Department of Information Engineering and Computer Science

Bachelor's Degree in Computer Science

FINAL DISSERTATION

Unsupervised learning for customer segmentation and its interpretability

Supervisor

Prof. Mauro Brunato

Student

Steve Azzolin

Academic year 2019/2020

Contents

Abstract

We are living in a period of great enthusiasm regarding Artificial Intelligence. Many companies have started to invest in these new technologies in order to be the first to introduce new products and new services in their market.

The insurance sector is undergoing a profound transformation, which reflects intense changes in the macroeconomic environment and technological progress. Thanks to the development of new technologies and the spread of such technologies to the mass, the insurance market is looking for new business opportunities, such as online insurances which are gaining more and more importance. One of the main differences between online insurances and traditional insurances is the lack of an intermediate agent that can lead to the potential loss of revenue. A proper segmentation of customers can, instead, lead to automated targeted offers, increase the probability of sale and encourage automated marketing initiatives.

Goal

The goals of the project were to develop a configurable pre-processing data pipeline that can be also reusable for many Machine Learning applications, to cluster insurance agencies' customers with Machine Learning techniques, and to identify the characteristics of such clusters. Once a significant customer segmentation, from the insurance point of view, has been identified, it will be possible to associate each group with a profile and a series of policies to be proposed.

Personal contribution

Considering that this project represents the base for the development of new innovative services to be proposed to potential customers, it was not possible to obtain a real dataset.

Data quality is fundamental for the success of a Machine Learning project, but the goal of this project was not to obtain an optimal clustering, but rather to develop tools and provide a background for future projects accompanied by real datasets. For these reasons this work does not focus on the results achieved by clustering, but on the process that led to these results trying to maintain a didactic approach.

Particular attention was reserved to the interpretability of clustering results, a characteristic that often lacks in unsupervised projects, through the development of a residual impurity score. The development of such a index was followed by an optimization step to find the best couple dimensionality reduction algorithm - clustering algorithm, with their relative hyperparameters. At the same time the optimization step was driven by the goal of verifying the presence of features that can describe, from an insurance point of view, each cluster. The most important features for each cluster were evaluated by the residual impurity score after the optimization process.

The development of the pipeline was followed by the deployment of such pipeline on the cloud, and by the development of a web application in order to test the model on new samples.

1 Introduction

1.1 Context and motivation

During the past decades, the interest towards Artificial Intelligence reached many peaks but also encountered many pitfalls, such as the *AI winter*[1]. A new AI wave started during the 2010s when training deep architectures became feasible. New training algorithms, super-computers, GPUs and TPUs contributed to the great success of Deep Learning. Deep Learning is a subfield of Machine Learning (ML) which in turn is a subfield of Artificial Intelligence. Machine Learning has the aim of learning from data and improving from experience without being explicitly programmed.

The strength of Machine Learning, which is learning from data, is also its weakness. Data is essential in order to resolve problems with Machine Learning. The limitation of labeled data is currently one of the main problems with Machine Learning. This seems in contrast with the trend of Big Data, but the truth is that Big Data is often unstructured and unlabelled.

One of the current challenges is to find ways to use unlabelled data along with labelled data to train algorithms. This would unlock a great amount of unused data that could bring to a new AI revolution. For example, the sub-task of Machine Learning called *semi-supervised learning* tries to combine the power of *supervised learning* with the great amount of data that *unsupervised learning* could use. For instance, it is possible to pre-train architectures with unsupervised learning and fine-tune parameters with supervised techniques.

Supervised learning aims at finding a function that maps an input to an output. The mapping function is inferred by data examples. Unsupervised learning aims at finding hidden patterns in the data. Often algorithms are used as black-boxes, so it is very difficult to understand why the algorithm produced a certain result.

According to the ANIA (National Association of Insurance Companies) 2018-2019 report [1], 218 insurance companies are active on the Italian territory (97 national, 3 with Extra-EU registered office, 118 with registered office in an EU country). According to the statistical bulletin IVASS (Institute for Insurance Supervision) 2018 [5], a total of 243393 insurance intermediaries (agents, brokers, etc.) are authorized to operate in Italy. As the IVASS 2018 Activity Report [6] testifies, the insurance sector is undergoing a profound transformation, which reflects intense changes in the macroeconomic environment and technological progress. In fact, the insurance sector has started to adopt technologies such as Cloud, Blockchain and Artificial Intelligence.

Blue Reply[2] is a computer consultancy company that collaborates with the main insurance companies in Italy. As part of the presented general propensity for new technologies, it started a Machine Learning Lab to figure out how to develop new services based on Machine Learning. My goal was to take the first step for the adoption of Machine Learning within the market in which the company operates. This was accomplished with the development of a data pre-processing configurable pipeline, the implementation of a baseline for clustering, and the deployment on the cloud of such baseline in order to cluster new samples coming from a web GUI. The ML Lab is still in the research and development phase, so the lack of clients implies the lack of a real dataset. This increased the need to develop a general and configurable baseline for clustering, in order to address many kinds of Machine Learning problems. For this reason, in this work, I will highlight steps and tools that brought me to the final result, rather than the final result itself.

Particular attention was reserved for the interpretation of the results of the clustering algorithm since interpretability plays an important role in this work. Considering that this project was an experimental

[1] https://en.wikipedia.org/wiki/AI_winter
[2] https://www.reply.com/en/

approach to the broad world of Machine Learning, the goal was to identify groups of customers to whom assign an insurance profile and policies, keeping, as stated above, a didactic and general approach. The link between clusters, profiles, and policies was thought to be still a manual action to be taken at a later stage. Thus, the creation of this link is not treated here. However, in order to make this link is necessary to highlight the characteristics, in terms of the business domain, of each cluster to allow a correct manual association. For this reason, part of my work was focused on the development of a metric to describe analytically a cluster, and to assign it a meaning from the insurance point of view. For instance, examples of meaningful clusters for the insurance market are:

- Customers with similar ages
- Customers that, despite different ages, share the same purchase power
- Customers that, despite different ages and different purchase power, share the same family conditions (number of sons, number of dependent elderly family members, etc..)

From these examples is possible to grasp the common characteristic behind these clusters, that is, they all have one or more features whose range of values within the cluster is lower than the range of values of the overall population.

The insurance market covers the 7.7% of the Italian GDP (Gross Domestic Product)[5]. Thanks to the development of new technologies and the spread of such technologies to the mass, the insurance market is changing and looking for new business opportunities. For example, online insurances are gaining more and more importance [12][3]. One of the main differences between online insurances and traditional insurances is the lack of an intermediate agent. This has some pros and some cons. For example, without an insurance expert the customer could be disoriented. On the other hand, online insurances are usually cheaper than traditional insurances. In fact, the lower cost is the first reason for which Italian customers look for online vehicles insurances [12]. From the insurance company point of view, the lack of interaction between the insurance agent and the customer could lead to the potential loss of revenue, that is when the insurance agent adopts marketing strategies such as *cross-selling*[3] or *up-selling*[4]. Thus, the development of new services is essential to exploit these new technologies in order to create new business opportunities. Proper segmentation of customers can lead to automated targeted offers, increase the probability of sale and encourage automated marketing initiatives such as cross-selling and up-selling.

1.2 The dissertation

After this introduction, the dissertation is organized as follows:

- *Chapter 2 - Background*: Introduces clustering and dimensionality reduction algorithms.
- *Chapter 3 - System description*: Overall description of the main components of this work.
- *Chapter 4 - Data generation*: This chapter describes the process of data generation, with its constraints.
- *Chapter 5 - The pipeline*: Presents the description of the pipeline, focusing on each step and their relative characteristics.
- *Chapter 6 - Optimization and interpretability*: This chapter describes the optimization process and the residual impurity score.
- *Chapter 7 - Deployment of the pipeline and the web app*: Offers an overview of the deployment of a Machine Learning model to the cloud and describes the web application.
- *Chapter 8 - Conclusion and future works*: Draws the conclusion of this work and outlines possible extensions.

[3] Proposing to the customer complementary products or services to the one just bought

[4] Encouraging the customer to purchase a greater quantity or an upgraded version of products or services

2 Background

This chapter consists of a brief overview of clustering and dimensionality reduction algorithms.

2.1 Clustering

Clustering is a sub-task of unsupervised learning with the aim of grouping similar items and assigning them to groups of similar instances. Clustering algorithms are not only used for *customer segmentation* like in this work, but also for *semi-supervised learning*, *search engines*, *image segmentation*, etc [4]. Algorithms described here measure the similarity of points with distance metrics. The distance of two points is measured in a multi-dimensional space. The Euclidean distance is a widely used distance metric, defined as:

$$d_1(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.1}$$

where p, q are two points and p_i, q_i their coordinates in a n-dimensional space. Another widely used distance metric is the Manhattan distance, defined as:

$$d_2(p,q) = \sum_{i=1}^{n}|p_i - q_i| \tag{2.2}$$

These two metrics can be generalized by the Minkowski distance (2.3). It represents the Manhattan distance when $h = 1$ (also called L1 norm) and Euclidean distance when $h = 2$ (also called L2 norm).

$$d_3(p,q) = \sqrt[h]{\sum_{i=1}^{n}|p_i - q_i|^h} \tag{2.3}$$

Clustering can be divided into *hard clustering* and *soft clustering*. Hard clustering assigns to each sample the label of the cluster to which the sample belongs. On the other hand, soft clustering assigns to the samples a score for each cluster. This score could be, for example, the distance of the point from each cluster. With soft clustering a sample can belong to multiple clusters but with different degrees.

There are three main clustering methods:

- *Density-Based Methods*: Clusters are assembled measuring the distance between points. The number of clusters is inferred by the algorithm.
- *Hierarchical Based Methods*: Clusters form a hierarchical tree-type structure called dendrogram. At the beginning every sample is an independent cluster. At each iteration, similar clusters are joined together until reaching some stopping criterion or until a single global cluster is available. The number of clusters is decided by the ML engineer analyzing the dendrogram.
- *Partitioning Methods*: These methods partition the dataset into k clusters by optimizing an objective criterion. The number of clusters k is a hyperparameter.

