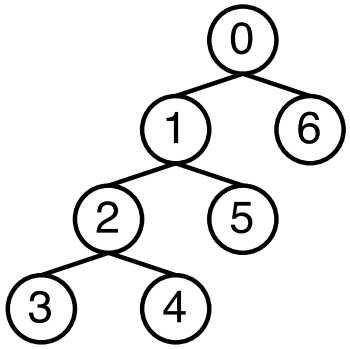


Figure 4.1: A maximum-height binary decision tree.

tree, and the training set, or an independent pruning set, should have been used in evaluating the fitness of individuals during EA computation.

This work corrects and extends the approach outlined in [77], making use of the well-known, elitist, multi-objective evolutionary algorithm NSGA-II [91]. A post-pruning strategy for J48 [274] is designed that seeks to optimize *two objectives*: the accuracy of the obtained tree (on the training dataset) and the number of its nodes. The novel approach is compared with the default post-pruning methodologies of both the algorithms J48 (*Error-Based Pruning*, see Section 1.2.1) and C5.0 [3]. In both cases (EA-based pruning and default pruning strategies), a third hold-out set is not necessary: this makes our comparison easier and, of course, it is advantageous for those cases in which training instances are scarce.

The work is organized as follows: Section 4.1.1 provides a sound justification for the evolutionary algorithm approach to the decision tree post-pruning problem. Section 4.1.2 presents the proposed approach in detail. Section 4.1.3 is devoted to the experimental analysis of the achieved solution. Finally, the obtained results are discussed in Section 4.1.4.

4.1.1 On the complexity of the decision tree pruning problem

Let us now focus our attention on the pruning problem viewed as a search problem. We are interested in establishing suitable lower and upper bounds to the search space and, to this end, we restrict our attention to *binary trees*, which makes it simpler to compute the bounds. More general lower and upper bounds can then be easily derived. Notice that binary decision trees are always *full*, that is, each node has zero or two children, and thus the pruning of a full binary decision tree, for any given internal node, either removes both subtrees or maintains both of them.

The search space consists of all the different pruned trees that can be obtained from a fully grown tree, that is, from the tree generated by the TDIDT recursive procedure.

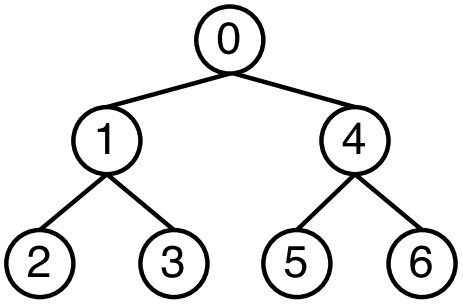


Figure 4.2: A balanced and complete binary decision tree.

Let n be the number of nodes of the given (fully grown) tree. A lower bound on the cardinality of the search space is given by the number of pruned trees that can be obtained from the highest full binary decision tree that can be generated with n nodes at our disposal. Consider the case with $n = 7$. The height of the highest full binary decision tree with 7 nodes is $h = 3$ (see Figure 4.1). The number of distinct pruned trees that can obtained from it is 4: the complete tree, the one obtained by deleting nodes 3 and 4, the one obtained by deleting nodes 2, 3, 4, and 5; and the one consisting of the root 0 only. In general, if the height of the highest full binary decision tree is h, the number of its distinct pruned trees is $h + 1$, i.e., $O(h)$.

An upper bound on the cardinality of the search space is given by the number of pruned trees that can be obtained from the *perfect* binary tree, whose levels are all complete (see Figure 4.2). In such a case, the number of pruned trees can be determined by a recursive formula, as precisely stated by the following proposition.

Proposition 1. *Let us consider a perfect binary decision tree of height h. The number of pruned trees that can be obtained from it is expressed by the following recursive equation:*

$$f(h) = \begin{cases} 1 & \textit{if } h{=}0\,; \\ f(h-1)^2 + 1 & \textit{otherwise}\,. \end{cases} \tag{4.1}$$

Proof. Let $h = 0$. There is only one possible perfect binary decision tree of height 0 which just consists of the root node. By definition, $f(0) = 1$ and indeed such a tree can generate only one pruned tree, that is, the root node itself.

Let $h > 0$. Since the tree is a perfect binary tree, it consists of a root node with exactly two perfect and binary subtrees, whose height is $h - 1$. By the inductive hypothesis, $f(h - 1)$ pruned trees can be obtained from each of the two subtrees. The total number of pruned trees that can be obtained from the entire tree is thus equal to the number of possible combinations of the subtrees' pruned trees, which is equal to $f(h - 1)^2$, plus the pruned tree consisting of its root only. The resulting value is then $f(h) = f(h - 1)^2 + 1$. □

An upper bound on the cardinality of the search space is thus $O(f(h)) = \Omega(2^h)$, which is a function that grows very fast. As an example, we have that:

$$\begin{aligned} f(0) &= 1 & f(4) &= 677 \\ f(1) &= 2 & f(5) &= 458.330 \\ f(2) &= 5 & f(6) &= 210.066.388.901 \\ f(3) &= 26 & f(7) &= 4,412788775 * 10^{22} \ . \end{aligned}$$

The above formula can be easily generalized to cope with non-perfect, non-binary, and, therefore, non-full, decision trees.

Let N be the root of the given tree, let $f'(N)$ be the function that computes the number of its pruned trees (we identify the tree with its root), and let $\mathcal{C}(N)$ be the set of all children of the node N. The function f', that generalizes f, can be defined as follows:

$$f'(N) = \begin{cases} 1, & \text{if } N \text{ is a leaf ;} \\ 1 + \displaystyle\prod_{M \in \mathcal{C}(N)} f'(M) & \text{otherwise .} \end{cases} \tag{4.2}$$

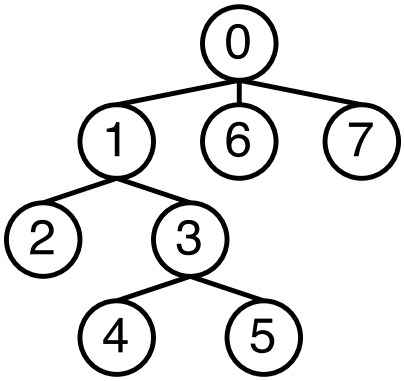
Clearly, both f and f' grow too fast to allow a systematic search of the space of all the pruned trees.

To see how the number of solutions may grow in real-world cases, consider, as an example, the decision tree generated by applying WEKA's J48 on the UCI's *Soybean* benchmark dataset (683 instances, 36 attributes [331]), disabling pruning and collapsing operations (`weka.classifiers.trees.J48 -O -U -M 2`). The resulting model has 207 nodes (141 leaves) and, according to the previous formula, it admits 131.165.197.804 possible pruned trees. For the sake of comparison, the default EBP pruned version has 93 nodes (61 leaves).

To conclude our analysis, it is worth mentioning that also optimal, linear computational complexity (with respect to the tree size) methods to perform post-pruning of decision trees do exist. Nevertheless, to the best of our knowledge, they all require the presence of a separate, hold-out set. An example is given by *Reduced Error Pruning* (REP) [273], where the additive property of the error rate for decision trees is exploited to design a bottom-up strategy to derive the smallest version of the most accurate subtree with respect to the hold-out set. In [110], the importance of having a large hold-out set is specifically remarked, to avoid the bias of REP towards overpruning.

4.1.2 Evolutionary algorithm-based pruning

This section presents a novel, *wrapper-based* approach to the pruning of decision trees. In wrapper-based pruning, a search algorithm explores the search space of all



(a) A full-grown decision tree with nodes labelled in a pre-order fashion.

0	1	2	3	4	5	6	7
1	1	0	1	0	0	0	0

(b) The corresponding gene, representing a solution.

Figure 4.3: A full-grown decision tree and the corresponding gene representation.

possible pruned trees that can be obtained from a single, unpruned and uncollapsed decision tree, and potential candidates are evaluated step-by-step.

The methodology is implemented on J48, the WEKA implementation of the algorithm C4.5 and with the evolutionary search algorithm known as NSGA-II (see Chapter 1) through the jMetal framework [102].

Representation of solutions and initial population

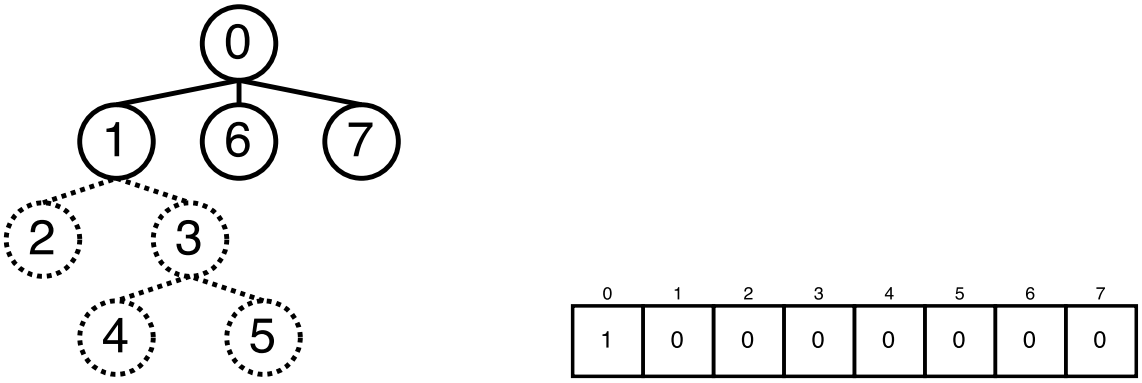
Given a training dataset, a fully grown, uncollapsed and unpruned J48 decision tree is first built.

Each solution to the search problem is conveniently represented as a binary array, whose size is equal to the number of nodes of the original J48 tree given as input. Each cell of the arrays tracks whether the subtree rooted at the specific node of the tree is being kept or not. In order to establish a correspondence between the tree and the array, nodes are numbered according to a pre-order visit of the tree.

As an example, the tree represented in Figure 4.3a corresponds to the binary array of Figure 4.3b. The decision tree obtained by removing the subtree rooted at node 1 and, consequently, the subtree rooted at node 3, is represented in Figure 4.4a. Its corresponding gene representation is shown in Figure 4.4b. Basically, each gene keeps track of the subtree status with respect to the original, fully grown decision tree.

The *initial population* has been generated with the following schema to ensure both its correctness and its heterogeneity.

A binary array corresponding to a particular solution is initially set to represent the fully grown tree. Then, a "non-pruning probability threshold" t is established (empirical evaluation suggested a random value in the range $[0.85, 0.95]$ for the threshold). For each cell of the array, the strategy proceeds as follows. The distance



(a) The pruned decision tree, with nodes labelled in a pre-order fashion.

(b) The gene representing the pruned tree.

Figure 4.4: The pruned decision tree and the corresponding gene representation.

d of the corresponding node from the root of the tree is computed (starting from 1), and, then, d random values are generated: if at least one of them is greater than t, the subtree corresponding to the array cell is pruned, that is, the cell is set to 0.

The idea behind such a generation strategy is that pruning at higher levels should be more difficult than pruning at lower levels, as pruning a shallow node removes most of the tree. Obviosly, pruning operations must be carried out in such a way that the resulting tree is a valid tree: if the subtree rooted at node N is removed, then all subtrees rooted at descendants of N must be removed as well.

Operators

Let us now consider the crossover and mutation operators, that in the evolutionary algorithm are used to generate new individuals.

Given two parent solutions, two children solutions are generated via *crossover* by simply performing a pairwise AND and a pairwise OR of the two corresponding binary arrays. Thus, one child will contain the subtrees which are in common between the two parents, whereas the other one will contain the union of the subtrees of the two parents. Correctness of the generated solutions straightforwardly follows from the correctness of the parents.

Given a binary array of length l corresponding to a solution, *mutation* is carried out as follows. A random number n, $1 \leq n \leq l$ is generated, and n random flips are carried out in the binary array, preserving the correctness of the generated solution: if the subtree rooted at N is removed, then all subtrees rooted at descendants of N must be removed as well, and if the subtree rooted at node N is restored, then all subtrees rooted at ancestors of N must be restored as well.

Fitness function

A bi-objective fitness function is employed. The first objective is to maximize the accuracy of the pruned tree on the training dataset and, to this end, we used the standard evaluation method provided by J48. The second objective is to minimize the size of the pruned tree and, for such a purpose, a simple function to count the number of nodes is used, also provided by WEKA. The two objectives are clearly antithetical: pruning a tree may possibly reduce the accuracy of it on the training set, but it never increases it.

Decision method

As it happens with any other multi-objective optimization algorithm, the result of our post-pruning method is a set of non dominated solutions.

To evaluate the quality of the proposed method, the solutions it returns are compared with the solutions provided, on the same training datasets, by C4.5 and by C5.0. Thus, an *a posteriori* decision-making method must be designed to select a single solution (or a subset of solutions) in a systematic and controllable way. The proposed strategy is inspired by the *Minimum Description Length* (MDL) principle (see, e.g., [233]), which is based on the assumption that any regularity in a given dataset can be exploited to compress the data.

Specifically, each non dominated tree can be considered as a *theory*, which can be used to "explain" the training data. Each theory will have a related (possibly empty) set of *exceptions*, which are not "captured" by the model. Thus, we may define the *coding cost* of a candidate solution as the coding cost of the theory (in the present case, the relative size of the tree with respect to the original one) plus the coding cost of the exceptions, that is, the error rate of the tree, calculated over the entire training set. Both values, denoted here by $SIZE$ and ER, respectively, belong to interval $[0, 1]$, and they can be arranged into a weighted sum as follows:

$$W * ER + (1 - W) * SIZE \ , \tag{4.3}$$

where $W \in [0, 1]$ is a weight which can be modified by the user in order to vary the pruning aggressiveness.

The candidate solution with the *lowest combined value* is selected. Intuitively, a large value of W tends to favor larger, but more accurate, models, as far as the training set is concerned, whereas smaller W values should result in more general trees being selected. Intentionally, the weight W plays a similar role as the confidence factor in C4.5 and C5.0, and this makes it possible to better compare the results.

4.1.3 Experimental task

This section provides an experimental comparison among three post-pruning approaches, that is, the standard C4.5 and C5.0 EBP pruning and our wrapper-based

Table 4.1: The UCI datasets under study.

Dataset	# inst	# attrs	Predictors type	# nodes unpr
Adult	48842	14	nominal, numerical	31145
Bank	41188	20	numerical	6820
Breast W	699	9	numerical	53
Credit Card	30000	23	numerical	5613
Credit G	1000	20	nominal, numerical	433
Diabetes	768	8	numerical	69
Eye State	14980	14	numerical	1579
Labor	57	16	nominal, numerical	16
Sonar	208	60	numerical	29
Spambase	4601	57	numerical	361
Voting	435	16	numerical	51
Waveform 5k	5000	40	numerical	555

EA for post-pruning. All experiments have been carried out on an Intel Core i5 processor running at 2.4 GHz, equipped with a main memory of 8 GB.

Datasets

Experiments are based on 12 standard UCI datasets[1], which have been selected in order to maximize the variability in terms of number of instances as well as number and types of attributes.

As witnessed by the recent literature, UCI datasets are a standard *de facto* in the machine learning community (see, e.g., [77, 110, 115, 152]). The chosen datasets are detailed in Table 4.1, where, for each case, we show the number of nodes in the respective uncollapsed and unpruned J48 decision tree (*# nodes unpr*).

Methods

The experimental phase has been designed as follows. Each dataset has been partitioned into a training set (75%) and a test set (25%), according to a stratified approach. Then, on each dataset, the three methods, namely, J48, C5.0, and EA-based, for post-pruning have been applied.

As for C4.5 and C5.0, the followed approach is similar to the one adopted in [152]. In particular, a set of 27 decision trees has been trained for each dataset by varying

[1]https://archive.ics.uci.edu/ml/datasets.html

Table 4.2: Number of evaluations and computation time per dataset.

Dataset	# evaluations	Computation time (sec)
Adult	75000	3200
Bank	100000	720
Breast W	10000	< 1
Credit Card	60000	400
Credit G	100000	46
Diabetes	10000	1
Eye State	100000	188
Labor	10000	< 1
Sonar	10000	< 1
Spambase	100000	50
Voting	10000	1
Waveform 5k	60000	37

the confidence factor over in the range $[0.001, 0.49]$ (this is the widest implemented range). The tested values range from 0.1 to 0.49 (included) by steps of 0.02, other than the very low values 0.005 and 0.001 — the lower the confidence factor, the more aggressive the pruning performed. For the sake of comparison, recall that WEKA's default confidence factor is 0.25.

Collapsing option has also been activated. Moreover, in the case of C5.0, trees have been generated non-bagged, winnowing has been disabled, while global pruning and soft thresholds remained active.

On the other hand, the EA-based experiment has been designed as follows. On each dataset 30 independent runs have been executed, each with a different seed value. This led to 30 sets of non dominated solutions which have been merged into a final, single set (from which all dominated solutions have been eliminated). The population size has been set to 100, while crossover and mutation probabilities have been set to 0.9 and 0.2, respectively.

Moreover, a series of experiments has been performed for each dataset to empirically assess the minimum amount of evaluations, in the range $[10000, 100000]$, that are necessary to reach a satisfactory solution (given the population size of 100, 10000 evaluations correspond to 100 evolution steps). Table 4.2 summarizes the number of evaluations, and the required computation time, for each independent run over the considered datasets.

Results

For each dataset, the results of the three post-pruning methods have been compared. The results are shown in a number of figures (from Figure 4.5 to Figure 4.16). Each graph shows the relation that emerges between the predictive accuracy (on the test set) and the size (number of leaves) of each tree produced by a specific setting of the parameter that governs the pruning aggressiveness (confidence interval for C4.5 and C5.0, weight W for the EA-based wrapper), generating three curves for each dataset. Each point in the graphs represents a model.

Even if the focus was not on learning more precise trees than classical methods, in 7 out of 12 cases, the EA-based pruning produced at least one tree with equal or better predictive accuracy than the best tree produced by C4.5 or by C5.0, regardless its size. These are: *Breast W*, *Credit Card*, *Diabetes*, *Labor*, *Spambase*, *Sonar*, and *Voting*.

In the case of *Breast W* (Figure 4.7) the EA-based pruning generated a tree with 20 leaves and 97.7% accuracy (best overall accuracy), and the smallest tree generated by classical methods has 5 leaves and an accuracy of 94.2%, while the wrapper has been able to produce a tree with only 2 leaves and an accuracy of 93.7%.

In the case of the dataset *Credit Card* (Figure 4.8), a tree with 24 leaves and 82.2% accuracy (best overall accuracy) has been produced, as well as the same smallest result as C5.0 (2 leaves, 81.7% accuracy), and the evolutionary approach also surpassed the smallest tree generated by J48 (7 leaves and 81.8% accuracy).

As for *Diabetes* (Figure 4.10), the EA-based pruning matches the best result obtained by J48 (3 leaves and 75.5% accuracy), and proposes a smaller but almost as much as accurate tree (2 leaves and 75.0% accuracy), while C5.0 is capable of achieving the same result but with a (slightly) bigger tree (3 leaves, 75.0% accuracy).

In the case of *Labor* (Figure 4.12), J48, C5.0 and the EA-based wrapper are all capable of generating the best tree (2 leaves and 85.7% accuracy), while in the case of *Sonar* (see Figure 4.13), the wrapper constantly surpasses J48 in terms of classification performance, and it is also capable of achieving the same accuracy result of C5.0 (78.4%) with a slightly smaller tree (9 vs 11 leaves).

As for *Spambase* (Figure 4.14), we may observe that the results obtained by the three approaches tend to be quite similar; however, the wrapper is capable of generating the best tree (141 leaves and 92.7% accuracy) as well as a very small (smaller than those proposed by J48 and C5.0) yet accurate enough tree (12 leaves, 89.6% accuracy).

Finally, in the case of *Voting* (Figure 4.15), the three methods all produce the best (and smallest) tree (2 leaves and 96.3% accuracy).

In the case of the dataset *Adult* (Figure 4.5), the best accuracy is obtained by C5.0; however, while the most accurate model generated by J48 has 144 leaves and 56.7% accuracy, its smallest tree has 17 leaves and 56.5% accuracy. In comparison, the wrapper produced a pruned tree with 10 leaves, while still retaining an accuracy

of 56.1%. Given the exceptionally big unpruned tree, probably, a better result would be obtained with more EA evaluations.

As for the dataset *Bank* (Figure 4.6), even if the wrapper is surpassed by both J48 and C5.0 in terms of accuracy, it is still capable of generating a pruned tree with 7 leaves, while still retaining a test set accuracy of 91.1%, and a tree with only 3 leaves, keeping an accuracy of 90.1%. This contrasts with the smallest tree generated by J48 (25 leaves and 91.5% accuracy) and the one produced by C.50 (13 leaves and 91.2% accuracy.

In the case of *Eye State* (Figure 4.11), while the wrapper does not reach the same accuracy as C4.5 and C5.0, it is able of providing a variety of much smaller trees, without loosing too much accuracy.

Finally, in the case of *Credit G* (Figure 4.9), the wrapper achieved an overall higher accuracy than J48 (74.0% vs 73.6%), but lower than C5.0 (74.8%), while in the case of *Waveform 5k* (Figure 4.16) the wrapper produced more accurate trees than J48, and, while the most accurate model is generated by C5.0 (120 leaves and 77.9% accuracy), the wrapper generated two smaller, yet accurate enough, models (one with 76 leaves and 77.7% accuracy and the other with 39 leaves and 75.4% accuracy).

4.1.4 Discussion

Results reported in Section 4.1.3 clearly show that the proposed EA approach to pruning decision trees is capable of matching, and sometimes surpassing, the performances of classical C4.5 and C5.0 strategies in terms of the size-to-accuracy ratio of the generated trees. In particular, the proposed method is capable of producing a more variegate set of solutions, often characterized by smaller trees, which, nevertheless, preserve most of the accuracy of those traditionally pruned.

The proposed EA-based pruning approach starts from a full-grown C4.5 decision tree. Given that, as reported in [7], the trees grown by C5.0 tend to behave better than those grown by C4.5, we may speculate that better results could be achieved by applying the EA method to the former instead of the latter. Moreover, there are several EA-related aspects that could be taken into account to enhance our results.

First, the classical selection strategy implemented in NSGA-II has been improved in, for instance, the algorithm ENORA [178, 179]. Second, independently from the selection strategy, state-of-the-art implementations of EAs do not require explicit and fixed setting of the crossover and mutation rates, which are, instead, considered as characteristics of adaptiveness of each of the solutions. Empirically, adaptation has been observed to better the performance of traditional operators in terms of convergence to the Pareto optimal front and in diversity of the final solutions. Finally, in the last few years, the use of fitness functions is being progressively substituted by convergence directly based on the hypervolume, which seems to behave better than traditional fitness-based methods [168], and it would be interesting to understand

its effects on our method as well. Overall, this entire work should be considered as a worthy proof-of-concept of the idea of EA-based post-pruning, which in our opinion deserves some further investigation. Previous to this work, to the best of our knowledge, only Chen et al. [77] approach to (non-oblique) decision tree pruning has used evolutionary algorithms. Their method and the present one differ from each other in various nontrivial aspects. First of all, Chen's pruning is applied to a fully-grown ID3 decision tree, which is a predecessor of C4.5 [272]. Concerning the representation of the solutions, the hereby presented encoding is similar to theirs, except for the fact that it denotes nodes, that is, subtrees, instead of edges. On the contrary, the proposed crossover operator is very different from the one of [77], where its application partitions each of the two parents into a prefix and suffix substrings (then, to generate the offspring, the prefix of the first parent is combined with the suffix of the second parent, and vice versa). Moreover, the present mutation strategy is more aggressive than Chen's, where a single random bit flip is performed: empirically, allowing to perform more than one flip per individual has proven to help avoiding local optima convergence. Additionally, in [77], the single-objective fitness function is evaluated directly on the test dataset. As pointed out in [42], this is a serious methodological mistake, since such data should be kept aside for validation purposes, and not being used for tuning the algorithm. Finally, UCI datasets were also used in [77], although the authors have relied on just 4 datasets, making the present experimental setting more comprehensive and variegate.

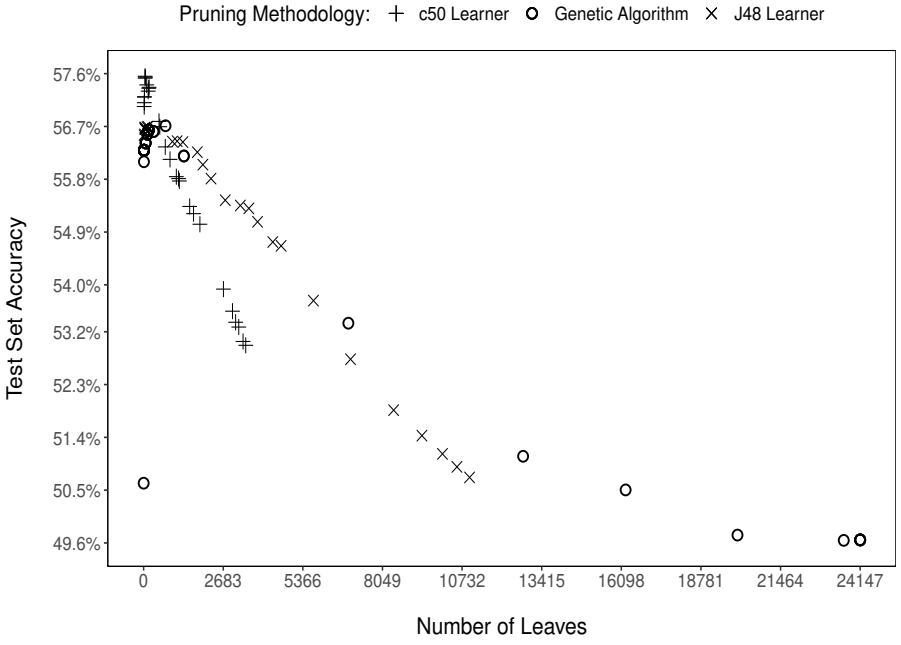


Figure 4.5: Results on the Adult dataset.

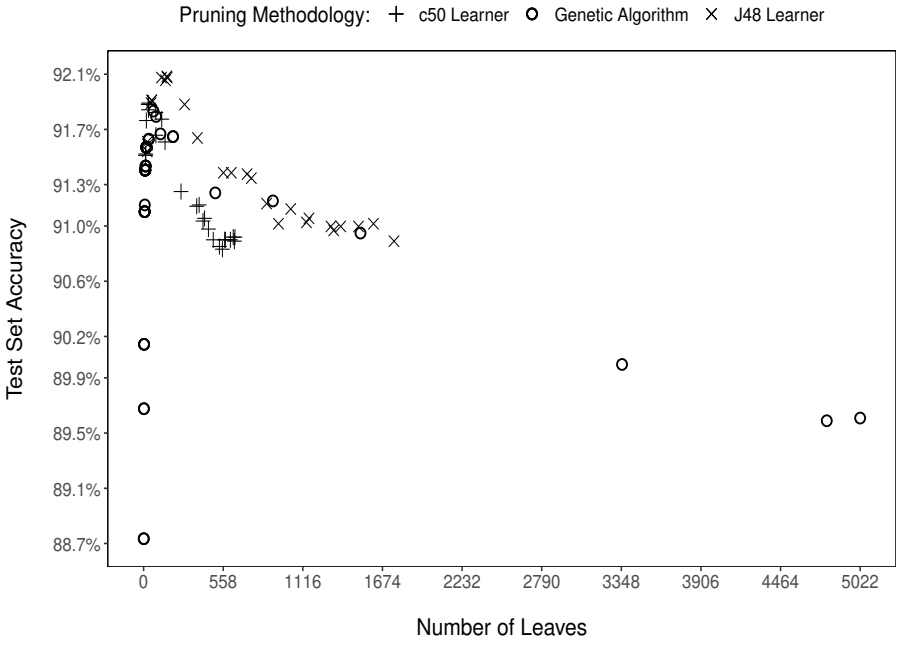


Figure 4.6: Results on the Bank dataset.

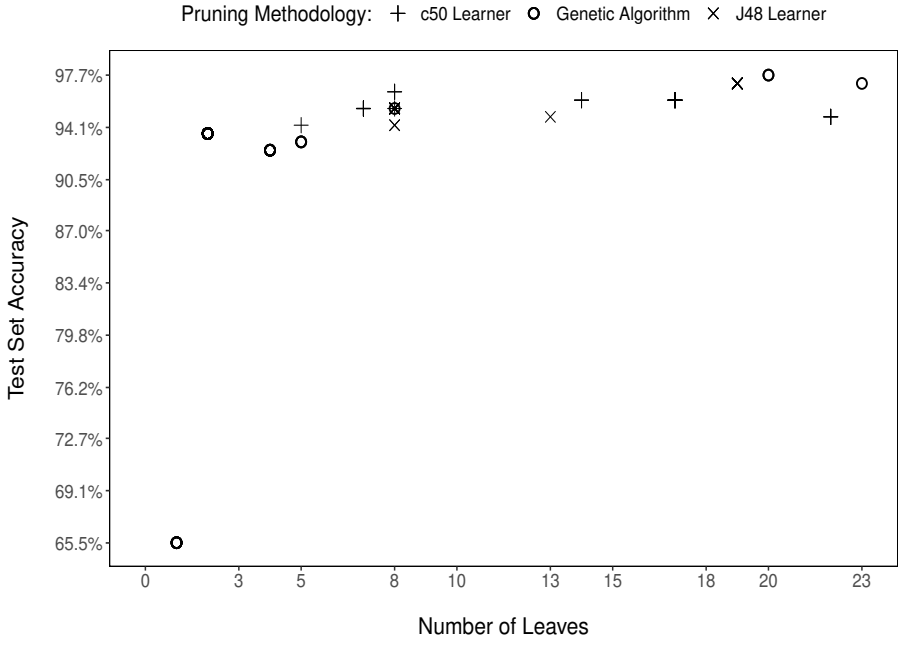


Figure 4.7: Results on the Breast W dataset.

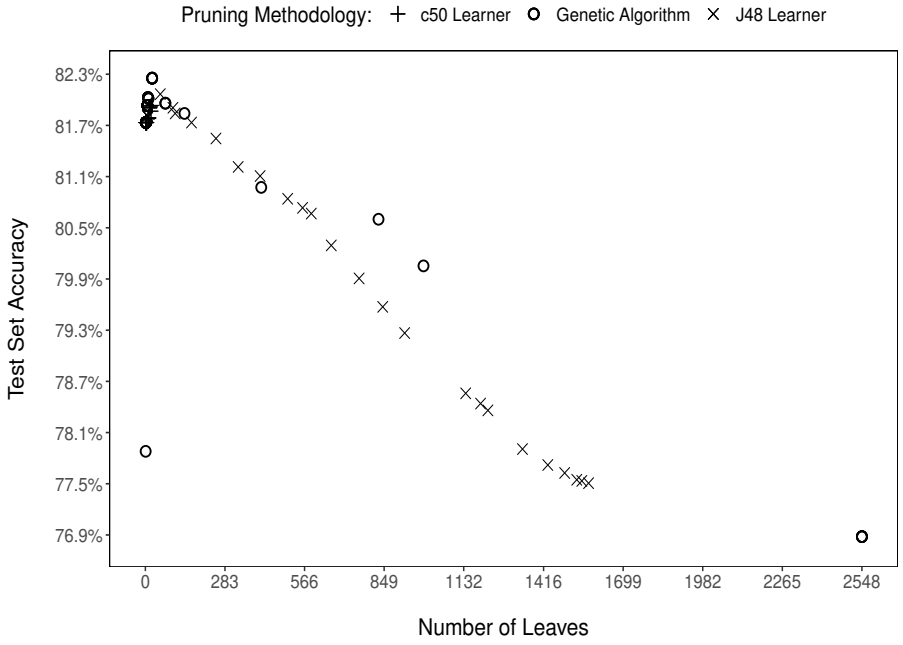


Figure 4.8: Results on the Credit Card dataset.

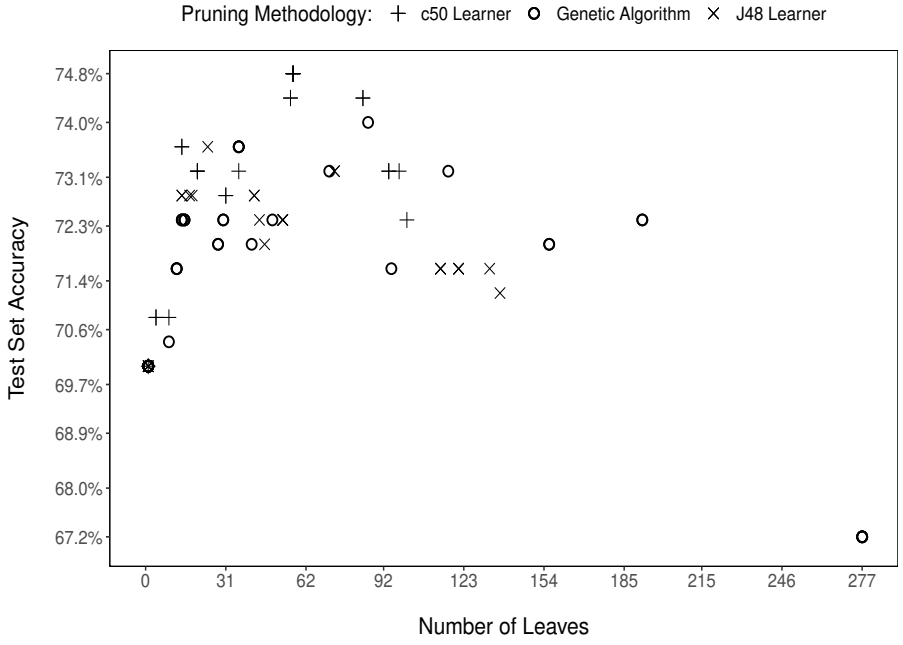


Figure 4.9: Results on the Credit G dataset.

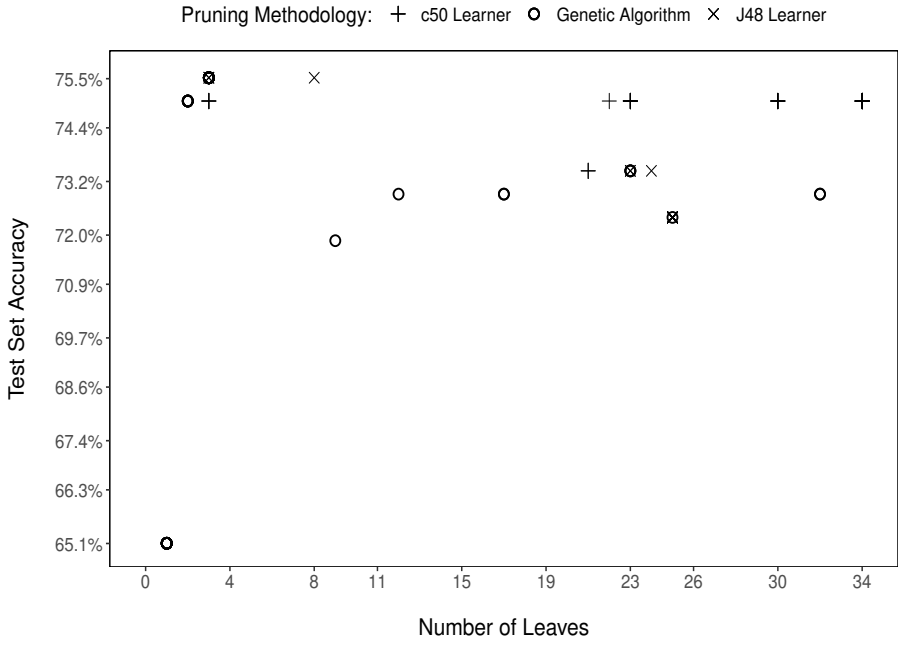


Figure 4.10: Results on the Diabetes dataset.

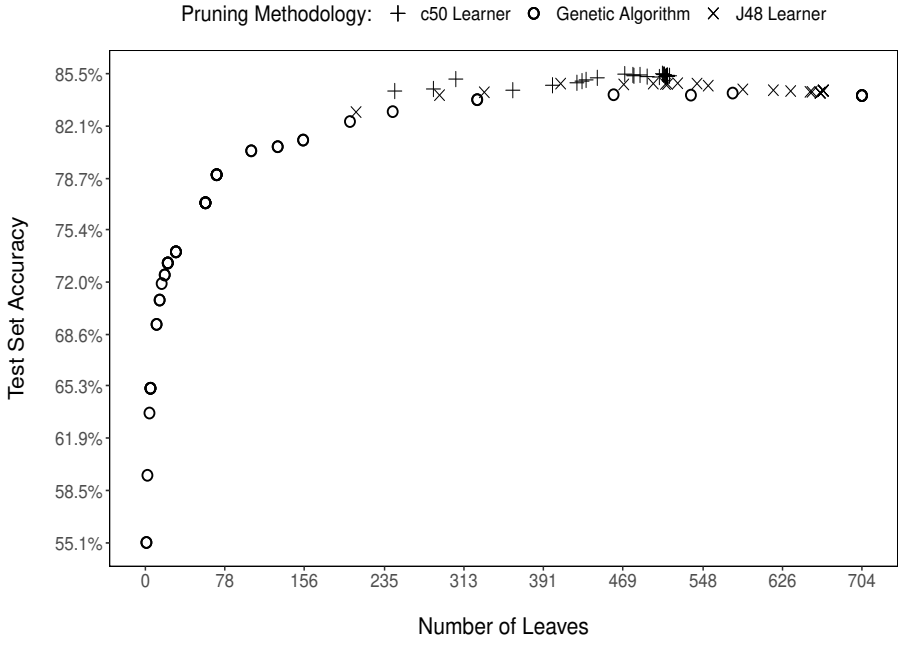


Figure 4.11: Results on the Eye State dataset.

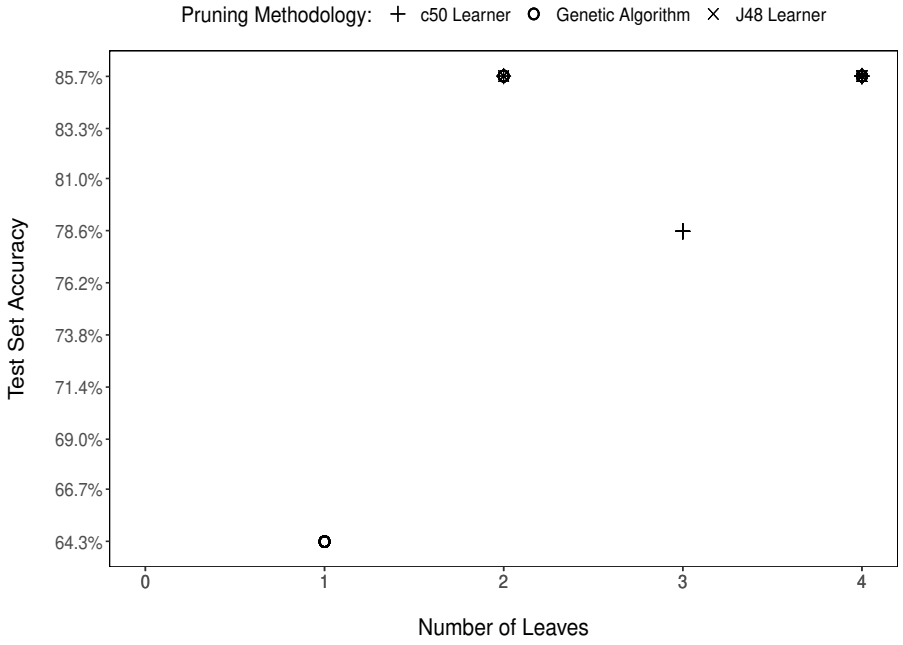


Figure 4.12: Results on the Labor dataset.

