University of Rome Tor Vergata

MASTER'S OF DATA SCIENCE

Thesis in

Applied Machine Learning

Machine Learning techniques for contracts lifecycle prediction in Enel

Federico Francone

Academic Year 2018-2019

Supervisor: Giorgio Gambosi

Co-Advisors: Paola Console, Nicola Sacchi

Ringraziamenti

Vorrei ringraziare prima di tutto la mia famiglia, che mi è stata sempre vicino con calore e affetto. In particolare ci tengo a ringraziare i miei genitori, che con il loro aiuto hanno reso possibile tutto il mio percorso universitario.

Voglio poi esprimere la mia gratitudine ai colleghi del Master, soprattutto quelli che con me hanno condiviso momenti di gioia ma anche di difficoltà e soddisfazione; in particolare Daniele per aver allietato lo studio di tutti gli esami con la sua mentalità sempre positiva, Andrea per tutte le risate e il supporto nelle situazioni più particolari.

Allo stesso modo, ci tengo a ringraziare i colleghi dell'Enel, specialmente Josè, Iannis, Esin e Anna, che hanno reso decisamente più piacevole il mio periodo in azienda. Ringrazio anche i miei amici storici Dandi, Bruno e Sofia.

Infine, un ringraziamento speciale a tutte quelle persone che sono per me un solido e costante punto di riferimento, sempre pronte a dare senza aspettarsi nulla in cambio: Clarissa, Franci, Andrea e Bea.

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Research objective . . . 2
1.3 Process Description . . . 2
1.3.1 Static and Rolling approaches . . . 3
1.3.2 Predictions Pipeline . . . 5

2 Data Preprocessing **6**
2.1 Raw Data Description . . . 6
2.2 Dataset Construction . . . 6
2.2.1 Dataset Aggregation . . . 7
2.3 Data Cleaning . . . 8
2.4 Feature Engineering . . . 9
2.5 Target . . . 11
2.6 Missing Data Imputation . . . 12
2.7 Handling Categorical Variables . . . 12
2.7.1 One Hot Encoding . . . 13
2.7.2 Clustering Categories . . . 13
2.7.3 K-Fold Target Encoding . . . 14
2.7.4 Categorical Embedding with Neural Networks . . . 15
2.7.5 Final choice to handle categories . . . 15
2.8 Descriptive Statistics . . . 16
2.8.1 Static . . . 16
2.8.2 Rolling . . . 20
2.9 Feature Scaling . . . 22
2.10 Preliminary Feature Selection . . . 23
2.11 Data Resume . . . 24
2.12 Train Test Split . . . 24

3 Machine Learning Models **25**
3.1 Algorithms Overview . . . 25
3.2 Hyperparameter Tuning approach . . . 26
3.2.1 Dimensionality Reduction . . . 26
3.3 Classification Task . . . 27
3.3.1 Static . . . 28
3.3.2 Rolling . . . 33

3.4 Regression Task 36
3.4.1 Static 37
3.4.2 Rolling 38
3.5 Summary Best Results 41
3.6 Single Lifetime Approach 41
3.6.1 Relevant Comparisons 43

4 Deep Learning Models 45
4.1 Overview 45
4.2 Optimization Approach 46
4.3 Classification Task 48
4.3.1 Custom Loss Function 48
4.3.2 Static Results 49
4.3.3 Rolling Results 52
4.4 Regression Task 53
4.4.1 Static Scores 53
4.4.2 Rolling Scores 55
4.5 Recurrent Neural Networks 55
4.5.1 Reshaping Data 55
4.5.2 RNN Results 57
4.6 Multi-Task learning: Two outputs model 58
4.6.1 Filtering Choices 58
4.6.2 Undersampling 60

5 Conclusions 61
5.1 Most performing models 62
5.2 Further studies 64

A Codes
A.1 Custom K-Fold Target Encoding Class . . . . . .
A.2 Feature Engineering and Lifetime cuts creation . . . . . .
A.3 Embedding Preprocessing . . . . . .
A.4 Recurrent Neural Networks Preprocessing . . . . . .
A.5 Run Grid Search with scores summary . . . . . .

B Figures

Bibliography

List of Figures

1.1 Orders irregularity shown through a specific consumption curve. . . 2

2.1 Elbow curve showing how many contracts (Y axis) exceed the *Header Amount* by the factors reported on the X axis. 9
2.2 Aggregated data to cluster suppliers. 13
2.3 Categorical Embedding example. 16
2.4 Cardinalities of categorical features 17
2.5 Label Distribution. Fully consumed contracts are ~44% of the dataset. 17
2.6 %Used (left) and %Life (right). 18
2.7 Life % passed until first order. On the left related to all the contracts, while on the right only for those entirely consumed (>90%) at the 1st order. 18
2.8 Numerical Features Distributions. 18
2.9 Numerical features in relation to target. 19
2.10 Average label of most frequent classes. 19
2.11 Regression Target distribution. 20
2.12 Label Distribution of Rolling dataset. 20
2.13 BoxPlots of % used and % life for Rolling data. 21
2.14 Variables distributions after the Encoding (2.7). 21
2.15 Correlations of features with target after Encoding (2.7). 22
2.16 Average interval between two orders (left) and standard deviation (right) in respect to life%. 22

3.1 Cumulative Explained Variance. 27
3.2 Dataset plotted in two dimensions. 27
3.3 Cumulative feature importance Random Forest. 30
3.4 Logistic F1 at varying regularization. 30
3.5 Knn F1 at varying the number of neighbors and distance metric. . . 31
3.6 Boundary Svm Gaussian Kernel in 2D. 32
3.7 Boosting Feature Importances. 32
3.8 Features Importance Rolling Classification. 34
3.9 Optimal Threshold choice for Logistic Regression. 35
3.10 ROC Curves Rolling Classification. 35
3.11 Results Rolling Classification for each life%. 36
3.12 Regression Plots (Static). 38
3.13 Feature Importance Rolling (Regression). 39

3.14 Rolling performance per each life% (Regression). 40
3.15 Regression % error in relation to Nominal Duration. 40
3.16 Regression % errors in relation to Nominal Duration for each lifetime. 41
3.17 Regression average % error for each lifetime at varying Nominal Duration. 41

4.1 Ideal decreasing Learning Rate. 47
4.2 Left: overfitting and underfitting. Right: bias-variance trade-off [7]. . 47
4.3 Custom Loss function in Keras backend. 48
4.4 Grid search results using encoded categories. 49
4.5 From left: Accuracy, F1 and Loss Curves. 49
4.6 Custom F1 metric with Keras backend. 50
4.7 Original and Embedded size for each Categorical Variable. 50
4.8 Result using only categorcal variables (1). 51
4.9 Results with only numerical inputs (2). 51
4.10 Results using a mixed input (3). 51
4.11 Results rolling classification per each lifetime. 53
4.12 Predictions against True values. 54
4.13 Rolling Regression results using *NNs* 55
4.14 Average error at varying *Nominal Duration* for each lifetime considered. 56
4.15 Results GRU per each life %. 57
4.16 LSTM error based on nominal duration percentiles. 57
4.17 Two outputs model construction. 58
4.18 Contradictions between the two output predictions in multitask learning. 59
4.19 Multi-output results with undersampling and target projection. . . . 60

B.1 Multi-Task Learning with Neural Networks. . . . . .
B.2 Correlation Encoded variables Static Data. . . . . .
B.3 Example of two rows of scaled Rolling Dataset, having 32 predictors.
B.4 Schema of a ANN with concatenation of embedding layers and numerical inputs (red selection). . . . . .

Chapter 1

Introduction

1.1 Context

Recently, many companies have been increasing the exploitation of machine learning models in order to guide their Business decisions. Indeed, a Company reflects its own strategy through a wide variety of decisions which can be much more effective if supported by information derived from data, rather than relying only on personal intuition. Basing on this approach there is the definition of data driven decision making, which can be beneficial in economical terms in several ways, as in detecting future risks or opportunities. A direct benefit of it may be, for instance, to a firm which operates in the financial market, where it can be crucial to have an accurate prediction on stock prices because that might truly correspond on saving or earning millions of euros. Another more indirect way of using data to support decision may be for increasing internal efficiency, as in the case of predicting the breakdown of a machine in a production process.
The more indirect the economical impact is, the more difficult is to estimate the amount of savings or earnings due to the information given by data. That being highlighted, in this project it is put the focus on the estimation of lifetime and consumption of contracts of *Enel S.p.A.* with its own suppliers, where the latter are referred to entities providing both physical components and services. In particular, each contract with a supplier is characterized by a certain duration, which does not correspond to its effective life. Indeed a contract may be entirely consumed way before its own nominal end date.
There are two extreme situations which it is wanted to prevent; the first consists of the stipulation of a contract which is never consumed. In such case, the company has spent time and energy to negotiate for something it has never really used. The opposite situation is the one where a contract is consumed much before it is supposed to. In such other case, the company has to stipulate a new contract much earlier than expected, risking to be without a supplier in the meanwhile. Estimating the actual life of a contract and its consumption curve is extremely useful for a better planning and programming, even though the benefit has an indirect nature and thus it is not easy and immediate to estimate. Above all, with such an eventual improvement in planning, Enel would increase its bargaining power when negotiating with its suppliers.

1.2 Research objective

In this framework it has been investigated the use of different supervised machine learning techniques to estimate both the consumption of each contract and its remaining lifetime. In particular, for the first task, classification algorithms have been exploited, while regression techniques for the second one. A particular focus has been given to explore the most performing ways to handle categorical features having a very high number of classes. Further, several neural networks configurations have been exploited, including a multi-output structure which allows to satisfy the two required tasks at the same time, that is using one single model.

1.3 Process Description

Once a contract with a specific supplier is stipulated, several information are collected by Enel's Global Procurement functional area. First, the *Validity Start Date* and *Validity End Date*, which express the range of time within which the contract is valid. Between these two dates, several orders can be emitted, each one characterized by its own amount in economical terms and a different time interval passed from the previous one. Such irregularity is graphically illustrated in the figure below, where an example of cumulated orders over time for a real contract is presented.

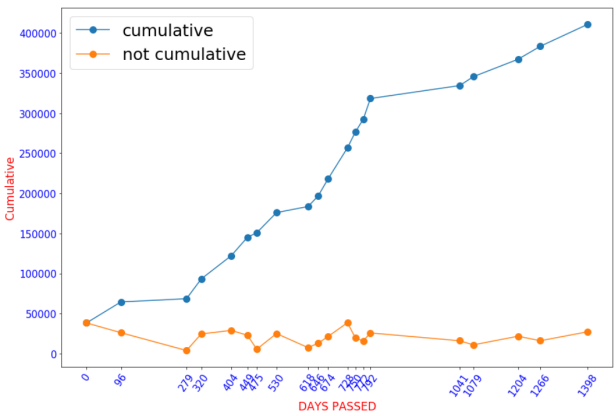


Figure 1.1. Orders irregularity shown through a specific consumption curve.

The information is collected in such a way that contracts and orders are widespread among the same ingested data. What specifies whether an observation in the dataset is a contract or an order is the *Contract Type* boolean field.
Given this statement, the most meaningful initial attributes describing contracts and orders are introduced in table 1.1, where the last two columns refer to attributes which make sense only in the case a record represent respectively a contract or an order.

The ones listed are the most explicative features of the whole process; the remaining ones, all having a categorical nature, are mainly describing with less or more details the Company's functional areas and units responsible for each agreement. That is the case of the *Global Business Line*, *Global Purchase Unit*, *Country*

Feature Name	Description	Only Contracts	Only Orders
Purchase Document	Unique ID of each contract.	Y	N
Validity Start Date	Date in which the contract starts to be valid.	Y	N
Validity End Date	Date until which the contract is active.	Y	N
Contract	ID of each order, identical to the one of the associated contract.	N	Y
Contract Type	Boolean (Yes/No) which specifies whether an observation in the dataset is a contract or an order.	N	N
Purchase Document Date	Date of stipulation. In case of orders it is the emission date.	N	N
VMD Code	Unique ID for each different supplier.	Y	N
QMG Code	Merchandise group ID to which the supplier belongs to.	Y	N
Header Amount	Maximum amount that Enel can use until the contract is fully consumed. If the row is an order, it corresponds to the individual order amount.	N	N
Variant Code	Boolean field specifying eventual changes in terms of time and/or header amount of a contract. All variants are already considered into the ingested data.	Y	N
Active Contract Flag	Boolean column stating whether a contract is active or not.	Y	N

Table 1.1. Main features of raw data.

and *Geographical Area*.
Finally, some of them give extra information about the kind of supply, as for the *Job Type* and *Supplier Type Code* fields.

Consumption Threshold

According to Business needs, the most relevant part of this project is linked to the estimation of contracts lifetime. However, it is hard considering this separated from their consumption. In particular, those contracts which are not fully used consequently terminate on their *Validity End Date*. It is necessary to take off such contracts to avoid misleading performances in the lifetime prediction, keeping only those which are fully consumed. An important modeling choice to achieve that has been followed; it consists of estimating first the consumption level of each contract in a binary way - just if it is going to be fully used or not – then performing a continuous prediction of the lifetime (as it is going to be explained in paragraph 1.3.2). This modeling solution is also based on the choice of the threshold above which a contract may be considered as fully consumed which, as agreed with the Business area, has been set to 90% of the Header Amount.

1.3.1 Static and Rolling approaches

As mentioned in the previous paragraphs, the goal of this project is predicting the consumption amount and lifetime of each supply agreement. It is fundamental to specify that to standardize the contracts in terms of their durations, intervals have been considered in percentage terms, rather than absolute values. That is because in the prediction phase, it will be needed to predict a set of contracts, each being at a different stage of its lifetime, and dealing with percentages makes it easier to face such situation. This fact is at the basis of the target creation process, which is going to be discussed in section 2.4.

Both for classification and regression tasks, two different approaches may be exploited:

1. **Static**: with prediction made at T_0, day of the stipulation. Here, only information available in the creation phase of the contract can be used, neglecting anything derived from its specific evolution.

2. **Rolling**: predicting after a certain time has already passed since the Validity Start Date. Clearly, all information related to the evolution occurred within the range of time passed can be exploited.

A first important observation is that the second approach is expected to yield better results, thus is always preferable. However, in the prediction phase we will have some contracts not yet started, for which only the first approach can be used. For the rolling approach, several further choices can be done. As already stated, it is possible to look behind, from the *Validity Start Date* until the current lifetime stage. The main issue is the way and 'depth' at which considering the evolution. Indeed, it might be chosen any number of lifetime percentages to split the range of time passed into different bins. Since the number of bins created is always discrete, in the prediction phase there will be often the issue of predicting a contract which is at a lifetime percentage the model has not been trained on. In such situation, the forecast would be made considering the closest lifetime cut at which the model has been trained on. Clearly, the higher the number of bins, the better the approximation in the prediction phase, at the expense of more computational cost. The split into N different lifetime cuts can result either in a set of N different models – one for each lifetime cut – or in a unique model dealing with all of them. Thus, in the Rolling configuration there are still two possibilities:

1. **Direct Rolling**: Choosen a certain lifetime %, cutting all the contracts until that point of life and applying a model directly on this (E.g: at 20% life). Because of the previous explanation, in this case many different models must be maintained at the aim of dealing with different lifetimes in the prediction phase.

2. **Parametric Rolling**: Several lifetimes % are chosen, and a new dataset is built with more rows for each contract, each referred to a certain cut. A unique model is enough to overcome the prediction issue.

The last approach can be considered 'parametric' with respect to the lifetime %, since a predictive model would train on contracts cutted at different times of their life. This clearly allows to be more efficient, compared to the direct rolling perspective. It is extremely important to specify that the size of the parametric dataset is not a multiple of the size of the original data, since some samples must be taken off. The latter are those contracts which, considered at a certain percentage of their life, have already expired. In the previous paragraph it has been highlighted that the threshold level of consumption needed to predict in a binary way is 90%. Therefore, when following the second approach, all the contracts having a cumulated consumption higher than that, must be removed. Indeed it would be as much easy as much misleading and not beneficial for any model to predict that a contract will

be consumed, looking just at the feature corresponding to the cumulated percentage used. An illustration of such situation is given in the table below, where it is assumed to have cut all the contracts three times, respectively at their 20%, 40% and 60% of life.

ID	Header Amount	....	N° Orders	% Life	% Used	Label
A	100 K	....	2	20%	12%	1
A	100 K	....	6	40%	28%	1
A	100 K	....	15	60%	94%	1

Table 1.2. Toy-example of a contract expired on its $60th$ % of life.

In such example we can see how contract A at 60% life has already terminated, because used at 94%. While in the third case any model could look exclusively at the % used to understand if a contract is going to be classified as 0 or 1, in the first two snapshots it would have to find somewhere else such information. All the rows having a similar situation to the third sample would be easily predicted as 1. Such good and easy prediction not only would distort the performances, which would (apparently) increase, but would also make the model rely too much on the %Used feature; that means learning with a bias. For this reason, it follows that chosen N lifetime cutoffs, the new dataset will have less than $N*$~40'000 rows, where ~40'000 is approximately the number of distinct contracts we are going to deal with (see section 2.11). That being said, after several trials of datasets built with different number of time cuts, in this project it has been chosen to use 10 lifetimes percentages, ranging from 0% of life until 90%. It has to be pointed out the distinction between the prediction made at the 0% of life and the one performed at T_0, which is based on the fact that on the first day of validity some contracts may be already fully consumed and, for reasons just explained, they have to be taken off. This fact makes the number of active contracts at the 0% of life be necessarily lower than the one at T_0.

1.3.2 Predictions Pipeline

It has not yet been explained in detail the final exploitation of classification and regression models. The idea is that, given a new contract, it is first predicted by the classification model whether it is going to be consumed or not. Then, two are the scenarios:
1. It is predicted to be a 0, corresponding to not consumed. Then its lifetime will simply correspond to the whole nominal duration agreed in the stipulation phase.
2. It is predicted to be consumed, a 1. In such case, it is relevant to understand when this is going to happen and, in order to find that out, the regression model is used.

Being regression eventually applied this way in pipeline after classification, its related dataset has to be further filtered, specifically by excluding those contracts which are not going to be entirely consumed. This fact results in a smaller number of samples for the regression task, compared to the classification one.

Chapter 2

Data Preprocessing

2.1 Raw Data Description

Global Procurement functional area, consists of 5'680'670 samples and 27 columns - mostly categorical - related to contracts stipulated between Enel and its suppliers after 2012. Firstly, it has to be recalled that each record does not actually correspond to a contract, since most of the rows represent rather orders which are associated to a certain contract in the dataset. The list of the most relevant initial features has been mentioned in the previous chapter (1.3).

In table 2.1 some hypothetical rows of the dataset are shown. The interpretation is the following: two contracts are represented, identified as A and B. The first one has been stipulated on 2009, begins to be active on 2010 and expires on 2020, if not consumed before. The dates have been simplified reporting only the year, enough for this toy example. Two orders related to contract A have been emitted, one in 2011 and the other in 2012, both valid being them between the range time agreed. Clearly, the dots stay for the remaining characteristics; for instance, for the two orders there would be the amount in terms of euros, but the purpose of this example is to give just an idea of how the dataset is composed, highlighting the fact that both contracts and orders are included.

2.2 Dataset Construction

Before computing any relevant statistics or doing any other step, it is necessary to transform the dataset into a new one where each row corresponds to a single

Contract ID	**Type**	**Start Date**	**End Date**	**Order ID**	**Doc. Date**	**....**
A	Contr	2010	2020	/	2009	....
/	Order	/	/	**A**	2011	....
/	Order	/	/	**A**	2012	....
B	Contr	2015	2017	/	2014	....
....	....	....	....	....	....	....

Table 2.1. Hypothetical samples of the raw Dataset describing Contracts and its associated Orders.

contract. To obtain such new dataframe an intermediate step has been performed, associating contracts and orders. Without getting deep into the programming codes, what it is obtained after a *Join* operation is given in the table below.

Contract ID	**Duration**	**Header Amount**	**Order Date**	**Order Amount**	**....**
A	3650	1 Mln	2011	100 K	....
A	3650	1 Mln	2012	200 K	....
B	730	200 K	2014	50 K	....
....	....	....	....	....	....

Table 2.2. Orders are associated to Contracts with a Join operation.

It follows a brief explanation of the specific example: from table 2.1, it can be noticed that contract A lasts 10 years, thus 3650 days. Its Header Amount is assumed to be 1 million, even though it was omitted in the first table, as well as the amounts of the two orders, respectively 100'000 and 200'000 euros. What we can state given these information, is that in the first two years it has been used 30% of the maximum allowed.

Besides the details of the example, it is relevant to highlight that with this having done, the table 2.1 has been transformed into an equivalent one - 2.2 - which is much easier to handle, having indeed only contracts. However, these ones are still described in as many rows as many orders have been done for each, for instance in the example there are two rows for contract A, whose associated orders are indeed two.

Because of that, the final step of aggregating data in order to have each row corresponding to a different contract is necessary. This step is going to be described in the next paragraph.

2.2.1 Dataset Aggregation

To obtain a compact dataframe where each row corresponds to a different contract, it is crucial to consider which information is worth to keep and extrapolate. Surely, once the dataset is aggregated, some information is going to be lost and some converted; for instance, considering the situation we have got so far, that is table 2.2, it is easy to realize that the *Order Date* is likely to be neglected in the final transformation, replacing it maybe with something describing time intervals. When performing such operation, new features have to be consequently created, as it is going to be explained more in detail in section 2.4. As it is going to be explained later on, the issue in creating features concerns only the rolling approach, mentioned in the previous chapter; for the static model the only feature to keep, without the need of aggregated functions, corresponds to the nominal duration. Given the nature of the rolling dataset, illustrated in table 1.2, aggregating data through a *GroupBy Operation* is usually performed looking not at the entire length of a contract, but only up to a specific life percentage of it.

In the following table there is the compacted version prototype of the dataset with some of the new features created.

ID	Nominal Duration	Header Amount	....	N° Orders	Avg Order	Avg Interval	% Used	Slopes Ratio
A	3650	1 Mln	....	2	150	300	30	1.5
B	730	200 K	....	1	50	180	25	0.5
....	....	....	....	....	....	....	....	....

Table 2.3. Aggregated Dataset. After a *GroupBy* Operation, each row represents a certain contract. Also, some of the created features are shown.

A brief explanation of the values in the example is going to be given in the feature engineering section (2.4), with the purpose of reporting the basic logic of the process.

Being the grouped operation performed, we finally got to the point where each row corresponds to a single contract.

