Assignment 2

xtj271 and cdv245

4/6/2021

Gu, Kelly and Xiu (2020) describe an asset's excess return as an additive prediction error model:

$$r_{i,t+1} = E_t(r_{i,t+1}) + \epsilon_{i,t+1} \quad \text{where} \quad E_t(r_{i,t+1}) = g(z_{i,t})$$

where $i = 1, ..., N$ is the stock index and $t = 1, ..., T$ is the period specific index. The objective is the predict excess returns with the P-dimensional vector of predictors: $z_{i,t}$.

1) Preparing and exploring the data:

We choose $z_{i,t}$ such that it contains the following financial variables: mvel1, retvol, chmom, maxret, dolvol, turn, mom12m, mom6m, baspread, ill. We also include as macroeconomic variables the treasury bill rate (tbl) and the long-term government bond yield (lty). We further generate interactions between the macroeconomic and the financial variables and include it in the analysis. Lastly, we generated dummies for each industry classifier contained in sic2 (a total of 72 dummy variables). We chose for simplicity not to include it in the report so the line that creates the dummies is commented out. All time-series regressors are already lagged one period with respect to the excess returns that we aim to predict (so we do not need to perform any further lagging).

A quick inspection of the macro variables: One can clearly see that both are downwards sloped over the sample period. The volatility seems to be higher for the long term yield but the the crisis in th early 2000s and in 2007/8 seem to have had a larger effect on the short term yields.

Table 1: Variable summary table

var	N	mean	sd	min	q25	median	q75	max
baspread	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	1.00
chmom	1250871	0.00	0.56	-1.00	-0.47	0.00	0.47	1.00
dolvol	1250871	0.00	0.57	-1.00	-0.49	0.00	0.50	0.99
ill	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	1.00
lty	1250871	0.05	0.01	0.02	0.04	0.05	0.06	0.07
maxret	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	1.00
mktcap	1250871	2885.60	15115.93	0.03	53.84	225.44	1045.13	750709.58
mom12m	1250871	0.00	0.56	-1.00	-0.47	0.00	0.47	1.00
mom1m	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	1.00
mom6m	1250871	0.00	0.57	-1.00	-0.49	0.00	0.49	1.00
mvel1	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	0.99
ret.adj	1250871	0.25	15.15	-66.66	-6.98	0.09	7.15	66.66
retvol	1250871	0.00	0.58	-1.00	-0.50	0.00	0.50	1.00
tbl	1250871	0.03	0.02	0.00	0.00	0.03	0.05	0.06
turn	1250871	0.00	0.57	-1.00	-0.49	0.00	0.50	1.00

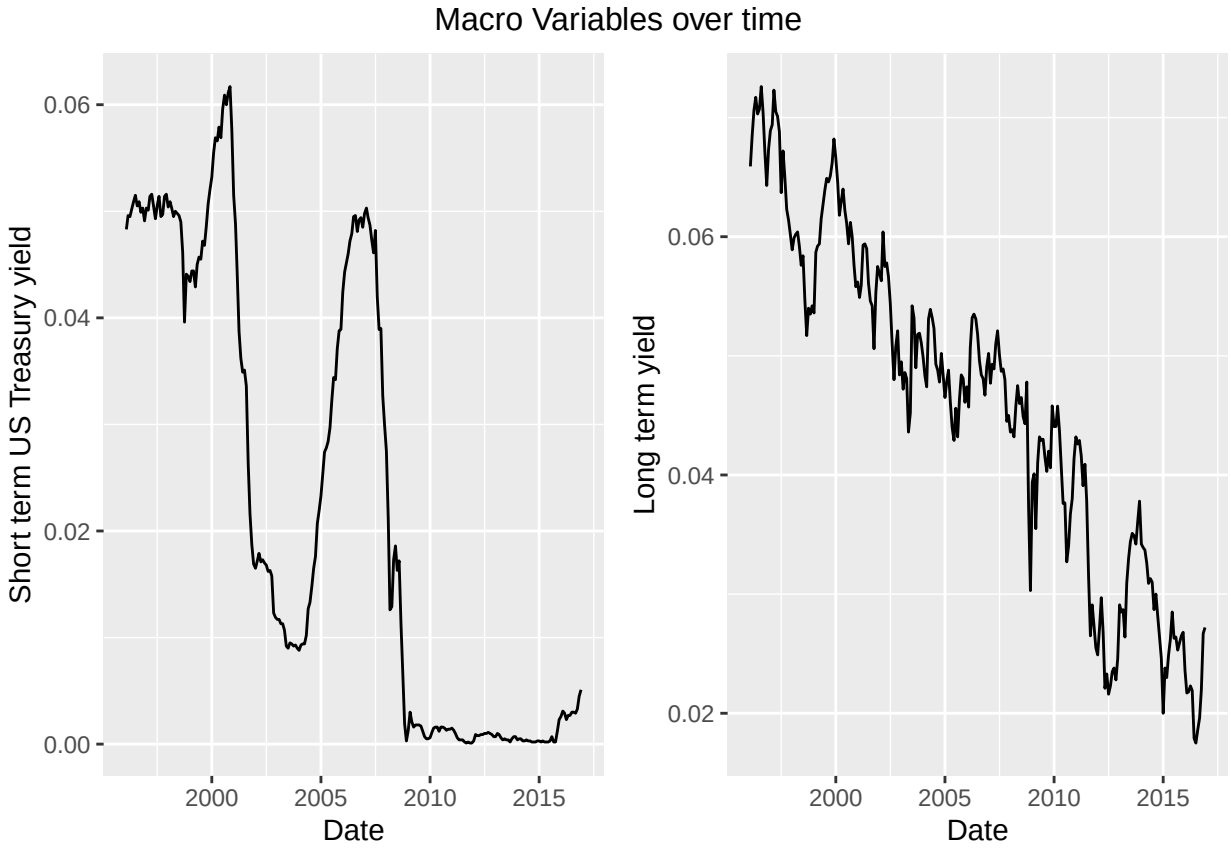


Table 1 reports descriptive statistics for the excess return and the predictors: The sample contained 1.250.871 and it is worth to point out that many variables seem to have a mean at exactly or close to 0. We chose to include market cap in the table necause this variable is used to create a weighted average in question 5 but it will not be used as a predictor in the models.

