Imperial College Business School

IMPERIAL COLLEGE BUSINESS SCHOOL LONDON

Euro Area Yield Curve Prediction Using Machine Learning Models

Student ID Number: 01566339

August 2019

Contents

List of Figures

1 Client Specification

Morgan Leon is an American multinational Investment Bank which specializes in the development of sophisticated investments and trading strategies of Fixed Income Securities which are subject to Interest Rate changes. The head of Interest Rate Strategy team is interested in the evaluation of model performance of different machine learning methods for the prediction of changes in interest rates so that his team will be able to make more accurate trading decisions. This will be achieved by introducing the Linear Regression based models (as Benchmark models) and investigating the usefulness of machine learning predictive models within the Fixed Income market. In addition, he is interested to see whether variable selection and data compression techniques can offer potential improvements on the model performances.

2 Motivating Framework

The focus of this research paper is on the prediction of changes in yield curve, which is the centrepiece of bond markets. Forecasting future interest rate changes is becoming more and more important nowadays, particularly because interest rate is a key financial variable that influence the decisions of regulatory bodies, investment managers and consumers. By having an informed prediction of how interest rates will move in the future, markets will be able to proactively adapt to changing conditions. The daily variation and fluctuation around an unstable average level of interest rates decrease the prediction ability of traditional statistical models [Bauer, 2017]. With the non-linear and non-stationary environment of interest rate markets, the use of more advanced models with complex features becomes more attractive. Although there is enormous research and development on the use of machine learning in Finance, it is applicable to the Equity Market (e.g. [Gu et al., 2018]). While the Bond Market is one of the most important sources of finance, much less scientific work has been done focusing on machine learning in the Bond Market (e.g [Nunes et al., 2018],[Ganguli and Dunnmon, 2017]).

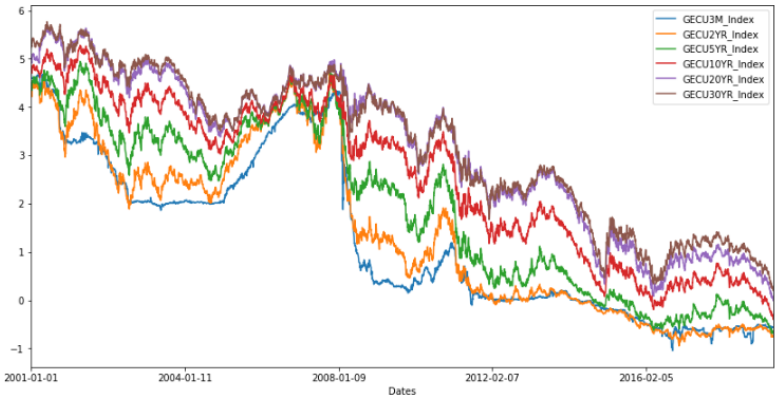


Figure 1: Euro Area Yield Curve.

3 Machine Learning Methodologies

3.1 Simple and Penalized Linear Regressions

Simple linear regressions are still widely used in practise due to their simplicity and limited computation power required. However, despite their popularity, linear models tend to suffer from low bias and high variance as well as model interpretability as the number of predictors increases [James et al., 2014].

In order to overcome this difficulty, 2 well-known strategies are introduced; Shrinkage and Dimension Reduction through Ridge, Lasso, Elastic Net and Principal Component Analysis. Lasso model will be used as the benchmarking predictive model. With Shrinkage approach a penalty term is introduced in the objective function and depending on the type of shrinkage, coefficients estimated towards zero or be exactly zero.

3.2 Permutation Feature Importance Measurement

The importance of a feature is measured by calculating the decrease in the models prediction R^2, since in this case the model relied on the feature for the prediction. A feature is unimportant if dropping its values leaves the model R^2 unchanged, because in this case the model did not consider the feature for the prediction.

3.3 Principal Component Analysis and Partial Least Squares Regression

Principal Component Analysis (PCA) and Partial Least Squares (PLS) are popular techniques for deriving a low-dimensional set of features from a large set of variables that contain the majority of the explanatory power of the full feature set. This is achieved by computing linear combinations of the variables.

While PCA focus only on the explanatory variables and seeks to find hyperplanes of maximum variance, PLs which is an extension of PCA, takes into account both the respond and explanatory variables and seeks to find hyperplanes of maximum variance as well as correlation between them [Bianchi and Tamoni, 2019].

3.4 Artificial Neural Networks

For the prediction of the targets, the paper focuses on the FeedForward Neural Network whose model was inspired by the way the human brain is designed. The way the FeedForward Neural Network works can be illustrated in Figure 2. To begin with, the ANN consists of an input layer with the initial predictors (called neurons), a number of hidden layers (which are non-linear transformations of the initial predictors allowing the model to learn more complex functions), and an output layer with the predicted values of the targets. In addition, as it can be seen from Figure 2, each neuron in each layer is directly connected to the neurons of the subsequent layer.