2.3 Data Cleaning

Before getting the final dataset to preprocess, data has to be cleaned. In a first moment, according to Business communications, it has been filtered by those samples either meaningless or with known poor data quality. Basing on that, contracts having the following characteristics have been taken off:

1. Purchase Document Type different from 'Z', 'A' and 'V'. In particular, it must be specified that type 'Z' corresponds to *Framework Agreements* ('accordi quadro'), which are the ones leaving more freedom about the user (Enel) consumption and are not binding from any business perspective.

2. Header Amount lower than 10'000 euros.

3. Validity Start Date earlier than 2012.

Then, data has also been filtered by samples either not relevant or incoherent with important assumptions. This is the case of contracts having:

1. Value 'Yes' in the column 'Active Contract Flag', corresponding to active contracts, not terminated yet. These are not useful to build any machine learning model, simply because we don't have yet any value for target variable for both regression and classification problems.

2. Nominal Duration lower than 15 days, since it would not be valuable for the Business to make prediction for such short agreements.

3. Sum of total consumption higher than 1.2 times the Header Amount. Such factor of 1.2 has been chosen based on the distribution of exceeding contracts, illustrated in figure 2.1 which shows how based on such choice, most of the contract are kept. In particular, from such curve it can be noticed that the number of contracts exploited more than their 100% is more than 5000, and

the absolute value of the slope is very high until a multiplier of the *Header Amount* equal to ~1.1. Finally, for a value around 1.2, the curve begins to be flat.

4. Effective expiring date beyond 105% of life, in the sense that there have been orders beyond the stipulated end date, which is not possible given the variants already included (as stated in table 1.1). However, for short differences as 5%, that has been considered an acceptable error.
   Again an elbow curve analogue to the previous case has been used to support such decision.

5. First order antecedent to Validity Start Date.

6. Missing data, represented by the string "nd" in this specific raw data. Contracts having such value in any of the relevant features described in chapter 1.3 are a very residual case, since they amount to less than 1% of the data.

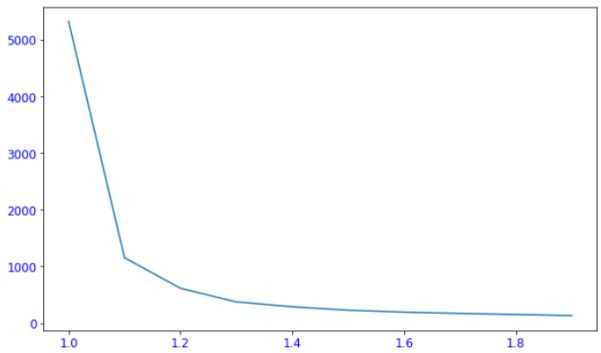


Figure 2.1. Elbow curve showing how many contracts (Y axis) exceed the *Header Amount* by the factors reported on the X axis.

2.4 Feature Engineering

As mentioned in paragraph 2.2.1, it is a matter of fact that punctual information is going to be lost when grouping and compacting data. In practice, it should be kept somehow all information which might be useful to achieve the two prediction tasks, related to lifetime and consumption. For the rolling approach, the list of features extrapolated for each life percentage -with the relative description- is given below, specifying whenever relevant the related maths behind the example 2.2 - 2.3.

1. **Nominal Duration**: the duration according to the stipulation. It is the difference between the *Validity End Date* and *Validity Start Date*. As in the reference example, it has been reported in days.

2. **Number of Orders**: how many orders have been done.

3. **Order Rate**: the frequency with which orders have been sent, expressed as number of orders per month.

4. **Days until first order**: days passed between *Validity Start Date* and first order emitted.

5. **Days from last order**: days passed from last order emitted until last time considered, corresponding to the life% of the current sample.

6. **Avg Order Amount**: average of orders amount.

7. **Std Order Amount**: the standard deviation of the order amounts (euros).

8. **Avg Intervals**: average interval of time between two orders (days).

9. **Std Intervals**: standard deviation of the intervals between the orders (days).

10. **Percentage Used**: how much it has been consumed in percentage since the beginning, namely the *Validity Start Date*.

11. **Actual/Ideal Slope**: It is defined as:

$$RS = \frac{A}{I} \tag{2.1}$$

    where A is the slope of the curve representing the cumulated amount consumed within a certain interval - the Actual Slope - and I the Ideal Slope. The latter corresponds to the amount that should be consumed every month to make a contract expire exactly on its *Validity End Date*.
    It is considered as Ideal because, for reasons explained in chapter 1, the perfect situation is neither consuming the contract much earlier, nor consuming nothing of it. In the reference example, it has been calculated in the following way: the *ideal slope*, obtained dividing the Header Amount by the Nominal Duration, would be equal to 100'000 euros consumed per year. If we consider only the first two years, the *actual slope* is 150'000 euros per year, since it has been ordered the double amount from 2010 and 2012. By dividing the two slopes, it is obtained 1.5. In general, a value higher than 1 represents a situation in which it has been drawing faster than expected and, on the contrary, if lower than 1 it means the consumption has been slower than supposed to.

12. **Life %**: percentage of life considered for the contract. The role and the weight of such feature are going to be discussed in the next Chapters.

13. **Seasonality and Initial Quarter**: percentage of orders emitted in each quarter, weighted by their frequency not to favor the ones passed more times than the others. It then corresponds to 4 features. The initial quarter is an integer value between 1 and 4 associated to the quarters which might be useful to catch a temporal information, specially for short contracts which last only one or two trimesters.

14. **Acceleration(%used)**: it is defined as:

$$A(I) = \frac{\sum_{j=0}^{N} I_j}{\sum_{k=0}^{M} I_k} \tag{2.2}$$

where j and k are natural numbers expressing the days passed since the *Validity Start Date*, *s.t* $j \in [t_{-1}, t]$ and $k \in [t_{-2}, t_{-1}]$. The generic I_j or I_k, *s.t.* $I < H$, refer to the orders emitted in a specific date, which must be lower than the Header Amount *H*. Finally, *t* refers to the last time considered, *t-1* and *t-2* to the two previous lifetime cuts; N and M are respectively the number of orders emitted in the two intervals. This formula is then a ratio between the % of consumption in the last and previous intervals considered. If higher than one, it means the contract has been used more quickly in the last interval (e.g between 30% and 40% of life passed) than in the preceding one (between 20% and 30%).

15. **Acceleration(n° orders)**: same of the previous one but considered with respect to the number of orders emitted rather than their amounts. It is then expressed by:

$$A(N) = \frac{N_j}{Nk} \tag{2.3}$$

where the contraints of j and k are the same of the previous formula.

2.5 Target

Any supervised model requires a label for each sample to be trained on. Recalling the two tasks of predicting the whole lifetime and the final consumption of each contract, the former in regression while the latter in binary classification (section 1.3), two labels have been created. Summing them up, we have obtained:

- **Classification problem**: Given any contract i, its percentage of Header amount used P_i within its own *Nominal Duration* range and the consumption threshold L (section 1.3) equal to 0.9, its related label will be:

$$y_i = \begin{cases} 1, & \text{if } P_i \geq L \\ 0 & \text{otherwise} \end{cases}$$

- **Regression problem**: given T_c as the time of the actual range of time passed for the *i-th* fully consumed contract and T_n its nominal duration, the regression target for such *i-th* contract is defined as:

$$y_i = 100 * \frac{T_c^i}{T_n^i} \tag{2.4}$$

Since the fraction is always between 0 and 1, the result will necessarily be a continuous value between 0 and 100, reflecting the lifetime, with respect to the Nominal Duration. General insights behind such choice have been exposed in section 1.3.1).

2.6 Missing Data Imputation

When building new features as done in section 2.4, it is necessary to handle exceptional situations and ensure that everything makes sense. Here, the extreme situations to handle are the ones where no orders have been sent. In fact, in such circumstance, features from 3 to 9 in the list 2.4 do not behave normally. For instance, the average length of order intervals does not make sense if there are no orders at all.
The contracts with no orders reflect a very big portion of the dataset, as it is going to be shown in section 2.8. Therefore, a particular attention has to be given to this problem. There are many ways to accomplish the imputation of missing data. Most common ones, as presented in [26], are replacing them with the mean or the median of the whole feature, in order not to drastically affect its distribution. In some other cases, it might be instead more opportune to assign a zero, if that is coherent with its meaning. Also, putting out of scale values can be a reasonable strategy, in order for a model to discriminate them from the common range of values. Finally, it might be possible to assign a certain value basing on the domain knowledge of the project. Obviously, the best choice depends on the situation. In this thesis missing values have been handled with a logical imputation, which means giving a reasonable meaning to the replacing values. Before getting into the details, at the basis there is the fact that zero orders can be seen either as N orders of zero euros each or as one single order emitted after the Nominal Duration, when the contract is already closed. Following this approach, whenever the number of orders is equal to zero, the average order import and the Actual/Ideal Slope have been set to zero, besides clearly setting to zero the orders rate too. Same for the two standard deviations, related respectively to the order amount and the intervals. As concerning the remaining columns related to time intervals, namely 'Days until first order', and 'Avg Intervals', exploiting the approach based on considering zero orders as one single order of any amount sent after the contract final date, when valid time is over, they have been set as the longest possible interval plus one day. Clearly, since 'Days from last order' looks at it in the opposite way, it has been assigned -1, meaning that the order has still to arrive (indeed, it will arrive after one day).

2.7 Handling Categorical Variables

Being data cleaned, before applying a predictive algorithm it is necessary to transform the categorical variables into numerical ones. Among the initial features, excluding the Header Amount which is the only original one to be already continuous, all the features present in figure 2.4a must be converted. The easiest case is for those having only two categories, which have been converted into binary assigning then values 0 and 1. For the remaining ones, some other methods are necessary. Label Encoding has to be excluded since such approach may be used only for features having an ordinal nature, rather than nominal.

2.7.1 One Hot Encoding

One of the most common way to handle categories is to transform the i-th feature, having N classes, into N-1 binary columns where for each row, there are all zeros and a 1 assigned to the column corresponding to the original class. This process, known as One Hot Encoding, creates then new variables, namely dummies, which are all independent between each other.
However, such technique cannot be used to convert features having very high cardinality, since the dimension of the problem would explode (e.g: ~12'000 columns just to encode *VMD Code*). Besides that, although linear models such as Logistic Regression work well with dummy variables, some other algorithms such as Random Forest do not handle them quite well, as explained in [20].

2.7.2 Clustering Categories

Another way to make categorical variables suitable to an algorithm without drastically increase the dimensionality would be to group categories. To achieve that, either K-Nearest-Neighbours clustering or traditional clustering methods may be used. The second approach has been considered in this thesis; it has been exploited a clustering algorithm in order to group the *VMD Code* classes into a much smaller number of sets. In order to do that, a new matrix has been built by considering aggregated features for each supplier, like shown in figure 2.2.

VMD Code	days_for_new_stipulation	%forniture	%diff_merchandise	avg header	std header %	avg rate mese	std rate mese %	avg durata nominale	std durata nominale %	avg utilizzo medio%	std utilizzo medio%
46974	9.31	98.0	7.33	21153.37	137.21	0.16	87.50	394	71.07	95.09	15.13
DR\0030000165	7.94	0.0	0.38	621.27	100.51	2.15	16.74	13	7.69	98.37	12.21
158703	13.02	100.0	22.97	253995.65	762.38	0.22	159.09	261	54.02	90.43	26.61

Figure 2.2. Aggregated data to cluster suppliers.

The optimal number of clusters to separate the categories may be choosen either looking at the lowest cost function of direct clustering methods - such as *K-Means*- over a grid of number of clusters, or by using stopping criteria in hierarchical clustering techniques.
Nevertheless, the limit of such methods is the fact that the target of the original problem is not taken into account, since it is an unsupervised method. This way, all the features as the ones reported in the previous figure would be given the same importance, when actually it is well known that for instance the average percentage used is surely much more important than any other feature, being it the continuous version of the final target. That represents a very big drawback in using them, although there are some versions of Automatically Weighted Clustering Algorithms - as in [11] - which iteratively find out the relative feature importances. Indeed even in this case, the higher importance would be assigned to those variables which -once standardized- have more variability. In summary, clustering would result in grouping samples -the single categories in this case- basing on some similarity which is not necessarily linked to the final target it is wanted to predict. That being said, other methods are preferable.

2.7.3 K-Fold Target Encoding

Another way to encode categorical variables is through a K-Fold approach. Chosen a certain threshold and considering one feature at a time, it consists of splitting the training set into K folds, and assigning to each category in the *k-th* fold, the target average of such category in the K-1 folds, providing that the number of its samples in such K-1 folds overcomes the threshold. If instead it does not, it is assigned the average of the whole feature, still calculated not over the full Training Set, but over the K-1 folds. Similarly, in the Test Set, for each known category, it is assigned its average in the Training providing that its frequency overcomes the threshold; otherwise, the mean of the whole population -the whole feature- in the Training is assigned, as described in [19]. In this way the value assigned to each category is clearly target oriented, as opposed to the clustering approach described in the previous section. Further, the K-fold approach is useful to avoid any data leakage - which implies overfitting - since it results in assigning each time different values to each class. Also the threshold role has the same purpose: if the number of samples of a class in the K-1 folds is too low, it is too risky to rely on its average target. On the other hand, if it is higher than the threshold, it is more consistent from a statistical point of view.

The use of such kind of encoding has the big advantage of keeping the original dimensionality of the problem, since each categorical variable is transformed in numerical without creating more features, as it happens instead in One Hot Encoding.

Customized Target Encoding with Noise

In this project, the K-Fold target encoding approach just described has been slightly customized to increase its robustness. In particular, in order to avoid overfitting, there is a relevant improvement which has been made. Before getting into the details, this modification puts the basis on the fact that the *Sample Mean* neglects the variability of the samples over whom it is calculated. Indeed, the same average value of the target in the k-1 folds may be obtained by having all values close to the average or widely ranging around it. In other words, the *Standard Deviation* is extremely important to consider. That being said, in the customized version of Target Encoding, it has been added a certain component of Noise proportional to the standard deviation. More precisely, when considering the target value for each class in the k-1 folds, it has been computed in a first step both the average and the standard deviation. Then, it has been assigned to the class in the *k-th* fold a value extracted from a Gaussian with average and variance respectively equal to the target mean and a proportion of the standard deviation in the K-1 fold.

Formally, considering any feature x, chosen an arbitrary number K, each fold will have $d = \frac{N}{K}$ rows, with N being the number of rows of the whole dataset. Then, $\forall k_j \in K, \forall l = 0, 1, ..., d$, the encoding of each category x_l in each fold is defined as a random draw from the following Normal distribution:

$$x_l \sim \mathcal{N}(m_l,\, 0.1 * \sigma_l) \tag{2.5}$$

where

$$m_l = \begin{cases} \frac{\sum_{k \in K: k \neq k_j} \sum_{i=0}^{d} y_i \delta_i}{\sum_{k \in K: k \neq k_j} \sum_{i=0}^{d} \delta_i}, & \text{if } \sum_{k \in K: k \neq k_j} \sum_{i=0}^{d} \delta_i \geq T \\ \frac{\sum_{k \in K: k \neq k_j} \sum_{i=0}^{d} y_i}{\sum_{k \in K} d_k}, & \text{otherwise} \end{cases}$$

where δ_i is a boolean which is 1 if the *i-th* category in the *k-th* fold is equal to the *l-th* category which has been considered, else 0. The previous formula was related to the mean of the K-1 folds; similarly, the standard deviation is computed over the k-1 folds, obtaining the related σ_l; 0.1 is a multiplier chosen small enough to avoid assigning the same punctual value to the same category.

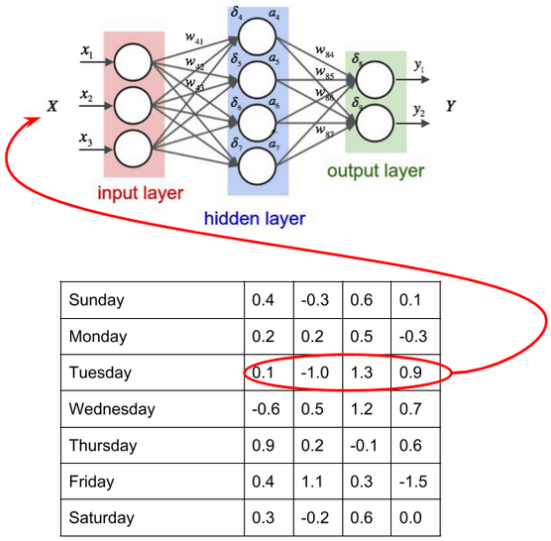
This way all the advantages previously described are kept, but adding noise also guarantees that any model would not strictly rely on precise encoded values of categories, so that the model learns to generalize better. In order to build such method, a custom Python Class has been created, as shown in the code shown in the appendix A.1.

2.7.4 Categorical Embedding with Neural Networks

Recently the Embedding technique, mostly used in Natural Language Processing models, has been demonstrated to be performing in handling categorical features, as shwon in [10]. This consists of mapping M categories to a lower dimensional space P with $P << M$. Each categorical feature is first One Hot Encoded -thus transformed in M new variables, with M equal to its original number of classes- then the weights which put in relation the encoded vector of dimension M and the embedded version of dimension P are learned by the neural network, a topic discussed in chapter 4. In figure 2.3 is illustrated an example of Entity Embedding for a feature whose classes are the days of the week -having then dimension equal to 7- which are then embedded in a subspace of dimension 4. It can be seen how in such new space Sunday is for instance much more similar to Saturday than to Wednesday. It is important to specify that the Neural Network will take one input for all the continuous variable, plus one Embedding Input for each categorical column. In the appendix it has been shown an example of the network input schema - B.4 - and the python function created to preprocess data with categorical embedding, in A.3. So, the essence of embedding is to catch the relationships of the various classes. This fact of considering the interactions is definitely different from One Hot Encoding, where the new transformed variables are independent each other. Entity Embedding has been proved to be very efficient even when the number of classes is very high [10]. Nevertheless, it must be considered that it does not make very sense to apply such method when the dispersion of the feature is very high, in the sense that there are many classes appearing only once or with a very low frequency, as for the case of *VMD Code* and *QMG Code*.

2.7.5 Final choice to handle categories

Given the overall picture highlighting the main strengths and weaknesses of different techniques to handle categorical variables, it has been considered opportune to use



Sunday	0.4	-0.3	0.6	0.1
Monday	0.2	0.2	0.5	-0.3
Tuesday	0.1	-1.0	1.3	0.9
Wednesday	-0.6	0.5	1.2	0.7
Thursday	0.9	0.2	-0.1	0.6
Friday	0.4	1.1	0.3	-1.5
Saturday	0.3	-0.2	0.6	0.0

Figure 2.3. Categorical Embedding example.

a mix of them in order to exploit the advantages of each one. So in the end, looking at the list of categorical features illustrated in 2.4a, it has been used:

1. **One Hot Encoding** for the four features having only two categories, resulting in the creation of a binary column for each one.

2. **Customized Target Encoding** for the first two categories -VMD Code and QMG Code- having too high dispersion.

3. **Categorical Embedding or Customized Target Encoding** for all the remaining intermediate cases.

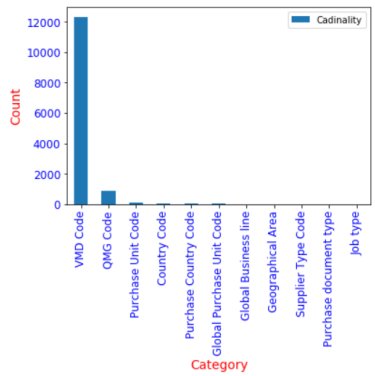
2.8 Descriptive Statistics

2.8.1 Static

So far, data has been transformed in a compact and more useful way, new features have been created and the required labels have been deduced. Next necessary step before giving it in input to a machine learning model is to convert all the values into numerical, which is going to be done in the next section. Before that, it is extremely useful to carry out descriptive statistics to better understand the dataset and, of course, to explain the phenomenon.
First, after compacting raw data, which was about 5,5 millions of rows- in the way described in paragraph 2.2.1, the new aggregated data has got 38'550 contracts, net of the ones removed in 2.3.
About this, it has to be considered that in the *Join Operation* resulted in table 2.2, all the orders not associated to any contract have been automatically taken off. Furthermore, it has to be reminded the whole process of data cleaning described in 2.3, which provides other reasons for the reduction of the size of the dataset. The

	Cadinality
VMD Code	12336
QMG Code	880
Purchase Unit Code	76
Country Code	60
Purchase Country Code	13
Global Purchase Unit Code	13
Global Business line	8
Geographical Area	5
Supplier Type Code	5
Purchase document type	4
Job type	3
Variant code	2
Contract Type	2
Main material group qualification flag	2
Professional services flag	2
Energy convey flag	1
Active contract flag	1

(a) Absolute values.



(b) Bar Chart.

Figure 2.4. Cardinalities of categorical features

number of unique classes for each category is shown in figure 2.4a.

The same information can be seen graphically in the bar chart 2.4b.

It is evident and reasonable that *VMD Code* - reflecting the single suppliers - is the one with highest cardinality, with more than 12'000 different classes, about one third of the dataset. Even the merchandise group - *QMG Code* - has got a high number of categories. The remaining ones have an acceptable number of classes.

Then, it is important to get the knowledge of the label distribution, both for consumption and regression tasks. The Pie Chart 2.5 shows the percentage of contracts whose label is 1, while the plot 2.6 shows the two continuous target distributions, corresponding to the whole life percentage passed (right) and consumption level (left).

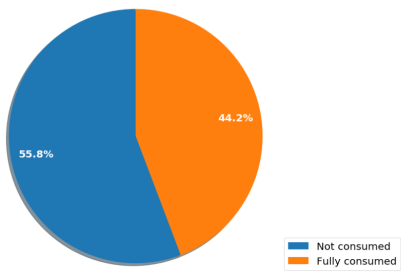


Figure 2.5. Label Distribution. Fully consumed contracts are ~44% of the dataset.

Since a supply agreement can be consumed in any number of orders, with the extreme case where it is consumed all at once, it is interesting to consider some statistics basing on this, such that:

- **% contracts with at least 1 order:** 86.98%.
- **% life passed until 1st order** is shown in the two boxplots of figure 2.7.

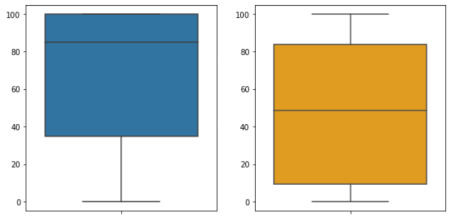


Figure 2.6. %Used (left) and %Life (right).

The one on the left is related to the percentage calculated among all the samples, while on the right is strictly referred to the contracts fully consumed at the first order. We can see how in both the cases such first order is emitted roughly around the 10% of life, according to the *50%-th* percentile.

