FACULTY OF ECONOMICS AND
BUSINESS ADMINISTRATION

InClass Kaggle competition:
Product return rate prediction

Technical report

Course: Machine Learning

Teacher: Prof. Dr. Dries Benoit
Lukas De Kerpel (TA)

Students: Team 05

ABEDIKOUPAEI, MELIKA	02202978	melika.abedikoupaei@ugent.be
DANG, LE	02204518	le.dang@ugent.be
DENG, LISHUANG	01905012	lishuang.deng@ugent.be
OWUOR, VICTOR ODHIAMBO	02001807	victorodhiambo.owuor@ugent.be
SI, JUNTIAN	02206130	juntian.si@ugent.be

Academic year: 2023–2024

GHENT
UNIVERSITY

Contents

1 Introduction

Product return is a tricky thing in real life. It not only adding extra cost for the retailers but also trigger environmental issue. In this assignment, we built up 12 different parameter models to predict product yearly return rates by using four weeks of sales data.

In the data cleaning part, we try to impute all the missing data by the mean or the mode method. For example, we impute the “Channel” column in the sales_yr1 and sales_yr2 tables by the mode of the number of channels in this brand or product.

In the feature engineering part, we derive 24 features in total. Some features are very important for later modeling such as the number of size types, online sales percentage, and so on.

In some models, we combined more than 2 models to work together, for example, we used the random forest model to find out 8 potential important variables and then used the cross-validation(cv) method to calculate the best 5 variables combination for the final GAM model.

In the hyperparameter tuning part, we combined 4 methods. Random search, Grid search, Cross-validation, and Manual. First, we randomly set up a grid of hyperparameter values. Secondly, the best parameters are found by using cross validation method. However, for some models, the best parameters don’t perform significantly. Thus, we also tuned these parameters manually.

In the end, comparing all models with validation mean absolute error (MAE), we find that the Random Forest model is the best to determine the product’s yearly return rate.

2 Data Preparation

2.1 Data splitting

- The sales data from the year 2021 in the sales_yr1 table are used for model training (The big training set).
- The first 4 weeks sales data from the year 2022 in the sales_yr2 table are used for the product yearly return rate prediction.
- In the big training set, randomly selected 70% of data for the model training, and the remaining data for the model validation.

2.2 Data cleaning

2.2.1 Missing values

In the ’Missing Values’, Here is a summary of what we did:

In the product table:

- NA values in the “price” column are imputed by the mean of price of their respective category. Negative price is regard as NA values.
- NA values in the ’brand’ column are filled with the brand name from the same product ID (pid) which has brand value. NA values cannot be imputed, we filled with ’others’.
- NA values in the ’sku_size’ column, we didn’t impute them. Instead, we changed the size type into numeric and non-numeric size type. (see further in the feature emerging part)

In the sales_yr1 and sales_yr2 tables:

Calculated the mode for ’channel’ within each product and filled NA values with the mode value.

2.2.2 Outliers

- Used the Tukey method to detect outliers in the net_sales_amount in sales_yr1 and sale_yr2 tables.
- Unnormal return_rate values which is larger than 1 in the training set, we replace with 1.

2.2.3 String manipulation

The ’String Manipulation’ section involved cleaning and transforming text-based columns. Here’s a summary:

- Remove Currency Symbol“\” from ’Price’.
- removing redundant ’brand’ and ’channel’ symbol such as CH// and brand::
- removing redundant category names from ’subcategory’ column such as “Formal Attire-Petite” in the subcategory, remove “Formal Attire-”

2.3 Feature engineering

Based on the original dataset, we created 13 new features (The red names in the Feature Name column below) Here is a list for our features in the train and test set:

Feature Name	**Description**
pid	Product ID, a unique identifier assigned to each product.
brand	The brand or manufacturer of the product.
season	The season to which the product belongs, indicating the time of the year it is associated with (summer, winter).
category	The general category to which the product belongs (e.g., clothing, electronics, accessories).

Continued on next page

Table Continued from previous page

Feature Name	**Description**
subcategory	A more specific subcategory within the general category, providing additional detail about the product.
subsubcategory	A further refinement of the product's categorization.
numeric_size	The numeric representation of the product size type.is it based on number then this col would be 1 or based on S,M,L then it would be 0
count_size	The variety of sizes available for the product.
price	The selling price of the product.
total_sales	Total number of units sold for the product.
total_return	Total number of units returned for the product.
offline_sales	Number of units sold through offline channels.
offline_return	Number of units returned from offline sales.
online_sales	Number of units sold through online channels.
online_return	Number of units returned from online sales.
online_percentage	Percentage of total sales that occurred online. Calculated as (online_sales / total_sales) * 100.
unique_channel_count	Count of unique channels through which the product was sold.
SR	Indicating the relationship between sales and returns. From 0 to 5 represents different relationships. 0 means no sales and no returned 1 means no sales but have returned 2 means the number of sales is less than the return 3 means the number of sales is equal to the return 4 means the number of sales is more than the return 5 only sales and no return
SO	Stockout, product completely sale in store=0 or completely online sales =1 or both channels =2.
month_number	The numerical representation of the month in which the sales and returns calculated.
introduction_time	The percentage of the time that first appear during the time period.
introduction_season	The season that the product is introduced, 1 for January to March 2 for April to June 3 for July to September 4 for October to December
first_return	The percentage of the first return time occurs during the time period.
return_rate	Return_ rate = (-total_return / total_sales) for the first 4 weeks of the year

Continued on next page

Table Continued from previous page

Feature Name	Description
yearly_return_rate	The return rate during the year 2021

3 Model Training

3.1 Linear baselines

3.1.1 Linear regression

Specification

- Calculate the correlation matrix.
- Using the forward selection to find a level of the features and select the best features.

$$\begin{gathered} yearly\_return\_rate = \beta_0 + \beta_1 \times numeric\_size + \beta_2 \times count\_size + \beta_3 \times price \\ +\beta_4 \times total\_sold + \beta_5 \times total\_returned + \beta_6 \times online\_percentage \\ +\beta_7 \times unique\_channel\_count + \beta_8 \times SR + \beta_9 \times SO + \beta_{10} \times first\_return \\ +\beta_{11} \times introduction\_time + \beta_{12} \times season\_Winter + \beta_{13} \times subcategory\_Petite \end{gathered}$$

- Fitting the best features with linear regression model. In this step, we found out that two variables (total_returned and SO) are not significantly. Thus, we dropped these two variables and fit the model again.
- VIF Analysis. All VIF values are lower than 5. There is no collinearly among the predictors.