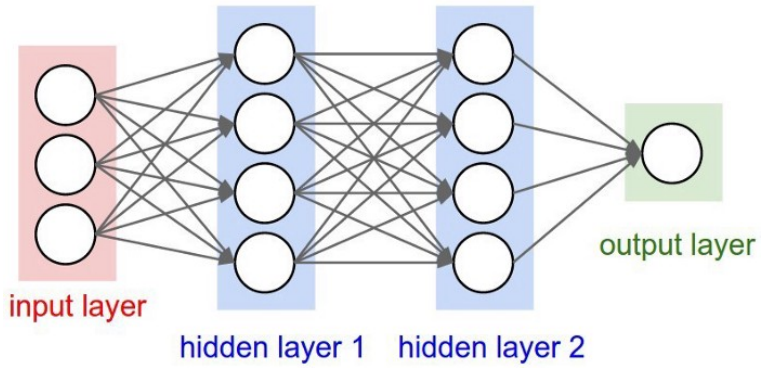


Figure 2: Multi-layer Neural Network.

Network Architecture:

The performance of the Neural Network model depends heavily on the selected number of hidden layers. For testing the predictive performance of Neural Networks, the networks that are tested, start with a single hidden layer and are expanded to 3 hidden layers (excluding the output layer). The number of hidden neurons were determined using the geometric pyramid rule as suggested in [Masters, 1993]. This means that the number of neurons follow a pyramid shape, with the number decreasing from the input to the output following a geometric sequence. The specific Model Architectures used in the paper can be seen in Table 3.

Activation Function:

The Activation Function used in the models is chosen to be the Rectified Linear Activation Function (ReLU). ReLU is a piecewise linear function that outputs the input directly if it is positive, otherwise, it outputs zero. Because of its non-saturating non-linearity, ReLU function works better than other non-linear functions (e.g. tahn and sigmoid); it is able to train faster (due to computational efficiency) without making significant difference in the accuracy [Glorot et al., 2011].

Optimization and Regularization:

In some cases, when training a large network, there is a point during training where the model will stop generalizing and it tends to learn the tiniest details presented in the data. One way to prevent overfitting is by reducing the complexity of the model using regularization. Regularization will shrink many of the weights towards zero. Each neural network is trained by minimizing the mean squared loss function with the penalizing term L1 as illustrated in Table 3.

Adding to the penalty term, early stopping is introduced for stopping the training as soon as the performance on the validation set decreases as compared to the performance on the validation set at the prior training epoch.

Simultaneously, in order to find optimal values weights of neural network to minimize the loss function, different optimization techniques were used such as Adam, Nadam and Rmsprop Optimizers.

Dense layer:

The final layer of the models is a typical Dense layer, with a number of activation units equal to the number of distinctive classes or number of percentage change predictions using Sigmoid activation function which returns values from -1 to 1.

Initializer:

Initialization of the weights of neural networks is another parameter which must be tuned since it can speed up the learning process if it is chosen appropriately. The initializers used where the Normal, Glorot_Unifrom and Uniform functions.

Hyper-parameter Tuning Neural Networks					
Model Architecture	Regularisation L1	Optimisation	Initialiser	Activation Function	Epochs
[4, 2, 1]	0.0001	Adam	Glorot_Unifrom	Relu	300
[15, 6, 1]	0.00001	Nadam	Normal		
[128, 64, 32, 1]	0.000001	Rmsprop	Uniform		
[32, 16, 8, 4, 1]					
[64, 32, 16, 8, 4, 1]					

Figure 3: Multi-layer Neural Network Hyper-Parameter Tuning.

3.5 Random Forest Regression

A Random Forest consist of an arbitrary number of simple trees as illustrated in Figure 4, where at every split are trained on different parts of the same training set with the goal of reducing the variance and are used to determine the final outcome. The responses of the trees are averaged to obtain an estimate of the dependent variable [James et al., 2014].

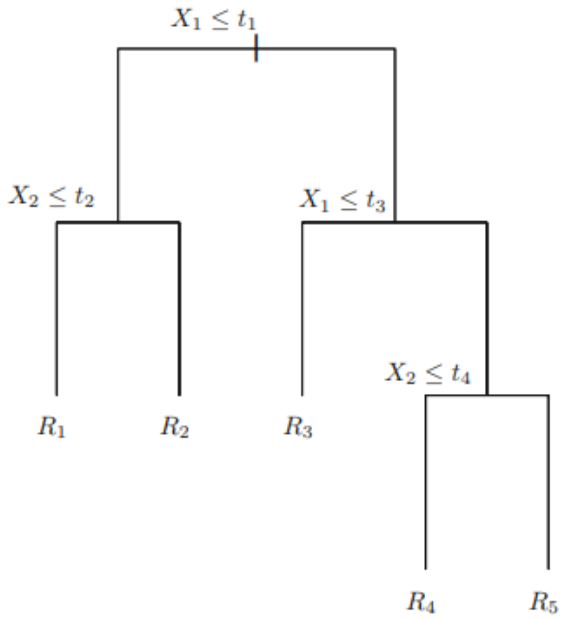


Figure 4: Random Forest Regression.