Lastly, we show summary of industry specific characteristics. We do that by counting the number of stocks in each industry as well as the average market capitalization and average adjusted return. Because of the

Table 2: Industry summary table

sic2	mktcap	ret.adj	N_permnos
73	2546.70	-0.41	1841
60	1757.11	0.95	1554
28	3935.22	-0.07	1195
36	2380.97	0.02	845
38	1529.24	0.36	709
35	2973.75	0.36	630
48	6506.16	-0.30	498
13	2975.08	0.30	390
49	3891.76	0.82	336
63	4293.04	0.90	335
80	997.18	0.08	270
50	857.60	0.21	268
59	2818.03	-0.13	258
87	724.24	0.11	233
99	10657.14	-1.64	233
20	5520.62	0.77	232
37	4055.37	0.61	211
58	2092.74	0.25	202
61	6664.78	-0.24	189
62	4670.76	0.73	181
67	922.63	0.36	168
51	1904.92	0.29	152
79	1432.67	0.24	143
34	1288.91	0.60	135
27	818.33	0.01	134
33	1021.80	0.31	132
39	772.68	-0.32	117
30	1251.56	0.40	110
65	505.22	-0.02	108
26	3965.51	0.39	100

number of categories we chose to sort for the number of firms per industry and display only the 30 industries with the most stocks associated to them. the top industry contains over 1800 stocks while many of the excluded industries contain only 10 or even less stocks.

2) Arbitrage Pricing Theory (Ross, 1976)

Arbitrage Pricing Theory (Ross, 1976) assume that in a competitive and frictionless market, the return of an asset i follows the stochastic process:

$$R_i = a_i + b_i' f + \epsilon_i$$

where b_i is the $(K \times 1)$ vector of loadings, f is the $(K \times 1)$ vector of time-varying factors, a_i is the combination of static factors and ϵ_i is the market noise where $E(\epsilon_i) = 0$.

$$E(R_i) = a_i + b_i' f = g(z_i)$$

where $z_i := [a_i, f']'$ is the $(K + 1 \times 1)$ vector of factors. Here, the function of factors is linear and known, therefore it can be estimated with a regression. If this function were non-linear or unknown we might prefer

using other methods. In the case of time series analysis, we will make use of this framework in order to predict excess returns: $r_i := R_i - R_{rf}$ at time t using an unknown function of factors from the previous period. In particular we were interested in:

$$r_{i,t+1} = E_t(r_{i,t+1}) + \epsilon_{i,t+1} = g(z_{i,t}; \theta_{i,t}) + \epsilon_{i,t+1}$$

where $g(.)$ is a flexible function of predictors (z) and the weights (θ). However the function imposes some rigidities. First, it is independent of i and t, meaning that it retains the same form over time and we can estimate the one period ahead expected return. Furthermore, it only uses information from the stock i at period t but not from any other stocks nor previous periods. These are some shortcomings of the model against models that allow using information from the whole cross-section of returns or longer time horizons.

3) Hyperparameter tunning:

3.1) Hyperparameter selection procedure:

Hyperparameter selection is done to penalize overfitting. Consider the following L2 linear objective function to minimize the distance between prediction as given by the Mean Absolute Prediction Error and the data without penalization:

$$L(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} |r_{i,t+1} - g(z_{i,t}; \theta_{i,t})|$$

It is unwise to train your model with the entire data set by minimizing this criterion function. If you do it, the prediction will overfit the training data set but will perform badly in out of sample predictions, as the model will capture most of the noise that should have been disregarded from the model. Instead, if you want to predict, it is better to penalize this overfitting with a hyperparameter, e.g. (λ) in this case, and validate your model several times for a range of values of the hyperparameter, choosing the value that minimizes the Mean Prediction Error (MPE) in the validation sample, and modifying the criterion function as follows:

$$L(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} |r_{i,t+1} - g(z_{i,t}; \theta_{i,t})| + \phi(\theta; \lambda; ...)$$

3.2) Limitations of hyperparameter tunning:

Tuning lambda to minimize this new objective function might reduce variance, but introduces a bias in predictions. Unlike OLS, now our estimates are biased by design as we are not minimizing anymore the distance between prediction and testing data points. There is a trade-off between bias and variance in prediction.

In the paper, the authors say that adaptive hyperparameter tuning using the data from a validation sample is to basically mimic a reliable out of sample prediction. They do argue that the validation sample fits are not truly an out of sample representation. One potential alternative to tuning parameters dynamically from a validation set could be to select the hyperparameters (e.g. from a grid search) that result in the most favorable information criteria (e.g. the BIC).

Source for the BIC idea: "On Tuning Parameter Selection in Model Selection and Model Averaging: A Monte Carlo Study" by Hui Xiao and Yiguo Sun

3.3) Data splitting:

We split the dataset in three different samples, keeping the period from January 1st 2012 until December 31st 2016 for out-of-sample testing, and splitting the remaining observations before and including December 31st, 2011 into a training and a validation set where each set takes 50% of the observations. We chose to conduct the initial split on date with the *split()* function and use the *rsample:initial_time_split()* for splitting the training date into a training and validation set with probability of 0.5. We also initialize two empty data frames: *MSPE_df* and *hyper_df*. We will use these to collect the minimum, maximum and chosen parameter value for each hyperparameter. In other words, for each machine learning model we will append 3 lines to the *MSPE_df* for each hyperparameter that has to be tuned and one line for hyperparameters that were set by us. The resulting MSPE for each model will also be collected in *MSPE_df* after each model's predictions were calculated.

The aim is to use the training data set to fit a model and then make out-of-sample predictions with the validation data for different hyperparameter values. Then select the hyperparameter that minimized the Mean Absolute Prediction Error in the cross-validation step. The final prediction is done in the testing sample and the resulting prediction error is the one we use to compare across different models.

4) Machine learning models:

A few comments to guide you through this section: We will run through 4 machine learning models: The Ridge, the Lasso, a Tree and a Forest. We also do include the code for our Neural Network (NN) but we do not touch on it in the report itself. For each model, we describe how it works and what hyperparameters we chose to discuss. The logic by which hyperparameters are tuned as well as the targeted objective function will also be discussed. The code for each model contain a block about fitting and tuning the actual model, a few lines of code that collect the hyperparameters in a dataframe after each model as well a block of code that performs the portfolio sort for each model after it was fitted. Please note that whenever we name a hyperparameter in this section it will be reported in the Table called "Hyperparameter-Tuning Overview" at the very end.

