University of Zurich[UZH]

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Machine Learning in International Asset Pricing

Master's Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Quantitative Finance

Author

Riccardo Tegazi

riccardo.tegazi@uzh.ch

Supervisor

Prof. Dr. Markus Leippold

Hans Vontobel Professor of Financial Engineering

Department of Banking and Finance

University of Zurich

Assistant

Gianluca De Nard

Date of Submission: 7 August, 2020

Part of this research was done during an internship at Unigestion, in Genève. Special thanks to Prof. Dr. Robert Kosowski, Alisa Shishkina, Ravi Ramakrishnan and Dr. Vladyslav Dubikovskyy.

Executive Summary

Machine learning has gained widespread attention in finance, in particular in asset pricing, where Gu et al. (2020a) showed the superior predictive power of algorithms such as neural networks and tree ensembles. We conduct a comparative experiment of six different algorithms across three regions: United States, Europe and Japan, that we also combine in a unique universal balanced data set. We introduce state-of-the-art developments in machine learning in terms of methods (XGBoost and VASA), model explainability (SHAP values) and hyperparameters optimization (Bayesian search), but also in covariance matrix estimation (Analytical Shrinkage) and portfolio construction (Efficient Sorting).

We confirm the supremacy of complex non-linear algorithms over linear ones both from a predictive and financial performance standpoints. Neural networks are outperformed by tree ensembles in region-specific universes, while they are more competitive in the universal one. We show how restricting only to stocks with full history of returns, in the balanced data set, can decrease overall performances, because of the absence of stocks with smaller capitalization, and stock-specific metrics can highlight extreme errors made by our models. However, we can achieve similar financial results with Long-Only strategies, while Efficient Sorting can enhance the performances of Long-Short portfolios. We perform statistical tests to assess the significance of differences in predictive power and strategies' risk-adjusted returns, showing how even large gaps can often be seen as random fluctuations. Finally, we make an extensive use of SHAP values for non-linear algorithms, to show how they can provide invaluable insights on their behaviour, with a solid theoretical framework derived from game theory, that unifies several other approaches used in the field of explainability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning has gained a huge popularity and a widespread adoption in many fields in recent years, thanks to astonishing successes in difficult tasks, such as mastering the game Go, see Silver et al. (2016), and recommending films to see to streaming users, see Bennett et al. (2007), only to mention two instances well known also to the general public. Together with the always greater and cheaper availability of computational power and sources of data, this lead to several applications of machine learning in finance. Moreover, in the field of asset pricing, there has always been an extensive research to detect factors that are able to predict stocks excessive returns, see Fama and French (2008). As a consequence, hundreds of potential 'anomalies' have been found, for instance Harvey et al. (2016) review 313 papers that study cross-sectional return patterns. Therefore, machine learning has been identified has a potential solution to leave to algorithms the task of identifying the relevant predictors, with the capacity to handle large amount of data and also non-linear relationships, that cannot be identified by simple linear methods or by experience.

The paper by Gu et al. (2020a) has been a major contribution, performing a really extensive comparison of several machine learning methods on a large universe of United States stocks and showing promising performance of algorithms such as neural networks and tree-ensembles. Since then, there has been increasing interest to extend the set of machine learning techniques and the universes of stocks. On the first path, Gu et al. (2020b) use autoencoders to build a latent factor conditional asset pricing model, Chen et al. (2019) utilize generative adversarial networks to enforce no-arbitrage con-

ditions and recurrent Long-Short-Term-Memory (LSTM) networks to model hidden economic states, while De Nard et al. (2020) try to step back to linear methods, but adding variables subsampling (VASA). The second path has mainly been followed by practitioners in the asset management industry, who try to exploit this machine learning predictive power in different regions, for instance Wolff and Echterling (2020) apply a set of algorithms to both an American and a European index. We will refer mainly to the Gu et al. (2020a) framework to analyze three region-specific universes (Europe, Japan, United States), but also to De Nard et al. (2020) for the study of a 'universal' balanced data set. The main difference is that we will work with next-year returns, instead of next-month ones. In terms of algorithms, instead of standard Gradient Boosting Machines, we will use XGBoost, which has become the state-of-the-art model for predictions with tabular data. Moreover, we use Bayesian search to tune hyperparameters of most complex models. Finally, we will make an extensive use of SHAP values, introduced by Lundberg and Lee (2017), to explain our complex models: they give invaluable insights on their behaviour and this is particularly meaningful in the field of asset pricing. Especially from a practitioner point of view, being able to explain our own models is fundamental, not only to keep them under control, but also to report to other stakeholders, such as senior management or clients, who might not be able to deeply understand models from a mathematical point of view.

Chapter 2

Algorithms and Methodology

2.1 Problem Setup

The purpose of our problem is to predict assets' ranked excess returns over the next year, from t to $t + 12$, denoted by $r_{i,t+12}$ for each asset $i \in \{1, ..., N\}$, where N is the number of stocks in the universe, and $t \in \{1, ..., T\}$. An asset's ranked excess return can be defined as an additive prediction model:

$$r_{i,t+12} \coloneqq \mathbb{E}_t[r_{i,t+12}] + \epsilon_{i,t+12}, \tag{2.1}$$

where

$$\mathbb{E}_t[r_{i,t+12}] \coloneqq \ g(z_{i,t}), \tag{2.2}$$

i.e. we assume the conditional expectation $\mathbb{E}_t[r_{i,t+12}]$, taken at time t, to be a function $g(\cdot)$ of the P-dimensional vector $z_{i,t}$, with P being the number of predictors.

We note the $\mathbb{E}_t[r_{i,t+12}]$ depends only on $g(\cdot)$ and $z_{i,t}$, in particular our prediction of the excess return of stock i at time t is based only on the information of the stock itself at that time, we do not use other stocks or previous dates. What we learned form the past is only contained in the training of the model $g(\cdot)$.

Predicting the rank of returns, instead of raw returns, is an uncommon choice in this field. The reason we do it is to simplify the problem to get higher accuracy, because for most of our applications, even

if we were to predict the raw returns, we would anyway use only the rank. However, we have to pay a price for this, as we will see in Section 2.6.1.

2.2 Methodology

To evaluate and compare the performance of our models, we need to leave aside some data that we will use for testing. Moreover, we need validation sets to tune hyperparameters. As we are dealing with time-series data, this must be done preserving the temporal ordering. Therefore, at each date we will split data in the following way:

- A training sample, made by the first 80% of the observations. It is used to fit the models with all the different combinations of hyperparameters that we want to consider
- A validation sample, made by the next 20% of data, used to tune hyperparameters. With the models estimated on the training set, we build predictions for the validation sample and we select the hyperparameters with the best performance. In this way, we can optimize these hyperparameters according to an out-of-sample measure of performance. The hyperparameters we will tune are: the penalization weight in LASSO, the number of predictors to subsample K in VASA, number of trees, learning rate, maximum depth of trees and dropout rate of columns in XGBoost and learning rate, dropout rate, L1-penalization weight and batch size in Neural Networks.
- A testing sample, made only by the last date of the whole sample. It is the date in which we virtually want to make our truly out-of-sample forecast and that we will use to test our models.

We note that after having selected hyperparameters, we retrain each model on the whole training data, i.e. both on the training and validation sample. It is important to insert one-year buffers between the training and the validation samples and between the validation and the testing samples, because we work with next-year returns. Otherwise, we would have overlapping between the different samples, which would results in an underestimation of both the validation and the testing error, with a potential bias in the selection of hyperparameters. From a practical point of view, we will repeat

this procedure each month since November 2010, in order to have at least ten years of training data with two years of buffers. As we move ahead, we have two ways to select the whole training data:

- The expanding window, i.e. using all the previous observations available.
- The rolling window, i.e. using only the most recent x-years of data.

In region-specific universes, we will optimize and train one model with the expanding and one with the rolling window for each algorithm and we will combine them with an equally weighted average of the predictions. The purpose is to reduce the impact of the oldest data points and capture more short-term patterns. In the universal balanced universe, we will train models only on the rolling window, because of the greater computational effort. Moreover, we re-train linear models every month and non-linear ones every six months.

2.2.1 Performance Evaluation

To measure the out-of-sample predictive accuracy of our methods, we will mainly use the out-of-sample R^2, as in Gu et al. (2020a):

$$R^2_{\text{oos}} = 1 - \frac{\sum_{(i,t)\in\mathcal{T}_{TEST}} \left(r_{i,t+12} - \widehat{r}_{i,t+12}\right)^2}{\sum_{(i,t)\in\mathcal{T}_{TEST}} r^2_{i,t+12}}, \tag{2.3}$$

where $\mathcal{T}_{TEST}$ indicates that we are computing the R^2 only on testing samples. As a cross-sectional performance measure, we will use also the Pearson correlation coefficient:

$$R_{\text{oos}} = \frac{\sum_{i=1}^{n} \left(r_{i,t+12} - \bar{r}_{i,t+12}\right)\left(\widehat{r}_{i,t+12} - \bar{\widehat{r}}_{i,t+12}\right)}{\sqrt{\sum_{i=1}^{n} \left(r_{i,t+12} - \bar{r}_{i,t+12}\right)^2}\sqrt{\sum_{i=1}^{n} \left(\widehat{r}_{i,t+12} - \bar{\widehat{r}}_{i,t+12}\right)^2}}, \tag{2.4}$$

this should give results in line with R^2, although absolute values may be inflated because of the demeaning applied in the formula.

When we will have a fixed number of stocks, we will also compute the asset specific R^2, as suggested by De Nard et al. (2020):

$$R^2_{\text{oos},i} := 1 - \frac{\sum_{t\in\mathcal{T}_{TEST}} \left(r_{i,t+12} - \hat{r}_{i,t+12}\right)^2}{\sum_{t\in\mathcal{T}_{TEST}} r^2_{i,t+12}}, \tag{2.5}$$

this is extremely useful to spot eventual extreme predictions that may be hidden by the cross-sectional R^2. They are particular dangerous because they could prompt big bets in stocks where we have high confidence, but this could result in large losses and a mismatch between prediction accuracy and portfolio out-of-sample performance.

Model comparison

Once we have our performance measurements, we would like to understand whether differences are statistically significant or they might be only random fluctuations. To this purpose, We will perform Diebold and Mariano (1995) tests for out-of-sample predictive accuracy between each couple of models. We will follow the adaptation proposed by Gu et al. (2020a), which compares the cross-sectional average average of prediction errors of the two model, instead of individual returns. The test-statistic for the algorithm (1) versus algorithm (2) will be $\mathrm{DM}_{12} = \bar{d}_{1,2}/\hat{\sigma}_{\bar{d}_{1,2}}$, with

$$d_{12,t+12} = \frac{1}{n_{3,t+12}} \sum_{i=1}^{n_{3,t+1}} \left(\left(\hat{e}^{(1)}_{i,t+12} \right)^2 - \left(\hat{e}^{(2)}_{i,t+12} \right)^2 \right), \tag{2.6}$$

where $\hat{e}^{(1)}_{i,t+12}$ and $\hat{e}^{(2)}_{i,t+12}$ indicate prediction errors for the stock i predicted at time t by each method.

2.3 Algorithms

We will now go into the details of the algorithms we will compare.

2.3.1 Ordinary Least Squares

We begin with the most simple algorithm in our panel: Ordinary Least Squares (OLS). While it is supposed not to be competitive in terms of forecasting power, because of the high number of predictors and potential autocorrelation and heteroskedasticity in the data, it is anyway a useful benchmark, also from a variables' importance point of view.
OLS assumes that the conditional expectations $g(\cdot)$ are linear functions of the predictors :

$$g(z_{i,t}; \alpha, \beta) = \alpha + z'_{i,t}\beta, \tag{2.7}$$

where α is the intercept and β the P-dimensional vector of predictors weights. In other words, using Equations (2.1) and (2.2):

$$r_{i,t+12} = \alpha + z'_{i,t}\beta + \epsilon_{i,t+12}. \tag{2.8}$$

We find the weights minimizing the sum of squared residual as loss function:

$$\mathcal{L}(\alpha, \beta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(r_{i,t+12} - g(z_{i,t}; \alpha, \beta)\right)^2. \tag{2.9}$$

2.3.2 LASSO

To tackle some of the pitfall of OLS, in particular the potential overfitting to large-error data points, we introduce the LASSO regression. It simply consists in adding a L_1 penalization of the weights in the loss function in the Equation (2.9). LASSO not only shrinks parameters towards zero, but it induces sparsity, setting a subset of the coefficients to zero. Hence, it can also be seen as an automatic variable selection method. The loss functions becomes:

$$\mathcal{L}(\alpha, \beta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(r_{i,t+12} - g(z_{i,t}; \alpha, \beta)\right)^2 + \lambda \sum_{p=1}^{P} |b_p|, \tag{2.10}$$

where λ is the weight of the L_1 norm of the weights and it is an hyperparameter to tune.
To understand why LASSO forces some coefficients to zero, when λ is big enough, we simply notice that minimizing the loss function in Equation (2.10) is equivalent to minimizing the standard least-squares in Equation (2.9), adding the constraint:

$$\sum_{p=1}^{P} |b_p| \leq \gamma, \tag{2.11}$$

for the corresponding value of γ.

2.3.3 VASA

VASA (Variable Subsampling Aggregation) has been recently introduced by De Nard et al. (2020). In general, VASA works combining a set of independent submodels, where each submodel is trained

only on a subset of the features. The probability of selecting a certain feature should not be uniform, but should depend on a measure of variables importance. Submodels can be combined through a simple average or in more complex ways.

The strength of VASA lays in the consistent random selection of features, because it allows to capture more diverse relationships than a single model fit on all the data, where a small subset of features may be dominant, reducing potential biases coming from model validation. This is particularly true when submodels are linear.

In our setting, the VASA function $g(\cdot)$, is made by a combination $f(\cdot)$ of the submodels g^{BASE}. At time $t, (1 \leq t \leq T)$, for the stock $i, (1 \leq i \leq N)$, we have :

$$\hat{r}_{i,t+12} = g\left(z_{i,t}\right) := f\left[\hat{g}^{BASE}\left(\tilde{z}_{i,t,1}\right), \ldots, \hat{g}^{BASE}\left(\tilde{z}_{i,t,B}\right)\right], \tag{2.12}$$

where B is the number of submodels and $\tilde{z}_{i,t,b}$ is the subset of the input space where the b-th submodels is trained on and has $K_b \leq P$ number of predictors.

It is clear that the VASA framework is quite general and several choices has to be made, following De Nard et al. (2020):

- The function $f(\cdot)$ will perform a simple average of the B submodels:

$$\hat{r}_{i,t+12} = \frac{1}{B} \sum_{b=1}^{B} \hat{g}^{BASE}\left(\tilde{z}_{i,t,b}\right). \tag{2.13}$$

- The submodels will be Ordinary Least Squares.

- The subsampling probability of the p-th predictor will be

$$q_p = R^2_{i,p} / \sum_{j=1}^{P} R^2_{i,j}, \tag{2.14}$$

  where $R^2_{i,p}$ is the R^2 that we get from a one factor regression of r_i on the p-th predictor. In this way, we will sample more often those variables which seem to have more explanatory power.

In the end, to analyse factors' importance, De Nard et al. (2020) rank covariates according to their total frequency in all VASA submodels. As we have just seen, this will mainly depend on the R^2 of

the univariate regressions. Moreover, a covariate may appear in every model, but with a very low coefficient, hence its impact would be overestimated by the frequency. Therefore, we will use as a measure of importance the average coefficient across all submodels of the predictors, considering it zero (no impact) if it was not selected. In this way, we will give importance both to the frequency and to the size of coefficients.

2.3.4 Decision Trees

There are many algorithms that work by partitioning the input space in hyper-cubic regions and they are usually referred as tree-based. We will focus only on Classification And Regression Trees (CART), one of the first and most popular decision tree model, introduced by Breiman et al. (1984). Decision trees are completely non-parametric and each hyper-cubic region can be seen as group of similar observations, where we usually predict a constant value in a regression setting. This makes them highly interpretable, because every step is equivalent to a binary decision, where we ask if a certain quantity is above or below a certain threshold.

To build a decision tree model, we need to decide the sequence of variables we use to split and the respective thresholds. Trying all possible combinations and minimizing mean squared error is computationally unfeasible, even for a fixed number of splits. Therefore, greedy algorithms are always implemented. In general, we start with a single node and then proceed adding one node at time. At each step, we want to add two new leaf nodes, having a certain number of variables available. The variable and its threshold are selected through exhaustive search, and we will predict the average of the data in the new region. In the end, we must decide when to stop adding trees. We can do it when the reduction in the error becomes small, but we could lose future useful splits. Hence, standard practice is to tune the number of splits or to grow a large tree and then prune it, i.e. collapsing the nodes that result redundant.

Once we have our tree, $\mathcal{T}$, with K leaves and depth D, we can express it as:

$$g\left(z_{i,t};\theta,K,D\right)=\sum_{k=1}^{K}\theta_k 1_{\{z_{i,t}\in C_k(D)\}}, \tag{2.15}$$

where $\{C_k(D)\}_{k=1}^K$ is the set of partitions of the input space.

In the CART framework, at each split we minimize the so-called l_2 impurity of the new branch:

$$H(\theta, C) = \frac{1}{|C|} \sum_{z_{i,t} \in C} (r_{i,t+12} - \theta)^2 , \tag{2.16}$$

with $|C|$ being the number inputs. When C is defined, we can see that the optimal choice of θ is $\theta = \frac{1}{|C|} \sum_{z_{i,t} \in C} r_{i,t+1}$, as we have already mentioned.

Interactions and nonlinearities are naturally introduced by decision trees, with a straightforward interpretation of their functioning. However, they have several weaknesses: splits are aligned with the axes, hence it can be tough to split two regions whose optimal border is at 45 degrees, they produce piece-wise constant outputs and their hard decisions make them prone to overfitting. This is why we only use them in ensemble methods.

2.3.5 (Extreme) Gradient Boosting Machine

The term *boosting* is used for any ensemble algorithm which works iteratively fitting a sequence of weak learners, that will be combined (e.g. summed) to build a strong learner. A weak learner is a regressor or classifier whose performance is only slightly better than random. In our case, weak learners will consist of decision trees.

The first and classic example is AdaBoost, see Freund and Schapire (1997). AdaBoost gives to each weak learner a weight, according to the training error of the step in which the learner is created. Moreover, after every step, it also gives a weight to every data point, corresponding to its current training error. Breiman (1984) noticed that boosting can be viewed as an optimization algorithm, selecting the appropriate cost function. Friedman (2001) introduced the first gradient boosting algorithm, calling it Gradient Boosting Machine (GBM).

In our regression setting, we want to learn the conditional expectation $g(\cdot)$, by minimizing the loss function in Equation (2.9). Suppose we want to perform S stages of boosting: at each stage s ($1 \leq s \leq S$), we will have a partial model g_s. To improve it, we want to add a new estimator h_s, such that:

$$g_{s+1}(z) = g_s(z) + h_s(z) = r. \tag{2.17}$$

Obviously, we can write

$$h_s(z) = r - g_s(z), \tag{2.18}$$

which shows that, at each stage, we are fitting the residuals. In other words, each model tries to correct the errors of the previous one.

If we simplify our notation of the loss function to:

$$\mathcal{L}(r, z) = \frac{1}{2}(r - g(z))^2, \tag{2.19}$$

we can easily see that boosting is a gradient descent algorithm, because:

$$h_s(z) = -\frac{\partial L}{\partial g} = r - g(z). \tag{2.20}$$

We will use a boosting system called XGBoost, introduced by Chen and Guestrin (2016). It gained a huge popularity because of its efficiency and effectiveness. On the data-science platform Kaggle, in 2015, 17 out 29 winning solutions were using XGBoost. XGBoost modifies the gradient boosting framework and the greedy algorithm for split finding. We leave details in the Appendix A.

We conclude this section introducing a fundamental regularization technique employed in Gradient Boosting. As with LASSO regression, we can add a shrinking parameter. The simplest way is to add a scaling constant to the importance of each tree that we add. Therefore, Equation (2.17) will become:

$$g_{s+1}(z) = g_s(z) + \nu h_s(z), \tag{2.21}$$

where the parameter ν is called the *learning rate* of the boosting process. It is straightforward to notice that a smaller learning rate will likely imply the need for a higher number of boosting stages S, but it can significantly enhance generalization power. Friedman (2001) found that this is also empirically true and that it is usually convenient to set values of $\nu < 0.1$ and to select S by early stopping.

Another possible regularization approach is subsampling, as suggested by Hastie et al. (2009), who also shows that it helps in building more accurate models. In the same perspective of VASA, we will use feature subsampling, as we think it can give a similar outcome in terms of variance reduction, but

with higher probability to get more uncorrelated trees, smoothing the impact of dominant predictors.

2.3.6 Random Forest

Random Forest is another popular ensemble methods, based on decision trees. They were introduced by Breiman (2001) as an evolution of the procedure called *bagging*, from bootstrap aggregating. Bagging consists in drawing B random samples (with replacement) from the input data, fitting for each one an independent regression tree and, in the end, averaging all the predictions of the B trees. In this way, the ensemble is able to reduce the variance of our models, without increasing the bias. This reduction of the variance might be harmed if many of the trees are correlated, which is likely to happen if there are some dominant predictors, that always take control of the first and most important splits. Random Forest attempt to solve this issue: it does not bootstrap samples but, at each candidate split, it uses only a random subset of the features.
As with the *Gain* for XGBoost (see Appendix A.2), we have a straightforward way to compute features importance also with Random Forests. The *Importance* of a feature will be the total reduction of impurity brought by this feature. This score will be normalized to have the sum across features of their *Importance* equal to 1.

2.3.7 Neural Networks

Neural Networks are among the most powerful algorithms in machine learning. They have gained huge popularity thanks to the great successes achieved in difficult fields, such as computer vision, recommender systems, natural language processing and in games, although it could be noticed that most of these problems can be seen as unsupervised or reinforcement learning, and not supervised learning. The field of Neural Networks is usually called *Deep Learning*, thanks to their ability to find hidden structures and (non-linear) interactions in the data. However, their strength is also their weakness, because their large parametrization makes also them prone to overfitting and difficult to understand. Their theoretical justification is mainly based on the theorem of universal approximation (Hornik et al. (1989), Cybenko (1989)), which claims that a feed-forward network can approximate arbitrary well real-valued continuous functions on compact subsets of $\mathbb{R}^n$, although it does not explain

how this networks could be constructed.

We will use the simplest form of neural networks, called *artificial* or *feed-forward*. They are made of basic units, called *neurons*, stacked in different layers:

- An *input* layer, that represents the data we feed in the model.
- One or more *hidden* layers, which create the interactions and non-linear transformations of inputs. Each neuron in a hidden layer applies a non-linear transformation, called *Activation Function*, to a linear combinations of the outputs of the previous layer.
- An *output* layer, that gives us our prediction. In a regression setting, it is made by a single neuron with a linear activation function.

When training a neural networks, our goal is to optimize the weights of the linear combinations.

In mathematical terms, given our set of P predictors $(z_1, \dots, z_P)$, we define a neural networks with L layers and activation function $f(\cdot)$. $K^l, (1 \leq l \leq L)$, is the number of neurons in the l-th layer and x_k^l is the output of the k-th neuron in the l-th layer. Exploiting the recursive structure of the network, the output of the single neuron can be written as:

$$x_k^{(l)} = f\left(\sum_{i=1}^{K^{(l-1)}} x_i^{(l-1)} w_{k,i}^{(l-1)} + w_{k,0}^{(l-1)}\right), \tag{2.22}$$

where $w_{k,i}^{(l)}$ is the weight corresponding to the i-th element of the $(l-1)$-th layer, for $i \geq 1$, while $w_{k,0}^{(l)}$ represents the intercept in the neuron.

The single neuron of the output layer will forecast:

$$g(z_{i,t}; w) = \sum_{i=1}^{K^{(L-1)}} x_i^{(L-1)} w_i^{(L-1)} + w_0^{(L-1)}. \tag{2.23}$$

We will now outline the most important steps of the optimization of neural networks. Rewriting Equation (2.22) in matrix form, we get:

$$\begin{aligned} z^{(l)} &= W^{(l)} x^{(l-1)} \\ x^{(l)} &= f\left(z^{(l)}\right), \end{aligned} \tag{2.24}$$

where $z^l \in \mathbb{R}^{K^{(l)}}$ is the vector of inputs of the layer l, $x^l \in \mathbb{R}^{K^{(l)}}$ is the vector of outputs, $W^l \in \text{Mat}\left(K^{(l)}, 1 + K^{(l-1)}; \mathbb{R}\right)$ is the matrix of weights between the layers $(l-1)$ and l and $f(\cdot)$ is applied to $z^{(l)}$ component wise.

In our regression context, we want to minimize the penalized l_2 objective function, i.e. the penalized sum of square residuals:

$$\mathcal{L} = SSR(W) + \phi(W; \cdot), \tag{2.25}$$

where ϕ is regularization term, which helps to keep under control the size of the weights and model complexity.

This problem is highly non-linear and non-convex, hence using standard gradient descent would be unfeasible. Standard practice is to use optimized versions of Stochastic Gradient Descent (SGD), which needs to compute the gradient only on a batch of the input data. More details on this and other ways to improve the training of networks are given below in Implementation Details.

The last issue is the computation of the gradient: what we have previously described is the Forward propagation, which is how the information goes from the input to the outputs. Note that, for the first epoch, weights are randomly initialized (see Implementation Details). Once we have our prediction, we can compute the loss and we need to compute the gradient to update our weights. This is done trough *Backpropagation*. In the output layer, we can compute the error term:

$$\delta^{(L)} = r - x^{(L)}, \tag{2.26}$$

and its gradient will be:

$$\nabla_{W^{(L)}} \text{SSR}(W) = \delta^{(L)} v^{(L-1)'}. \tag{2.27}$$

Then, we can go back layer by layer and compute first the error term:

$$\delta^{(l)} = f'\left(x^{(l)}\right) \odot \left(W^{(l+1)'} \delta^{(l+1)}\right), \tag{2.28}$$

where $\odot$ is the Hadamard product, and, in the end, the gradient:

$$\nabla_{W^{(L)}} SSR(W) = \delta^{(l)} v^{(l-1)'}. \tag{2.29}$$

Implementation details

To train neural networks in an efficient and meaningful way, it is fundamental to utilize a set of tools that allow to speed up optimization, reduce overfitting and improve generalization power. In this section, we will draw an outline of all of those we used:

- Learning Rate Shrinkage. As previously mentioned, we do not use a standard version of SGD. The main issue is that, when we approach the (local) optimum, if we make too big steps, we may jump out from the minimum and lose all the optimization done. The solution is shrinking the learning rate as the gradient goes to zero. In particular, we use the Adam (Adaptive Moment Estimation) algorithm from Kingma and Ba (2014). Adam updates the learning rate using bias-corrected estimates of the first and second moment of the gradient, controlling the decay through two parameters called β_1 and β_2.

- Early stopping. Early stopping addresses the problem of when stopping the optimization. Indeed, we talked about going backward and forward in tree, with each run called *epoch*, but we did not say when we have reached the solution. This is particularly important, because performing too many steps implies increasing the computational effort and potential overfitting, while not making enough steps could prevent the network from learning. Moreover, in simple cases, in can be proved that Early Stopping is equivalent to adding an l_2 penalization term in the objective function, but at a much lower computational costs. Classical Early Stopping means stopping the training when some metric does not improve anymore, for instance the validation error. In our case, we adopt an hybrid approach, treating the number of epochs as a tuning parameter, mainly to keep consistency in the length of training.

- Batch Normalization. As we have already said, in each SGD step, we only compute the gradient in a sample of the inputs, called batch. The problem is that the networks training is a dynamical process and the weights get frequently updated. This means that layers have to continuously adapt to new distributions of inputs, the so-called *Internal Covariate Shift*. Ioffe and Szegedy (2015) solved this issue through *Batch Normalization*: after each layer, Batch Normalization normalizes the batches and applies an affine transformation, whose scale and shift values are parameters to tune, to preserve the representative power of the layer.

- He Initialization and Ensemble. Given the non-convex nature of our problem, it is crucial to avoid poor local minima. A careful initialization of weights can help to tackle this risk. Although it is standard practice to sample initial weights from a normal distribution, several choices can be made on the mean and the variance to be used. He et al. (2015) showed that, if using the ReLU activation function, a clever choice is zero mean and variance $2/n_{in}$, where n_{in} is the number of inputs. This makes the variance of the input of each neuron equal to 1. Even with the He initialization, we have no guarantees that the single choice of weights is not problematic. Therefore, we use a simple ensemble approach, training the same model with different random seed multiple times. This method has been proven to be effective even with a relatively small number of independent models, see Hansen and Salamon (1990).

2.4 Model tuning

In this paragraph, we expand the topic, introduced in Section 2.2, of the hyperparameters to tune when we train each single model. Even though hyperparameters can be very different among models, the tuning always addresses the same general issue: keeping under control the model complexity, in order to reduce overfitting and increase out-of-sample performance. Obviously, more complex models tend to have more hyperparameters, but, in practice, the bigger effort required to tune them must be compensated with a higher performance.

2.4.1 Hyperparameters

LASSO

To train a LASSO model, we need to define the value λ for the weight of the penalization term in the loss function. The greater the λ, the more the weights will be shrunk towards zero.

VASA

In our VASA discussion, we left unanswered two questions: how many predictors should be used in each submodel (K_b) and how many submodels we need to employ (B). First of all, as in De Nard et al.

(2020), we fix the same number of predictors in every submodel, i.e. $K_b \equiv K$, for each $1 \leq b \leq B$. Moreover, given the low computation expense of training even a large number of OLS submodels, we fix B to be high enough to have a stable behaviour. Therefore, the only hyperparameter to tune remains K, the number of predictors we will sample in each submodel.

XGBoost

As already introduced in Section 2.3.5, the two most important hyperparameters in a gradient boosting model are the learning rate ν and the number of boosting steps S. Ideally, our tuning procedure should be able to select a large S when ν is small, and vice versa. This means that, if we give a smaller weight to each tree, we will train more of them to reach a similar level of training risk. XGBoost is an ensemble approach, hence we cannot forget that is composed by a sequence of decision trees. Although it is very effective in controlling model complexity while fitting the trees, as we show in Appendix A.2, the maximum depth of each tree is a fundamental parameter. Therefore, we treat it as tuning parameter, because it can have a major impact on the modelling capacity of the ensemble. In the end, we will tune also the percentage of features that we use to fit each tree, as we perform feature subsampling.

Random Forest

One of the strengths of Random Forest is that it usually produces good results without much need of tuning. This was true also in our case, hence we decided to fix the number of trees (as in VASA with the number of submodels) and the number of features that we use at each split, while we only tune the maximum depth of the trees, in order to control model complexity and the overfitting risk.

Neural Networks

Neural Networks usually require a significant effort in hyperparameters tuning, because their number can be high and the learning can be completely frustrated if the optimization does not work smoothly. First of all, we keep the structure constant, i.e. the number of layers and neurons, using a combination that has been proven to be effective in similar problems, especially by Gu et al. (2020a). The most

important hyperparameter becomes the learning rate, which controls the impact of each update of the weights, in particular the speed that we have when we approach a minimum in the optimization. Another relevant parameter for the optimization is the batch size: as we saw in Section 2.3.7, we evaluate the gradient only one batch at time. Standard gradient descent would use only one batch with all the input data, while standard SGD would use one sample for each batch. We choose one way in the middle, in order to keep the benefits from SGD, but without slowing down the learning too much.

Related to the optimization are other two tunable parameters: the weight of the l_1 penalization, that represent the ϕ in Equation (2.25), and the number of epochs, i.e. the number of time that we go through the network to update weights (see Section 2.3.7, Implementation Details).

2.4.2 From Grid Search to Bayesian Learning

Once we have decided what we need to tune, the next question question becomes how we can actually do it.

The simplest approach is usually to define a grid for each single hyperparameter and try all possible combinations on the validation sample, as explained in Section 2.2. It is feasible when we are sure of the intervals in which we want to look for our optimal choice and the computational effort required is acceptable. This can be true when the model is particularly quick to train or the number of combinations is low. Therefore, we will use this technique in our linear models, LASSO and VASA, and in Random Forest.

When the dimension of the grid search increases, grid search can quickly become unfeasible. A major achievement in this field has been the proof that, for complex models such as neural networks, random search 'over the same domain is able to find models that are as good or better within a small fraction of the computation time', as stated and demonstrated in Bergstra and Bengio (2012).

We can observe that, both grid and random search, have one important weakness: they do not have memory, i.e. at each step, they do not use the information that the combinations already tried could provide. In the end, they just try a certain set of combinations and pick the one with the lowest validation error. One way of exploiting this information is Bayesian optimization: we want to use the past information before selecting the next combination of hyperparameters to try. After a certain

number of steps, it might be clear that it is useless to explore certain regions, because they yield poor results. This means that we will need to perform less steps of optimization. Moreover, we do not have to specify a discrete grid, but we can just define a certain interval for every parameters, hence we can find solutions that we would have never found with our pre-defined grid. This is exactly what the Tree-structured Parzen Estimator (TPE) is able to achieve, as showed in Bergstra et al. (2011).

