Bankruptcy Prediction

DS5220: Supervised Machine Learning Project Report

Viral Pandey
MS in Data Science, 1st Year

Camellia Debnath
MS in Data Science, 1st Year

I. Abstract

The financial health of any company is reflected in its revenue, net and gross profits, liabilities and various other econometric measures. Those measures, in turn, can be used to make judgments regarding how the company is performing in the market, and also how they might perform in the future. In this report, we analyzed the results for Bankruptcy Prediction for Polish Companies in the emerging markets. The predictions were based on 64 econometric ratios published by those companies, and we compared various supervised machine learning classification models for this task, starting from the simple Logistic Regression, Naive Bayes to the more complex Neural Network. Quadratic Discriminant Analysis gave the best performance when it came to predicting correctly.

II. Introduction

The purpose of this project is to identify the financial condition of a company based on the historical data of successful and not so successful companies. This can give meaningful insights on what direction a company is headed. To achieve something like this, we can build a mathematical model based on different numerical indicators of the existing companies and how they are performing in the market and use it to evaluate the financial health and the future of the company. Machine Learning Algorithms have improved so much in recent years especially for classifying which class does a data point belong to. Using these classification algorithms, we can classify a company and identify the factors which affects significantly to the financial health of a particular company.

A. About the Data

The Dataset used for this project is the Polish companies bankruptcy data Data Set [1]. The dataset is about bankruptcy prediction of Polish companies. The data was collected from Emerging Markets Information Service, which is a database containing information on emerging markets around the world. The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.
Basing on the collected data, five classification cases were distinguished, that depends on the forecasting period:

- 1st Year: The data contains financial rates from 1st year of the forecasting period and corresponding class label that indicates bankruptcy status after 5 years. The data contains 7027 instances (financial statements), 271 represents bankrupted companies, 6756 firms that did not bankrupt in the forecasting period.
- 2nd Year: The data contains financial rates from 2nd year of the forecasting period and corresponding class label that indicates bankruptcy status after 4 years. The data contains 10173 instances (financial statements), 400 represents bankrupted companies, 9773 firms that did not bankrupt in the forecasting period.
- 3rd Year: The data contains financial rates from 3rd year of the forecasting period and corresponding class label that indicates bankruptcy status after 3 years. The data contains 10503 instances (financial statements), 495 represents bankrupted companies, 10008 firms that did not bankrupt in the forecasting period.
- 4th Year: The data contains financial rates from 4th year of the forecasting period and corresponding class label that indicates bankruptcy status after 2 years. The data contains 9792 instances (financial statements), 515 represents bankrupted companies, 9277 firms that did not bankrupt in the forecasting period.
- 5th Year: The data contains financial rates from 5th year of the forecasting period and corresponding class label that indicates bankruptcy status after 1 year. The data contains 5910 instances (financial statements), 410 represents bankrupted companies, 5500 firms that did not bankrupt in the forecasting period.

The data set is multivariate in nature and has 64 econometric ratios (Table: II). As one can see, companies can be repeated in these different data-sets and since there is no unique identification of the companies across the datasets, it does not make sense to combine all the data. Hence, we fit all the models separately on each data set.

III. Methods

This project was carried our in various steps such as Exploratory Data Analysis, Data Pre-processing, Model Fitting, and finally, selecting the best model. Data Pre-processing consisted of splitting the data-set into train, test and validation sets, taking care of missing values, trying out several imputation techniques, taking care of data imbalance, and taking care of correlation among the features. The next step consisted of fitting various supervised machine learning classification models on the processed training data, and using the validation set for k-fold cross-validation for hyper-parameter tuning. Finally,

we compared the various models based on their performance on the part of the data set we had kept aside for testing. All these methods are described in detail below.

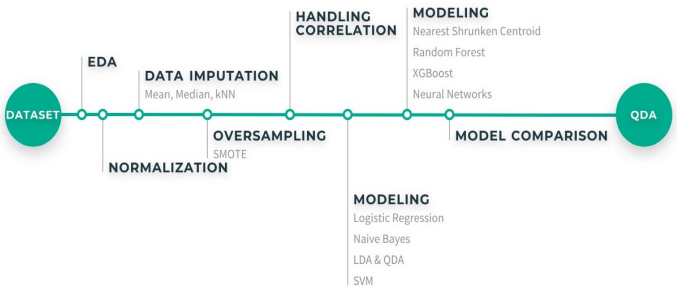

Fig. 1. Timeline

A. Exploratory Data Analysis

We performed exploratory data analysis on the training data set to get a better idea about the nature of the data. One of the striking characteristics of the data-set was the data imbalance between the majority and the minority classes, i.e., the data-set had significantly more rows corresponding to companies which had not gone bankrupt, compared to the ones which did. We can see this from the following plot.

