Machine Learning in Finance Group Project Report

Laurin van den Bergh, Julia Bingler, Michael Hodel, Julian Müller

April 21, 2020

1 Introduction

Credit ratings are an important determinant of financial decision-making. Usually, a variety of factors are taken into consideration when assigning a credit rating to a specific company. Furthermore, in addition to present and near past credit ratings, future credit ratings are of high importance for investors. As these are not observable, one relies on models to make predictions on the future credit rating. In the present approach, we considered a variety of machine learning algorithms to predict the long-term credit rating of S&P 500 companies.

2 Data Preparation

2.1 Data Sources

Since the task given left quite some room for interpretation and for experimenting, when downloading the data, we were faced with several questions, such as "Which additional variables could be good predictors?" and "How can we make sure we get enough data to train the models effectively?". The latter led to the idea of gathering data for more companies than just the ones listed on the S&P 500 on the 29th of June 2018, as specified by the list SP500_CompanyList.csv on OLAT, as this was the list we took as the basis. For all datasets but CRSP we decided to download all variables, since we always have the opportunity to delete variables later. We downloaded the ratings as suggested on OLAT.

We decided to use the CRSP database as a foundation for our data, since this is a huge and well organized database of US equities, including the S&P 500 companies. Despite the fact that only the latters are the ones for which we need to predict the credit

ratings, we first used the entire CRSP company universe to extend the database for the algorithms. The monthly CRSP data also gave us access to some additional variables, such as "number of shares outstanding" and "stock price", which we used to calculate the market capitalization for each security.

We checked that all S&P 500 companies were included in the CRSP dataset. Using the PERMNO we could verify that all but two companies are included. The two companies that were not included turned out to have been added to the S&P 500 mid 2017, which is after the period we had to consider for the project (January 2010 - January 2017). Therefore we did not consider those two companies further. Thus we confirmed the S&P 500 stocks to be a subset of our data. We exported two lists of unique identifiers (CUSIP and PERMNO) from the CRSP data to download other data from WRDS for the corresponding firms. We created a binary indicator to indicate if a firm belongs to the S&P 500. An issue here was that despite using the same Python and Pandas versions, converting True and False to 1 and 0 using the .astype(int) function yielded panda data type 'int32' for non-mac users but either 'uint8' or 'int64' for mac users, which became a problem later. When fitting our first models we quickly realized that the non-S&P 500 data differs too much from the S&P 500 data, even when screening for large market capitalizations (an approach we chose since S&P 500 companies are selected that way). Thus for all models we proceeded with the subset containing only S&P 500 firms.

Other than the mandatory data on financial ratios on firm level and firm fundamentals, we also included data on financial ratios on industry level, inflation and on treasury bill returns with maturities 30 days, 90 days and one year, as they might be important, since interest rates and lending money are two very interconnected topics. It turned out, however, that interest rates and inflation were no relevant predictors, probably since they were almost constant during the time period from 2010 to 2017.

2.2 Merging

When merging the different datasets we decided on performing 'inner merges' on the date and the available security identifier. This means that at every merge, we took the intersection of the datasets regarding the tuple [date, identifier], which as a key specifies the observation. These inner merges allowed us to keep the numbers of missing values per variable at a minimum, but came at the cost of drastically reducing the dataset. Since we started with a large dataset, this was a compromise we were willing to take, since most of the keys matched anyway, resulting in a reasonably sized final dataset of 49362

observations. When merging the industry level data we used the sector codes provided by CRSP additionally to the date to match the datasets, instead of the security identifier. An issue with the merging was that the credit ratings file contained the 9-digit CUSIP, but the financial ratios file contained only the 8-digit CUSIP. Thus we had to convert the 9-digit CUSIP codes into 8-digit CUSIP codes by removing the last digit, which is a check digit.

2.3 Missing Values

2.3.1 Imputation

First, we dropped all observations with null values as response variables, because imputation would not really make sense here. Since most machine learning algorithms don't work with null values, we had to either get rid of observations and / or variables which exhibit null values - which is obviously undesirable, since one loses data and therefore information that could be valuable for the models - or estimate them. Accordingly, we first estimated the ones we could with a reasonable approach. We decided that it would be reasonable to first linearly interpolate each variable for each company along the time. We did this for each null value that has at least one preceding and at least one following non-null value. This interpolation removed only a minority, namely 11.4%, of the null values. For the remaining null values for which there was at least one null value for the given variable and company, we backward- and forward-filled the null values along the time. This filling removed another 29.3% of null values. The great majority of null values, namely 62.7%, thus came from variables for which certain companies exhibit only null-values and thus could not be estimated by either interpolating or backward- and forward-filling.