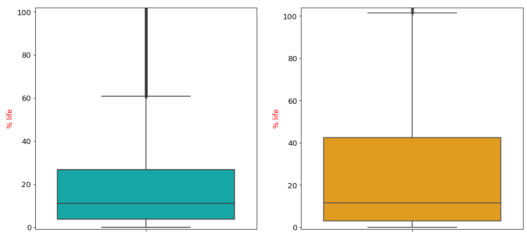
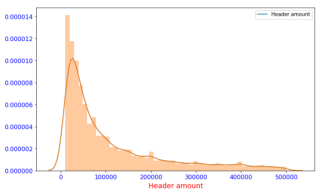


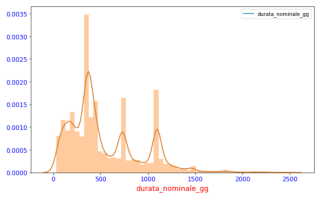
Figure 2.7. Life % passed until first order. On the left related to all the contracts, while on the right only for those entirely consumed (>90%) at the 1st order.

Besides these exceptional situations, it is interesting to see the distribution of the Header Amounts - expressed in terms of euros - and Nominal Duration.

The former, which seems to be drawn from a lognormal distribution, is reported in figure 2.8a, while the latter is shown in picture 2.8b, both with the related Kernel Density Estimation (KDE) plots.



(a) Header Amount.



(b) Nominal Duration.

Figure 2.8. Numerical Features Distributions.

From those, it is evident that the agreed Header Amount tends to be lower than 100'000 euros and to be roughly equivalent to 40'000 euros. It must be recalled that for Business reasons, supply agreements having Header Amount lower than 10'000 euros had been taken off in section 2.3. As concerning the deal in terms of duration, it is evident how most of the contracts can last at maximum 1500 days - roughly 4 years - and the highest value of the density is almost at one year and half. Then,

since the resulted KDE seems to reflect a mixture of gaussians, there are two other peaks around 700 days, thus two years, and 1200, about three years.

In plot 2.9 we can have a broad idea of both the numerical features in relation to the target. In both cases, there doesn't seem to be any particular dependency with the label.

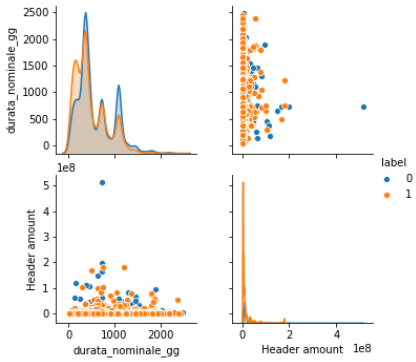


Figure 2.9. Numerical features in relation to target.

As concerning categorical variables, having most of them a very high cardinality, it is hard to visualize some graphs. For instance, if it is wanted to show a bar chart of the frequency for each class within the categorical feature, that would be practically impossible in the case of the first categories illustrated in fig. 2.4a, having hundreds of classes. Thus, a more compact way to visualize their properties is needed.
The method chosen consists of considering only the first N classes whose cumulative frequency roughly reaches the 85% of the dataset - displaying at maximum 4 of them - and showing then their target average and standard deviation, as illustrate in figure 2.10. It must be said that the respective means have been rescaled taking into account the light unbalance of the target classes, so that a bar higher than 0.5 means that a specific class tends to be associated with used contracts -class 1 -, while the closer to zero, the more it is associated to class 0.

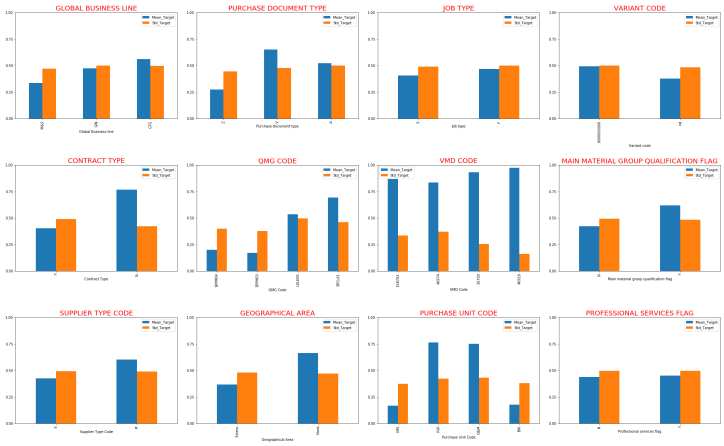


Figure 2.10. Average label of most frequent classes.

For instance, looking at the bars, a Purchase Unit Code equal to 'ROI' seems

to be associated more often to fully used contracts rather than 'IMS', as well agreements stipulated in the 'Nord' area leads to have class 1 more often than the one abroad - 'Estero' - and so on. On the other hand, features as Professional Service Flag and Variant Code have average value around 0.5 in all their classes, besides having a quite large standard deviation, suggesting a lack of any pattern with the classification target.

As concerning regression target, related to the whole % lifetime of (consumed) business agreements, the histogram 2.11 highlights a major concentrations in the tails, suggesting that contracts seem to expire either in the first 10% of life or in the last 20%.

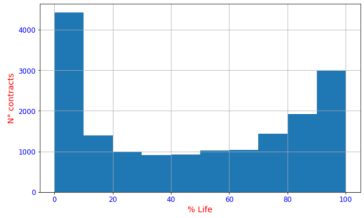


Figure 2.11. Regression Target distribution.

2.8.2 Rolling

As mentioned in the previous section, chosen 10 lifetime cuts for each sample, the model will have a dataset composed by contracts considered at different points of their life. Since the number of samples increases and data is filtered off by the contracts progressively consumed, new descriptive statistics have to be carried out.

First, the label distribution has changed like it is illustrated in figure 2.12. In particular, the % of fully consumed contracts has dropped to less than 30%, thus classes have become unbalanced.

In figure 2.13 the two boxplots related to the % of consumption and the target in the regression task are illustrated.

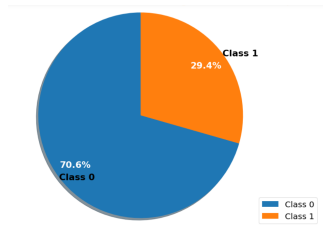


Figure 2.12. Label Distribution of Rolling dataset.

As discussed in section 2.7, one of the two ways followed to handle categorical features has been through the custom Target Encoding; clearly, it requires *Train Test split* before being applied, which is going to be presented in paragraph 2.12. Any machine learning model would be trained on the Training Set and then tested on data it has never seen. The assumption at the basis of this is that data on the test set has to be distributed similarly to the one on the training set.

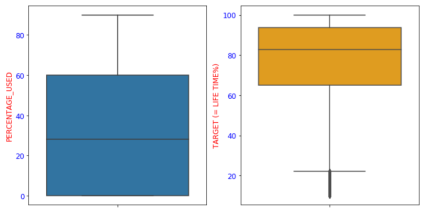


Figure 2.13. BoxPlots of % used and % life for Rolling data.

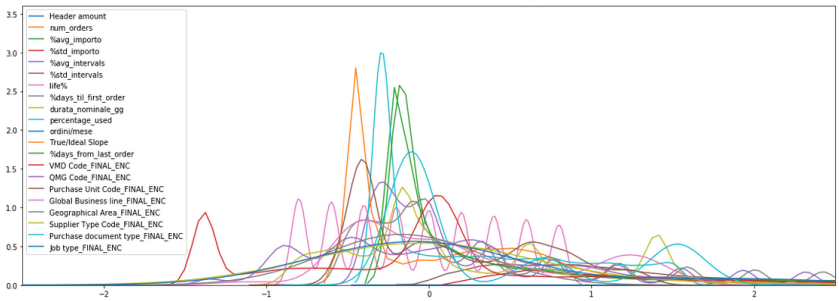


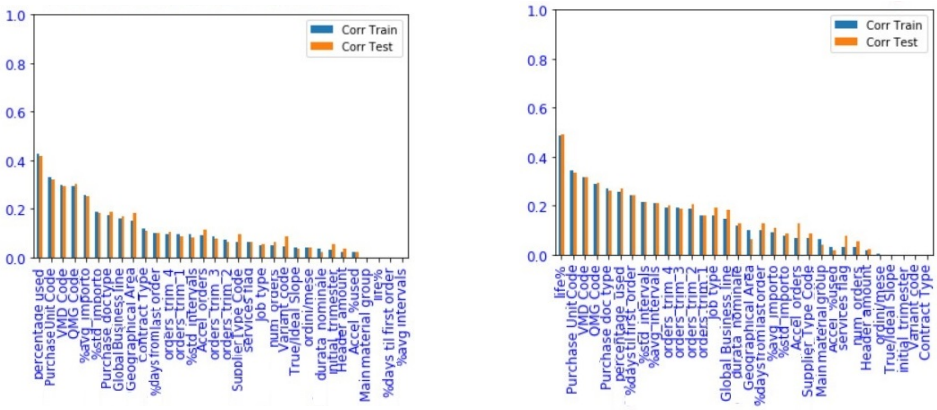
Figure 2.14. Variables distributions after the Encoding (2.7).

Thus, being data splitted and scaled, it is possible to have an overview of the distribution of the features in the Training Set in picture 2.14. Here, it can be noticed that about half of the features is distributed as gaussians, while the remaining ones seem to have distributions coming from a mixtures of gaussians.

After performing the custom Target Encoding of categorical features (sec. 2.7), we can have an idea of the goodness of it by looking at the correlation between each of them and the labels, given in figure 2.15. We can see how for the classification dataset, the highest correlation is about 35%, while for regression it is a bit higher than 40%. The fact they are not too high reflects the good result of the custom encoding, particularly due to the noise added. Comparing also for both cases the blue and orange bars, referred to Train and Test, we can see how the correlations in the former are roughly equal to the ones in the latter. That is a further demonstration of the quality of the custom encoding method.

It must be also underlined that in the Rolling dataset, the percentage of samples with exactly zero o or one order has raised to 51%. That, again, is due to the fact that the more contracts' lives go on, the more likely their related samples sequence will be truncated at a certain life %. Clearly, for not consumed contracts, the associated sequence of 10 samples related to their different lifetimes will entirely occur in the data, while it will be clearly truncated for those which are consumed. This is basically the reason why the dataset has become unbalanced.
The average width of orders intervals of all contracts and their standard deviation are illustrated in the two boxplots in picture 2.16, from which we can deduce that every 4% to 12% of life passed, one order is emitted.

In the Appendix B.2 it is also illustrated the correlation of encoded variables, which demonstrates the absence of high multicollinearity. Further exploratory analysis on the Rolling data has been conducted in a similar way to the static data



(a) Classification: correlation with label. **(b)** Regression: correlation with target.

Figure 2.15. Correlations of features with target after Encoding (2.7).

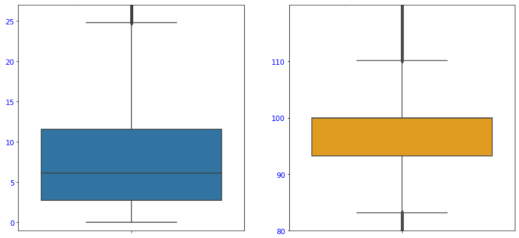


Figure 2.16. Average interval between two orders (left) and standard deviation (right) in respect to life%.

description, but only the main changes earlier explained have been reported.

2.9 Feature Scaling

Being the dataset cleaned, new features created and all columns made numerical, the only step needed before giving it in input to a predictive model is scaling the features. That is fundamental for linear models, such as Linear Regression, SVM or Neural Networks, because features having wider ranges would otherwise result in higher coefficients or weights than others. In the case of scale invariant algorithms, such as Random Forest or Gradient Boosting, even if scaling is not necessary, it is advised to speed up algorithms from a computational point of view, since CPUs work better with small numbers. Most common ways to scale numerical features are:

1. **Traditional Scaling**: rescaling the features to a new range. For instance, to make them between 0 and 1, for each value x it is computed $\frac{x-min}{max-min}$.

2. **Normalization**: which works if the feature is distributed as a Gaussian. It consists of computing $\frac{x-mean}{std}$, using mean and standard deviation of the feature.

In this thesis the following more robust way to perform scaling has been chosen, by using the Scikit Learn package [2]:

3. **Robust Scaling**, which keeps track of the outliers of the distribution by using percentiles. In fact it is computed by appling $\frac{x-50\%}{75\%-25\%}$, where the 50% is the *50-th* percentile -that is the median- while the others correspond to the *75-th* and *25-th* percentile of the feature. This way, only the values inside the denominator range will be scaled to small numbers, while the farther a value from either the 1st or the 3rd percentile, the higher the scaled number.

A major strength of such Robust Scaling is that it can be applied to any feature, even if not originally distributed as a Gaussian which instead, as seen before, is an important requirement when applying normalization.

2.10 Preliminary Feature Selection

One optional but absolutely recommended step to build models, concerns *Feature Selection.*
The goal is to keep the most relevant and not redundant information, in the sense that it should be attempted to be as more parsimonious as possible. Doing this way, not only the dimension decreases and so does the computational time, but it also improve the ability of a model to generalize well on new data. Basing on this concept, those columns having always the same value have been taken off since they would surely not add valuable information to any machine learning model. This is the case of the features *'Energy convey flag', 'Migrated Flag', 'Created by migration flag', 'Completely migrated flag'*, constantly valorised with the string 'N'.
In general, there are several ways to select features; a preliminary selection which is usually made consists of taking off highly correlated features. Again, that is due to the fact that the ideal dataset contains a certain set of variables, each giving different information. If two features have a pattern according to whom they are highly dependent each other, it is useless to keep both. Now, the *dependence* between variables is actually a broad concept, since they might have various kinds of relationship, like linear, polynomial, exponential etc. The easiest way to check if they are dependent is to see the *Pearson Correlation Coefficient*, which can have any value between 0 and 1, with the latter corresponding to highest linear correlation of two vectors. In general, if they are linearly correlated, then they are clearly highly dependent and thus certainly bringing redundant information. On the other hand, if they are dependent, it does not necessarily follow they have high linear correlation. That being specified, we can conclude that removing highly correlated features is surely going to help any model. In this project, among two correlated features, it has not been chosen randomly which one to discard; indeed such decision has been slightly supported by a first run of Random Forest models in order to compute features importance and removing the one having the lowest. Given that, the features *'Purchase Country Code', 'Country Code', 'Global Purchase Unit Code'* have been removed. Such analysis has been computed even in the Rolling datasets and the same features have been taken off.

2.11 Data Resume

Before getting into the application of different predictive algorithms, a brief summary of the datasets is given below:

- **Static**:
  - Classification: 38'550 samples, 14 features.
  - Regression: 17'051 samples, 14 features.
- **Parametric Rolling**:
  - Classification: 304'564 samples , 32 features.
  - Regression: 89'596 samples , 32 features.

For completeness, it is possible to have an overview of the whole set of 32 features in the appendix -figure B.3 - where 2 rows of the Rolling Dataset are shown, without the related Target which varies basing on the task. Clearly, negative numbers are due to the fact that features have been encoded and scaled. It must be also reminded that the Static dataset contains a subset of those features, more precisely only the Nominal Duration among the ones created.

2.12 Train Test Split

Any machine learning model would be trained on the Training Set and then tested on data it has never seen. More precisely, a portion of the training is usually kept to validate the model, referring to the choice of its best hyperparameters. Actually, a more sophisticated and robust alternative to that is using a K-fold Cross Validation approach, which is the one exploited in this thesis.
What is fundamental is to never spy in any way observations on the test data, otherwise the performance would clearly increase on that, but would be drastically lower when using a different test set. In order to avoid that, in this thesis the split between train and test has been carried out with an easy ad hoc procedure. In particular, the rolling dataset is constructed in such way that a single contract occurs in more than one row. Thus, applying the classical split on data would result in having different lifetime cuts of the same contract both in the training and in the test set.
To avoid that, set the test size to 20%, the split has been performed considering the whole contracts' sequences instead of the individual records.

Chapter 3

Machine Learning Models

In the previous chapters it has been explained in detail the goal of this project, divided in the two tasks of predicting contracts consumption as a binary classification and their lifetime in regression. Furthermore, for both of them, both static and rolling approaches have been exploited, as described in section 1.3.1.

3.1 Algorithms Overview

The goal of ML models is to learn a function in N dimensions which can generalize well enough on unseen data. To do that, some specific algorithms have been chosen in this project, among a wide range of different techniques. First of all, linear models have been investigated, to have a first idea of how a linear boundary - or a linear function in the case of regression - perform. This is the case of *Logistic Regression*, *Linear Regression* or *SVM with linear kernel*. Those models have the advantage to provide also a direct importance of the variables by looking at the coefficients found; besides suggesting a further feature selection, this fact may be useful to explain more deeply the phenomenon. In particular SVM - both in regression and classification - is robust to outliers, since only support vectors are considered to make predictions on the test and because of that, it is even not very sensitive to overfitting. Since the 'best' shape of the function to learn is a priori unknown, after the above cited linear models, other supervised algorithms have been exploited to go beyond linearity. This is the case of ensemble techniques such as *Random Forest* and *Gradient Boosting*, which are able to handle big datasets providing fast computations (as long as they are properly handled, as discussed in the next paragraph) and are also known to yield good results in many problems. Even though such results are more difficult to interpret, such *decision tree* based techniques provide a first important comparison between linear and non linear models. When the performance are better in such second case, it has been considered relevant to extend the analysis to other algorithms. This is the case of *SVM* with more complex kernels - such as *gaussian* or *polynomial* - and *K-NN*, used for both the tasks, and polynomial regression exclusively for the lifetime task. However, since such techniques are computationally heavy with this amount of data, they have been used only after taking opportune precautions, which are going to be described in the next paragraph.

Besides the expectations previously listed, there is something above all which

is reasonable to expect as a result. It is first necessary to remind the nature itself of the Rolling data, which contains several snapshots of each contract related to the different lifetime percentages considered (further details in section 1.3.1). Given that, it is expected that the higher the life % of a certain contract, the more accurate the prediction. This expectation is based on the fact that clearly, a higher % corresponds to a higher relative range of time within which the evolution of the contract can be observed.

3.2 Hyperparameter Tuning approach

For each of them, *parameter tuning* has been conducted through the use of *Grid Search*, either classical or randomized. The main goal of those searches, aimed at finding the optimal combination of parameters, is to reduce overfitting. This is usually achieved by tuning the specific regularization coefficients of the various models. Also, in most of the cases, the best grid of regularization parameters has the double effect of decreasing also computational time of executions, as for the cases of reducing the number of features or the depth of tree based algorithms. In order to get closer and closer to an optimal solution, parameter tuning has been performed iteratively, meaning that given a grid of parameters and its best combination found, it is then created a new grid where the new parameters to explore are distributed around the best ones of the previous run. Such process is repeated until the new solution does not show any relevant improvement. The tuning of hyperparameters of multiple steps in sequence - as for instance feature selection and the model itself - has been carried out using Grid Search on Python Pipelines.
One of the Python customized functions used to run grid search over pipelines is reported in the code section of the appendix A.5.

3.2.1 Dimensionality Reduction

For the same purpose of a better generalization on new data, even the use of dimensionality reduction methods- such as *PCA*- has been explored. Indeed, this allows to reduce the dimension by projecting features on a subspace where the new variables -namely principal components- are a linear combination of the original ones. The new subspace keeps most of the information of the previous one. More precisely, the information can be interpreted as the total amount of variability of the features of a dataset. In the extreme case of a variable that takes a single value, the variance is zero and not surprisingly the information too. On the contrary, the higher the variance, the higher the contribute to the information. The cumulative explained variance of the N principal components for the Static dataset is reported in figure 3.1.

Clearly, with a number of components equal to the number of original features, the total information kept is 100%. However, since the purpose is to reduce the original dimension, it is relevant to understand a relative optimal number of components. One way to do that when the main goal is not being the most possible parsimonious, is setting a threshold of desired explained variance and seeing with how many components that is reached. In this thesis, it has been chosen to use 10 principal components, which explain about 90% of the total variance.

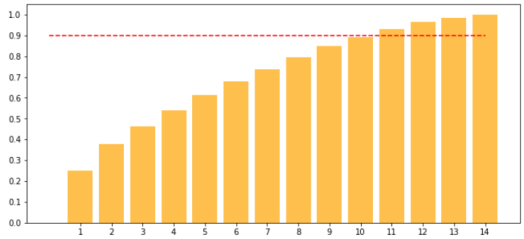


Figure 3.1. Cumulative Explained Variance.

It is also interesting to plot the data in two dimension - that is using only two components- to have a broad idea of how observations are disposed and in particular how separable they are. This is illustrated in figure 3.2.

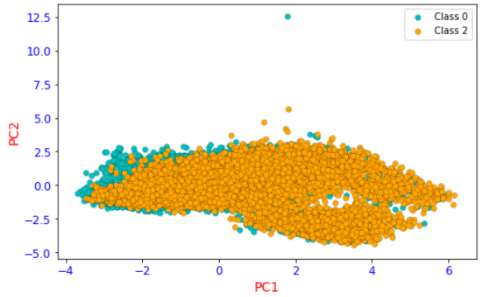


Figure 3.2. Dataset plotted in two dimensions.

Here, data does not look like really separable. Though, it must be considered on one hand that the two components explain about 40% of the total variability, so it is not reflecting the true representation. On the other hand, what is not separable in two dimensions, can be on a higher dimensional space.

3.3 Classification Task

In this section the detailed performances concerning the binary prediction of contracts consumption are going to be reported. As mentioned in the beginning of the chapter, this is going to be done according to both the static and rolling approaches. First of all, it must be considered that mainly **Weighted F1-score** and **Balanced Accuracy** have been exploited as performance metrics, corresponding to the following formulas:

$$F_1 = (1 + \beta^2) * \frac{P*R}{\beta^2*P+R}, \qquad \text{with } \beta = 1$$

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

where P and R refer to *Precision* and *Recall*, while *TP, FN, FP* and *TN* represent the number of true positives, false negatives, false positives and true negatives, respectively.

The first corresponds to an armonic mean of precision and recall, meaning that it gives a high penalty if one of them is low, while the second reflects how many times the model has correctly guessed the true label.
Accuracy is not a good performance metric when target classes are imbalanced. Indeed in this case, that might correspond to a high score by simply predicting the label with the highest frequency. Based on that there is the concept of **Baseline Accuracy**, corresponding to the one of a silly model predicting always the same label. The baseline is useful to understand how well the model has been doing compared to a meaningful reference.