Tree-structured Parzen Estimator (TPE)

Following Bergstra et al. (2011), the mathematical framework that we will use is called Sequential Model-Based Global Optimization (SMBO), see Hutter et al. (2011) for details. The main intuition is that, when we have a function $f : \mathcal{X} \to \mathbb{R}$ that is costly to evaluate, we approximate it with a surrogate function which is cheaper to compute. At each step, the true function f will evaluate only the point x^* that maximizes the surrogate function. We will spend more time on selecting the hyperparameters to try, but we will save several calls to the expensive original function.
There two fundamental choices to be made now: the criterion used to optimize the surrogate function and how to model f through the observation history $\mathcal{H}$:

- The criterion is the Expected Improvement (EI), defined as the the expectation, under some probability model M of f, that f will negatively exceed a certain threshold y^*:

$$\mathrm{EI}_{y^*}(x) := \int_{-\infty}^{\infty} \max\left(y^* - y, 0\right) p_M(y \mid x) dy \tag{2.30}$$

- The TPE does not try to directly model $p(y \mid x)$, but defines $p(x \mid y)$ using two non-parametric densities :

$$p(x \mid y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{2.31}$$

  Here, it is fundamental to notice that '$l(x)$ is the density formed by using the observations $\left\{x^{(i)}\right\}$ such that corresponding loss $f\left(x^{(i)}\right)$ was less than y^* and $g(x)$ is the density formed by using the remaining observations' (see Bergstra et al. (2011)). Therefore, this is where the history of the previous steps comes into play.

Inserting Equation (2.31) through the Bayes' rule in Equation (2.30), we can get:

$$\mathrm{EI}_{y^*}(x) = \frac{\gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma \ell(x) + (1-\gamma) g(x)} \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}, \tag{2.32}$$

which show that, to have the highest improvement, we need points x with high probability under $l(x)$ and low probability under $g(x)$.

We will utilize the implementation of TPE given by the package *hyperopt*, see Bergstra et al. (2013).

2.5 Variables' Importance

Analysing variables' importance is crucial, because is the main way we have to understand what is driving the forecasts of our model. Given the relevance of the topic of *explainability* in the whole world of Machine Learning, in the last years there have been major improvements in the tools available to dig into the behaviour of complex models. In this section, we will explain one of the most recent and powerful approaches: SHAP values.

2.5.1 SHAP Values

SHAP (SHapley Additive exPlanations), introduced by Lundberg and Lee (2017), is a game-theoretic approach that allows to analyze the impact of each single data point on the model and to measure the interactions among features. In general, the main contribution of SHAP has been to provide a general framework to build *explanation models*, which are every simplified interpretable approximations of complex models. In this framework, they restrict to 'Additive feature attribution methods' and show that they include other popular algorithms, such as LIME (Ribeiro et al. (2016)) and DeepLift (Shrikumar et al. (2017)). Moreover, they prove that, with this playground, there is a unique solution that satisfies three important theoretical properties and provide fundamental algorithmic improvements to compute SHAP values in reasonable amounts of time, implemented in the *Python* package *SHAP* .

Available at https://github.com/slundberg/shap

We will now look at this points in more detail, following Lundberg and Lee (2017). Let f be the original model to explain and g the explanation model. All the considered approaches are called *local methods*, because they try to explain a single prediction based on a certain input x. The input x is usually mapped into a simplified version through a function $h_x : x = h_x(x^*)$. A local method should guarantee that $g\left(z'\right) \approx f\left(h_x\left(z'\right)\right)$ if $z' \approx x'$.

Definition 2.1 (Additive feature attribution methods)**.** Additive feature attribution methods have an explanation model that is a linear function of binary variables:

$$g\left(z'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i', \tag{2.33}$$

where $z' \in \{0,1\}^M$, M is the number of simplified input features and $\phi_i \in \mathbb{R}$. Therefore, ϕ_i represents the impact of the feature i. △

Let us now look at the three properties that we want to satisfy with our solution.

Definition 2.2 (Local accuracy)**.**

$$f(x) = g\left(x'\right) = \phi_0 + \sum_{i=1}^{M} \phi_i x_i', \tag{2.34}$$

which means that the explanation model $g(x')$ must exactly match the original model $f(x)$at least when the $x = h_x(x')$, with $\phi_0 = f\left(h_x(\mathbf{0})\right)$, i.e. the model forecast when all simplified inputs are set to zero. △

Definition 2.3 (Missingness)**.**

$$x_i' = 0 \Longrightarrow \phi_i = 0, \tag{2.35}$$

i.e. a feature with a null simplified input, cannot have a non-zero impact on the output. △

Definition 2.4 (Consistency)**.** Let $f_x\left(z'\right) = f\left(h_x\left(z'\right)\right)$ and $z'\backslash i$ be equivalent to $z_i' = 0$. For any couple of models f and f', if

$$f_x'\left(z'\right) - f_x'\left(z'\backslash i\right) \geq f_x\left(z'\right) - f_x\left(z'\backslash i\right) \tag{2.36}$$

for every input $z' \in \{0,1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$. △

The following theorem is the fundamental theoretical result: it shows that we have one and only one additive feature attribution methods satisfying the three properties.

Theorem 2.1. *There exists only one explanation model g of the form given in Definition 2.1 that follows Definitions 2.2, 2.3 and 2.4:*

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left[f_x\left(z'\right) - f_x\left(z' \backslash i\right) \right] \tag{2.37}$$

where $|z'|$ is the number of non-zero component in z', and $z' \subseteq x'$ represents all vectors z' whose non-zero components are a subset of the non-zero components of z'.

The theorem was proved by Young (1985), exploiting results from cooperative game theory, where the features' effects ϕ_i are known as Shapley values, from Shapley (1953).
The last step is to apply this framework to our features' importance problem. The SHAP values will be the solution of the Equation (2.37), with $f_x(z') = f(h_x(z')) = E[f(z) \mid z_S]$ and S is the set of non-zero indexes in z'. Note that we are implicitly defining $h_x(z') = z_S$ and then we approximate $f(z_S)$ with $E[f(z) \mid z_S]$, because a general model cannot handle inputs of varying dimension.
The rest of the work are computational optimizations that allow to compute SHAP values in a reasonable amount of time. We only highlight that the outputs are exact for tree-based models, while they are only approximate for neural networks, where the computational load is significantly higher. For the same reason, we can compute also SHAP interaction values only for tree-based models, defined as:

Definition 2.5 (SHAP Interaction Values)**.**

$$\Phi_{i,j}(f, x) = \sum_{S \subseteq \mathcal{M} \backslash \{i,j\}} \frac{|S|!(M - |S| - 2)!}{2(M-1)!} \nabla_{ij}(f, x, S), \tag{2.38}$$

where $i \neq j$ and

$$\begin{aligned} \nabla_{ij}(f, x, S) &= f_x(S \cup \{i,j\}) - f_x(S \cup \{i\}) - f_x(S \cup \{j\}) + f_x(S) \\ &= f_x(S \cup \{i,j\}) - f_x(S \cup \{j\}) - [f_x(S \cup \{i\}) - f_x(S)] \end{aligned}, \tag{2.39}$$

with $\mathcal{M}$ being the set of all input features. △

For all the details on the *TreeExplainer*, see Lundberg et al. (2018).

2.6 Portfolio Construction

The final goal of our exercise of our exercise it to exploit our predictions of assets' excess return to construct profitable portfolios. We will evaluate one long-short strategy for all the investment universes, while we only construct two different optimized strategies for the universal balanced dataset. In this section, we will briefly go through the construction processes of these portfolios.
In general, at each stock i in our portfolio, with $1 \leq i \leq N$, at time t, we assign a weight $w_{t,i}$. If weights satisfy:

$$\sum_{w_{t,i}<0} |w_{t,i}| = \sum_{w_{t,i}>0} |w_{t,i}| = 1, \tag{2.40}$$

their sum will be zero and the portfolio is called *dollar-neutral* and the total exposure is of two dollars. While, if we just impose that the sum of weights is equal to 1, the portfolio is called *fully invested* and the gross exposure is not bounded.

2.6.1 Quantile-Based Portfolio

The simplest way to construct a trading strategy is to rank the stocks according to our predictions, split them in B quantiles and

- buy the stocks in the best quantile, if we want a long-only strategy;
- buy the stocks in the best quantile and short those in the worst quantile, if we want a long-short strategy.

The vector of forecasts, at time t, is $\hat{r}_t = (\hat{r}_{t,1}, \ldots, \hat{r}_{t,N})'$, we define $\{(1), (2), \ldots, (N)\}$ as a permutation of $1, 2, \ldots, N$ such that stocks are correctly sorted and grouped in quantiles:

$$\hat{r}_{t,(1)} \leq \hat{r}_{t,(2)} \leq \ldots \leq \hat{r}_{t,(N)}. \tag{2.41}$$

We can now define the weights w_t^{LS} of the long-short (LS) strategy as:

$$\begin{aligned} w_{t,(1)}^{\text{LS}} = \ldots = w_{t,(d)}^{\text{LS}} &:= -1/d, \\ w_{t,(d+1)}^{\text{LS}} = \cdots = w_{t,(N-d)}^{\text{LS}} &:= 0, \text{ and} \\ w_{t,(N-d+1)}^{\text{LS}} = \ldots = w_{t,(N)}^{\text{LS}} &:= 1/d, \end{aligned} \tag{2.42}$$

where d is the largest integer number such that $n \geq N/B$. The portfolio is dollar-neutral and its return will be $\rho_t^{\text{LS}} = r_t' w_t^{\text{LS}}$. We have to highlight that, predicting only the ranked returns, when we short a stock, we do not have an estimate of its expected return. Even if it is in the worst quantile, it might have a positive expected return. Therefore, this can have a negative impact on our performance in periods when all the market goes very well. Predicting raw returns would allow to short stocks only if we predict a negative value.
With regards to the long-only (LO) strategy, the weights will be:

$$w_{t,(N-d+1)}^{\text{LO}} = \ldots = w_{t,(N)}^{\text{LO}} := 1/d, \tag{2.43}$$

it is fully invested and the return will be $\rho_t^{\text{LO}} = \boldsymbol{r_t}' \boldsymbol{w}_t^{\text{LO}}$. We can actually build a long-only strategy for every quantile, in order to check that their performance is monotonic, which would be a proof that our predictions are actually able to provide out-of-sample accuracy.

2.6.2 Efficient Sorting Portfolio

We now introduce the first optimized portfolio, proposed by Ledoit et al. (2019) and called *Efficient Sorting*. The idea is to optimize the portfolio, keeping the same value of expected return given by the quantile-based long-short strategy, but minimizing the variance, to get a new dollar-neutral long-short portfolio. The new problem can be formulated as:

$$\begin{aligned} & \min_{\boldsymbol{w}} \boldsymbol{w}' \hat{\Sigma}_t \boldsymbol{w} \\ \text{subject to} \quad & \hat{\boldsymbol{r}}_t' \boldsymbol{w} = \hat{\boldsymbol{r}}_t' \boldsymbol{w}_t^{\text{LS}} \text{ and} \\ & \textstyle\sum_{w_i<0} |w_i| = \sum_{w_i>0} |w_i| = 1 \end{aligned} \tag{2.44}$$

where $\hat{\Sigma}_t$ is an estimator of the covariance matrix of $\boldsymbol{r}_t$.

2.6.3 Covariance Matrix Estimation

Here we will give only a brief overview on the topic of covariance matrix estimation, as it is not the main topic of this work. We only note that it is one of the most ancient and well-known problems in the field of asset pricing, because of its importance and difficulty. The first historical establishment of the whole field of (mathematical) portfolio theory, by Markowitz (1952), already made clear that optimal weights depended also from the (inverse of the) covariance matrix. The problem has traditionally been that the covariance matrix cannot be observed and you need to perform an inversion in high dimensions. This is critical both from a computational point of view and because base estimators, such as the sample covariance matrix, tend to be already ill-conditioned. This led to the famous definition, given by Michaud (1989), of portfolio optimization as an 'error-maximization procedure'.

The research has focused on two different issues to tackle: the dynamic behaviour and the cross-section of the covariance matrix. On the former, the seminal work was done by the introduction of ARCH models, by Engle (1982), and their evolution, GARCH, by Bollerslev (1986), but these approaches suffered with the extension in high dimensions. In more recent year, one of the major developments has been the Dynamic Conditional Correlation (DCC) model of Engle (2018). On the latter, one of the most important contributions was Ledoit and Wolf (2004), that opened the way for the *shrinking* of the covariance matrix. Their intuition was to approximate the covariance matrix with a convex linear combination of a highly structured estimator, F, and the sample covariance matrix, S:

$$\hat{\Sigma}_{\text{Shrink}} = \delta F + (1 - \delta) S \tag{2.45}$$

, where $\delta \in [0, 1]$ is the shrinkage constant and its optimal value can be estimated analytically. It is evident that the name *shrinkage* comes from the fact that the estimator is shrunk toward the structured estimator F. It is also clear that the following evolution would have been to develop a non-linear (NL) shrinkage estimator: this was achieved by Ledoit and Wolf (2018), 'exploiting a deep connection between nonlinear shrinkage and non-parametric estimation of the Hilbert transform of

the sample spectral density'. This is the estimator we will actually utilize.
In the end, we just mention that the state-of-the-art model is the DCC-NL estimator, proposed by Engle et al. (2019), that combines the DCC model with non-linear shrinkage.

2.6.4 Markowitz Portfolio

Having a good estimator of the covariance matrix, the NL Shrinkage, we will exploit it also to build another optimized portfolio. We call it *Markowitz*, because it is a fully invested portfolio, where we fix our desired level of expected returns and we minimize the variance. Following De Nard et al. (2018), without short-sale constraints, we define the problem as:

$$\begin{aligned} & \min_{\boldsymbol{w}} \boldsymbol{w}' \hat{\Sigma}_t \boldsymbol{w} \\ \text{subject to} \quad & \hat{\boldsymbol{r}}_t' \boldsymbol{w} = \rho_t \text{ and} \\ & \sum_{w_i} w_i = 1, \end{aligned} \tag{2.46}$$

where ρ_t is our target return.

2.6.5 Performance Evaluation and Comparison

For each strategy, we will have a sequence of realized returns ρ_t, with $1 \leq t \leq T$. To assess its value, we will look at the average return, the standard error and their ratio, the Sharpe ratio.
In the end, we want to test statistical significance also for portfolio performances, as we did in Section 2.2.1 for predictive accuracy. We will employ two robust Sharpe ratio tests develop by Ledoit and Wolf (2008). The first one involves the use of a heteroskedasticity and autocorrelation robust (HAC) kernel to estimate the asymptotic covariance matrix of the two series of returns that we are using. The second one is a non-parametric method based on Studentized bootstrap. We bootstrap from the Student's t-test and from the bootstrap distribution we can compute quantiles to build confidence intervals and reject the null hypothesis if they don't contain zero. The p-value is the ratio of bootstrap statistics bigger than the sample statistic. To account for the time-series nature of our problem, we use the circular bootstrap of Politis and Romano (1992).

Chapter 3

Empirical Results

In this chapter we will show all our empirical results. We will utilize three different raw data sets: the Russell 1500 for United States stocks, the union of MSCI Europe and the 1000 biggest stock in Europe in Compustat for European stocks and the union of MSCI Japan and the 1000 biggest stock in Japan in Compustat for Japanese stocks. We will apply our algorithms to these three universes separately, but we will also build a *universal* dataset that we call 'balanced', because we keep only the stocks that have the full history of returns. This leaves us with 1105 stocks.
We build 101 stock-level predictive factors and 24 industry dummies from the Global Industry Classification Standard (GICS ®) of MSCI .

3.1 Universal Balanced Data Set

We begin with the universal balanced data set, because it is the one where we conducted the most in-depth analysis. The idea to retain only stocks with full history of returns is suggested by De Nard et al. (2020), although they also apply the rule on every covariate. The main reason for us is to be able to analyze stock-specific predictive power and to easily implement the optimized portfolios, that require the estimation of the covariance matrix.

See https://www.msci.com/gics

3.1.1 Predictive Accuracy

Tables 3.1 and 3.2 show average out-of-sample correlation and R^2 for our six algorithms.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
4,23%	1,51%	2,78%	3,99%	4,01%	2,57%

Table 3.1 – Average OOS Correlation

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
-3,41%	-1,19%	-1,52%	-0,40%	-0,03%	-1,15%

Table 3.2 – Average OOS R^2

Firstly, we note that correlation and R^2 are not completely according, especially OLS and VASA are better under the correlation perspective. Keeping the R^2 as the most important measure, we see that OLS struggles in this high dimensional setting, as in related works, see De Nard et al. (2020) for instance. All the methods have a negative R^2, even though XGBoost is very close to a null value and is the best performer, strictly followed by Neural Networks. In the middle, we have Random Forest and the 'regularized' linear methods, which are able to achieve two percentage points better than standard OLS, confirming the need for dimension reduction for linear algorithms in this setting. Figures B.1 and 3.1 show the time series of out-of-sample correlation and R^2 for each month that we predict. We can notice that one important quality of XGBoost is its robustness, as it never goes to very negative values and this makes the difference especially in the first year. Neural Networks pay a high price for the bad performance of the initial period, but since end of 2012, they have a limited variance with some important spikes. In Table 3.3 we look at the p-values of the pairwise Diebold-Mariano test, that we introduced in Section 2.2.1. The under-performance of OLS is significant against any other algorithm. The same for the over-performance of XGBoost and Neural Networks, while the difference between does not seem robust. Also the similar predictive power of LASSO, VASA and Random Forest is confirmed by high p-values.

In the end, we want to look at the specific correlation and R^2, that we defined in Equation (2.5). We can consider the correlation and R^2 above as an average measure of performance at each point of time. As every average, they might hide very different behaviours among stocks: in particular,

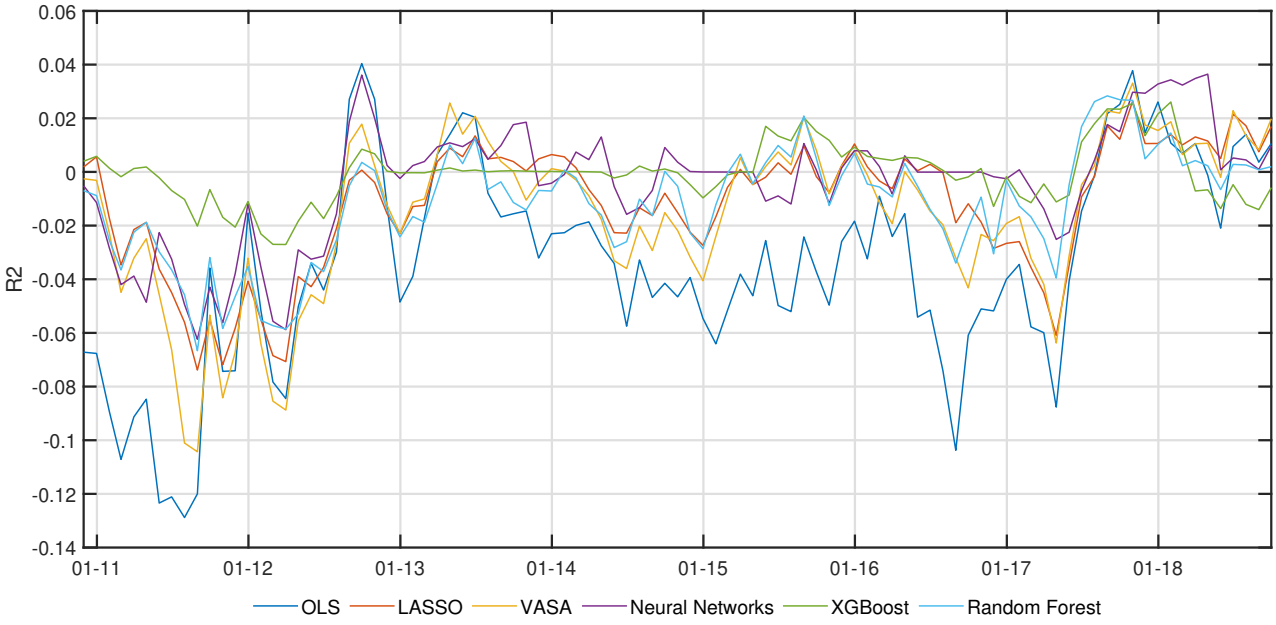


Figure 3.1 – OOS R^2 through time.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,08%	0,09%	0,00%	0,01%	0,12%
LASSO	-	-	10,79%	0,62%	1,62%	39,56%
VASA	-	-	-	0,10%	1,40%	20,18%
Neural Networks	-	-	-	-	27,33%	0,74%
XGBoost	-	-	-	-	-	0,05%
Random Forest	-	-	-	-	-	-

Table 3.3 – p-values of the Diebold-Mariano Test.

we would be worried by very negative outliers, i.e. stock that we predict in a completely wrong way. Tables 3.4 and 3.5 show average specific out-of-sample correlation and R^2.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
6,0%	3,5%	4,5%	6,4%	2,2%	3,6%

Table 3.4 – Average Specific OOS Correlation

Correlation numbers are in line with the standard average we saw above, while R^2 are significantly lower. This should be due that, when we compute the specific R^2, for each stock we have only around 100 values. Looking at correlation, we could say that the specific performance is quite in line with the

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
-23,3%	-20,7%	-21,1%	-19,7%	-15,9%	-20,0%

Table 3.5 – Average Specific OOS R^2

cross-sectional one. More information can be found with the boxplots in Figures B.2 and B.3, where we show the distribution of specific R^2 for each algorithm. We see the same discrepancy between correlation and R^2 that we have on cross-sectional results. The distribution of specific correlation is almost symmetric and without outliers, while the R^2 gives us a significant amount of stocks with very negative outliers. Even though these extreme values may be due to the computation of R^2 with few data points, we have another confirmation of the XGBoost being the best performer, as its worst R^2 is two points better than those of the other algorithms.

3.1.2 Portfolios' Performances

We will now analyze the performance of the portfolios that we build on top of our predictions.

Deciles and Long-Short Portfolios

Firstly, we look ad decile-based portfolios, they are described in Section 2.6.1, where we fix $B = 10$. Tables 3.6 and 3.7 show the out-of-sample performance for the Long-Only, where we buy the best decile, and Long-Short portfolios where we buy the best decile and sell the worst one, according to the predictions of each algorithm.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	15,8%	14,3%	14,0%	13,1%	15,6%	13,9%
Volatility	14,8%	14,0%	14,2%	14,1%	13,9%	13,9%
Sharpe Ratio	1,069	1,024	0,984	0,929	1,118	1,002

Table 3.6 – Long-Only, based on the best Decile D10, Portfolios' performances. Average returns and volatilities are annualized.

In general the outcome is not astonishing: in the Long-Short case, only XGBoost and OLS are able to achieve a Sharpe ratio higher than 0.9, with the latter result that is very surprising, given its predictive accuracy. To understand where this low performance is coming from, we can look at the

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	7,80%	5,40%	5,46%	4,01%	7,18%	6,01%
Volatility	8,34%	8,32%	8,94%	8,35%	7,95%	8,09%
Sharpe Ratio	0,935	0,650	0,611	0,481	0,903	0,743

Table 3.7 – Long-Short Decile-based Portfolios' performances. Average Returns and Volatilities are annualized.

two components of the strategy. The performance of the Long-side is actually better, with Sharpe ratios close or above one for all the methods: average returns are at decent levels, but the volatility is pretty high. XGBoost is the best with an average annualized return of 15.6% and a Sharpe ratio of 1.12. It is evident that shorting the worst decile is ruining our performance, as we were suspecting in Section 2.6.1, because we are not able to detect low performing stocks and we do not know if we are predicting a positive or negative return. Looking at all deciles in detail, shown in Table B.1, we can see that, indeed, average returns of the predicted worst decile go from 7.9% to 9.1%, which are obviuously too high to be shorted. However, we see that average returns are usually increasing with the deciles, this means that are predictions are able to discern at least best performing stocks.
Tables 3.8 and 3.9 show the p-values for the pairwise comparisons of the Sharpe ratios. In this case, all the p-value are pretty high, as it could be expected, since Sharpe ratios are on similar ranges on both strategies. The only lowest p-values are for the D10 portfolios of XGBoost against LASSO and VASA, which is reasonable as they are the only two methods with a Sharpe ratio lower than one. We do not report results for the HAC Sharpe ratio test, as they are almost identical to the bootstrap version; this holds also for all the other portfolios.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	73,5%	22,4%	39,3%	77,2%	70,1%
LASSO	-	-	14,0%	41,1%	41,3%	82,4%
VASA	-	-	-	82,0%	10,0%	26,1%
Neural Networks	-	-	-	-	17,4%	46,9%
XGBoost	-	-	-	-	-	20,4%
Random Forest	-	-	-	-	-	-

Table 3.8 – Studentized Bootstrap p-values for the D10 Long-Only Portfolio.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	35,1%	31,1%	19,6%	91,6%	56,5%
LASSO	-	-	97,8%	65,5%	24,8%	56,1%
VASA	-	-	-	61,9%	34,3%	61,1%
Neural Networks	-	-	-	-	23,0%	40,1%
XGBoost	-	-	-	-	-	48,1%
Random Forest	-	-	-	-	-	-

Table 3.9 – Studentized Bootstrap p-values for the Long-Short Portfolio.

Efficient Sorting Portfolios

Table 3.10 presents results for Efficient Sorting portfolios, whose construction is described in Equation (2.44), using the Analytical Shrinkage technique for covariance estimation, see Section 2.6.3.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	8,61%	4,94%	7,02%	6,26%	5,61%	4,15%
Volatility	4,42%	6,10%	6,07%	6,40%	4,41%	5,25%
Sharpe Ratio	1,948	0,809	1,158	0,978	1,274	0,791

Table 3.10 – Efficient Sorting Portfolio performances. Average Returns and Volatilities are annualized.

All the portfolios show a lower volatility and a higher Sharpe ratio than the simple decile-based Long-Short portfolios. In particular, the Sharpe ratio doubles for OLS and Neural Networks.
In this case, the over-performance of the OLS is actually statistically significant against the other methods, as we can see in Table 3.11.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,4%	4,2%	0,6%	14,6%	1,0%
LASSO	-	-	4,6%	63,5%	19,6%	95,2%
VASA	-	-	-	55,7%	77,0%	22,0%
Neural Networks	-	-	-	-	49,7%	56,7%
XGBoost	-	-	-	-	-	17,0%
Random Forest	-	-	-	-	-	-

Table 3.11 – Studentized Boostrap p-values for the Efficient Sorting Portfolio.

Markowitz Portofolios

The second optimized portfolio we build for the universal balanced data set is the Markowitz one, described in Section 2.6.4. We set as target return ρ as the average return of the stocks ranked in the best quintile by our predictions. The summary of results is shown in Table 3.12.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Returns	22,3%	21,3%	22,5%	21,9%	21,0%	21,3%
Vol	20,8%	20,8%	20,9%	20,6%	20,7%	20,6%
Sharpe Ratio	1,069	1,026	1,075	1,060	1,013	1,036

Table 3.12 – Markowitz Portfolio performances. Average Returns and Volatilities are annualized.

As expected, we have significantly higher values of returns and volatilities than the previous portfolios. This happen because we set a pretty ambitious target return, therefore the optimization needs to place big bets to achieve this level of expected returns. Obviously, this means that also the out-of-sample volatility will increase. Values are almost identical across algorithms, because here we are using predictions only to set the expected target. These targets will be pretty close at each point of time and, after we set them, the optimization runs in the same way independently of the original method. For these reasons, we do not test the statistical significance of Sharpe ratio differences.

3.1.3 Variables' Importance

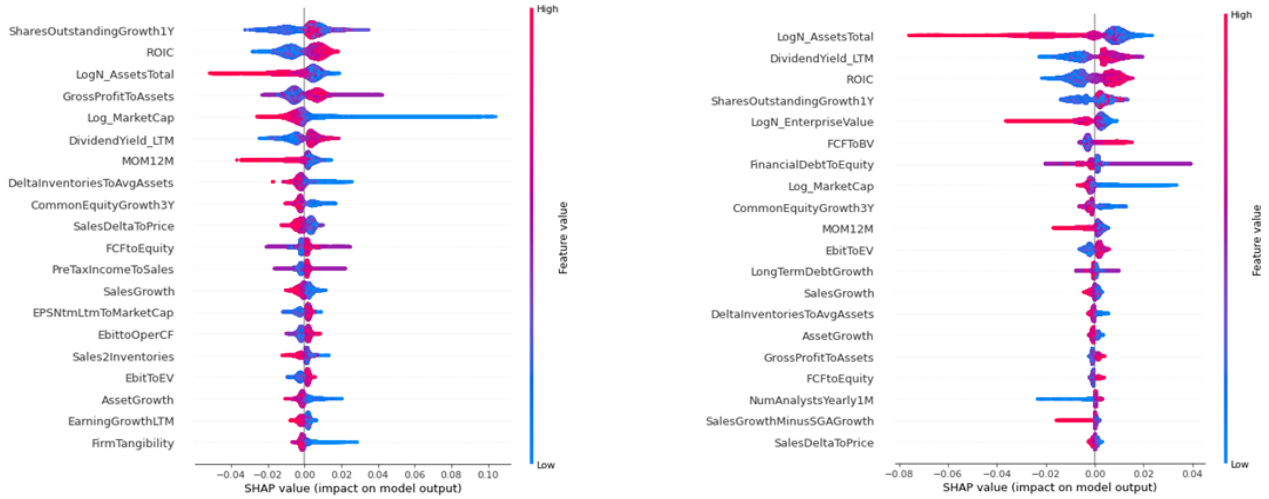
At this stage, we have analyzed the performance of our models from both the predictive power and economic value perspectives. Now, we want to understand which factors have been identified as most important and, ideally, how they affected the model. These are the measures that we will consider:

- OLS: regression coefficients;
- LASSO: regression coefficients;
- VASA: average regression coefficients across submodels, see Section 2.3.3;
- Neural Networks: SHAP values, see Section 2.5;
- XGBoost: Gain, see Appendix A.12, and SHAP values;

- Random Forest: Importance, see Section 2.3.6, and SHAP values.

In Figure B.4 we show the outcome for linear methods, with the average of both raw coefficients and their absolute value. In Figure B.5 we show the averages for the remaining algorithms. In general, we see a high degree of agreement among methods on the most important variables: Dividend Yield and Market Cap are always in the top three, except for OLS. In lower positions there is more variability, even if some variables are quite common, for instance ROIC, Earning-Price and Book-Price ratios. Linear methods give more importance to industry dummies. Random Forest has a significantly higher difference between the first and the second most important variable: this is very clear comparing the SHAP values with XGBoost and Neural Networks, as they are the same measure.
XGBoost and Random Forest have two different metrics, hence it may be interesting to see if they tend to give similar outputs. For the former, Gain and SHAP give almost overlapping features in the top ten, but positions are not always matching, see Figures B.5b and B.5c. For the latter, instead, the top eleven variables are the same, and only two of them have exchanged positions, see Figures B.5d and B.5e.
In the end, we want to leverage more the information given by SHAP values, through summary and dependency plots. In Figure 3.2, we see the SHAP summary plot for the XGBoost and Random Forest model trained in May 2018. They show, for the twenty most important predictors, the SHAP values of every feature for every sample (each dot is prediction). This gives us a clear picture of the distribution of the impacts each feature has on the model output. The color represents the feature value (red high, blue low). For instance, a high ROIC increases the predicted ranked returns, while a high Momentum or Market Cap decreases it.

Comparing XGBoost and Random Forest, we can check the each variable has the same qualitative impact on model. It would be problematic if, for instance, Market Cap had a positive impact on one model and a negative one on the other. We see that is actually true, but, obviously, the specific shape and tails of SHAP values can be slightly different. If we look at the growth of outstanding shares, the shape of the points is pretty similar, but XGBoost is relying more on it, indeed SHAP value have bigger absolute values and it is the most important factor, while it is only forth for Random Forest. We leave the summary plot for Neural Networks in Appendix B.6: for them, we are not able to



(a) XGBoost **(b)** Random Forest

Figure 3.2 – SHAP Summary plots for tree-based methods trained as of May 2018.Each dot represents a single prediction.

compute interaction effects, hence we only see linear relationships between the SHAP values and the features. This explains why we see only bars, that are symmetric around the mean.

Moving to interactions, we can show more in detail the dependency plot of each feature and we can add the interaction effect with another feature. We show some of these plots for the XGBoost model trained as of May 2018 in Figure 3.3. Here, it is even easier to visualize and understand the non-linear impacts of the predictors. Looking at ROIC plots, we see how the dependency is linear only when ROIC is around its mean. When we add the interaction of Momentum (MOM12M), we can clearly notice that the impact of ROIC is amplified when Momentum is high (red points) and reduced when Momentum is low (blue points).

The dependency plots for the the equivalent Random Forest model are shown in Appendix B.7. Sticking to the ROIC reference, we note how the relationship here tends to have a bigger linear region, but the general shape is highly comparable to XGBoost. For Neural Networks, these plots would be all straight lines, as they are missing interactions effects.