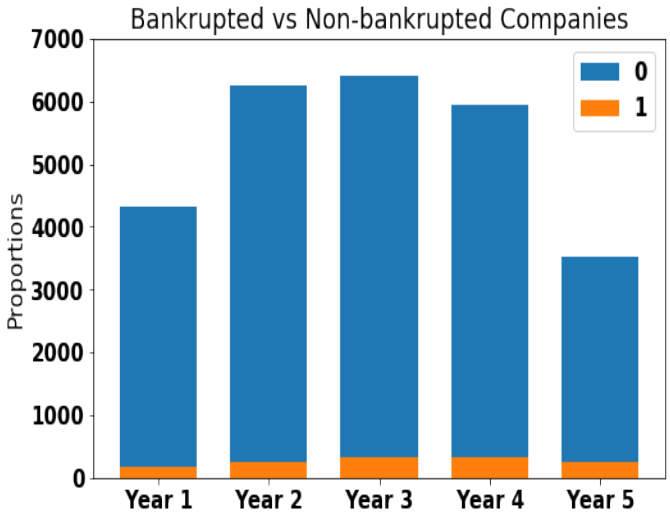

Fig. 2. Data Imbalance

In this figure, blue represents the companies which did not go bankrupt, and orange represents the ones which did. As we can clearly see, the data set is highly skewed. According to our analysis, Year 1 data-set has 3.856% , Year 2 has 3.932%, Year 3 has 4.713%, Year 4 has 5.260% and Year 5 has 6.937% of minority class data-points.

Our analysis also shed light on the fact that this dataset consists of a lot of missing values. We plotted a sparsity matrix for a visual representation of the missing values across the columns and rows. The sparsity matrix for the 1st Year dataset is as follows:

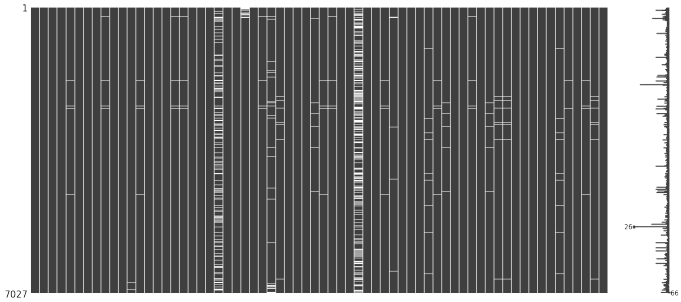

Fig. 3. Missing Values across the dataset

This sparsity matrix is a visual representation of the data-set for the first year, containing 7027 entries, and 64 attributes. The white horizontal bars represent the missing values in the data-set, as we can clearly see, there are many missing values in the data-set. To be precise, the percentage of rows containing missing values for the data-sets from years 1 through 5 are as follows: 54.55, 59.81, 53.48, 51.29 and 48.71. Our Exploratory data analysis also revealed that there were presence of highly correlated features in the data-set. We found that the features which correspond to Attributes 13, 19, 20, 23, 30, 31, 39, 42, 43, 44, 49, 56, 58, 62 had high linear correlation among them. The definition of these attributes can be found in the Appendix (Table: II).

B. Data Preprocessing

As the first step of Data Preprocessing, we first normalized our training and test data-set so that each of 64 variables of our data-set were in the range of 0 - 1. Our Exploratory Data Analysis revealed some of the issues we needed to take care of before fitting, namely, missing values, data imbalance and existence of correlated features. To begin with, we tried different data imputation techniques to handle the missing values, such as mean imputation, median imputation and k-Nearest Neighbours (k-NN) imputation. Mean Imputation technique involves replacing missing values with the mean of the existing values for that feature. Similarly, Median Imputation involves replacing the missing values with the median of the existing values for that feature. k-NN involves replacing the missing values by considering its k nearest neighbours in the feature space. Out of all the imputation techniques, we found that Mean Imputation performed relatively better than other techniques on all our models.