2.3.2 Dropping Variables and Observations

Given that the majority of the null values still remain, we had to either use other estimation methods and / or drop variables and / or companies until there are no null values left. Since we have a lot of data and the first approach would require more complex estimation - e.g. replacing null values by taking data from 'similar' companies (whereby one could for example determine the similarity between companies by the smallness of the euclidean distance between them in a vector space after normalizing) - and assumably yield less realistic estimations, we decided for the dropping. Now there is a trade-off

between the number of of variables to drop and the number of companies to drop: The fewer the number of variables that would be dropped - given, that one drops the variables for which there are the most companies with null values - the greater the number of companies that would have to be dropped afterwards to yield no remaining null value, and vice versa. The graph depicting the variables for which at least one company exhibits only null values, sorted by the number of companies with only null values for this variable, shows approximately an exponential decay. We decided for a threshold of 200 companies which demanded for the worst 16 variables to be dropped, which we did. After that, we dropped all companies that still had at least one remaining null value. With this approach, we managed to keep many companies and the majority of the variables (91.75 %) after imputation.

2.4 Feature Engineering

In order to get better results but also to be able to apply certain algorithms in the first place, we had to do feature engineering. The main parts of feature engineering were dealing with categorical variables, adding lags and changes and transforming our response variable. Further, we deleted redundant variables, since they yield no additional information. One problem we ran into in this section was that we had a variable for the company location, named 'loc', that could therefore not be accessed using the dot notation due to the conflict with the pandas .loc[] function and that we therefore renamed.

2.4.1 Categorical Variables

Since most machine learning algorithms don't work with non-numeric predictor variables, we had to either get rid of them or convert them to numeric 'dummy' variables. When looking at all the categorical variables, a great share of them exhibited a lot of categories. In order to not inflate the already big number of variables in our data frame, we decided to drop all categorical variables with more than 12 categories. Luckily, many of those variables that we dropped because of this reason are either company identifiers or variables of geographical nature and should not have too much predictive power in the first place.

2.4.2 Lagged Variables

Another crucial part of feature engineering is to generate lagged variables. Lagged variables allow to account for the fact that past developments might influence today's out-

comes. There are multiple factors credit agencies take into consideration when assigning a credit rating. One of them is the past history of borrowing and paying off debts. Debt has also a big influence on ratings, which is why we added lagged variables for most relevant ratios involving debt. We differentiated monthly, quarterly and annual lags. Since debt service is an important aspect regarding credit ratings, we added monthly lagged variables for debt service variables. Furthermore, the amount of debt compared to equity and further financial indicators are important indicators of financial health of a company and how much value would be lost in the case of a default. Since the financial health of a company is usually assessed quarterly, we added variables lagged by three months accounting for these aspects. Finally, for a further outlook, it is necessary to operate with larger lags, which is why we added twelve-month lags for fundamental variables, which usually don't change drastically in a short period of time. With this measure we also take possible seasonalities into account. Furthermore, since financial ratios are usually more meaningful when compared across institutions or when looking at their change, we calculated the relative change of the most important financial ratios over time instead of simply using the lagged variables. A change in financial ratios could to some extent hint on how a company will perform in the future, or at least influence how credit rating analysts might think that a company will perform in the future, which is decisive for the rating decision. One problem here was that lags introduce null values and infinity values. We fixed this problem by adding the lagged variables before dealing with the missing values.

2.4.3 Response Variable

We have an ordered response variable. Thus, we also want to treat it as such, since e.g. the difference between an AA+ rating and an AA rating is smaller than the difference between AA+ and B- . The credit rating, our response variable, also exhibited a large number of categories, namely 23. To be able to look at the distribution of the credit rating, we transformed it into an ordered categorical variable. The distribution looks approximately normal. Since the worst categories, the ones from CCC down to D, had only very few observations, we decided to group them into the category "CCC or lower". The reasoning behind this is that very infrequent categories might actually decrease performance due to the additional categories, which could be wrongly predicted for observations. Given that the junk status for credit ratings starts at BB, we think that binning the worst categories should not severely impact the usefulness of the model. However, we have to be aware

that binning the worst categories could bias the model accuracy to our favour.