We produced these analyses for each model trained. Obviously, it is impossible to show or comment them all, but in practice, especially when we have interaction values, they provide invaluable insights into the model behaviour, that simple metrics such as Gain or Importance do not provide. We could also compare the dependency plots for a certain feature across time, to see if its impact has changed

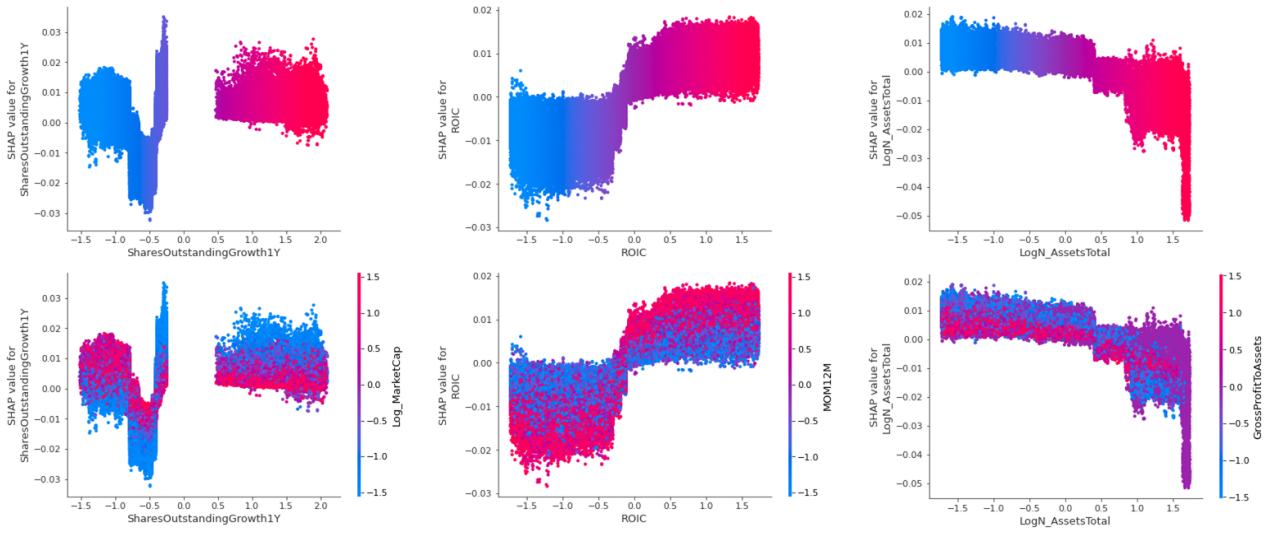


Figure 3.3 – SHAP Dependency Plots of the top 3 variables for XGBoost. The line above shows the SHAP values of a certain feature against the feature itself. In the line below, we add also the interaction with another feature, which is chosen as the one with the highest correlation of SHAP values in a vertical slice of the SHAP dependency plot. Each dot represents a single prediction.

and if this is matching the factor performance in reality or different market conditions. The same holds for different training windows. We do not report the evolution of features' importance through time, because the most relevant are usually very persistent.

3.1.4 Hyperparameters

We have described all the hyperparameters that we tune in Section 2.4.1. We show in Table 3.13 the ranges in which we perform our searches:

We report the combinations chosen for each trained model in the Appendix, Figures B.8, B.9 and B.10. We only note that, for LASSO and VASA, it could seem that we did not leave enough freedom to hyperparameters. But, in practice, too high values of the $L1$ penalization and too low values of the number of predictors led always to very poor performance, even though they were sometimes picked when the validation set was probably unreliable. In general, we can see an increase in model complexity during the last two years across all models. Comparing with the out-of-sample R^2 through time in Figure 3.1, we see that this led first to a drop in performance at the beginning of 2017, but after mid 2017, we have had some of the best results of the whole backtest.

ALGORITHM	HYPERPARAMETR	RANGE	SEARCH
LASSO	L1 Penalization Weight	$[10^{-4},10^{-2}]$	Grid
VASA	Number of Sub-models	100	Fixed
	Number of Predictors	[20,40]	Grid
Random Forest	Number of Trees	500	Fixed
	Maximum Depth	[3,6]	Grid
Neural Networks	Number of Layers	3	Fixed
	Number of Neurons	32-16-8	Fixed
	Learning Rate	$[\exp(-7), \exp(-4.5)]$	Bayesian
	L1 Penalization Weight	$[\exp(-11.5), \exp(-5)]$	
	Epochs	[13,17]	
	Batch Size	{32,64}	
	Dropout Rate	[0,0.175]	
XGBoost	Number of Trees	[10,250]	Bayesian
	Learning Rate	$[\exp(-8), \exp(-4)]$	
	Maximum Depth	[3,7]	
	Columns Dropout Rate	[0.025, 0.15]	

Table 3.13 – Hyperparameters ranges.

3.2 European Data Set

The first region-specific data set that we present is the European one. We remind that in the three 'raw' datasets we keep all the stocks and not only those with the fully history of returns, as in the Universal analysis.

3.2.1 Predictive Accuracy

We report average out-of-sample Correlation and R^2 in Tables 3.14 and 3.15.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
12,9%	11,7%	12,2%	14,2%	16,4%	16,2%

Table 3.14 – Average OOS Correlation

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
0,58%	1,39%	1,19%	1,76%	2,48%	2,45%

Table 3.15 – Average OOS R^2

Similarly to the Universal data set, the OLS is the worst performer in terms of R^2, although it looks better from a correlation perspective. The regularized linear methods are a small step above and not at the same level of Neural Networks. Tree-based methods look out-performing, while the difference between them is very small.
Figures B.11 and 3.4 report Correlation and R^2 for each month. The ranking among algorithms are rather constant over time. We only highlight a spike in Neural Networks' accuracy at the beginning of 2013 and in the very last months, as in the Universal case.

To assess the statistical significance of the differences in predictive power, we look at the p-values of pairwise Diebold-Mariano tests in Table 3.16. Firstly, we can note that XGBoost and Random Forest have low p-values against all the other methods, hence their over-performance is robust. While, between them, the small difference has a high p-value, as we could expect. Neural Networks find confirmation of their superiority to OLS and VASA, but they struggle with LASSO, with a p-value of 9.16%. Among linear methods, the performance of VASA, although more than double than OLS, does not have a strong statistical significance, while LASSO has more relevant p-values even though

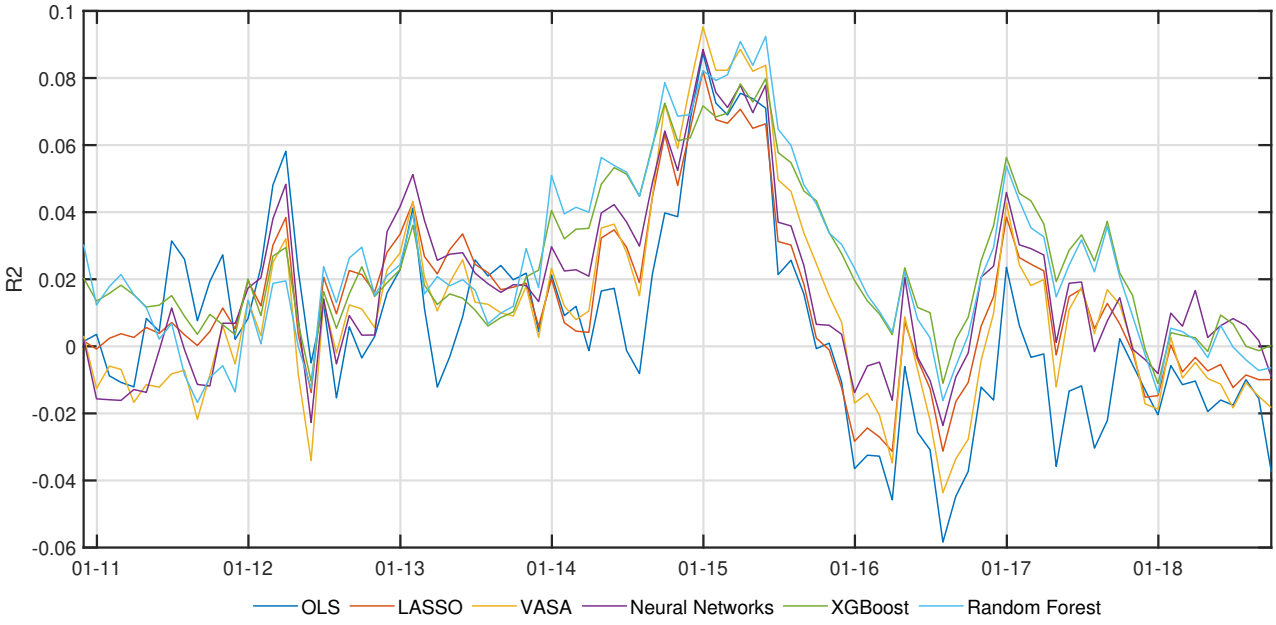


Figure 3.4 – OOS R^2 through time.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	1,26%	11,57%	0,74%	0,09%	0,21%
LASSO	-	-	30,36%	9,16%	0,66%	1,64%
VASA	-	-	-	1,71%	0,02%	0,00%
Neural Networks	-	-	-	-	2,98%	5,07%
XGBoost	-	-	-	-	-	39,25%
Random Forest	-	-	-	-	-	-

Table 3.16 – p-values of the Diebold-Mariano Test.

its R^2 is only 0.2% higher.

For all the three region-specific data sets, we do not report specific Correlation and R^2, because they do not have a fixed number of stocks and there are stocks that go in and out of the universe. Therefore, it would not make much sense comparing results that come from different number of data points.

3.2.2 Portfolios' Performances

For the same reasons, in the region-specific data sets, we will only look at decile-based portfolios, since the computation of the covariance matrix with a varying set of stocks would be complicated.

We always fix B=10 (see Section 2.6.1) and construct our deciles and the Long-Short Portfolios. Table 3.17 shows the results of Long-Only strategies, while Table 3.18 reports the Long-Short portfolios. Table B.2 in the Appendix gives details for all the deciles.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	17,5%	17,1%	17,6%	18,7%	19,2%	19,8%
Volatility	16,4%	16,2%	16,2%	16,3%	17,0%	16,6%
Sharpe Ratio	1,066	1,057	1,085	1,148	1,130	1,194

Table 3.17 – Long-Only, based on the best Decile D10, Portfolios' performances. Average returns and volatilities are annualized.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	12,9%	10,0%	11,2%	12,5%	13,2%	13,6%
Volatility	6,4%	8,6%	9,3%	8,0%	8,4%	8,2%
Sharpe Ratio	2,005	1,162	1,205	1,557	1,576	1,660

Table 3.18 – Long-Short Decile-based Portfolios' performances. Average Returns and Volatilities are annualized.

The Long-Only outcomes are in line with predictive power of each algorithm. In particular average returns follow the ranking of accuracy, while volatilities are almost all on the same level. The only exceptions seems XGBoost, which has a higher volatility and, therefore, a Sharpe ratio lower than Neural Networks.

In the Long-Short portfolios we have the same pattern, but with the big surprise of OLS performance. OLS is able to obtain a much lower volatility with a good level returns, hence the best Sharpe ratio with a good margin over tree-based methods. Looking at the performance of the worst decile that we short, see Table B.2, we can notice that OLS has the best D1, with an average return of 4.9%, while the second best has 5.9%. This implies that when we add the short leg, OLS is rewarded for being more precise in predicting 'bad' stocks.

In this case, it is particularly interesting to analyse the statistical significance, as OLS *exploit* in the Long-Short portfolio could have not been guessed from its accuracy. Studentized Bootstrap p-values for the two strategies are shown in Table 3.19

As in the Universal case, we do not report p-values based on the HAC standard errors test, because they are perfectly in line with the Studentized Bootstrap. Given the very similar Sharpe ratios,

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	65,1%	84,2%	24,0%	40,3%	13,4%
LASSO	-	-	82,8%	6,0%	10,8%	1,8%
VASA	-	-	-	18,8%	32,1%	10,8%
Neural Networks	-	-	-	-	77,2%	61,1%
XGBoost	-	-	-	-	-	13,4%
Random Forest	-	-	-	-	-	-

Table 3.19 – Studentized Bootstrap p-values for the D10 Long-Only Portfolio.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	2,2%	2,8%	16,4%	34,1%	45,1%
LASSO	-	-	88,6%	7,6%	15,6%	7,6%
VASA	-	-	-	4,8%	15,8%	3,8%
Neural Networks	-	-	-	-	95,4%	68,5%
XGBoost	-	-	-	-	-	54,7%
Random Forest	-	-	-	-	-	-

Table 3.20 – Studentized Bootstrap p-values for the Long-Short Portfolio.

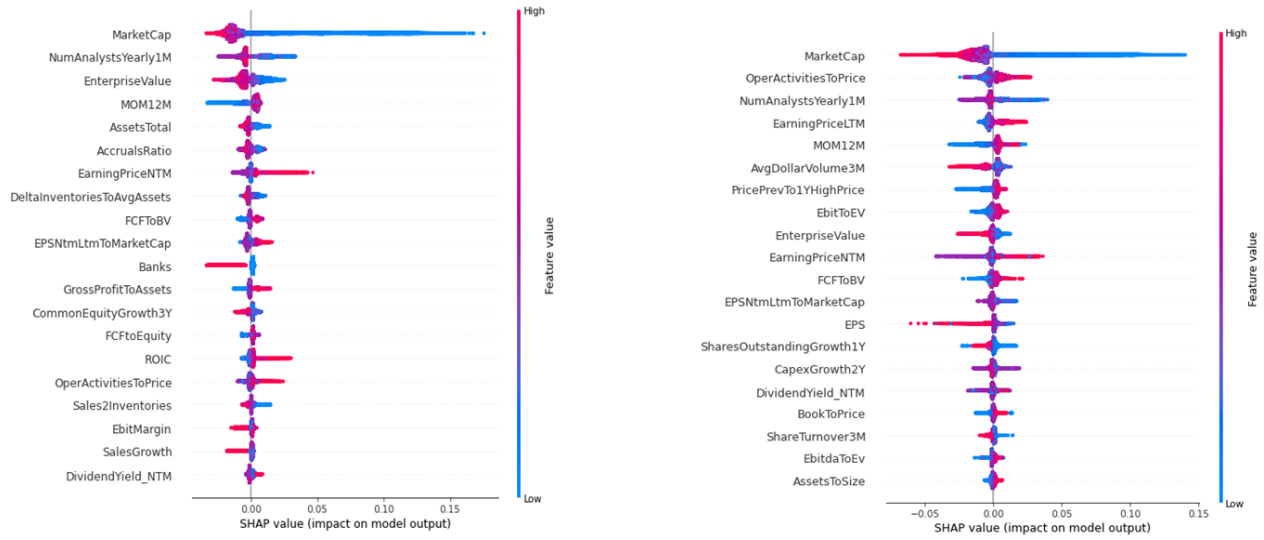
Long-Only p-values are quite high. Random Forest, the best performer, has slightly more significant p-values, but it achieves less than 10% only against LASSO, the worst performer. With Long-Short portfolios, differences were a bit more large, but the picture is anyway equal and, excluding OLS, again only Random Forest is able to have some good margin, when compared with LASSO and VASA. The anomaly of OLS does not seem to be confirmed: p-values are low only against the other linear-methods, but they do not provide significance with non-linear algorithms. Therefore, it is very likely that this low variance achieved by OLS is mainly due to randomness.

3.2.3 Variables' Importance

In this European data set the level of performance is rather different than in the Universal one, hence we want to analyze what are the factors that manly drove this difference. For a list of measures used, see Section 3.1.3. We remind that for region-specific universes, we make an equal-weighted average of a model trained on expanding window and another on a rolling window. Therefore, when we will talk about averages of variables' importance, they are averaged both across training windows and through time.

In Figure B.12 we show the averages of raw coefficients and their absolute value, while in Figure B.13, we report the averages of the metrics we use for non-linear algorithms. The first thing to notice is the dominance of Market Cap as most important variables: it is first in every algorithm with every metric, except for OLS, where it is anyway the only non-industry dummy in the top twenty. Moreover, the difference between Market Cap and the second-ranked variable is always rather high and this is particularly extreme for Random Forest, as it was in the Universal case. Other variables that are common among the best twenty are the Ebit-Ev ratio, one-year Momentum, Earning-Price ratio and Free Cash Flow-Book Value ratio. However, also other size-related features, such as Enterprise Value or the Number of new analysts in one year, are often important.

We now look at SHAP plots of model trained as of May 2018, to get more insights on the behaviour of our non-linear models. With regards to Market Cap, clearly we see negative weights in linear methods, hence we expect the same in non-linear ones. This is confirmed by SHAP summary plots, that we show in Figure 3.5 for XGBoost and in Figure B.14 for Neural Networks, but with an important difference. For XGBoost and Random Forest, we can see that the impact of Market Cap is not symmetric, because small caps are highly promoted, i.e. they are predicted to have high returns, while big caps are less penalized penalized in proportion. This asymmetry is even more



(a) Rolling Window **(b)** Expanding Window

Figure 3.5 – SHAP Summary plots for XGBoost models trained as of May 2018.

clear from SHAP Dependency plots, that we report in the Appendix, see Figures B.15 and B.16

for XGBoost, Figures B.17 and B.18 for Random Forest. In this particular moment, May 2018, size-related variables were even more dominating than on average. However, some of them, such as Enterprise Value, have a more linear impact, and others, such as the Number of Analysts Growth in one year, only depends on the sign.

In the end, we note that, except for dominant predictors, models trained on rolling and expanding windows can provide a good level on diversity on the set of most important features and their relative important. Hence, this model averaging can be a good way to increase stability of predictions and avoid too big bets on a certain predictor, if there is not a strong agreement among windows.

3.2.4 Hyperparameters

In conclusion of the European data set discussion, we briefly look at model tuning. For the general framework, see again Section 2.4.1. In region-specific data sets we use slightly different search ranges, that we report in Table 3.21. In these cases, we give slightly more freedom to hyperparameters, as we can bear more model complexity, while in the Universal dataset overfitting was more incumbent.

ALGORITHM	HYPERPARAMETER	RANGE	SEARCH
LASSO	L1 Penalization Weight	$[10^{-4},10^{-1}]$	Grid
VASA	Number of Sub-models	100	Fixed
	Number of Predictors	[20,60]	Grid
Random Forest	Number of Trees	500	Fixed
	Maximum Depth	[3,6]	Grid
Neural Networks	Number of Layers	3	Fixed
	Number of Neurons	32-16-8	Fixed
	Learning Rate	$[\exp(-7), \exp(-4.5)]$	Bayesian
	L1 Penalization Weight	$[\exp(-11.5), \exp(-6)]$	
	Epochs	[13,17]	
	Batch Size	{32,64}	
	Dropout Rate	[0,0.175]	
XGBoost	Number of Trees	[10,350]	Bayesian
	Learning Rate	$[\exp(-8), \exp(-3.5)]$	
	Maximum Depth	[4,9]	
	Columns Dropout Rate	[0, 0.15]	

Table 3.21 – Hyperparameters ranges.

All the hyperparameters for each model are reported in the Appendix in Figure B.19 for LASSO (20 steps of grid search, values logarithmically spaced), VASA (grid search on all integer values) and Random Forest (grid search on all integer values), Figure B.20 for Neural Networks (15 steps of Bayesian search) and Figure B.21 for XGBoost (20 steps of Bayesian search). In unreported results, we noticed that the average standard deviation of validation error is always at least twice higher for Neural Networks than for XGBoost. This is an indication that Neural Networks are more sensitive to hyperparameters choices, with important implications: they might be less reliable both when the computational resources to tune them is limited and when the validation set may be not well representing the testing set.

We highlight another benefit of averaging models trained on different time-windows: we are actually averaging two different models and in many cases they can have very different levels of model complexity, see for instance the Maximum Depth of Random Forest in Figure B.19c. Therefore, we can smooth extreme choices of hyperparameters by one single model and have a more stable behaviour through time.

3.3 Japanese Data Set

The second region-specific universe that we consider is made by Japanese stocks.

3.3.1 Predictive Accuracy

Tables 3.22 and 3.23 show average out-of-sample correlation and R^2 for the six algorithms in the Japanese universe.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
15,0%	8,1%	13,3%	15,3%	17,5%	16,2%

Table 3.22 – Average OOS Correlation.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
1,36%	0,25%	1,35%	1,79%	2,86%	2,42%

Table 3.23 – Average OOS R^2.

In this case, LASSO is the worst performer, while OLS is at the same level of VASA. As in the European universe, Neural Networks are one step above linear methods, but also one below tree-based methods. XGBoost is over-performing Random Forest, with an R^2 of 2.86% against 2.42%. The evolution of out-sample metrics for each month are given in Figures B.22 and 3.6.

We notice also here more variability in the rankings in the first months, after which XGBoost and Random Forest lead the way almost always. Neural Networks have a quite big variance, with big spikes but also big drops, as the one in the latest dates.
The p-values of the Diebold-Mariano tests, shown in Table 3.24, give us some interesting insights. In particular, we can see that the difference between XGBoost and Random Forest is more significant than those between them and Neural Networks. This is surprising given the lower Correlation and R^2 of Neural Networks. Therefore, from this point of view, we cannot claim with confidence a true superiority of tree-based models. Nonetheless, the test confirms that linear-methods are really under-performing and LASSO is particularly poor in this case.

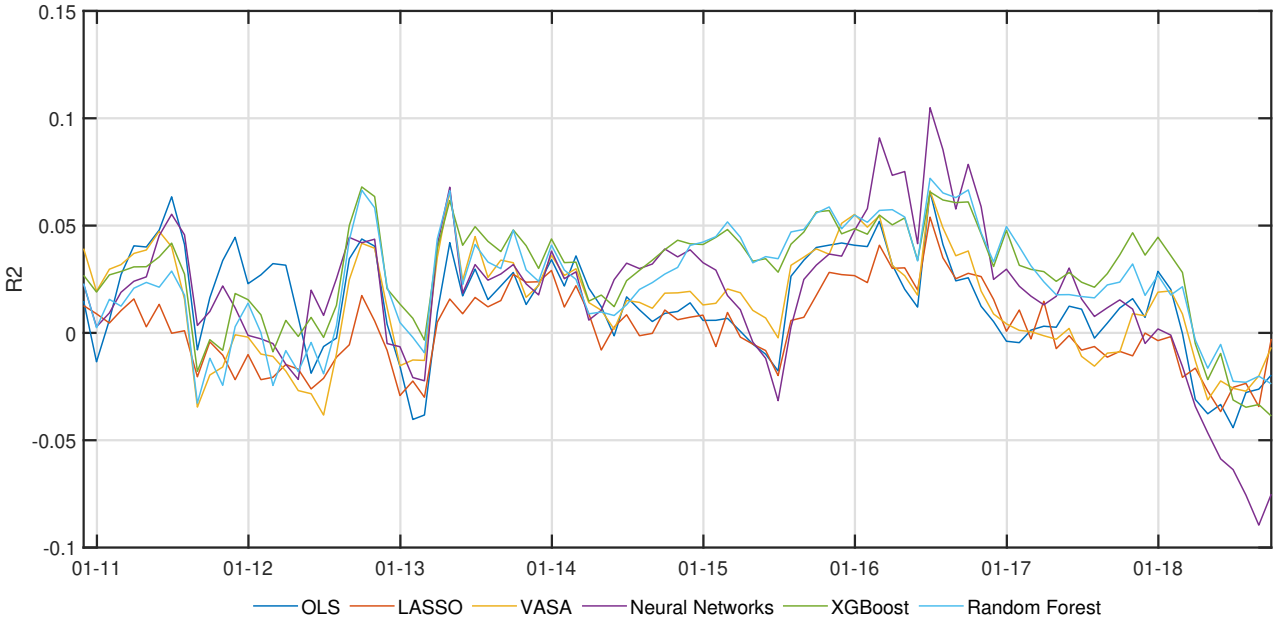


Figure 3.6 – OOS R^2 through time.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,73%	39,89%	8,05%	0,45%	8,16%
LASSO	-	-	0,00%	0,00%	0,00%	0,00%
VASA	-	-	-	5,97%	0,00%	0,42%
Neural Networks	-	-	-	-	8,51%	36,81%
XGBoost	-	-	-	-	-	4,56%
Random Forest	-	-	-	-	-	-

Table 3.24 – p-values of the Diebold-Mariano Test.

3.3.2 Portfolios' Performances

We will now look at how our predictions translate into economic performance, through the quantile-based portfolios described in Section 2.6.1, again with B=10, i.e. considering deciles.
Tables 3.25 and 3.26 show average returns, volatility and Sharpe ratios for Long-Only and Long-Short portfolios.

The Long-Only strategies provide slightly better outcomes than what we saw in the Universal and European universes, although at the same level of magnitude. The ranking of algorithms is not completely matching with predictive power: surprisingly, Random Forest is the worst performer and also XGBoost is just above LASSO. The picture changes when we go the Long-Short strategy,

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	15,3%	14,1%	15,6%	16,0%	16,1%	14,8%
Volatility	12,8%	12,4%	12,1%	13,2%	13,8%	13,6%
Sharpe Ratio	1,195	1,137	1,285	1,212	1,168	1,088

Table 3.25 – Long-Only, based on the best Decile D10, Portfolios' performances. Average returns and volatilities are annualized.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	10,6%	6,7%	9,8%	10,5%	11,6%	10,4%
Volatility	7,5%	6,6%	8,1%	8,3%	6,6%	6,3%
Sharpe Ratio	1,412	1,010	1,214	1,267	1,754	1,659

Table 3.26 – Long-Short Decile-based Portfolios' performances. Average Returns and Volatilities are annualized.

where tree-based models re-take their usual leadership, being the only ones which are are really able to boost the Sharpe ratios with the shorting. The increase for OLS is marginal, null for Neural Networks, while LASSO and VASA Sharpe ratios decrease. This effect is easily explainable by the performance of the worst deciles, shown in Table B.3: LASSO, VASA and Neural Networks produce an average return of 7.5%, 5.8% and 5.4%, while OLS, Neural Networks XGBoost and Random Forest are below 5%, hence they are better in forecasting 'bad' stocks. Among these, tree-based models get an advantage from having also a lower variance.

In the end, we report p-values for the Sharpe ratio tests in Tables 3.27 and 3.28.

In the Long-Only case, we observe only high p-values, as it could be expected given the similar values

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	49,5%	21,8%	85,0%	68,1%	20,0%
LASSO	-	-	6,6%	42,5%	81,2%	61,5%
VASA	-	-	-	41,5%	18,4%	8,0%
Neural Networks	-	-	-	-	39,9%	8,0%
XGBoost	-	-	-	-	-	6,8%
Random Forest	-	-	-	-	-	-

Table 3.27 – Studentized Bootstrap p-values for the D10 Long-Only Portfolio.

for all the algorithms. The same happens in the Long-Short strategy, here a bit more surprisingly, since some difference looked more relevant. Only p-values of tree-based methods against LASSO, VASA and Neural Networks, the worst performers, are actually lower than the others, but still far

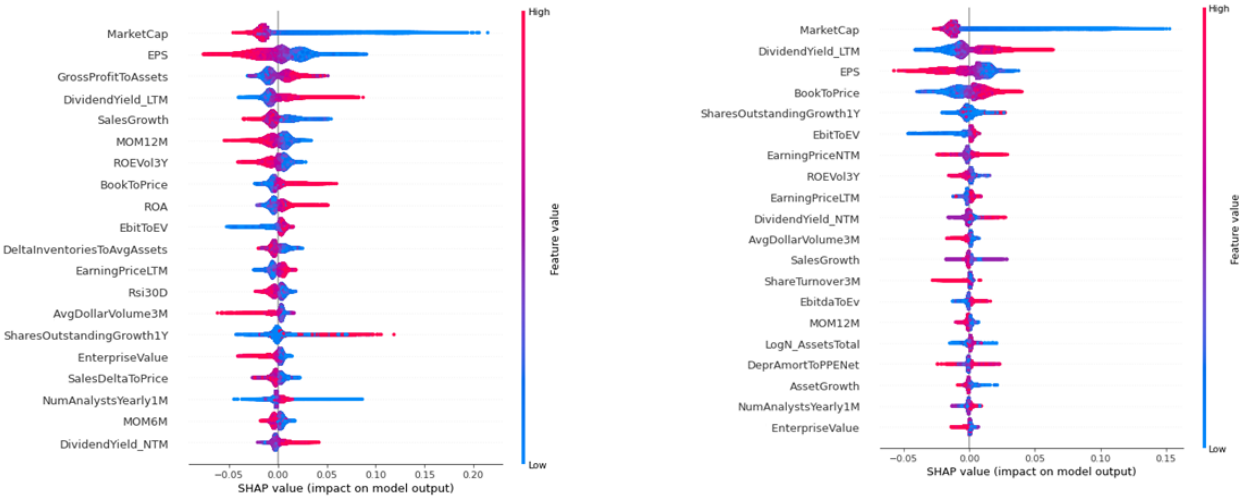
	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	18,2%	33,3%	62,1%	49,7%	59,3%
LASSO	-	-	34,9%	40,1%	10,4%	6,4%
VASA	-	-	-	87,2%	25,7%	22,6%
Neural Networks	-	-	-	-	11,0%	25,0%
XGBoost	-	-	-	-	-	53,9%
Random Forest	-	-	-	-	-	-

Table 3.28 – Studentized Bootstrap p-values for the Long-Short Portfolio.

from the boundaries of statistical significance.

3.3.3 Variables' Importance

To analyze variables' importance, we follow the same procedure described in Section 3.2.3. In Figure B.23 we show the average for the coefficients of linear methods, in Figure B.24 the averages of the measures used for the other non-linear methods. We notice many similarities with the European analysis: Market Cap is dominating in every algorithm, except for OLS and LASSO, where industry dummies are outstanding. In particular, Banks are particularly penalized, also by Neural Networks. There is a strong agreement also on the impact of Earning Per Share (EPS) and Book-to-Price ratios, which are in the top five for every metric. Other common feature that we find in out top twenties include: Sales-to-Price, Earning-to-Price and the Dividend Yield. In general, we can say that, even though Market Cap is clearly the most important predictor, other size-related features such ah Enterprise Value, Assets Total and the the Number of analyst in the last year, are less present than in Europe. They are replaced by these several quality and profitability ratios.
This comparison between Europe and Japan also holds if we look at SHAP plots for the models trained as of May 2018. We show Summary plots in Figures 3.7 for XGBoost and in Figure B.25 for Neural Networks and Random Forest. Moreover, we report SHAP dependency plots for the top three variables of XGBoost in Figures B.26 and B.27,of Random Forest in Figures B.28 and B.29. We remind that we can show them only for tree-based models, because we can compute interaction values there. The shape of Market Cap impact is very similar to what we have already seen: there is big skew towards lower capitalized stocks, where we predict high returns. Similarly, EPS SHAP values are negatively correlated to EPS itself, but with a more symmetric impact. However, it is



(a) Rolling Window **(b)** Expanding Window

Figure 3.7 – SHAP Summary plots for XGBoost models trained as of May 2018.

interesting to note that for XGBoost (see Figure B.27b) the shape of EPS SHAP values is more linear, while it is more tilted towards high EPS stocks for Random Forest (see Figure B.28c). Among the many interaction effects, we only highlight Figure B.29e as an example. It shows a clear separation between red and blue points, i.e. points with high and low and Assets Total. The former group is highly impacted by the respective value of Sales-to-Growth, both when it is large or small, while the latter is marginally influenced by this ratio.

3.3.4 Hyperparameters

In conclusion of the study on the Japanese set of stocks, we report the results of model tuning. The ranges used for hyperparameters searches are the same of the European data set, see Section 3.2.4 and Table 3.21. The combinations selected for each model that we trained are shown in the Appendix: see Figure B.30 for LASSO, VASA and Random Forest, Figure B.31 for Neural Networks and Figure B.32 for XGBoost.

3.4 United States Data Set

We will now conclude our analysis with the universe made by American stocks.

3.4.1 Predictive Accuracy

Tables 3.29 and 3.30 show average out-of-sample correlation and R^2 for the different algorithms.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
5,95%	6,80%	6,10%	8,60%	11,22%	11,41%

Table 3.29 – Average OOS Correlation.

OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
-1,96%	0,12%	-0,26%	0,28%	2,48%	0,99%

Table 3.30 – Average OOS R^2.

We first notice the usual best performance by tree-ensemble methods, with some disagreement among the two metrics: while correlation assigns the first spot to Random Forest with a tight margin, XGBoost's R^2 is more than double, achieving 2.48% versus 0.99%. Going down in the ranking, Neural Networks confirm their third position. However, they are close to linear methods, where LASSO and VASA are one step above OLS.
In Figure 3.8 we show R^2 for each month. The over-performance of XGBoost is really evident from 2013 to the end of 2015, with another spike across 2016 and 2017, but also in other periods, it is consistently in the first spots, together with Random Forest. In general, the rankings are quite stable since 2013, as we have already seen in the European and Japanese cases. Figure B.33 reports the evolution of correlation, where we do not have the great peak of XGBoost, which explains the disagreement between correlation and R^2.

The p-values of the Diebold-Mariano test, reported in Table 3.31, provide us a substantial confirmation of our impressions. The under-performance of OLS is significant against any other algorithm,

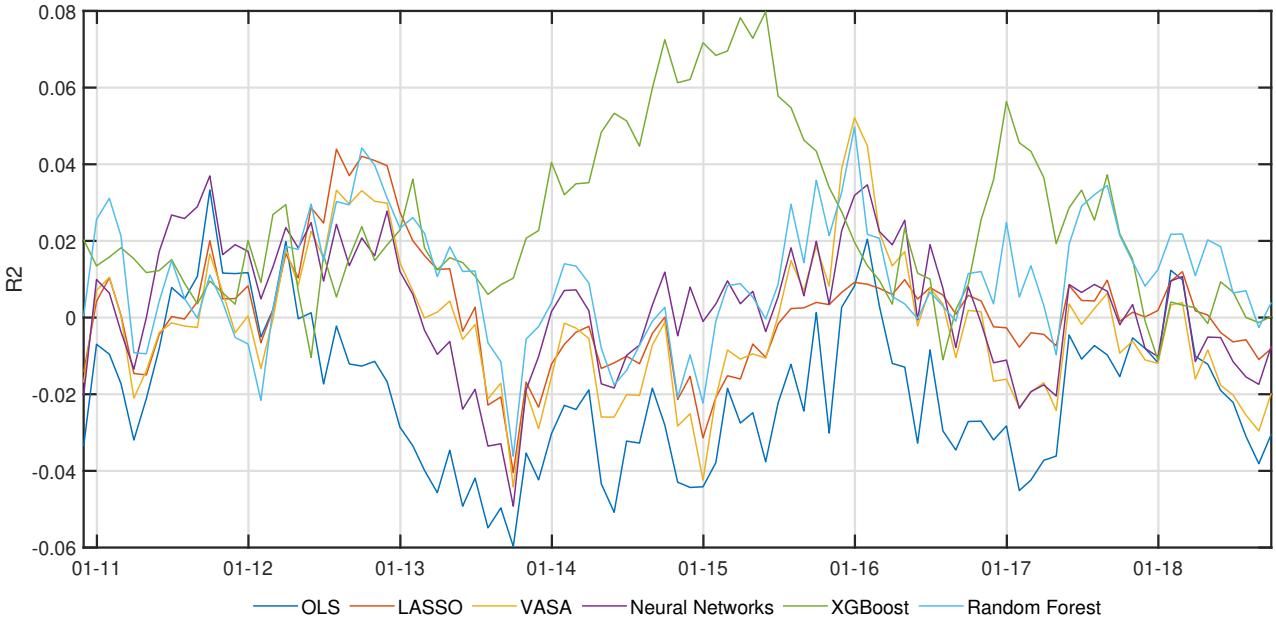


Figure 3.8 – OOS R^2 through time.

while LASSO, VASA and Neural Networks can be put on the same level, having p-values included between 7.08% and 36.15%. The best predictive power of XGBoost and Random Forest s is almost completely confirmed, with only the p-values of the latter against Neural Networks being rather high (8.30%), while the difference between them is not significant (22.45%).

