Introduction to Machine Learning (Seminar)

Group Project Summary – Group 1

Zurich, April 18, 2021

Tim Ehrensperger, 18-719-062, tim.ehrensperger@uzh.ch
Marco Heiniger, 18-733-824, marco.heiniger@uzh.ch
Pascal Huser, 10-927-671, pascaljosef.huser@uzh.ch
Marc Tschudi, 18-729-269, marc.tschudi@uzh.ch

Contents

List of Figures

List of Tables

1 Data Preprocessing

For preprocessing, as well as for model building, we let us guide by the structure provided by the recommended textbook 'Python Machine Learning' by Raschka (2015). Furthermore, we followed the idea of having the data reveal its secrets through systematic examination of it.

As the original data consisted of separate yearly data sets and the financial indicators contained were periodically measured values, we decided to first examine these data sets one by one to get a feel for their similarities and differences. A first analysis step showed us the differences in the number of observations among the yearly data sets, Figure 1. It was due to these findings, we decided to add a 'Year' column (our feature engineering) to account for the yearly characteristics of each data set, which had to be considered later, when all data sets are merged for the machine learning task.

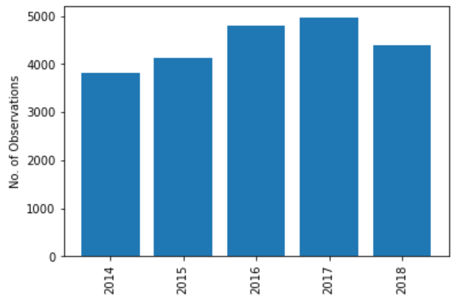

Figure 1: Distribution of yearly observations

Following this, we implemented the encoding of the 'Strategy' class labels, which allowed us to get a first impression of the class label distribution among our data sets, Figure 2. It can easily be seen that there are strong class imbalances in each data set, which are consistent over the different years.

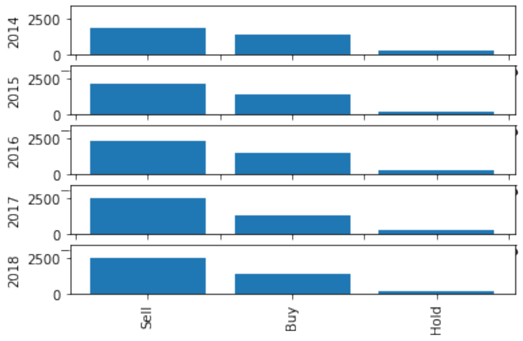

Figure 2: Distribution of yearly strategies

1.1 Zero and NaN Values

A first examination of zero and NaN values led to the discovery of unhelpful and duplicate features, which immediately got removed. Further comparisons of zero and NaN feature percentages over time led us to remove further features with percentages over thresholds of 50% and 40%, respectively. The threshold difference results due to our strong believe, that a zero value has a higher information content than a NaN value. In a similar fashion we took on the task of checking observations for NaN values, where we decided to get rid of rows with NaN value content higher than 75%, which led to deletion of 11% of our initially 22'077 rows.

1.2 Outlier Handling

We decided to perform outlier handling per sector per year, in order to take into account the specific treats of every sector and the macroeconomic environment the companies were exposed to. We decided to replace data points above or below a certain quantile threshold (0.05 and 0.95) with the respective quantile value. We used this outlier handling method because we strongly believe that within a sector, differences between companies present in the S&P 500 index will not heavily differ from each other.

1.3 Imputation of Missing Values

Consistent with our outlier handling, also the imputation of missing values was performed on a per sector per year basis. Due to this deep focus on the subparts present within every yearly data set and the handling method chosen for outlier handling, we decided to impute NaN values by using the median value. A last zero values check showed us, caused by the imputation before, a few additional features to drop.

1.4 Partitioning

For the partitioning of the complete data set into training and test sets, we decided to work with a test size of 30%, a value in the range suggested by Raschka (2015). Additionally, we worked with the stratify parameter to ensure maintaining the class label proportions at levels found in the original data set.

1.5 Class Imbalance

As we already saw before, there was a strong class imbalance in our data set which needed to be handled. We decided to balance our training set by using the SMOTE (Synthetic Minority Over-Sampling Technique) algorithm, which does not simply duplicate already existing observations for oversampling but synthesizes new data points out of already existing ones (Brownlee (2020)).

1.6 Feature selection

The last big step before starting to train our data set was feature selection. We decided to work with the Random Forest feature selection algorithm, which led us, after extensive evaluation cycles, to 75 features by using a threshold of 0.0055 relative feature importance. Being aware of the fact that 75 features to train a model is a high number, and therefore results in a complex model, we nevertheless are confident about it, because of the structured approach we used to select the features to be dropped.

2 Algorithms and Results

We trained our models using the following algorithm groups: LDA & QDA, Decision Tree Random Forest, Support Vector Machine and Keras Neural Network. Subsequently, there's a short presentation of every algorithm and the obtained results.

2.1 LDA and QDA

Linear Discriminant Analysis estimates for every class of the k possible classes the probability, by which a new input belongs to that class. It finally assigns the input to the class, for which the estimated probability is the highest, under the assumption that all k classes share the same covariance matrix. Since the method of LDA is widely used among machine learning tasks with multiple classes and features, we trained one of our models using this approach. The achieved score of approximately 57% was neither bad nor outstanding high.