2.1.1 K-means

The K-means algorithm is a partitioning method for clustering. The number of clusters k is defined by the ML engineer as a hyperparameter. The K-means algorithm aims at choosing centroids that

minimize the *inertia*, that is the objective criterion.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2) \tag{2.4}$$

Each of the k disjoint clusters C is defined by a centroid, that is the mean μ of the samples x in that cluster. Centroids do not have to be necessarily points of the dataset. Inertia (2.4) can be seen as a measure of how internally coherent clusters are. It assumes convex clusters [10]. A cluster is said to be convex if given any two points, it contains the whole line segment that joins them.

The algorithm works by moving centroids at every iteration in order to minimize the inertia. At the first iteration, centroids are randomly initialized. To avoid blocking on local minima, the algorithm is executed n_init times and the best solution is kept. To speed up the computation a more sophisticated technique for initializing the random centroids was proposed, which takes the name of K-means++. It selects centroids far away from each other, so the risk of reaching sub-optimal solutions is reduced [2].

2.1.2 DBSCAN

DBSCAN is a Density-Based method for clustering. It considers clusters as areas of high density separated by areas of low density. For this reason, clusters can have any kind of shape and not strictly convex as in the case of K-means. Figure (2.1) shows the behaviour of K-means and DBSCAN on different datasets.

The algorithm requires two important hyperparameters, *min_samples* and *eps*. DBSCAN works by picking a random sample from the dataset and checking if in its neighborhood, within a radius of *eps*, there are at least *min_samples* points. If yes, then the picked sample is classified as *core-sample*. If a point is not a *core-sample* and is at least *eps* in distance from any *core-sample*, then the point is classified as an anomaly with the default label of -1. Every *core-sample* forms a clusters with all points reachable from it. So, higher *min_samples* or lower *eps* indicate higher density necessary to form a cluster. DBSCAN is deterministic and very susceptible to hyperparameters changes.

The main drawback of the sklearn's[1] implementation of DBSCAN is that it does not expose the method *predict*(). So it is not possible to use DBSCAN to predict the cluster of new samples. To overcome this limitation we can use a supervised learning algorithm, called K-Nearest-Neighbor (KNN) [4]. We train the KNN algorithm on the core-samples using the cluster label as target of the supervised learning algorithm.

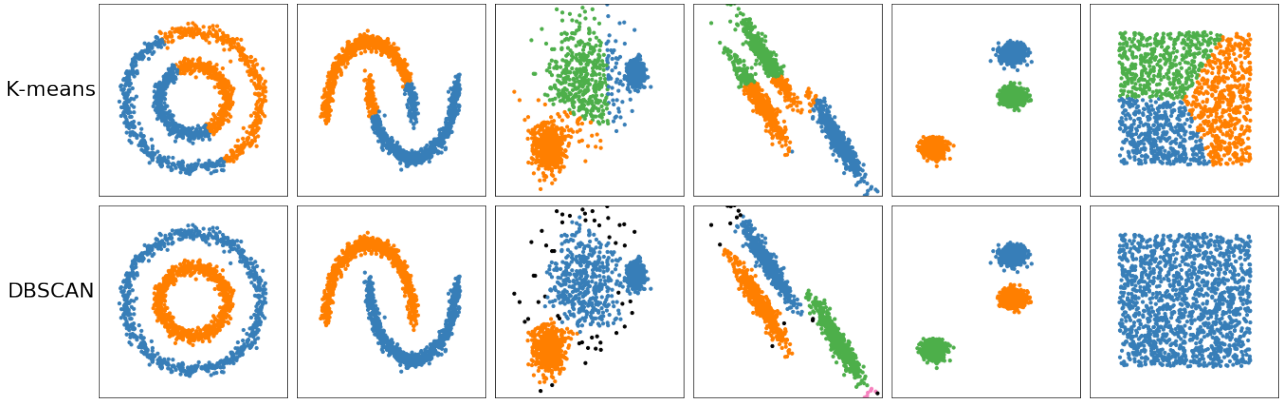


Figure 2.1: K-means and DBSCAN on different toy datasets

[1] sklearn (abbreviation of Scikit-learn) is an open source Python library for Machine Learning

2.2 Dimensionality reduction

Machine Learning problems can involve dataset with thousands or even millions of features. One of the biggest problem with these high-dimensional datasets is that the training of algorithms can be time consuming. Moreover, with so high-dimensional datasets, finding good solutions can become much more complex due to the problem of the *curse of dimensionality*. The curse of dimensionality refers to the distribution of points in high-dimensional spaces. In higher-dimensional spaces, points tend to be much more sparse than lower-dimensional spaces [4]. Some dimensionality reduction algorithms can also be used for *anomaly* and *novelty detection* as will be explained in section 5.0.1.

Another useful application of dimensionality reduction algorithms is to enable the plotting of datasets with high dimensionality. The inability to plot data is pretty serious since many companies are adopting data-driven approaches and *DataViz* is the only way to communicate results or problems to customers, managers, or decision-makers.

There are a lot of algorithms for dimensionality reduction, here an introduction to *PCA* and *Autoencoder* is provided.

2.2.1 PCA

PCA, or Principal Component Analysis, finds the hyperplanes that account for the largest amount of variance and it linearly projects the samples onto them. Keeping the hyperplanes that preserve the largest amount of variance will most likely lose less information than the other projections [4]. The i^{th} hyperplane is called the i^{th} principal component. Thus, each principal component explains a certain amount of variance of the original dataset.

The number of principal components to keep in the compressed dataset is a hyperparameter of the algorithm. If dimensionality reduction is applied for DataViz, than this hyperparameter will be typically set to 2 (for 2D plots) or 3 (for 3D plots). Otherwise, we can set this hyperparameter to any number between 1 and the original dimensionality of the dataset. To select the best number of principal components to keep, we can consider the cumulative principal components' explained variance as illustrated in Figure 2. Thus, we can choose the value of this hyperparameter that reaches the best trade-off between the number of principal components to keep and their explained variance.

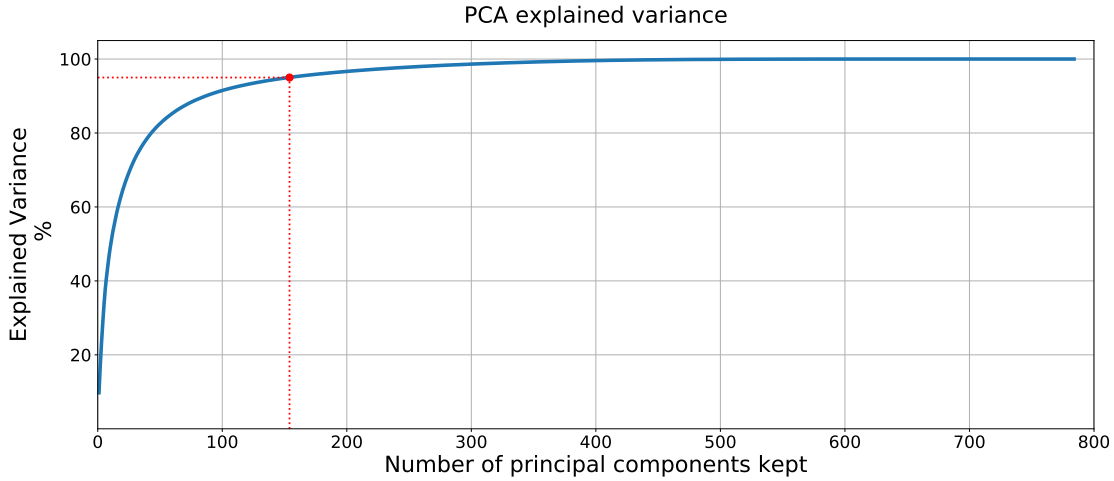


Figure 2.2: Explained variance as a function of the number of principal components kept. With ~ 150 principal components, we preserve the 95% of variance of the original dataset, whose dimensionality is ~ 800

Some variations to the original PCA are *Randomized PCA* (It finds an approximation of the principal components quicker than computing the exact values), and *Kernel-PCA* (It allows non-linear projections).

2.2.2 Autoencoder

The Autoencoder is a special kind of Artificial Neural Network architecture that learns a *latent representation* of the input data. Undercomplete Autoencoders learn a lower dimensionality representation of the input data, while overcomplete Autoencoders learn a higher dimensionality representation. For this reason, undercomplete Autoencoders are suitable for dimensionality reduction. Autoencoders can also be used to denoise images, to pre-train deep neural networks, and as generative models[2] [4].

The general structure of an undercomplete Autoencoder is presented in Figure (2.3). The Autoencoder tries to find a latent representation that allows the reconstruction of the input data as closest as possible to the original input.

Autoencoders trained in this work are implemented with Tensorflow, a popular open source library for numerical computation.

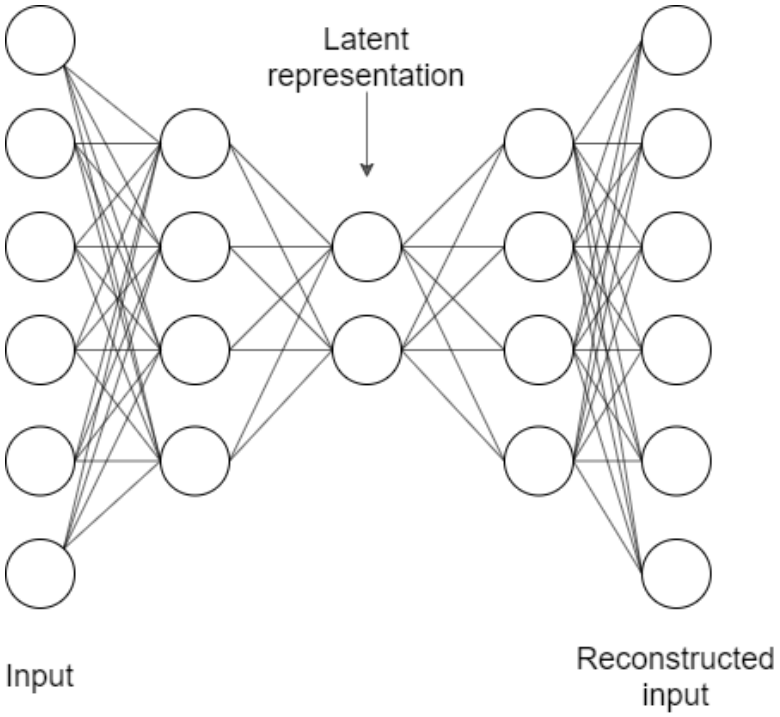


Figure 2.3: Undercomplete Autoencoder

[2]Randomly generate new data that looks like the training data [7]

3 System description

Figure (3.1) illustrates the overall structure of the system. As we can see, a web proxy mediates the communication between the web application and the IBM back-end, in order to overcome the IBM API's limitations on CORS (Cross-Origin Resource Sharing[1]). The last step of *Deployment* stores the dataset used to train the pipeline and some statistics about clusters to an instance of Cloud Firestore[2], to allow the web application to retrieve easily the dataset and to plot clusters. The trained pipeline is deployed on *IBM Watson Machine Learning*[3].