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,07%	0,78%	0,00%	0,00%	0,00%
LASSO	-	-	12,42%	36,15%	0,01%	0,06%
VASA	-	-	-	7,08%	0,01%	0,00%
Neural Networks	-	-	-	-	1,28%	8,30%
XGBoost	-	-	-	-	-	22,45%
Random Forest	-	-	-	-	-	-

Table 3.31 – p-values of the Diebold-Mariano Test.

3.4.2 Portfolios' Performances

We now look at the financial performance of our predictions, considering as usual decile-based strategies.

Tables 3.32 and 3.33 show average returns, volatility and Sharpe ratios for Long-Only and Long-Short

portfolios. The performances of Long-Only portfolios are substantially aligned with the predictive

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	12,3%	20,5%	19,5%	20,3%	23,4%	23,2%
Volatility	15,3%	13,7%	14,3%	13,7%	14,8%	14,4%
Sharpe Ratio	0,803	1,492	1,357	1,480	1,584	1,607

Table 3.32 – Long-Only, based on the best Decile D10, Portfolios' performances. Average returns and volatilities are annualized.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
Avg. Returns	1,3%	7,9%	7,2%	8,7%	11,2%	10,2%
Volatility	6,8%	7,3%	7,9%	7,4%	5,6%	6,2%
Sharpe Ratio	0,196	1,085	0,920	1,168	1,982	1,647

Table 3.33 – Long-Short Decile-based Portfolios' performances. Average Returns and Volatilities are annualized.

accuracy of each algorithms. XGBoost and Random Forest lead the way with average returns above 23% and volatility below 15%, followed by Neural Networks, LASSO and VASA, that have slightly lower volatility, but return in the order of 20%. At the bottom, we find OLS, with significantly lower returns and higher volatility. We highlight that in the US data set we achieve the best results in term of returns and Sharpe ratios for the Long-Only strategy across universes.

Adding the short leg, only XGBoost increases its Sharpe Ratio, while Random Forest remains constant and all the other algorithms' performances drop consistently. However, this does not seem related to a better quality of XGBoost's predictions in the lowest deciles, as we can see from Table B.4. In D1, performances are all on the same level, with average returns between 11.6% and 13.0%, hence the Long-Short differences are mainly driven by D10. Tree-ensemble methods are able to maintain a good Sharpe ratio, because they started from high returns in D10 and they manage to have a low volatility, but the drop in average returns is much bigger than what we saw in the Japanese case.

Looking at the pairwise Sharpe Ratio tests, in the Long-Only case, see Table 3.34, only the OLS under-performance is statistically significant, having all p-values below 1%, while all the others are above 11%. In the Long-Short case, see Table 3.35, where differences in Sharpe ratios are bigger, we can notice the significance of XGBoost superiority on all algorithms, excluding Random Forest, which is superior to OLS and VASA. OLS' poor performance is confirmed also here, while the re-

maining p-values are above 17%; in particular, we note again that Neural Networks are not able to beat LASSO and VASA. As in the previous universes, we do not report HAC-based p-values, because they perfectly agree with the Studentized Bootstrap outcomes.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,8%	0,2%	0,2%	0,4%	0,8%
LASSO	-	-	21,8%	90,8%	23,2%	27,7%
VASA	-	-	-	11,4%	12,6%	15,4%
Neural Networks	-	-	-	-	22,2%	27,1%
XGBoost	-	-	-	-	-	98,2%
Random Forest	-	-	-	-	-	-

Table 3.34 – Studentized Bootstrap p-values for the D10 Long-Only Portfolio.

	OLS	LASSO	VASA	Neural Networks	XGBoost	Random Forest
OLS	-	0,8%	1,6%	0,4%	0,4%	0,8%
LASSO	-	-	46,7%	59,7%	4,2%	20,2%
VASA	-	-	-	17,4%	0,8%	5,6%
Neural Networks	-	-	-	-	4,0%	23,8%
XGBoost	-	-	-	-	-	21,0%
Random Forest	-	-	-	-	-	-

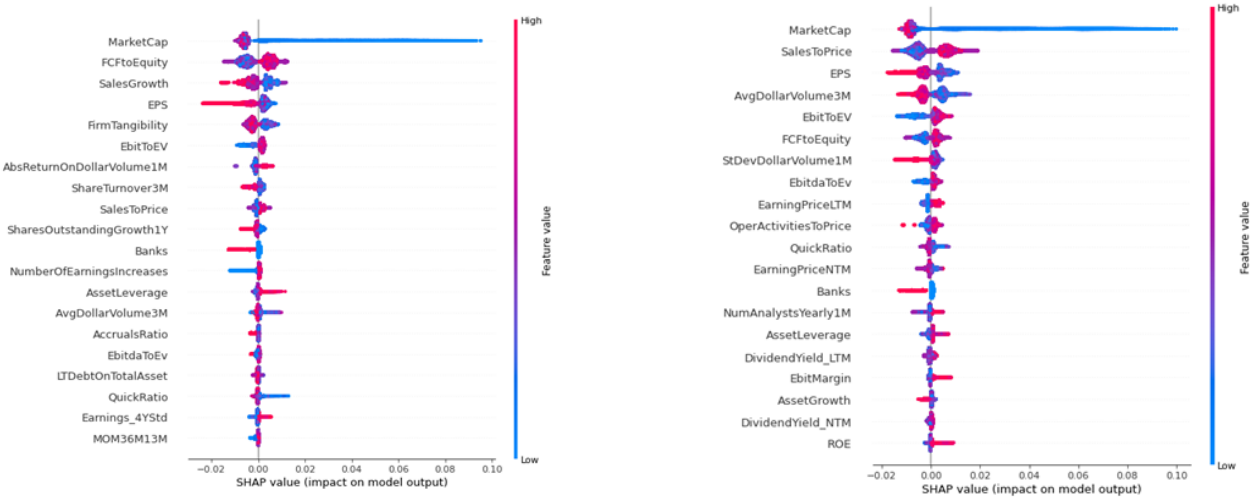
Table 3.35 – Studentized Bootstrap p-values for the Long-Short Portfolio.

3.4.3 Variables' Importance

We now try to understand the drivers of our performances, trough the analysis of most important variables. Firstly, we report average coefficients of linear methods in Figure B.34 and average metrics for non-linear algorithms in Figure B.35. We note many similarities with the European and Japanese plots: Market Cap is always the most important predictor, except for Neural Networks, where the Earning-Per-Share (EPS) ratio is able to claim the first spot of SHAP values. Anyway, it is not a huge surprise, as EPS has been one of the most important variables across all universes, algorithms and metrics. Averages show more similarities of US with Japan than with Europe, especially because, after Market Cap, other size-related features do not often appear in the top twenties. Also the importance of Sales-to-Growth and Sales-to-Price is a common phenomenon between US and Japan. At the same time, ratios such as Earning-to-Price, Operational Activities-to-Price and

Ebit/Ebitda-to-Enterprise Value remain strong predictors through all the universes. Another persistent phenomenon is the great reliance of Random Forest on a very few predictors, while XGBoost usually has less extreme values in the first positions.

We conclude with our overview of SHAP models for tree-ensemble methods trained as of May 2018. We show summary plots in Figure 3.9 for XGBoost and Figure B.36 for Neural Networks and Random Forest. Here, we only want to highlight how the absence of interaction values is a major hurdle in our understanding of Neural Networks' behaviour. Since we are able to get only a linear approximation of the impact, we only know the slope of the line and its mean, hence we will never be able to understand if they have an asymmetric pattern, like XGBoost. Moreover, the mean might be misleading, because if we have a large number of stocks with an high Market Cap, which are penalized, the mean of SHAP values will be negative, as we see in Figure B.36a, even if SHAP value are very high for stocks with low capitalization.



(a) Rolling Window **(b)** Expanding Window

Figure 3.9 – SHAP Summary plots for XGBoost models trained as of May 2018.

In the end, we report the SHAP dependency plots for the top three features of XGBoost, see Figures B.37 and B.38 and Random Forest, see Figures B.39 and B.40, as of May 2018. The shapes of Market Cap and EPS impacts are in line with those already seen in the previous cases, while we have a good presence of Sales-to-Price and Sales-to-Growth ratios. We see the former for the first time: its impact depends only on the signs and the correlation with SHAP values is positive.

We have already shown the latter with the Random Forest trained with rolling window in Japan (Figure B.29) and the pattern is the same: stocks with high Sales-to-Growth are penalized and also here the relationship mainly depends on the sign of the feature.

3.4.4 Hyperparameters

As last step for the United States data set, we give a brief overview on model tuning. The search ranges are the same for all the three region-specific universes, see Section 3.2.4 and Table 3.21. The actual choices are shown in Figure B.41 for LASSO, VASA and Random Forest, Figure B.42 for Neural Networks and Figure B.43 for XGBoost.

Chapter 4

Conclusions

In the end, we will try to draw some conclusions on what we have learned during this work and how it could extended and improved.

The most important split in this thesis is between balanced and non-balanced data sets. They significantly differ from several points of view: predictive accuracy, variables' importance and financial performance, even though we get similar outcomes in the Long-Only portfolios, that are the most meaningful in our setup, and in line with related works, such as Gu et al. (2020a). Our main intuition on the source of these differences is given by the great explanatory power of SHAP plots. They show that Market Cap is much more important in non-balanced data set and, in particular, we can see that the effect is strongly skewed towards low cap stocks, that our non-linear models strongly promote. If we consider that the balanced dataset is very likely to have stocks with an higher capitalization, since they have never gone out of the index in more than twenty years, this could explain the gaps. We notice that our universes do not contain penny stocks and they contain only rather liquid stocks, but anyway this asymmetry could harm our out-of-sample results when accounting for trading costs. Among Europe, Japan and US non-balanced data sets, we do not observe huge differences, as average R^2 and Sharpe ratios are in the same ranges. One discrepancy is that in US we struggle in predicting low performing stocks: this may be due to the small cap bias, which might be more penalizing in this universe. However, in unreported results, we observed this phenomenon also in stock-specific *balanced* data sets. In Europe and Japan we have more smooth outcomes across deciles, even if in

Europe we rely more on other size features, while in Japan quality and profitability predictors have a stronger impact.

In terms of methodology, we have made two uncommon choices for the field of machine learning in asset pricing: using yearly returns and ranking them. They were motivated by two practitioners' observations: machine learning predictions should have been used to feed another optimization model, which has an average holding period of a stock of one year, filtering out a certain percentage of the worst stocks. This did not seem to bring us particular additional value, but it costed us a good amount of information, especially the ranking of returns. This is clear when we build the short leg of portfolios: we should short a stocks only if we predict a negative return, while we were forced to sell a full decile without knowing the sign of expected returns. In periods of strong upwards market movements, even the worst decile can have a positive performance. However, this also shows how this machine learning framework is flexible and is able to bring value in different conditions. We also note that, if we compare again this work with Gu et al. (2020a), the amount of historical data considered is very different, as they have data from 1957 to 2016, while we go from 1998 to 2019. This means that we always have less training data, but also that we test in more recent periods, where the general predictability could be different, for instance for the rise of quantitative investors who started to exploit factors' anomalies.

In terms of possible expansions of the work, we have already mentioned the obvious direction of new algorithms, with XGBoost and VASA being two clear examples of this. Another possible direction to take, without changing the framework, could be to move towards a more performance oriented feature engineering: the standard practice is to use 'financially meaningful' ratios, to use some common knowledge and have more explainability. But in other machine learning worlds, such as Kaggle competitions, where the need for the highest accuracy is extreme, winning solutions always rely on building even thousands of feature and then selecting them on a forecasting power basis . If we want to slightly modify our general framework, we highlight two ideas suggested by de Prado (2018): use fractionally differentiated prices instead of returns and transform the problem in classification, using a dynamic 'triple-barrier' labelling. The former suggestion is based on the observations that using return is an overshooting on the memory-stationarity dilemma, because we do not need so much

See, for instance, one winning solution from a recent data science competition: `https://www.kaggle.com/c/data-science-bowl-2019/discussion/127469`

stationarity, while we could get value from having more memory in our time series. The former idea is to dynamically label observations: we set two horizontal barriers that, if hit, will assign a label of buy or sell to the stock, but also a vertical (time) barrier that, if hit, will give a neutral label. Dynamically setting these thresholds allows us to adjust our labelling depending on the volatility present in the market, which means that we would be able to risk-adjust our expected returns.

Appendix A

XGBoost Specifications

In this section, we are going through some mathematical details of the XGBoost framework, in order to understand its functioning and its particular variables' importance metric.

A.1 Objective function

XGBoost lays in the gradient boosting framework, hence its general principles are the same as in Section 2.3.5. For each stock $i = 1, \ldots, N$, at time t, we attempt to predict the excess return $r_{i,t+12}$ with:

$$\hat{r}_{i,t+12} = \sum_{s=1}^{S} g_s(z_{i,t}), \quad g_s \in \mathcal{G}, \tag{A.1}$$

where S is the number of trees and $\mathcal{G}$ is the set of all possible CARTs. With a general loss function l, at time t we will optimize the objective function:

$$\mathcal{L} = \sum_{i=1}^{N} l(r_{i,t+12}, \hat{r}_{i,t+12}) + \sum_{s=1}^{S} \Omega(g_s), \tag{A.2}$$

where Ω is a measure of model complexity, hence its sum is a regularization term. From Equation (A.1), we can see that at each step k of the summing of trees, we can write:

$$\hat{r}_{i,t+12}^{(k)} = \sum_{s=1}^{k} g_s(z_{i,t}) = \hat{r}_{i,t+12}^{(k-1)} + g_k(z_{i,t}). \tag{A.3}$$

This allows us to rewrite Equation (A.2), at the step k, as

$$\mathcal{L}^{(k)} = \sum_{i=1}^{N} l(r_{i,t+12}, \hat{r}_{i,t+12}^{(k-1)} + g_k(z_{i,t})) + \sum_{s=1}^{S} \Omega(g_s). \tag{A.4}$$

To account for a general loss function, we perform a Taylor expansion of the objective function up to the second order:

$$\mathcal{L}^{(k)} = \sum_{i=1}^{N} \left[l(r_{i,t+12}, \hat{r}_{i,t+12}^{(k-1)}) + p_i g_k(z_{i,t}) + \frac{1}{2} q_i g_k^2(z_{i,t}) \right] + \Omega(g_k) + \text{constant}, \tag{A.5}$$

where

$$\begin{aligned} p_i &= \partial_{\hat{r}_i^{(k-1)}} l(r_{i,t+12}, \hat{r}_{i,t+12}^{(k-1)}), \\ q_i &= \partial^2_{\hat{r}_i^{(k-1)}} l(r_{i,t+12}, \hat{r}_{i,t+12}^{(k-1)}). \end{aligned} \tag{A.6}$$

Removing all constants, what we really need to minimize remains:

$$\mathcal{L}^{(k)} \sim \sum_{i=1}^{N} \left[p_i g_k(z_{i,t}) + \frac{1}{2} q_i g_k^2(z_{i,t}) \right] + \Omega(g_k). \tag{A.7}$$

A.2 Model Complexity, Structure Score and Gain

One of the greatest intuition of XGBoost is a particular reformulation of the regularization term, which allows to express the objective function as a sum over the leaves.

First of all, we express the single tree as:

$$g_k(z) = w_{t(z)}, \;\; w \in \mathbb{R}^L, \;\; t : \mathbb{R}^P \to \{1, 2, \ldots, L\}, \tag{A.8}$$

where w is the vector of scores on leaves, t assigns each data point to the corresponding leaf and L is the number of leaves.

Model complexity is defined as:

$$\Omega(g) = \gamma L + \frac{1}{2} \lambda \sum_{l=1}^{L} w_l^2. \tag{A.9}$$

Therefore, Equation (A.7) becomes:

$$\begin{aligned}\mathcal{L}^{(k)} &\sim \sum_{i=1}^{N}\left[p_i g_k(z_{i,t}) + \frac{1}{2}q_i g_k^2(z_{i,t})\right] + \gamma L + \frac{1}{2}\lambda\sum_{l=1}^{L} w_l^2 \\ &= \sum_{l=1}^{L}\left[P_l w_l + \frac{1}{2}\left(Q_l + \lambda\right)w_l^2\right] + \gamma L,\end{aligned} \tag{A.10}$$

where $P_l = \sum_{i\in I_l} p_i$, $Q_l = \sum_{i\in I_l} q_i$ and $I_l = \{i : t(z_{i,t}) = l\}$ is the set of indices of inputs contained in the l-th leaf. From here, we can directly get our solutions:

$$\begin{aligned}w_l^* &= -\frac{P_l}{Q_l + \lambda}, \\ \mathcal{L}^* &= -\frac{1}{2}\sum_{l=1}^{L}\frac{P_l^2}{Q_l + \lambda} + \gamma L,\end{aligned} \tag{A.11}$$

where the last equation indicates how good a tree structure t is. Using this expression, it is straightforward to define our greedy algorithm that we will use to make our splits. We define the *Gain* as:

$$Gain = \frac{1}{2}\left[\frac{P_{Left}^2}{Q_{Left} + \lambda} + \frac{P_{Right}^2}{Q_{Right} + \lambda} - \frac{P_{Prev}^2}{Q_{Prev} + \lambda}\right] - \gamma, \tag{A.12}$$

where the four terms represent, respectively, the score on the new left leaf, the score on the new right leaf, the score on the previous leaf and the regularization term. At each step, we only need to select the split that maximizes the *Gain*. Moreover, thanks to the embedded regularization term, we automatically delete all the splits whose Gain is not big enough, which is equivalent to pruning. In conclusion, we also note that Gain gives us a direct way to measure features' importance. At each split of each tree, we only need to attribute the Gain to predictor that we are using in the split and to accumulate these scores through all the trees.

Appendix B

Further Empirical Results

B.1 Universal Balanced Data Set

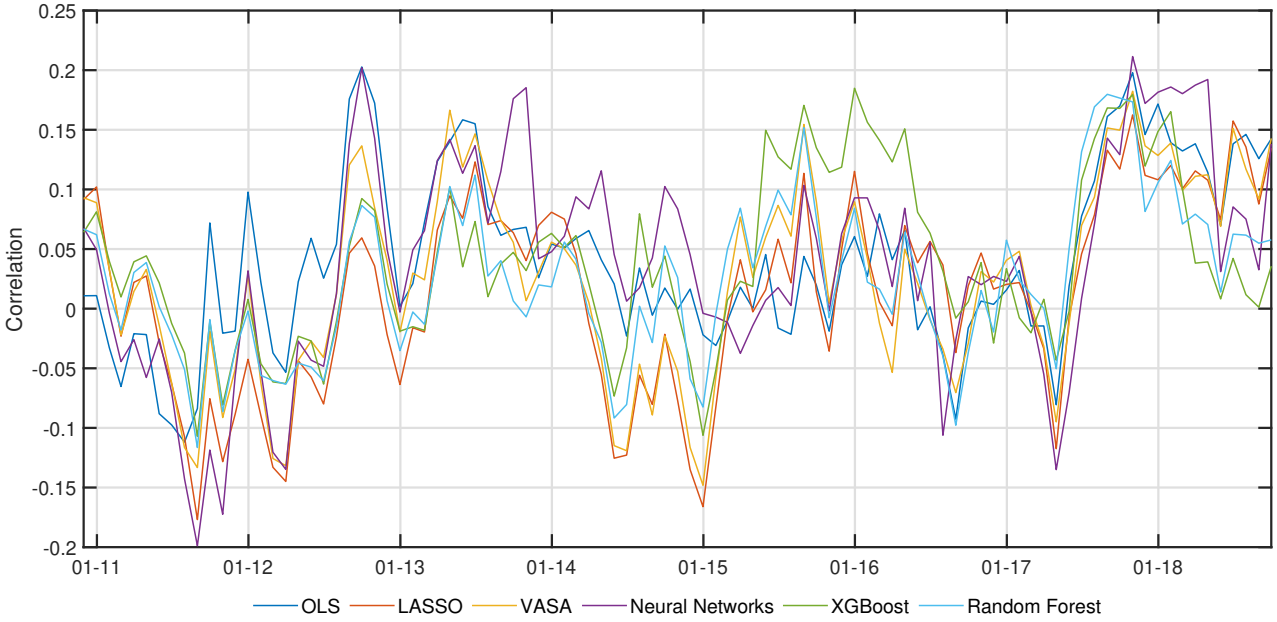
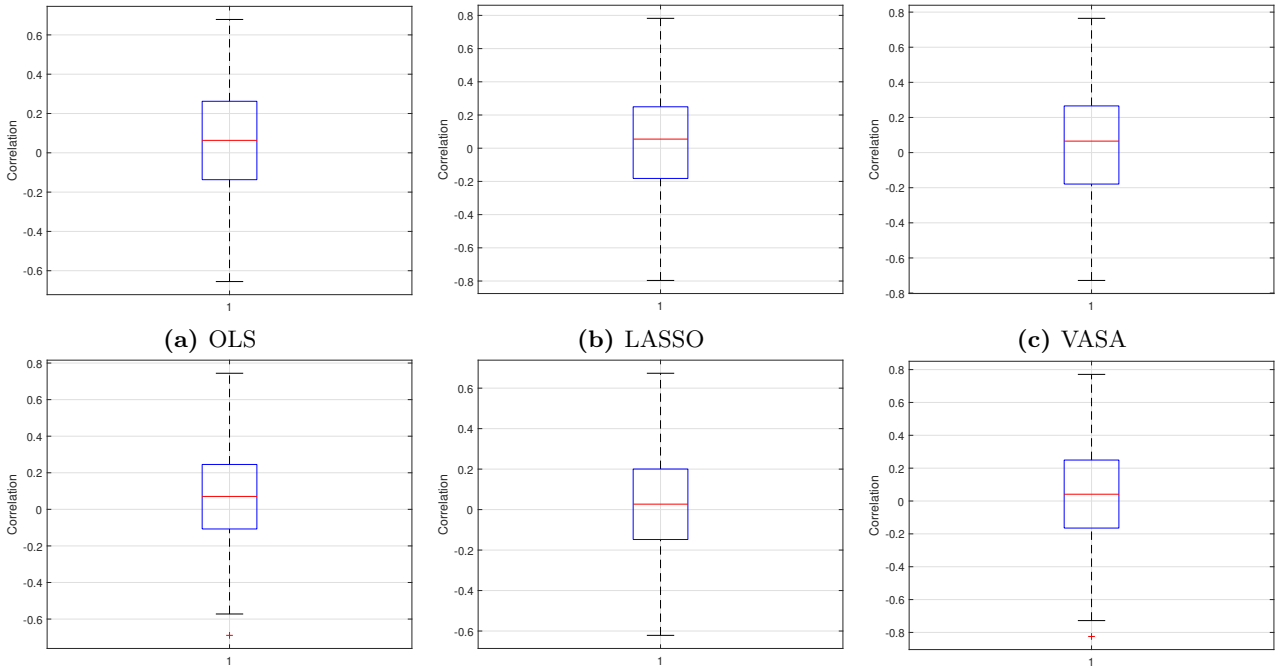


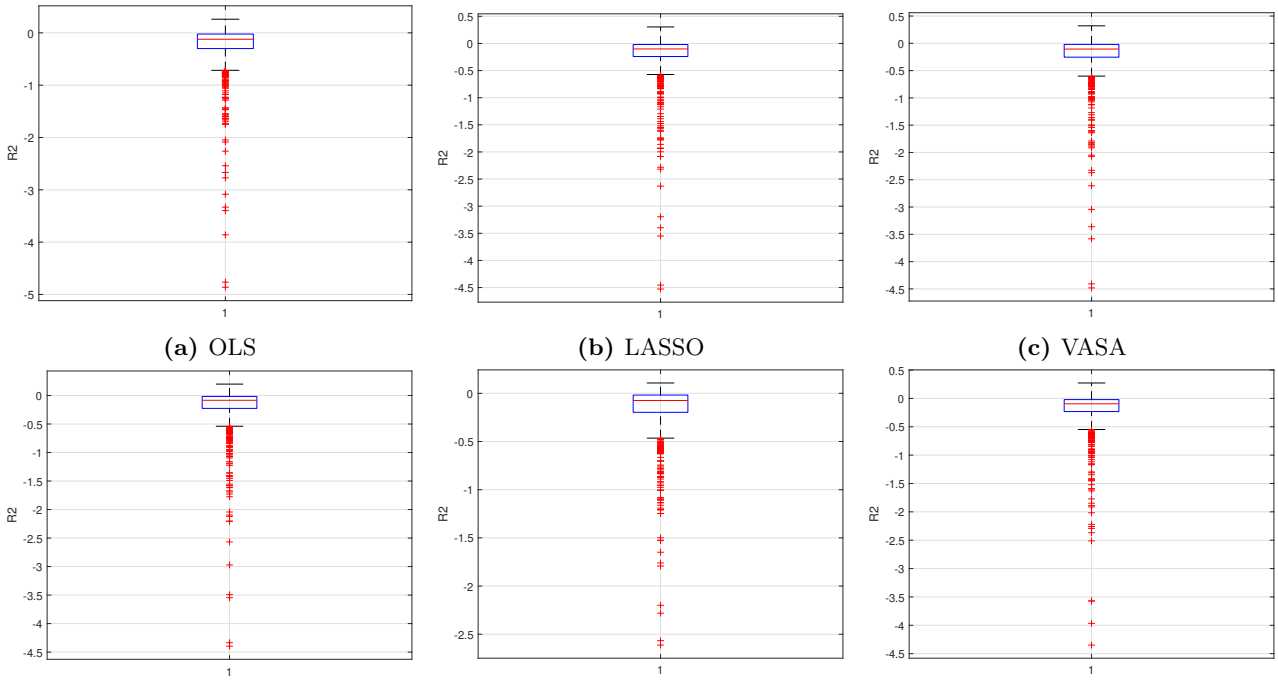
Figure B.1 – OOS Correlation through time.



(a) OLS **(b)** LASSO **(c)** VASA

(d) Neural Networks **(e)** XGBoost **(f)** Random Forest

Figure B.2 – Boxplots for stock-specific OOS Correaltion



(a) OLS **(b)** LASSO **(c)** VASA

(d) Neural Networks **(e)** XGBoost **(f)** Random Forest

Figure B.3 – Boxplots for stock-specific OOS R^2

	OLS									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	8,0%	8,8%	9,2%	10,1%	11,1%	10,8%	12,7%	13,8%	13,4%	15,8%
Volatilty	11,1%	11,5%	11,6%	11,3%	12,3%	11,9%	12,1%	12,5%	12,9%	14,8%
Sharpe Ratio	0,724	0,768	0,791	0,894	0,903	0,904	1,054	1,110	1,044	1,069

	LASSO									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	8,9%	10,1%	11,0%	8,9%	12,6%	11,7%	12,5%	11,8%	12,3%	14,3%
Volatilty	11,5%	11,8%	12,1%	11,8%	11,8%	12,4%	12,3%	12,2%	13,1%	14,0%
Sharpe Ratio	0,773	0,853	0,911	0,752	1,069	0,945	1,021	0,966	0,934	1,024

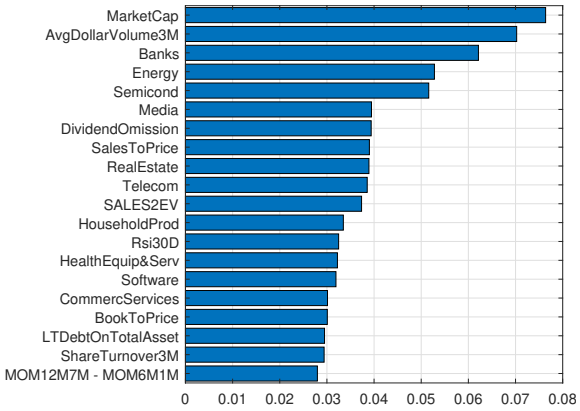
	VASA									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	8,5%	9,5%	10,1%	12,0%	10,6%	11,4%	12,7%	13,0%	12,4%	14,0%
Volatilty	11,4%	11,7%	11,8%	11,6%	11,6%	12,1%	12,4%	12,5%	13,4%	14,2%
Sharpe Ratio	0,749	0,817	0,857	1,029	0,911	0,940	1,025	1,039	0,930	0,984

	Neural Networks									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	9,1%	9,4%	9,8%	9,6%	11,3%	11,6%	13,7%	11,7%	13,6%	13,1%
Volatilty	11,6%	11,6%	11,7%	11,9%	11,5%	12,2%	12,5%	12,6%	12,9%	14,1%
Sharpe Ratio	0,788	0,809	0,838	0,802	0,986	0,945	1,092	0,928	1,055	0,929

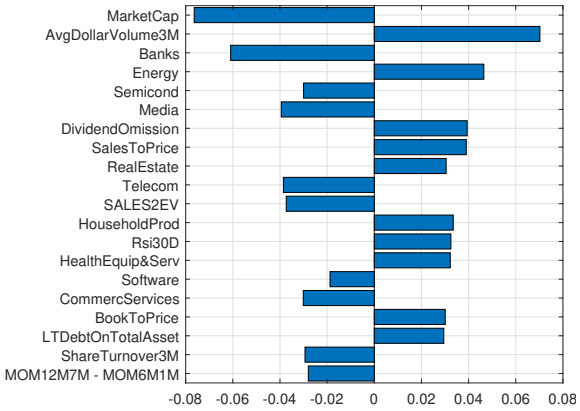
	XGBoost									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	8,4%	9,6%	9,5%	10,3%	11,0%	11,8%	12,1%	12,7%	13,5%	15,6%
Volatilty	11,4%	11,9%	12,6%	11,9%	12,2%	11,8%	12,2%	12,3%	12,7%	13,9%
Sharpe Ratio	0,734	0,809	0,752	0,870	0,895	1,004	0,989	1,035	1,067	1,118

	Random Forest									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	7,9%	9,8%	11,4%	11,5%	11,1%	10,3%	11,4%	13,6%	13,2%	13,9%
Volatilty	12,0%	11,7%	12,6%	12,4%	11,6%	12,1%	12,3%	12,1%	12,7%	13,9%
Sharpe Ratio	0,658	0,838	0,911	0,928	0,953	0,846	0,929	1,121	1,043	1,002

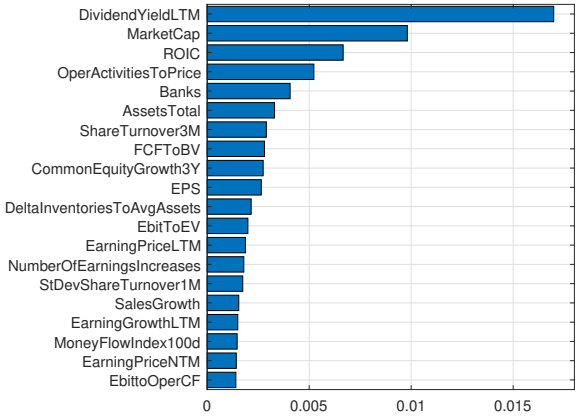
Table B.1 – Decile-based Portfolios' Performances. Average returns and volatilities are annualized. D1 is the (predicted) worst-performing decile, D10 is the (predicted) best-performing decile.