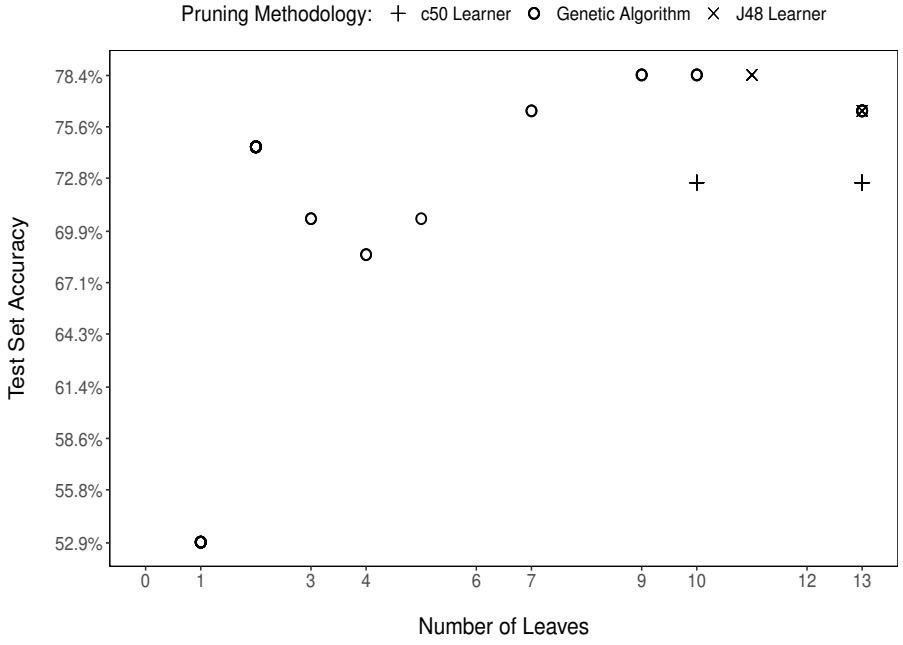


Figure 4.13: Results on the Sonar dataset.

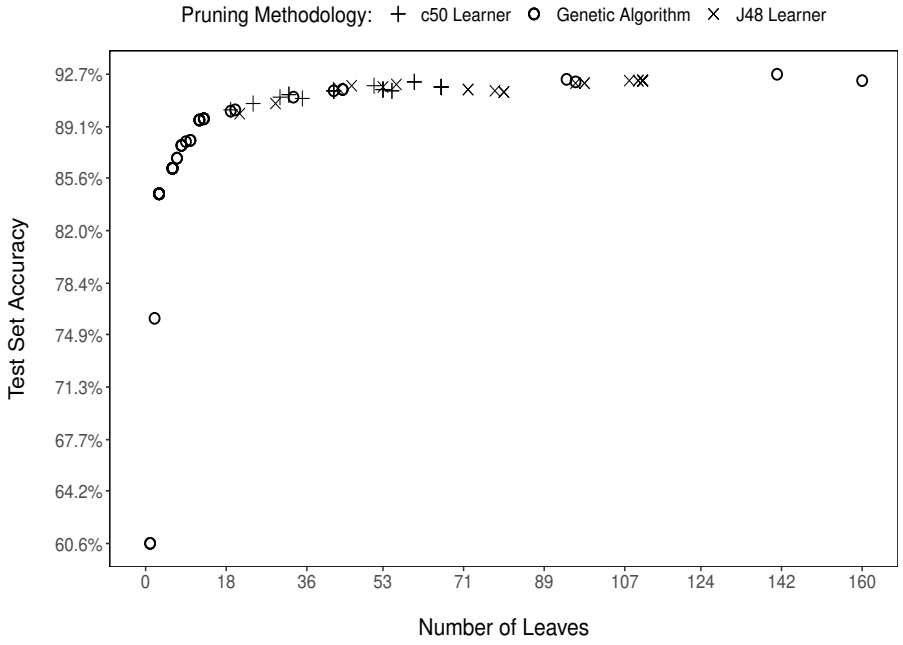


Figure 4.14: Results on the Spambase dataset.

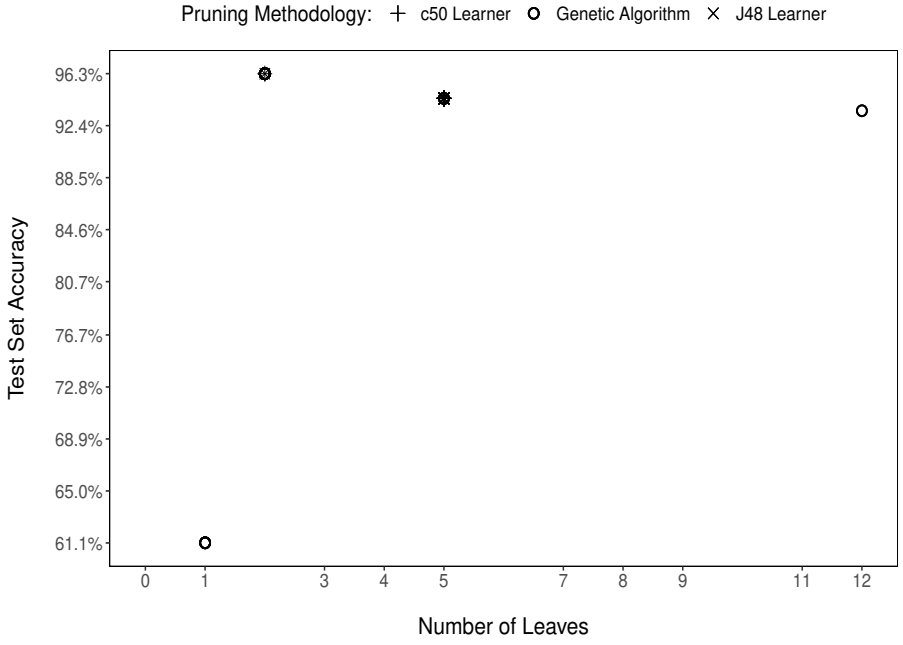


Figure 4.15: Results on the Voting dataset.

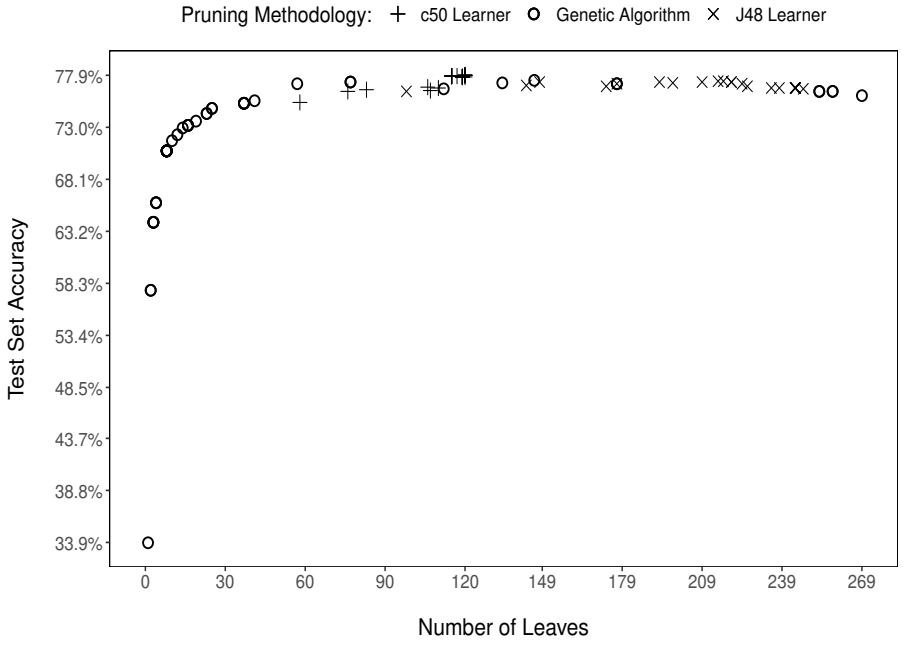


Figure 4.16: Results on the Waveform 5k dataset.

5

Working with Sequential Data

In the present chapter, we focus on the analysis of sequential data. Specifically, Section 5.1 presents and motivates the development of J48S [66], which is an extension to J48, the WEKA's Java implementation of C4.5 [125] decision tree inducing algorithm. Then, based on the fact that text can be considered as a kind of sequential information where single words are items that make up a sequence, Section 5.2 describes a task dealing with transcription-based phone call classification in contact centers, where J48S has been profitably employed [68].

5.1 J48S: a novel decision tree approach for the handling of sequential data

This section presents a novel decision tree model based on the well-known inducer C4.5 [273] and the algorithm VGEN [122] for the extraction of frequent patterns (for an introduction to both tools, see Sections 1.2.1 and 1.1.1, respectively). The proposed algorithm is able to mix the use of categorical, numerical, and sequential attributes during the same execution cycle, leading to several benefits, most importantly: *(i)* a simplification of the data preparation phase, especially in the preprocessing of sequential (such as textual) attributes, and *(ii)* a very high interpretability of the generated models.

As we shall see in Section 5.2, the proposed solution has been successfully applied to a text classification task in the context of a real business case, with the aim of detecting problematic phone calls. This should not come as a surprise, since call conversation transcripts may be regarded as (a kind of) sequential data, where sequences are made of words.

In the next two sections, all the modifications that have been made to integrate VGEN and J48S are thorougly described. Then, Section 5.1.3 briefly recaps the main characteristics of the novel algorithm, and provides a solid justification for its development.

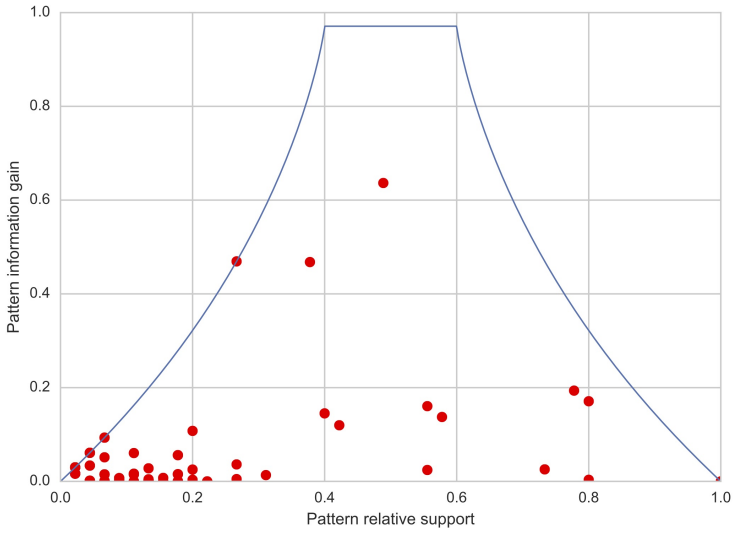


Figure 5.1: Pattern information gain and its theoretical upper bound with respect to the support on a randomly generated dataset.

5.1.1 Adapting VGEN

As discussed in Section 1.1.1, VGEN produces frequent generator patterns in a top-down fashion, until a user-defined minimum support threshold is reached. Since the goal here is not that of obtaining frequent patterns, but rather discriminative patterns that may help in the classification process, the *information gain* criterion has been integrated into the algorithm. For the sake of simplicity, let us consider the two-class classification problem; the concepts can be easily extended to multi-class classification by means of an one-vs-all approach.

First, note that the information gain of a pattern may be calculated by partitioning the instances in the dataset into those that satisfy it, and those which do not. As observed in [79], the information gain can be considered as a function of the support of the pattern: patterns having a very high or very low support are typically not very informative, the former being simply too common, and the latter having a too limited coverage in the dataset. On the basis of these assumptions, in [79], a strategy to derive an upper bound to the information gain based on the support of the pattern is presented. The behavior of such an upper bound is illustrated as the solid curve in Figure 5.1, which considers data taken from a randomly generated binary class dataset: as expected, from left to right, the upper bound increases as long as the support increases, until it peaks, and then starts decreasing as the relative support reaches 1. The graph also shows the information gain and support of each pattern extracted by VGEN on the same dataset (red dots). Thus, given the upper bound calculation strategy, it is possible to provide a stopping criterion

for the pattern growing phase (remember that, as a pattern grows, its support may only decrease). During the pattern generation process a dedicated variable stores the best information gain found so far. Then, given a pattern, one may decide if it is worth growing as follows: if the support of the pattern corresponds to a point in the curve where the derivative is positive or zero, then growing it may only decrease its upper bound or leave it unchanged. In such a situation, if the best information gain found so far is already greater or equal to the information gain upper bound of the pattern, its growing process may be stopped. This strategy constitutes a second stopping condition, which is employed together with the minimum support threshold method.

VGEN has also been modified with respect to its output. As we shall see, VGEN is called at each node during the tree construction phase to obtain a candidate pattern to guide the split operation. So, we are not interested in obtaining a set of patterns (as returned by the standard VGEN algorithm), but rather in the single most useful pattern at that stage. Note, however, that sometimes we cannot just take the most informative one, as it might be too complex and thus overfit training data. Hence, we proceed as follows. The best pattern (in terms of information gain) for each encountered pattern size is stored. Then, at the end of the algorithm, the following formula (again, inspired by the MDL principle [285]), is evaluated for each of the stored patterns:

$$W * (1 - pattern_{rIG}) + (1 - W) * pattern_{rlen} \tag{5.1}$$

where $W \in [0, 1]$ is a weight that can be customized by the user, $pattern_{rIG} \in [0, 1]$ is the relative information gain of the pattern with respect to the highest information gain observed, and $pattern_{rlen} \in [0, 1]$ is the relative length of the pattern, with respect to the longest pattern that has been discovered (in number of items). Finally, the pattern the minimizes the value of such formula is selected. Thus, intuitively, the larger the user sets the weight, the longer and more accurate (on the training set) the extracted pattern will be. On the contrary, selecting a low weight should lead to a shorter, and more general pattern being selected. In essence, setting W allows one to control the level of abstraction of the extracted patterns.

5.1.2 Adapting J48 to handle sequential data

The decision tree J48 is capable of handling both categorical and numerical attributes. Its extension J48S adds the possibility of managing also sequences, which are represented as properly formatted strings. The design is inspired by [112], in which the authors propose a decision tree construction strategy for sequential data. The present work improves [112] mainly over three aspects: *(i)* it relies on a well-established algorithm such as J48, instead of designing a new one; *(ii)* the novel decision tree is capable of mixing the usage of sequential and classical (i.e., categorical or numerical) data during the same execution cycle; *(iii)* the proposed implementation allows the tuning of the abstraction level of the extracted patterns.

The first modification concerns the splitting criterion. Instead of relying on the gain ratio, the so-called *normalized gain* [186] is used, which is defined as:

$$NormIG(X,T) = \frac{Gain(X,T)}{log_2 n},\ n \geq 2 \tag{5.2}$$

where n is the split arity. Note that this allows for a seamless integration between the decision tree learner and the modified VGEN algorithm, since the normalized gain is exactly equal to the information gain for binary splits, as in the case of a pattern-based split (recall that the criterion is the presence or absence of the pattern in the instance). At each node of the tree, the learner determines the most informative attribute among the categorical and numerical ones. Then, it calls VGEN along with two fundamental parameters: *(i)* the user-defined weight for controlling the abstraction level of the extracted pattern, W; and *(ii)* the information gain of the best attribute found so far, so that it can be used to prune the pattern search space as discussed in 5.1.1. Once VGEN exits, if a pattern is returned, its normalized gain is compared with the best previously found. If it is better, then, a binary split is created on the basis of the presence or absence of the pattern. Otherwise, the best numerical or categorical attribute is used.

Algorithm 2 describes the main recursive procedure used in the tree building phase, that is in charge of determining the split of an internal node, or of labelling it as a leaf. It takes in input a node (initially, the root node of the tree, in which all instances reside), and proceeds as follows. If the given node is pure (i.e., all instances have the same label) or another stopping criteria is met (e.g., based on a minimum cardinality constraint), then the node is transformed into a leaf of the tree. Otherwise, the algorithm determines the best attribute for the split (*best_attr*, together with its normalized gain value, *best_ng*) in the following way. First, it evaluates all static attributes (categorical and numerical), determining their *normalized gain* value (variable *a_ng*), following the default J48/C4.5 strategy (see Section 1.2.1). Then, it focuses on string attributes that encode sequences: the best *closed frequent pattern* is extracted for each attribute (*pat*, together with its normalized gain value, *pat_ng*), calling VGEN algorithm. Observe that the sequential pattern mining algorithm is intentionally called after determining the best normalized gain among static attributes, so as to use such a value for search space pruning purposes. Finally, the attribute with the maximum normalized gain is selected to split the node, and the algorithm is recursively called on each of the children.

Table 5.1 lists the six parameters that have been added to J48S, with respect to the original J48. Observe, in particular, the role of *maxGap*: it allows to tolerate gaps between itemsets which, in a text classification context, means that it is possible to extract and apply n-grams capable of tolerating the presence of some noisy or irrelevant words. Note that when *maxGap* is set to values larger than 1, it greatly affects the computation time of the pattern extraction phase: in general the higher its value, the slower the algorithm. As an example, Figure 5.2 presents an archetypal

Algorithm 2 Node splitting procedure (J48S)

```
1: procedure NODE_SPLIT(NODE)
2:     if NODE is "pure" or other stopping criteria met then
3:         make NODE a leaf node
4:     else
5:         best_attr ← null
6:         best_ng ← 0
7:         for each numerical or categorical attribute a do
8:             a_ng ←get information gain of a
9:             if a_ng > best_ng then
10:                best_ng ← a_ng
11:                best_attr ← a
12:        for each sequential string attribute s do
13:            pat, pat_ng ←get best frequent pattern in s
14:            if pat_ng > best_ng then
15:                best_ng ← pat_ng
16:                best_attr ← pat
17:        children_nodes ← split instances in NODE on best_attr
18:        for each child_node in children_nodes do
19:            call NODE_SPLIT(child_node)
20:            attach child_node to NODE
21:    return NODE
```

J48S decision tree, built by integrating the extension to WEKA data mining suite [331]. The reference dataset is characterized by three features: *sequence_attribute*, which is of type string and represents a sequence of itemsets; *attribute_numeric* which has an integer value; *attribute_nominal*, which may only take one out of a predefined set of values. The class is binary, and has two labels: *class_0* and *class_1*. The first test made by the tree on a given instance is whether its sequential attribute contains the pattern $(A, B) > D$ or not, meaning that there should be an itemset containing A, B, followed (within the maximum gap constraint) by an itemset containing D. If this is the case, the instance is labelled with *class_0*, otherwise the tree proceeds by testing on the numerical and nominal attributes.

5.1.3 Discussion

As discussed in the Introduction, sequences play a major role in the extraction of information from data, and may be sometimes complemented by other, "static" kinds of data, which can be numerical or categorical. Unfortunately, different kinds of data typically require different kinds of preprocessing techniques and classification algorithms to be managed properly, which usually means that the heterogeneity of data

Table 5.1: Custom parameters in J48S.

Parameter name	Default	Description
maxGap	2	max gap between itemsets (1 = contiguous)
maxPatternLength	20	max length of a pattern, in number of items
maxTime	30	max running time of the algorithm, per call
minSupport	0.5	min support of a pattern
patternWeight	0.5	weight used in VGEN for the result extraction
useIGPruning	True	activate pruning of the pattern search space

```
sequence_attribute !contains (A,B)>D
|   attribute_numeric <= 20: class_1
|   attribute_ numeric > 20
|   |   attribute_nominal = value_1: class_0
|   |   attribute_nominal = value_2: class_1
sequence_ attribute contains (A,B)>D: class_0
```

Figure 5.2: A typical J48S decision tree built by means of WEKA.

and the complexity of the related analysis tasks are directly proportional. Moreover, since multiple algorithms have to be combined to produce a final classification, the final model may probably lack in interpretability. This is a fundamental problem in domains in which understanding and validating the classification process is as important as the accuracy of the classification itself, such as in the case of production business systems or life critical medical applications. As previously discussed, the key characteristic of J48S is that it is capable of managing static (categorical and numerical) as well as sequential data during the same execution cycle, meaning that a single model may be used for the whole analysis process. Moreover, as we shall also see in the application scenario presented in Section 5.2, the resulting decision tree models are intuitively interpretable, so that a domain expert may easily read and validate them.

5.2 A combined approach to speech conversation analysis in a contact center

Nowadays, more and more companies strive to extract relevant knowledge regarding their business. Although such data may in fact be an important source of strategic information, it is sometimes stored in an unstructured and hard to exploit form as, for instance, in the case of data generated from text or audio flows. Specifically, the ability to analyze conversational data plays a major role in contact centers, where the core part of the business still focuses on the management of oral interactions [288].

The analysis of texts originating from oral conversations has to face two main challenges: (*i*) the intrinsic characteristics of conversational dynamics, which are different from those of written texts, and (*ii*) the presence of a transcription process, which introduces a bias between the spoken interaction and its written form. The combined effect of these two factors asks for a specific treatment, different from the ones exploited for written compositions.

In this section, we focus on the specific case of a real company, to root our work on actual issues emerging from practice, but the solutions we propose are of general validity.

We analyze a dataset of recorded anonymized agent-side call conversations produced by a wide range survey campaign made on a large part of the Italian population by Gap Srlu company which, as we have already mentioned, is an Italian business process outsourcer specialized in contact center services. Such an analysis is extremely valuable to the company, as it may be easily integrated in a broader, existing decision support system as the one we presented in Chapter 2. It should be noted that, to the best of our knowledge, this work is one of the few to tackle with speech analytics applied to a real business case, in Italian language.

The ultimate goal of the work is the introduction of a reliable speech analytics solution having a real operational impact on company processes, such as, for instance, in assessing the overall training level of human employees. We specifically aim at determining whether it is possible to develop a full-fledged speech analytics system at a relative low industrial cost by relying on interpretable models only, that can be validated by a domain expert before being put into production. As we shall see, this led us to neglect two approaches that are commonly used in natural language processing, i.e., neural networks and Conditional Random Fields (CRFs).

In order to develop the system, the first step is that of assessing the feasibility of building a proprietary transcription model trained on a relatively small dataset. The comparison with a commercial cloud transcription solution is then mandatory in order to evaluate its costs and performances.

The second main challenge concerns the tagging of conversation transcripts. We investigate potentialities and limitations of a machine learning solution. More precisely, we try to establish whether an approach based on machine learning may be

reliable enough, and more efficient than a classical linguistic solution based on domain expert-defined regular expressions. As a by-product, we assess the effectiveness of a combined approach to tagging that mixes the two.

Finally, the impact on the text analysis process of a solution based on J48S decision tree induction algorithm (see Section 5.1) is evaluated. The outcome of such an evaluation is that the proposed solution has at least two major key benefits: the simplification of the data preparation phase and the high interpretability of the final model which, as we have already mentioned, is one of our main goals.

The rest of the section is organized as follows. Section 5.2.1 outlines the general framework of our speech analytics process. It first gives a short account of related work on the analysis of call interactions in contact centers, and then it describes the specific speech analytics task which we focus on, that is, the semantic tagging of conversations recorded in the context of a specific outbound survey. Section 5.2.2 deals with the first part of the analytics workflow, namely, the transcription of phone conversations. Section 5.2.3 focuses on the second phase of the analysis, which consists of the semantic tagging of telephone conversation transcripts. Section 5.2.4 focuses on J48S which, as he have seen in Section 5.1, handles sequential data, such as textual data, for classification purposes. Section 5.2.5 discusses the operational impact of the proposed speech analytics solution. Finally, we provide an overall assessment of the work done.

5.2.1 The general framework

In this section, we first give a short account of related work on the analysis of speech conversations in a contact center, and then we outline the general structure of the solution that we propose.

A short account of related work

As we have already pointed out, speech analytics may play an important role in the domain of contact centers. Among the benefits that its use can lead, we would like to mention the possibility to exploit it to distinguish between well-behaved and problematic calls.

Existing solutions can be roughly partitioned into those that operate directly on raw audio data and those that turn them into textual data.

An analysis directly based on raw audio features has been proposed in [39], where the authors look for specific sound patterns which can be interpreted as, for instance, turn taking, hesitation, or voice overlap. Similarly, in [259], the authors rely on patterns based on the so-called *speaking rate*, which is expressed as the number of spoken words per minute. Both approaches take into consideration notable beahavioural patterns, emerging from an acoustic analysis, with the goal of obtaining a classification of conversations.

As an alternative, the analysis may rely on textual information. This is the case with [73, 346]. These approaches consist of two fundamental steps. The first step generates the textual data from the raw audio files by making use of an automatic transcription process. Then, text mining techniques are applied to the resulting texts with the final objective of determining the overall quality of the interaction.

A more advanced approach is proposed in [135, 237], which makes use of advanced natural language processing (NLP) techniques, such as *noun group*, *named entity recognition*, and *divergence of corpus statistics*, to analyze the obtained transcriptions in the context of a broader architecture for conversation analysis.

In [45], the DECODA project is illustrated, which aims at facilitating the development of robust speech data mining tools in the context of contact center evaluation and monitoring. The main contribution is the proposal of a French language corpus originated from the Parisian public transportation call center, which ought to reduce the development cost of speech analytics systems by limiting the need for manual data annotation. Based on such a corpus, several studies have been conducted, including theme identification [111, 241, 243, 261], dialogue classification [196, 242], named entity and semantic concept extraction [138], and speech summarization [314].

Nevertheless, as we have already mentioned, to the best of our knowledge, no systematic work on speech analytics applied to a real business scenario has been presented for the Italian language, yet.

The analysis of speech conversations in a contact center

In this section, we give a general overview of the developed system. The work done is inspired by the approach followed in [135, 237], and it sets the basis for a flexible and modular framework for conversation analysis in a contact center.

The system consists of two main modules: the first one is in charge of the automatic transcription of speech conversations, while the second one analyzes the generated texts. The analysis step actually includes two distinct phases, i.e., a lexical and a logical one.

Such a modular approach allows us to provide specific problem-tied solutions and makes it easier to analyze and compare various methods. As a matter of fact, the proposed solution has enabled us to test and combine different methodologies both for the transcription and the text analysis activities, highlighting the best performing configuration.

We focus our attention on the analysis of agent side calls originated in the context of an outbound survey service, carried out by Gap Srlu. In order to guarantee the privacy of the individual agents, all data have been anonymized. As explained in Section 2.2, agents performing outbound calls ought to complete a predefined script in order for a call to be considered successful. Such a rigid structure makes it easier to analyze a conversation and to establish whether the agent has followed all the required steps or not. Although such an analysis may seem trivial, it has very deep

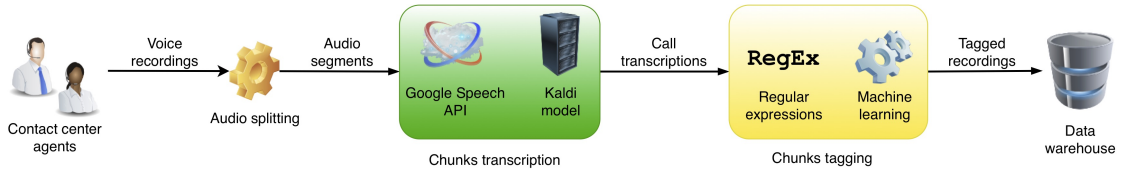


Figure 5.3: The call analysis workflow.

practical implications for the company. As a matter of fact, the common practice in the domain of call centers is that the overall quality of phone conversations is manually checked by supervising staff by simply listening to a random number of calls. Such an operation is clearly time-consuming and it requires a lot of enterprise resources. An automatic analysis module may simplify it a lot by identifying "problematic" calls that require further investigation, thus reducing the time needed for verification and increasing the overall efficacy of the process. Based on the analysis tasks, the supervising staff may then identify the need for further training or specific deficiencies that have to be solved.

The workflow of the process of call analysis is depicted in Figure 5.3. During each phone conversation, two distinct voice recordings are generated, one for each side of the conversation. The two recordings have exactly the same length, meaning that they are filled with silence phases when the other side is speaking. The first step in the processing of the raw audio is the splitting of the agent-side recording, which is done by considering the pauses (see Section 5.2.2, proprietary corpus (spontaneous)). The result is a set of *segments*, which are individually passed to the following transcription step. Transcription may be performed either by means of Google Cloud Speech API, or by exploiting an internal, self-trained Kaldi model, as explained in Section 5.2.2. Once the transcripts have been generated, they can be tagged with proper keywords, encapsulating the semantic content of the conversation. As explained in detail in Section 5.2.3, transcribed segments may be tagged following two main strategies: the first one relies on *regular expressions* defined by domain experts, the other one makes use of machine learning algorithms. Finally, results of the analysis are stored in the company's data warehouse. Data may then be used for reporting purposes, or may form the basis for more advanced tasks such as the firing of automatic rule-based actions.

5.2.2 Transcription of phone conversations

In this section, we address the problem of speech-to-text transformation of phone calls, with reference to the transcription of agent side phone conversations on a specific outbound survey service. To this end, we analyze and compare the performance of Kaldi ASR framework [269] and Google Cloud Speech API [147].

Table 5.2: Corpora used for model training and evaluation

Corpus name	# utterances		Recording time	
	training	test	training	test
CLIPS	1025	-	2h 30m	-
QALL-ME	1208	-	2h 20m	-
Proprietary (read)	3467	-	4h 28m	-
Proprietary (spontaneous)	201	339	30m	35m

Transcribing with internal Kaldi model

Kaldi is a free, open-source toolkit for speech recognition written in C++, and available for both Windows and Unix-like systems for research purposes. The core library supports the modeling of arbitrary phonetic-context sizes, and acoustic modeling with Gaussian mixture models (SGMM) as well as deep neural networks. Other than having a large online documentation, the software comes with a variety of *recipes*, i.e., sets of already worked-out scripts to carry out automatic speech recognition (ASR) tasks, that can be tailored to the specific use case.

Data resources and data preparation For the training of the Kaldi model, we relied on four data sources, each composed of a collection of manually transcribed speech utterances. Two of them, namely, CLIPS and QALL-ME corpora, are freely available for download on their respective websites. The remaining two have been developed inside Gap Srlu, and are proprietary datasets. Table 5.2 lists the amount of data contained in each dataset. Note that the overall amount of transcribed instances available for training is rather low (less then 10 hours of speech) and that just about half of them are generated by the company. Nevertheless, as we shall see in Section 5.2.2, they allowed us to reach a good enough level of transcription performances, in a cost-effective manner.

CLIPS. CLIPS is a collection of Italian speech and text corpora, freely available for research purposes [210]. The entire dataset consists of about 100 hours of speech, equally represented by male and female voices, for a total amount of about 300 speakers. Recordings have been captured in 15 Italian cities, selected on the basis of linguistic and socio-economic principles of representativeness. For each city, several kinds of speech have been included, ranging from radio and television broadcasts to telephone conversations. For the purposes of our work, we focused on a subset of the telephone speech corpora, whose audio is stored in monophonic .wav format files, 32 bit, 8kHz, 128Kb/s. Overall, we considered 1025 transcribed utterances, for a total conversation time of about 2 hours and a half. Each utterance is the result of a role play, in which a human person acts as a tourist who is calling a hotel reception to require a service or ask for information. Although using such a small amount of the entire corpus may seem an unusual choice, we purposely did that

in order to keep training data as coherent to the domain as possible. The latter consists of phone recordings, that typically exhibit specific characteristics in terms of audio quality, presence of noise, and way to speak. On the contrary, tv, radio, and similar recordings are characterized by a significantly different phraseological articulation and sound quality.

QALL-ME. The QALL-ME benchmark consists of 4501 single-speaker utterances [225], declined in several languages. The data were produced by 113 different speakers, mostly native-ones, and equally distributed between males and females. Utterances are divided into *spontaneous* and *read* ones. The first are produced by speakers within a scenario which prompts them to ask for information such as, for instance, the opening times of a local museum, the prices of a certain restaurant, and the time a particular movie is showing. The latter are predefined texts read out loudly by speakers. Audio is stored in monophonic .wav format files, 32 bit, 8kHz, 128Kb/s. For the aims of our work, we selected 1208 transcribed spontaneous utterances in Italian language, for a total recording time of about 2 hours and 20 minutes.

Proprietary corpus (read). It consists of 3467 transcribed utterances, for a total recording time of about 4 hours and a half. Following the typical workflow of the specific outbound survey service under study, a series of typical, agent-side phrases have been prepared, each corresponding to an utterance. A set of selected, Italian-native contact center agents has then been instructed, through a specific data gathering protocol, to read each of the phrases while wearing a typical phone headset, that has been used to record the voice. Each audio file has then been saved in monophonic .wav format, 32 bit, 8 kHz, 128Kb/s Windows PCM.

Proprietary corpus (spontaneous). It consists of agent-side recordings taken from successful calls made for the specific outbound service under study. Audio files have been automatically generated by the company CRM systems, and saved in monophonic .wav format, 32 bit, 8 kHz, 128Kb/s. To integrate such data with that of the other datasets, as a first step, each call recording has been split into segments, on the basis of conversation silences. In order to do that, a Python script relying on the *pydub* library has been coded. Pydub has the ability to split audio considering: (*i*) *minimum silence length* for the pause to be detected, which, in our case, has been set to 750ms; (*ii*) *silence threshold*, i.e., the noise cut-off level used in identifying silence, which we empirically set to -34dB; (*iii*) *keep silence*, by which an amount of silence is kept at the beginning and end of each segment in order to avoid abrupt cuts, set to 450ms. Out of the obtained segments, we kept those lasting at least 3 seconds. The result is a net speaking time of about 65 minutes, over 540 segments.

Training of the model A Kaldi ASR model consists of two parts, a language model and an acoustic model. The first basically defines, by analyzing the training transcripts, the set of phrases that are to be accepted by the recognizer, establishing

the likelihood of a word following another. In other words, the language model gives the prior probability of a word sequence. The latter represents the relationship between an audio signal and the linguistic units, such as phonemes, that make up speech. As we shall discuss, for the training of both models, we relied on the GMM-based *vystadial* Kaldi recipe, which has been adapted for our purposes of building an off-line transcriber for the Italian language. As already pointed out, we decided to neglect more advanced neural network-based acoustic models for a matter of industrial cost: we wanted to determine whether the already present, general purpose system infrastructure could be used to develop the speech analytics solution, without relying on more specialized and expensive hardware.

Language model. The language model has been built by making use of trigrams. The full vocabulary is made of 1873 distinct Italian words. For common Italian words, the pronunciation dictionary, i.e., the mapping between words and phonemes, has been extracted from the Italian version of the *Festival* text-to-speech software [87] by means of a custom script. For custom, or domain-oriented words, pronunciations have been manually defined. As a result, a set of 48 phonemes in ARPAbet format has been established, including a list of frequent onomatopoeiae. Although the size of the dictionary may seem small, recall that we are going to analyze agent-side outbound phone conversations, which ought to show a relatively little variance. Thus, also the vocabulary used by the agents is expected to be rather small.

Acoustic model. As mentioned before, given the scarcity of available training data, and the lack of dedicated hardware, for the training of the acoustic model we relied on GMMs, neglecting the use of neural networks, which would have required a far greater number of already transcribed utterances, other than increasing the overall development cost. In particular, we relied on a customization of *vystadial_en* recipe [198], building a triphone model (LDA + MLLT feature transformation, Maximum Mutual Information objective function), where a triphone is a sequence of three phones that captures the context of single phone. The model has been trained on a dedicated virtual machine running CentOS 7, and featuring a 3.1GHz quad-core processor as well as 8 GB RAM. Despite the relative modest computing resources, the training process took approximately 5 hours.

Transcribing with Google Cloud Speech API

Google Cloud Speech API [147] is a general purpose ASR service that can be used to transcribe conversations in over 80 languages. One of its distinctive features is that it allows for an easy integration into developer applications. Both online (real time) and offline (batch) transcription is supported. Moreover, the service can be tailored to a specific domain, by providing a set of words or phrases that may be pronounced. Such a functionality turns out to be extremely useful, because it allows one to detect personalized names, acronyms, and so on.

We tested the API by developing a Python script that performs a batch tran-

scription of a set of already split segments, providing also a set of custom words to tailor the transcriptions to the specific outbound service of choice.

A critical evaluation of the two solutions

The two approaches (Kaldi and Google Cloud Speech API) have been evaluated against the same set of 339 manually transcribed segments (see Table 5.2). The outcome is as follows: the word error rate of the former is 28.77%, while the latter has performed better, showing a word error rate of 18.70%. It should be observed that such a result considers common stopwords as well, which can mask the true performance of automatic speech recognition for the intended task. Thus, in an attempt to concentrate on "important" terms, we have also evaluated the two transcription systems against the same 339 segments, but neglecting Italian stopwords (as listed in the package NLTK). The results we obtained are not significantly different (in fact, they are sligthly better), showing a word error rate of 28.33% and 18.51% for Kaldi and Google, respectively.

While it is not surprising that Google Speech surpassed the Kaldi solution in terms of raw transcription accuracy, it should be remarked that the in-house model has been trained on a reduced quantity of data, at a relatively low expense for the company. Moreover, while Kaldi is a free solution, the use of Google Speech requires an amount of fees per second of conversation to be paid. Finally, as we shall see in the next section, the quality of the transcriptions generated by Kaldi is good enough to allow for their tagging.

Even though it is true that the evaluation has been carried out on a single service, for which the Kaldi model has been specifically trained, it can be argued that such a model can be easily extended to deal with other kinds of outbound calls by simply increasing the training set, with a limited effort for the company. Notice that, as the model will become more capable of transcribing heterogeneous conversations, it will be possible to apply it, with increasing success, to the inbound case as well.

5.2.3 Semantic tagging of phone conversation transcripts

In this section, we concentrate upon the semantic tagging of the conversation transcripts generated by Kaldi and Google (that may thus be affected by noise and errors). Such a process involves the definition of a suitable conceptual schema to represent the keywords within the existing Gap's enterprise-wide data warehouse (see Chapter 2), and the set up of linguistic as well as machine learning strategies to guide the tagging process.

The database conceptual schema

In order to tag the transcribed phone conversations, we first defined a suitable conceptual (database) schema for the encoding of relevant information. The proposed

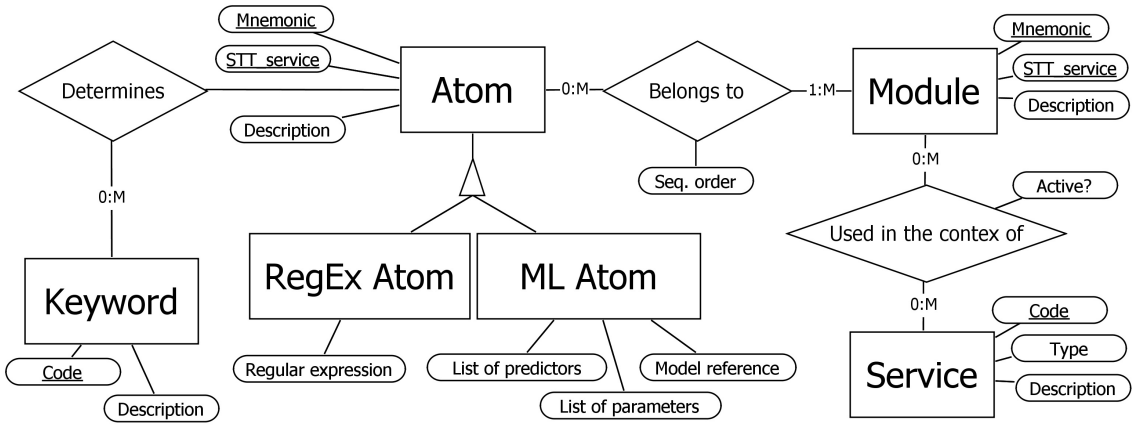


Figure 5.4: A conceptual schema for the tagging process.

schema is depicted in Figure 5.4, using the Entity-Relationship model. As a matter of fact, it is an extension to the company's data warehouse schema discussed in Chapter 2.