4.1) Ridge:

Consider the objective criterion function for a elastic net:

$$L(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}|r_{i,t+1} - g(z_{i,t};\theta_{i,t})| + \alpha\lambda\sum_{j=1}^{P}|\theta_j| + (1-\alpha)\frac{1}{2}\lambda\sum_{j=1}^{P}\theta_j^2$$

The R package glmnet allows us to implement combinations of this model to train our data and make predictions. By setting $\alpha = 0$ in the elastic net criterion, we obtain the Ridge criterion:

$$L(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}|r_{i,t+1} - g(z_{i,t};\theta_{i,t})| + \frac{1}{2}\lambda\sum_{j=1}^{P}\theta_j^2$$

The parameter of interest for the ridge model is the penalization parameter lambda. In order to tune this hyperparameter, we use the training data to fit the ridge model for a range of λ and then select the one that minimizes the criterion, as given by the Mean Absolute Prediction error. As can be seen in the ridge criterion function, complexity is penalized with an L2 norm (more precisely as the sum of squared elements in the theta vector). That selected lambda is then used to fit the model. Our fitted "best-model" is then used to make out of sample predictions with the testing data. The corresponding boundaries for lambda as well as the chosen lambda are appended to hyper_df. The MPE is collected in MPE_df and plotted as a black horizontal line in the graph below. The plot visualizes the relationship between complexity of

Table 3: Current MPE comparison table

Model	MPE value
RIDGE	12.21

the model (in terms of shrunken coefficients) and the performance. The lower the penalization parameter lambda the less restricted the model is (thus more complex) the model is. Two things should be noted here: Firstly, the validation MPE is strictly lower the the error in the training set. Additionally, the prediction on the testing data is below the training and validation errors. As discussed in class, one would expect to see the highest MPE for the out of sample testing data as this data is new and could not be used to fit the model. The observed relationship could be due to the time series character of the data. The testing set contains only data from 2012 onward, while both validation and training contain observations from e.g. the financial crisis or even the older periods of high volatility. If the testing data happens to represent a period of uncharacteristically low volatility, the model might actually make better out-of-sample prediction not because the out of sample fit is so perfect but because of structural differences in testing and training data. The second point that should be noted is that the MSPE seems to be surprisingly invariant to the choice of lambda. In the current state, we consider a large range of lambda with the highest being 1*e^3. Accordingly, the chosen lambda is 411.

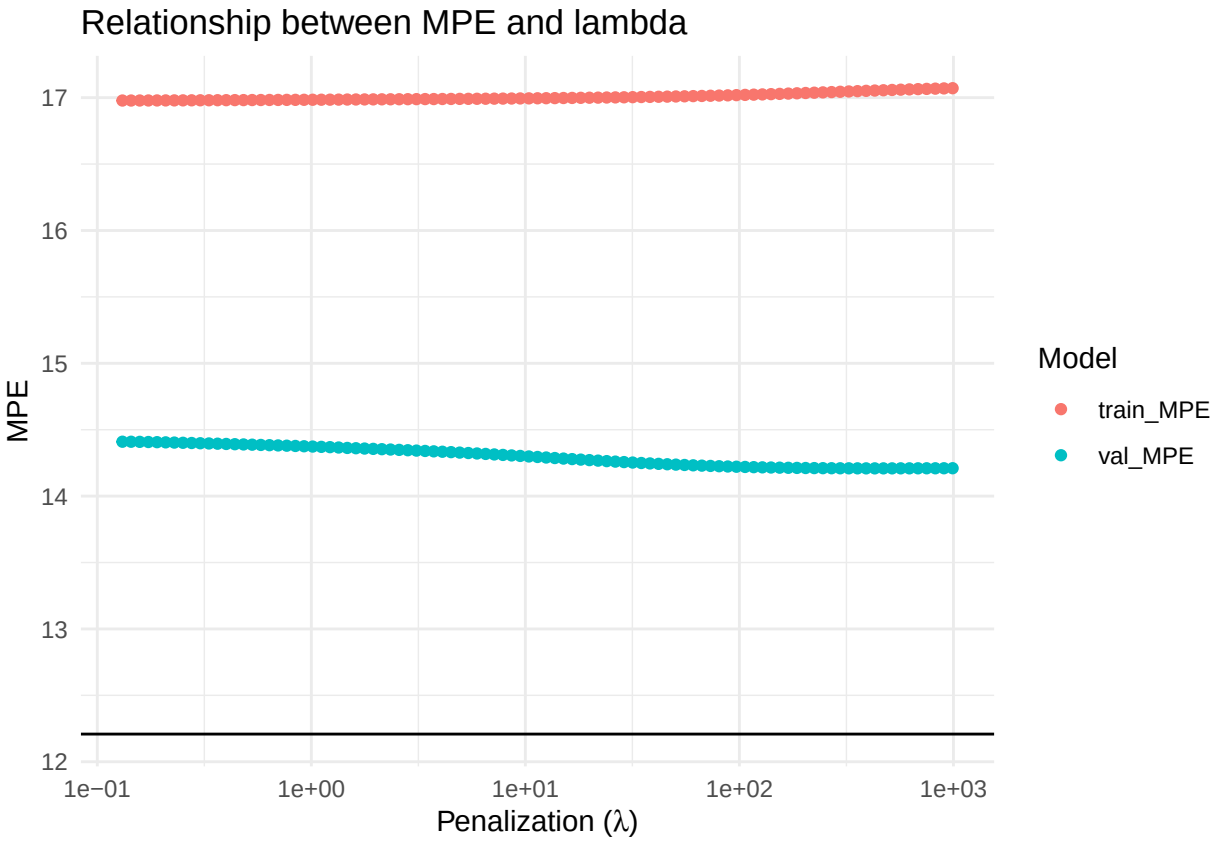
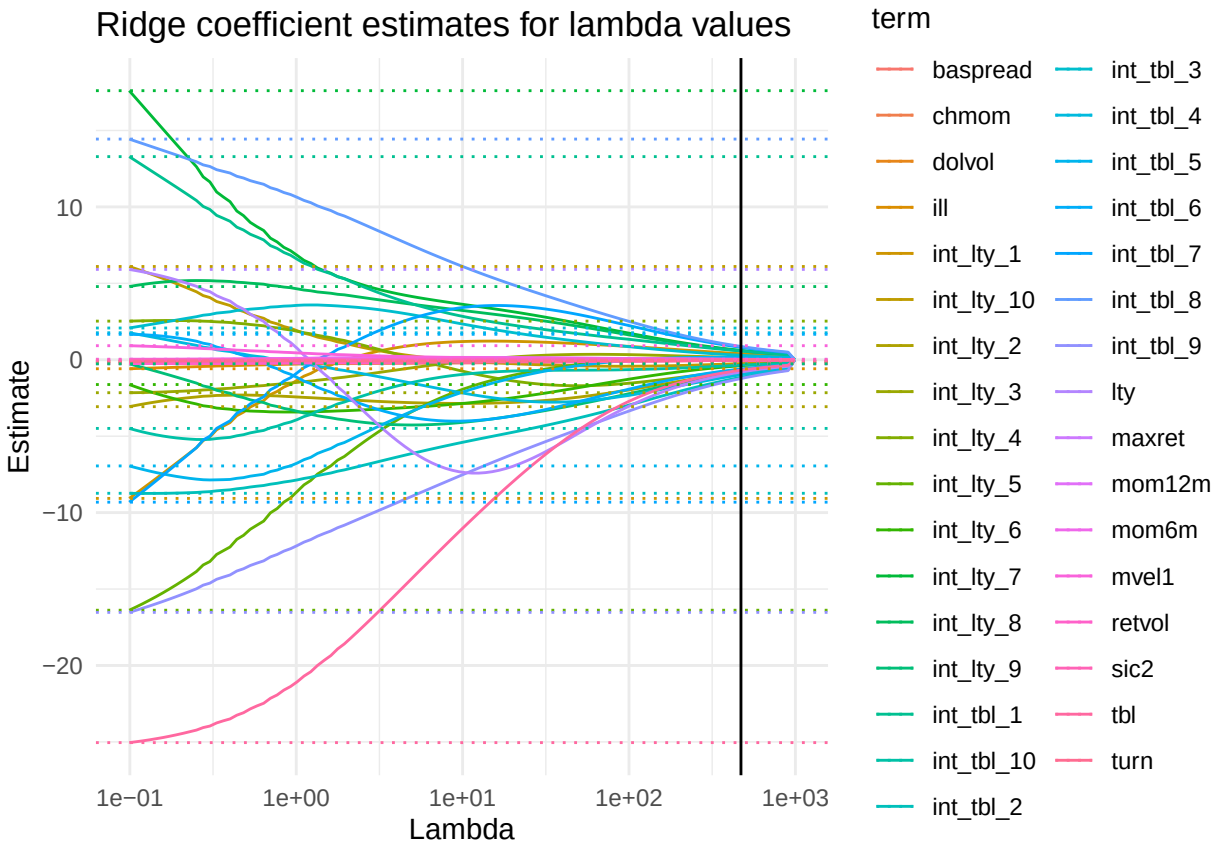