3.3.1 Static

In this section classification results of predictions made at T_0 - when the contract has not yet started to be valid - are going to be discussed (further explanation in section 1.3.1). For the Static dataset, as mentioned in section 2.8, the classes are not strongly unbalanced, indeed about 55% and 45%; thus the baseline accuracy is 55%. Given that, the two scores should be most of the times quite close. However, since classes are slightly unbalanced, models will tend to have higher recall for class 0 - non consumed contracts - than for class 1. This is the reason why in table 3.1 the accuracy - in parenthesis- is always equal or slightly higher than F1 measure.

The first algorithm which has been exploited is **Random Forest**. The reason behind that is the expected complex shape of the boundary separating the contracts. Assuming that basically means that a simple linear boundary would not work as well as a set of multiple regions within which the observations will be classified either as 0 or 1. If not properly handled, *Random Forest* can actually risk to overfit by creating too many regions based on the training data. Besides that, the model suffers the multicollinearity problem. To overcome that, both dimensionality reduction and feature selection methods have been exploited, expecting a small decrease in performance compensated by a better parsimony.

To outweigh the weaknesses of such method, tuning has been focused on:

- *Number of Decision Trees* built.
- *Criterion* for doing each split, namely *Gini Index* and *Entropy*.
  Then, to avoid overfitting by pruning each Tree:
- *Maximum depth* of each Tree.
- *Min impurity decrease*, having the role of a stopping criteria when building each Tree. Indeed, it is stopped to build the Tree when the split over a leaf results in a quite balanced division of classes. Indeed, the ideal situation for a leaf is having a high impurity degree, which means its relevance is proportional to the unbalance level over the leaf itself. In such case, the model would learn that for new observations satisfying the set of conditions to obtain that specific leaf, the target is either mostly 0 or mostly 1.
- *Min samples leaf*, to avoid building leafs just for a few samples.

The strength of an ensemble method such as Random Forest is to build many decorrelated Trees; a contribute to do that is choosing different features at each split. The default parameter for doing this, demonstrated to be empirically the best in [8], is 'sqrt', corresponding to a number of features equal to the square root of the number of total features. The best grid found consists of using Gini Index, 200 Decision Trees, maximum depth equal to 10 and 20 sample per leaf at minimum.
Another ensemble technique which has been exploited is *Gradient Boosting*, having the main strength of building at each step trees over the residuals of the previous iteration. From this algorithm it is expected a very high computational speed, since usually it exploits very short decision trees. As for *Random Forest*, complicated boundaries are expected when using such method. As concerning the rest of algorithms which have been investigated within this framework, the observations stated at the beginning of the chapter are the most relevant.

Before analysing the output of each traditional machine learning model earlier mentioned at varying the parameters tuned, the summary of their performances is given in the following table 3.1.

Model Type	**Model**	**F1 Train (Acc)**	**F1 Test (Acc)**	**Feats**
RF	**RF**	75.6 (75.6)	73.6 (73.6)	14/14
RF	**PCA + RF**	71.5 (71.5)	71.3 (71.3)	10/14
RF	**Feat Sel(Gini)+RF**	75.6 (75.7)	74.1 (74.2)	9/14
RF	**Feat Sel(Lasso)+RF**	76.1 (76.1)	74.2 (74.3)	9/14
Logistic	**Regularized Logistic**	72.6 (72.7)	72.9 (73.1)	9/14
Logistic	**PCA + Logistic**	72.1 (72.1)	72.0 (72.0)	10/14
Knn	**Knn**	74.8 (75.1)	73.4 (73.6)	14/14
Knn	**PCA + Knn**	73.6 (73.8)	72.4 (72.5)	10/14
SVM	**SVM Linear**	66.5 (68.5)	65.6 (67.9)	14/14
SVM	**SVM Gaussian**	75.1 (75.0)	73.7 (73.7)	14/14
SVM	**PCA + Best SVM**	74.7 (74.9)	73.2 (73.1)	10/14
SVM	**SGD**	71.9(72.2)	72.3 (72.6)	14/14
Boosting	**XGBoost**	74.5 (74.6)	73.8 (73.9)	14/14
Boosting	**PCA + Best Boost**	73.1 (73.2)	73.2 (73.2)	10/14
Ensemble	**Hard Voting**	75.3 (75.3)	74.1 (74.1)	/
Ensemble	**Soft Voting**	75.3 (75.3)	74.2 (74.3)	/

Table 3.1. Perfomances Classification (Static)

As concerning *Random Forest*, it can be noticed that the performances increase when doing Feature Selection, which is basically done in two ways:

1. Basing on mean impurity decrease Gini (or Entropy) Index, then taking only the amount of features whose cumulative importance is higher than a threshold (E.g 90%), as illustrated in figure 3.3.

2. Using regularization techniques such as **Lasso**. Even though this is usually exploited for linear models, it handles multicollinearity between features, which creates also problems to Random Forest, as explained in [6].

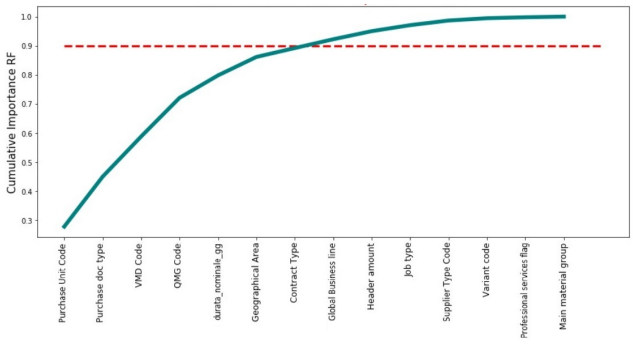


Figure 3.3. Cumulative feature importance Random Forest.

As regarding **Logistic Regression**, the grid has explored the tuning of:

- *Type of regularization*: besides lasso, even **Ridge** and **Elastic-Net**, where the latter is a combination of the first two, as described in [4].
- *Penalty coefficient* for the regularization in matter, in Python referred as 'C', where smaller values correspond to higher penalization strength. For Ridge, it corresponds to shrinking variables coefficients towards zero, while in Lasso to set some of them exactly to zero, selecting this way less features.
  Then, besides these ones to reduce overfitting:
- *Solver*, related to the optimization algorithm aimed at minimizing the regularized cost function.

The best resulted in using Lasso, $C = 0.0048$ - corresponding to 9 features selected - and solver= 'liblinear'.

A resume of the performances at varying the parameters is given in figure 3.4. Increasing C, thus reducing penalization and selecting more features - the model with Lasso keeps improving both on Train and Validation, while the one using Ridge starts getting worse soon. Anyway, results of Logistic are worse than the previous ones, suggesting that a linear boundary is probably not advisable for this data.

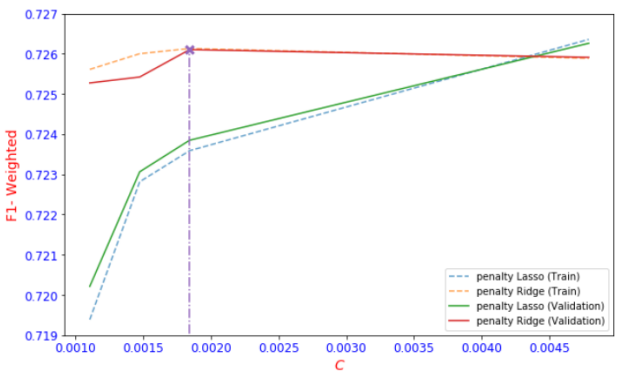


Figure 3.4. Logistic F1 at varying regularization.

For **K-Nearest Neighbours**, it has been explored a set of different choices for the following hyperparameters:

- *N° Neighbors* to consider for each sample when averaging.
- *Distance Metric*, such as *'Euclidean'*, or *'Manhattan'*.
- *Weights*: 'uniform' if the K neighbors are given the same weight -regarldess how distant they are- or 'distance', if the weight decreases with distance when computing average.

In picture 3.5 it is evident that for K=1 the model is extremely in overfitting, since considering 1 neighbor only corresponds to consider the *i-th* sample itself. Indeed the performance for 1 neighbor is 100% on Train, but much lower on the Test Set. Best performance is obtained for K=20 and uniform manhattan distance. From the table of results, it can be noticed that with 10 features rather than 14, the test performance has decreased of 1%.

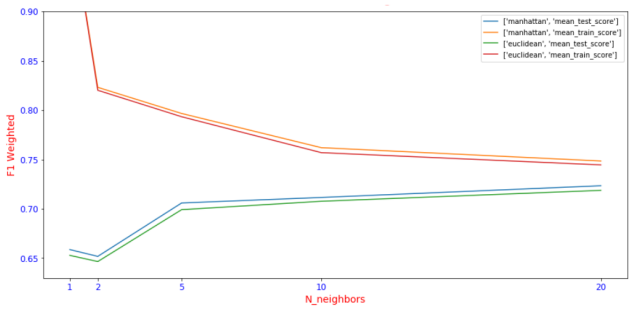


Figure 3.5. Knn F1 at varying the number of neighbors and distance metric.

As concerning **Suppor Vector Machine** the set of hyperparameters investigated concerns both regularization - parameter 'C' - and types of *Kernel*. As for the latter, only *Linear* and *Gaussian ('Rbf')* have been explored. Given the fact that the former has been computationally very expensive, *Polynomial* kernel has not been tried. Also, since it is evident it does not fit well the data, even a *Sigmoid* Kernel has been neglected since it would result in a linear boundary, which would give poor performances. Gaussian Kernel, with C= 10 and default gamma value resulted the best solution for this data, again enlightening that a non-linear boundary, as the one in figure 3.6, seems to be more appropriate.

Even *Stochastic Gradient Descent Classifier*, a Scikit Learn version of SVM which speeds up thanks to the gradient descent algorithm, has been tempted, even though not with high test scores.

Furthermore, several kinds of **Gradient Boosting** methods have been used, in particolar exploring *GradientDescentClassifier*, *XGBRFClassifier* and *XGBClassifier* ScikitLearn packages. The last one resulted in the best effectiveness and efficiency, in terms of scores and execution time.

Although a broad gamma of hyperparameters is avaialable to build a Boosting model, in this project it has been basically tuned the followings:

- *Number of Decision Trees* built.

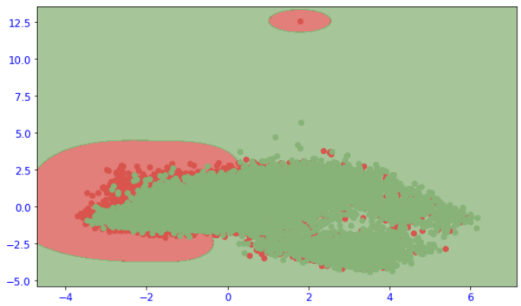


Figure 3.6. Boundary Svm Gaussian Kernel in 2D.

- *Maximum depth* of each Tree, which for Boosting is tipically 1 or 2.
- *Learning rate*, determining the step - thus the speed - at which the booster learns.
- *alpha and lambda*, corresponding to 'l1' and 'l2' norms used for regularization on weights, as for Lasso and Ridge.
- *Min child weight*, used again to control overfitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- *Subsample ratios*, both related to samples and columns, More precisely, it is tuned the percentage of training samples to Train each Tree and also the percentage of columns, for the same reason mentioned in Random Forest, that is building decorrelated Trees.

The best Boosting model found has 150 trees, each with depth=3, learning rate= 0.1, alpha= 0.7, lambda= 0.8 and both subsample ratios equal to 0.8. With this setting of hyperparameters, the features importance is given in figure 3.7.

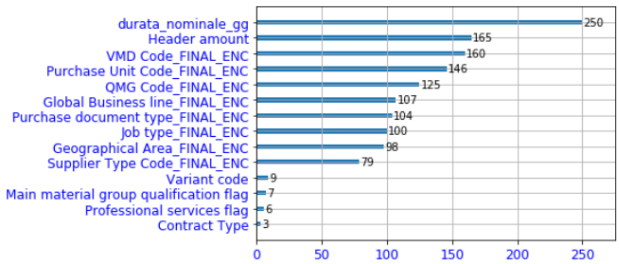


Figure 3.7. Boosting Feature Importances.

Finally, an ensemble approach such as **Voting Classifier** has been carried out. In the same way Random Forest is considered an ensemble method since it exploits many different individual models - the Trees - averaging then the results, this one is an ensemble technique because exploiting several different models. This fact of

averaging many models of very different nature is what makes it more robust, as different studies have shown [22] [27] [12] . There are two ways of averaging models:

1. **Hard Voting** consists of a assigning a label to each sample basing on a simple majority vote. Thus, if there are model A, B and C and the first two of them categorize a sample as class 1, while the third as class 0, the final label will be 1.

2. **Soft Voting** outputs instead the final label as the average probability of each model. Thus, if the probabilities of model A, B and C are repectively 0.4, 0.4 and 1, the average is 0.6 and the sample will be classified as 1, even though two of them predicted the final class to be 0.

The idea behind the latter is that two models having probability 0.4 for a certain sample, do not have a strong confidence on the final prediction, indeed at 0.5 they would classify the sample in the other class. On the other, the third model has high confidence on the prediction.

To run the two voting models, the best algorithms so far mentioned have been considered, then:
1. Random Forest, with feature selection.
2. Svm with Gaussian Kernel.
3. Gradient Boosting (XGB).
The results in the table 3.1 show that for both cases the scores are among the best given all the considered models. In particular, the second type of Voting Classifier is slightly better, as expected for the previous reasons.

3.3.2 Rolling

So far performance for classifiers on the Static task have been reported. A similar approach was followed also for the Rolling task based on a distinct dataset, which is composed by 304'564 samples and 32 features (for additional details see section 2.11) and has been built according to the procedure described in detail in section 1.3.1. In such case, a further *Feature Selection* has been conducted through a combination of both a-priori ideas of dependency with the target derived from the previous exploratory analysis, and features importance resulted from a *Random Forest* algorithm. The result is that from the set of 32 variables, five of them have been eliminated, namely the initial trimester and the four ones related to seasonality. Then, new data has got 27 features.

The algorithms and the tuning procedure exploited are analogue to the ones in the previous section; the results of the different models are reported in table 3.2.

First, we can notice how in most of the algorithms, performance on the test set are roughly equivalent or slightly better than in the train. On one hand, the fact they are similar is good because it reflects we are not overfitting. On the other hand, it must be reminded that the custom encoding (discussed in section 2.7) is done in such a way that noise is added only on the training set. Some of the predictive techniques used for the static data have been neglected because computationally too heavy. This is the case of *K-NN* and *SVM*.

As concerning Random Forest, the best model hyperparameters configuration resulted in: *Gini Index* as impurity metric, *Max Depth* of each of 200 *Decision*

Model Type	Model	F1 Train (Acc)	F1 Test (Acc)	Feats
RF	**RF**	77.2 (76.3)	74.8 (73.7)	27/27
RF	**PCA + RF**	68.5 (67.0)	69.6 (68.2)	21/27
RF	**Feat Sel(Gini)+RF**	76.1 (75.1)	74.0 (72.8)	11/27
RF	**Feat Sel (Lasso)+RF**	77.2 (76.2)	75.0 (73.9)	23/27
Logistic	**Regularized Logistic**	72.9 (75.1)	72.9 (75.3)	27/27
Logistic	**PCA + Logistic**	72.2 (72.2)	72.9(72.8)	12/27
Knn	**PCA + Knn**	86.4 (86.7)	74.1 (74.6)	15/27
SVM	**SGD**	71.0 (74.4)	71.0 (74.4)	27/27
Boosting	**XGBoost**	77.5 (78.5)	76.6 (77.7)	27/27
Boosting	**PCA + Best Boost**	76.3 (77.6)	74.9 (76.3)	21/27
Ensemble	**Soft Voting**	79.0 (78.9)	77.6 (77.4)	/

Table 3.2. Rolling Performances (classification).

Tree equal to 10, *min samples leaf*= 20. Moreover, since classes are unbalanced, a parameter related to *class weights* has been set. Relative features importance are displayed in figure 3.8.

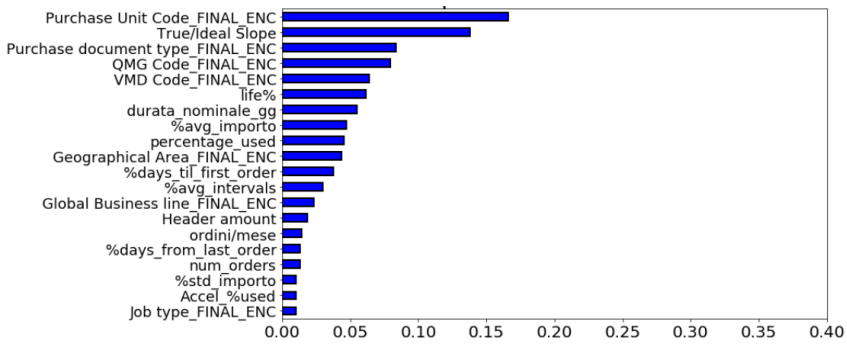


Figure 3.8. Features Importance Rolling Classification.

If performances are assessed not only according to their magnitude but also based on models parsimony, we could conclude that *Random Forest* improves when using only 11 features, selected based on the mean impurity decrease of each one. Also, the one derived by a Lasso feature selection provide both the most parsimonious and effective test score. Performance decreases when using a linear model such as Logistic Regression, whose best parameters correspond to *Ridge Regularization ('l2' norm)*, penalty coefficient set to 0.0005 and using 'liblinear' solver.

Recalling that Logistic uses a *Sigmoid Function* to give in output a score between 0 and 1, the default threshold equal to 0.5 has been changed in such a way to maximize the difference between True Positive Rate and False Positive Rate, consequently changing the final *F1* and *Accuracy* scores. By doing that, it has been obtained the best score reported in the table. In particular, the best threshold found is equal to 0.283. A visual way to understand that is given in figure 3.9.

As said at the beginning of the chapter, *K-NN* is weak with respect to large dataset; that's due to the fact that, given N observations in M dimensions, it has to compute N^N distances between vectors in M dimensions. PCA technique has been

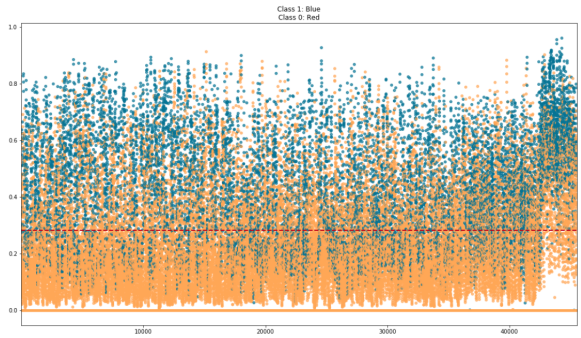


Figure 3.9. Optimal Threshold choice for Logistic Regression.

exploited both in order to reduce the weight of them and to partially mitigate the effect of overfitting. However, given the scores reported in the previous table, it can be noticed how the model on test data does not generalize as well as on the training set. Manhattan distance with uniform weights and 5 neighbours have been used.
As concerning Gradient Boosting, as for the static problem, *XGBClassifier* Python Class has been exploited. Among the many parameters set, there have been used 200 *Trees*, *learning rate* = 0.8 and max depth equal to 1, corresponding to one split only for each tree.

Again, *Tree-Based Methods* have been the most performing ones. The comparison of *ROC Curves* for the best models of Random Forest, Boosting and Logistic Regression is shown in figure 3.10.

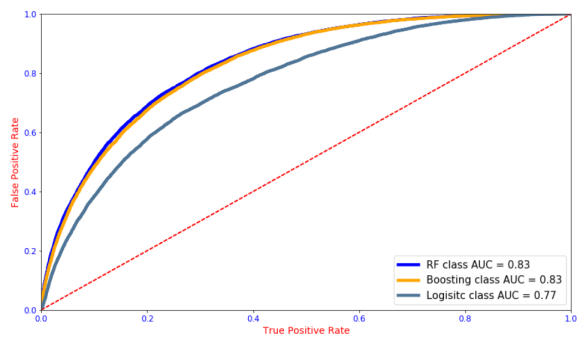


Figure 3.10. ROC Curves Rolling Classification.

Once again, the *Ensemble* tehnique of *Soft Voting Classifier* has been adopted using the most performing models, namely Boosting and Random Forest, without any relevant improvement though.

Besides the global performances so far considered, as mentioned in the paragraph 3.1 it is fundamental to evaluate a model based on the partial performances, that is with respect to the each % of life considered. That is shown in figure 3.11 for the two best methods.

Finally, we can notice those results to be much coherent with the expectation discussed in paragraph 3.1, according to which a rolling model is supposed to have higher performance at increasing the life% of the samples.

	% samples	acc_RF	F1_RF	acc_xgb	F1_xgb
0% life	12.459	0.686	0.684	0.723	0.721
10% life	11.235	0.678	0.682	0.726	0.718
20% life	10.736	0.692	0.699	0.739	0.730
30% life	10.435	0.697	0.707	0.753	0.744
40% life	10.121	0.716	0.727	0.760	0.748
50% life	9.804	0.742	0.754	0.785	0.771
60% life	9.455	0.769	0.781	0.798	0.784
70% life	9.101	0.792	0.802	0.820	0.800
80% life	8.640	0.823	0.832	0.846	0.824
90% life	8.016	0.862	0.869	0.894	0.867

Figure 3.11. Results Rolling Classification for each life%.

3.4 Regression Task

So far the focus has been put on the classification task, pursued by using traditional machine learning algorithms. Before deepening a specular analysis for regression, it has to be recalled how the target has been built. As explained in sections 1.3.1 and 2.5, an important modeling choice has been to create the regression target based on the whole lifetime % of contracts. This has got the double effect of having the Target feature between 0 and 100 and, above all, that the target does not depend on the temporal length of the contracts (at least numerically). Once again, it is also recalled a second important choice, consisting of predicting lifetime only of observations with label equal to 1 (motivation in section 1.3). That being said, it is possible to see how several models fit the data. The indexes used to measure performances are **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**. The former gives the average error as an absolute value, thus it is often not very useful to have a direct understanding of the model goodness, since such value depends on the specific range of the target. A similar observation can be made regarding the latter which, being it not expressed in % and, thus, strictly linked to the target values. However, since in this case the target is a % - then between 0 and 1 - the MAE would correspond to a sort of **Mean Absolute Percentage Error (MAPE)**, because the average error would be surely between 0 and 1. By the way, *MAPE* itself is not going to be used since it would often result as infinite value. Indeed, summing up for each sample the absolute error divided by the true target, it would explode when the denominator -the real target- is zero, which happens for several samples. A partial solution to fix this issue has been automatically made by removing those contracts having a nominal duration lower than 15 days, as described in section 2.3. Nevertheless, *MAPE* would result in extremely high value - even though not infinite - wherever the target is very close to 0, which occurs to all those contracts lasting many years and for which 15 days becomes an almost irrelevant length.