MAE test in the validation set as follows

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
Linear regression	0.1105	0.0857	0.1079	0.1052	4.63E-05	0.8572

From the correlation matrix, we found that All correlation values are lower than 0.35. In the final model, the Adjusted R-squared is 0.4103. That means the linear model doesn't perform well. The linear relationship is not so obvious.

3.1.2 Ridge regression

Specification

- We fit the model with almost all the variables (20 /24), which is more than linear regression. Features used for Ridge regression: [“return_rate” , “first_return”, “season”, “introduction_season”, “introduction_time”, “SO”, “SR”, “unique_channel_count”, “online_percentage”, “ total_returned”, “total_sales”,”total return”, ”offline_return”, ”offline_sales”,”online_sales”, ”online_return”,“price”, “count_size”, “numeric_size”,”subcategory”]

Hyperparameter tuning

- Set a grid of lambda 10^{10} to 10^{-3}
- Find the best lambda by using 10 fold Cross-Validation method.

Name	Range	Selected value
lambda λ	[10 ŝeq(10, -3, length = 100)]	0.0013

Table 2: Ridge regression hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
Ridge regression	0.0901	0.0684	0.0872	0.0785	0.0001	0.9082

Ridge regression has better performance than multiple linear regression, due to the shrinking.

3.1.3 Lasso regression

Specification

Use the same predictors as the ridge regression to fit the model.

Hyperparameter tuning

- Grid search and cross-validation method combination 3.

Name	Range	Selected value
lambda λ	[10 ŝeq(10, -3, length = 100)]	0.001

Table 3: Lasso regression hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
lassoreg	0.0897	0.0679	0.0874	0.0767	9.319	0.9082

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
GAMs	0.088	0.0684	0.0878	0.0820	6.54e-06	0.9068

The performance of Lasso regression is even better than the ridge regression because lasso model can shrink coefficient to zero (6 coefficient of predictors are zero in the model) . In a way, it helps the model to select variables.

3.2 Generalized additive models

3.2.1 Smoothing splines

Specification

- Use the RandomForest model to select 8 potential variables based on variable importance.
- Get the best degree of these 8 variables by cv method.
- Make all combinations which selects only 5 features from 8 potential variables.
- Fit the GAM model on each combined features and relative best degree value.
- Repeat previous step in all combinations feature groups.
- Get the lowest MRE model as the best model.

Hyperparameter tuning

- Grid search and cross-validation method combination 4.

Predictor	**Range** df_λ	**Selected value** df_λ
`return_rate`	[2, 20]	3
`introduction_time`	[2, 20]	2
`online_percentage`	[2, 20]	2
`total_sold`	[2, 20]	2
`price`	[2, 20]	6

Table 4: Smoothing spline hyperparameters.

Results

Compared to linear regression, GAM model has better performance. Here we don't assume that predictors and yearly return rate have linear relationship anymore. And we try to release this assumption and make the regression line more flexible.

3.3 Tree-based models

3.3.1 Decision tree

Specification

- Using cross-validation to find the best size. (the range of size is 1 to 7)
- Pruned the tree with the identified optimal size 7.
- Features used for the tree-based method:

  ["return_rate" , "first_return", "season", "introduction_season", "introduction_time", "SO", "SR", "unique_channel_count", "online_percentage", " total_returned", "total_sold", "price", "count_size", "numeric_size"]. This variable set would be used for most of the left models.

Hyperparameter tuning

We use automatically tuning method from the cv.tree() function to find out the best number of terminal nodes 5.

Name	**Range**	**Selected value**
Terminal nodes	[1,2,3,4,5,6,7]	7

Table 5: Decision tree hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
Regression decision tree	0.0890	0.0637	0.0884	0.0744	3.63E-05	0.8882

Regression decision tree dose not perform better than the GAM, but it helps us to get to know the variables importance for the further analysis.

3.3.2 Bagging

Specification

The same variables set as we mention in the decision tree method.

Hyperparameter tuning

- Grid search and manually search combination.

- Find out the best value by 10-fold cross-validation method.

Manually change the grid range and repeat previous step again for a few times. 6.

Name	**Range**	**Selected value**
Terminal nodes	[10, 20, 30]	20
Trees	[50, 100, 150]	100

Table 6: Bagging hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
bagging	0.0776	0.0542	0.0827	0.0843	8.44E-17	0.9183

Bagging tree has the best performance so far in the same variables set.

3.3.3 Random forest

Specification

- The same variables set as Ridge model..

Hyperparameter tuning

- Grid search and manually search combination. 7.

Name	**Range**	**Selected value**
Terminal nodes	[18, 19, 20]	19
Trees	[60, 68, 100]	68
Predictors	[4, 5, 6]	5

Table 7: Random forest hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
RandomForest	0.0768	0.0533	0.0816	0.0810	4.29E-05	0.9412

By randomly selecting fewer variables multiply times, Random Forest is slightly better than the bagging model.

3.3.4 Gradient boosting

Specification

- The same variables set as we mention in the decision tree method.

- A gaussian distribution is set in the 'gbm' model
- 80% of the data is used for training in each fold.

Hyperparameter tuning

· Each model undergoes 10-fold cross-validation to assess its performance, measuring the training error.8.

Name	**Range**	**Selected value**
Trees	[300, 500, 700]	700
Interaction depth	[2, 4, 5]	4
Shrinkage	[0.01, 0.03, 0.1]	0.03

Table 8: Gradient boosting hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
gbm	0.0779	0.0539	0.0820	0.0772	0.0001	0.8757

Gradient boosting needs to fit more times than the bagging and Random Forest model to get the optimal result.

3.4 Support vector regression

3.4.1 Linear kernel

Specification

The same variables set as decision tree model.