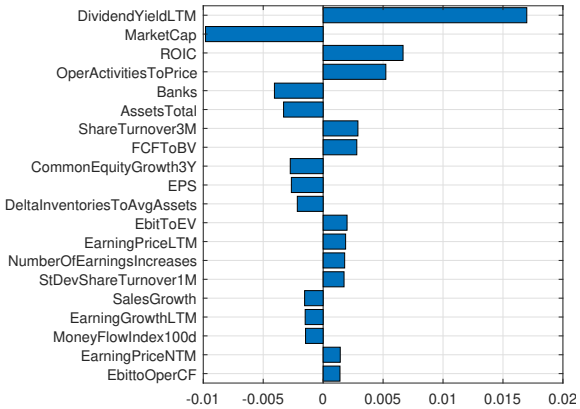
(a) OLS Average |Coefficient|



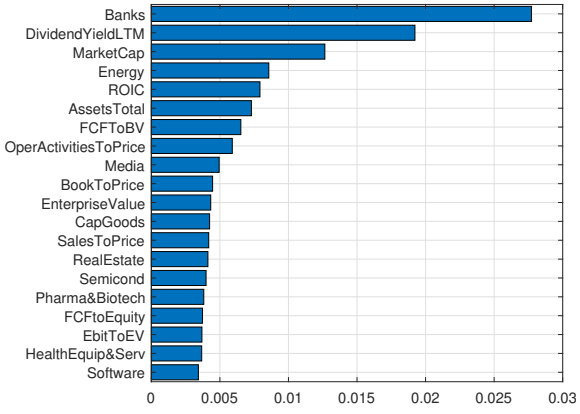
(b) OLS Average Coefficient



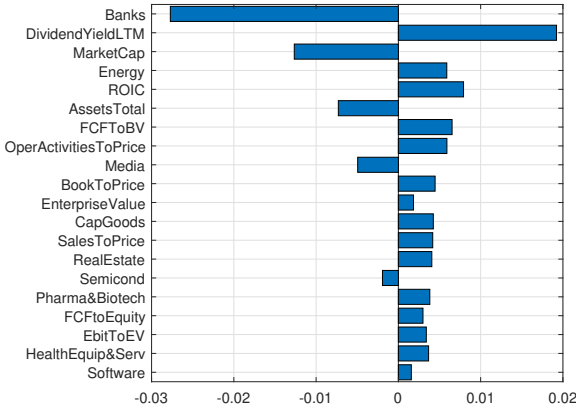
(c) LASSO Average |Coefficient|



(d) LASSO Average Coefficient

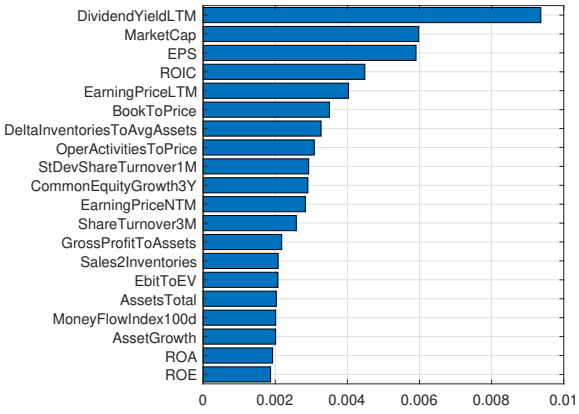


(e) VASA Average |Coefficient|

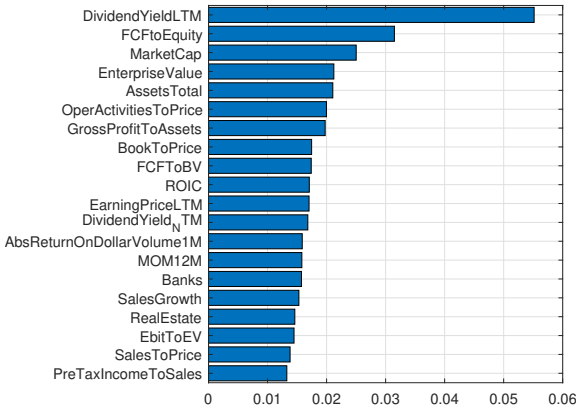


(f) VASA Average Coefficient

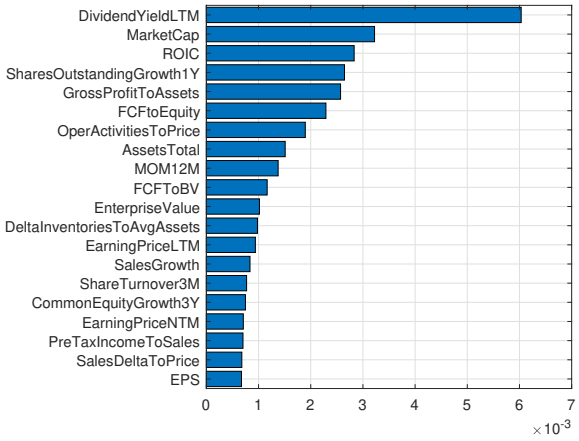
Figure B.4 – Averages of linear methods' coefficients and their absolute values.



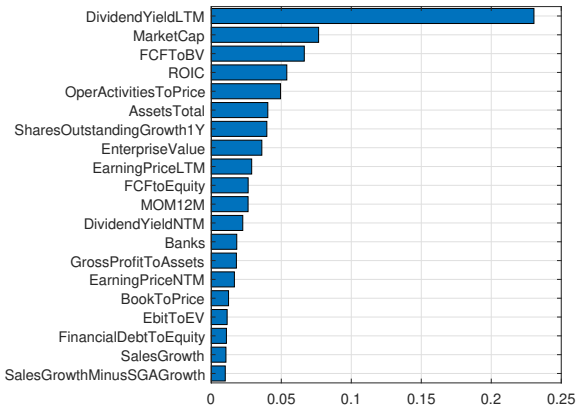
(a) Neural Networks Average |SHAP|



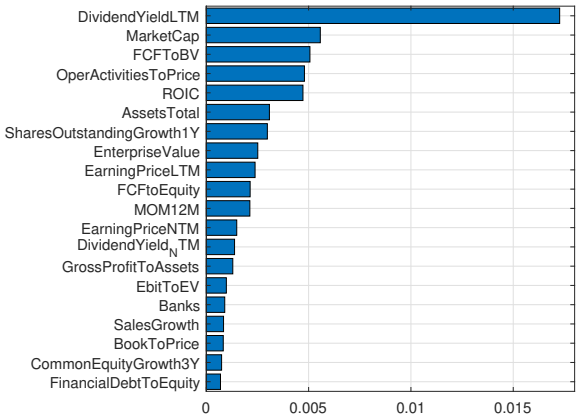
(b) XGboost Average Gain



(c) XGboost Average |SHAP|



(d) Random Forest Average Importance



(e) Random Forest Average |SHAP|

Figure B.5 – Averages of variables importance measures for non-linear methods.

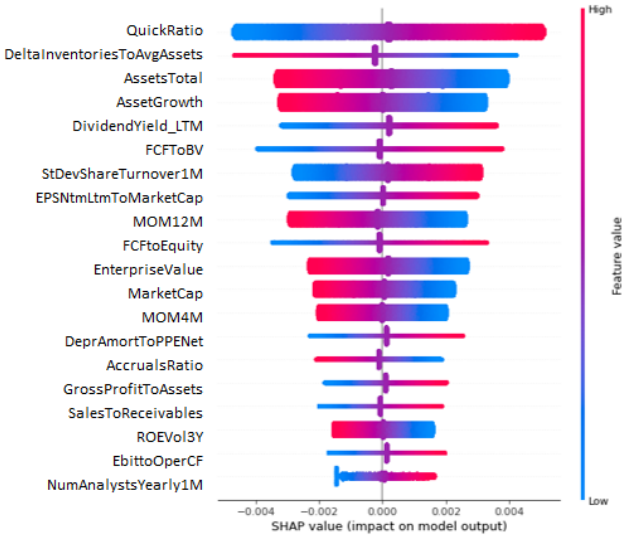


Figure B.6 – SHAP summary plot for Neural Networks model trained in May 2018.

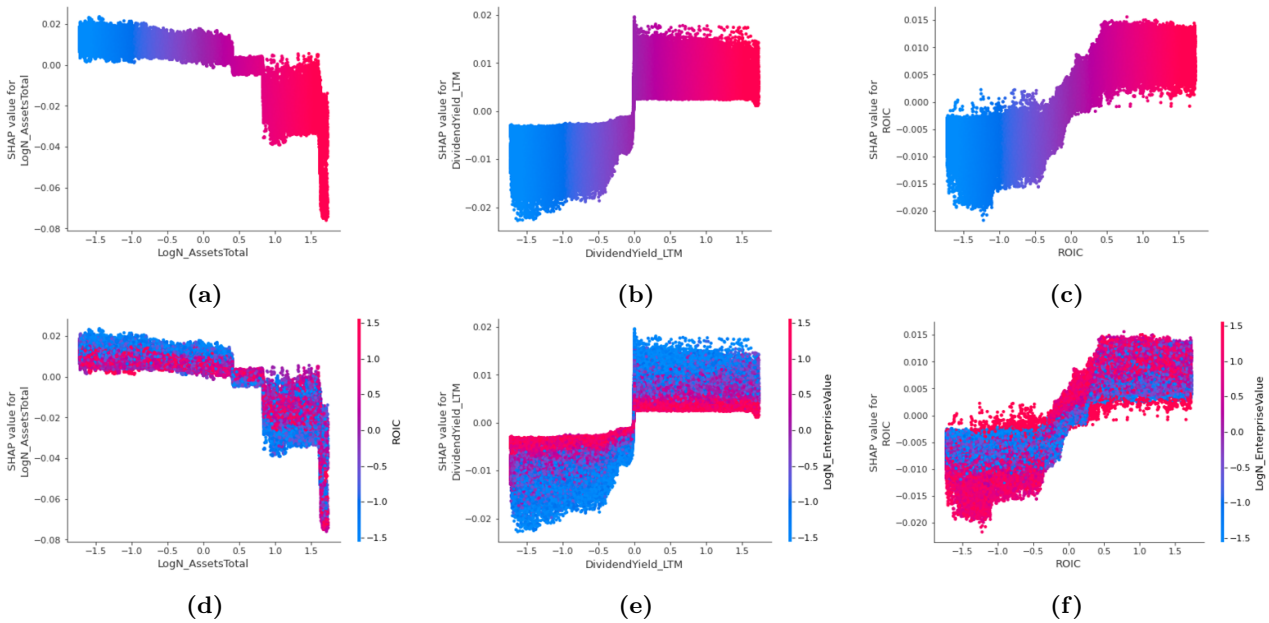
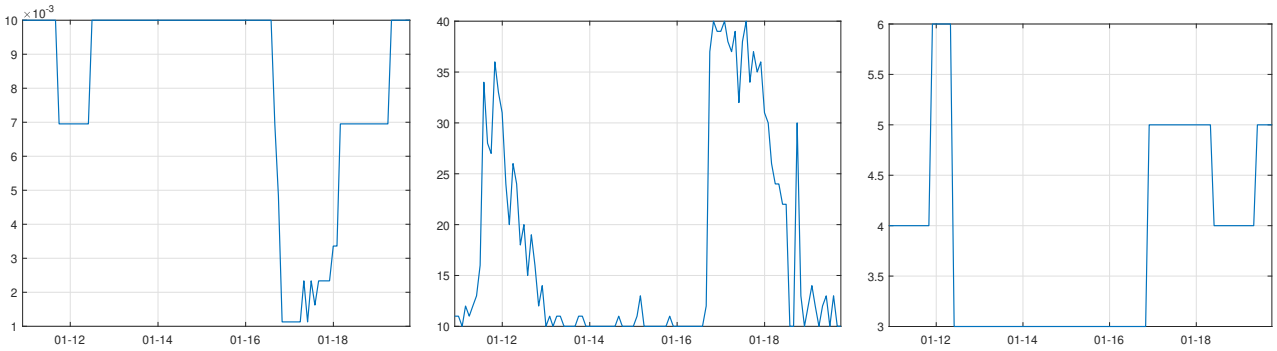


Figure B.7 – SHAP Dependency Plots of the top 3 variables for Random Forest. The line above shows the SHAP values of a certain feature against the feature itself. In the line below, we add also the interaction with another feature, which is chosen as the one with the highest correlation of SHAP values in a vertical slice of the SHAP dependency plot. Each dot represents a single prediction.

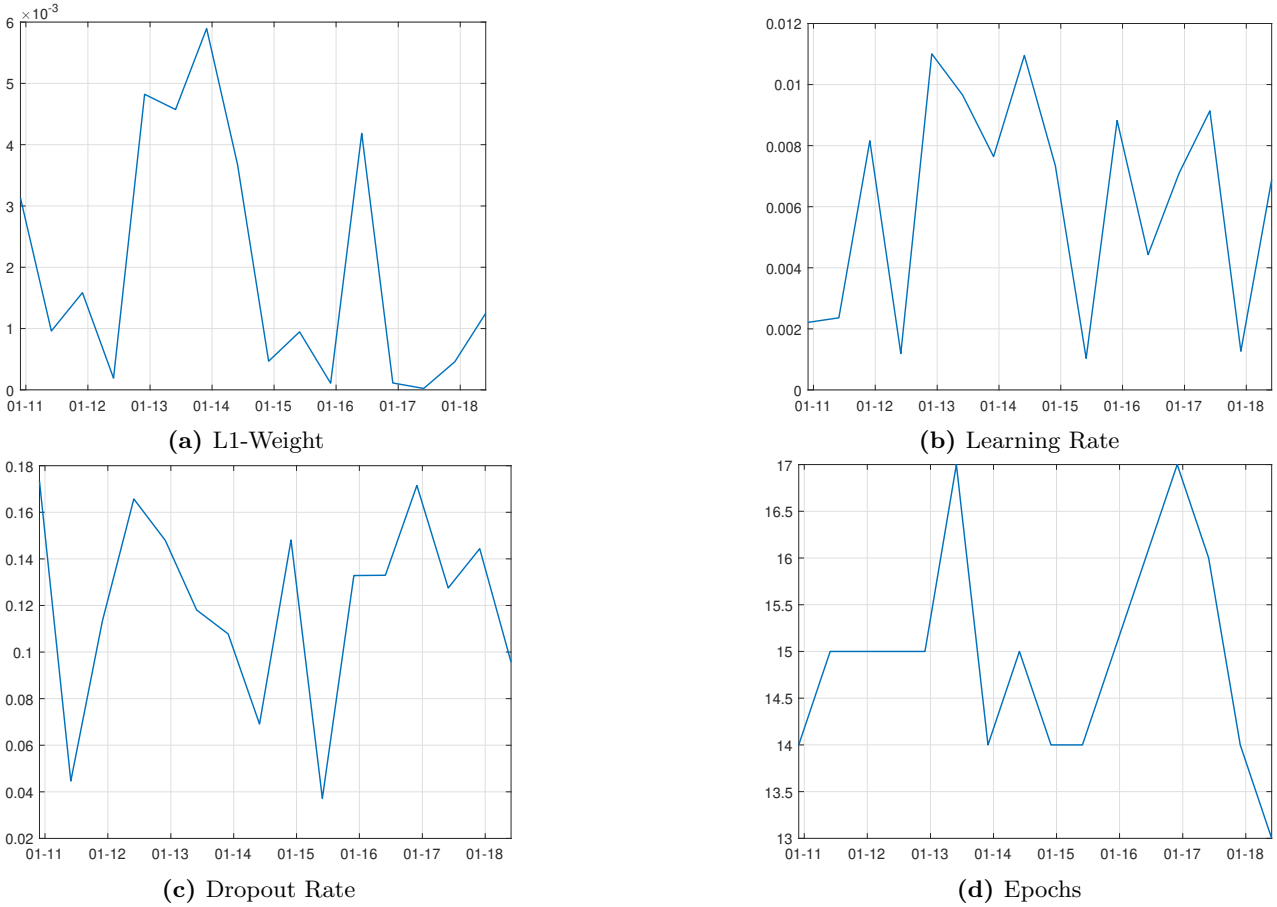


(a) LASSO L1-Weight

(b) VASA Number of features in sub-models

(c) Random Forest Maximum Depth

Figure B.8 – Hyperparameters choices for LASSO, VASA and Random Forest.



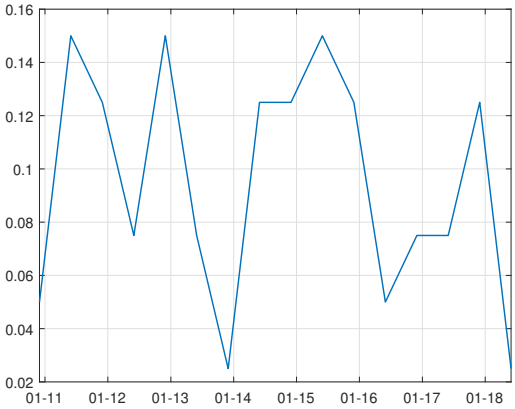
(a) L1-Weight

(b) Learning Rate

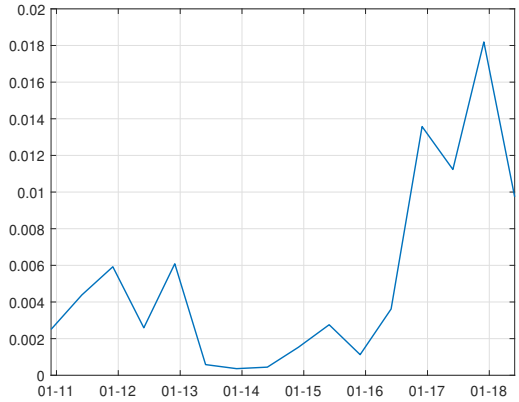
(c) Dropout Rate

(d) Epochs

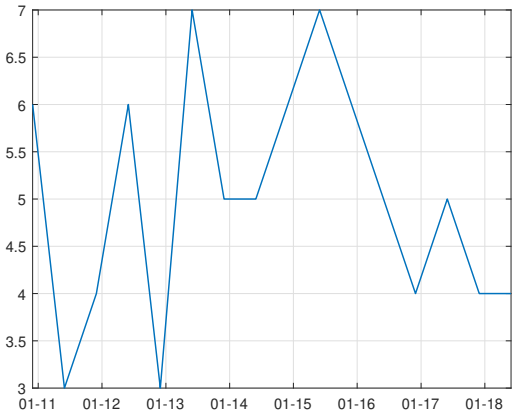
Figure B.9 – Neural Networks Hyperparameters choices.



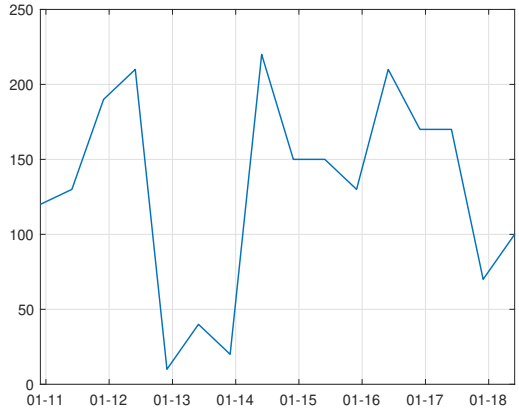
(a) Columns Dropout Rate



(b) Learning Rate



(c) Maximum Depth



(d) Number of trees

Figure B.10 – XGBoost Hyperparameters choices.

B.2 European Data Set

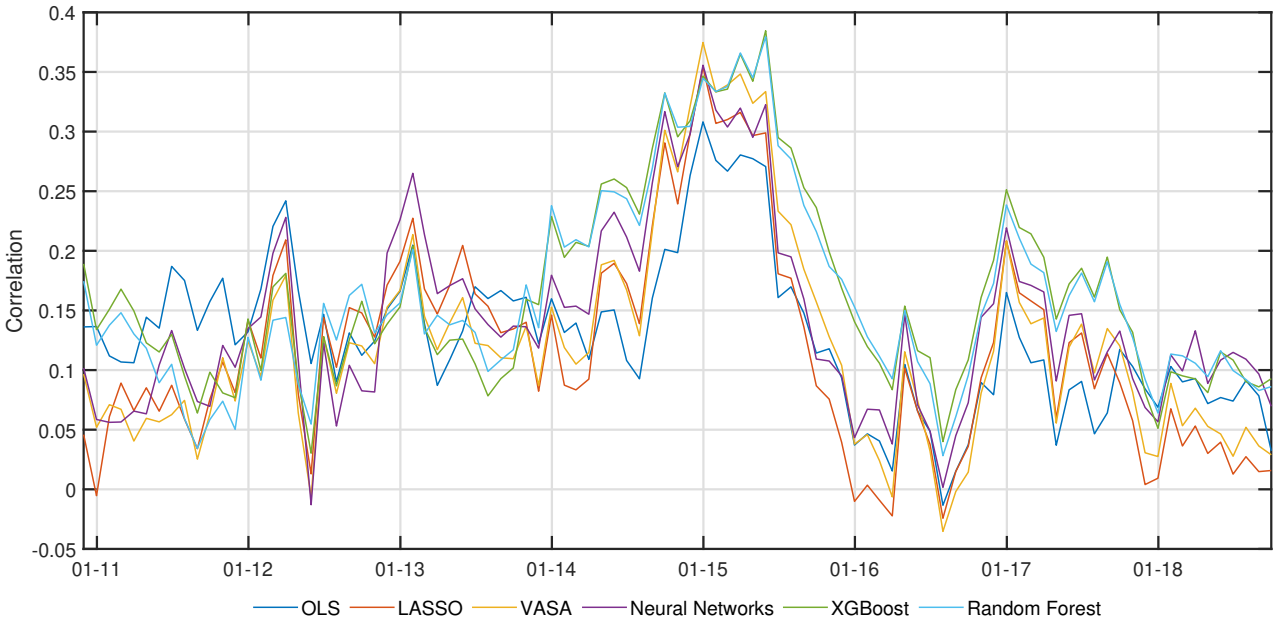


Figure B.11 – OOS Correlation through time.

	OLS									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	4,6%	7,8%	7,1%	7,4%	9,4%	8,0%	11,1%	11,3%	14,4%	17,5%
Volatilty	18,0%	16,4%	15,7%	15,8%	15,7%	16,4%	15,7%	15,3%	15,7%	16,4%
Sharpe Ratio	0,253	0,477	0,455	0,469	0,598	0,489	0,704	0,737	0,918	1,066

	LASSO									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	7,1%	7,3%	7,5%	8,7%	9,2%	9,3%	10,2%	12,0%	12,0%	17,1%
Volatilty	18,0%	16,2%	15,9%	15,9%	16,2%	15,3%	16,4%	16,1%	15,5%	16,2%
Sharpe Ratio	0,397	0,454	0,472	0,546	0,572	0,606	0,619	0,746	0,777	1,057

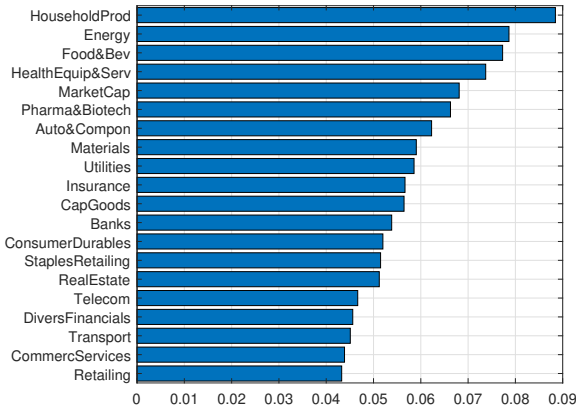
	VASA									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	6,4%	7,4%	7,6%	8,4%	9,1%	10,7%	10,9%	10,1%	13,1%	17,6%
Volatilty	19,0%	16,7%	16,4%	15,7%	15,7%	15,1%	16,2%	15,8%	16,1%	16,2%
Sharpe Ratio	0,337	0,443	0,463	0,534	0,576	0,706	0,675	0,639	0,815	1,085

	Neural Networks									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	6,1%	5,7%	7,8%	9,4%	8,4%	8,2%	11,0%	10,1%	13,1%	18,7%
Volatilty	19,0%	16,8%	16,4%	15,7%	15,4%	14,8%	15,9%	15,6%	15,6%	16,3%
Sharpe Ratio	0,324	0,336	0,475	0,598	0,546	0,555	0,689	0,647	0,838	1,148

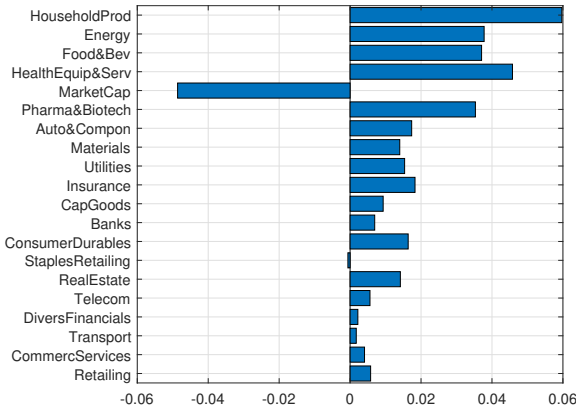
	XGBoost									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	5,9%	6,9%	7,8%	7,4%	7,9%	9,6%	10,4%	11,5%	11,5%	19,2%
Volatilty	20,0%	16,7%	15,3%	15,1%	14,7%	14,8%	16,0%	16,0%	16,7%	17,0%
Sharpe Ratio	0,297	0,415	0,507	0,486	0,540	0,649	0,653	0,721	0,693	1,130

	Random Forest									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	6,2%	8,5%	7,4%	7,2%	7,8%	10,0%	11,0%	12,3%	10,4%	19,8%
Volatilty	19,5%	16,8%	15,3%	15,2%	14,5%	15,4%	15,4%	16,2%	16,8%	16,6%
Sharpe Ratio	0,317	0,505	0,484	0,474	0,538	0,649	0,717	0,760	0,620	1,194

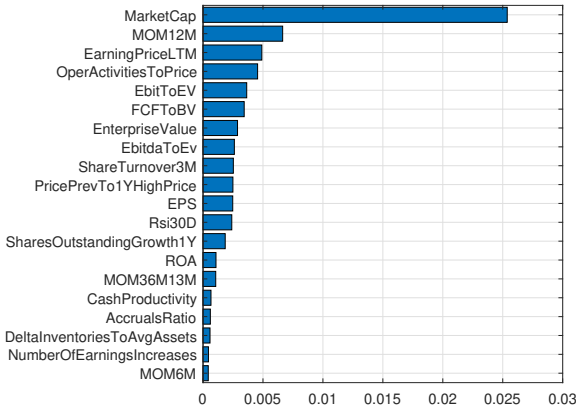
Table B.2 – Decile-based Portfolio Performances. Average returns and volatilities are annualized. D1 is the (predicted) worst-performing decile, D10 is the (predicted) best-performing decile.



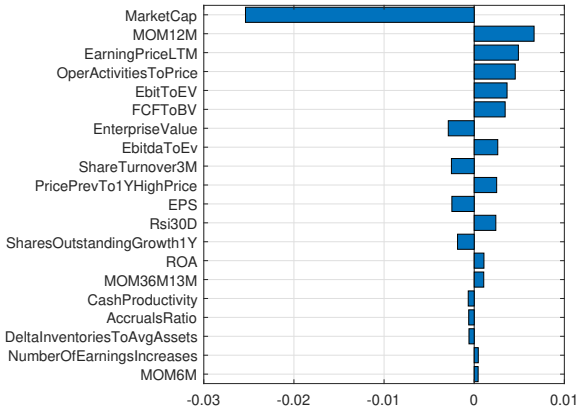
(a) OLS Average |Coefficient|



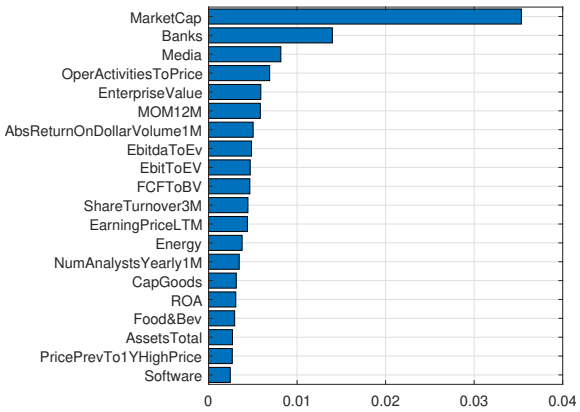
(b) OLS Average Coefficient



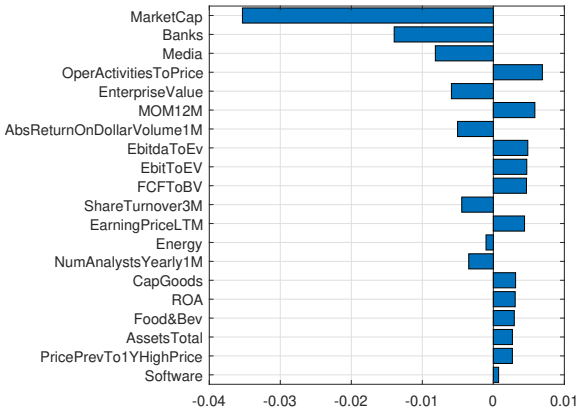
(c) LASSO Average |Coefficient|



(d) LASSO Average Coefficient

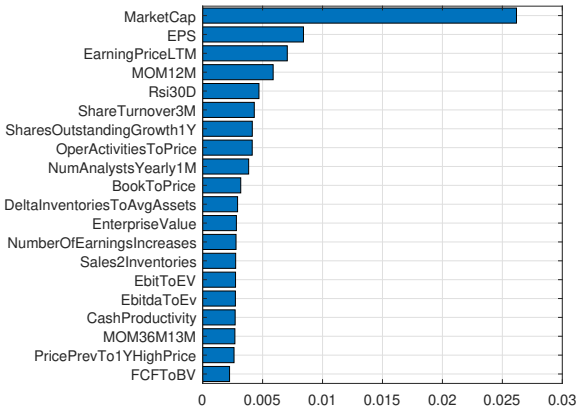


(e) VASA Average |Coefficient|

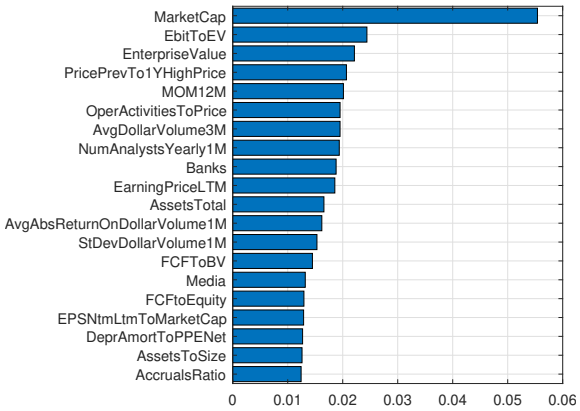


(f) VASA Average Coefficient

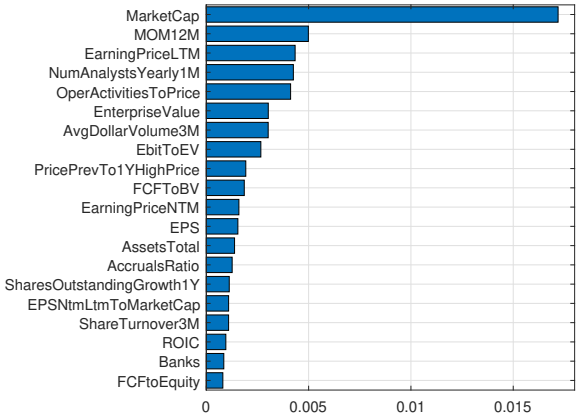
Figure B.12 – Averages of linear methods' coefficients and their absolute values.



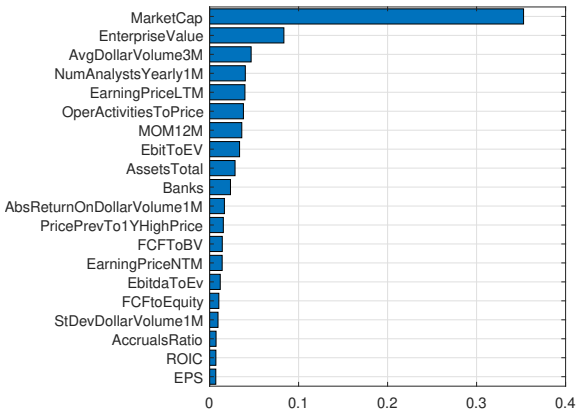
(a) Neural Networks Average |SHAP|



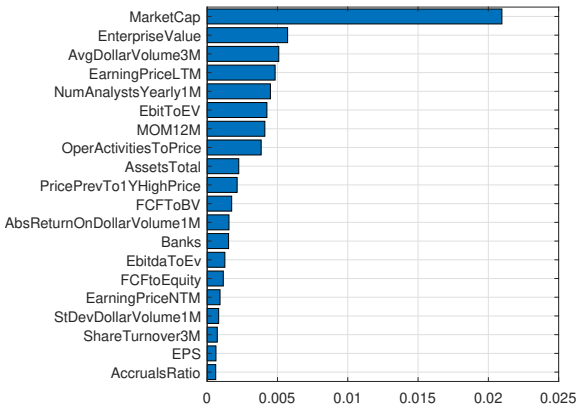
(b) XGboost Average Gain



(c) XGboost Average |SHAP|

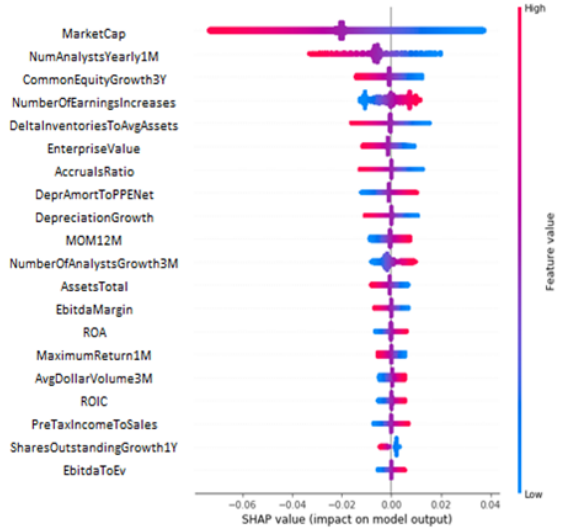


(d) Random Forest Average Importance

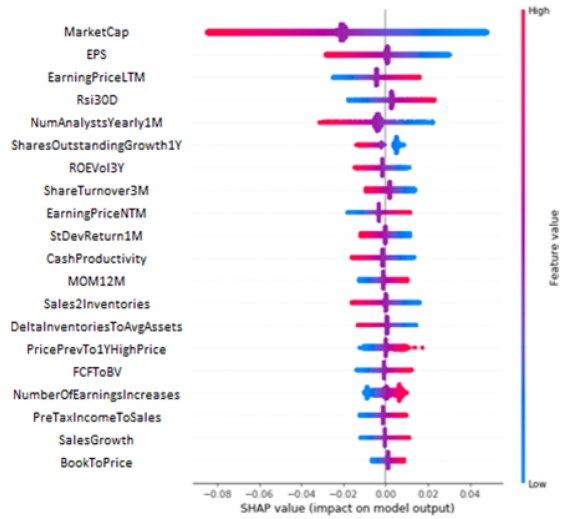


(e) Random Forest Average |SHAP|

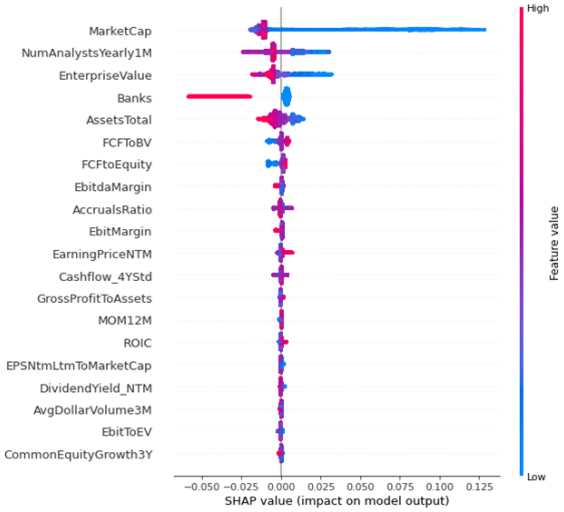
Figure B.13 – Averages of variables importance measures for non-linear methods.