2.5 Feature Importances

Since we have a lot of variables in our cleaned data set and some models work better with fewer or only the most important variables, we decided to create one additional data set where only a fraction of all the features are kept, namely the most important ones regarding predictive power. For that, we decided to use a simple random forest classifier, extract the estimated feature importances and select the variables based on those feature importances. When plotting the feature importances, the importance decay was less exponential and more linear than expected. What was especially striking was that the outstanding shares and the correlated market capitalisation seem to by far be the most important of the given predictors. It was also great to see that, as we expected, some of the created change and lag variables ranked higher in terms of feature importance than the variable they were based upon. Also notable is the fact that the industry level variables are grouped very strongly together. For our decimated data set, we decided to keep all the features that have an importance greater than 0.005, which is about when the group of the industry level variables starts.

3 Algorithms

3.1 Approach

After we had finished exploring our data sets, doing feature engineering and null value handling and had a data set ready on which we could apply the classifiers, we decided that each of us would get familiar with two classification algorithms that she / he thinks could perform well and are appealing. That way, we looked into logistic regression, random forests, support vector machines, LDA / QDA and k-nearest neighbors. After the first results, we focused on the best models.

3.2 Performance Metrics

We decided to only consider the accuracy metric to assess model performance. The reason is that other common multi-class performance metrics such as precision and recall are not that interesting in our context, since our goal is to accurately predict as many observations as possible. A metric like recall only gives us an idea of what fraction of each

respective class can be correctly classified by the algorithm, whereas precision tells us how likely a certain prediction actually is correct. In our example, where we classified 18 distinct classes, which would result in 18 different performance measures for each metric. This is an amount too large to simply interpret and a critique on the model would be extremely difficult and subjective, since weighing each of the 18 measures is not trivial nor is there a definitive answer to that. Simply weighing them at 1/18 each also defeats the purpose, since there is no way to interpret the resulting average in a way that is beneficial to an investor. So we opted for the most intuitive and simple to understand metric 'accuracy'.

3.3 Benchmark

To be able to know how well we actually do with our models, it was necessary to determine the naive classifier. The share of correct predictions when making predictions without applying machine learning can then serve as a benchmark. If we have a model that performs insignificantly better or even worse than this benchmark, we know that the model is not good. This benchmark is determined by the relative frequency of the most frequent response, which in our case is the rating BBB. The naive classifier would thus always predict BBB and would predict correctly around 14.11% of the time. Interestingly, the observations with the 5 most frequent ratings make up over half of all the observations and the observations with the 10 most frequent ratings make up over 90%.

3.4 Best Approach: Random Forests

3.4.1 First Results

We first used the entire CRSP dataset. We did a manual split of the training and testing data sets to ensure that only SP500 companies are in the test set, as required in the project outline. For that we used StratifiedShuffleSplit, splitting the SP500 data and stratifying it. We split 50:50 such that the final test set makes up about 20% of the full dataset. One part of the 50:50 split we upsampled to give more weight to those observations, and merged back into the non-SP500 data. Even though it is not fully stratified, it is at least a bit closer to being stratified.

Unfortunately and against our initial intuition, given the very bad accuracy of the random forest when considering all companies for training, namely 40%, it looks like the companies not in the S&P 500 don't help the algorithm to predict the S&P 500

companies and actually worsen its performance. Thus, we decided to only look at the S&P 500 companies.

3.4.2 Hyperparameter Tuning

Considering only S&P 500 companies for training, we could then regularly split the training and the testing sets. After dropping all non-S&P 500 companies, some variables exhibited only one value, which we dropped since they contain no information. We used a pipeline and k-fold cross validation. As already expected, random forests turned out to perform very well - already with the default parameters. The cross validation suggests an expected accuracy on the test data of about 96-97%. All estimates seem to lie in that range. Also, the confusion matrix didn't suggest major miscategorizations, since most were just one or two categories off, which is fine given that credit ratings involve some degree of subjectivity and ratings are ordered. Then, we performed hyperparameter tuning using grid search, which yielded the best parameters 'class_weight' : None, 'min_samples_leaf' = 1, 'min_samples_split' = 2 and the corresponding best accuracy of 96.66%.