Obviously, here the purpose is to minimize the errors, thus lower scores correspond to better performances. As concerning dimensionality reduction methods, as for the classification case, with 10 principal components it is explained more than 90% of variance. The dataset plotted in two dimension for Regression is not

Model Type	Model	MAE Train (RMSE)	MAE Test (RMSE)	N° feats
RF	**RF**	21.5 (26.6)	22.6 (28.4)	14/14
RF	**PCA + RF**	21.8 (26.9)	23.1 (28.8)	12/14
RF	**Feat Sel(MSE)+RF**	21.6 (26.8)	22.7 (28.5)	8/14
RF	**Feat Sel (Lasso)+RF**	21.8 (27.0)	22.6 (28.4)	7/14
Lin Reg	**Regularized Reg**	24.7 (29.8)	24.7 (29.8)	9/14
Poly Reg	**Lasso + Poly Reg**	23.8 (29.1)	23.8 (29.2)	67/120
Knn	**Knn**	21.0 (26.4)	23.2 (29.5)	14/14
SVM	**PCA+ SVM Linear**	24.8 (30.4)	24.8 (30.3)	10/14
SVM	**SVM Gaussian**	21.0 (27.6)	22.4 (29.4)	14/14
SVM	**SGD**	25.4(31.0)	24.6 (31.0)	14/14
Boosting	**XGBoost**	21.7 (27.3)	23.1 (28.7)	14/14
Boosting	**Feat Sel+XGBoost**	20.4 (26.4)	23.3 (29.1)	7/14
Ensemble	**Voting Regressor**	21.3 (26.7)	22.5 (28.5)	/

Table 3.3. Regression Performances (Static).

reported.

3.4.1 Static

Models treated in regression are pretty much the same ones run in Classification. In table 3.3 *MAE* and *RMSE* for each one are reported. The reason why *RMSE* scores are generally worse is based on the fact that it is derived from *MSE* which penalizes more the bigger errors, since it considers the squared error .

Tuning for *Random Forest* is roughly the same done for classification, with two main differences:
1. The criterion is neither *Gini Index* nor *Entropy* anymore, but it is minimized the *mean squared error*. It could be even minimized the *MAE*, but since *MSE* is a quadratic function, thus also strictly convex, the optimization process is much faster and more effective minimizing the latter.

The same scores of *Random Forest* are obtained with less variables after doing feature selection; these models are clearly better, being more parsimonious. 2. The number of features to randomly consider at each split is now equal to the whole set of features. That is due to the fact that recent studies [8] have proved this to be more efficient for regression problems, rather than taking the square root of the n° features.

Linear Regression has given poor performances; several types of regularization have been examined and the best model has been obtained selecting 9 features with Lasso.

Then, complexity has been increased with **Polynomial Regression** of different degrees, from 2 to 4. It has not gone farther because of the drop in scores and exponential increase of computations. After tuning, the best degree has been found for degree equal to 2. This implies creating new polynomial features, that is adding

to the original variables, both the square of each and, in turn, multiplication of each with all the others, in order to extrapolate their interaction. Doing that, 14 features become 120; with Lasso, only 67 of them have been selected.

As concerning *K-Nearest-Neighbors* and *SVM*, the set of hyperparameters to be tuned is the same to the classification case. For the latter, the best performing is, once again, the Gaussian Kernel.

When applying *Gradient Boosting*, the only thing changed is that the objective function is linear, being a regression problem, while previously it was the logistic loss function, being a binary problem (otherwise multi softmax). It does not show any relevant drop in the results even with half of the variables kept with feature selection.

The most performing technique in this case coincides with the Voting Regressor method, which in this case simply corresponds on averaging the outputs of different models, chosen again to be the most performing ones. With this, consisting of a combination of Gaussian Svm, Random Forest and Boosting, the top performance is reached, consisting of 22,5% of *MAE* and 28.5% of *RMSE*.

A visual overview of the different model performances is given in the plots 3.12, where for each model it has been displayed the prediction -y axes- versus the actual target - x axes - for the test set samples.

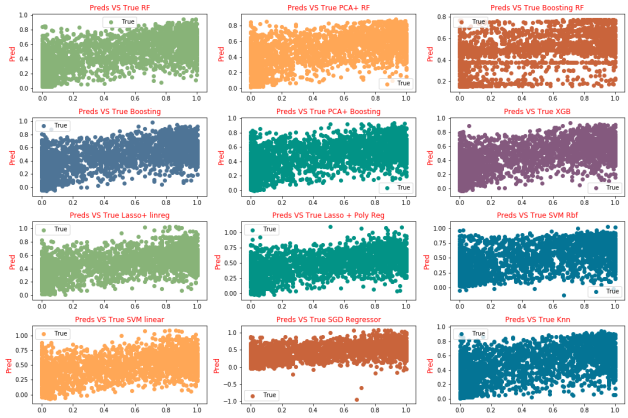


Figure 3.12. Regression Plots (Static).

Clearly, the more such points are disposed along the bisector, the better the performances is. The opposite is also true: the farther they are, the more inaccurate the prediction. It is interesting to see for instance that SGD Regressor (bottom center) has a much tighter band, but this is highly biased because located only in the upper side of the plane.

3.4.2 Rolling

As concerning Rolling Dataset, the results are reported in table 3.4.

It is possible to notice immediately that most of the features do not seem to be quite relevant. The most striking situation is for Random Forest run after a

Model Type	Model	MAE Train (RMSE)	MAE Test (RMSE)	N° feats
RF	**RF**	13.6 (18.3)	14.8 (19.5)	27/27
RF	**PCA + RF**	14.0 (18.7)	15.3 (20.6)	21/27
RF	**Feat Sel(MSE)+RF**	14.0 (18.7)	15.0 (19.7)	7/27
RF	**Feat Sel (Lasso)+RF**	13.6 (18.5)	14.8 (19.6)	18/27
Lin Reg	**Regularized Reg**	16.8 (21.2)	17.1 (21.4)	23/27
Poly Reg	**Lasso + Poly Reg**	15.9 (20.5)	16.3 (20.8)	78/406
Knn	**PCA+ Knn**	13.0 (17.0)	15.7 (20.8)	12/27
SVM	**SGD**	16.8 (22.5)	17.3 (22.3)	27/27
Boosting	**XGBoost**	21.7 (27.3)	15.1 (19.8)	27/27
Boosting	**Feat Sel+XGBoost**	13.6 (18.6)	15.0 (19.5)	23/27
Ensemble	**Voting Regressor**	13.6 (18.3)	14.7 (19.4)	/

Table 3.4. Regression Performances (Rolling).

feature selection consisting of 7 variables out of 27. That is partially explained by the feature importance of Random Forest, reported in figure 3.13.

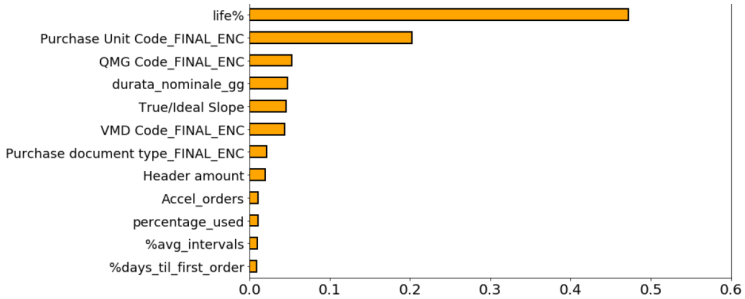


Figure 3.13. Feature Importance Rolling (Regression).

From that, it emerges how life% individually has about half of the total information according to that model, since its feature importance is about 0.5. The explanation of that is exclusively due to the progressive elimination of contracts when they have expired, as mentioned in section 1.3.1. More precisely, contracts having a low target value - meaning a low lifetime% - will consequently have only the first few associated samples, while framework agreements having high target values, will have almost all the related snapshots in the dataset. The result of that is a strict dependency between such feature and regression target which makes life% be the most correlated feature to the target, as earlier illustrated (figure 2.15).

Clearly, it would be desired any model to rely less on such a feature. Especially because in the prediction phase, if a model has to predict a contract with a low life%, then it cannot carry any relevant conclusion basing only on that, since the first snapshots are included for almost all of them; on the other hand, if it is in an advanced stage of its life, then the model can only conclude that it is not one of those with a short life, which is already evident.

However, we can see such feature as the price to pay to have one single model, rather than N separated models for each of the N lifetimes at whom it is desired

to make prediction with a rolling approach. It basically plays the role of a model selector, in the sense that it allows somehow the parametric model to canalize the prediction to a specific model.

As done for classification, even in this case the performances are reported for each life%, as illustrated in figure 3.14.

	% samples	MAE_RF	RMSE_RF	MAE_xgb	RMSE_xgb
0% life	18.425	22.033	27.375	22.619	27.989
10% life	14.235	20.314	24.623	20.309	24.617
20% life	12.524	17.651	21.213	17.467	21.325
30% life	11.492	15.334	18.815	15.383	18.884
40% life	10.414	13.178	16.036	13.198	16.062
50% life	9.329	10.810	13.096	10.917	13.346
60% life	8.131	8.532	10.326	8.786	10.883
70% life	6.918	6.451	7.876	6.787	8.545
80% life	5.335	4.666	5.533	5.327	6.693
90% life	3.195	2.396	2.849	3.945	5.361

Figure 3.14. Rolling performance per each life% (Regression).

The procedure used for parameter tuning is the same of the previous sections and results for each paramaters combination are thus omitted.
For a complete analysis of the error it is needed an ad-hoc exploration of it basing on the specific context. For example we might wonder whether at increasing nominal duration of contracts, the error tends to be higher or lower. In figure 3.15 we can notice as the higher % of errors are related to contracts with very short lifetime, while at increasing their nominal duration the % of error also decreases.

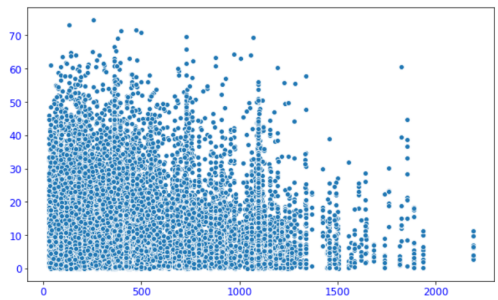


Figure 3.15. Regression % error in relation to Nominal Duration.

This is also shown below (figure 3.15), where for each duration quartile is reported the average percentage error.

The final set of plots in figure 3.17 summarizes both the average error basing on lifetime percentage and on nominal duration.

	mean		std	
	delta errore gg	delta errore %	delta errore gg	delta errore %
durata_binned_quartiles				
(29.999, 221.0]	22.770460	16.970553	20.803092	12.973345
(221.0, 394.0]	52.754508	15.834342	44.991605	13.187325
(394.0, 730.0]	78.074583	13.793755	75.461884	12.708046
(730.0, 2191.0]	129.667879	12.442258	119.331566	11.266254

Figure 3.16. Regression % errors in relation to Nominal Duration for each lifetime.

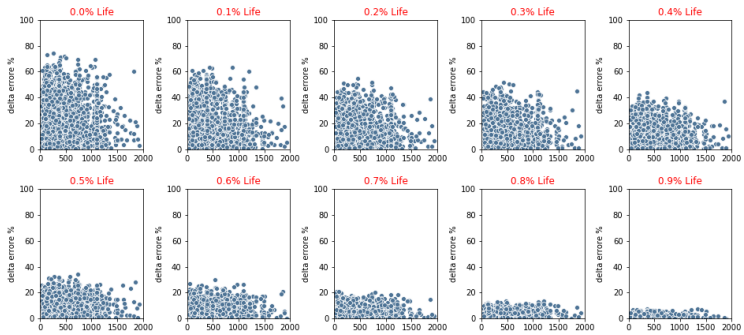


Figure 3.17. Regression average % error for each lifetime at varying Nominal Duration.

3.5 Summary Best Results

Before going forward, best test results obtained by the previous supervised models are reported in table 3.5.

Static

Task	Model Type	Test Score
Class	Boosting	73.8% (F1)
Reg	RF	23.1% (MAPE)

Rolling

Task	Model Type	Test Score
Class	Boosting	76.6% (F1)
Reg	RF	14.8% (MAPE)

Table 3.5. Best Performances.

Such results are the best ones in terms of highest weighted *F1 score* and lowest *MAE* - which in our case plays the role of a *MAPE* - without taking into account parsimony. In other words, ideally it should be picked the best model not only giving the best scores on the test, but one 'enough good' which is also parsimonious, for instance in terms of features used; this is the reason why the expensive ensemble methods such the ones based on averaging several models have not been considered in the summary, even though giving slightly better results.

3.6 Single Lifetime Approach

In paragraph 1.3.1 it was explained the parametric approach to provide a rolling prediction. In particular, it was highlighted the fact the parametric dataset is composed by a series of snapshots of each contract, considered in different stages of their lives, as shown in table 1.2.
Furthermore, when discussing features importance derived from the Random Forest

models, a deep focus on the life% attribute has been put, concluding that its high importance is the price to pay to have one single model rather than many different models for each lifetime considered.

Theoretically, we could indeed build a different machine learning model for each lifetime.It might be absolutely possible that for instance Boosting would be the most performing up to the 40% of life, while Logistic Regression for higher lifetimes. Clearly, it must be considered not only the overall **effectiveness** but also the **efficiency**, trying to find the best *trade-off* between the two. In fact, in terms of effectiveness, having several different models would be likely to lead to better results, but it would be extremely more expensive in terms of development and maintenance.
Since in real world high efficiency is often required at the price of less effectiveness, one single (parametric) model is surely going to be preferred. To have a broad idea of the loss in terms of performances, it would be interesting to pick a specific lifetime and see how worse the parametric model is in comparison.
Such single lifetime approach as been exploited considering the 30% of life.

The datasets consists now of 31720 samples for the classification task and 10223 for regression, consequently to the progressive elimination of consumed contracts and to the requested pipeline, consisting of applying regression on lifetime percentage only on the fully used contracts (sec 1.3.1). Also, given the different composition of data, some variables have been eliminated due to their high correlations, resulting in a total of 23 features.

Best results for each model used to predict consumption in binary classifications are reported in table 3.6, from which we can deduce how linear models perform much worse than non linear ones.

Model Type	**Model**	**F1 Train (Acc)**	**F1 Test (Acc)**	**Feats**
RF	**RF**	77.0 (76.3)	73.6 (72.8)	23/23
RF	**PCA + RF**	68.7 (67.6)	69.3 (68.2)	21/23
RF	**Feat Sel(Gini)+RF**	77.3 (76.6)	73.3 (72.5)	11/23
RF	**Feat Sel (Lasso)+RF**	77.3 (76.6)	73.5 (72.6)	22/23
Logistic	**Regularized Logistic**	70.4 (72.6)	69.7 (72.1)	23/23
Logistic	**PCA + Logistic**	69.8 (68.9)	70.1 (69.3)	21/23
Knn	**PCA + Knn**	81.0 (81.4)	73.3 (73.7)	15/23
SVM	**SVM Linear**	69.6 (70.2)	69.8 (70.5)	23/23
SVM	**SVM Gaussian**	82.4 (81.9)	74.1 (73.5)	23/23
SVM	**PCA + Best SVM**	75.5 (74.8)	72.3 (71.4)	10/23
SVM	**SGD**	70.1 (71.9)	70.2 (72.2)	21/23
Boosting	**XGBoost**	74.5 (74.6)	74.5 (75.6)	23/23
Boosting	**PCA + Best Boost**	78.4 (79.2)	74.0 (74.8)	21/23
Ensemble	**Hard Voting**	78.1 (77.5)	74.4 (73.6)	/
Ensemble	**Soft Voting**	77.3 (76.6)	73.8 (73.0)	/

Table 3.6. Direct Rolling: 30% lifetime performances.

In table 3.7 results of continuous predictions of lifetime are reported.

Model Type	Model	MAE Train (RMSE)	MAE Test (RMSE)	Feats
RF	**RF**	13.5 (16.6)	15.0 (18.4)	23/23
RF	**PCA + RF**	14.0 (17.0)	15.5 (18.9)	21/23
RF	**Feat Sel(MSE)+RF**	13.5 (16.7)	15.0 (18.5)	12/23
RF	**Feat Sel (Lasso)+RF**	13.5 (16.7)	15.0 (18.4)	19/23
Lin Reg	**Regularized Reg**	15.4 (18.7)	15.7 (19.1)	19/23
Poly Reg	**Lasso + Poly Reg**	15.4 (18.7)	15.7 (19.1)	19/23
Knn	**PCA+ Knn**	15.2 (18.5)	15.7 (18.5)	12/23
SVM	**Linear Kernel**	16.0 (19.6)	16.2 (19.9)	23/23
SVM	**Gaussian Kernel**	15.4 (19.0)	15.8 (19.5)	23/23
SVM	**SGD**	16.2 (19.6)	16.1 (19.9)	23/23
Boosting	**XGBoost**	13.4 (17.0)	15.2 (18.6)	23/23
Boosting	**Feat Sel+XGBoost**	14.7 (18.1)	15.4 (18.8)	19/23
Ensemble	**Voting Regressor**	14.1 (17.2)	15.1 (18.5)	/

Table 3.7. Regression Performances Rolling 30%.

3.6.1 Relevant Comparisons

In order to know the loss in terms of effectiveness when dealing with all the lifetimes all together, results on the test set have been compared in table 3.8. In order to compare not only the scores derived from both the parametric and the direct approach (last two columns), but also the improvement of predictions made at 30% of contracts lives in respect to the one computed before they begin to be valid (static approach), it is necessary to perform a small change on the latter. More precisely, because of the progressive elimination of samples -corresponding to consumed contracts- the static dataset has got 38850 samples, while the number of contracts (samples) 'survived' until the 30% of their life is 31720. Therefore, to have a meaningful comparison between the static performance and the rolling at 30%, the static dataset has been filtered in such way to have the same samples of the rolling datasets.

Task	Static	Comparable Static	Parametric Rolling 30%	Direct Rolling 30%
Class (F1)	73.8%	71.6%	74.4%	74.5%
Reg (MAPE)	23.1%	16.3%	15.3%	15.0%

Table 3.8. Comparison between best performances of parametric and direct rolling approaches.

As concerning classification task, everything seems to be linear and coherent, since the (comparable) static F1 is lower than the parametric rolling, which in turn is slightly lower - 0.1% - than the direct rolling. That means that the loss of effectiveness from passing from several individual models - one for each stage of life - to one parametric is only 0.1% of weighted F1.

With a similar reasoning, we can conclude that the effectiveness loss for regression correspond to 0.3% of average percentage error. Moreover, the fact that the static performance in regression are much worse than the one in the comparable static, gives us some more information. Considering that the latter is based on a subset of the static dataset, with roughly ~7000 samples less than the former corresponding to exploited contracts, we can conclude that in general it is harder to predict well the lifetime of contracts which are going to be end soon, than those having a longer lifetime. More specifically, the contracts which are consumed in the first 30% of life, contribute about the 7% of average percentage error in prediction, which is roughly one third of the overall error.

The overall set of validation results obtained for the direct rolling approach are omitted, since same methods previously used for regression and classification have been here exploited too. Clearly, in the previous table, the best results obtained have been reported, mainly derived again from tree based methods such as Boosting and Random Forest.

Chapter 4

Deep Learning Models

The results of the traditional Machine Learning models previously explored suggest that linear models do not suit well with both Rolling and Static datasets. Thus, more complex boundaries and curves for classification and regression have been considered as viable approaches to increase model performance. **Artificial Neural Networks (ANN)** have been the most emerging tool in the Artificial Intelligence field in the last years due to their ability to learn complex relations from data. It must be noticed that without any hidden layer, they would result in linear models [24]. Thus, it has been investigated the use of several ANNs configurations increasing gradually their complexity, ranging from a single hidden layer network up to *Deep Neural Networks*.

4.1 Overview

Starting from the datasets described in section 2.11, the first big issue has been handling categorical variables either with **Embedding Layers** or giving them as a numerical input derived from the **Custom Target Encoding**. As explained in 2.7, even when exploiting Categorical Embedding, the two features having highest dispersion -namely *VMD Code* and *QMG Code*- have been encoded, since attempting to learn weights of classes whose frequency is one would be pointless. In both cases, Classification and Regression tasks have been clearly faced by constructing networks of different nature. The previous configurations have been explored both on the *Static* and *Rolling* datasets, given their different set of features. Moreover, as concerning the Rolling approach, both **Traditional and Recurrent Neural Networks** have been exploited.

Feature Selection has not been carried out since it is not needed when dealing with Neural Networks. In fact these are able to learn weights which allow to discriminate the importance of each variable by themselves, as shown in [16]. Thus, the three correlated variables - namely *'Purchase Country Code', 'Country Code', 'Global Purchase Unit Code'* - removed in 2.10 have been kept.

4.2 Optimization Approach

For each Network Configuration, iterative Grid Searches (3.2) have been exploited. However, given the high amount of *hyperparameters* to be tuned in Deep Learning, such search has been gradually done with a procedure aimed at first finding the best network structure in terms of number of hidden layers and neurons, then exploring several combination of parameters mainly related to regularization, in order to reduce overfitting. The general procedure used for building each ANN is roughly described in the following steps:

1. **One hidden Layer**. The number of neurons together with the **batch size** have been tuned. The former is initialized to be either the double number of inputs, either the same or the half, while batch size is set to either 32 or 128.

2. Adding **Dropout**, between hidden and output layers. This consists of randomly dropping a set of links at each batch, so that the neural network does not learn to strictly rely on some specific nodes. The probability at which such links between neurons are removed is initialized around 0.5, since recent studies have proven that to be a reasonable choice to prevent overfitting, as explained by *J. Benjo et al.* in [5].

   Deductions from first results: first conclusions are drawn to get an idea of the goodness of the level complexity. If the best validation loss is obtained with high number of neurons, which correspond to a more complex model, then more layers are going to be added(even thought one layer with enough neurons could theoretically represent any multilayer structure, as explained in [17] and [18]. With a similar logic, preferred low probabilities for dropout mean that most of the links are useful, thus it is probably better not to lower the complexity. Finally, the batch size implies bigger gradients in the **back-propagation**(referenza appendice), thus bigger modification of the weights. If a low batch size is preferred -meaning a lower validation loss- then the dataset has probably many close local minima which are easier to achieve with smaller improvements.

3. **Adjusting Network Structure**: given the behaviour in the first grid, it is tried either a gradually increasing number of hidden layers or a smaller number of neurons in the first and unique hidden layer.