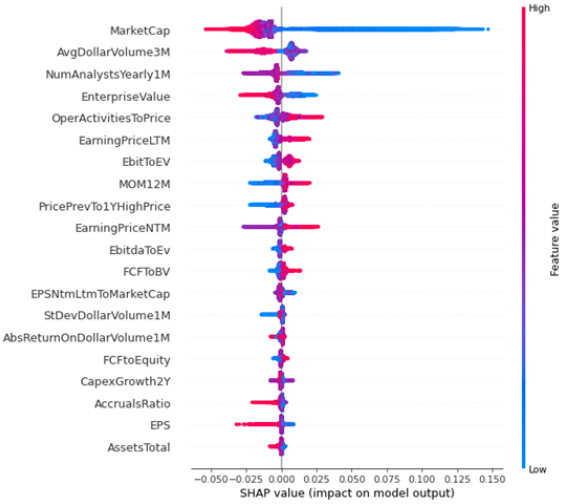
(a) Neural Networks, Rolling Window



(b) Neural Networks, Expanding Window



(c) Random Forest, Rolling Window



(d) Random Forest, Expanding Window

Figure B.14 – SHAP Summary plots for Neural Networks and Random Forest models trained as of May 2018.

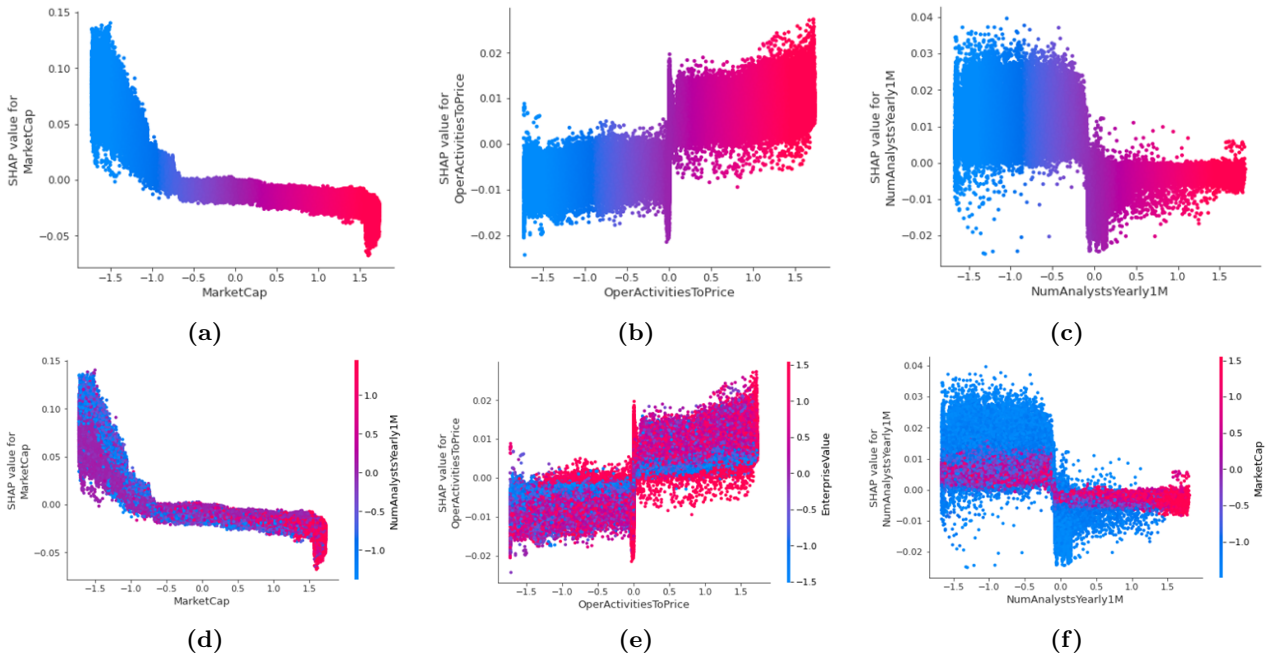


Figure B.15 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the expanding window as of May 2018.

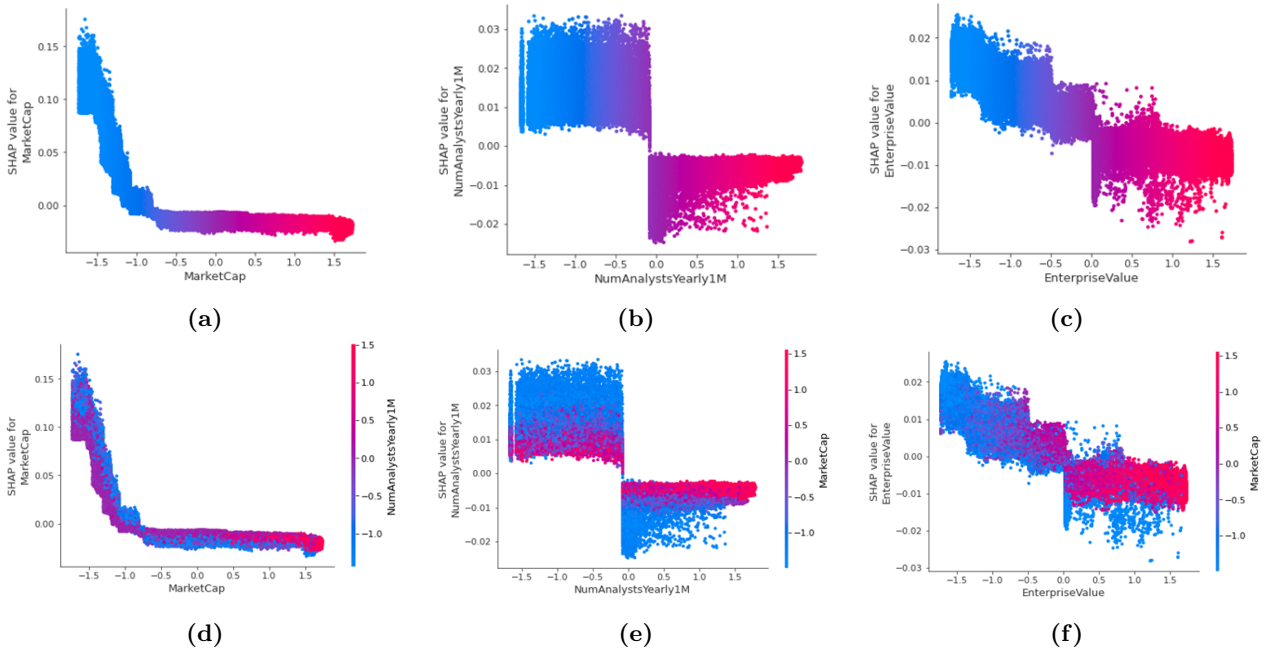


Figure B.16 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the rolling window as of May 2018.

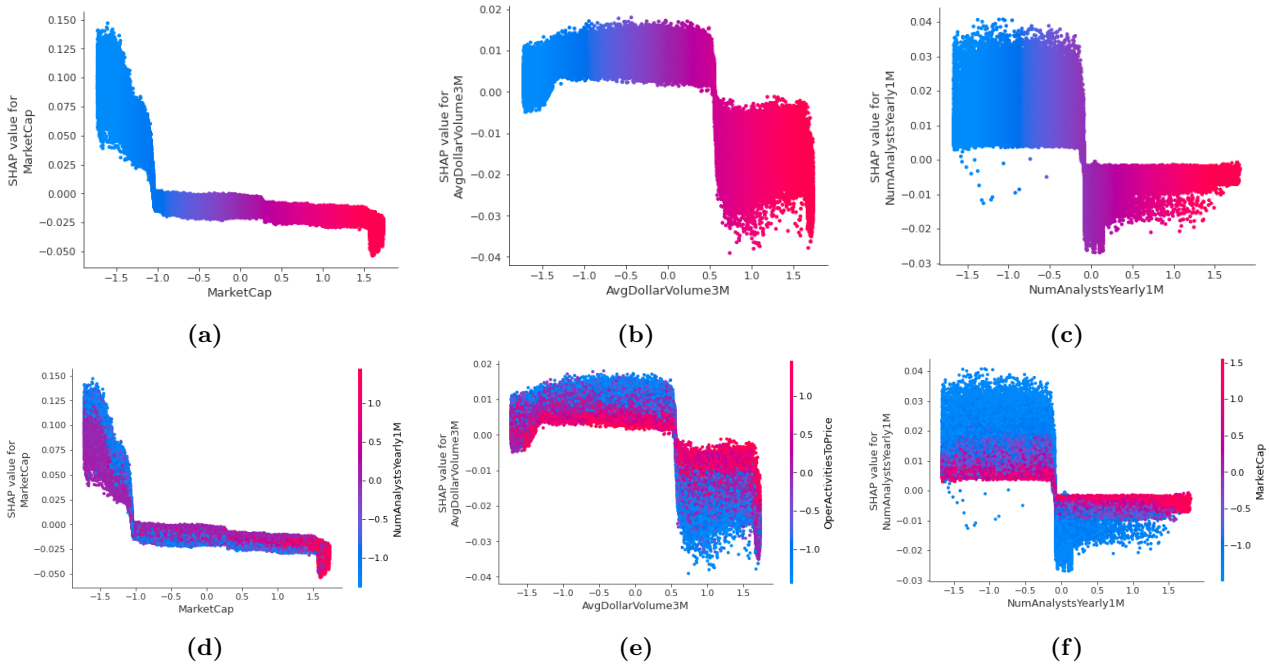


Figure B.17 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the expanding window as of May 2018.

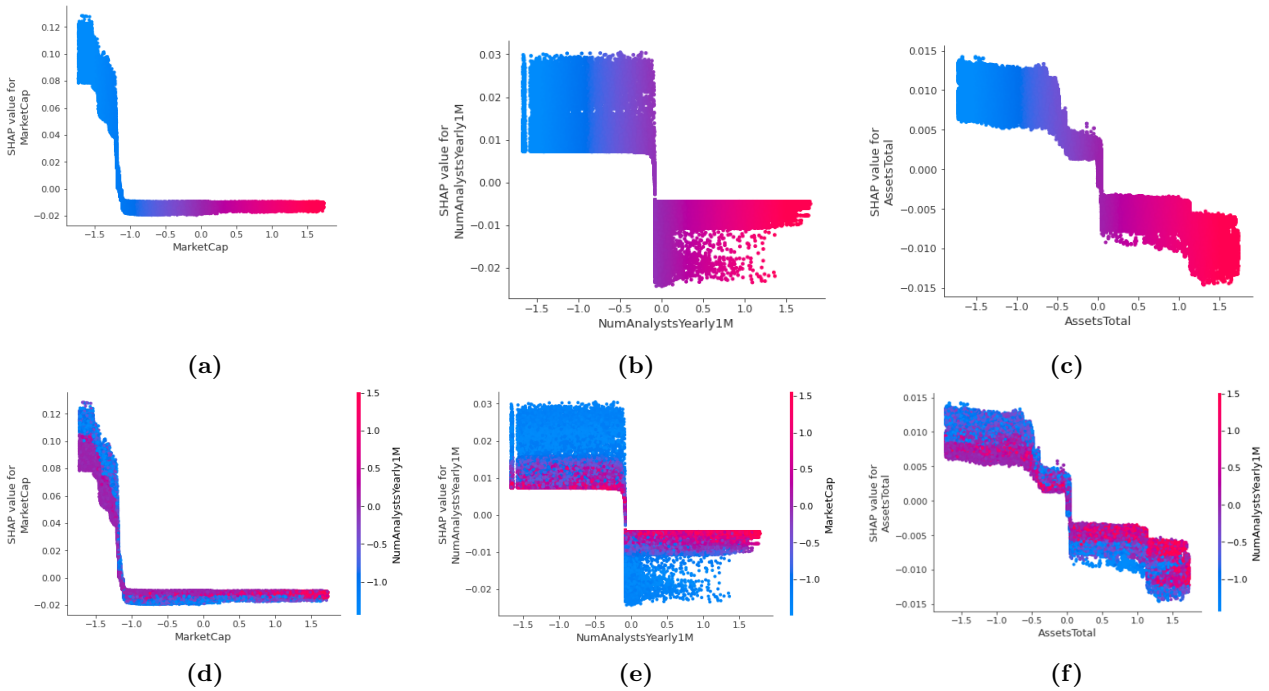
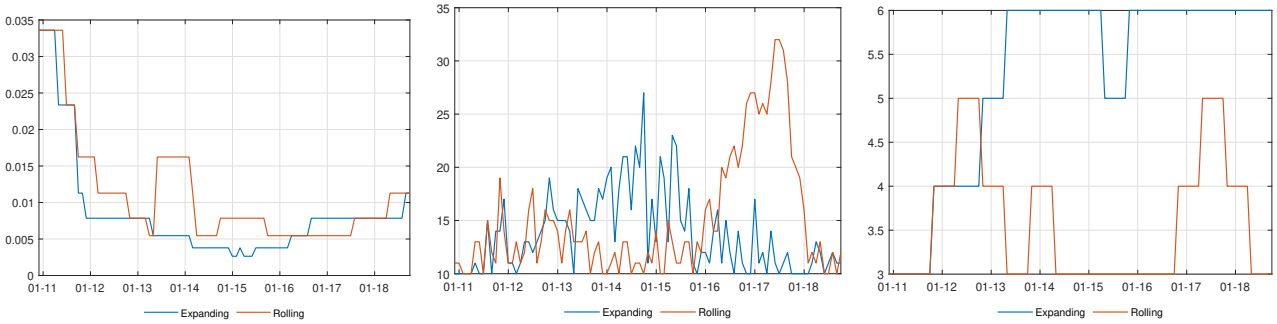


Figure B.18 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the rolling window as of May 2018.

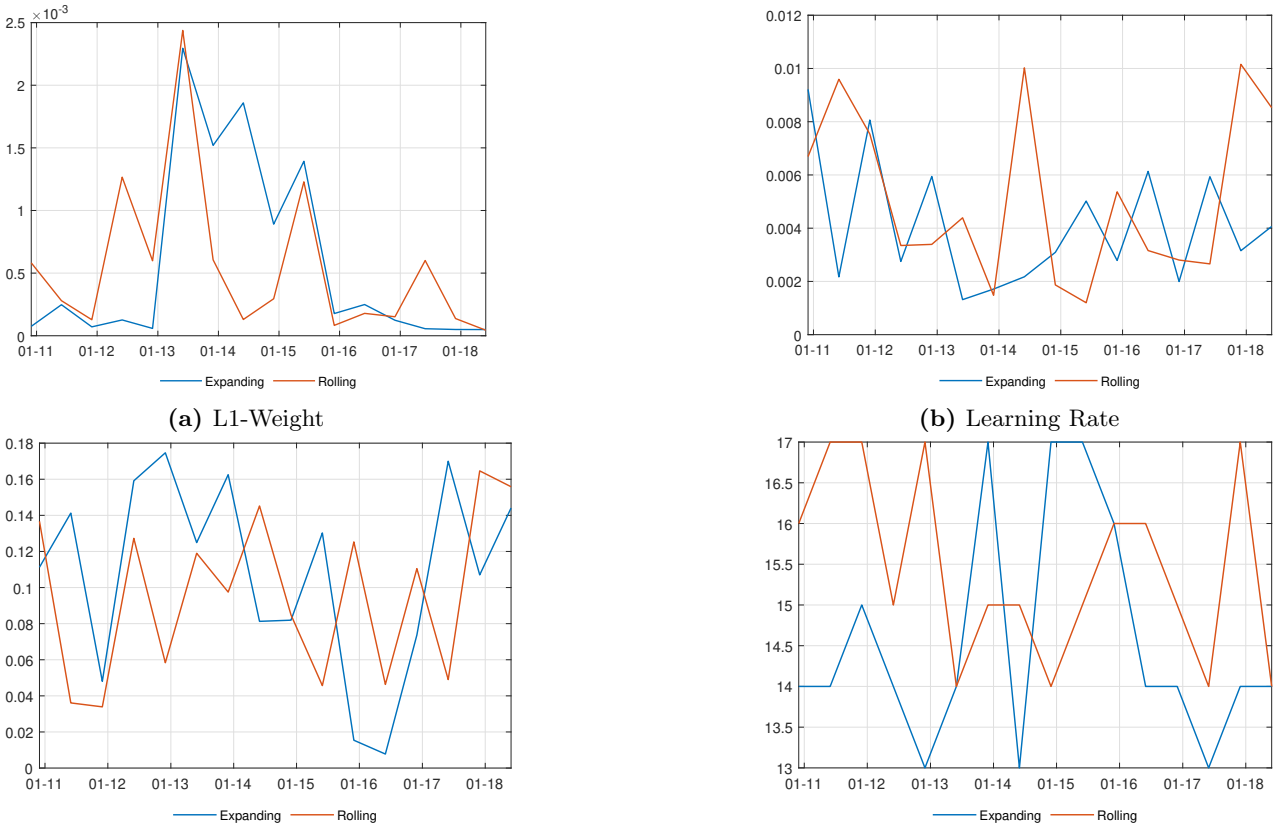


(a) LASSO L1-Weight

(b) VASA Number of features in sub-models

(c) Random Forest Maximum Depth

Figure B.19 – Hyperparameters choices for LASSO, VASA and Random Forest.



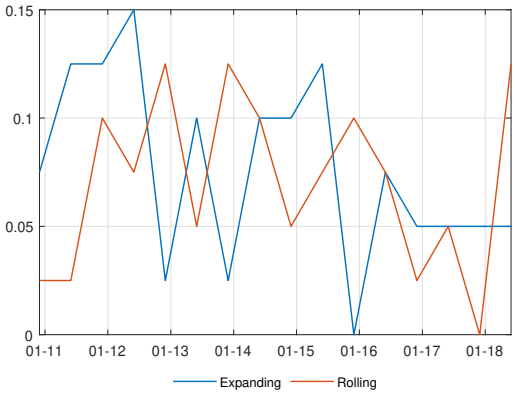
(a) L1-Weight

(b) Learning Rate

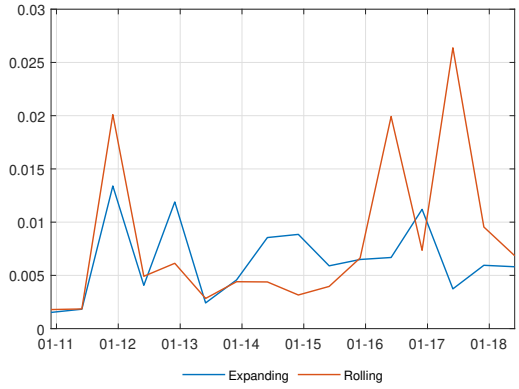
(c) Dropout Rate

(d) Epochs

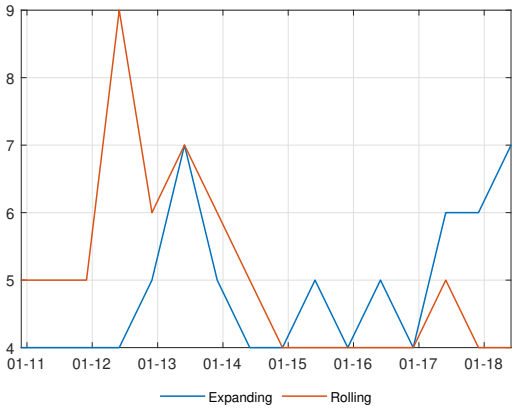
Figure B.20 – Neural Networks Hyperparameters choices.



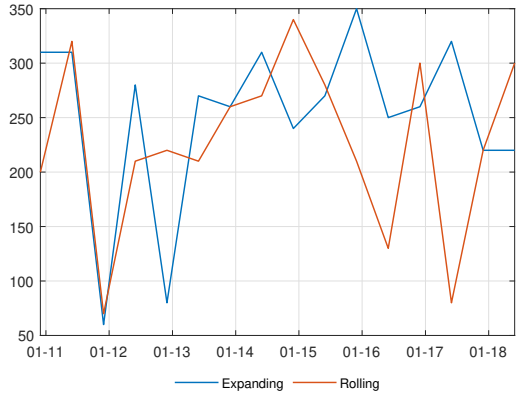
(a) Columns Dropout Rate



(b) Learning Rate



(c) Maximum Depth



(d) Number of trees

Figure B.21 – XGBoost Hyperparameters choices.

B.3 Japanese Data Set

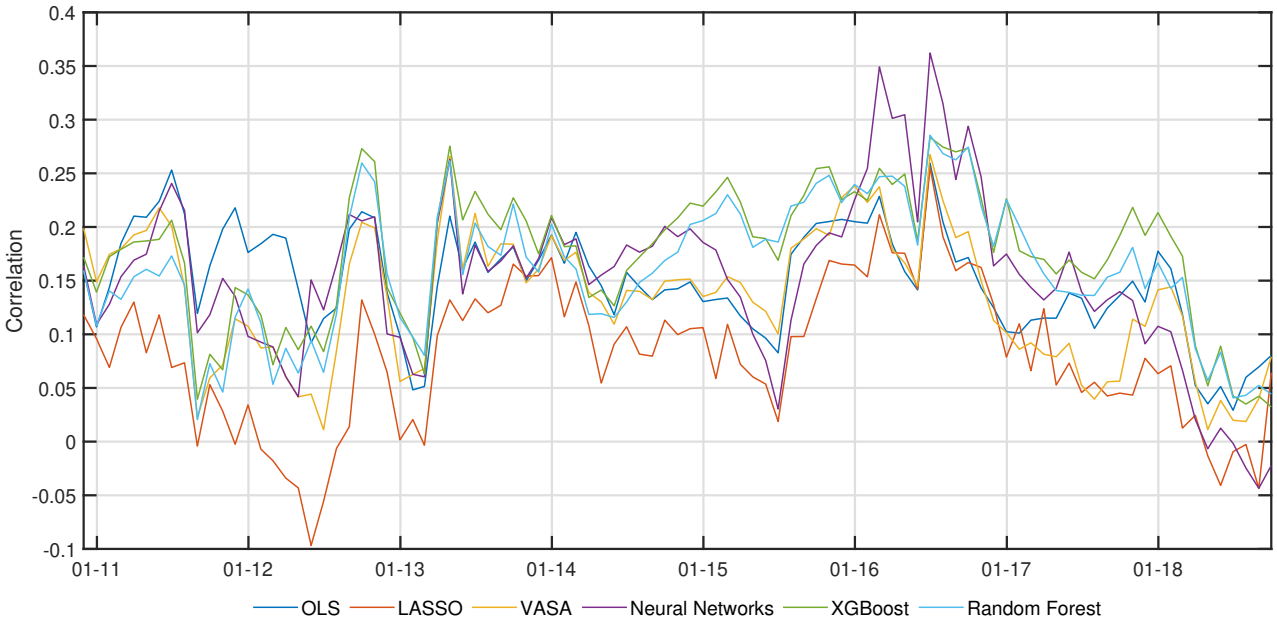


Figure B.22 – OOS Correlation through time.

	OLS									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	4,8%	6,3%	7,0%	8,6%	9,8%	9,6%	12,2%	13,3%	14,2%	15,3%
Volatility	13,4%	12,3%	11,8%	12,5%	12,7%	12,6%	12,5%	12,4%	12,7%	12,8%
Sharpe Ratio	0,354	0,514	0,598	0,688	0,773	0,760	0,980	1,075	1,118	1,195

	LASSO									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	7,5%	9,2%	7,9%	9,6%	9,1%	8,6%	11,3%	10,9%	13,0%	14,1%
Volatility	12,3%	12,9%	12,3%	12,7%	12,3%	11,9%	12,7%	12,7%	12,8%	12,4%
Sharpe Ratio	0,610	0,716	0,643	0,753	0,737	0,725	0,889	0,860	1,016	1,137

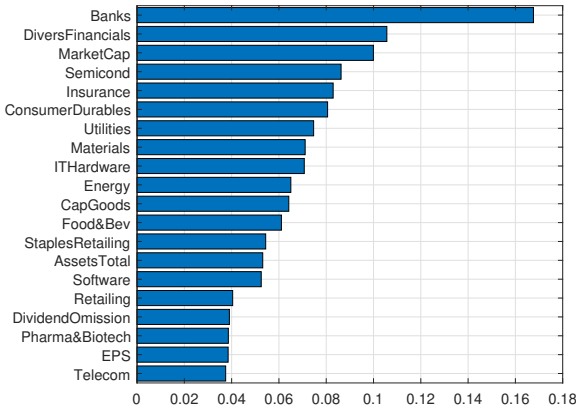
	VASA									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	5,8%	8,4%	8,7%	7,8%	9,3%	9,3%	11,4%	11,6%	13,8%	15,6%
Volatility	13,1%	12,8%	12,7%	12,6%	12,2%	12,5%	13,2%	12,7%	12,2%	12,1%
Sharpe Ratio	0,442	0,653	0,682	0,622	0,763	0,750	0,867	0,914	1,132	1,285

	Neural Networks									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	5,4%	6,9%	8,9%	7,7%	9,4%	10,6%	11,3%	11,7%	12,8%	16,0%
Volatility	13,5%	12,3%	12,4%	12,2%	12,0%	12,8%	13,2%	12,8%	12,9%	13,2%
Sharpe Ratio	0,402	0,562	0,714	0,633	0,788	0,826	0,857	0,912	0,990	1,212

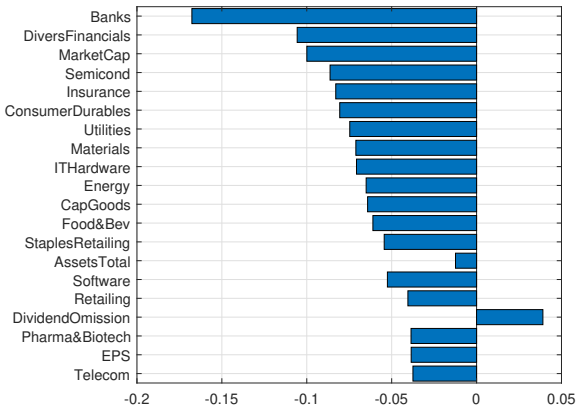
	XGBoost									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	4,5%	7,3%	7,6%	9,6%	8,0%	10,2%	11,3%	12,8%	13,5%	16,1%
Volatility	12,8%	12,3%	12,5%	12,5%	12,3%	11,9%	12,6%	12,3%	12,5%	13,8%
Sharpe Ratio	0,348	0,591	0,609	0,771	0,653	0,861	0,892	1,040	1,085	1,168

	Random Forest									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	4,4%	8,6%	9,8%	9,6%	8,9%	8,6%	11,5%	11,6%	13,7%	14,8%
Volatility	12,9%	12,8%	12,5%	12,3%	12,1%	12,3%	12,6%	12,7%	12,7%	13,6%
Sharpe Ratio	0,340	0,669	0,782	0,775	0,736	0,702	0,912	0,912	1,082	1,088

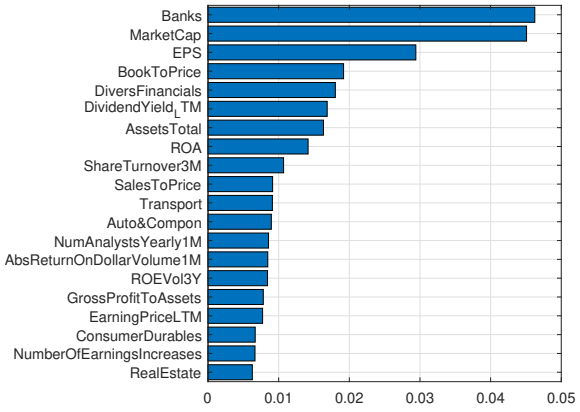
Table B.3 – Decile-based Portfolios' Performances. Average returns and volatilities are annualized. D1 is the (predicted) worst-performing decile, D10 is the (predicted) best-performing decile.



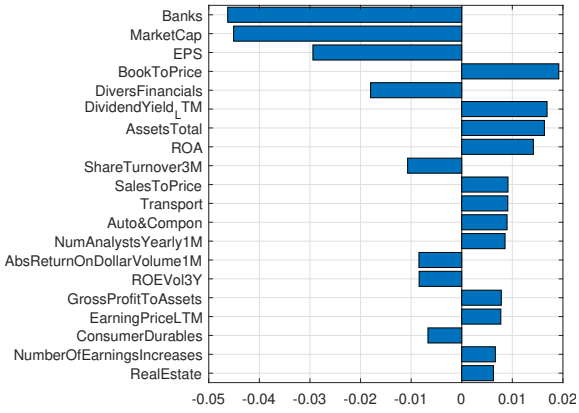
(a) OLS Average |Coefficient|



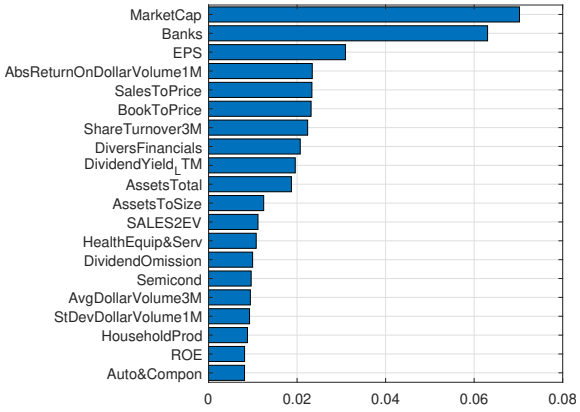
(b) OLS Average Coefficient



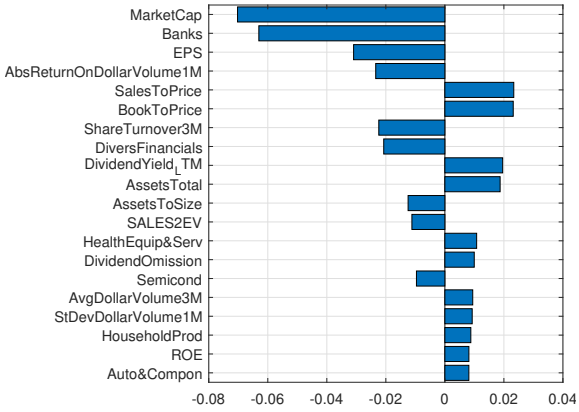
(c) LASSO Average |Coefficient|



(d) LASSO Average Coefficient

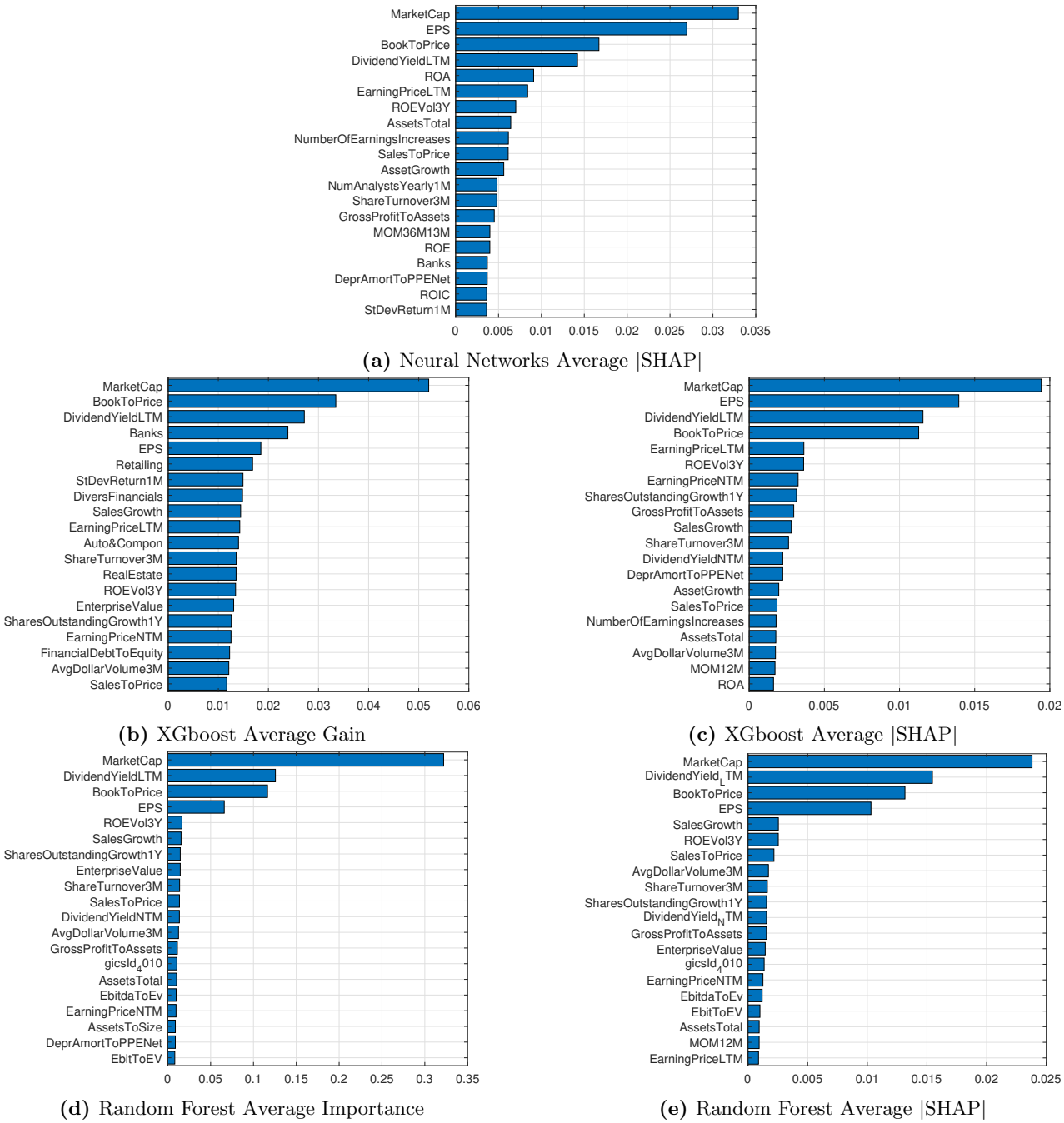


(e) VASA Average |Coefficient|



(f) VASA Average Coefficient

Figure B.23 – Averages of linear methods' coefficients and their absolute values.