Table 3 reports the final prediction error in the testing sample for the ridge model. This table will be updated each time we fit a new model and thus will serve as a comparison of these coefficients. The ridge shrinks the value of the coefficient estimates towards zero for larger values of λ, as shown below. Note that for a $\lambda = 0$, the ridge coefficients are equivalent to the Least Squares estimates. The vertical line is the chosen λ^* that minimized the out of sample MPE:



4.2) Lasso:

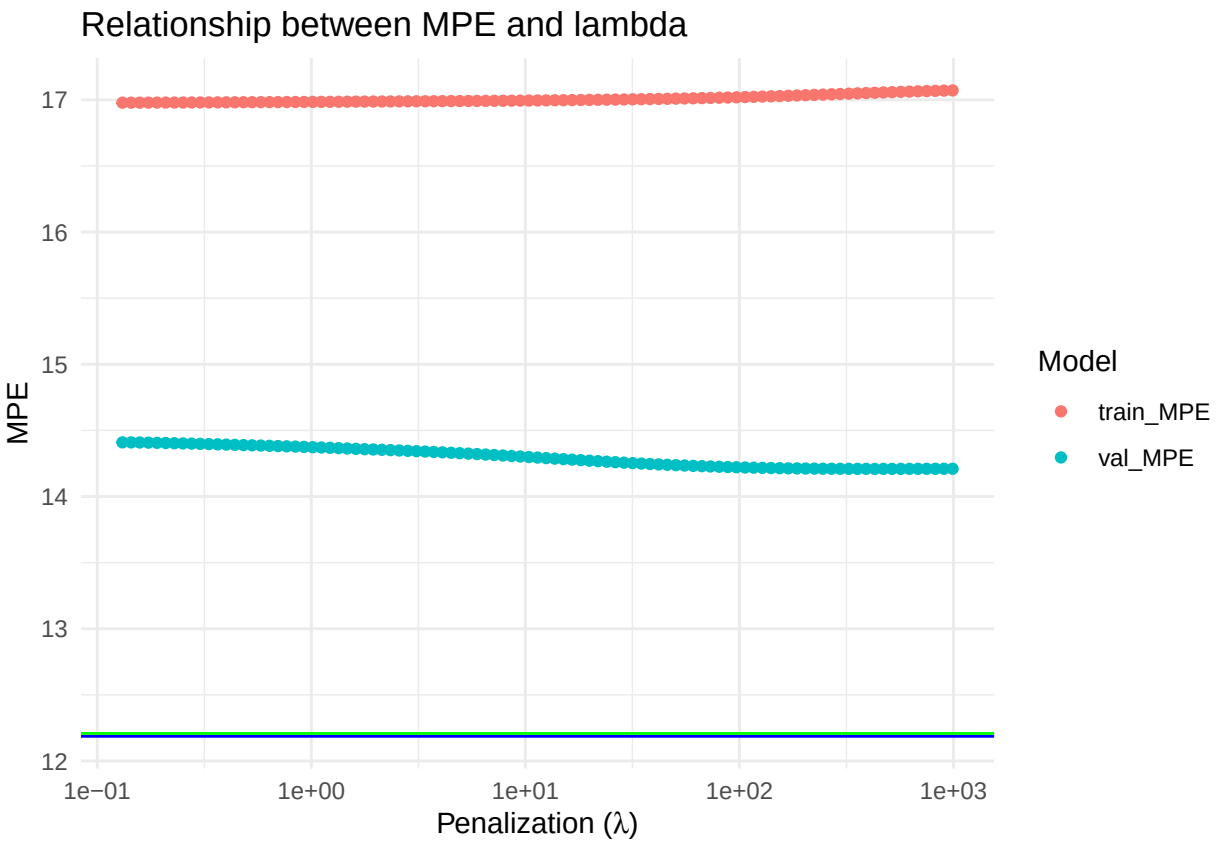
By setting $\alpha = 1$ in the elastic net criterion, we obtain the Lasso criterion:

$$L(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} |r_{i,t+1} - g(z_{i,t}; \theta_{i,t})| + \lambda \sum_{j=1}^{P} |\theta_j|$$