Next we faced the issue of handling data imbalance, i.e., the disparity of representation between the two classes of data. There are several ways to handle data imbalance. One way is to draw a random sample from the majority class data which is similar in size to the minority class. But the minority class being very small in size and also the overall size of

the data being not that large, undersampling would greatly reduce the size of the training data-set, which is not good for modeling purpose. Other option we have is to oversample the minority class by creating artificial data points of that class.

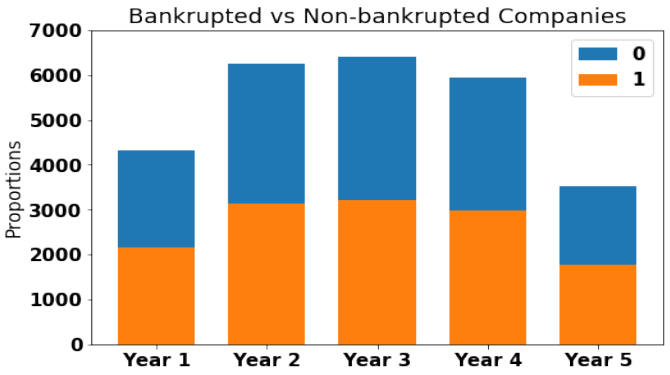

Fig. 4. Oversampling the minority class using SMOTE

To oversample the minority class, we decided to use Synthetic Minority OverSampling Technique (SMOTE) on our training data-set. SMOTE is an over-sampling approach in which the minority class is over-sampled by creating synthetic examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen [2]. We have used default value of 5 nearest neighbors.

Before implementing any kind of Supervised Machine Learning Classification Algorithms on our data-set, it is important that there should be no multi-collinearity i.e. the variables should not be highly correlated with each other. To identify such variables, we calculated correlation matrix and extracted all the attribute pairs which either had correlation coefficient of 0.5 and larger or -0.5 and smaller. This gave us Attributes 13, 19, 20, 23, 30 31, 39, 42, 43, 44, 49, 56, 68 and 62. These attributes were highly correlated and if you look at Table: II, you can see that all these econometric ratios are similar. Hence we ended up keeping only Attribute 13 and dropping others to remove collinearity in the data-set.

C. Model Fitting

1) Logistic Regression: Logistic Regression is a very popular supervised machine learning classification method. The elegance of Logistic Regression lies in the simplicity of its implementation and interpretability. The logistic regression model can be represented as follows [3]:

$$y = \sigma(w^T x) \tag{1}$$

Where $\sigma(.)$ is the logistic sigmoid function. Here y is the probability of a class as given by this equation, and w and x are the weight and feature vectors respectively.

We used the `LogisticRegression` function of Python `scikit-learn` library to implement this. Since Logistic Regression was one of the very first models we implemented, we experimented with various versions of it. We implemented Logistic Regression on our training dataset, first without any sort of oversampling, and then after oversampling. There was only a very slight improvement on performance on test data after oversampling. We also tried regularization for feature selection. We tried both L1 and L2 regularization. L1 enforces linear penalty, while L2 implements quadratic or euclidean penalty. for our Logistic Regression models, using oversampled training data. L1 performs slightly better than L2. However, Logistic Regression had an overall very poor performance on the test dataset [Table: I].

2) Naive Bayes: Naive Bayes is a classification method which combines Bayes Theorem of conditional Probability and the assumption that all the features are independent of each other [3]. Bayes Theorem is given as:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \tag{2}$$

Where $p(C_k|x)$ is the conditional probability distribution of the k^{th} class C_k given the features x, $p(C_k)$, $p(x)$ are the prior probabilities of the class C_k and the features x, and $p(x|C_k)$ is the conditional probability distribution of x. The Naive Bayes Classification formula is given as follows:

$$\hat{y} = \underset{k\in 1,\ldots,K}{\operatorname{argmax}}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{3}$$

Where $\hat{u}$ is the predicted class, and K is the total number of classes. Naive Bayes can also make assumption about the nature of distribution of the features. For our modeling purpose, we assumed that the data has a Gaussian Distribution, i.e. ,

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(v-\mu_k)^2}{2\sigma_k^2}\right) \tag{4}$$

Where μ_k and σ_k^2 are the mean and variance of the values in the feature x associated with the k^{th} class.

