Alma Mater Studiorum
University of Bologna

Campus of Cesena
Department of Informatics - Science and Engineering
Master's degree in Engineering and computer science

AutoML: A new methodology to automate data pre-processing pipelines

Thesis in the subject of
Data Mining

supervisor
Professor Matteo Golfarelli

presented by
Joseph Giovanelli

in collaboration with
Professor Alberto Abelló
Dr. Besim Bilalli

March 19th 2020

Thesis advisor: Professor Matteo Golfarelli Joseph Giovanelli

AutoML: A new methodology to automate data pre-processing pipelines

Abstract

It is well known that we are living in the Big Data Era. Indeed, the exponential growth of Internet of Things, Web of Things and Pervasive Computing systems greatly increased the amount of stored data. Thanks to the availability of data, the figure of the Data Scientist has become one of the most sought, because he is capable of transforming data, performing analysis on it, and applying Machine Learning techniques to improve the business decisions of companies. Yet, Data Scientists do not scale. It is almost impossible to balance their number and the required effort to analyze the increasingly growing sizes of available data. Furthermore, today more and more non-experts use Machine Learning tools to perform data analysis but they do not have the required knowledge. To this end, tools that help them throughout the Machine Learning process have been developed and are typically referred to as AutoML tools. However, even with the presence of such tools, raw data (i.e., without being pre-processed) are rarely ready to be consumed, and generally perform poorly when consumed in a raw form. A pre-processing phase (i.e., application of a set of transformations), which improves the quality of the data and makes it suitable for algorithms is usually required.
Most of AutoML tools do not consider this preliminary part, even though it has already shown to improve the final performance. Moreover, there exist a few works that actually support pre-processing, but they provide just the application of a fixed series of transformations, decided a priori, not considering the nature of the data, the used algorithm, or simply that the order of the transformations could affect the final result. In this thesis we propose a new methodology that allows to provide a series of pre-processing transformations according to the specific presented case. Our approach analyzes the nature of the data, the algorithm we intend to use, and the impact that the order of transformations could have.

To all who have always been there.

Acknowledgments

The success of this thesis is certainly merit of my supervisor, Matteo Golfarelli, who gave me the opportunity to undertake the amazing experience at UPC, Universitat Politècnica de la Catalunya, who always offered me technical support, despite the distance, and continues to offer me professional growth opportunities. I would like to thank Alberto Abello and Besim Bilalli in the same way, they followed me step by step throughout the research. The welcome in Barcelona was the best and your collaboration was precious. Not least I can thank the two corresponding research groups. Regarding the prof. Golfarelli's one, in Cesena, I would like to thank Enrico, Matteo, Anna, Nicola and Sara for their support, even in the most disoriented moments. In the same way I would like to thank the whole research group of prof. Abello in Barcelona, especially Moditha, Rediana and Jam. The Spanish experience would not have been the same without you.

Afterwards, I'd like to thank someone who is not here today, for all the support she gave me, not in just these last five years. Thanks Ilaria, for a long time you were everything and, if it were not for you, I would be a different person today. I would like also to thank Elena, Patrizio, Giovanni, Giacomo and Teresa. I think it is a shared hope to be able to understand the importance of the things not only when we have lost them. About this I would really like to start from here and thank the people who have always been there for me. First of all, my parents. Sometimes we do our worst just with the people who least deserve it, those who are closest to us. Thank you Mum. Thank you Dad. Not only for making this day possible but also for enduring me day after day and continuing to give me love in the most sincere way, without expecting anything back. I love you both. Thanks brother, your speeches are always the most teaching ones. You have always been a reference point for me and you always will be. When I was a child I thought there was no better family, I wish you knew that I still think so. Thanks Alba for bringing bright into this family, which is now yours too, with Blue and Lupo.

A lot things have been going on and there are some people that have always hugged me, talked to me and helped me to get up. Those people are my Friends. Not just simple friends, those of a lifetime. In all these years, I messed up, a lot, I haven't been myself at all, sometimes perhaps too much but in any case they accepted me and they never made me feel alone. Who knows

me, knows how much this matters to me. I want to thank Alessandro who has always been ready to come to me and pick me up, every time I went to slam, not just figuratively. I would like to thank in the same way Edoardo T., Edoardo C., Alessandra, Eugenio, Sara, Michele, Federica, Claudio and Camilla G. I have a really different relationship with each of you, but each one is, in his own, special. You have always been close to me, most of you since kindergarten and, above all, you have always been a shoulder to lean on. I also would like to thank Filippo, Camilla P., Giacomo B., Giovanni, Giacomo C., Alberto, Francesca, Lodovico and Giulia. They would all deserve more than just a thank you line, but believe me, you would have to read until tomorrow. My gratitude can never overcome laughter, tears, smiles, hugs and so on.
Last but not least, in these five years I have had the possibility to meet a lot of wonderful people. I collaborated with them and developed numerous projects but not only; between beers, football games, dinners and anything else we have had the opportunity to become friends more than ever. Thanks Matteo, Andrea C., Marcello, Diego P., Luca, Giulia, Andrea P., Marco, Andrea D., Eugenio, Silvio, Diego M. and Vincenzo. Although, among all of them, I feel I should thank in a particular way Giuseppe. We have been roommates, soccer teammates, classmates, project mates and so forth. In these five years we have shared a lot, thanks for everything.

Thank you all, I feel really lucky to have you all in my life.

Contents

1 Introduction 13

2 Towards AutoML 17
- 2.1 Machine Learning . . . 18
- 2.2 The Machine Learning role in Data Science . . . 25
  - 2.2.1 Domain & Data understanding . . . 26
  - 2.2.2 Data Preparation . . . 28
  - 2.2.3 Data mining . . . 33
- 2.3 The AutoML approach . . . 39
- 2.4 The state-of-the-art solutions . . . 42
  - 2.4.1 Distributed tools . . . 42
  - 2.4.2 Cloud-based tools . . . 44
  - 2.4.3 Centralized tools . . . 46

3 Bayesian techniques for AutoML 51
- 3.1 Bayesian techniques and the SMBO algorithm . . . 52
  - 3.1.1 Gaussian Processes (GP) regression . . . 55
  - 3.1.2 Tree-structured Parzen Estimator (TPE) approach . . . 56
  - 3.1.3 Sequential Model-based Algorithm Configuration (SMAC) . . . 56
- 3.2 CASH and DPSO problems . . . 58
  - 3.2.1 SMBO as a CASH resolution . . . 61
  - 3.2.2 SMBO as a DPSO resolution . . . 63

4 Automated Data Pre-processing 67
- 4.1 General architecture . . . 68
- 4.2 Offline phase . . . 74
  - 4.2.1 Intermediate table building . . . 75
  - 4.2.2 SMBO experiments and insights interpretation . . . 79
  - 4.2.3 Meta-learning process . . . 85

4.3 Online phase 93
4.3.1 Data pipeline prototypes building 93
4.3.2 Data pipeline prototypes optimization 96

5 Evaluation 97
5.1 Data Pre-processing importance 98
5.2 Evaluation of the Automated Data Pre-processing approach 100

6 Conclusions and future developments 107

References 111

Listing of figures

2.1 Machine Learning process scheme 18
2.2 Graphic representation of the outcome of a classifier 21
2.3 Graphic representation of the cross validation technique 24
2.4 Decision Tree example 34
2.5 Example of how Decision Tree splits work 35
2.6 Two-dimensional representation of K-Nearest Neighbor 36
2.7 Trend of the growth of human and machine-generated data 39
2.8 MLBase infrastructure 43
2.9 Auto-Sklearn infrastructure 47
2.10 Auto-Sklearn Pre-processing operators and machine learning algorithms 48
2.11 A Quemy's pipeline instance 49
2.12 Quemy's reseacrh space 49

3.1 SMBO algorithm example 53
3.2 Gaussian Processes (GP) interpolation 55
3.3 Example of a Regression Decision Tree 57
3.4 Example of Random Forest 57
3.5 Machine Learning problems scheme 59
3.6 A combined hierarchical hyper-parameter optimization problem example 61
3.7 Hierarchical Dependencies in CASH and DPSO problems 63
3.8 Quemy's experiments results, accuracy changes in 100 iterations 64
3.9 Quemy's experiments results, best pipelines explored 65

4.1 Case in which the global application of the transformations is incorrect . 69
4.2 Previous case applying the transformations only to compatible attributes 69
4.3 Domain and Co-domain of the considered transformations 70
4.4 Naive approach to consider all the data pipeline prototypes 71
4.5 Online phase of our approach 72
4.6 Online phase on how to find the data pipeline prototypes 72

4.7 Offline phase that allows us to build the Dependency table 74
4.8 Table resulting from the compatibility analysis 75
4.9 Compatibility analysis, Encode-Normalize representation 76
4.10 Compatibility analysis, Discretize-Normalize representation 76
4.11 Table of constraints not considering the used framework 77
4.12 Intermediate table construction 78
4.13 Graphical representation of the performed SMBO experiments 80
4.14 Result label extraction from the winning data pipeline 81
4.15 Enumeration of all possible pipelines 82
4.16 Graphs depicting the number of valid and invalid results 83
4.17 Graphs depicting the labels about the valid results 84
4.18 Graphs depicting the labels about the valid results 84
4.19 Meta-learning working 86
4.20 Graphs depicting the labels before and after the grouping 87
4.21 Effects of Rebalancing step 90
4.22 Example of assigning an order to the no_order instances 91
4.23 Comparison of how to order no_order instances 91
4.24 Study of meta-learners with different seeds 92
4.25 Comparison of performances between GBM and XGBoost 93
4.26 Table of constraints after the SMBO experiments 94
4.27 BPMN scheme representing the possible data pipeline prototypes 95

5.1 Comparison between Pre-processing and Modeling optimizations 99
5.2 Comparison between our approach and the Quemy's one 101
5.3 Estimation of how much the winning approach improves the result . . . 103
5.4 Discretization transformation's role in our pipeline 104
5.5 Comparison between our approach and the Pseudo-exhaustive one . . . 105

List of Tables

2.1 Confusion Matrix 21

3.1 A combined hierarchical hyper-parameter optimization problem example 62

4.1 Rules for validating and assigning the result label to two configurations . 82
4.2 Feature - Rebalance results 89
4.3 Feature - Rebalance results with oversampling 89

5.1 Comparison between Pre-processing and Modeling optimizations results 99

1 Introduction

Coca-Cola[25], with more than 500 drink brands sold in more than 200 countries, is the largest beverage company in the world. A monitoring system throughout all the supply chain generates a large amount of data and the company exploits it by supporting new product development.

Heineken[26], a worldwide brewing leader, is looking to catapult its success in the United States by leveraging the vast amount of data they collect. From data-driven marketing[29], to the Internet of Things[24], to improving operations through data analytics[17], Heineken looks to improve its operations, marketing, advertising and customer service.

As we can see, the concept of Machine Learning[10] is growing in popularity, specially in the world of e-commerce and business activity. In Computer Science, Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed. In a nutshell, it means to acquire the capacity to observe, see patterns such as grouping of similar objects and then apply what it has been discovered. This is typically a human skill but

machines are even better in it because they can use more data and data with more dimensions. Most of the methods used in Machine Learning are exploited in Data Science[9] to analyze and extract information from the data. Vice versa, some Data Science techniques are frequently applied to improve Machine Learning results. In particular, in this thesis we analyze in detail the Pre-processing[20] techniques. Data Scientists transform the data through these techniques based on the nature of the data and the Machine Learning algorithm to be used. The Pre-processing step has become really important in the process of extracting knowledge from data, because nowadays more and more raw data perform poorly or cannot be directly consumed by Machine Learning algorithms as they are. All this comes from an exponential growth of Internet of Things, Web of Things[15], and Pervasive Computing[33] systems that produce large amounts of data, often unstructured, or otherwise not suitable for the algorithms in question. This growth has also led to an increase in the need to analyze such data and derive profit from it. The Data Scientist has become one of the most sought figures of the twenty-first century, but the numerous skills expected (IT, mathematics, statistics, business, cooperation) make it difficult to increase the number of Data Scientists. All this leads to non-expert users performing data analysis and Machine Learning without having the adequate knowledge. The result is that non-experts are overwhelmed by the large amount of available and applicable techniques, hence automatic tools that help them throughout the Machine Learning process are required. These tools are typically referred to as AutoML tools. Yet, they focus more on the application of Machine Learning algorithms and pay little attention to the Pre-processing part. Moreover, the few ones that actually support the automation of this phase provide just the application of a fixed series of transformations, called pipelines[32]. A fixed pipeline means that it is decided a priori, and it does not consider the nature of the data, the used algorithm, or simply that the order of transformations could affect the final result. The fixed pipeline approach is widely used, since trying out several different Pre-processing pipelines on a data-set is a highly expensive operation and the number of possible pipelines increases with the increase of the number of transformations involved.

In this thesis, we propose a new methodology that allows to recommend a Pre-processing pipeline according to the specific presented case. We studied the transformations, trying to understand their domain, co-domain and how they work. Thanks to this study we realized that, in some cases the semantics of transformations imposes a predetermined order.

In other cases, instead, it is the used technology that does not allow some pipelines. In this way we managed to decrease the total search space and therefore the number of pipelines to be tested. Moreover, since some constraints are not imposed neither by the semantics of the transformations nor by the used technology, but rather by the nature of the data-set and by the used algorithm, we had to perform some experiments. Specifically, we collected several different data-sets and, considering the most used Machine Learning algorithms, we tested various Pre-processing pipelines. In this way, we discovered hidden insights that the semantics of transformations did not show. But above all, we collected enough data to be able to use the Machine Learning algorithms themselves to discover the dependencies between transformations, data-sets, and used algorithms. We were able to discard some other pipelines, and hence test just the promising ones. In order to evaluate the pipelines and choose the best one, we used the Bayesian techniques[3], which is state-of-the-art in this regard. Our approach takes into account the nature of the data, the algorithm we intend to use, and the impact that the order of transformations could have. This will open the road to more effective AutoML tools.

In chapter 2, we give a comprehensive introduction to Machine Learning, Data Science, and the application of these branches to real-case problems. In addition, we offer an analysis of the state-of-the-art of the available AutoML tools, with related pros and cons. In chapter 3, we provide a detailed explanation about Bayesian techniques and how they are applied in the AutoML field. Indeed, they underlie most AutoML tools, including ours. In chapter 4, we illustrate our approach and in 5, we discuss the results of some experiments performed with the aim to evaluate it. Finally, in 6, we discuss the contribution given by our research, the limitations of this work and outline future work.

2

Towards AutoML

In order to have a clear comprehension of the thesis, in this chapter it is given an introduction to the topic. Before talking about AutoML, which stands for Automated Machine Learning, a background of Machine Learning is needed and, no less important is its link with the Data Science field. In fact, the value of the entire project resolves around this connection.
Once this knowledge is provided, we expose the objectives of the AutoML but also the need and the causes that led to the development of this branch. Further, we present a state-of-the-art overview and the related works on this topic.

2.1 Machine Learning

We use machines to solve problems, thus we write down a sequence of instructions and compose an algorithm, which in turn is capable of transforming an input to an output. Although this approach allows us to to solve a lot of problems, some others are not easily solved with an algorithm. For example, we can devise an algorithm to sort an array but not to distinguish spam and legitimate emails. We know the input and the output, respectively an email document and an answer yes/no indicating whether the message is spam or not, but we do not how to transform the input to the output.

We know that humans learn from their past experiences and machines follow instructions given by humans. But what if humans can train the machines to learn from past data? In other words, we would like to collect a set of messages, which we know if they are spam or not, and we would like the computer to learn how to distinguish them when a new email arrives.

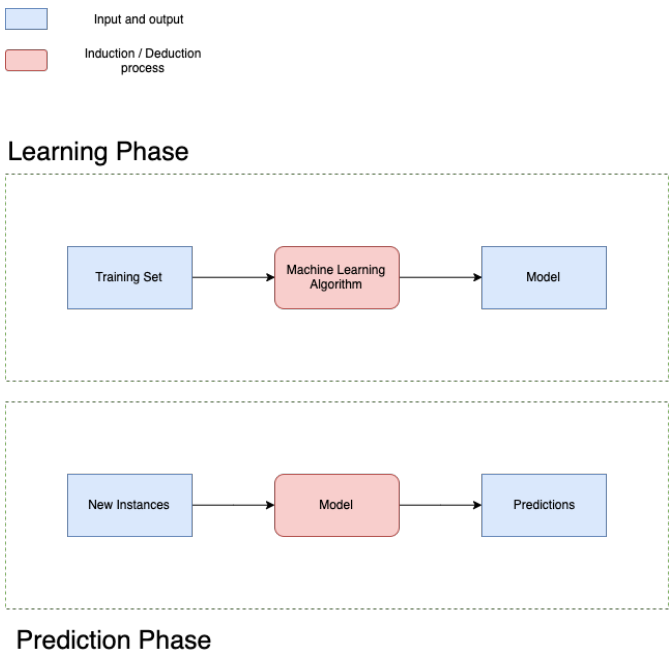


Figure 2.1: This scheme summarizes a typical Machine Learning flow.

In Figure 2.1 we can see a scheme that describes well the above mentioned example. The collected set of messages, from which the machine will learn to tell spam email from legitimate ones, is also called *Train Set*, sometimes abbreviated to TS. This name is coming from the fact that the computer is using it to train itself to understand the *class* (or *label*) of messages. In this case, the class is one of "spam" / "not spam". The *Machine Learning Algorithm* is the core of the process, indeed, applying it the machine can *learn*. This phase is called *Training* or *Learning Phase*. The result is *Model*, the outcome of an induction process on the training set data. Through this a new message can be categorized: a *prediction* of the class can be inferred. Machine learning (ML) tasks are typically classified into three broad categories, depending on the nature of the learning method:

- Supervised learning, examples of inputs and related desired outputs are presented, thus the goal is to learn a general rule that maps inputs to outputs;
- Unsupervised learning, no desired output is given and the goal is to find a structure in the input;
- Reinforcement learning, the idea here is to learn through a dynamic environment. Some actions are performed and we want to understand which ones have a positive effect and which not. To achieve this goal, in addition to past data, the environment tells us, through a score, how well we are doing.

Another categorization of Machine Learning tasks arises when one considers, instead, the kind of input and output:

- Classification task, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or (multi-label classification) more of these classes. This is typically tackled in a supervised way;
- Regression task, also a supervised problem, the outputs are continuous rather than discrete. For instance, predict the price of a used car given some car attributes that could affect a car's worth, such as brand, year, mileage, etc.;
- Clustering task, a set of inputs has to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task;

- Density estimation, the goal is to find the distribution of inputs in some space;
- Dimensionality reduction, simplifies inputs by mapping them into a lower-dimensional space.

Machine Learning covers a wide range of problems and, in order to fully understand them, each one requires its own in-depth study on the subject.
In this thesis, we focus on Supervised learning, in particular Classification problems. An example would be the problem of distinguishing between "spam" and "non spam" emails.
First of all, we list the three different Classification types:

- Binary, assign an instance to one of two possible classes (often called positive and negative one),
- Multiclass, assign an instance to one of $n > 2$ possible classes;
- Multilabel, assign an instance to a subset $m \leq n$ of the possible classes.

Regardless of the number of the classes, the problem can be formalized as follows:

- The Machine Learning algorithm is provided with a set of input/output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$,
- The learned model consists of a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ which maps inputs into their outputs (e.g. classify emails).

Training a model implies searching through the space of possible models (aka hypotheses). Such a search, typically aims at fitting the available training examples well according to a chosen *performance measure*. There are several measures with different meanings, and they are explained below.
Considering a Binary Classification problem, we have instances belonging to the *positive* class, for example "spam", and others to the *negative* class, "not spam". Usually the positive class is the one we are looking for, the one we are interested in distinguishing. Given the N instances to be classified, the result of each of the classification attempts can be:

- True Positive (TP): a positive instance has been correctly assigned to the positives;

- True Negative (TN): a negative instance has been correctly assigned to negatives;
- False Positive (FP): a negative instance has been incorrectly assigned to positives. Also called Type I or False error;
- False Negative (FN): a positive instance has been incorrectly assigned to negatives. Also called Type II or Miss error.

In Figure 2.2 we have a graphic representation.

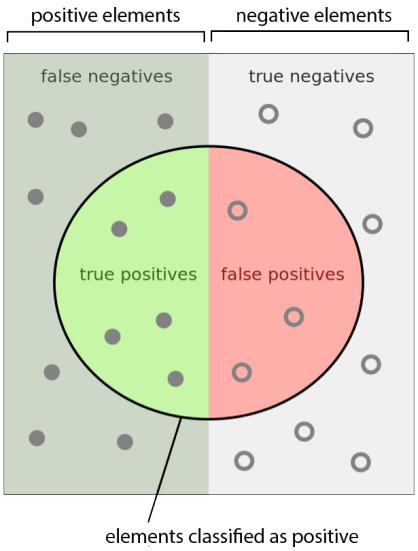


Figure 2.2: Graphic representation of the outcome of a classifier. From Wikimedia Commons, the free media repository.

The Confusion Matrix (Table 2.1) evaluates the ability of a classifier based on these indicators.

	Predicted Class		
Actual Class		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

Table 2.1: Confusion Matrix.

