UNIVERSITY OF MILAN

DEPARTMENT OF COMPUTER SCIENCE

MASTER'S DEGREE IN COMPUTER SCIENCE

MASTER THESIS

MACHINE LEARNING METHODS TO PREDICT DEBTORS SOLVENCY

Supervisor:
Prof. Giorgio Valentini

Co-Supervisors:
Dr. Luca Cappelletti
Dr. Fabrizio Trovato

Author:
Vittorio Triassi
ID. 938344

ACADEMIC YEAR 2019 - 2020

Index

Acknowledgments

At the very beginning of this dissertation, I would like to thank those without whom this work would not have been possible.

First of all, I would like to acknowledge my supervisor for his guidance and attention to detail and my co-supervisors for their advice and availability throughout the whole project.

I would like to thank my girlfriend and my friends for all the support and affection shown.

And finally, my biggest thanks goes to my family. To them, I owe everything.

Abstract

Credit risk assessment and debt solvency are well-known issues for credit institutions, which have to regularly evaluate credit applications trying to reduce as much as possible their financial exposure in the event that a loan is granted. In this context, Machine Learning plays a key role, since being able to make predictions on an individual or a company creditworthiness is crucial. However, debt solvency is a key issue also for debt collection agencies, and in this specific context, this problem is more challenging from a Machine Learning standpoint. In fact, while credit institutions and financial companies have access to all the information they need to understand whether or not an applicant will be able to pay their debt back, the access to relevant information is more difficult or even unfeasible with debt collection agencies.

Focus of the following dissertation is exactly the problem of debt solvency related to debt collection agencies. Most of the studies on the problem of debt solvency, mainly focused on what is known as Financial Credit Risk (FCR). FCR is usually related to banks, and represents the risk of default coming from a debtor who does not pay their debt back. To our knowledge, there is no literature for debt solvency with debt collection agencies, and since the two problems share some similarities, in this thesis we will investigate some of the methods proposed for FCR to address the debt collection problem, which is a more challenging prediction problem, due to the reduced information available for the prediction. This problem can be formalized as a binary classification. The positive class, labels the debts that will likely be paid, and the negative one, just the opposite. We defined two main prediction problems: a partial and a full debt solvency. Three different supervised Machine Learning methods are proposed: tree-based models like Decision Tree and Random Forest which allow for a higher explainability, and a neural network model. Results show that Random Forests outperform the other models. Random Forests further tuned with the Bayesian Optimization to identify the optimal sets of hyperparameters, achieve an accuracy of about 72.15% for the partial and 76.80% for the full debt solvency using multiple holdouts. The features used for the prediction showed individually no significant correlation with the output variable. However, a ranking of these was extracted in order to understand which are the most relevant.

The whole dissertation is divided into five chapters. In Chapter 1, there is a brief introduction whose aim is to give an overview on debt solvency which differs from FCR. In Chapter 2, data cleaning, encoding and visualization are covered in detail. In Chapter 3, a wide discussion on the Machine Learning models and feature selec-

tion techniques is done along with the importance of performing model selection. In Chapter 4, experimental results on the tests carried out are shown and motivated. Finally, in Chapter 5, conclusions are drawn. To our knowledge, this is the first work that tries to predict debt solvency to support the decisions of a debt collection agency.

Chapter 1

Introduction

The following thesis aims to shed some light on the problem of *debt solvency*, in which some insights with the aid of Machine Learning models are meant to be gained. The reason why these models are used, is largely because of their great ability to generalize over several problems. The debt solvency problem is formalized as a classification task, in which learning algorithms learn how to label two or more classes. In this very specific case, a binary classification task will be addressed. We tackle the problem of debt solvency through a pipeline that allows to evaluate data on debtors and eventually produce a prediction expressing their creditworthiness in output. Two classes are therefore defined: a positive and a negative one. The positive class represents a debtor who paid their debt back. The negative one, just the opposite. All this work has been carried out in collaboration with a debt collection agency based in Northern Italy, which has provided us with a considerable amount of data on their dossiers. These dossiers can be transformed into records made of several attributes that identify the debts and the individuals or the companies who run into them, which will be eventually fed to the Machine Learning models.

1.1 Financial credit risk vs Debt collection

Something that it is worth to point out is that the following thesis is not meant to be a review on a quite widespread topic known as *Financial Credit Risk* (FCR). When trying to evaluate FCR, it is possible to identify two kinds of problems:

1. *Credit rating*, usually expressed through grades and aims to assign the creditworthiness of a business.

2. *Bankruptcy prediction*, which aims to predict whether or not a firm is likely to go bankrupt.

Credit risk can be summarized as the risk of default coming from a debtor who does not pay their debt back. The first reviews on FCR found in the literature are mainly based on statistical models [8, 11]. Recently, the scientific community developed a deeper interest on new Machine Learning models [2, 12]. Although the FCR is different than the problem addressed here, which is more related to the credit collection, it was noticed that both problems share some similarities. Since Machine Learning models perform quite well on FCR, it was considered of interest to use them to carry out experiments on this task too.

To ensure that a substantial difference between FCR and debt collection is perceived, some more details should be provided. Financial Credit Risk is mainly referred to banks that have to figure out whether a client who asks for a loan will be eventually able to pay it back or not, while still ensuring very little risk to the exposure of the bank. As a consequence, the bank has all the powers to ask the client to provide all the necessary documentation to prove their creditworthiness. The client in fact, cannot afford not to do so, otherwise no loan is granted. On the other hand when talking of debt collection, the agencies do not have the same power and here is when a problem arises.

Debt collection agencies are usually not able to obtain large information about debtors, apart from what they already know on them. Moreover, several might be the aspects that remain unknown to them. For instance, they do not have access to too sensitive information and therefore there is very little they can do. For this reason, it would be helpful to investigate more on the data they already have stored and collected for years to understand whether any pattern among them can be recognized to improve their collection rates.

Most of the studies addressing the FCR problem in literature, carried out experiments on different types of methods which should be further discussed.

Statistical methods

Several have been the statistical methods used to address the FCR problem. It is worth to mention the Linear Discriminant Analysis (LDA), the Logistic Regression (LR) and the Multiple Discriminant Analysis (MDA). Despite these methods carry limitations that neglect the complex interactions of financial variables, they are still quite popular in some of the most famous rating agencies. Just recently, linear and generalized additive models to predict the default risk in German companies, were proposed [15].

Artificial Neural Networks

Artificial Neural Networks (ANNs) have been tested in several different architectures. Thanks to the great ability of ANNs to generalize over complex problems, they proved to be very effective in detecting financial patterns [5]. Despite ANNs are able to approximate any function, they also have weaknesses. ANNs in fact, lose in terms of explainability and it is not possible to easily understand what drove their final predictions. They can be imagined as black boxes in which the financial feature vectors are fed in input and predictions are returned in output. Recent studies have been proposing new methods that allow for an easier interpretation, and are part of what is known as Explainable Artificial Intelligence (XAI). Overall, ANNs applied to the problem of FCR have been quite effective and outperformed the traditional statistical methods [3].

Decision Trees

Also Decision Trees (DTs) have been applied on the context of the FCR prediction. Among the several advantages that they show, the most interesting are: easier results explainability compared to ANNs, reliability in the predictions, and the ability to handle both categorical and continuous features. There are multiple approaches to build the trees, and the most famous are ID3, C4.5, CART and MARS. More precisely, C4.5 and CART, proved to be quite effective when trying to address the prediction of business failure [14].

Ensemble methods

Ensemble methods are built by combining multiple classifiers that are individually trained and eventually aggregated to produce the final prediction [27]. The reason why ensemble methods have proved to be so successful has its roots in a trade-off between the accuracy of the single classifiers and their diversity. These methods have been used a lot in literature since they are able to obtain better performance compared to their base learners [25, 26].

1.2 Machine Learning models for the debt solvency problem

Very little is found on the debt collection problem and their applications using Machine Learning models. The majority of the studies carried out in fact, mainly refers to the Financial Credit Risk and the very few attempts towards debt collection did not produce significant results, which shows a lack of knowledge on this topic. As previously mentioned in Section 1.1, some similarities between FCR and debt collection are recognized and therefore it is of interest to try to investigate more on the problem of the prediction of debt collection using some of the approaches found in literature for FCR.

Both problems can be formulated as learning problems and hence Machine Learning models can be used. As far as the FCR problem is concerned, it is possible to start from a dataset including data on debtors labeled as good or bad creditors, with several financial attributes that describe the debtor's condition. By using learning algorithms, it is possible to predict the likelihood that a debtor will be a bad creditor or, when talking about companies, if these will go bankrupt. This problem belongs to the family of the Supervised Learning algorithms since data are already labeled with the classes that have to be identified. Similarly, it is possible to approach the debt solvency problem. Also here, the starting point is a dataset built upon debtors data and several attributes available on them. Of course, the amount of attributes, that in Machine Learning are called features, is far below the one available for banks or other credit institutions and this raises the question whether reliable predictions can still be obtained.

So far, an overall picture of the problem was given. In real applications, the predictive process needs a very complex pipeline that has to be properly built. The steps involved in a Machine Learning project, whose aim is to extract business insights and make predictions on a specific task, are several. First of all, it is important to collect the data and make them “computer readable”. After data acquisition is done, we need to understand whether or not there are features that introduce bias in the data. If this is the case, these features should be promptly dropped to avoid too optimistic predictions and the model would not correctly generalize over the task. Right after, data preprocessing is necessary since data are generally “dirty” and things like taking care of missing values, feature scaling, encoding and spotting relationships among the data, might be crucial for the success of the predictions produced by the learning models. Then, feature selection has to be performed to understand which are the features that most positively impact on the models performance. Even if it is not this the case, feature selection may help reduce the dimensionality of the dataset burdening less on the model training. Selecting a subset of features among their whole list may also be beneficial to remove the ones that introduce noise in the models. Once these steps are performed, the Machine Learning models can be trained and eventually evaluated on several metrics.

A wide discussion on Machine Learning models, that are adapted to the debt solvency problem, will be covered in Chapter 3. Finally, a choice among the models used should be made. This is what is known as model selection, in which the model that best performed over the others is selected and the set of its hyperparameters is properly tuned with the ultimate goal of producing the optimal model used to make the final predictions.

Chapter 2

Data construction, preprocessing and visualization

2.1 Dataset

As far as the dataset, its first version was made up of approximately 650k rows and 18 columns. Each row refers to a single dossier that involves a particular debt. For each dossier there is access to some information on the debtor. Then, new columns became available and these were appended to the existing ones, resulting in 22 columns this time. The format used to store the dataset is the comma-separated values (csv) that comes in handy with libraries that are later presented. Let us now give a look at the list of features available. These give an idea about the information available for each dossier.

2.1.1 Features

1. `IDENTIFICATIVO_INTERNO`: internal ID
2. `codice_fiscale`: fiscal code or VAT number
3. `cap`: ZIP code
4. `comune`: municipality name
5. `ente`: name of the body to whom the dossier is assigned
6. `numDatoriLavoro`: number of employers
7. `numContiCorrenti`: number of bank accounts
8. `numImmobili`: number of properties owned
9. `numMezzi`: number of vehicles owned
10. `valMezzi`: value of vehicles owned
11. `avvisi_anno_minimo`: year in which the dossier is created by the body

12. `AVVISI_ANNO_MASSIMO`: year in which the dossier stops being under the responsibility of that body
13. `avvisiDurata`: difference (in years) between `AVVISI_ANNO_MASSIMO` and `avvisi_anno_minimo`
14. `AVVISI_CONTEGGIO`: number of notifications the debtor receives in the range of years (`avvisi_anno_minimo - AVVISI_ANNO_MASSIMO`)
15. `AVVISI_PRODOTTO`: type of debt, generally local taxes (i.e. garbage, water, etc), to be paid. There are several of them and they are mapped as shown below. The mapping is provided by the debt collection agency.
    - `RISC. COATT. TASI: TASI`,
    - `TASI: TASI`,
    - `RISC. COATT. ICI: IMU`,
    - `ICI: IMU`,
    - `IMU: IMU`,
    - `RISC. COATT. IMU: IMU`,
    - `TBON: TBON`,
    - `TRIBUTO DI BONIFICA: TBON`,
    - `TIA: TARI`,
    - `TIA-G: TARI`,
    - `CODSTR: CODSTR`,
    - `RISC. COATT. CDS: CODSTR`,
    - `SANZ.:  CODSTR`,
    - `RISC. COATT. SERSCO: SERSCO`,
    - `SERSCO: SERSCO`,
    - `TARI: TARI`,
    - `TARI-G: TARI`,
    - `RISC. COATT. TARI: TARI`,
    - `RSU: TARI`,
    - `RSUG: TARI`,
    - `VRSU: TARI`,
    - `TARES: TARI`,
    - `TARES-G: TARI`,
    - `TIAP: TARI`,
    - `CIMP: PUBBLICITA`,
    - `ICP: PUBBLICITA`,
    - `ACQUA: ACQUA`,

- `CANIDR: ACQUA`,
- `SII: ACQUA`,
- `SII-GORI: ACQUA`
- `CORRP: CORRP`,
- `CDEM: CDEM`,
- `CFIN: CFIN`,
- `CIRR: CIRR`,
- `CREG: CREG`,
- `TOSAP: TOSAP`,
- `AMB: AMB`,
- `TIRR: TIRR`,
- `CANTRS: CANTRS`,
- `CANDEP: CANDEP`,
- `GAS: GAS`,
- `FOREST.:  FOREST`,
- `AENTR: AENTR`,
- `CANASD: CANASD`,
- `COSAP: COSAP`

16. `anno_creazione`: year in which the dossier is taken over by the debt collection agency and is not under the responsibility of the original `ente` anymore

17. `annoPagamento`: year in which *any amount* of the debt (when the target variable is `pagato`) is paid, or the *whole amount* of the debt (when the target variable is `pagato_completamente`) is paid. `annoPagamento` represents the year in which such operation takes place

18. `durataPratica`: difference (in years) between `annoPagamento` and `anno_creazione`

19. `importoDovuto`: amount due (in euros)

20. `importoPagamento`: amount paid (in euros)

21. `FASE_RAGGIUNTA`: stage reached by the dossier. There are six of them and they are mapped as shown below. The higher the mapped value is, the more severe is the debt owed.

    - `SOLLECITO`: 1,
    - `INGIUNZIONE`: 2,
    - `INTIMAZIONE`: 2,
    - `PREAVVISO DI FERMO`: 3,
    - `FERMO AMMINISTRATIVO`: 4,

    - `PIGNORAMENTO`: 5

22. `valore_veicolo`: value of vehicles owned by a debtor (it replaces the column `valMezzi`). Value of vehicles are obtained through a basic algorithm that computes the current value of the vehicle taking into account its year of registration and the horse power.

2.1.2 Target variables

For our purposes, two prediction problems are defined:

1. **Partial debt solvency**: label → pagato.

$$pagato = \begin{cases} 1, & \text{if importoPagamento} > 0 \\ 0, & \text{otherwise} \end{cases}$$

2. **Full debt solvency**: label → pagato_completamente.

$$pagato\_completamente = \begin{cases} 1, & \text{if importoPagamento} = \text{importoDovuto} \\ 0, & \text{otherwise} \end{cases}$$

   Note that $pagato\_completamente = 1 \Rightarrow pagato = 1$ and $pagato = 0 \Rightarrow pagato\_completamente = 0$. The opposite implications do not hold.

2.2 Sanitization

The first step mainly involves a preliminary analysis on the input variables. It is important to understand if there are malformed values. When this is the case, it becomes necessary to start thinking about possible approaches to sanitize them. It turns out that a few variables have missing values, while others have outliers to be removed or typos.

2.2.1 Dropped input variables

After a careful selection, the variables identified as not relevant for the encoding part or that are carrying a bias which would make a future prediction too optimistic, are dropped. These variables are the following:

- `IDENTIFICATIVO_INTERNO`: it does not carry any useful information, it is a simple identifier
- `comune`: it gets dropped after is sanitized and validated with `cap`, it gets translated into geographical coordinates so it cannot be dropped from the beginning
- `cap`: it gets dropped after is sanitized and validated with `comune`, it gets translated into geographical coordinates so it cannot be dropped from the beginning
- `regione`: information on municipalities that is not used (yet). We might think about one-hot-encoding these values eventually

- `nazione`: information on municipalities that is not used (yet). It might become more useful when dealing with geographical heat maps. Geographical heat maps are a data visualization technique that aims to highlight areas with low or high density for a certain phenomenon
- `valMezzi`: replaced by `valore_veicolo`
- `annoPagamento`: it introduces a bias with respect to the target variables `pagato` and `pagato_completamente`
- `importoPagamento`: it introduces a bias with respect to the target variables `pagato` and `pagato_completamente`
- `durataPratica`: it introduces a bias with respect to the target variables `pagato` and `pagato_completamente`

2.2.2 Dropped records

Part of the sanitization involves basic checks such as making sure that the min and max values for each variable are acceptable. If they are not, these records are dropped. More precisely:

- Since `durataPratica` comes from the difference between `annoPagamento` and `anno_creazione`, negative values are not expected. Thus, we drop all the records with `durataPratica` < 0. We also drop all the records with `durataPratica` > 12, as after a careful analysis, it turns out that it would exceed the range of values we think it is reasonable to keep for such attribute
- We drop the very few records with `importoDovuto` < 0, as it would be inconsistent with the dataset
- We also drop all the records with `comune` AND `cap` null, since no geographical information from these records can be extracted
- After a careful analysis on the context of our data, we drop all the values that exceed a reasonable range for the `AVVISI_ANNO_MASSIMO` variable. More specifically, we drop all the dossiers with `AVVISI_ANNO_MASSIMO` < 1994 (see Figure 2.1) OR `AVVISI_ANNO_MASSIMO` > 2019 (see Figure 2.2).

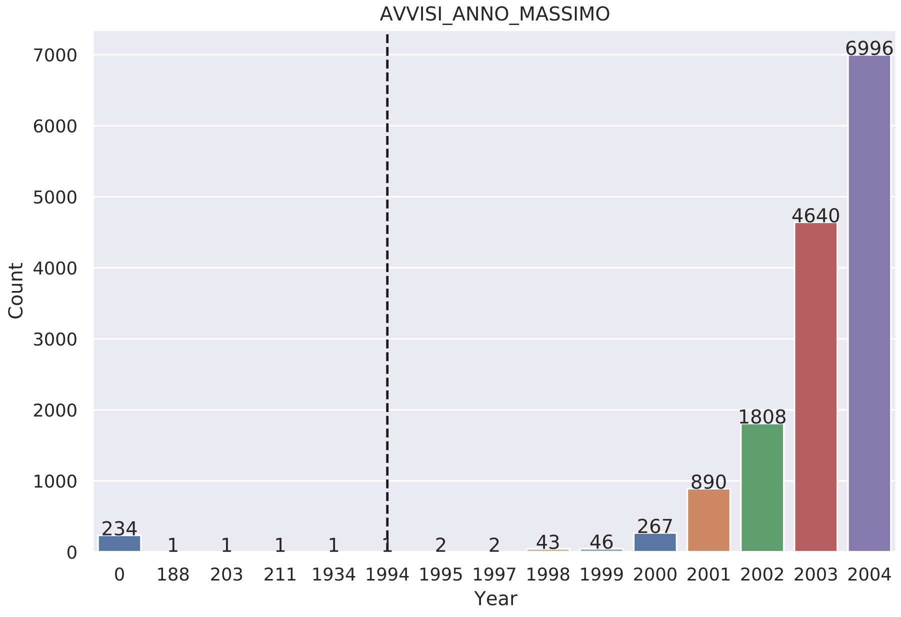


Figure 2.1: Minimum 15 values for the variable `AVVISI_ANNO_MASSIMO`.

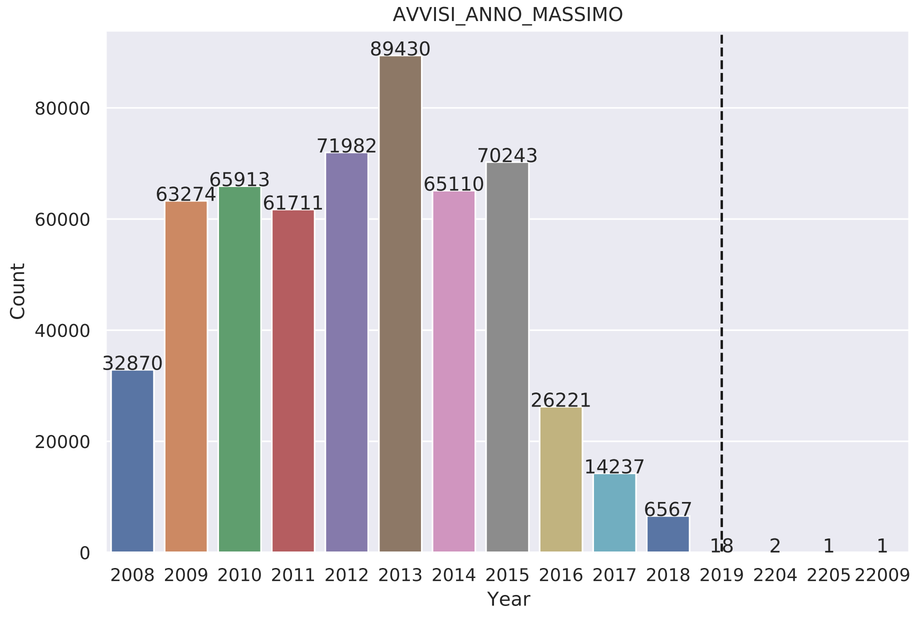


Figure 2.2: Maximum 15 values for the variable `AVVISI_ANNO_MASSIMO`.

- For the same reason, we take care of `avvisi_anno_minimo`. We drop all the dossiers with `avvisi_anno_minimo` < 1994 (see Figure 2.3) OR `avvisi_anno_minimo` > 2019 (see Figure 2.4).

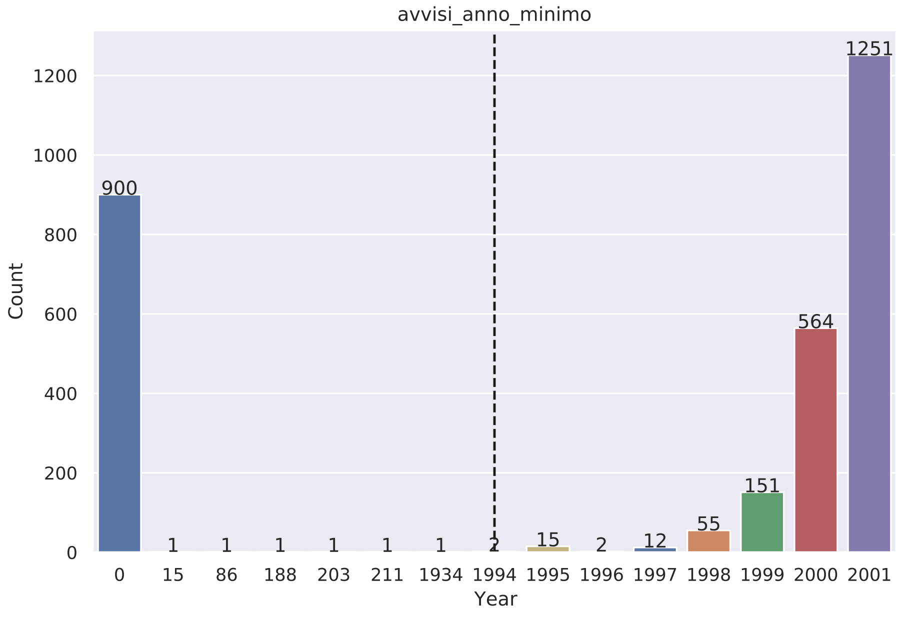


Figure 2.3: Minimum 15 values for the variable `avvisi_anno_minimo`.

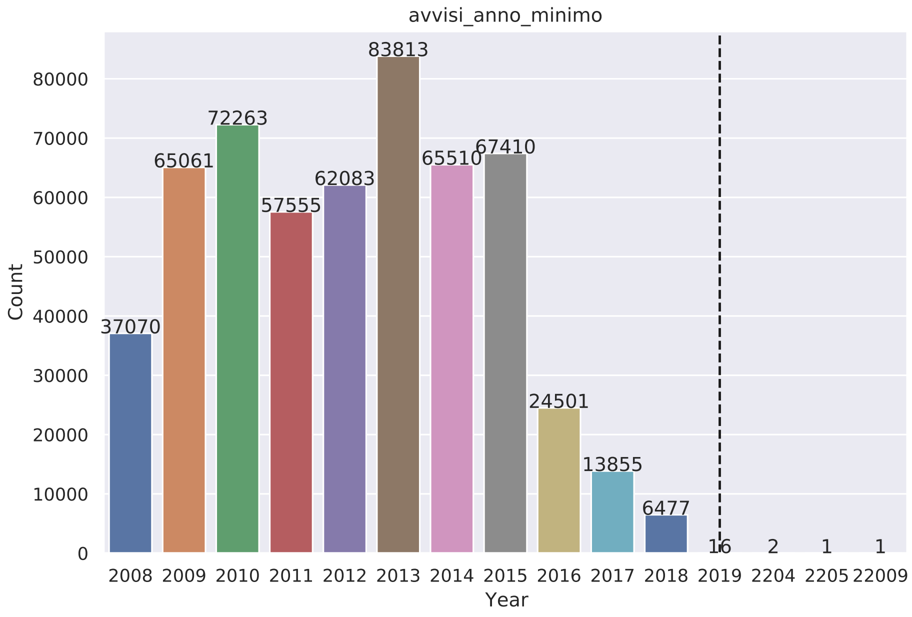


Figure 2.4: Maximum 15 values for the variable `avvisi_anno_minimo`.