Figure (3.2) focuses on the interaction between the web application, the web proxy, and the IBM backend.

Each step in Figure (3.1) will be described in detail in the following chapters.

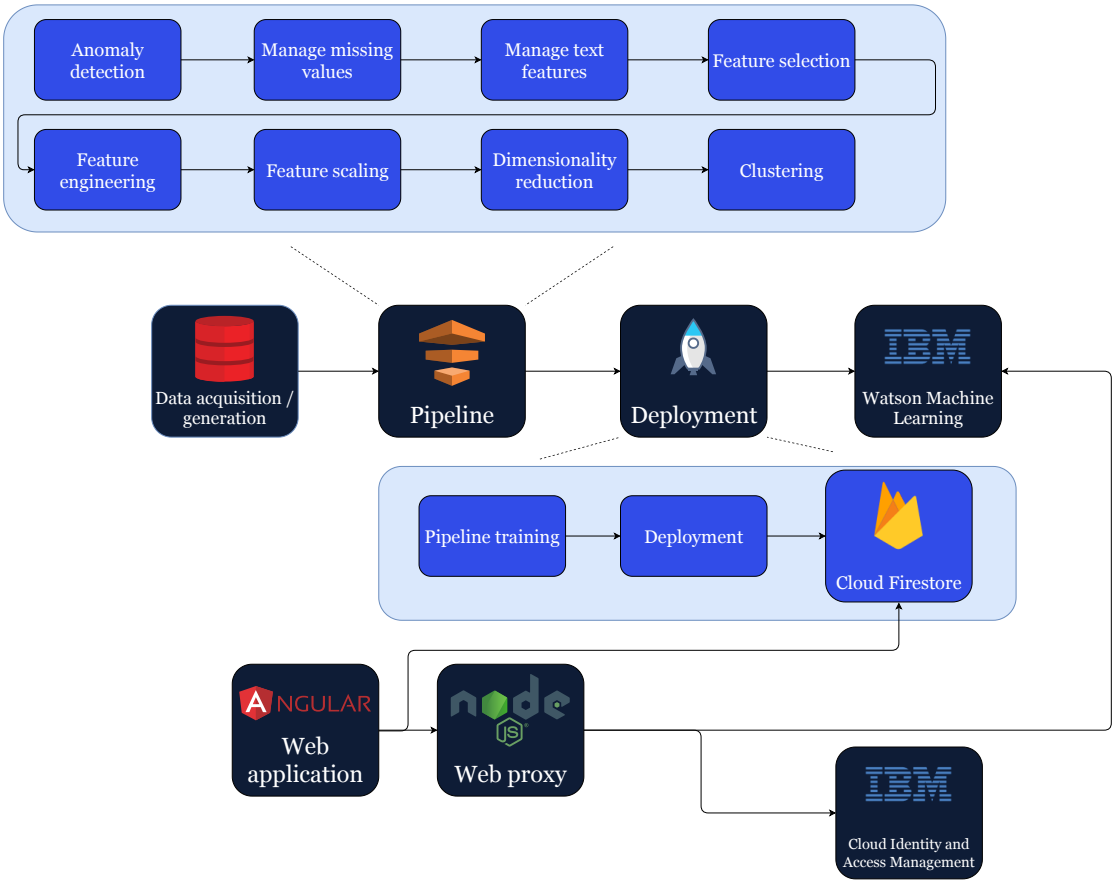


Figure 3.1: Overall structure of the system

[1]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[2]NoSql Database offered by Firebase, which is a web application development platform
[3]https://www.ibm.com/cloud/machine-learning

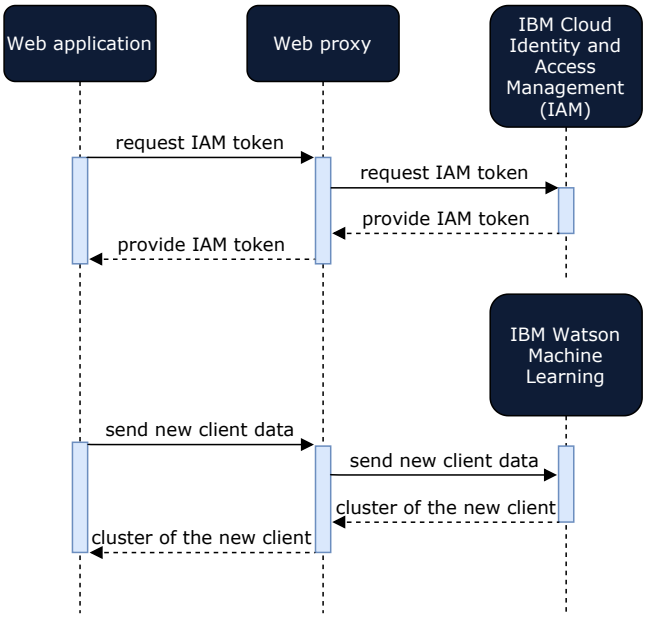


Figure 3.2: Sequence diagram for interactions between the web app, the web proxy the IBM back-end

4 Data generation

As stated in the introduction, the dataset used in this work is synthetic. This chapter describes the process of the random data generation.

The list with required features provided by Blue Reply contains:

- *Personal data*: (age, gender, marital status, educational qualification, etc.)
- *Geographic data*: (district, latitude and longitude, etc.)
- *Family data*:(number of sons, age of the youngest son, number of dependent family members, etc.)
- *Insurance information*(income bracket, heritage, propensity to delegate and risk tolerance, etc.)

Rather than generating fully random data, a series of constraints validated by Blue Reply were adopted to generate plausible data. Algorithms (1,2,3) illustrate a brief and simplified summary of the random generation for some features. Moreover, geographic data was generated coherently and referring to a real database[1]. This means that only *city_id* is random, the other information like region, population, latitude and longitude are taken from the database referring to the chosen city.

Algorithm 1: Generation of the age

```
generateAge (person,minAge,maxAge)
    person.age ← random(minAge, maxAge+1)
    return person
```

Algorithm 2: Generation of the marital status

```
availableStatuses ← {All available statuses e.g married, widow, divorced, single...}
generateMaritalStatus (person,isMarried)
    if isMarried then
        person.status ← "married"
        return person

    end ← false
    while ¬ end do
        end ← True
        status ← random(availableStatuses \{"married"})
        if person.age ≤ 24 ∧ status ∈ {"widow","divorced"} then
            end ← False
        else if person.age ≥ 60 ∧ status ∈ {"cohabitant"} then
            end ← False
        end
    end

    person.status ← status
    return person
```

[1] https://github.com/matteocontrini/comuni-json

Algorithm 3: Generation of the number of sons, the age of the youngest son and the number of dependent family members

```
generateFamily (person,maxSons,minAgeForSons)
    if person.age > 28 then
        if random(0,2) ≠ 0 then
            person.n_sons ← random(0, maxSons+1)
        else
            person.n_sons ← 0
        end
    else if person.age > 22 then
        person.n_sons ← random(0,2)
    else
        person.n_sons ← 0
    end

    // if he/she has sons, is more likely to be married
    if person.n_sons > 0 then
        upperValIsMarried ← 2
    else
        upperValIsMarried ← 3
    end

    probIsMarried ← random(0, upperValIsMarried)
    if (probIsMarried == 0 ∧ person.age ≥ 23) ∨ person.n_sons ≥ 2 then
        isMarried ← 1
    else
        isMarried ← 0
    end

    person ← generateStatus(person, isMarried)
    person.n_fam_dependent ← person.n_fam_old + person.n_sons + isMarried

    if person.n_sons ≥ 1 then
        person.age_younger_son ← random(0, person.age - random(minAgeForSons + 2 *
            person.n_sons, person.age ) ) // estimate a real age for the youngest son,
            not too close to the age of the person
    if person.age_younger_son ≥ 25 then
        // if True, probably sons are no more dependent on their parents

        probRemainDependent = random(0,5)

        if probRemainDependent ≠ 0 then
            // 1/4 prob that sons are still dependent on their parents
            person.n_fam_dependent -= person.n_sons

    return person
```

At the end of the data generation process, an exploratory data analysis was performed to explore the distribution of the population. Figure (4.1) is an overview. Moreover, the generated dataset is saved on *IBM Cloud Object Storage*[2] to allow the pipeline to access the data.

[2]https://www.ibm.com/cloud/object-storage

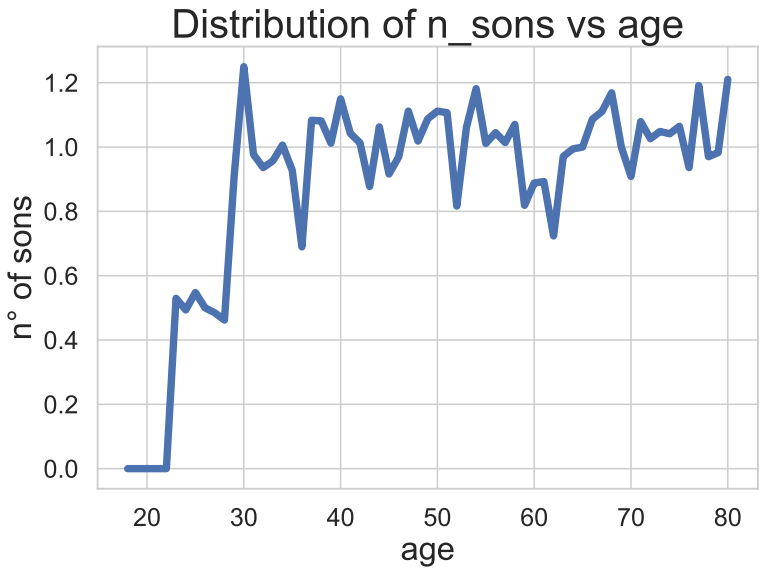


Figure 4.1: Example of exploratory data analysis performed on the generated data

5 The pipeline

The data pre-processing pipeline is organized as follows:

- Anomaly detection
- Handling missing values
- Handling text features
- Feature selection
- Feature engineering
- Feature scaling

Each step of the pipeline was developed as a transformer, i.e, a class implementing the methods *fit* (to fit the transformer), *transform* (to apply the transformation) and *fit_transform* (to perform fit and transform in a single line of code). Below the description of each step.