Each transcription segment is typically tagged by one or more *keywords*, that is, a word or a combination of words encapsulating a semantic meaning, which expresses what is going on in the corresponding part of the call. Examples of such keywords are *first_survey_question*, *greetings_formula*, and *closing_formula*. A keyword is thus a very general concept that allows one to manage in a uniform way highly heterogeneous phone calls.

A keyword is detected by means of one or more *atoms* (the name suggests the fact that they are the fundamental building blocks of the structure), which can be thought of as strategies to analyze textual segments. Each atom determines one and only one keyword, and is designed to handle the transcriptions generated by a specific speech-to-text approach, e.g., Kaldi or Google Speech, in the context of a service offered by the company. As an example, each service typically has its own greetings formula, that has to be detected in a dedicated way.

Atoms are currently specialized into two non-overlapping categories: RegEx and Machine Learning (ML), though the generality of the schema makes it easier to add other ones. The former relies on regular expressions designed with the help of domain experts to detect the semantic content of the text. The latter makes use of machine learning models to identify such content. Specifically, an ML-Atom is designed to be as general as possible. As such, it contains: (*i*) a reference to the specific model that has to be used, e.g., a logistic regression one, which can be coded directly in an SQL function or into an external script; (*ii*) a list of parameters to be fed to the model, such as the regression coefficients; and, (*iii*) a list of the attributes to extract from the textual information to be used as predictors, such as the presence or absence of specific (combinations of) words.

Each atom belongs to a *module*. Modules are basically lists of atoms that have to be checked together, in a specific order. As an example, a module may refer to three questions which have to be asked in sequence during an outbound interview, each detected by a single atom. Like atoms, also modules are declared for a specific speech-to-text service. In this regard, it is worth noting that a module should be homogeneous, that is, it should only contain atoms pertaining to the same service.

Finally, modules are used in the context of an inbound or outbound service offered by the contact center, an example of which is the toll-free number of an airline company.

Data description

In this section, we give a short account of the actual data for the considered outbound survey service. The list of defined keywords is the following:

- *age*: the agent asked the interviewed person his/her age;
- *call_permission*: the agent asked the called person for the permission to conduct the survey;
- *duration_info*: the agent informed the called person about the duration of the survey;
- *family_unit*: the agent asked the called person about his/her family unit;
- *greeting_initial*: the agent introduced himself/herself correctly at the beginning of the phone call;
- *greeting_final*: the agent pronounced the scripted goodbye phrases;
- *person_identity*: the agent asked the called person for a confirmation of his/her identity;
- *privacy*: the agent informed the called person about the privacy implications of the phone call;
- *profession*: the agent asked the interviewed person about his/her job;
- *question_1*: the agent asked the first question of the survey;
- *question_2*: the agent asked the second question of the survey;
- *question_3*: the agent asked the third question of the survey.

Table 5.3 shows some hand-made transcriptions with the associated keywords. Once again, observe that keywords are general concepts, which are independent from the specific service under consideration.

Table 5.3: Some exemplary, hand-made transcriptions with the associated keywords. Data has been anonymized, and punctuation has been added to the transcriptions for ease of reading.

Phrase (It)	**Phrase (En)**	**Tags**
Si pronto buongiorno sono X dalla X di X, parlo con la signora X?	Hello, my name is X and I am calling from X of X, am I talking with Mrs X?	greeting_initial, person_identity
Lei è pensionato. Ultima domanda, senta, a livello statistico la data solo di nascita... millenovecento...?	You are a pensioner. Last question, listen, statistically, the birth date only... nineteen hundred...?	age, profession
Ho capito. Posso chiederle il nome di battesimo?	Understood. May I ask you for your first name?	person_identity
Mi permette? Trenta secondi, tre domande velocissime...	May I? Thirty seconds, three quick questions...	duration_info

To allow for the training of the machine learning models, and for the evaluation of all the approaches, a set of 4884 text segments originated from 482 distinct outbound sessions of the considered survey service have been manually tagged by domain experts, so that each instance is characterized by the transcription, and by a list of Boolean attributes that track the presence or absence of each specific keyword. Each session has been independently transcribed by Kaldi and Google, in order to evaluate the tagging performance based on both services. The transcriptions may indeed contain some errors, and this is deliberate, since we are not interested in assessing the tagging performances over perfectly transcribed data, but rather in assessing the noise tolerance of the tagging models.

Table 5.4 reports, for each tag, the number of instances in which it is present or not. The resulting dataset has then been split into a training (75%, 3696 instances) and a test (25%, 1188 instances) set, according to a *stratified random sampling by group* approach, where each single session is a group on its own. This allowed us not to fragment segments belonging to a single session between the two sets and, moreover, to preserve the keyword distribution between them.

Tagging with Regular Expressions

Domain expert-defined regular expressions (*RegEx*) have already been used in order to extract knowledge from textual data, e.g., in [140].

In the present work, 12 *RegEx* atoms, which encapsulate the semantic content of utterance transcriptions, have been manually defined, one for each of the keywords that have to be recognized. By exploiting regular expressions, these atoms

Table 5.4: Number of instances in which each tag is present or not, in the full dataset.

Keyword name	# present	% present	# not present	% not present
age	638	13.1	4246	86.9
call_permission	565	11.6	4319	88.4
duration_info	491	10.1	4393	89.9
family_unit	506	10.4	4378	89.6
greeting_initial	560	11.5	4324	88.5
greeting_final	453	9.3	4431	90.7
person_identity	600	12.3	4284	87.7
privacy	440	9.0	4444	91.0
profession	391	8.0	4493	92.0
question_1	516	10.6	4368	89.4
question_2	496	10.2	4388	89.8
question_3	500	10.2	4384	89.8

make it possible to deal with the plain text of the transcriptions, without requiring any further data transformation process, like, for instance, stemming or stopwords removal. As the defined regular expressions try to locate linguistic patterns inside transcriptions, we refer to this tagging strategy as a *linguistic* approach.

In order to make it evident how difficult is the process of defining suitable regular expressions, Table 5.5 provides some complexity measures on them. As it can be observed, expressions are typically quite long when measured in terms of the number of characters and, most notably, they are inherently complex, as the corresponding Deterministic Finite Automata (DFA) have a large number of states. As an example, the regular expression *.*(you born|age|how old are you|you were born in|(in|from) which year|less than eighty).**, translated and reworked from Italian into English, recognizes the atom *age*.

Evaluation of tagging performance Each of the 1188 segments in the Kaldi and Google test sets has been evaluated against the presence of every possible keyword, looking for the concordance between its manual annotations and the ones given by the regular expressions. This is, to all intents and purposes, a *supervised classification* problem, and the performances of our model, shown in Table 5.6, can be measured by standard metrics:

- *accuracy*, which tracks the fraction of times when a tag has been correctly identified in a segment as present or absent;

- *precision*, which is the fraction of segments in which a specific tag has been identified as present by the method, and in which the tag is indeed present;

- *recall*, that reports the proportion of segments presenting the specific tag, that have been in fact identified as such by the method;

- *true negative rate* (TNR), which shows the proportion of segments not presenting the specific tag, that have been classified as negative by the method.

The above-described metrics will be used throughout the experimentations to test the performance of the different approaches to tagging. As shown by Table 5.6, the regular expressions are capable of reaching a very high accuracy in tagging the segments, considering both Kaldi and Google transcriptions. Such a result, however, is not indicative of the true performance, since it is biased by the large disproportion between the instances in which the tags are present and those in which they are not. More useful indicators are the true negative rate, the precision and, especially, the recall, which reveals the tags that are most difficult to identify. As a matter of fact, looking at Kaldi transcribed data, satisfactory results are obtained for most of the tags, except for *call_permission*, that has a fairly low precision and recall. Then, also the tags *greeting_initial*, *person_identity*, and *profession* show a recall performance somewhat under the average, and seem in general hard to identify. As for the results on Google transcriptions, accuracy tends to be slightly lower than Kaldi, with the notable exception of *call_permission*. Moreover, while precision seems to be better in general, recall is overall significantly lower. While this may come as a surprise, given the fact that Google transcripts are in general less error prone than Kaldi ones, it can, however, be easily explained, since it turns out that domain experts have defined the regular expressions based on Kaldi transcriptions only. This highlights the necessity of dealing with Kaldi and Google transcriptions separately, an approach that we are indeed going to pursue in the following sections.

Table 5.5: Complexity measures on the defined regular expressions.

Keyword name	**# chars in regex**	**# states in DFA**
age	94	79
call_permission	143	102
duration_info	158	104
family_unit	135	97
greeting_initial	30	23
greeting_final	143	99
person_identity	189	121
privacy	101	65
profession	103	85
question_1	107	80
question_2	244	126
question_3	209	141

Table 5.6: Tagging performance with the linguistic approach over Kaldi (K) and Google (G) transcriptions.

	Accuracy		Precision		Recall		TNR	
Keyword	**K**	**G**	**K**	**G**	**K**	**G**	**K**	**G**
age	0.971	0.902	0.906	0.988	0.884	0.435	0.985	0.999
call_permission	0.918	0.949	0.697	0.732	0.539	0.598	0.969	0.981
duration_info	0.982	0.942	0.884	0.635	0.953	0.922	0.985	0.944
family_unit	0.969	0.944	0.857	0.897	0.864	0.684	0.982	0.987
greeting_initial	0.949	0.919	0.987	0.988	0.549	0.488	0.999	0.999
greeting_final	0.991	0.977	0.991	1.000	0.916	0.747	0.999	1.000
person_identity	0.944	0.934	0.818	0.925	0.657	0.609	0.981	0.991
privacy	0.991	0.908	0.963	1.000	0.937	0.030	0.996	1.000
profession	0.970	0.908	0.969	1.000	0.646	0.101	0.998	1.000
question_1	0.967	0.987	0.990	1.000	0.723	0.873	0.999	1.000
question_2	0.978	0.964	0.943	0.990	0.833	0.728	0.994	0.999
question_3	0.960	0.964	0.942	0.976	0.653	0.690	0.995	0.998

We conclude by remarking that the definition of suitable regular expressions comes at a cost. As already discussed, each keyword is evaluated by a corresponding regular expression, which must be defined by domain experts on the basis of their personal knowledge, through a delicate, empirical, and long refining process.

Tagging with Logistic Regression

Let us consider now an alternative approach to tagging based on machine learning techniques. One of its advantages is that it allows us to get rid of the support from domain experts in the definition of suitable rules for the identification of tags in the transcriptions.

Since we are interested in determining the presence or absence of a tag in a segment, we focus on a subset of machine learning algorithms, namely, classification algorithms. Moreover, given that the classification task under consideration is a binary one, and that we are interested not only in getting a result, but also in somehow tracking the decision process that has been followed, we opt for *logistic regression*.

Logistic regression generates an interpretable model, as it exposes the relative importance of the features and how they contribute to the overall outcome. It should be remarked that interpretability is a fundamental requirement for the company, since it allows for the validation of the model by domain experts, that, in turn, constitutes an essential step before entering the production phase. This, together with assessments regarding the complexity of the problem, led us to this solution, rather than, for instance, to the use of support vector machines, Conditional Random

Table 5.7: Original number of attributes per dataset, considering Kaldi and Google transcription models.

	Number of predictors		
Model	**Unigram datasets**	**Bigram datasets**	**Trigram datasets**
Kaldi	177	154	102
Google	444	747	664

Fields, or neural networks.

Data preparation Starting from the data described in Section 5.2.3, the training and test datasets have been prepared independently considering Kaldi and Google transcripts as follows. First of all, the text has been converted to lowercase, and all non-alphabetical characters have been removed from each segment. Next, all Italian stopwords have been also removed. Then, *Porter2* stemming algorithm for Italian has been applied. Finally, we extracted from the processed text all unigrams, bigrams, and trigrams with a frequency of at least 1%, as calculated on the training dataset. All operations have been carried out by means of custom-made *Python* scripts, by making use of the *NLTK* library.

Three training and test dataset pairs have then been created for each keyword, considering the different n-gram lengths. Each dataset consists of a set of predictor attributes that track whether the corresponding n-gram is present or not in the transcription, plus the class. As we have already mentioned, such a setting has been replicated for both Kaldi and Google transcriptions, so to train two independent sets of models, following the observation expressed at the end of Section 5.2.3. Thus, given the 12 possible tags, the 3 different n-gram lengths, and the 2 transcription models that have been considered, we generated a total of 72 training and test dataset pairs. For each dataset, Table 5.7 shows the number of originally considered predictors. As it can be seen, the number of attributes generated from the Google transcriptions is considerably higher than Kaldi's. This might be explained by the small dictionary employed by the latter transcription model.

Training of the models In order to train the models, we relied on WEKA's *Logistic* algorithm [331]. However, given the very high number of predictors involved, an attribute selection step has been applied in order to reduce the dimension of the datasets before actually proceeding with the training phase.

Recall that, as explained in Section 1.1.3, the design of a feature selection step entails the selection of a search strategy, to guide the incremental generation of the feature set, and of an evaluation strategy, to score the candidates (subsets of attributes).

As for the search strategy, we considered WEKA's *BestFirst* [263], which is an implementation of *beam search*, that searches the space of attribute subsets by

Table 5.8: Number of attributes per dataset after the attribute selection step, considering Kaldi (K) and Google (G) transcriptions.

	# preds unigram		# preds bigram		# preds trigram	
Keyword name	**K**	**G**	**K**	**G**	**K**	**G**
age	8	45	11	66	8	68
call_permission	8	17	9	21	9	71
duration_info	14	18	11	22	10	64
family_unit	15	20	12	65	10	58
greeting_initial	11	18	14	69	13	72
greeting_final	11	14	13	17	11	58
person_identity	8	17	9	72	9	80
privacy	10	13	12	70	10	69
profession	11	22	9	22	24	66
question_1	11	15	13	74	16	72
question_2	10	14	10	66	9	68
question_3	12	14	10	58	12	62

greedy hill climbing, augmented with a backtracking capability.

As for the evaluation strategy, WEKA's multivariate filter *CfsSubsetEval* [153] has been used. Such a filter evaluates the worthiness of a subset of attributes by considering the individual predictive power of each feature, together with the degree of redundancy between them, preferring subsets of features that are highly correlated with the class while having low intercorrelation among them.

The results of the attribute selection phase are reported in Table 5.8. As it turned out, many of the attributes have been discarded, meaning, perhaps not surprisingly, that only certain n-grams are useful for determining the presence or absence of each specific keyword. Once again, observe that the Google based datasets have a higher number of attributes than Kaldi's.

Evaluation of tagging performance We considered the performance of the models based on, respectively, unigrams, bigrams, and trigrams on each of the 1188 segments in the Kaldi and Google test sets. The results are reported in Table 5.9.

Considering both Kaldi and Google transcripts and models, precision does not show a consistent behaviour among the n-gram sizes, while in many cases recall decreases sharply with the larger n-gram sizes. It is also interesting to observe that, as the n-grams grow in size, the accuracy seems to slightly decrease. This behavior can be explained in two ways: (*i*) there might not be a sufficient number of instances in the datasets to justify the adoption of bigrams and trigrams; (*ii*) the presence or absence of a keyword in calls generated in the context of the specific service that has been considered is not a difficult concept to learn, and thus unigrams are sufficient for the task.

Table 5.9: Tagging performance with the Logistic Regression approach, over Kaldi (K) and Google (G) transcriptions.

	Accuracy		Precision		Recall		TNR	
Keyword	**K**	**G**	**K**	**G**	**K**	**G**	**K**	**G**
age-uni	0.966	0.953	0.897	0.957	0.854	0.988	0.984	0.788
age-bi	0.961	0.953	0.940	0.968	0.768	0.976	0.992	0.842
age-tri	0.924	0.934	0.963	0.940	0.470	0.982	0.997	0.701
call_permission-uni	0.931	0.959	0.798	0.765	0.560	0.713	0.981	0.981
call_permission-bi	0.910	0.947	0.713	0.727	0.404	0.552	0.978	0.982
call_permission-tri	0.902	0.942	0.705	0.736	0.305	0.448	0.983	0.986
duration_info-uni	0.967	0.955	0.908	0.807	0.773	0.696	0.991	0.982
duration_info-bi	0.963	0.951	0.912	0.821	0.727	0.628	0.992	0.986
duration_info-tri	0.913	0.939	0.833	0.785	0.234	0.500	0.994	0.986
family_unit-uni	0.963	0.972	0.893	0.930	0.758	0.868	0.989	0.989
family_unit-bi	0.955	0.943	0.944	0.960	0.636	0.625	0.995	0.996
family_unit-tri	0.900	0.915	1.000	0.918	0.099	0.441	1.000	0.993
greeting_initial-uni	0.977	0.949	0.921	0.900	0.872	0.759	0.991	0.985
greeting_initial-bi	0.961	0.963	0.922	0.957	0.714	0.795	0.992	0.993
greeting_initial-tri	0.931	0.940	0.947	0.964	0.406	0.639	0.997	0.996
greeting_final-uni	0.994	0.997	0.991	0.990	0.950	0.980	0.999	0.999
greeting_final-bi	0.990	0.989	0.965	0.968	0.933	0.909	0.996	0.997
greeting_final-tri	0.986	0.985	0.964	1.000	0.891	0.838	0.996	1.000
person_identity-uni	0.937	0.943	0.750	0.868	0.679	0.733	0.971	0.980
person_identity-bi	0.932	0.935	0.811	0.859	0.533	0.683	0.984	0.980
person_identity-tri	0.910	0.923	0.800	0.891	0.292	0.559	0.991	0.988
privacy-uni	0.991	0.998	0.955	0.990	0.946	0.990	0.995	0.999
privacy-bi	0.988	0.995	0.945	0.990	0.928	0.960	0.994	0.999
privacy-tri	0.973	0.979	0.891	0.976	0.811	0.792	0.990	0.998
profession-uni	0.987	0.975	0.893	0.873	0.958	0.881	0.990	0.985
profession-bi	0.929	0.944	1.000	0.889	0.125	0.514	1.000	0.993
profession-tri	0.919	0.911	-	0.769	0.000	0.184	1.000	0.994
question_1-uni	0.984	0.993	0.968	0.972	0.891	0.955	0.996	0.997
question_1-bi	0.985	0.995	0.984	0.982	0.883	0.973	0.998	0.998
question_1-tri	0.980	0.987	0.991	1.000	0.832	0.873	0.999	1.000
question_2-uni	0.984	0.990	0.924	0.970	0.917	0.949	0.992	0.996
question_2-bi	0.976	0.991	0.933	0.992	0.817	0.934	0.993	0.999
question_2-tri	0.974	0.977	0.932	0.959	0.800	0.853	0.993	0.995
question_3-uni	0.985	0.990	0.942	0.973	0.911	0.931	0.993	0.997
question_3-bi	0.976	0.987	0.936	0.964	0.823	0.914	0.993	0.996
question_3-tri	0.973	0.985	0.942	0.981	0.790	0.879	0.994	0.998

Tagging with a hybrid approach

In this section, we explore a simple hybrid approach to tagging that combines the linguistic and machine learning based strategies.

In developing it, we took into account that, even though the company is definitely interested in finding patterns of problematic calls, it is also true that the supervising staff ought not to be overwhelmed with false notifications (a correct call signaled as problematic) carrying requests for verification.

On the basis of empirical evaluations, it has been found out that a simple combination of the results of the linguistic and machine learning approaches leads to satisfactory results: a keyword is considered to be present if at least one of the two individual strategies marks it as present.

Table 5.10 reports the detailed figures, obtained from the combination between the results given by the regular expressions and the unigram logistic models. As it can be observed for both Kaldi and Google transcripts, the accuracies typically show a slight improvement over the two individual approaches. Moreover, as expected, given the specific combination strategy, the precision has decreased a little bit, while the recall has gained a considerable boost. This is exactly what had to be accomplished: it is now more likely to predict a tag as present, and thus the problematic calls are limited to the most serious cases.

A critical evaluation of the proposed solutions

We conclude the section with a comparison of the proposed solutions. Consider Table 5.11. As it turns out, the regular expression approach has an overall accuracy comparable to the one exhibited by the unigram-based machine learning approach, which has already been shown to be the best n-gram choice for the problem. However, the unigram approach exhibits a clearly better behaviour for what concerns the recall which, as we already pointed out, is the key performance indicator for the considered problem. In addition, the definition of suitable regular expressions is a tedious and error-prone process, that has to be done manually, which in turn consumes a significant amount of company time and personnel resources. As for the hybrid approach, it has proven itself capable of boosting recall, although at the cost of a slight decrease in precision. Finally, it should be observed that the results based on Kaldi transcripts are just slightly inferior to those obtained on Google ones, especially when considering the hybrid solution. Thus, at least for the considered tagging task, the in-house and low-cost developed solution seems to be sufficient to accomplish the intended goals.

5.2.4 An alternative approach to the tagging of segments

In this section, we illustrate the setup and the results of an alternative approach to the tagging of segments carried out with J48S decision tree algorithm, that has

Table 5.10: Tagging performance with the hybrid approach, over Kaldi (K) and Google (G) transcriptions.

	Accuracy		Precision		Recall		TNR	
Keyword	**K**	**G**	**K**	**G**	**K**	**G**	**K**	**G**
age	0.966	0.962	0.897	0.943	0.854	0.836	0.984	0.989
call_permission	0.923	0.955	0.699	0.723	0.610	0.737	0.965	0.976
duration_info	0.982	0.941	0.879	0.637	0.961	0.958	0.984	0.939
family_unit	0.962	0.968	0.800	0.880	0.879	0.922	0.973	0.977
greeting_initial	0.981	0.942	0.917	0.900	0.910	0.721	0.990	0.985
greeting_final	0.995	0.995	0.991	0.983	0.958	0.964	0.999	0.999
person_identity	0.946	0.949	0.784	0.846	0.803	0.796	0.965	0.976
privacy	0.992	0.999	0.955	0.993	0.955	1.000	0.995	0.999
profession	0.987	0.981	0.886	0.925	0.969	0.905	0.989	0.991
question_1	0.985	0.993	0.969	0.953	0.898	0.980	0.996	0.994
question_2	0.984	0.992	0.924	0.963	0.917	0.970	0.992	0.995
question_3	0.985	0.993	0.934	0.978	0.919	0.958	0.993	0.997

been presented in Section 5.1.

Data preparation

Starting from the data described in Section 5.2.3, the training and test datasets have been prepared as follows. As done for the logistic regression case, the text has been converted to lowercase, and all non-alphabetical characters have been removed from each segment. Then, stopwords have been discarded, and *Porter2* stemming algorithm has been applied. A training and a test dataset have been created for each keyword and transcription model, where each of the instances is characterized by just two attributes: the processed text of the segment, and an attribute identifying the presence or absence of the specific keyword.

Table 5.11: Average performance per method, over Kaldi (K) and Google (G) transcriptions.

	Accuracy		Precision		Recall		TNR	
Keyword	**K**	**G**	**K**	**G**	**K**	**G**	**K**	**G**
Regular expressions	0.966	0.942	0.912	0.928	0.763	0.575	0.990	0.992
Logistic, unigram	0.972	0.973	0.903	0.916	0.839	0.870	0.989	0.973
Logistic, bigram	0.961	0.966	0.917	0.923	0.691	0.789	0.992	0.980
Logistic, trigram	0.940	0.951	-	0.910	0.494	0.666	0.995	0.895
Hybrid	0.974	0.973	0.886	0.894	0.886	0.896	0.985	0.985

Training of the model

All algorithm parameters have been left with their default values, except for *minSupport*, which has been set to 0.01 (1%), in order to allow for the extraction of uncommon patterns, without performing any specific kind of tuning.

Evaluation of tagging performance

Following the same approach as the one used for the evaluation of the *RegEx* and logistic regression strategies, we measured the performance of J48S in determining the presence or absence of each keyword in the same test set made of 1188 segments. As shown by the results reported in Table 5.12, J48S is capable of matching the accuracy of the hybrid approach. Moreover, though having a slightly worse precision than regular expression and logistic models, it surpasses the hybrid method score. As for the recall, it is only beaten by the hybrid solution, which nonetheless is much more complex, making use of two distinct strategies.

All in all, we can conclude that J48S is able of providing results comparable to the ones of the hybrid approach. Moreover, it is worth mentioning that: (*i*) the measured performances have been obtained training the algorithm with the default parameters (except for the minimum support value), and thus we may expect to be able to obtain better values through a dedicated tuning phase; (*ii*) the complexity of the data preparation phase is greatly reduced, since the decision tree is capable of directly exploiting textual information, allowing one to avoid the n-gram preprocessing step, and the related a-priori choices of the n-gram sizes, frequencies, and feature selection strategies altogether; (*iii*) sequential patterns may be more resistant to noise in the data than bigrams and trigrams, as they are capable of skipping irrelevant words thanks to the *maxGap* parameter; (*iv*) finally, and most importantly, the constructed trees are much more understandable to domain experts (which are not computer scientists) than regression models. As an example, in Figure 5.5 we show a very simple model that recognizes the presence of the tag *greeting_final*. Observe how the decision tree is capable of combining patterns having different length: given a transcription to be classified, the model starts by testing whether the text contains the single term *recontact*; if this is not the case, then it checks the transcription against the presence of the word *thank* followed by the word *cooperation*; if this pattern is not satisfied either, and the utterance does not contain the term *inform*, finally the instance is given class 0 (no *greeting_final* tag has been detected).

5.2.5 On the operational impact of the speech analytics process

The introduction of a speech analytics process in a contact center may have a really wide and deep impact. Improving customer experience, reducing costs, identifying selling opportunities, reducing attrition and churn, evaluating agents, and rising

Table 5.12: Tagging performance with J48S, over Kaldi (K) and Google (G) transcriptions.

	Accuracy		Precision		Recall		TNR	
Keyword	**K**	**G**	**K**	**G**	**K**	**G**	**K**	**G**
age	0.964	0.957	0.935	0.964	0.793	0.985	0.991	0.821
call_permission	0.923	0.958	0.671	0.776	0.681	0.678	0.955	0.983
duration_info	0.971	0.955	0.891	0.821	0.828	0.677	0.988	0.985
family_unit	0.971	0.970	0.877	0.929	0.864	0.855	0.985	0.989
greeting_initial	0.973	0.974	0.924	0.921	0.827	0.910	0.992	0.986
greeting_final	0.988	0.990	0.965	0.989	0.916	0.899	0.996	0.999
person_identity	0.944	0.940	0.763	0.830	0.752	0.758	0.970	0.972
privacy	0.992	0.995	0.955	0.990	0.955	0.960	0.995	0.999
profession	0.984	0.974	0.897	0.893	0.906	0.844	0.991	0.989
question_1	0.990	0.992	0.985	0.972	0.927	0.946	0.998	0.997
question_2	0.980	0.990	0.914	0.963	0.883	0.956	0.991	0.995
question_3	0.983	0.992	0.919	0.974	0.919	0.948	0.991	0.997
Average	0.972	0.974	0.891	0.919	0.854	0.868	0.987	0.976

service quality are some of the sought-after advantages. Among them, the improvement of service quality and the overall evaluation of the workforce are very important business goals, and constitute the main driving factors of the present study.

Most of the operational processes that take place in a contact center are quite complex and, in a modern company, widely based on business intelligence. The experimentation illustrated in the present work shows that with a relatively small effort it is possible to develop a fully fledged in-house speech analytics solution, the adoption of which may imply a refactoring of operational processes in order to get the best results in terms of service quality and costs control.

In detail, Gap Srlu makes use of a call quality assurance process having two dimensions: *external*, aimed at the needs of customers and of final users, and *internal*, pointing to operational issues such as, for instance, agent management. The process consists of a series of steps:

1. *call scope evaluation*, that is, determining if the call is pertinent to the service;
2. *checking of mandatory contents*, e.g., checking whether or not all the relevant questions of an outbound survey have been asked;
3. *proper data collection and actions to be performed*, e.g., sending a payment reminder as one of the results of a call;
4. *correct call outcome classification*, by which the call outcome manually set by the agent is validated;

```
transcription !contains recontact
|  transcription !contains thank>cooperation
|  |  transcription !contains inform: 0
|  |  transcription contains inform
|  |  |  transcription !contains inform>company_name: 0
|  |  |  transcription contains inform>company_name
|  |  |  |  transcription !contains allow: 0
|  |  |  |  transcription contains allow: 1
|  transcription contains thank>cooperation
|  |  transcription !contains ok>finish>thank>behalf
|  |  |  transcription !contains allow>inform
|  |  |  |  transcription !contains cooperation>day: 0
|  |  |  |  transcription contains cooperation >day: 1
|  |  |  transcription contains allow>inform: 1
|  |  transcription contains ok>finish>thank>behalf: 1
transcription contains recontact: 1
```

Figure 5.5: J48S decision tree recognizing the presence (class = 1) of the tag *greeting_final*.

5. *evaluation of soft skills*, e.g., checking whether the agent has been polite during the call.

Speech analytics clearly impacts on all the above-listed steps, as the exploitation of conversation transcripts may potentially play a major role in each of the tasks. Moreover, all phases are susceptible of being automatized, and, by formulating suitable business rules, several automatic actions may be performed. Examples include signalling to the operational staff the most problematic calls, or issuing specific training tasks to cope with the deficiencies emerged in the handling of the calls.

Highlighting some detail, in order to better identify relevant parts of the agent side speech, Gap Srlu operational staff has organized the semantic tags discovered by the speech analytics process in a set of logical rules. The Boolean values of the rules identify in a simple way what is considered to be compliant and what is not, according to the service operational guidelines. As a further step, a partitioning of the tags into three groups helps the operational staff in identifying the severity level of the flaws discovered in a call.

Let us now give an example that shows how tags, logical rules, and severity levels work together in the process. The operational guidelines state that the initial greeting and the call permission items have to be evaluated together, and the corresponding check is considered to be passed only if both of them are present. This can be represented by means of a simple logical rule: *greeting_initial AND call_permission*. Moreover, the tag partitioning leads to establish the severity level of potentially missing tags. According to the guidelines, in our example the tags *greeting_initial* and *call_permission* are both categorized in the same set associated

Table 5.13: Corrispondence among tags, logical rules, and severity levels.

Tag	Logical rule	Severity
age, person_identity	age ∧ person_identity	1
greeting_initial, call_permission	greeting_initial ∧ call_permission	2
question_1, question_2, question_3	question_1 ∧ question_2 ∧ question_3	3

with the severity level 2 (mid-level, which means that the call has some problems, but the survey is still valid). Table 5.13 presents some further concrete examples taken from the service operational guidelines (all proposed rules rely on conjunctions; of course, any other logical connective can in principle be used).

To summarize, by combining semantic tags, logical rules, and partitioning, each single call may be analyzed in detail and a clear assessment pattern emerges. Such a method gives the operational staff the ability to gain solid insights in a faster way and with more precise results with respect to a manual call monitoring procedure. The supervising staff confirms an improvement of 15% in the overall quality of the process with respect to the old procedure, which required a manual sampling of the calls to be monitored, followed by a completely handmade evaluation of their contents. Considering the time spent for the analysis, the gain is about 17%. If we restrict our attention to the detection of surveys to be cancelled for serious flaws committed by the agents, then the increment is sensible, with results from the field indicating a gain of more than 300%.

It is worth to underline once more that, as shown, these results an be accomplished by relying on the Kaldi-based proprietary transcription model, confirming that the operational fallout is pretty good, despite its higher word error rate with respect to the Google Speech API solution. Moreover, the tested solution has virtually no costs except for the internal one and this is clearly a relevant benefit for a company.

As already pointed out, the Gap Srlu IT and operational staffs are evaluating further automatic steps that can be possibly added to the call quality assurance process. As an example, on the basis of the logical rules outcomes and the severity level of the flaws, a set of e-learning modules could be suggested to the involved agents. This would lead to an instance of what is known as the "learning analytics cycle" [83], i.e., an ongoing (and almost fully automated) circular process involving learners (the agents), data (the transcribed speech), metrics (the speech analysis), and interventions (the e-learning and supervisoring). This seems to be a very effective and interesting evolution for the operational impact of the entire project.

5.2.6 Discussion

In this work, the experimentation of a speech analytics process for an Italian contact center has been described, which consists of two distinct phases: a transcription

step, where agent side conversations are transformed to text, and a tagging step, where semantic content is attached to the transcripts. As for the transcription phase, the feasibility of building an in house solution, relying on a small amount of training data has been confirmed, by effectively developing a model using the Kaldi framework and comparing its results with those obtained by Google Cloud Speech API. As for the tagging phase, the tested machine learning based strategies have proven themselves to be reliable, and more efficient than classical solutions based on regular expressions.

Finally, the decision tree inducing algorithm J48S, applied to the problem of tagging, has exhibited a performance comparable to the hybrid solution, while relying on just a single, highly interpretable model.

Overall, based on the experimental evidence that we provided, we can safely claim that the considered speech analytics process can be built at a low-cost for the company, relying on in-house developed models with respect to both the transcription and the tagging phases.

Such an analytics module can be easily integrated in the architecture presented in Chapter 2, for instance, as an extension of the operator performance assessment tool.

6

Working with Time Series Data

This chapter deals with the analysis of time series data. In particular, Section 6.1 describes J48SS [67], which is a further extension to J48S decision tree inducing algorithm (see Section 5.1). Then, Section 6.2 presents the results of the application of J48SS to a real case problem, that is, a pollution analysis task on data collected in the city of Wrocław, Poland [64]. As we shall see, J48SS excels when run on datasets where each instance is characterized by heterogeneous information, such as, for instance, in the considered pollution analysis task. For the cases in which an instance is represented just by a time series and possibly some derived attributes (as in the case of the UCR datasets tested in Section 6.1), better results might be obtained through the use of dedicated algorithms, like recurrent neural networks.

6.1 J48SS: a novel decision tree approach for the handling of time series

Time series play a major role in many domains. As an example, in economy, they may be used for stock market price prediction [249]; in medicine, they may help in forecasting the patient arrival rate in emergency centers [326]; in geophysics, they convey important information about the evolution of the temperature in the oceanic waters [281].

This work focuses on the general problem of *time series classification*. Given a training dataset of labelled time series data, the goal is to derive a model capable of assigning a label to new, unlabelled instances. Various techniques have been used in the past for such a purpose, ranging from support vector machines [189] to neural networks [190]. However, when understanding and validating the decision process is as important as the accuracy degree of the prediction itself, decision trees are still a popular choice among classification models (see, for instance, [25]).

The main contribution of this work is a novel decision tree learner based on the J48S extension [66] of C4.5 [274], called J48SS, which is capable of exploiting static (categorical and numerical) attributes as well as sequential and time series data during the same execution cycle. As mentioned in the Introduction, such a mixture of heterogeneous data can actually be observed in many domains. For in-

stance, we may think of a fault prediction scenario, where an industrial machine may be characterized by (*i*) static attributes, such as the machinery model and its age of service, (*ii*) sequential attributes, storing information about the history of states and warnings the machine has gone through in the past, and (*iii*) time series, tracking the machine production rate over time, or other kinds of sensor data. Based on such pieces of information, one may perform a classification task, with the aim of identifying, for instance, the fault that most likely is affecting the machine, if any. Clearly, the availability of a single, universal model for the classification of such heterogeneous data greatly reduces the data preparation effort, and eases the subsequent analysis phase. The proposed algorithm makes use of the concept of *shapelet* (see Section 1.1.2), originally introduced in [337]. A shapelet may be thought of as a time series pattern that is useful for classification purposes. In the recent years, several contributions have employed such a primitive for time series analysis [148, 149, 162, 192, 280, 284, 330], although, to the best of our knowledge, the present approach is the first one that relies on multi-objective evolutionary computation for the shapelet generation process, in the context of decision tree classification. Moreover, unlike most of the previous studies, the proposed solution does not require the generated shapelets to be part of the training set, and it provides an effective strategy to control their degree of complexity, which can be easily adapted to the requirements of the specific classification task.

Experimental results show that J48SS is capable of producing interpretable models, that nevertheless achieve better classification performances than previous solutions based on single decision trees, while keeping a low computation time. Moreover, preliminary insights suggest that J48SS trees might be combined in smaller and possibly more accurate ensemble models than those proposed in the past, although at the price of a loss in interpretability.

In the literature, some very recent approaches have been presented that achieve a higher accuracy on pure time series classification than the one provided by J48SS (see, e.g., [292]). Nevertheless, these approaches typically lack two distinctive features of the novel decision tree, namely, the ability of natively dealing with mixed data, and the interpretability of the generated models.

The work is organized as follows: Sections 6.1.1 and 6.1.2 present respectively the implementation of NSGA-II and the extensions made to J48S in order to handle time series classification. Section 6.1.3 describes the experiments carried out with the proposed algorithm, and presents the collected results with respect to single models and ensembles. The last section provides an overall assessment of the work done.

6.1.1 Using NSGA-II to extract time series shapelets

Evolutionary algorithms have already been successfully applied to various phases of the decision tree induction process in the past [42]. Recall that the description

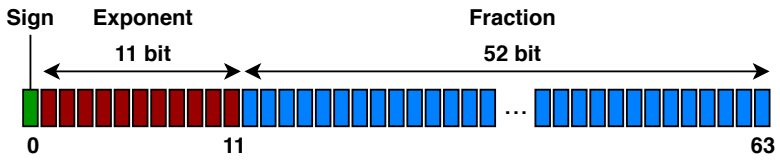


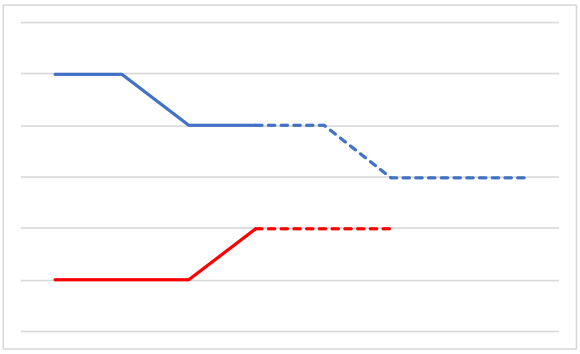
Figure 6.1: IEEE 754 double-precision floating point format.

of the implementation of an evolutionary algorithm involves the description of: (*i*) how the single solutions are represented; (*ii*) how the initial population is set up; (*iii*) which evolutionary operators (crossover, mutation) are employed; (*iv*) which fitness function is used, and (*v*) in a multi-objective setting, which decision method is adopted to select a single solution from the resulting Pareto front. All these points are discussed in the following. As we shall see, unlike the majority of previous contributions, the proposed solution does not require shapelets to be part of the specific training set, but they may evolve freely through the evolutionary operators. All code pertaining to the evolutionary algorithm has been implemented in the jMetal framework [102].