- After a careful inspection, we also come up with dropping all the records with `anno_creazione` < 2005, as it seems that the debt collection agency starts receiving dossiers from other bodies just after this year, and the very few that show up before our threshold are clearly inconsistent with the dataset (see Figure 2.5)

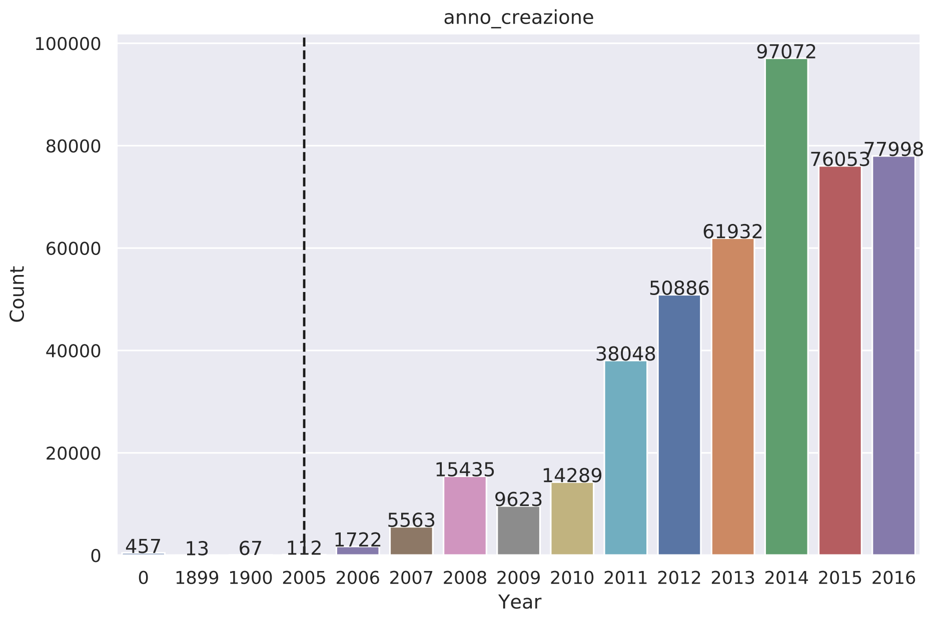


Figure 2.5: `anno_creazione` last 15 values displayed according their counts.

- Finally, all the records that have `AVVISI_ANNO_MASSIMO` $>$ `anno_creazione` and `avvisi_anno_minimo` $>$ `anno_creazione` get dropped, since these records would be inconsistent with their semantic meaning defined in Section 2.1.1.

2.2.3 Municipality and ZIP code

Most problems come from free-text variables such as `comune`. Other issues are encountered with the variable `cap` that has many invalid values. Our intuition suggests us to first sanitize the `comune` and `cap` variables and then to transform this information into geographical one, by converting the ZIP codes for our municipalities into *latitudes* and *longitudes*. In this way, we are introducing two more valuable features that hopefully increase the accuracy of our subsequent analyses and avoid the introduction of a huge number of variables through one-hot-encoding. In order to sanitize the two aforementioned variables, we take advantage of the Italian dataset provided by *pgeocode* [19] which stores the whole list of Italian ZIP codes with their municipality names and geographical coordinates. Since a few thousand records includes foreign municipalities or country names, we decide to make a mapping that maps these records to the capital of their countries. To do so, we use the *geopy* package [9].

2.2.4 Summary

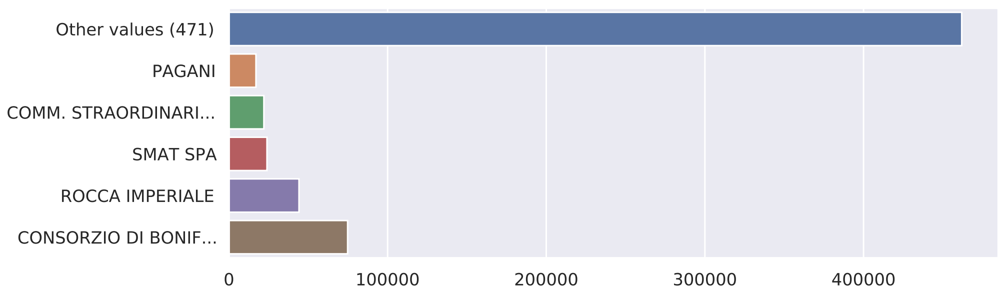
In Table 2.1, a summary for all the input variables after the sanitization is shown. For real and integer variables, we include their range of values and their mean. For categorical or ordinal values we report their number of unique distinct values.

Table 2.1: List of variables contained in the sanitized dataset. Target variables are presented in **bold**.

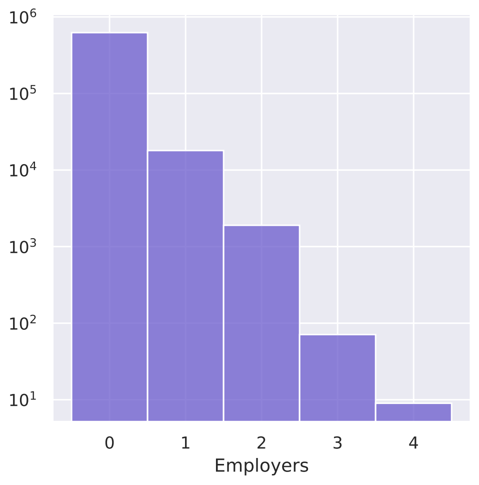
Variable	Range	Mean	Unique Values
ente	-	-	476
numDatoriLavoro	(0 - 4)	0.03	-
numContiCorrenti	(0 - 11)	0.18	-
numImmobili	(0 - 167)	0.21	-
numMezzi	(0 - 268)	0.47	-
avvisi_anno_minimo	(1994 - 2019)	2011.21	-
AVVISI_ANNO_MASSIMO	(1994 - 2019)	2011.44	-
avvisiDurata	(0 - 14)	0.23	-
AVVISI_CONTEGGIO	(1 - 185)	1.52	-
AVVISI_PRODOTTO	-	-	23
anno_creazione	(2006 - 2020)	2014.72	-
importoDovuto	(0 - 785513)	570.41	-
FASE_RAGGIUNTA	-	-	5
valore_veicolo	(0 - 7782600)	10641.88	-
lat	(-41.28 - 60.16)	42.57	-
long	(-121.09 - 174.77)	12.09	-
codice_fiscale	-	-	407869
pagato	**(0 - 1)**	**0.40**	**2**
pagato_completamente	**(0 - 1)**	**0.31**	**2**

2.2.5 Distributions

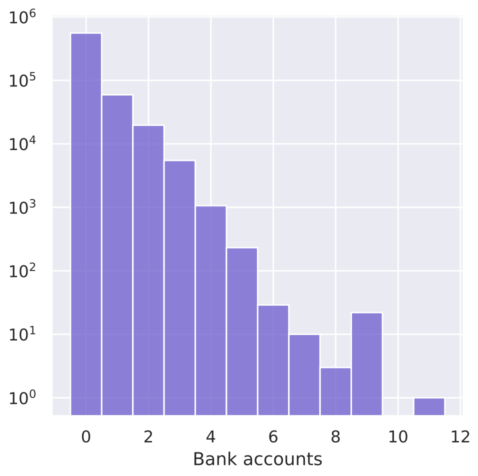
The following histograms show the distribution of the variables listed in Table 2.1.



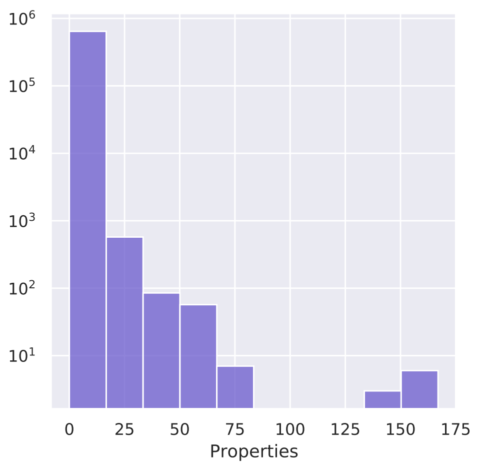
(a) ente



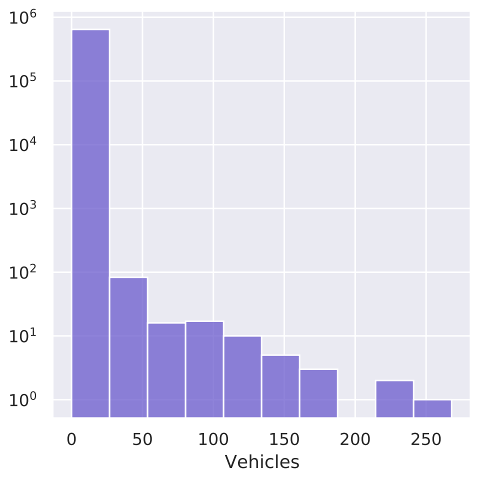
(b) numDatoriLavoro



(c) numContiCorrenti

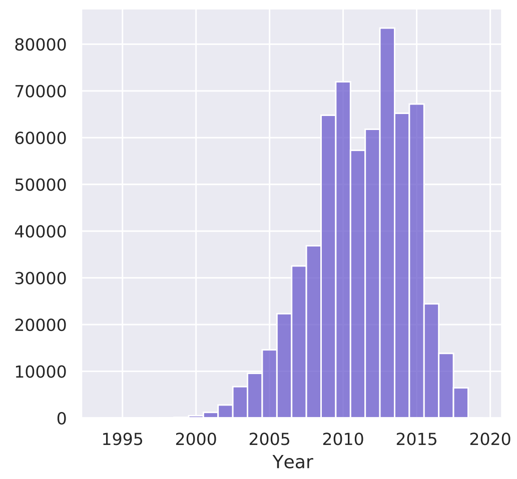


(d) numImmobili

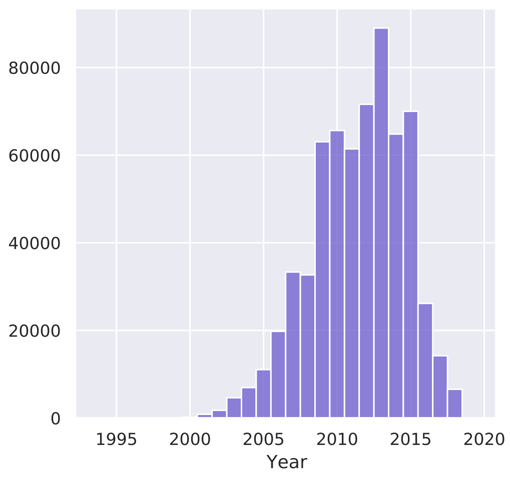


(e) numMezzi

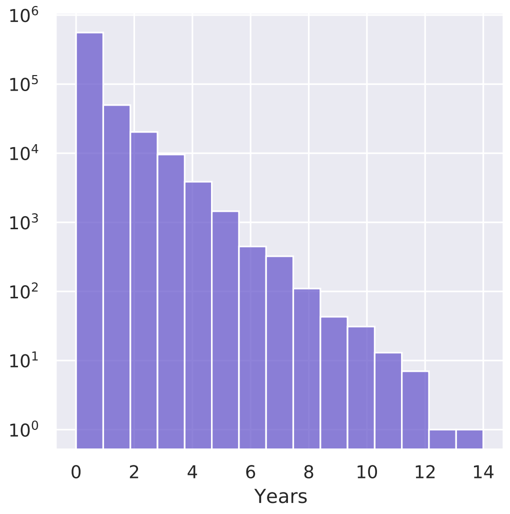
Figure 2.6: Overview of the variables 1-5 listed in Table 2.1.
Ordinate values are expressed in logarithmic scale.



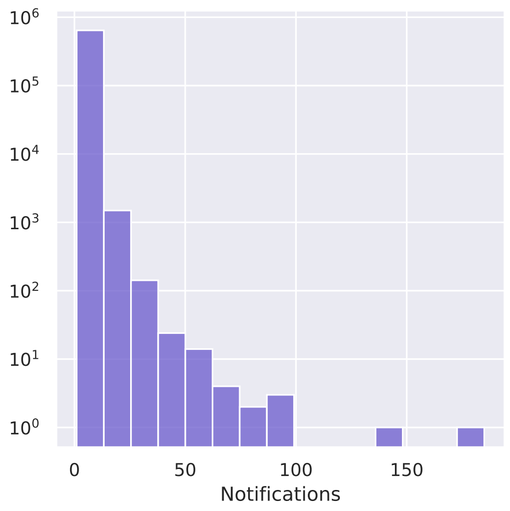
(a) avvisi_anno_minimo



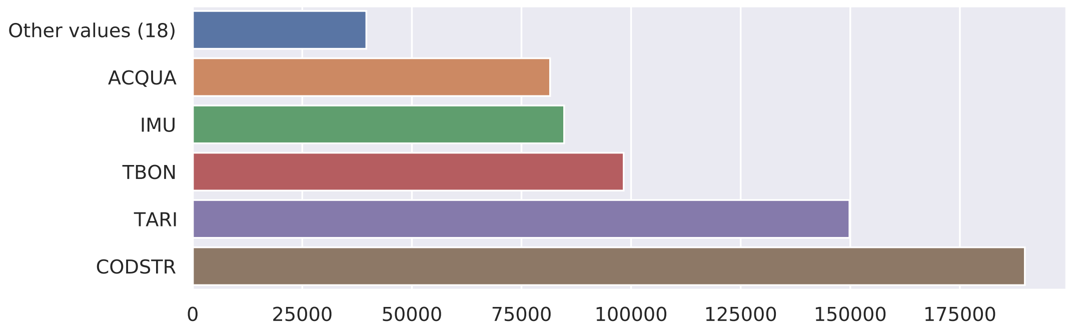
(b) AVVISI_ANNO_MASSIMO



(c) avvisiDurata

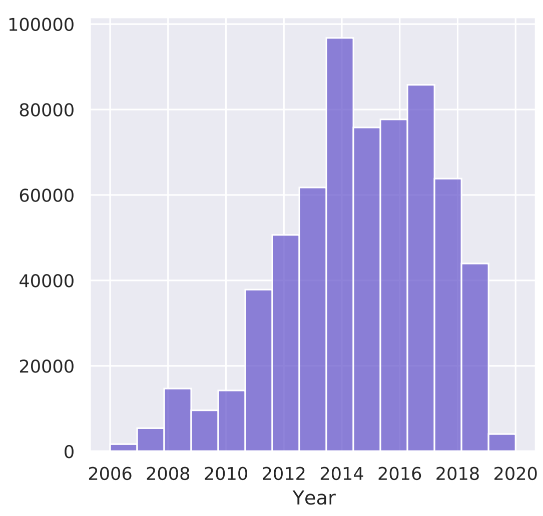


(d) AVVISI_CONTEGGIO

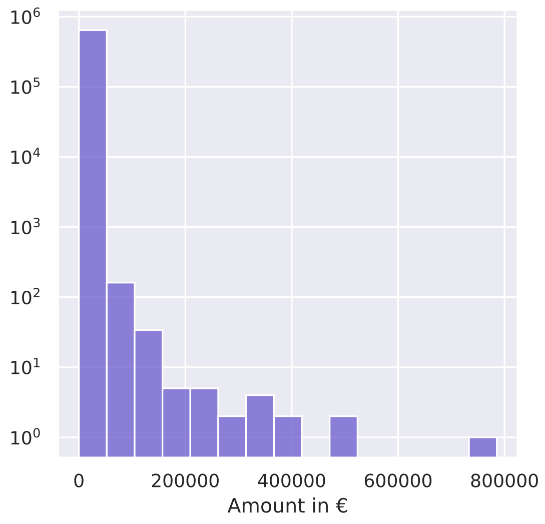


(e) AVVISI_PRODOTTO

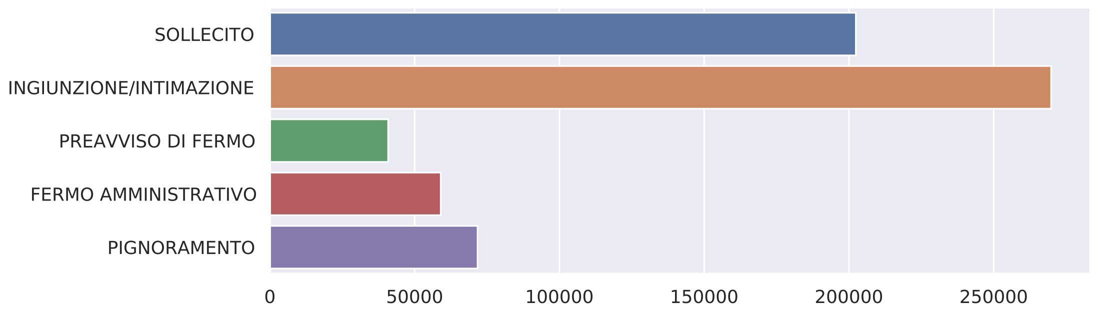
Figure 2.7: Overview of the variables 6-10 listed in Table 2.1.
In (c) and (d), ordinate values are expressed in logarithmic scale.



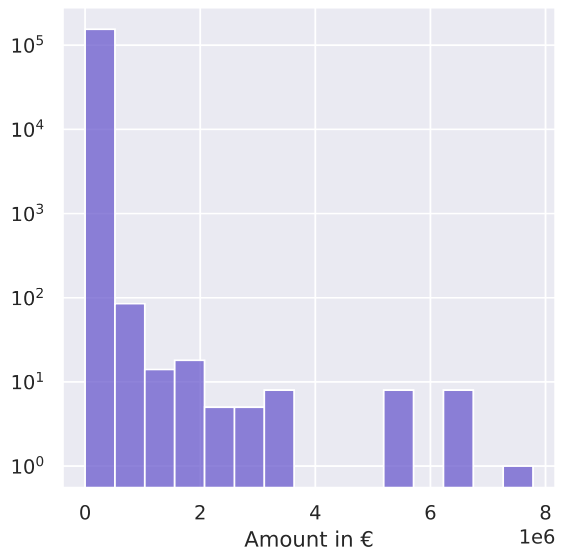
(a) anno_creazione



(b) importoDovuto

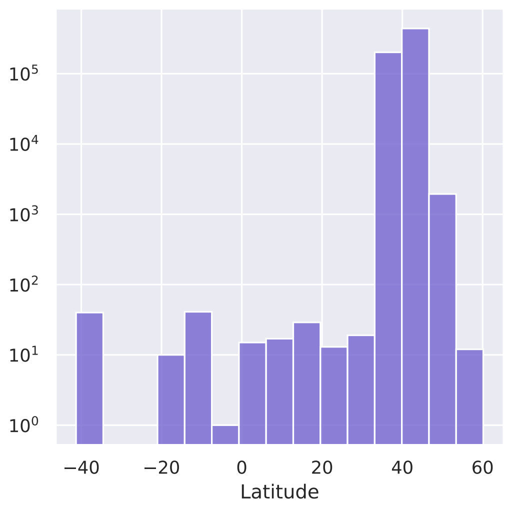


(c) FASE_RAGGIUNTA

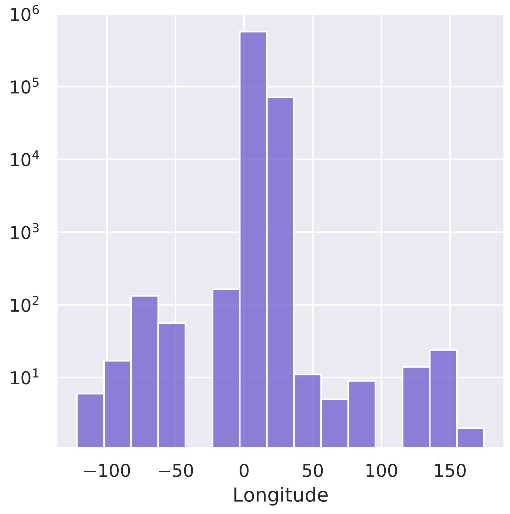


(d) valore_veicolo

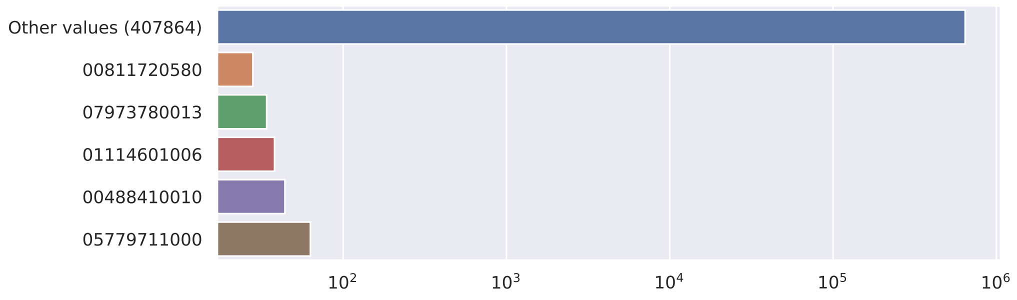
Figure 2.8: Overview of the variables 11-14 listed in Table 2.1.
In (b) and (d), ordinate values are expressed in logarithmic scale.



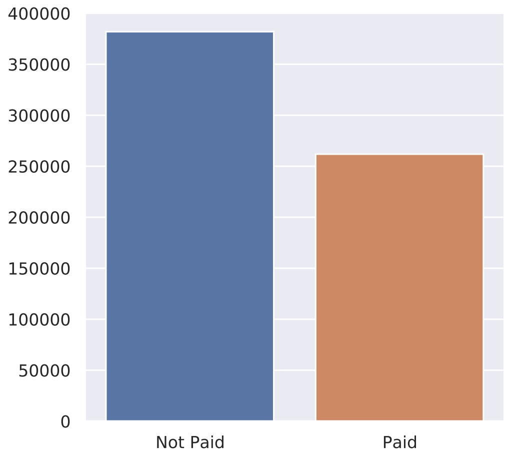
(a) lat



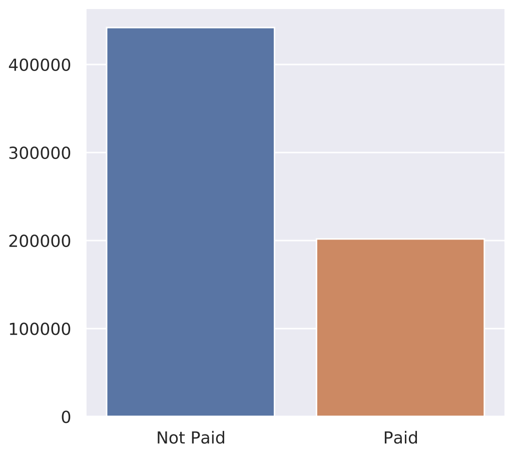
(b) long



(c) codice_fiscale



(d) pagato



(e) pagato_completamente

Figure 2.9: Overview of the variables 15-19 listed in Table 2.1.
In (a), (b) and (c) values are expressed in logarithmic scale.

2.3 Encoding

After the sanitization and the normalization of the input variables, we take care of the encoding and the normalization of the dataset variables. Since we need to feed our data to learning models, it is important that we take care of how we encode them since it can influence models performance. The encoding is applied to categorical and ordinal variables, while we normalized integer and real input variables through variable scaling techniques.

2.3.1 One-hot-encoding of categorical variables

An important issue about one-hot-encoding is how many records we have for each category. If we have too few records with a particular category, we should rather set a threshold between *top k records* and *others*. That is why it becomes essential to split the `ente` and `AVVISI_PRODOTTO` variables according to this idea. In our case, we come up with two thresholds:

- If the count for a specific `ente` is at least 5000, the value of `ente` is encoded as a single category, otherwise it is encoded as `ente_other`. The reason why we do so is because by setting such value as a threshold, we can guarantee that our records are almost equally split into two groups. All the records that have `ente` with a count > 5000, account for the 50.2% of the examples, while all the others account for the remaining part (see Figure 2.10). This allows us not to have categories for `ente` with too few values. To one-hot-encode `ente`, 16 bits are required (after the threshold is applied, there are 15 independent *bodies*, and 1 in which all the others fall into, therefore 16 bits are needed).

- If the count for a specific `AVVISI_PRODOTTO` is at least 13000, the value of `AVVISI_PRODOTTO` is encoded as a single category, otherwise it is encoded as `AVVISI_PRODOTTO_other`. Here, we pick 13000 as threshold after having inspected the number of occurrences for each `AVVISI_PRODOTTO`.
  All the `AVVISI_PRODOTTO` with at least 2% of frequency in the dataset are considered independent categories (see Figure 2.11). The ones that are below such value shall be aggregated into a single one instead. Doing so, we prevent too sparse categories to be created and compact them into a single one that accounts for almost 10% of the records. To one-hot-encode `AVVISI_PRODOTTO`, 6 bits are required (after the threshold is applied, there are 5 independent *notifications*, and 1 in which all the others fall into, therefore 6 bits are needed).

After having performed the following splits, we create two more variables:

- `frequenze_ente`
- `frequenze_prodotto`

These represent the frequencies for each `ente` and `AVVISI_PRODOTTO` normalized respect to the number of samples in the dataset, and will be useful when training learning models.