In the cells TP, TN, FP and FN there are the absolute frequencies of the relative classifications and we can define the following metrics:

$$Accuracy: \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N}$$

$$Misclassificationerror: \frac{FP + FN}{TP + TN + FP + FN} = 1 - Accuracy$$

Accuracy is the most widely used metric to synthesize the information of a Confusion Matrix but is not appropriate if the classes differ a lot in terms of the number of instances they contain. Considering a Binary Classification problem in which we have

- 9990 records of the first class;
- 10 records of the second class;

A model that always returns the first class will have an accuracy of $\frac{9990}{10000} = 99.9\%$. A data-set like this one is called imbalanced data-set.
Precision and Recall are two metrics used in applications where the correct classification of positive class records is more important. Considering the positive class the "rare" one, we can have a clearer idea of the classifier's behavior on these instances:

- Precision measures the fraction of record results actually positive among all those who were classified as such. High values indicate that few negative class records were incorrectly classified as positive;
- Recall measures the fraction of positive records correctly classified. High values indicate that few records of the positive class were incorrectly classified as negatives;
- F-measure is defined as the harmonic mean of Precision and Recall and, indeed, conveys the balance between them.

$$Precision : \frac{TP}{TP + FP}$$

$$Recall : \frac{TP}{TP + FN}$$

$$F - measure : 2 * \frac{Precision * Recall}{Precision + Recall}$$

Precision, Recall and F-measure are metrics that can be calculated for each class by reversing the positive class with the negative and vice versa.
In the case we have more than one class, the confusion matrix will be $n \times n$, therefore a column and a row for each class, and the above metrics can be calculated for each class considering the class in question as positive and all the others as negative.
Last but not least, a metric similar to the Accuracy, but which avoids inflated performance estimates on imbalanced data-sets, is Balanced Accuracy [4,18]. It is the average of recall scores per class or, equivalently, raw Accuracy where each instance is weighted according to the inverse prevalence of its true class. Thus for balanced data-sets, that metric is equal to Accuracy.
Now that we know what performance measures are, we are going to understand how they can be used properly. In fact, if we measured the performance on the data used for training, we would overestimate its goodness.
It was already mentioned that training a model aims at learning a function which maps inputs into outputs. Even if this process is done through the instances at our disposal, the learned model should perform well on unseen data. Ideally, in a Machine Learning problem we have enough amount of data to both build the model and *test* it. Test the model means check effectively that it *generalizes*, that is, it also performs well on non-train data. This means evaluating the above metrics on the new instances. The way this is done is to split the whole amount of data into two portions: Train and Test Set. The algorithm is trained in the Train Set and tested in the Test Set. This method is called *Hold-Out* and generally it could be applied by keeping 60% (or 80%) of the data for the train and the remaining 40% (or 20%) for the test.
However, in practice it happens that the only data available is that of the Training Set, or better said, it may happen that the data available is not enough for training and testing the model. This is where sampling techniques come into play to evaluate the model, without incurring *Over-fitting* or *Under-fitting*. Indeed the risk would be that of keeping the Train-

ing set big and the Test set small which leads to better accuracy in the learning phase. The model perfectly fits the data but it does not "learn the rule" and thus it does not generalize. The latter means that the model does not perform well on unseen data (Over-fitting). On the other hand, if the data for training is reduced, the algorithm may not have enough instances to build a good model and thus not capture enough patterns. This means it would perform poorly both in Train and Test Set (Under-fitting). Therefore, what is required is a method that provides sample data for training the model and also leaves sample data to validate it. K-Fold cross validation does exactly that. It can be viewed as repeated Hold-Out and we simply average scores after K different runs. In practice, the data set is divided into groups (folds) of equal number, iteratively excludes one group at a time and tries to predict it with the groups not excluded. Every data point gets to be in the Test Set exactly once, and gets to be in the Train Set k-1 times. This significantly reduces Under-fitting as we are using most of the data for fitting, and also significantly reduces Over-fitting as most of the data is also being used in Test Set. In Figure 2.3 we can observe a working example.

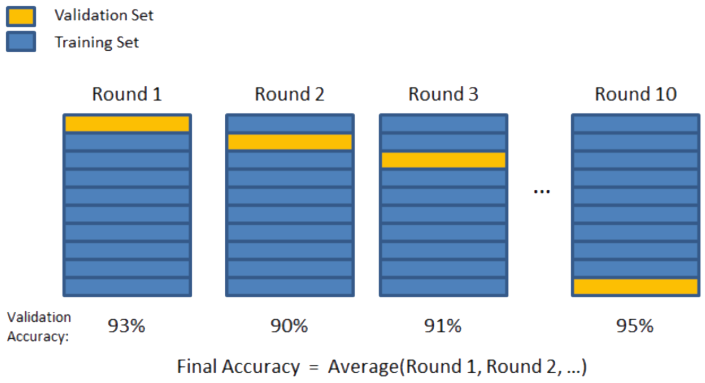


Figure 2.3: Graphic representation of the cross validation technique. From Georgios Drakos's article in Medium, Cross-Validation [11].

This method is a good choice when we have a minimum amount of data and we get sufficiently big difference in result quality between folds. Through the number of folds we can control the used amount of data to train and test the learners in the folds. As a general rule, we

choose k=5 or k=10, as these values have been shown empirically to yield test error estimates that suffer neither from excessively high *Bias* nor high *Variance*.

- The Bias error is an error from erroneous assumptions in the learning algorithm. High Bias can cause an algorithm to miss the relevant relations between features and target outputs (Under-fitting);
- The Variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (Over-fitting).

Leave-one-out cross validation is K-fold Cross Validation taken to its logical extreme, with K equal to N, the number of instances in the set. That means that N separate times, the function approximator is trained on all the data except for one instance and a prediction is made for that instance. As before the average error is computed and used to evaluate the model. The evaluation given by Leave-One-Out Cross Validation is recommended in cases where the data-set is particularly small and you want to "lose" the least possible number of instances for the training but, in the meanwhile, we want to validate all the instances. A drawback of this approach is the cost involved for building as many models as the number of instances. A crucial point of the Cross Validation techniques is that their purpose is not to come up with our final model: assuming to use 5 folds, we do not use these 5 models to do any real prediction, for that we want to use all the data because we have to come up with the best model possible.

2.2 The Machine Learning role in Data Science

Nowadays several different companies are using Machine Learning techniques behind the scenes both to impact our everyday lives and to improve their own businesses. Specifically, a massive quantity of data is collected to make predictions and hence to take decisions that are more profitable for the company. The application of Machine Learning methods to large databases, with the purpose of extracting knowledge and useful information, is called *Data Mining*. The analogy is that a large volume of earth and raw material is extracted from a mine, which when processed leads to a small amount of very precious material; similarly, in Data

Mining, a large volume of raw data is processed to construct a simple model with valuable use. However, Data Mining is just a piece of a larger process called *Knowledge Discovery in Databases*, also abbreviated to KDD. The goal of knowledge discovery is wider than just applying Machine Learning techniques and finding a suitable model. We are interested on what the data itself could bring, for example to detect similarities and to find interesting and unexpected patterns. Most of the time, we want to describe better the data, to reason about the information we find, and to understand and explain why there are certain patterns. Discovering knowledge from data should therefore be seen as a process containing several steps:

1. Domain & Data understanding;
2. Data Preparation, also called Pre-processing;
3. Data mining to discover patterns;
4. Post-processing of discovered patterns;
5. Results manipulations and interpretation, it is important to explain, read and present the results obtained in the correct way.

Below we will go deeper on the first three phases of this process, since they are those on which this thesis focuses.

2.2.1 Domain & Data understanding

Certainly a fundamental requirement for carrying out a useful analysis is having understood the domain of the data. Without that the Data Scientist, therefore the expert in charge to discover and reason about data, would not be able to take conscious decisions in the mining process. He has to know what the input means, how to interpret the results; he has to be able to recognize if a valuable outcome appears and, if not, why.
On the other hand, what makes a Data Science task different from a common Machine Learning one is not the domain knowledge to acquire but rather that of data. Knowing the nature of the data allows the Data Scientist to manipulate and transform it coherently. Usually in Machine Learning problems we have just numerical data, of the set of real numbers. In our

case the data-sets are more heterogeneous.
First of all, since we defined what a Classification task consist of, but we did not have formally defined what a data-set is, we proceed to explain it.

We said:

- The Machine Learning algorithm is provided with a set of input/output pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

The set of all the (x_i, y_i) pairs provided as input constitutes the data-set. Specifically, each pair is considered an instance, or *record*, and while y_i is the class, therefore a single value belonging to a finite set, x_i is a more complex component. Indeed this latter is formed by columns, commonly called *features* or *attributes*. These columns represent the characteristics of each instance; in other words they provide a description of it. Previously we also said:

- The learned model consists of a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ which maps inputs into their outputs (e.g. classify emails).

Even here, considering what we have just explained, we can go deeper in what the model consists of. Think, for example, of a supermarket chain that has hundreds of stores all over a country, selling thousands of goods to millions of customers. The point of sale terminals record the details of each transaction: date, customer identification code, goods bought and their amount, total money spent, and so forth. We now know that these represent the features. What the supermarket chain wants, is to be able to predict who are the likely customers for a product. Considering the product's type as class, the Machine Learning algorithm is trying to find some patterns in the features that allow the supermarket to determine the product's type. So, the learned model $f : \mathcal{X} \longrightarrow \mathcal{Y}$ establishes relations between the domain of the features and the domain of the classes.
Features can be of the following types:

- Numerical, express a measurable quantity and they could be of two kinds:
  - The ones where the difference between the values has a meaning, i.e. there is a unit of measurement, for instance the date or the temperature in Celsius degrees;

  - The ones which also have an absolute zero and hence the ratio between the values has a meaning too, e.g. age, mass, length and so on;

- Categorical, they express qualitative phenomena and stand out in:
  - Nominals, different names of values. We can only distinguish them, e.g. gender or eyes color;
  - Ordinals, values which allow us to sort objects based on the attribute value such as a rating or the hardness of a mineral.

There are actually many other types of data, such as images, text, space-time data and so on. Generally the area of data mining focuses on categorical and numerical data, also called structured data. Another type that we could run into in this category is the Timestamp, sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second.

2.2.2 Data Preparation

As we just said in the previous section, knowing the nature of the data allows the Data Scientist to manipulate and transform it coherently. The transformations which he uses are called Pre-processing transformations because they are applied before the Machine Learning process. It has been proven that they improve the classifier to predict better[2]. In litterature , there are several different transformations:

- Imputation;
- Rebalancing;
- Features Engineering;
- Discretization;
- Normalization;
- Encoding.

For each one we have many available operators; in this section we will go into detail on each of them.
We start talking about the *Imputation techniques*. Sometimes, for some instances, the value of some attributes could not be present. This could happen because the information was not collected, e.g. the interviewee did not indicate his age and weight, or the attribute is not applicable to all objects, e.g. the annual income does not make sense for children. However, there are several different ways to handle it:

- Delete the objects that contain them, if the data-set is sufficiently numerous;
- Let the algorithm handle them;
- Manually fill in the missing values, generally too time consuming;
- Automatically fill in missing values, using the so called *Imputation techniques*.

These techniques are divided into:

- Univariate imputation techniques, which enter a constant value in place of missing values or estimate them with the mean (or the mode) of the attribute in question;
- Multivariate imputation techniques, for each instance predict the value of the missing attribute based on other known attributes. In this case, Data Mining algorithms would be used to prepare input data for other Data Mining algorithms.

Instead, regarding the *Re-balancing techniques*, we are using them when we have to deal with data-sets in which one or more classes have a far greater, or lesser, number of instances than the others. These kinds of data-sets are called imbalanced data-sets and, using the data as it is, could create problems when Machine Learning algorithms are applied. Since the goal of the algorithms is to maximize the predictive accuracy, that is equivalent to minimize the misclassification error, for the classifier is more convenient to have a prediction tending towards the majority class. In that way we would have a model which will not generalize. The Re-balancing, also called re-sampling techniques were conceived to equilibrate the number of instances for each class, making the algorithm learn on a balanced training data-set.
There are two main different approaches:

- Under-sampling, involves dropping some instances from the majority classes;
- Over-sampling, involves supplementing the instances of the minority classes.

Both have several different techniques, most of them based on the K-Nearest Neighbor Machine Learning algorithm. In a nutshell, the points of the training set are drawn in a multi-dimensional space and, instead of discarding/adding points randomly, the sampling is done in such a way as to maintain consistency with existing points. For the Under-sampling family we can mention the *NearMiss* [37] and *CondensedNearestNeighbor* [16] techniques which try to keep the distribution as representative as possible. For the Over-sampling family we can mention the *SMOTE* [6] technique which on the other hand generates synthetic data points based on the distance between the points in the multi-dimensional representation.
The *Feature Engineering techniques* are used because, as the dimensionality increases (number of features in the data-set), the data becomes progressively more scattered and many Clustering and Classification algorithms have difficulties when dealing with data-sets that have high dimensions. The definitions of density and distance between points becomes less significant, fundamental in algorithms such as the aforementioned K-Nearest Neighbor. This phenomenon is called *Curse of Dimensionality* and the Features Engineering techniques are used to deal with it:

- Principal Component Analysis (PCA), it is a projection method that transforms objects belonging to a p-dimensional space into a k-dimensional space (with k < p) preserving the maximum information in the initial dimensions (the information is measured as total variance of the data-set);
- Feature selection, it aims at completely discarding some features from the analysis. In particular, it is performed because some of them could be:
  - Redundant, therefore duplicate the information contained in other attributes due to a strong correlation between information;
  - Irrelevant, for instance the student's ID is often useless to predict the average of the grades.

  There are different techniques:

  - Exhaustive approach, test all possible subsets of attributes and choose the one that provides the best results on the test set using the predicted accuracy of the mining algorithm as goodness function. Given n attributes, the number of possible subsets is $2^n - 1$;
  - Non-exhaustive approaches:
    * Embedded approaches, the selection of attributes is an integral part of the Data Mining algorithm. The algorithm itself decides which attributes to use (e.g. Decision Trees);
    * Filtered approaches, the selection phase takes place before mining and with criteria independent of the algorithm used (e.g. sets of attributes are chosen whose element pairs have the lowest correlation level);
    * Heuristic approaches, approximate the exhaustive approach using heuristic search techniques.

Discretization means transformation of numerical attributes into categorical attributes, aggregating values in intervals or categories. Indispensable to use some mining techniques (e.g. Association Rules) and it can also be used to reduce the number of categories of a discrete attribute.
Discretization requires to:

- Find the most suitable number of intervals;
- Define how to choose split points.

And there are two kinds of techniques:

- Unsupervised, do not exploit the knowledge about the class to which the elements belong;
- Supervised, exploit the knowledge on the class to which the elements belong.

For the unsupervised family we can list the following techniques:

- Equi-Width, the range is divided into intervals of equal length;

- Equi-Frequency, also called Equi-Height, the range is divided into intervals with the same, or similar, number of elements;
- K-medians, k groupings are identified in order to minimize the distance between the points belonging to the same grouping.

Regarding the supervised discretization, the intervals are positioned in order to maximize their "purity". We fall into a classification problem: starting from classes, the intervals composed of a single element, contiguous classes are merged recursively. A statistical measure of purity is the entropy of the intervals. Each value v of an attribute A is a possible boundary for division into the intervals $A \leq v \wedge A > v$. We choose the value which preserves the greatest information gain, i.e. the greatest reduction in entropy. The process is applied recursively to the sub-intervals thus obtained, until a stop condition is reached, for example until the information gain obtained becomes less than a certain threshold d.
Normalization techniques involve the application of a function that maps the entire set of values of an attribute into a new set so that each value in the starting set corresponds to a single value in the arrival set. This allows the entire set of values to respect a certain property and this is necessary to compare variables with different variation intervals. For example, think of having to compare a person's age with his income. In a classification problem, where you want to predict the job of a person, an income difference of 50€ between two people means they are similar, instead, 50 years of difference lead you to consider a completely different class. We mention the two main *Normalization* techniques:

- Max-Min normalization, the A attribute is rescaled so that the new values fall between a new range: $NewMinA$ and $NewMaxA$. The new value x' is calculated from the starting value x as follow:

$$x' = \frac{x - Min_A}{Max_A - Min_A} * (NewMaxA - NewMinA) + NewMinA$$

- Z-score normalization, it changes the distribution of the A attribute so that it has mean 0 and standard deviation 1:

$$x' = \frac{x - \mu_A}{\partial_A^2}$$

μ_A is the mean of the A attribute and δ_A its standard deviation.

In conclusion, there are the *Encoding techinques*. Often some algorithm implementations are not working with categorical features. This causes an incompatibility between the data we have and the data accepted. We have to convert the categorical features into numerical: encode. There are mainly two techniques types:

- Ordinal Encoding, it transforms each categorical feature to one new feature of integers (0 to n_categories - 1);
- One-Hot Encoding, it transforms each categorical feature with n_categories possible values into n_categories binary features, with just one of them 1, and all others 0.

2.2.3 Data mining

Many learning techniques seek structural descriptions of what is learned, descriptions that can become quite complex as sets of rules or decision trees. Since these descriptions can be understood by people, they serve to explain what has been learned; in other words, to explain the basis for new predictions. There are also other techniques, such as the Artificial Neural Networks (ANN), called black box techniques, which generally perform very well but are effectively incomprehensible. Experience shows that, in many applications of Machine Learning to Data Mining, the explicit knowledge structures acquired and the structural descriptions are, at least, as important as the ability to obtain good results on new examples. People often use data mining to gain knowledge, not just predictions. Getting knowledge from the data is certainly something that enriches the result.
Below we illustrate three of the most used algorithms in the field of Data Mining. In particular, since there are many peculiarities for each one, we will only argue the basic idea on which they are based. We will explain the functioning of these particular algorithms, and not others, both for the high expressiveness of some of them and also because they have been used in this thesis project.
Decision Trees are one of the most widely used classification techniques that allow to represent a set of classification rules with a tree. A tree is a hierarchical structure consisting of a set of nodes, linked by labeled and oriented arcs. There are two types of nodes:

- Leaf nodes which identify classes;
- The others which are labeled based on the attribute that partitions the instances.

The partitioning criterion represents the label of the arcs and each root-leaf path represents a classification rule.

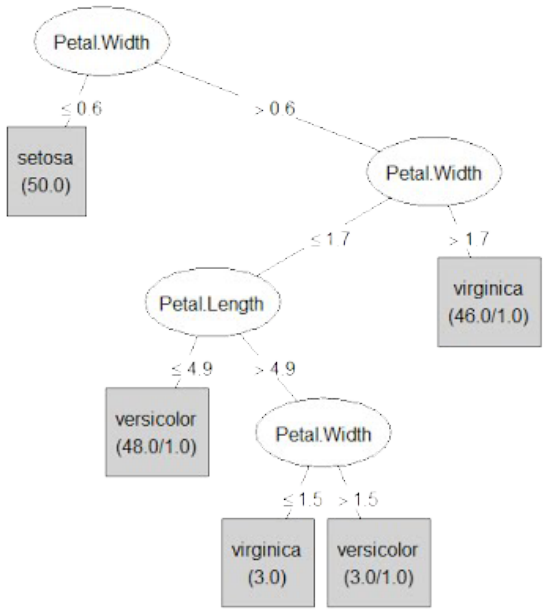


Figure 2.4: Decision Tree example on the Iris data-set, built using the well-known machine learning framework Weka.

In Figure 2.4 we see an example of a Decision Tree built on the Iris data-set. Iris is a genus of plants that contains over 300 species. In the data-set in question there are 154 instances of Iris classified according to three species: *Iris setosa*, *Iris virginica* and *Iris versicolor*. The four features considered are the length and width of the sepal and petal. We can notice that in the tree's leafs some numbers are reported. It is not a coincidence that the sum of these is the size of the data-set, indeed they represent the number of instances of the training set classified according to the leaf. When only one number is present in the leaf node, it means that all the instances in question have been correctly classified. Instead, in the case that two

numbers are present, the first number represents the instances correctly classified, the second number counts the elements classified as such but belonging to another class.

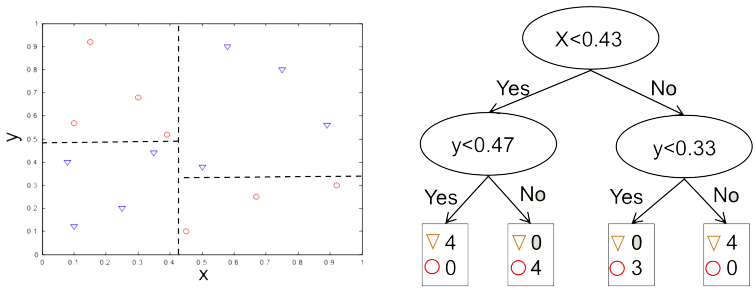


Figure 2.5: Example of how Decision Tree splits work.

Decision tree expressivity is limited to the possibility of performing search space partitions with conditions that involve only one attribute at a time: the decision boundary are parallel to the axes. In Figure 2.5 we have an example. On the other hand, Decision trees are robust to strongly correlated attributes because, in each split, they automatically assesses which attributes to divide; and if there is a correlation between two attributes, one of them will not be considered. A complex issue to address is to find the optimal split point in each attribute but a Discretization technique can be used to manage the complexity of the search.
K-Nearest Neighbor is an algorithm that, unlike the others, does not build models but classify the new records based on their similarity to the instances in the Train Set, for this reason they are called lazy learners. We can say that, in a certain way, the Train Set is the model itself.
When a new instance needs to be classified, the nearest k points (neighbors) are used to perform the classification. The basic idea is: "if it walks like a duck, quacks like a duck, then it's probably a duck".
In Figure 2.6 a graphical representation is shown.

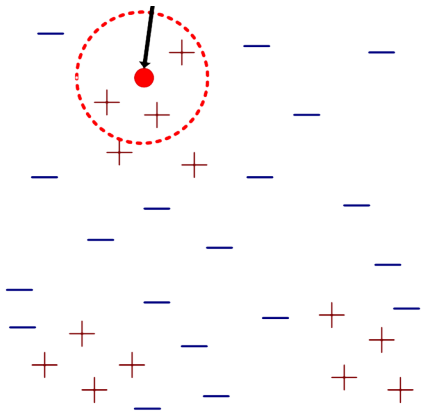


Figure 2.6: Two-dimensional representation of K-Nearest Neighbor.

They require:

- The training set;
- A metric to calculate the distance between records;
- The value of k, which is the number of neighbors to use.

The classification process involves:

- To calculate the distance to the instances in the training set
- To identify the k nearest neighbors
- To use the classes of the nearest neighbors to determine the class of the unknown instance (e.g. choosing the one that appears most frequently)

The classification of a new instance z is obtained through the majority voting process among D_z, therefore the k closest elements of the training set D:

$$y_z = argmax_{y \in \mathcal{Y}} \sum_{(x_i, y_i) \in D_z} \mathcal{I}(y_i = y)$$

where $\mathcal{Y}$ is the set of class labels and and function $\mathcal{I}$ returns 1 if its argument is TRUE, 0 otherwise.

To operate correctly, the attributes must have the same scale of values and must therefore be normalized during the Pre-processing phase. For example a difference of 0.5 is more significant on an attribute in which the range varies from 1.5 to 2.5 rather than on an attribute that varies from 50 to 150. Moreover, they are very sensitive to the presence of irrelevant or related attributes that will distort the distances between objects. For example if in a data set of commercial products, in addition to standard attributes such as product category, year of production and so on, we have the attributes "price without taxes" and "price with taxes", as high values of one correspond to high values of the other, the price will have more importance than normal. A Pre-processing step, such as feature selection or PCA, solves the problem.

Naive Bayes classifiers represent a probabilistic approach by modeling probabilistic relationships between attributes and the class.

The conditional probability $P(A|C)$ is the probability that the event A occurs knowing that the event C has occurred. $P(A, C)$ is the joint probability of the two events A and C, therefore the probability that both occur and it is defined as the conditional probability $P(A|C)$, multiplied by the probability that C actually occurs:

$$P(A, C) = P(A|C)P(C) = P(C|A)P(A)$$

Consequently we can define:

$$P(C|A) = \frac{P(A, C)}{P(A)}$$
$$P(A|C) = \frac{P(A, C)}{P(C)}$$

and with a simple mathematical equation we can reach the Bayes theorem:

$$P(C|A) = \frac{P(A,C)}{P(A)} = \frac{P(A|C)P(C)}{P(A)}$$
$$P(A|C) = \frac{P(A,C)}{P(C)} = \frac{P(C|A)P(A)}{P(C)}$$

Let the vector $A = (A_1, A_2, ..., A_n)$ describe the set of attributes and let C be the class variable. If C is linked in a non-deterministic way to the values assumed by A we can treat the two variables as random variables and capture their probabilistic relationships using $P(C|A)$:

- Before the training phase, the probabilities $P(C)$ and $P(A)$ are calculated through the training set;
- During the training phase, the probabilities $P(C|A)$, called also *a priori probabilities*, are learned for each combination of values assumed by A and C;
- Knowing these probabilities and applying the Bayes theorem, for a test record with certain attributes values a, we calculate for each class c, the *posterior probability* $P(c|a)$. Then, we classify the test instance with the class which maximizes that posterior probability.