For fitting Gaussian Naive Bayes model, we used the `GaussianNB` function of the Python `scikit-learn` library. We fit the model to our oversampled training data-set. Also we fit two different models. First, we fit a Gaussian NB model to the training data-set without dropping any of the correlated features, which did not give us any good performance. Since Naive Bayes makes inherent assumption that the features are independent, we tried fitting another model to the training data-set after dropping the correlated variables. But we did not achieve any significant improvements [Table: I]. We think there could be two possible reasons behind this: first, even though we removed the correlated features from the training data-set, there could still exist features in the data-set having non-linear dependency. Correlation just takes care of linear dependency. The other possible reason could be the underlying assumption

of Normality. We assumed that the features were normally distributed, which could quite possibly may not be the case.

3) Linear and Quadratic Discriminant Analysis: Linear and Quadratic discriminant analyses(LDA and QDA) are classification methods which work by projecting features of higher dimensions into a lower dimensional space [3]. Linear Discriminant Analysis is the special case when we assume that the individual features have equal covariance matrices. For ease of understanding, lets consider the case of two classes: we take a D-dimensional input vector x and project it down to one dimension using

$$Y = w^T x \tag{5}$$

where w is the weight vector.

We place a threshold on y and classify $y \geq -w_0$ as class C_1 and otherwise class C_2. This can be done in the following steps: first we calculate the between class variance, or the distance between means of different classes, and then we find the within class variance, signifying the distance between the mean and samples of each class. Then we construct the lower dimensional space which maximizes the between class variance and minimizes the within class variance.

We use the `LinearDiscriminantAnalysis` and `QDA` functions from Pythons `scikit-learn` library to implement LDA and QDA models. We used the oversampled training data-set for this purpose. We found that LDA performs worse than QDA for all the 5 years data-sets [3]. QDA, infact, performs significantly better compared to LDA. This behavior is quite expected since LDA assumes that the individual features have identical covariance matrices. Since this is not the case in practice, the low performance of LDA is not surprising. QDA does not make this assumption, and hence, and this is reflected in the better performance.

4) Support Vector Machines: Support Vector Machines are supervised machine learning classification techniques, which are good for separating classes which are not linearly separable. The idea behind this classification technique is to find the maximum margin hyperplane (or line, in case of a 2-dimensional plane) in the p-dimensional feature space which separates the classes. Support Vector Machines use different Kernel Functions, which projects the non-linear non-separable input space into a higher dimensional linear separable space.

We used the `SVC` function of the Python `scikit-learn` library to fit SVM models to our oversampled training data using linear and polynomial kernels (degrees 2, 3 and 4). SVM failed to perform well when classifying the test data. Moreover, we did not notice much difference in the performance of the different kernels [Table: I]. However, the performance actually worsens when we increase the polynomial degree to more than 3.

5) Nearest Shrunken Centroids: Nearest Shrunken Centroids (NSC) are another type of methods for classification, where classify the data by shrinking the class-wise mean toward the overall mean, for each feature separately. The result is a regularized version of the nearest centroid classifier, or equivalently a regularized version of the diagonal-covariance form of LDA [4].

We used the `NearestShrunkenCentroids` function of the Python `scikit-learn` library to fit the NSC models to our oversampled training dataset. For the shrinkage parameter, we used k-fold cross validation to find the optimum shrinkage threshold. Similar to most other models (Except QDA), this model performs poorly on the test dataset [Table: I].

6) Random Forest: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub samples of dataset and uses averaging to improve the predictive accuracy and control over fitting. The sub-sample size is always the same as original input size but the samples are drawn with replacement unless specified. We used the `Random Forest` from `scikit-learn` library to fit on the oversampled training data [5]. The hyperparameters here were the number of estimators and maximum depth of the tree. We used hyperparameter tuning on the cross validation set where the number of estimators and the depth were selected randomly from the set [100, 150, 200, 250, 500] and depth from [10, 25, 50, 100] respectively. The optimum values were 100 and 25 for which the model was developed. Random Forest performance is average as the fit is not excellent but not very poor either.

