Forecasting Firm Failure via Random Forests and Logit

Mila Andreani, Lorenzo Bellomo, Andrea Pugnana,
Laura State, Jack Tacchi

May 14, 2021

Abstract

Starting from the work of [1], firm failure prediction has been extensively studied in both economics and statistical literature. However, only a few studies have proposed a Machine Learning approach. Therefore, this project aims to answer two main research questions: "Can clustering algorithms be used to find novel and useful ways of clustering Italian firms?" and "Can Machine Learning algorithms perform better than Logit at predicting firm failure?".

To address these questions, Clustering and Classification algorithms were performed on the AIDA dataset, which contains spreadsheet data of Italian firms.

Contents

1 Dataset and Preprocessing

1.1 Dataset

The dataset used in this project comes from AIDA database, which contains balance sheet data of Italian firms. In particular, there are yearly records for over 2 million companies from 2001 to 2019. The variables used in the analysis were:

- company name, ateco code, legal form, ID
- location (longitude/lattitude)
- account closing date (last entry)
- number of employees
- total payables + costs overall + retained earnings
- profits + revenues + total assets
- shareholder funds, working capital
- dichotomous variable (fail/no fail)

For each of the variables related to accounting data, there were also lagged columns for each of the five years before the last available year.
Regarding failed firms, the whole database contains roughly 110,000 firms that went bankrupt. Due to computational constraints and an imbalanced dataset in terms of failed firms, a subsample has been drawn from the original AIDA database as follows:

- Only firms that failed after 2011 were considered. This reduction resulted in a sample of around 20k firms
- In order to avoid issues arising from the imbalance, roughly 50k additional companies that did not fail were sampled. The final sample consists of roughly 70k observations

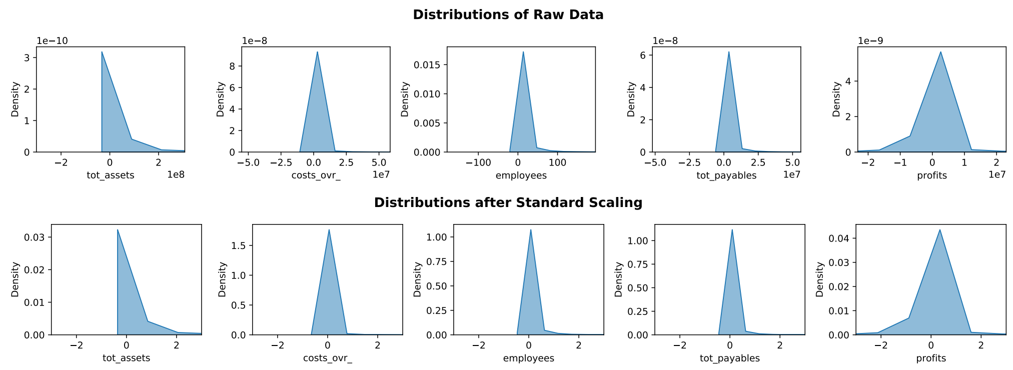


Figure 1: Kernel density estimates of the first 5 variables before (upper row) and after scaling (lower row).

- Finally, given the nature of data, all of the rows which contained any missing values for the selected features were removed. Missing value imputation has been discarded as it may have produced data inconsistent with balance sheet rules
- The final dataset consists of 37,328 rows, of which 35% are failed firms

1.2 Preprocessing

Each variable, except for longitude/latitude, ateco code and the account closing date, was normalised as follows:

$$\hat{x}_{ij} = \frac{x_{ij} - \overline{x}_i}{\sigma_i} \tag{1}$$

where columns are indexed by i and rows by j. Put simply, each variable individually had its mean $\overline{x}_i$ subtracted and was then divided by its standard deviation σ_i. This is also known as standard scaling.

For clustering, the scaling was done on the full dataset, whereas for classification it was performed on the training and test set independently. The resulting changes in the variables' distribution are shown in Figure 1.

2 Clustering

2.1 Introduction and Preprocessing

Clustering analysis was performed on balance sheet data only. Additionally, the dataset had been subsampled to improve the computational efficiency of the algorithms. For instance, for Hierarchical Clustering only 1,000 observations

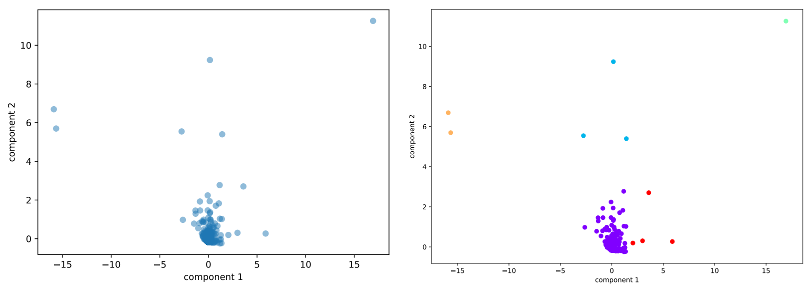


Figure 2: Left: Principal Component Analysis: the reduced data space of the first two principal components, with a ration of approx. 60% and 30% of variance explained. Right: Hierarchical Clustering with $n = 5$ (number of clusters), linkage function *ward*. Clusters are colour-coded.

were used. Only data from the latest available time point were taken into account, NaN values were dropped.
In addition, a Principal Component Analysis (PCA) was performed, either for data visualisation purposes only or before applying the clustering, as in the case of Hierarchical Clustering. Results are shown in Figure 2, left side. The data points form a dense cloud.

2.2 Hierarchical Clustering

Hierarchical Clustering crucially depends on the linkage function used to compute the distance between different clusters. Therefore, the clustering was performed using four different linkage functions: *single, ward, complete* and *average*.
Figure 3 shows the resulting dendrograms after performing clustering with the different linkage functions. The average silhouette scores computed for an increasing number of clusters and for the four different linkage functions are shown in Figure 4. The general trend suggests that the average silhouette score drops with a higher number of clusters. Given that the aim of the algorithm is to maximise the score function, a small number of clusters is preferred.
As a result, it is not possible for the Hierarchical Clustering approach to find meaningful subpartitions of the dataset, as the data forms a highly dense cloud. The best found clustering from the Hierarchical Clustering, with $n = 5$ clusters and the *ward* linkage function, is shown in Figure 2 Right.

2.3 KMeans

This section describes the kind of experiments that were run on the data using `KMeans` [5, 7] as a clustering algorithm. The results obtained and some plots are shown in section 2.3.

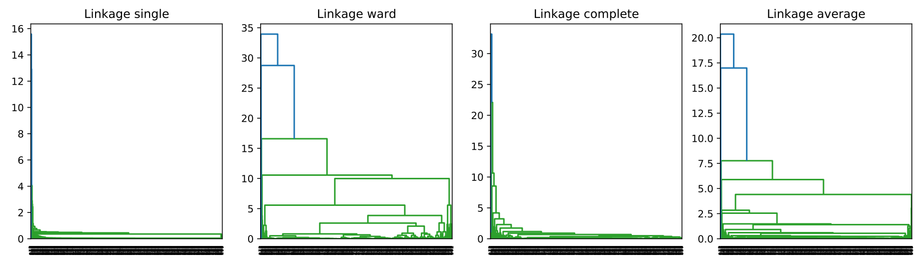


Figure 3: Hierarchical Clustering: Dendrograms for linkage function *single, ward, complete, average* (left to right).

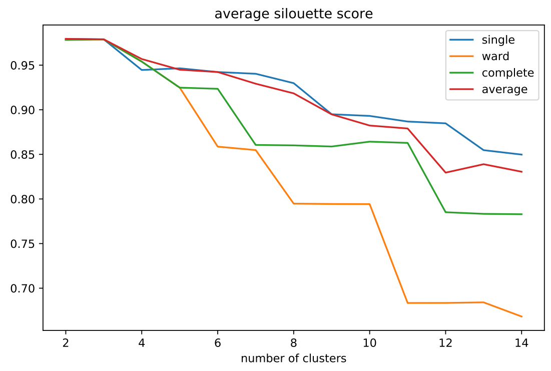


Figure 4: Hierarchical Clustering: Average Silhouette Score for different linkage functions and over an increasing number of clusters.

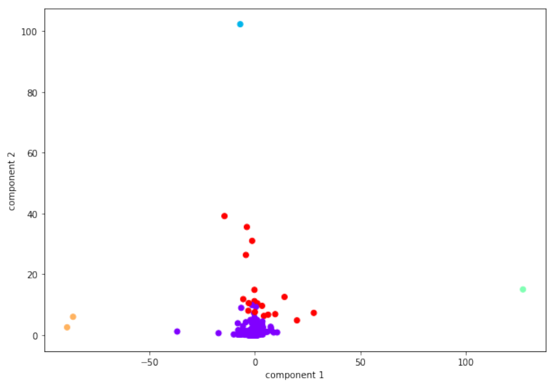
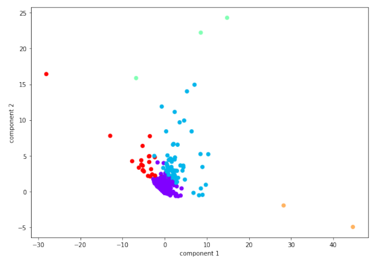
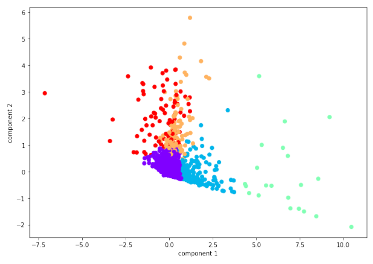


Figure 5: KMeans with $k = 5$



(a) Second layer of `KMeans`, $k = 5$



(b) Third layer of `KMeans`, $k = 5$

Figure 6: `KMeans` applications after the first phase of outlier removal

All the experiments were run on a random sample of $10,000$ items and using $k \in [2, 6]$.
The results obtained are plotted in Figure 5, showing the components derived from dimensionality reduction. Results are far from ideal, as the majority of the points are grouped in one cluster. Indeed, only about a dozen points are left out of the majority cluster and split into the remaining 4 labels.
Results do not vary significantly for different choices of k, thus only results for $k = 5$ are displayed. As a matter of fact, `KMeans` has been run on a dataset where no *outlier detection* was performed.

Afterwards, `KMeans` was used as a tool to perform outlier detection. Since most of the points are labeled as part of the majority cluster, `KMeans` was performed, recursively, three times on the majority cluster emerging from the previous step. The first, second and third `KMeans` layers are shown in Figures 6, 6a, 6b.
Although the proposed approach improves the results of the analysis, when inspecting the sizes of the five clusters at layer five the following is obtained:

- Cluster 1: 9541
- Cluster 2: 20
- Cluster 3: 34
- Cluster 4: 158
- Cluster 5: 201

After three "layers" of `KMeans`, a clear majority cluster emerges (blue in Figure 6b), along with several smaller ones. This approach highlights the inner structure of the data, and specifically, of the dense cloud of points, which gets denser and denser the deeper you inspect it.
This kind of data are by definition not a great fit for `KMeans`, which tends to work better when data are located in "blobs".

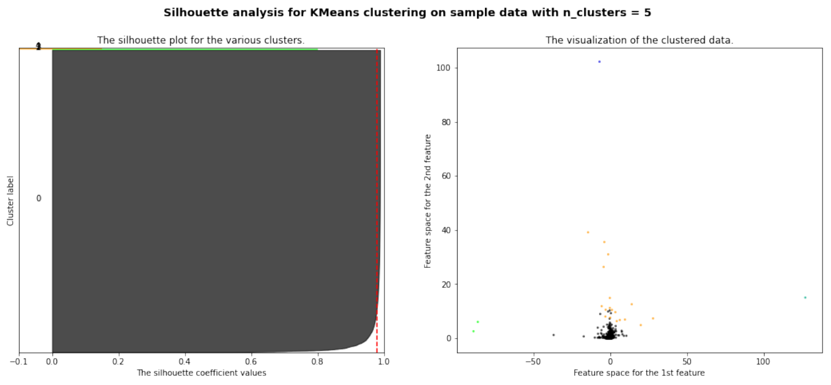


Figure 7: Silhouette Score, second outlier removal layer, $k = 5$

Figure 7 shows a plot displaying the *silhouette* score for a run of the algorithm. As expected, the *silhouette* score is an uninteresting metric for totally unbalanced cluster sizes, as the results are one-sided and completely shifted towards the majority

2.4 DBScan

`DBScan` [3] (*Density-Based Spatial Clustering of Applications with Noise*) is a clustering algorithm dating back to 1996.
The main aim of such algorithm is to form clusters out of areas with highly dense structures of data points. This algorithm is resilient to noise, and it can also identify *non-linearly separable* clusters. Moreover, the number of clusters is not a parameter.
A brief overview of the algorithm is shown in section 2.4.1, some algorithm techniques used for parameter estimation are shown in section 2.4.2. Section 2.4.3 shows the results obtained on the dataset.

2.4.1 The Algorithm

The main aim of the authors when designing `DBScan` was to provide an algorithm that required minimal knowledge of a dataset to obtain quality clustering results. Indeed, the main goal the authors was to provide an algorithm where the number of clusters was not a parameter, while simultaneously *aiding* the user in the selection of the two parameters that were required to run the algorithm.
`DBScan` works by dividing the data points in three groups:

- *Core* points: points in a dense area.
- *Border* points: points at the edge of a dense area.
- *Noise* points: either outliers or points that are in a sparser area.

To distinguish between those three types, `DBScan` employs two parameters:

- ϵ: radius of the neighbourhood of point
- **min_points**: minimum number of points required in the neighbourhood of a point to "make it part of a cluster"

Specifically, the first phase of the algorithm consists in a scan over all the data points, in order to label them with the three types described before. This labelling is performed as follows:

- A point p is a *core* point if at least **min_points** points are within distance ϵ of it (including p).
- A point p is considered a *border* point if it is in the neighbourhood of a core point (within a circle of radius ϵ), without satisfying the core point requirement itself.
- Any other point is labeled as *noise*.

After this preliminary step, a full labelling of the data cloud has been performed. The next step consists of actually identifying the clusters. This is done by building a neighbourhood graph, and by finding the connected components in it. Connected components (formed by *core* and *border* points) are put in the same cluster. This allows the algorithm to generate cluster labels without actually getting provided with the number of clusters to form.
This algorithm excels when the data points tend to have a close to uniform density, excluding noise points. `DBScan` is also resistant to non-linearly separable clusters.
The weakness of the algorithm lies in the fact that the density is "driven" by the two parameters. This means that results will not be satisfactory when data points are in clouds of different densities.

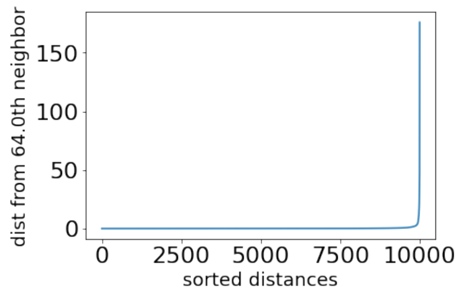


Figure 8: $min\_points = 64$, sorted distances w.r.t. 64-th neighbour

2.4.2 Parameter Estimation

The algorithm requires two parameters to run, but how are those chosen?
The idea is the following:

- Try several possible **min_points**, such as several power of twos.
- For each possible choice k of **min_points**, plot the sorted distances of each point from its k-th closest neighbour. This allows us to create a plot such as the one shown in Figure 8.
- Pick *epsilon* as one corresponding to the elbow of the curve. This separates points into those that will form a cluster and those that will be labeled as noise.

The plot shown in Figure 8 is far from ideal. This is due to the curve being extremely sharp, and not providing a clear *elbow*.
Other parameter choices yield similar results, which makes us assume that results will be non-ideal.
In order to still test the algorithm, several parameter configurations were also *brute-forced*:

- **min_points** $= 2^i \ \forall i \in [5, 10)$
- $\epsilon = 0.001 \cdot i \ \forall i \in [0, 70)$

Results were then checked for instances that formed at least two clusters of at least 50 points each.

2.4.3 DBScan Experiments and Results

In this section, the results obtained are presented. The approach proposed in this project specifically aims at performing clustering on the raw scaled data, whereas the components obtained via the dimensionality reduction are used only for plotting the data.

A sample of $10,000$ observations is randomly sampled from the original dataset and used to run the algorithm to improve the computational efficiency. Subsequently, the optimal configuration of parameters is found with a brute-force approach. In particular, the most interesting results are obtained by setting $min\_points = 64$ and $\epsilon = 0.9$.

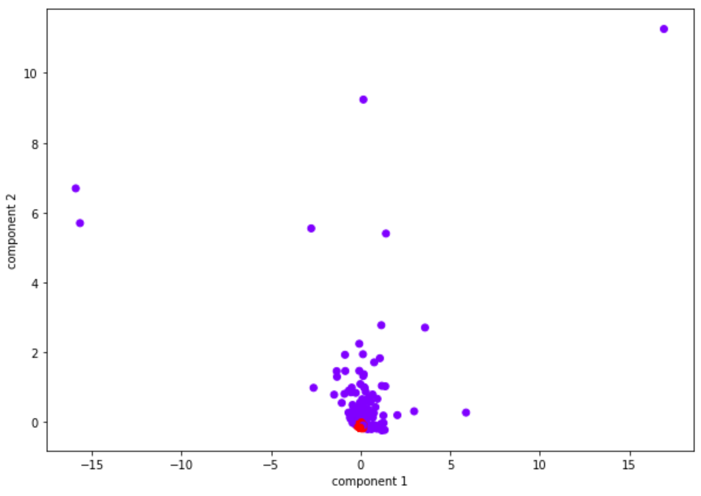


Figure 9: $min\_points = 64$, $\epsilon = 0.9$

This parameter configuration creates 4 clusters of sizes respectively $1,960$, 141, 415, 269. However, as shown in Figure 9, only the majority cluster is clearly visible (the red points), whereas the remaining data points tend to get lost in the noise.
This configuration of parameters also labels $7,215$ points as noise (displayed in pink in the Figure). This kind of result is, of course, a red flag for bad clustering results. This comes from the fact that `DBScan` is not built for clustering this kind of data.
By trying different values for the parameters, different results are obtained. For example, by lowering ϵ, more points are labelled as noise, whereas a bigger ϵ causes the algorithm to label most of the data points as belonging to one single cluster.

This result is coherent with findings of section 2.3, in which it has been shown that the data are distributed in a very dense cloud characterised by a non-uniform density. This kind of setting is exactly the one in which `DBScan` performs the worst, as the parameters are not resilient to varying densities.

2.5 Mean Shift Clustering

Mean Shift Clustering [4] is another non-parametric clustering algorithm that is tailored towards defining clusters based on the density of the data. Similarly to DBScan, it does not require a predefined number of clusters and is resilient to noise and *non-linearly separable clusters*.

Mean Shift will try to locate local maxima of density, the so-called “modes” of the data. Unfortunately, it can have trouble correctly classifying structures that have more than one mode per cluster.

2.5.1 The Algorithm

The Mean Shift algorithm is based on the repetition of a few basic steps:

- For a given point, calculate a ”Gaussian blob” with the point at the centre; this area is referred to as a *region of interest*.
- Calculate the centre of mass of the region of interest. As the area is defined as a Gaussian, the weighting of points closer to the centre will be larger, while those of the points at the extremity will be lower.
- Calculate the vector between the centre of the region of interest and to its centre of mass; this vector is called the *mean shift vector* and is where the algorithm gets its name.
- Shift the region of interest by the mean shift vector and re-calculate the centre of mass.
- Repeat the above steps until the size of the mean shift vector is below a certain threshold. When this happens, the algorithm will have located a mode within the date.

In order to cluster the data, the above steps are performed for every data point. Then, the points who ended up near the same mode are grouped together into *basins of attraction*; and these basins form the clusters. In this way, each cluster corresponds to a mode found within the data, with the final number of clusters being between 1 and the total number of data points.

2.5.2 Parameter Estimation

Although Mean Shift does not have a predetermined number of clusters, the amount of clusters it finds can be very sensitive to the choice of *bandwidth* [6]. Bandwidth determines the size of the region of interest: the larger the bandwidth, the more points will be used to calculate the centre of mass and the fewer the number of modes that will be found. This effect can be seen in Figure 10, where the number of clusters falls sharply until a bandwidth of around 0.3. Furthermore, in Figure 11, it can be observed that the silhouette scores of the clusterings appears to gradually plateau for bandwidths above 0.5. A bandwidth of 0.385 was found to be optimal for the current data.

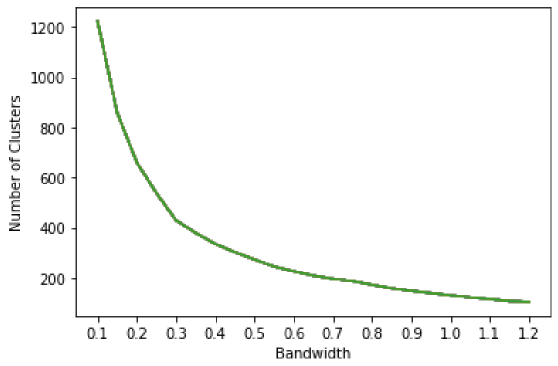


Figure 10: Plot of bandwidth against number of clusters

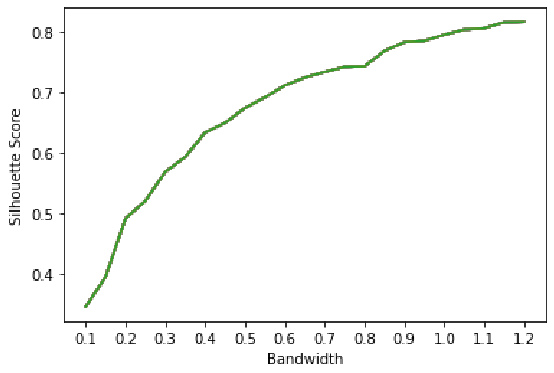


Figure 11: Plot of bandwidth against silhouette score

2.5.3 Mean Shift Results

The Mean Shift Clustering was run using the same subsample of $10,000$ normalised data points as the DBScan algorithm. Using the bandwidth of 0.385 that was found in the previous sections, Mean Shift found 347 clusters. However, once again, a single overly dominant cluster emerges with only a few small clusters surrounding it. Here, the largest cluster contains 8,991 data points, while the second and third largest contain 146 and 39 respectively. There are then 6 clusters that contain between 15 and 30. The number of outliers is also incredibly high: 283 of the 347 clusters (81.5%) contain only a single data point. If all of the outliers were put into a single cluster, that cluster would be the second largest.

Figure 12 displays a plot of the Mean Shift clusters. Although it is not obvious from the graph alone just how dominant the main cluster is, it is clearly larger and more dense than any of the other clusters.

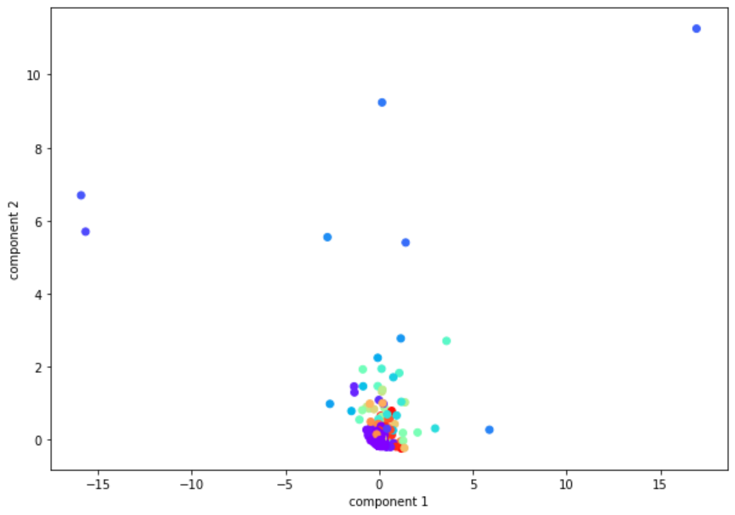


Figure 12: Mean Shift clustering, plotted with the first principal component on the x-axis and the second on the y-axis. The dominant cluster is displayed in purple

2.6 Clustering Results

After running the clustering with all four of the chosen algorithms, it was repeatedly found that the results were dominated by a single, massive, extremely dense supercluster. Indeed, this is further supported by the finding that the silhouette scores for the clusterings were consistently very high when a small number of clusters were present and consistently much lower for larger numbers of clusters; suggesting that additional clusters only served to artificially split the supercluster. Furthermore, even when examining only the centre of the supercluster, the same results are obtained: concentric clusters of increasing density with no meaningful partitions.

One possible explanation for this come from the data, which was taken from Italian firms. Businesses in Italy are predominantly small-medium enterprises and these types of firms tend to have very similar spreadsheet profiles and thus appear very close to one another within the data set.

Regardless, no novel and meaningful ways of grouping the firms was found. However, it is possible that the presence of the extremely dense supercluster was obscuring other patterns in the data. It may therefore be pertinent for future researchers to remove said cluster if they wish to examine the rest of data for novel findings.

3 Classification

3.1 Random Forest

Random Forests is a method introduced for the first time in [2]. In particular, random forests are an ensemble algorithm aimed at mitigating the bias-variance trade-off of decision trees. They are vastly used in several applications due to their relative simplicity and their performance advantages over traditional decision trees. The algorithm is described as follows:

- as a first step, the training set is bootstrapped;
- l features out of the k features available are randomly selected;
- a decision tree is grown on the bootstrapped training set using only the selected features;
- this procedure is repeated for B times, growing B trees;
- as a final step the predictions of each tree are then averaged in order to obtain the final predictions.

One of the main advantages of Random Forests is that, differently from other machine learning algorithms, they require only a few hyper-parameters to be tuned. In this sense, Random Forests represent a competitive tool with respect to traditional parametric and non-parametric statistical models. In this work six different Random Forests were trained on six different set of features. For each of them, the hyper-parameters have been selected by running a Grid Search Cross Validation over 5 folds of the training set. The parameters that were tuned are the following:

- the loss function used by each tree to determine the splits, i.e. Gini and cross-entropy;
- the number of trees B. 100, 200, 500, 1000, 2000 trees were tested;
- he number l of features randomly sampled by the original set of k features. The criteria were either the $log_2(k)$ or $\sqrt{k}$ as suggested in the majority of the literature.

The final combination of hyper-parameters is the one delivering the best performing random forest in terms of accuracy.
It is worth highlighting that in this setting accuracy is a reasonable choice to test the performance of the classifier, while in extremely imbalanced situations accuracy might be a misleading metric.

3.2 Methodology

3.2.1 Lasso, Ridge Regression, Elastic Net and Logit

One of the aims of this project is to compare the predictive accuracy of Random Forests against Logit, a parametric statistical model largely used in the literature for classification tasks. Starting from the dataset described in Section 2.1, the independent variable is modelled as:

$$Y_i = f(\mathbf{X}_i, \mathbf{Z}_i)$$

where $Y \in [0, 1]$. Referring to the work of Altman [1], the vector of covariates $\mathbf{X}$ is composed of:

- X_1: Working Capital/Total Assets
- X_2: Retained Earnings/Total Assets
- X_3: Earnings before Interest and Taxes/Total Assets
- X_4: Market Value of Equity/ Total Debt
- X_5: Sales/Total Assets

and where $\mathbf{Z}$ consists of balance sheet data variables.

In order to tackle the issues arising from multicollinearity and the curse of dimensionality, a two-step approach is applied to the original dataset. The first step consists of eliminating lags and correlated features from the original dataset of 62 covariates to avoid multicollinearity. Figure 13 shows the heat map (or correlation matrix) of the 9 final, selected features.
The resulting set of 9 features is then used in the second step to obtain a Logit model as parsimonious as possibile. Specifically, Lasso, Ridge Regression and Elastic Net were run in order to select the most important features. A 5-fold cross validation was used to optimise the parameter λ of each model.
Eventually, none of the above-mentioned methods eliminated any variables, so the final set of covariates in the Logit model is:

- Shareholders' Funds
- Total Payables
- Profits
- Working Capital
- Earning before Taxes over Total Assets (EATA)
- Revenues
- Revenues over Total assets (RVTA)

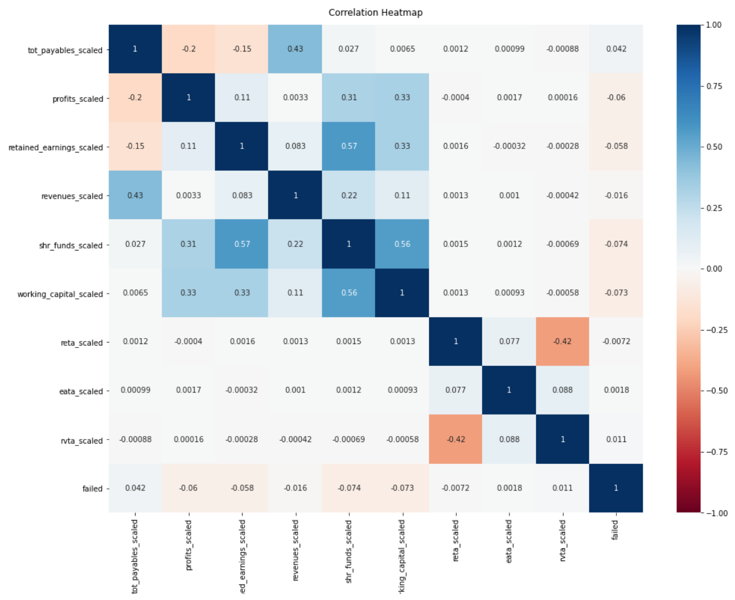


Figure 13: Correlation matrix of the final set of 9 features used in the Logit model.

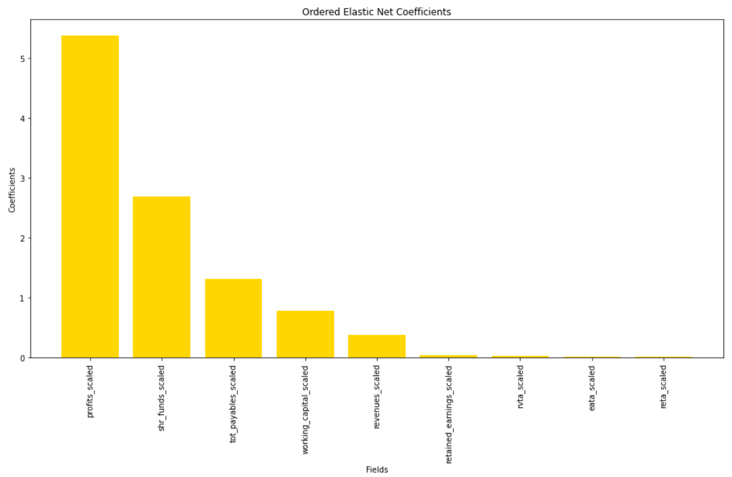
- Retained Earnings

The coefficients related to each variable do not vary drastically across the four models (Lasso, Ridge Regression, Elastic Net and Logit). For instance, Figure 14 displays the coefficients for Elastic Net and Ridge Regression. In both cases, the coefficients are in the same order in terms of weight.

3.2.2 Random Forests Variable Importance

Several specifications of Random Forests have been run, one for the sake of comparability and the others to improve performances. Each specification concerning different variables has been separately tuned using the Grid Search CV described previously. In particular, 6 different sets of features have been used to train the Random Forest:

- the complete set of features
- the top 10 features (in terms of variable importance) (Figure 15)
- the top 5 features (in terms of variable importance) (Figure 15)
- the ratios from [1]
- the ratios from [1], the firms' ATECO code and the dimension categories
- the 9 features used in the Logit model



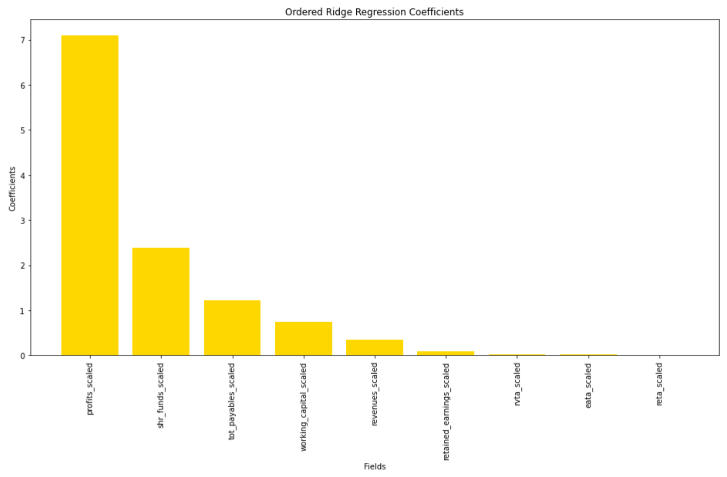


Figure 14: Coeficcients for Elastic Net and Ridge Regression

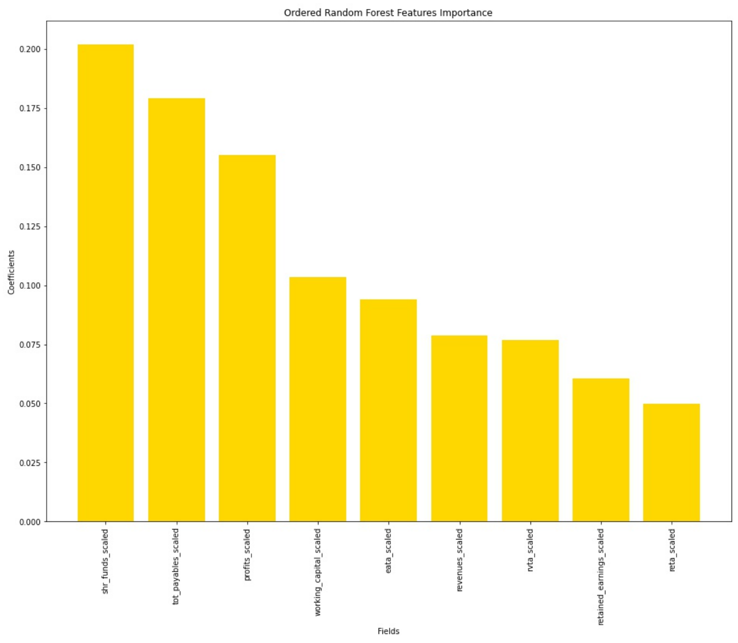


Figure 15: Variable importance computed using the complete set of features. The variable importance plot shows a different order of features to the one identified through the classical statistical models from the previous section.

Model	AUC	Accuracy
All features	0.81	0.83
Top10	0.78	0.81
Top5	0.77	0.80
Ratios+Categories	0.79	0.82
Logit Specification	0.78	0.81
Ratios	0.78	0.81

Table 1
Random Forests results

Model	AUC	Accuracy
Random Forest	0.78	0.81
Logit	0.70	0.78
Lasso	0.71	0.78
Ridge	0.71	0.78
Elastic Net	0.71	0.78

Table 2
Logit and Random Forests results

3.2.3 Results

The results obtained with the 6 different specifications of the Random Forest are reported in Table 1.
Results show that the Random Forest trained with the complete set of features is the best performing model in terms of both AUC and Accuracy. However, the rest of the specifications, trained with a smaller number of features, deliver similar results. This is due to the fact that many of the 62 features are correlated: even though Random Forest can definitely handle multicollinearity better that parametric models, using a big number of correlated variables results in a computationally inefficient model.
Finally, we compare the results of the Logit model and the Random Forest trained with the Logit covariates. As shown in Table 2, the Random Forest outperforms the Logit model in terms of both Accuracy and ROC. As a matter of fact, the non parametric approach offered by the Random Forest delivers an improvement of 8% and 3% in AUC and Accuracy, respectively.

4 Conclusion

This project highlights two main empirical results. First, the clustering algorithms do not extract any significant patterns from the Italian firms dataset, due to the nature of the data. Second, good interpretability and the small number of parameters required to be tuned make Random Forests particularly suitable for practical applications, such as firm failure prediction. In particular, Random Forests outperform Logit, the latter being one of the most used statistical models for classification tasks. In this sense, this Machine Learning algorithm presents a useful tool for uncovering complex patterns in data.

References

[1] Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The Journal of Finance*, 23(4):589–609.

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

[3] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press.

[4] Fukunaga, K. and Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. pages 32–40. IEEE Transactions on information theory.

[5] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[6] Ren, Y., Kamath, U., Domeniconi, C., and Zhang, G. (2014). Boosted mean shift clustering. pages 646–661. Springer-Verlag Berlin Heidelberg.

[7] Steinhaus, H. (1956). Sur la division des corps matériels en parties. *Bulletin de l'Académie Polonaise des Sciences*, Cl. III — Vol. IV(12):801–804.