The main advantage of probabilistic reasoning over logical reasoning lies in the possibility of reaching rational descriptions even when there is not enough deterministic information on the functioning of the system. Since it is based on the calculation of the probabilities they are very robust to: irrelevant features and noise:

- The noise is canceled during the calculation of $P(A|C)$;
- If A is an irrelevant attribute, $P(A|C)$ is uniformly distributed with respect to the values of C and therefore its contribution is also irrelevant.

A drawback could be that related attributes can reduce effectiveness since the assumption of conditional independence does not apply to them. However, we already know that this problem can be resolved through feature selection or PCA techniques.

We are overwhelmed with data. The amount of data in the world, in our lives, seems ever-increasing and there is no end in sight. In fact, it has been reported that 2.5 quintillion bytes of data is being created everyday and the 90% of stored data in the world, has been generated in the past two years only [27]. Specifically, human and machine-generated data is experiencing an overall 10x faster growth rate than traditional business data, and machine data is increasing even more rapidly at 50x that growth rate. In Figure 2.7 the reported data from [28].

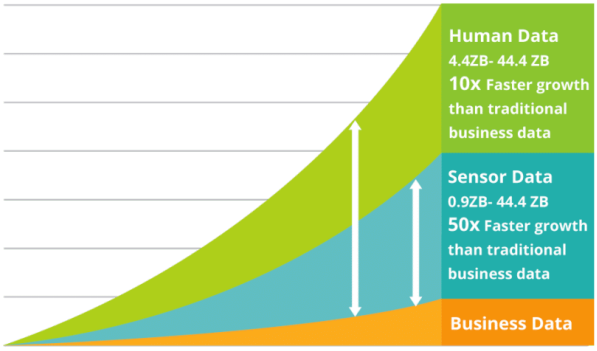


Figure 2.7: Trend of the growth of human and machine-generated data, from the article *IoT, Big Data and AI - the new "superpowers" in the digital universe* published in Forbes [28].

By business data we mean all the data generated in the traditional way so far by companies, such as interviews, questionnaires and market surveys. However over the years the advancement of technology has led to a change in the market. With human-generated data we means UGC, short for User-Generated Content, the term which describes any form of content such as video, blogs, discussion form posts, digital images, audio files, and other forms of media that is created by consumers or end-users of an online system or service. It is becoming an important part of content marketing, with consumers forming a part of a brand's strategy. It is powerful because it builds connections between like-minded people, whether the opportunity is to share a common experience or win a prize.

We instead refer to pervasive computing, or ubiquitous computing, the growing trend of embedding computational capability into everyday objects to make them effectively communicate and perform useful tasks in a way that minimizes the end user's need to interact with computers as computers. Pervasive computing devices are network-connected and constantly available. Such devices, equipped with sensors, measure a substantial amount of data, called machine-generated data.
It is precisely these phenomena, UGC and pervasive computing, that led to a dizzying increase in human and machine-generated data and led us to the era of data ubiquity: data is central to all of our existences, whether we're a giant enterprise or an individual person.
As a consequence of this unceasing growth of data, the need to organize and exploit the stored data has also increased dramatically. Above all this need is coming from companies, to make more conscientious business decisions. Essentially, it can be said that the Data Scientist is the expert that the companies are looking for, capable of extrapolating insights and analyzes. His figure must have heterogeneous skills, ranging from technology to knowledge of the market and business, up to the ability to use Machine Learning techniques and programming languages:

- Math & Statistic skills which comprises Machine Learning, Statistical Modeling, Data Science fundamentals, etc.;
- Programming & Database skills which comprises of Computer Science fundamentals, scripting and statistical languages, Databases, Big Data tools, etc.;
- Domain Knowledge & Soft skills which comprises of being passionate about the business, curios about data, strategic, proactive, creative and so on;
- Communication & Visualizazion skills which comprises of being able to translate data-driven insights into decisions and actions, know how to use visualization tools, be able to engage with senior management, etc.

However, although this exponential growth in data has led to the new business position of Data Scientist and with it to new opportunities, it has been reported that the demand is far greater than the supply. Data Scientists cannot scale: it is almost impossible to balance the

number of qualified experts of this field and the required effort to analyze the increasingly growing sizes of available data[14]. This gap has led to more and more non-expert users to approach this world and carry out data analysis, using Data Mining techniques. The process itself, known as Knowledge Discovery in Databases (KDD), consists of several steps, and users are overwhelmed by the amount of Machine Learning algorithms and Pre-processing techniques. These users require off-the-shelf solutions that will assist them throughout the whole process.

Indeed, the problems are related to these two main KDD phases. Regarding the Modeling phase, the difficulty is building a high-quality Machine Learning model; the process to achieve this result is iterative, complex and time-consuming. The Data Scientist needs to select among a wide range of possible algorithms (e.g. Decision Trees, K-Nearest-Neighbor, Naive Bayes, etc.) and to tune numerous *hyper-parameters* of the selected algorithm. An algorithm hyper-parameter is a parameter that the Machine Learning algorithm cannot learn by itself and, hence, it must be set a priori. For instance, common parameters in the Decision Tree are the attributes splits chosen by the algorithm; instead, a hyper-parameter would be the number of instances reacquired in a leaf to be split again. Regarding the Pre-processing step, the issue is to find the techniques to use. To automate this process we have to take in consideration more variables:

- There are several different transformations, e.g. Imputation, Re-balancing, Features Engineering, Discretization, Normalization, Encoding;
- For each transformation we have several operators, e.g. Normalization Min-Max, Normalization Z-score, etc.;
- For each operator we have different parameters;
- The order in which they are applied affects the result.

Moreover, the problem is more complicated because the transformations application depends on both the chosen algorithm and the data-set itself. A sequence of Pre-processing transformations, with related operators and parameters, is called data pipeline. Instead a Machine Learning pipeline (ML pipeline) consists of a data pipeline and a Machine Learning algorithm with its hyper-parameters defined. The techniqueof automatically configuring ML

pipelines is called AutoML.
Unfortunately, existing solutions either do not recommend Pre-processing operators or they recommend them in a very poor way, not giving too much importance to this step. Indeed, it has been noticed that the automation of the Modeling problem is performed in 16 out of 19 selected publications while only 2 publications study the automation of Data Pre-processing[7]. This fact represent an issue because generally raw data do not perform very well. Indeed, pervasive computing systems greatly increased the amount of machine-generated data and, since we are talking about data generated by sensors, we are rarely having data ready to be consumed. All in all, the Pre-processing phase plays a key role in Machine Learning and it is require a tool which treats all the problems (Modeling and Pre-processing ones) in all its complexity.

2.4 The state-of-the-art solutions

In this section, we provide an overview of several tools and frameworks that have been implemented to automate Modeling and Pre-processing problems. In general, they can be classified into three main categories: distributed, cloud-based and centralized. In order to be able to work with large quantities of data, distributed and cloud-based distributions use clusters and therefore different techniques than centralized solutions. A cluster is a set of machines that work together so that, in many respects, they can be viewed as a single system. For data-sets with not exaggerated quantities of instances, the overhead of using the cluster is not worthwhile and centralized solutions are preferred.

2.4.1 Distributed tools

MLbase[22,35] has been the first work to introduce the idea of developing a distributed environment for Machine Learning algorithm selection and hyper-parameter optimization. It is based on Apache Spark, an open source framework for distributed computing and a unified analytics engine for big data processing. In particular MLlib is the Spark library which allows to use the Machine Learning algorithm in distributed manner. MLBase exploit Spark's functionalities and MLlib, it consists of three components (Figure 2.8):

- ML Optimizer, this layer aims to automating the task of ML pipeline construction. The optimizer solves a search problem over feature engineering and ML algorithms included in MLI and MLlib. The ML Optimizer is currently under active development;
- MLI[34], an experimental API for feature engineering and algorithm development that introduces high-level ML programming abstractions. A prototype of MLI has been implemented against Spark, and serves as a testbed for MLlib;
- Apache Spark's distributed ML library. MLlib was initially developed as part of the MLbase project, and the library is currently supported by the Spark community. Many features in MLlib have been borrowed from ML Optimizer and MLI, e.g. the model and algorithm APIs, multimodel training, sparse data support, design of local / distributed matrices, etc.

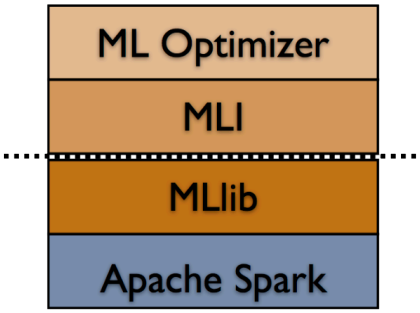


Figure 2.8: MLBase infrastructure. From the official MLBase documentation.

TransmogrifAI[1] is a really recent tool written in Scala. Currently, TransmogrifAI supports eight different binary classifiers, five regression algorithms and it expects a minimal human involvement.
MLBox is a Python-based AutoML framework covering several processes, including Preprocessing, optimization and prediction. It supports *model stacking* where a new model is

trained from combined predictors of multiple previously trained models and it uses hyperopt, a distributed asynchronous hyper-parameter optimization library to perform the hyper-parameter optimization process.

2.4.2 Cloud-based tools

Google Cloud AutoML is a suite of Machine Learning products that allows developers with limited experience in the field of Machine Learning to train high-quality models based on business needs. It is based on Google's cutting-edge Transfer Learning (TL)[36] and Neural Architecture Search (NAS)[12] technologies. Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. Over the years, Google has memorized a lot of data about different problems which allows to have really valid recommendations. Instead, Neural Architecture Search is a technique for automating the design of Artificial Neural Networks.
Google Cloud AutoML build models in different domains and for various tasks:

- AutoML Vision and AutoML Video Intelligence allow respectively to get insights from image and perform content detection;
- AutoML Natural Language and AutoML Translation detect the structure and meaning of the text and translate dynamically from one language to another;
- AutoML Tables, automatically develops and deploys the latest generation machine learning models on structured data.

Focusing on AutoML Tables, it performs automatically both model building and some data Pre-processing. The data Pre-processing that AutoML Tables does includes:

- Normalization and discretization of numeric features;
- Application of one-hot encoding for categorical features;
- Performing basic processing for text features;
- Extraction of date and time-related features from Timestamp columns.

Moreover, missing values are handled according to the type of the features:

- Numerical, a 0.0 or −1.0 is imputed;
- Categorical, an empty string is imputed;
- Text, an empty string is imputed;
- Timestamp, a timestamp set to −1 is imputed.

Then, when the model training kicks off, AutoML Tables takes the data-set and starts training for multiple model architectures at the same time. This approach enables AutoML Tables to determine quickly the best model architecture, without having to serially iterate over the many possible model architectures. AutoML Tables tests includes:

- Linear Regression, one of the most simple but effective regression algorithm;
- Feedforward Deep Neural Network, a kind of Artificial Neural Networks which in these last years allows to solve a wide range of problems;
- Gradient Boosted Decision Tree, classifier which combines several different Decision Trees in order to take the best of everyone;
- AdaNet, a particular algorithm which learn the structure of a neural network as an ensemble of subnetworks;
- Ensembles of various model architectures, like in Gradient Boosted Decision Tree and AdaNet, the goal is to define a combined learner that performs better than a basic one.

Azure AutoML uses *collaborative filtering* to search for the most promising pipelines efficiently based on a database that is constructed by running millions of experiments of evaluation of different pipelines on many data-sets. With collaborative filter we refer to a class of tools and mechanisms that allow the retrieval of predictive information regarding the interests of a given set of users. Collaborative filtering is widely used in recommendation systems, in this case what is recommended is the ML pipeline.
Amazon Sage Maker provides its users with a wide set of most popular Machine Learning,

and Deep Learning frameworks to build their models in addition to automatic tuning for the model parameters. Moreover, Amazon offers a long list of pre-trained models for different AI services that can be easily integrated to user applications, such as image and video analysis, voice recognition, text analytics, forecasting and recommendation systems.

2.4.3 Centralized tools

Several tools have been implemented on top of widely used centralized machine learning packages which are designed to run in a single node (machine). In general, these tools are suitable for handling small and medium sized data-sets. Since the problem to address is a optimization problem, find a maximum or minimum of a unknown function, various are the techniques used in this category. However we list only two because they are the most used:

- Genetic algorithms, which are inspired by the branch of genetics, and allow to evaluate different starting evaluations (as if they were different biological individuals) and, by recombining them (analogous to sexual biological reproduction) and introducing elements of disorder (analogous to random genetic mutations), they produce new solutions (new individuals) that are evaluated by choosing the best (environmental selection) in an attempt to converge towards "excellent" solutions.

- Bayesian optimization techniques, which are based on Bayes Theorem, start from a bunch of random evaluations and try to approximate the objective function through regression techniques and a probabilistic model. There are three different implementation of it:

  - Using Gaussian Procces (GP);
  - Tree-structured Parzen Estimators (TPE);
  - Sequential Model-based Algorithm Configuration (SMAC).

  As this approach has captured more attention recently, a more comprehensive explanation will be provided in the next chapter.

Auto-Weka[21] is considered as the first and pioneer Machine Learning automation framework[21]. It has been implemented in Java on top of Weka, a popular Machine Learning

library that has a wide range of machine learning algorithms. Auto-Weka applies Bayesian optimization using SMAC and TPE implementations for both algorithm selection and hyper-parameter optimization (Auto-Weka uses SMAC as its default optimization algorithm but the user can configure the tool to use TPE). No Pre-processing recommendation is done. Auto-Sklearn [13] has been implemented, instead, on top of the competitor framework Scikit-Learn, a Python package. Auto-Sklearn used SMAC as a Bayesian optimization technique too but in order to improve the quality of the result introduced a meta-learer. Indeed, the outcome of these techniques of optimization depends a lot on the start evaluations. The idea is to retrieve data-sets similar to the one in input and feed SMAC with the solutions of those similar data-sets. The component in charge of this task is called meta-learner since it is built through a Machine Learning algorithm and the similarity through data-sets is computed thanks to the meta-data (data that describes data). In addition, ensemble methods were also used to improve the performance of output models. With ensemble methods we refer to a set of techniques which aim to improve the quality of the result by combining the results of the best learner found. In Figure 2.9 we can see the infrastructure just described.

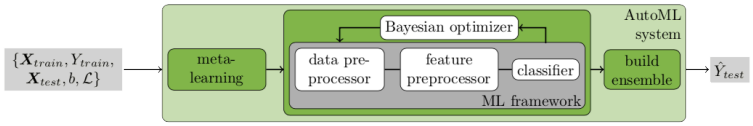


Figure 2.9: Auto-Sklearn infrastructure, from *Efficient and Robust Automated Machine Learning* published in NIPS 2015 [13].

Unlike the previous tool, this one allows, although poor, some Pre-processing. The included transformations are as follows, in the following fixed order:

- Encoding (just one operator);
- Imputation (just one operator);
- Normalization (just one operator);

- Balancing (just one operator);
- Features Pre-processing (thirteen operators).

The Modeling step follows with a choice of fifteen algorithms. In Figure 2.10 the Pre-processing operators and Machine Learning algorithms used in [13] are listed.

name	#λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)
Bernoulli naïve Bayes	2	1 (-)	1 (-)
decision tree (DT)	4	1 (-)	3 (-)
extreml. rand. trees	5	2 (-)	3 (-)
Gaussian naïve Bayes	-	-	-
gradient boosting (GB)	6	-	6 (-)
kNN	3	2 (-)	1 (-)
LDA	4	1 (-)	3 (1)
linear SVM	4	2 (-)	2 (-)
kernel SVM	7	2 (-)	5 (2)
multinomial naïve Bayes	2	1 (-)	1 (-)
passive aggressive	3	1 (-)	2 (-)
QDA	2	-	2 (-)
random forest (RF)	5	2 (-)	3 (-)
Linear Class. (SGD)	10	4 (-)	6 (3)

(a) classification algorithms

name	#λ	cat (cond)	cont (cond)
extreml. rand. trees prepr.	5	2 (-)	3 (-)
fast ICA	4	3 (-)	1 (1)
feature agglomeration	4	3 ()	1 (-)
kernel PCA	5	1 (-)	4 (3)
rand. kitchen sinks	2	-	2 (-)
linear SVM prepr.	3	1 (-)	2 (-)
no preprocessing	-	-	-
nystroem sampler	5	1 (-)	4 (3)
PCA	2	1 (-)	1 (-)
polynomial	3	2 (-)	1 (-)
random trees embed.	4	-	4 (-)
select percentile	2	1 (-)	1 (-)
select rates	3	2 (-)	1 (-)
one-hot encoding	2	1 (-)	1 (1)
imputation	1	1 (-)	-
balancing	1	1 (-)	-
rescaling	1	1 (-)	-

(b) preprocessing methods

Figure 2.10: Auto-Sklearn Pre-processing operators and machine learning algorithms, from *Efficient and Robust Automated Machine Learning* published in NIPS 2015 [13].

The work of A. Quemy in [32] focuses on applying the Bayesian techniques just to the Pre-processing pipeline, ignoring the Modeling phase, but extending the research space of Data Preparation, increasing the number of operations of each transformation. Below an instance of the data pipeline (Figure 2.11) and the considered research space (Figure 2.12) is shown.

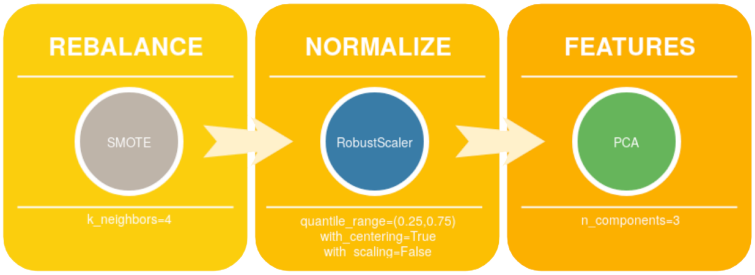
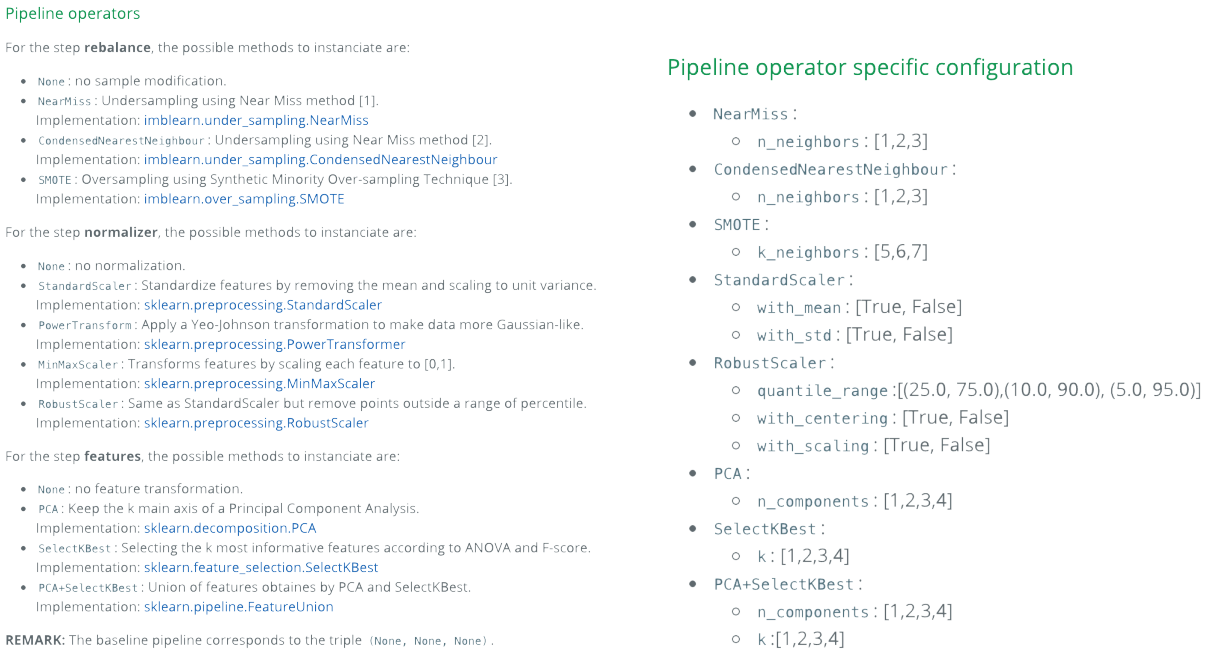


Figure 2.11: A Quemy's pipeline instance, from *Data Pipeline Selection and Optimisation* published in DOLAP 2019 [32].



Pipeline operators

For the step **rebalance**, the possible methods to instanciate are:

- `None` : no sample modification.
- `NearMiss` : Undersampling using Near Miss method [1].
  Implementation: imblearn.under_sampling.NearMiss
- `CondensedNearestNeighbour` : Undersampling using Near Miss method [2].
  Implementation: imblearn.under_sampling.CondensedNearestNeighbour
- `SMOTE` : Oversampling using Synthetic Minority Over-sampling Technique [3].
  Implementation: imblearn.over_sampling.SMOTE

For the step **normalizer**, the possible methods to instanciate are:

- `None` : no normalization.
- `StandardScaler` : Standardize features by removing the mean and scaling to unit variance.
  Implementation: sklearn.preprocessing.StandardScaler
- `PowerTransform` : Apply a Yeo-Johnson transformation to make data more Gaussian-like.
  Implementation: sklearn.preprocessing.PowerTransformer
- `MinMaxScaler` : Transforms features by scaling each feature to [0,1].
  Implementation: sklearn.preprocessing.MinMaxScaler
- `RobustScaler` : Same as StandardScaler but remove points outside a range of percentile.
  Implementation: sklearn.preprocessing.RobustScaler

For the step **features**, the possible methods to instanciate are:

- `None` : no feature transformation.
- `PCA` : Keep the k main axis of a Principal Component Analysis.
  Implementation: sklearn.decomposition.PCA
- `SelectKBest` : Selecting the k most informative features according to ANOVA and F-score.
  Implementation: sklearn.feature_selection.SelectKBest
- `PCA+SelectKBest` : Union of features obtaines by PCA and SelectKBest.
  Implementation: sklearn.pipeline.FeatureUnion

REMARK: The baseline pipeline corresponds to the triple `(None, None, None)`.

Pipeline operator specific configuration

- `NearMiss` :
  - `n_neighbors` : [1,2,3]
- `CondensedNearestNeighbour` :
  - `n_neighbors` : [1,2,3]
- `SMOTE` :
  - `k_neighbors` : [5,6,7]
- `StandardScaler` :
  - `with_mean` : [True, False]
  - `with_std` : [True, False]
- `RobustScaler` :
  - `quantile_range` :[(25.0, 75.0),(10.0, 90.0), (5.0, 95.0)]
  - `with_centering` : [True, False]
  - `with_scaling` : [True, False]
- `PCA` :
  - `n_components` : [1,2,3,4]
- `SelectKBest` :
  - `k` : [1,2,3,4]
- `PCA+SelectKBest` :
  - `n_components` : [1,2,3,4]
  - `k` :[1,2,3,4]

Figure 2.12: Quemy's research space, from *Data Pipeline Selection and Optimisation* published in DOLAP 2019 [32].