4. **Parameters Optimization**: being the main best structure caught, other hyperparameters such as **learning rate** and **weight decay** are tuned. The idea behind these two is to do bigger steps at the beginning, when we are likely far from a local minimum, and smaller steps when going on with the iterations, as illustrated in figure 4.1. They help doing this, even though optimizers such as **Adam (Adaptive Moment Estimation)** have already got an adaptive nature for updating the learning rate at each iteration, as described in [13].

5. **Adding more regularization**. Theoretically, the more hidden layers we add, the more the model tends to go towards overfitting. This may be compensated by regularization terms added to the loss function -such as l1 and l2

norms of weights-, which give more penalty to higher weights. However, these ones have not been tuned in this framework, since the Dropout itself can do regularization effectively and efficiently by itself, as stated in [5].

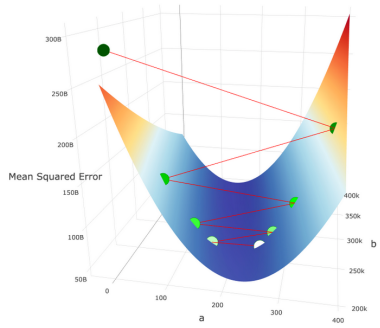


Figure 4.1. Ideal decreasing Learning Rate.

It is fundamental to highlight that tuning the number of **epochs** has been intentionally avoided. Indeed, put it sufficiently high, **Early stopping** technique has been exploited to interrupt the training when performances on Validation stop improving much. The idea is that by going on with the epochs, the perfomances on the Training and Test sets tend to have the shapes of figure 4.2, where after a certain point the Train loss (MSE) keeps decreasing but the Validation one starts to increase. Before starting to increase, it usually gradually slowers the decrease; that is the reason why early stopping is used, to truncate the training when we are about to reach a local optimum on validation. Finally, it must be underlined that **activation functions** on hidden layers have been left as default- since for instance **Relu** activation has been proven to be a good compromise between effectiveness and speed in traditional ANNs, same as **Tanh** activation the most appropriate for RNNs. Clearly, the activation function on the output layer strictly depends on the task, thus it has been set basing on that.

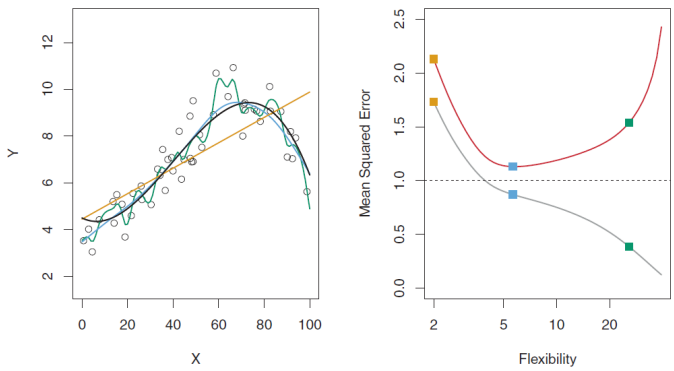


Figure 4.2. Left: overfitting and underfitting. Right: bias-variance trade-off [7].

4.3 Classification Task

It must be pointed out that when fitting the model, class weights have been specified basing on the % of class 0 and class 1, in order to give more weight to the less frequent label.

4.3.1 Custom Loss Function

Both for Static and Rolling datasets, the output layer of the network is composed of one single node with a **sigmoid** activation, which ensures the output to be between 0 and 1. Then, the predicted class of each sample is 1 if the score is higher than 0.5, else 0, treating such values as if it were a probability. Above all, much attention has to be put on the **Loss Function** L(p) used to predict a binary label, which in **Keras** is by default the **Binary Cross-Entropy**, whose formula is

$$L(p) = ylog(p) + (1 - y)log(1 - p) \tag{4.1}$$

However, if the classes are unbalanced, as for the *Rolling* case, minimizing such objective function results in practice in maximizing the *accuracy*, which in turn leads the model to predict almost always the most frequent class and then having poor performance in the *recall* of the less frequent one. To avoid this, a new **customized loss** which highly penalizes predicting mostly one single class was created by using the Keras backend API. The code to do that is reported in figure 4.3, from which it is possible to see how the optimization of the 1st term consists of minimizing the sum of *False Positive and False Negative errors*.

```
def loss_f1(y_true, y_pred):
  neg_y_true= 1- y_true
  neg_y_pred= 1- y_pred

  FP= K.sum(neg_y_true * y_pred)
  TN= K.sum(neg_y_true* neg_y_pred)

  TP= K.sum(y_true*y_pred)
  FN= K.sum(y_true* neg_y_pred)

  FPE= FP/ (FP+TN) #FP error
  FNE= FN/ (FN+ TP)

  return (K.square(FPE + FNE) + K.square(FNE - FPE))/2
```

Figure 4.3. Custom Loss function in Keras backend.

To that it is added a second term which corresponds to the squared difference between the previous two, aimed at avoiding to mostly minimizing only one of the two. In fact, without the second term and with unbalanced classes, one effective minimization might lead to predicting again one single class, in order to reduce either false positive or false negative error; minimizing also their difference, leads instead to a more balanced performance in which te number of false positive and false negative are roughly equal. Therefore, optimizing such personalized objective function results in maximizing the *F1 score*.
Such custom loss function has been used when dealing with the *Rolling* dataset, while for the *Static* there is no need since classes are pretty balanced.

4.3.2 Static Results

ANN from Encoding

As regarding the Static dataset, whose baseline accuracy is ~55% by predicting always class 0, using all numerical inputs after encoding categories (2.7) it has been obtained a maximum accuracy of 73.9%. Following the greedy procedure earlier explained, it has been obtained a final Grid whose results are illustrated in figure 4.4.

	n_layers	nodi_hiddens	batch_size	dropout	lr	Train Accuracy	Test Accuracy	Train F1	Test F1
2	2	[64, 32]	32	0.5	0.001	0.734	0.739	0.733	0.739
6	2	[64, 32]	64	0.5	0.001	0.733	0.737	0.731	0.736
0	1	[64, 32]	32	0.5	0.001	0.733	0.734	0.731	0.733
4	1	[64, 32]	64	0.5	0.001	0.731	0.734	0.73	0.733
5	1	[16, 16]	64	0.5	0.001	0.729	0.733	0.728	0.733
3	2	[16, 16]	32	0.5	0.001	0.729	0.732	0.728	0.73
1	1	[16, 16]	32	0.5	0.001	0.728	0.731	0.727	0.729
7	2	[16, 16]	64	0.5	0.001	0.725	0.726	0.723	0.724

Figure 4.4. Grid search results using encoded categories.

The best model found is composed by two hidden layers - having respectively 64 and 32 neurons- 32 samples entering in batch as input, dropout probability set to 0.5 and Adam's learning rate equal to 0.001. In figure 4.5 various curves, related to both Train and Validation- are represented at increasing the number of epochs. First one on the left represents the Accuracy.

The one in the middle, is related to a **custom F1 metric** built using the Keras backend -namely *Tensorflow*- as shown in the snippet 4.6.

Finally, it is plotted the curve related to the Loss function described in the previous paragraph.

ANN with Categorical Embedding

By converting the categorical features into new lower dimensional vectors through **Embedding** [2.7], there is the problem of choosing the new embedding size for each categorical variable. Clearly, if the reduction is too ambitious- that is it is tried to transform high dimensional vectors into very low dimensional ones, the Network may not manage to learn relationships able to keep the most of the original information. On the contrary, if the new dimension is not enough lower than the original one, the embedded components may not bring much more information than the individual

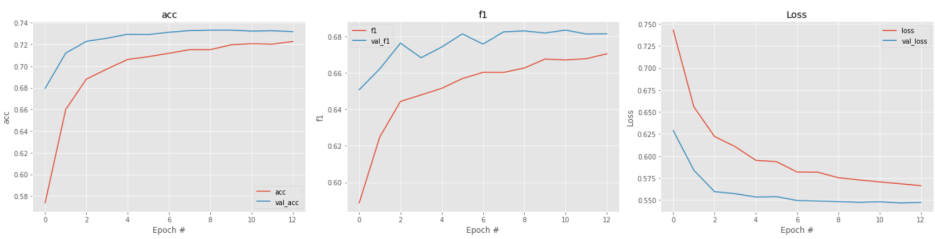


Figure 4.5. From left: Accuracy, F1 and Loss Curves.

```
def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    # tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)
```

Figure 4.6. Custom F1 metric with Keras backend.

	original_cardinality	embedded_size
Purchase Unit Code	76	19
Country Code	59	16
Purchase Country Code	13	7
Global Purchase Unit Code	13	7
Global Business line	8	6
Geographical Area	5	4
Supplier Type Code	5	4
Purchase document type	4	4
Job type	3	3

Figure 4.7. Original and Embedded size for each Categorical Variable.

classes. As for other hyperparameters, such embedding size has to be tuned. As a starting point, there are various rules of thumb, such as for Dropout or batch size. In this project the following options to calculate the embedding size of a categorical variable having cardinality D have been considered:
1. embedding size= k*D, where $k \in [0, 1]$, corresponding to a fraction of D.
2. embedding size= $\sqrt{D}$.
3. embedding size= min(1.6* D** 0.56), 600), which comes from recent studies experiments [1].

The last method resulted to be the best. In particular, the relative embedding size for each categorical variable applying that formula is reported in table 4.7.

As concerning the construction of the ANN, it has been done in three steps:
1. Only Categories: First, an ANN is built taking only the categorical inputs, using Embedding layers.
2. Only Numeric: Separately, an ANN based exclusively based on numerical inputs is trained.
3. Mixed inputs: Finally, a mixed ANN is constructed by concatenating the Embedding layers with the numerical one. The ideal would be to train both the networks -one related to categories and the other to numerical inputs- separately and concatenating them afterwards. Nevertheless, such two blocks of inputs have been merged immediatly -that is as a new concatenated input layer- due to computational issues when trying the ideal approach.

The fact of separating the networks is useful to have a broad idea of the improvement when passing from using only categorical variables - excluding the two encoded - and then even the numerical ones. In other words, we expect the mixed network

	n_layers	nodi_hiddens	batch_size	dropout	Train Accuracy	Test Accuracy	Train F1	Test F1
2	2	[32, 16]	128	0.5	0.731	0.729	0.731	0.729
6	2	[32, 16]	512	0.5	0.73	0.729	0.729	0.729
5	1	[128, 64]	512	0.5	0.731	0.728	0.73	0.728
1	1	[128, 64]	128	0.5	0.733	0.727	0.731	0.727
4	1	[32, 16]	512	0.5	0.73	0.728	0.729	0.727
7	2	[128, 64]	512	0.5	0.732	0.727	0.732	0.727
0	1	[32, 16]	128	0.5	0.728	0.726	0.727	0.726
3	2	[128, 64]	128	0.5	0.733	0.725	0.733	0.726

Figure 4.8. Result using only categorcal variables (1).

	param_batch_size	param_dropout	param_epochs	param_nodi_hidden	mean_test_score	mean_train_score
1	512	0.5	80	128	0.701323	0.702448
3	512	0	80	128	0.699784	0.703547
2	512	0	80	32	0.692747	0.695134
0	512	0.5	80	32	0.691788	0.693215

Figure 4.9. Results with only numerical inputs (2).

to perform better than the separate one, otherwise there is something wrong, unless if either categorical or continuous features do not carry much information.

In particular, the results of the grid searches for the separate categorical and numerical ANNs, are illustrated respectively in figures 4.8 and 4.9. As we can see, the best performance obtained on the test in respect to the weighted F1 score are 72.9% when using only categorical features and 70.2% when using the numerical ones, which include the encoded versions of 'VMD Code' and 'QMG Code'.

To have a direct representation of the third version, with merged embedded and numerical inputs, it has been plotted through the Keras built-in function one of the two hidden layer configurations explored, as illustrated in B.4. The red square highlights the numerical input, which have dimension equal to 8, which is indeed the number of numerical, binary and encoded features. That obviously does not have any embedding layer, rather it goes into the concatenation like this given its nature.

It is interesting to notice a substantial improvement -aligned with the expectations- when applying such mixed inputs approach, whose results are reported in figure 4.10. We can notice that the performance has increased to 75.2% with a NN composed of one layer having 64 neurons, dropout equal to 0.5 and batch size 32.

	n_layers	nodi_hiddens	batch_size	dropout	Train Accuracy	Test Accuracy	Train F1	Test F1
1	1	[64, 64]	32	0.5	0.751	0.752	0.751	0.752
3	2	[64, 64]	32	0.5	0.753	0.751	0.753	0.751
0	1	[32, 16]	32	0.5	0.75	0.75	0.749	0.749
2	2	[32, 16]	32	0.5	0.749	0.744	0.749	0.744

Figure 4.10. Results using a mixed input (3).

A final overview of the performances of each deep learning model is given in table 4.1.

Neural Network Inputs	F1 Train (Acc)	F1 Test (Acc)	N° feats
Numerical + Encoding	73.3 (75.4)	73.9 (73.9)	17/17
Only Embedded	73.1 (73.1)	72.9 (72.9)	9/17
Only Numerical	70.2 (70.4)	70.1 (70.9)	8/17
Numerical + Embedded	75.1 (75.1)	75.2 (75.2)	17/17

Table 4.1. Best results for traditional ANN on static dataset.

4.3.3 Rolling Results

Given the unbalance of classes when dealing with the rolling dataset, the same scheme of the previous section has been followed, with attention on specifying the class weights parameter in the Keras fit method, in particular giving more weight to the minor class. Results are displayed in the table 4.2.

Most performing NN corresponds to the case where categorical variables are handled with embedding layers and, obviously, adding also the numerical input. Though, it is important to notice that with the only numerical features -comprehensive of the two encoded variables with very high dispersion- the performances are very similar, thus they must be the most informative part of the data.

In addition to the configurations explored for the static dataset, here also **undersampling** has been tried in orde to face classes unbalance. Doing that the performances apparently decrease, since the *F1 score* is usually calculated as a *weighted* mean of the relative *F1* for the two classes, in particular giving more weight to the majority class.
That is done this way because for this project there is not any specified preferences between false positive or false negatives, otherwise we could penalize the predictions if they do not predict well one specific class. That being considered, the simple average of the performances would result in lower scores and -clearly- undersampling would outperform the other models in terms of F1. Moreover, in the case of the mixed input configuration, it has also been tried to train separately the two networks- one for categorical variables and the other for numerical ones- merging them in a common layer above which a new neural network is built. Even if close to the ideal case described in the previous paragraph, this is slightly different; in fact, given the technical issues when trying to combine them with the Keras API, it

Neural Network Inputs	F1 Train (Acc)	F1 Test (Acc)	N° feats
Numerical + Encoding	75.7 (77.2)	75.7 (77.2)	27/27
Undersampling	74.6 (74.7)	73.9 (74.0)	27/27
Only Embedded	70.7 (74.3)	70.7 (74.2)	6/6
Only Numerical	76.2 (76.9)	76.4 (76.9)	21/21
Numerical + Embedded	78.6 (79.4)	77.5 (78.4)	27/27

Table 4.2. Best Classification results ANN on rolling dataset.

	% samples	acc	F1
0% life	12.459	0.732	0.728
10% life	11.235	0.734	0.726
20% life	10.736	0.750	0.739
30% life	10.435	0.756	0.742
40% life	10.121	0.765	0.749
50% life	9.804	0.785	0.769
60% life	9.455	0.798	0.781
70% life	9.101	0.815	0.797
80% life	8.640	0.843	0.826
90% life	8.016	0.881	0.866

Figure 4.11. Results rolling classification per each lifetime.

Categorical Inputs Handling	Num layers	hidden neurons	Batch Size	Dropout	Learning Rate
Custom Encoding	1	64	128	0.5	0.001
Feature Embedding	1	64	32	0.5	0.01

Table 4.3. Best configurations for Rolling Classification.

has been simply exported the output at a specific layer of each of the two separate networks, then it was concatenated as a new input to give to another NN.

To get an idea of the scores for each lifetime cut, partials scores are reported in figure 4.11.

Best network structures and params are reported in table 4.3.

4.4 Regression Task

4.4.1 Static Scores

It is reminded there are ~17'000 samples and 17 features in the prediction at t_0 of fully used contracts whose label is 1. As previously explained, it is possible to exclude the three correlated features, but NNs do not need that. Actually, all the configurations have been run both with 17 features and 14, with better performance in the former.

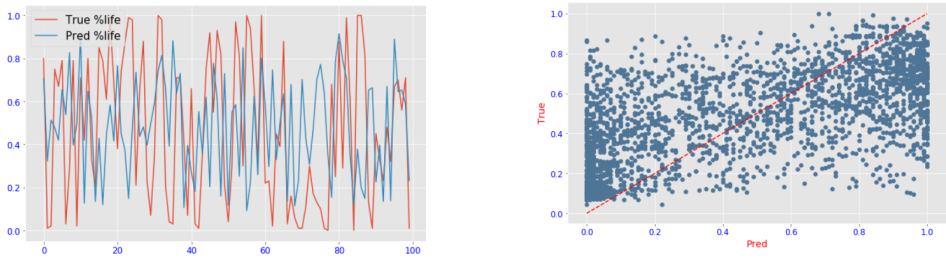
With an analogue procedure to the one applied for the classification case, both encoding and embedding of categories have been applied, with results summarized in table 4.4.

Categorical Inputs Handling	MAE Train (RMSE)	MAE Test (RMSE)	N° feats
Custom Encoding	23.0 (29.0)	23.2 (29.2)	17/17
Feature Embedding	22.7 (28.1)	23.1 (28.6)	17/17

Table 4.4. Best results for traditional ANN on static dataset.

Categorical Inputs Handling	Num layers	hidden neurons	Batch Size	Dropout	Learning Rate
Custom Encoding	3	[64, 32,16]	128	0.5	0.01
Feature Embedding	1	64	32	0.5	0.0001

Table 4.5. Best configurations for Static Regression.



(a) (b)

Figure 4.12. Predictions against True values.

All the parameters combinations are not going to be reported due to their high amount, although the respective best structures found out are reported in table 4.5.

The configuration exploiting embedding layers seems to be slightly better. Not only because it has slightly better test scores, but also because it achieve that using only one layer rather than three, which is computationally lighter.

It has to be highlighted that being the target a percentage, *MAE* plays a role similar to the *MAPE*, since it corresponds to the average percentage error. Further, the latter cannot be used in this case since many target values are very close or exactly equal to zero; indeed it is calculated by averaging the error divided by the true value of each sample and -because the denominator is often zero- it explodes to infinite.

Besides *MAE* and *RMSE* scores, it is possible to get an idea of how good continuous predictions are, taking a random sample of them and comparing with the actual values, as in the plot 4.12a. Here, an hundred random samples resulted from the embedding configuration are plotted.

Such graph may be good to have a first idea of how the model is performing. Above all, it let us understand whether the prediction range is actually among 0 and 1 or not. For instance, in this specific case, it seems to barely predict contracts lifetimes to be under 20% and in any case, it seems catching batter longer contracts than shorter ones.

A more complete way to visualize that is plotting the all predictions against the true values, as in the graph 4.12b

From that, it is clearly how most of the predictions are pretty far from the ideal case (red line). Most of the errors seem to be for the extreme case of too short or too long contracts and predictions look at least coherent since the amount of points very far from the true values is much lower than the ones close to the red line.

	mean		
	durata_nominale_gg	delta errore gg	delta errore %
life%			
0.0	486.167212	125.225107	27.084243
10.0	492.674431	100.285893	21.984752
20.0	509.937425	87.995271	18.666712
30.0	522.622295	81.300252	16.553463
40.0	533.391462	72.389811	14.268995
50.0	547.928110	67.483705	12.417274
60.0	551.700649	57.156189	10.338607
70.0	559.003268	47.916595	8.483617
80.0	555.364407	37.616359	6.658190
90.0	542.733491	29.481751	5.480396

(a) Performances at varying the lifetime.

	mean		std	
	delta errore gg	delta errore %	delta errore gg	delta errore %
durata_binned_quartiles				
(29.999, 221.0]	24.828002	18.468684	21.631683	13.395346
(221.0, 394.0]	61.404446	18.412555	47.885218	14.011667
(394.0, 730.0]	90.299480	16.076414	77.950300	13.072108
(730.0, 2191.0]	154.968849	14.855283	125.247752	11.847055

(b) Overall performances based on different *Nominal Duration* quartiles.

Figure 4.13. Rolling Regression results using *NNs*

4.4.2 Rolling Scores

Following the same approach just used, best results found are reported in table 4.6.

Categorical Inputs Handling	MAE Train (RMSE)	MAE Test (RMSE)	N° feats
Custom Encoding	15.6 (20.2)	16.0 (20.5)	27/27
Feature Embedding	15.3 (20.0)	15.9 (20.5)	27/27

Table 4.6. Best Regression Results Rolling.

We can see how, in respect of the best static score, corresponding to 23.1% of average error, the best performance for rolling is improved about 8%, being its error 15.9%.
The best parameters combination is instead presented in table 4.7.

Categorical Inputs Handling	Num layers	hidden neurons	Batch Size	Dropout	Learning Rate
Custom Encoding	1	128	64	0.5	0.001
Feature Embedding	2	64	32	0.5	0.0001

Table 4.7. Best configurations for Rolling Regression.

With analogue considerations as the ones made for the traditional machine learning models, the average error is plotted in respect to life and duration.

Again, given any lifetime%, the higher the nominal duration, the lower the error, as illustrated in the plots in figure 4.14.

4.5 Recurrent Neural Networks

4.5.1 Reshaping Data

Since the Rolling dataset is composed by a certain number of copies of each contract considered in different lifetime percentages, it would be very powerful a method which allows to keep track of the evolution. That is, in other words, a method with memory.
One way to achieve that would be creating new lagged features for each sample

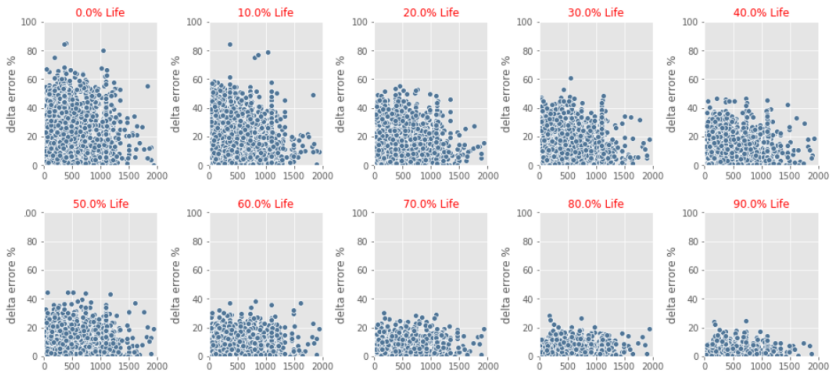


Figure 4.14. Average error at varying *Nominal Duration* for each lifetime considered.

corresponding to the features in the previous steps. The new input variables would contain this way both current and past information for each contract and they could be then given to any machine learning model. However, doing this way, once decided the maximum window until which creating lags, it would be difficult to handle the first snapshots of each contract; for instance, if it has been established to use 3 lags, at 10% of any contract life, there could be created only one real lag for each numerical features, since the first and unique snapshot would be related to the 0% of life.