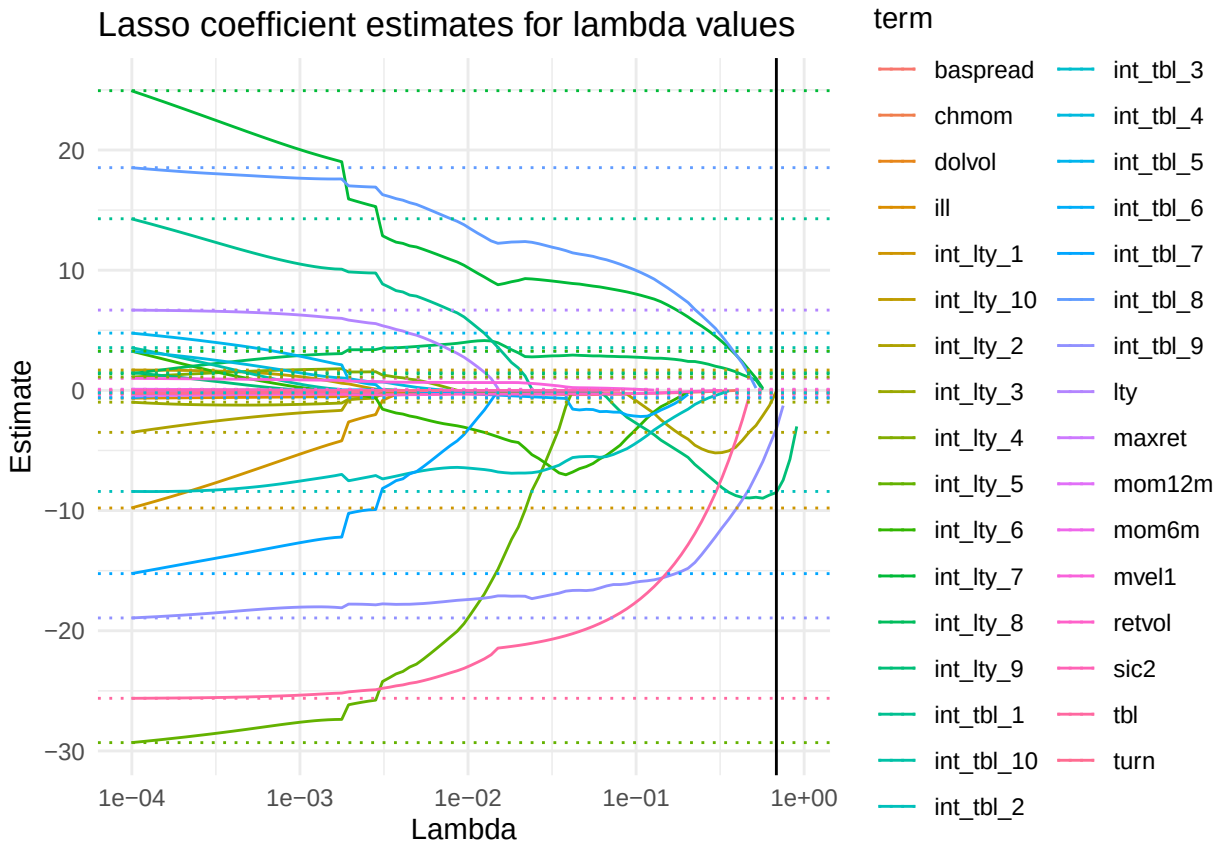
Equivalently to the Ridge, the parameter to be tuned in the lasso is also lambda. This means that we can basically recreate the analysis from before after changing the alpha in the glmnet call to 1 (indicating a fully L1 penalization): We estimate the out of sample MPE after training and cross-validating the model, and we plot both MPE together in the same graph (Lasso as the blue line and Ridge as the green line). We can observe that they are almost identical:

Table 4: Current MPE comparison table

Model	MPE value
RIDGE	12.21
LASSO	12.19



As said before, Table 4 compares the prediction errors of all currently fitted models. It can be seen that ridge and lasso perform very similar, with the lasso being slightly better. Note that in contrast to the ridge, the lasso sets the coefficients of less important predictors to zero for higher levels of complexity as given by λ and thus perform variable selection. This is due to the geometrical properties of the L1 penalization used in the lasso (penalize complexity in a linear way rather than quadratic as seen in the ridge criterion function). As can be seen in the pot is that the lambda chosen in the Lasso model is very high (thus meaning that a lot of variables are excluded). The range for lambda is between 0 and 1000 and the final choice is 411 as can be seen in the hyperparameter summary table 7 below.



4.3) Regression Tree:

How does a tree work: Gu, Kelly and Xiu point out that trees are fully nonparametric models unlike the two previously considered models. A tree i supposed to make smart splits of the data which will eventually split the space of predictors into rectangular sectors $R_1, R_2, ...R_J$. For a variable x in a specific sector (called partition in slides) R_j of the predictor space, the function f(x) is estimated by taking the average of all y training data points for which variable x is in the current partition. To decide what splits to conduct, select a variable j and a threshold value s that together define a split and thus create two new partitions. One region would contain values where variable j is smaller than value s, the other region contains all larger or equal to s. According to slide 39 in Lecture 2, variable j and value s are selected to minimize the residual sum of squares (RSS):