However, a problem that persists in the Quemy's solution is the fixed order of transformations.

TPOT [30] framework represents another type of solutions that has been implemented on top of Scikit-Learn. It is based on genetic programming and explore many different possible pipelines of feature engineering and learning algorithms. Then, it finds the best one out of

them. The TPOT approach is valid but it is expensive, in fact, Genetic Programming (GP) optimization methods are typically criticized for optimizing a large population of solutions, which can sometimes be slow and wasteful for certain optimization problems. Recipe[8] follows the same optimization procedure as TPOT using genetic programming but it adds a grammar, that avoids the generation of invalid pipelines, and speeds up the optimization process. Second, it works with a bigger search space of different model configurations than Auto-SkLearn and TPOT.

Recipe seems the solutions of all the problems if it were not for that the pipelines produced are customized for the data at hand. The non-expert users have generally no knowledge about the right transformations order.

3

Bayesian techniques for AutoML

In this chapter we analyze in details the Bayesian techniques, state-of-the-art regarding the optimization field. In particular we focus on their incarnation, the Sequential Model-Based Optimization (SMBO) algorithm, which allows to find valuable solutions in a very wide search space. We discuss different implementations focusing on their pros and cons.
Afterwards, we introduce the reader to a formal definition of the two problems faced in a typical Machine Learning flow. Only then, we can describe how these techniques are exploited by AutoML tools.

In an optimization problem we are searching for the best solution among a set of feasible solutions. It can be simply formalized as follow:

$$\max_{x \in A} f(x)$$

A contains all the feasible solutions, or candidates, and it is typically d-dimensional ($A \subseteq \mathbb{R}^d$). A specific solution $x \in A$ is evaluated through the function $f : \mathbb{R}^d \longrightarrow \mathbb{R}$, also called the objective. In general, in this kind of problems, f has no special structure like concavity or linearity that would make the optimization easier. In fact, we consider it as a "black-box" function, that we do not know how it works; we only know that it maps certain inputs, $x \in A$, to certain outputs, $f(x) \in \mathbb{R}$. The aim consists of finding x that maximizes $f(x)$.
The naive approach to the problem would be to systematically evaluate all possible candidates and choose the x which lead to the highest value of $f(x)$. Since this method effectively evaluates all the solutions and always find the best one, it is called Exhaustive Search. The drawback of this approach is that, generally, it cannot be applied in real-cases problems. Indeed, the difficulties encountered are: a large number of candidates, which are part of a high-dimensional space, and a really time-consuming evaluation function f. The result is that not all candidates can be evaluated and we have to find a way to choose the most promising ones. Bayesian techniques are part of the family of "surrogate methods", which create in fact a surrogate model to approximate the objective and, based on this, choose the evaluations to perform. In contrast to the other methods, Bayesian techniques build the surrogate model through Bayesian statistics and choose where to evaluate the objective function using a Bayesian interpretation of that surrogate. Specifically, it starts by evaluating the objective function on a configuration of initial points, then the process becomes iterative: the surrogate model is constructed on the basis of the made observations and through an acquisition function, therefore the Bayesian interpretation of the surrogate, the candidate for the next observation is decided. The process ends when a termination condition is reached, generally expressed in terms of the number of observations to be done or in terms of time. Given its iterative nature and the fundamental role of the model, this algorithm is called Sequential Model-Based Optimiza-

tion (SMBO). In Algorithm 1 we provide a pseudo-code of it, the budget is expressend in terms of number of observations.

Algorithm 1: Sequential Model-Based Optimization

Observe f at n_0 points according to an initial space-filling experimental design
Set $n = n_0$
while $n < N$ **do**
    Create\Update the model using all available data
    Find x_n, the maximizer of the acquisition function
    Observe $y_n = f(x_n)$
    Increment n
end
Return the solution: the point evaluated with the highest $f(x)$

In Figure 3.1, we provide an example run with a 1-dimensional continuous input.

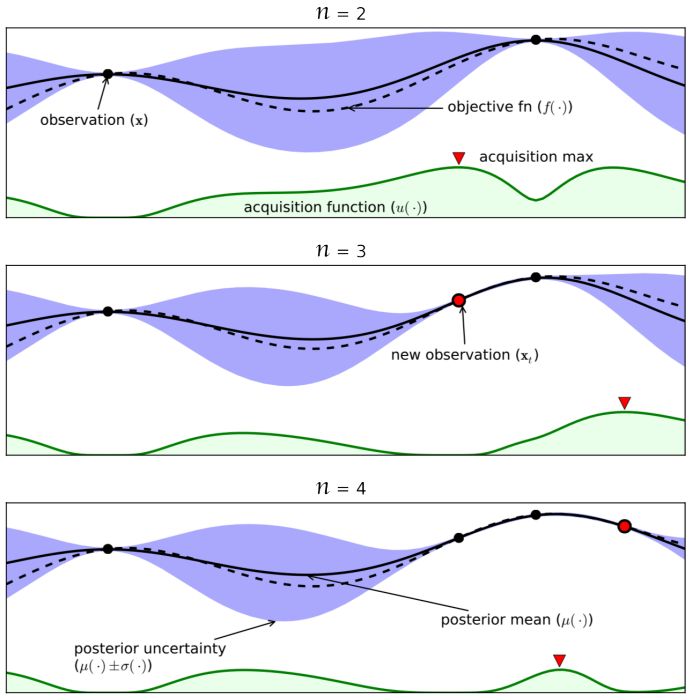


Figure 3.1: SMBO algorithm example which shows the working, through some iterations. From A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, published in ArXiv 2010 [3].

In that case the objective function is represented by the black dotted line, while the already measured observations by the black circles. The surrogate model is composed of two different elements:

- The posterior mean $\mu(x)$, represented by the black solid line, which tries to estimate the objective by interpolating the current available observations;
- The posterior uncertainty $\sigma(x)$, represented by the shaded purple area, which is a confidence interval containing $f(x)$ with probability of 95%;

It can be observed that the uncertainty in the observations already made is zero, since those values are actually measured and there is no doubt. Furthermore, as the iterations progress, the whole model converges towards the objective function.
The bottom green distribution represents the acquisition function calculated according to the surrogate. The point which maximizes that function (red triangles in figure) is the most promising candidate and, hence, the one that is chosen to be the new observation in the next iteration (red circles in figure). The acquisition function tends to be high for high posterior mean values (Exploitation) and for high posterior uncertainty values (Exploration). The explanation of this behavior lies in wanting to maximize $f(x)$, looking for high value of its estimation $\mu(x)$, and in the meanwhile to explore new promising areas, considering high uncertainty. An example of acquisition function is the Expected Improvement (EI):

$$EI_n(x) = E_n[[f(x) - f_n^*]^+]$$

where

$$a^+ = max(a, 0)$$

Let x be a new candidate, $f(x)$ is its evaluation through the objective, and, f_n^* is the highest evaluation so far. Merely, we have as much improvement as the difference between this two factors $[f(x) - f_n^*]^+$ is greater, but $f(x)$ is unknown until after the evaluation. For this reason we use E_n, the $f(x)$ expectation according to the surrogate, built with the already done evaluations $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$.

The most substantial difference between the several implementations is the methodology used to build the model. We give, below, a brief overview of them.

3.1.1 Gaussian Processes (GP) regression

In Bayesian statistics, whenever we have an unknown phenomenon, we suppose that it is coming, by nature, from some random prior probability distribution. GP regression takes this prior distribution to be as a multivariate normal. To interpolate the already done evaluations, $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$, various functions, passing through these points, are generated. Each of them represent a belief according to the problem constraints (the observations) and what we can do, as it is shown in Figure 3.2, is compute the mean μ and the variance σ^2.

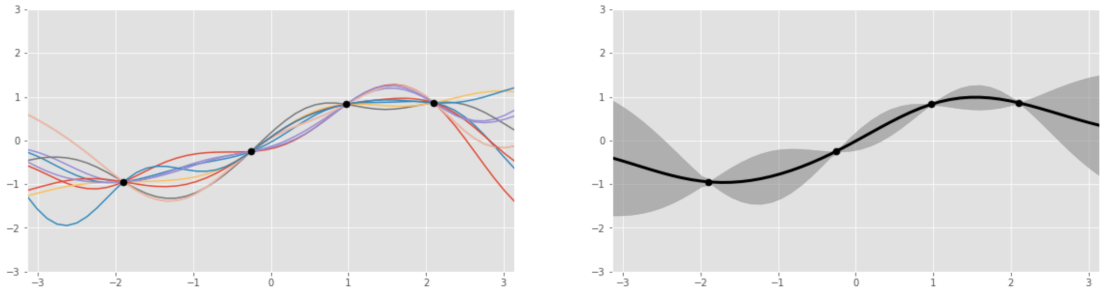


Figure 3.2: An example of Gaussian Processes (G) regression. From *An intuitive guide to Gaussian processes* in Medium - Towards Data Science [19].

The random functions are drawn specifying a covariance function or kernel Σ_0. The kernel is chosen in a way that, two points x_i, x_j, close in the input space, have a large positive correlation. This would encode the belief that they should have more similar function values than points that are far apart.

3.1.2 Tree-structured Parzen Estimator (TPE) approach

The name of this approach is given by its two principal characteristics:

- Tree-structured, because this algorithm manages the d dimensions of $x \in A \subseteq \mathbb{R}^d$ in a tree structure. Indeed, unlike what we have for simplicity seen so far, there are optimization problems that involve more than one dimension. In these cases, it may happen that some dimensions have dependencies to each other and, in literature, the tree structure is extremely suitable to model this kind of constraints. A further advantage, derived by using this structure, is that it supports not only continuous dimensions, but also categorical ones;

- Parzen Estimator, because to build the surrogate it uses a density estimation technique called Parzen estimator.

In particular, while the other approaches are estimating the objective through a surrogate that can be seen as $p(y|x)$, where y is the presumed value of the objective in a certain point and x is the observations done so far, a tree of Parzen estimators models the surrogate through $p(x|y)$ and $p(y)$. The idea is to learn these two probability distribution applying the density estimation technique in question on the available observations. Once we have estimated them we can derive $p(y|x)$ applying the Bayesian's rule.

3.1.3 Sequential Model-based Algorithm Configuration (SMAC)

Another way to build the desired model is run an ensemble of Regression Trees. In the previous chapter of this thesis, 2.2.3 Data Mining, we treated the Decision Tree algorithm but only with regard to the Classification problem. Fortunately the approach does not change, indeed as we already know, what changes in the Regression problem is just the nature of the result, that would be continuous instead of categorical. We can estimate the objective through a Regression Tree, the dimensions of the observations would be the features of our data-set and the functioning of the Decision Tree remains unchanged. In Figure 3.3 we have an example with just one continuous dimension.

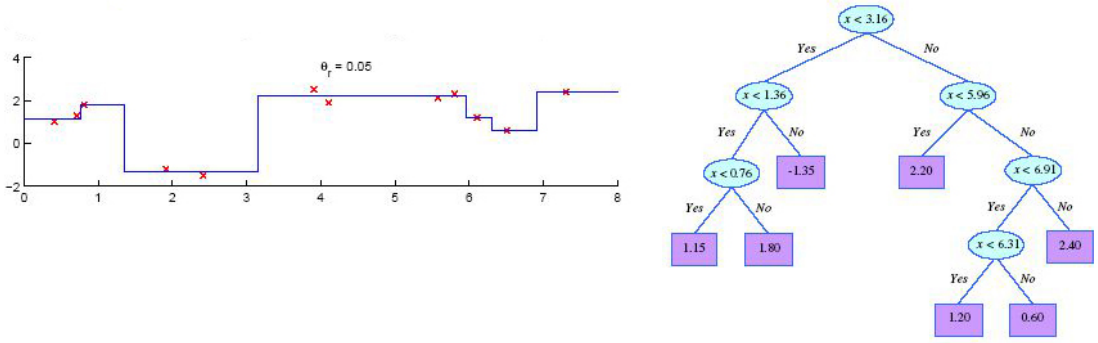


Figure 3.3: Example of a Regression Decision Tree.

As we mentioned in the state-of-the-art section (2.4.3), an ensemble method is a way to combine different results from different learners in order to improve the quality of the final outcome. Several are the ensemble techniques available but, using the Decision Trees, the most known one is Random Forest. It operates by constructing a multitude of trees at training time and outputting the class that is the mode of the classes (Classification) or the mean of the predictions (Regression) of the individual trees (Figure 3.4).

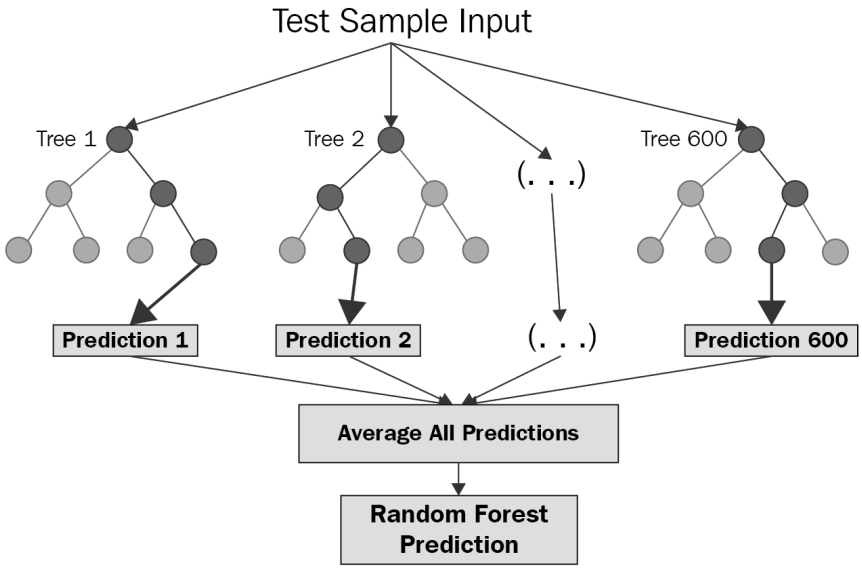


Figure 3.4: Example of Random Forest, from *Random Forest Regression* published in Towards Data Science [5].

Since the Decision Tree algorithm is the same for each of the built trees, to have a different result from each one of them, they are fed by different samples of training set and are able to use a limited number of all the features.
The Sequential Model-based Algorithm Configuration (SMAC) implementation uses Random Forest as regressor to build the model and, for this reason, allows to work with:

- Conditional dependencies between the dimensions;
- Categorical dimensions;
- High-dimensional data;
- Structured data.

3.2 CASH and DPSO problems

As the definition suggests, AutoML aims to automate the Machine Learning process. According to what we have said in 2.3 Automated Machine Learning, the two phases that we focus on are: Data Pre-processing and Modeling. In fact, a Machine Learning pipeline is composed by:

- A data pipeline, related to the Pre-processing phase;
- An algorithm configuration, related to the Modeling phase.

Regarding the Modeling phase we have a pool of algorithms and the aim is to select the one which performs best. Besides, for the chosen algorithm we have to provide its hyperparameter configuration.
Instead, a data pipeline consists of a succession of transformations in a specific order. It is important to choose the right transformations and concatenate them in the right order because both of these aspects affect the result. Furthermore, we have to choose an operator for each transformation and, for each operator, we need to find the parameters configuration (the ones that perform best).
Over time, these problems have been formalized as follows:

- Data Pipeline Selection and Optimization (DPSO), which refers to the problems contained in the Pre-processing phase;
- Combined Algorithm Selection and Hyper-parameter optimization (CASH), which refers to the problems contained in the Modeling phase.

A scheme is provided in Figure 3.5.

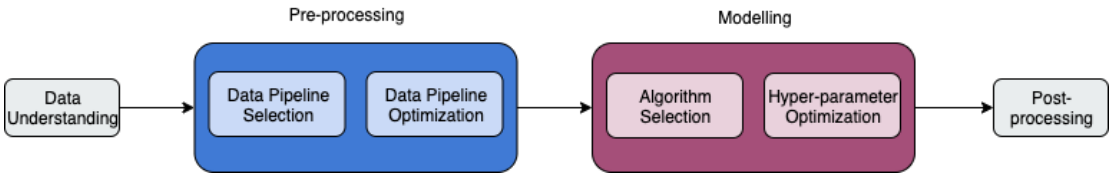


Figure 3.5: Machine Learning problems scheme.

CASH has been formalized in Auto-WEKA: Combined Selection and Hyper-parameter Optimization of Classification Algorithms[21].
Given:

- A data-set D divided into D_{train}, D_{test};
- A set of algorithms $\mathcal{A} = \{A^1, \ldots, A^k\}$ with associated hyper-parameter spaces $\Lambda^1, \ldots, \Lambda^k$;
- And a loss function $\mathcal{L}(A^i_\lambda, D_{train}, D_{test})$;

we are searching for:

$$A^*_{\lambda^*} \in \text{argmin}_{A^i \in \mathcal{A}, \lambda \in \Lambda^i} \mathcal{L}(A^i_\lambda, D_{train}, D_{validation}) \qquad \text{(CASH)}$$

The data-set D is divided into D_{train} and D_{test}, respectively to build and to evaluate the overall performances, calculated through the loss function $\mathcal{L}$. This latter is nothing more than one of the metric explained in 2.1, Machine Learning and Data Science. Since the problem is formalized as a minimization problem, we are trying to minimize the error, e.g Misclassification

Error, but it can be turned as a maximization problem by replacing the loss function with a metric like Accuracy, Balanced accuracy, etc. Regardless of the problem we are setting, the problem itself is set up as an optimization problem and, as such, it assumes we know the configuration space we are searching in (the set of algorithms $A^1, \ldots, A^k$ and the related hyper-parameter spaces $\Lambda^1, \ldots, \Lambda^k$).
Therefore, with the above formula, we aim to find the best algorithm A^* in the set of algorithms and its best hyper-parameters λ^* in the related hyper-parameter space. DPSO has been formalized by A. Quemy in Data Pipeline Selection and Optimization [32]. In this case, since we have a pipeline, the order of the transformation is an extra issue.
In order to formalize also this problem as an optimization one, it has been simplified: it requires the prototype of the data pipeline we are going to build. A data pipeline prototype is defined as a concatenation of transformations. In the prototype, for each transformation, we do not specify which operator we are going to use, but, we decide the steps of the Pre-processing pipeline (the transformation's order). However, as in the previous problem, for each transformation we have to know the available operators and their parameter's search space. Solving the optimization problem means finding the right configuration for each transformation (optimal operator and optimal parameters values), not caring about the order. Anyway, to support different order combinations, despite few, Quemy includes the *None* operator in each transformation. In that way, a transformation could be not present.
Thereafter, given:

- A data-set D divided into D_{train}, D_{test};
- A data pipeline prototype P with a configuration space $\mathcal{P}$;
- The algorithm A, for which the given pipeline P transforms the data;
- And a loss function $\mathcal{L}(P, A, D_{train}, D_{test})$;

we are searching for:

$$P^* \in \operatorname{argmin}_{P \in \mathcal{P}} \mathcal{L}(P, A, D_{test}) \qquad \text{(DPSO)}$$

3.2.1 SMBO as a CASH resolution

As we mentioned SMBO provides a methodology to solve optimization problems, so CASH can be solved through it. To do that, CASH has to be reformulated as a single combined hierarchical hyper-parameter optimization problem. We define $\Lambda = \Lambda^1 \cup \cdots \cup \Lambda^k \cup \{\lambda_r\}$, where λ_r is the root-level hyper-parameter that selects between algorithms $\Lambda^1, \ldots, \Lambda^k$. The hyper-parameters of each subspace Λ^i are made conditional on λ_r being instantiated to A_i. For instance, considering just two algorithms, Decision Tree and K-Nearest Neighbor, and just two hyper-parameters for each, the space looks like the following:

DecisioneTree.num_obj = [2, 3]
DecisioneTree.pruning = [True, False]

K-NearestNeighbor.k = [3, 4]
K-NearestNeighbor.distance_measure = [1 / distance, 1 - distance]

And the dependencies can be interpreted as in Figure 3.6

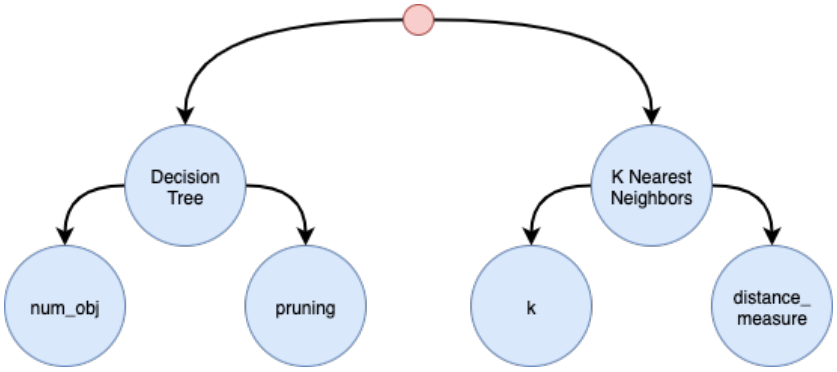


Figure 3.6: A combined hierarchical hyper-parameter optimization problem example.

Then, by denoting with Λ^1 the hyper-parameter space related to the Decision Tree algorithm, and with Λ^2 the K-Nearest Neighbor's one; in order to apply SMBO, we can reformulate

CASH by introducing λ_r and joining the hyper-parameter spaces, Λ^1 and Λ^2, in a single one, Λ.

Λ^1 = { [2, 3], [True, False]}
Λ^2 = { [3, 4], [1/ distance, 1 - distance]}

λ_r = {DecisionTree, K-NearestNeighbor}
$\Lambda = \Lambda^1 \cup \Lambda^2 \cup \lambda_r$

The consequence is that we can consider the Λ space as the domain of our Bayesian problem and the performances evaluation of those algorithms as the time-consuming function to be discovered. Indeed, searching for the optimal solution means looking for the best algorithm and, in the meanwhile, optimizing its hyper-parameters. Therefore, solving CASH. The enabling factor is the additional λ_r parameter: it allows to consider just the right hyper-parameter space according to the algorithm, and ignoring the others, so that SMBO can be performed. In Table 3.1, all the possible combinations of the previous example are listed.

Λ^1	Λ^2	λ_r
2, True		DecisionTree
2, False		DecisionTree
3, True		DecisionTree
3, False		DecisionTree
	3, 1 / distance	K-NearestNeighbors
	3, 1 - distance	K-NearestNeighbors
	4, 1 / distance	K-NearestNeighbors
	4, 1 - distance	K-NearestNeighbors

Table 3.1: A combined hierarchical hyper-parameter optimization problem example.

An example of a real application is Auto-Weka which, unlike other frameworks, does not make any further changes. It simply performs SMBO on a wider range of algorithms and

hyper-parameters. As already mentioned in the previous chapter, 2.4.3 Centralized tools, the two implementations used by it are SMAC and TPE.