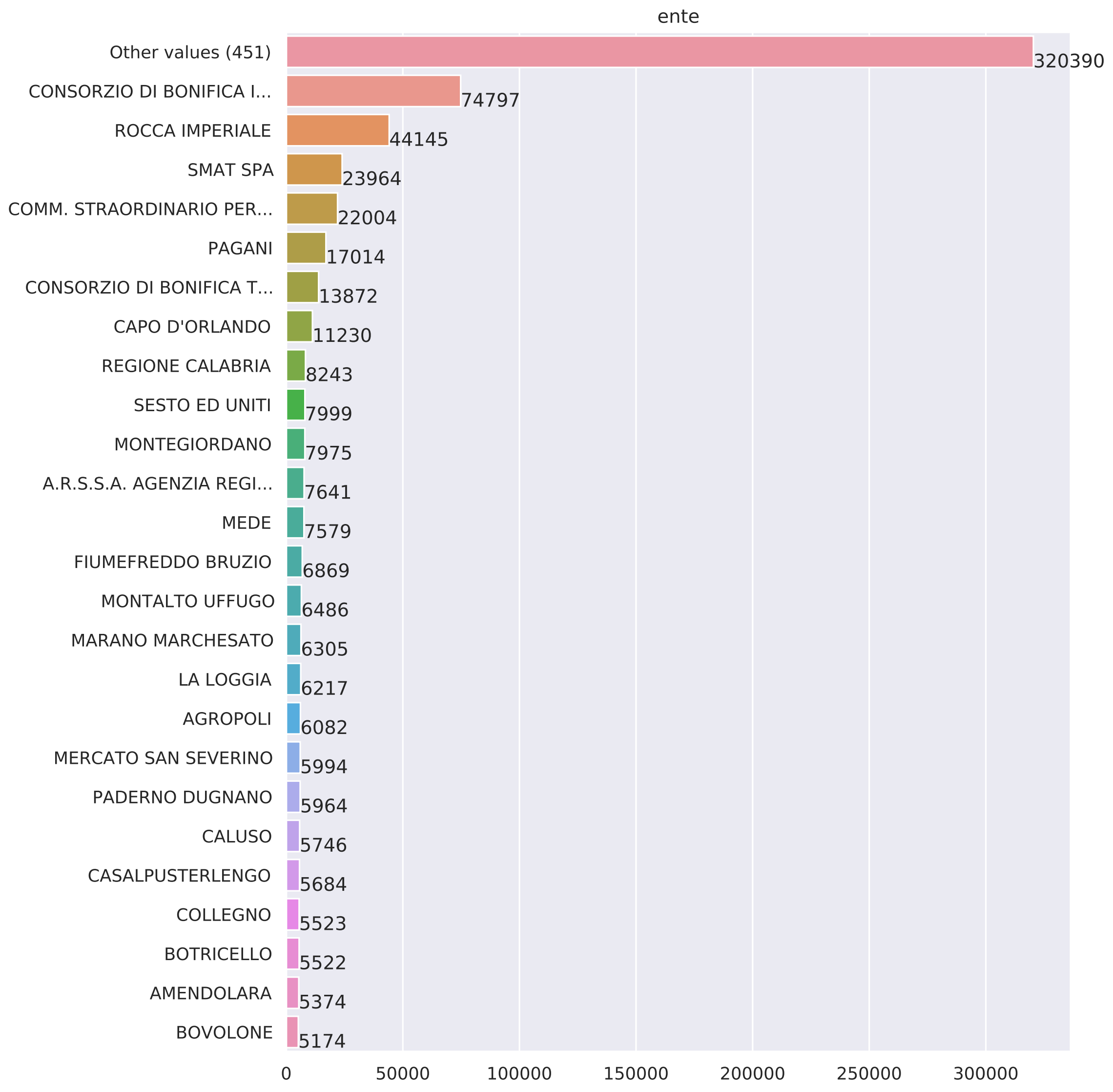


Figure 2.10: Inspection of the variable `ente`.

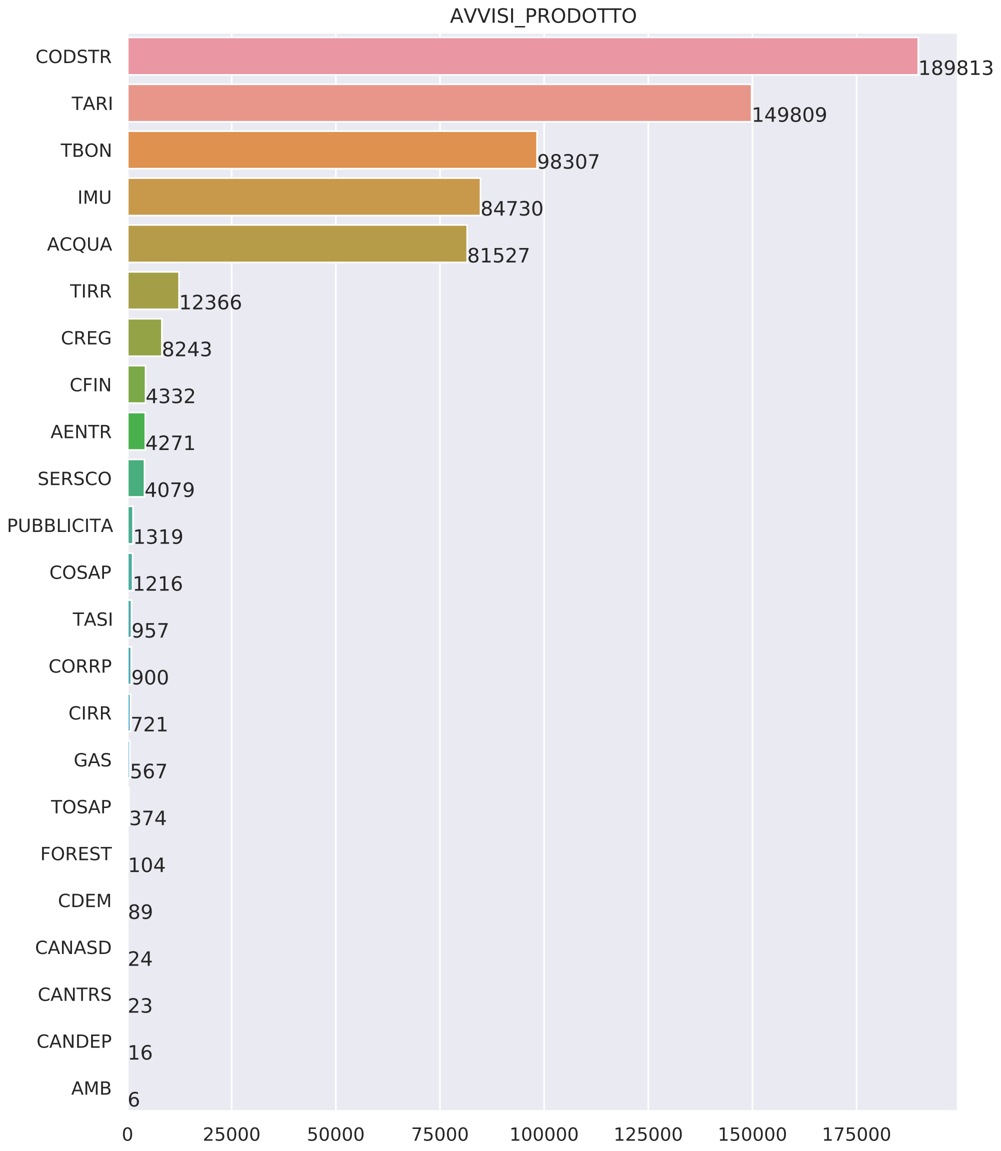


Figure 2.11: Inspection of the variable `AVVISI_PRODOTTO`.

2.3.2 Encoding of ordinal variables

FASE_RAGGIUNTA contains a scale from 1 up to 5 which shows the severity of the debt. If we encoded it through a one-hot encoding, we would lose this relationship and our model could not be blamed. On the other hand, if we preserve this relationship by introducing an ordinal encoding, our model understands that something encoded with value 5 has to be taken more seriously than a debt with a severity value of 1. That is why for the following variable an OrdinalEncoder is used. This encodes features with an ordinal relationship as an integer array, using unary coding (also known as *thermometer code*).

2.3.3 Input variable scaling

For integer and real valued inputs, several of the machine learning algorithms that will be used, perform better with variables on a similar scale, it is important to select the right scalers. For our purpose, two scalers are chosen:

- MinMaxScaler that preserves the shape of the distribution and scales the data to a given range (by default $feature\_range = [0, 1]$).

  The transformation is given by:

  $$X_{MinMaxScaler} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

  Such scaler is used for the following variables:

  1. numDatoriLavoro
  2. numContiCorrenti
  3. avvisiDurata
  4. avvisi_anno_minimo
  5. AVVISI_ANNO_MASSIMO
  6. anno_creazione

- RobustScaler that transforms the input variable by removing the median and it scales the data between the 1st (Q1) and 3rd (Q3) quantile. More precisely, it scales the data to the interquartile range $25\% - 75\%$. RobustScaler is usually used to reduce the effect of outliers. In fact, according to *Tukey's fences*, outliers can be defined as those values that are outside the range $[Q1 - Q3]$. The reason why RobustScaler is applied is because there are several input variables that have distributions containing values that are far from the median but still acceptable. That is why, even though they might appear as outliers, they are still useful for our classification problems and they cannot be dropped. RobustScaler comes in handy because contrary to MinMaxScaler that normalizes everything in a fixed range ($[0, 1]$), this distributes values taking advantage of a larger range, preserving in this way smaller values that occur more often than bigger ones. The transformation is given by:

$$X_{RobustScaler} = \frac{X - X_{Q1}}{X_{Q3} - X_{Q1}}$$

Such scaler is used for the following variables:

1. `numImmobili`
2. `AVVISI_CONTEGGIO`
3. `importoDovuto`
4. `valore_veicolo`
5. `lat`
6. `long`
7. `numMezzi`

In addition to this, we take care of filling the missing values for the variable `valore_veicolo` with the median, otherwise too many records would be wasted. Below, Table 2.2 summarizing the list of variables fed to the models and their encoding is provided.

Table 2.2: List of variables with their scalers and encoding.

Variable	**Scaler**	**Encoding**
numDatoriLavoro	MinMaxScaler	Real
numContiCorrenti	MinMaxScaler	Real
numImmobili	MinMaxScaler	Real
numMezzi	RobustScaler	Real
avvisi_anno_minimo	MinMaxScaler	Real
AVVISI_ANNO_MASSIMO	MinMaxScaler	Real
avvisiDurata	MinMaxScaler	Real
AVVISI_CONTEGGIO	RobustScaler	Real
anno_creazione	MinMaxScaler	Real
importoDovuto	RobustScaler	Real
FASE_RAGGIUNTA	-	Ordinal
valore_veicolo	RobustScaler	Real
lat	RobustScaler	Real
long	RobustScaler	Real
frequenze_prodotto	-	Real
frequenze_ente	-	Real
AVVISI_PRODOTTO	-	Categorical (6 binary variables one-hot-encoded)
ente	-	Categorical (16 binary variables one-hot-encoded)

2.4 Data Visualization

In the following section, two techniques are shown. More precisely, we cover the *Principal Component Analysis (PCA)* and the *t-Distributed Stochastic Neighbor Embedding (t-SNE)*. Such techniques come in handy for different purposes. Here, these are used to see if it is possible to gain more insights on the data by plotting them into two dimensions. In fact, as already mentioned in the previous sections, the data are made up of tens of dimensions. It seems to be quite clear that it is not possible to plot something that is more than three dimensions. That is why, by taking advantage of these techniques, we try to reduce the dimensionality of the data to plot them into two dimensions and eventually understand something more on them.

2.4.1 PCA - Principal Component Analysis

Without going into too much detail, the reason why PCA is used is to project the feature vectors into a lower dimensional space. As seen before, approximately 40 features for each record are collected. This means that we are into a 40-dimensional space. Since it is not feasible to plot and visualize such data, we take advantage of PCA that allows us to project the feature vectors into a 2-dimensional space.

To achieve the following task, the PCA implementation provided by `sk-learn` is used and allows to look for the two main components that contain the highest explained variance. Thus, two tests are carried out: the first one runs PCA in its classical way, while the second one first performs the Nystroem Kernel approximation [29] and finally runs PCA. The reason why PCA is performed with and without the Nystroem kernel is because we would like to understand if better results on the visualization can be achieved. Nystroem is an efficient technique that obtains a low-rank approximation of kernels. To achieve that, it randomly subsamples a number of features on which the kernel is evaluated. Since the Nystroem kernel should retain more explained variance, it might be interesting to find out that a higher separation is obtained.

Before running PCA on the data, is important to mention a couple of things. PCA requires that the data get properly scaled. This means that they should have zero mean and unit variance. Even though we have already added some `sk-learn` scalers in Section 2.3.3, we did not achieve that. In fact, `MinMaxScaler` is missing unit variance. That is why a `StandardScaler` becomes necessary before the data get fed to the PCA algorithm and it is defined as:

$$z = \frac{x - \mu}{\sigma}$$

where μ and σ represent the mean and the standard deviation of the training samples.

For the sake of clarity, both results are shown (on data before and after having performed the `StandardScaler`).

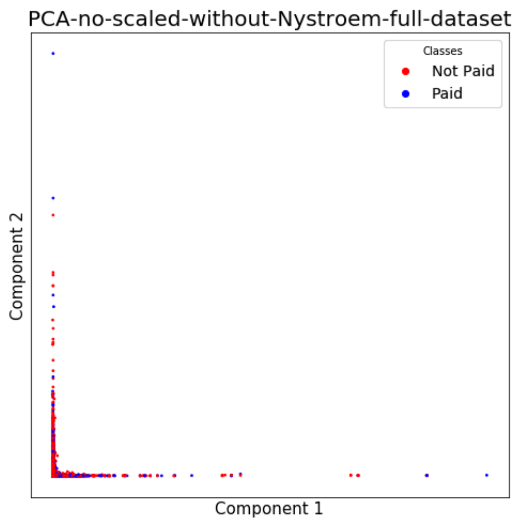


Figure 2.12: PCA performed with n_components = 2, without `StandardScaler` on full dataset.

The Figure 2.12, suggests that without scaling the data, there is no correlation at all between the two classes.

In Figure 2.13 we perform another time the PCA, but first, the `StandardScaler` is applied.

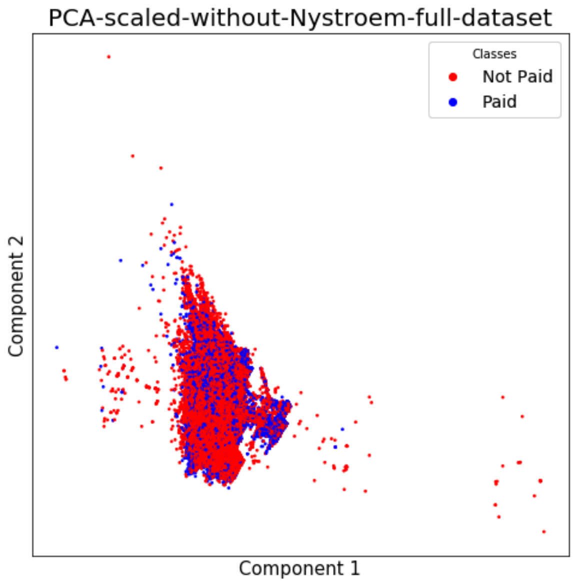


Figure 2.13: PCA performed with n_components = 2 and `StandardScaler` on full dataset.

No useful insight can be extracted yet, but the data look different. By giving a look at how much variance is retained in the first two components, we obtain around 10% of it. What we are going to do next is to try the same approach applying the Nystroem kernel approximation to see if anything changes.

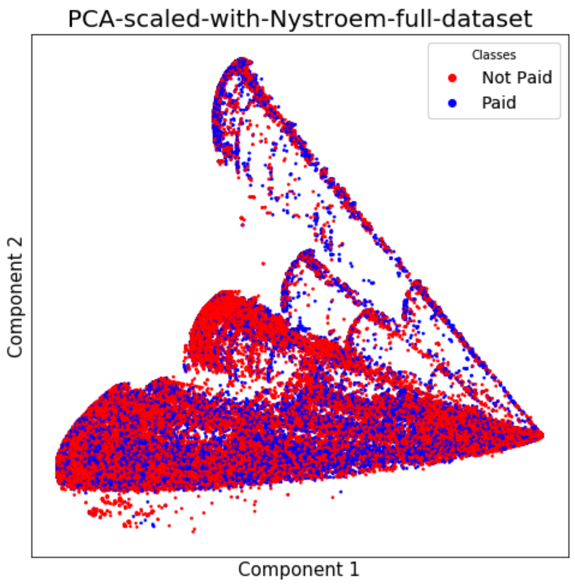


Figure 2.14: PCA performed with n_components = 2, `StandardScaler` and Nystroem kernel approximation on full dataset.

The data look once again different, but still, no useful insight can be extracted. What we can say is that the variance retained from the first two components increases up to 25% this time, but no separation between the two classes seems to be possible.

2.4.2 t-SNE - t-Distributed Stochastic Neighbor Embedding

As seen with PCA, also with t-SNE [17] we aim at getting some useful insight from the data. Unfortunately, so far, PCA has not been as useful as hoped. That is why we try to perform another technique usually used to deal with non-linear dimensionality reduction: t-SNE. t-SNE is able to group data points in such a way that similar objects have a small intra-cluster distance and dissimilar objects have a higher inter-cluster distance. Since it is not deterministic, every time it is run, it can provide a different result, but despite this, it still remains a good tool that is worth a try.

So far, we have been quite disappointed with the results of PCA. In fact, no separation could be achieved. t-SNE despite taking longer with its execution, is usually used to provide a better separation between classes.

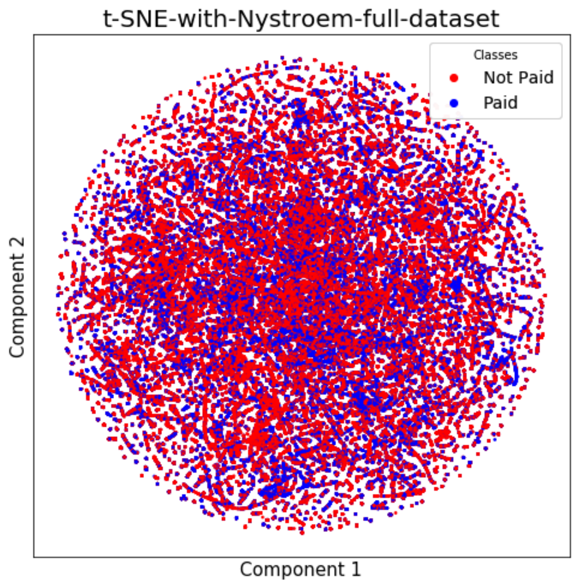


Figure 2.15: t-SNE performed with n_components = 2, `StandardScaler` and Nystroem kernel approximation on full dataset.

Once again, as it turns out from Figure 2.15, no clear separation seems to be possible to be obtained. This would suggest that further feature selection will become essential to try to discriminate the two classes. We will address the following task with Boruta feature selection, that will be later presented. Moreover, in Figure 2.15, the Nystroem kernel was applied. We did not show the t-SNE without Nystroem because very few differences could be appreciated. Both tests were performed though.

2.4.3 Features correlation

Before carrying out feature selection techniques, it might be interesting to first have an overview of the correlations of the features by plotting a correlation matrix. A correlation matrix is a square matrix whose entries are the features that get plotted one against another to check if any correlation among them is present. In Figure 2.16, features correlation is shown in the range $[-1, 1]$. The principal diagonal is obviously colored with the highest intensity $(+1)$ since a feature is plotted against itself so the correlation is trivial. In the other cases, when finding blue shades it means that a positive linear correlation is found. With red ones, a negative linear correlation instead. If the correlation is close to 0, no linear correlation is found. There are several methods to compute the correlation between two features. The one shown in Figure 2.16 is computed with *Pearson*, which is the standard correlation coefficient.

It would be definitely interesting to find out that the target variables (`pagato` and `pagato_completamente`) are correlated with any other feature. Unfortunately, it seems that they are not strongly correlated with anything specifically and therefore, no particular insight that justifies a correlation between a specific debtor feature and creditworthiness can be deducted.

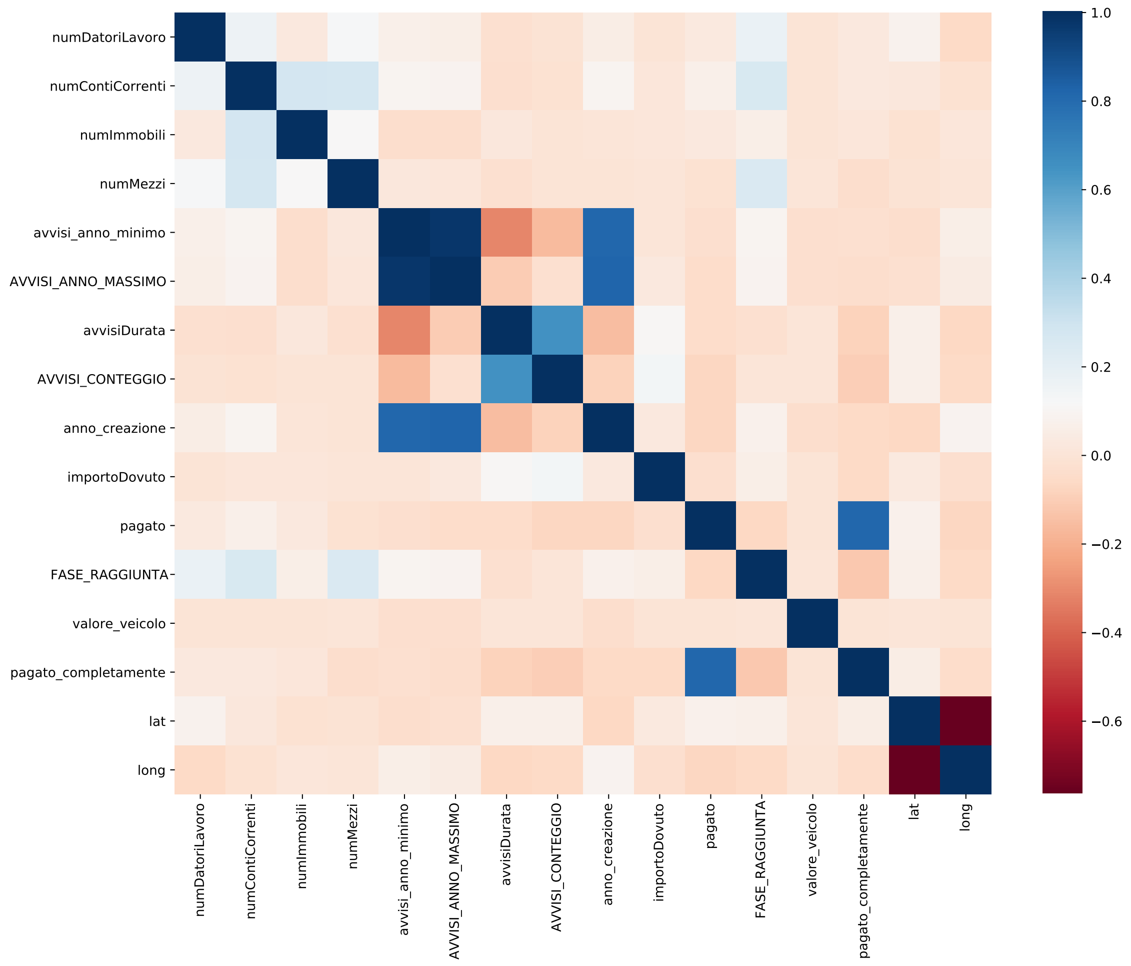


Figure 2.16: Features correlation matrix

The most anti-correlated features with our target variables are shown in Table 2.3 and 2.4. The remaining ones have lower absolute correlation coefficients and therefore are not included.

Table 2.3: Top 5 anti-correlated variables with the target variable **pagato**.

	pagato
anno_creazione	-0.075344
AVVISI_CONTEGGIO	-0.072801
long	-0.069796
FASE_RAGGIUNTA	-0.063242
avvisiDurata	-0.046370

Table 2.4: Top 5 anti-correlated variables with the target variable **pagato_completamente**.

	pagato_completamente
FASE_RAGGIUNTA	-0.121651
AVVISI_CONTEGGIO	-0.100079
avvisiDurata	-0.083087
importoDovuto	-0.061700
anno_creazione	-0.059400

Chapter 3

Methods

In this section, the Machine Learning models used in the experiments are described.

3.1 Overview

Learning models are the core part of this work since they are particularly powerful in extracting patterns present into data that otherwise would not be as easy to recognize. As previously mentioned in Section 2.1.2, two main classification problems are defined. In order to address a classification task, well-suited models have to be chosen.

For our purposes, the following are selected:

- Decision Tree
- Random Forest
- Feedforward Neural Network

Other models could be used, but a practical decision has to be made. One of the main problems that arises when dealing with Machine Learning is that depending on the models selected, results might be hard to interpret. This problem, known as *explainability*, involves the ability to extract insights at a human level from the models. In fact, even if it is possible to predict with very good performance on some tasks, it would be also interesting from a business perspective to understand what determines those results and what drove those predictions.

3.2 Decision Tree

The first model that is presented here is the *Decision Tree*. Decision Tree is part of the *Supervised Learning* algorithms, which are those algorithms where the training samples are labeled, and a function that best approximates them has to be computed. This is in contrast to the other family which is known as *Unsupervised Learning* algorithms in which data are not labeled and patterns among them have to be discovered.

Decision Trees are non-parametric, which means that no specific assumption on the distribution of the data is made. Even though it is a quite simple model, Decision Tree achieves good performance on several tasks and this is one of the reason it is selected.

They can be used to solve classification and regression problems and therefore are divided into two types:

- **Classification trees**: if the target variable is made of discrete values
- **Regression trees**: if the target variable is made of continuous values

Since the target variables to be predicted in our tasks are discrete, Decision Tree classifiers are used [20]. The reason why this model is widely used despite their simplicity is mainly because they can be visually clear in decision making and can provide explainable decisions. The typical structure of a tree is represented by a root, branches and leaves. Here, each internal node represents a test on a feature, each branch represents the result of a test and the leaves contain the class labels.

The basic idea is that simple *decision rules* are learned and these are helpful to understand to which class a specific record belongs to. The internal nodes behave in such a way that they split the instance space in sub-spaces. The way a Decision Tree makes the splits of the attributes highly affects the performance of the tree itself. In fact, there are several algorithms that are used to split a node into more sub-nodes. The final goal is to create sub-nodes that increase the purity of our decisions as much as possible. That is why it is worth to mention some splitting criteria that Decision Trees might actually implement. The most common are the *Gini impurity* and the *Entropy* criteria.