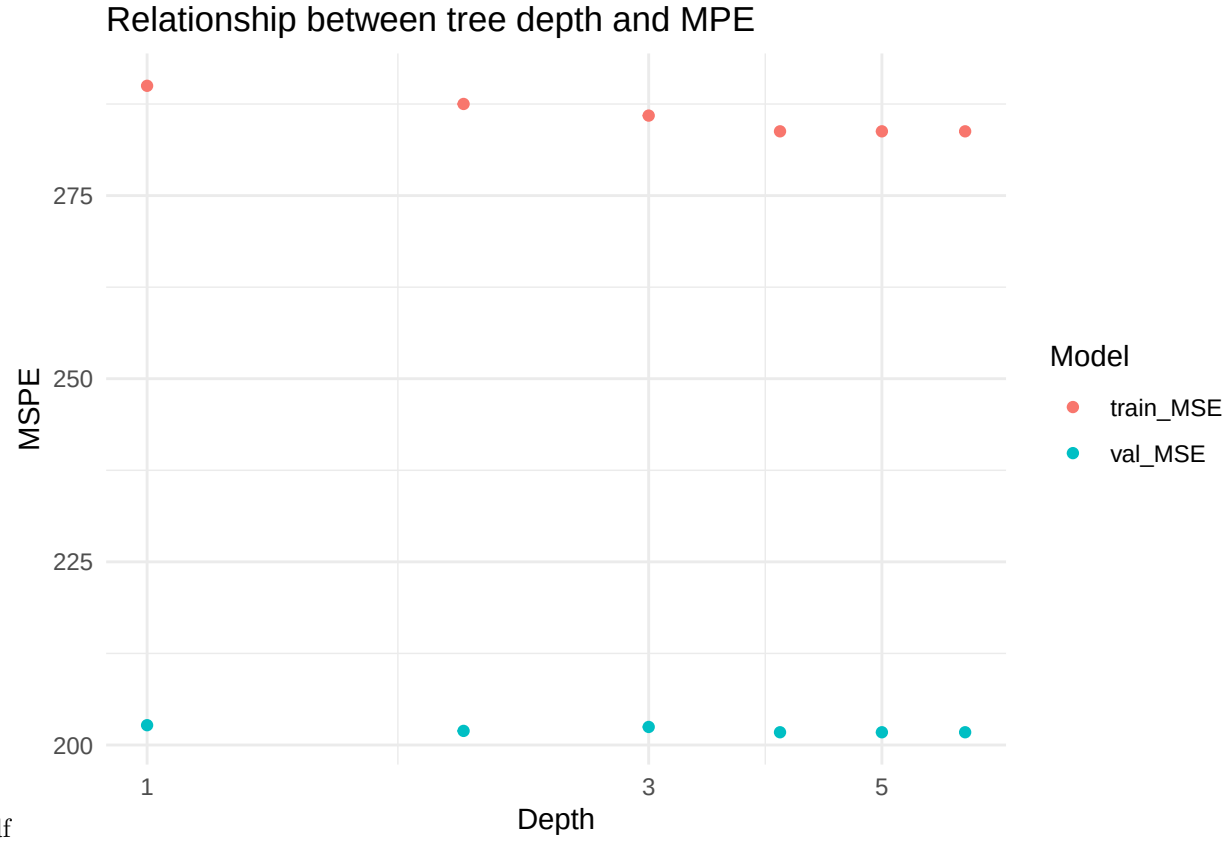
$$RSS = \sum_{i:x_i \in R_1(j,s)} (y_i - \tilde{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \tilde{y}_{R_2})^2$$

When plotting a tree, it becomes easy to see how a specific predictor is used to split the data for a certain threshold of this predictor into two new branches. The logic of such a split is considered "smart" because it follows some rules on e.g. how much each split has to improve the actually prediction in order for it to be attempted. Thus, by selecting the most relevant predictors to split by and the specific threshold the prediction error is minimized. A final prediction is then done by averaging over the outcome of each of these sectors.

For modeling a regression tree, a different set of hyperparameters than in the Lasso or Ridge has to be considered. For this exercise we focus on tuning the max_depth parameter which is a control option for the *rpart* library that is used to grow the tree. Tuning is done by looping over a range of tree depths to calculate the validation MSPE for each iteration. The results can be used to select the depth that gives the smallest MSPE. A final tree is then fitted using that depth control option and to make a prediction on the training data set. Aside from the max_depth control option we also changed two other controls from the

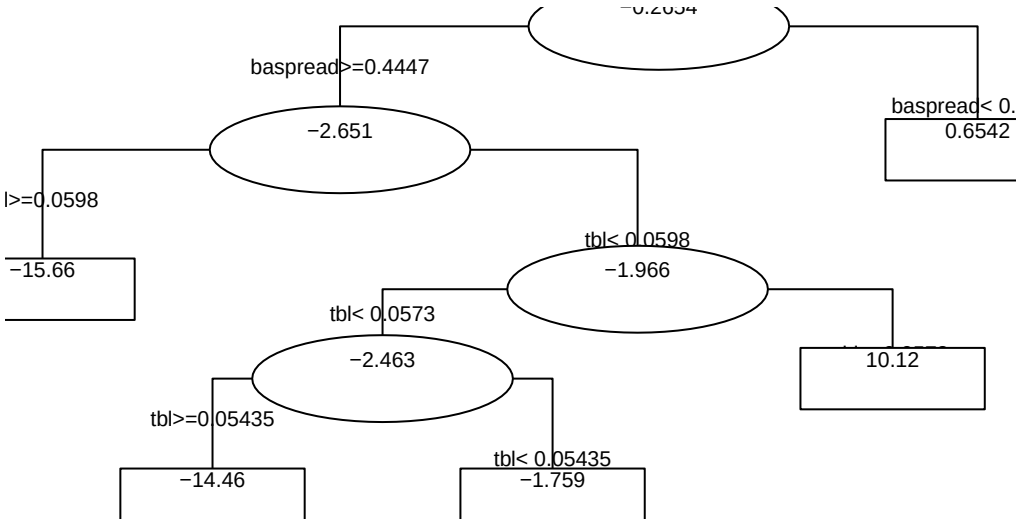
rpart library, namely *minsplit* and *cp*. These changes were not done using a grid search but were manually set because our tree would not make a single split with the default settings of $minsplit = 20$ and $cp = 0.01$. To encourage our tree to grow, we had to change the minimum number of observations in a node for a split to be attempted (*minsplit*) down to 10 and the complexity parameter *cp* down to 0.005. That parameter defines by how much the overall lack of fit has to be improved as a minimum for a split to occur. Again, an overview of all relevant parameter can be found in Table 7 below.

Similarly to before, the sensitivity of the prediction error with respect to complexity is visualized in the following graph. Here, depth is plotted as the x-axis instead of lambda. It can be seen how the training MPE decreases with a growing number of branches but is unchanged after the 4th branch has been grown. The validation MPE is also declining and then constant after the 4th branch but the magnitude of these changes is neglectable. Nevertheless, a maximum depth of 4 is the result of our hyperparameter tuning for the tree.



-1.pdf

Next, we visualize the tree. It can be seen that the variables chosen to perform splits are *baspread* and interestingly the macroeconomic variable *tbl* multiple times. This is however not a problem per se: If we look at the left branch (*baspread >= 0.447*), the next split is on *tbl < 0.0598* and then again on *tbl < 0.0573*. This implies that for observations with *baspread >=0.447* & *tbl* between 0.0573 and 0.0598 the outcome will be different from those with *baspread >=0.447* but *tbl < 0.0573*. -1.pdf



Accordingly, the values for *minsplit*, *cd* as well as the boundaries for max_depth are appended to the hyperparameter data frame. The resulting MPE for the prediction using the fitted tree is plotted as a purple line in the same graph as the Lasso and the Ridge from before. The prediction overlap to a point where it becomes hard to compare the results graphically, so we also report the table of MPEs for all models that were fitted so far. -1.pdf