Hyperparameter tuning

- Grid search with cross-validation method to find out the best value. 9.

Name	**Range**	**Selected value**
Kernel	["linear"]	"linear"
Cost C	[0.1, 1, 5,10]	5

Table 9: SVR with linear kernel hyperparameters.

Results

Linear SVM doesn't perform well because our data set is not so obvious linear relationship.

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
Linear SVM	0.0831	0.0459	0.1005	0.0807	2.04e-05	0.9380

3.4.2 Non-linear kernel

Specification

- The same variables set as decision tree model.

Hyperparameter tuning

Using the 'tune' function for cross-validation in both polynomial and radial function with a combination stated in table 9. (cost=5 and d=2) and (cost=1, gamma=0.5) are the best Combinations.11.

Name	**Range**	**Selected value**
Kernel	["polynomial"]	'polynomial'
Cost C	[0.1, 1, 5]	5
Degree d	[2, 3, 4]	2

Table 10: SVR with Polynomial kernel hyperparameters.

Name	**Range**	**Selected value**
Kernel	["radial"]	'radial'
Cost C	[0.1, 1, 5]	1
Gamma γ	[0.5, 1, 2]	0.5

Table 11: SVR with Radial kernel hyperparameters.

Results

Method	Mean of Abs. Errors	Median	SD	IQR	Min	Max
polynomial_svm	0.0788	0.0490	0.0982	0.0723	1.68e-05	1.2539
radial_svm	0.0779	0.0466	0.0968	0.0775	0.00019	0.9497

Nonlinear kernel svm models have better performance than the linear kernel model.

3.5 Neural networks

3.5.1 Multi-layer perceptron

Specification

- The same variables set as decision tree model.
- the hidden units were initially set to tune()
- initial regularization parameter was set between 1e-100 Inf
- epochs were set between 20 and 200
- All the other parameters like learning rate, activation function and dropout were automatically selected by the tuning algorithm.

Hyperparameter tuning

- Regular grid and space-filling design grid search tune the model.
- Racing method to find the optimal values.

The associated learning convergence curve is as below

Name	Range	Selected value
Hidden layers	[Dense, Dense]	Automatic selection
Hidden units	[1 - 10]	9
Activation function	[Linear]	Automatic Selection for regression
Penalty	[1e-100,Inf]	0.00001
Epochs	[50 – 200]	198

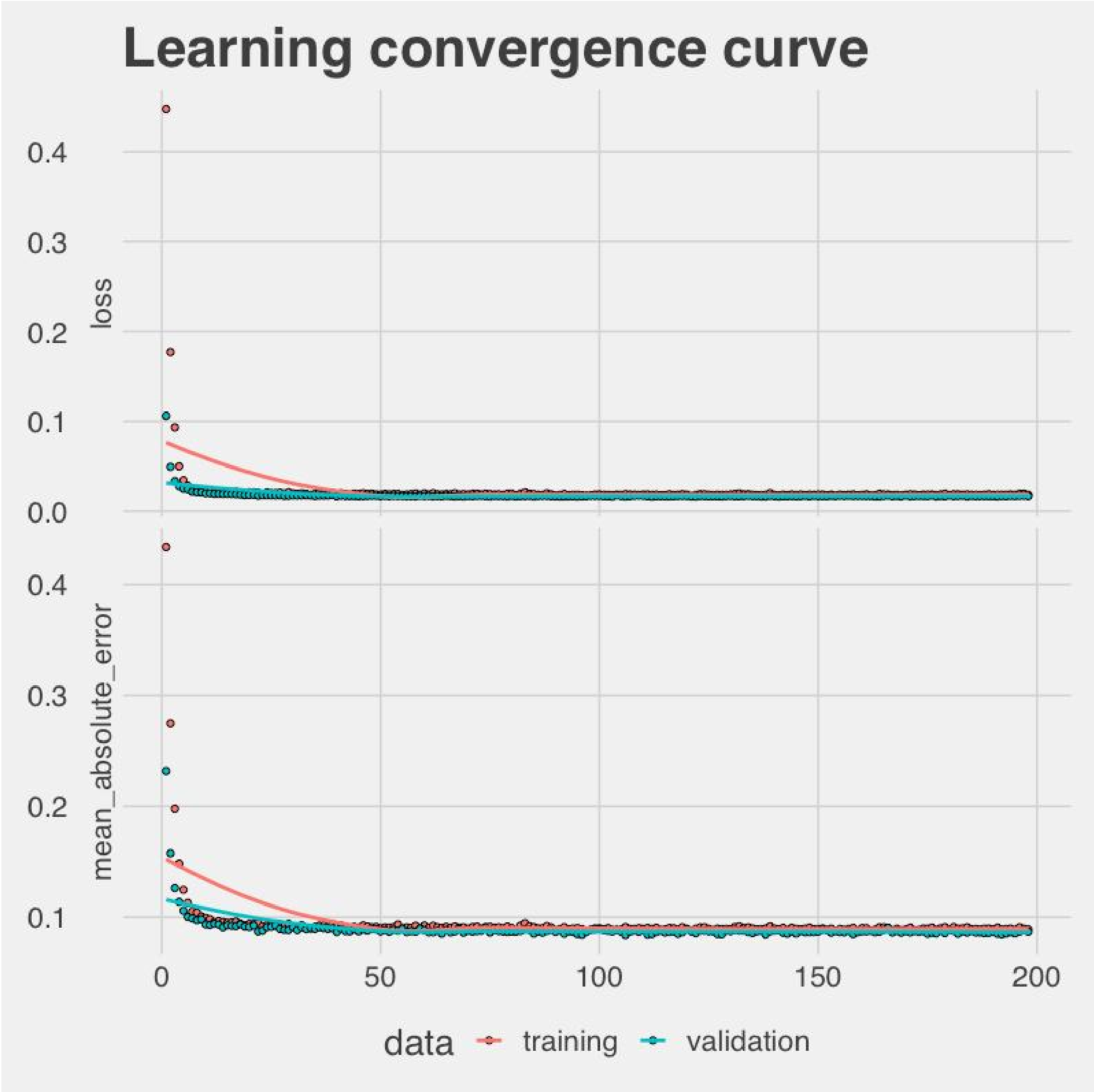

Figure 1: Figure 1: MLP learning convergence.