The *Gini impurity* is defined as:

$$\text{Gini} = 1 - \sum_{j=1}^{c} p_j^2$$

and the *Entropy* criteria, defined as:

$$\text{Entropy} = - \sum_{j=1}^{c} p_j \log p_j$$

where c represents the classes and p the probability of picking a sample with class j.

As we can see, both are computed in a similar way and represent the quality of a split. The *Entropy* though, requires more computation since a logarithmic function is used, that is why it might be slightly slower than *Gini*.

So far, mainly the pros of Decision Trees were discussed. Among the cons, it is important to mention the problem of *overfitting*. Decision Trees are quite prone to overfit, and as a consequence they lose in generalization performance if such problem is not properly addressed. To tackle overfitting, we can prune the trees by removing parts of them that are redundant for the classification.

Decision Trees have several hyperparameters that have to be accordingly tuned. Out of the several that could be possibly chosen, two are here selected when running the holdouts:

- `max_depth`: the maximum depth of the Decision Tree
- `min_samples_split`: the minimum number of samples required to split an internal node

The reason why these are the hyperparameters chosen is that they are the ones that mainly affect the performance of the Decision Tree if tested in different combinations. Of course other hyperparameters can be accordingly tuned but this is something that is addressed during the *model selection* step which will take care of a wider parameters space.

3.3 Random Forest

Let us move forward by introducing another model known as *Random Forest* [4]. This is an ensemble method that allows to use multiple base learners rather than just one to solve a specific task. In fact, we do not simply rely on one prediction produced by a single estimator. Several estimators are combined to return a more robust prediction instead. The most common techniques when dealing with ensemble methods are: *bagging*, *boosting* and *stacking*.

Bagging can be defined as a parallel ensemble because each model is built in such a way that is independent from the others. The following technique is usually used when we aim to reduce the variance of the learners. The basic idea in bagging is to create subsets of data that come from training samples randomly drawn with replacement. The final goal is to aggregate the learners trained on the bootstrapped samples. If we are dealing with a regression task, we simply average the results of the learners and take that as the final hypothesis. If a classification task is performed, the predicted class is selected through a voting mechanism instead, where the class that is voted the most is the one chosen as final prediction.

Boosting is another of the most used ensemble methods in machine learning. The following method is mainly used to reduce the bias in the model. The idea in this case is to perform a sequential process in which the learners are stacked one by one and together form the ensemble.

The third and last ensemble method considered is *stacking*. In bagging and boosting the ensemble was always performed by using one learning algorithm. In contrast to that, when dealing with the stacking, several models are trained by using different algorithms and in the end their results are combined to produce a prediction.

Random Forests train base decision trees using bagging, but also using a randomization for the selection of features to be evaluated at each node of the decision trees. Hence Random Forests leverage randomization at two levels to train the base learners:

1. At sample level by bootstrapping
2. At feature level by randomly selecting the features used at each node of the base Decision Trees.

In particular, Random Forests have been successfully applied to the prediction of financial risk assessment.

As previously explained when talking about the bagging technique, running a Random Forest would usually end up in getting better performance than their base learners and also more stable results.

Let us recall the two possible approaches:

- **Random Forest classifiers**: the mode of the classes to predict is returned
- **Random Forest regressors**: the average prediction of the trees is returned

Since the target variables to be predicted are discrete, Random Forest classifiers are used. The reason why Random Forests are chosen is because they usually outperform Decision Trees and it would be interesting for our purposes making a comparison between these two tree-based models.

Random Forests share most of the Decision Trees parameters but for the sake of clarity, the ones that are used when running the holdouts are listed below:

- `max_depth`: the maximum depth of each Decision Tree
- `min_samples_split`: the minimum number of samples required to split an internal node
- `n_estimators`: the number of trees to use in the forest

As mentioned when discussing the Decision Tree, there are several more hyperparameters that could be tuned. Their tuning is discussed when dealing with the *model selection* step. The ones here presented are the hyperparameters that most affect the performance of a Random Forest and are therefore included.

3.4 Feedforward Neural Network

Artificial Neural Networks (ANN) have proved to be very efficient when dealing with non-linear financial data and are generally used for a lot of classification tasks with several different architectures. Among them, it is worth to mention: Multi-Layer-Perceptron (MLP), Self-organizing-map (SOM), Radial Basis Function Network (RBFN), and Deep Neural Networks (DNN) [1].

For our purposes a Feedforward Neural Network (FFNN) is selected. The main difference between a DNN and a FFNN is simply the number of hidden layers. In deep models, there is a large number of them and this introduces a way higher number of parameters that, at least in our case, is not necessary.

Sometimes in fact, a deeper network might even worsen the model performance. This would be motivated by the way the back-propagation is performed. For a deeper network, the complexity of the model increases, more parameters (weights) need to be learned and the model can decrease its generalization capabilities. If more layers are stacked, this behaviour would be stressed even more. To tackle this problem, the training parameters should be accordingly tuned, but even more important would be the number of the input data available. In fact, if very little data is available, is definitely not beneficial to build a deep architecture.

FFNNs are very powerful in solving predictive tasks. The main goal of a FFNN is to approximate some function *f**. Since we are dealing with a classification problem, let us take for instance a classifier y = $f^*(\mathbf{x})$. By approximating such function, it becomes possible to map an input $\mathbf{x}$ to a class y. In a FFNN such mapping is defined as $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ and the network learns the parameters $\boldsymbol{\theta}$ that produce the best approximation [10].

So far, Decision Tree and Random Forest were discussed. It would be now interesting to make a comparison between these and neural models. The main differences between Neural Networks and tree-based models can be summarized as follows.

A Neural Network is more computationally demanding and there are way more hyperparameters that is possible to tune. As a consequence, it happens that these models are trained by empirically tuning the hyperparameters until the model generalizes well over the data.

Moreover, if the interpretability of the results is important, Neural Networks are not as good as tree-based models can be. In fact, other than requiring way more data, after these networks are trained, they can be thought as black boxes where data are fed in input and predictions in output are returned, without being really able to understand what caused those results. Of course, Neural Networks performance strictly depend on the task is meant to solve and on the features available.

Again, it should now be clear that with Neural Networks, interpretability gets lost, therefore, if explainability is a priority they might not be the best fit. On the other hand, if explainability is not a priority they have proven to be very effective in solving a lot of tasks and can be very useful.

Here, the focus is on the architecture used to train our FFNN model.

In Figure 3.1, the architecture of our FFNN is shown. This is built as follows:

- An input layer $\in \mathbb{R}^{38}$ is provided. Here, the dataset is fed to the network record by record. Each record is represented by a vector made of 38 features.
- Two hidden layers $\in \mathbb{R}^{128}$ are added in order to provide a larger space in which weights can be computed.
- Two hidden layers $\in \mathbb{R}^{64}$ are added, in which the number of neurons is slightly decreased, going from 128 of the former layers to 64.
- Two hidden layers $\in \mathbb{R}^{32}$ are added, in which the number of neurons is slightly decreased, going from 64 of the former layers to 32.
- A dropout of 0.3 is here added. Dropout has proved to be effective in preventing our network from overfitting [24].
- Three hidden layers $\in \mathbb{R}^{16}$ are added, in which the number of neurons is slightly decreased, going from 32 of the former layers to 16.
- Output layer $\in \mathbb{R}^{1}$ that returns the predictions.
- When going from the input layer until the last hidden layer, the ReLU is used as activation function. Only in the last layer, when returning the class predictions, the Sigmoid is used [18].

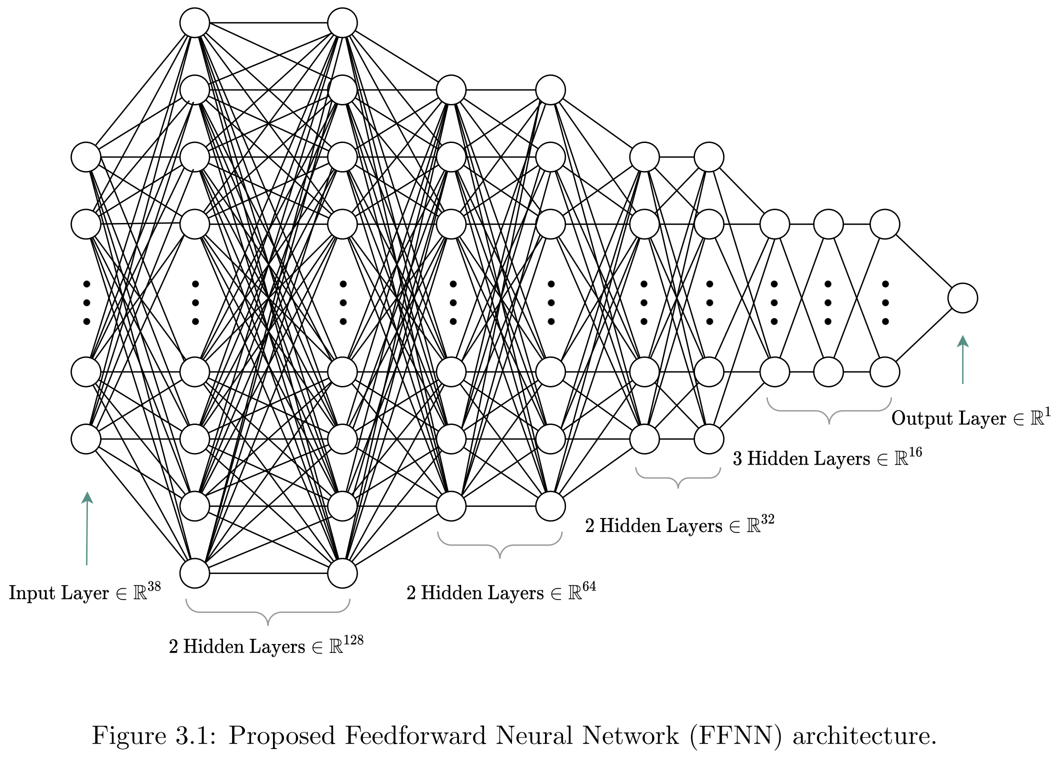

Output Layer $\in \mathbb{R}^1$
3 Hidden Layers $\in \mathbb{R}^{16}$
2 Hidden Layers $\in \mathbb{R}^{32}$
2 Hidden Layers $\in \mathbb{R}^{64}$
Input Layer $\in \mathbb{R}^{38}$
2 Hidden Layers $\in \mathbb{R}^{128}$

Figure 3.1: Proposed Feedforward Neural Network (FFNN) architecture.

Below, the hyperparameters chosen while running the holdouts are provided:

- `epochs`: number of epochs to train the model
- `batch_size`: number of samples for gradient update
- `optimizer`: *nadam*, optimizer that implements the NAdam algorithm
- `loss`: *binary_crossentropy*, objective function

As previously mentioned when presenting the other models, these are just the most common hyperparameters that are generally used in Neural Networks. A more efficient process is the model selection, that is usually performed to find the best set of parameters to use when running a model. In Section 3.6, the model selection approach applied in this work, is discussed.

3.5 Feature selection

Datasets usually come with a lot of features and most of them might not be as significant as it might look. Sometimes, having too many features can badly affect computational performance ending up in what is known as the *curse of dimensionality*. Moreover, too many not informative features can worsen model predictions. Despite our dataset is not affected by a high dimensionality, it is still desirable to select the right subset of features and to throw the not relevant ones away. Several are the techniques available to perform this step but in this case we focused on two in particular. The first one is called *Boruta* feature selection [13] and will help us to compute what is known as the all-relevant set of features, while the second one is a feature selection method embedded in Random Forest that aims to provide a feature ranking so that a subset of top k features can be used to make predictions.

3.5.1 Boruta

Boruta allows to automatically perform feature selection by deciding which features are important and which are not. While most of the feature selection methods try to solve a *minimal-optimal* problem, Boruta aims to compute the *all-relevant* set of features present in the dataset which is a more complex problem to solve.

The main difference between these two approaches is that in a minimal-optimal problem, only the non-redundant attributes are selected. In the all-relevant one, all the relevant attributes are selected instead. As a consequence, the selected attributes can be in some way redundant and this is because there are some of them that carry information already expressed by others. While this might not look particularly efficient, it becomes very useful when trying to understand more about a specific subject as in this case.

In order to solve the all-relevant problem, this algorithm makes use of feature importance scores. One of the Machine Learning models that lends itself well with that is *Random Forest* that is the model used by default by the algorithm. For our purpose the *BorutaPy* package is used [22]. Let us dive deeper in the Boruta algorithm by looking at the pseudocode presented in its original work [13].

Algorithm 1 Boruta Feature Selection (pseudocode extracted from [13])

```
1. Extend the information system by adding copies of all variables (the information system is always extended by at least 5 shadow attributes, even if the number of attributes in the original set is lower than 5).
2. Shuffle the added attributes to remove their correlations with the response.
3. Run a random forest classifier on the extended information system and gather the Z scores computed.
4. Find the maximum Z score among shadow attributes (MZSA), and then assign a hit to every attribute that scored better than MZSA.
5. For each attribute with undetermined importance perform a two-sided test of equality with the MZSA.
6. Deem the attributes which have importance significantly lower than MZSA as ‘unimportant’ and permanently remove them from the information system.
7. Deem the attributes which have importance significantly higher than MZSA as ‘important’.
8. Remove all shadow attributes.
9. Repeat the procedure until the importance is assigned for all the attributes, or the algorithm has reached the previously set limit of the random forest runs.
```

Boruta takes in input several hyperparameters but here the most important are mentioned:

- `n_estimators`: the number of estimators used by the ensemble method, that in this case is Random Forest.
- `max_iter`: the number of iterations the algorithm will run.

It is quite clear that the higher `max_iter` is, the more accurate the output the algorithm returns will be, but for big datasets such process can be time-consuming, that is why a trade-off between speed and performance is important to be reached. In any case, Boruta returns two lists of features: the *confirmed* and the *rejected* ones. Among the ones that get rejected are included also the features that could not be properly processed by the algorithm because of the too small number of iterations.

Boruta feature selection is performed on the problems 1 and 2 presented in Section 2.1.2.

3.5.2 Embedded methods

Feature selection can be usually performed by following three different approaches. These are known as *filter*, *wrapper* and *embedded* methods. To have a deeper understanding of why embedded methods are here preferred, it is worth to provide an overview on all the three approaches.

A *filter* method is generally preferred when a very large number of features has to be handled. They are quite robust to overfitting but at the same time it might be possible that they end up not selecting the best features.

A *wrapper* method usually gives better performance than the filter but it is also slower and prone to overfitting. In this method, it is possible to understand how the features interact between each other.

Finally, an *embedded* method tries to take advantage of the benefits of both previously discussed approaches. In this method, the feature selection step is performed while the model is training itself over the data and this is also the reason why they are called “embedded”. Embedded feature selection is prone to overfitting (as seen in the filter method) and runs faster than the wrapper.

What it is interesting about this technique is that it is possible to obtain reliable results while training the model. This means that rather than performing the feature selection and the training as two different tasks, it is possible to combine them and take advantage of something that is internally computed from the model. For our purposes, we will use the “feature importance” concept behind Random Forest. Tree-based models in fact are already equipped with everything needed to perform such step.

Thanks to a ranking, it would be possible to select a subset of features that would be prioritized over their whole list, and the top ranked features can lead to better overall classification performance. Another reason why a feature ranking becomes necessary is because new data might become available, and they might have a slightly different structure than the ones already collected.

3.6 Model selection methods

In next sections, the models will be executed and a reasonable set of *hyperparameters* will be provided. Even though these are chosen with some basic understanding of the tasks and the models themselves, they are not meant to be the optimal ones.

What is usually done in this case is performing something called *model selection*. Model selection is a step that allows for selecting a model among several candidates. This enables to understand which model best performs over the others.

As seen, every learning algorithm requires a set of several hyperparameters in order to be run. Such set might have large impact over the final model performance, and more importantly, it is not known from the beginning, since it can vary depending on the task is meant to solve. *Hyperparameter optimization* allows to select the set of the optimal hyperparameters that best perform for a specific learning algorithm.

This step can be carried out in several different ways. Here, some of them will be discussed, and are the following:

- Grid Search
- Random Search
- Bayesian Optimization

For each of them, pros and cons of their use are presented and a decision about which one will better suit our needs is eventually made.

3.6.1 Grid Search

An important concept that will accompany us throughout the next sections is something called "parameter space“, which can be also referred to as "search space". More precisely, a *parameter space* can be thought as the space of all the possible parameters that define a model, or as in this case, a learning model. The first approach here proposed is called *Grid Search* and it consists of defining a grid of hyperparameters in which each of them is made of a list of values to test.

A Grid Search, first defines a search space, and then it tests all its possible combinations. Of course it is a quite time-consuming process, since all the combinations have to be run. It can be seen as a brute-force attempt to understand which is the set of hyperparameters that best perform on the tasks of interest.

3.6.2 Random Search

Bearing in mind that the concept of "parameter space" still holds, let us move to the discussion of another possible approach for hyperparameter optimization: *Random Search*. Random Search randomly selects a set of hyperparameters from the pool of parameter space available and tests them on the model. The reason why such technique might come in handy is mainly because it allows to spot set of hyperparameters that would not be easily guessed otherwise. Overall, if no other means are available or known, Grid Search and Random Search still prove to be quite useful for hyperparameter optimization.

3.6.3 Bayesian Optimization

In contrast to Grid Search, where a brute-force approach is attempted, and Random Search, where random combinations of hyperparameters are tried, what we are about to introduce now has solid statistic foundation. The third approach here proposed is known as *Bayesian Optimization* [23].

Bayesian Optimization (BO) is a technique usually used when it is necessary to optimize some expensive objective function *f*. When talking of objective functions we are referring to the underlying functions that have to be approximated when performing a Machine Learning task and that eventually output a metric. However, the objective function can be quite hard to optimize, since its direct evaluation can be too expensive. In fact, while in algorithms like gradient descent is possible to compute the derivatives and suggest a direction for the optimization, sometimes this is not possible or too expensive to do. What is possible to do is trying to approximate a *surrogate function* instead.

A surrogate function is a function that aims to approximate the objective function investing less effort than it would be otherwise required. The surrogate functions are built by sampling *points* and can be represented through *Gaussian Processes* [21]. These can be defined as probability distributions over functions and are the core of the BO execution. In fact, the Gaussian Processes already include the Bayes ideas and this is very useful when trying to optimize the surrogate function. In our case, the "points" mentioned before, are the training data. The basic idea here is to sample a function and take advantage of the Bayes rule, trying to *exploit* and *explore* the regions of the function in such a way that a maximum is found in as few steps as possible while going through the parameter space.

Exploitation and exploration are the two key concepts to understand here and a compromise between them should be reached. As already mentioned, the effectiveness of the BO is to be linked to the adoption of the Bayes theorem, that enables to investigate with more knowledge into the surrogate function the regions that are more promising for finding the global maximum.

The Bayes rule is here recalled just for the sake of completeness:

$$\mathrm{P(A|B)} = \frac{\mathrm{P(B|A)P(A)}}{\mathrm{P(B)}}$$

where:

- P(A), also called the *prior*, is the probability of the event A being true

- P(B), or *marginal likelihood*, is the probability of the event B being true
- P(A|B) is called the *posterior* and defines the probability of observing the event A if B is true
- P(B|A) is called the *likelihood* and is the probability of observing the event B if A is true

As previously mentioned, exploitation and exploration are two important concepts in the BO and should be further investigated.

The *exploitation* tries to sample where good predictions are found. As a consequence, it might tend to get stuck longer in small regions and this would not be beneficial for the optimization, since the algorithm might mistakenly think to have found the global maximum when in fact it is simply stuck in a local one.

On the other hand, there is the *exploration* which tends to sample in regions with high uncertainty. It is worth to point out that when dealing with Gaussian Processes, there is very good knowledge in close proximity of the sampled points and almost no knowledge the more we move away from them. Which despite it might not seem a good outcome, it is, on the contrary, very effective for further exploration. In fact, the algorithm knows the regions that were unexplored and depending on the distributions of the priors, it might favour those regions or not for further sampling.

Once the BO is left running for a number of iterations on a parameter space, it should produce what is called the set of the optimal hyperparameters for the learning model. Among the three approaches so far proposed, the BO seems to be more robust and will be therefore used and commented in the results chapter.

Chapter 4

Results

4.1 Experimental set-up

The Machine Learning methods presented in Chapter 3, are here discussed in terms of metrics, performance evaluation and implementation details.

4.1.1 Tasks

Two variants of the partial and full debt solvency are considered, ending up with four classification problems:

- **Partial debt solvency**:
  1. Predict if a debt will be paid at least partially
  2. Equal to the previous one but debts with a specific fiscal code/VAT are included either in the train or in the test set
- **Full debt solvency**:
  1. Predict if a debt will be entirely paid
  2. Equal to the previous one but debts with a specific fiscal code/VAT are included either in the train or in the test set

4.1.2 Metrics

In order to evaluate the results obtained from the execution of the models so far discussed, it is important that some metrics are fixed and properly motivated. Here, three of them are selected and they respectively are:

- Accuracy
- AUROC
- AUPRC

In Machine Learning there are a lot of different metrics available, and their choice mainly depends on the task is meant to accomplish. In fact we might be dealing with classification problems, regression problems, statistical problems, etc. Since most of this work revolves around classification tasks, the most useful metrics for this kind of problem are selected.

The *accuracy* is probably the easiest metric to understand and is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

To motivate *AUROC* and *AUPRC* is necessary to define what an *ROC* curve is. A ROC (receiving operating characteristic) curve is a plot able to show the performance of a binary classifier at all thresholds. This curve plots the true positive rate (TPR) against the false positive rate (FPR). More formally:

True Positive Rate (TPR) also known as *recall* or *sensitivity* is defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

False Positive Rate (FPR) is defined as:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Let us now move to the *AUC* or also *Area Under the ROC Curve*, which "measures the entire two-dimensional area underneath the entire ROC curve" [7]. By computing the *AUROC* it is possible to assign a score between 0 and 1 to each model and understand which one performs better over the data.

The reason why also the *AUPRC* is selected, is because this metric is well-suited to measure performance with imbalanced datasets. *Area Under Precision-Recall Curve* is an AUC where on the x-axis there is the *recall* and on the y-axis the *precision.* Even if our data are not extremely imbalanced, it is still worth seeing the difference between AUROC and AUPRC results.

4.1.3 Multiple Holdouts

Before letting the model doing any computation, it is important that the data are properly split. The reason why such splitting becomes necessary is because the model has to learn to generalize over the problem is trying to solve. This splitting is commonly known as the division of the original dataset into two parts: the *training* set and the *test* set. The ratio typically used to perform this split is 80% for the training data and 20% for the test data. Other combinations can be obviously tried. What is interesting here is the basic idea to let the model train itself on a wide portion of the data and test its ability to generalize on unseen ones.

Another problem arises when trying to evaluate the performance of Machine Learning models. When splitting the data into training and test set, there is no guarantee that the actual split might end up in some unreliable predictions. The reason why this happens is because every time the data get split, unless the process of their splitting can be reproducible, there is no control on the way the samples get assigned to the training nor the test set. In other words, a single test on a single split can not be trusted. To overcome this problem, multiple experiments with different splits have to be run and averaged. By doing so, it is possible to prevent our results from being unstable due to fluctuations caused by more or less favourable splits. Such technique is known as *multiple holdouts* or *Monte-Carlo cross-validation*.

4.1.4 Implementation

In order to carry out the analyses, a set of tools needs to be identified and chosen. In this case we decide to stick with the *Python* programming language (version 3.7.6) that offers a wide range of libraries such as *Pandas, NumPy* and *Scikit-learn* that come in handy to better manipulate dataframes and perform analyses on them.

Moreover, a private repository on GitHub is created. By doing so, it is also possible to set up a continuous integration service like Travis CI [6]. Travis is responsible for running the unit tests every time commits to the GitHub repository are pushed. In this way, greater control over the codebase is guaranteed. In fact, in the event anything breaks, Travis will notify it and the build will fail.

Getting back to the execution of *multiple holdouts*, 100 of them are run with a ratio of 80/20 for the train and test set for each model and with the following sets of hyperparameters:

1. Decision Tree:
    - max_depth: 25
    - min_samples_split: 2
2. Random Forest:
    - n_estimators: 500
    - max_depth: 25
    - min_samples_split: 2
3. Feedforward Neural Network:
    - epochs: 50
    - batch_size: 1024

After that, their results are properly stored. In fact, by fixing the `random_state` argument of the `train_test_split` function provided by `sk-learn` to an integer value, it is possible to ensure reproducibility of the outputs, unless, some more randomness is introduced in other operations.

4.2 Experimental results

First, results obtained from the execution of the models without feature selection are shown. After that, the feature selection is performed and a comparison between performance is made.

4.2.1 Predictions without feature selection

Although a preliminary overview on the classification problems was already done in Section 4.1.1, it is worth to explain how we came up with such division. As seen, the original and more general task of debt prediction can be defined as a binary classification. More precisely, there are two classes that can be predicted: the positive one, which represents a paid debt, and a negative one, which represents just the opposite.

With further analyses, two more interesting tasks can be defined: a *partial solvency* and a *full solvency* prediction. Furthermore, since some fiscal codes and VAT numbers might occur multiple times in the data, a deduplication of their dossiers is considered and performed. What we want to understand is whether or not clients with multiple dossiers can affect the overall performance of the models. That is why, a deduplication of these dossiers is taken care of. More precisely, we make sure that if a fiscal code or a VAT number occurs more than once in the dataset, the data are split in such a way that these dossiers are not included in both, train and test set, at the same time. This brings us to the four classification tasks that are here performed.

Figures 4.1, 4.2 and 4.3 should provide a good overview of the models results before feature selection is performed. For even more detailed results, in Tables 4.1, 4.2 and 4.3 the complete performance values are provided.