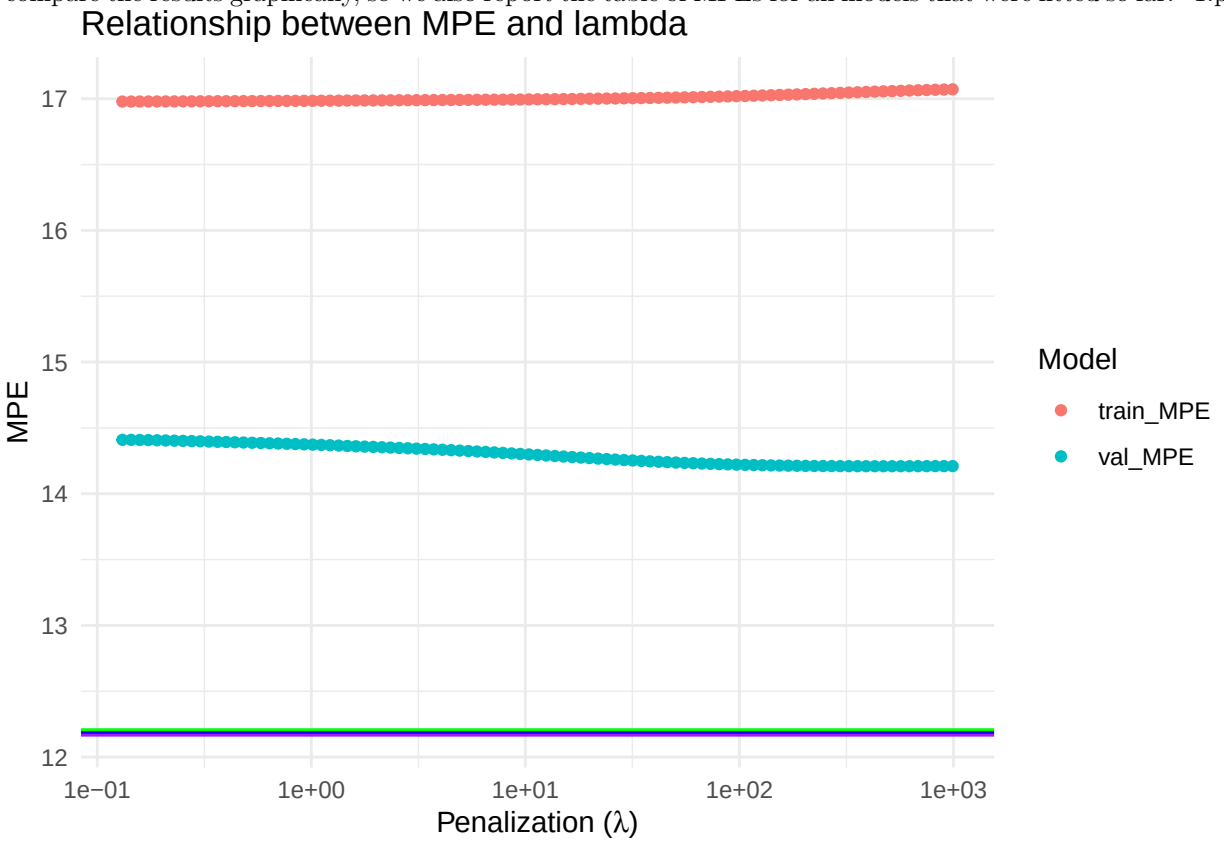


Table 5 shows how similar the prediction error for the regression tree (with respect to the training sample) is compared to Lasso and Ridge. The MPE is the lowest for the tree.

Table 5: Current MPE comparison table

Model	MPE value
RIDGE	12.21
LASSO	12.19
TREE	12.17

4.4) Random Forest:

How does a forest work: According to Gu, Kelly and Xiu, regression trees are the prediction method most likely to overfitting. Therefore combining the forecast from multiple trees could help to reduce overfitting. A random forest is following the algorithm provided below. For a specified number of trees in the forest, a random sample from the data is taken to grow each tree. After fitting N = ntree numbers of trees, the prediction is averaged over all trees. This averaging should make the prediction more robust compared to the prediction of a single tree.

1. Given training data set
2. Select number of trees to build (ntrees)
3. for i = 1 to ntrees do
4. Generate a bootstrap sample of the original data
5. Grow a regression tree to the bootstrapped data
6. for each split do
7. | Select m variables at random from all p variables
8. | Pick the best variable/split-point among the m
9. | Split the node into two child nodes
10. end
11. Use typical tree model stopping criteria to determine when a tree is complete (but do not prune)
12. end (Source: https://uc-r.github.io/random_forests)

Each time a bootstrap sample is selected for a tree i (the authors call that procedure bagging) the rest of the data serves a out-of-bag sample (like a quasi validation data set). This can be used to calculate the out-of-bag prediction error (OOB) The objective function for each individual tree is the same as in thee section above.