3.4.3 Final Model

We then created our final model where we only use the S&P 500 companies and the parameters determined by the hyperparameter tuning. We first check the fit on the training data again to compare it to the default model and quantify the expected increase in accuracy. Then, we predict the test data to get the final score of the model. The prediction accuracy of our final model on the test set is 97.175%. Interestingly, the vast majority (88.3%) of the false predictions are only one category off and the remaining false predictions are all two categories off, yielding no prediction which was more than two categories off, which is remarkable. Thus, the prediction can be considered a very good estimate of the true rating.

3.4.4 Forecasting

Initially, there was some confusion about if we should use random train-tests splits which are independent of the time or use the last part of our time series as the test set. The first case allows far better performance but is further apart from a real-world use case. Thus, we decided to also do forecasting with our best performing algorithm in the end. Since the non-S&P 500 data didn't help when training the previous model, we only considered

past S&P 500 data for the forecasts. We used an appropriate splitting of the test and training data, such that the last 20% of the dates are in the testing data set and the first 80% are in the training data set. Using default parameters, we only managed a mean accuracy on the validation sets of around 21%, which is insignificantly better than our benchmark. Thus we again performed parameter tuning using grid search, which yielded a final forecasting accuracy on our test set of 76.51%.

3.5 Other models

3.5.1 Approach

We applied various other classifiers as well, namely LDA and QDA, logistic regression, adaptive boosting, gradient boosting, neural networks, support vector machines, k nearest neighbors and decision trees. Here, we will discuss the most important approaches. In our notebook 'other_models.ipynb', we uniformly performed hyperparameter tuning using grid search to extract the best parameters and the corresponding prediction accuracy.

3.5.2 Models

The other classifiers yielded some very interesting results. The neural networks for example yielded with 18.78% the worst performance of all classifiers we tried, against the assumption that neural networks generally perform well. Also, we didn't manage to beat the default parameters by using a simple grid search. The bad performance of neural networks is probably best explained by the fact that they are very sensitive and thus are very prone to overfitting and also the fact that we just lack the understanding of the theory behind them to be able to properly tune them.

Probably most interesting is the k nearest neighbors algorithm. Using grid search for hyperparameter tuning and cross validation on the k nearest neighbors classifier with the full data set yielded already a fairly good prediction accuracy of 88.98%. However, as the algorithm is sensitive to predictors with very low predictive power, it yielded far better results when using only the data set with the top features that we created using a random forest classifier. Performing hyperparameter tuning on the smaller data set yielded a prediction accuracy of 96.72%, which almost beats our random forest and can thus be considered our second-best approach.

Also notable are the performances of the support vector machines (94.5%), of the gradient boosting (93.7%) and the simple decision tree (92.2%). The other approaches

performed significantly worse, with logistic regression yielding ca. a 43%, LDA yielding ca. a 37% and adaptive boosting yielding ca. a 29% prediction accuracy.

4 Discussion

In this section, we discuss the biggest issues we faced in the process and how we solved them, how we evaluate our approach and which open questions remain.

4.1 Runtime

Runtime was probably what limited us the most. For example, we faced rather long runtimes for code with collection membership tests using lists. In those cases, we could greatly decrease the runtimes when the order was irrelevant by using sets instead, which use hash tables, which are way more efficient. Runtime was especially a problem when it came to hyperparameter tuning using grid searches. Therefore we decided to comment out the grid searches and just leave the classifiers with the best parameters, such that the notebooks can be run in a reasonable time. The results of the grid searches can be found in the gridsearch_results folder. Generally, we managed to solve quite a few problems more elegantly and efficiently using different approaches or already built-in functions, for example sklearn-functions.

4.2 Evaluation

By dropping companies that for some variables exhibit only null values, which can be outliers and thus constitute observations that might be harder to predict, we might have achieved better accuracy than if we had left these companies in the dataset. Also, as already mentioned, binning the worst categories could also introduce a bias to our favour.

4.3 Further Research

Some aspects would have benefited from more time. If we could, we would probably create additional data sets with different numbers of predictors and also do more extensive hyperparameter tuning. Also, we would be interested in exploring additional predictor variables that we have not considered so far and more extensive feature engineering.