(a) Neural Networks Average |SHAP|

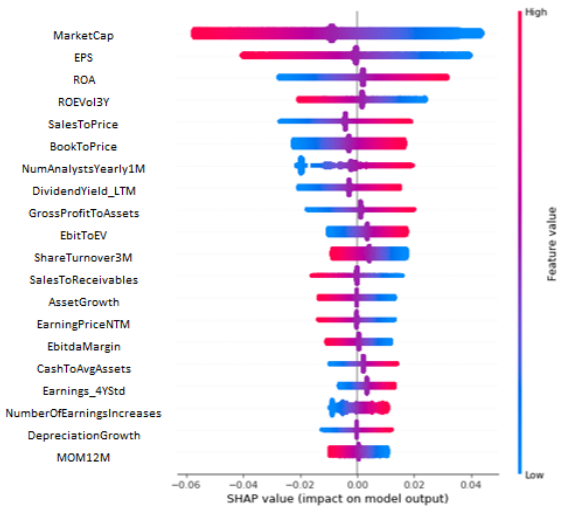
(b) XGboost Average Gain

(c) XGboost Average |SHAP|

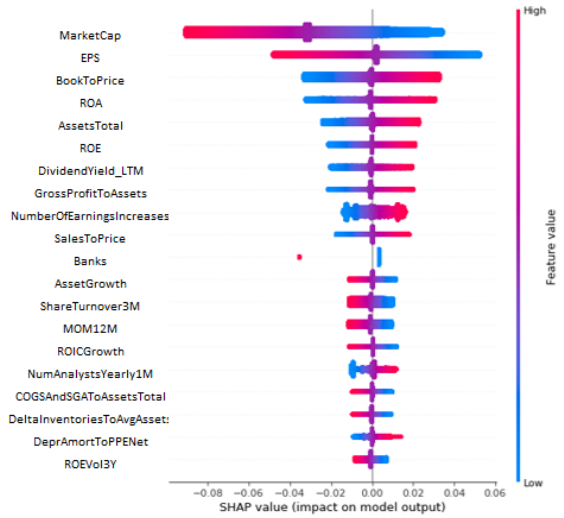
(d) Random Forest Average Importance

(e) Random Forest Average |SHAP|

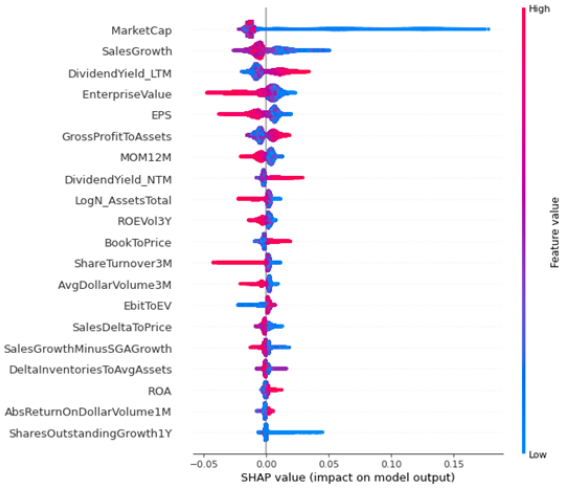
Figure B.24 – Averages of variables importance measures for non-linear methods.



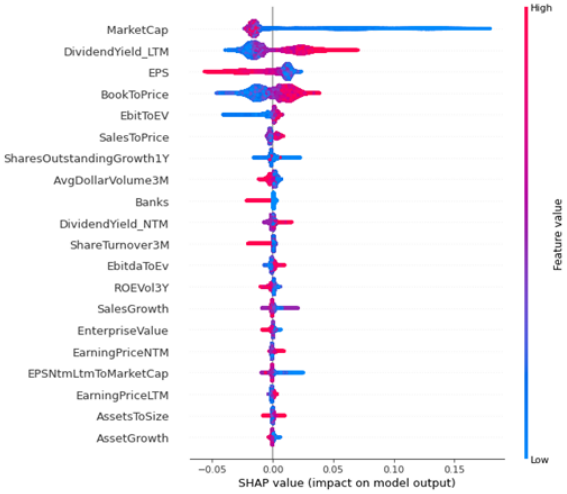
(a) Neural Networks, Rolling Window



(b) Neural Networks, Expanding Window



(c) Random Forest, Rolling Window



(d) Random Forest, Expanding Window

Figure B.25 – SHAP Summary plots for Neural Networks and Random Forest models trained as of May 2018.

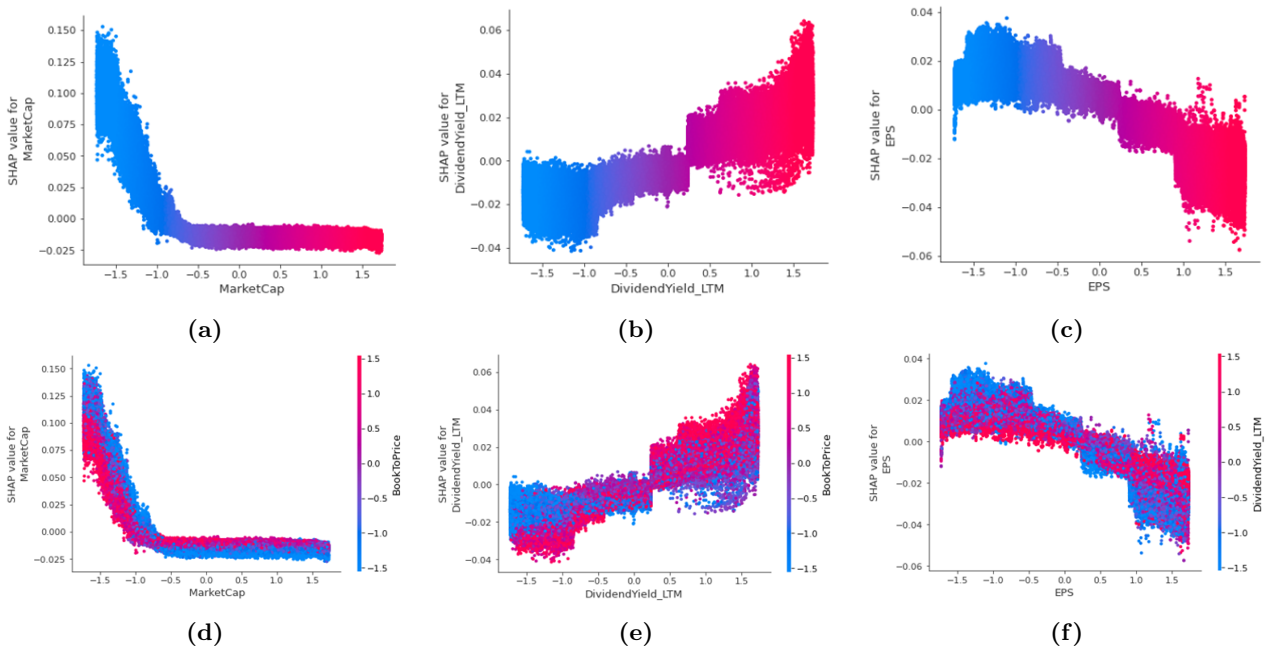


Figure B.26 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the expanding window as of May 2018.

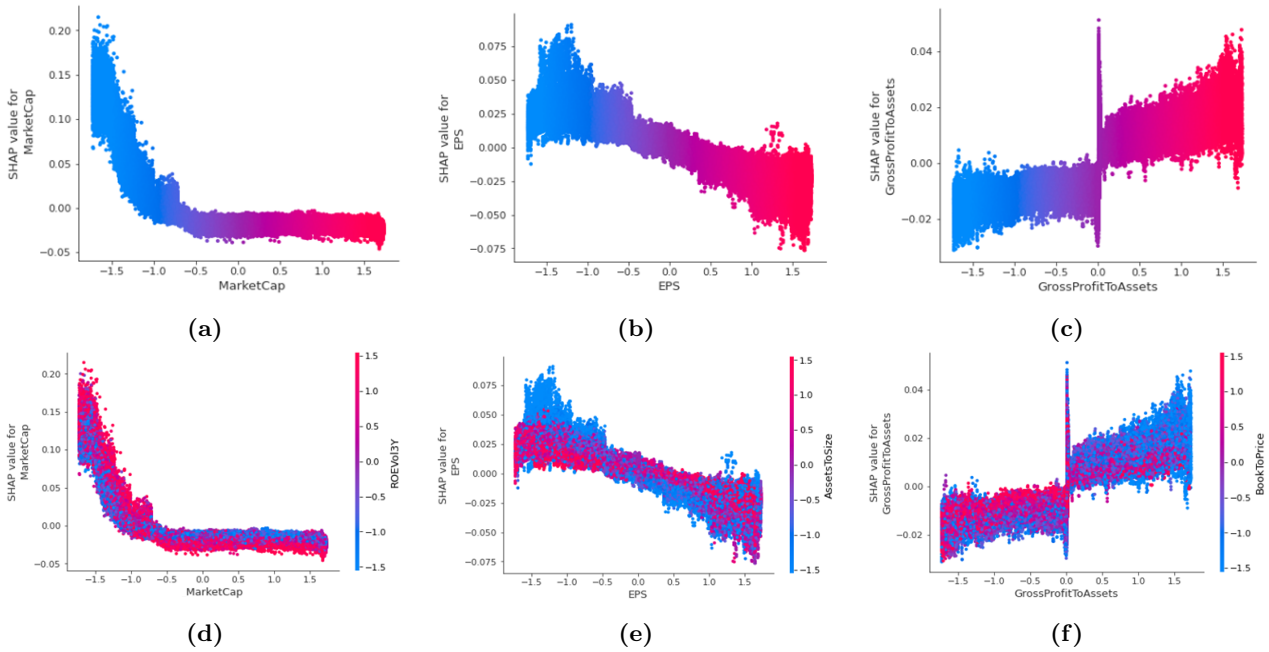


Figure B.27 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the rolling window as of May 2018.

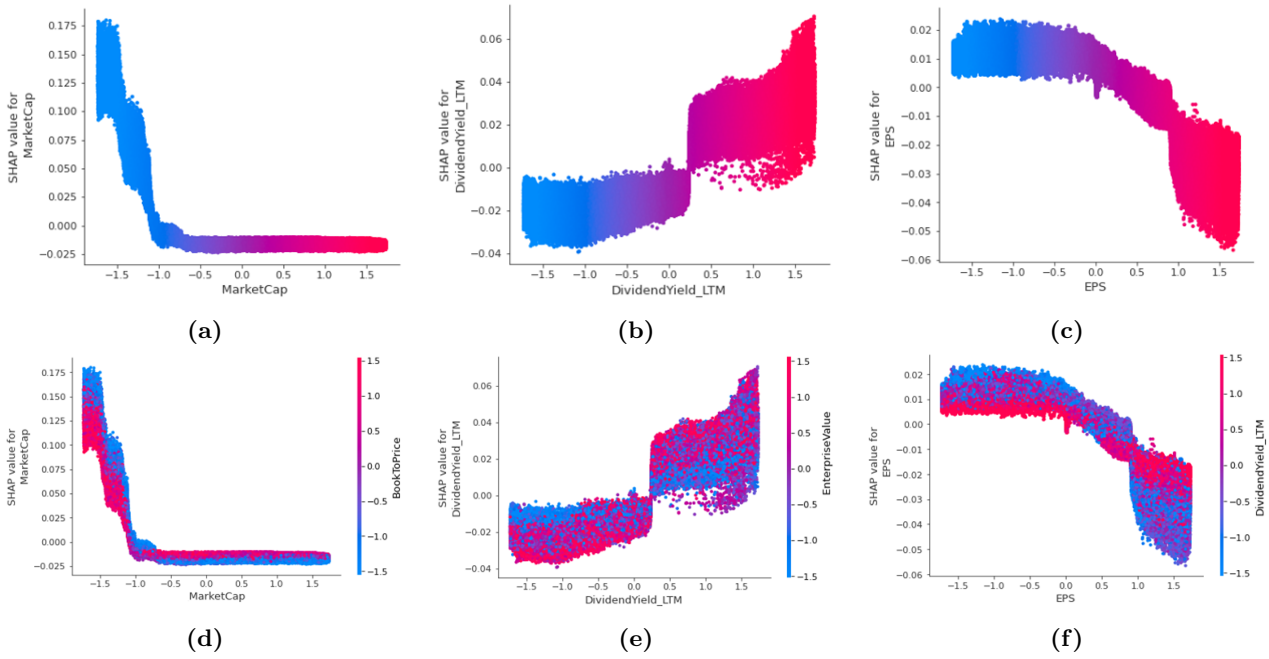


Figure B.28 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the expanding window as of May 2018.

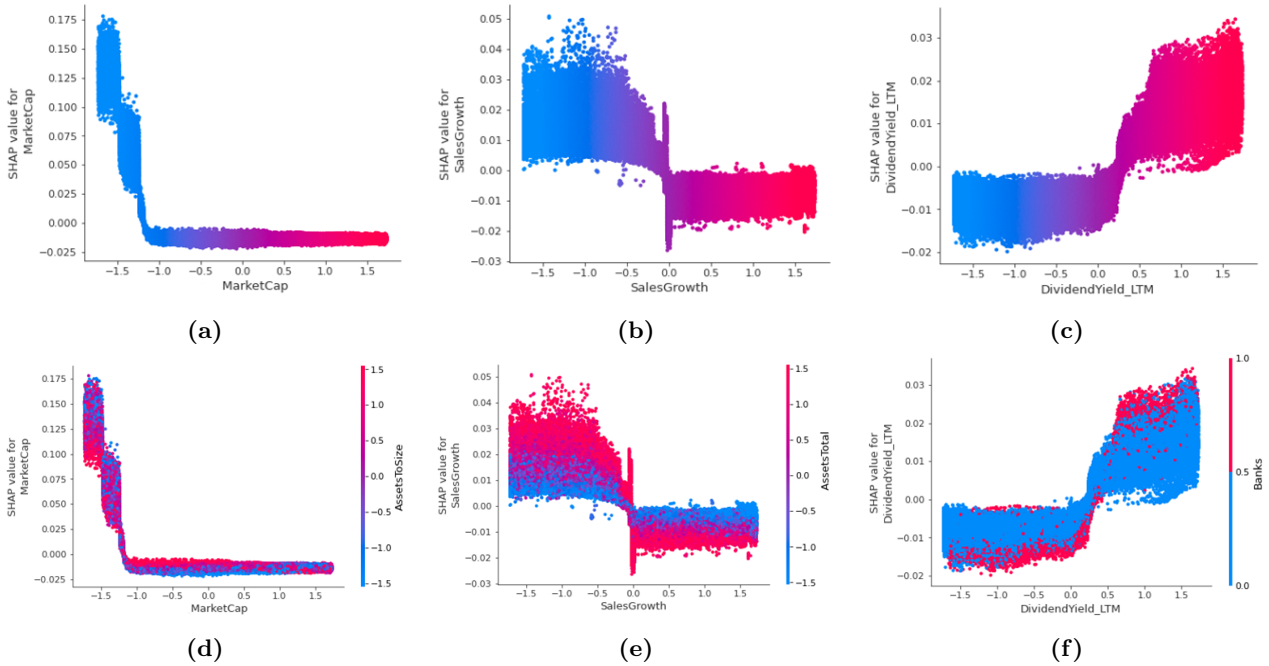
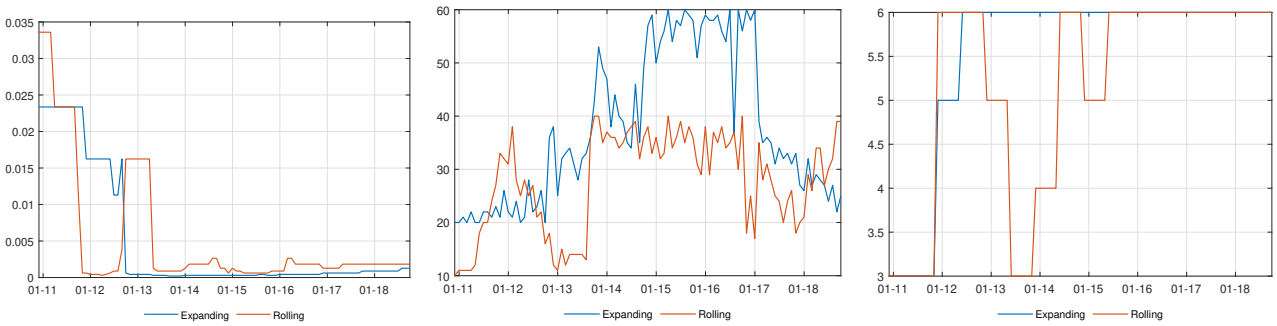


Figure B.29 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the rolling window as of May 2018.

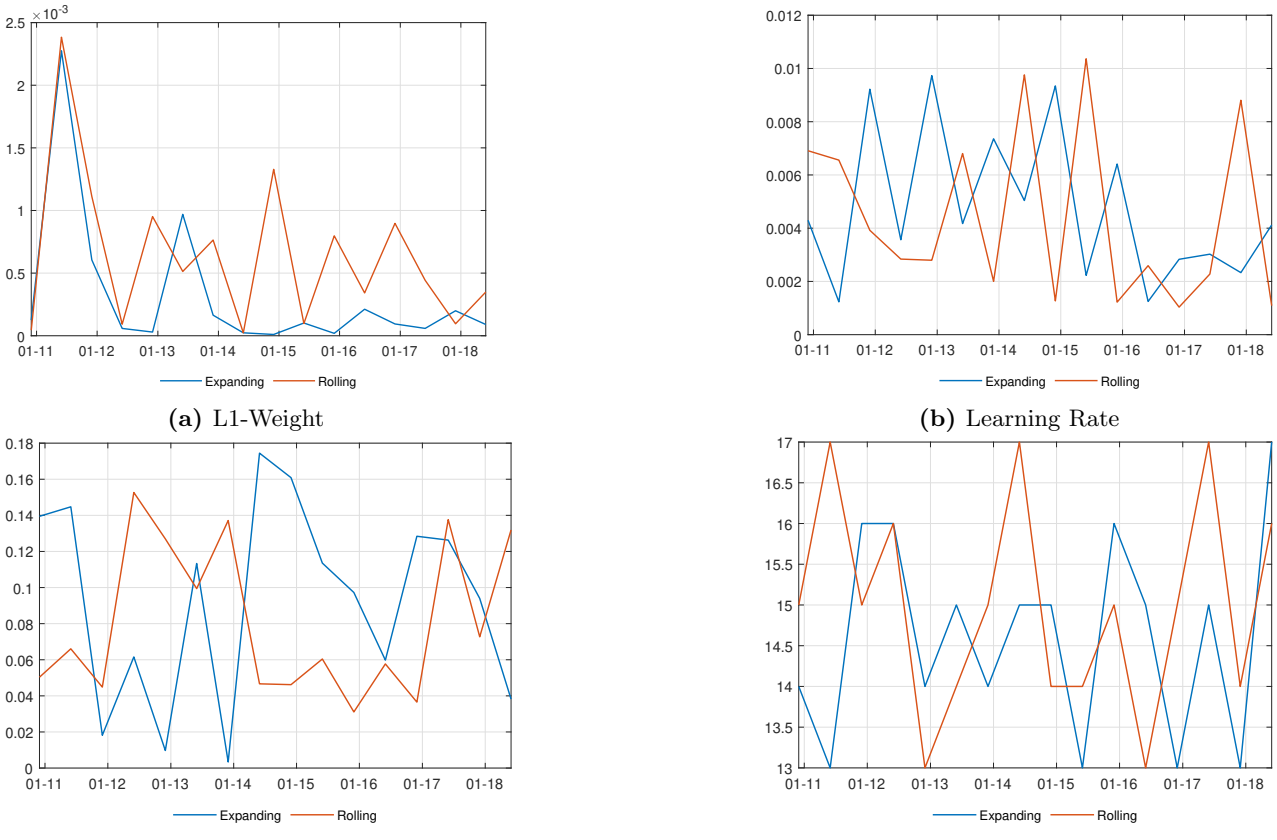


(a) LASSO L1-Weight

(b) VASA Number of features in sub-models

(c) Random Forest Maximum Depth

Figure B.30 – Hyperparameters choices for LASSO, VASA and Random Forest.



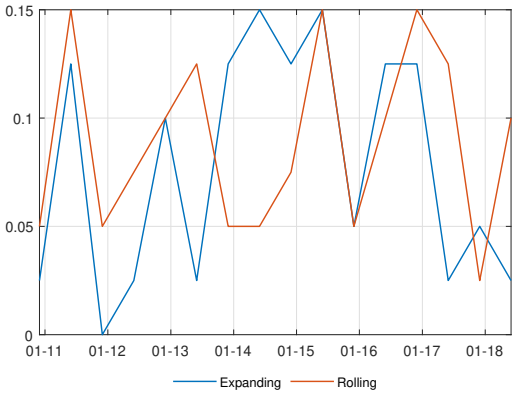
(a) L1-Weight

(b) Learning Rate

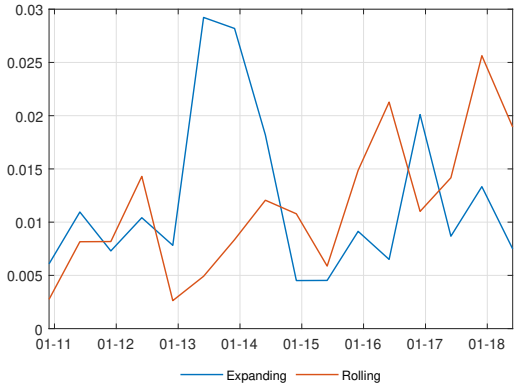
(c) Dropout Rate

(d) Epochs

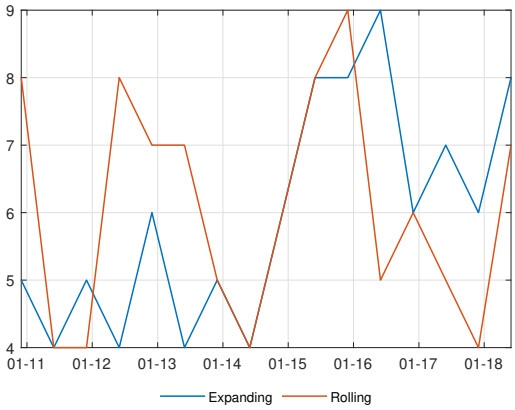
Figure B.31 – Neural Networks Hyperparameters choices.



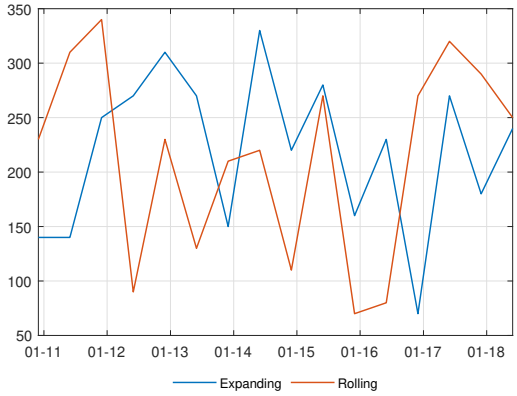
(a) Columns Dropout Rate



(b) Learning Rate



(c) Maximum Depth



(d) Number of trees

Figure B.32 – XGBoost Hyperparameters choices.

B.4 United States Data Set

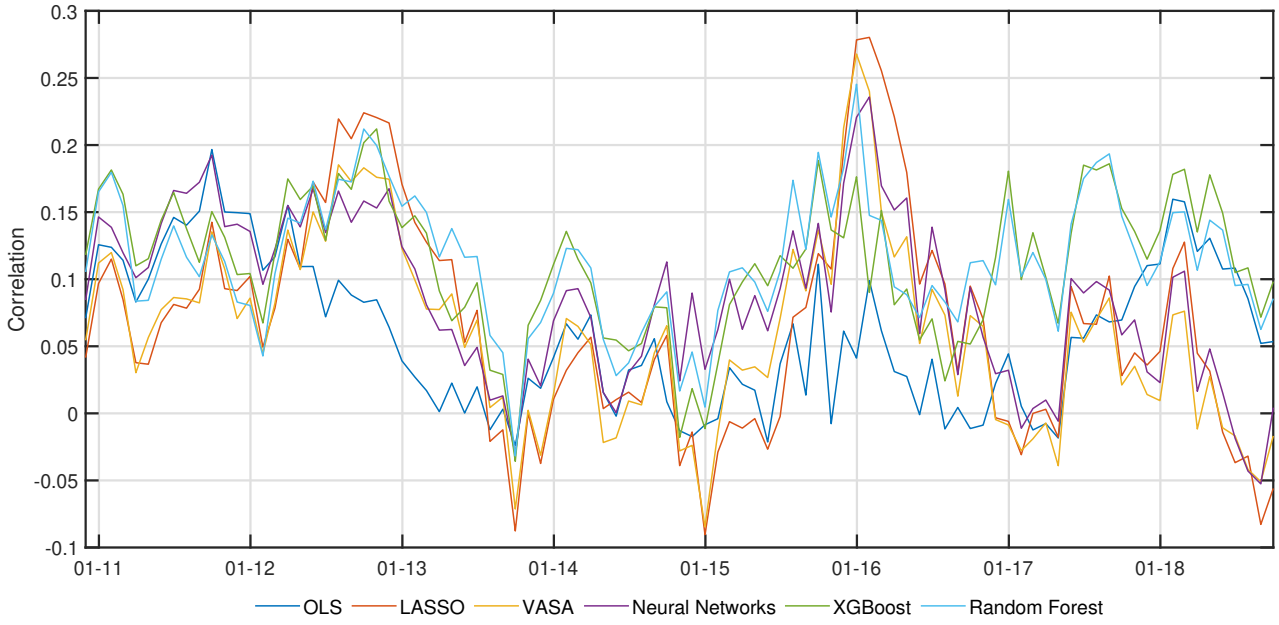


Figure B.33 – OOS Correlation through time.

	OLS									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	10,9%	12,3%	10,3%	13,5%	10,9%	12,3%	12,0%	12,9%	13,8%	12,3%
Volatility	13,0%	12,1%	12,4%	13,0%	12,9%	13,8%	13,7%	13,9%	14,2%	15,3%
Sharpe Ratio	0,842	1,016	0,832	1,039	0,844	0,894	0,878	0,929	0,971	0,803

	LASSO									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	12,6%	12,5%	11,4%	12,9%	14,0%	11,9%	12,4%	12,7%	16,2%	20,5%
Volatility	11,6%	11,7%	11,7%	12,7%	12,0%	12,7%	12,8%	13,3%	13,6%	13,7%
Sharpe Ratio	1,084	1,073	0,972	1,017	1,162	0,936	0,964	0,956	1,189	1,492

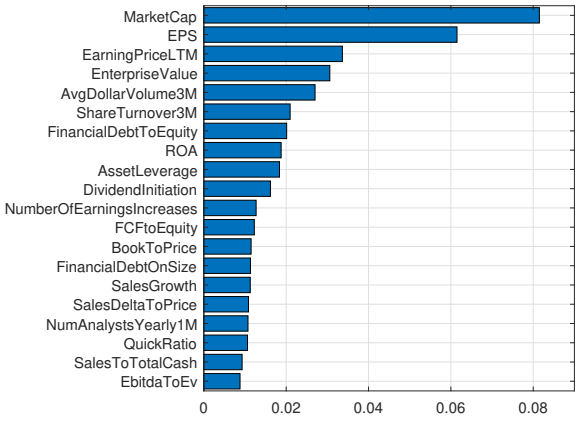
	VASA									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	12,2%	12,5%	12,5%	13,8%	13,1%	12,2%	12,9%	13,6%	14,9%	19,5%
Volatility	13,4%	11,6%	11,9%	11,7%	12,3%	12,5%	12,8%	12,8%	13,2%	14,3%
Sharpe Ratio	0,913	1,075	1,051	1,182	1,066	0,976	1,006	1,060	1,131	1,357

	Neural Networks									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	11,6%	11,9%	11,5%	12,3%	11,2%	13,2%	13,4%	13,1%	16,7%	20,3%
Volatility	12,8%	12,4%	12,4%	12,0%	12,5%	12,1%	12,3%	12,7%	13,8%	13,7%
Sharpe Ratio	0,906	0,964	0,932	1,021	0,901	1,096	1,090	1,028	1,207	1,480

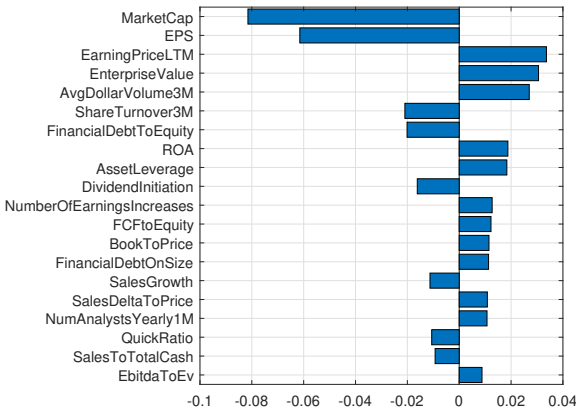
	XGBoost									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	12,3%	11,8%	11,1%	11,5%	13,8%	12,4%	11,8%	12,9%	14,0%	23,4%
Volatility	14,8%	12,5%	11,6%	11,7%	11,9%	11,4%	12,4%	13,0%	13,1%	14,8%
Sharpe Ratio	0,830	0,944	0,955	0,981	1,162	1,091	0,948	0,995	1,066	1,584

	Random Forest									
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Avg. Returns	13,0%	12,2%	10,5%	13,2%	12,8%	11,5%	12,6%	12,9%	14,9%	23,2%
Volatility	14,9%	12,5%	11,9%	12,1%	11,5%	11,9%	12,4%	12,8%	13,5%	14,4%
Sharpe Ratio	0,874	0,974	0,883	1,089	1,113	0,973	1,016	1,006	1,101	1,607

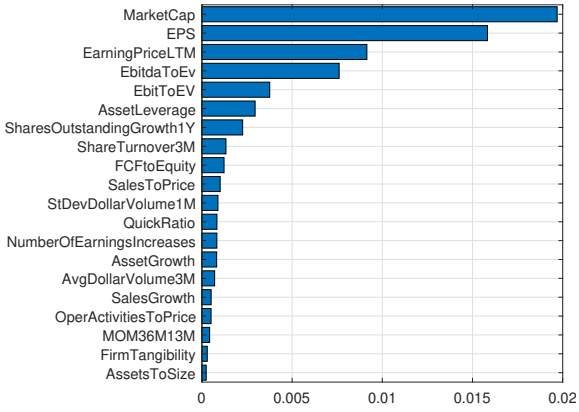
Table B.4 – Decile-based Portfolios' Performances. Average returns and volatilities are annualized. D1 is the (predicted) worst-performing decile, D10 is the (predicted) best-performing decile.