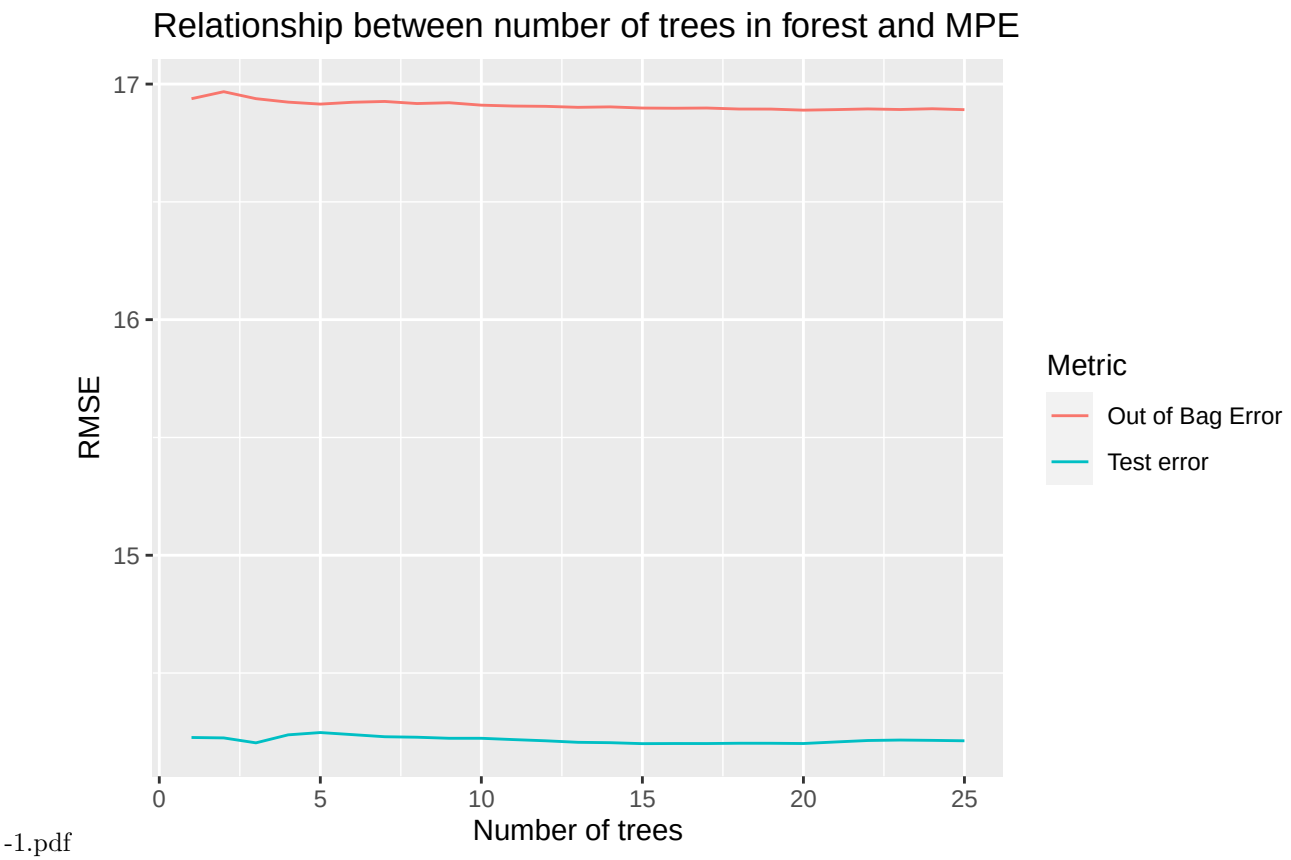
Among the hyperparameters of interest in the random forest the following three were selected for this assignment: The number of trees the forest should grow, *mtry* (according to documentation the number of randomly selected predictors as potential candidates for each split) and *nodesize* (which defines the minimum number of terminal nodes). For hypertuning, the number of trees was fixed to 25 (which is a fairly low number but was set to reduce computation time). A grid was created containing all combinations of *mtry* and *node_size*. Note that we use a package called *ranger* which was suggested by *https://uc-r.github.io/random_forests* to perform faster than the *randomForest*. Ranger is a C++ implementation of Breiman's algorithm also mentioned in Gu, Kelly and Xiu. We then perform a grid search and loop over each combination of hyperparameters to calculate the out-of-bag error. As before, we then train a tree with the hyperparameter combination that gave the best overall results. We decided to also plot the sensitivity of the OOB-error wrt. the number of trees that are grown in the forest. Recall that the maximum number of trees was set to 25. The plot shows that the prediction error seems to be insensitive to the number of trees over which the final prediction is averaged. As mentioned before, plotting the final prediction in the same graph as the Lasso, Ridge and Tree prediction would lead to messy results, so only the table displaying the currently collected numerical predictions is shown to visualize the comparison between all models.

Table 6: Current MPE comparison table

Model	MPE value
RIDGE	12.21
LASSO	12.19
TREE	12.17
FOREST	12.17

```
## [1] 4
```

```
## Growing trees.. Progress: 20%. Estimated remaining time: 3 minutes, 20 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 27 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 26 seconds.
## Growing trees.. Progress: 20%. Estimated remaining time: 3 minutes, 52 seconds.
## Growing trees.. Progress: 36%. Estimated remaining time: 2 minutes, 47 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 58 seconds.
## Growing trees.. Progress: 68%. Estimated remaining time: 1 minute, 14 seconds.
## Growing trees.. Progress: 96%. Estimated remaining time: 8 seconds.
## Growing trees.. Progress: 20%. Estimated remaining time: 2 minutes, 56 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 28 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 27 seconds.
## Growing trees.. Progress: 4%. Estimated remaining time: 12 minutes, 24 seconds.
## Growing trees.. Progress: 24%. Estimated remaining time: 3 minutes, 16 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 46 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 36 seconds.
## Growing trees.. Progress: 100%. Estimated remaining time: 0 seconds.
```



-1.pdf

Table 7: Hyperparameter-Tuning Overview

Type	LASSO	RIDGE	TREE	FOREST	NN
Lambda min		0.12			
Lambda max		991.32			
Lambda choice		470.96			
Lambda min	0				
Lambda max	0.99				
Lambda choice	0.68				
Rpart control minsplit			10		
Rpart control cp			0.005		
Depth min			1		
Depth max			6		
Depth choice			4		
Numer of trees				25	
mtry min				5	
mtry max				7	
mtry choice				7	
nodes min				5	
nodes max				7	
nodes choice				7	

Table 8: Comparison of model performance on buy 10 sell 1 portfolio

	Ridge101	Lasso101	Tree101	Forest101
estimate	0.70	1.50	2.4100e+00	1.87
nw_tstat	385.15	575.71	2.6412e+16	239.92

Interestingly, the rounded prediction error is the same for the tree and the forest.

4.5) Hyperparameter-tuning Summary:

Finally we report all the collected hyperparameters in the table below: The table contain one column for each model. For each parameter that was tuned, 3 lines were appended, first the minimum value of the potential range, then the maximum value and third the chosen value - which gave the best prediction. In case a parameter was set manually by the us, only one line stating that parameter was appended. As said before the range of most parameters was chosen to be relatively narrow to reduce computation time but could easily be scaled up to larger grids. Also the table contains a column for the NN, which is included in the code bit not in the report - hence that column will stay empty for now.

5)

After collecting the average return predictions of our 4 models based on a portfolio that buys the *permnos* in quantile 10 and sells quantile 1 (quantile with respect to the predicted next month returns y_hat) the following table summarizes the results:

As a side note, when looking at the quantiles for *y_hat_tree*, we realized that because of the number of splits we allowed the tree to make, the predictions containing only 3 unique values. So the code could not split this variable with three distinct values into 10 different quantiles. We will think about increasing the split threshold for the final exam but for now the portfolio that is supposed to buy the top quantile (usually 10)

will buy quantile 3 for the tree (as quantile 3 - 7 are all the same and q3 is the highest possible prediction a stock could be assigned by the tree). Our Forest also only generated 7 distinct predictions so we used the same fix for the Forest as for the tree before, but still call the variables "Tree101" and "Forest101" in the table because it represents the portfolio that buys the stocks with the highest predicted return. As said this is something we will look into again after the feedback. Note we did not yet implement a comparison to the CAPM like in the exercise.