3.2.2 SMBO as a DPSO resolution

Also in this case, since DPSO is formalized as an optimization problem, SMBO is proposed as a valid solution. We saw its application to the Modeling step but in reality the process does not change if we apply it to a sequence of steps, therefore a pipeline. Figure 3.7 provides two graphical representation about the dependencies in these problems: the ones related to the Modeling problem (top), with two algorithms and two hyper-parameters for each, and ones related to the Pre-processing problem (bottom), with two transformations, two operators each, and in turn, two parameters for each.

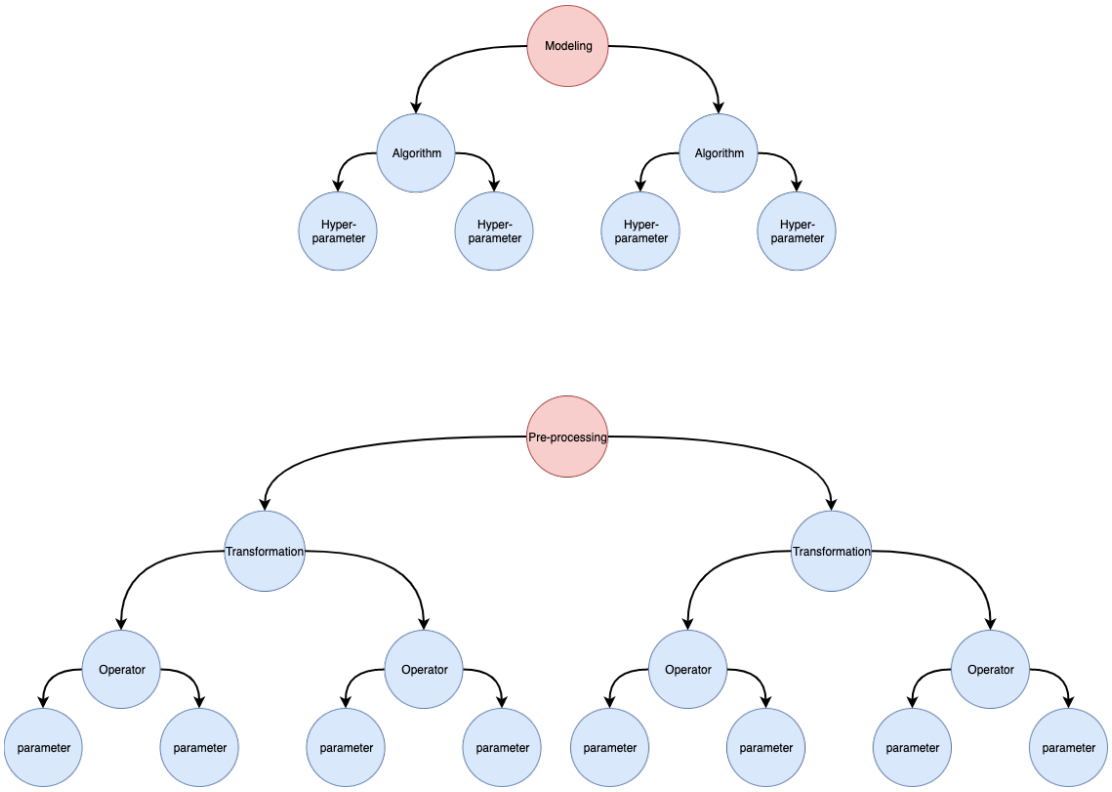


Figure 3.7: Hierarchical Dependencies in CASH (top) and DPSO (bottom) problems.

The process of selecting the best algorithm, and its hyper-parameters configuration, is identical to selecting the best transformation's operator, and its parameters configuration. The problem has just one layer more and SMBO can be customized to deal with it.
This approach is, indeed, the one adopted by A. Quemy. His aim was to demonstrate the effectiveness of the SMBO algorithm on the data pipeline. For this reason he had to run, on different data-sets with different algorithms, both exhaustive search and SMBO; so that he could compare the results and demonstrate the goodness of the solutions found by SMBO.
It creates a data pipeline prototype composed by three transformations in the following order: Re-balancing, Normalization and Features Engineering. For each step he includes some concrete operators, with a configuration space, but it includes also the "None" option, to give the pipeline the ability to skip a step. The chosen data-sets are three of the well-known UCI datasets: Iris, Wine and Breast; and the four used algorithms (SVM, Random Forest, Neural Network and Decision Tree) are evaluated through the 10 cross-validation method.
The framework he used was Scikit-learn, in Figure 3.8 we can see the results of SMBO and the exhaustive research in comparison. Just after 20 iterations, SMBO reaches its best solution, positioning itself very close to the optimal one.

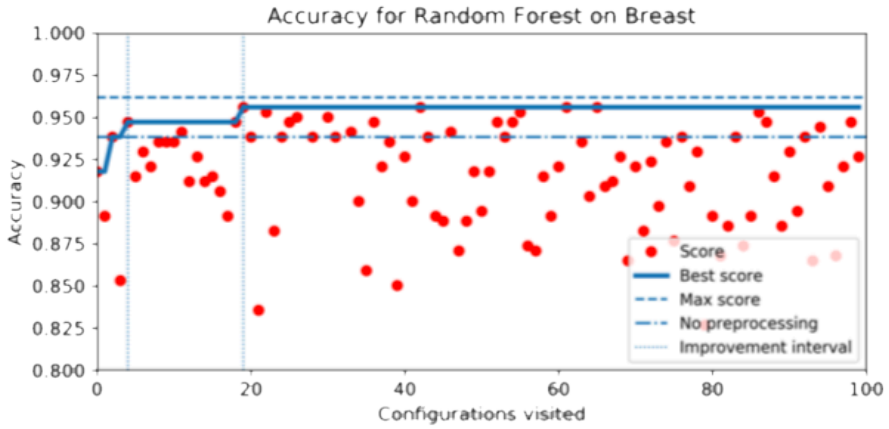


Figure 3.8: Accuracy with SMBO for 100 configurations explored. The results are computed using Random Forest on Breast. From *Data Pipeline Selection and Optimization* [32].

The goodness of each solution is measured applying the Data Mining algorithm after the Pre-processing pipeline. Since the hyper-parameters are fixed to the default ones, the accuracy of the learner actually measures the effectiveness of the considered data pipeline. In the Figure 3.9, instead, there are the pipelines of these solutions: on top the optimal one, found by the exhaustive search, and, in the bottom, the ones found by SMBO.

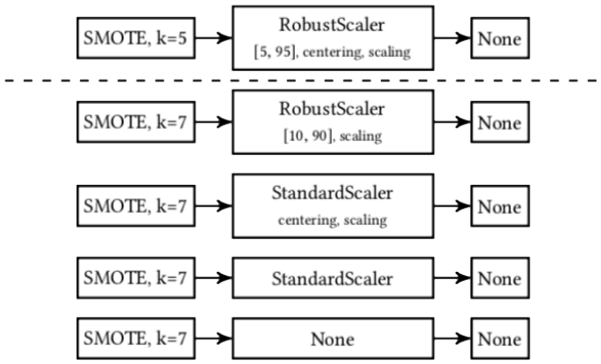


Figure 3.9: The optimal pipeline (top) and the best pipelines found by SMBO. The results are computed using Random Forest on Breast. From *Data Pipeline Selection and Optimization* [32].

We can say that Quemy demonstrates the effectiveness of SMBO on the data pipeline and he provides a DPSO solution approach. However, the case of study was really specific:

- Very limited number of transformations;
- Data-sets with just numerical attributes;
- Transformations applied globally, to all the features;
- Data pipeline prototype fixed and built ad hoc for the used data-sets.

All in all, this study could not be considered a general solution for DPSO. It is comprensible because he had to run both exhaustive search and SMBO; indeed an exhaustive search on a

longer pipeline would have been unfeasible. Now that we know the effectiveness of SMBO on this problem, we can use the Quemy's solution as a starting point, and extend it, for the purpose of solving the current limitations.

4

Automated Data Pre-processing

Following, we are going to present our solution to the DPSO problem.
First of all, we start from the limitations of the Quemy's approach [32] and we describe the general architecture that emerges as these issues are fixed. This first analysis will pave the way to introduce the reader to the two main aspects of our architecture.
In fact, afterwards, we are going into details about the Offline and Online Phases. In the former we explain how the components that constitute our approach are created. In the latter we, instead, explaining how these components interact and how the data are modified through the whole flow.

As we already said, the Quemy's solution to the DPSO problem was valid but he chose a really specific case of study:

1. Very limited number of transformations;
2. Data-sets with just numerical attributes;
3. Transformations applied globally, to all the features;
4. Data pipeline prototype fixed and built ad hoc for the used data-sets.

Below, we are going to discuss how each bullet has been faced and solved.
Starting from the top, we simply extend the number of available transformations. Quemy used:

- Rebalancing, with three different operators: NearMiss (undersampling), CondensedNearestNeighbour (undersampling) and SMOTE (oversampling);
- Normalization, with four operators: StandardScaler (removes the mean and scales to unit variance), PowerTransform (transforms the distribution to a Gaussian-like one), MinMaxScaler (scales each feature between 0 and 1) and RobustScaler (same as StandardSclaer but removes points outside a percentile range);
- Feature Engineering, with three options: PCA, Features Selection and union of features obtaines by PCA and Features Selection.

We added the followings:

- Discretization, with two different operators: KBins Discretization (creates categorical attributes with K categories) and Binarization (creates categorical attributes with just two categories);
- Imputation, with two operators: Univariate Imputation (imputes a constant, either number or sting) and Multivariate Imputation (estimates the missing values through the other features, by running a Data Mining algorithm);

- Encoding, which includes Ordinal Encoding (transforms each distinct value into a integer) and One Hot Encoding (create a binary attribute for each distinct value).

The second problem is that he used data-sets with only numerical attributes. This fact allowed him to apply all the transformations globally, therefore to all the attributes, without checking that they were effectively compatible. In fact, he selected transformations that could only work with numeric data-sets.
Instead, we support data-sets with numerical, categorical and mixed features and, because of that, we cannot apply all the transformations to all the features. Indeed, some would not make sense, for example in Figure 4.1 we cannot normalize categorical features.

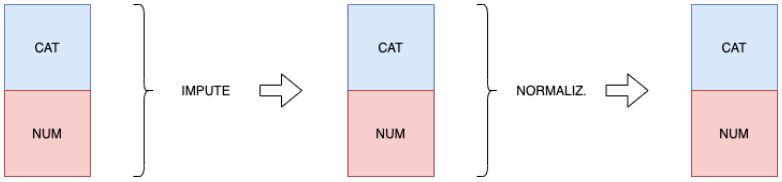


Figure 4.1: Case in which the global application of the transformations is incorrect.

Hence, we apply transformations only to compatible features (Figure 4.2).

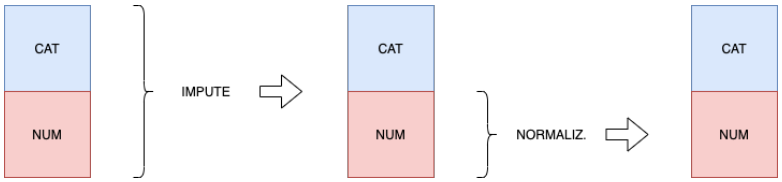


Figure 4.2: Previous case in which the adopted application is only to compatible attributes.

In that way, we defined the domain and co-domain of the transformations. They are listed in Figure 4.3.

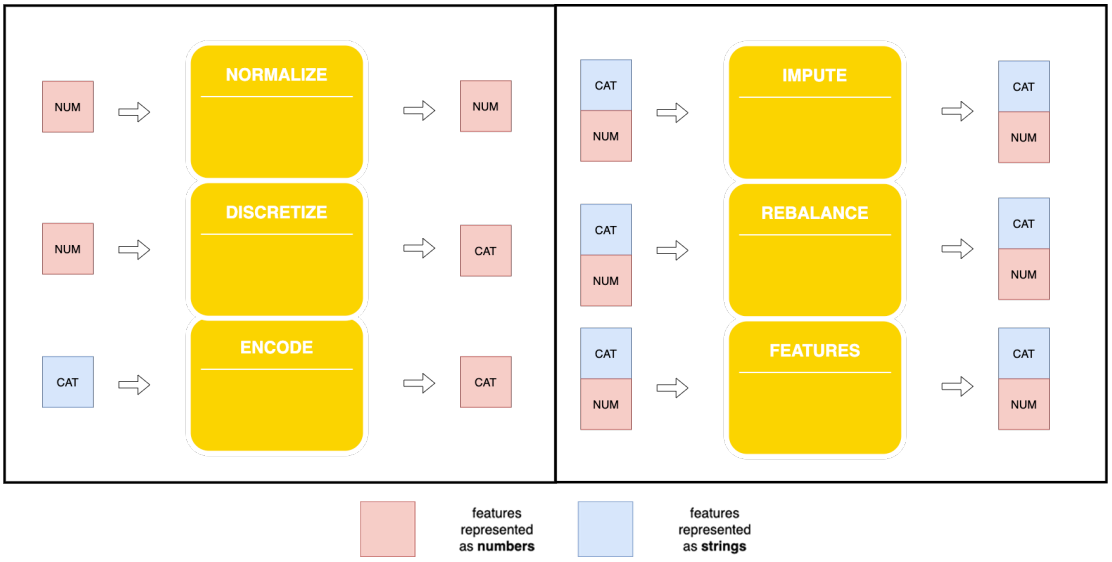


Figure 4.3: Domain and Co-domain of the considered transformations.

As we can see, the categorical attributes (CAT in Figure 4.3) are not always colored in blue. This is because we distinguish between the type of the feature, categorical or numerical, and its representation, string (blue) or number (red). Indeed, even if a categorical attribute is encoded, it would not make sense to consider it as numerical, and hence allow the application of transformations such as Normalization or Discretization. Although its representation is numerical, in our approach, the attribute remains categorical and only transformations compatible with categorical attributes can be applied.

After all, the data pipeline prototype built by Quemy was fixed. Yet still, different transformation orders are not considered. We already know that including the *None* operator in each transformation, pipelines where some transformations could not be present are included too. However, even in this case some combinations cannot be generated, e.g. the Features step can never stand before the Re-balance or Normalize ones.

All the considerations made so far correspond to changes to Quemy's implementation. Now we will see how to incorporate its modified solution into a larger architecture in order to consider different prototypes of data pipelines.

A Naive approach to consider the order could be: given the transformations, generate all the

possible data pipeline prototypes; run SMBO on each of them; and then take the winner (Figure 4.4).

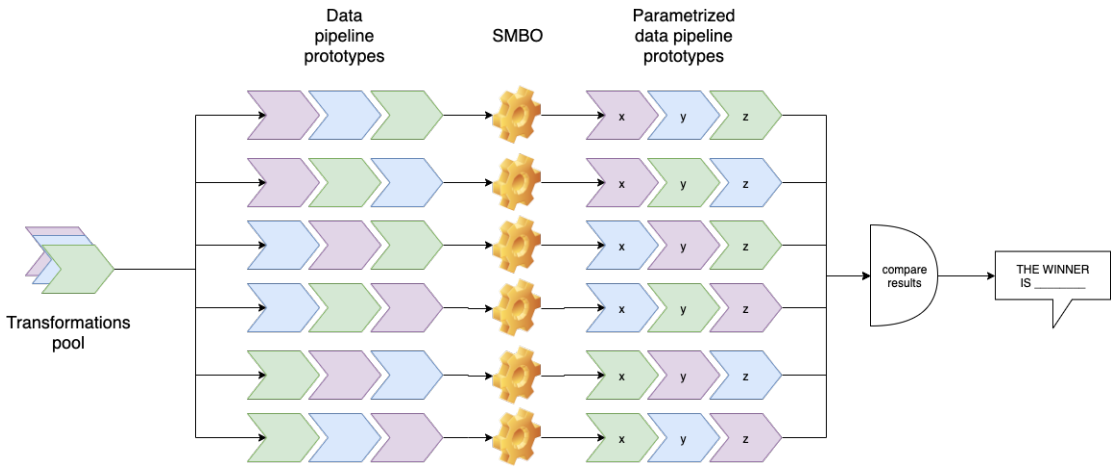


Figure 4.4: Naive approach to consider all the data pipeline prototypes.

Unfortunately, this approach is computationally speaking unfeasible.
The basic idea is to divide the selection of the data pipeline from its optimization.

1. Understand the constraints that lie behind the considered transformations, and in that way, build the promising data pipeline prototypes;

2. Then optimize them, hence, apply SMBO. We will use the Quemy's code but with our modifications, because it allows more transformations, mixed data-sets and the transformations are applied just to the compatible attributes.

Since the best data pipeline prototype depends on: the chosen Data Mining algorithm and the data-set itself, we need both to be specified. An example of the flow we aim to build is represented in Figure 4.5.

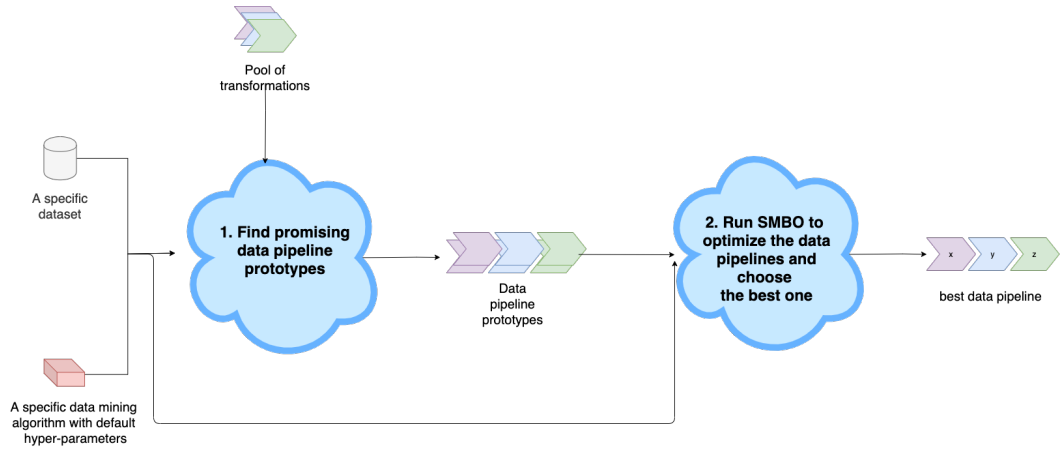


Figure 4.5: Online phase of our approach.

As we said, we want to build the data pipeline prototypes by understanding the constraints between the transformations we are going to use. Indeed, this is literally what we have done: by studing the relationships between the transformations, we built a table that, given a pair of transformations, tells us which one should appear before the other. We refer to that table as Dependency table, since it represents the order dependencies between the transformations. Then, through these constraints, we were able to build the desired pipeline prototypes. In Figure 4.6 we depict the architecture, specifically zooming in on the first step.

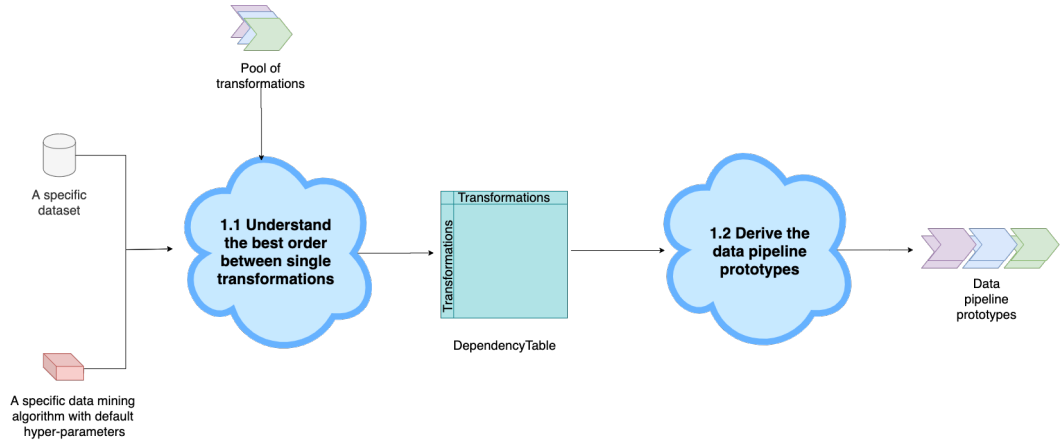


Figure 4.6: Online phase on how to find the data pipeline prototypes.

So far, we explained the desiderata. In fact, we can call this phase, Online phase, therefore the flow that is performed once the system is built.
However, unlike steps 1.2 and 2, we do not know the algorithm that allows us to solve the step 1.1. What we need is to do some "Offline" work, specifically, learn from data.
For this reason we performed some experiments, through SMBO, with: a pool of data-sets, a pool of algorithms and the already known pool of transformations. These experiments consist in testing how the order of the transformations affects the result. Precisely we were interested to know, given a pair of transformations what is the best order to concatenate them (i.e., find the order that yields better performance for the Machine Learning algorithm). The aim was to build our own Training set, and then, apply one or more learners on top of it. Given a data-set and an algorithm, those learners would predict the order between two transformations. We refer to this process as Meta-learning because, to complete the goal, we used the so-called meta-data (the data that describes the data). More precisely the meta-data are common information (such as number of features, number of instances), statistical indexes (such as mean of instances per class) and so on, that can be extracted from data-sets themselves.
However, in some cases, or better for some transformation's pair, the data were explicit and we had statistical confidence to assure that a specific order is, in general, better than the other. Hence, in these situations, the insights got from the experiments were enough and the meta-learner turned out unnecessary. Since the high number of experiments to perform, given the no-small number of combinations between transformations, data-sets and algorithms; it was decided to reduce the number of pairs of transformations to test. In particular, for some of them we could decide a-priori the order in which to concatenate, because of some constraints of the used framework or because the other way around would not make sense. We, hence, built a *Intermediate table* and we integrate it through the experiments insights and the Meta-learning process outcome.
In Figure 4.7 there is the representation of the above explained process.