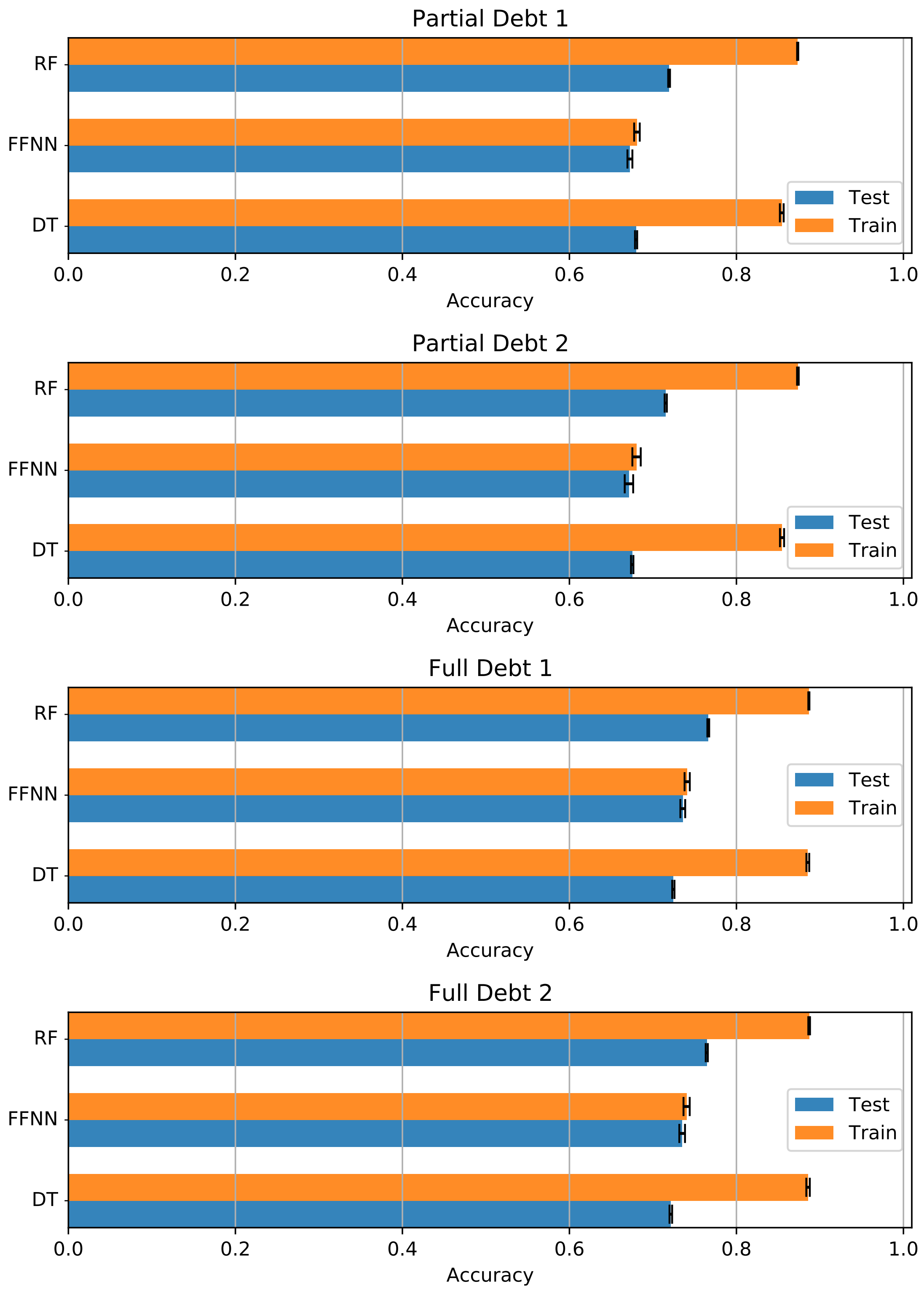


Figure 4.1: **Accuracy** results on the four classification problems presented in 4.1.1 without feature selection.

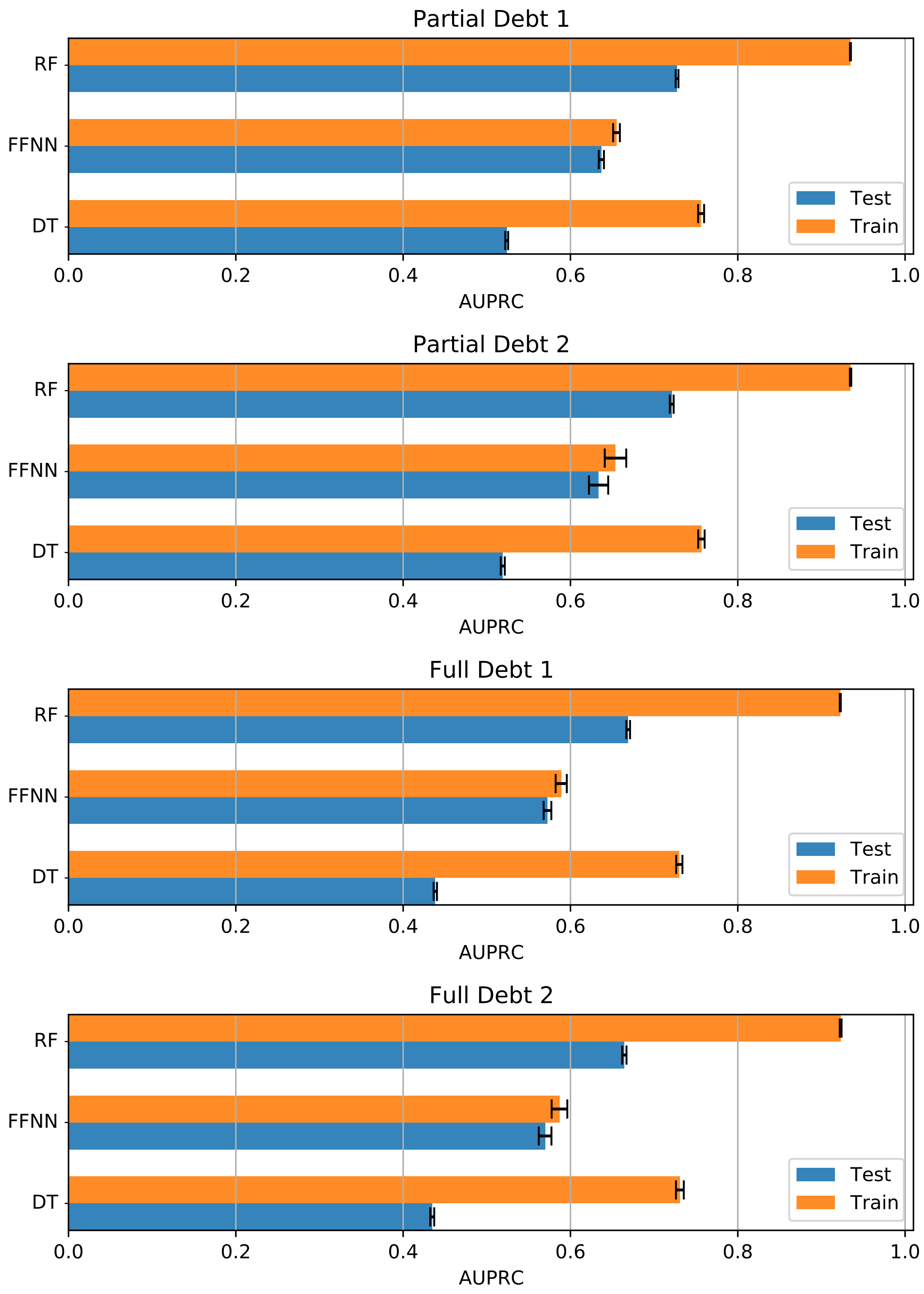


Figure 4.2: **AUPRC** results on the four classification problems presented in 4.1.1 without feature selection.

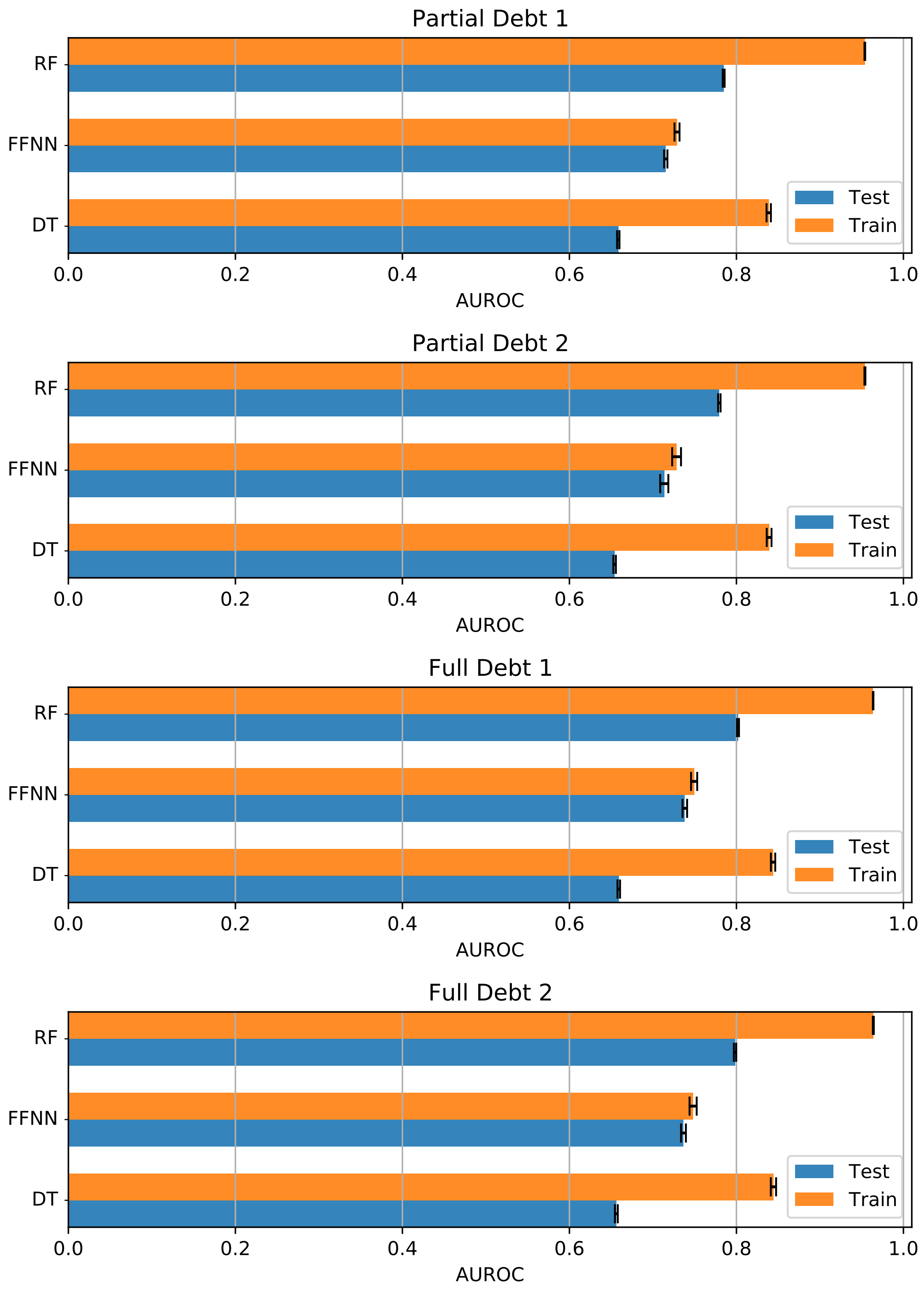


Figure 4.3: **AUROC** results on the four classification problems presented in 4.1.1 without feature selection.

Table 4.1: **Accuracy** results on 100 holdouts without feature selection.

Model	Train Accuracy	Test Accuracy	Task
Decision Tree	85.42%	67.98%	Partial Debt 1
Decision Tree	85.46%	67.51%	Partial Debt 2
Decision Tree	88.54%	72.44%	Full Debt 1
Decision Tree	88.57%	72.14%	Full Debt 2
Random Forest	**87.33%**	**71.93%**	**Partial Debt 1**
Random Forest	**87.35%**	**71.53%**	**Partial Debt 2**
Random Forest	**88.67%**	**76.62%**	**Full Debt 1**
Random Forest	**88.70%**	**76.44%**	**Full Debt 2**
Feedforward Neural Network	68.08%	67.24%	Partial Debt 1
Feedforward Neural Network	68.03%	67.12%	Partial Debt 2
Feedforward Neural Network	74.10%	73.57%	Full Debt 1
Feedforward Neural Network	74.03%	73.50%	Full Debt 2

Table 4.2: **AUPRC** results on 100 holdouts without feature selection.

Model	Train AUPRC	Test AUPRC	Task
Decision Tree	75.61%	52.37%	Partial Debt 1
Decision Tree	75.65%	51.91%	Partial Debt 2
Decision Tree	73.01%	43.84%	Full Debt 1
Decision Tree	73.07%	43.47%	Full Debt 2
Random Forest	**93.47%**	**72.74%**	**Partial Debt 1**
Random Forest	**93.48%**	**72.10%**	**Partial Debt 2**
Random Forest	**92.25%**	**66.90%**	**Full Debt 1**
Random Forest	**92.29%**	**66.44%**	**Full Debt 2**
Feedforward Neural Network	65.50%	63.69%	Partial Debt 1
Feedforward Neural Network	65.38%	63.36%	Partial Debt 2
Feedforward Neural Network	58.89%	57.25%	Full Debt 1
Feedforward Neural Network	58.68%	56.97%	Full Debt 2

Table 4.3: **AUROC** results on 100 holdouts without feature selection.

Model	Train AUROC	Test AUROC	Task
Decision Tree	83.86%	65.85%	Partial Debt 1
Decision Tree	83.91	65.41%	Partial Debt 2
Decision Tree	84.38%	65.90%	Full Debt 1
Decision Tree	84.43%	65.61%	Full Debt 2
Random Forest	**95.37%**	**78.47%**	**Partial Debt 1**
Random Forest	**95.38%**	**77.95%**	**Partial Debt 2**
Random Forest	**96.37%**	**80.18%**	**Full Debt 1**
Random Forest	**96.39%**	**79.83%**	**Full Debt 2**
Feedforward Neural Network	72.88%	71.53	Partial Debt 1
Feedforward Neural Network	72.83%	71.35%	Partial Debt 2
Feedforward Neural Network	74.93%	73.81%	Full Debt 1
Feedforward Neural Network	74.81%	73.65%	Full Debt 2

By looking at Figures 4.1, 4.2 and 4.3, Random Forest seems to be the model that best performs on all the classification tasks for the metrics considered. The second best model is the Feedforward Neural Network that despite it is outperformed on the Accuracy metric for the tasks *Partial Debt 1* and *Partial Debt 2* from the Decision Tree, it generalizes better on the other problems. From Tables 4.1, 4.2 and 4.3, there is no real difference between the performance produced by *Partial Debt 1* and *2*, as well as *Full Debt 1* and *2*. This would suggest that more dossiers for the same clients do not affect the models predictions.

To understand and validate the results obtained so far, a *Wilcoxon signed-rank* test, is performed. Wilcoxon signed-rank test is a non-parametric test for paired data and it is used to compare two samples on a single one to check if their mean differs [28]. More precisely, the *Wilcoxon signed-rank* test allows for:

- A better understanding among the models run, by testing which one performs better over the others
- Finding out whether or not running Boruta feature selection has a positive impact over the classification problems

Below, Win-Tie-Loss Tables 4.4, 4.5, 4.6 and 4.7 summarize the results scored from the models. The results presented here, come from the execution of the Wilcoxon test on the 100 holdouts results which are compared with a level of significance $\alpha = 0.01$. Each model competes against the others and depending on the statistical test results, there might be a *Win*, a *Tie* or a *Loss*.

The tables are split according to the four classification tasks as follows:

Table 4.4: Win-Tie-Loss table comparing models on the task **Partial Debt 1**.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	1	0	1	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	0	0	2	1	0	1	1	0	1

Table 4.5: Win-Tie-Loss table comparing models on the task **Partial Debt 2**.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	1	0	1	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	0	0	2	1	0	1	1	0	1

Table 4.6: Win-Tie-Loss table comparing models on the task **Full Debt 1**.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

Table 4.7: Win-Tie-Loss table comparing models on the task **Full Debt 2**.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

As we can see, Wilcoxon results confirm what already came out from the histograms previously presented.

4.2.2 Predictions using Boruta feature selection

In Figures 4.4, 4.5 and 4.6, histograms representing the comparison between the performance obtained before and after running Boruta are shown.

At first glance, running Boruta does not show a large improvement in the models. More precisely, Decision Tree and Random Forest seem to achieve almost the same performance regardless of the feature selection. A slight improvement is noticed with Feedforward Neural Network instead.

Further analysis is necessary though, since we might be losing insights that are not clear enough just by looking at the histograms.

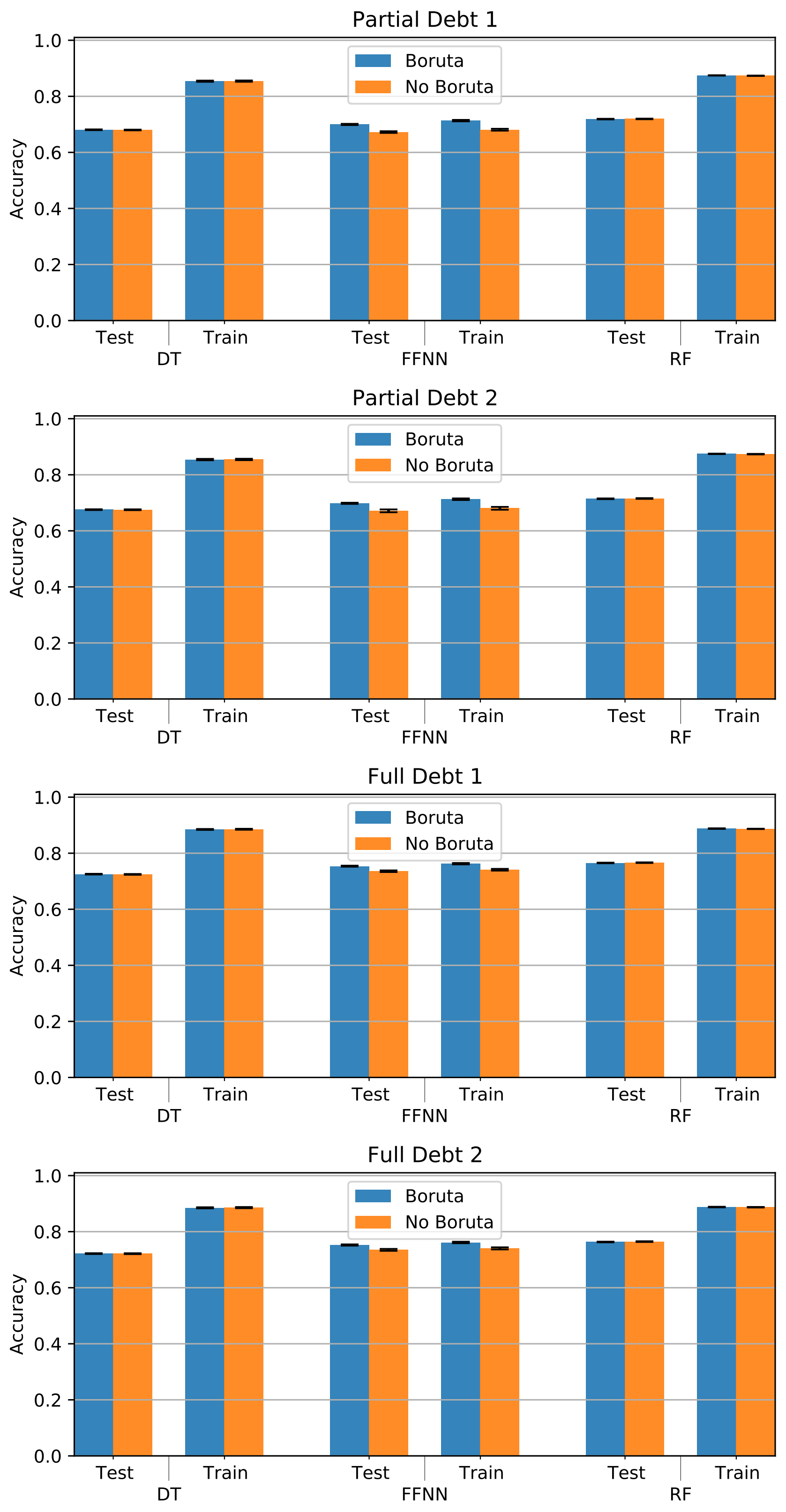


Figure 4.4: **Boruta** vs **No Boruta** comparison for the **Accuracy** metric on the four classification problems presented in 4.1.1

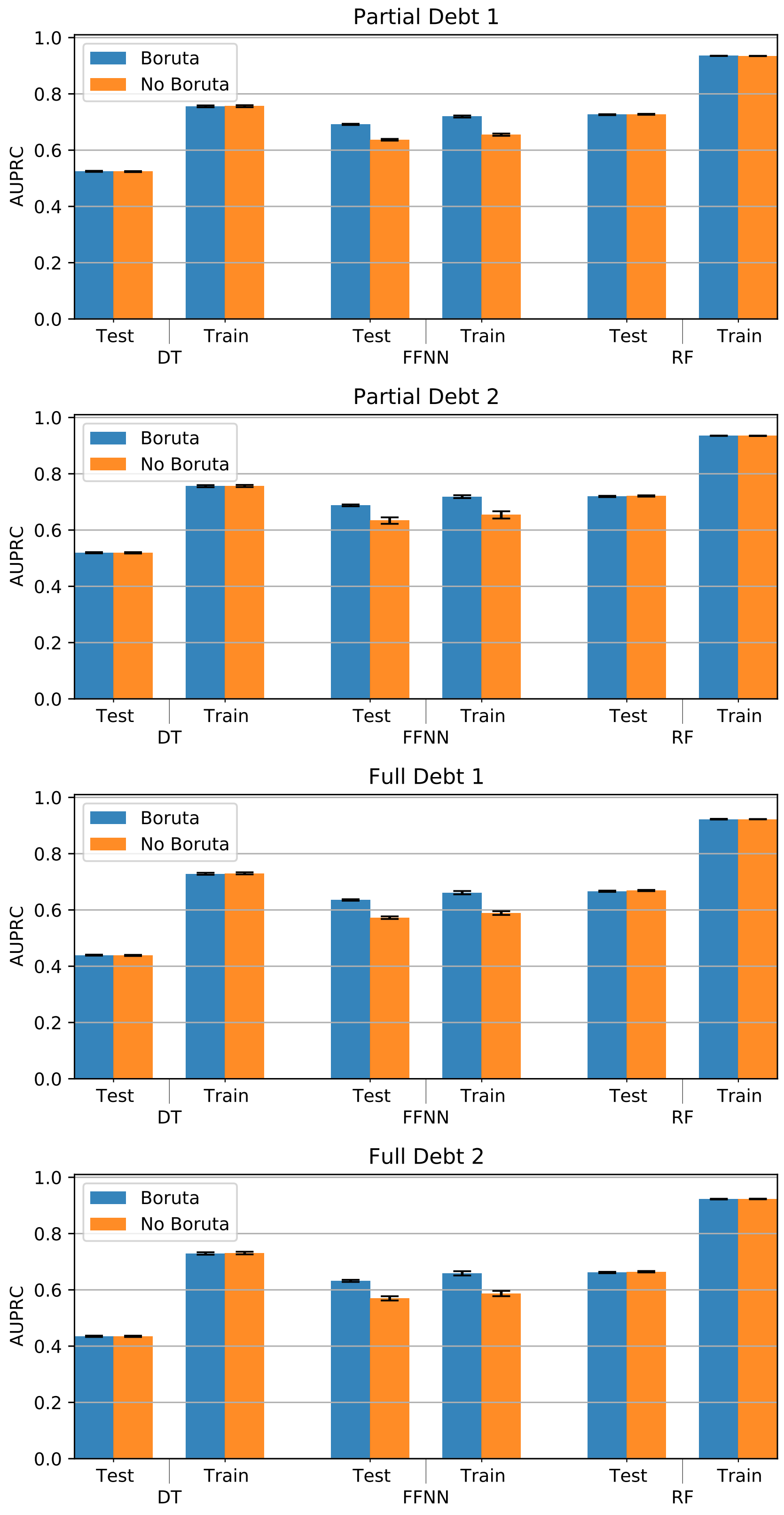


Figure 4.5: **Boruta** vs **No Boruta** comparison for the **AUPRC** metric on the four classification problems presented in 4.1.1