5.0.1 Anomaly detection

The goal of *Anomaly detection* is to find anomalies, or outliers. Anomalies are samples that greatly differ from the distribution of normal samples and, in many cases, they negatively affect generalization performances of Machine Learning models. Anomaly detection can be performed with several techniques, for example:

Visualization. The simplest idea to find outliers is to plot feature-by-feature all samples in order to visually identify potential outliers. A simple but non-scalable approach.

Dimensionality reduction. The core idea is to reduce the dimensionality of the dataset and then restore the original dimension. The goal is to preserve only relevant features in the forward step while forgetting the irrelevant details of the data. After the back step, the data will not be exactly reconstructed, but an error will be introduced between the original full-dimensional data and the full-dimensional reconstructed dataset. This reconstruction error is assigned to each sample as an anomaly score. The greater the reconstruction error is, the greater the probability that the sample is an anomaly. This procedure works thanks to the fact that the dimensionality reduction algorithm will have the largest reconstruction error on those samples that are hardest to model, or in other words, those that occur the least often and are the most anomalous [9]. Suitable dimensionality reduction algorithms are, among the others, PCA and Autoencoders.

Neighborhood. Based on the concept of distance. Given the hyperparameter K, the distance of a sample to its K^{th} nearest neighbor is its anomaly score. The neighborhood can be measured via an algorithm like K-Nearest Neighbors. A variation of this procedure is that if the sample X hasn't *min_samples* neighbors in its neighborhood, than X is marked as an anomaly. This is a short description of anomaly detection with the DBSCAN algorithm.

Tree based. An example of tree-based algorithm for anomaly detection is *IsolationForest* [8]. It randomly selects a feature and a split value between the maximum and minimum values of the selected feature. A simplified version of this procedure is presented by Algorithm (4). The number of splits required to isolate a sample is equivalent to the path length from the root node to the leaf. The average path length over all trees of the forest is a measure of normality and our decision function. Samples with short paths are highly likely to be anomalies [8].

The aim of the anomaly detection step, rather than being simply finding outliers on the training data, was to find anomalies on new samples coming from the web application. This is known as *Novelty*

Algorithm 4: A simplified version of the tree-construction algorithm. The complete code is available in the official paper [8]

input : X - input data
output: An IsolationTree (iTree)
iTree (X)
  if $|X| \leq 1$ **then**
    `// leaf node`
    return *Node(left* $\leftarrow$ *NULL,*
      right $\leftarrow$ *NULL)*
  else
    let Q be a list of attributes in X
    randomly select an attribute q $\in$ Q
    randomly select a split point p from max and min values of attribute q in X

    $X_l \leftarrow$ filter(X, q $<$ p)
    $X_r \leftarrow$ filter(X, q $\geq$ p)
    return *Node(left* $\leftarrow$ *iTree*(X_l)*,*
      right $\leftarrow$ *iTree*(X_r)*,*
      splitAttr $\leftarrow$ *q,*
      splitValue $\leftarrow$ *p)*
  end

detection. At inference time a possible usage scenario is to advise the user of the web app that while inserting the data of a customer, he/she probably committed a typographical error, such as adding a zero at the end of the income bracket of a part-time worker.

Since the generated dataset lacks the ground truth, it is not possible to compare different algorithms to choose the best one for our dataset. To provide a benchmark of anomaly detection algorithms, the supervised problem of *Credit Card Fraud Detection*[1] was addressed with algorithms for unsupervised anomaly detection. This problem is a classification problem with two classes, the positive class (fraud) and the negative class (not fraud). Figure (5.1) shows the results of the two best performing algorithms, PCA and Autoencoders. Since PCA performed better than Autoencoder in this task, and considering that PCA requires much less tuning than Autoencoders, the anomaly detection step in the pipeline is based on PCA. Performances are evaluated by $Precision$, $Recall$ and $F_1 score$:

$$Precision = \frac{tp}{tp + fp} \tag{5.1}$$

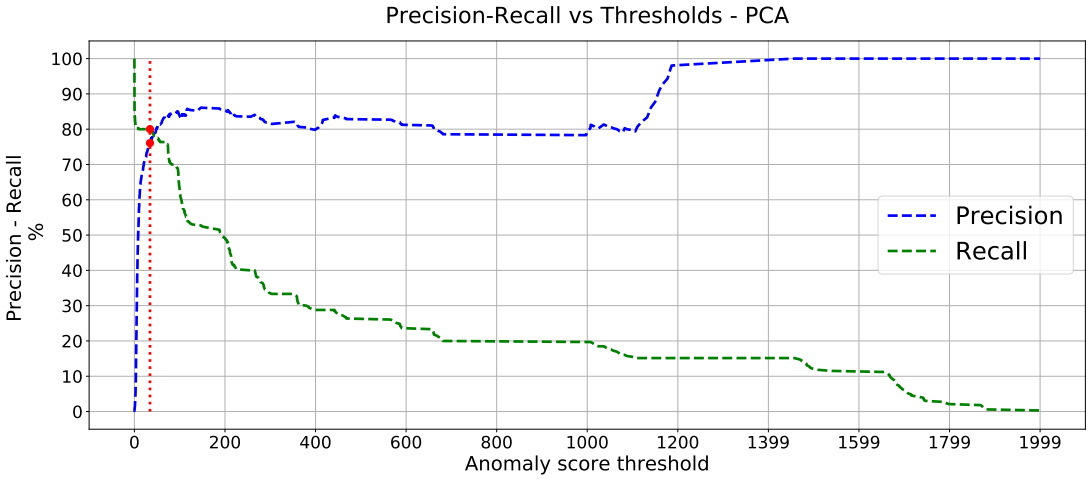
$$Recall = \frac{tp}{tp + fn} \tag{5.2}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

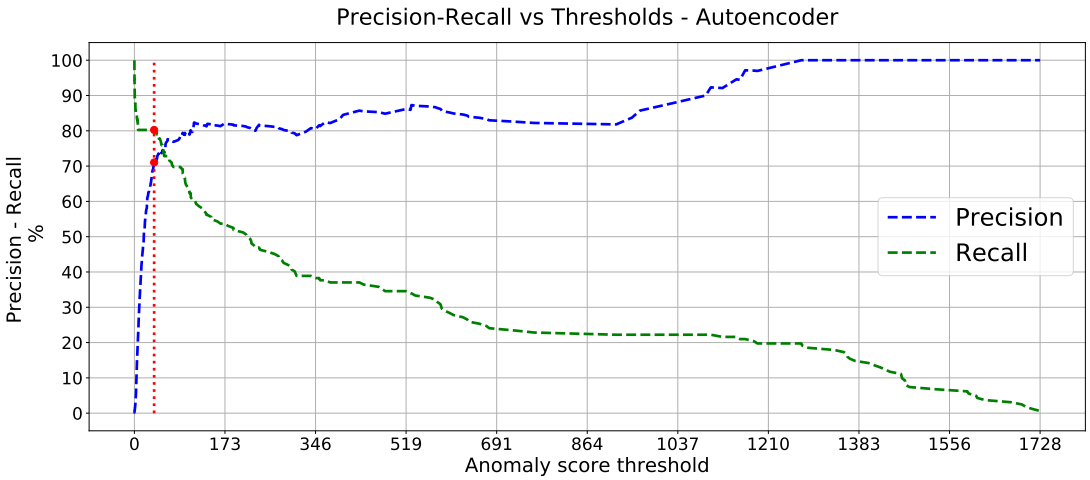
where tp are the true positives (i.e fraud samples classified correctly), fp the false positives (i.e normal samples classified as frauds), and fn the false negatives (i.e fraud samples classified as non-fraud). Additionally, a confusion matrix was plotted to highlight errors made by the unsupervised classifier. Such matrix displays on the rows the true class of the samples, while on each column the actual class predicted by the classifier. A perfect classifier would have non-zero elements only on the main diagonal.

Following the general and didactic approach of this work, an additional notebook was developed. Its aim is to investigate the anomaly scores assigned by the vanilla anomaly detector of the pipeline to the synthetic training set. An example of a high level data analysis is illustrated in Figure (5.2).