Results

Grid/ Method	**Mean**	**SD**	**Median**	**IQR**	**Minimum**	**Maximum**
Racing	0.0806	0.0005	0.0809	0.0003	0.0798	0.0810
SFD - Grid	0.0874	0.0113	0.0827	0.0113	0.0801	0.12
Regular Grid	0.102	0.021	0.0979	0.0463	0.0799	0.128

The racing method produced the grid with the best performing MAE but overall not significantly different from MAE produced by regular and SFD grids

4 Model Evaluation

12.

Model	Val. MAE (%)
Linear baselines	
Logistic regression	0.1024
Ridge regression	0.0910
Lasso regression	0.0902
Generalized additive models	
Smoothing splines	0.0880
Tree-based models	
Decision tree	0.0890
Bagging	0.0782
Random forest	0.0774
Gradient boosting	0.0817
Support vector regression	
Linear kernel	0.0831
polynomial kernel	0.0779
Neural networks	
Multi-layer perception	0.0797

Table 12: Validation set metrics.

5 Conclusion

Based on the sales data, product yearly return rate dose not perform strong linear relationships with all the predictors. Thus nonlinear models are better fit in this subject.

Compared with MAE among 15 the models, we found that Random Forest model has the best performance. Thus, we use Random Forest model to refit training data (train and validation set) to predict the test data.

A Appendix.1

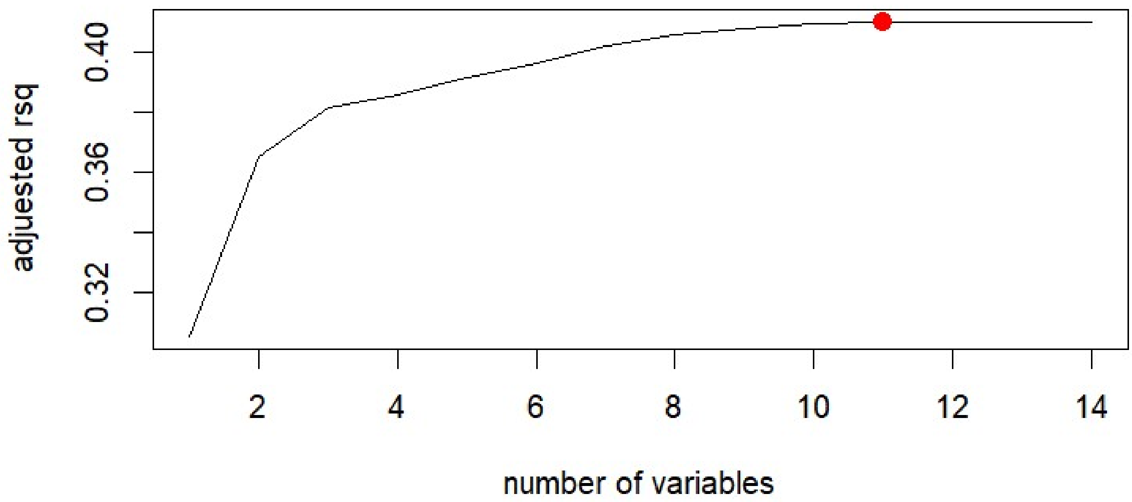

Figure 2: Forward selection for the best number of variables

As we can see, the maximum r square is located at 13. So we take 13 variabels to fit our linear regression model.

B Appendix.2

	count_size	price	total_sold	total_returned	offline_sales	offline_return	online_sales	online_return	online_percentage	unique_channel_count	yearly_return_rate
count_size	1.00000000	0.18835325	0.2638171	-0.19017429	0.2497190501	-0.13342059	0.1689736	-0.1604248242	-0.2182370	0.241032164	-0.09176258
price	0.18835325	1.00000000	-0.1445794	-0.01511906	-0.1146668349	0.05621192	-0.1241117	-0.0324412163	-0.0743816	-0.147719652	0.10397479
total_sold	0.26381714	-0.14457944	1.0000000	-0.36892761	0.8791648554	-0.40191994	0.7362116	-0.2690554355	-0.1568125	0.754446809	-0.12449186
total_returned	-0.19017429	-0.01511906	-0.3689276	1.00000000	-0.1197528781	0.30681673	-0.5621754	0.9598758762	-0.1955702	-0.111390627	-0.23574681
offline_sales	0.24971905	-0.11466683	0.8791649	-0.11975288	1.0000000000	-0.42820507	0.3247673	0.0003911249	-0.3858796	0.862692090	-0.18547788
offline_return	-0.13342059	0.05621192	-0.4019199	0.30681673	-0.4282050655	1.00000000	-0.1895943	0.0276060578	0.1659443	-0.387430855	0.02856652
online_sales	0.16897362	-0.12411168	0.7362116	-0.56217545	0.3247672567	-0.18959432	1.0000000	-0.5345775835	0.2367851	0.272231443	0.01632420
online_return	-0.16042482	-0.03244122	-0.2690554	0.95987588	0.0003911249	0.02760606	-0.5345776	1.0000000000	-0.2542957	-0.002839757	-0.25601573
online_percentage	-0.21823703	-0.07438160	-0.1568125	-0.19557022	-0.3858796009	0.16594429	0.2367851	-0.2542956832	1.0000000	-0.422057130	0.31881111
unique_channel_count	0.24103216	-0.14771965	0.7544468	-0.11139063	0.8626920900	-0.38743086	0.2722314	-0.0028397572	-0.4220571	1.000000000	-0.19610029
yearly_return_rate	-0.09176258	0.10397479	-0.1244919	-0.23574681	-0.1854778813	0.02856652	0.0163242	-0.2560157263	0.3188111	-0.196100287	1.00000000

Figure 3: Table used for calculating correlations between variables in Linear regression