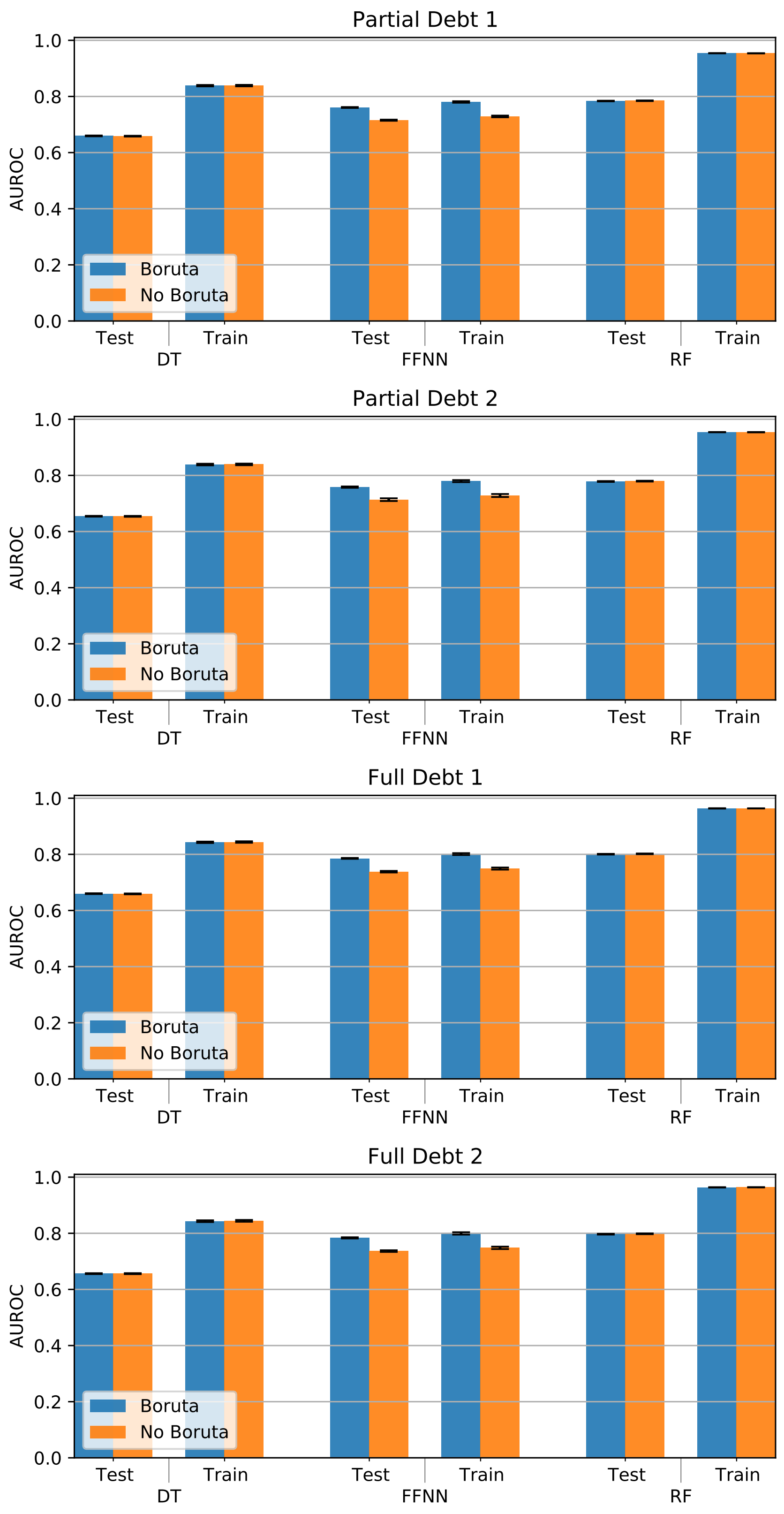


Figure 4.6: **Boruta** vs **No Boruta** comparison for the **AUROC** metric on the four classification problems presented in 4.1.1

While keeping in mind that the dimensionality of the dataset is not particularly high, it is still expected that by performing feature selection, some features will not be considered useful for the classification tasks. More precisely, by feeding the sanitized dataset to Boruta, the following features are thrown away:

1. `valore_veicolo`
2. `ente_A.R.S.S.A. AGENZIA REGIONALE PER LO SVILUPPO E PER I SERVIZI IN AGRICOLTURA IN LIQUIDAZIONE`
3. `ente_FIUMEFREDDO BRUZIO`
4. `ente_MARANO MARCHESATO`

Although all the bodies are encoded from us as explained in Section 2.3.1, there are some of them that are rejected from the algorithm. This might suggest that not all the bodies created are improving the performance of the model and for this reason some of them should be dropped. In addition to this, it is worth saying that we arbitrarily set a number of maximum iterations for the execution of the algorithm. The number of iterations can always be modified, but since a trade-off between the time and the results should be met, we set the maximum number of iterations = 15 and the number of estimators used by the ensemble method = *auto*. The reason why these two hyperparameter values are chosen is because they allow us to end the algorithm in a reasonable amount of time without losing too much in performance. This implies that by setting a higher value for the iterations, a more accurate convergence can be reached and as a consequence also the list of the rejected features might slightly change.

What should suggest us to carry out more experiments is the rejection of the feature `valore_veicolo` instead. In fact, since the original dataset would have missing values on this attribute for most of the entries, we decided to fill it by imputing the median value as mentioned at the end of Section 2.3.3. The fact that Boruta throws `valore_veicolo` away might imply that the imputation of the median for too many records would not bring any useful value to the prediction of the classification problems. That is why it is worth running Boruta again on the entries of the dataset where `valore_veicolo` is not imputed in order to have as much as possible an unbiased result.

It turns out that even with only the entries where `valore_veicolo` is not imputed (around 150k records), Boruta rejects this feature. This would apparently suggest that `valore_veicolo` is not a high-predictive feature for the classification problems we are trying to address and should therefore be dropped.

To understand and validate the insights obtained from Boruta, also here the *Wilcoxon signed-rank* test is performed. As already mentioned, this is a non-parametric test for paired data, that in this case are results data *before* and *after* performing Boruta. By running the Wilcoxon test on the results coming from the 100 holdouts with a significance $\alpha = 0.01$, it is possible to make a comparison between the results obtained performing the feature selection and the results obtained without it.

First, let us show the Win-Tie-Loss tables for all the models after having performed Boruta feature selection.

Table 4.8: Win-Tie-Loss table comparing models on the task **Partial Debt 1** with feature selection.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

Table 4.9: Win-Tie-Loss table comparing models on the task **Partial Debt 2** with feature selection.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

Table 4.10: Win-Tie-Loss table comparing models on the task **Full Debt 1** with feature selection.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

Table 4.11: Win-Tie-Loss table comparing models on the task **Full Debt 2** with feature selection.

Model	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Decision Tree	0	0	2	0	0	2	0	0	2
Random Forest	2	0	0	2	0	0	2	0	0
Feedforward Neural Network	1	0	1	1	0	1	1	0	1

Random Forest is still the model that better performs and generalizes on the data. However, in contrast to the results presented in Section 4.2.1, by running Boruta, it is possible to build a Feedforward Neural Network (FFNN) that performs better against the Decision Tree for all the tasks. Before, indeed, FFNN was outperformed on two out of four tasks by the Decision Tree.

As a final step, it would be interesting now to separately compare each Machine Learning model performance *before* and *after* Boruta. In fact, thanks to this comparison it is possible to understand whether running Boruta has a positive impact on the performance of the models or not.

Decision Tree

In Table 4.12, the comparison coming from the execution of the *Decision Tree* model *before* and *after* Boruta for each task is shown. Performing Boruta on Decision Tree has a positive impact for three out of four classification tasks. In *Full Debt 2*, a *Tie* for the metrics AUPRC and AUROC is obtained. This would suggest that running Boruta on this task does not show any improvement than when it is not run. Or more formally, the performance of the models are statistically identical in this case.

Table 4.12: Win-Tie-Loss table comparing the four classification tasks performed on **Decision Tree** for **Boruta vs No Boruta**

Task	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Partial Debt 1	1	0	0	1	0	0	1	0	0
Partial Debt 2	1	0	0	1	0	0	1	0	0
Full Debt 1	1	0	0	1	0	0	1	0	0
Full Debt 2	1	0	0	0	1	0	0	1	0

Random Forest

In Table 4.13, the comparison coming from the execution of the *Random Forest* model *before* and *after* Boruta for each task is shown. Here, running Boruta does not seem to have any real advantage. In fact, better performance are obtained without feature selection for all the tasks.

Table 4.13: Win-Tie-Loss table comparing the four classification tasks performed on **Random Forest** for **Boruta vs No Boruta**

Task	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Partial Debt 1	0	0	1	0	0	1	0	0	1
Partial Debt 2	0	0	1	0	0	1	0	0	1
Full Debt 1	0	0	1	0	0	1	0	0	1
Full Debt 2	0	0	1	0	0	1	0	0	1

Feedforward Neural Network

In Table 4.14, the comparison coming from the execution of the *Feedforward Neural Network* model *before* and *after* Boruta for each task is shown. Here, running Boruta improves the performance for all the tasks.

Table 4.14: Win-Tie-Loss table comparing the four classification tasks performed on **Feedforward Neural Network** for **Boruta vs No Boruta**

Task	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Partial Debt 1	1	0	0	1	0	0	1	0	0
Partial Debt 2	1	0	0	1	0	0	1	0	0
Full Debt 1	1	0	0	1	0	0	1	0	0
Full Debt 2	1	0	0	1	0	0	1	0	0

4.2.3 Predictions using embedded methods for feature selection

In section 3.5, feature selection was introduced and two techniques would be eventually performed:

- Boruta
- Embedded methods

So far, the results obtained performing feature selection with Boruta have been shown and commented. In Section 3.5.2, the focus of our discussion moved to embedded methods. As seen, embedded methods are already equipped with what needed to rank the features based on their importance. This is particularly useful, since such information is already available when holdouts are computed.

While Boruta was performed on all the models selected in Section 3.1, from now on, we will exclusively focus our attention on the model that outperformed the others: *Random Forest*.

In fact, one of the goals of this dissertation, is to provide a final model that can be used to make predictions on the tasks of interest. That is why it is important that the discussion is narrowed down as much as possible and focused on the tuning of the model that can return the best outcome. This should not be interpreted as the other methods not being good, but at least on this dataset and on these classification tasks, Random Forest is the model that better suits our needs.

The next steps of our analysis are split as follows:

- First, a ranking of the features based on their importance has to be provided
- Then, it would be interesting to understand if a subset of the top k features coming from the embedded method performs better than when all of them are used

- Finally, depending on whether the subset performs better or not, the set of optimal hyperparameters for the model has to be found. This last step though, is covered in the next section, which will take care of the model selection.

Feature ranking

In Figure 4.7, the feature ranking obtained when the embedded method is performed on Random Forest is shown.

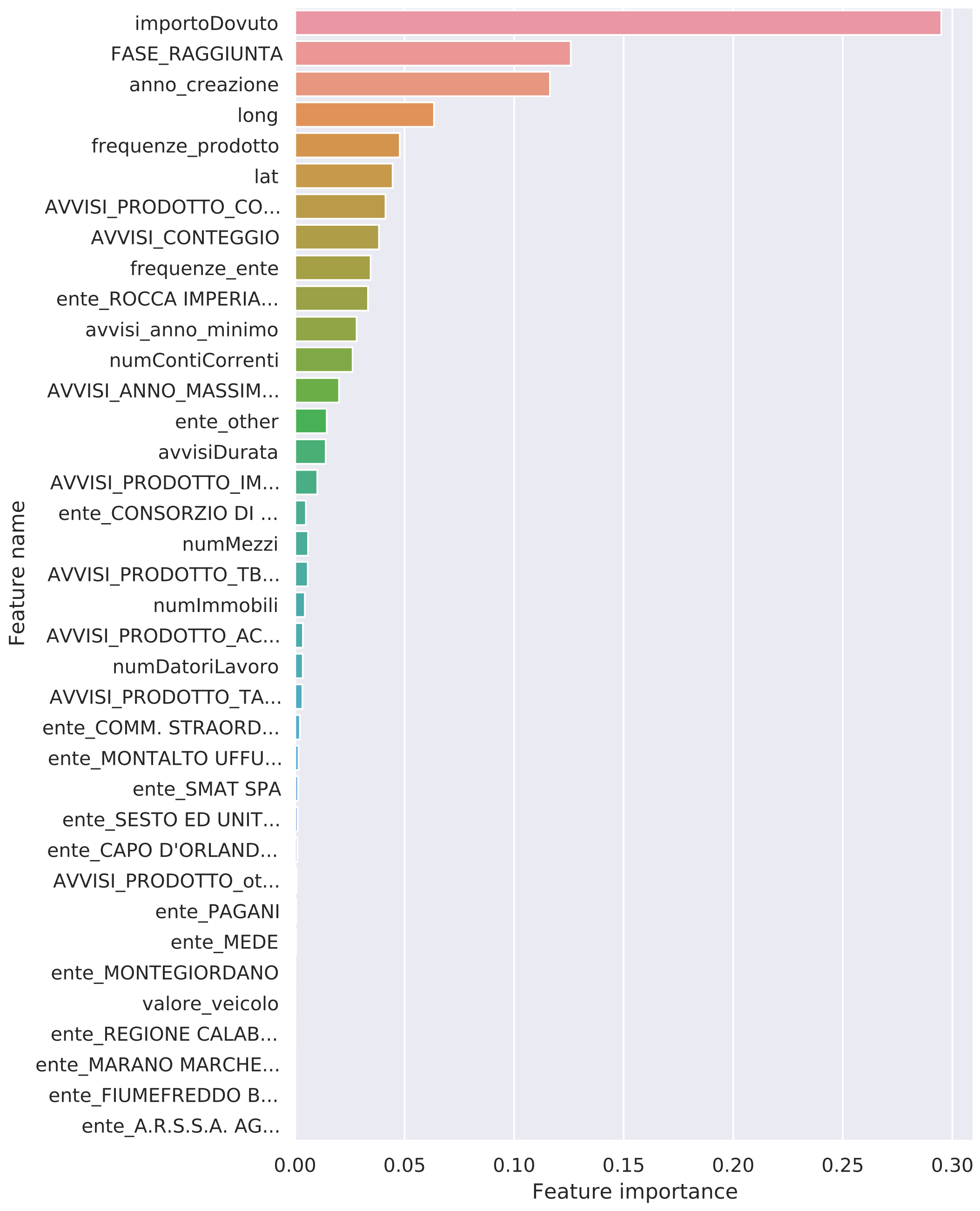


Figure 4.7: Feature ranking obtained with the embedded method.

Here, a couple of things should be observed. In Section 4.2.2, Boruta recognized a list of features that would eventually end up being rejected. Not surprisingly, those same features are in the tail of the ranking shown in Figure 4.7. This is the proof that the features that Boruta marked as not relevant are also among the least informative ones for the embedded method.

What comes in handy now is that by considering a subset of the top k features, it is possible to understand which are the most useful that should provide the best results on the classification tasks.

In fact, had the stakeholder hand over new data, the top k features should be prioritized over the others. In other words, if the stakeholder has to invest time and money to obtain information on the value of the vehicles (i.e. valore_veicolo), thanks to this ranking, we would highly advise to invest their resources into something more valuable as the number of bank accounts owed by the debtor, or even better, specific information about the stage the dossier reached.

Since some of the top features identified were encoded from us (i.e. body names, latitude, longitude, etc.), a filtering of them should be made and properly listed so that as previously explained, the stakeholder can prioritize them over the others. In fact, it would not be wise to prioritize `ente_ROCCA_IMPERIALE` over `avvisi_anno_minimo`, since the first one is obtained after the encoding and strictly depends on this dataset. Had we different data and different bodies, the importances of some of the features listed in Figure 4.7 might rank in different positions. That is why in Figure 4.8, the bodies (i.e. ente_) and the type of debts (i.e. AVVISI_PRODOTTO_) are removed.

However, someone might argue that `frequenze_ente`, `frequenze_prodotto`, `lat` and `long` are still included, despite they are encoded only later.

The reason why they are still included in Figure 4.8 is because:

- `lat` and `long` are obtained after the sanitization step is executed and therefore will always be available.
- `frequenze_ente` and `frequenze_prodotto` are computed based on the occurrences of the bodies and type of debts present in the dataset, and also here, they will always be available after the encoding is performed. Furthermore, both features proved to be particularly informative when performing Boruta and also in the case of the embedded method.

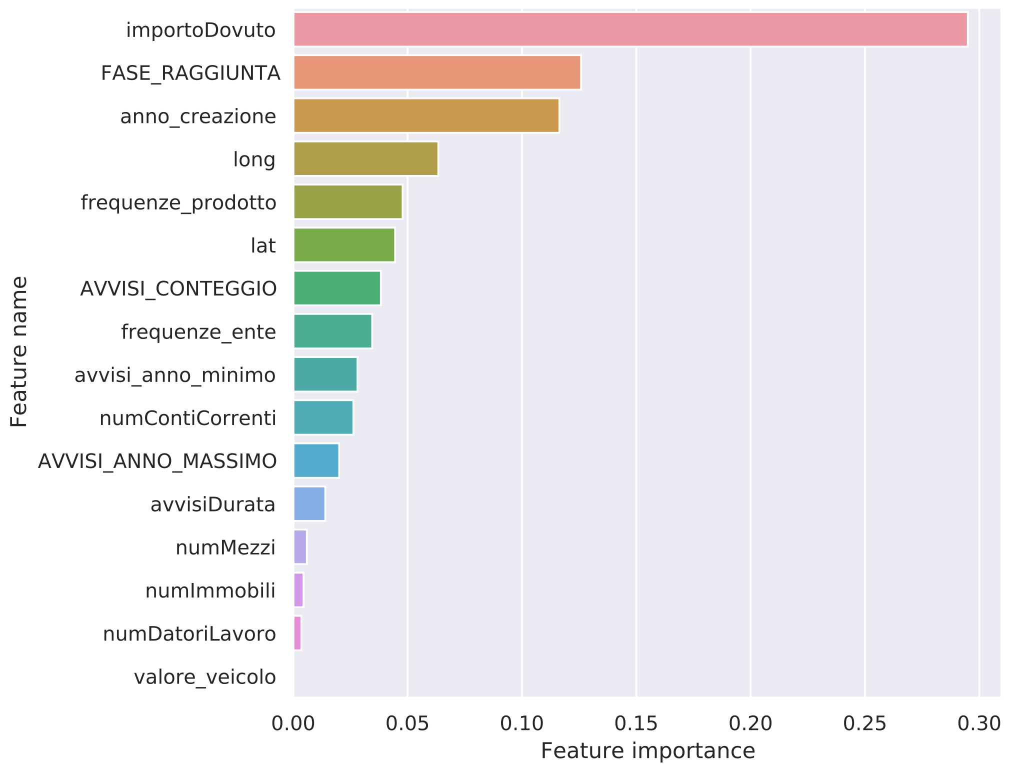


Figure 4.8: Filtered feature ranking obtained with the embedded method.

Experiments on subset of features

Another goal was set at the beginning of this section: understand how a subset of the top k features would perform on the classification tasks. Such threshold k is completely arbitrary. In Figure 4.8, 16 features are listed. Among them, it was already stated that `valore_veicolo` was the least informative.

It is worth saying that what we are trying to do now, carries very little expectations. Our dataset is not particularly large, and therefore our intuition would suggest that cutting down on the number of features used to make predictions, should not provide any real benefits. What is expected here, is that by feeding a subset of features, despite these are the top k as far as their importance, they will worsen the model performance.

For our purposes, the first 12 features shown in Figure 4.8 are selected. The reason why such threshold is chosen is because features from 13 until 16 mainly provide information on *vehicles*, number of *properties* owned and details on the number of *employers*. Information that so far have proved to carry very little value for the predictive tasks. Therefore, it would be worthwhile to understand how much new predictions made on such subset differ from the full list, made of 38 features. In Figures 4.9, 4.10 and 4.11, the results obtained from the execution of this comparison are shown.

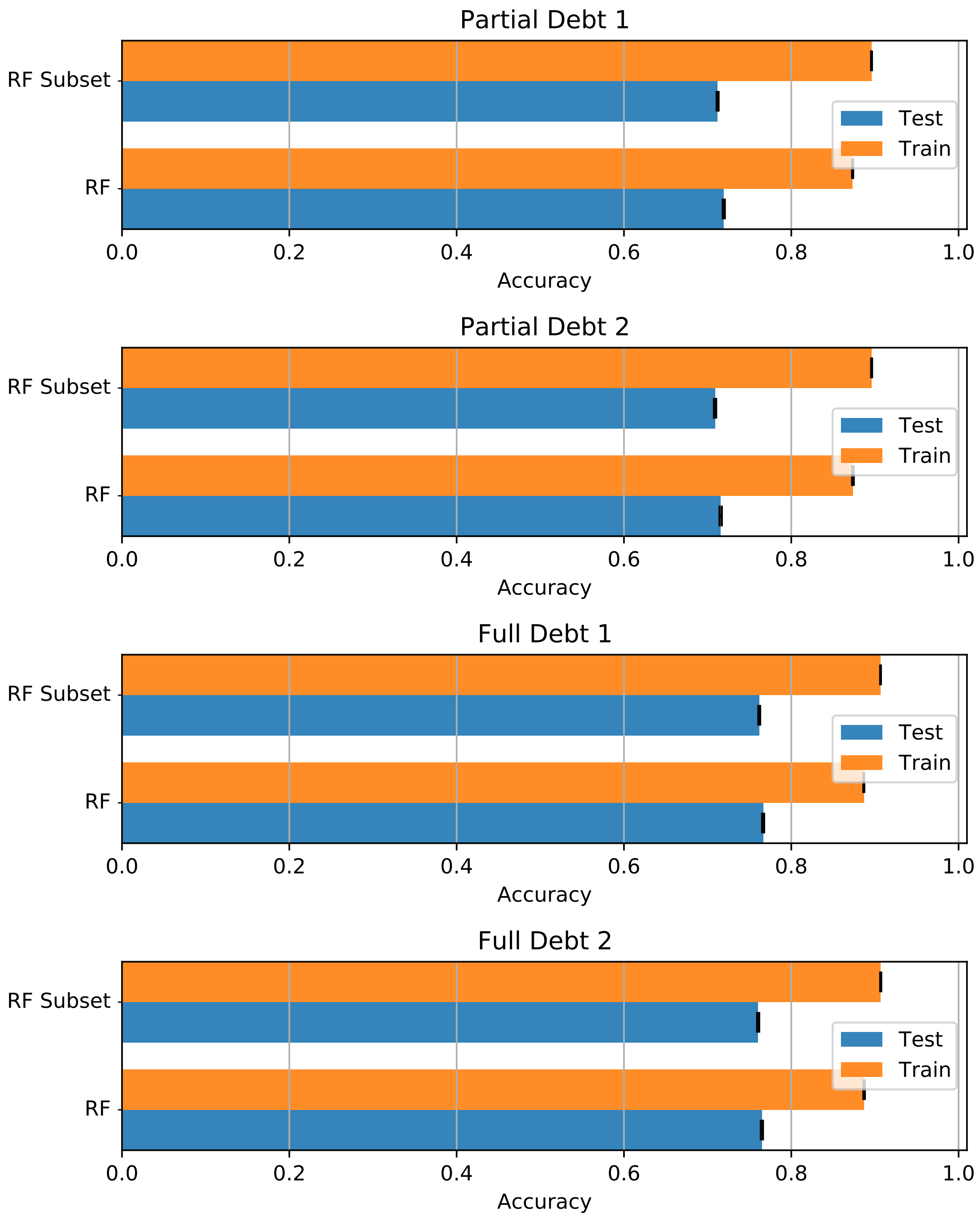


Figure 4.9: Comparison between the results obtained from *Random Forest* and *Random Forest Subset* on the **Accuracy** metric.

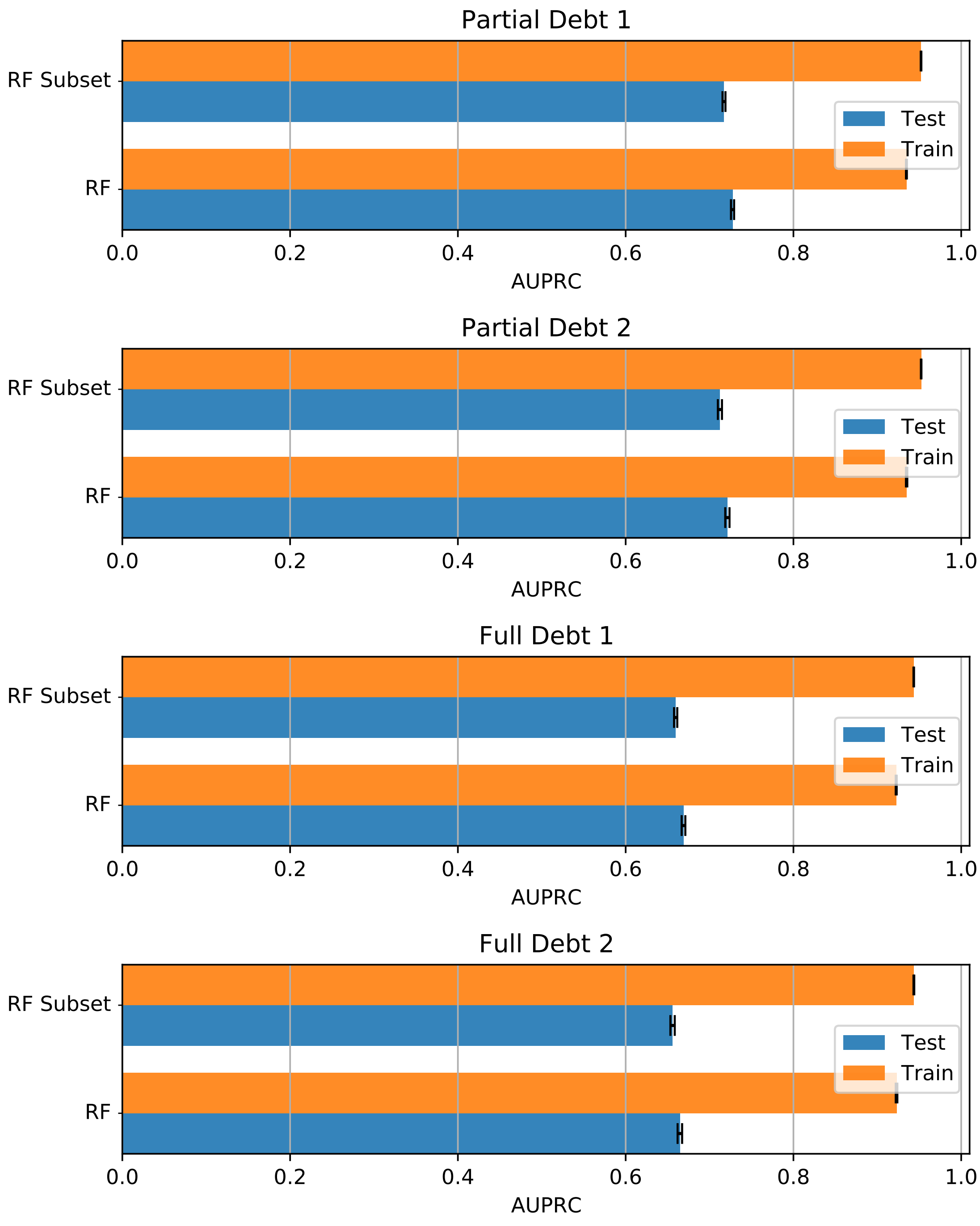


Figure 4.10: Comparison between the results obtained from *Random Forest* and *Random Forest Subset* on the **AUPRC** metric.