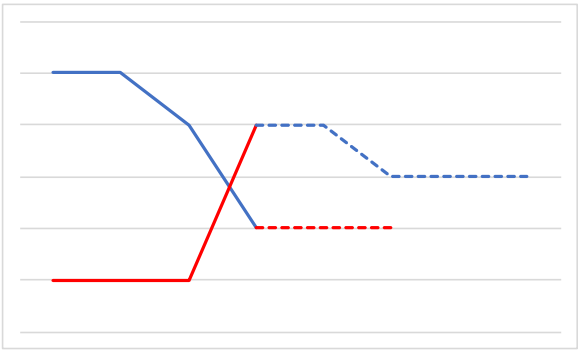
Representation of the solutions and initial population Each shapelet in the population is represented by an ordered list of binary arrays. Each binary array in the list corresponds to a floating point value, encoded in *IEEE 754 double-precision* 64 bit format (see Figure 6.1): the first, leftmost bit establishes the sign of the number, the following 11 bits represent the exponent, and the remaining 52 bits encode the fraction. As we shall see, this kind of representation makes it extremely convenient to apply the *mutation* operator. Given n the number of elements in the population, each of the instances is initialized in the following way. A time series is randomly selected from the dataset, and then a begin and end index are randomly generated. The shapelet is then simply extracted from the portion of the time series that lies between the two indexes.

Crossover operator A strategy similar to the *single-point* crossover [105] is used. Given two parent solutions, i.e., two lists of binary arrays, two random indexes are generated, one for each parent. Such indexes define the beginning of the *tails* of the two lists, which are then swapped between the parents, generating two offspring. A graphical representation of the crossover operation, where the two indexes happen to have the same value, is shown in Figure 6.2. Observe that, given the operator, the two offspring may have different lengths than those of the two parents.

Mutation operator A proper mutation operator should not try to purposely improve a solution, since this would bias the evolution process of the population. Instead, it should cause random, unbiased changes in the solution components [105]. In the present implementation, mutation is carried out by performing random flips in the binary array representation of the elements composing the shapelet. Recall that the binary arrays follow the double-precision IEEE 754 notation. Thus, the higher



(a) Shapelets before the crossover operation.



(b) Shapelets after the crossover operation.

Figure 6.2: Crossover operation. Dotted lines represent the tails of the shapelets, that are being swapped.

the index in the array, the less significant the corresponding bit. The probability of flipping the ith bit is thus given by $P_{mut} - P_{mut} * log_{65}(65 - i - 1)$, where P_{mut} is the overall mutation probability, established by the *mutationP* parameter. Intuitively, such a formula penalizes flipping the most significant bits in the representation. The rationale behind this choice is that abrupt changes of the values of the shapelets are unwanted, since they might hinder the convergence of the algorithm to a (local) optimum. Finally, observe that randomly flipping the bits may lead to undesired results, encoded in the IEEE 754 format as *NaN* (Not a Number). To solve such infrequent cases, a further random flip is carried out in the exponent section of the binary array.

Fitness function One of the distinctive features of the proposed algorithm is the bi-objective fitness function it exploits. The first objective is to maximize the *Information Gain* (IG) of the given shapelet (which corresponds to the normalized gain, since a split over a shapelet is always binary), as calculated on the training instances belonging to the specific tree node. To determine the IG of a shapelet, first its minimum Euclidean distance with respect to every time series is determined, following a sliding window approach. Then, the resulting values are dealt with following the

same strategy as the one used by J48/C4.5 for splitting on numerical attributes (see [274]). Note that, in principle, any other distance metric may be used in the objective, by simply defining a proper procedure. The second objective is to reduce the *functional complexity* [318] of the shapelet, in order to avoid unwanted overfitting phenomena that might occur if considering only the first objective. To do so, the well-known *Lempel-Ziv-Welch* (LZW) compression algorithm [327] is relied on: the (decimal) string representation of the shapelet is compressed through LZW, then the ratio between the lengths of the compressed and the original strings is evaluated. Note that such a ratio is typically less than 1, except for very small shapelets, in which the overhead of the LZW algorithm might make the compressed string actually larger than the original one. Finally, NSGA-II is set to minimize the ratio: the underlying idea is that "regular", well-generalizing shapelets should be more prone to compression than complex, overfitted ones. As a side effect, also extremely small (typically singleton) and uninformative shapelets are discouraged by the objective. Observe again that, since a bi-objective fitness function is used, the final result is a set of Pareto-optimal solutions, with respect to the Information Gain and the compression ratio. This is an extremely important characteristic of the present approach as different problems may require different functional complexities of the shapelets in order to be optimally solved. As we shall see, a trade-off between the two objectives may be easily achieved. It is worth mentioning that also other fitness functions have been tested, inspired by the approaches described in Section 1.4. Among them, the best results have been given by:

- an early-stopping strategy inspired by the separate-set evaluation discussed in [129]. The training instances are divided into two equally-sized datasets. Then, the evolutionary algorithm is trained on the first subset, with a single-objective fitness function aimed at maximizing the information gain of the solutions. The other dataset is used to separately assess a second information gain value for each of the solutions. During the algorithm computation, the best performing individual according to the separated set is kept track of, and the computation is stopped after k non improving evolutionary steps. Finally, the best individual according to the separated set is returned;

- a bootstrap strategy inspired by the one presented in [119]. The evolutionary algorithm evaluates each individual along 100 different datasets (each having the same size of the original one), obtained from a random sampling (with replacement) of the original dataset. Then, two objectives are optimized: maximize the average information gain of the shapelet, as calculated along the 100 datasets, while minimizing its standard deviation, in an attempt to search for shapelets that are good in general.

Nevertheless, both of the strategies provided inferior accuracy performances than the previously discussed bi-objective fitness function. While for the first approach this might be explained by the fact that the number of training instances is greatly

reduced (a problem that is exacerbated in the smallest nodes of the tree), the reason why the second one performed so poorly is still unclear, and will be investigated in further studies.

Decision method As it happens with any multi-objective optimization task, the result of the shapelet generation phase is a set of non-dominated solutions, in this case with respect to the two objectives: maximizing the information gain, while minimizing the compression ratio of the individuals. Thus, to extract a single shapelet out from the set, a decision method is needed. To this end, each solution is evaluated with respect to the following formula, similarly to what is done for J48S and sequential frequent patterns:

$$(1 - W) * ComprRatio_{rel} + W * (1 - InfoGain_{rel}) \tag{6.1}$$

where $W \in [0, 1]$ is a weight that can be customized by the user (by the *patternWeight* parameter), $ComprRatio_{rel} \in [0, 1]$ is the relative compression ratio of the shapelet, and $InfoGain_{rel} \in [0, 1]$ is the relative information gain of the shapelet, with reference to the respective highest values observed in the population. The shapelet that minimizes the value of such a formula is selected as the final result of the optimization procedure, and is returned to the decision tree for the purpose of the splitting process. Intuitively, the larger the user sets W, the more accurate on the training set the extracted shapelet will be. On the contrary, smaller values of W should result in less complex solutions being selected.

An overview of the algorithm NSGA-II applied to the shapelet extraction problem is presented in Figure 6.3. For more details regarding the inner workings of the chosen evolutionary algorithm, see Section 1.4.1.

6.1.2 Adapting J48 to handle time series data

Let us now focus on the execution of J48SS on time series data. As in the classical C4.5 algorithm, the learning procedure is made of two distinct phases: a growing one, in which the tree is built, and a final pruning step, that discards the non-relevant parts of the tree, improving its generalization capability and readability (for details on this latter phase see, for example, [65]). Algorithm 3 shows the main recursive procedure used in the tree building phase. It takes in input a node (initially, the root node of the tree, in which all instances reside), and proceeds as follows. If the given node is pure (i.e., all instances have the same label) or another stopping criteria is met (e.g., based on minimum cardinality), then the node is transformed to a leaf of the tree. Otherwise, the algorithm determines the best attribute for the split. It first evaluates all static attributes (categorical and numerical), determining their *normalized gain* value (see Section 5.1.2), following the default J48/C4.5 strategy. Then, it focuses on string attributes that encode sequences: for each of them, the best *closed frequent pattern* is extracted, following the strategy described in Section 5.1. Finally, for each string attribute of the dataset that encodes a time series, it

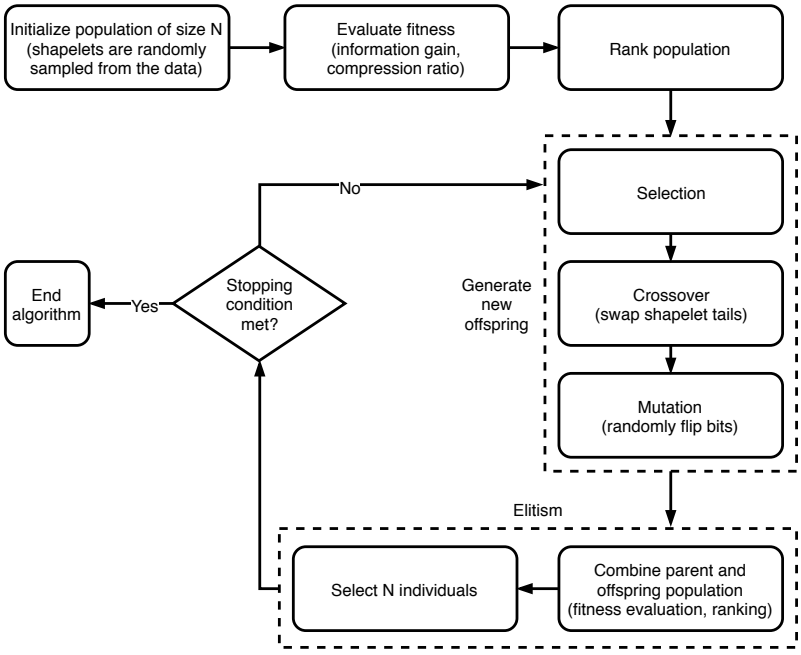


Figure 6.3: Overview of the algorithm NSGA-II applied to the shapelet extraction problem.

runs NSGA-II with respect to the set of instances belonging to the node (details are provided in the remainder of this section). As a result, a single shapelet is returned. The shapelet is evaluated as if it was a numerical attribute, considering the Euclidean distance between each instance's time series and itself (adopting also the *subsequence distance early abandon* strategy to speed-up the computation, as suggested in [337]). In this way, its normalized gain value is calculated, and compared with the best found one. As a final step, the attribute with the maximum normalized gain is used to split the node, and the algorithm is recursively called on each of the children. As for the input parameters of new algorithm specific to the shapelet extraction process, they are listed in Table 6.1. Apart from *patternWeight*, which determines the degree of *complexity* of the shapelet returned to the decision tree by NSGA-II (see Algorithm 3), they all control the evolutionary process:

- *crossoverP* determines the probability of breeding two individuals of the population;
- *mutationP* determines how often an element undergoes a random mutation;
- *popSize* determines the number of individuals in the population;
- *numEvals* sets the number of evaluations (and, thus, generations) that are going to be carried out during the optimization process.

Table 6.1: Custom parameters for time series shapelet extraction in J48SS.

Parameter name	Default	Description
crossoverP	0.8	crossover probability to be used in the EA
mutationP	0.1	mutation probability to be used in the EA
numEvals	500	# of evaluations to be carried out by the EA
patternWeight	0.75	weight for the extraction of the final shapelet
popSize	100	population size to be used by the EA

Note that the number of evaluations (*numEvals*) should typically be higher than the population size (*popSize*): high *popSize* values, combined with small *numEvals* values, will indeed reduce the heuristic to a blind random search. Each parameter has already a default value that has been empirically determined, though a dedicated tuning phase of *patternWeight* and *numEvals* is advisable in order to obtain the best performances.

6.1.3 Experimental task

This section presents the experimental tasks which J48SS has been evaluated on. First the considered datasets are described, which have been selected from a well-known, public repository. Then, the experimental workflow is detailed. Finally, the results are presented.

Datasets

All experiments are based on a selection of 16 UCR [78] datasets since, as witnessed by the recent literature, such a collection is a standard *de facto* in the machine learning community (see, for instance, [162, 192, 284, 292, 330]). Moreover, the chosen datasets make our results comparable to the ones presented in [284]. Table 6.2 details the number of training and test instances, and the length of the time series in the datasets under study. Observe that, although the considered time series have fixed length, J48SS is capable of handling also time series of heterogeneous sizes. Every dataset has been pre-processed by means of a dedicated *Python* script. As a result, each instance is described by the following set of heterogeneous attributes (other than the class label), so to try to exploit the capabilities of the novel algorithm:

- the original time series, encoded as a string;
- the minimum, maximum, average, and variance numerical values obtained from the original time series;
- the time series *skewness* and *kurtosis* [297] numerical values;

Table 6.2: The UCR datasets under study, with the tuned values of *patternWeight* and *numEvals*.

Dataset	|train|	|test|	TS length	# classes	patternWeight	numEvals
Adiac	390	391	176	37	0.4	800
Beef	30	30	470	5	0.5	800
ChlorineC	467	3840	166	3	0.9	800
Coffee	28	28	286	2	0.0	800
DiatomSize	16	306	345	4	0.0	400
ECGFiveD	23	861	136	2	0.4	200
FaceFour	24	88	350	4	0.5	200
GunPoint	50	150	150	2	0.8	800
ItalyPower	67	1029	24	2	0.6	800
Lighting7	70	73	319	7	0.9	800
MedicalIm	381	760	99	10	0.2	400
MoteStrain	20	1252	84	2	0.9	200
SonyAIBO	20	601	70	2	0.2	200
Symbols	25	995	398	6	1.0	800
Trace	100	100	275	4	0.5	400
TwoLead	23	1139	82	2	0.7	200

- the slope time series, obtained from the original one and encoded as a string;
- the minimum, maximum, average, and variance numerical values obtained from the slope time series;
- the slope time series *skewness* and *kurtosis* [297] numerical values.

Methods

Based on the selected UCR datasets, the performance of J48SS has been evaluated against the accuracy results presented in [284] (*Random Shapelet*). This work has been chosen for comparison as it is relatively recent and presents thorough details on the experimentation phase. Moreover, like in J48SS, it embeds the shapelet generation phase in the decision tree inducing algorithm. In short, Random Shapelet works by randomly selecting a certain percentage of shapelet candidates (with respect to all the possible shapelets obtainable from traning set data) to be evaluated in each node during the decision tree construction phase. Parameters such as the shapelet sizes to be considered have to be tuned in advance.

For each dataset, 100 single models have been built by varying the initial seed used by the evolutionary algorithm, in order to get statistically reliable results. The two parameters that are most likely to control the degree of generalization of the models have been tuned by means of averaging the results of 100-times 10-fold cross-validation on training data: first *patternWeight*, over the range $[0.1, 0.2, \ldots, 1.0]$; then, *numEvals*, considering the values $[200, 400, 800]$. Table 6.2 reports the adjusted values of the two parameters for each dataset. Note that, for time constraints, tuning was not performed on a grid-search, although such an approach would have allowed us, in principle, to find better combinations of the two values. All other parameters have been left at their default values.

Also, a series of experiments aimed at collecting statistics on the running time of the algorithm, the model sizes, and the generalization capability of the evolutionary approach have been conducted by varying the *numEvals* parameter only, which has been tested with the values $[200, 400, 800, 1600]$ (all other parameters have been left at their default values). Again, on each selected dataset and for each value of the parameter, 100 models have been built by varying the evolutionary algorithm seed in order to get statistically meaningful results.

Although the main focus of the work is on evaluating single, interpretable trees, it is also interesting to get at least some preliminary insights on the performances of ensembles of J48SS models. To this end, 100 ensemble models, each made of 100 trees, have been built through Weka's *Random Subspace* [161] method by varying the initial seed used by the evolutionary algorithm and keeping the tuned values for *patternWeight* and *numEvals*. The average accuracy results have been compared with those presented in [192] (*Generalized Random Shapelet Forests*) which, to the best of our knowledge, has been the only attempt so far of embedding shapelets in a forest inducing algorithm for time series classification (as an example, in [330], a shapelet extraction algorithm is used to transform the dataset *before* applying a random forest classifier). Similarly to the Random Shapelet approach, also in Generalized Random Shapelet Forests a random set of shapelet candidates is considered at each node. Moreover, taking inspiration from Random Forest [55], an ensemble of trees is built, considering a random subset of training instances for each tree.

Along with the accuracy results, computation times are reported, despite the fact that the referenced paper does not present any absolute computation time. Note that the Random Subspace function has a seed that should be independently changed with respect to the one used by the evolutionary algorithm in order to get proper statistically accurate results; nevertheless, this fact has been neglected due to time constraints. The same constraints led us to test ensembles made by only 100 trees, which is a relatively small number compared, e.g., to the 500 trees that have been suggested as a suitable forest size for traditional random forests [54].

Although we acknowledge the limits of such a preliminary experimentation, and look forward to thoroughly investigate the performances of different kinds of ensembles of J48SS models in future work, the obtained results are still useful to establish

a baseline.

Results

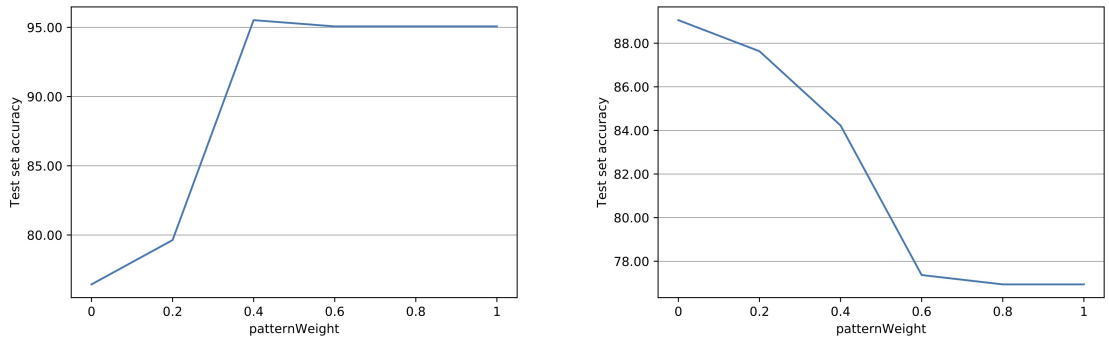
This section presents the results of the experiments carried out on the selection of UCR datasets, with respect to single trees and ensembles of J48SS models.

Results of single J48SS models Table 6.3 presents the results of the experimentation, along with the accuracies reported in [284] (*Random Shapelet*), which were also calculated over 100 executions of the algorithm. The *Random Shapelet* approach results are obtained by sampling different percentages of the total number of shapelets (ranging from 0.10% to 50%). The asterisk (*) symbol refers to tests that were reported not to converge in a reasonable amount of time (several weeks). Regarding J48SS, we may observe that, despite the mild tuning, the algorithm is capable of achieving an overall better accuracy result than *Random Shapelet*. Even better performances may be achieved by means of a dedicated full-parameter tuning phase, using a grid-search approach, and/or extending the search range. Nevertheless, observe that, on some datasets, J48SS has shown inferior performances with respect to the largest sampling variants of the competitor. However, as reported in [284], such higher performances of *Random Shapelet* are obtained at the price of long training times: typically hundreds of seconds are required for building a single model on the smaller datasets, and thousands of seconds (or worse) are needed on the bigger ones, when the larger sampling values are used. On the contrary, while J48SS training times tend to increase with the number of evaluations, they are relatively low. Specifically, Table 6.4 reports the average of the training times (in seconds) for the J48SS models on the 16 datasets. With respect to the importance of weight tuning, Figure 6.4 shows the average accuracy over 100 executions for the datasets *ECGFiveD* and *SonyAIBO*, over different values of *patternWeight* (all other parameters have been left at their default values). As can be seen, the two datasets behave very differently with respect to the values of the parameter: while in *ECGFiveD* the best performances are provided by high-performing shapelets (with respect to the Information Gain), *SonyAIBO* tends to favor simple patterns. This confirms the importance of tuning the value of *patternWeight*. Concerning the generalization capabilities of the evolutionary approach, Figure 6.5 shows the trend of the test set accuracy of the J48SS models, over the number of evaluations, for three selected datasets, leaving all other parameters at their default values. The typically observed behavior is the one of *DiatomSize*: the accuracy tends to grow as the number of evaluations increases, till it peaks, and then starts to decrease, due to overfitting in the evolutionary algorithm. There are also other two notable tendencies, that have been rarely seen and typical, for instance, of datasets *Beef* and *ECGFiveD*: in the former, the accuracy increases consistently, although we may speculate that it would peak, and then decrease if given a sufficiently high number of evaluations; in the latter, the accuracy decreases steadily, potentially suggesting that the chosen

Table 6.3: Accuracy of the Random Shapelet approach, compared with the results obtained by J48SS. Note that, when larger sampling is used in Random Shapelet, computation times may be orders of magnitude larger than those required by J48SS. Asterisk characters (*) represent missing values, due to too long computation times (several weeks).

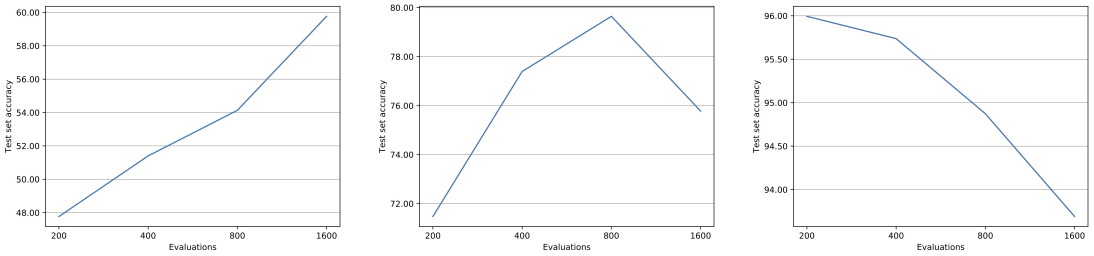
Dataset	**Random Shapelet**										**J48SS**
	0.10	**0.20**	**0.50**	**1.00**	**2.00**	**10.00**	**20.00**	**25.00**	**33.33**	**50.00**	
Adiac	45.4	47.7	49.6	50.6	51.9	51.6	51.3	51.7	50.3	51.8	**60.5**
Beef	40.0	39.3	36.8	35.5	33.6	32.4	31.6	32.0	31.8	31.2	**55.4**
ChlorineC	54.2	54.8	55.7	55.9	56.0	57.2	57.2	55.5	58.1	*	**61.4**
Coffee	69.0	70.3	73.1	74.1	76.4	76.9	78.1	78.3	78.3	78.3	**100.0**
DiatomSize	83.6	**85.4**	82.5	80.5	78.6	77.4	79.5	79.8	79.9	79.7	82.7
ECGFiveD	94.2	96.2	96.7	97.2	97.3	**97.7**	97.3	97.0	96.7	96.1	96.2
FaceFour	72.1	71.6	73.8	72.9	72.8	74.2	74.6	75.0	75.5	74.4	**84.5**
GunPoint	89.3	88.3	87.9	87.3	87.3	84.4	82.9	82.7	82.8	82.9	**91.8**
ItalyPower	87.1	87.8	90.0	90.4	90.8	92.4	93.0	93.0	93.1	**93.2**	92.7
Lighting7	61.9	61.0	58.1	56.9	56.9	63.50	63.5	63.9	64.9	63.0	**66.0**
MedicalIm	58.3	58.5	58.8	58.9	58.8	59.2	59.6	59.8	59.7	60.6	**65.8**
MoteStrain	78.4	78.9	79.2	79.2	79.2	81.5	81.8	**82.3**	82.2	82.0	81.5
SonyAIBO	83.8	84.9	85.5	86.9	86.3	86.8	87.2	87.9	88.6	**89.8**	89.2
Symbols	78.0	76.1	76.0	75.2	76.8	79.5	80.6	**80.8**	80.7	80.6	77.6
Trace	94.4	94.5	**95.0**	**95.0**	94.6	93.4	93.1	93.1	93.0	93.0	**95.0**
TwoLead	82.2	85.6	87.4	89.5	90.2	91.4	**92.1**	**92.1**	**92.1**	92.0	90.6
Average	73.2	73.8	74.1	74.1	74.2	75.0	75.2	75.3	75.5	N/A	**80.7**

fitness function is failing in avoiding the overfitting. As for the interpretability of the generated models, Figure 6.6 shows the average sizes in total number of nodes of the trees built on the 16 UCR datasets, together with their standard deviations. As can be seen, decision trees are typically small, with the evident exceptions of *Adiac*, *ChlorineC*, and *MedicalIm*. Of course, the size of the decision trees may be tuned by customizing two parameters of J48SS that govern the *pre-* and *post-pruning* aggressiveness (already present in J48). Figure 6.7 shows one of the J48SS models that have been trained on the *Beef* dataset. Starting from the root, the tree first checks the distance between the specific shapelet and the time series attribute named *TS* (the pattern may be easily visualized, for example by plotting its list of values). If such a value is less than 0.109855, then it tests the distance between another shapelet and the time series named *TS_slope*. If the distance is less than 0.01709, finally it assigns the class *2* to the instance. Also, observe that not only time series attributes are used by the tree, but also numerical ones, such as the *kurtosis* value



(a) Dataset *ECGFiveD* (b) Dataset *SonyAIBO*

Figure 6.4: Average test set accuracy of J48SS, with respect to different values of *patternWeight*.



(a) Dataset *Beef*. (b) Dataset *DiatomSize*. (c) Dataset *ECGFiveD*.

Figure 6.5: Average test set accuracy of J48SS, with respect to the number of evaluations of the evolutionary algorithm.

of the original time series and the *average* value of the slope time series. The specific decision tree has a test set accuracy of 53.33%, a value which is largely superior to the ones provided by the *Random Shapelet* approach. Figure 6.8 shows the shapelet tested at the root of the tree with respect to two time series of the train portion of dataset *Beef*, the first being representative of class *1*, while the second being typical of class *5*. As it can be seen, the time series having class *1* has a bigger distance than the one having class *5*, which is consistent with the splits taken in the decision tree. Finally, observe that, although in this case the shapelet is quite short with respect to the time series, this does not represent a common phenomenon in our experiments: in fact, other shapelets have emerged whose length is comparable to the one of the considered time series; indeed, the fitness function only indirectly influences the length of the generated shapelets.

Table 6.4: Average of J48SS training times (in seconds), given the number of evaluations of the evolutionary algorithm.

Dataset	200 evals	400 evals	800 evals	1600 evals
Adiac	14.1	20.8	37.8	98.9
Beef	3.7	5.2	7.72	14.8
ChlorineC	24.5	30.1	44.2	86.4
Coffee	1.1	1.4	2.0	4.0
DiatomSize	1.9	2.5	4.1	10.9
ECGFiveD	0.8	1.1	1.4	2.2
FaceFour	2.1	2.8	4.3	8.2
GunPoint	1.3	1.5	1.9	2.6
ItalyPower	0.8	0.9	1.3	2.4
Lighting7	4.6	6.6	10.2	18.6
MedicalIm	8.3	11.0	17.4	38.4
MoteStrain	0.6	0.7	0.9	1.3
SonyAIBO	0.6	0.8	1.0	1.9
Symbols	2.7	3.9	6.3	14.5
Trace	2.2	3.0	4.6	9.5
TwoLead	0.5	0.7	0.9	1.5

Results of Ensembles of J48SS Models

Let us now briefly report the outcomes of the experimentation with ensembles of J48SS models. Table 6.5 reports the average of the accuracies and training times of the generated models (*EJ48SS*), with respect to the accuracies reported in [192] (*Generalized Random Shapelet Forests*, *gRSF*). Again, note that the EJ48SS models are made of just 100 trees, with respect to the 500-tree forests used in gRSF. Although the results should just be considered as a baseline for future studies on J48SS ensembles, it is interesting to observe that in 8 out of 16 datasets J48SS ensembles have been able to match or outperform the performance gRSF, despite being much smaller considering the total number of trees. Moreover, taking into account the observations done about the importance of properly tuning the parameters of the single decision trees, it seems reasonable to assume that better results might be obtained without overly-increasing the size of the ensembles. As already stated before, such considerations, along with the assessment of different kinds of ensemble methods, are going to be addressed in future work.

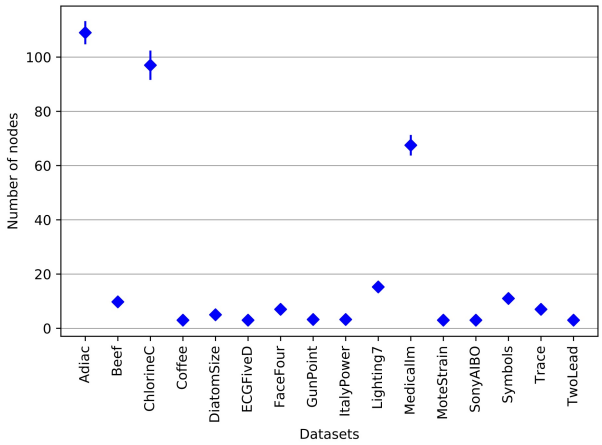


Figure 6.6: Average and standard deviation of the decision tree size per dataset.

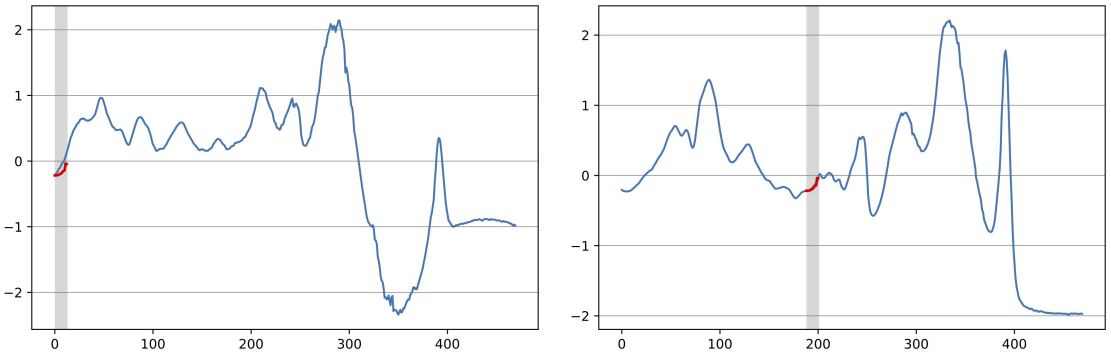
```
d(TS,[-0.21788,-0.21741,-0.21622,-0.21433,-0.20888,-0.20046,-0.19195,-0.18197,-0.15951,-0.14946,-0.13987,-0.036398,-0.04448]) <= 0.109855
|   d(TS_slope,[-0.080505,4.35E-4]) <= 0.01709: 2
|   d(TS_slope,[-0.080505,4.35E-4]) > 0.01709
|   |   d(TS_slope,[-0.0429]) <= 5.95E-4
|   |   |   kurtosis <= 0.197753: 5
|   |   |   kurtosis > 0.197753: 3
|   |   d(TS_slope,[-0.0429]) > 5.95E-4: 5
d(TS,[-0.21788,-0.21741,-0.21622,-0.21433,-0.20888,-0.20046,-0.19195,-0.18197,-0.15951,-0.14946,-0.13987,-0.036398,-0.04448]) > 0.109855
|   slope_avg <= -0.003305: 4
|   slope_avg > -0.003305: 1
```

Figure 6.7: One of the J48SS models trained on dataset *Beef*.

6.1.4 Discussion

The experimental results reported in the previous section allow us to conclude that J48SS is capable of achieving a competitive classification performance with respect to the task of time series data classification. Moreover, as a further advantage over previous methods, the trees built by the proposed algorithm are easily interpretable, and powerful enough to effectively mix decision splits based on several kinds of attribute. Finally, such a flexibility allows one to reduce the data preparation effort.

As a matter of fact, the datasets used for the experimental tasks did not take full advantage of the capabilities of the new algorithm, as they consist of time series only, and all static attributes are simply derived ones. In order to fully exploit the potentialities of J48SS, a dataset should contain sequential and/or time series data, and it should include meaningful static (numerical or categorical) attributes, somehow independent from the previous ones. The latter condition means that static attributes should not simply synthesize information already contained in the sequences/time series, but add something per se. Such a dataset has proven to be difficult to find in the literature, maybe because of the lack, until now, of an algorithm capable of training a meaningful model on it. Nonetheless, Section 6.2 describes the application of J48SS to a real analysis task in the context of pollution modelling, in which such a mixture of attributes can actually be found.



(a) Time series typical of class *1*

(b) Time series typical of class *5*

Figure 6.8: Two exemplary time series from the *Beef* dataset, with respect to the shapelet tested in the root of the decision tree of Figure 6.7. The best matching position according to the Euclidean Distance has been enlightened.

Table 6.5: Accuracy and training time (in seconds) of the J48SS ensemble models (100 trees), with respect to *Generalized Random Shapelet Forests* (500 trees).

Dataset	**gRSF acc.**	**EJ48SS acc.**	**EJ48SS tr. time (s)**
Adiac	74.2	**78.1**	2196.8
Beef	**80.0**	67.5	269.1
ChlorineC	67.3	**72.2**	2230.9
Coffee	**100.0**	**100.0**	49.0
DiatomSize	**96.4**	86.8	57.6
ECGFiveD	**100.0**	**100.0**	6.2
FaceFour	**100.0**	92.8	36.8
GunPoint	**100.0**	98.0	40.2
ItalyPower	94.0	**95.6**	19.3
Lighting7	69.9	**73.1**	410.5
MedicalIm	73.3	**75.5**	392.2
MoteStrain	92.1	**92.6**	6.5
SonyAIBO	**92.5**	90.4	6.4
Symbols	**96.8**	76.1	215.2
Trace	**100.0**	99.1	61.5
TwoLead	**100.0**	97.1	5.8
Average	**89.8**	87.2	

Algorithm 3 Node splitting procedure (J48SS)

```
1: procedure NODE_SPLIT(NODE)
2:     if NODE is "pure" or other stopping criteria met then
3:         make NODE a leaf node
4:     else
5:         best_attr ← null
6:         best_ng ← 0
7:         for each numerical or categorical attribute a do
8:             a_ng ←get information gain of a
9:             if a_ng > best_ng then
10:                best_ng ← a_ng
11:                best_attr ← a
12:        for each sequential string attribute s do
13:            pat, pat_ng ←get best frequent pattern in s
14:            if pat_ng > best_ng then
15:                best_ng ← pat_ng
16:                best_attr ← pat
17:        for each time series string attribute t do
18:            shap, shap_ng ←get shapelet in t using NSGA-II
19:            if shap_ng > best_ng then
20:                best_ng ← shap_ng
21:                best_attr ← shap
22:        children_nodes ← split instances in NODE on best_attr
23:        for each child_node in children_nodes do
24:            call NODE_SPLIT(child_node)
25:            attach child_node to NODE
26:    return NODE
```

6.2 Assessing the role of temporal information in air pollution modelling

In this work, we focus on assessing how different ways of handling temporal information may bring to different results in a real classification scenario. To do that, we consider an environmental problem, that is, the problem of identifying the relationships between the concentrations of the pollutants NO_2 (nitrogen dioxide), NO_X (nitrogen oxides), and $PM_{2.5}$ (particulate matter), and a set of variables, describing, among others, meteorological conditions and traffic flow, in the city of Wrocław in Poland. Specifically, we build on a previous work [187], where the authors tried to model such relationships by means of atemporal regression models. Instead, we consider the task as a classification one, which allows us to apply the recently proposed J48SS decision tree inducer (see Section 6.1), that, as we have seen, generates models capable of handling both temporal (sequences and time series) and atemporal (categorical and numerical) attributes. We compare its performance with those reported by J48 (WEKA's [331] implementation of Quinlan's C4.5 [274]) and Random Forest [55] runs on both an atemporal, as well as a transformed version of the original dataset, that makes use of lagged variables, showing that, at least in this case, time series and temporal sequences give rise to models with superior performance.

The content is organized as follows: Section 6.2.1 briefly discusses some previous work, and presents the original problem and dataset, as considered in [187]. Section 6.2.2 describes the data preparation step, and the differences with respect to the original work. Section 6.2.3 discusses the results that have been obtained with J48 and Random Forest neglecting temporal information, which are useful to establish a baseline. In Section 6.2.4, we formulate the problem with classical models, based on the use of lagged variables. In Section 6.2.5, we discuss the performance of J48SS on this problem. Finally, in Section 6.2.6 we provide an assessment of the work done and point out some possible future research directions.

6.2.1 Background

Over the recent years, machine learning-based environmental pollution studies have been gaining more and more traction in the scientific community. For instance, in [94], decision trees are used to predict $PM_{2.5}$ levels in the city of Quito, the capital of Ecuador, based on wind (speed and direction) and precipitation levels. In [332], the correlation between wind data and a wide range of pollutants in China's Pearl River delta region is investigated. In [296], Classification and Regression Trees (CART) and Ensemble Extreme Learning Machine (EELM) algorithms are used to model hourly $PM_{2.5}$ concentrations in the city of Yancheng, China. Finally, in [231], a machine learning approach based on two years of meteorological and air pollution data analysis is built to predict the concentrations of $PM_{2.5}$ from wind and precipitation levels in Northern Texas.

The present work builds on the findings presented in [187], where an environmental study conducted in Wrocław (Poland) is reported. The overall goal of the study was that of determining how the levels of specific pollutants, namely, NO_2, NO_X, and $PM_{2.5}$, are related to the values of other attributes, such as weather conditions and traffic intensity, with the purpose of building an *explanation* model (in opposition to a *prediction* model). In such an explanation model, the value of a pollutant at a certain time instant is linked to the value of the predictor attributes from the same time instant; discovering such a relationship is extremely important, as it may allow the city's government to identify the most critical factors that influence pollution levels, and to take them into consideration when developing proper corrective measures.

The considered dataset spans over the years 2015–2017, and it records information at one hour granularity. The dataset attributes are listed in Table 6.6: they are all numerical, with the exception of *holiday_or_not*, which has a binary categorical value, and *wind_direction_categ*, that may take the values N, NE, E, SE, S, SW, W, and NW. The attribute *traffic* refers to the number of vehicle crossings recorded at a large intersection equipped with a traffic flow measurement system, whereas the air quality information have been recorded by a nearby measurement station. We may broadly classify the predictors into three categories:

- *traffic intensity*: traffic;
- *timestamp*[1]: year, month, dom, dow, hour, date, holiday_or_workday;
- *weather conditions*: air_temp, wind_speed, wind_direction (categorical and numerical), rel_humidity, air_pressure.

Three atemporal Random Forest models have been trained on such data to perform a regression task on each of the three pollutants: NO_2, NO_X and $PM_{2.5}$. The conclusion was that in modelling nitrogen oxides concentration at a certain time, the most important variable is the traffic flow at the same time, while for $PM_{2.5}$ meteorological conditions seem to play a more predominant role. Starting from such a result, we now want to asses if more reliable explaining models can be found by taking into account the historical values of the predictor variables.

6.2.2 Data preparation

Let us now turn our attention to the description of the dataset used in this work. Although data are essentially the same as the one described in Section 6.2.1, it is important to underline once more that there is a fundamental difference between our work and the work reported in [187]: we consider a classification problem, instead

[1]We use the term *timestamp* to refer to the kind of variables that identify a specific time instant, to distinguish them from those we will consider to be proper temporal features, i.e., the ones that encode historical values.

Table 6.6: Features in the original dataset.