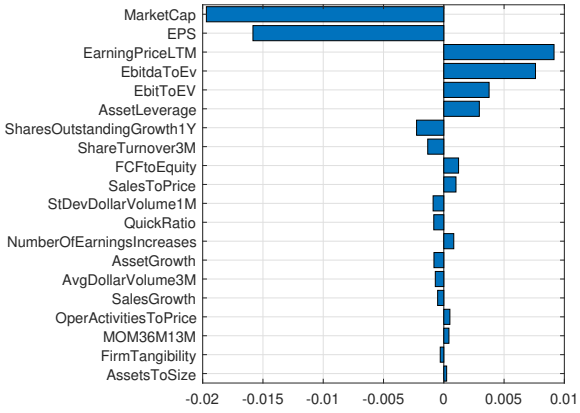
(a) OLS Average |Coefficient|



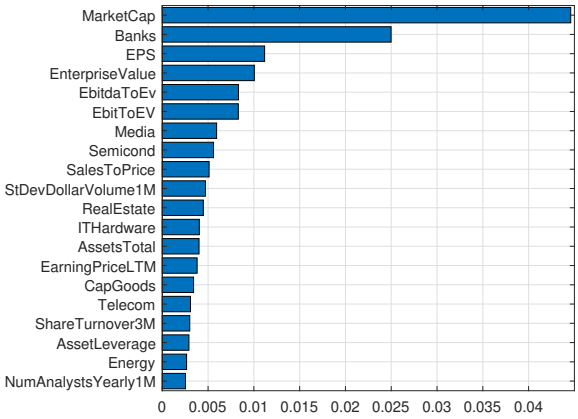
(b) OLS Average Coefficient



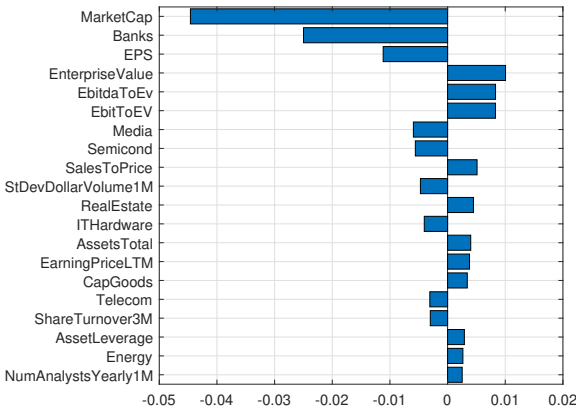
(c) LASSO Average |Coefficient|



(d) LASSO Average Coefficient

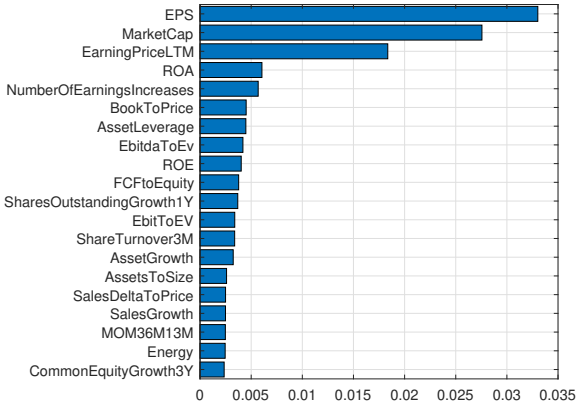


(e) VASA Average |Coefficient|

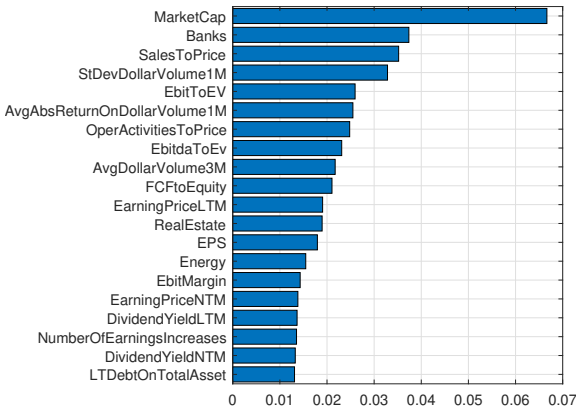


(f) VASA Average Coefficient

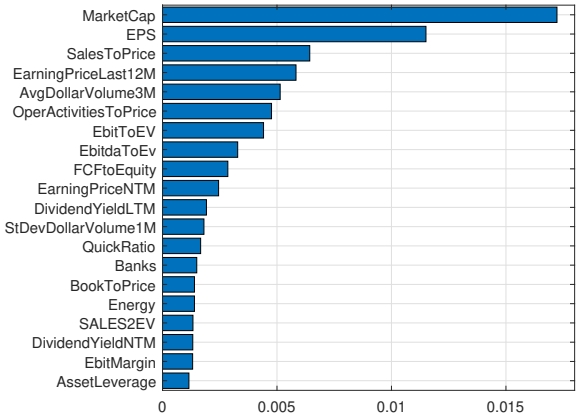
Figure B.34 – Averages of linear methods' coefficients and their absolute values.



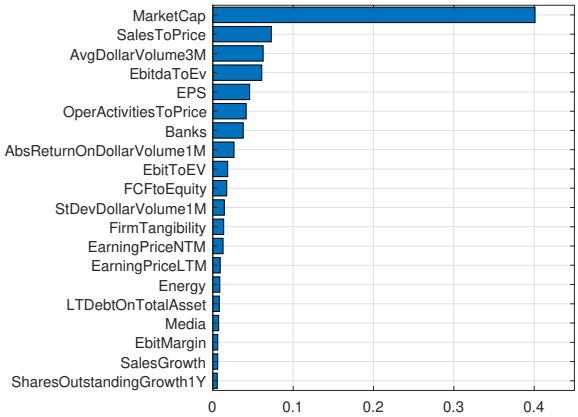
(a) Neural Networks Average |SHAP|



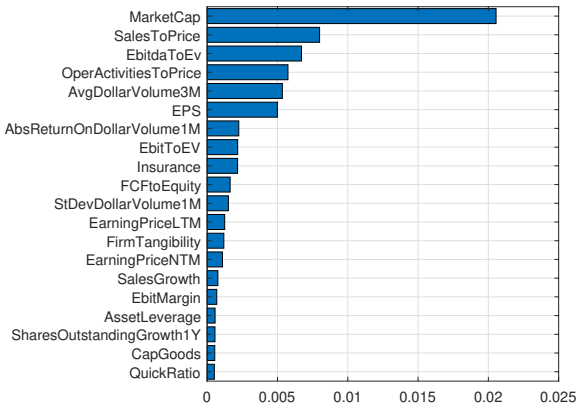
(b) XGboost Average Gain



(c) XGboost Average |SHAP|

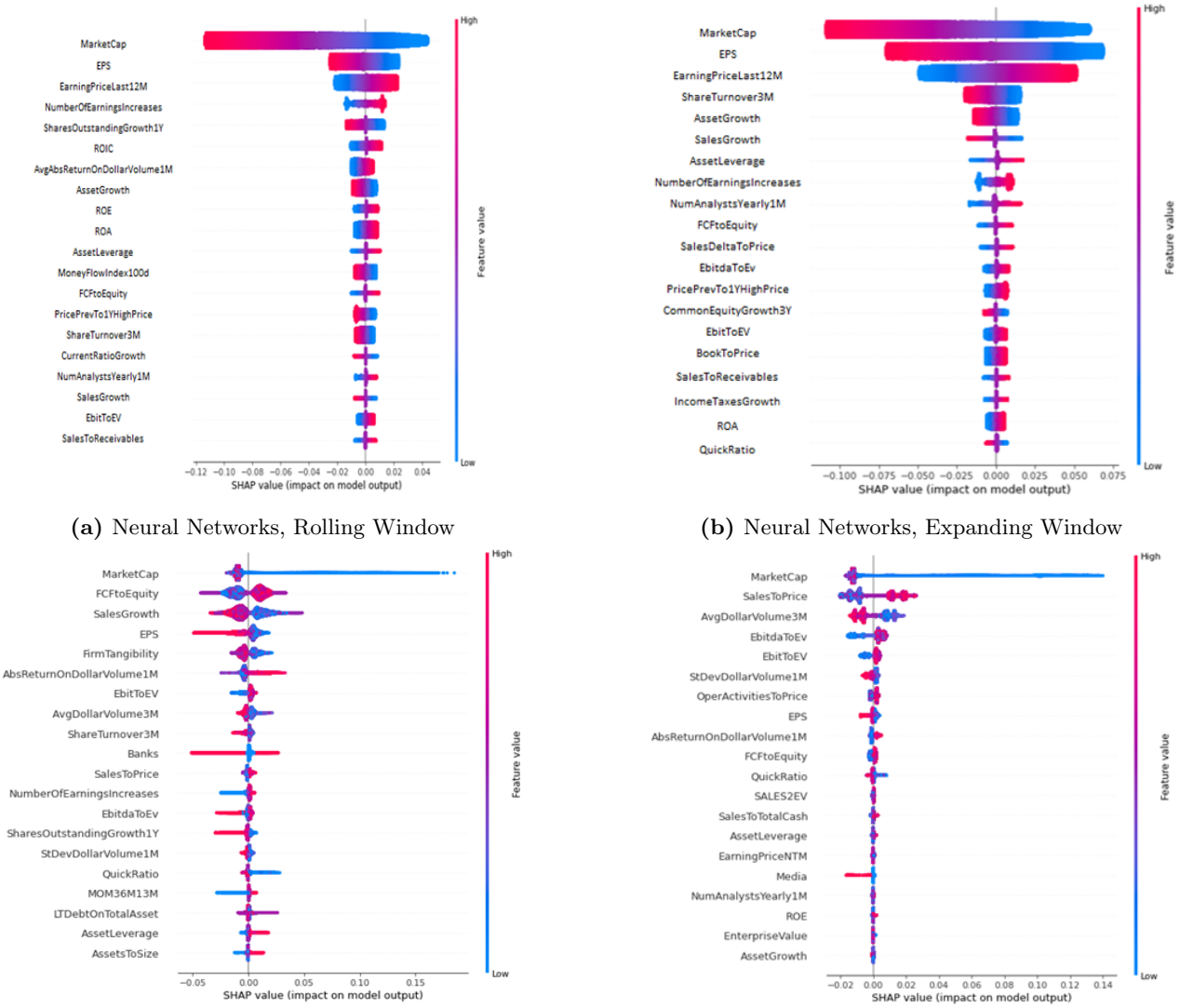


(d) Random Forest Average Importance



(e) Random Forest Average |SHAP|

Figure B.35 – Averages of variables importance measures for non-linear methods.



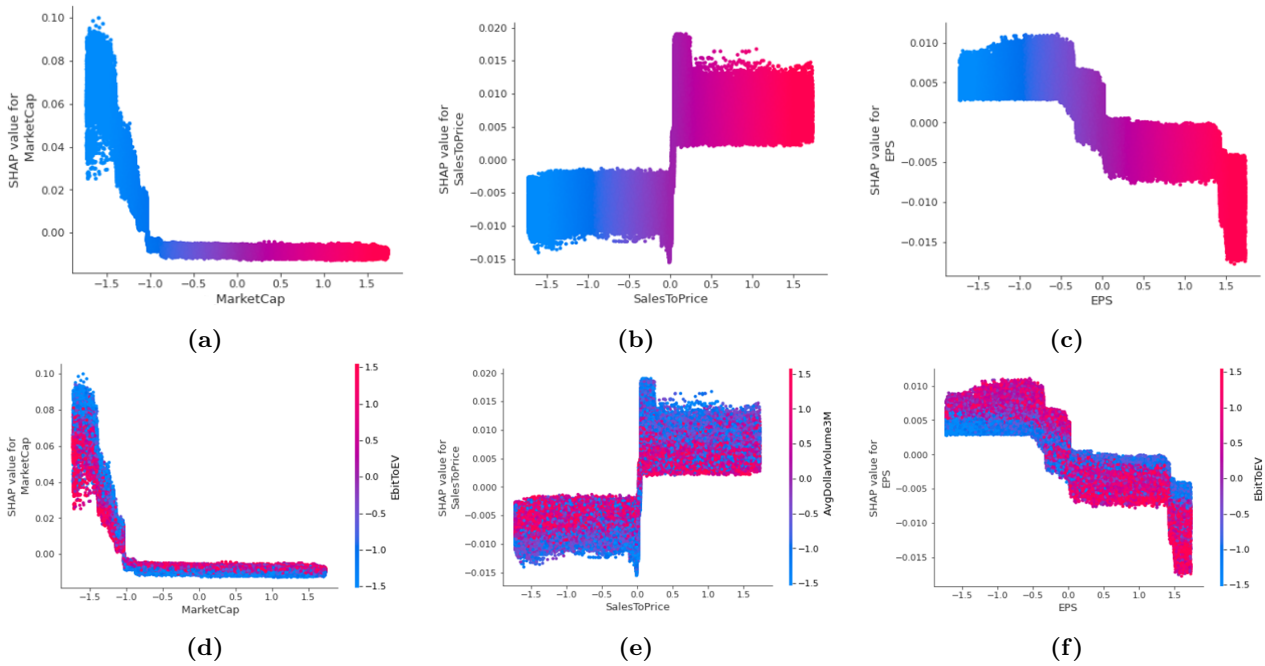
(a) Neural Networks, Rolling Window

(b) Neural Networks, Expanding Window

(c) Random Forest, Rolling Window

(d) Random Forest, Expanding Window

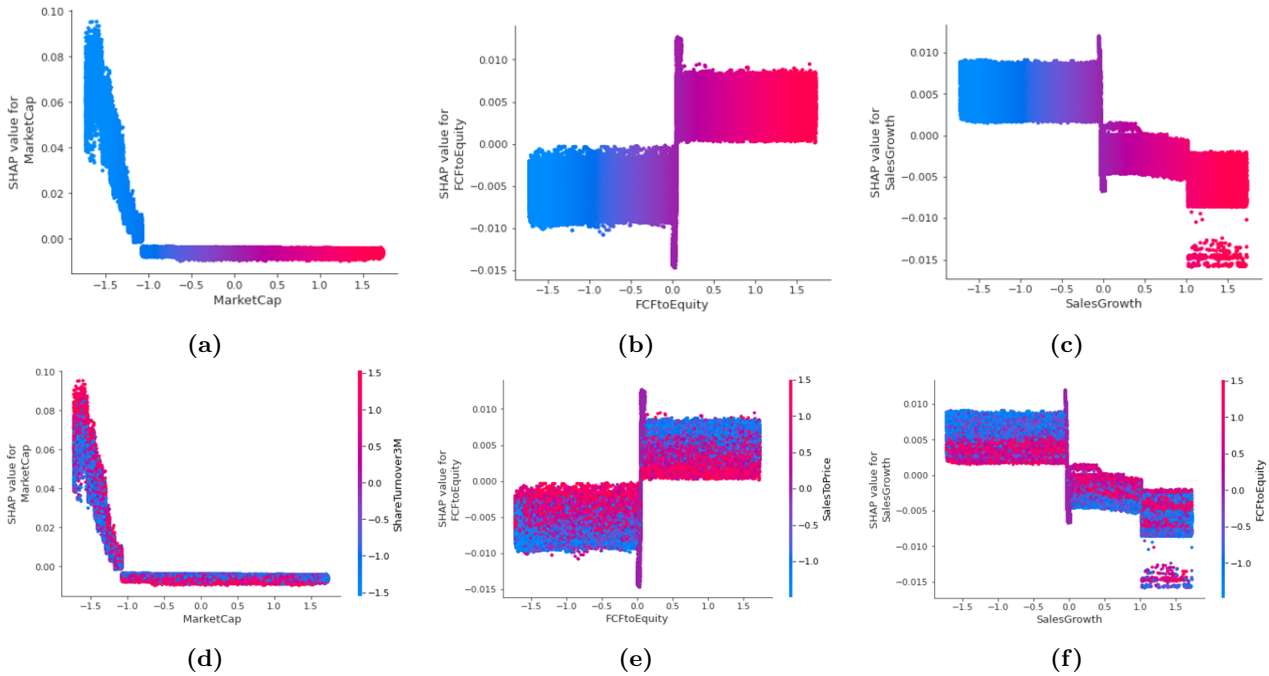
Figure B.36 – SHAP Summary plots for Neural Networks and Random Forest models trained as of May 2018.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure B.37 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the expanding window as of May 2018.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure B.38 – SHAP Dependency Plots of the top 3 variables for XGBoost, trained on the rolling window as of May 2018.

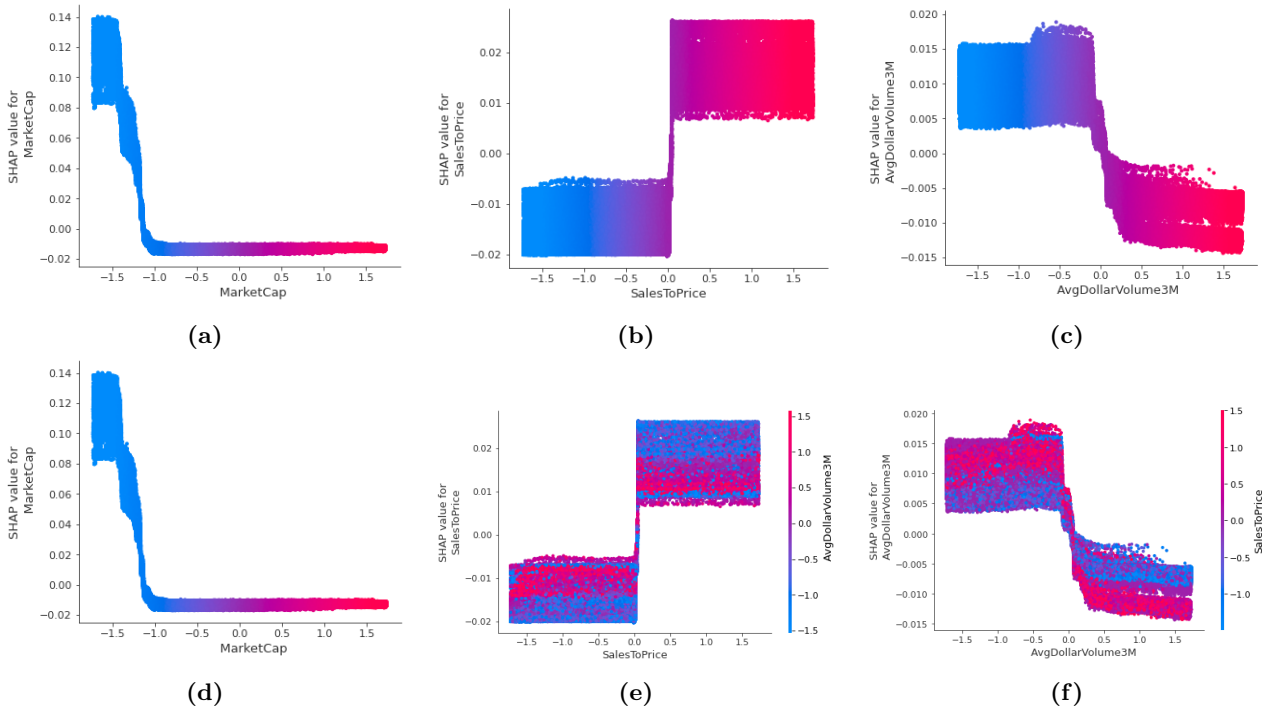


Figure B.39 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the expanding window as of May 2018.

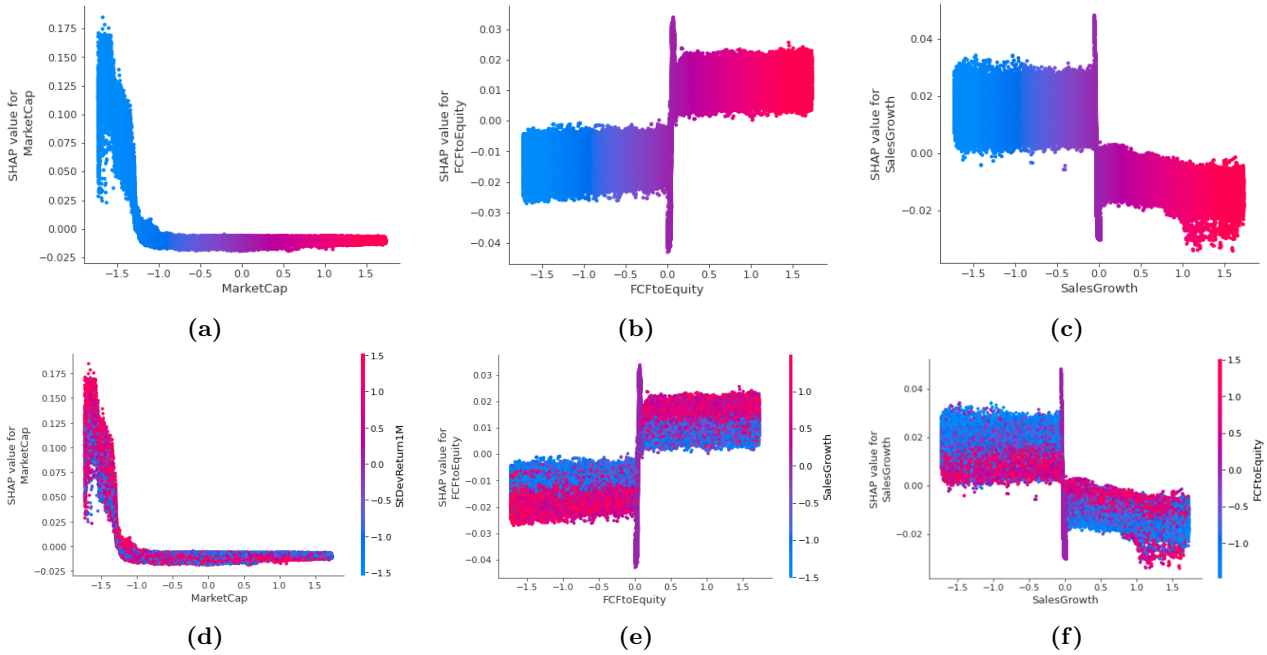
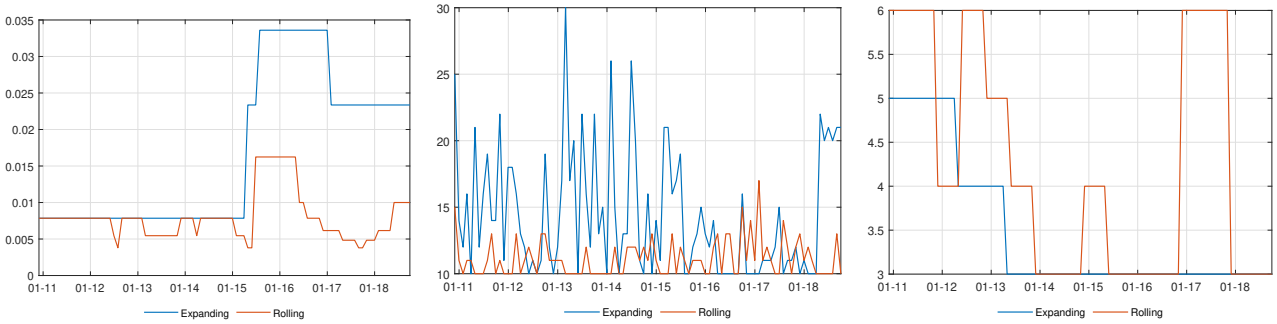


Figure B.40 – SHAP Dependency Plots of the top 3 variables for Random Forest, trained on the rolling window as of May 2018.

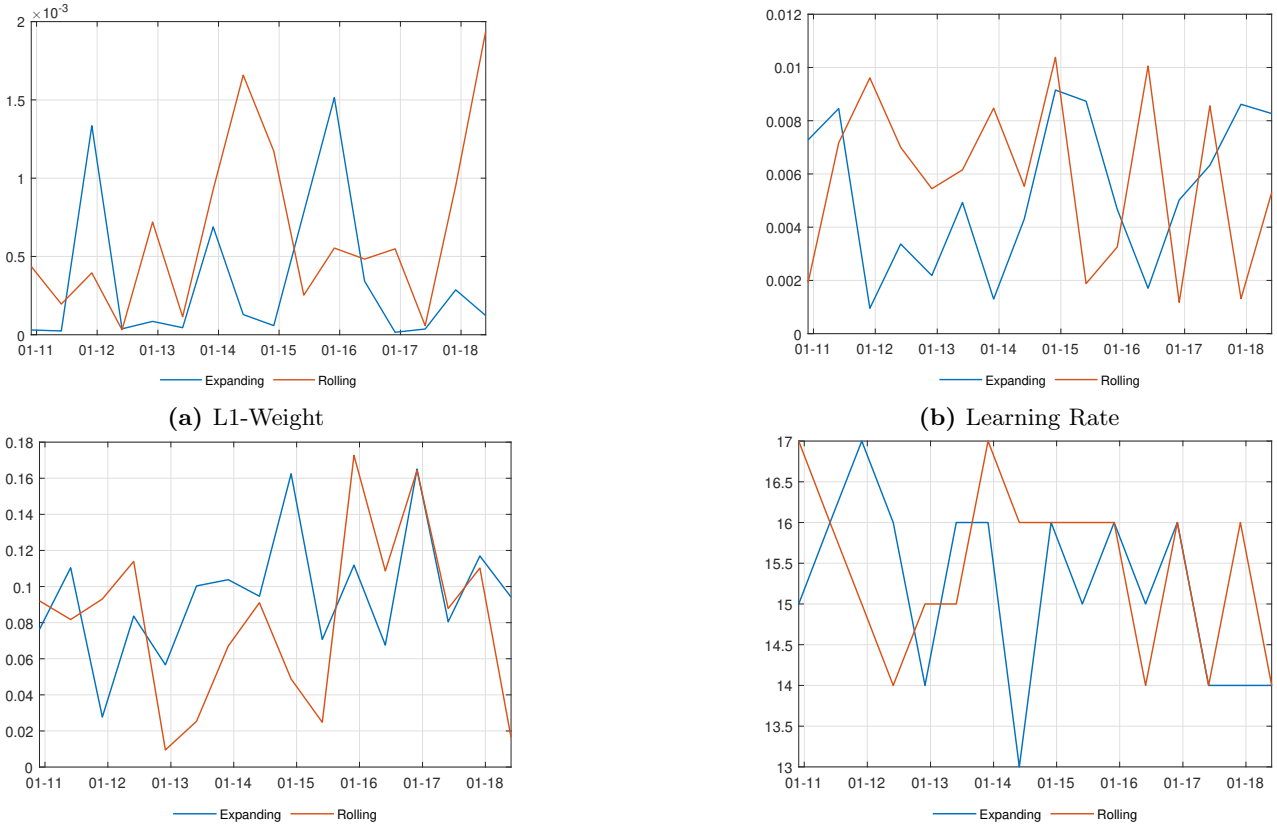


(a) LASSO L1-Weight

(b) VASA Number of features in sub-models

(c) Random Forest Maximum Depth

Figure B.41 – Hyperparameters choices for LASSO, VASA and Random Forest.



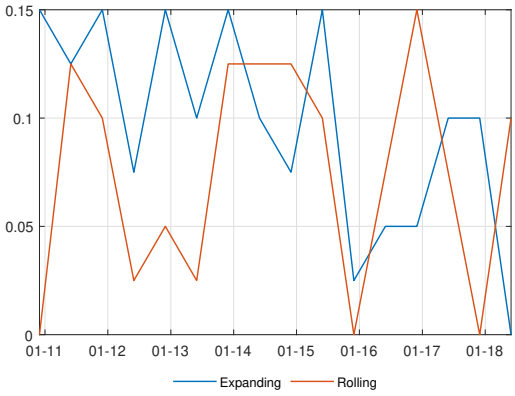
(a) L1-Weight

(b) Learning Rate

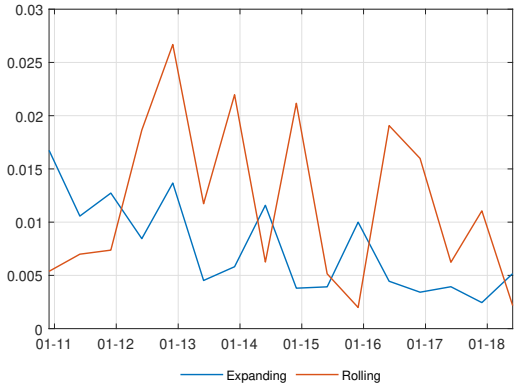
(c) Dropout Rate

(d) Epochs

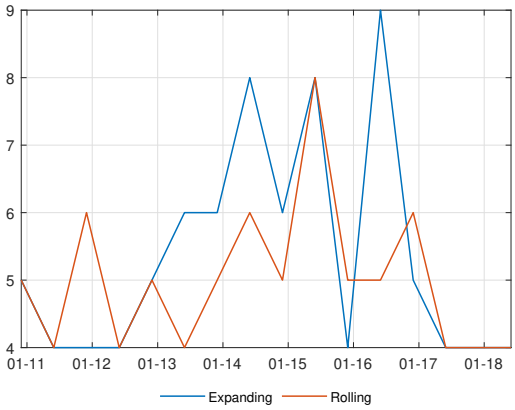
Figure B.42 – Neural Networks Hyperparameters choices.



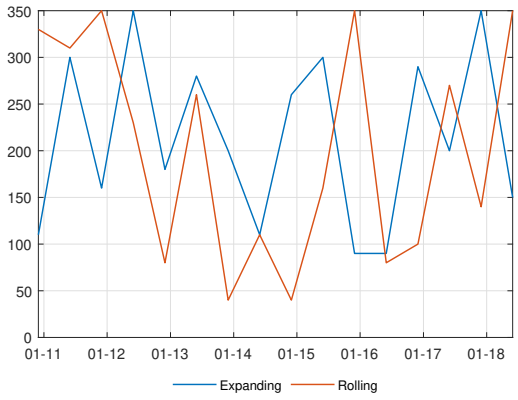
(a) Columns Dropout Rate



(b) Learning Rate



(c) Maximum Depth



(d) Number of trees

Figure B.43 – XGBoost Hyperparameters choices.

Bibliography

J. Bennett, S. Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

J. Bergstra, Y. Bengio, R. Bardenet, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proc. of the 30th International Conference on Machine Learning*, 2013.

T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31:307–327, 1986.

L. Breiman. Classification and regression trees. *Statistics Department, University of California, Berkeley*, Technical Report 486, 1984.

L. Breiman. A decision-theoretic generalization of on-line learning and an application to boosting. *Machine Learning*, 45:5–32, 2001.

L. Breiman, J. Friedman, and C. Stone. Classification and regression trees. *Belmont, Calif.: Wadsworth*, 1984.

L. Chen, M. Pelger, and J. Zhu. Deep learning in asset pricing. *Available at SSRN 3350138*, 2019.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *22nd SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2:303–314, 1989.

G. De Nard, O. Ledoit, and M. Wolf. Factor models for portfolio selection in large dimensions: The good, the better and the ugly. *University of Zurich, Department of Economics, Working Paper No. 290*, 2018.

G. De Nard, S. Hediger, and M. Leippold. Subsampled factor models for asset pricing: The rise of vasa. *Available at SSRN: https://ssrn.com/abstract=3557957*, 2020.

M. L. de Prado. Advances in financial machine learning. *Wiley*, 2018.

F. X. Diebold and R. S. Mariano. Comparing predictive accuracy. *Journal of Business & Economic Statistics*, 13:3:253–263, 1995.

R. F. Engle. Autoregressive conditional heteroskedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50:987–1007, 1982.

R. F. Engle. Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of Business & Economic Statistics*, 20(3): 339–350, 2018.

R. F. Engle, O. Ledoit, and M. Wolf. Large dynamic covariance matrices. *Journal of Business & Economic Statistics*, 37:2:363–375, 2019.

E. F. Fama and K. R. French. Dissecting anomalies. *The Journal of Finance*, 63:1653–1678, 2008.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

S. Gu, B. Kelly, and D. Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33:2223–2273, 2020a.

S. Gu, B. Kelly, and D. Xiu. Autoencoder asset pricing models. *Journal of Econometrics*, 2020b.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.

C. R. Harvey, Y. Liu, and H. Zhu. ... and the cross-section of expected returns. *Review of Financial Studies*, 29:5–68, 2016.

T. Hastie, R. Tibshirani, and J. Freedman. The elements of statistical learning. *Springer, New York, Second Edition*, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Coello C.A.C. (eds) Learning and Intelligent Optimization. LION 2011. Lecture Notes in Computer Science*, 6683, 2011.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, page 448–456, 2015.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*, page arXiv:1412.6980, 2014.

O. Ledoit and M. Wolf. Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, 30(4):110–119, 2004.

O. Ledoit and M. Wolf. Robust performance hypothesis testing with the sharpe ratio. *Journal of Empirical Finance*, 15:5:850–859, 2008.

O. Ledoit and M. Wolf. Analytical nonlinear shrinkage of large-dimensional covariance matrices. *University of Zurich, Department of Economics, Working Paper No. 264, Revised version, Available at SSRN: https://ssrn.com/abstract=3047302*, 2018.

O. Ledoit, M. Wolf, and Z. Zhao. Efficient sorting: A more powerful test for cross-sectional anomalies. *Journal of Financial Econometrics*, 17.4:645–686, 2019.

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

S. M. Lundberg, B. Nair, M. S. Vavilala, M. Horibe, M. J. Eisses, T. Adams, D. E. Liston, D. K.-W. Low, S.-F. Newman, J. Kim, and S.-I. Lee. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2:749–760, 2018.

H. Markowitz. Portfolio selection. *The Journal of Finance*, 7:77–91, 1952.

R. O. Michaud. The markowitz optimization enigma: Is 'optimized' optimal? *Financial Analysts Journal*, 45:31–42, 1989.

D. Politis and J. Romano. Robust performance hypothesis testing with the sharpe ratio. *LePage, R., Billard, L. (Eds.), Exploring the Limits of Bootstrap. John Wiley, New York,*, pages 263–270, 1992.

M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2.28:307–317, 1953.

A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner,

I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

D. Wolff and F. Echterling. Stock picking with machine learning. *Available at SSRN 3607845*, 2020.

H. P. Young. Monotonic solutions of cooperative games. *International Journal of Game Theory*, 14.2:65–72, 1985.

Eidesstattliche Erklärung

Der/Die Verfasser/in erklärt an Eides statt, dass er/sie die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

.....	.....
Ort, Datum	Unterschrift des/der Verfassers/in