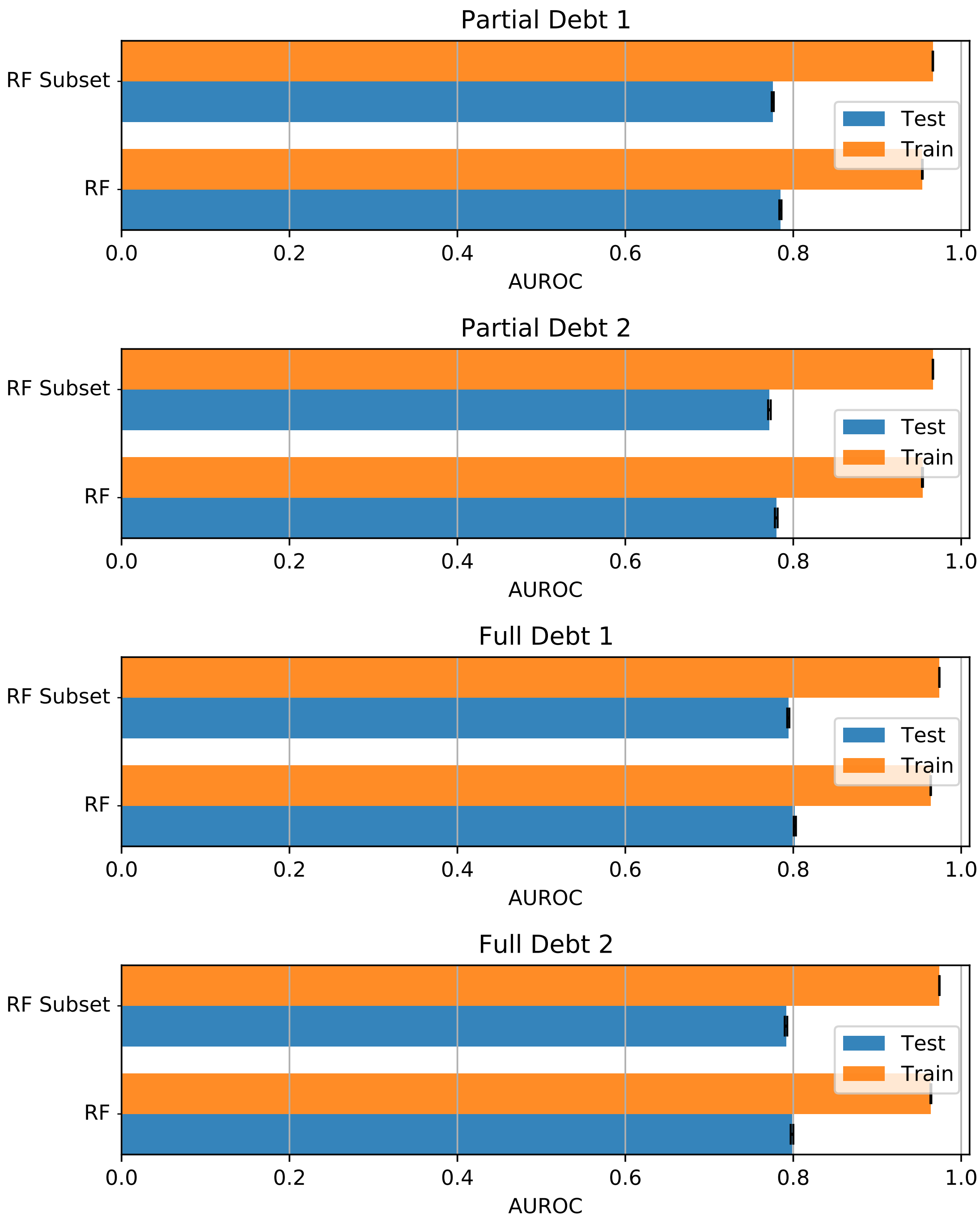


Figure 4.11: Comparison between the results obtained from *Random Forest* and *Random Forest Subset* on the **AUROC** metric.

As expected, better performance are obtained when using all the available features than when just a subset is selected. However, the two feature rankings proposed in Figures 4.7 and 4.8 still prove to be useful in case the stakeholder has to prioritize the purchase of some features over the others.

It is worth noticing that despite only 12 features out of the 38 available are used, still very comparable results are produced. This is interesting because, as already said, for new datasets there might not be available all the 38 features but reliable predictions are still possible to be obtained.

In Tables 4.15, 4.16 and 4.17, the exact performance values for each metric on the train and test set for both models are provided.

Table 4.15: **Accuracy** results on 100 holdouts comparing **Random Forest on all features** vs **Random forest on feature subset**.

Model	Train Accuracy	Test Accuracy	Task
Random Forest (on all features)	**87.33%**	**71.93%**	**Partial Debt 1**
Random Forest (on feature subset)	89.59%	71.19%	Partial Debt 1
Random Forest (on all features)	**87.35%**	**71.53%**	**Partial Debt 2**
Random Forest (on feature subset)	89.60%	70.88%	Partial Debt 2
Random Forest (on all features)	**88.67%**	**76.62%**	**Full Debt 1**
Random Forest (on feature subset)	90.67%	76.17%	Full Debt 1
Random Forest (on all features)	**88.70%**	**76.44%**	**Full Debt 2**
Random Forest (on feature subset)	90.68%	76.04%	Full Debt 2

Table 4.16: **AUPRC** results on 100 holdouts comparing **Random Forest on all features** vs **Random forest on feature subset**.

Model	Train AUPRC	Test AUPRC	Task
Random Forest (on all features)	**93.47%**	**72.74%**	**Partial Debt 1**
Random Forest (on feature subset)	95.21%	71.72%	Partial Debt 1
Random Forest (on all features)	**93.48%**	**72.10%**	**Partial Debt 2**
Random Forest (on feature subset)	95.22%	71.23%	Partial Debt 2
Random Forest (on all features)	**92.25%**	**66.90%**	**Full Debt 1**
Random Forest (on feature subset)	94.34%	65.96%	Full Debt 1
Random Forest (on all features)	**92.29%**	**66.44%**	**Full Debt 2**
Random Forest (on feature subset)	94.35%	65.59%	Full Debt 2

Table 4.17: **AUROC** results on 100 holdouts comparing **Random Forest on all features** vs **Random forest on feature subset**.

Model	Train AUROC	Test AUROC	Task
Random Forest (on all features)	**95.37%**	**78.47%**	**Partial Debt 1**
Random Forest (on feature subset)	96.62%	77.55%	Partial Debt 1
Random Forest (on all features)	**95.38%**	**77.95%**	**Partial Debt 2**
Random Forest (on feature subset)	96.63%	77.15%	Partial Debt 2
Random Forest (on all features)	**96.37%**	**80.18%**	**Full Debt 1**
Random Forest (on feature subset)	97.39%	79.40%	Full Debt 1
Random Forest (on all features)	**96.39%**	**79.83%**	**Full Debt 2**
Random Forest (on feature subset)	97.40%	79.13%	Full Debt 2

To gain better insights on the results shown in Figures 4.9, 4.10, 4.11 and Tables 4.15, 4.16 and 4.17, also here the Wilcoxon signed-rank test is performed.

The results are presented in Table 4.18, and come from the execution of the Wilcoxon test on the values obtained from 100 holdouts which are compared with a level of significance $\alpha = 0.01$. The Random Forest executed on the feature subset competes against the other equipped with all the features, and depending on the statistical test results, there might be a *Win*, a *Tie* or a *Loss*.

Table 4.18: Win-Tie-Loss table comparing the four classification tasks performed on **Random Forest Subset** vs **Random Forest**

Task	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Partial Debt 1	0	0	1	0	0	1	0	0	1
Partial Debt 2	0	0	1	0	0	1	0	0	1
Full Debt 1	0	0	1	0	0	1	0	0	1
Full Debt 2	0	0	1	0	0	1	0	0	1

As expected, the Wilcoxon test validates what previously presented. The Random Forest equipped with all the features performs better than the one run on a subset.

4.2.4 Predictions using model selection

As it turns out from the results presented in Section 4.2.3, making predictions while considering the whole list of features produces better results than when just a subset is used. However, it is also noticed that the difference in performance between them is not as high. As far as this discussion is concerned, from now on the whole list of features will be retained to carry out further analyses.

Let us now move the discussion to one of the most significant steps, which was discussed in Section 3.6 and known as *model selection*. As seen, three approaches were presented and one of them is investigated more in depth, the Bayesian Optimization (BO).

Among the three metrics so far presented, it is important that one of them is selected and used to perform its maximization with the BO. For our purposes the AUPRC is selected since it is particularly useful when evaluating datasets in which imbalanced classes are present. Despite the imbalance here is not extremely high, it is still worth to evaluate the BO on this metric rather than on a less sensitive one as in the case of the Accuracy.

In Section 3.6, it turned out that the parameter space was especially relevant for the BO execution. The parameter space in fact, allows the algorithm to evaluate the performance on a wide set of hyperparameters. When the first holdouts were performed, a fixed and completely arbitrary but reasonable set of hyperparameters was chosen. If other combinations had performed better, there would not have been any way to find it out. By defining a larger parameter space, it is possible to get the best out of the model chosen, that in this case, we remember being the Random Forest.

For our purposes, the parameter space is defined and meant to be optimized as follows:

- $350 \leq$ `n_estimators` ≤ 1000: the number of trees to use in the forest
- $15 \leq$ `max_depth` ≤ 40: the maximum depth of each Decision Tree
- $2 \leq$ `min_samples_split` ≤ 30: the minimum number of samples required to split an internal node
- $0.5 \leq$ `max_samples` ≤ 0.999: the number of samples to draw to train each Decision Tree

The reason why these hyperparameters are selected for the *hyperparameter optimization* is because these are the ones that most affect the predictions in Random Forest and are therefore further tested. It is worth to mention that previous tests on a wider parameter space were carried out. In that case, the BO tried to optimize other hyperparameters like:

- The *criterion* to use to measure the quality of a split. Both, the *Gini* and *Entropy* were tested, but it turned out that better performance were obtained with *Gini* and that is why *Entropy* was not further explored.
- The number of *max features* to retain when trying to perform the best split. Also in this case, it turned out that the optimization of this hyperparameter did not bring any significant improvement and its default approach was preferred instead.

As far as the intervals are concerned, these values come from a basic understanding of the model itself and some empirical tests performed before launching the very first holdouts. Wider ranges might be tried but a higher number of init points and iterations should be given.

Speaking of init points and number of iterations, these were set as follows:

- `init_points` $= 10$
- `n_iter` $= 60$

The init points should be thought as random samples drawn on the function to optimize. It should be provided a sufficient number of them, in order not to leave too many regions unexplored. Also in this case, 10 seemed to be an acceptable value given the complexity of our data.

The number of iterations represents how many steps the BO will run after the init points are sampled. While it is left running trying to optimize each one of the four classification tasks, it will at some point produce some maximum. At the end of the whole execution, the set of the optimal hyperparameters, is returned combined with the highest AUPRC score obtained on the test set for the task performed.

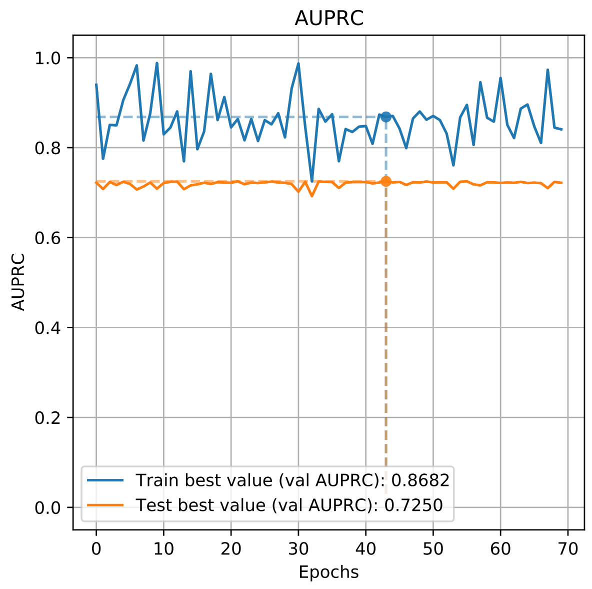
To have a deeper understanding on how the BO is performed, it is important to add some details. The BO is run after the dataset is split into three parts:

- Train set
- Validation set
- Test set

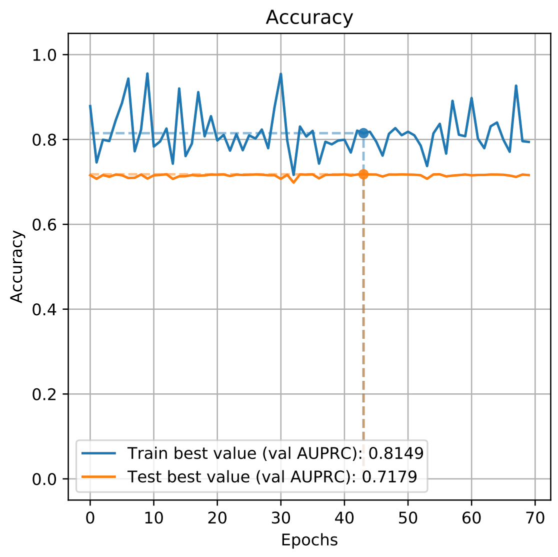
The proportion of these splits are the following: 60% for the train, 20% for the validation and 20% for the test set. More precisely, the model selection is performed by optimizing the AUPRC metric, training on the train set and evaluating on the validation set. For each task, the plots showing the BO execution with its AUPRC optimization, and hyperparameters tuning are respectively presented.

In Figures 4.12, 4.14, 4.16 and 4.18, the metrics AUPRC, Accuracy and AUROC when optimizing over the AUPRC with the Bayesian Optimization on the four classification tasks, are shown. Just like the metrics, in Figures 4.13, 4.15, 4.17 and 4.19, the hyperparameters optimization processes are shown. The iterations that produce the best values on the AUPRC test scores and consequently the optimal hyperparameters, are marked with a circle.

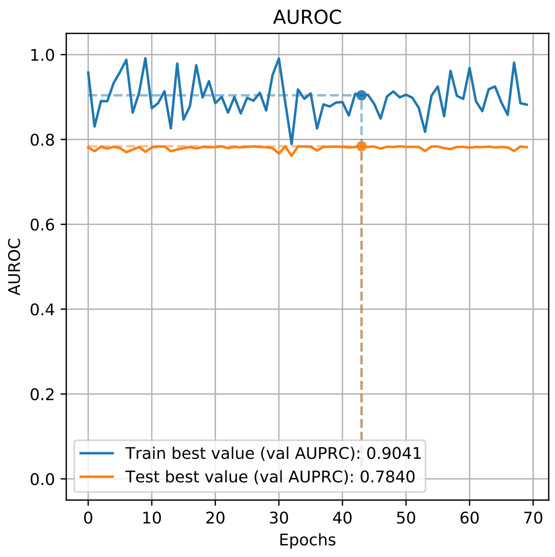
Partial Debt 1



(a) AUPRC

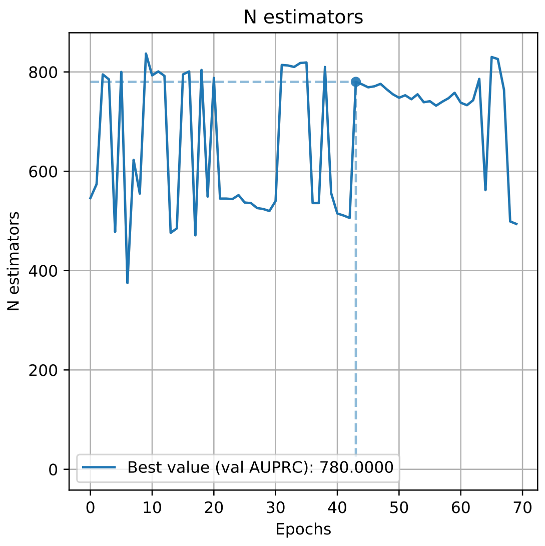


(b) Accuracy

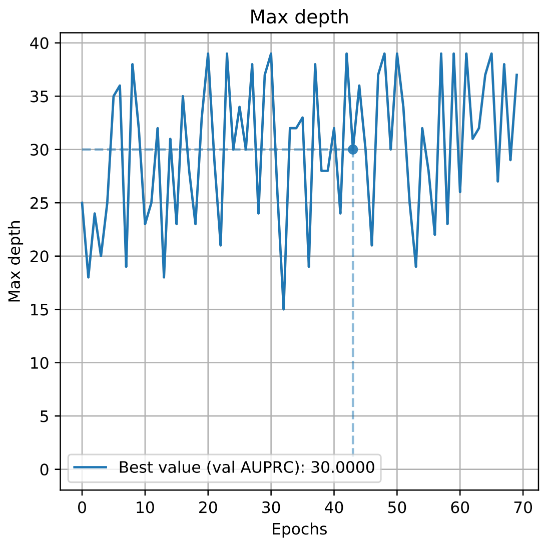


(c) AUROC

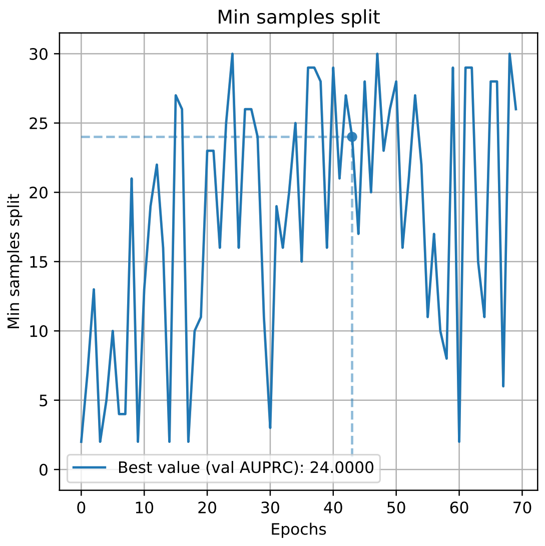
Figure 4.12: Overview of the metrics when optimizing the **AUPRC** metric with the BO on the classification task **Partial Debt 1**.



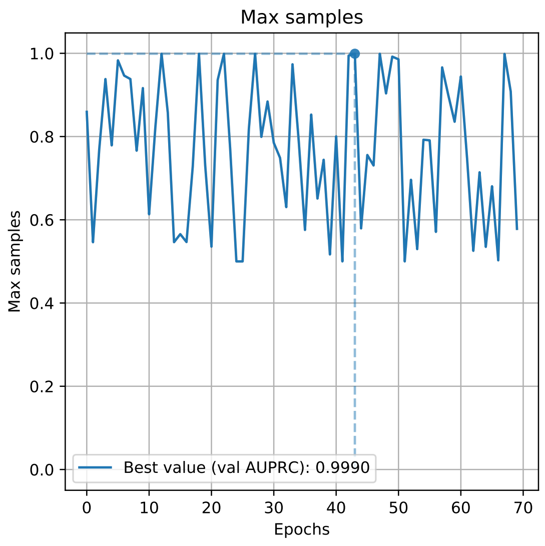
(a) n_estimators



(b) max_depth



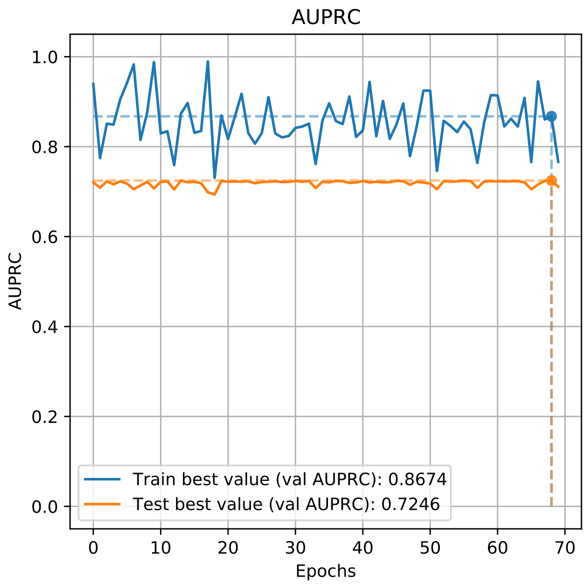
(c) min_samples_split



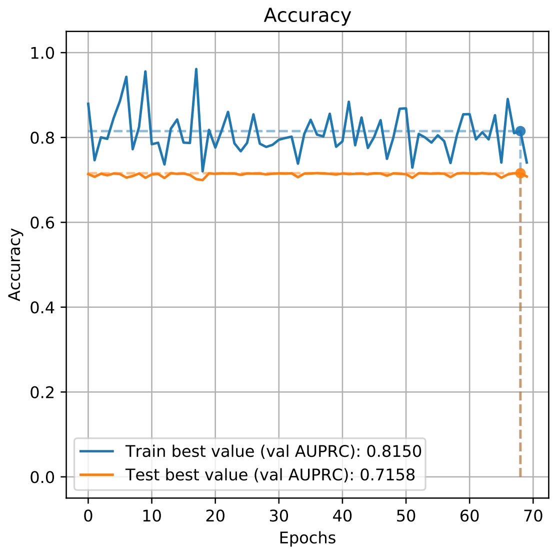
(d) max_samples

Figure 4.13: Overview of the hyperparameters when optimizing the **AUPRC** metric with the BO on the classification task **Partial Debt 1**.

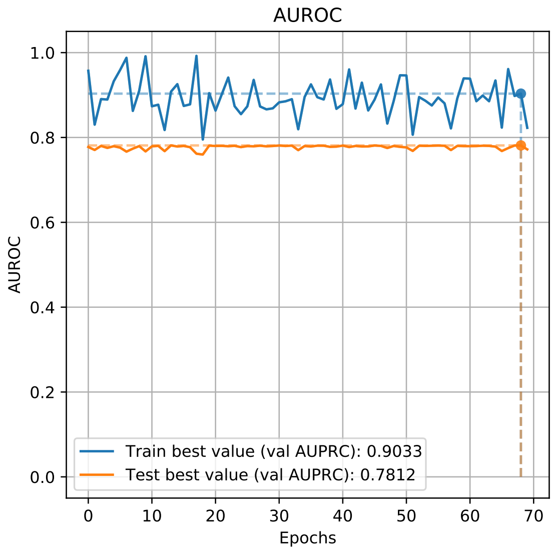
Partial Debt 2



(a) AUPRC

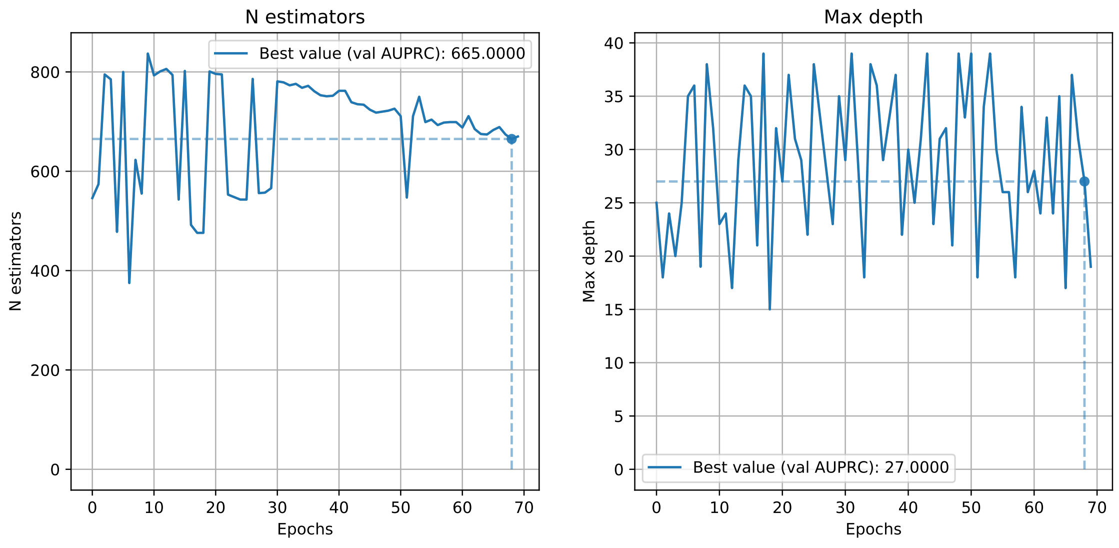


(b) Accuracy



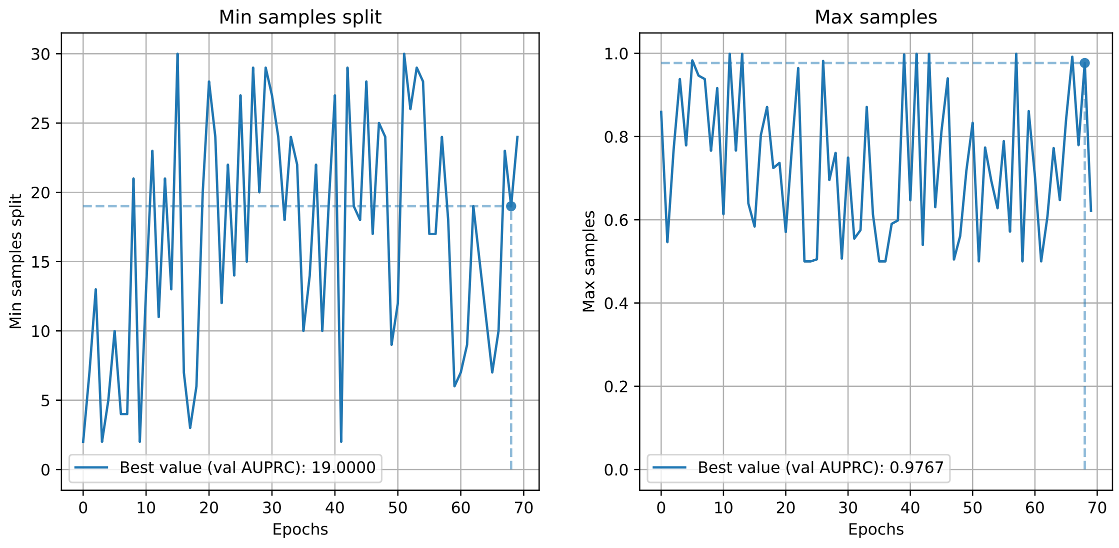
(c) AUROC

Figure 4.14: Overview of the metrics when optimizing the **AUPRC** metric with the BO on the classification task **Partial Debt 2**.