Feature	Description
year	year which the instance refers to
month	month which the instance refers to
dom	day of the month which the instance refers to
dow	day of the week which the instance refers to
hour	hour which the instance refers to
date	full date which the instance refers to
holiday_or_workday	whether the day which the instance refers to is a workday or not
traffic	hourly sum of vehicle numbers at the considered intersection
air_temp	hourly recording of the air temperature
wind_speed	hourly recording of the wind speed
wind_direction_num	hourly recording of the wind direction (numerical, degrees)
wind_direction_categ	hourly recording of the wind direction (discretized)
rel_humidity	hourly recording of the relative air humidity
air_pressure	hourly recording of the air pressure
NO2_conc	hourly recording of the NO_2 concentration level
NOX_conc	hourly recording of the NO_X concentration level
PM25_conc	hourly recording of the $PM_{2.5}$ concentration level

of a regression one. In that respect, the first step has been the choice of a sensible discretization technique for the pollutant attributes. To this end, we relied on official European Union directives [6, 11], that led us to define the following classes[2]:

- NO_2: intervals $[0, 40)$, $[40, 80)$, $[80, \infty)$;
- NO_X: intervals $[0, 40)$, $[40, 200)$, $[200, \infty)$;
- $PM_{2.5}$ intervals $[0, 25)$, $[25, \infty)$.

Such a discretization led to the class distributions depicted in Figure 6.9, which, as can be seen, are rather unbalanced. As a consequence, in order to evaluate the performances of the classifiers developed in this work, we shall use the *F1 score* [291], instead of the more common *accuracy* index. In short, F1 is defined as the harmonic mean of *precision* and *recall*:

$$F1 = \frac{2PR}{P + R} \, . \tag{6.2}$$

Thus, an F1 score reaches its best value at 1 (perfect precision and recall) and its worst at 0. Intuitively, it allows one to evaluate the balance between precision and

[2]Note that the European directives identify the two relevant values of 0 and 200 for NO_2 concentrations. However, we chose here to rely on different interval boundaries, since in the considered data there are just 4 instances with values over 200. Although this is a rather arbitrary choice, it does not compromise the goal of the work, namely, assessing the role played by temporal information in the overall classification task.

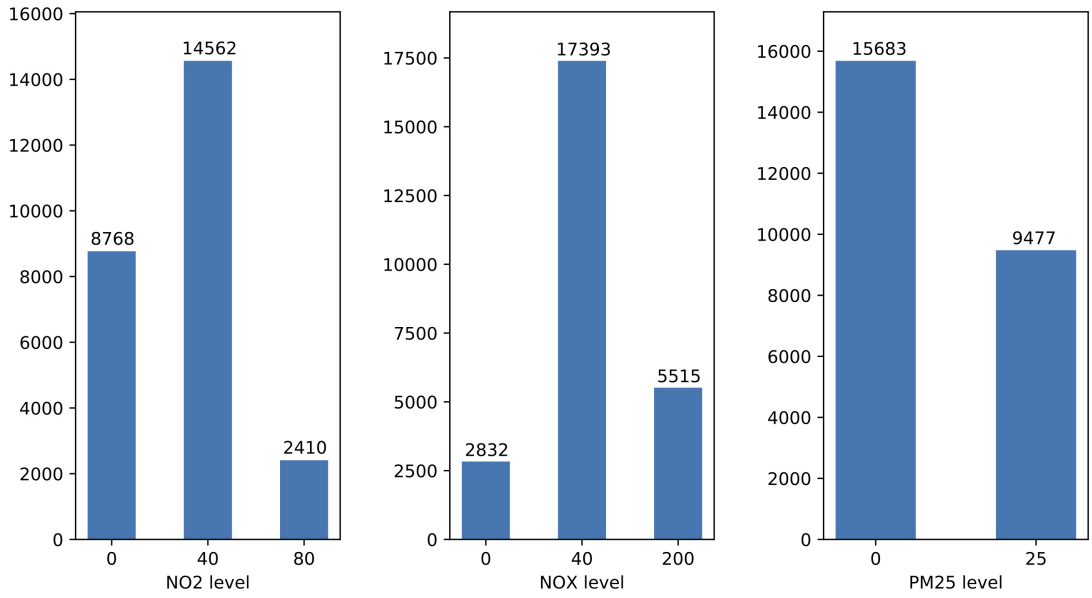


Figure 6.9: Class distributions (in number of instances) for the pollutants NO_2, NO_X, and $PM_{2.5}$. Labels on the x axis refer to the lower bounds of each discretization interval.

recall, which is extremely useful for evaluating classifiers performance in situations where there is an uneven class distribution, and false positive kinds of error have the same importance as false negative ones.

Let us now focus on the predictors, which have also undergone some changes with respect to the original features. First, a *season* attribute has been added, that (as it can be expected) tracks the season to which an instance refers (Winter, Spring, Summer, Autumn). Moreover, the timestamp attributes *month*, *dom*, *dow*, and *hour* have been replaced by two attributes each, based on the two trigonometric transformations:

$$SIN(2 * \pi * x/\delta) \quad , \quad COS(2 * \pi * x/\delta) \, , \tag{6.3}$$

where x represents the original value, and δ is the length of the period, e.g., 24 for the attribute *hour*, and 7 for the attribute *dow*. Relying on the two trigonometric transformations allows us to take into account the periodicity of the timestamp attributes. In this way, for instance, the value 11 pm hours becomes close to that of midnight and of 1 am hours. A similar motivation is behind replacing *wind_direction_num* by its two trigonometric transformations, although the idea here is not that of dealing with periodicity, but, again, that of treating in the same way wind directions that are close to each other, e.g., 359° and 1°. Figure 6.10 shows the result of such a transformation on the values of the attribute *wind_direction_num*. Observe

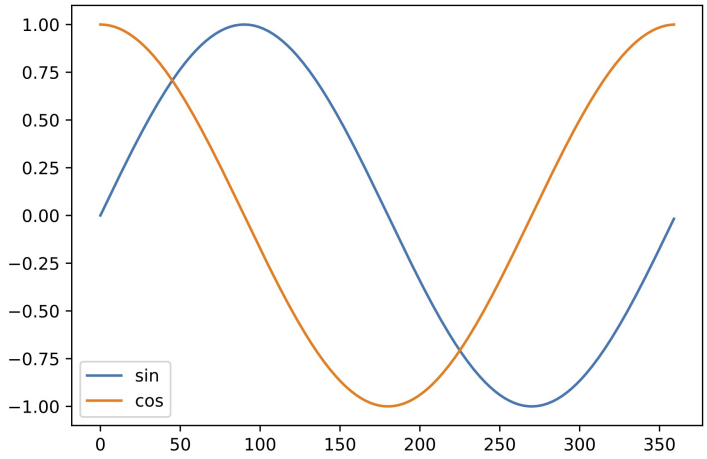


Figure 6.10: Result of the two trigonometric transformations applied to the attribute *wind_direction_num*.

that both trigonometric transformations are necessary in order to derive the original value, e.g., to distinguish 100° from 260°.

The final step consists of identifying which attributes might carry a temporal content, that is, which attributes can be interpreted as collection of historical values. They are *traffic*, *air_temp*, *wind_speed*, *wind_direction_num* (trigonometric transforms), *wind_direction_categ*, *rel_humidity*, and *air_pressure*. Sections 6.2.4 and 6.2.5 present two different approaches by which it is possible to model such temporal aspects.

6.2.3 Classification with atemporal data

Before delving into how temporal data may be handled, let us establish a baseline by developing a set of atemporal classification models. We consider WEKA's J48 decision tree learner and Random Forest ensemble technique, that will be compared to single J48SS trees and ensembles of J48SS trees, respectively. The considered predictors are thus *season*, *month* (trigonometric transforms), *dom* (trigonometric transforms), *dow* (trigonometric transforms), *hour* (trigonometric transforms), *holiday_or_workday*, *traffic*, *air_temp*, *wind_speed*, *wind_direction_num* (trigonometric transforms), *wind_direction_categ*, *rel_humidity*, and *air_pressure*, for a total of 18 attributes. The dataset has been partitioned into a training (66%) and a test (33%) set according to a stratified approach. Then, to account for the uneven class distribution, proper instance weights have been derived by means of Scikit-learn's compute_class_weight function [15], and have been used for model learning.

Three Random Forest models have been considered (Scikit-learn's implementation [16]), each performing classification on a different pollutant. For each of them,

Table 6.7: Hyperparameter search space and best parameters for Scikit-learn's RandomForestClassifier algorithm, over the three datasets.

Parameter	Search space	Atemporal	Lag_1_2_3_4	Lag_6_12_18_24
n_estimators	10, 25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 400, 500, 700, 1000	NO_2: 150	NO_2: 1000	NO_2: 175
		NO_X: 300	NO_X: 700	NO_X: 300
		$PM_{2.5}$: 1000	$PM_{2.5}$: 1000	$PM_{2.5}$: 1000
criterion	gini, entropy	NO_2: gini	NO_2: gini	NO_2: entropy
		NO_X: entropy	NO_X: entropy	NO_X: entropy
		$PM_{2.5}$: entropy	$PM_{2.5}$: entropy	$PM_{2.5}$: entropy
min_samples_split	2, 4, 6, 8, 10, 15, 20	NO_2: 15	NO_2: 20	NO_2: 20
		NO_X: 8	NO_X: 10	NO_X: 15
		$PM_{2.5}$: 2	$PM_{2.5}$: 2	$PM_{2.5}$: 4
max_features	sqrt, log2	NO_2: log2	NO_2: sqrt	NO_2: sqrt
		NO_X: log2	NO_X: sqrt	NO_X: sqrt
		$PM_{2.5}$: log2	$PM_{2.5}$: sqrt	$PM_{2.5}$: sqrt

we proceeded in the following way. To start with, we performed a hyperparameter tuning step by means of 3-fold cross-validation on training data. Table 6.7 reports the hyperparameters search space and the best performing ones, that have been then used to train the final model (column *Atemporal*). Observe that the Random Forest classifier does not have a deterministic behaviour, but relies on an initial seed to guide some random choices during the learning process. Thus, in order to get some confidence intervals over the test set performance, we trained 10 different models by varying the initial seed. We then considered the average and standard deviation of the F1 score, as shown in Figures 6.11 through 6.13 (label *RF_atemporal*). As a side note, observe also that the F1 score returned by each model is actually a macro average of the F1 scores calculated for each class. Similarly, we trained three single J48 decision trees over the same data. However, unlike the previous case, we did not perform any tuning over these models, because, as we shall see in Section 6.2.5, this allows us to perform a fair comparison with J48SS. Performance results for J48 are also reported in Figures 6.11 through 6.13 (label *J48_atemporal*). Note that, being J48 a deterministic algorithm, no standard deviation is observed.

As it could be expected, Random Forest is capable of obtaining F1 scores considerably higher than J48, in all three classification tasks. Moreover, standard deviations exhibited by Random Forest are quite small, which is the result of them being composed of a relatively large number of trees.

6.2.4 Classification with classical models and lagged variables

In this section, we report the outcomes of the classification by means of classical models and lagged variables. Recall that in Section 6.2.2 we identified the at-

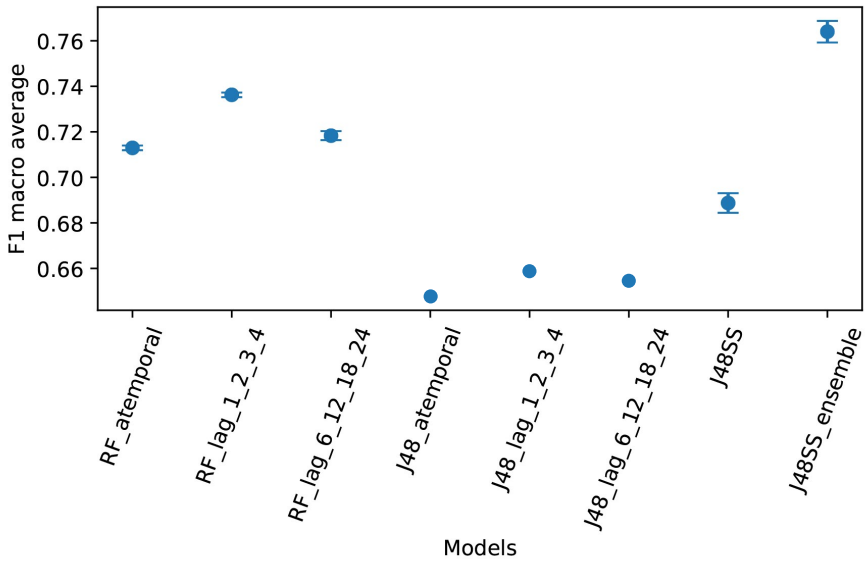


Figure 6.11: Average and standard deviation of the F1 macro average scores over the dataset NO_2.

tributes that may be susceptible of carrying temporal information, namely, *traffic*, *air_temp*, *air_pressure*, *wind_speed*, *wind_direction_num* (trigonometric transforms), *wind_direction_categ*, and *rel_humidity*.

We want to assess now the impact on the classification performances of models learned by J48 and Random Forest using the historical data of these attributes. We modeled the temporal aspect by means of lagged variables, which is a commonly used technique in the literature (see, for instance, [328]). In short, a lagged variable is a delayed variable that can be used to keep track of historical values for a given attribute. For instance, given the amount of traffic at the current instant, we may as well be interested in knowing how many vehicles crossed the same intersection two hours ago.

We started from the same set of attributes as described in Section 6.2.3 and we added, for each of the 8 mentioned attributes, a set of lagged variables. Specifically, we considered two different datasets, characterized by the use of either: *(i)* lagged variables for 1, 2, 3, and 4 hours before the current value, or *(ii)* lagged variables for 6, 12, 18, and 24 hours before. In both cases, the resulting dataset encompasses 50 attributes (18 atemporal + 32 lagged).

We trained three Random Forest and three J48 models, following the same tuning, training, and test protocol followed in Section 6.2.3. The results of the tuning phase are presented in Table 6.7 (columns *Lag_1_2_3_4*, and *Lag_6_12_18_24*), while their performances on the test set are depicted in Figures 6.11 through 6.13 (labels *RF_lag_1_2_3_4*, *RF_lag_6_12_18_24*, *J48_lag_1_2_3_4*, and *J48_lag_6_12_18_24*).

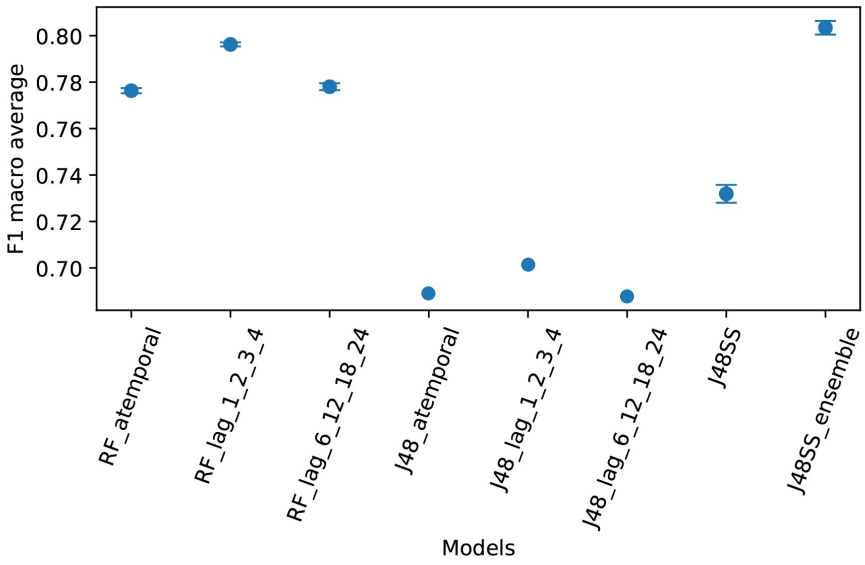


Figure 6.12: Average and standard deviation of the F1 macro average scores over the dataset NO_X.

We observe that the performances of the models trained on the lagged datasets tend to be higher than or equal to those of the ones built on the atemporal features only, with the exception of J48 on the $PM_{2.5}$ classification task; such a behaviour might be explained by the high number of attributes that the decision tree has to consider, and might be improved with an additional, intermediary feature selection step.

It should finally be observed that the choice of the number and granularity of lagged variables to employ is a rather arbitrary choice which may also be susceptible of tuning. As we shall see in the next section, relying on J48SS allows one to deal with temporal information in a much more natural way.

6.2.5 Classification with J48SS

J48SS makes it much easier to integrate temporal information starting from the atemporal dataset discussed in Section 6.2.3: for each of the 8 temporal attributes identified in Section 6.2.2, we simply build a string storing the past 24 values. Thus, we end up with a total of 26 features (18 atemporal + 8 temporal). Recall that J48SS makes use of a further parameter, denoted by $W \in [0, 1]$ (weight), to evaluate the *abstraction* level of each temporal pattern extracted from either a sequence or a time series and, thus, control overfitting (as suggested in [69]). Intuitively, larger values of W should lead to more powerful patterns being extracted, in terms of class discrimination performance (at least, on the training set). Conversely, smaller values of W should result in less complex, more general features being selected. We tuned

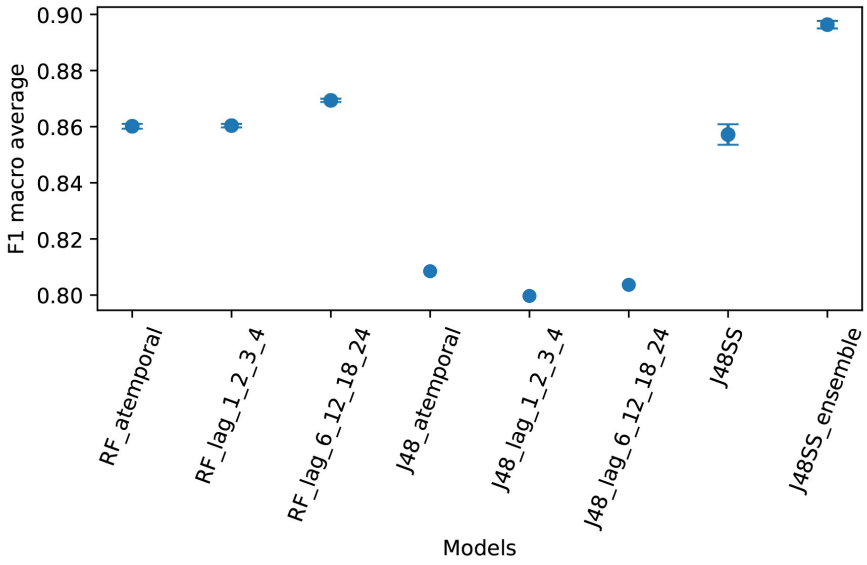


Figure 6.13: Average and standard deviation of the F1 macro average scores over the dataset $PM_{2.5}$.

W over a stratified training data split (considering 66% as actual training data, and 33% as validation data), finding that its best values are 0.9, 1.0, and 0.75 for NO_2, NO_X, and $PM_{2.5}$ classification tasks, respectively. Observe that we did not perform any tuning on the atemporal part of J48SS, which behaves exactly in the same way as the original J48. Then, for each prediction task, we averaged the performance on the test set of 10 single J48SS models; this is necessary since, as mentioned before, J48SS extracts time series shapelets by means of an evolutionary algorithm, which in turn is non deterministic and makes use of an initial random seed to guide the search. Results are depicted in Figures 6.11 through 6.13 (label *J48SS*).

As a final step, we considered ensembles of J48SS models. Specifically, we built three models, each composed of 10 (tuned) J48SS trees, relying on the WEKA's RandomSubSpace method [161]. In short, RandomSubSpace tries to build a decision tree ensemble that maintains the highest possible accuracy on training data, and improves on generalization accuracy as it grows in complexity. The trees in the ensemble are constructed systematically by pseudorandomly selecting random samples of features instead of the entire feature set, that is, the trees are constructed in randomly chosen subspaces. We did not perform any tuning over the ensemble method hyperparameters, relying on the default choice for the subspace size, i.e., 50%. As usual, the results, computed over the execution of 10 models, are presented in Figures 6.11 through 6.13 (label *J48SS_ensemble*).

As it can be seen, single J48SS trees score better than plain J48 models, although they typically perform worse than Random Forests. The observed standard devi-

ation is higher than that of Random Forest; however this is expected, being these models composed of just a single decision tree. More importantly, very small ensembles of J48SS trees are capable of achieving results higher than large Random Forest ensembles, that may in turn include up to 1000 trees. This is a clear indication that frequent patterns and shapelets are capable of extracting more information than lagged variables. Moreover, observed standard deviations are also reduced, which is a trend that is expected to continue as the ensemble size grows. Finally, using time series and sequences in J48SS allows one to reduce the arbitrariness related to the lagged variable approaches, although, as we have witnessed, one has still to decide the appropriate length for the temporal attributes' histories.

6.2.6 Discussion

In this work, we considered how different ways of encoding temporal information may impact on the performance of a classification task. We considered a real case scenario, that is, that of assessing the relationships between concentrations of the pollutants NO_2, NO_X, and $PM_{2.5}$, and a set of variables describing, among others, meteorological conditions and traffic flow in the city of Wrocław, in Poland. Through a series of experiments, we showed that accounting for the historical values of the features by means of lagged variables helps in improving the overall accuracy results. Moreover, an ever higher performance is obtained by relying on J48SS. Although the J48SS approach to the management of temporal data is quite natural and less complex than relying on lagged variables, some degree of arbitrariness still remains for what concerns the choice of the length of the histories for the temporal attributes. Moreover, it may be sensibly argued that the most straightforward way to deal with the consider pollution analysis would have been that of employing regression, instead of classification. In fact, this is a current limitation of J48SS, which may be extended in the future to deal with regression tasks.

Finally, observe that, although J48SS is capable of handling temporal information, it does so by testing existential conditions over single attributes (i.e., establishing the presence of a pattern). More powerful models could be developed by taking into account relationships between the values of different temporal features, a task on which we focus on in Section 7.1.

7

Combining Learning and Temporal Logic

In this chapter we discuss how machine learning and temporal logic can be combined, considering two possible aspects: (*i*) the case when temporal logic is used as a means inside of a machine learning algorithm to perform data analysis, as in the interval temporal logic-based decision tree presented in Section 7.1; and, (*ii*) a setting in which machine learning techniques are employed to generate temporal logic formulas, as in the task of natural language English utterance formalization, discussed in Section 7.2. Finally, as a conclusive development, in Section 7.3 we propose the architecture of a system, in which formal methods and machine learning techniques are seamlessly combined to perform anomaly detection and predictive maintenance tasks [60]. As we have already mentioned, we believe the integration between machine learning and statistical learning solutions with logics and formal methods techniques to constitute an original, thrilling research opportunity that promises to shed light on new ways of dealing with complex, real-world problems [172].

7.1 Interval temporal logic decision tree learning

As we have seen in Sections 5.1 and 6.1, J48SS is capable of handling categorical, numerical, sequential and time series data during the same execution cycle. This allows us to successfully handle analysis tasks characterized by heterogeneous datasets, such as the one described in Section 6.2. Nevertheless, the way in which the algorithm deals with temporal information has one, essential, limitation: the decision tree is just capable of handling existential conditions over single temporal attributes, that is, determining if a given instance satisfies a specific pattern (either a list of itemsets or a shapelet). A natural way to extend its capabilities is that of taking into account relationships between the values of different temporal attributes. This section discusses a convenient solution based on temporal logic (the original paper can be found in [71]).

A decision tree can be seen as a structured set of rules: every node of the tree can be thought of as a decision point, and, in this way, each branch becomes a

conjunction of such conditional statements, that is, a *rule* of the form $X_1 \wedge X_2 \wedge \cdots \wedge X_n \rightarrow C$, where the left-hand part represents tests on the instance's attributes and the right-hand part is the predicted class. A conditional statement may have many forms: it can be a yes/no statement (for binary categorical attributes), a categorical value statement (for non-binary categorical attributes), or a splitting value statement (for numerical attributes); the arity of the resulting tree is two if all attributes are binary or numerical, or more, if there are categorical attributes with more than two categories. Each statement can be equivalently represented with *propositional letters*, so that a decision tree can be also seen as a structured set of *propositional logic* rules.

Temporal classification: static solutions. As we have already discussed, just focusing on the static aspects of data is not always adequate for classification. Within static decision tree learning, temporal information may be aggregated in order to circumvent the absence of explicit tools for dealing with temporal information (for example, a patient can be labelled with a natural number describing how many times he/she has been running a fever during the observation period); the ability of a decision tree to perform a precise classification based on such processed data, however, strongly depends on how well data are prepared, and therefore on how well the underlying domain is understood. Alternatively, decision trees have been proposed that use *frequent patterns* [79, 122, 215] in nodes, considering the presence/absence of a frequent pattern as a categorical attribute [66, 67, 112] (such as J48SS). Nevertheless, as briefly stated before, despite being the most common approach to (explicit) temporal data classification, frequent patterns in sequences or series have a limited expressive power, as they are characterized by being *existential* and by intrinsically representing temporal information with instantaneous events.

Our approach: interval temporal logic decision trees. A different approach to temporal classification is mining temporal logic formulas, and since temporal databases universally adopt an interval-based representation of time, the ideal choice to represent temporal information in data is *interval* temporal logic. The most representative propositional interval temporal logic is Halpern and Shoham's Modal Logic of Allen's Relations [154], also known as HS. Its language encompasses one modal operator for each interval-to-interval relation, such as *meets* or *before*, and the computational properties of HS and its fragments have been studied in the recent literature (see, e.g. [57, 58, 59]). The very high expressive power of HS, as well as its versatility, make HS the ideal candidate to serve as the basis of a temporal decision tree learning algorithm. Based on these premises, we propose in this work a decision tree learning algorithm that produces HS-based trees. Our proposal, Temporal ID3, is a direct generalization of the ID3 algorithm [272], founded on the logical interpretation of tree nodes, and focuses on data representation and node generation; we borrow other aspects, such as splitting based on information gain and the overall learning process from the original algorithm. The accuracy of a decision tree and its resilience to over-fitting also depends on the stopping criterion

and possible post-pruning operations, but we do not discuss these aspects here.

Existing approaches to temporal logic decision trees. Learning temporal logic decision trees is an emerging field in the analysis of physical systems, and, among the most influential approaches, we mention learning of automata [31] and learning Signal Temporal Logic (STL) formulas [43, 72, 250, 279]. In particular, STL is a point-based temporal logic with *until* that encompasses certain metric capabilities, and learning formulas of STL has been focused on both the fine tuning of the metric parameters of a predefined formula and on learning the innermost structure of a formula; among others, decision trees have been used to this end [52]. Compared with STL decision tree learning, our approach has the advantage of learning formulas written in a well-known, highly expressive interval-based temporal logic language; because of the nature of the underlying language and of the interval temporal logic models, certain application domains fit naturally into this approach. Moreover, since our solution generalizes the classical decision tree learning algorithm ID3, and, particularly, the notion of information gain, it is not limited to binary classification only. Moreover, in [50] a first-order framework for TDIDT is presented with the attempt to make such paradigm more attractive to inductive logic programming (ILP). Such a framework provides a sound basis for logical decision tree induction; in opposition, we employ the framework to represent *modal*, instead of first-order, relational data. Additionally, our approach should not be confused with [230], in which the term *interval* indicates an uncertain numerical value (e.g., *the patient has a fever of 38 Celsius* versus *the patient has a fever between 37.5 and 38.5 Celsius*), and in which an algorithm for inducing decision trees on such uncertain data is presented that is based on the so-called Kolmogorov-Smirnov criterion, but the data that are the object of that study are not necessarily temporal, and the produced trees do not employ any temporal (logical) relation. In [33, 191, 317], the authors present two other approaches to a temporal generalization of decision tree learning. In the former, the authors provide a general method for building point-based temporal decision trees, but with no particular emphasis on any supporting formal language. In the latter, the constructed trees can be seen as real-time algorithms that have the ability to make decisions even if the entire description of the instance is not yet available. Finally, in [84], a generalization of the decision tree model is presented in which nodes are possibly labelled with a timestamp to indicate when a certain condition should be checked.

Summarizing, our approach is essentially different from those presented in the literature in several aspects. As a matter of fact, by giving a logical perspective to decision tree learning, we effectively generalize the learning model to a temporal one, instead of introducing a new paradigm. In this way, instances that present some temporal component are naturally seen as timelines, and, thanks to the expressive power provided by HS, our algorithm can learn a decision tree based on the temporal relations between values, instead of the static information carried by the values.

7.1.1 Preliminaries: interval temporal logic

Let $\mathbb{D} \subseteq \mathbb{N}$. In the *strict* interpretation, an *interval* over $\mathbb{D}$ is an ordered pair $[x, y]$, where $x, y \in \mathbb{D}$ and $x < y$, and we denote by $\mathbb{I}(\mathbb{D})$ the set of all intervals over $\mathbb{D}$. If we exclude the identity relation, there are 12 different Allen's relations between two intervals in a linear order [28]: the six relations R_A (adjacent to), R_L (later than), R_B (begins), R_E (ends), R_D (during), and R_O (overlaps), depicted in Figure 7.1, and their inverses, that is, $R_{\overline{X}} = (R_X)^{-1}$, for each $X \in \mathcal{A}$, where $\mathcal{A} = \{A, L, B, E, D, O\}$. Halpern and Shoham's modal logic of temporal intervals (HS) is defined from a set of propositional letters $\mathcal{AP}$, and by associating a universal modality $[X]$ and an existential one $\langle X \rangle$ to each Allen's relation R_X. Formulas of HS are obtained by

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle X \rangle \varphi \mid \langle \overline{X} \rangle \varphi \,, \tag{7.1}$$

where $p \in \mathcal{AP}$ and $X \in \mathcal{A}$. The other Boolean connectives and the logical constants, e.g., $\rightarrow$ and $\top$, as well as the universal modalities $[X]$, can be defined in the standard way. For each $X \in \mathcal{A}$, the modality $\langle \overline{X} \rangle$ (corresponding to the inverse relation $R_{\overline{X}}$ of R_X) is said to be the *transpose* of the modalities $\langle X \rangle$, and vice versa. The semantics of HS formulas is given in terms of *timelines* $T = \langle \mathbb{I}(\mathbb{D}), V \rangle$[1], where $\mathbb{D}$ is a linear order and $V : \mathcal{AP} \rightarrow 2^{\mathbb{I}(\mathbb{D})}$ is a *valuation function* which assigns to each atomic proposition $p \in \mathcal{AP}$ the set of intervals $V(p)$ on which p holds. The *truth* of a formula φ on a given interval $[x, y]$ in an interval model T is defined by structural induction on formulas as follows:

$$\begin{array}{lll}
T, [x,y] \Vdash p & \text{if} & [x,y] \in V(p), \text{ for } p \in \mathcal{AP} \,; \\
T, [x,y] \Vdash \neg\psi & \text{if} & T, [x,y] \nVdash \psi \,; \\
T, [x,y] \Vdash \psi \vee \xi & \text{if} & T, [x,y] \Vdash \psi \text{ or } T, [x,y] \Vdash \xi \,; \\
T, [x,y] \Vdash \langle X \rangle \psi & \text{if} & \text{there is } [w,z] \text{ s.t } [x,y] R_X [w,z] \text{ and } T, [w,z] \Vdash \psi \,; \\
T, [x,y] \Vdash \langle \bar{X} \rangle \psi & \text{if} & \text{there is } [w,z] \text{ s.t } [x,y] R_{\bar{X}} [w,z] \text{ and } T, [w,z] \Vdash \psi \,.
\end{array}$$

HS is a very general interval temporal language and its satisfiability problem is undecidable [154]. Our purpose here, however, is to study the problem of formula *induction* in the form of decision trees, and not of formula *deduction*, and therefore the computational properties of the satisfiability problem can be ignored at this stage.

7.1.2 Motivations

In this section, we present some realistic scenarios in which learning a temporal decision tree may be convenient, and, then, we discuss aspects of data preprocessing related to the temporal component.

[1] We deliberately use the symbol T to indicate both a timeline and an instance in a dataset.

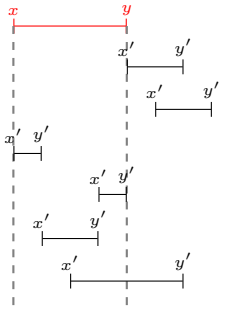
HS	Allen's relations	Graphical representation
$\langle A \rangle$	$[x,y]R_A[x',y'] \Leftrightarrow y = x'$	
$\langle L \rangle$	$[x,y]R_L[x',y'] \Leftrightarrow y < x'$	
$\langle B \rangle$	$[x,y]R_B[x',y'] \Leftrightarrow x = x', y' < y$	
$\langle E \rangle$	$[x,y]R_E[x',y'] \Leftrightarrow y = y', x < x'$	
$\langle D \rangle$	$[x,y]R_D[x',y'] \Leftrightarrow x < x', y' < y$	
$\langle O \rangle$	$[x,y]R_O[x',y'] \Leftrightarrow x < x' < y < y'$	

Figure 7.1: Allen's interval relations and HS modalities.

Learning. There are several application domains in which learning a temporal decision tree may be useful. Consider, for example, a medical scenario in which we consider a dataset of classified patients, each one characterized by its medical history, as in Figure 7.2, top. Assume, first, that we are interested in learning a *static* (propositional) classification model. The history of our patients, that is, the collection of all relevant pieces of information about tests, results, symptoms, and hospitalizations of the patient that occurred during the entire observation period, must be processed so that temporal information is subsumed in propositional letters. For instance, if some patient has been running a fever during the observation period, we may use a proposition $fever$, with positive values for those patient that have had fever, and negative values for the others (as in Figure 7.2, bottom, left). Depending on the specific case, we may, instead, use the actual temperature of each patient, and a static decision tree learning system may split over $fever < t$, for some threshold temperature t, effectively introducing a new propositional letter, and therefore a binary split. Either way, the temporal information is lost in the preprocessing. For example, we can no longer take into account whether $fever$ occurred before, after, or while the patient was experiencing headache (*head*), which may be a relevant information for a classification model. By generating, instead, the timeline of each patient (as in Figure 7.2, bottom, right), we keep all events and their relative qualitative relations. By learning a decision tree on a preprocessed dataset such as the one in Figure 7.2 (bottom, left), we see that the attribute *head* has zero variance, and therefore zero predictive capabilities; then, we are forced to build a decision tree using attribute $fever$ alone, which results in a classifier with 75% accuracy. On the contrary, by using the temporal information in the learning process, we are able to distinguish the two classes: $C1$ is characterized by presenting both *head* and $fever$, but not overlapping, and this classifier has, in this toy example, 100% accuracy. In this example, the term *accuracy* refers to the *training set accuracy* (we do not consider independent trainining and test data), that is, the ability of the classification system to discern among classes on the data used to train the system itself; it should not be confused with *test set accuracy*, which measures the real classification

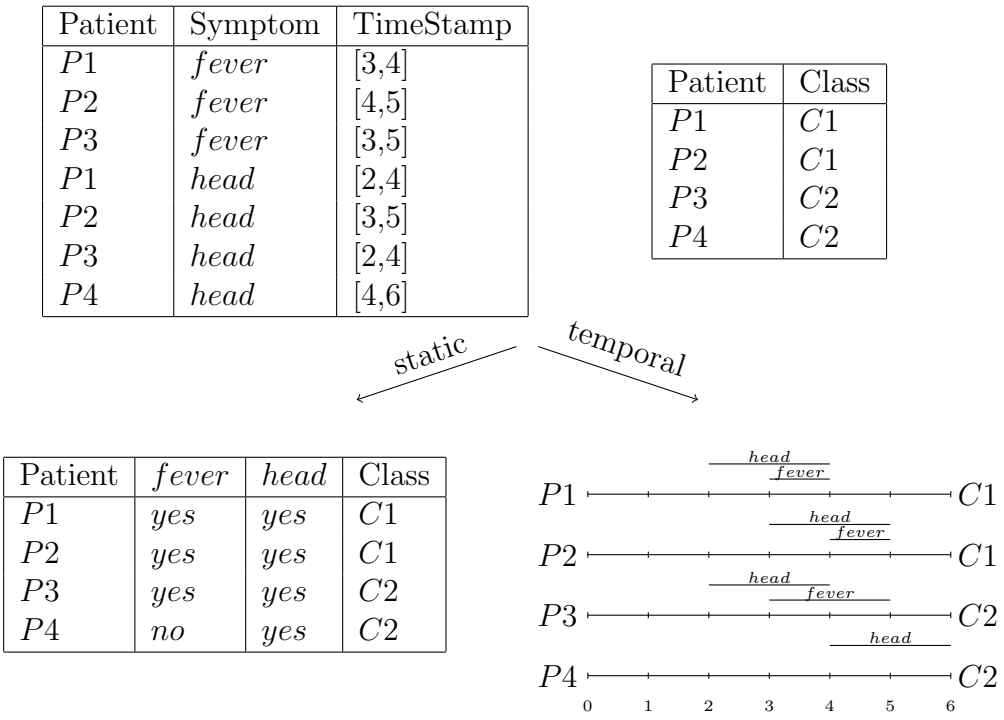


Figure 7.2: Example of static and temporal treatment of information in the medical domain.

performances that can be expected on future, real-life examples.

Alternatively, consider a problem in the natural language processing domain. In this scenario, a timeline may represent a *conversation* between two individuals. It is known that, in automatic processing of conversations, it is sometimes interesting to label each interval of time with one or more *context*, that is, a particular topic that is being discussed [29, 41, 270], in order to discover the existence of unexpected or interesting temporal relations among them. Suppose, for example, that a certain company wants to analyze conversations between selling agents and potential customers: the agents contact the customers with the aim of selling a certain product, and it is known that certain contexts, such as the price of the product (*price*), its known advantages (*advantages*) over other products, and its possible minor defects (*disadvantages*) are interesting. Assume that each conversation has been previously classified between those that have been successful and those that ended without the product being acquired. Now, we want to learn a model able to predict such an outcome. By using only static information, nearly every conversation would be labelled with the three contexts, effectively hiding the underlying knowledge, if it exists. By keeping the relative temporal relations between contexts, instead, we may learn useful information, such as, for example, *if price and disadvantages are not discussed together, the conversation will be likely successful.*

Preprocessing. Observe, now, how switching from static to temporal information influences data preparation. First, in a context such as the one described in our first example, numerical attributes may become less interesting: for instance, the information on *how many times* a certain symptom occurred, or its *frequency*, are not needed anymore, considering that each occurrence is taken into account in the timeline. Moreover, since the focus is on attributes *relative temporal positions*, even categorical attributes may be ignored in some contexts: for instance, in our scenario, we may be interested in establishing the predictive value of the relative temporal position of $fever$ and *head* regardless of the sex or age of the patient. It is also worth underlining that propositional attributes over intervals allow us to express a variety of situations, and sometimes propositional labelling may result in *gaining* information, instead of loosing it. Consider, again, the case of fever, and suppose that a certain patient is experiencing low fever in an interval $[x, y]$, say, a given day, and that during just one hour of that day, that is, over the interval $[w, z]$ strictly contained in $[x, y]$, he/she has an episode of high fever. A natural choice is to represent such a situation by labelling the interval $[x, y]$ with lo and its sub-interval $[w, z]$ with hi. On the other hand, representing the same pieces of information as three intervals $[x, w], [w, z], [z, y]$ respectively labelled with lo, hi, and lo, which would be the case with a point-based representation (or with an interval-based representation under the homogeneity assumption), would be unnatural, and it would entail hiding a potentially important information such as: "*the patient presented low fever during the entire day, except for a brief episode of high fever*". Building on such considerations, our approach is based on propositional, non-numerical attributes only.