Quadratic Discriminant Analysis (QDA), which is similar to LDA, differs from LDA by relaxing the strong underlying assumptions and allowing class specific means and covariance matrices. This makes it the preferred choice for problems which lack a common covariance matrix. But since there is one in our task and this is reflected in the accuracy score of the QDA method which is by far the worse than the one of the LDA model.

2.2 Random Forest and Decision Tree

Decision Tree and Random Forest are, due to their intuitive classification process and their flexibility on input data, two of the most used algorithms for classification problems (Raschka, 2015). Both classifiers were extensively used to train our model on a variety of parameter values via GridSearch always under the control of cross validation techniques.

The GridSearch algorithm on the Decision Tree classifier was performed over different values of the following parameters: criterion function to measure the quality of a split, maximum depth of the tree, minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node. The highest resulting test score was 60%. The GridSearch algorithm on the Random Forest classifier was performed over the following parameters: number of trees in the forest, maximum depth of a tree, minimum number of samples required to split an internal node and minimum number of samples required to be at a leaf node. The hereby highest resulting test score was 61%.

2.3 Support Vector Machine

SVMs delivered disappointing results in our case. Nevertheless, we included the algorithms within our Python Notebook. For both linear and polynomial kernel functions we didn't reach test scores above 50%. As recommended in the script, we worked with the standardized version of the train and test sets. With growing parameter C, the run times went into the hours very fast such that we are not completely sure to have found the best parameters. Nevertheless, it seems very unlikely that the score would improve onto the level of the Random Forest because the difference is just too big.

2.4 Keras

We decided to also invest time using Keras, a deep learning framework built on top of TensorFlow 2.0, to diversify our analysis. It is not only the most used framework among top-5 winning teams on Kaggle, but also used by state-of-the-art research institutions like CERN, NASA, etc. (Keras (2021)).

Keras consists of a sequential API, which allows layer-by-layer model creation, and a functional API which allows more complex models (Brownlee (2017)). We decided to use the sequential model, because of it's simplicity.

The tuning parameters of Keras' sequential API are the number of layers used, the batch size and the epochs. Our starting point was a two layer model, over which we ran the Grid-Search-Algorithm to determine the optimal batch size-epochs-combination. As almost every resulting optimal combination only achieved a test score between 52% and 54%, we decided to increase the number of layers by one. After again letting the Grid-Search-Algorithm look for the optimal parameter combination, we achieved test scores from 54% to 56%. Unfortunately, an additional layer increase to four layers, did not further improve our test score above 54%. Therefore, our best test score, when using the neural network model, was 56.08%, which was achieved by the following parameter values: 3 layers, batch size 100, 30 epochs.

3 Conclusion

Before starting to work on the multi-class classification problem, we asked ourselves what accuracy score we would be aiming for and what would be acceptable for us, knowing that class imbalance will highly influence our work. We concluded that a score between 65% and 75% would be a good result. Higher scores didn't seem possible, otherwise every Machine Learning student would have to become rich with buying and selling shares. Table 1 summarizes our highest achieved scores per algorithm used.

Method	**Best score**
Random Forest	0.6144
Decision Tree	0.6039
Keras Neural Network	0.5658
LDA and QDA	0.5660
Support Vector Machine	0.4109

Table 1: Best test scores achieved per algorithm

Random Forest delivered the best test score (and a much higher weighted F1-score than Decision Tree). But, in conclusion, we didn't reach 65% accuracy. When we started running our algorithms without having balanced the dataset or handled imputation and outliers well enough, our test scores were at around 53%. In explanation, it was best for all algorithms to just predict the strategy 'Sell' every time. And because 'Sell' is the correct strategy in 53% of the cases in our data, you get this score. Of course such a prediction algorithm is useless. In the following, we managed to push the score up to 61%, predicting 'Sell' and 'Buy' in an acceptable manner. 'Hold' is the best strategy in much less cases, but it is also predicted correctly very rarely which is a bit disappointing. When looking back to where we started from and the different evolutionary steps we took as a group and with our knowledge gained, we are okay with our end product.

Lastly, we are very convinced that if we would get new real world data and would predict a strategy with our algorithms, we would achieve similar scores to what we see in our test score. We have put value on a realistic dataset. We didn't improve our test scores artificially by removing too many outliers or deleting rows with too many missing values. Of course if you would do so, you would improve your test score but would move away from reality. We are sure to have found a good middle way with our process described in the Data Preprocessing chapter.

For further improvement of our test scores, more computational power could be invested in grid search and every single parameter, for example include our feature selection threshold (0.0055) into a grid search algorithm with very long run time.

References

Brownlee, J. (2017). *How to use the keras functional api for deep learning.* `https://machinelearningmastery.com/keras-functional-api-deep-learning/`. (Accessed: 2021-04-18)

Brownlee, J. (2020). *Smote for imbalanced classification with python.* `https://https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/`. (Accessed: 2021-04-18)

Keras, K. (2021). *Keras documentation.* `https://keras.io/`. (Accessed: 2021-04-18)

Raschka, S. (2015). *Python machine learning.* Packt publishing ltd.