7) Boosting: XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning. It is an advanced form of gradient boosting algorithm with features for regularization, parallel processing and built in cross validation [6]. We used XGBoost instead of the other popular algorithm AdaBoost as it performs for generic loss functions as compared to AdaBoost which is derived mainly for exponential loss functions. We used `XGBoostclassifier` of `XGBoost` library to fit the data on the oversampled training set.. The hyperparameters here were the maximum depth and L2 regularization penalty. We used hyperparameter tuning on the cross validation set where the penalty and the depth were selected randomly from the set [0.1, 0.2, 0.5, 0.7, 0.9] and [5, 10, 20, 50, 100] respectively. The optimum values were 0.5 and 10 for which the model was developed. XGBoost performs a decent fit on the model and gives good predictions.

8) Neural Networks: Neural Networks are a class of supervised learning algorithm that learns a function mapping by training on given data. It learns by minimizing the objective loss function

$$L(W) = -\sum_{i=1}^{N} [Y_i \log \pi_i + (1 - Y_i) \log(1 - \pi_i)] \tag{6}$$

Here, W is the weight matrix and

$$\pi = \frac{1}{1 + \exp\left(-(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p)\right)} \quad (7)$$

The weights are updated by calculating the predicted labels and hence calculating the errors. The error is then used to calculate the gradients which are used to update the weights by backpropagation [3]. We use `MLPClassifier` of `Scikit-Learn` for fitting on the oversampled training data. The hyper parameters in this case were number of layers, number of neurons, non-linear activation function, optimizer, L1 regularization penalty and learning rate. The hyperparameters were chosen using 64 random trials on cross validation set.. The number of layers, number of neurons, activation function and optimizer were drawn randomly from their respective sets with uniform probability. The learning rate and L1 regularization penalty were both drawn geometrically with the former choosing from .0001 to .1 and the latter choosing from 1e-7 to 1e-2. Drawing geometrically from a set A and B means drawing uniformly in the log domain between log(A) and log(B), exponentiating to get a number between A and B. The optimal model had the following hyper parameters: number of layers = 1, number of neurons = 10, non-linear activation function = relu, optimizer = adam, L1 regularization penalty = 1e-5 and learning rate = 0.01. Surprisingly, Neural Network performs very poorly and is not able to make good predictions.

IV. Results

Since our data-set is imbalanced, accuracy would not have been a good measure of performance on test data-set. Since even if we classified every single instance as the majority class, we would still get an accuracy of 96% or more on our test data-sets. Hence, we used Precision, Recall and F1 scores for our performance measurement. Precision is the ratio between the total number of True Positives and the sum of True Positives and False Positives. Recall is the ratio between the total number of True Positives and the sum of True Positives and False Negatives. F1 score is simply the harmonic mean of Precision and Recall.

We observed that most of the models performed best on Year 5 test dataset. The reason for this may be that Year 5 itself being a smaller dataset compared to the other years, the data imbalance proportion was lesser. We observed that Quadratic Discriminant Analysis performed the best among all the models. Quite contrary to popular belief, Neural Networks failed to perform well on this particular dataset. The reason behind this could be attributed to the fact that neural networks require huge amount of data to perform well, but the size of our datasets ranged from only 5000 to 10,000 instances per year. Also, Neural Networks are complicated models that require extensive hyper parameter tuning, which was out of scope for this project. As expected, plain Logistic Regression and Linear Discriminant Analysis fail to capture the complex non-linear decision boundary of our dataset. We get slightly better performance from XGBoost and Random Forest, but even those are not encouraging enough to use them for prediction. It is surprising that in spite of various existing state of the art techniques, LDA and QDA still perform better than many of the models. A likely reason is that the data can only support simple decision boundaries such as linear or quadratic, and the estimates provided via the Gaussian models are stable. [4].

V. Discussion

The striking nature of the data-set that we observed was the fact that it is hard to find trends in the data-set, i.e., most of our supervised machine learning models failed to work. There could be various reasons behind this. For example, whether a company will actually go bankrupt depends on many factors other than just the econometric ratios, such as change in market conditions, company management, competitors etc. We did not have any such predictors available. However, from the data-set that did have available to us, we realized that its possible to do feature selection, and therefore dimensionality reduction, if one possesses a good domain knowledge. For example, our highly correlated variables were mostly related to ratios involving sales and profits, which make sense intuitively. Also, observing the feature weights of regularized logistic regression, we found that higher weights were given to features indicating the profits and liabilities of a company, and we know that whether a company will go bankrupt or not is largely depend on its profits and its liabilities too. Future scope for predicting bankruptcy could involve working on data-sets which have more features available with regards to market conditions and company characteristics.