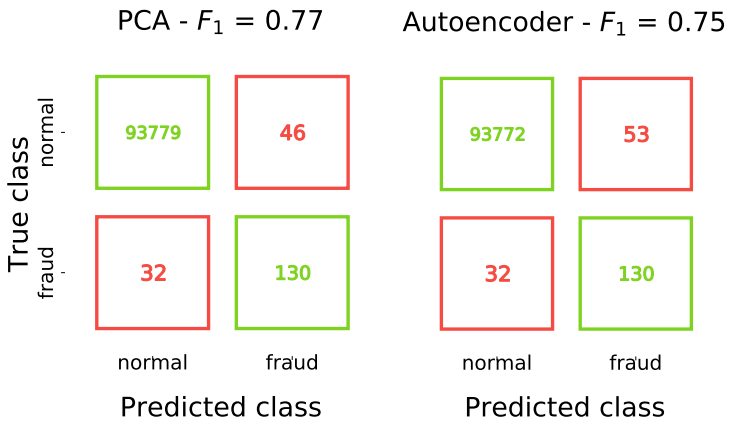
[1] https://www.kaggle.com/mlg-ulb/creditcardfraud



(a) With an anomaly score threshold of 34, the algorithm achieves 80% of recall and 76% of precision on the training set



(b) With an anomaly score threshold of 38, the algorithm achieves 80% of recall and 71% of precision on the training set



(c) Confusion matrix of the test set

(d) Confusion matrix of the test set

Figure 5.1: Precision, Recall and confusion matrix of PCA and Autoencoder for Credit Card Fraud Detection

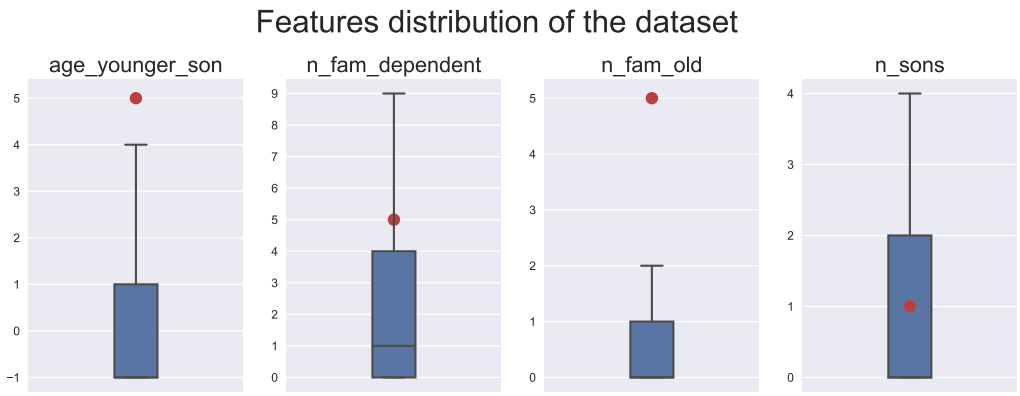


Figure 5.2: Values of four features of the sample with the highest anomaly score in the training set (red points), plotted against the entire dataset (boxplots). The analyzed sample has at least two features whose value greatly differ from the dataset's distribution. The value "-1" in *age_younger_son* corresponds to "no sons"

5.0.2 Handling missing values

The problem of missing values is common with real-world datasets. The causes can be anomalies in sensors or simply a general lack of information. Usually, in Python, missing values are encoded with *NaNs*. Often Machine Learning models can't deal with NaN values, so a proper imputation is required. There are several techniques for data imputation, for example:

Drop NaNs. If few samples of the dataset have NaNs, it is possible to drop those samples. It's also possible to drop entire features, if the number of NaNs is very high for that feature.

Constant imputation. NaNs are imputed with constant values. For example, in the context of surveys, a missing value could be replaced by the default value "I prefer not to answer".

Statistical imputation. NaNs of a feature f are imputed with statistics obtained from f without considering those NaNs. A predefined statistic is computed for each feature and NaNs are replaced with the computed statistical value. Values used to impute the training set must be preserved to impute the test set. Suitable statistics are, for example, mean and mode. The former is usually used with continuous features, while the latter with discrete features.

Machine Learning. An ad-hoc algorithm can be trained on non-NaN samples, of all the features except f. The target of the supervised task is the feature f, in order to predict the value for all NaNs of f. If multiple features contain NaNs, then multiple algorithms must be trained at the cost of a higher computational cost.

In the pipeline, the first three techniques are implemented. *RowNaNDropper* is a custom transformer that accepts as input a list of features to check. If NaNs are present in these features, it drops samples with those NaNs. For statistical imputation the sklearn's transformer *SimpleImputer*[2] is used. To link features with their relative imputation technique, a wrapper data structure was developed. Since SimpleImputer by default applies its transformation to the entire dataset passed as input, the sklearn's transformer *ColumnTransformer* was used. It allows to apply different transformations to different subsets of features.

The result is a fully configurable step where the Machine Learning engineer can choose the best imputation technique for each feature and, if constant imputation is chosen, can specify the constant value to use as imputation. If the Machine Learning engineer needs to implement a Machine Learning model for imputation, he will have to implement a transformer and he will have to place it inside the ColumnTransformer with its relative features. The basic structure of ColumnTransformer is reported below.

[2]Supports mean, median, most_frequent and constant imputations

```
ColumnTransformer([
    (name="mean", transformer=SimpleImputer(strategy="mean"),
     columns=mean_column_NaN),
    ("mode", SimpleImputer(strategy="most_frequent"), mode_column_NaN),
    ("ml_model", FutureTransformer(...), ml_model_column_NaN),
])
```

5.0.3 Handling text features

Similarly to missing values, Machine Learning models can't deal with text features. Sklearn's transformers *OrdinalEncoder* and *OneHotEncoder* are two of the most common reversible techniques for handling text features.

OrdinalEncoder creates a dictionary that links every textual value to an integer. Thus, a textual array is converted to an integer array.

OneHotEncoder creates a one-hot representation of each value to be converted. A one-hot representation consists of a vector with all zeros but one, which corresponds to the textual value. The length of such a vector is equal to the cardinality of the feature to encode. It is suitable for low-cardinality features, otherwise a very sparse vector will be created.

ColumnTransformer is again used to differentiate the encoding scheme across features, as illustrated above.

5.0.4 Feature selection

The feature selection step allows to filter out the most promising features to build the final model, and at the same time to drop attributes that provide no useful information for the task.

This step is more suited for supervised tasks where, for example, we can train an algorithm with different subsets of features tracking which subset obtained the best score[3]. Others techniques involve the use of statistical indexes to measure the correlation between features, such as the Pearson correlation[4].

In this work the feature selection step is limited to dropping features that surely have no information, such as incremental ids.

5.0.5 Feature engineering

Feature engineering is referred to the process of extracting new composite features from the original feature space or to extract features from raw data.

This step is not always needed. In some cases we may have access only to already composite features where the raw data is not accessible. Moreover, the rise of Deep Learning opened up to automatic feature extraction from raw data [11]. If Deep Learning techniques are not suitable for the specific task, there are also plain libraries to perform automatic feature engineering, such as Featuretools[5]

In this work the feature engineering step performs binning of high cardinality discrete features, such as *n_people_city*. The goal of this feature is to express the dimension of the city where a customer lives, not to express the exact number of people. Since geographical information was taken from a real database, the number of people in each city is exact. To avoid the potential problem of discriminating customers only for a few hundreds of people of difference in their cities, the feature is binned. The same applies also for ages. The sklearn's transformer *KBinsDiscretizer* bins the input data into *n_bins* bins, where *n_bins* is a user-defined parameter. My implementation of the feature engineering transformer can apply KBinsDiscretizer only to certain features, and not to the entire dataset as the original KBinsDiscretizer. To this end, a custom wrapper was developed to allow the ML engineer to link features to be binned with their relative parameter *n_bins*:

[3] An example is Recursive Feature Elimination (RFE) provided by sklearn

[4] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[5] https://docs.featuretools.com/en/stable

```
to_be_binned = MyWrapper([('n_people_city', n_bins_people),(...),(...)])
fe_step = FeatureEngineer(to_bin=to_be_binned)
```

5.0.6 Feature scaling

Feature scaling is a very important step in a Machine Learning pipeline. Algorithms that use distance-based metrics, such as the Euclidean distance, are very susceptible to the magnitude of features. The algorithm can be biased by features with a broad range of values, because the distance will be governed by these particular features. At the same time, also not distance-related algorithms are affected by scaling. For example, Gradient Descent[6] will have a faster convergence if the feature space is equally scaled. If features are not scaled properly, parameters will change quickly on small ranges and slowly on large ranges. Figure (5.3) testifies that different algorithms have different impacts.

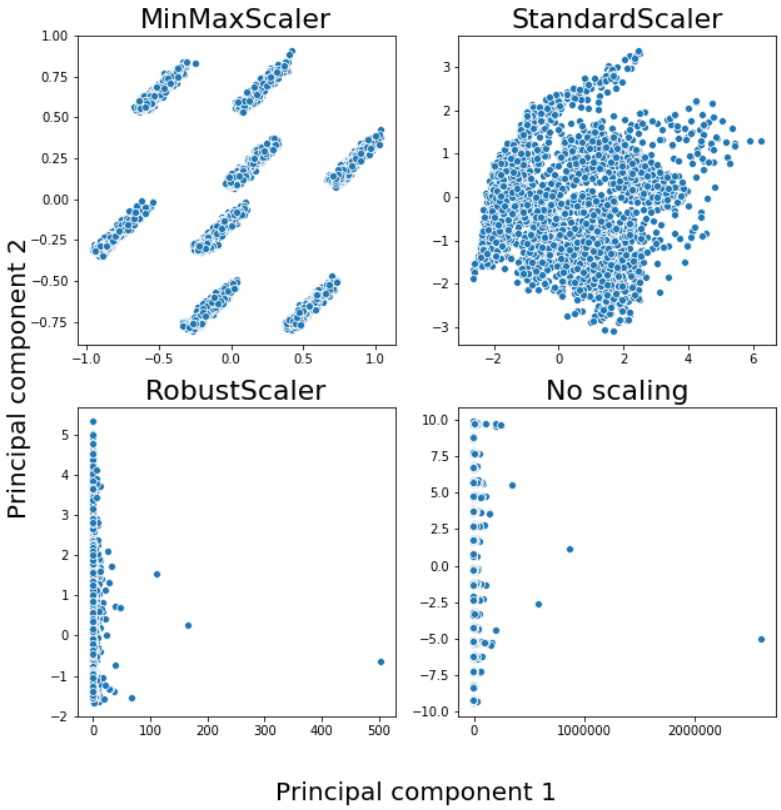


Figure 5.3: Comparison of different scaling algorithms provided by sklearn, applied to the same dataset. First scaling is applied, then the dimensionality is reduced by means of PCA

5.0.7 Assembling

Once all steps are defined, the entire pipeline can be assembled as follows:

```
nan_pipe = ColumnTransformer([
    ("numerical_nan", SimpleImputer(strategy="mean"), mean_col_nan),
    ("categorical_nan", SimpleImputer(strategy="most_frequent"), mode_col_nan),
] + hand_imputed_col)
nan_pipe = Pipeline([
    ('drop_row_nan', RowNaNDropper(c, col_to_check=drop_row_nan)),
    ('nan_pipe', nan_pipe)
```

[6]Popular algorithm to train Neural Networks

```
])
txt_pipe = ColumnTransformer([
    ("ordinal", OrdinalEncoder(), ordinal_col_nan),
])
final_pipe = Pipeline([
    ('missing_values', nan_pipe),
    ('text_features', txt_pipe),
    ('feature_selection', FeatureSelector(c, useless_col=useless_col)),
    ('feature_eng', FeatureEngineer(c, to_discretize=to_discretize)),
    ('scaling', StandardScaler())
])
```

The main difference between *Pipeline* and *ColumnTransformer* is that Pipeline applies its transformations to each column, while with ColumnTransformer one can specify to which columns the transformer will be applied. The anomaly detection step is kept outside of $final\_pipeline$, so at inference time we can check separately whether the test sample is an anomaly or not, without computing all the other steps of $final\_pipeline$. The parameter c is present in all custom transformers and consists of a class with handy functions, such as the configuration of a log-tracking tool for hyperparameters and metrics. It is possible to configure any kind of log-tracking tool, from a custom database to business services such as Comet[7]. In this work, Comet was used to compare different Autoencoder architectures for the anomaly detection benchmark (Figure (5.1)). To make the configuration faster, the skeleton to configure Comet is provided.