7.1.3 Learning interval temporal logic decision trees

In this section we describe a generalization of the algorithm ID3 that is capable of learning a binary decision tree over a temporal dataset, as in the examples of the previous section; as in classical decision trees, every branch of a temporal decision tree can be read as a logical formula, but instead of classical propositional logic we use the temporal logic HS. To this end, we generalize the notion of information gain, while, at this stage, we do not discuss pre-pruning, post-pruning, and purity degree of a sub-tree [56, 273].

Data preparation and presentation. We assume that the input dataset contains timelines as instances. For the sake of simplicity, we also assume that all timelines are based on the same finite domain $\mathbb{D}$ of length N (from 0 to $N-1$). The dataset $\mathcal{T}$ can be seen as an array of n structures; $\mathcal{T}[j]$ represents the j-th timeline of the dataset, and it can be thought of as an interval model. Given a dataset $\mathcal{T}$, we denote by $\mathcal{AP}$ the set of all propositional letters that occur in it.

Temporal information. We are going to design the learning process based on the same principles of classical decision tree learning. This means that we need to define a notion of splitting as well as a notion of information conveyed by a split,

and, to this end, we shall use the truth relation as defined in Section 7.1.1 applied to a timeline. Unlike the atemporal case, splits are not performed over attributes, but, instead, over propositional letters. Splitting is defined *relatively to an interval* $[x, y]$, and it can be local, if it is applied on $[x, y]$ itself, or temporal, in which case it depends on the existence of an interval $[z, t]$ related to $[x, y]$ and the particular relation R_X such that $[x, y]R_X[z, t]$ (or the other way around). A *local split* of $\mathcal{T}$ into $\mathcal{T}_1$ and $\mathcal{T}_2$, where $[x, y]$ is the *reference* interval of $\mathcal{T}$, and p is the propositional letter over which the split takes place is defined by:

$$\begin{array}{rcl} \mathcal{T}_1 & = & \{T \in \mathcal{T} \mid T, [x, y] \Vdash p\} \ , \\ \mathcal{T}_2 & = & \{T \in \mathcal{T} \mid T, [x, y] \Vdash \neg p\} \ . \end{array} \tag{7.2}$$

On the contrary, a *temporal split*, in the same situation, over the temporal relation R_X, is defined by:

$$\begin{array}{rcl} \mathcal{T}_1 & = & \{T \in \mathcal{T} \mid T, [x, y] \Vdash \langle X \rangle p\} \ , \\ \mathcal{T}_2 & = & \{T \in \mathcal{T} \mid T, [x, y] \Vdash [X] \neg p\} \ . \end{array} \tag{7.3}$$

Consequently, the *local information gain* of a propositional letter p is defined as:

$$LocalGain(p, \mathcal{T}) = Info(\mathcal{T}) - \left((\frac{|\mathcal{T}_1|}{|\mathcal{T}|}) Info(\mathcal{T}_1) + (\frac{|\mathcal{T}_2|}{|\mathcal{T}|}) Info(\mathcal{T}_2) \right) \ , \tag{7.4}$$

where $\mathcal{T}_1$ and $\mathcal{T}_2$ are defined as in (7.2), while the *temporal information gain* of a propositional letter p is defined as:

$$TempGain(p, \mathcal{T}) = Info(\mathcal{T}) - \min_{X \in \mathcal{A}} \left\{ (\frac{|\mathcal{T}_1|}{|\mathcal{T}|}) Info(\mathcal{T}_1) + (\frac{|\mathcal{T}_2|}{|\mathcal{T}|}) Info(\mathcal{T}_2) \right\} \ , \tag{7.5}$$

where $\mathcal{T}_1$ and $\mathcal{T}_2$ are defined as in (7.3) and depend on the relation R_X. Therefore, the information gain of a propositional letter becomes:

$$Gain(p, \mathcal{T}) = \max\{LocalGain(p, \mathcal{T}), TempGain(p, \mathcal{T})\} \ , \tag{7.6}$$

and, at each step, we aim to find the letter that maximizes the gain.

The algorithm. Let us analyze the code in Figure 7.3. At the beginning, the timelines in $\mathcal{T}$ are not assigned any reference interval, and we say that the dataset is *unanchored*. The procedure *FindBestUnanchoredSplit* systematically explores every possible reference interval of an unanchored dataset, and, for each one of them, calls *FindBestAnchoredSplit*, which, in turn, tries every propositional letter (and, implicitly, every temporal relation) in the search of the best split. This procedure returns the best possible triple $< X, p, g >$, where X is an interval relation, if the best split is temporal, or it has no value, if the best split is local, p is a propositional letter, and g is the information gain. *Temporal ID3* first creates a root node, and then

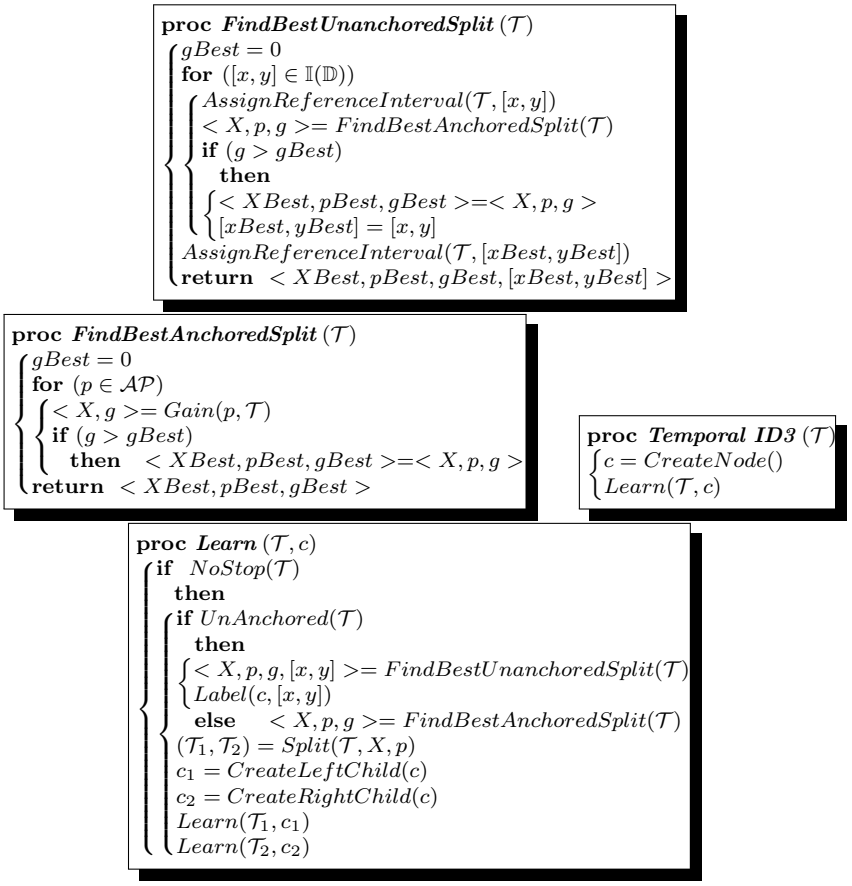


Figure 7.3: The algorithm Temporal ID3.

calls *Learn*. The latter, in turn, first checks possible stopping conditions, and then finds the best split into two datasets $\mathcal{T}_1$ and $\mathcal{T}_2$. Of these, the former is now *anchored* (to the reference interval returned by *FindBestUnanchoredSplit*), while the latter is still unanchored. During a recursive call, when $\mathcal{T}_1$ is analyzed to find its best split, the procedure for this task will be *FindBestAnchoredSplit*, called directly, instead of passing through *FindBestUnanchoredSplit*. So, in our learning model, all splits are binary. Given a node, the 'lefthand' outgoing edge is labelled with the chosen $\langle X\rangle p$, or just p, when the split is local, whereas the corresponding 'righthand' edge is labelled with $[X]\neg p$ (or just $\neg p$); also, the node is labelled with a new reference interval if its corresponding dataset is unanchored. After a split, every $T \in \mathcal{T}_1$ (the existential dataset, which is now certainly anchored) is associated with a new *witnessing* interval: in fact, those instances satisfy $\langle X\rangle p$ on $[x, y]$, and, for each one of them, there is a possibly distinct witness. Witnesses are assigned by the function *Split*; while the witnessing interval of an instance may change during the process, its reference interval is set only once.

Consider, now, the function *AssignReferenceInterval* and the example shown in Figure 7.4. As can be seen, neglecting the temporal dimension, one may classify the

Patient	Symptom	TimeStamp
$P1$	$fever$	[0,2]
$P2$	$fever$	[1,3]
$P3$	$head$	[0,2]
$P4$	$head$	[1,3]

Patient	Class
$P1$	$C1$
$P2$	$C1$
$P3$	$C2$
$P4$	$C2$

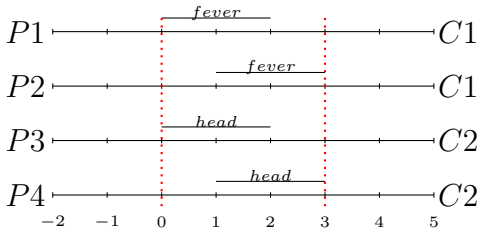


Figure 7.4: Example of a problematic dataset.

instances with just a single split based on the presence of the symptom fever (or headache). On the contrary, given the temporal dataset with domain $\mathbb{D} = \{0, 1, 2, 3\}$ it is not possible discriminate the classes within a single step. A natural solution consists of augmenting $\mathbb{D}$ in such a way to simulate the behaviour of an infinite domain model. In our example, it suffices to consider $\mathbb{D} = \{-2, -1, 0, 1, 2, 3, 4, 5\}$, so that a single split may be based on the rule: $\langle L \rangle fever \rightarrow C1$, otherwise $C2$ holding on $[-2, -1]$ (or, equivalently, its inverse formulation on $[4, 5]$). Thus, the function *AssignReferenceIntervals*, while searching all possible reference intervals, takes into consideration two extra points at each side of the domain. Although it is possible to obtain a similar result by adding less than four points (in our example, -2 and -1 suffice), this is no longer true if we include the possibility that Temporal ID3 is called on a *subset* of HS modalities, for example, for computational efficiency reasons. Adding four points, on the other hand, guarantees that the most discriminative split can always be found.

Analysis. We now analyze the computational complexity of Temporal ID3. To this end, we first compute the cost of finding the best splitting. Since the cardinality of the domain of each timeline is N, there are $O(N^2)$ possible intervals. This means that, fixed a propositional letter and a relation R_X, computing $\mathcal{T}_1$ and $\mathcal{T}_2$ costs $O(nN^2)$, where n is the number of timelines. Therefore, the cost of *FindBestAnchoredSplit* is obtained by multiplying the cost of a single (tentative) splitting by the number of propositional letters and the number of temporal relations (plus one, to take into account the local splitting), which sums up to $O(13nN^2|\mathcal{AP}|)$. The cost of *FindBestUnanchoredSplit* increases by a factor of N^2, as the **for** cycle ranges over all possible intervals, and therefore it becomes $O(13nN^4|\mathcal{AP}|)$. We can increase the efficiency of the implementation by suitably pre-computing the value of $\langle X \rangle p$ for each temporal relation, each propositional letter, and each interval, thus eliminating a factor of N^2 from both costs.

If we consider $\mathcal{AP}$ as fixed, and N as a constant, the cost of finding the best splitting becomes $O(n)$, and, under such (reasonable) assumption, we can analyze the complexity of an execution of *Learn* in terms of the number n of timelines. Two cases are particularly interesting. In the worst case, every binary split leads to a very unbalanced partition of the dataset, with $|\mathcal{T}_1| = 1$ and $|\mathcal{T}_2| = n - 1$ (or the other way around). The recurrence that describes such a situation is:

$$t(n) = t(n-1) + O(n) \ , \tag{7.7}$$

which can be immediately solved to obtain $t(n) = O(n^2)$. However, computing the worst case has only a theoretical value; we can reasonably expect Temporal ID3 to behave like a randomized divide-and-conquer algorithm, and its computational complexity to tend towards the average case. In the average case, every binary split leads to a non-unbalanced partition, but we cannot foresee the relative cardinality of each side of the partition. Assuming that every partition is equally probable, the recurrence that describes this situation is:

$$t(n) = \frac{1}{n}\sum_{k=1}^{n}(t(k) + t(n-k)) + O(n) \ . \tag{7.8}$$

We want to prove that $t(n) = O(n\log(n))$. To this end, we first prove a useful bound for the expression $\sum_{k=1}^{n} k\log(k)$, as follows:

$$\begin{aligned}
\sum_{k=1}^{n}(k\log(k)) &= \sum_{k=1}^{\lceil\frac{n}{2}\rceil-1}(k\log(k)) + \sum_{k=\lceil\frac{n}{2}\rceil}^{n}(k\log(k)) \\
&\leq \sum_{k=1}^{\lceil\frac{n}{2}\rceil-1}(k\log(\tfrac{n}{2})) + \sum_{k=\lceil\frac{n}{2}\rceil}^{n}(k\log(n)) \\
&= (\log(n)-1)\sum_{k=1}^{\lceil\frac{n}{2}\rceil-1} k + \log(n)\sum_{k=\lceil\frac{n}{2}\rceil}^{n} k \\
&= \log(n)\sum_{k=1}^{n} k - \sum_{k=1}^{\lceil\frac{n}{2}\rceil-1} k \\
&\leq \tfrac{1}{2}\log(n)n(n+1) - \tfrac{1}{2}\tfrac{n}{2}(\tfrac{n}{2}+1) \\
&= \tfrac{1}{2}(n^2\log(n) + n\log(n)) - \tfrac{1}{8}n^2 - \tfrac{1}{4}n \ .
\end{aligned}$$

Now, we prove, by induction, that $t(n) \leq an\log(n) + b$ for some positive constants a, b, as follows:

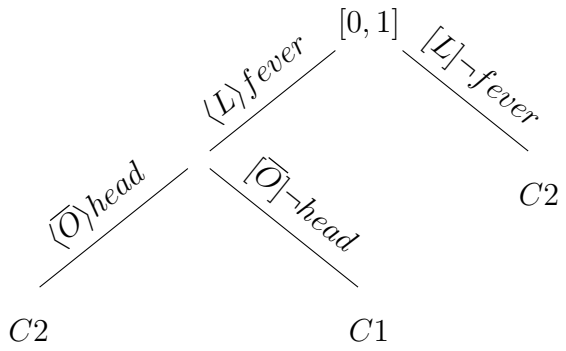


Figure 7.5: A decision tree learned by Temporal ID3 on the example in Figure 7.2.

$$
\begin{aligned}
t(n) &= \tfrac{1}{n}\sum_{k=1}^{n}(t(k)+t(n-k))+O(n) \\
&= \tfrac{2}{n}\sum_{k=1}^{n} t(k)+O(n) \\
&\leq \tfrac{2}{n}\sum_{k=1}^{n}(ak\log(k)+b)+O(n) && \text{inductive hypothesis} \\
&= \tfrac{2}{n}\sum_{k=1}^{n}(ak\log(k))+\tfrac{2}{n}\sum_{k=1}^{n} b+O(n) \\
&= \tfrac{2a}{n}\sum_{k=1}^{n}(k\log(k))+2b+O(n) \\
&\leq \tfrac{2a}{n}(\tfrac{1}{2}(n^2\log(n)+n\log(n))-\tfrac{1}{8}n^2-\tfrac{1}{4}n) \\
&\quad +2b+O(n) && \text{proved above} \\
&= an\log(n)+2a\log(n)-\tfrac{an}{4}-\tfrac{a}{2} \\
&\quad +2b+O(n) \\
&\leq an\log(n)+b && \text{if } \tfrac{an}{4} \geq 2a\log(n)-\tfrac{a}{2}+b+O(n)\ .
\end{aligned}
$$

Example of execution. Consider our initial example of Figure 7.2, with four timelines distributed over two classes. Since this is a toy example, there are many different combinations of intervals, relations, and propositional letters that give the same information gain. Figure 7.5 gives one possible outcome, generated by a Python-based prototype implementation of the algorithm, which seems to indicate that, looking at the entire history, the class $C2$ is characterized by presenting headache and overlapping fever, or no fever at all.

There are several running parameters that can be modulated for an execution of Temporal ID3, and further analysis is required to understand how they influence the final result, and, particularly, the properties of the resulting classifier. The most

important ones are: *(i)* how to behave in case of two splits with the same information gain; *(ii)* how to behave in case of more than one possible witness interval for a given timeline; *(iii)* how to behave in case of more than one optimal reference interval for a given unanchored temporal dataset. If we allow, in all such cases, a random choice, the resulting learning algorithm is not deterministic anymore, and different executions may result in different decision trees. This is a typical situation in machine learning (e.g., in algorithms such as *k-means clustering*, or *random forest*), that involves some experience in order to meaningfully assess the results.

7.1.4 Discussion

Machine learning is generically focused on a non-logical approach to knowledge representation. However, when learning should take into account (qualitative) temporal aspects of data, a logical approach can be associated to classical methods, and besides decision tree learning, interval temporal logics has been already proposed as a possible tool, for example, for temporal rules extraction [92]. Focusing these approaches on fragments of interval temporal logics whose satisfiability problem is decidable (and tractable) may result into an integrated systems that pairs induction and deduction of formulas, intelligent elimination of redundant rules, and automatic verification of inducted knowledge against formal requirement. Also, using a logical approach in learning may require non-standard semantics for logical formulas (e.g., fuzzy semantics, or multi-valued propositional semantics); these, in turn, pose original and interesting questions on the theoretical side concerning the computational properties of the problems associated with these logics (i.e., satisfiability), generating, *de facto*, a cross-feeding effect on the two fields. In this work, through the development of our interval temporal logic based decision tree prototype, we have taken a first step towards these research directions. As for future work, we plan to experiment with the use of Temporal ID3 on a selection of real datasets.

7.2 Synthesis of LTL formulas from natural language texts

In Section 7.1 we have looked at how temporal logic can be used as a means inside of a machine learning algorithm to help in classification tasks. In this section, we deal with the opposite direction, i.e., we consider a setting in which machine learning techniques are employed to generate temporal logic formulas. Specifically, we focus on the task of natural language English utterance formalization through linear temporal logic formulas (the original work can be found in [70]).

Linear temporal logic (LTL) is a formalism which is widely used in model checking (see, for instance, [137, 175]); moreover, LTL formulas make it possible to unambiguously describe the relationships among occurrences of events over time, a capability which turns out to be quite important in many automated reasoning fields. For these reasons, LTL can be considered as a preferred means to formalize and then reason upon natural language texts, which is most useful in tasks such as requirement specification and, more generally, the analysis of temporally-declined semantic content in texts [307].

Despite of their usefulness, the correct specification and interpretation of temporal logic formulas typically requires a strong mathematical background, and this severely limits their applicability by untrained domain experts. In such contexts, a system capable of automatically translating between English and temporal logic would be of great help.

Concerning the LTL-to-English translation, it may be achieved in a relatively easy way, by simply parsing the logical formula by means of an attribute grammar and applying some heuristics to make the English translation sound as natural as possible [282]. As for the opposite direction, several issues have to be dealt with, such as the inherent ambiguity of natural language, the possible lack of an explicit context in the sentences, or reference to background knowledge which may not be included in the utterance. For all these reasons, the task of translating a free, unconstrained English text into a free, unbounded LTL formula is still an open problem, although several partial solutions have been proposed over the years (see Section 7.2.2).

In this part of the thesis, we give an overall assessment of the state of the art for what concerns English-to-LTL translation. In addition, we make a critical evaluation of some of the currently available tools which may be adapted and combined for such a task, and we outline some possible future research directions.

The work is organized as follows. In Section 7.2.1, we introduce the problem of English-to-LTL translation. Then, in Section 7.2.2, we provide a short state of the art. Next, in Section 7.2.3, we outline possible future research directions. Finally, in Section 7.2.4, the outcomes of an experimental evaluation of some of the most significant tools proposed in the literature that may be used to develop an English-to-LTL translation architecture are reported. Finally, a brief discussion summarizes the main contributions of the work done.

7.2.1 Problem definition

The problem of extracting temporal logic formulas from a natural language utterance may be considered as an instance of the *Semantic Parsing* task, that is, the process of mapping a natural-language sentence into a formal, typically machine-understandable representation of its meaning [329].

It must be observed that such a problem differs from the one of inferring logical formulas from examples. In the latter, a dataset of logs, or traces, resulting from the execution of a given system is considered, with the goal of determining which formulas characterize and distinguish between examples that describe a good behaviour of the system from those that do not (see, for instance, [247]).

In the present work, we focus on *Linear-Time Temporal Logic* (LTL for short) [267]. An LTL formula is built from a finite set of proposition letters by making use of Boolean connectives and the temporal modalities **X** (next) and **U** (until). Formally, LTL formulas can be defined as follows:

- if $p \in \mathcal{AP}$, then p is an LTL formula;
- if ψ and ϕ are LTL formulas, then $\neg\psi$, $\phi \vee \psi$, $\mathbf{X}\psi$, and $\phi \mathbf{U} \psi$ are LTL formulas.

On the basis of these temporal operators, additional modalities can be defined, most commonly the Boolean connectives $\wedge$ (and), $\rightarrow$ (imply), **true**, and **false**, and the temporal modalities **G** (globally) and **F** (eventually).

As for the semantics, let $w = a_0, a_1, a_2, \ldots$ $(a_i \in 2^{\mathcal{AP}})$ be an infinite sequence of truth evaluations for proposition letters in $\mathcal{AP}$, and let $w(i)$ denote the truth evaluation at position i. The satisfaction relation $\models$ between w and an LTL formula can be defined as follows:

- $w \models p$ if $p \in w(0)$;
- $w \models \neg\psi$ if $w \not\models \psi$;
- $w \models \phi \vee \psi$ if $w \models \phi$ or $w \models \psi$;
- $w \models \mathbf{X}\psi$ if $w(1) \models \psi$;
- $w \models \phi \mathbf{U} \psi$ if $\exists\, i \geq 0$ s.t. $w(i) \models \psi$ and $\forall\, 0 \leq k < i$ $w(k) \models \phi$.

The semantics of the derived modalities is then defined as follows:

- $\mathbf{true} \equiv p \vee \neg p$;
- $\mathbf{false} \equiv \neg\mathbf{true}$;
- $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$;
- $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$;

- $\mathbf{F}\psi \equiv \mathbf{true}\mathbf{U}\psi$;
- $\mathbf{G}\psi \equiv \neg\mathbf{F}\neg\psi$.

As an example, consider the following statement:

> *"After the button is pressed, the light will turn red until the elevator arrives at the floor and the doors open."*

Such a situation can be captured by the following LTL formula:

$$p \rightarrow \mathbf{X}\ (\ q\ \mathbf{U}\ (\ s \wedge v\ )\ )$$

where p, q, s, and v are proposition letters corresponding to the button being pressed, the light turning red, the elevator arriving, and the doors opening, respectively.

7.2.2 A short state of the art

Over the years, several authors have devised methodologies to translate English sentences into LTL formulas. However, they typically do not deal with the most general case of translating natural, unbounded, English sentences into general, unbounded LTL formulas: assumptions are made that either reduce the generality of the input text or the output formula.

In [103], the authors present a pattern-based approach to the encoding of property specifications for finite-state system verification. Among other contributions, they provide a set of 55 LTL-based formulas that should capture typical patterns that come into play in the design of concurrent and reactive systems, such as the existence of a specific condition or the conditional response to a stimulus. A detailed description of such patterns, together with a repository of examples, are available on the SAnToS laboratory website [8]. Such a set of patterns has been later extended to deal with more advanced scenarios, such as, for instance, timed property specifications [150] and real-time systems [197].

In [252], the authors start from the LTL repository available in [8], identifying the 8 patterns that, according to their analysis, are the most frequently used in temporal requirement specification, covering over 80% of the cases they encountered, that all lie in the aerospace domain. Then, they develop a set of shallow classifiers (such as Random Forests [55] and Support Vector Machines [86]) that are capable of detecting the presence of such patterns in textual technical specifications. As for the predictor attributes, they make use of a bag-of-words approach, enriched with information derived from Part of Speech (PoS) tagging. It should be observed that no solution is given to the problem of instantiating a detected pattern on the specific textual data. Thus, such an approach is not able of deriving a complete translation from English to the restricted set of LTL formulas they consider.

In [343], an algorithm is presented to translate a property, which is specified by making use of a predefined subset of English (referred to as *controlled English)*,

to LTL. The approach relies on syntactical properties only, being based on text processing techniques like grammatical dependency parsing.

In [213], LTL is used as a formalization technique within an integrated system for generating, managing, and executing controllers for autonomous robots. The idea is that a user should be capable of instructing a robot by means of natural language, so no assumption is made on the form of English text that is given as input to the system. A textual instruction is processed making use of tools for speech tagging, dependency parsing, and null element restoration (see, for instance, [128]). Then, all verbs are identified, together with their arguments. Finally, each verb is mapped into a set of *senses* in VerbNet [293]. Although in principle no restriction is made on the kind of input phrases that a user may submit to the system, in practice the translation is based on a mapping between *senses* and LTL formulas, that is in turn based on different, manually specified, combinations of so-called *macros*.

In [335], a framework for requirement consistency management is presented. In order to formalize requirements, the system automatically translates natural language descriptions of functionalities into a logic representation. However, a restricted English grammar is considered, to avoid problems such as semantic ambiguity and to make the overall translation process easier.

In [139], the ARSENAL framework for translating natural language requirements into analyzable formal representations is discussed. Given an input sentence, the first step exploits both domain-specific and domain-independent knowledge to identify entity n-grams, such as, for instance, "Lower Desired Temperature", which are then converted into single terms like, e.g., `Lower_Desired_Temperature`. In addition, common expressions, such as "is greater than or equal to", are converted into simple terms, like `dominates`, in an attempt to regularize the input text and reduce its complexity. Next, a dependency parsing step is performed to extract grammatical relationships between phrase elements. These pieces of information, together with PoS tags, are then fed to a so-called *semantic processor*, that guides the translation from English to an internal, intermediate representation. It should be observed that such a module relies on a set of hand-made rules, that inevitably are domain-specific and can only work for restricted scenarios. Finally, the intermediate representation can be converted into several formalisms, including LTL.

Last but not least, a *controlled natural language* is defined in [290], which can be used to specify restrictions on how a system model interacts with its environment. Sentences formulated in such a constrained language are then automatically translated to LTL by means of hard-coded rules.

Broadening the attention scope to other kinds of (temporal) logical formalisms, the following contributions are worth mentioning.

In [113], the authors present a tool for the automatic translation of natural language sentences into formulas of the action-based temporal logic ACTL, in the context of the formalization of reactive systems requirements. A corpus of sentences is analyzed and, starting from it, a context-free grammar is manually built that

allows one to parse the considered instances. Such a grammar is augmented with attributes, that allow one to generate the target ACTL formula during the parsing process.

In [248], a translation method is discussed that converts constrained English utterances into a representation level based on Kamp's *Discourse Representation Theory* [188]. Such an intermediate representation is then translated into an ACTL temporal logic formula.

In [104], the authors present a methodology to translate natural language instructions into a formal logical description of goals, that can then be issued to and followed by robots. The target formalisms are CTL* and first-order dynamic logic (FDL). The translation process is carried out by means of a hand-made combinatory categorial grammar [303].

In [155], hand-made attribute grammars are employed to generate Computation Tree Logic (CTL) formulas from Hardware Description Language (HDL) code comments written in English.

Finally, in [306], a two-phase approach to parsing natural language into formal logic is presented. The first phase aims at extracting the generic structure of the logical expression associated with a given natural language utterance. The general idea is that of building a dependency parse of the input utterance, and then attaching to its nodes λ-expressions chosen from a pre-specified finite set, by means of a specifically trained model. The node assignment to λ-expressions is then evaluated to a generic logical form structure. The second phase instantiates the discovered pattern on the specific utterance data. Some considerations are made about the possibility of adopting LTL as a target formalism in future work.

Overall, the typical approach followed by these studies can be summarized as follows: given an input English utterance, preprocess it to extract syntactical information, which may include part of speech tagging, dependency parsing, semantic role labelling, and so on. Then, enrich the input with these pieces of information. Finally, run an attribute grammar-based parser, or rely on some hand-made rules, to derive a translation into a target logical format. A notable exception is the work of [306], where a fully-supervised learning setting is considered.

7.2.3 Possible future research directions

In the following, on the basis of the analysis of related work done in the previous section, we outline some possible approaches to the problem of translating open English utterances into general LTL formulas. For each of them, we identify existing tools and solutions from the literature that can possibly be exploited.

LTL synthesis via classical semantic parsing

Classical approaches to semantic parsing involve the definition of a suitable grammar, which it is possible to rely on to parse a given phrase, and to contextually

generate a desired output. Typically, attribute grammars or combinatory categorial grammars have been considered in the literature for this task (see Section 7.2.2), although specifically developed grammars have also been proposed [246].

As already pointed out, the problem of generating LTL formulas from English texts can be viewed as an instance of semantic parsing. Some frameworks that allow one to develop semantic parsers are available in the literature, most notably KRISP [193], SEMPRE [46], Cornell Semantic Parsing Framework [38], SippyCup [18], and WASP [22].

However, the main difficulty remains that of defining a suitable grammar, a task that for real-world applications cannot be performed by hand, especially when, instead of restricting to a specific domain, we refer to a general translation scenario. In an attempt to facilitate this task, several approaches to grammar induction have been proposed over the years. Some of them are specifically oriented to natural language processing, like, for instance, those presented in [100, 156, 302, 345]; others are more general, e.g., [203, 258]. Nevertheless, they all require a quite large amount of training data, a problem that is shared also by the (more promising) strategy presented in the next section.

LTL synthesis as a translation problem

Machine translation can be viewed as the use of software to translate text or speech from one language to another. Several approaches to translation have been proposed over the years.

The first proposed one was the so-called *rule-based paradigm*, in which explicit rules are given that guide the translation. The translation can be done in two different ways: either directly mapping the source language into the target one, or relying on some sort of interlingual representation. Of course, the richness and complexity of natural language imply that such handcrafted approaches are only suitable for very constrained domains. Nevertheless, some platforms for rule-based translation are available, such as, for instance, Apertium [120].

Subsequently, *statistical machine methods* have gained in popularity. In such a case, translations are not generated by ad-hoc developed rules anymore, but are, instead, generated on the basis of statistical models, whose parameters are trained on bilingual text corpora.

At the moment, the most popular approach is the one based on *neural machine translation*, where a large artificial neural network is used to predict, given an utterance in the source language, the likelihood of a sequence of words in the target language. Several frameworks have been proposed in the literature to develop general-purpose neural networks, such as, for instance, PyTorch [14], Tensorflow [20], and Keras [9]. Recently, OpenNMT [195] has been presented as an open source toolkit specifically oriented to neural machine translation. The system prioritizes efficiency, modularity, and extensibility, and it allows one to develop state-of-the-art solutions, based on Convolutional Neural Networks, LSTM, attention mechanisms,

and word- as well as character-based embeddings, through a simple general-purpose interface, that in principle requires only source and target files to be provided. Besides language translation, the framework also supports other sequence generation tasks, such as, for example, summarization, image-to-text and speech recognition[2].

It is worth pointing out that most of the translation models from the literature are based on a sequence-to-sequence architecture [309]. Nevertheless, since a logic formula can be represented by a tree-like structure, it might be better to opt for a sequence-to-tree approach [223], or for other *constrained decoding* techniques (see, for instance, [17]).

Since the English-to-LTL mapping can be viewed as a language translation problem, it makes sense to think of training a neural network model to perform such a task. The best candidate framework for doing that seems to be, to date, the previously-discussed OpenNMT. Unfortunately, a major problem has to be solved in order to actually pursue this approach, that is, the lack of a large training dataset in which English utterances are paired with the corresponding LTL formulas. The University of Kansas provides an online repository of temporal logic formulas that represent the content of technical specifications [8]. However, just about a hundred of English-to-LTL pairings are available. LTLStore[3] is an online repository of around one thousand LTL formulas, collected from previous studies. Although they lack an English counterpart, they may still be useful as *monolingual* data [294].

Various approaches can be followed to address the problem of the scarcity of training data. We would like to outline three of them.

The first, most trivial one consists of randomly generating a large set of LTL formulas by means of a suitable grammar. Then, rules may be derived to translate each formula into an English utterance. Also, some post-processing techniques can be applied to make the resulting text more realistic, such as replacing proposition letters by a set of words that represent them (e.g., `button_pressed` might be mapped to *"the button is pressed"*). Of course, the generated sentences would still make use of a very constrained kind of English. Nevertheless, such an artificial training dataset may be used as a starting point to determine which are the most promising translation techniques.

The second approach consists of finding a kind of "bridging dataset", i.e., a tranining dataset that maps each English utterance in a formal representation that, in turn, can be more or less easily converted into LTL by means of hardcoded rules. As an example, it may be worth investigating the data used in semantic parsing competitions, such as, for instance, SemEval[4], that over the years included some tracks concerned with the evaluation of temporal content in natural language phrases (this is the case with TempEval). Other possibly useful resources are the TimeML [271] annotated corpus TimeBank [283] and AQUAINT [2], ACE (Automatic Content Ex-

[2]Additional details can be found on OpenNMT website: `http://opennmt.net/`.
[3]LTLStore project website: `https://gitlab.lrz.de/i7/ltlstore`.
[4]SemEval 2019 website: `http://alt.qcri.org/semeval2019/`.

traction) [1] and WikiWars [229] corpora, that all provide texts manually labelled with events, temporal expressions, and relationships.

A third recent research line has managed to train both neural machine and statistical machine translation systems using monolingual corpora only [36, 37, 201, 202]. Although accuracy results are still far from those guaranteed by state of the art, bilingual data-based models, such approaches are worth some further investigation.

LTL synthesis through evolutionary computation

Both LTL synthesis via semantic parsing and LTL synthesis via machine translation follow consolidated paths in the field. Now, we work out an alternative path that makes use of evolutionary computation.

Recall, from Section 1.4, that *Evolutionary Algorithms* (EAs) are adaptive meta-heuristic search algorithms, inspired by the process of natural selection, biology, and genetics. Unlike blind random search, they are capable of exploiting historical information to direct the search into the most promising regions of the search space and, in order to achieve that, their basic characteristics are designed to mimic the processes that, in natural systems, lead to adaptive evolution [105].

The English-to-LTL mapping problem may be thought of as an optimization one (that in turn can be solved via evolutionary computation): given an input utterance, the task is that of finding the temporal formula that best matches the content of the text.

In the following, the main issues concerning the exploitation of an evolutionary algorithm for the English-to-LTL mapping problem are discussed. They can be summarized as follows: (*i*) how the single solutions are represented; (*ii*) how the population is initialized; (*iii*) which evolutionary operators (crossover, mutation) are employed; (*iv*) which fitness function is used.

Solution representation and population Each instance in the population represents an LTL formula, which may be conveniently coded by means of a tree-based data structure. As for the initialization of the population, one may proceed as follows. In the first step, all proposition letters are extracted from the given utterance. This can be done by means of a suitable predicate extraction process, that can rely on available tools, such as, for instance, PredPatt [12, 339], or on custom-tailored solutions based on *Semantic Role Labeling* or *Open Information Extraction*, making use of natural language processing frameworks such as Stanford CoreNLP [226], the SpaCy-based [19] AllenNLP [134], and TextPro [266]. Once all proposition letters have been extracted, candidate LTL formulas may be randomly generated, ensuring both to use all of the propositional letters (since the goal is that of fully encoding what is happening in the utterance), and to respect LTL syntactical correctness constraints.

Evolutionary operators The representation of LTL formulas by means of tree-like structures makes it quite straightforward to implement the crossover and mutation operators. Both of them can, indeed, modify or generate a solution by acting on subtrees, e.g., by adding, removing, raising, or swapping them, always making sure that each proposition letter is appearing at least one time in the resulting formula.

Fitness function In order to establish how good a given formula is, it is necessary to determine how well it captures the pieces of information contained in the utterance. This is the most difficult step, since a properly designed fitness function should be capable of capturing a semantic parallelism between the logical formula and the natural language text, with an accuracy well beyond the one that can be provided, for instance, by commonly used techniques such as Latent Dirichlet Allocation (LDA) [49].

A possible solution can be that of extracting a suitable model from the text, capable of capturing all pieces of information that are relevant for the evaluation of the LTL formula. In order to do that, an initial step, as already discussed in the case of population initialization, can be the extraction of all proposition letters from the utterance. Then, these letters can be arranged in a Kripke structure [200], that is, a model that keeps track of how and when they hold. To generate such a model, the best approach seems to be the one pursued in [319], where a deep learning solution is developed that is capable of assigning time intervals to the verbs in a given text[5] (a similar work is discussed in [209]). Once such time intervals have been determined, the structure can be derived in a fairly straightforward way and, based on it, the (still non trivial) question now becomes "how much and how well the given LTL formula is describing what is happening in the Kripke model?". Finally, a second objective must be considered, that is, generating easy to read, and thus understandable, formulas. Such a condition can be enforced by using fairly natural metrics such as the nesting degree of the formula, the number of Boolean and/or temporal operators employed, and so on.

LTL synthesis through event identification and ordering

Finally, the English-to-LTL translation problem can be addressed with an approach which turns out to be somehow more hardwired than the ones discussed before, but, nevertheless, makes use of some advanced machine learning methodologies.

The idea is that of first determining the syntactic structure of the utterance and the events it refers to, then identifying the main agent and action of each of them, and finally establishing an ordering over them. Based on such pieces of information, it should be easier to derive a set of rules for the generation of the corresponding LTL formula, at least up to a fixed, maximum degree of nesting. Of course, the translated utterances could be fed in as training data to any other solution developed

[5] A working implementation can be found at: `https://hub.docker.com/r/sidvash/temporal`.

according to the approaches described in Section 7.2.3. In the following, we give a more detailed account of these phases.

As for the syntactic analysis, several frameworks can be found in the literature, most notably SpaCy [19], AllenNLP [134], which is based on SpaCy, Stanford CoreNLP [226], NLTK [222], and TextPro [266]. All of them allow one to perform the following syntactic analysis tasks, that are necessary to carry out the translation:

- *part of speech tagging*, by which components such nouns, verbs, and adjectives are identified;
- *construction of the dependency tree*, that establishes the relationships between "head" words and words which modify such heads;
- *coreference resolution*, that is, the identification of all the expressions that refer to the same entity in the text;
- *recognition of noun/prepositional/verb phrases*, that may help in the later identification of proposition letters (for instance, *"the big blue ocean"* should be considered as a single entity).

Then, proposition letters are to be extracted, relying on tools like, e.g., PredPatt [12, 339], or on custom solutions, based on the previously-introduced frameworks, making use of either:

- *semantic role labeling*, that is, the process that assigns labels to words or phrases in a sentence that indicate their semantic role, such as, for instance, agent or action, or
- *open information extraction*, that generates a structured, machine-readable representation of the relevant data in the text, typically in the form of n-ary propositions.