References

[1] Zikeba, Maciej and Tomczak, Sebastian K and Tomczak, Jakub M, "Ensemble Boosted Trees with Synthetic Features Generation in Application to Bankruptcy Prediction," Expert Systems with Applications, Elsevier, 2016.

[2] N. V. Chawla and K. W. Bowyer and L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal Of Artificial Intelligence Research, Volume 16, pages 321-357, 2002, 10.1613/jair.953.

[3] C. M. Bishop, Pattern Recognition and Machine Learning, 1st Ed, 233 Spring Street, NY: Springer, 2006.

[4] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Data Mining, Inference, and Prediction, 2nd Ed, CA: Springer, 2008.

[5] sklearn.ensemble.RandomForestClassifier . Available: http://www.scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[6] Jason Brownlee ,How to Tune the Number and Size of Decision Trees with XGBoost in Python, 2016/09/06. Available: https://www.machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/

APPENDIX

TABLE I
MODEL PERFORMANCE

Models	Precision	Recall	F1 Score
Year 1			
Logistic Regression	0.53	0.69	0.49
Naive Bayes	0.10	0.50	0.10
LDA	0.74	0.53	0.53
QDA	0.66	0.91	0.72
SVM	0.51	0.51	0.04
Nearest Shrunken Centroids	0.49	0.5	0.49
Random Forest	0.77	0.70	0.73
Boosting	0.81	0.79	0.81
Neural Networks	0.53	0.72	0.43
Year 2			
Logistic Regression	0.52	0.62	0.44
Naive Bayes	0.50	0.48	0.14
LDA	0.64	0.94	0.70
QDA	0.62	0.86	0.67
SVM	0.49	0.5	0.05
Nearest Shrunken Centroids	0.48	0.5	0.5
Random Forest	0.70	0.64	0.67
Boosting	0.74	0.69	0.71
Neural Networks	0.54	0.68	0.47
Year 3			
Logistic Regression	0.52	0.61	0.48
Naive Bayes	0.52	0.51	0.10
LDA	0.65	0.94	0.70
QDA	0.65	0.91	0.71
SVM	0.51	0.51	0.04
Nearest Shrunken Centroids	0.48	0.5	0.49
Random Forest	0.68	0.67	0.68
Boosting	0.75	0.74	0.75
Neural Networks	0.53	0.65	0.32
Year 4			
Logistic Regression	0.52	0.69	0.48
Naive Bayes	0.51	0.51	0.08
LDA	0.79	0.57	0.6
QDA	0.64	0.90	0.67
SVM	0.54	0.51	0.08
Nearest Shrunken Centroids	0.81	0.51	0.51
Random Forest	0.69	0.66	0.67
Boosting	0.71	0.68	0.69
Neural Networks	0.55	0.64	0.54
Year 5			
Logistic Regression	0.58	0.78	0.78
Naive Bayes	0.60	0.57	0.58
LDA	0.75	0.53	0.53
QDA	0.83	0.96	0.87
SVM	0.54	0.51	0.08
Nearest Shrunken Centroids	0.81	0.51	0.51
Random Forest	0.76	0.76	0.76
Boosting	0.79	0.81	0.80
Neural Networks	0.56	0.68	0.54