In addition to the steps presented above, dimensionality reduction and clustering are also inserted into the pipeline, providing a greater modularity of the code. Once all steps are assembled, the pipeline needs to be fitted on the training data. The $final\_pipeline.fit()$ call starts the fitting procedure. The $.fit()$ of the first step is automatically invoked, then $.transform()$ produces the output that will be fed to the next step. The procedure is iterated until the last step is reached, where only $.fit()$ will be called, without $.transform()$.

[7] https://www.comet.ml/site/

6 Optimization and interpretability

As stated in the introduction, interpretability plays an important role in this work. But as section 2.1 states, many clustering algorithms work by simply clustering together near points in the feature space, without providing any feedback on how coherent these clusters are. To analyze from an insurance point of view how clusters are composed and their characteristics, an ad-hoc index was developed, which is called *residual impurity*.

The idea behind this index is to measure via some statistical descriptors the distribution of points in the original dataset and, after clustering is performed, to compare via the same statistical descriptors the distribution of points within each cluster with the initial condition.

The development of the mentioned index has been iterative.

6.1 Residual impurity index

6.1.1 First step

The first step involved the definition of the residual impurity index. The statistical descriptors chosen are *standard deviation* for ordered features and *Gini impurity* for unordered features. The standard deviation σ is a measure of the amount of dispersion of a set of values X. A low standard deviation indicates that the values are close to the mean, while a high standard deviation indicates that the values are spread over a wider range.

$$\sigma_X = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}} \tag{6.1}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ represents the mean value of X.

Gini impurity is usually used for Decision Trees[1] learning. It consists of a measure of impurity, and is defined as:

$$G(p) = 1 - \sum_{i \in J} {p_i}^2 \tag{6.2}$$

where J is the set of elements of X and p_i is the normalized fraction of elements of X with value i. The residual impurity index is defined by equation (6.3) and (6.4). Note that $L.R.I$ is the local residual impurity, i.e, the residual impurity of a single feature within a cluster.

$$R.I.(C) = \frac{1}{|C|}\sum_{c \in C}\frac{1}{|f_1| + |f_2|}\sum_{f \in f_1 \cup f_2} L.R.I(c, f) \tag{6.3}$$

$$L.R.I.(c, f) = \begin{cases} \frac{\sigma_{fc}}{\sigma_f}, f \in f_1 \\ \\ \frac{g_{fc}}{g_f}, f \in f_2 \end{cases} \tag{6.4}$$

where f_1 is the set of ordered features, f_2 is the set of unordered features, σ_{fc} and g_{fc} are standard deviation and Gini impurity of the feature f in the cluster c, σ_f and g_f are standard deviation and Gini impurity of the feature f in the entire dataset.

[1] Tree-based supervised algorithm

6.1.2 Second step

The second step was driven by the fact that a cluster, to be meaningful from an insurance point of view, does not need to have a low residual impurity score for every feature. As reported in the introduction, examples of meaningful clusters for the insurance market are:

- Customers with similar ages
- Customers that, despite different ages, share the same purchase power
- Customers that, despite different ages and different purchase power, share the same family conditions (number of sons, number of dependent elderly family members, etc..)

So, to account this characteristic, rather than averaging all features together in the computation of the residual impurity, only the K features with the lowest local impurity score are taken into account. K is a configurable parameter, set to a default value of 3.

6.1.3 Third step

The approach adopted in this step consisted of using the residual impurity score not only to evaluate a given clustering, but also to choose the algorithm that achieves the lowest residual impurity score. In fact, since the residual impurity score is designed to express the informative power of clusters from an insurance point of view, and the goal of clustering is to find meaningful clusters, the residual impurity score can be used to measure the performances of the algorithms.

In particular, the residual impurity score was used as an objective function to be minimized within a process of *Bayesian optimization*. Bayesian optimization aims at finding the best hyperparameters for an algorithm through the exploration of the hyperparameters space, together with the evaluation of an objective function. The hyperparameter optimization implemented in this work can be represented by the following equation:

$$x^* = \underset{x \in X}{\operatorname{argmin}}\, R.I.(G(P(D, x_d), x_c)) \tag{6.5}$$

where x^* is the set of hyperparameters that yields the lowest value of the score, X is the hyperparameters space, x_c and x_d are two subsets of x that respectively contain the hyperparameters for the clustering and dimensionality reduction algorithm, D is the dataset, G is an algorithm for clustering, and P is an algorithm for dimensionality reduction.

Bayesian optimization is implemented in many libraries and in this work Hyperopt[2] has been used. In the code below, an example of hyperparameter space for DBSCAN is presented by using the Hyperopt APIs:

```
from hyperopt import hp

space = {
    'eps': hp.uniform(label='eps', low=0.001, high=10),
    'min_samples': hp.uniform('min_samples', 3, 10)
}
```

Bayesian optimization, in contrast to random[3] and grid search[4], keeps track of past evaluation results which are used to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function. Thus, compared to random search, it invests more time evaluating hyperparameters that appear more promising from past results, rather than randomly sampling the hyperparameters from the space.

The experiments shown below are structured as follows: The Bayesian optimization process is performed for a fixed number of iterations and for each couple dimensionality reduction - clustering

[2] https://github.com/hyperopt/hyperopt

[3] Random search randomly samples hyperparameters from a user-defined space

[4] Grid search tests every possible combination of parameters defined by the user

algorithm to test. At each iteration, the sampled hyperparameters and the chosen algorithms are trained and the R.I score is calculated. At the end of the optimization process, the hyperparameters with the lowest R.I score are kept. All random seeds are fixed, to ensure reproducibility.

A useful property of the residual impurity score defined by (6.3) and (6.4) is that $R.I.$ and $L.R.I.$ can be interpreted independently. In fact, table (6.1) shows the R.I. values for two experiments, one with the couple (Kernel-PCA + K-means) and the other with the couple (PCA + DBSCAN). Values close to zero mean that, on average, clusters are homogeneous, while values close to one mean that clusters are heterogeneous. Table (6.2) and (6.3) show the shape of clusters for the experiments reported in table (6.1). As we can see, both algorithms isolated many tiny clusters and preserved a single huge cluster (that for DBSCAN is the cluster of outliers). Table (6.4) and (6.5) analyze the L.R.I values for three clusters and clearly show that the low R.I score obtained by the two algorithms come from the tiny clusters that are very homogeneous, while bigger clusters are much more impure.

Experiment	**R.I.**
Experiment with Kernel-PCA + K-means	0.06
Experiment with PCA + DBSCAN	0.06

Table 6.1: Outcome example of R.I. for two experiments.

Cluster n°	**# samples normalized**
0	80 %
1	$\leq$ 1 %
2	$\leq$ 1 %
3	$\leq$ 1%
4	$\leq$ 1 %
5	$\leq$ 1 %
6	$\leq$ 1 %
7	$\leq$ 1 %
8	$\leq$ 1%
9	$\leq$ 1%
10	$\leq$ 1%
11	$\leq$ 1%
12	$\leq$ 1%
13	4 %
14	$\leq$ 1%
15	3 %
16	3 %
17	$\leq$ 1%

Table 6.2: Shape of clusters for the experiment with Kernel-PCA + K-means

Cluster n°	**# samples normalized**
-1	78 %
0	$\leq$ 1%
1	$\leq$ 1%
..	..
74	$\leq$ 1%
75	$\leq$ 1%

Table 6.3: Shape of clusters for the experiment with PCA + DBSCAN

6.1.4 Fourth step

From table (6.1) it can be observed that the optimization process has obtained a very good score. However this score is given by the fact that many small clusters, with a very low impurity, have been created (from table 6.2 to 6.5). Since pure clusters are the clear minority of our dataset, the optimization process did not manage to find a good solution.

To reduce this phenomenon, some regularization terms were added to the formulation of R.I. Regularization is a widely used technique in Machine Learning to constraint an optimization problem in order

Feature name / Cluster n°	0	13	14
prop_delegate	1.0	1.0	0.0
n_sons	1.0	0.0	0.0
n_fam_old	1.0	0.0	0.0
age_younger_son	1.0	0.0	0.0
prop_risk	1.0	0.56	0.0

Table 6.4: L.R.I values related to the experiment of table (6.2). Only the three best features of cluster 0, 13 and 14 are reported. Cluster 0 has a value of 1 for all features

Feature name / Cluster n°	-1	0	1
n_sons	1.0	0.0	0.0
n_fam_old	1.0	0.0	0.0
age_younger_son	1.0	0.0	0.0

Table 6.5: L.R.I values related to the experiment of table (6.3). Only the three best features of cluster -1, 0 and 1 are reported. Cluster -1 has a value of 1 for all features.

to reduce the risk of overfitting[5]. It penalizes complex models while encouraging generalization.