The next step is that of correlating proposition letters by making use of Boolean connectives and temporal modalities. To this end, other than relying on the dependency tree and on specific keyword extraction, it is possible to consider:

- *temporal expression recognition and normalization*, that is, the task of identifying sequences of tokens in a given text that denote a point in time, a duration, or a frequency, and of interpreting them. A lot of work has been done in this respect (see, e.g., [97, 205, 219, 341]), including a number tools that can be used for the task: PTime [13], SUTIME [76], UWTIME [207], ClearTK's TimeML module [5], HeidelTime [305], cogCompTime [256], SynTime [342], and TextPro's TimePro module [266]. In addition, AllenNLP and Stanford coreNLP contain some modules that can be used as well. The typical performance is higher than 80% precision and recall on benchmark data such as the Platinum dataset from the TempEval3 workshop [315].

- *temporal relation identification*, i.e., the task of determining a partial or total ordering of events (e.g., identified by verbs) happening in an utterance. While this can still be considered an open problem [93], a lot of work has already been done in the literature [48, 130, 218, 235, 253, 254, 255], also driven by important domain requirements, such as those in the medical field [118, 206, 312]. Some tools are available that either (*i*) encode temporal relationships by means of graphs, as it happens with CATENA [236], ClearTK's TimeML module [5], TextPro's TempRelPro module [266], cogCompTime [256], CAEVO [4], and TIPsem [21, 220], or, perhaps more naturally, (*ii*) assign a time interval to each event, like in [319] (we have already referred to it in Section 7.2.3) and [209].

Other resources that can be successfully exploited are WordNet [114], a well-known lexical database for the English language that groups words into sets of synonyms and records a number of relations among them, VerbNet [293], which is the largest on-line verb lexicon currently available for English, PPDB [131], a database containing hundreds of million paraphrase pairs, which capture many meaning-preserving syntactic transformations, and VerbOcean, a repository that encompasses the semantic relationships that in English language typically hold between verbs, including a *happens-before* relation [80].

7.2.4 An experimental evaluation of existing tools

This section is devoted to a critical experimental evaluation of some of the tools that have been previously discussed. For such a purpose, two datasets have been considered.

The first one is the *US Government's Accident Injuries* collection[6], that contains information on all accidents, injuries, and illnesses reported by mine operators and contractors beginning with 1/1/2000. Each instance corresponds to an accident and is characterized, among all other pieces of information, by a free-text narrative describing the event.

The second dataset has been provided by *Main Roads Western Australia*[7], and it includes information on fatal, serious, and medical car crashes recorded between 2013 and 2017 in Western Australia. As it happens with the previous collection, each instance is characterized by a narrative describing how the incident occurred.

The two datasets are representative of two different kinds of narratives: in the mining case, the textual description is typically short, and involves a single participant. On the contrary, the car crash narratives are longer and more articulated, may involve several participants, and tend to thoroughly describe the event.

[6] `https://catalog.data.gov/dataset/accident-injuries`.

[7] `https://www.mainroads.wa.gov.au/Pages/default.aspx`.

For reference, the following representative utterance from the mining accidents dataset is taken into account:

> *Employee was cleaning up at the Primary Crusher with the Dingo skid steer. The employee slipped and fell while operating the skid steer and the machine pinned him against the cement retaining wall.*

As for the car crashes dataset, the following representative instance is considered:

> *B drove east on Gibbins Road, Coolup. It appears B has stopped at the intersection of Maryfield Road and allowed a vehicle to pass before proceeding into the intersection. B2 was driving north on Maryfield Road, has observed B's vehicle at the intersection noticing that B appeared to be looking straight ahead. As B has proceeded into the intersection, B2 has applied the brakes on his vehicle, however he was unable to avoid B's vehicle and has 't-boned' it, colliding with the drivers side of B's vehicle. B has died at the scene from injuries sustained. A preliminary test conducted on B2 returned a negative reading.*

The remainder of this section considers two main tasks that, according to Section 7.2.3, have to be carried out in order to synthesize an LTL formula that correctly formalizes an utterance, that is, the extraction of the proposition letter candidates, and the identification of the temporal relationships among them.

Concerning the extraction of proposition letters, we may identify two distinct steps: first, *coreference resolution*, by which all references to the same entity are identified and normalized[8]; second, the *extraction of predicates*, i.e., tuples composed of an agent, the action that is being carried out, and possible other arguments.

As for coreference resolution, AllenNLP [134] relies on the state-of-the-art algorithm presented in [208]. Tested on the mining accidents dataset instance, it identifies two entities with coreferences, highlighted in red and blue colours:

> ***Employee*** *was cleaning up at the Primary Crusher with* ***the Dingo skid steer***. ***The employee*** *slipped and fell while operating* ***the skid steer*** *and the machine pinned* ***him*** *against the cement retaining wall.*

Although the result is fairly accurate, the algorithm fails to detect the *the machine* reference to the blue entity.

[8] Observe that coreference resolution implies entity identification which, nevertheless, is a much easier task, that in general can be reliably solved by investigating the dependency tree and the PoS tags of the given utterance.

Table 7.1: Predicates extracted by PredPatt on the mining accidents dataset instance. Results have been reworked for the ease of reading.

Agent	Action
Employee	was cleaning up
Employee	slipped
Employee	fell
the Dingo skid	steer
the Dingo skid	pinned Employee

Turning to a more challenging example, this is the result of its execution on the second text[9]:

> ***B*** *drove east on Gibbins Road,* ***Coolup****. It appears* ***B*** *has stopped at the intersection of Maryfield Road and allowed a vehicle to pass before proceeding into the intersection.* ***B2*** *was driving north on Maryfield Road, has observed* ***B's*** *vehicle at the intersection noticing that* ***B*** *appeared to be looking straight ahead. As* ***B*** *has proceeded into the intersection,* ***B2*** *has applied the brakes on* ***his*** *vehicle, however* ***he*** *was unable to avoid* ***B's*** *vehicle and has 't-boned' it, colliding with the drivers side of* ***B's*** *vehicle.* ***B*** *has died at the scene from injuries sustained. A preliminary test conducted on* ***B2*** *returned a negative reading.*

Focusing on just two entities, it is immediately clear that the algorithm has erroneously associated *Coolup* with *B2*. Even worse, judging from the result it seems that *B2* has been applying the brakes on *B*'s vehicle, and that *B* was unable to avoid himself. These two errors are enough to totally misinterpret the semantic content of the utterance.

Once the entities have been found and normalized, it is possible to look for the candidate proposition letter. In this respect, we investigated the use of PredPatt [12, 339], since it is still a major component in many recent, state-of-the-art NLP tools (see, for instance, [319]). To do that, we relied on two versions of the chosen instances in which coreference resolution has been already correctly solved by hand.

The predicates extracted by PredPatt on the mining data utterance are reported in Table 7.1. As it can be seen, the algorithm outputs 5 candidates. Again, with the exception of the fourth candidate, the result is quite natural and intuitive, except perhaps for the fact that the fourth and fifth candidates refer to the *the Dingo skid* entity instead of *the Dingo skid steer*, considering *steer* as a verb.

Table 7.2 reports the predicates extracted by PredPatt algorithm on the car crash dataset instance. Here, there are some spurious candidates as, for example,

[9]Note that the coreference resolution process extracted more than two entities. However, for the sake of clarity, we concentrate here on the two main ones, that are, *B* and *B2*.

Table 7.2: Predicates extracted by PredPatt on the car crashes dataset instance. Results have been reworked for the ease of reading.

Agent	Action
B	drove east
It	appears
A vehicle	allowed to pass
[UNK]	was driving north
B	appeared to be looking straight ahead
B	has proceeded into the intersection
B2	has applied the brakes on B2's vehicle
[UNK]	avoid B's vehicle
[UNK]	't-boned' it
[UNK]	has died at the scene from injuries
A preliminary test	returned negative reading

the one with *It* as the main agent. Moreover, for several results, the agent is not clear (*UNK*), or simply wrong (*A vehicle allowed to pass*). Finally, some predicates are entirely missed, like in the case of *B has stopped at the intersection*.

Finally, let us consider the task of extracting the temporal relationships that hold between each pair of predicates. Our original intention was that of evaluating cogCompTime [256] as the tool for generating an ordering graph over the set of events, and the approach presented in [319] as the tool for assigning time intervals to events, since they are two recently proposed solutions. Unfortunately, since various weeks cogCompTime is experiencing some server problems (`http://groupspaceuiuc.com/temporal/#`), that do not allow us to experiment with it. Thus, in the following, just the time interval approach presented in [319] is evaluated with respect to the two selected utterances.

Figure 7.6 shows its results when applied on the mining dataset instance. As it can be seen, the two occurrences of *steer* are still erroneously recognized as actions. Moreover, while the interval assigned to *operating* seems to be an appropriate choice, the tool fails in identifying its overlap with the interval related to *cleaning*. Furthermore, the events *fell* and *slipped* appear to be swapped.

Considering the car crashes instance, as depicted in Figure 7.7, the outcomes are unfortunately quite poor, and do not seem to follow any reasonable pattern.

At this point, by looking at the underwhelming results obtained by the tools we tested, we decided not to proceed with the LTL formula synthesis task any further.

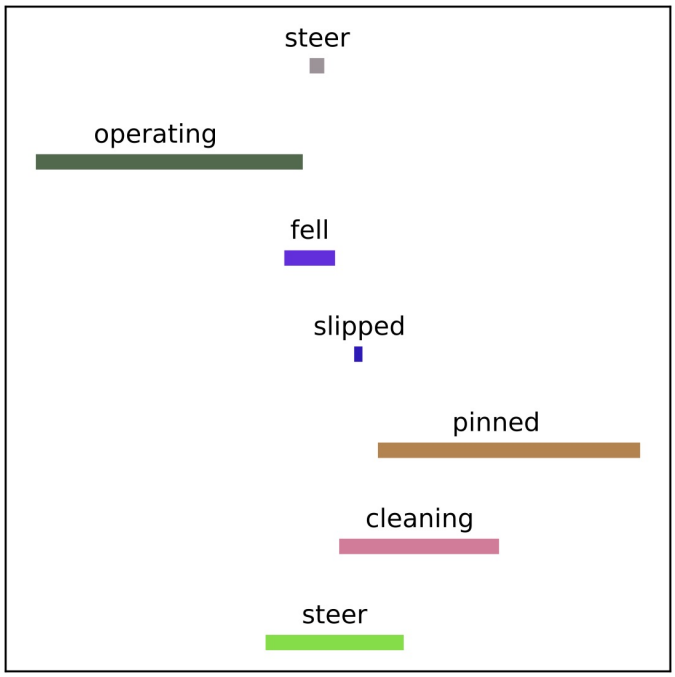


Figure 7.6: Events timeline in the mining dataset instance.

7.2.5 Discussion

Linear-time temporal logic (LTL) is a logical formalism which is commonly adopted in formal verification, in particular in consistency and model checking, and it can be a very useful tool for capturing the temporal content of natural language utterances. Such capabilities can be exploited in many important applications, including disambiguation of technical requirements, log analysis, and automated reasoning over temporal data.

Unfortunately, as it emerges from our comprehensive analysis of the literature and of existing tools, a general enough solution, that is capable of translating free, natural English texts into unbounded, general LTL formulas is still missing.

In this work, we provided a comprehensive review of the state of the art, as defined by the relevant literature; moreover, an empirical evaluation has been conducted on some of the currently available NLP tools, based on two real world instances. Results show that further research work is needed on methods and tools for various crucial tasks, such as coreference resolution, predicate extraction, and temporal relation identification, in order to achieve a performance which is good enough to allow for the synthesis of LTL formulas from unconstrained, natural language texts. We highlighted some possible research directions that can be followed to tackle such a complex problem.

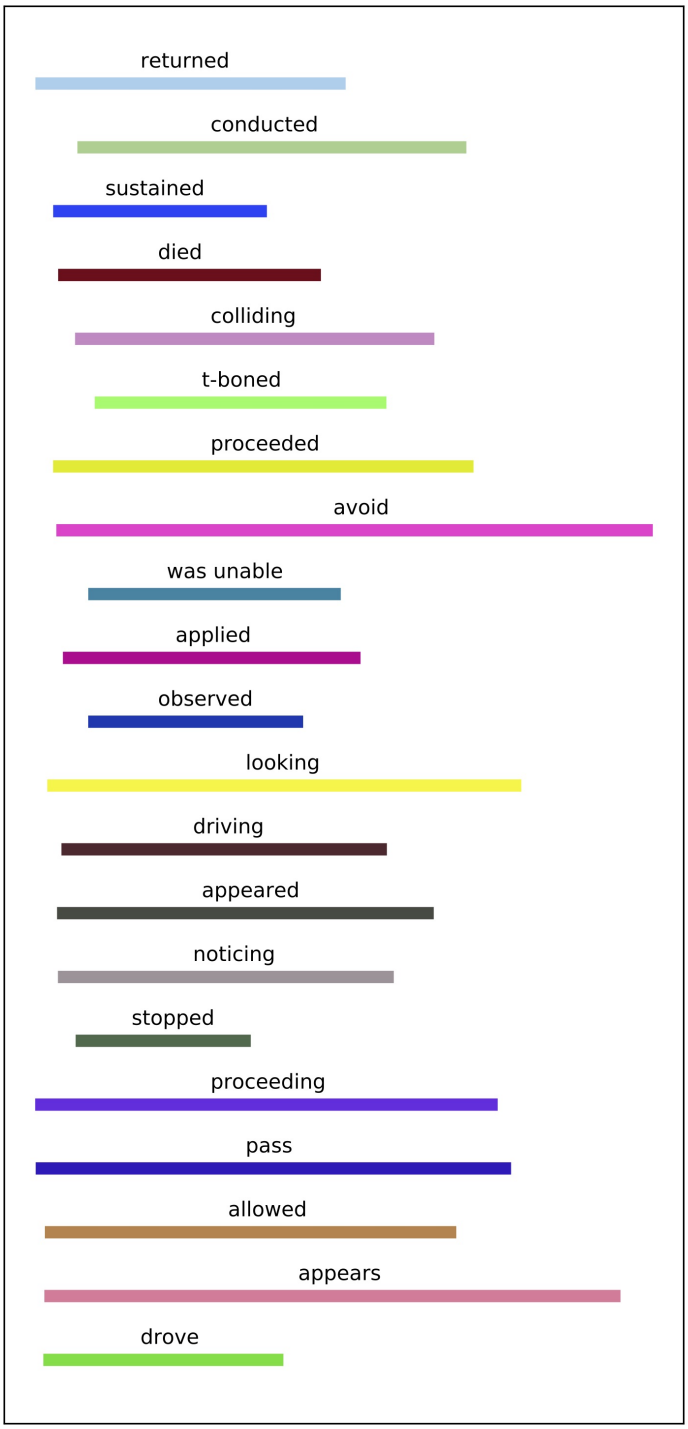


Figure 7.7: Events timeline in the car crashes dataset instance.

7.3 Pairing monitoring with machine learning for smart system verification and predictive maintenance

In Section 7.1 we have seen how temporal logic may be used within a machine learning algorithm to perform classification; conversely, in Section 7.2, we considered how machine learning methodologies may be relied on to extract temporal logic formulas from natural language texts. In this final section, we present the architecture of a system in which machine learning and formal methods are seamlessly combined and employed together to perform tasks such as anomaly detection and maintenance prediction.

Over the last decades, thanks to new microelectronic technologies, even the simplest measuring devices have progressively evolved into complex digital systems capable of performing several processing and communication operations related to the measured values (smart sensors). A smart sensor can be defined as a device that takes its input from the physical environment and uses built-in computational resources to perform some predefined functions upon detection of specific input, thus processing data before passing them on [127]. Such devices may be used to set up monitoring and control mechanisms in a variety of contexts, including smart grids [174], and they have several applications in agriculture [320], health-care [151], and industry [251]. As an example, one may think of an industrial plant, where the behaviour of all the different subsystems and components is tracked by means of dedicated smart sensors embedded in the machinery.

In such a complex environment, formal methods can be exploited to devise methods and techniques for the automated verification of software and hardware systems, a task which is of paramount importance in many critical domains. However, the inherent complexity of the systems and the non-trivial ways in which their components may interact among themselves and with the environment, make it very difficult (and sometimes impossible) to specify in advance all the relevant properties that have to be guaranteed (or, dually, avoided). To overcome these limitations, some approaches that complement formal verification with model-based testing and monitoring have been recently proposed in the literature (see, for instance, [75, 136]).

In this work, we focus our attention on *monitoring*, a runtime verification technique which is gaining more and more attention within the formal method and verification community. We outline a novel framework for online system verification that integrates monitoring with machine learning and can be applied in anomaly detection and predictive maintenance tasks. As we shall see, the proposed approach allows a domain expert to start with the specification of the most important and natural properties to monitor. Then, the framework autonomously discovers new relevant properties by means of an iterative refinement process, becoming capable, over time, of identifying undesired behaviours in advance, with a considerably higher

level of detail and coverage than the original specification.

In the following sections, after a brief introduction to the concept of system monitoring, we envision a framework that combines it with machine learning, also identifying some of the most promising approaches concerning the latter. We then conclude with a short recap of the distinctive features of the proposed approach.

7.3.1 System monitoring

Monitoring [211] is a runtime verification technique that is gaining much interest in the realm of formal methods for automated system verification.

Classic, static techniques, such as, for instance, model checking [82], perform an exhaustive analysis of the system: satisfaction or violation of a property (typically expressed by some temporal logic formula) by the system is established by exploring all possible behaviours. On the contrary, monitoring allows one to detect satisfaction or violation of a property by analyzing a single behaviour, called *execution trace* or *run*, of the system. It is therefore a lightweight technique, but the gain in efficiency is paid in terms of expressivity: monitorable properties are a subset of the class of specifications expressible in temporal formalisms commonly used for automated verification.

Even though monitoring has been mostly studied in the setting of linear time temporal logics [158], a notable effort has been devoted to the investigation of monitoring branching-time properties (see, for instance, [124]). A comparison of the two settings can be found in [23].

The basic principles on which monitors (and the theory of monitorability) are built are the ability of reaching a verdict by just observing a finite prefix of a single execution trace (a *finite trace*) and the fact that, once a verdict is reached, it is irrevocable, as it is a guarantee that the system satisfies or violates the property, independently from all the other possible (unobserved) behaviours it might exhibit. This latter feature ensures the *soundness* of a monitor. We introduce and discuss the notion of *completeness* of a monitor below.

We say that a property is *positively monitorable* if every system satisfying it features a finite trace that witnesses the satisfaction; conversely, we say that a property is *negatively monitorable* if every system violating it features a finite trace that witnesses the violation. A *monitorable* property is a property that is either positively or negatively monitorable.

Safety properties are negatively monitorable, as their violation is witnessed by a finite trace. Dually, co-safety properties are positively monitorable. On the other hand, there exist properties like

it is possible to reach a `success` *state, but it is not possible to reach it in less than 3 steps*

which are neither positively nor negatively monitorable. A monitor observing a system execution trace that is compatible with the property, e.g., a `success` state

is reached at the third execution step, indeed, cannot issue a satisfaction verdict, thus stating that the property is fulfilled by every run, as there might be a run along which a `success` state is reached in less than 3 steps. Moreover, a system might violate the property by never reaching a `success` state at all; in this case, no finite trace witnesses the violation, as the `success` state might be reached at the next step, or through a completely different run.

Instead, the property

it is not possible to reach a `success` *state in less than 3 steps*

is negatively monitorable, as every system violating it features a finite trace that reaches a `success` state in less than 3 steps.

Given a positively (respectively, negatively) monitorable property P, we say that a monitor is *satisfaction-complete* (respectively, *violation-complete*) for P if every system satisfying P features a finite trace that is accepted (respectively, rejected) by the monitor.

It is clear that sound and complete monitors only exist for monitorable properties. Thus, characterizing the set of monitorable properties is one of the major challenges in the field. However, some work has also been done in the attempt of weakening the notion of monitorability (see [24] for an account of such a research direction) or of extending the realm of applicability of such a technique to specifications that are deemed not amenable to being runtime verified [124].

7.3.2 Integrating system monitoring and machine learning

As it is emerging from the most recent literature in the field, machine and statistical learning solutions can be successfully combined with formal methods techniques to deal with complex real-world problems [172]. One challenging scenario is that of failure detection and predictive maintenance. As we have already pointed out, formal methods allow one to synthesize monitors to check whether some good properties (safety properties) are violated by a system during its operation and to detect those system's behaviours (traces) that violate them. Then, machine learning approaches may be applied to analyze such traces.

For instance, frequent patterns (Section 1.1.1) or shapelets (Section 1.1.2) may be extracted. As for a more formal characterization of a pattern, the task of learning temporal logic formulas from examples is attracting an increasing interest. For instance, this can be the case of the decision tree learning algorithm *Temporal ID3*, which we discussed in Section 7.1. Also, a technique to learn general, unrestricted linear temporal logic (LTL) formulas from a set of examples in the form of infinite, ultimately periodic words, without relying on user-defined templates, is illustrated in [247]. Variants and extensions of the latter result are considered in [74, 334]. More precisely, the case of LTL interpreted over finite traces is dealt with in [74], while parametric linear temporal logic (pLTL) is analyzed in [334]. Finally, observe that the task of extracting a temporal logic formula from a set of traces can be

thought of as an optimization problem (similarly to what we discussed in Section 7.2.3), thus also evolutionary algorithms may be relied on for such a purpose.

In the following, we briefly outline a possible integration of monitoring and machine learning, that can be decomposed into five main phases.

Specification of the initial set of properties. Domain experts are asked to specify the most significant properties that the system under consideration should exhibit. These properties are then formalized in a suitable temporal logic and a monitor that checks them against execution traces is synthesized (obviously, monitorability of the relevant formulas is a critical issue here).

Monitoring of system properties. The monitor checks whether the system satisfies/violates the considered properties during its execution (monitoring).

Detection of a failure. Traces for which the monitor reaches a verdict of failure are collected. These are considered to be *failure* traces. In addition, traces generated by the system during previous normal behaviour are extracted, and considered to be *normal* traces. Of course, the length of the time window that is used to extract failure traces depends on the specific application domain, and it must be carefully chosen according to the results of a dedicated tuning phase, possibly with the help of domain experts.

Mining of the relevant behaviour patterns. Failure and normal traces are put together to generate a dataset for supervised machine learning. Each instance is characterized by a trace that can be numerical (as in the case of a temperature signal) or categorical (this is the case, for instance, with a sequence of system calls made in a Unix system). Moreover, each instance is labeled with a binary class, that can be either *failure* or *normal* behaviour. Machine learning algorithms are run on the dataset, with the goal of extracting the (temporal) logic formulas that best characterize and discriminate between the two classes.

Extension of the pool of properties. The (temporal) logic formulas extracted during the mining phase are added to the monitoring pool of properties (possibly after their transformation into safety properties), and the process restarts from the monitoring phase.

As it can be observed, the proposed framework works in an iterative way. It starts from a set of basic properties, and new ones are then added over time. The discovered logical properties are closely related to the original ones and, in principle, they should allow the system to discover anomalous behaviours earlier and with ever increasing accuracy and coverage.

7.3.3 Discussion

Thanks to the new microelectronic technology advancements, measuring devices can now be easily transformed into complex digital systems capable of processing data, and of interacting with one another and with the surrounding environment. As a result, their emerging behaviour is typically non trivial, and difficult to formalize into a set of logical properties to be used for standard system verification, even for a pool of domain experts. For such a reason, approaches that complement formal verification with model-based testing and monitoring have been recently investigated in the literature.

In this work, we outlined a possible solution to such a problem which integrates machine learning and monitoring techniques. The proposed formal verification framework starts with the runtime verification of the most important and natural properties that should be guaranteed during the execution of the analyzed system. Then, it autonomously discovers new ones in an iterative way, becoming capable of identifying unwanted behaviours earlier, and with ever higher detail and coverage than the original specification. In the long run, the verification tool is expected to gain the ability of performing advanced tasks such as self-diagnosis, predictive maintenance, adaptation, and robustness enhancement.

A practical implementation of the system introduced here in the context of a real-world scenario represents a natural development of the present work.

Conclusions

In this thesis, we presented several data science applications, with a particular attention to real case studies.

Since all starts from the data, we first presented the development of a data warehouse-based decision support system for an Italian contact center company. Then, we described several, classical analysis tasks that rely on the warehouse, including employee performance evaluation, phone call outcome prediction and inbound call notes classification. A characteristic common to all the solutions we developed is their interpretability, a concept which we specifically focused on investigating the possibility of performing decision tree pruning through evolutionary computation.

Then, we unravelled the main topic of this work, that is, how to handle temporal information in data mining tasks. We presented J48S, a novel decision tree induction algorithm that is capable of handling categorical and numerical attributes, as well as sequences, during the same execution cycle. The model has been applied within a more general speech analytics process inside the previously mentioned contact center company, following the observation that text can also be considered as a sequence of words.

Always with an eye on the extension of the temporal capabilities of the decision tree, we then discussed J48SS, that adds to J48S the possibility of dealing with time series data. Such an algorithm has proven its value on an environmental study set in the city of Wrocław, Poland, where it outperformed more classical solutions based on the use of lag variables. The pollution classification task has also allowed us to highlight an important point regarding J48SS: the decision tree is mostly useful in those situations in which classification has to be performed on heterogeneous datasets, where temporal and atemporal data are equivalently present and important. On the other hand, when dealing with temporal data only, such as time series, dedicated algorithms may be preferred, as the results of the application of J48SS on the UCR time series datasets have shown.

Next, since a lot of work on the management of temporal information has been done in automated reasoning and formal verification, a natural direction in which to proceed with the research was that of investigating how such solutions can be combined with machine learning algorithms. We explored two different paths: first, we showed how temporal logic can be used within a machine algorithm, presenting Temporal ID3, the prototype of an interval logic-based decision tree model; then, conversely, we investigated how machine learning techniques can be employed to generate temporal logic formulas, considering the task of natural language English utterance formalization. Finally, as a conclusive development, we proposed the

architecture of a system, in which formal methods and machine learning techniques are seamlessly combined to perform tasks such as anomaly detection and predictive maintenance.

This latter topic represents, at the same time, the culmination of the entire research path, and the starting point for a new journey. We began with some basic data collection and modelling tasks, and this led us to the first analytics applications. Then, we naturally considered temporal information under different aspects, and in ever increasing complexity. The final outcome of our work is an integrated framework that allows us to deal with both temporal as well as different kinds of atemporal information. We believe, as it is emerging from the most recent literature in the field, a new, thrilling research direction to be that of combining machine and statistical learning solutions with logics and formal methods techniques, to deal with complex real-world problems, that can be characterized by several kinds of sequential and temporal information. This would allow, in principle, to combine the best of both worlds, i.e., inference from the data, and a high interpretability of the generated models and solutions. But this is a story for another day.

Bibliography

[1] ACE project website. https://www.ldc.upenn.edu/collaborations/past-projects/ace. Accessed: 2019-04-10.

[2] AQUAINT corpus website. https://catalog.ldc.upenn.edu/LDC2002T31. Accessed: 2019-04-12.

[3] C5.0: An Informal Tutorial. https://www.rulequest.com/see5-unix.html. Accessed: 2017-06-24.

[4] CAEVO corpus website. https://www.usna.edu/Users/cs/nchamber/caevo/. Accessed: 2019-04-10.

[5] ClearTK GitHub page. https://cleartk.github.io/cleartk/. Accessed: 2019-04-16.

[6] European Union air quality standards. http://ec.europa.eu/environment/air/quality/standards.htm. Accessed: 2019-05-21.

[7] Is C5.0 Better Than C4.5? http://rulequest.com/see5-comparison.html. Accessed: 2017-06-24.

[8] Kansas state university CIS department, laboratory for specification, analysis, and transformation of software (SAnToS laboratory), property pattern mappings for LTL. http://patterns.projects.cs.ksu.edu/. Accessed: 2019-04-03.

[9] Keras framework website. https://keras.io. Accessed: 2019-04-12.

[10] The key word in "data science" is not data, it is science. https://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/. Accessed: 2019-06-21.

[11] NOx level objectives. http://www.icopal-noxite.co.uk/nox-problem/nox-level-objectives.aspx. Accessed: 2019-05-21.

[12] PredPatt GitHub page. https://github.com/hltcoe/PredPatt. Accessed: 2019-04-18.

[13] PTime website. http://ws.nju.edu.cn/ptime/. Accessed: 2019-04-18.

[14] PyTorch GitHub page. https://github.com/pytorch. Accessed: 2019-04-12.

[15] Scikit-learn's compute_class_weight function. `https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html`. Accessed: 2019-05-22.

[16] Scikit-learn's RandomForestClassifier. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. Accessed: 2019-05-22.

[17] Semantic parsing with AllenNLP. `https://github.com/allenai/allennlp/blob/master/tutorials/getting_started/semantic_parsing.md`. Accessed: 2019-04-17.

[18] Sippycup GitHub page. `https://github.com/wcmac/sippycup`. Accessed: 2019-04-17.

[19] SpaCy website. `https://spacy.io/`. Accessed: 2019-04-16.

[20] TensorFlow framework website. `https://www.tensorflow.org/`. Accessed: 2019-04-15.

[21] TIPSem GitHub page. `https://github.com/hllorens/otip`. Accessed: 2019-04-08.

[22] WASP semantic parser website. `http://www.cs.utexas.edu/users/ml/wasp/`. Accessed: 2019-04-07.

[23] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proceedings of the ACM on Programming Languages*, 3:52:1–29, 2019.

[24] L. Aceto, A. Achilleos, A. Francalanza, A. Ingólfsdóttir, and K. Lehtinen. An operational guide to monitorability. In *Proceedings of the 17th International Conference on Software Engineering and Formal Methods (SEFM 2019)*, volume 11724, pages 433–453, 2019.

[25] A. S. Adesuyi and Z. Munch. Using time-series NDVI to model land cover change: a case study in the Berg river catchment area, Western Cape, South Africa. *International Journal of Environmental, Chemical, Ecological, Geological and Geophysical Engineering*, 9(5):537–542, 2015.

[26] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th IEEE International Conference on Data Engineering (ICDE 1995)*, pages 3–14, 1995.

[27] R. Alcaraz, M. Gacto, and F. Herrera. A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems. *IEEE Transactions on Fuzzy Systems*, 19(4):666–681, 2011.

[28] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[29] R. Alluhaibi. Simple interval temporal logic for natural language assertion descriptions. In *Proceedings of the 11th International Conference on Computational Semantics (IWCS 2015)*, pages 283–293, 2015.

[30] J. B. Alonso, J. Cabrera, M. Medina, and C. M. Travieso. New approach in quantification of emotional intensity from the speech signal: Emotional temperature. *Expert Systems with Applications*, 42(24):9554–9564, 2015.

[31] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[32] R. Anirudha, R. Kannan, and N. Patil. Genetic algorithm based wrapper feature selection on hybrid prediction model for analysis of high dimensional data. In *Proceedings of the 9th International Conference on Industrial and Information Systems (ICIIS 2014)*, pages 1–6, 2014.

[33] S. Antipov and M. Fomina. A method for compiling general concepts with the use of temporal decision trees. *Scientific and Technical Information Processing*, 38(6):409–419, 2011.

[34] M. Antonelli, D. Bernardo, H. Hagras, and F. Marcelloni. Multiobjective evolutionary optimization of type-2 fuzzy rule-based systems for financial data classification. *IEEE Transactions on Fuzzy Systems*, 25(2):249–264, 2017.

[35] M. Arathi and A. Govardhan. Effect of Mahalanobis distance on time series classification using shapelets. In *Proceedings of the 49th Annual Convention of the Computer Society of India (CSI 2015)*, volume 2, pages 525–535. Springer, 2015.

[36] M. Artetxe, G. Labaka, and E. Agirre. Unsupervised statistical machine translation. *CoRR*, abs/1809.01272, 2018.

[37] M. Artetxe, G. Labaka, and E. Agirre. An effective approach to unsupervised machine translation. *CoRR*, abs/1902.01313, 2019.

[38] Y. Artzi. Cornell SPF: Cornell semantic parsing framework. *arXiv preprint arXiv:1311.3011*, 2013.

[39] H. Atassi and Z. Smékal. Automatic identification of successful phone calls in call centers based on dialogue analysis. In *Proceedings of the 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom 2014)*, pages 425–429, 2014.

[40] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 429–435, 2002.

[41] R. Baeza-Yates. Challenges in the interaction of information retrieval and natural language processing. In *Proceedigs of the 5th International on Computational Linguistics and Intelligent Text Processing (CICLing 2004)*, pages 445–456, 2004.

[42] R. C. Barros and A. A. Freitas. A survey of evolutionary algorithms for decision-tree induction. In *Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, pages 291–312, 2011.

[43] E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proceedings of the 12th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2014)*, pages 23–37, 2014.

[44] N. Barua, M. Choudhury, and A. Borkakoty. *Business Process Outsourcing: Its Prospects and Challenges*. Daya Publishing House, 2009.

[45] F. Béchet, B. Maza, N. Bigouroux, T. Bazillon, M. El-Bèze, R. D. Mori, and E. Arbillot. DECODA: A call-centre human-human spoken conversation corpus. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 1343–1347, 2012.

[46] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*, pages 1533–1544, 2013.

[47] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.

[48] S. Bethard, T. A. Miller, D. Dligach, C. Lin, and G. Savova. Neural temporal relation extraction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*, pages 746–751, 2017.

[49] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[50] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[51] V. Bolón-Canedo and A. Alonso-Betanzos. *Recent Advances in Ensembles for Feature Selection*, volume 147 of *Intelligent Systems Reference Library*. Springer, 2018.

[52] G. Bombara, C. Vasile, F. Penedo, H. Yasuoka, and C. Belta. A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control (HSSC 2016)*, pages 1–10, 2016.

[53] A. Bonifati, G. Mecca, P. Papotti, and Y. Velegrakis. Discovery and correctness of schema mapping transformations. In *Schema Matching and Mapping*, pages 111–147. Springer, 2011.

[54] H. Boström. Concurrent learning of large-scale random forests. In *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence (SCAI 2011)*, volume 227 of *Frontiers in Artificial Intelligence and Applications*, pages 20–29, 2011.

[55] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[56] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[57] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Metric propositional neighborhood logics on natural numbers. *Software and System Modeling*, 12(2):245–264, 2013.

[58] D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computer Science*, 560:269–291, 2014.

[59] D. Bresolin, P. Sala, and G. Sciavicco. On begins, meets, and before. *International Journal of Foundations of Computer Science*, 23(3):559–583, 2012.

[60] A. Brunello, D. Della Monica, and A. Montanari. Pairing monitoring with machine learning for smart system verification and predictive maintenance. In *Proceedings of the 1st Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis (OVERLAY@AIxIA 2019)*, 2019.

[61] A. Brunello, P. Gallo, E. Marzano, A. Montanari, and N. Vitacolonna. An event-based data warehouse to support decisions in multi-channel, multi-service contact centers. *Journal of Cases on Information Technology*, 21(1), 2019.

[62] A. Brunello, F. Jiménez, E. Marzano, J. Palma, G. Sánchez, and G. Sciavicco. Towards semi-automatic human performance evaluation: The case study of a contact center. *Intelligent Data Analysis*, 22(4):867–880, 2018.

[63] A. Brunello, F. Jiménez, E. Marzano, A. Montanari, G. Sánchez, and G. Sciavicco. Multiobjective evolutionary feature selection and fuzzy classification of contact centre data. *Expert Systems*, 36(3), 2019.

[64] A. Brunello, J. Kamińska, E. Marzano, A. Montanari, G. Sciavicco, and T. Turek. Assessing the role of temporal information in modelling short-term air pollution effects based on traffic and meteorological conditions: A case study in Wrocław. In *Proceedings of the International Workshop on Business Intelligence and Big Data Applications at the 23rd European Conference on Advances in Databases and Information Systems (BBIGAP@ADBIS2019)*, 2019.

[65] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. Decision tree pruning via multi-objective evolutionary computation. *International Journal of Machine Learning and Computing*, 7(6):167–175, 2017.

[66] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. J48S: A sequence classification approach to text analysis based on decision trees. In *Proceedings of the International Conference on Information and Software Technologies (ICIST 2018)*, pages 240–256. Springer, 2018.

[67] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. A novel decision tree approach for the handling of time series. In *Proceedings of the International Conference on Mining Intelligence and Knowledge Exploration (MIKE 2018)*, pages 351–368. Springer, 2018.

[68] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. A combined approach to the analysis of speech conversations in a contact center domain. *Submitted to Journal*, 2019.

[69] A. Brunello, E. Marzano, A. Montanari, and G. Sciavicco. J48SS: A novel decision tree approach for the handling of sequential and time series data. *Computers*, 8(1):21, 2019.

[70] A. Brunello, A. Montanari, and M. Reynolds. Synthesis of LTL formulas from natural language texts: State of the art and research directions. In *Proceedings of the 26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*, 2019.

[71] A. Brunello, G. Sciavicco, and I. E. Stan. Interval temporal logic decision tree learning. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence (JELIA 2019)*, pages 778–793, 2019.

[72] S. Bufo, E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, and L. Bortolussi. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Proceedings of the 6th International Symposium on Leveraging*

Applications of Formal Methods, Verification and Validation (ISoLA 2014), pages 391–403, 2014.

[73] F. Cailliau and A. Cavet. Mining automatic speech transcripts for the retrieval of problematic calls. In *Proceedings of the 13th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2013)*, pages 83–95, 2013.

[74] A. Camacho and S. A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019), to be published*, 2019.

[75] I. Cassar, A. Francalanza, L. Aceto, and A. Ingólfsdóttir. A survey of runtime monitoring instrumentation techniques. In *Proceedings of the 2nd International Workshop on Pre- and Post-Deployment Verification Techniques (PrePost@iFM 2017)*, pages 15–28, 2017.

[76] A. X. Chang and C. D. Manning. SUTime: A library for recognizing and normalizing time expressions. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 3735–3740, 2012.

[77] J. Chen, X. Wang, and J. Zhai. Pruning decision tree using genetic algorithms. In *Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI 2009)*, volume 3, pages 244–248, 2009.

[78] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The UCR time series classification archive. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`, July 2015. Accessed: 2019-06-19.

[79] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE 2007)*, pages 716–725, 2007.

[80] T. Chklovski and P. Pantel. VerbOcean: Mining the web for fine-grained semantic verb relations. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, pages 33–40, 2004.

[81] M. Chowdhuri and S. Bojewar. Emotion detection analysis through tone of user: A survey. *Emotion*, 5(5):859–861, 2016.

[82] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[83] D. Clow. The learning analytics cycle: Closing the loop effectively. In *Proceedings of the 2nd ACM International Conference on Learning Analytics and Knowledge (LAK12)*, pages 134–138, 2012.

[84] L. Console, C. Picardi, and D. Dupré. Temporal decision trees: Model-based diagnosis of dynamic systems on-board. *Journal of Artificial Intelligence Research*, 19:469–512, 2003.

[85] O. Cordón, F. Herrera, M. del Jesus, L. Magdalena, A. Sánchez, and P. Villar. A multiobjective genetic algorithm for feature selection and data base learning in fuzzy-rule based classification systems. In B. Bouchon-Meunier, L. Foulloy, and R. R. Yager, editors, *Intelligent Systems for Information Processing*, pages 315–326. Elsevier Science, 2003.

[86] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[87] P. Cosi, F. Tesser, R. Gretter, C. Avesani, and M. Macon. Festival speaks Italian! In *Proceedings of the 7th European Conference on Speech Communication and Technology (Eurospeech 2001)*, pages 509–512, 2001.

[88] V. K. Dabhi and S. Chaudhary. A survey on techniques of improving generalization ability of genetic programming solutions. *arXiv preprint arXiv:1211.1119*, 2012.

[89] A. Davletcharova, S. Sugathan, B. Abraham, and A. P. James. Detection and analysis of emotion from speech signals. *Procedia Computer Science*, 58:91–96, 2015.

[90] K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms*. Wiley, London, UK, 2001.

[91] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[92] D. Della Monica, D. de Frutos-Escrig, A. Montanari, A. Murano, and G. Sciavicco. Evaluation of temporal datasets via interval temporal logic model checking. In *Proceedings of the 24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*, pages 11:1–11:18, 2017.

[93] L. Derczynski. *Automatically Ordering Events and Times in Text*, volume 677 of *Studies in Computational Intelligence*. Springer, 2017.

[94] J. K. Deters, R. Zalakeviciute, M. Gonzalez, and Y. Rybarczyk. Modeling $PM_{2.5}$ urban pollution using machine learning and selected meteorological parameters. *Journal of Electrical and Computer Engineering*, 5106045:1–14, 2017.

[95] V. Dhar. Data science and prediction. *Communications of the ACM*, 56(12):64–73, 2012.

[96] S. Dinakaran and P. R. J. Thangaiah. Role of attribute selection in classification algorithms. *International Journal of Scientific & Engineering Research*, 4(6):67–71, 2013.

[97] W. Ding, G. Gao, L. Shi, and Y. Qu. A pattern-based approach to recognizing time expressions. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (AAAI 2019)*, 2019.

[98] P. Ducange, G. Mannara, F. Marcelloni, R. Pecori, and M. Vecchio. A novel approach for internet traffic classification based on multi-objective evolutionary fuzzy classifiers. In *Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017)*, pages 1–6, 2017.

[99] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, 2012.

[100] A. D'Ulizia, F. Ferri, and P. Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36(1):1–27, 2011.

[101] H. Duong, T. Truong, and B. Le. Efficient algorithms for simultaneously mining concise representations of sequential patterns based on extended pruning conditions. *Engineering Applications of Artificial Intelligence*, 67:197–210, 2018.

[102] J. J. Durillo, A. J. Nebro, and E. Alba. The jMetal framework for multi-objective optimization: Design and architecture. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 4138–4325, 2010.

[103] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE 1999)*, pages 411–420, 1999.

[104] J. Dzifcak, M. Scheutz, C. Baral, and P. W. Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 4163–4168, 2009.

[105] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[106] A. Ekbal, S. Saha, and C. Garbe. Feature selection using multiobjective optimization for named entity recognition. In *Proceedings of the 20th International Conference on Pattern Recognition (ICPR 2010)*, pages 1937–1940, 2010.

[107] M. ElAlami. A filter model for feature subset selection based on genetic algorithm. *Knowledge-Based Systems*, 22(5):356–362, 2009.

[108] C. Emmanouilidis, A. Hunter, J. MacIntyre, and C. Cox. A multi-objective genetic algorithm approach to feature selection in neural and fuzzy modeling. *Journal of Evolutionary Optimization*, 3(1):1–26, 2001.

[109] F. Esposito, D. Malerba, and G. Semeraro. Decision tree pruning as a search in the state space. In *Proceedings of the 6th European Conference on Machine Learning (ECML 1993)*, pages 165–184, 1993.

[110] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[111] Y. Estève, M. Bouallegue, C. Lailler, M. Morchid, R. Dufour, G. Linarès, D. Matrouf, and R. D. Mori. Integration of word and semantic features for theme identification in telephone conversations. In *Natural Language Dialog Systems and Intelligent Assistants*, pages 223–231. Springer, 2015.

[112] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*, pages 230–238, 2008.

[113] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, 4(3):243–263, 1994.

[114] C. Fellbaum. WordNet. *The Encyclopedia of Applied Linguistics*, 2012.

[115] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[116] A. Ferranti, F. Marcelloni, and A. Segatori. A multi-objective evolutionary fuzzy system for big data. In *Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016)*, pages 1562–1569, 2016.

[117] A. Ferranti, F. Marcelloni, A. Segatori, M. Antonelli, and P. Ducange. A distributed approach to multi-objective evolutionary generation of fuzzy rule-based classifiers from big data. *Information Sciences*, 415:319–340, 2017.

[118] O. Ferret, X. Tannier, A. Névéol, and J. Tourille. Temporal information extraction from clinical text. In *Proceedings of the 15th Conference of the European*

Chapter of the Association for Computational Linguistics (EACL 2017), pages 739–745, 2017.

[119] J. Fitzgerald, R. M. A. Azad, and C. Ryan. A bootstrapping approach to reduce over-fitting in genetic programming. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO 2013)*, pages 1113–1120, 2013.

[120] M. L. Forcada, M. Ginestí-Rosell, J. Nordfalk, J. O'Regan, S. Ortiz-Rojas, J. A. Pérez-Ortiz, F. Sánchez-Martínez, G. Ramírez-Sánchez, and F. M. Tyers. Apertium: A free/open-source platform for rule-based machine translation. *Machine Translation*, 25(2):127–144, 2011.

[121] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Proceedings of the 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2014)*, pages 40–52, 2014.

[122] P. Fournier-Viger, A. Gomariz, M. Šebek, and M. Hlosta. VGEN: Fast vertical mining of sequential generator patterns. In *Proceedings of the 16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2014)*, pages 476–488, 2014.

[123] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[124] A. Francalanza, L. Aceto, and A. Ingolfsdottir. Monitorability for the Hennessy–Milner logic with recursion. *Formal Methods in System Design*, pages 1–30, 2017.

[125] E. Frank, M. A. Hall, and I. H. Witten. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann Publishers Inc., 4th edition, 2016.

[126] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pages 144–151. Morgan Kaufmann Publishers Inc., 1998.

[127] R. Frank. *Understanding Smart Sensors*. Artech House, 2013.

[128] R. Gabbard, S. Kulick, and M. P. Marcus. Fully parsing the Penn Treebank. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (NAACL-HLT 2006)*, 2006.

[129] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic programming, validation sets, and parsimony pressure. In *Proceedings of the European Conference on Genetic Programming (EuroGP 2006)*, pages 109–120. Springer, 2006.

[130] D. Galvan, N. Okazaki, K. Matsuda, and K. Inui. Investigating the challenges of temporal relation extraction from clinical text. In *Proceedings of the 9th International Workshop on Health Text Mining and Information Analysis*, pages 55–64, 2018.

[131] J. Ganitkevitch, B. V. Durme, and C. Callison-Burch. PPDB: The paraphrase database. In *Proceedings of the Human Language Technologies Conference of the North American Chapter of the Association of Computational Linguistics (NAACL-HLT 2013)*, pages 758–764, 2013.

[132] N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing & Service Operations Management*, 5(2):79–141, 2003.

[133] J. García-Nieto, E. Alba, L. Jourdan, and E. Talbi. Sensitivity and specificity based multiobjective approach for feature selection: Application to cancer diagnosis. *Information Processing Letters*, 109(16):887–896, 2009.

[134] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. E. Peters, M. Schmitz, and L. Zettlemoyer. AllenNLP: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640, 2018.

[135] M. Garnier-Rizet, G. Adda, F. Cailliau, J.-L. Gauvain, S. Guillemin-Lanne, L. Lamel, S. Vanni, C. Waast-Richard, et al. CallSurf: Automatic transcription, indexing and structuration of call center conversational speech for knowledge extraction and query by content. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, pages 2623–2628, 2008.

[136] M. Gerhold, A. Hartmanns, and M. Stoelinga. Model-based testing of stochastically timed systems. *Innovations in Systems and Software Engineering*, 15(3-4):207–233, 2019.

[137] R. Gerth, D. A. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification (PSTV 1995)*, pages 3–18, 1995.

[138] S. Ghannay, A. Caubrière, Y. Estève, N. Camelin, E. Simonnet, A. Laurent, and E. Morin. End-to-end named entity and semantic concept extraction

from speech. In *Proceedings of the 2018 IEEE Workshop on Spoken Language Technology (SLT 2018)*, pages 692–699, 2018.

[139] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner. ARSENAL: Automatic requirements specification extraction from natural language. In *Proceedings of the 8th NASA Formal Methods International Symposium*, pages 41–46, 2016.

[140] S. Gold, N. Elhadad, X. Zhu, J. J. Cimino, and G. Hripcsak. Extracting structured medication event information from discharge summaries. In *Proceedings of the 2008 American Medical Informatics Association Annual Symposium (AMIA 2008)*, pages 237–241, 2008.

[141] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[142] M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*, volume 5. McGraw-Hill New York, 2009.

[143] A. Gomariz, M. Campos, R. Marin, and B. Goethals. ClaSP: An efficient algorithm for mining frequent closed sequences. In *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013)*, pages 50–61, 2013.

[144] A. Gómez-Skarmeta, F. Jiménez, and J. Ibáñez. Pareto-optimality in fuzzy modeling. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT 1998)*, pages 694–700, 1998.

[145] A. Gómez-Skarmeta, F. Jiménez, and G. Sánchez. Improving interpretability in approximative fuzzy models via multiobjective evolutionary algorithms. *International Journal of Intelligent Systems*, 22(9):943–969, 2007.

[146] I. Gonçalves and S. Silva. Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *Proceedings of the European Conference on Genetic Programming (EuroGP 2013)*, pages 73–84. Springer, 2013.

[147] Google Inc. Google Cloud Speech API. `https://cloud.google.com/speech/`, 2017. Accessed: 2017-12-18.

[148] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2014)*, pages 392–401, 2014.

[149] J. Grabocka, M. Wistuba, and L. Schmidt-Thieme. Scalable discovery of time-series shapelets. *arXiv preprint arXiv:1503.03238*, 2015.

[150] V. Gruhn and R. Laue. Patterns for timed property specifications. *Electronic Notes in Theoretical Computer Science*, 153(2):117–133, 2006.

[151] P. Gupta, D. Agrawal, J. Chhabra, and P. Dhir. IoT based smart healthcare kit. In *Proceedings of the International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT 2016)*, pages 237–242, 2016.

[152] L. O. Hall, R. Collins, K. W. Bowyer, and R. Banfield. Error-based pruning of decision trees grown on very large data sets can work! In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002)*, pages 233–238, 2002.

[153] M. A. Hall. *Correlation-Based Feature Selection for Machine Learning*. PhD thesis, The University of Waikato, 1999.

[154] J. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.

[155] C. B. Harris and I. G. Harris. Generating formal hardware verification properties from natural language documentation. In *Proceedings of the 9th IEEE International Conference on Semantic Computing (ICSC 2015)*, pages 49–56, 2015.

[156] C. B. Harris and I. G. Harris. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. In *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE 2016)*, pages 966–971, 2016.

[157] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2nd edition, 2009.

[158] K. Havelund and D. Peled. Runtime verification: From propositional to first-order temporal logic. In *Proceedings of the 18th International Conference on Runtime Verification (RV 2018)*, pages 90–112, 2018.

[159] D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 1993)*, pages 1002–1007, 1993.

[160] F. Herrera. Genetic fuzzy systems: Taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1:27–46, 2008.

[161] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[162] L. Hou, J. T. Kwok, and J. M. Zurada. Efficient learning of timeseries shapelets. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, 2016.

[163] J. Hühn and E. Hüllermeier. FURIA: An algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319, 2009.

[164] Z. Hui, B. H. Tu, and S. Kawasaki. Wrapper feature extraction for time series classification using singular value decomposition. In *Proceedings of the 1st World Congress of the International Federation for Systems Research: The New Roles of Systems Sciences For a Knowledge-based Society (IFSR 2005)*. JAIST Press, 2005.

[165] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters*, 5(1):15–17, 1976.

[166] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 2005.

[167] H. Ishibuchi. Multi-objective pattern and feature selection by a genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 1069–1076, 2000.

[168] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima. Performance comparison of NSGA-II and NSGA-III on various many-objective test problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2016)*, pages 3045–3052, 2016.

[169] H. Ishibuchi, T. Murata, and I. Turksen. Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89(2):135–150, 1997.

[170] H. Ishibuchi, T. Nakashima, and T. Murata. Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences*, 136(1-4):109–133, 2001.

[171] H. Ishibuchi and Y. Nojima. Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. *International Journal of Approximate Reasoning*, 44(1):4–31, 2007.

[172] N. Jansen, J.-P. Katoen, P. Kohli, and J. Kretinsky. Machine learning and model checking join forces. *Dagstuhl Reports*, 8(3):74–93, 2018.

[173] A. Jara, R. Martínez, D. Vigueras, G. Sánchez, and F. Jiménez. Attribute selection by multiobjective evolutionary computation applied to mortality from infection in severe burns patients. In *Proceedings of the International Conference on Health Informatics (HEALTHINF 2011)*, pages 467–471, 2011.

[174] M. Jaradat, M. Jarrah, A. Bousselham, Y. Jararweh, and M. Al-Ayyoub. The internet of energy: Smart sensor networks and big data management for smart grid. *Procedia Computer Science*, 56:592–597, 2015.

[175] C. Jard and T. Jéron. On-line model checking for finite linear temporal logic specifications. In *Proceedings of the International Conference on Computer Aided Verification (CAV 1989)*, pages 189–196, 1989.

[176] H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh. A framework for OLAP content personalization. In *Proceedings of the 2010 East European Conference on Advances in Databases and Information Systems (ADBIS 2010)*, pages 262–277. Springer, 2010.

[177] F. Jiménez, A. Gómez-Skarmeta, H. Roubos, and R. Babuska. Accurate, transparent, and compact fuzzy models for function approximation and dynamic modeling through multi-objective evolutionary optimization. In E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 653–667. Springer Berlin Heidelberg, Berlin, Germany, 2001.

[178] F. Jiménez, A. Gómez-Skarmeta, G. Sánchez, and K. Deb. An evolutionary algorithm for constrained multi-objective optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1133–1138, 2002.

[179] F. Jiménez, E. Marzano, G. Sánchez, G. Sciavicco, and N. Vitacolonna. Attribute selection via multi-objective evolutionary computation applied to multi-skill contact center data classification. In *Proceedings of the IEEE Symposium on Computational Intelligence in Big Data (IEEE CIBD 15)*, pages 488–495, 2015.

[180] F. Jiménez, G. Sánchez, J. García, G. Sciavicco, and L. Miralles. Multi-objective evolutionary feature selection for online sales forecasting. *Neurocomputing*, 2016.

[181] F. Jiménez, G. Sánchez, and J. M. Juárez. Multi-objective evolutionary algorithms for fuzzy classification in survival prediction. *Artificial Intelligence in Medicine*, 60(3):197–219, 2014.

[182] F. Jiménez, G. Sánchez, and P. Vasant. A multi-objective evolutionary approach for fuzzy optimization in production planning. *Journal of Intelligent and Fuzzy Systems*, 25(2):441–455, 2013.

[183] X. Jin and J. Han. Expectation maximization clustering. In *Encyclopedia of Machine Learning*, pages 382–383. Springer, 2011.

[184] Y. Jin, editor. *Multi-objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*. Springer, Warsaw, Poland, 2006.

[185] H. Jongen. *Optimization Theory*. Kluwer Academic Publishing, 2004.

[186] B. H. Jun, C. S. Kim, H.-Y. Song, and J. Kim. A new criterion in selection and discretization of attributes for the generation of decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1371–1375, 1997.

[187] J. A. Kamińska. The use of random forests in modelling short-term air pollution effects based on traffic and meteorological conditions: A case study in Wrocław. *Journal of Environmental Management*, 217:164–174, 2018.

[188] H. Kamp. A theory of truth and semantic representation. *Formal Semantics: The Essential Readings*, pages 189–222, 1981.

[189] A. Kampouraki, G. Manis, and C. Nikou. Heartbeat time series classification with support vector machines. *IEEE Transactions on Information Technology in Biomedicine*, 13(4):512–518, 2009.

[190] F. Karim, S. Majumdar, H. Darabi, and S. Chen. LSTM fully convolutional networks for time series classification. *arXiv preprint arXiv:1709.05206*, 6:1662–1669, 2018.

[191] K. Karimi and H. J. Hamilton. Temporal rules and temporal decision trees: A C4.5 approach. Technical Report CS-2001-02, Department of Computer Science, University of Regina, 2001.

[192] I. Karlsson, P. Papapetrou, and H. Boström. Generalized random shapelet forests. *Data Mining and Knowledge Discovery*, 30(5):1053–1085, 2016.

[193] R. J. Kate and R. J. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (ACL 2006)*, 2006.

[194] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2011.

[195] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*, pages 67–72, 2017.

[196] S. Koço, C. Capponi, and F. Béchet. Applying multiview learning algorithms to human-human conversation classification. In *Proceedings of the 13th Annual Conference of the International Speech Communication Association (Interspeech 2012)*, pages 2322–2325, 2012.

[197] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 372–381, 2005.

[198] M. Korvas, O. Plátek, O. Dušek, L. Žilka, and F. Jurčíček. Free English and Czech telephone speech corpus shared under the CC-BY-SA 3.0 license. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2014)*, pages 4423–4428, 2014.

[199] V. Kreinovich, C. Quintana, and L. Reznik. Gaussian membership functions are most adequate in representing uncertainty in measurements. In *Proceedings of the North American Fuzzy Information Processing Society Conference (NAFIPS 1992)*, pages 618–624, 1992.

[200] S. A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.

[201] G. Lample, A. Conneau, L. Denoyer, and M. Ranzato. Unsupervised machine translation using monolingual corpora only. In *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*, 2018.

[202] G. Lample, M. Ott, A. Conneau, L. Denoyer, and M. Ranzato. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*, pages 5039–5049, 2018.

[203] P. Langley and S. Stromsten. Learning context-free grammars with a simplicity bias. In *Proceedings of the 11th European Conference on Machine Learning (ECML 2000)*, pages 220–228, 2000.

[204] P. L. Lanzi. Fast feature selection with genetic algorithms: A filter approach. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (IEEE CEC 1997)*, pages 537–540, 1997.

[205] E. Laparra, D. Xu, and S. Bethard. From characters to time intervals: New paradigms for evaluation and neural parsing of time normalizations. *Transactions of the Association of Computational Linguistics*, 6:343–356, 2018.

[206] H. Lee, Y. Zhang, M. Jiang, J. Xu, C. Tao, and H. Xu. Identifying direct temporal relations between time and events from clinical notes. *BMC Medical Informatics and Decision Making*, 18(S-2):23–34, 2018.

[207] K. Lee, Y. Artzi, J. Dodge, and L. Zettlemoyer. Context-dependent semantic parsing for time expressions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, pages 1437–1447, 2014.

[208] K. Lee, L. He, M. Lewis, and L. Zettlemoyer. End-to-end neural coreference resolution. *CoRR*, abs/1707.07045, 2017.

[209] A. Leeuwenberg and M. Moens. Temporal information extraction by predicting relative time-lines. *CoRR*, abs/1808.09401, 2018.

[210] F. A. Leoni, F. Cutugno, and R. Savy. CLIPS Corpus (Corpora e Lessici di Italiano Parlato e Scritto). `http://www.clips.unina.it`, 2005. Accessed: 2018-02-16.

[211] M. Leucker and C. Schallhart. A brief account of Runtime Verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[212] T. W. Liao. Clustering of time series data: A survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[213] C. Lignos, V. Raman, C. Finucane, M. P. Marcus, and H. Kress-Gazit. Provably correct reactive control from natural language. *Autonomous Robots*, 38(1):89–105, 2015.

[214] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (SIGMOD 2003)*, pages 2–11, 2003.

[215] W. Lin and M. Orgun. Temporal data mining using hidden periodicity analysis. In *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems (ISMIS 2000)*, pages 49–58, 2000.

[216] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*, volume 454. Springer Science & Business Media, 2012.

[217] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[218] Z. Liu, T. Mitamura, and E. H. Hovy. Graph-based decoding for event sequencing and coreference resolution. *CoRR*, abs/1806.05099, 2018.

[219] H. Llorens, L. Derczynski, R. J. Gaizauskas, and E. Saquete. TIMEN: An open temporal expression normalisation resource. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 3044–3051, 2012.

[220] H. Llorens, E. Saquete, and B. Navarro. TIPSem (English and Spanish): Evaluating CRFs and semantic roles in TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval 2010)*, pages 284–291, 2010.

[221] D. Lo, S.-C. Khoo, and J. Li. Mining and ranking generators of sequential patterns. In *Proceedings of the 2008 SIAM International Conference on Data Mining (SIAM 2008)*, pages 553–564, 2008.

[222] E. Loper and S. Bird. NLTK: The natural language toolkit. *CoRR*, cs.CL/0205028, 2002.

[223] W. Ma, Z. Ni, K. Cao, X. Li, and S. Chin. Seq2tree: A tree-structured extension of LSTM network. 2017.

[224] N. R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys*, 43(1):1–41, 2010.

[225] B. Magnini, E. Cabrio, R. Gretter, M. Kouylekov, and M. Negri. QALL-ME Benchmark (Question Answering Learning Technologies in a Multilingual and Multimodal Environment). `http://qallme.itc.it`, 2007. Accessed: 2018-02-10.

[226] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford CoreNLP natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, pages 55–60, 2014.

[227] S. T. March and A. R. Hevner. Integrated decision support systems: A data warehousing perspective. *Decision Support Systems*, 43(3):1031–1043, 2007.

[228] C. Martínez, F. Jiménez, and G. Sánchez. Multi objective evolutionary search. Available at `https://sourceforge.net/projects/moea/files/`, 2017.

[229] P. P. Mazur and R. Dale. WikiWars: A new corpus for research on temporal expressions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*, pages 913–922, 2010.

[230] C. Mballo and E. Diday. Decision trees on interval valued variables. *Symbolic Data Analysis*, 3(1):8–18, 2005.

[231] A. Mbarak, Y. Yetis, and M. Jamshidi. Data - based pollution forecasting via machine learning: Case of Northwest Texas. In *Proceedings of the 2018 World Automation Congress (WAC 2018)*, pages 1–6, 2018.

[232] A. McGovern, D. H. Rosendahl, R. A. Brown, and K. K. Droegemeier. Identifying predictive multi-dimensional time series motifs: An application to severe weather prediction. *Data Mining and Knowledge Discovery*, 22(1-2):232–258, 2011.

[233] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD 1995)*, pages 216–221, 1995.

[234] M. O. Mendez, M. Matteucci, V. Castronovo, L. Ferini-Strambi, S. Cerutti, and A. Bianchi. Sleep staging from heart rate variability: Time-varying spectral features and hidden Markov models. *International Journal of Biomedical Engineering and Technology*, 3(3-4):246–263, 2010.

[235] P. Mirza. Extracting temporal and causal relations between events. *CoRR*, abs/1604.08120, 2016.

[236] P. Mirza and S. Tonelli. CATENA: CAusal and TEmporal relation extraction from NAtural language texts. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING 2016)*, pages 64–75, 2016.

[237] G. Mishne, D. Carmel, R. Hoory, A. Roytman, and A. Soffer. Automatic analysis of call-center conversations. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM 2005)*, pages 453–459, 2005.

[238] T. Mitsa. *Temporal Data Mining*. Chapman and Hall/CRC, 2010.

[239] D. C. Montgomery, C. L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. John Wiley & Sons, 2015.

[240] F. Mörchen and A. Ultsch. Optimizing time series discretization for knowledge discovery. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD 2005)*, pages 660–665. ACM, 2005.

[241] M. Morchid, R. Dufour, M. Bouallegue, G. Linarès, and R. D. Mori. Theme identification in human-human conversations with features from specific speaker type hidden spaces. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association (Interspeech 2014)*, pages 248–252, 2014.

[242] M. Morchid, R. Dufour, P. Bousquet, M. Bouallegue, G. Linarès, and R. D. Mori. Improving dialogue classification using a topic space representation and a gaussian classifier based on the decision rule. In *Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014)*, pages 126–130, 2014.

[243] M. Morchid, G. Linarès, M. El-Bèze, and R. D. Mori. Theme identification in telephone service conversations using quaternions of speech features. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association (Interspeech 2013)*, pages 1394–1398, 2013.

[244] R. Moskovitch and Y. Shahar. Classification-driven temporal discretization of multivariate time series. *Data Mining and Knowledge Discovery*, 29(4):871–913, 2015.

[245] L. Mukkala, J. Arvo, T. Lehtonen, T. Knuutila, et al. Current state of ontology matching. A survey of ontology and schema matching. Technical report, University of Turku, 2015.

[246] S. Muresan, T. Muresan, and J. L. Klavans. Lexicalized well-founded grammars: Learnability and merging. Technical report, Columbia University, 2005.

[247] D. Neider and I. Gavran. Learning linear temporal properties. In *Proceedings of the 2018 Formal Methods in Computer Aided Design Conference (FMCAD 2018)*, pages 1–10, 2018.

[248] R. Nelken and N. Francez. Automatic translation of natural language system specifications into temporal logic. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV 1996)*, pages 360–371, 1996.

[249] M. Nerlove, D. M. Grether, and J. L. Carvalho. *Analysis of Economic Time Series: A Synthesis*. Academic Press, 2014.

[250] L. Nguyen, J. Kapinski, X. Jin, J. Deshmukh, K. Butts, and T. Johnson. Abnormal data classification using time-frequency temporal logic. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control (HSCC 2017)*, pages 237–242, 2017.

[251] S. Nihtianov and A. Luque. *Smart Sensors and MEMS: Intelligent Sensing Devices and Microsystems for Industrial Applications*. Woodhead Publishing, 2018.

[252] A. P. Nikora and G. Balcom. Automated identification of LTL patterns in natural language requirements. In *Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE 2009)*, pages 185–194, 2009.

[253] Q. Ning, Z. Feng, and D. Roth. A structured learning approach to temporal relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 1027–1037, 2017.

[254] Q. Ning, H. Wu, H. Peng, and D. Roth. Improving temporal relation extraction with a globally acquired statistical resource. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL-HLT 2018)*, pages 841–851, 2018.

[255] Q. Ning, Z. Yu, C. Fan, and D. Roth. Exploiting partially annotated data for temporal relation extraction. *CoRR*, abs/1804.08420, 2018.

[256] Q. Ning, B. Zhou, Z. Feng, H. Peng, and D. Roth. CogCompTime: A tool for understanding time in natural language. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*, pages 72–77, 2018.

[257] M. O'Mahony. *Sensory Evaluation of Food: Statistical Methods and Procedures*. Taylor and Francis Inc., 2018.

[258] H. M. Pandey, A. Chaudhary, and D. Mehrotra. Grammar induction using bit masking oriented genetic algorithm and comparative analysis. *Applied Soft Computing*, 38:453–468, 2016.

[259] M. A. Pandharipande and S. K. Kopparapu. A novel approach to identify problematic call center conversations. In *Proceedings of the 9th International Joint Conference on Computer Science and Software Engineering (JCSSE 2012)*, pages 1–5, 2012.

[260] M. Paprzycki, A. Abraham, R. Guo, and S. Mukkamala. Data mining approach for analyzing call center performance. In *Proceedings of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2004)*, pages 1092–1101, 2004.

[261] T. Parcollet, M. Morchid, G. Linarès, and R. D. Mori. Quaternion convolutional neural networks for theme identification of telephone conversations. In *Proceedings of the 2018 IEEE Workshop on Spoken Language Technology (SLT 2018)*, pages 685–691, 2018.

[262] T. Park and H. Kim. A data warehouse-based decision support system for sewer infrastructure management. *Automation in Construction*, 30:37–49, 2013.

[263] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, 1984.

[264] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The Prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[265] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[266] E. Pianta, C. Girardi, and R. Zanoli. The TextPro tool suite. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, 2008.

[267] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57, 1977.

[268] G. Polese. A decision support system for evidence based medicine. *Journal of Visual Languages & Computing*, 25(6):858–867, 2014.

[269] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The Kaldi speech recognition toolkit. In *Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding (ASRU 2011)*, pages 1–4, 2011.

[270] I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1–2):1–36, 2005.

[271] J. Pustejovsky, J. M. Castaño, R. Ingria, R. Saurí, R. J. Gaizauskas, A. Setzer, G. Katz, and D. R. Radev. TimeML: Robust specification of event and temporal expressions in text. *New Directions in Question Answering*, pages 28–34, 2003.

[272] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[273] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[274] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[275] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4(1):77–90, 1996.

[276] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2008.

[277] E. Rahm. Towards large-scale schema and ontology matching. In *Schema Matching and Mapping*, pages 3–27. Springer, 2011.

[278] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[279] A. Rajan. Automated requirements-based test case generation. *SIGSOFT Software Engeneering Notes*, 31(6):1–2, 2006.

[280] T. Rakthanmanon and E. Keogh. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *Proceedings of the 2013 SIAM International Conference on Data Mining (SIAM 2013)*, pages 668–676, 2013.

[281] N. Ramesh, M. A. Cane, R. Seager, and D. E. Lee. Predictability and prediction of persistent cool states of the tropical Pacific Ocean. *Climate Dynamics*, 49(7-8):2291–2307, 2017.

[282] A. Ranta. Translating between language and logic: What is easy and what is difficult. In *Proceedings of the International Conference on Automated Deduction (CADE 2011)*, pages 5–25. Springer, 2011.

[283] N. Reimers, N. Dehghani, and I. Gurevych. Temporal anchoring of events for the TimeBank corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2016.

[284] X. Renard, M. Rifqi, W. Erray, and M. Detyniecki. Random-shapelet: An algorithm for fast shapelet discovery. In *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA 2015)*, pages 1–10, 2015.

[285] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[286] K. Roitero, M. Soprano, A. Brunello, and S. Mizzaro. Reproduce and improve: an evolutionary approach to select a few good topics for information retrieval evaluation. *Journal of Data and Information Quality*, 10(3):12, 2018.

[287] L. Rokach and O. Maimon. *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Company Inc., 2nd edition, 2014.

[288] M. Saberi, O. Khadeer Hussain, and E. Chang. Past, present and future of contact centers: A literature review. *Business Process Management Journal*, 23(3):574–597, 2017.

[289] S. Salcedo-Sanz, M. Naldi, A. Pérez-Bellido, J. Portilla-Figueras, and E. Ortíz-García. Evolutionary optimization of service times in interactive voice response systems. *IEEE Transactions on Evolutionary Computation*, 14(4):602–617, 2010.

[290] T. Santos, G. Carvalho, and A. Sampaio. Formal modelling of environment restrictions from natural-language requirements. In *Proceedings of the 21st Brazilian Symposium on Formal Methods: Foundations and Applications, (SBMF 2018)*, pages 252–270, 2018.

[291] Y. Sasaki. The truth of the F-measure. *Teach Tutor Mater*, 1(5):1–5, 2007.

[292] P. Schäfer and U. Leser. Fast and accurate time series classification with WEASEL. In *Proceedings of the 2017 ACM Conference on Information and Knowledge Management (CIKM 2017)*, pages 637–646. ACM, 2017.

[293] K. K. Schuler, A. Korhonen, and S. W. Brown. VerbNet overview, extensions, mappings and applications. In *Proceedings of the Conference of the North American Chapter of the Association of Computational Linguistics on Human Language Technologies (NAACL-HLT 2009)*, pages 13–14, 2009.

[294] R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. *CoRR*, abs/1511.06709, 2015.

[295] M. Shah, J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning DTW-shapelets for time-series classification. In *Proceedings of the 3rd IKDD Conference on Data Science (CODS 2016)*, page 3, 2016.

[296] Z. Shang, T. Deng, J. He, and X. Duan. A novel model for hourly $PM_{2.5}$ concentration prediction based on CART and EELM. *Science of the Total Environment*, 651:3043–3052, 2019.

[297] R. Shanmugam and R. Chattamvelli. *Statistics for Scientists and Engineers*, chapter 4, pages 89–110. Wiley-Blackwell, 2016.

[298] W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10(5):335–347, 1989.

[299] L. Siguenza Guzman, V. Saquicela, and D. Cattrysse. Design of an integrated decision support system for library holistic evaluation. In *Proceedings of the International Association of University Libraries Conference (IATUL 2014)*, pages 1–12, 2014.

[300] R. Spraque. A framework for the development of decision support systems. *Management Information System Quarterly*, 4(4):1–26, 1980.

[301] M. Srinivas and L. Patnaik. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):656–667, 1994.

[302] B. Starkie. Inferring attribute grammars with structured data for natural language processing. In *Proceedings of the 6th International Colloquium on Grammatical Inference: Algorithms and Applications (ICGI 2002)*, pages 237–248, 2002.

[303] M. Steedman. *The Syntactic Process*, volume 24. MIT press Cambridge, MA, 2000.

[304] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The Data Tamer system. In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR 2013)*, 2013.

[305] J. Strötgen and M. Gertz. HeidelTime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation (SemEval 2010)*, pages 321–324, 2010.

[306] G. Sturla. *A Two-phased Approach for Natural Language Parsing Into Formal Logic*. PhD thesis, Massachusetts Institute of Technology, 2017.

[307] W. Sun, A. Rumshisky, and Ö. Uzuner. Temporal reasoning over clinical text: The state of the art. *Journal of the American Medical Informatics Association*, 20(5):814–819, 2013.

[308] Z. Sun. A framework for developing management intelligent systems. In *Decision Management: Concepts, Methodologies, Tools, and Applications*, pages 503–521. IGI Global, 2017.

[309] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[310] T. Suzuki, T. Furuhashi, S. Matsushita, and H. Tsutsui. Efficient fuzzy modeling under multiple criteria by using genetic algorithm. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 1999)*, volume 5, pages 314–319. IEEE Computer Society, 1999.

[311] J. Taylor. *Decision Management Systems: A Practical Guide to Using Business Rules and Predictive Analytics*. Pearson Education, 2011.

[312] J. Tourille. *Extracting Clinical Event Timelines: Temporal Information Extraction and Coreference Resolution in Electronic Health Records. (Création de Chronologies d'Événements Médicaux: Extraction d'Informations Temporelles et Résolution de la Coréférence dans les Dossiers Patients Électroniques)*. PhD thesis, University of Paris-Saclay, France, 2018.

[313] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj. Temporal attention-augmented bilinear network for financial time-series data analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[314] J. Trione, B. Favre, and F. Béchet. Beyond utterance extraction: Summary recombination for speech summarization. In *Proceedings of the 17th Annual Conference of the International Speech Communication Association (Interspeech 2016)*, pages 680–684, 2016.

[315] N. UzZaman, H. Llorens, J. F. Allen, L. Derczynski, M. Verhagen, and J. Pustejovsky. TempEval-3: Evaluating events, time expressions, and temporal relations. *CoRR*, abs/1206.5333, 2012.

[316] H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Proceedings of the 4th International Conference on Tools with Artificial Intelligence (TAI 1992)*, pages 200–203, 1992.

[317] V. Vagin, O. Morosin, M. Fomina, and S. Antipov. Temporal decision trees in diagnostics systems. In *Proceedings of the 2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD 2018)*, pages 1–10, 2018.

[318] L. Vanneschi, M. Castelli, and S. Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO 2010)*, pages 877–884, 2010.

[319] S. Vashishtha, B. V. Durme, and A. S. White. Fine-grained temporal relation extraction. *CoRR*, abs/1902.01390, 2019.

[320] G. Vellidis, M. Tucker, C. Perry, C. Kvien, and C. Bednarz. A real-time wireless smart sensor array for scheduling irrigation. *Computers and electronics in agriculture*, 61(1):44–50, 2008.

[321] M. Venkatadri and K. Srinivasa Rao. A multiobjective genetic algorithm for feature selection in data mining. *International Journal of Computer Science and Information Technologies*, 1(5):443–448, 2010.

[322] A. Viel, A. Brunello, A. Montanari, and F. Pittino. An original approach to positioning with cellular fingerprints based on decision tree ensembles. In *Proceedings of the 14th International Conference on Location Based Services (LBS 2018)*, pages 49–70, 2018.

[323] A. Viel, A. Brunello, A. Montanari, and F. Pittino. An original approach to positioning with cellular fingerprints based on decision tree ensembles. *Journal of Location Based Services*, 2018.

[324] A. Villagra, D. Pandolfi, J. Rasjido, C. Montenegro, N. Serón, and M. G. Leguizamón. Repair algorithms and penalty functions to handling constraints in an evolutionary scheduling. In *Proceedings of the 16th Congreso Argentino de Ciencias de la Computación (CACIC 2010)*, pages 41–51, 2010.

[325] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE 2004)*, pages 79–90, 2004.

[326] L.-Y. Wei, D.-Y. Huang, S.-C. Ho, J.-S. Lin, H.-E. Chueh, C.-S. Liu, and T.-H. H. Ho. A hybrid time series model based on AR-EMD and volatility for medical data forecasting: A case study in the emergency department. *International Journal of Management, Economics and Social Sciences*, 6:166–184, 2017.

[327] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[328] A. S. Wilkins. To lag or not to lag?: Re-evaluating the use of lagged dependent variables in regression analysis. *Political Science Research and Methods*, 6(2):393–411, 2018.

[329] Y. Wilks and D. Fass. The preference xemantics family. *Computers & Mathematics with Applications*, 23(2-5):205–221, 1992.

[330] M. Wistuba, J. Grabocka, and L. Schmidt-Thieme. Ultra-fast shapelets for time series classification. *arXiv preprint arXiv:1503.05018*, 2015.

[331] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 4th edition, 2016.

[332] J. Xie, Z. Liao, X. Fang, X. Xu, Y. Wang, Y. Zhang, J. Liu, S. Fan, and B. Wang. The characteristics of hourly wind field and its impacts on air quality in the Pearl River delta region during 2013–2017. *Atmospheric Research*, 227:112–124, 2019.

[333] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia. The impact of feature selection on defect prediction performance: An empirical comparison. In *Proceedings of the IEEE 27th International Symposium on Software Reliability Engineering (ISSRE 2016)*, pages 309–320, 2016.

[334] Z. Xu, M. Ornik, A. A. Julius, and U. Topcu. Information-guided temporal logic inference with prior knowledge. In *Proceedings of the 2019 American Control Conference (ACC 2019)*, pages 1891–1897, 2019.

[335] R. Yan, C. Cheng, and Y. Chai. Formal consistency checking over specifications in natural languages. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE 2015)*, pages 1677–1682, 2015.

[336] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SIAM 2003)*, pages 166–177, 2003.

[337] L. Ye and E. Keogh. Time series shapelets: A new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009)*, pages 947–956, 2009.

[338] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60, 2001.

[339] S. Zhang, R. Rudinger, and B. V. Durme. An evaluation of PredPatt and Open IE via stage 1 semantic role labeling. In *Proceedings of the 12th International Conference on Computational Semantics (IWCS 2017)*, 2017.

[340] J. Zhao, P. Papapetrou, L. Asker, and H. Boström. Learning from heterogeneous temporal data in electronic health records. *Journal of Biomedical Informatics*, 65:105–119, 2017.

[341] X. Zhong and E. Cambria. Time expression recognition using a constituent-based tagging scheme. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web (WWW 2018)*, pages 983–992, 2018.

[342] X. Zhong, A. Sun, and E. Cambria. Time expression analysis and recognition using syntactic token types and general heuristic rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*, pages 420–429, 2017.

[343] L. Žilka. Temporal logic for man. Master's thesis, Brno University of Technology, 2010.

[344] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[345] S. Zvada and T. Gyimóthy. Using decision trees to infer semantic functions of attribute grammars. *Acta Cybernetica*, 15(2):279–304, 2001.

[346] G. Zweig, O. Siohan, G. Saon, B. Ramabhadran, D. Povey, L. Mangu, and B. Kingsbury. Automated quality monitoring for call centers using speech and NLP technologies. In *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL 2006)*, pages 292–295, 2006.