A better way to achieve the same goal would be by using *Recurrent Neural Networks (RNN)*, which are built in such way to keep track of the past information, as explained in [3] and [9]. Before giving data in input to any *RNN*, it must be reshaped in a proper way. The objective is to make it pass from two to three dimensions, that is transforming samples in **tensors**. Each specific tensor would represent the whole contract sequence, composed by all the various lifetimes it has been cut.

Before doing that it must be considered that for construction each contract will have ten samples if it has not been consumed within its 90% of life, otherwise only some of the first snapshots. Given this fact that each contract has got a different number of associated samples in the dataset, the maximum sequence length -that is 10 - has to be set as a reference point, known as the **timestep**. Then, **padding** is applied to those contracts whose sequences are shorter than 10, with a procedure similar to the one presented in [14]. After applying padding, each original row, corresponding to a certain contract lifetime, is converted into a matrix whose number of rows is equal to ten and having a number of columns equal to the number of features. Even though in NLP problems it is often irrelevant applying padding before or after the actual sequence, in this case it has to be applied necessarily before, meaning that the first three samples of a contract- corresponding to its first three lifetimes- will be positioned in the last three rows of the matrix, leaving all zeros in the first seven rows. In summary, each sample -which originally is a vector- becomes a padded sequence represented by a matrix. That is the explanation for the three dimensions of the new data, which is not anymore a matrix but a set of matrices. In particular, the three dimensions of the new reshaped data refer respectively to the number of original samples, the timestep and the number of features.

The Python custom function used to preprocess data for *RNNs* is reported in the code section of the appendix A.4.

4.5.2 RNN Results

In this project, RNN have been exploited for the regression task, since they are usually exploited for continuous predictions. **Gated recurrent units (GRU)** and **Long Short Term Memory (LSTM)** techniques have been used. Again, after rescaling the target between 0 and 1, a sigmoid activation function has been applied in the output layer, consisting of one single neuron. The best overall *Test Mae* is 15.7% and *RMSE* 19.6%, while on the train respectively 13.5% and 18.2%, obtained with LSTM. Results with GRU are pretty much the same, just slightly lower, but computationally lighter. Best results per life using GRU are displayed in figure 4.15.

	mean		
	durata_nominale_gg	delta errore gg	delta errore %
life%			
0%	486.116973	101.469744	22.249504
10%	492.674431	90.920743	20.091408
20%	509.937425	82.496248	17.644945
30%	522.622295	76.944357	15.883272
40%	533.391462	66.962558	13.491566
50%	547.928110	62.087381	11.802777
60%	551.700649	60.640914	11.119237
70%	559.003268	61.084100	10.716500
80%	555.364407	56.283991	9.879524
90%	542.733491	39.829140	8.106766

Figure 4.15. Results GRU per each life %.

The error based on percentiles of nominal duration are instead reported in figure 4.16.

	mean		std	
	delta errore gg	delta errore %	delta errore gg	delta errore %
durata_binned_quartiles				
(29.999, 221.0]	24.089741	17.953474	21.026310	13.136403
(221.0, 394.0]	56.078358	16.794130	44.805641	13.135148
(394.0, 730.0]	83.593875	14.773706	74.961182	12.531700
(730.0, 2191.0]	140.281661	13.481416	118.975274	11.199560

Figure 4.16. LSTM error based on nominal duration percentiles.

The properties seem to be pretty much the same of the models used so far, in the sense that predictions are less accurate for short contracts than for those having longer nominal duration. Besides that, it is coherent that the higher the life percentage, the better the performance.

Given the high computations involved in running *RNNs* due to their high amount of weights to learn, parameter tuning has been minimal. Thus, it is omitted to report the best combination of parameters.

4.6 Multi-Task learning: Two outputs model

The last approach investigated through this framework consists of multi-outputs models. The purpose would be predicting the level of consumption and lifetime of each contract using one single model rather than two different ones. That is possible by putting two neurons rather than one in the output layer and building an appropriate loss function to minimize. The goal is that optimizing the latter benefits both the predictions. A direct way to construct a common loss function is through a weighted average of the two individual losses. Clearly, this makes sense only if the two targets are both continuous and consequently the loss function have the same nature. In other words, it would not make much sense to just average a binary cross-entropy objective function, used for binary targets, with a mean squared error, which is used for continuous predictions. Therefore, to follow this direction, the classification target has been converted into a continuous one, representing the final percentage used of contracts, in terms of their header amount. That being done, a multi-task learning NN is built using the keras functional API as in the figure below, giving same weights to the two loss functions to average.

```
def NN(nodi_hiddens= nodi_hiddens, dropout= dropout, lr= lr, n_layers= n_layers,
                loss_func= loss_func, metrics= metrics):

  numerical_input= Input(shape=(container_reg['X_train'].shape[1],))
  for i in range (n_layers):
    if i==0:
      drop= Dropout(dropout)(numerical_input)
      dense = Dense(nodi_hiddens[0], activation='relu')(drop)
    else:
      drop= Dropout(dropout)(dense)
      dense = Dense(nodi_hiddens[i], activation='relu')(drop)

  drop= Dropout(dropout)(dense)
  output_life = Dense(1, activation= 'sigmoid', name='output_life')(drop)
  output_usage= Dense(1, activation= 'sigmoid', name='output_usage')(drop)
  model = Model(inputs= [numerical_input], outputs=[output_life, output_usage])
  model.compile(optimizer='adam',
      loss={'output_life': 'mean_squared_error', 'output_usage': 'mean_squared_error'},
      loss_weights={'output_life': 1.0, 'output_usage': 1.0},  metrics= ['mae'])

  return model
```

Figure 4.17. Two outputs model construction.

A big benefit of using such model would be the double check of consumption and lifetime prediction, meaning that theoretically their output should be compatible. For instance, if the lifetime prediction is lower than 100%, that means the contract has been consumed more than 90%, reminding that is the threshold level above which a contract can be considered as fully used. Then, if the consumption prediction is instead lower than such threshold, the two outputs are somehow contradicting each other and the overall estimation can be considered less reliable. On the contrary, if they agree, the results could be interpreted as much more reliable than all the previous models.
In the appendix B.1 an example of double output neural network is presented.

4.6.1 Filtering Choices

As described in the first paragraphs of the thesis 1.3.2, regression models are applied on a subset of the classification samples, in the sense that it is relevant to predict only lifetime of fully consumed contracts. However, in the multi-output model,

the dataset must necessarily be the same. Keeping only the subsamples of fully consumed contracts would not make much sense, since the consumption target would be in that case always between 90% and 100% which is irrelevant from a business perspective. What is relevant would be eventually to know how much a contract is partially used.
A more reasonable option would be keeping all the samples, handling the ones not used with some opportune strategy. This might consists of setting the target to 100% to all those contracts which have not been consumed; the maximum lifetime would correspond to the whole lifetime duration in the agreement, that is the nominal duration. Doing this way, results may look good at a first sight, having:

- *Test MAE*: 9.8%, *Test RMSE*: 16,3%.
- *Train MAE*: 8.7%, *Train RMSE*: 15,5%.

Results seem to outperform all the previous model. However, it is extremely important to make them comparable, reminding that previously regression predictions had been made on ~77'000 samples related to contracts fully used, while now we are considering the whole set of ~304'000 samples. After filtering the test samples in order to make test scores compatible, the results become:

- *Test MAE (Comparable)*: 28.7%.
- *Test RMSE (Comparable)*: 36,5%.

Furthermore, the contradictions between the two outputs amount to more than 85% of the total predictions, as shown in figure 4.18.

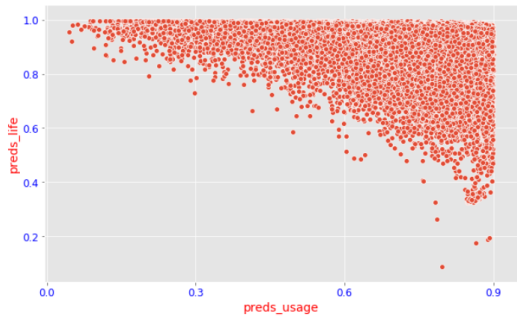


Figure 4.18. Contradictions between the two output predictions in multitask learning.

What seems to be reasonable, is that from that plot it emerges that the higher the prediction related to consumption, the lower the one related to lifetime. That makes absolutely sense, since at the basis of the whole process there is the fact that contracts which are not consumed more than 90%, expire after their 100% of life. Nevertheless, the image shows exactly the contradictions mentioned in the previous paragraph, corresponding to contracts partially used - whose life is supposed to be 100% - which are predicted to last less, as if they were fully consumed.

4.6.2 Undersampling

Two ways to improve such issue have been explored. The first one consists of projecting the target of partially used contracts basing on the rate at which they have been consumed during their nominal lifetime. For instance, if one has been used at 50% during its 100% life, it is assumed to finish after the double of its lifetime, which is 200%. Such projection has been made setting an upper bound to 1000% of maximum life, to avoid the target ranging within an extremely wide space.
The second change consists of filtering the dataset to make it more balanced in terms of consumed contracts and not, recalling that with the parametric rolling dataset classes distribution is roughly 70% and 30%. The process of taking off samples to make classes more balanced is known as undersampling and it has been applied in order to improve performance.

Skipping the results on the whole data, the following image shows the scores in the regression task, comparable with all the previous model results, meaning again that only the same test samples are considered.

	n_layers	nodi_hiddens	batch_size	dropout	lr	Train MAE	Test MAE	Train RMSE	Test RMSE
3	2	[32, 32]	128	0.5	0.01	83.012	79.034	147.546	147.981
1	1	[32, 32]	128	0.5	0.01	95.748	91.846	163.775	160.044
0	1	[16, 16]	128	0.5	0.01	90.598	106.613	156.874	171.833
2	2	[16, 16]	128	0.5	0.01	91.818	113.305	161.082	208.847

Figure 4.19. Multi-output results with undersampling and target projection.

The average percentage error on the test set is about 80%, which is clearly a very poor performance. The reason behind that is that data has been manipulated and forced to be something which is not very natural. For instance, the assumption earlier mentioned - related to the target projection - does not make much sense from a business perspective. In fact, if only half of the header amount has been exploited for one contract, when nominal duration has passed, time is over. There is no any other time for that to be consumed, thus it does not make any sense to project it beyond the maximum time it can last.
The purpose of this section has been to investigate the powerful tool of multi task learning with Keras, which would be very helpful in other problems, even though it seems there is no way it may fit to this particular process.

Chapter 5

Conclusions

In this project we have experimented several techniques to achieve the two required tasks, consisting of predicting consumption and lifetime of contracts, respectively in binary classification and regression (further details in chapter 1.2). A major part of the whole work is related to the construction of the dataset from the raw data extracted. Here, several modeling choices have been accomplished. First of all, it has been explored the realization of two main datasets aimed at making predictions based on the two achievements which is wanted to pursue, related to making predictions following both the *static* and *rolling* approaches. In particular, the former purpose is to apply the forecast in the moment a contract has been stipulated, while the latter to do that when it has already started to be valid - thus eventually consumed - for a certain period of time. In addition to that, two kinds of rolling approach have been investigated, referred to as *direct* and *parametric*. As regarding the latter, a different number of snapshots representing different moments of a contract lifetime may be chosen; among several attempts, 10 lifetimes percentages have been chosen to build such parametric dataset. Details regarding the two mentioned approaches and their specific realization have been discussed respectively in sections 1.3.1 and 2.2. Once specified the split into two different paths to follow both for classification and regression, important considerations have been made in the targets creation phase. Given the major importance of predicting the lifetime of business agreements and the need of performing that exclusively on consumed contracts, it has been chosen to apply in a first phase the classification model in order to filter out only those with label 1. Afterwards a regression prediction is carried out on such filtered contracts, as explained in section 1.3.2. To do that, it has been necessary to take a further decision concerning the creation of the target itself. In particular, as mentioned in section 1.3 and discussed more in detail in the paragraph 2.5, it has been chosen to deal with % of lifetimes rather than absolute ranges of time passed (i.e days, months...). Such choice provides also a standardization in the sense that this way the regression target does not strictly dependent on the stipulated duration. Given these first preliminary modeling choices, one of the most important phases of the project is related to the way categorical variables having high cardinality have been handled, in particular by exploiting a sophisticated target encoding procedure and categorical embeddings derived from neural networks. Relating to the former, a customized Python class has been developed

(Appendix A.1), where a big relevance has been given to the way noise has been punctually added according to specific criteria, in order to reduce overfitting. In section 2.7 a detailed explication of both the two methodologies has been reported. Relevant added value to the work is also in the fact that information concerning the evolution of framework agreements has been caught by the creation of several features built in the data aggregation phase. Each of the new features is consequence of a specific kind of signal which is desired to keep track about, as described in the paragraph 2.4.
After the decisions on theoretical approaches and technical operations related to the preprocessing step, the second part of the work has focused on the predictive models used to pursued the two tasks. Firstly, traditional supervised techniques have been exploited, as presented in chapter 3. The reasons behind the use of each of those algorithms and its parameter tuning approach have been exposed in section 3.1, with the related results presented in the following paragraphs. At the end of that chapter it has been reported the comparison between the direct and parametric methodologies, highlighting the essential convenience in using the second one. In a similar way, in chapter 4 the exploitation of *Neural Networks* has been explored, first giving the general insights of the optimization methodology pursued for building each *NN* (see section 4.2), followed by the results given each set of configuration, approach - that is static and parametric - and task. Particular attention has been given to the analysis of *Recurrent Neural Networks* used for the purpose of keeping track of the step by step evolution of each contract.
Finally, a multitask learning *NN* configuration, consisting of a double output, has been investigated in section 4.6 in the attempt of achieving the two tasks with a single model, even though results have shown that it seems not to be affordable given this data.

5.1 Most performing models

Before drawing any conclusion about the best models found, it is fundamental to specify that depending on the specific environment and business needs, there are various ways to assess a model quality. The most basic as well as relevant division could be expressed in terms of **effectiveness** and **efficiency**. Basing on the first criterion, it is considered the best model the one performing best in terms of the chosen mathematical scores, for instance those having highest *F1* or lowest *MAPE*. On the other hand, efficiency is always referred to the resources and time spent for achieving a specific result, which are often given as external boundaries and constraints. The optimal solution is the one combining the two aspects, thus attempting to achieve the best **trade-off** between efficacy and effectiveness.

That being said, a summary of the best models obtained through this project is going to be reported, both in terms of most effective algorithms and the best trade-off considering also efficiency. In the two tables below that is shown for the static approach, where the number of features is reported as the unique term of efficiency, even though other aspects such as execution time and general model complexity have been considered in this matter.

As we can notice, for classification, at the cost of 1% of *weighted F1*, we have

Classification

Static	Model Type	F1 Test	Feats
Effectiveness	NN (Embedding)	75.2%	17/17
Efficiency	RF (Feat. Sel.)	74.2%	9/14

Regression

Static	Model Type	MAPE Test	Feats
Effectiveness	SVM (Rbf)	22.4%	14/14
Efficiency	RF (Feat. Sel.)	22.6%	7/14

Table 5.1. Final Trade-off Static.

obtained a more parsimonious model, using 9 features rather than 17. Furthermore, tree based methods opportunely tuned (e.g setting maximum depth of each decision tree) are much faster than neural networks. As concerning regression, a Random Forest performed after selecting 7 out of 14 features is much more preferable than a SVM with gaussian kernel, at the cost of increasing the MAPE of 0.2%.

In a similar way, the summary for the rolling approach is reported in the following tables, recalling that the best score in terms of mean percentage error is a weighted average - with respect to the number of samples - of the individual scores for each lifetime percentage. In this case, different best trade-off may be chosen and, since the choice is subjective basing on what it is wanted to maximize, or - better said - until which extent efficiency is preferred to effectiveness, it is highlighted in grey the best solution proposed.

Classification

Rolling	Model Type	F1 Test	Feats
Effectiveness	Soft Voting	77.6%	/
Efficiency (1)	NN(Embedding)	77.5%	27/27
Efficiency (2)	Boosting	76.6%	27/27

Regression

Rolling	Model Type	MAPE Test	Feats
Effectiveness	Voting Regressor	14.7%	/
Efficiency (1)	RF	14.8%	27/27
Efficiency (2)	RF (Feat. Sel.)	15.0%	7/27

Table 5.2. Final Trade-off Rolling.

For classification, the most performing result is given by the very heavy ensemble method consisting of averaging the results of different models; accepting a 0.1% of lower test *MAPE*, we might choose a *NN* exploiting embedding layers to handle categorical features. For reasons mentioned at the beginning of this paragraph, it is proposed to use a much faster algorithm such as *Gradient Boosting*, at the cost of 1% of *MAPE* in respect to the most effective benchmark. With a similar reasoning, in regression it is descarded the solution given by the voting regressor, suggesting as a primary choice a random forest algorithm. In particular, the one using 7 features - selected basing on the mean decrease of *MSE* - might be much more advisable rather than the one using the whole set of 27 features, allowing just a 0.2% of lower *MAPE*.

Given the best results obtained and the overall picture of the project, we can draw the conclusion that tree based methods such as Random Forest and Boosting are the most advisable tools given this problem. Both traditional and recurrent Neural networks have not shown any relevant advantage which is worthy to keep at the expense of their high computational cost. It must be also pointed out that a parametric rolling approach has been preferred to many individual models built ad hoc for each contract evolution phase, accepting to have a small decrease in performances which, as seen in the previous chapters (in particular in section 3.6), can be considered almost irrelevant.

5.2 Further studies

This work represented a first important step into the estimation of contracts lifetime and consumption. Although the results may be considered promising, the predictions might be improved in different ways. If the Business goals should be such that even a small increase in the test scores is preferred at the cost of sacrificing more time and resources, a first suggestion could be to keep going on with parameter tuning, especially in neural networks where, given their computational weight, it has not been deepened much. In particular, besides exploring a wider range of hyperparameters such as the number of layers or neurons, further investigation may be conducted in relation to other parameters such as the weight decay of the learning rate, its momentum, or even regularization terms based on *l1* and *l2* penalty aimed at decreasing the power of high weights. As regarding this matter, a bayesian approach for finding parameters might represent an important potential improvement, as shown in [15], [25] and [21].

In general, what could be improved with the current Business needs consists of changes in the categorical variables handling. First, it has to be pointed out that within this framework the noise proportion and the threshold in the custom K-fold target encoding have not been treated as hyperparameters, but they have been fixed; tuning them would be clearly an expensive effort, but it could slightly improve the test results. Other efficient approaches such as binary or hash encoding [23] might lead to some improvements, at the expense of decreasing the problem interpretability, since new 'abstract' features would be created from the original ones. A deeper analysis would be also obtained trying different feature selection methods, such as regression based step-wise selection, either forward or backward. The latter consists of a *greedy* approach where starting from the original set of features, it is removed at each step the one less relevant in terms of lowest *p-value*, until all the remaining features are significant according to a chosen confidence level. Another approach which could bring to meaningful results, consists of keeping track of the past evolution of each contract through new lagged features, referred to the previous lifetimes values. Once they are eventually built, the new wider input variables may be given to any traditional machine learning algorithm. Instead of that, recurrent neural network have been preferred within this framework, since their nature allows to avoid building so many lagged features which, given the specific case of this problem, would result in a very high dimension. A further suggestion might be using survival models to estimate the remaining lifetime. Additionally, the creation of more lifetime snapshots, which would result in a higher number of samples a model would be trained on, could bring to better performances. Anyway, even if the scores were roughly the same obtained with the 10 lifetimes used, such model would allow a better approximation for any lifetime percentage.

Last but not least, regardless how the work might be extended in its complexity and strengthened in its robustness, further improvements could derive by augmenting the raw data extracted, in terms of new columns. These could eventually bring new information which, after being opportunely translated through new variables in the feature engineering process, could lead to much better results much more easily than any parameter tuning or any different algorithm would do without having such new features.