The following regularization terms were added during the optimization process:

$$R_1 = \lambda_1 * |C|^2 \tag{6.6}$$

where λ is an hyperparameter that regulates how much the model is penalized, and C is the set of clusters. This regularization term aims at penalizing solutions with a huge number of clusters, where every cluster would have a very limited fraction of samples. It encourages the optimization process to find solutions with a limited number of clusters.

$$R_2 = \lambda_2 * \sigma^2_{clusters\_size} \tag{6.7}$$

where $\sigma_{clusters\_size}$ represents how much the size of clusters varies. This regularization term aims at reducing the possibility of having one or two huge clusters, while the others have a very low fraction of items.

$$R_3 = \lambda_3 * \frac{|c_{-1}|}{|D|} \tag{6.8}$$

where $|D|$ is the number of samples in the dataset, and $|c_{-1}|$ is the number of samples belonging to the cluster labeled with -1. This applies only to DBSCAN (or other algorithms that behave similarly), where the cluster -1 represents the cluster of potential outliers. This term is applied under the assumption that in the dataset there are no outliers (such as in this work), or outliers had been filtered by the anomaly detection step presented in section 5.0.1.

All these terms were added to the original definition of R.I., resulting in:

$$R.I.(C) = R_1 + R_2 + R_3 + \frac{1}{|C|} \sum_{c \in C} \frac{1}{|f_1| + |f_2|} \sum_{f \in f_1 \cup f_2} L.R.I(c, f) \tag{6.9}$$

Tables from (6.6) to (6.10) are the equivalent of tables from (6.1) to (6.5) except for the optimization process. The former tables use the regularized version of R.I, while the latter tables use the non-regularized version.

Experiment	R.I.
Experiment with Kernel-PCA + K-means	0.56
Experiment with PCA + DBSCAN	1.0

Table 6.6: Outcome example of R.I. for two experiments, with regularization.

[5]When a Machine Learning algorithm adapts too much to the training data and loses the ability to generalize to unseen examples

Cluster n°	# samples normalized
0	5 %
1	5 %
2	5 %
3	9 %
4	6 %
5	3 %
6	14 %
7	8 %
8	11 %
9	5 %
10	8 %
11	11 %
12	2 %
13	8 %

Table 6.7: Shape of clusters for the experiment with Kernel-PCA + K-means, with regularization.

Cluster n°	# samples normalized
0	100 %

Table 6.8: Shape of clusters for the experiment with PCA + DBSCAN, with regularization.

Feature name / Cluster n°	4	6	12
n_fam_old	0.17	0.52	0.0
n_fam_dependent	0.36	0.33	0.22
n_sons	0.5	0.29	0.0
prop_delegate	0.7	0.88	0.0

Table 6.9: L.R.I values related to the experiment of table (6.7). Only the three best features of cluster 4, 6 and 12 are reported.

Feature name / Cluster n°	0
all features	1.0

Table 6.10: L.R.I values related to the experiment of table (6.8).

As can be noted, with regularization, Kernel-PCA + K-means found 14 clusters with similar size and with a residual impurity of 0.56, while PCA + DBSCAN does not seem to be suitable for the task. At this point, we can describe for each cluster which are the K features that better characterize it, i.e, the K features with the lowest L.R.I. score, as illustrated in Figure (6.1). We could also create a word cloud with the names of features and their relative importance across all clusters (Figure (6.2)).

6.1.5 Last step

The last step of the development of the residual impurity index involved another regularization term, which focuses on penalizing easy-to-isolate features. The clustering algorithm can more easily isolate binary features, that are features with only two possible values, than high-cardinality features. This phenomenon was observed in the previous index experiments, where very often the same binary features appeared between the K most important features. To this end, the following regularization term was added to the definition of L.R.I (6.4):

$$R_4 = \lambda_4 * \frac{1}{|f|} \tag{6.10}$$

where f is the feature to be regularized. The idea behind R_4 is to penalize low-cardinality features. Binary features have the highest penalization because $|f|$ is equal to 2.

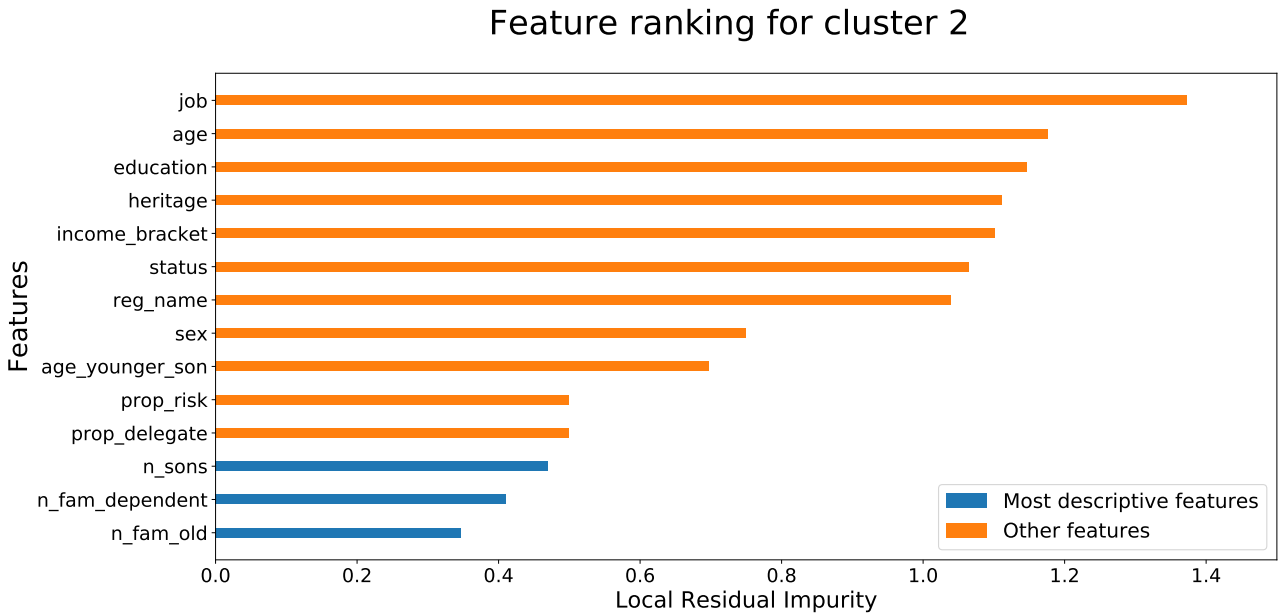


Figure 6.1: Example of feature ranking for a random cluster

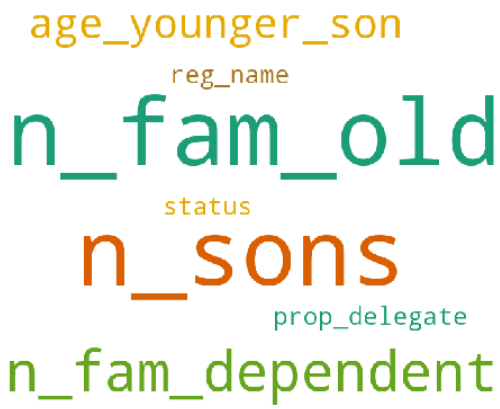


Figure 6.2: Word cloud with feature names across all clusters. The bigger the word, the more times the feature appeared relevant across clusters

7 Deploy of the pipeline and web app

7.1 Deployment of the pipeline

Deploy a Machine Learning pipeline means to put it into production where one can pass data to the model and get a prediction in return. To deploy the Pipeline defined in Chapter 5 we need to follow the general structure for a deployable function on IBM Watson, i.e, a Python closure. The structure is presented in the following code:

```
"""
'parameters' is a JSON that typically contains some configurations for
the pipeline (hyperparameters, etc.).
"""
def function_to_deploy(parameters):
    #Import packages
    #Create functions, variables etc.
    #Configure your pipeline
    #Train your algorithm/pipeline


    #'payload' is the data of new customers to be clustered
    #sent by the web app
    def score(payload):
        try:
            #Run the pipeline
            return {"results" : results}
        except Exception as e:
            return {"error" : str(e)}

    return score
```

During the deployment, IBM Watson runs *function_to_deploy*() once, then it creates an endpoint URL that will be used by clients to send a payload to *score*().

7.2 The web app

This work included the development of an Angular[1] web application, whose functionalities consists of providing, first, a GUI to insert information of new customers to be clustered (Figure (7.2)). Second, to provide a user-friendly interface to explore clusters and their relative most descriptive features (Figure (7.3)). Finally, to provide an easy interface to plot the clustered training set (Figure (7.4), but only if its dimensionality was reduced to 2D).

To allow the web application to access the data needed for Figure (7.3) and (7.4), a Firestore storage service was configured. Once the pipeline is deployed, the training set and the summary of the most descriptive features are stored on Firebase, to allow the web application to easily retrieve this information. Since the dataset can be very large, downloading the entire dataset every time that one needs to plot the data would take too long. Therefore to ensure a quick response by the web interface, the end of the training's timestamp is saved along with the data on Firebase. If the web application does not have a local timestamp or the local timestamp is older than the remote timestamp, the data is

[1] Open source framework to develop web application

downloaded and stored on the local storage. Finally the local timestamp is set to match the remote timestamp. If this is not the case, the web application retrieves the data previously saved on the local storage.

The web application does not communicate directly to the IBM back-end but, as anticipated in section 3, is mediated by a NodeJS proxy to overcome the IBM API's limitations on CORS. The proxy receives a payload by the web application, sends it to the deployed pipeline, and once it receives the result back the proxy responds to the web app, by adding to the response the CORS HTTP headers. Before sending a payload to the deployed model, the web application needs to obtain a IAM token (Identity and Access Management) to be authenticated. The request for this token is, again, mediated by the proxy.

The web application language is Italian, due to an explicit request by Blue Reply.