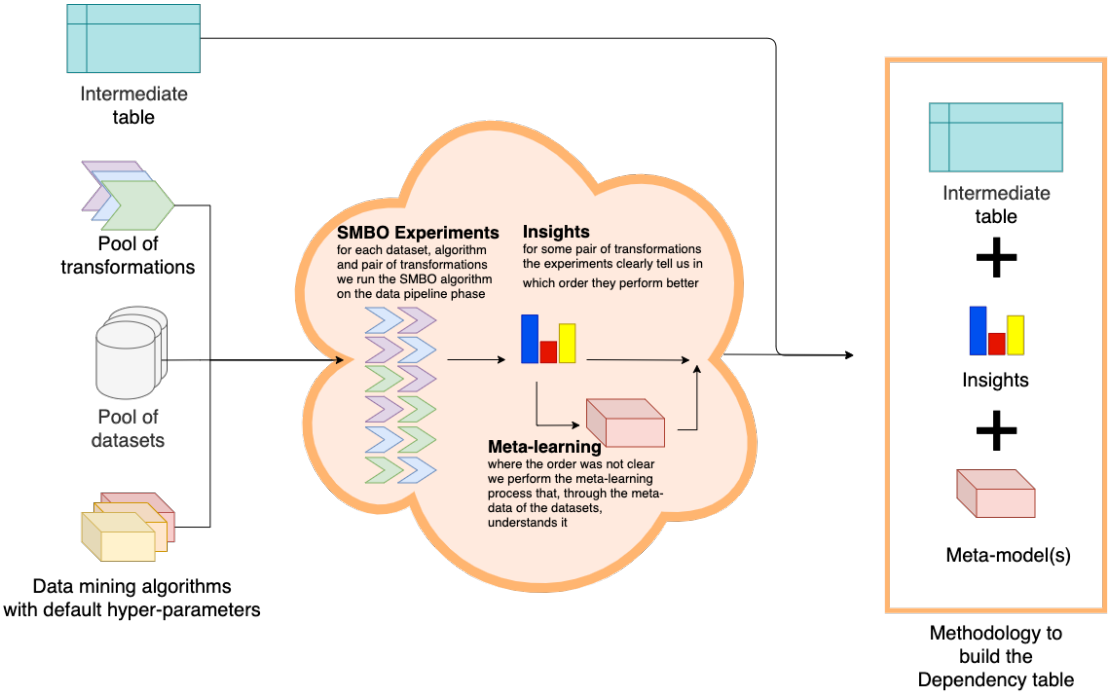


Figure 4.7: Offline phase that allows us to build the Dependency table.

All in all, the Intermediate table and the outcome given by the insights represent the static part of the Dependency table, the one that does not change, regardless of the data-set and the algorithm. But, as it stands, it would be incomplete. The meta-model(s), given data-set and algorithm in input, allow us to complete it. Thereafter, thanks to this knowledge acquired in advance, we can build the promising data pipelines and, running the SMBO algorithm on each of them, select the best one.

4.2 Offline phase

In the following sections we are going to explain: how the Intermediate table is built (4.2.1), the details about the performed experiments and the interpretation of their insight (4.2.2), and the used Meta-learning methodology (4.2.3).

4.2.1 Intermediate table building

In order to build the intermediate table, first of all, we checked if the framework we use imposes some constraints. As we said, Quemy uses Scikit-learn and, since we enriched his solution, we use it too. For each transformations pair we tested both the possible orders. In some cases, we noticed that just one of them was valid and, hence, we could assign the dependency without running any experiments. In Figure 4.8 that table is shown, with the related legend.

Legend	
Color	Description
	Compatible only doing first the transformation in the row
	Compatible only doing first the transformation in the column
	Compatible in any order
	Not compatible in any order

	Normalize	Encode	Discretize	Impute	Rebalance
Encode					
Discretize					
Impute					
Rebalance					
Features					

Figure 4.8: Table resulting from the compatibility analysis.

Observe that the table is triangular. This is because we consider each comparison only once. In the following we discuss in more details the cells where the order has been assigned.
Regarding the Encode-Normalize case, both configurations seem valid but, the Scikit-learn method used to select the features on which to apply the transformations cannot manage an array with different elements types as output. For this reason, we can see in Figure 4.9 that, the second option was discarded. The same considerations are valid also in the Encode-Discretize case.

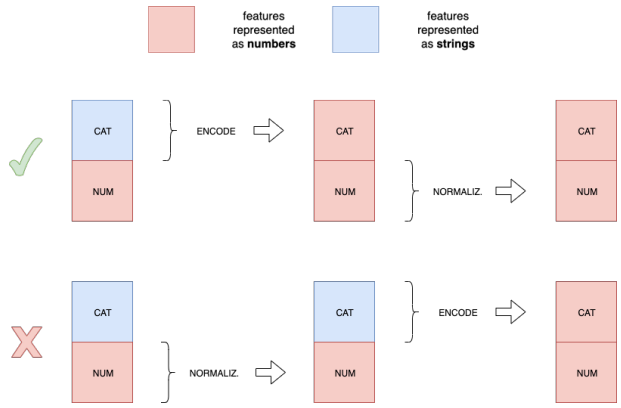


Figure 4.9: Graphical representation of the Encode-Normalize case, regarding the compatibility analysis.

In the other two cases in which the Encoding operation is involved, therefore the ones with Re-balancing and Features Engineering, we put Encoding first because the Scikit-learn operators of these two latter transformations, cannot operate with attributes in a string format. Instead, regarding the order fixed by the Imputation techniques, we noticed that the Encoding, Discretization, Re-balancing and Features Engineering operations do not work with missing values. Therefore, the Imputation step has to be applied before them.
Afterwards, we asked ourselves if all the compatible combinations actually make sense. For example, in the Discretize-Normalize case, we have that, according to the compatibility table, both of the configurations are valid (Figure 4.10).

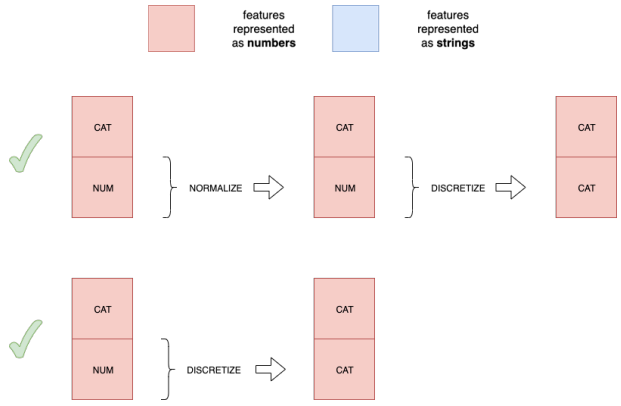


Figure 4.10: Graphical representation of the Discretize-Normalize case, regarding the compatibility analysis.

Although, in this case two considerations can be made:

- Applying the Normalization first, the Discretization sets aside the Normalization effects;
- Applying the Discretization first, there is no point in applying the Normalization afterwards, because we do not have any numerical values anymore.

For this reason we created a table which, regardless of the framework, describes the transformations' dependencies (Figure 4.11).

Legend	
Color	Description
	Makes sense only doing first the transformation in the row
	Makes sense only doing first the transformation in the column first
	Makes sense in any order
	Does not make sense in any order

	Normalize				
Encode		Encode			
Discretize			Discretize		
Impute				Impute	
Rebalance					Rebalance
Features					

Figure 4.11: Table of constraints not considering the used framework.

As we can see, the constraints which applied to the Encoding operations are no longer there, because those constraints were due to the used framework. Furthermore, we have more variations in the Normalize column. Leaving out the Encode-Normalize and Discretize-Normalize cases, because they were discussed above, the two remaining cells to be explained are Imputate-Normalize and Rebalance-Normalize.

Regarding the former, although in Scikit-learn the Normalizers can work with missing values, we do not see why the Normalization applied first should perform better. Indeed, we believe that, as with the other transformations, the Imputation first has more benefits.
Instead, regarding the Rebalance-Normalize, we put Rebalance first because the Normalization first could affect the sampling process.
All in all, in order to understand in which combinations we still do not have a pre fixed order, we merged the two tables, obtaining the Intermediate table (Figure4.12). For each cell, we kept the most binding one (i.e., the more restrictive one).

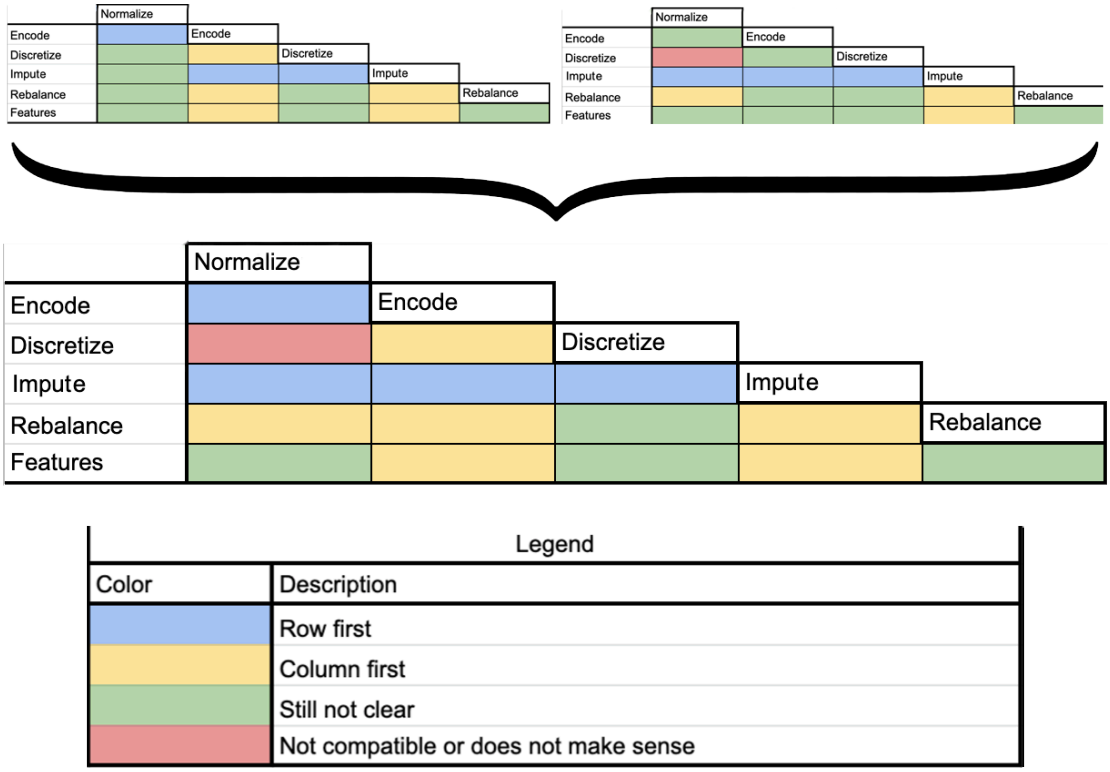


Figure 4.12: Intermediate table construction, by merging the two above mentioned tables.

Once again, the green cells in the resulting table are the ones where there are no constraints and hence we would like to understand in which order the transformations in question perform better. To do that, we performed the experiments explained in the following section.

4.2.2 SMBO experiments and insights interpretation

Since a cell identifies a pair of transformations, the idea is to run, for each pair, the SMBO algorithm on both the possible combinations. For instance, taking the pair Features-Rebalance, SMBO has been run once on the data pipeline prototype Features-Rebalance and once on Rebalance-Features.
Furthermore, to have a statistical reliability we had to perform the experiments with a collection of heterogeneous data-sets and a selection of different Data Mining algorithms. In that way we can assure that the performed experiments actually represent a wide real-cases problems.
Concerning the data-sets, we used the OpenML-CC18 suite. It contains 72 data-sets in total, that satisfy a large set of bench-marking requirements:

- The number of observations are between 500 and 100′000 to focus on medium-sized data-sets, that are not too small and not too big;
- The number of features does not exceed 5000 features to keep the run-time of algorithms low;
- The target attribute has at least two classes;
- Have classes with less than 20 observations;
- The ratio of the minority class and the majority class is above 0.05, to eliminate highly imbalanced data-sets which require special treatment for both algorithms and evaluation measures.

From these, we excluded data-sets which:

- Have more than 5′000′000 tuples, for reasons of time, given that the duration to train the models depends on the size of the data-set;
- Contains more than 10% of missing values and contains more than 10% of instances with missing values;

In total we considered 60 data-sets. Regarding the Data Mining algorithms, we used three of the main ones: Naive Bayes[23], K-Nearest Neighbor[31] and Random Forest[5]. Hence, taking again the Features-Rebalance case as example, for each data-set in the OpenML-CC18 suite, we run the experiments as shown in the Figure 4.13.

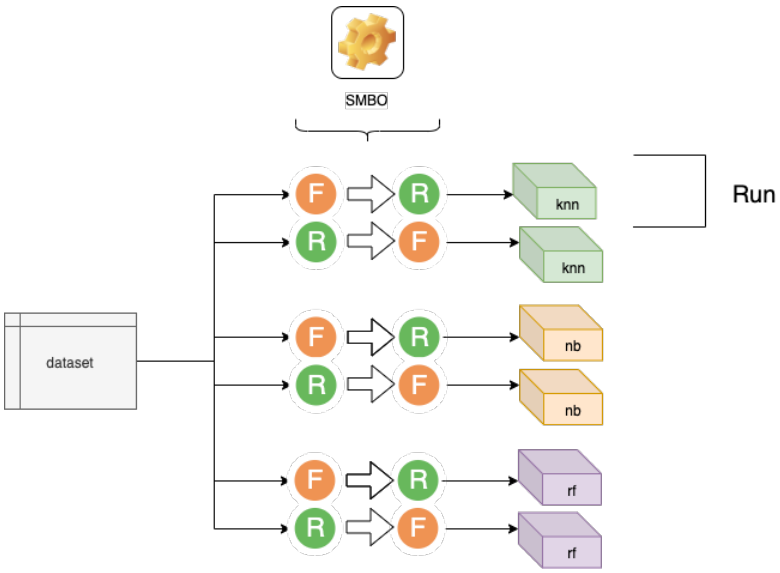


Figure 4.13: Graphical representation of the performed SMBO experiments.

The data-set is modified through the pipeline and given as input to the Data Mining algorithm. In each run, actually, this phenomenon happens as many times as the number of SMBO iterations. In fact, the optimization algorithm works on the operators of the data pipeline and tries to find the best ones, and the related best parameter values.
Given a data-set and an algorithm, we have two runs, or configurations: where the order of the transformations is the opposite. For each run we gave to SMBO a time budget to perform: 400 seconds. With this budget per run, just the Features-Rebalance experiment took more or less 48 hours.
Since the algorithm run each iteration with its default parameters, the accuracy measures the goodness of the found data pipeline. In particular, we observed many data-sets with an unbalanced class problem, hence, we used Balanced accuracy instead of accuracy. However, given

a data-set and an algorithm we could compare both configurations and, then, understand which data pipeline prototype performed best. Although, not in all cases, we could say that one was better than the other. SMBO has the possibility to disable the transformations and, if this happens, for that case, it would mean that the disabled transformations should not have been present. We would like to represent this information in the result and, to do that, we extracted a label from the winning data pipeline. In Figure 4.14 there are all the possible labels.

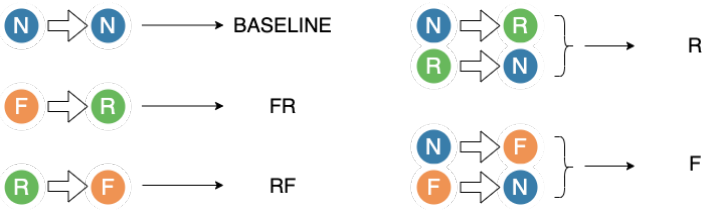


Figure 4.14: Result label extraction from the winning data pipeline.

The second issue is given by the very nature of the SMBO algorithm. According to the problem nature, in the cases where the winning pipeline has one or two disabled transformations, that pipeline can be found by both the configurations, because the order does not count. Unfortunately, due to the SMBO indeterminism, we cannot guarantee that this happens. In general, we cannot ensure that the algorithm, in both of the configurations, built a good model or had enough time. On the other hand, we can assure that one of the runs has not "worked well", if the winning data pipeline found by the winning configuration could have also been found by the other configuration, but actually failed. We called these cases as invalid cases.

In Figure 4.15 we listed all the possible cases. In particular we listed the possible pipelines for both configurations and then for each possible pair we defined a rule (Table 4.1). That rule specifies which configuration, according to the pipeline prototypes, must win *to be valid*. As we said, given the nature of the problem, an invalid case could occur just when the winning pipeline contains one or more transformations with the "None" operator. Hence, to be valid, those runs must draw.

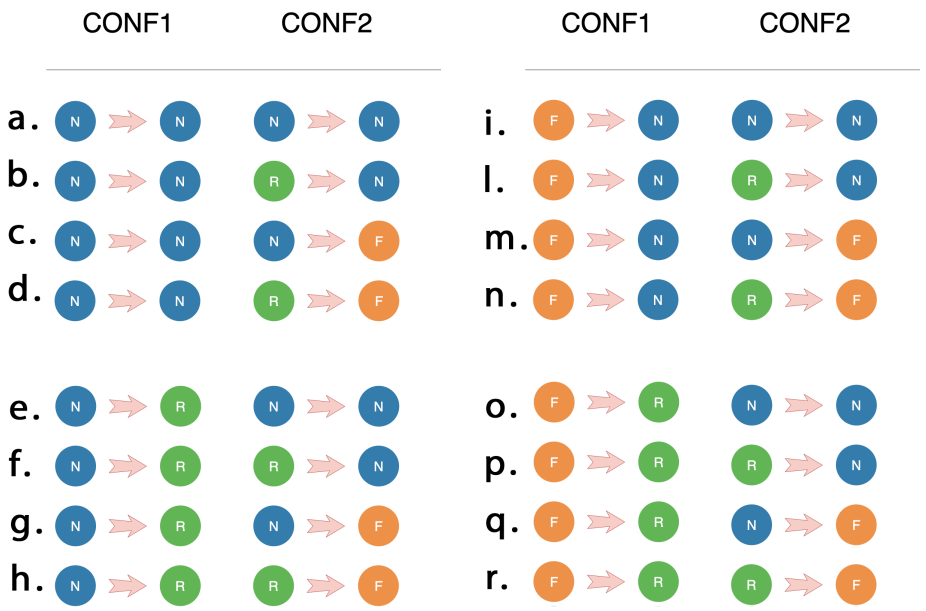


Figure 4.15: Enumeration of all possible pipelines.

	to be valid	result
a	DRAW	BASELINE
b	DRAW	BASELINE
c	DRAW	BASELINE
d	DRAW or CONF2	draw ⇒ BASELINE, CONF2 ⇒ RF
e	DRAW	BASELINE
f	DRAW or CONF1 or CONF2	R
g	DRAW	FoR
h	DRAW or CONF2	draw ⇒ R, CONF2 ⇒ RF
i	DRAW	BASELINE
l	DRAW	FoR
m	DRAW or CONF1 or CONF2	F
n	DRAW or CONF2	draw ⇒ F, CONF2 ⇒ RF
o	DRAW or CONF1	draw ⇒ BASELINE, CONF1 ⇒ FR
p	DRAW or CONF1	draw ⇒ R, CONF1 ⇒ FR
g	DRAW or CONF1	draw ⇒ F, CONF1 ⇒ FR
r	DRAW or CONF1 or CONF2	draw ⇒ DRAW, CONF1 ⇒ FR, CONF2 ⇒ RF

Table 4.1: Rules for validating and assigning the result label to two configurations.

Then, as we said, we extract the label from the winning pipeline. It could be that two runs found different pipelines but with the same Balanced accuracy. Whenever this happens, we extract the label from the simplest one.
It can be noticed that, since we are focusing on the data pipeline prototype, in the configurations where the two prototypes are the same (f. and m. in Figure 4.15), we are not caring about the Balanced accuracy, that could be different due to the parameters optimization. If it were not, we would have had to restrict the validation rule to a draw. All in all, we run the experiments to understand the order between the transformations that had no constraints in the Dependency table, therefore: Features - Normalize, Features - Discretize, Features - Rebalance, Discretize - Rebalance.
In each case, grouping by the algorithm, we observed less than 10% of invalid results (Figure 4.16).

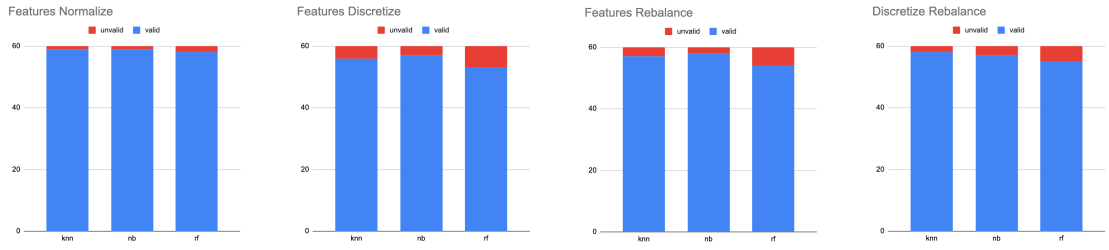


Figure 4.16: Graphs depicting the number of valid and invalid results.

Then, considering only the valid ones, for each data-set, according to the above rules, we extracted the label from the winning pipeline. Finally, for each algorithm, we counted the occurrences of each label, depicting the outcomes in Figures 4.17 and 4.18.

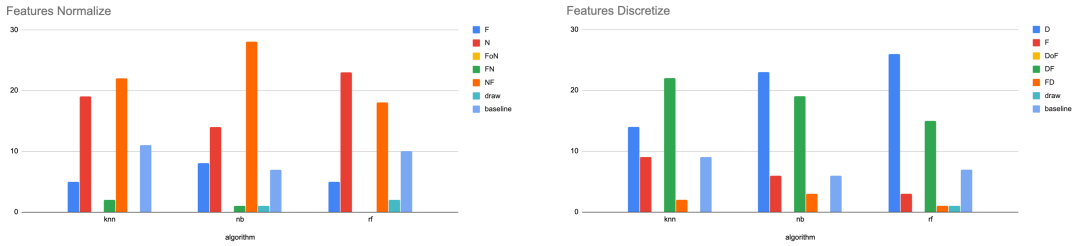


Figure 4.17: Graphs depicting the labels about the valid results.

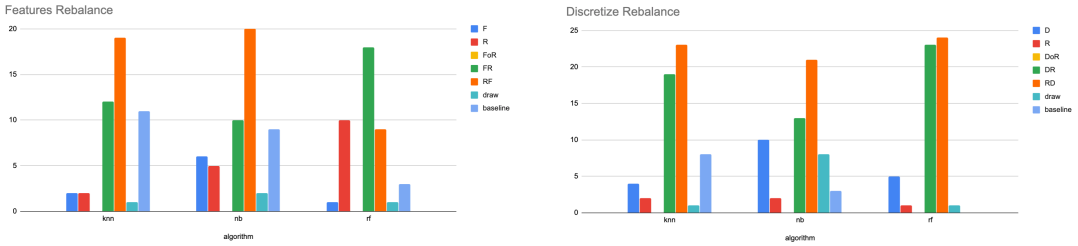


Figure 4.18: Graphs depicting the labels about the valid results.

The legend reports the labels explained in Figure 4.14. The only label that is not present, considering the Features - Rebalance case, is the FoR one. This is applied when a pipeline with only the Features transformation and a pipeline with only the Rebalance transformation draw (g. and l. cases in Figure 4.15).

In the cases of Features - Normalize and Features - Discretize (Figure 4.17), for each algorithm, we had a clear evidence of the winning order. First of all, we removed from the analysis the labels that SMBO could find regardless of the prototype. Therefore, we kept only the results that saw both transformations present (columns green and orange in Figure), excluding draws. In particular, we were searching for a statistical significant difference between these two frequencies. We run the chi-square test, which assures that the distributions, in those cases were not random. Plus, we also compared them with a binary-like distribution,

where the frequencies of the values are in a 90:10 proportion. We can assure with 95% of confidence that Normalization before Features Engineering performs better than the other way around and, likewise, that the Discretization before Features Engineering is better.
These evidences allow us to assign, in the Dependency table, an order to the mentioned transformations.
Unfortunately, the other two cases do not show any evidence. For this reason we have the meta-learning process to induce the order.