Appendix A

Codes

A.1 Custom K-Fold Target Encoding Class

```
import pandas as pd
import numpy as np

from sklearn  import base
from sklearn.model_selection import train_test_split, KFold



class KFoldTargetEncoderTrain_std(base.BaseEstimator, base.TransformerMixin):

    def __init__(self, colnames,targetName,n_fold=10,verbosity=True,discardOriginal_col=False, threshold= 1, showPrint= False):

        self.colnames = colnames
        self.targetName = targetName
        self.n_fold = n_fold
        self.verbosity = verbosity
        self.discardOriginal_col = discardOriginal_col
        self.threshold= threshold
        self.showPrint= showPrint

    def fit(self, X, y=None):
        return self

    def transform(self,X):

        assert(type(self.targetName) == str)
        assert(type(self.colnames) == str)
        assert(self.colnames in X.columns)
        assert(self.targetName in X.columns)
        assert(type(self.threshold) == int)

        kf = KFold(n_splits = self.n_fold, shuffle = False, random_state=2019)


        col_mean_name = self.colnames + '_' + 'enc_mean'
```

```
        col_std_name= self.colnames + '_' + 'enc_std'


        X[col_mean_name] = np.nan
        X[col_std_name] = np.nan


        for tr_ind, val_ind in kf.split(X):
            if self.showPrint is True:
                print('Indici delle Train-Folds: {},\033[1mTest Fold da fillare: {}\033[0m \n'.format(tr_ind, val_ind))

            X_tr, X_val = X.iloc[tr_ind], X.iloc[val_ind]

            diz_mean= X_tr.groupby(self.colnames)[self.targetName].mean().to_dict()
            diz_std= X_tr.groupby(self.colnames)[self.targetName].std().to_dict()


            frequenze= X_tr.groupby(self.colnames).size()
            cat_oltre_threshold= frequenze[frequenze >= self.threshold].index

            if self.showPrint == True:
                print('cat_oltre_threshold', cat_oltre_threshold)

            X.loc[X.index[val_ind], col_mean_name] = X_val[self.colnames].apply(lambda x: diz_mean[x] if x in cat_oltre_threshold else np.nan )
            X.loc[X.index[val_ind], col_std_name] = X_val[self.colnames].apply(lambda x: diz_std[x] if x in cat_oltre_threshold else np.nan)

            mean_of_target = X_tr[self.targetName].mean()
            std_of_target= X_tr[self.targetName].std()

            X[col_mean_name].fillna(mean_of_target, inplace = True)
            X[col_std_name].fillna(std_of_target, inplace = True)

        if self.verbosity:

            encoded_feature = X[col_mean_name].values
            print('Correlation between the new feature, {} and, {} is {}.'.format(col_mean_name,
                                                self.targetName,np.corrcoef(X[self.targetName].values, encoded_feature)[0][1]))

        if self.discardOriginal_col:
            if self.showPrint == True:
                print('columns: ', X.columns)
            X = X.drop(self.colnames, axis=1)
        return X


class KFoldTargetEncoderTest_std(base.BaseEstimator, base.TransformerMixin):
```

```python

    def __init__(self,train,colNames,Train_encoded_mean,Train_encoded_std, showPrint= False):

        self.train = train
        self.colNames = colNames
        self.Train_encoded_mean = Train_encoded_mean
        self.Train_encoded_std = Train_encoded_std
        self.showPrint= showPrint

    def fit(self, X, y=None):
        return self

    def transform(self,X):


        category_mean_tr = self.train[[self.colNames,self.Train_encoded_mean]].groupby(self.colNames).mean()
        category_std_tr= self.train[[self.colNames,self.Train_encoded_std]].groupby(self.colNames).mean()


        population_mean_tr= self.train[self.Train_encoded_mean].mean()
        population_std_tr= self.train[self.Train_encoded_std].mean()


        X[self.Train_encoded_mean]= X[self.colNames].map(category_mean_tr.squeeze().to_dict())
        X[self.Train_encoded_std]= X[self.colNames].map(category_std_tr.squeeze().to_dict())


        if self.showPrint== True:
            print('Filled {} Missing Values with Average mean and std equals to {}, {}'.format(
                                                                np.sum(np.sum(X.isna())), population_mean_tr , population_std_tr))

        X[self.Train_encoded_mean].fillna(population_mean_tr, inplace = True)
        X[self.Train_encoded_std].fillna(population_std_tr, inplace = True)

        return X
```

A.2 Feature Engineering and Lifetime cuts creation

```python
def aggregated_operations(x):

    d = {}
    d['header_utilizzato'] = x['Header amount_ordine'].sum()
    d['num_orders'] = x['Header amount_ordine'].count()
    d['avg_importo_ordine']= x['Header amount_ordine'].mean()
    d['std_importo_ordine']= x['Header amount_ordine'].std()
    d['last_order_date']= max(x['Document date_ordine'])
    d['first_order_date']= min(x['Document date_ordine'])
    d['avg_order_intervals']= round(np.mean(x['diff_intervals']), 1)
```

```
    d['std_order_intervals']= round(np.std(x['diff_intervals']), 1)

    d['order_trimesters']= x['Document date_ordine'].apply(lambda x: x.quarter).value_counts().to_dict()

    return pd.Series(d, index= agg_column_names)



def construct_lifes(df_not_aggregated, life_cols= life_columns):
    diz_df_lifes= {}
    diz_df_shapes= {}
    for life_col in life_cols:
        print('Processing', life_col)

        df_life= df_not_aggregated[df_not_aggregated[life_col] >= df_not_aggregated['Document date_ordine']]
        df_life_grouped= df_life.groupby(['Purchase document']).apply(aggregated_operations)
        df_life_grouped.reset_index(inplace=True, drop= False)

        suffix= life_col.split('life')[0]
        df_life_grouped.columns= ['Purchase document']+ [col + '_' + suffix for col in agg_column_names]



        lasciati_fuori= set(df_not_aggregated['Purchase document']) - set(df_life_grouped['Purchase document'])
        temp= pd.DataFrame(np.array(('Nessun_ordine '*len(lasciati_fuori)*(df_life_grouped.shape[1]-1)).split()).reshape(len(lasciati_fuori), (df_life_grouped.shape[1]-1)))
        temp.insert(0, 'Purchase document', lasciati_fuori)

        temp.columns= df_life_grouped.columns
        df_life_final= pd.concat([df_life_grouped, temp])

        diz_df_lifes[life_col]= df_life_final
        diz_df_shapes[life_col]= diz_df_lifes[life_col].shape

    return [diz_df_lifes, diz_df_shapes]

all_lifes_not_agg= construct_lifes(join_date_fasce_final, life_cols= life_columns)
```

A.3 Embedding Preprocessing

```
def embedding_preparation(train, test, max_emb_size= 50, emb_method= 'half', n_layers= 1, nodi_hidden1= 64, nodi_hidden2= 32, nodi_hidden3= 16,
                          with_dropout=True , dropout= 0.5, fraction= 0.33, concat_cont= False):

  Train= train.copy()
  Test= test.copy()

  categorical_vars= Train.select_dtypes(include= object).columns
```

A.3 Embedding Preprocessing

```
continuous_cols = [col for col in Train.columns if col not in categorical_vars ]


label_enc_custom= Customized_Label_Encoder()
Train= label_enc_custom.fit_transform_pd(Train, cols= categorical_vars)
Test= label_enc_custom.transform_pd(Test, cols= categorical_vars)


for col in continuous_cols:

  scaler= RobustScaler(with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0))
  Train[col]= scaler.fit_transform(pd.DataFrame(Train[col]))
  Test[col]= scaler.transform(pd.DataFrame(Test[col]))

X_train_categorical= Train[categorical_vars]
X_train_continuous= Train[continuous_cols]

X_test_categorical= Test[categorical_vars]
X_test_continuous= Test[continuous_cols]

if (X_train_categorical.shape[0] == X_train_continuous.shape[0]== y_train.shape[0])== False:
  print('Shape mistaken')
  raise ValueError

inputs_emb= []
true_inputs= []

final_X_train= []
final_X_test= []

continuous_input = Input(shape=(X_train_continuous.shape[1], ))
if concat_cont is True:
  inputs_emb.append(continuous_input)
  true_inputs.append(continuous_input)
  final_X_train.append(Train[continuous_cols])
  final_X_test.append(Test[continuous_cols])

for categorical_var in categorical_vars :
  current_input= Input(shape=(1,), dtype='int32')
  final_X_train.append(Train[categorical_var])
  final_X_test.append(Test[categorical_var])

  true_inputs.append(current_input)
  no_of_unique_cat  = X_train_categorical[categorical_var].nunique()



  if emb_method== 'half':
    embedding_output_size = int(min(np.ceil((no_of_unique_cat)/2), max_emb_size ))

  elif emb_method== 'sqrt':
    embedding_output_size = int(min(np.ceil(np.sqrt(no_of_unique_cat)
```

```
    ), max_emb_size ))

    elif emb_method== 'fraction':
      embedding_output_size = int(min(np.ceil(fraction*
    no_of_unique_cat), max_emb_size ))

    elif emb_method== 'paper_rule':
      embedding_output_size = int(min(np.ceil(1.6* no_of_unique_cat**
    0.56), 600 ))


    print('Categorical variable {} has got {} classes, Embedded in -->
    {}'.format(categorical_var, no_of_unique_cat ,
    embedding_output_size))

    cat_embedding= Embedding(output_dim= embedding_output_size,
    input_dim= no_of_unique_cat , input_length=1)(current_input)
    cat_embedding= keras.layers.Reshape((embedding_output_size,)) (
    cat_embedding)

    inputs_emb.append(cat_embedding)

  all_inputs = keras.layers.concatenate(inputs_emb)
  concat_inputs = keras.layers.concatenate(inputs_emb)




  diz= { 'final_X_train': final_X_train, 'final_X_test':final_X_test,
          'true_inputs': true_inputs,  'concat_inputs': concat_inputs,
     'continuous_input':continuous_input,
           'X_sets': {'X_train_cont': X_train_continuous, 'X_test_cont
    ': X_test_continuous, 'X_train_cat': X_train_categorical, '
    X_test_cat': X_test_categorical} }

  return    diz
```

A.4 Recurrent Neural Networks Preprocessing

```
def preprocess_X_rnn(X, max_len= max_len, show_print= False):

  unique_contr= pd.Series(X.index).unique()
  contr_counts= X.reset_index()['Purchase document'].value_counts()
  contr_counts

  sequenze= []
  for contr in unique_contr:
    n_snapshots= contr_counts.loc[contr]
    sequenze.append(n_snapshots)

  sequences_order= sequenze
  cumsum= np.array(sequences_order).cumsum()

  df= X.copy()

  max_len= max_len
  train_sequences= []
```

```python

for i in range(len(sequences_order)):
  if i==0:
    seq_len= sequences_order[i]
    seq= df.iloc[:seq_len]
    train_sequences.append(seq)

  elif i>0:
    previous_seq_len= cumsum[i-1]
    seq_len= sequences_order[i]

    seq= df.iloc[previous_seq_len: cumsum[i]]
    train_sequences.append(seq)

seqs_not_padded= []

for col in df.columns:
  if show_print:
    print('...performing sequences for', col, '...')
  seq_col= []
  iteraz= 0
  for blocco in train_sequences:

    for i in range(len(blocco)):
      if i==0:
        try:
          current_list= [blocco[col].iloc[i]]

          seq_col.append(current_list)

        except:
          print('Non riuscito')
      elif i>0:


        previous_list= seq_col[iteraz+i-1]

        current_number= blocco[col].iloc[i]

        current_list= previous_list.copy()
        current_list.append(current_number)

        seq_col.append(current_list)

  iteraz+= len(blocco)
  seqs_not_padded.append(seq_col)

padded_seqs= seqs_not_padded.copy()

for i in range(len(seqs_not_padded)):
  padded_seqs[i]= pad_sequences(seqs_not_padded[i], maxlen = max_len, padding = 'pre',  dtype= 'float64')

final_padded= []
for i in range(len(X)):

  lista_9_25= [[padded_seqs[col_index][i][0] for col_index in range
```

```
    (0, num_feats)]]

    for j in range(1, max_len):
      lista_9_25.append([padded_seqs[col_index][i][j] for col_index in range(0,num_feats)])

    final_padded.append(np.array(lista_9_25))

  final_padded= np.array(final_padded)
  print(final_padded.shape)

  return final_padded
```

A.5 Run Grid Search with scores summary

```
def summary_classifier(classifier, x_train, y_train, y_test, predictions):


    print('\n\033[1mtrain accuracy TP+TN/tot is {} \033[0m'.format(round(accuracy_score(y_train, classifier.predict(x_train)),3)))
    print('\033[1mtest accuracy TP+TN/tot is {} \033[0m'.format(round(accuracy_score(y_test, predictions),3)))

    print('\n\033[1mCLASSIFICATION REPORT\033[0m: \n', classification_report(y_test, predictions))
    print('\033[1mCONFUSION MATRIX\033[0m \n', confusion_matrix(y_test, predictions))

    print('\n\033[1mROC (Area under curve) Train is {a}, for ROC test is {b}\033[0m'.format(a= round(roc_auc_score(y_train, classifier.predict(x_train), average="weighted"),2), b= round(roc_auc_score(y_test, predictions, average="weighted"),2)))
    print('\n\033[1mF1 train is {a}, F1 test is {b} \033[0m'.format(a= round(f1_score(y_train, classifier.predict(x_train),
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
    average='weighted'),3), b= round(f1_score(y_test, predictions, average='weighted'),3)))


def summary_grid(grid_fit, scaled_train, Y_train, scaled_test, Y_test):

    print('best param combination: ', grid_fit.best_params_)




    print('\n\033[1m Train Accuracy is {}\033[0m'.format(round(accuracy_score(Y_train, grid_fit.predict(scaled_train)),3)))
    print('\033[1m Test Accuracy is {}\033[0m'.format(round(accuracy_score(Y_test, grid_fit.predict(scaled_test)),3)), '\033[0m')

    print('\n\033[1mCLASSIFICATION REPORT\033[0m: \n', classification_report(Y_test, grid_fit.best_estimator_.predict(scaled_test)))
    print('\033[1mCONFUSION MATRIX\033[0m \n', confusion_matrix(Y_test,
```

```
    grid_fit.best_estimator_.predict(scaled_test)))

    print('\n\033[1mROC train is {a}, ROC test is {b} \033[0m'.format(a= roc_auc_score(Y_train, grid_fit.predict(scaled_train)), b= roc_auc_score(Y_test, grid_fit.predict(scaled_test))))
    print('\n\033[1mF1 TRAIN: {a}, \nVALIDATION F1 (averaged on 5 folds): {b} \033[0m \n\033[1mF1 TEST: {c}\033[0m'.format(
        a= round(f1_score(Y_train, grid_fit.predict(scaled_train), average='weighted' ),3),
        b= round(grid_fit.best_score_,3),
        c= round(f1_score(Y_test, grid_fit.predict(scaled_test), average='weighted'),3)))



def run_model(estimator, task='class', param_grid= None, only_imp= False, random= False, verbose=5, cv=3, n_jobs= None, comparison= False):

    diz= cont_modello(task)

    X_train= diz['X_train']
    X_test= diz['X_test']
    ytrain= diz['ytrain']
    ytest= diz['ytest']

    if comparison== True:
        ytest= ytest_comparison_stat
        X_test= X_test_comparison_stat

    if only_imp== True:
        try:
            X_train= X_train_imp
            X_test= X_test_imp
        except:
            pass

    print('Shape di X_train e X_test sono: ', X_train.shape, X_test.shape)
    temp=time.time()

    "The balanced mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))"


    if param_grid is None:
        estimator.fit(X_train, ytrain, )
        preds= estimator.predict(X_test)
        print('execution time (min)=', round((time.time()-temp)/60, 1), ' finished at ', datetime.today())
        summary_classifier(estimator, X_train, ytrain, ytest, preds)
        return estimator

    else:
        if random== False:
            Grid_RF = GridSearchCV(estimator, param_grid, scoring='
```

```
f1_weighted', cv=cv, return_train_score=True,
                                n_jobs= n_jobs, verbose= verbose)
            Grid_RF.fit(X_train, ytrain)
            print('\n It is not a single model but a Grid Search \n')
            summary_grid(Grid_RF, X_train, ytrain, X_test, ytest)
            print('execution time (min)=', round((time.time()-temp)/60, 1), ' finished at ', datetime.today())

        elif random== True:
            Grid_RF = RandomizedSearchCV(estimator, param_distributions=param_grid,
                                         scoring='f1_weighted', cv=cv, return_train_score=True, verbose= verbose, n_jobs= n_jobs)
            Grid_RF.fit(X_train, ytrain)
            print('\n It is not a single model but a (Randomized) Grid Search \n')
            summary_grid(Grid_RF, X_train, ytrain, X_test, ytest)
            print('execution time (min)=', round((time.time()-temp)/60, 1), ' finished at ', datetime.today())
        return  Grid_RF
```

Appendix B

Figures

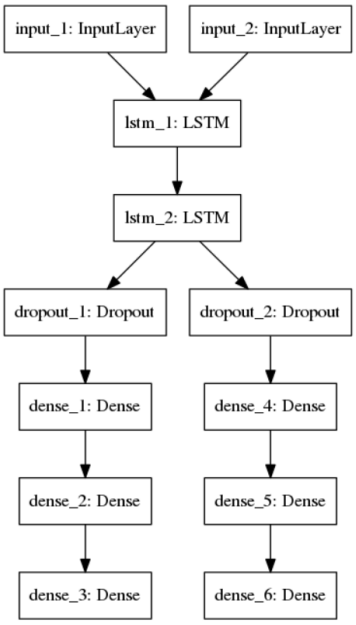


Figure B.1. Multi-Task Learning with Neural Networks.

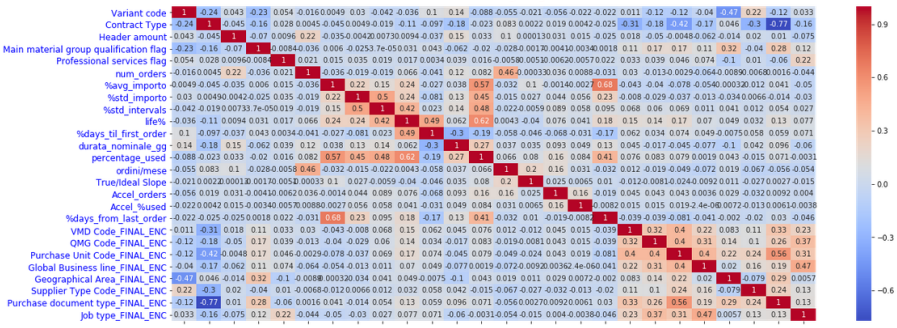


Figure B.2. Correlation Encoded variables Static Data.

Purchase document	Variant code	Contract Type	Header amount	Main material group qualification flag	Professional services flag	num_orders	%avg_importo	%std_importo	%avg_intervals	%std_intervals
EL\5800012947	-1.0	0.0	7.966689	0.0	0.0	-0.2	-0.054418	0.000000	-0.389517	0.000000
EL\5800012947	-1.0	0.0	7.966689	0.0	0.0	0.4	0.045404	0.015207	-0.250978	1.523618

Purchase document	life%	%days_til_first_order	durata_nominale_gg	percentage_used	ordini/mese	True/Ideal Slope	orders_trim_1	orders_trim_2
EL\5800012947	-0.75	-0.523732	1.419028	-0.151816	-0.109756	-0.340715	0.0	0.0
EL\5800012947	-0.50	-0.103753	1.419028	-0.101460	0.902439	-0.087173	0.0	0.0

Purchase document	orders_trim_3	orders_trim_4	initial_trimester	Accel_orders	Accel_%used	%days_from_last_order	VMD Code_FINAL_ENC
EL\5800012947	0.0	0.0	0.5	0.0	0.000000	-0.065815	0.048648
EL\5800012947	0.0	0.0	0.5	1.0	1.035197	1.268001	0.292853

Purchase document	QMG Code_FINAL_ENC	Purchase Unit Code_FINAL_ENC	Global Business line_FINAL_ENC	Geographical Area_FINAL_ENC	Supplier Type Code_FINAL_ENC	Purchase document type_FINAL_ENC	Job type_FINAL_ENC
EL\5800012947	0.041980	0.554572	0.232029	-0.468702	-0.425284	0.737483	-1.042676
EL\5800012947	0.047627	0.636475	0.893161	0.352800	-0.725370	0.691957	-1.051192

Figure B.3. Example of two rows of scaled Rolling Dataset, having 32 predictors.

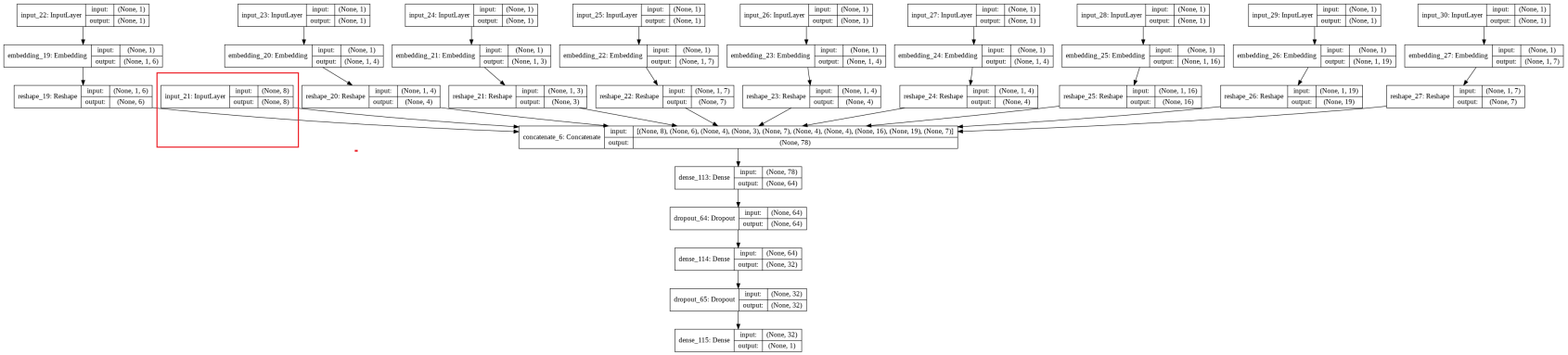


Figure B.4. Schema of a ANN with concatenation of embedding layers and numerical inputs (red selection).

Bibliography

[1] Embedding size rule used in fastai open source python library. https://github.com/fastai/fastai/blob/master/fastai/tabular/data.pyL14.

[2] Robustscaler python class, scikit learn https://scikit-learn.org.

[3] Understanding lstm networks, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[4] J. B. Ahire. Lasso, ridge and elastic net regularization, 2018.

[5] Y. Bengio, N. Srivastava, and G. Hinton. Dropout a simple way to prevent neural networks from overfitting. 2014.

[6] A. Dubey. Feature selection using random forest, 2018.

[7] T. H. R. T. Gareth james, Daniela Witten. *An introduction to Statistical Learning.* New York : Springer. c2001.

[8] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *ScienceDirect, Machine Learning*, 63(1):3–42, 2006.

[9] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[10] C. Guo and F. Berkhahn. Entity embeddings of categorical variables. *Neokami Inc.*, 2016.

[11] J. Huang and M. Ng. Automated variable weighting in k-means type clustering. *IEEE*, 2005.

[12] D. Jimenez. Dynamically weighted ensemble neural networks for classification. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, volume 1, pages 753–756 vol.1, May 1998.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[14] N. Kumar. Recurrent neural networks (rnn) explained — the eli5 way, 2019.

[15] D. D. Labs. Parameter tuning with hyperopt, 2017.

[16] P. Leray and P. Gallinari. Feature selection with neural networks. *Behaviormetrika*, 1999.

[17] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *ScienceDirect, Machine Learning*, 6.

[18] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, June 1991.

[19] Pourya. K-fold target encoding, 2019.

[20] R. Ravi. One hot encoding is making your tree-based ensembles worse, 2019.

[21] R. L. V. L. Reyes Pavón, Fernando Díaz. Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *ScienceDirect*, 36, 2009.

[22] A. Rojarath, W. Songpan, and C. Pong-inwong. Improved ensemble learning for classification techniques based on majority voting. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 107–110, Aug 2016.

[23] B. Roy. All about categorical variable encoding, 2017.

[24] W. S. Sarle. Neural networks and statistical models, 1994.

[25] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[26] R. Somasundaram and R. Nedunchezhian. Evaluation of three simple imputation methods for enhancing preprocessing of data with missing values. *International Journal of Computer Applications*, 21, 2011.

[27] Y. Zhang, H. Zhang, J. Cai, and B. Yang. A weighted voting classifier based on differential evolution. *Abstract and Applied Analysis*, 2014.