Figure 7.1: Homepage of the web application

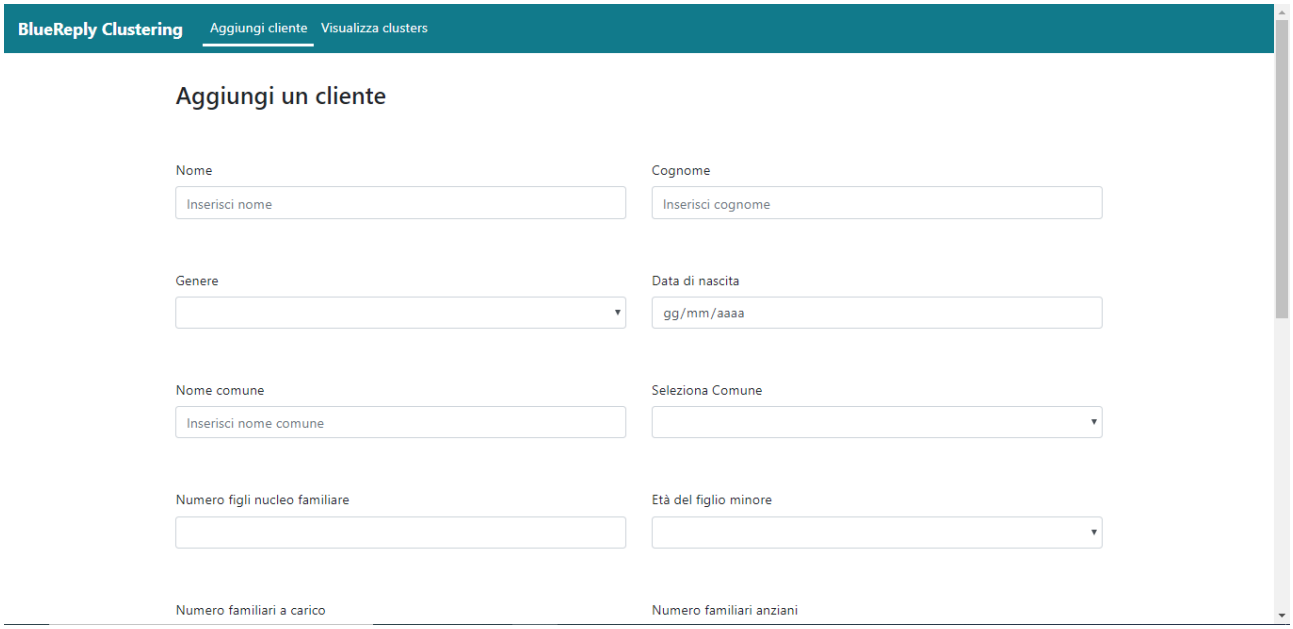


Figure 7.2: Form to insert the data of new customers

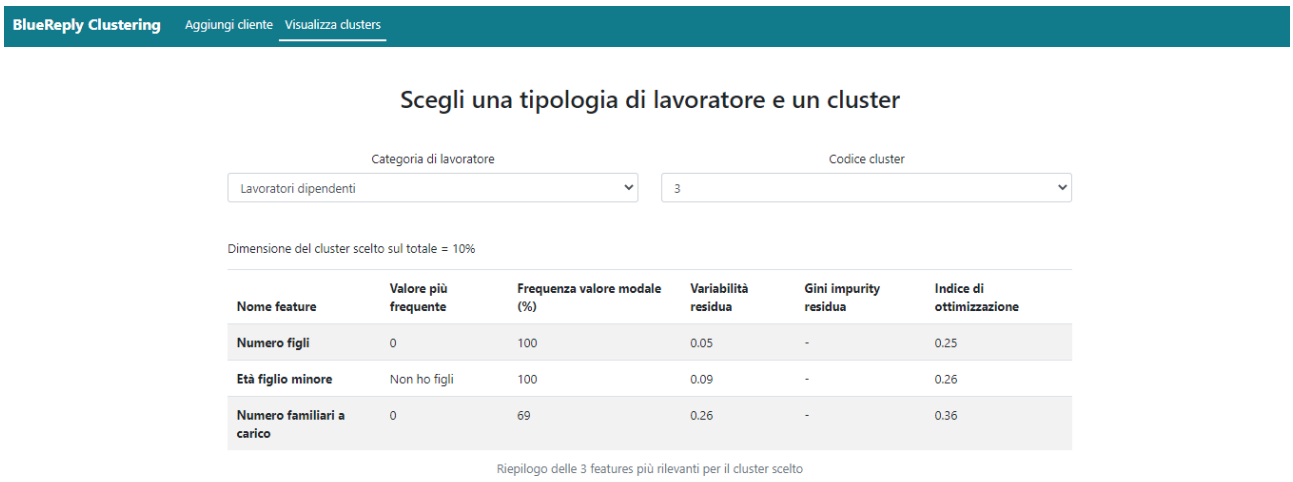


Figure 7.3: Summary of the most important features for every cluster. *Variabilità residua* is the plain L.R.I score , while *Indice di ottimizzazione* is the L.R.I score regularized by (6.10)

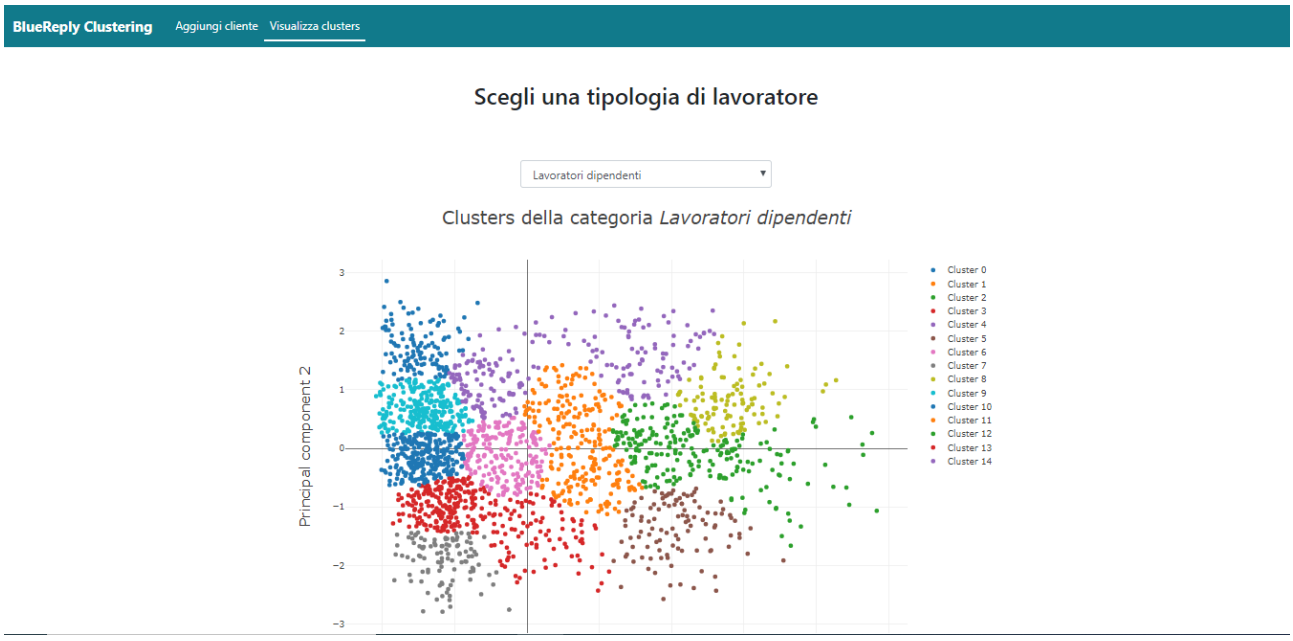


Figure 7.4: Plot of the training data

8 Conclusion and future works

The main goal of this work was to take the first step towards the development of new services for insurance companies, through the application of Machine Learning techniques to group customers into distinct marketing groups.

The main challenge of customers segmentation was to provide feedback, from an insurance point of view, of the composition of such clusters, in order to produce interpretable results. This was accomplished via the residual impurity score, an index that measures the residual impurity of each cluster and to highlights the most descriptive features for each cluster. This index was used also as an objective to minimize in a process of Bayesian optimization, with the aim of finding the best couple dimensionality reduction algorithm - clustering algorithm, with their relative hyperparameters.

Moreover, an Angular web application was developed to send to the deployed Machine Learning model the data of new customers and to allow easy exploration of clusters.

The generic and didactic approach adopted in this work was appreciated by Blue Reply. A new session of experiments with Machine Learning started after this work. This new experiment has the aim of grouping customers by analyzing their already purchased policies. Several other approaches are possible, such as, combining the personal information along with their past purchase to have a more detailed view of the customers' profile. This would also provide the possibility to investigate the correlation between personal information and the types of policies bought.

Recall that Machine Learning is a very broad subject, where clustering is not the only viable approach. In fact, in presence of a labelled dataset, many other algorithms can be tested. For instance, one could classify customers into different categories, predict churns, prevent or discover frauds, and develop a recommender system.

Bibliography

[1] ANIA. L'assicurazione italiana. https://www.ania.it/documents/35135/126701/LAssicurazione-Italiana-2018-2019.pdf/6975f9f6-77a4-985b-bcda-8fe835c55eee?version=1.0&t=1575543865117, 2018-2019. Accessed: 2020/04/30. Page 50.

[2] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

[3] Intermedia channel. Insurance report nielsen: 1,9 milioni di italiani hanno chiesto preventivi alle assicurazioni online. https://www.intermediachannel.it/2017/10/09/insurance-report-nielsen-19-milioni-di-italiani-hanno-chiesto-preventivi-alle-assicurazioni-online/, 2017. Accessed: 2020/04/30.

[4] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn Tensorflow*. O'Reilly, first edition edition, 2017.

[5] IVASS. I principali numeri delle assicurazioni in italia. https://www.ivass.it/pubblicazioni-e-statistiche/statistiche/numeri-assicurazioni/2018/Focus_I_Principali_numeri_2018.pdf, 2018. Accessed: 2020/04/30.

[6] IVASS. Relazione sull'attività svolta dall'istituto nell'anno 2018. https://www.ivass.it/pubblicazioni-e-statistiche/pubblicazioni/relazione-annuale/2019/Considerazioni_Presidente_IVASS_2018.pdf, 2018. Accessed: 2020/04/30.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2013.

[8] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, page 413–422, USA, 2008. IEEE Computer Society.

[9] A.A. Patel. *Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data*. O'Reilly Media, 2019.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Official sklearn's documentation for k-means. https://scikit-learn.org/stable/modules/clustering.html#k-means. Accessed: 2020/05/12.

[11] F. Shaheen and B. Verma. An ensemble of deep learning architectures for automatic feature extraction. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–5, 2016.

[12] Deloitte Touche Tohmatsu. Indagine sul mercato assicurativo auto in europa: L'ascesa delle polizze auto digitali. https://www2.deloitte.com/content/dam/Deloitte/it/Documents/strategy/161201_European%20motor%20study-HD-CMJN.pdf, 2016. Accessed: 2020/04/30. Page 15.