(a) n_estimators

(b) max_depth

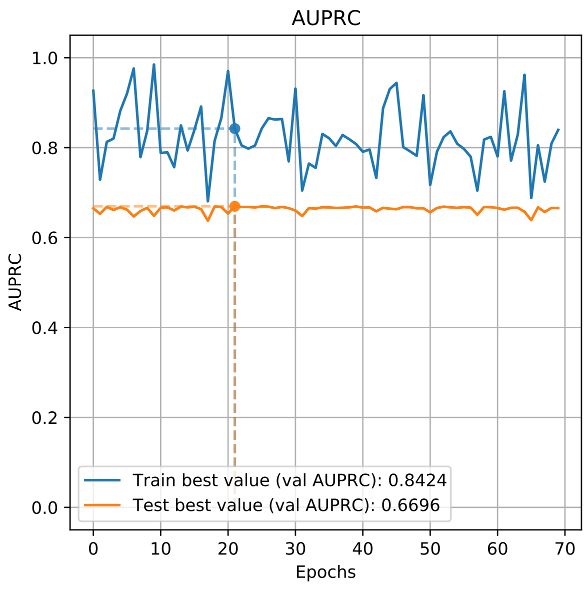


(c) min_samples_split

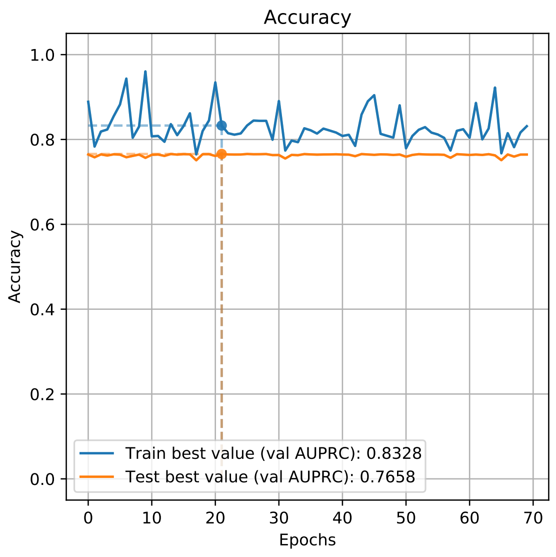
(d) max_samples

Figure 4.15: Overview of the hyperparameters when optimizing the **AUPRC** metric with the BO on the classification task **Partial Debt 2**.

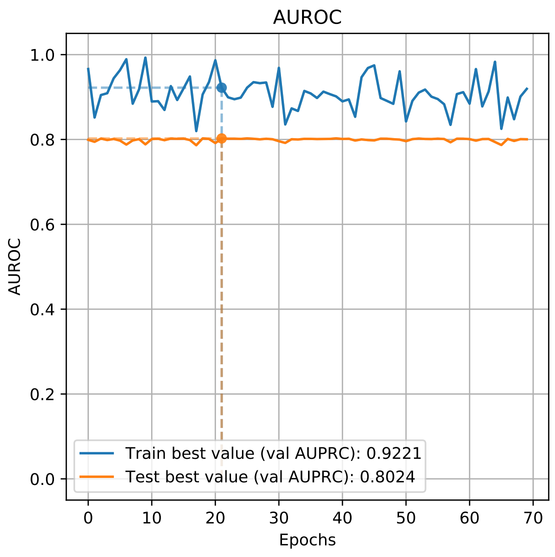
Full Debt 1



(a) AUPRC

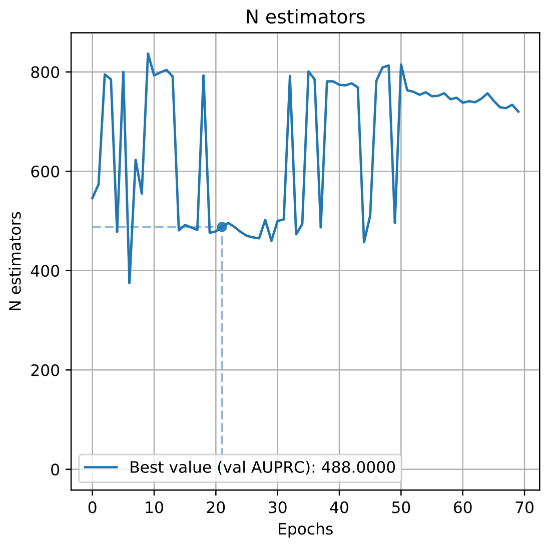


(b) Accuracy

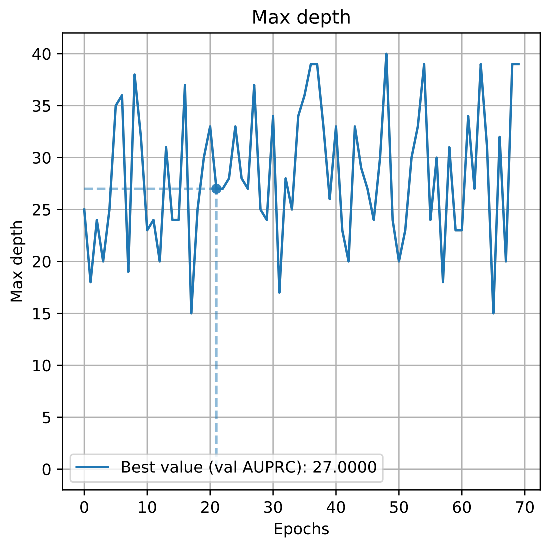


(c) AUROC

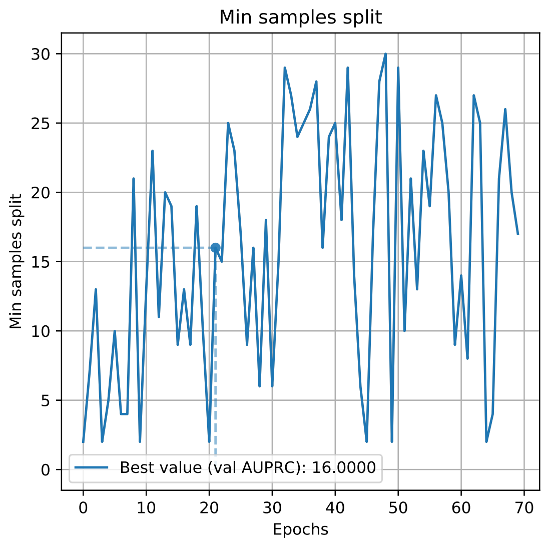
Figure 4.16: Overview of the metrics when optimizing the **AUPRC** metric with the BO on the classification task **Full Debt 1**.



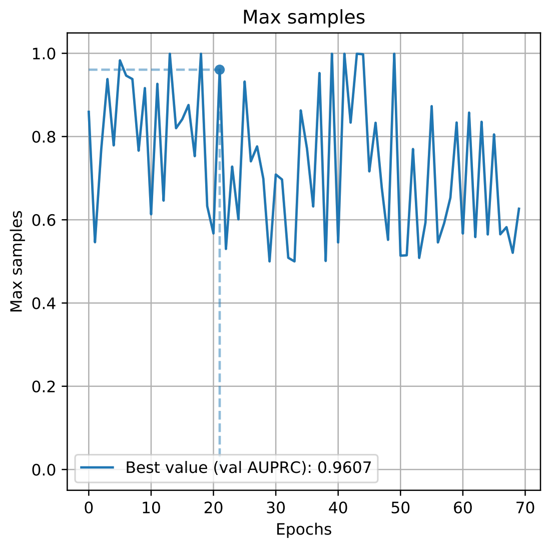
(a) n_estimators



(b) max_depth



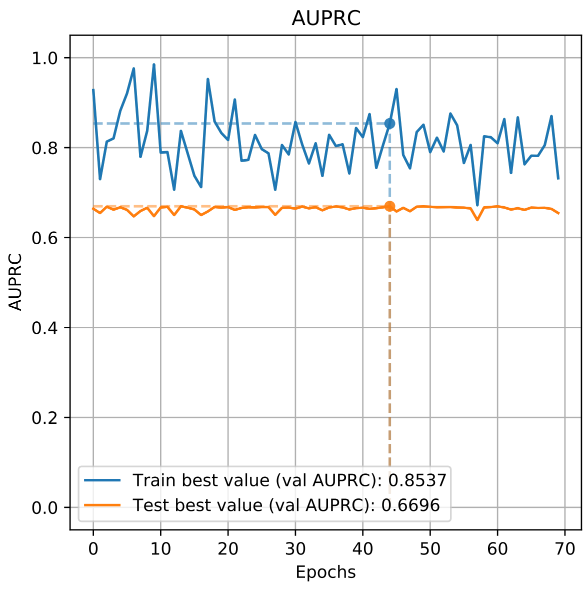
(c) min_samples_split



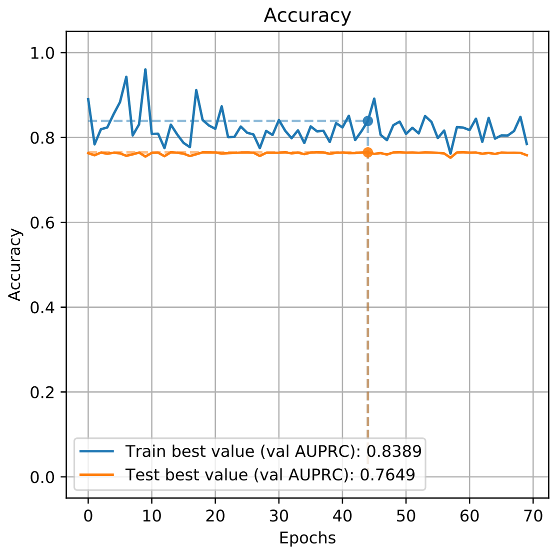
(d) max_samples

Figure 4.17: Overview of the hyperparameters when optimizing the **AUPRC** metric with the BO on the classification task **Full Debt 1**.

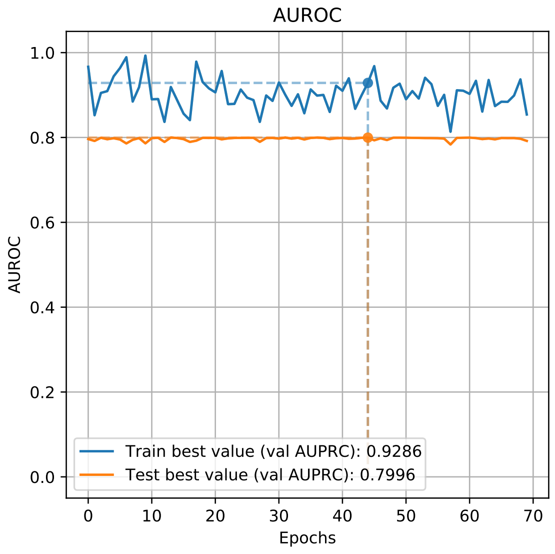
Full Debt 2



(a) AUPRC

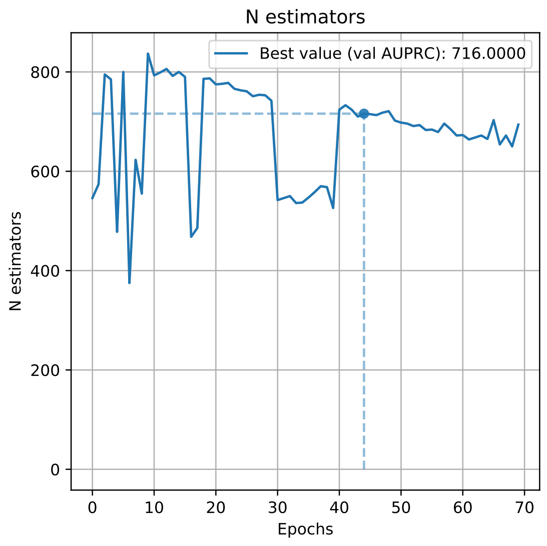


(b) Accuracy

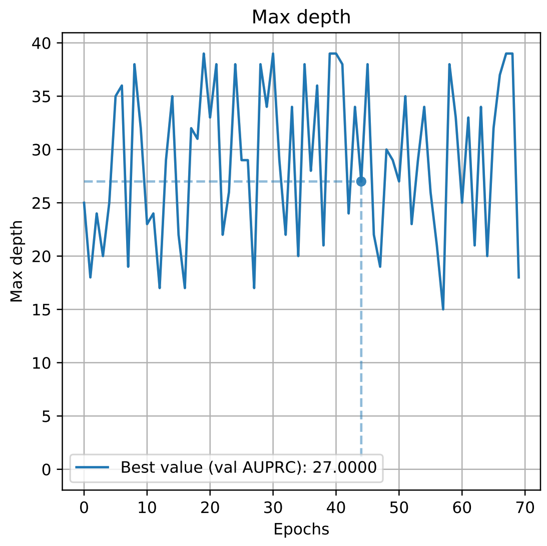


(c) AUROC

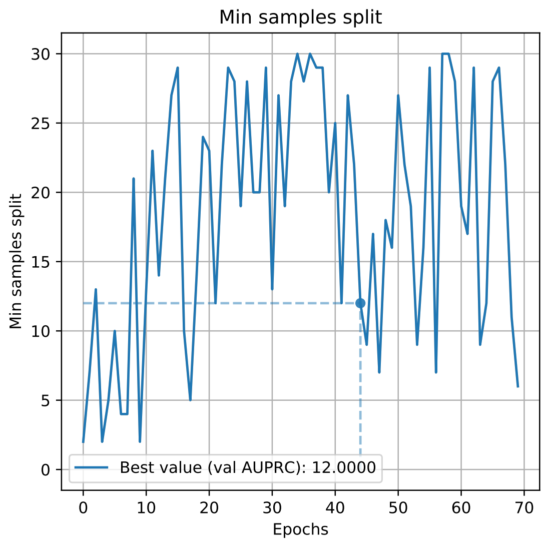
Figure 4.18: Overview of the metrics when optimizing the **AUPRC** metric with the BO on the classification task **Full Debt 2**.



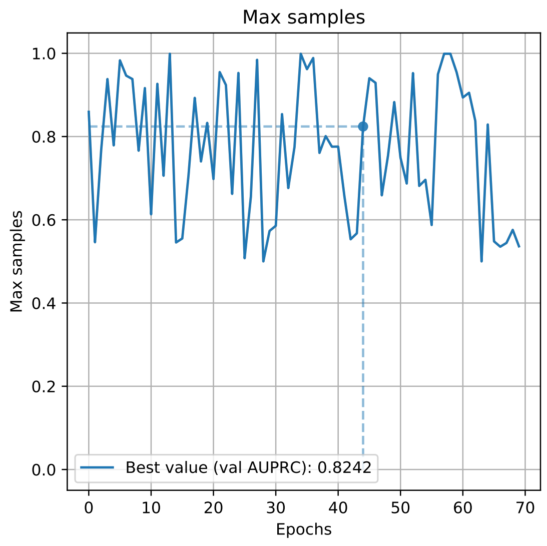
(a) n_estimators



(b) max_depth



(c) min_samples_split



(d) max_samples

Figure 4.19: Overview of the hyperparameters when optimizing the **AUPRC** metric with the BO on the classification task **Full Debt 2**.

Optimal hyperparameters

In Table 4.19, the sets of optimal hyperparameters produced after the Bayesian Optimization (BO), are summarized. It is worth to notice that despite the very first holdouts were run by selecting an arbitrary set of hyperparameters, these have proved to be quite close to the ones produced by the BO.

Table 4.19: Sets of **optimal hyperparameters** obtained from the BO for the four classification tasks on **Random Forest**.

Hyperparameters				**Task**
n_estimators	**max_depth**	**min_samples_split**	**max_samples**	
780	30	24	0.999	Partial Debt 1
665	27	19	0.976	Partial Debt 2
488	27	16	0.960	Full Debt 1
716	27	12	0.824	Full Debt 2

Evaluation on the test set by multiple holdouts

Once the sets of optimal hyperparameters are identified, it is possible to finally evaluate them on the test set for the final models. Also here, multiple executions are necessary in order to have stabler results. In this regard, 10 holdouts for each classification task are performed and compared to the performance obtained with the holdouts run without any model selection.

In Table 4.20, the results obtained by performing *Wilcoxon test* are shown. The Random Forest executed with model selection competes against the one without any optimization with a level of significance $\alpha = 0.05$.

Table 4.20: Win-Tie-Loss table comparing the four classification tasks performed on **Random Forest with model selection** vs **Random Forest no model selection**

Task	**Accuracy**			**AUPRC**			**AUROC**		
	W	**T**	**L**	**W**	**T**	**L**	**W**	**T**	**L**
Partial Debt 1	1	0	0	1	0	0	1	0	0
Partial Debt 2	1	0	0	1	0	0	1	0	0
Full Debt 1	1	0	0	1	0	0	1	0	0
Full Debt 2	1	0	0	1	0	0	1	0	0

It turns out that Random Forests with model selection perform better than when no model selection is carried out. This is quite satisfactory, since the BO allowed us to tune the model as efficiently as possible. In Tables 4.21, 4.22 and 4.23, a final comparison providing performance scores on all the metrics is shown. It is worth to notice that the BO produces sets of hyperparameters that allow to achieve slightly better performance on the test set and lower performance on the train set. This has to be seen as an improvement, since it means that less overfitting on the train set is reached compared to when looser sets of hyperparameters were used, which leads to more robust models.

Table 4.21: **Accuracy** results comparing Random Forest after model selection is performed vs Random Forest without any model selection.

Model	Train Accuracy	Test Accuracy	Task
Random Forest (with model selection)	**81.30%**	**72.15%**	**Partial Debt 1**
Random Forest (no model selection)	87.33%	71.93%	Partial Debt 1
Random Forest (with model selection)	**81.00%**	**71.82%**	**Partial Debt 2**
Random Forest (no model selection)	87.35%	71.53%	Partial Debt 2
Random Forest (with model selection)	**83.07%**	**76.80%**	**Full Debt 1**
Random Forest (no model selection)	88.67%	76.62%	Full Debt 1
Random Forest (with model selection)	**83.60%**	**76.65%**	**Full Debt 2**
Random Forest (no model selection)	88.70%	76.44%	Full Debt 2

Table 4.22: **AUPRC** results comparing Random Forest after model selection is performed vs Random Forest without any model selection.

Model	Train AUPRC	Test AUPRC	Task
Random Forest (with model selection)	**86.51%**	**73.08%**	**Partial Debt 1**
Random Forest (no model selection)	93.47%	72.74%	Partial Debt 1
Random Forest (with model selection)	**86.25%**	**72.57%**	**Partial Debt 2**
Random Forest (no model selection)	93.48%	72.10%	Partial Debt 2
Random Forest (with model selection)	**83.69%**	**67.46%**	**Full Debt 1**
Random Forest (no model selection)	92.25%	66.90%	Full Debt 1
Random Forest (with model selection)	**84.76%**	**66.95%**	**Full Debt 2**
Random Forest (no model selection)	92.29%	66.44%	Full Debt 2

Table 4.23: **AUROC** results comparing Random Forest after model selection is performed vs Random Forest without any model selection.

Model	Train AUROC	Test AUROC	Task
Random Forest (with model selection)	**90.14%**	**78.80%**	**Partial Debt 1**
Random Forest (no model selection)	95.37%	78.47%	Partial Debt 1
Random Forest (with model selection)	**89.94%**	**78.35%**	**Partial Debt 2**
Random Forest (no model selection)	95.38%	77.95%	Partial Debt 2
Random Forest (with model selection)	**91.88%**	**80.53%**	**Full Debt 1**
Random Forest (no model selection)	96.37%	80.18%	Full Debt 1
Random Forest (with model selection)	**92.50%**	**80.18%**	**Full Debt 2**
Random Forest (no model selection)	96.39%	79.83%	Full Debt 2

Chapter 5

Conclusion

Main goal of the following work was to provide insights on the topic of debt solvency by using Machine Learning models. To our knowledge in fact, this topic has not been covered in literature. Such problem can be formalized as a classification task and can be therefore addressed with Supervised learning algorithms. Two classification tasks were eventually identified: the partial and the full debt solvency. The first one aimed to predict whether or not a debtor is likely to partially pay their debt back. The second one aimed to predict whether or not the due amount is completely returned. To carry out the following tasks, Machine Learning models like Decision Tree, Random Forest and Feedforward Neural Network were used. Feature selection with different techniques was performed, and it did not eventually produce any significant advantage compared to when it was not used. In order to obtain reliable results, multiple holdouts were executed on each model. It turned out that Random Forest outperformed the other models and for this reason was selected for further analyses. Random Forests, as well as most of the Machine Learning models, require several hyperparameters in order to run, and that is why model selection with Bayesian Optimization was carried out. The Bayesian Optimization in fact, allows to select the optimal set of hyperparameters that maximize a metric. For the purpose of this work, the AUPRC was eventually maximized. It turned out that Random Forest after performing model selection, was able to achieve 72.15% of accuracy on the partial debt solvency and 76.80% on the full debt solvency. As known, classification and regression are the two possible approaches when dealing with Supervised learning problems. Here, the whole discussion was exclusively based on classification. Another task that it would be interesting to further address is a regression one. That is why future development might focus more on this approach. More precisely, it would be of interest to predict a time window in which the debt will be eventually settled. Such problem can be clearly defined also as a classification task where the classes to predict are not just two as in this dissertation but might be several. And finally, an issue that would be also interesting to cover more in detail is related to the explainability. Explainability represents a big problem when trying to understand the results provided by Machine Learning models, and that is why further experiments with more interpretable models like Additive Tree [16] might be carried out to gain clearer business insights.

References

[1] Peter Martey Addo, Dominique Guegan, and Bertrand Hassani. Credit risk analysis using machine and deep learning models. *Risks*, 6(2):38, 2018.

[2] Jodi L Bellovary, Don E Giacomino, and Michael D Akers. A review of bankruptcy prediction studies: 1930 to present. *Journal of Financial education*, pages 1–42, 2007.

[3] Antonio Blanco, Rafael Pino-Mejías, Juan Lara, and Salvador Rayo. Credit scoring models for the microfinance industry using neural networks: Evidence from peru. *Expert Systems with applications*, 40(1):356–364, 2013.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Chris Charalambous, Andreas Charitou, and Froso Kaourou. Comparative analysis of artificial neural network models: Application in bankruptcy prediction. *Annals of operations research*, 99(1-4):403–425, 2000.

[6] Travis CI. Travis ci - test and deploy with confidence.

[7] Machine Learning Crash Course Google Developers. Classification: Roc curve and auc.

[8] Augustinos I Dimitras, Stelios H Zanakis, and Constantin Zopounidis. A survey of business failures with an emphasis on prediction methods and industrial applications. *European Journal of Operational Research*, 90(3):487–513, 1996.

[9] geopy. geopy - pypi.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] David J Hand and William E Henley. Statistical classification methods in consumer credit scoring: a review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3):523–541, 1997.

[12] P Ravi Kumar and Vadlamani Ravi. Bankruptcy prediction in banks and firms via statistical and intelligent techniques–a review. *European journal of operational research*, 180(1):1–28, 2007.

[13] Miron B Kursa, Witold R Rudnicki, et al. Feature selection with the boruta package. *J Stat Softw*, 36(11):1–13, 2010.

[14] Hui Li, Jie Sun, and Jian Wu. Predicting business failure using classification and regression tree: An empirical comparison with popular classical statistical methods and top classification mining methods. *Expert Systems with Applications*, 37(8):5895–5904, 2010.

[15] Christian Lohmann and Thorsten Ohliger. The total cost of misclassification in credit scoring: A comparison of generalized linear models and generalized additive models. *Journal of Forecasting*, 38(5):375–389, 2019.

[16] José Marcio Luna, Efstathios D Gennatas, Lyle H Ungar, Eric Eaton, Eric S Diffenderfer, Shane T Jensen, Charles B Simone, Jerome H Friedman, Timothy D Solberg, and Gilmer Valdes. Building more accurate decision trees with the additive tree. *Proceedings of the National Academy of Sciences*, 116(40):19887–19893, 2019.

[17] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[19] pgeocode. pgeocode - pypi.

[20] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[21] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[22] scikit-learn contrib. Borutapy.

[23] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[25] Sergey Tulyakov, Stefan Jaeger, Venu Govindaraju, and David Doermann. Review of classifier combination methods. In *Machine learning in document analysis and recognition*, pages 361–386. Springer, 2008.

[26] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5(Jul):725–775, 2004.

[27] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. In *Italian workshop on neural nets*, pages 3–20. Springer, 2002.

[28] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[29] Christopher KI Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682–688, 2001.