C Appendix.3

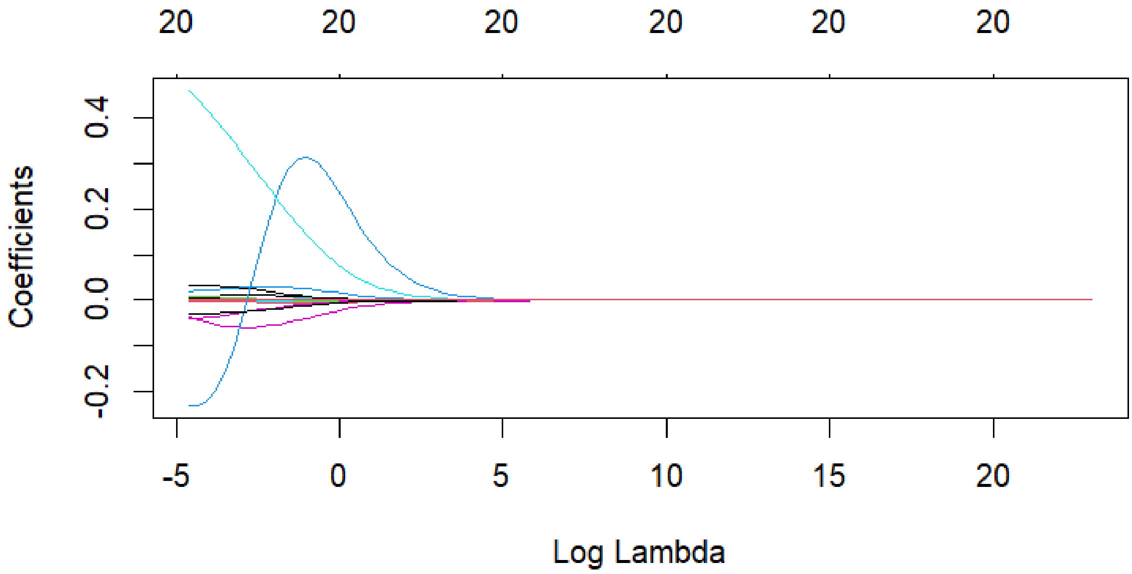

Figure 4: Coefficients and lambdas in the ridge model

D Appendix.4

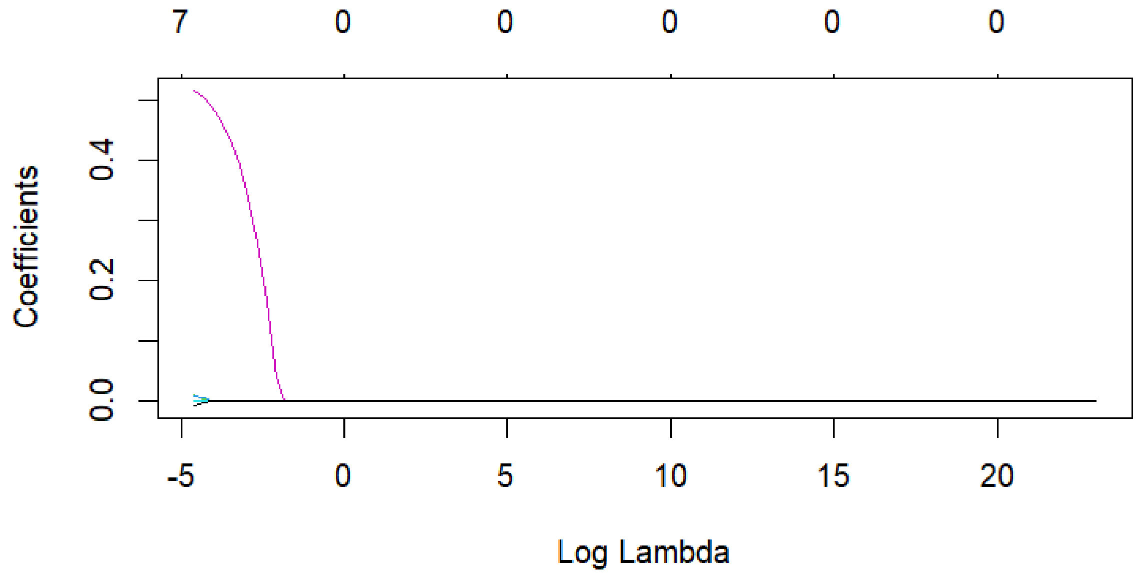

Figure 5: Coefficients and lambdas in the Lasso model

E Appendix.5

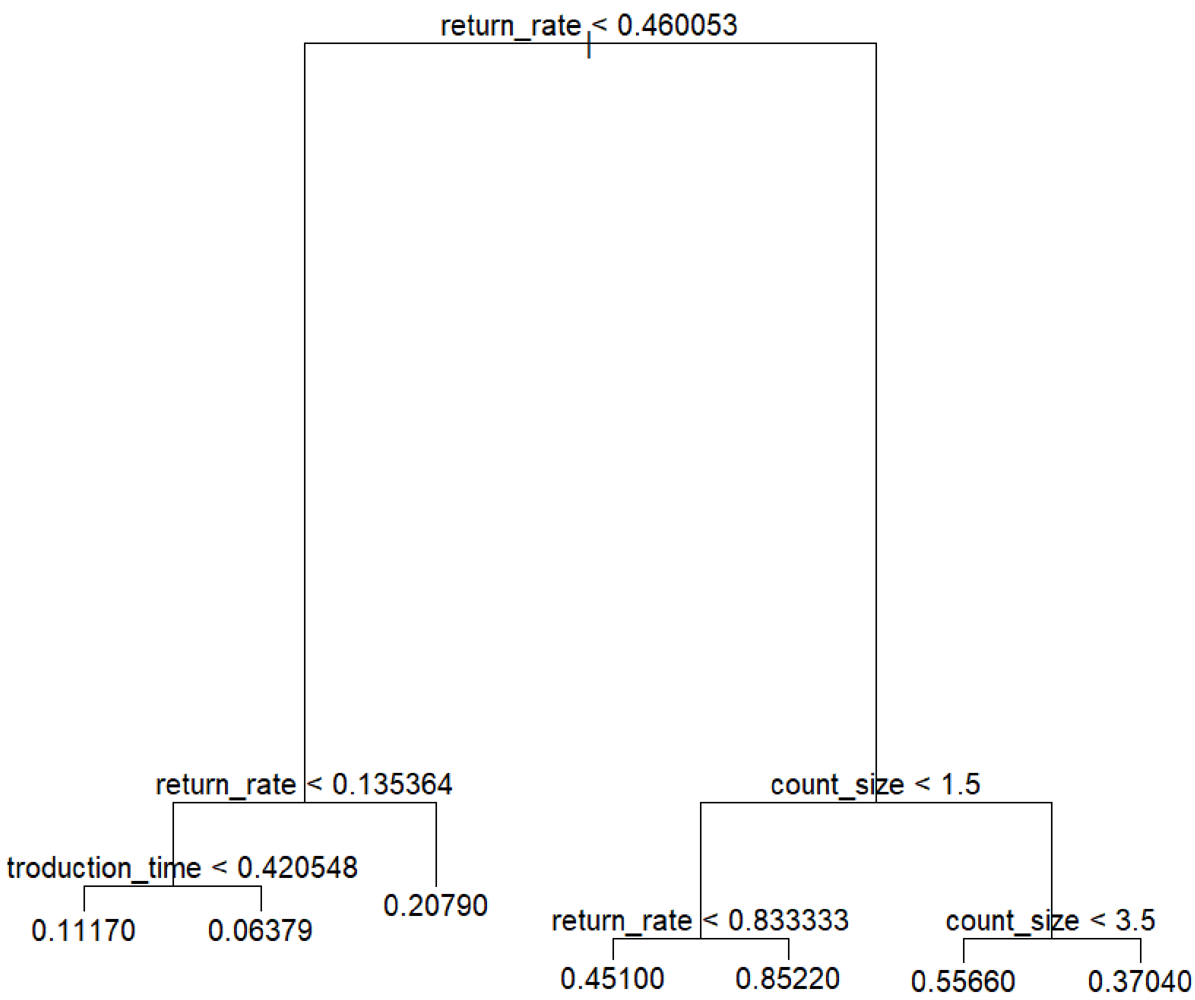

Figure 6: Pruned decision tree

F Appendix.6

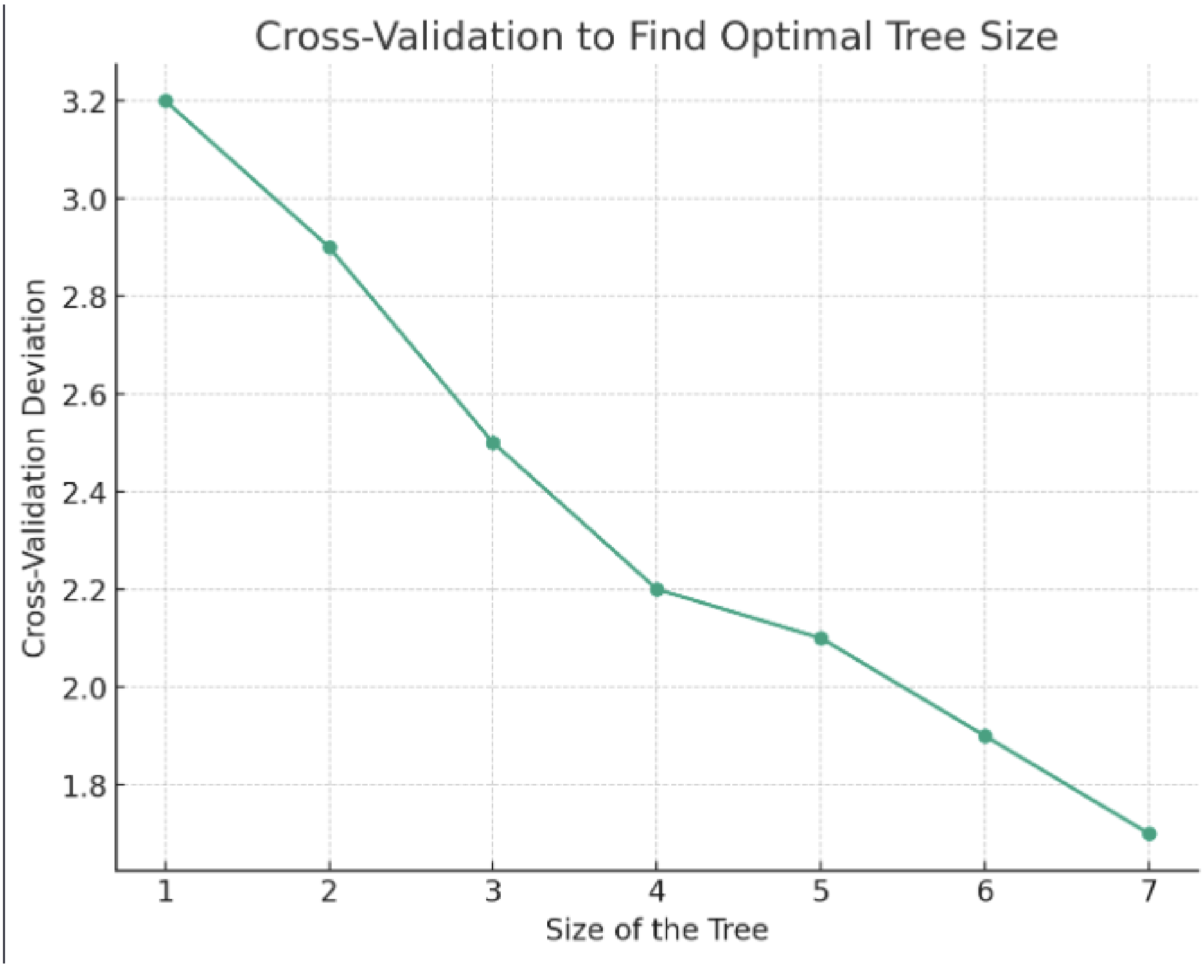

Figure 7: Choosing the size of the decision tree, we choose 7 here

G Appendix.7

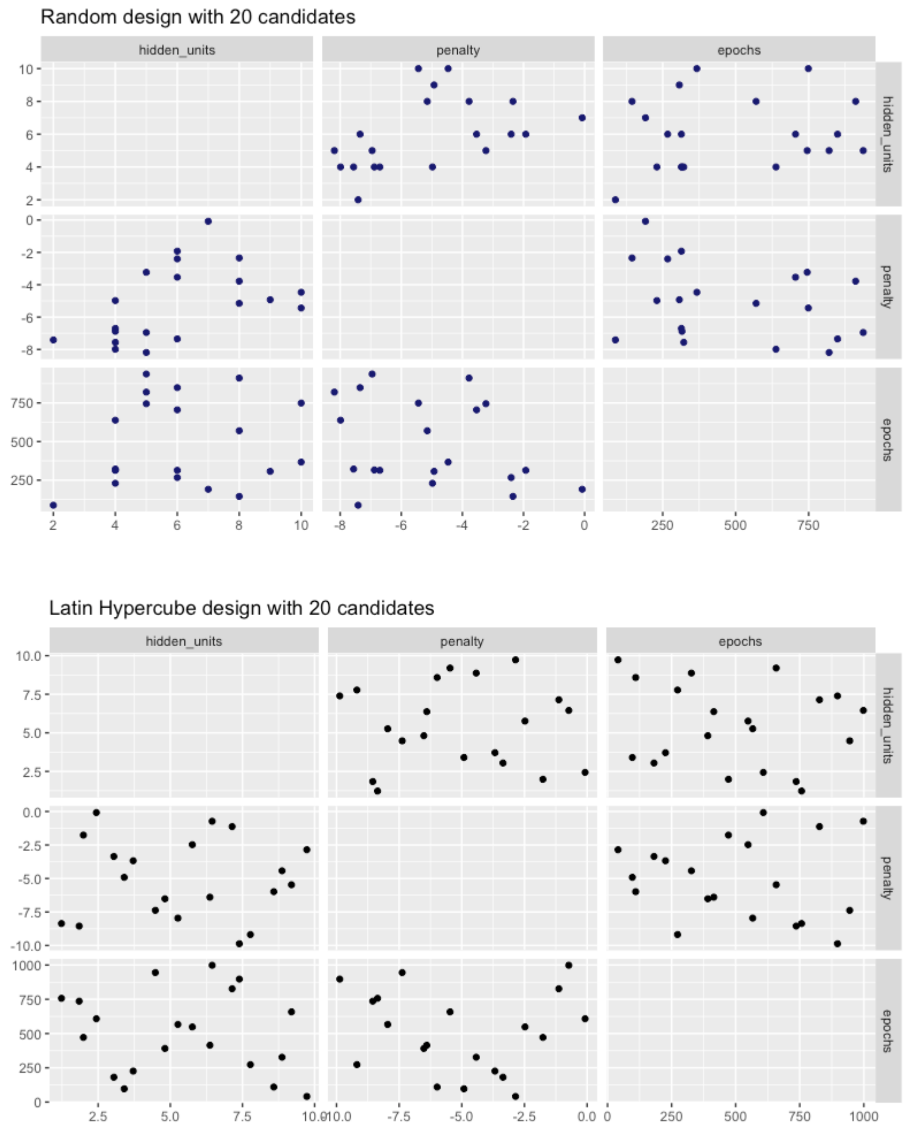

Figure 8: Grids using 10 folds cross validation

H Appendix.8

hidden_units	**penalty**	**epochs**	**mean**	**n**	**std_err**	**Model config**
5	1e-05	200	0.079870439	10	0.001426576	Model23
10	1e-05	200	0.080292501	10	0.001246539	Model24