4.2.3 Meta-learning process

So far, in the SMBO experiments, we classified the data-sets according to some labels. In each experiment we considered a pair of transformations and, for a specific data-set and algorithm, we were able to understand which is the best way to concatenate the transformations in question. Since we could not find a generic rule for all the experiments, the idea was to use the information about the data-sets to train a Machine Learning algorithm and learn to recognize the transformations' order. To be able to discriminate among different data-sets, we need to extract some information that describes them, the meta-data. Those meta-data would be the features of our Training Set, therefore we call them meta-features, and the label would be the class to predict. By training a learner, or better a meta-learner, (Offline phase) we will be able to predict (Online phase), for a given data-set and a given Data Mining algorithm, which is the best order to concatenate the specified transformations. Hence, this would mean, we are going to be able to complete the Dependency table. In Figure 4.19 we depict a scheme that summarizes the meta-learning process.
First of all, we defined the meta-features extracted from the data-sets:

- General: general information related to the data-set, also known as simple measures, such as number of instances, attributes and classes;
- Statistical: standard statistical measures to describe the numerical properties of a distribution of data;
- Information-theoretic: particularly appropriate to describe discrete (categorical) attributes and their relationship with the classes;

- Model-based: measures designed to extract characteristics like the depth, the shape and size of a Decision Tree (DT) model induced from a data-set;
- Landmarking: represents the performance of simple and efficient learning algorithms.

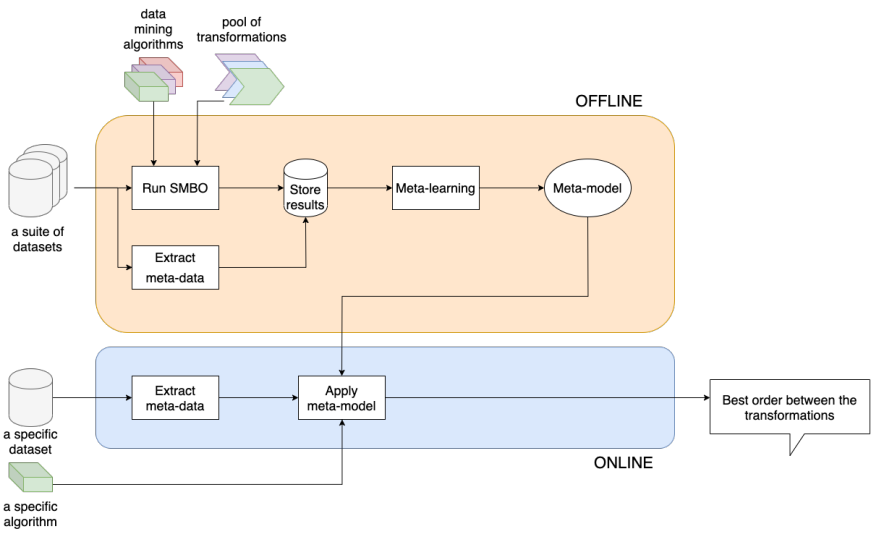


Figure 4.19: Meta-learning working.

Then, analyzing the experiments' results, we decided not only to build different meta-learners for different pairs of transformations, but also, to consider separately the results of each algorithm, and hence, to build different meta-learners for different algorithm results. To this end, since we have two pairs of transformations (Features - Rebalance and Discretize - Rebalance), and three data mining algorithms (K-Nearest Neighbor, Naive Bayes and Random Forest), in total we need six different meta-learners.

A further consideration to be done is that, since we are not interested in the labels where one or both of the transformations are disabled — the order is not crucial for learner purposes, we have grouped these labels into a class called *no_order*. Figure 4.20 illustrates how the class distribution changes in the Features - Rebalance case.

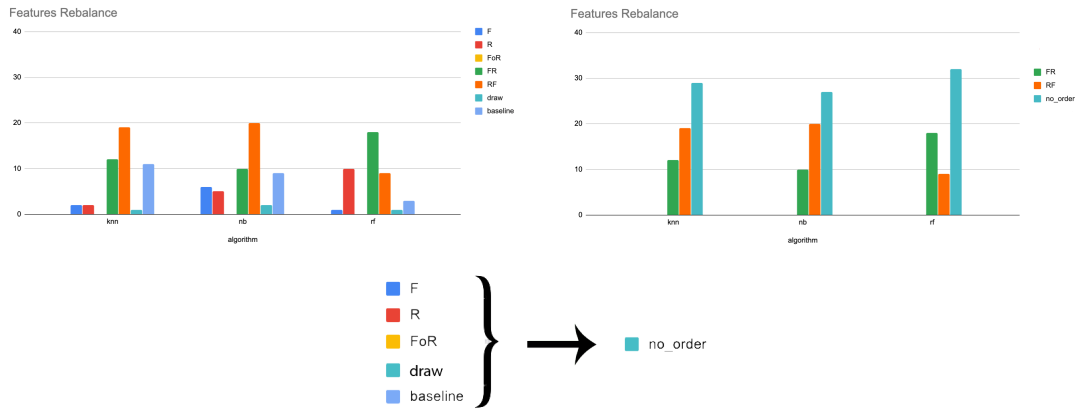


Figure 4.20: Graphs depicting the labels before (left) and after (right) the grouping.

We have analyzed the problems and we detected the following issues:

- It was not always possible to extract all the meta-features from the data-sets and this led us to have a high number of missing values;
- We have really few instances and, without taking the right countermeasures, we might build over-fitted models;
- We have imbalanced classes.

Regarding the first issue, high number of instances with missing values, we considered two different approaches:

- Impute those values, inserting a string could lead the algorithm to find some correlations in the data;
- Let the algorithm deal with them.

For this purpose, we considered different frameworks:

- H2O is available in Python and *R*, two wide-used programming languages in the Machine Learning field. In it we found algorithms capable of managing missing values during the model building;

- Scikit-learn is the Python library used for the SMBO experiments. As we already know, the algorithms in this package do not work with missing values. This forces us to use an imputation step. Moreover, the fact that those algorithms cannot work even with data in the form of strings, obliges us to impute a number;
- In Weka, we found both algorithms that manage missing values and techniques that impute numbers and strings.

After some trials, we decided to not impute the missing values and let the algorithm manage them. Hence, we discarded Scikit-learn.
In order to compare Weka and H2O we tried different Data Mining algorithms (with related Pre-processing techniques), in particular:

- In Weka:
  - K-Nearest Neighbor with PCA, Feature Selection and Normalization;
  - Decision Tree and Random Forest with Discretization;
- In H2O:
  - Random Forest;
  - XGBoost and GBM, boosting of Decision Trees;

Also, we tried the AutoML approaches of those frameworks: Auto-Weka and H2O AutoML. The former suggests a K-Nearest Neighbor approach, the latter a new ensemble method called Stacking. In Table 4.2 we show the results about the Features - Rebalance case, for simplicity we report only the ones without Pre-processing, which in this case was not of particular help.
The evaluation was made through the Leave One Out validation, in order to exploit the few instances in both training and testing phases.

Random Forest Training Set

Framework	Algorithm	Accuracy
Weka	Random Forest	0.72
Weka	K-NN	0.67
H2O	Random Forest	0.74
H2O	XGBoost	0.72
H2O	GBM	0.64
H2O	Stacking	0.51

K-NN Training Set

Framework	Algorithm	Accuracy
Weka	Random Forest	0.48
Weka	K-NN	0.46
H2O	Random Forest	0.53
H2O	XGBoost	0.56
H2O	GBM	0.55
H2O	Stacking	0.48

Naive Bayes Training Set

Framework	Algorithm	Accuracy
Weka	Random Forest	0.59
Weka	K-NN	0.54
H2O	Random Forest	0.52
H2O	XGBoost	0.50
H2O	GBM	0.59
H2O	Stacking	0.50

Table 4.2: Feature - Rebalance results.

The cells in blue are the ones that have achieved the best performance. Since the H2O framework performed better in all cases, we discarded Weka.
Regarding the imbalanced classes issue we tried to use Rebalancing techniques, in particular over-sampling techniques because of the low number of instances. In Table 4.3 we show the already seen H2O results, in the Features - Rebalance case, but with the over-sampling step.

Random Forest Training Set

Framework	Algorithm	Accuracy
H2O	Random Forest	0.72
H2O	XGBoost	0.67
H2O	GBM	0.76

K-NN Training Set

Framework	Algorithm	Accuracy
H2O	Random Forest	0.6
H2O	XGBoost	0.55
H2O	GBM	0.56

Naive Bayes Training Set

Framework	Algorithm	Accuracy
H2O	Random Forest	0.57
H2O	XGBoost	0.55
H2O	GBM	0.57

Table 4.3: Feature - Rebalance results with oversampling.

In green we can find the cases where the Rebalance step led to an improvement, in red a worsening. Surprisingly, they do not lead to a substantial improvement in all the cases. Let us take again the Features - Rebalance case, regarding Random Forest as meta-learner on the Random Forest Training Set; in Figure 4.21 there is the confusion matrix before and after the over-sampling step.

Prediction→ Actual ↓	FR	no_order	RF
FR	15	5	4
no_order	3	27	3
RF	0	0	2

Prediction→ Actual ↓	FR	no_order	RF
FR	16	3	4
no_order	1	25	3
RF	1	4	2

Figure 4.21: Confusion matrix of Features - Rebalance case, Random Forest Training Set, running Random Forest as meta-learner, before (left) and after (right) the over-sampling step.

We can notice that this Pre-processing transformation actually balanced the number of predictions per class, according to the class distribution. Unfortunately, this is not enough to distinguish well the classes and, therefore we have discarded this technique.
However, since our aim is to decide the order of the transformations, even when the meta-learner predicts *no_order* class, we have to assign an order. The choice has no impact when the instance classified as such is actually *no_order* because our choice will never be wrong. Instead, in the other situations, we have to favor one class over the other. As a matter of fact, the most convenient choice is to move the *no_order* instances according to the class that contains more mistakes, in order to turn them into correct predictions and, hence, have a greater accuracy. A hypothetical case is represented in Figure 4.22, where moving to the minority class is more convenient, since that is where the learner made more mistakes.

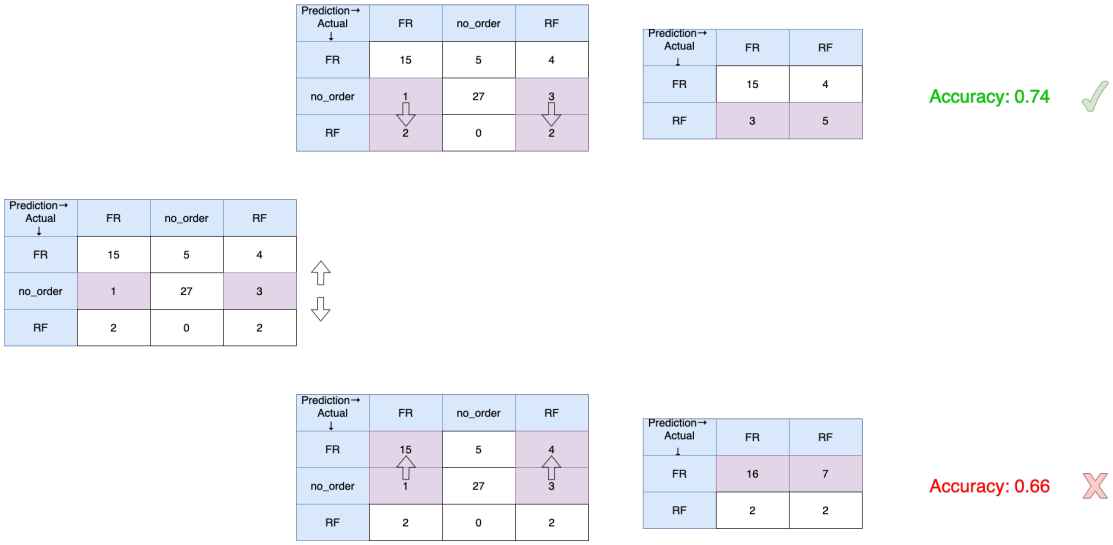


Figure 4.22: Example of assigning an order to the no_order instances. Majority class case on top, minority class case in the bottom.

Hence, in both Features - Rebalance and Discretize - Rebalance cases, for each meta-learner, we tried to move the *no_order* instances to both options. We compared the performances and in each case we noted which one reaches the best accuracy. In Figure 4.23 this comparison is shown.

FEATURES - REBALANCE				
training set	algorithm	majority size	minority size	winner
random forest	random forest	18	9	draw
random forest	gbm	18	9	majority
random forest	xgboost	18	9	majority
naive bayes	random forest	20	10	majority
naive bayes	gbm	20	10	majority
naive bayes	xgboost	20	10	majority
knn	random forest	19	12	majority
knn	gbm	19	12	minority
knn	xgboost	19	12	draw

DISCRETIZE - REBALANCE				
training set	algorithm	majority size	minority size	winner
random forest	random forest	24	23	minority
random forest	gbm	24	23	minority
random forest	xgboost	24	23	majority
naive bayes	random forest	21	13	majority
naive bayes	gbm	21	13	majority
naive bayes	xgboost	21	13	majority
knn	random forest	23	19	minority
knn	gbm	23	19	majority
knn	xgboost	23	19	majority

Figure 4.23: Comparison between assigning the no_order instances to the minority and majority class.

It can be noticed that choosing the majority class is almost always the best choice. In addition, where the minority class is chosen the classes are not so imbalanced. For these reasons we decided to be uniform and always move to the majority class.
Then, since we want the meta-learners to be all of the same type, we have made some comparison in order to elect the best one. Firstly, having noticed that the performance of the individual meta-learners do not vary much, we tried to understand if this fact was due to how these algorithms were instantiated from time to time (indeterminism) or if effectively one was always better than the other. For this purpose we ran those algorithms with different seeds, therefore randomizing their instantiation. In Figure 4.24 the results.

FEATURES - REBALANCE					
training set	algorithm	seed = 1234	seed = 567	seed = 89	max - min
random forest	random forest	0.7288	0.6949	0.7119	0.0339
random forest	gbm	0.7288	0.7288	0.7288	0
random forest	xgboost	0.6271	0.6271	0.6271	0
naive bayes	random forest	0.5439	0.5088	0.614	0.1052
naive bayes	gbm	0.5789	0.5789	0.5789	0
naive bayes	xgboost	0.5088	0.5088	0.5088	0
knn	random forest	0.5333	0.55	0.5667	0.0334
knn	gbm	0.55	0.55	0.55	0
knn	xgboost	0.5667	0.5667	0.5667	0

DISCRETIZE - REBALANCE					
training set	algorithm	seed = 1234	seed = 567	seed = 89	max - min
random forest	random forest	0.463	0.537	0.537	0.074
random forest	gbm	0.4815	0.4815	0.4815	0
random forest	xgboost	0.4815	0.4815	0.4815	0
naive bayes	random forest	0.5345	0.4483	0.5	0.0862
naive bayes	gbm	0.5	0.5	0.5	0
naive bayes	xgboost	0.5172	0.5172	0.5172	0
knn	random forest	0.4912	0.5614	0.5263	0.0702
knn	gbm	0.6316	0.6316	0.6316	0
knn	xgboost	0.5614	0.5614	0.5614	0

Figure 4.24: Study of meta-learners with different seeds.

As we can see, by varying the seed, the Random Forest performance varies as well. On the contrary, it seems that this phenomenon does not affect the performance of the boosting approaches. For this reason we discarded Random Forest as meta-learner and in Figure 4.25 we compared the performance of GBM and XGBoost.

FEATURES - REBALANCE		
training set	gbm	xgboost
random forest	0.7778	0.7037
naive bayes	0.5333	0.5
knn	0.5161	0.5161

DISCRETIZE - REBALANCE		
training set	gbm	xgboost
random forest	0.5319	0.5532
naive bayes	0.4706	0.6176
knn	0.6667	0.619

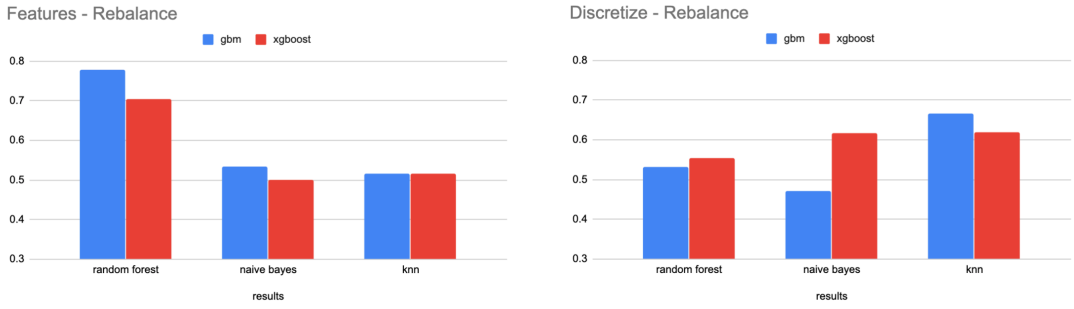


Figure 4.25: Comparison of performances between GBM and XGBoost.

Thus, the performance of the two algorithms are really close to each other in almost the cases. In fact, calculating the average, we have 58.27 for GBM and 58.49 for XGBoost. Moreover, since the latter always maintains an accuracy greater than 0.5, we chose it.
All in all, for the Features - Rebalance and Discretize - Rebalance cases, the meta-learners built on top of the experiment results are XGBoost. We have not used any Imputation, Rebalancing or other Pre-processing techniques. However, it is worth mentioning that whenever an instance is classified as a *no_order*, the actual assigned class will be the majority one.

4.3 Online phase

Now, we are going to explain how the bricks built in the offline phase are put together to achieve our goal. In particular we describe how to derive the data pipeline prototypes (4.3.1) and then how we optimize them (4.3.1).

4.3.1 Data pipeline prototypes building

We recall that we broke down this step in two phases:

- Understand the best order between single transformations, therefore build the Dependency Table;
- Derive the data pipeline prototypes.

Concerning the first one, we already know that almost the whole table is static. Indeed, after the SMBO experiments, regardless of the data-set and the used algorithm, the Dependency Table looks like the one in Figure 4.26.

Legend	
Color	Description
	Row first
	Column first
	Still not clear
	Not compatible or does not make sense

	Normalize				
Encode		Encode			
Discretize			Discretize		
Impute				Impute	
Rebalance					Rebalance
Features					

Figure 4.26: Table of constraints after the SMBO experiments.

Although, to complete it, we have to assign an order to the cells corresponding to the Features - Rebalance and Discretize - Rebalance. This would mean use the meta-models that we have built through the meta-learning process that we described above. In practice, we need just to extract the meta-features from the input data-set and give it to the right meta-models. In fact, for both Features - Rebalance and Discretize - Rebalance, the Data Mining algorithm we want to use selects the meta-model between the ones available. In each case we have three models, each one related to a Data Mining algorithm: Random Forest, Naive Bayes, K-Nearest Neighbor.

Once the Dependency table is completed, we are able to build the desired data pipeline prototypes. Indeed, the Dependency table is nothing more than a set of rules. In Figure 4.27, those rules are represented through the BPMN∗ (Business Process Model and Notation) graphical representation.

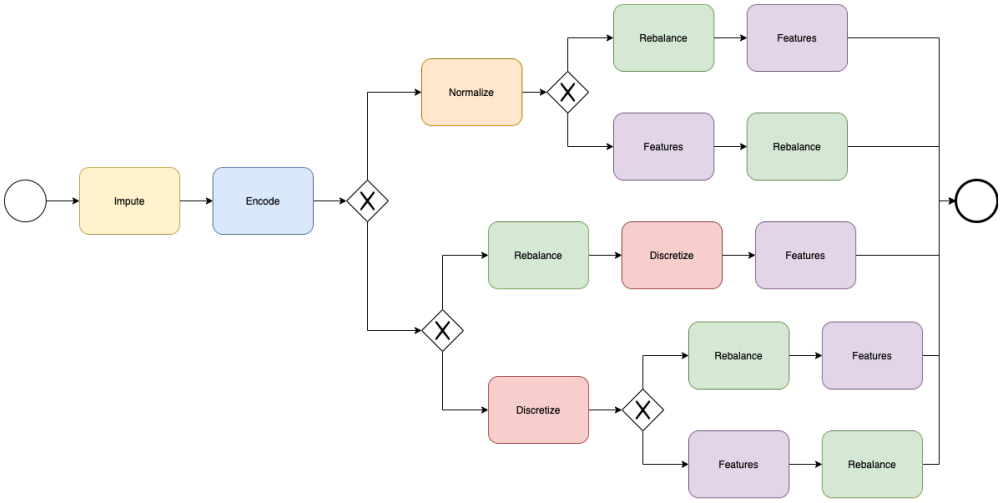


Figure 4.27: BPMN scheme representing the possible data pipeline prototypes.

As we expected, due to the compatibility constraints of the Scikit-learn framework, the Imputation and Encoding steps are at the beginning of the pipeline. Next the first decision taken, due to the only red cell in our table, was not to put Normalization and Discretization in the same pipeline (the combination does not make sense if applied in the same pipeline). As a matter of fact, two different branches emerge, one where Normalization is present and the other where Discretization is present. Since we do not know which choice is better, we should test both of the resulting pipelines. Proceeding with the one that includes the Normalization step, we have to understand just the order between the Rebalance and Features Engineering transformations. Instead, ragarding the other one, we have to understand also the relation between Discretization and Rebalance. As a matter of fact, in the Normalization

∗http://www.bpmn.org/

path, there can never be a conflict, since the relation to clarify is only one. In contrast, in the other path, the choices of the two meta-models could raise a conflict:

- The meta-model of Features - Rebalance could predict that Rebalance should be applied after Features;
- The meta-model of Discretize - Rebalance could predict that Rebalance should be applied before Discretize.

In all the other combinations there is no conflict, but if that happens, we have to explore both the options. Hence, the paths to test would raise to three.
Once we have generated those pipelines, we are going to optimize them.

4.3.2 Data pipeline prototypes optimization

With regards to the optimization of the obtained data pipeline prototypes, we just run the SMBO algorithm on each one of them. As we already discussed, the code of Quemy has been adapted to cover more generic cases:

1. More transformations;
2. Mixed data-sets, therefore with numerical and categorical attributes;
3. Transformations applied just to the compatible features.

In that way, any kind of data-set is supported and the prototypes we proposed can be ingested and optimized. Specifically, given a time budget, we need to split it between the pipelines to test. The difference between us and the approaches with fixed pipelines is that the latter spend their entire budget on a single pipeline. It is optimized very well, but by not exploring other orders of transformation combinations, it is not possible to know if that choice is the right one. Moreover, our approach allow us to not explore all the combinations but just the most promising ones, because we build those pipelines with previous knowledge, acquired through the SMBO experiments and the meta-learning process.
In the end, we compare the best performances reached by the optimization of the different pipeline prototypes and we retrieve the best one. The final outcome is, according our approach, the best data pipeline for the given data-set and algorithm.