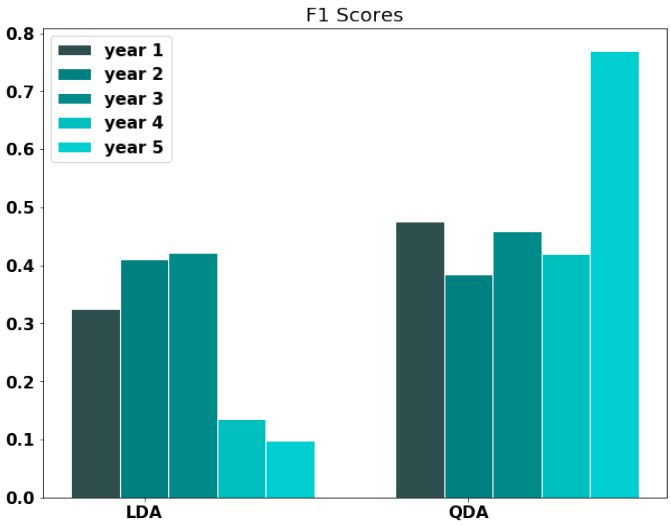

Fig. 5. F1 Scores for LDA and QDA

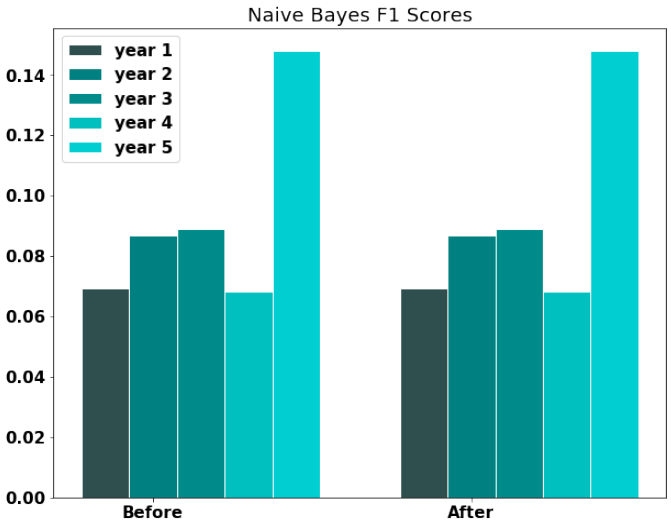

Fig. 6. F1 Scores for Naive Bayes before and after removing pairwise collinearity

TABLE II
DATA SET DESCRIPTION

Attribute	Description
X1	net profit / total assets
X2	total liabilities / total assets
X3	working capital / total assets
X4	current assets / short-term liabilities
X5	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
X6	retained earnings / total assets
X7	EBIT / total assets
X8	book value of equity / total liabilities
X9	sales / total assets
X10	equity / total assets
X11	(gross profit + extraordinary items + financial expenses) / total assets
X12	gross profit / short-term liabilities
X13	(gross profit + depreciation) / sales
X14	(gross profit + interest) / total assets
X15	(total liabilities * 365) / (gross profit + depreciation)
X16	(gross profit + depreciation) / total liabilities
X17	total assets / total liabilities
X18	gross profit / total assets
X19	gross profit / sales
X20	(inventory * 365) / sales
X21	sales (n) / sales (n-1)
X22	profit on operating activities / total assets
X23	net profit / sales
X24	gross profit (in 3 years) / total assets
X25	(equity - share capital) / total assets
X26	(net profit + depreciation) / total liabilities
X27	profit on operating activities / financial expenses
X28	working capital / fixed assets
X29	logarithm of total assets
X30	(total liabilities - cash) / sales
X31	(gross profit + interest) / sales
X32	(current liabilities * 365) / cost of products sold
X33	operating expenses / short-term liabilities
X34	operating expenses / total liabilities
X35	profit on sales / total assets
X36	total sales / total assets
X37	(current assets - inventories) / long-term liabilities
X38	constant capital / total assets
X39	profit on sales / sales
X40	(current assets - inventory - receivables) / short-term liabilities
X41	total liabilities / ((profit on operating activities + depreciation) * (12/365))
X42	profit on operating activities / sales
X43	rotation receivables + inventory turnover in days
X44	(receivables * 365) / sales
X45	net profit / inventory
X46	(current assets - inventory) / short-term liabilities
X47	(inventory * 365) / cost of products sold
X48	EBITDA (profit on operating activities - depreciation) / total assets
X49	EBITDA (profit on operating activities - depreciation) / sales
X50	current assets / total liabilities
X51	short-term liabilities / total assets
X52	(short-term liabilities * 365) / cost of products sold)
X53	equity / fixed assets
X54	constant capital / fixed assets
X55	working capital
X56	(sales - cost of products sold) / sales
X57	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
X58	total costs /total sales
X59	long-term liabilities / equity
X60	sales / inventory
X61	sales / receivables
X62	(short-term liabilities *365) / sales
X63	sales / short-term liabilities
X64	sales / fixed assets