10	1e-10	125	0.080755591	10	0.000973592	Model12
10	1e-05	50	0.080954451	10	0.001004143	Model06
10	1e-10	50	0.081238281	10	0.001306631	Model03
10	1e-10	200	0.081298193	10	0.001312349	Model21
10	1e-05	125	0.081337037	10	0.000989104	Model15
5	1e-10	200	0.081430428	10	0.001110794	Model20
5	1e-10	125	0.081438982	10	0.001229884	Model11
5	1e-05	125	0.081597907	10	0.001209955	Model14
5	1e-10	50	0.082323881	10	0.001161389	Model02
5	1e-05	50	0.082696694	10	0.001174234	Model05
5	1	200	0.097551338	10	0.001555825	Model26
5	1	125	0.097885539	10	0.001474648	Model17
10	1	125	0.098582178	10	0.001833293	Model18
10	1	200	0.098589213	10	0.001614523	Model27
5	1	50	0.103941544	10	0.002813022	Model08
10	1	50	0.117595594	10	0.002832309	Model09
1	1e-10	200	0.127260079	10	0.002541278	Model19
1	1e-10	125	0.127625219	10	0.002379545	Model10
1	1	200	0.127690249	10	0.002393661	Model25
1	1e-10	50	0.12779343	10	0.002520817	Model01
1	1	125	0.12792364	10	0.002370717	Model16
1	1e-05	125	0.127978801	10	0.002881961	Model13
1	1e-05	200	0.128318394	10	0.002373846	Model22
1	1	50	0.12838667	10	0.002455277	Model07
1	1e-05	50	0.128429494	10	0.002330917	Model04

From the table we can conclude that simplest models are those that impose larger penalty values and/or have fewer hidden units. You can see that using regular random grid-search, the model with the least test mean absolute error will be the one that includes 5 hidden units, a regularization lambda of 0.00001 and where the number of epochs equals 200. This is farthest right in the panel.