5 Evaluation

In this chapter we provide the results of some evaluation experiments. Firstly, we made a study regarding the importance of the Pre-processing step. We compare the performances reached by optimizing the data pipeline with the ones achieved by optimizing the algorithm hyper-parameters. Subsequently, in order to demonstrate the validity of our approach, we compared it other works from the state of the art. In the following we illustrate the results and discuss the insights obtained.

To understand how much the data pipeline is significant, we performed some tests. We used the same pool of data-sets and the same set of Data Mining algorithms utilized in the SMBO experiments. Our aim is to understand when it is more convenient to optimize the Pre-processing pipeline and when the algorithm. Hence, for each data-set and algorithm, we ran SMBO on both phases. Then, we compared the two achieved performances and declared the winner.

Considering that we have used the same data-sets and the same algorithms to build also the meta-learners, we need to be careful because, otherwise, we could overestimate the performances. In particular, we cannot feed a meta-learner with data-sets already seen in training phase. For this reason, we used the meta-learners built during the Leave One Out validation. In other words, each data-set was fed to the meta-learners which have seen it just in the evaluation phase.

The optimization of the Modeling phase simply consists in assigning the entire time budget to a single SMBO run. Instead, in the Pre-processing phase, the budget must be divided into several runs. In fact, for that optimization, we used our approach, which involves testing at least two pipelines. We recall that the best data pipeline, and hence the best result, is found by comparing the Accuracies of those tested pipelines.

Figure 5.1 shows, grouping by the used Data Mining algorithm, the amount of data-sets where it was more convenient to optimize the Pre-processing pipeline and where the algorithm. Specifically, for a data-set, the most convenient optimization is the one that reached the highest Accuracy. For sure, an important insight is the clear value of the Pre-processing phase. Indeed, in many cases, it is better to spend the SMBO time budget on optimizing the data pipeline rather than the algorithm. We can also note that the choice of the used Data Mining algorithm affects a lot. In fact, for Random Forest the hyper-parameter optimization is more important than in the other cases. This is due to the fact that the Decision Trees are very robust to features' correlations, missing values and so on; hence they require much less Pre-processing. Furthermore, Random Forest has many hyper-parameters that influence its training. On the contrary, K-Nearest Neighbor and Naive Bayes are very sensitive to the data and have really few hyper-parameters to set.

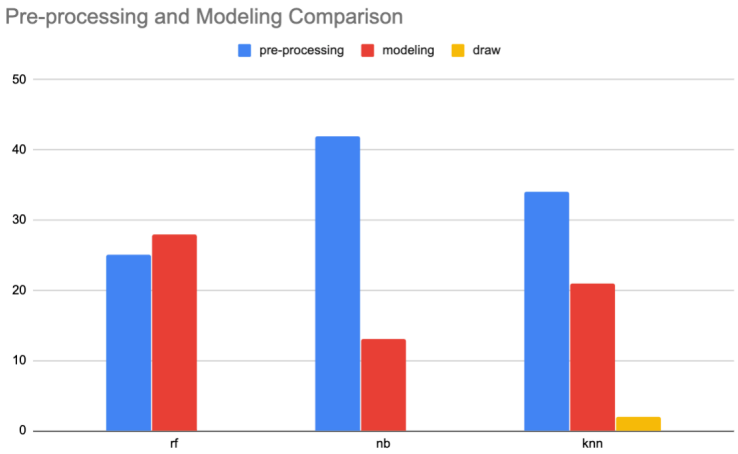


Figure 5.1: Comparison between Pre-processing and Modeling optimizations.

	Pre-processing	Modeling	Draw
Random Forest	25	28	0
Naive Bayes	42	13	0
K-Nearest Neighbor	34	21	2
Summary	101	62	2

Table 5.1: Comparison between Pre-processing and Modeling optimizations results.

The Table 5.1 reports the exact numbers of the graph and provides a summary.
We also wondered if there is something in the data-sets that led them to achieve better Accuracy by optimizing the algorithm rather than the Pre-processing pipeline, or vice versa. For this purpose, for each algorithm, we listed the data-sets where optimizing it was more convenient. Then, we counted those in common. If most of them were repeated, it would mean that there are some characteristics in the data-sets themselves that distinguish them and categorize them as such, regardless the used Data Mining algorithm. Among the 62 such cases, we counted only 34 distinct data-sets. Each of these data-sets appears, exclusively, in:

- Just one of the algorithms: Random Forest, Naive Bayes, K-Nearest Neighbor;

- Two of the algorithms: Random Forest and Naive Bayes, Random Forest and K-Nearest Neighbor, Naive Bayes and K-Nearest Neighbor;
- In all of the algorithms.

In the absence of any phenomenon that binds this distribution, the data-sets should be placed uniformly between the above cases. Since there are seven, the expected frequency for each one would be $34 * \frac{1}{7} = 4.85$. That would mean to have:

- In average $4.85 * 3 = 14.57$ data-sets present in just one of the algorithms;
- In average $4.85 * 3 = 14.57$ data-sets present in two of the algorithms;
- In average 4.85 data-sets present in all of the algorithms.

We counted effectively the data-sets for each case and we we had: 15 data-sets in just one algorithm, 10 in common between two algorithms and the remains 9 in common between all of the three algorithms. Already, at a glance, we can see that the observed frequencies of the cases where the data-sets are common to all the algorithms are twice the expected frequencies. But, in spite of this first sight, we were interested to prove it through a statistical test. Hence, we performed the Chi-square test, used to test the validity of a claim (null hypothesis) that is made about a population using sample data. The alternative hypothesis is the one believed if the null hypothesis is concluded to be untrue. Through this test we compared the expected and the observed distributions and we could tell if the the observed one is commonly distributed or not. The test outcome reports that with a confidence of 0.917 the two distributions differ and, hence, the observed frequencies are significantly not commonly distributed. We can say that this is an indication that the results are not simply random but there is some specific feature that leads certain data-sets to achieve better performance by optimizing the Modeling phase.

5.2 Evaluation of the Automated Data Pre-processing approach

Regarding the evaluation of the goodness of our approach, we arranged the experiments in the same way we performed those of the Data Pre-processing importance. Again, we used the

same pool of data-sets and same set of Data Mining algorithms. The only difference is that we no longer care about the result given by the optimization of the Modeling phase. Indeed, we are going to compare our approach with others that, in their turn, aim to optimize the Pre-processing phase.
First of all, we were interested in comparing our approach with Quemy's work, or better, the effectiveness of the pipeline found by this work with the fixed pipeline proposed by his work. In fact, the original approach from Quemy cannot work with the data-sets we use. Hence, we are comparing our approach with the upgraded Quemy approach. Specifically, when we ran his approach, we still had the fixed pipeline "Normalize ⟶ Rebalance ⟶ Features", but, since we have also data-sets with categorical attributes, transformations like Normalization are not applied to all the attributes, but just to the compatible ones. Moreover, some data-sets have missing values, hence, we add a further step: Imputation as first transformation of the pipeline. Once the experiments were performed we collected the results (Figure 5.2).

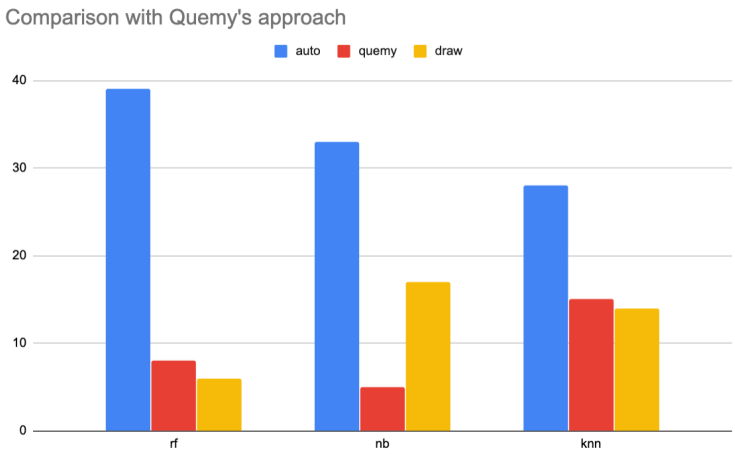


Figure 5.2: Comparison between our approach and the Quemy's one.

As we can see, our approach achieves better performances in most cases. A possible interpretation is that a fixed data pipeline prototype is not suitable for all the data-sets. Thanks to the knowledge acquired by the SMBO experiments and the data-set meta-features, we are

able, according to the case, to design the most promising pipelines. We have to split the budget between the different pipelines to test but we can effectively explore a wider search space than a fixed pipeline approach. Hence, the transformations order has proven to be more important than the parameter optimization. The cases where the Quemy's approach won are the ones where his proposed pipeline was effectively the best one and, evidently, the fact that his approach does not have to split the budget, allows to optimize it better. We can consider, instead, the draws as a success, because those cases are the ones where the our splitted time budget was enough to optimize the best data pipeline prototype, indeed by giving more budget to the Quemy's one, the performance did not improve.
In order to have a clear understanding of how much we improved the result we try to design an index, or score, that tries to express it. This, based on the Accuracy of the two approaches and the baseline (performance with no Pre-processing), is higher as much as the winning approach deviates from the baseline and as much as the losing approach is close to that. Hence, given the performances of the two approaches on a data-set with a specific algorithm, we denoted with *max* the winning approach's Accuracy and with *min* the one of the losing approach. The score is calculated as follows:

$$score = \frac{max - baseline}{min - baseline}$$

Figure 5.3 depicts the score in all the cases. In particular, in that figure the score was subtracted by 1, so that the draws have a score of 0, and, to have a better graphical visualization, the scores where the Quemy's approach wins have been made negative. With these graphs we can note that, we do not only win in most of the cases, but also when we do, we do it by making much more improvement than when the Quemy's approach wins.
Besides, we tried to understand the differences between the pipelines proposed by the two approaches. Since we already know the prototype of all the pipelines proposed by the Quemy's approach, because it is always fixed, we analyzed the prototypes that, in our approach, have prevailed. Specifically, we observed the cases where our approach won. In that way we could understand what was the factor that led us to success.

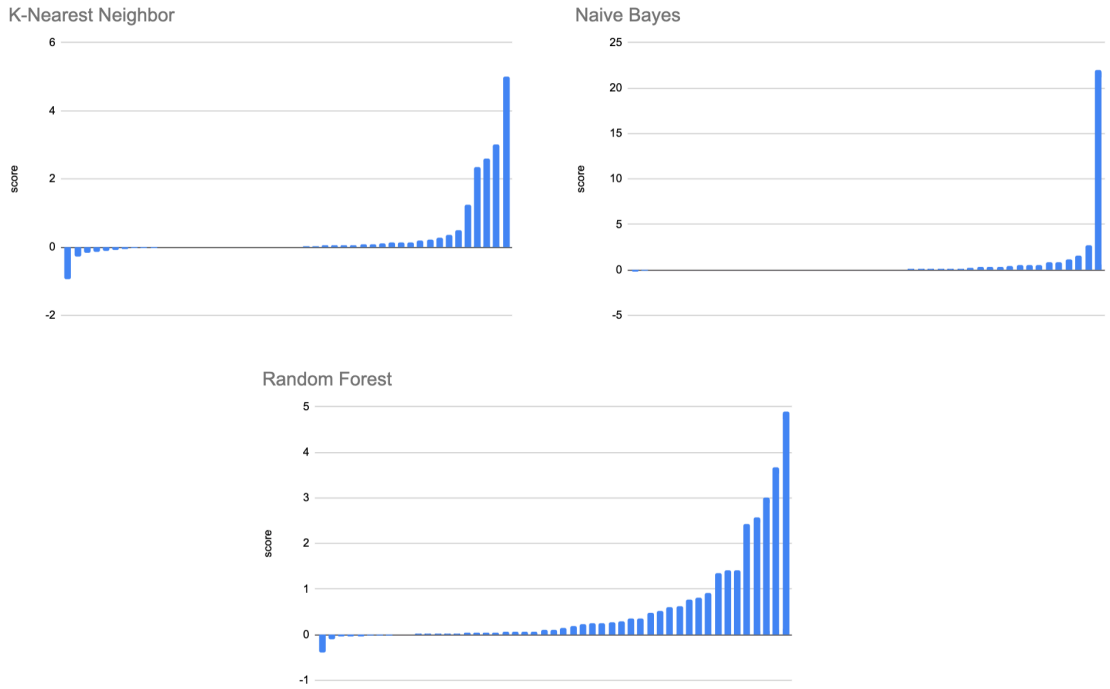


Figure 5.3: Estimation of how much the winning approach improves the result.

First of all, we recall that Quemy's pipeline prototype includes the Normalization step, instead, our approach tries to test also the prototype with the Discretization (see Figure 4.27 for the pipelines tested by our approach and Figure 2.11 for the Quemy's one). Regarding the cases won in our favor, Figure 5.4 shows when the Discretization transformation was part of the pipeline:

- Not present in the prototype (*not_in_prototype*);
- Present in the prototype but not in the final pipeline (*not_in_pipeline*), i.e. when SMBO has preferred to put the *None* operator in the step in question;
- Present in the prototype and also in the final pipeline (*in_pipeline*).

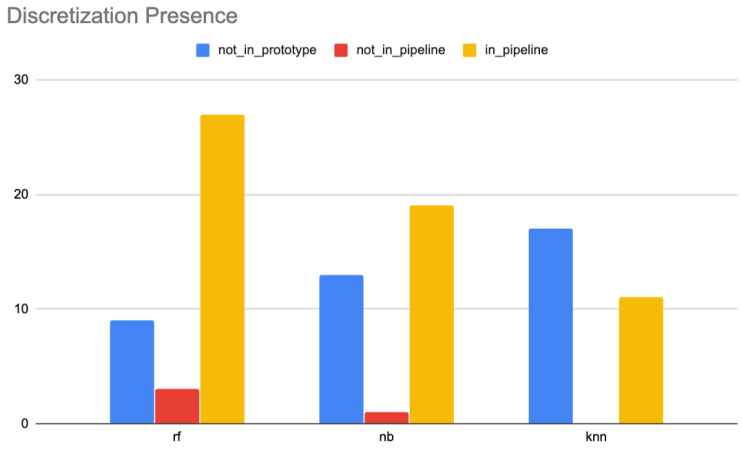


Figure 5.4: Discretization transformation's role in our pipeline.

We have a good part of the results that does not contain the Discretization step in the prototype and, hence, the substantial difference between the two proposed pipelines lies in the order. We counted 39 such cases, in a total of 100 victories in our favor. We can also see that, generally, when the Discretization prototype is chosen, SMBO instantiates this transformation with an operator different than the *None* one. This insight shows that our approach is actually understanding when a transformation is needed and when not. In the rest of the cases, 57 out of 100, the Discretization appears in both the prototype and the final configuration chosen by SMBO. Even if those cases are the majority, it cannot be said that the inclusion of the Discretization is the factor that makes us win. Before applying the meta-models, according to Figure 4.27, the number of pipelines that should be tested is five, of which three contain Discretization. We can say that it is reasonable that, more or less, 60% of the pipeline results contains Discretization inside.

Since our approach is not testing all the five pipelines, we were interested to compare our results with what we called *the Pseudo-exhaustive approach*, therefore the one that does. In order to remain fair to the other approaches, once again, the budget has been divided on the number of the tested pipelines. Regarding the Pseudo-exhaustive approach, that means dividing the budget between five. Figure 5.5 shows the results.

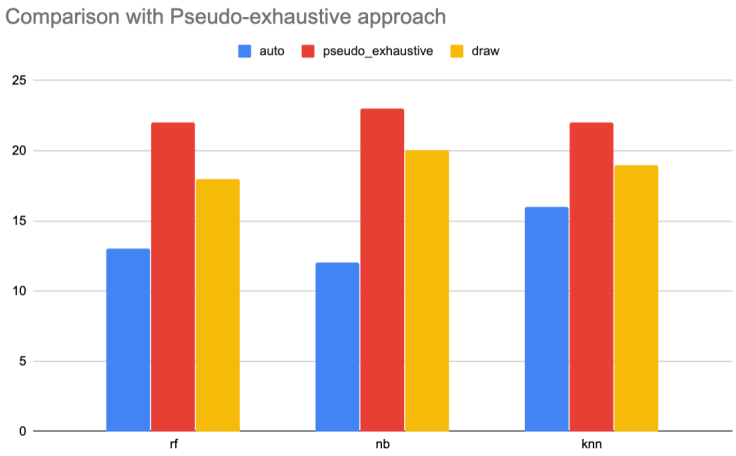


Figure 5.5: Comparison between our approach and the Pseudo-exhaustive one.

As it can be noted, we have the same result pattern with all the considered algorithms. The Pseudo-exhaustive approach won in most of the cases but, we do not consider this results as bad. Indeed, comparing an approach with one that tests all the possibilities and thinking of winning is against the odds. We already know that our meta-learning process has not 100% of Accuracy and, even when we are predicting correct, the Pseudo-exhaustive approach is also testing the correct pipeline. The only chance we have to win is that, since the Pseudo-exhaustive has to test more pipelines, the time given to the right one is not enough to find a solution as good as ours. After all, we can be satisfied given that in 165 cases we scored 41 victories. Moreover, we drew 57 times. Those are the cases where we found the optimal pipeline but the time we gave to optimize it was too much, and the Pseudo-exhaustive was able to find that solution too. Hence, we cannot consider those cases as negative, but instead successful. For sure, winning was not easy, but still it means that we should improve the meta-learning process. Anyway, looking on the bright side, those cases remark how much the order is important and how much the given time budget is secondary. In fact, even if we had more time to optimize the pipelines, we were optimizing the wrong ones. The pseudo-exhaustive, instead, had less time but had the possibility to optimize the right one, which was never pruned.

6

Conclusions and future developments

All in all, we are satisfied with the work that we have done. As has been argued in the chapter 2, Towards AutoML, the application of Machine Learning in real-case problems is a complex process that requires assistance if the user is not familiar with the techniques he has to work with. Moreover, in that section it has been seen that often the data with which the user has to deal is not adequate or algorithms do not perform best. Unfortunately, the current tools that aim to automate the Machine Learning process do not properly take care of the automation of the Pre-processing phase, which is in charge of transforming the data before the application of the Machine Learning algorithms. In fact, the few tools that consider it, simply apply the well-known SMBO optimization algorithm to a fixed Pre-processing pipelines, not considering that the transformations' order is very important to the performance. Our approach aims to find a methodology which recommends a Pre-processing pipeline taking into account the dependencies of the transformations' and how the final performances vary according to their order in the pipeline. Through the evaluation experiments we have demonstrated:

- The actual importance of the pre-processing phase;
- The success, in terms of performance, of our approach on a fixed pipeline approach.

In spite of the positive results, these experiments also show some improvement points. In particular, although the meta-learning approach can be considered effective, it has been noted that there is still room for improvement. Indeed, the comparison of our approach with the Pseudo-exhaustive one, which also considered the pipelines we excluded, did not have the best results. In future developments it would be interesting to be able to improve the built meta- models through:

- The execution of more experiments with a wider suite of data-sets and a larger set of algorithms;
- Experiments that consider not only dependencies between pairs of transformations.

Furthermore, given the numerous interesting insights obtained from the evaluation experiments, it would be worth to understand what are the characteristics that lead a data-set to achieve better performance by spending more time optimizing the algorithm rather than the Pre-processing pipeline, and vice versa.

References

[1] Bhaowal, M. (2018). Automl: The assembly line of machine learning. *DataEngConf*.

[2] Bilalli, B., Abelló, A., Aluja-Banet, T., & Wrembel, R. (2018). Presistant: Learning based assistant for data pre-processing. *Data Knowledge Engineering*.

[3] Brochu, E., Cora, V. M., & de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning.

[4] Brodersen, K. H., Ong, C. S., Stephan, K., & Buhmann, J. (2010). The balanced accuracy and its posterior distribution. *Pattern Recognition, International Conference on*, 0, 3121–3124.

[5] Chakure, A. (Jun 29, 2019). Random forest regression. *Towards Data Science*.

[6] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321–357.

[7] Crone, S., Lessmann, S., & Stahlbock, R. (2006). The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, (pp. 781–800).

[8] De Sá, A., Pinto, W., Oliveira, L. O., & Pappa, G. (2017). Recipe: A grammar-based framework for automatically evolving classification pipelines. (pp. 246–261).

[9] Dhar, V. (2013). Data science and prediction. *Commun. ACM*, 56(12), 64–73.

[10] Dietterich, T. G. (1997). Machine-learning research. *AI Magazine*, 18(4), 97.

[11] Drakos, G. (16 Aug 2018). Cross validation. *Medium*.

[12] Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural architecture search: A survey.

[13] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. (pp. 2962–2970).

[14] Frankel, S. (May 22, 2015). Data scientists don't scale. *Harvard Business Review*.

[15] Guinard, D. & Trifa, V. (2009). *Towards the Web of Things: Web Mashups for Embedded Devices*, (pp. 1506–1518).

[16] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3), 515–516.

[17] Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7), 2561 – 2573. Special Issue on Perspectives on Parallel and Distributed Processing.

[18] Kelleher, J., Mac Namee, B., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies.

[19] Knagg, O. (Jan 15, 2019). An intuitive guide to gaussian processes. *Towards Data Science*.

[20] Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Data preprocessing for supervised learning. *International Journal of Computer Science*, 1, 111–117.

[21] Kotthoff, L., Thornton, C., Hoos, H., Hutter, F., & Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18, 1–5.

[22] Kraska, T., Talwalkar, A., J.Duchi, Griffith, R., Franklin, M., & Jordan, M. (2013). Mlbase: A distributed machine learning system. *Conference on Innovative Data Systems Research*.

[23] Lowd, D. & Domingos, P. (2005). Naive bayes models for probability estimation. (pp. 529–536).

[24] Lu Tan & Neng Wang (2010). Future internet: The internet of things. 5, V5–376–V5–380.

[25] Marr, B. (2019a). Coca-cola: Driving success with ai and big data. *Bernard Marr*.

[26] Marr, B. (2019b). The incredible ways heineken uses big data, the internet of things and artificial intelligence (ai). *Bernard Marr*.

[27] Marr, B. (May 21, 2018). How much data do we create every day? *Forbes*.

[28] Misauer, L. (October 5, 2017). Iot, big data and ai – the new 'superpowers' in the digital universe. *Business2Community*.

[29] Mulvenna, M., Norwood, M., & Büchner, A. (1998). Data-driven marketing. *Electronic Markets*, 8(3), 32–35.

[30] Olson, R. & Moore, J. (2019). Tpot: A tree-based pipeline optimization tool for automating machine learning. (pp. 151–160).

[31] Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, 4(2), 1883. revision #137311.

[32] Quemy, A. (2019). *Data Pipeline Selection and Optimization*.

[33] Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4), 10–17.

[34] Sparks, E. & Talwalkar, A. (2013). Mli: An api for distributed machine learning. *International Conference on Data Mining*.

[35] Talwalkar, A. & Kraska, T. (2012). Mlbase: A distributed machine learning wrapper. *Big Learning Workshop at NIPS*.

[36] West, Jeremy; Ventura, D. W. S. (2007). Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*.

[37] Zhang, J. & Mani, I. (2003). KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction.