FIE 453 Final Project

Karoline Schauer
WU Vienna

Tomke Splettstoesser
CAU Kiel

Justus Valentin Vierow
CAU Kiel

In this project we use company specific and macroeconomic data to predict the return and volatility of stocks. For this matter we applied competitive machine learning classification algorithms. We find that XGBoost predicts significantly better than randomly and is also the best predictor of estimated volatility in comparison to other used algorithms. Continuing this we suggest a larger use of XGBoost in financial prediction, which is not the case so far in the literature.

Table of Contents

List of Tables

List of Figures

1 Introduction

Nearly seventy years ago, Kendall and Hill (1953) were the first to observe that stock prices seem to change fairly random over time. Nowadays,this phenomenon is known as Random Walk Hypothesis and connected to the concept of efficient market hypothesis, which states that all past information is priced in. Until today, predicting stock returns is arguably the most puzzling and hardest challenge in finance. What began with simple time-series models like AR(1) has shifted to more modern machine learning models in recent years. The reason behind that change is that on the one hand conventional statistical methods often fail to accurately display such complex issues and on the other hand they fail to find important patterns in the data. In terms of model inputs it was shown that company specific financial ratios have an impact on stock returns (Lewellen, 2004) as well as different macroeconomic factors (cite a few here), which motivated us to include both in our analysis.

We believe that it is almost impossible to predict the exact quantitative returns, which is why we use the classification method to predict whether the return is positive or negative and whether the returns squared are greater than or equal to 0.01 or not. The squared returns are used as an volatility estimator and we choose 0.01 as a boundary for squared returns, because 0.1 is in our opinion high return to consider for a one month time-frame. To classify we use 4 different algorithms, namely k-nearest neighbor, Neural-Networks, Random Forest and eXtreme Gradient Boosting, which are within the most used machine learning algorithms in finance (Emerson et al., 2019). We additionally try to tune these algorithms to improve our models and rank the algorithms by accuracy and kappa.
These classification models may give an indication for portfolio managers in terms of long/short position of a stock and the size of the position of a stock. Otherwise, these models could also be used an indication to sell or buy if both classification give the same result.

Our results show that classifying stock returns is difficult but we find that XGBoost gives a non-random result. Additionally, XGBoost performs better than the other

applied algorithms for the squared return. Therefore we suggest that XGBoost should be used more often for financial predictions and forecasting, which is not the case yet.

This paper consists of five sections. In Section 2 we introduce the used data and transformations/manipulations made to prepare the data. Section 3 deals with the machine learning algorithms used for our classification and their tuning parameters. In Section 4 we compare and discuss the results of our models, followed by a short conclusion in section 5.

2 Data

2.1 Variables

For our project we combine company-specific with macroeconomic data. For the company-specific data we use merged data from the CRSP and Compustat data. As for the macroeconomic data, we use data from the FRED database as it provides several features of the economy of the United States. Having only US companies in the provided data set, whereas the macroeconomic data from FRED is directly applicable for our project, since it provides data from the US economy.

The historically first used variable for predicting returns is the lagged return of the same security. This comes from the time-series econometrics relation to AR(1) and is reasonable. Instead of Revenue and Shares Outstanding we chose to use the ratio of both variables and therefore Earnings Per Share (EPS) is one of our six variables within the company-specific data. The advantage of using the ratio is to diminish the size effect and have similar amount of information more compact. In addition, we choose volume (VOL). The market value and the book value of a company can be very different so the book-to-market ratio (BM) gives us an overview on the size factor. This is why we are using the Return on Asset ratio (ROA). The last variable used regarding the CRSP and COMPUTSTAT dataset is an enterprise value(EV) multiple (EV/REV). This ratio is relevant as it gives more information on how valuable the company in respect to its revenue is.

Cunado and Perez de Gracia (2014) show that changes in oil prices have an impact on stock prices. Besides change of oil price we also include the oil price itself to reflect the level of energy prices. Current relevant example is the Ukraine war,which increased oil and gas prices sharply.

From a finance perspective we choose to include exchange rate variables. The main world currencies are the U.S. Dollar, the Euro and the Japanese Yen. Thus, we use the exchange rate between the USD and the Euro (USDEUR) as well as the exchange rate between the USD and the Yen (YENUSD). Besides the exchange rate two other factors (among many others) determine the market prices. The first one is the Federal Funds Effective Rate (FED), which is the central interest rate of the U.S. financial market. The second factor is the Treasury Bill Eurodollar Difference (TED Spread), which in this case is the spread between the 3-months LIBOR and the 3-months Treasury Bill. For a long-term view of the US economy, a 10-year treasury note (YIELD) is best suited to our model.

The representative market value in our model is given by the monthly S&P prices (SP). Since we predict returns we include also the S&P return as proxy for market returns. Another variable of this category is the ICE BofA Option-Adjusted-Spreads (SPREAD). These spreads show the difference between an OAS index of all bonds with a certain rating category and a spot Treasury curve. What is more, if we want to predict volatility we need a corresponding macroeconomic variable. This is why, we added the CBOE Volatility Index (VIX) variable to our model. This variable measures the short term volatility expectations of the market on the basis of stock index option prices.

The GDP is of tremendous importance for the macroeconomy of a country and therefore has to be included in such a model to reflect the National Accounts. For our project we decide not to use the absolute GDP of the U.S. but the Brave-Butters-Kelly Real Gross Domestic Product (BBKM). This index calculates monthly annu-

alized changes in the GDP growth of the United States.

Of importance to returns is also the consumer side of the macroeconomy. For this to have a representative of the prices of consumer goods we added the Consumer Price Index (CPI) to our model. Also, in order to reflect the labor market in the U.S. we included the unemployment rate (UNRATE) in our model.

To cover national uncertainty our model contains the Economic Policy Uncertainty Index for the United States (POLICY), which is an index that is based on daily news in the U.S. This is of relevance since if uncertainty is big people are less likely to invest in the stock market.

The last two macro variable for our models are the returns on Gold (GOLDRETURN) to include another asset known as a safe haven and the spread between the market and the risk-free rate(MARKETDIFF).
In Figure 1 we see the correlation plot of all used variables in our analysis. Summary statistics can be found in Table 1 and Table 2.

2.2 Data Manipulations

After merging all collected data we first calculated the needed ratios and lagged all variables for 1 month. For variables where a certain pattern might be important we also included 2 month lagged variables. To clean the data we dropped all missing (NAs) observations. Next we compute dependent binary variables for RET and RET^2. This way we end up with 113059 observations and 37 variables in our data-frame.
As preprocessing we demean and scale all variables, which is particulary important for the kNN but not so much for the tree-based algorithms. We see that using principal component analysis (PCA) is not useful for us because the first principal components do not explain enough variance (Figure 3). For this reason the trade-off would not be worth it to gain marginally in computation speed vs losing accuracy performance. One reason for that is, that we already did data pre-selection using Figure 2, which shows the highly correlated variables and in the process we removed

the highly correlated variables to end up with no variables that are very close to 1 or -1. Therefore, PCA does not seem useful to prevent over-fitting.

As common in machine learning models we split the data in training and test set. For the split of the data set, we decided not to do it randomly, but to split it by time, as it makes more sense to train the past to predict the future. The training set includes data from 31.05.2011-31.12.2017 containing around 70% of observations and the test sets contains data from 31.01.2018-31.12.2020 and contains the remaining 30% of observations.

3 Models

To fit the following algorithms and to find the optimal hyperparameters for our models we use 5-fold cross-validation, which resembles a split in training-validation set in a statistically sophisticated way. K-fold cross-validation is useful to break the bias-variance trade-off by keeping the training set large(less bias) and trying to deal with variance by resampling.

K=10 is generally seen as a good number of folds, but because of not enough computer power and time we only use a 3-fold cross validation. Lower number of folds mean less variance and more bias in comparison to larger K, which we took into consideration by looking at the results.

Also repeated cross-validation is a good choice, averaging over results from fold splits, but takes too much computation power we do not have.

3.1 K-Nearest Neighbors

K-nearest neighbor (kNN) is one of the oldest and simplest non-parametric machine learning algorithms, but still can have competitive performance results compared to other algorithms. First introduced by Fix and Hodges Jr (1951) and formalized by Cover and Hart (1967), the kNN algorithm found great success in various fields.

We consider a data set of pairs of observations $\{(\mathbf{x}_1, \theta_1), ..., (\mathbf{x}_n, \theta_n)\}$ in a metric space $\mathcal{X}$. $\mathbf{x}_i$ is a vector of predictors of individual i and θ_i the binary classification. Having a new pair $(\mathbf{y}, \theta_y)$ from which only $\mathbf{y}$ is observed kNN tries to find the best possible

guess of θ_y given the information of correctly classified data points by minimizing a distance function and finding the nearest neighbors: $\mathbf{x}' \in (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ is a nearest neighbor to $\mathbf{y}$ if:

$$min\, d(\mathbf{x}_i, \mathbf{y}) = d(\mathbf{x}', \mathbf{y}) \quad \forall\ i \in 1, 2, ..., n \tag{1}$$

The kNN approach computes the k closest neighbors of $\mathbf{y}$ and classifies θ_y by the majority of discrete outcomes. Especially if the outcome variable is binary it is useful to choose k equal to an uneven number because otherwise, it could be possible we do not find a majority of one discrete outcomes. We will use three ways to tune the kNN algorithm.

First, choosing the parameter k can have a big impact on the classification of θ_y and is therefore of great importance.
A small k increases the effect of outliers (noisy decisions) in the data and usually over-fits. This also leads to a smaller bias. High k decreases the impact of the nearest neighbor and therefore reduces the variance. This comes with the price of tendency to under-fit and results in a greater bias. Therefore, the optimal k can be defined as the best bias-variance trade off. In our case, we compare performance metrics of different k and then define the k with the best performance metric as "optimal".

Secondly, we will use different distance functions to compute the nearest neighbors, which can have big impact on performance of kNN algorithm (Hu et al., 2016). Mathematically, d is called a distance function if for any vector $\forall\ \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{X}$ the following axioms are satisfied:

1. For all $\mathbf{x} \in \mathcal{X}$, $d(\mathbf{x}, \mathbf{x}) = 0$ (Distinguishability)
2. For any distinct $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, $d(\mathbf{x}, \mathbf{y}) > 0$ (Positivity)
3. For any $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ we have $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ (Symmetry)
4. For any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{X}$, we have $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$ (Triangle Inequality)

Every function satisfying the previously listed axioms is called a distance function. Continuing this idea we can think of different functions satisfying these axioms. Starting from Minkowski distance first, we choose 4 different parameters q equal to one of the following elements $q \in \{1,2,3,4,\}$:

$$d(\mathbf{x}_a, \mathbf{x}_b) = \sqrt[q]{\sum_{j=1}^{P} |x_{aj} - x_{bj}|^q} \tag{2}$$

This includes the most commonly used Euclidean distance (q=2), which is derived from Pythagorean theorem and representing a straight line between two points and the the Manhattan distance(q=1), which shows better performance in high dimensional space compared to L2 (Aggarwal et al., 2001).

Third tuning parameter is the weighting function, also called kernel. It might be reasonable to give the nearest neighbor a bigger weight than the other nearest neighbor. We try out 4 different kernel functions. As a basis we use the rectangular unweighted function giving every nearest neighbor the same weight. Then we also use Gaussian Kernel with a normally distributed weight function and the Epanechnikov kernel. These 3 have the following kernel functions:

1. Uniform: $K(x, y) = \frac{1}{2}$
2. Gaussian: $K(x, y) = \frac{1}{\sqrt{2\pi}} \exp(\frac{1}{2} - ||x - y||^2)$
3. Epanachnikov: $K(x, y) = \frac{3}{4}(1 - ||x - y||^2)$

Additionally, we try the "optimal" kernel function introduced by Samworth (2012).

3.2 Neural Networks

Originally observed as a network of biological neurons, Deep Learning has shown great success in recent years in the machine learning field because of its high performance in areas such as image recognition and speech recognition. But neural networks are also used in finance, where for example Enke and Thawornwong (2005) showed that neural network classification can provide better risk-reward strategies than buy and hold and therefore did something similar to our project.

We use the single hidden layer neural network provided by the nnet R package. There the neural network consists of 3 layers in total, the input layer, the hidden layer and the output layer. The input layer simply consists of the input values, whereas the calculations are part of the hidden layer. The output layer produces combines and produces end results. Generally speaking, a neural network is a set of connected input and output units where each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights to be able to correctly predict the output target of a given set of input samples.

We use avNNet such that the same neural network is fitted using various random values. Averaging in our case means that scores are averaged first and after that pointed to predictions 1 or 0. Also we use bagging (bootstrap aggregation), because it often provides better predictive performance.
Our tuning parameters for neural networks are size and decay. Size means the number of units in the 1-hidden layer and decay is a regularization parameter to evade over-fitting. We try decay = (0.5, 0.1, 1e-2, 1e-3) and size = (1,5, 10, 20) as reasonable values to find the optimal parameters.

3.3 Random Forest

Originally decision trees and bagging were used in many data science applications. But even small noises in the data can make big difference because of the high variance. Random forests try to solve this problem by training multiple decision trees (that is why it is called forest) on changing sub-spaces of the features. Essentially, random forests are just a modification of decision trees and bagging building a large amount of not correlated trees and averaging them. This comes with the cost of slightly higher bias but reduced variance.
Khaidem et al. (2016) propose that to minimize forecasting error, it is better to use classification for predicting, which suits our project.
As tuning parameter we use mtry, the number of randomly selected variables when forming each split in the tree. This implies that each split includes a different random set of variables. Mtry can have significant impact on final accuracy and we try a quite big range of different mtry. Regarding the number of trees, we keep it at the

caret default value of 500 trees. This is caused by mainly two factors. The first one is that theoretically the performance of random forest should be mostly constant with reaching a certain tree threshold. The second reason is that with more trees the computation power is increasing as well.

3.4 XGBoost

XGBoost is scalable machine learning system for tree boosting, which got famous for outstanding performances in kaggle competitions (Chen and Guestrin, 2016). In comparison to regular gradient boosting, the XGBoost uses advanced regularization to avoid over-fitting, which gives better performance. Otherwise XGBoost uses the same idea as gradient boosting. However, not yet frequently used in finance we want to find out whether XGBoost can achieve similar superior results compared to other algorithms.

Since there are many tuning parameters to choose from, we use the tunelength argument from caret, which uses random selection to find the "optimal" model parameters. One of the main xgboost tuning parameters is the eta, also called the learning rate. It is the step size shrinkage to prevent overfitting. After each boosting step one receives the weights of new features and eta directly shrinks the weights to make boosting more conservative. Default values in caret is eta=0.3 and possible range is in [0,1]. Another parameter we tuned on is maxdepth, which equals the maximum depth of a tree.

Additionally the subsample parameter gives the ratio of training instances, meaning xgboost randomly samples the ratio amount prior to growing the trees, which prevents overfitting. Gamma is a regularisation parameter, which in comparison to maxdepth regularises "across" tree information not "inside" the tree. Overall there are more parameters that could prevent over-fitting.

4 Results

4.1 Fits

As expected we find that the number of nearest neighbors matter. More nearest neighbors lead to higher accuracy with the slope getting less steep with more nearest neighbors. This is likely the case because of the high effect of outliers, which is

diminished when using more nearest neighbors. For RET the Manhattan distance function is clearly superior to the other distance functions, this is coherent to results of Aggarwal et al. (2001) showing that L1 achieves better performance than L2 in high-dimensional space. Predicting RET^2 we do not see a difference regarding the distance functions. Unexpectedly, we do not observe a difference regarding kernel functions.
For the neural network fits we find for both RET and RET^2 that higher number of hidden units improve accuracy. However, the slope seems to be reduced from 10 hidden units on wards. As of weight decay we do not see a significant difference for RET, but find that a very low weight decay for RET^2 gives less accurate results (Figure 6 and Figure 7).

As for random forest fits, we see similar patterns for RET and RET^2 that accuracy seems higher with less random predictors selected. But considering the y-axis values, the difference seems to be not significant enough (Figure 8 and Figure 9).

For XGBoost we find that max tree depth of 2 and 3 seem to be the best, with max tree depth of 1 performing considerably less accurate. Other parameter do not seem to show a direct pattern of improving the fit.

Overall, we can say that all 4 algorithms fit equally within a range of [0.63,0.642] for RET and [0.721,0.733] for RET^2. It is not surprising that RET^2 fits better. This is widely known and reason for this is that on the one hand volatility is mean reverting, meaning in the long term volatility seems to tend towards the average and on the other hand volatility comes from uncertainty, which is easier to predict than how peopel react to the uncertainty.

4.2 Predictions

What we are really interested in is how well we predict. Applying our fitted models to the test set, the data set that our models have not seen yet, we get significantly worse results. This indicates to likely over-fitting our models. Similarly to comparing the fitted models we see that predicting RET^2 is more accurate than predicting RET.

For RET^2 all models predict more or less identically well with around 0.61 accuracy. But predicting the RET we find that XGBoost predicts significantly better than the other models. As mentioned we believe over-fitting is the problem XGBoost solves better than the other algorithms.

Another reason for bad prediction might be the time-frame used as test set. Since we train and validate with data from 2011-2018 and test with data from 2019-2020, the test sets contains the huge downturn in returns because of the corona crisis but the training set does not include huge crisis events. If we would be able to include data from the financial crisis around 2008 it might give a better prediction.

5 Conclusion

.....

.....

.....

References

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Cunado, J. and Perez de Gracia, F. (2014). Oil price shocks and stock market returns: Evidence for some european countries. *Energy Economics*, 42:365–377.

Emerson, S., Kennedy, R., O'Shea, L., and O'Brien, J. (2019). Trends and applications of machine learning in quantitative finance. In *8th International Conference on Economics and Finance Research (ICEFR 2019)*.

Enke, D. and Thawornwong, S. (2005). The use of data mining and neural networks for forecasting stock market returns. *Expert Systems with Applications*, 29(4):927–940.

Fix, E. and Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, California Univ Berkeley.

Hu, L.-Y., Huang, M.-W., Ke, S.-W., and Tsai, C.-F. (2016). The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, 5(1):1304.

Kendall, M. G. and Hill, A. B. (1953). The analysis of economic time-series-part i: Prices. *Journal of the Royal Statistical Society. Series A (General)*, 116(1):11–34.

Khaidem, L., Saha, S., and Dey, S. R. (2016). Predicting the direction of stock market prices using random forest. *CoRR*, abs/1605.00003.

Lewellen, J. (2004). Predicting returns with financial ratios. *Journal of Financial Economics*, 74(2):209–235.

Samworth, R. J. (2012). Optimal weighted nearest neighbour classifiers. *The Annals of Statistics*, 40(5):2733–2763.

Figures and Tables

Table 1: Summary statistic company data

Variable	N	Min	Mean	Max	St. Dev.
RET	113,059	−0.929	0.011	8.798	0.158
EPS	113,059	−88.430	0.339	149.910	2.331
VOL	113,059	0	303,353.000	31,003,122	792,266.400
BM	113,059	0.00000	0.003	1.525	0.017
ROA	113,059	−14.957	0.023	64.832	1.007
EV/REV	113,059	2.521	91,798.250	574,006,533.000	3,670,571.000

Table 2: Summary statistics macro data

Variable	N	Min	Mean	Max	St. Dev.
oilprice	113,059	19.230	68.689	107.080	24.267
policy	113,059	92.867	178.915	430.263	72.138
growth	113,059	−0.669	0.135	0.975	0.314
bbkm	113,059	−76.036	2.000	46.119	11.211
unrate	113,059	3.500	6.169	14.700	2.240
Yield	113,059	0.620	2.159	3.580	0.636
Spread	113,059	1.960	3.376	6.410	1.009
USDEUR	113,059	1.056	1.216	1.452	0.114
YENUSD	113,059	76.340	102.644	123.940	13.246
VIX	113,059	10.180	17.660	53.540	6.886
FED	113,059	0.050	0.614	2.410	0.753
TED	113,059	0.130	0.309	1.340	0.160
SP	113,059	1,131.420	2,155.522	3,756.070	629.055
SPreturn1	113,059	−0.699	0.008	1.891	0.106
Oilreturn1	113,059	−0.819	0.025	4.454	0.252
Goldreturn1	113,059	−0.441	0.003	0.861	0.073
Marketdiff1	113,059	−1.805	0.004	8.743	0.178

Table 3: Comparing Fits

	kNN	NN	RF	XGB
RET	0.631	0.635	0.642	0.638
RET^2	0.721	0.732	0.733	0.733

Table 4: Comparing Predictions

	kNN	NN	RF	XGB
RET	0.488	0.519	0.495	0.545
RET^2	0.604	0.609	0.611	0.619

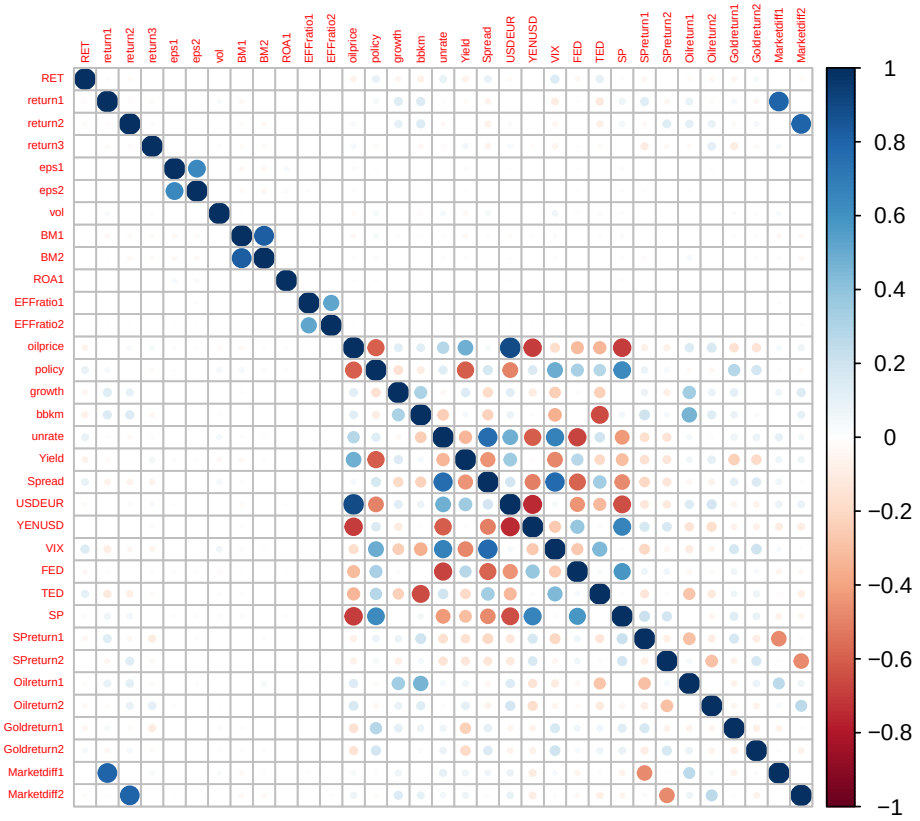

Figure 1: Correlation plot

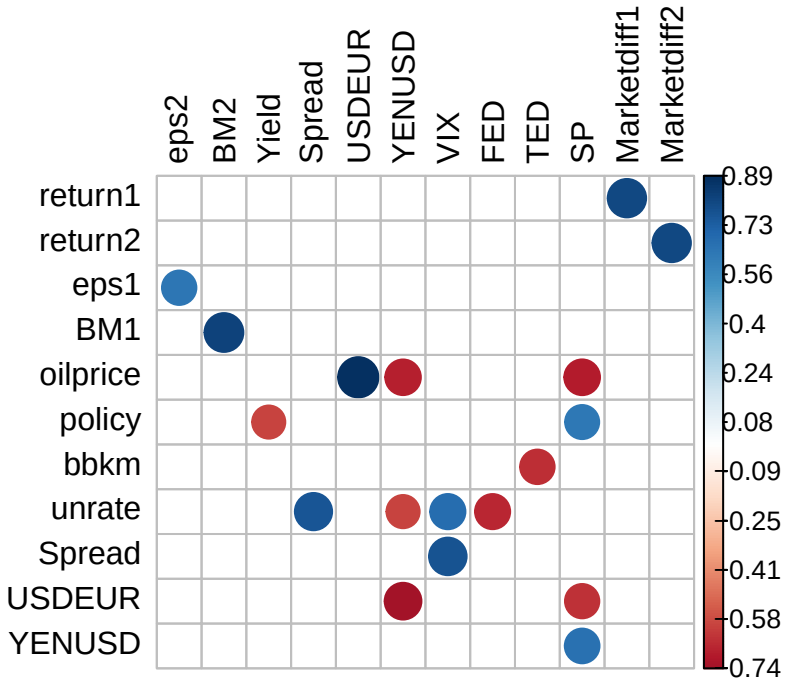

Figure 2: Highly correlated variables

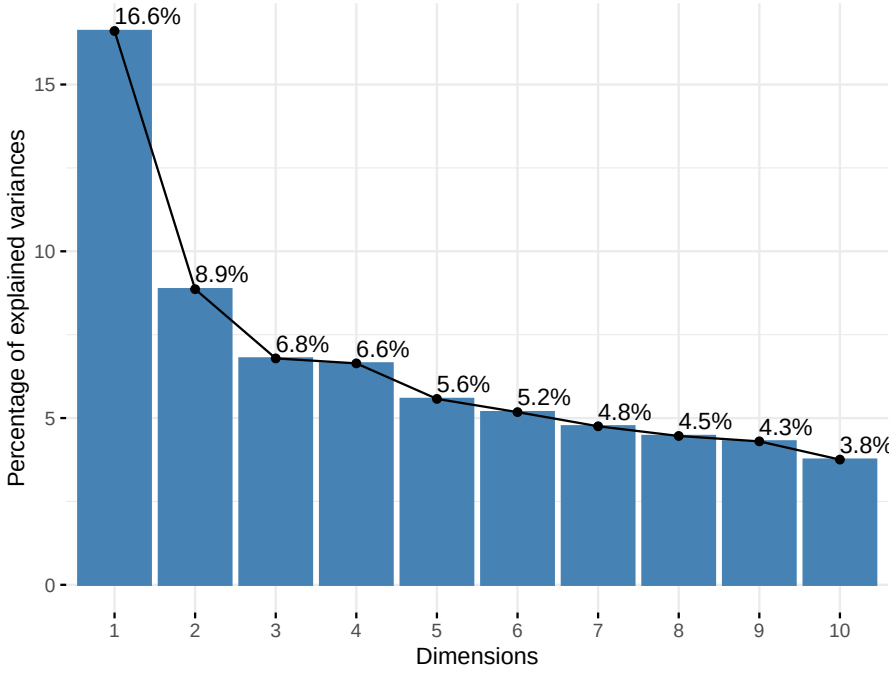

Figure 3: PCA Scree Plot

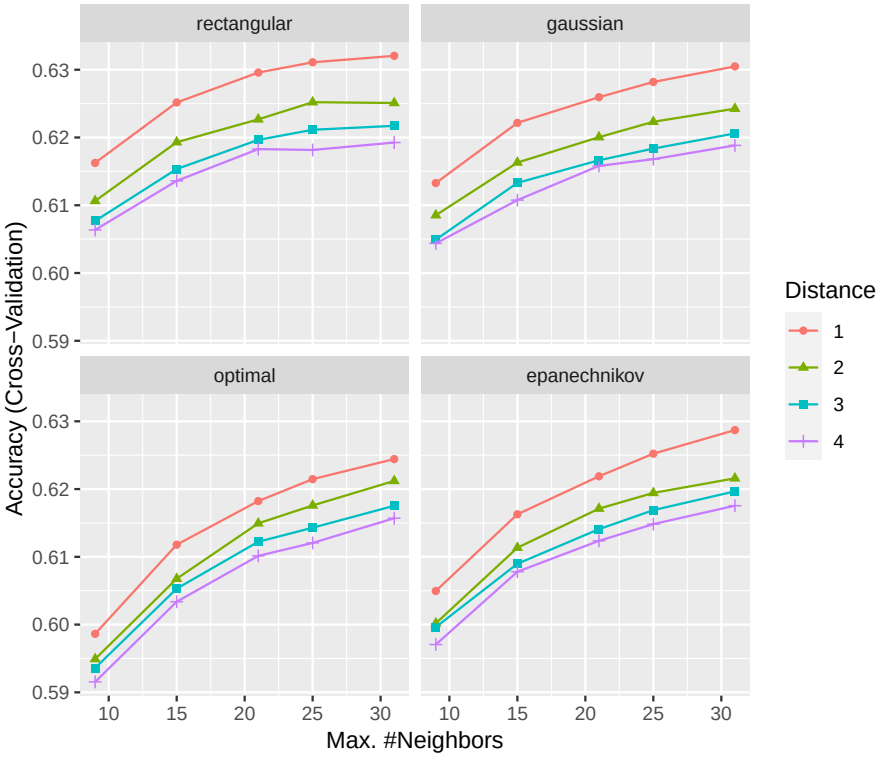

Figure 4: Fitting/Tuning kNN RET

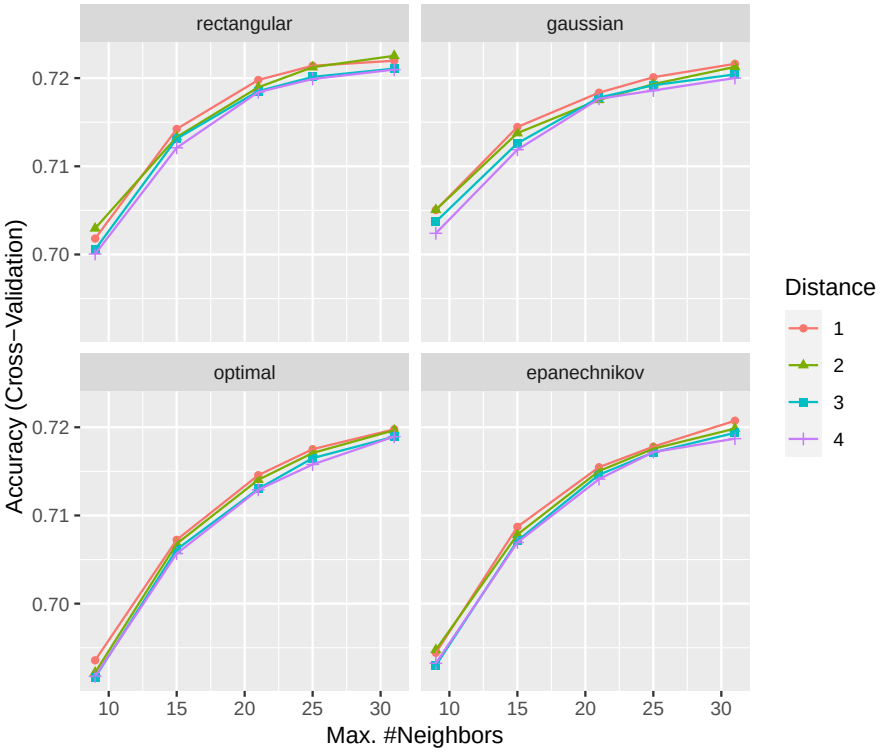

Figure 5: Fitting/Tuning kNN RET^2

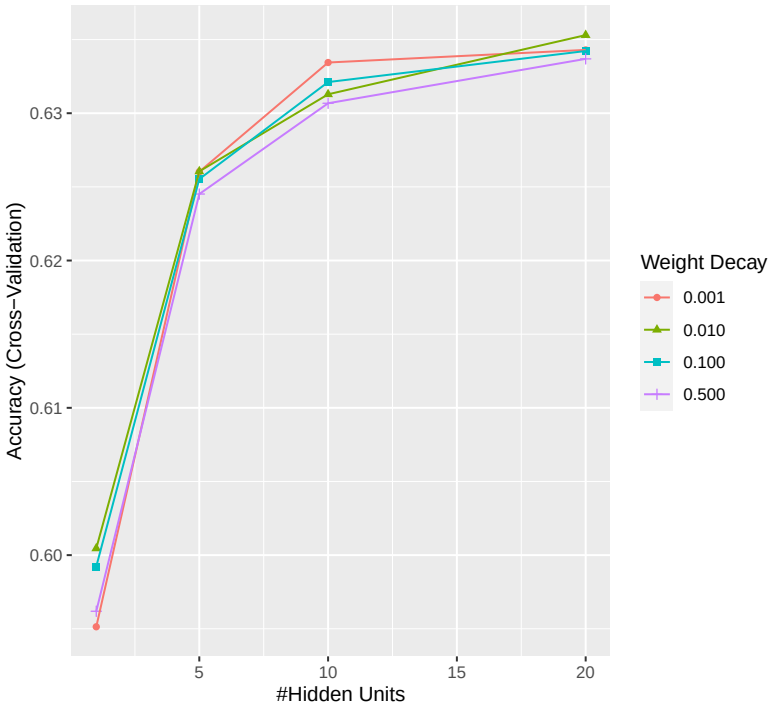

Figure 6: Fitting/Tuning neural network RET

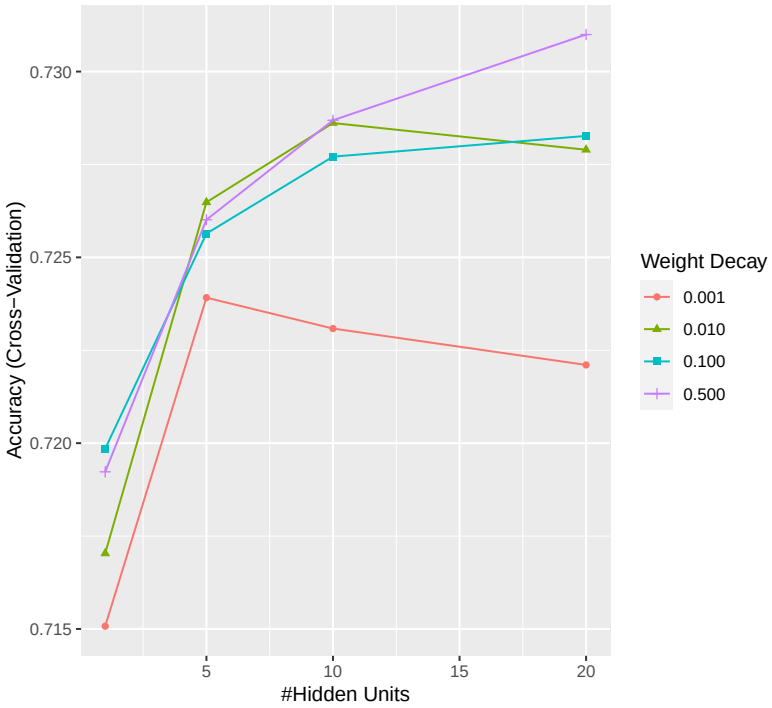

Figure 7: Fitting/Tuning neural network RET^2

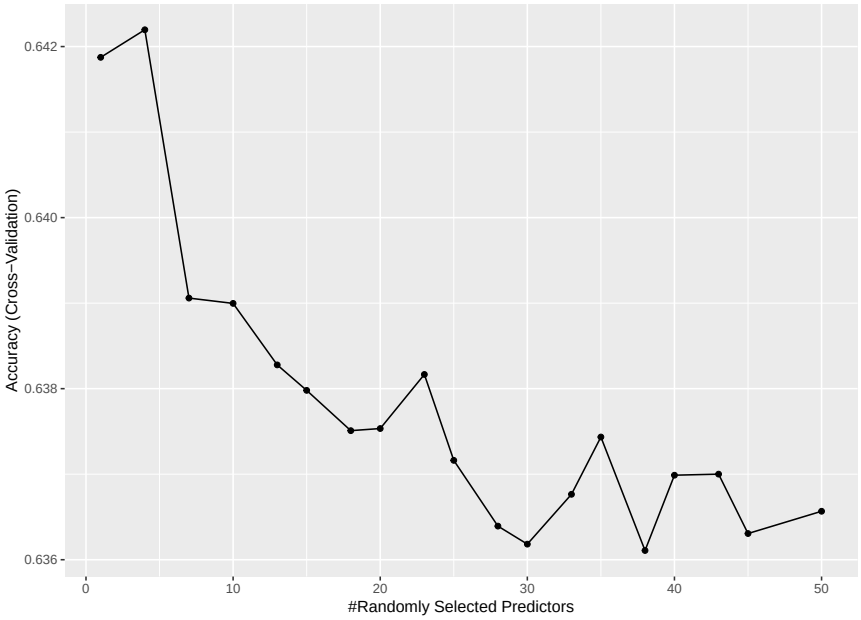

Figure 8: Fitting/Tuning Random Forest RET

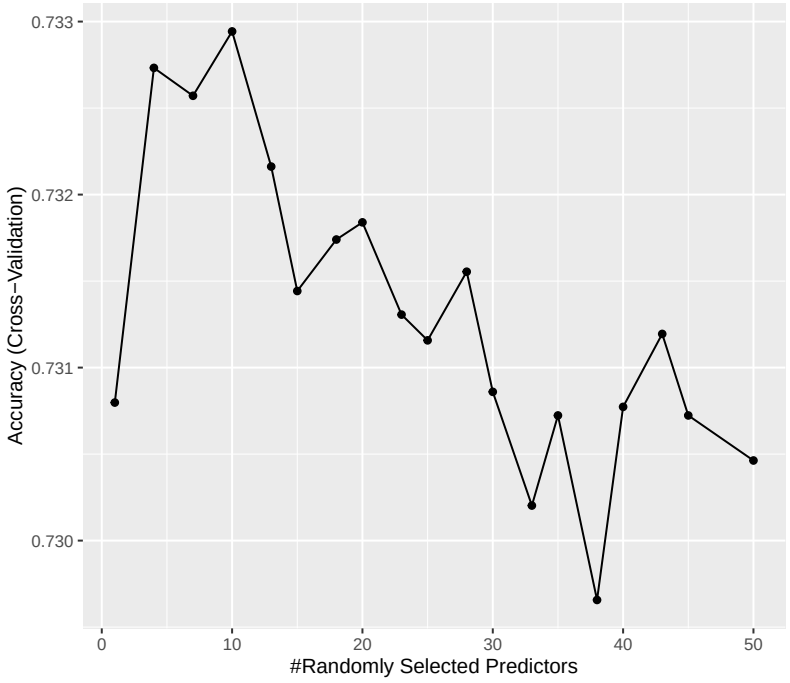

Figure 9: Fitting/Tuning Random Forest RET^2

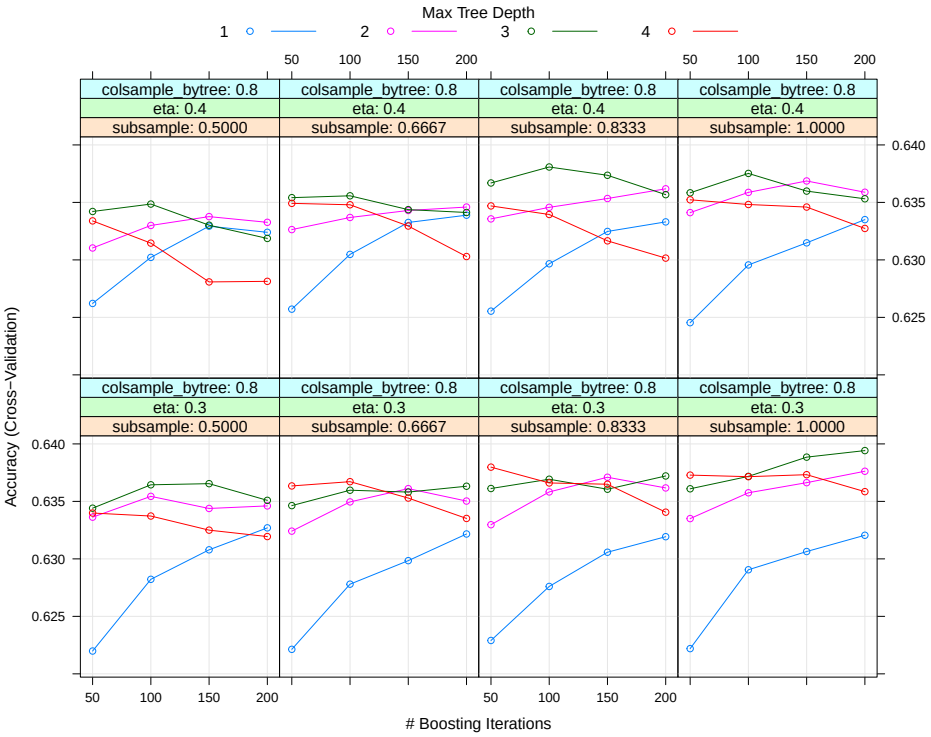

Figure 10: Fitting/Tuning XGBoost RET

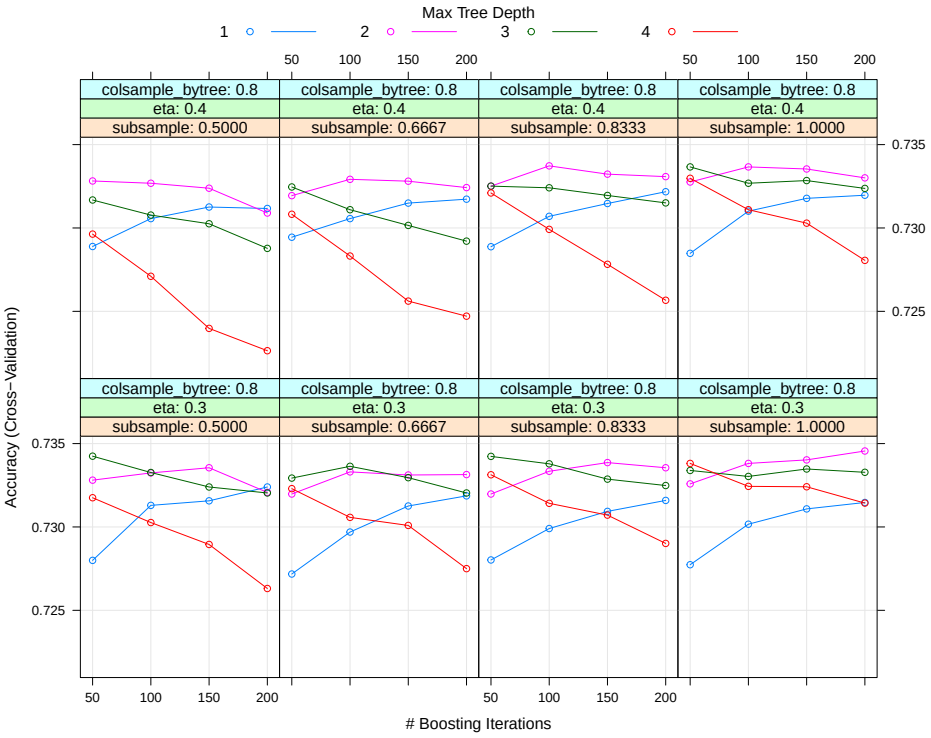

Figure 11: Fitting/Tuning XGBoost RET^2