I Appendix.8b

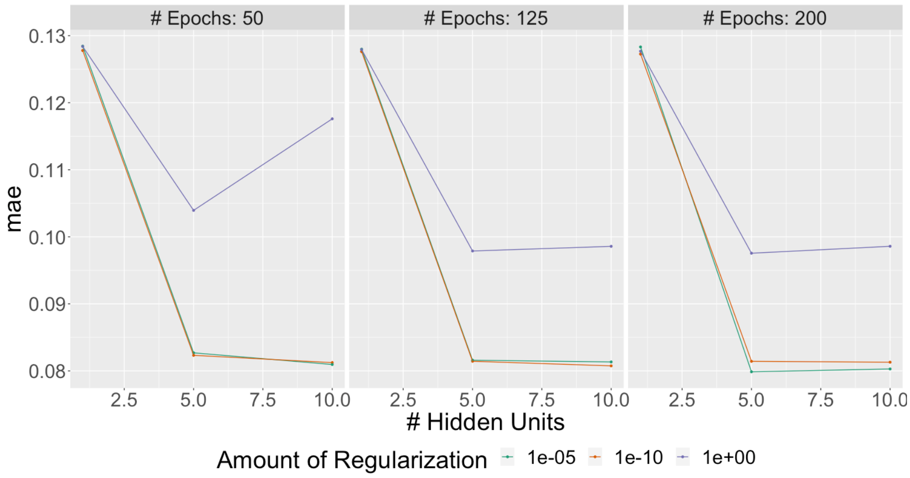

Figure 9: Regular gird

J Appendix.9

Space-filling design grid using a maximum entropy design with 20 candidate values

Hidden units	penalty	epochs	mean	n	std_err	Model config
9	1.03539844803594e-06	65	0.080141581	10	0.001081694	Model20
9	3.7511565369584e-05	98	0.080893628	10	0.001234361	Model04
7	1.34670578594659e-10	184	0.080951221	10	0.001039925	Model16
9	3.39794407580813e-06	148	0.081052379	10	0.001163215	Model05
4	4.17726267625665e-08	192	0.081279586	10	0.001237228	Model13
6	4.0054971597544e-07	111	0.08148341	10	0.001009677	Model02
4	0.00021054	83	0.081663902	10	0.000998817	Model10
5	1.10369527205928e-08	134	0.081708959	10	0.001191477	Model12
8	6.51108011395515e-10	90	0.081711309	10	0.000998086	Model06
5	3.05392422094145e-07	108	0.082141945	10	0.001231606	Model11
3	0.000440241	76	0.083347546	10	0.001305887	Model08
3	1.21445507527429e-05	63	0.08345174	10	0.001339792	Model18
10	0.001399322	55	0.083481489	10	0.001459276	Model03
6	0.003328904	132	0.084416865	10	0.001176576	Model07
2	0.017315688	120	0.092576302	10	0.001190674	Model15
2	2.79996830450631e-09	160	0.093296221	10	0.001050357	Model19
7	0.074659471	174	0.094202736	10	0.001160634	Model14
6	0.186528937	200	0.094293736	10	0.001158645	Model17
2	0.833369986	141	0.097977286	10	0.001905264	Model01
1	4.34639361919853e-09	163	0.128877258	10	0.002720835	Model09

K Appendix.10

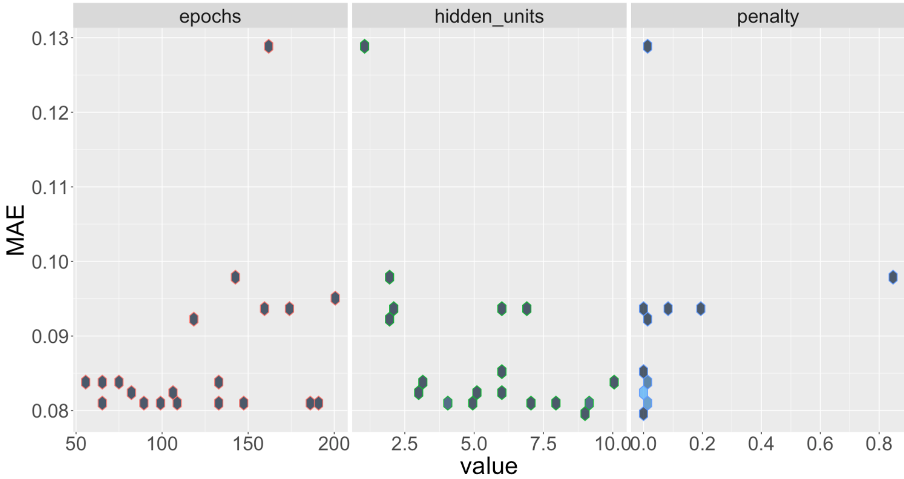

Figure 10: Space-filling design grid

L Appendix.11

Racing method

hidden_units	penalty	epochs	mean	n	std_err	Model config
9	0.00014837	198	0.07979632	10	0.00089273	Model18
5	5.4148289631411e-05	123	0.08086255	10	0.00118318	Model12
5	9.35775021790511e-08	170	0.08086495	10	0.00113856	Model08
9	1.63423493542111e-05	99	0.08103357	10	0.0010853	Model04

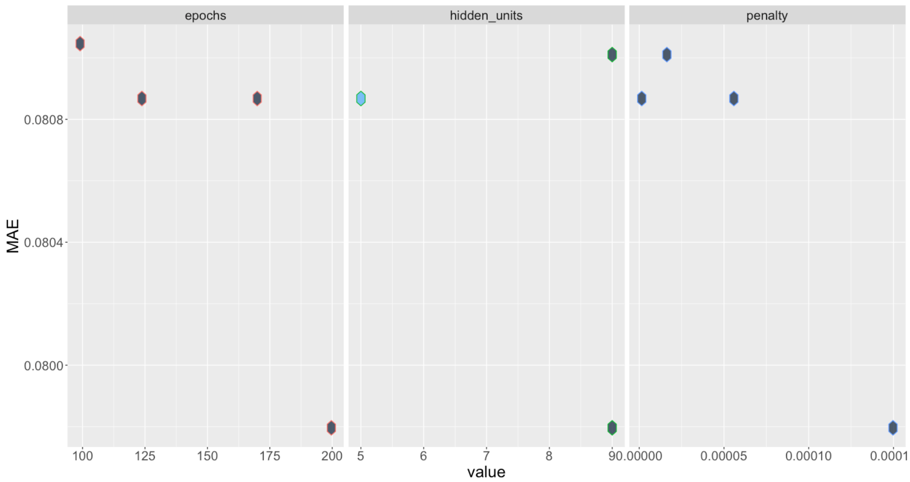

Figure 11: models produced by the racing method