Hyper-Parameter Tuning Random Forest	
Number of trees in Random Forest	[30,50,100,200]
Number of Features to consider at every split	[Auto, sqrt]
Maximum number of levels in tree	[3,5,10]
Minimum number of samples required to split a node	[2,5,7,10]
Minimum number of samples required at each leaf node	[1,2,5,10]
Bootstrap	[False, True]

Figure 5: Hyper-Parameter Tuning of Random Forest.

After tuning the hyper-parameters of Neural Networks and Random Forest in the validation set, the best models in the validation set are selected to be tested in the out-of-sample dataset. The best model out-of-sample is included in the model performance tables.

4 Research Design

4.1 Targets

The focus of the study is the Euro Area Government Yield Curve for the following reasons. First of all, the term structure and yield curve of government bonds has been proven to be a useful indicator of economic activity. Large number of activities in financial markets is determined by the yield curve and its shape reflects the performance of the economy. Secondly, this bond class has higher liquidity compared to other bond classes and the size of the market is considerably higher.

The study is conducted by recursively forecasting the overlapping one-day ahead percentage change in 3-month and 2-year yields.

Apart from predicting the percentage changes in yields, Artificial Neural Networks were also used to determine a 2-class problem which predicts whether upcoming percentage change in yields for 2-month and 3-year maturities was positive or negative.

4.2 Features

The machine learning methodologies will be compared against the benchmark model which will be based on the linear regressions framework. All of the machine learning models are designed to approximate the empirical specification

$$E_t\left[{r_{t+1}}^{(n)}\right] = \hat{\alpha} + \hat{\beta}^T \mathbf{x}_t$$

as well as its extension which includes macroeconomic information which is not spanned by the current term structure. The first application concerns the forecasting of future bond yield percentage changes based on cross-section of yields. The second application consists of forecasting future bond yield percentage changes based on both the term structure and a large set of 128 macroeconomic and financial variables.

The variables were selected to represent 8 main categories of macroeconomic time series: Interest Rates, Yields, Equities, Currencies, Economic Indicators, Volatilities, Commodities and Swap Rates of Europe, US, Japan and UK. Additional features were added, directly calculated from the previous features, mainly yield spreads as well as up to three days of past percentage changes in yields. Full list of series is given in Appendix and it is similar to lists used in other research papers (e.g. [Stock and Watson, 2002], [Ludvigson and Ng, 2009], [Bloomberg, 2017]). The sample observations are taken daily from the Bloomberg Professional Database and cover the period 2001:02 - 2019:07.

4.3 Data split and normalization:

Initially, the data is divided in three sub-samples: a training set used to train the model; a validation set used to select the hyper-parameters which optimize the objective function; a test set which represents the out-of-sample model performance. The training and validation periods together cover 80% of the data and the remaining 20% is used for the out-of-sample testing.

Finally, all interest rates and yields variables were replaced by their percentage change since we are interested in predicting the daily percentage change of the yields rather than the level of the yield itself. The remaining variables were normalized by subtracting the mean and dividing by the standard deviation after splitting the data into the three sub-datasets. Normalization played an important role since the data considered has very different scales in some cases. We recursively refit the machine learning methods at each time t, meaning that each time we refit, we increase the in-sample period by one daily observation.

4.4 Forecasting Evaluation

For the comparison of the predictive performance of each individual machine methodology, two metrics were used. The first metric is considered to be the out-of-sample Mean Squared Error (MSE), i.e,

$$MSE^{(n)} = \frac{1}{T - t_o - 1} \sum_{t=t_0}^{T-1} ({r_{t+1}}^{(n)} - \hat{r}_{t+1}^{(n)})^2$$

where $\hat{r}_{t+1}^{(n)}$ is the one-step ahead forecast of daily percentage change of the yield for maturity n and t_0 is the first date in the prediction sample.

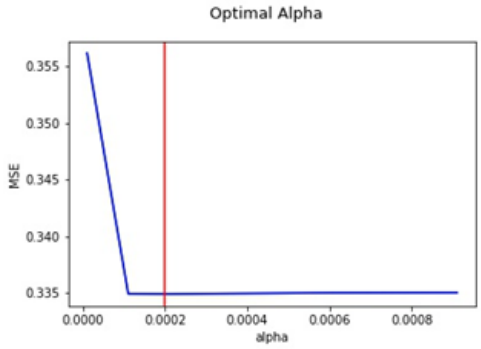
In addition to the MSE, the out-of-sample predictive R^2_{oos} is used for the comparison of the forecasts obtained from each methodology as suggested by [Gu et al., 2018] which is calculated as

$$R^2_{oos} = \frac{1}{T - t_o - 1} \frac{\sum_{t=t_0}^{T-1} ({r_{t+1}}^{(n)} - \hat{r}_{t+1}^{(n)})^2}{\sum_{t=t_0}^{T-1} ({r_{t+1}}^{(n)})^2}.$$

5 Results and Discussion

5.1 Variable Importance

To begin with, LASSO regression is used for the selection of a subset of variables with the highest predictive power. The tuning parameter is estimated by minimizing the overall loss measure (MSE) as illustrated in Figure 6.



(a)

Lasso hyperparameter tuning	
3 months	2 years
Optimal alpha: 0.0002	Optimal alpha: 0.0006

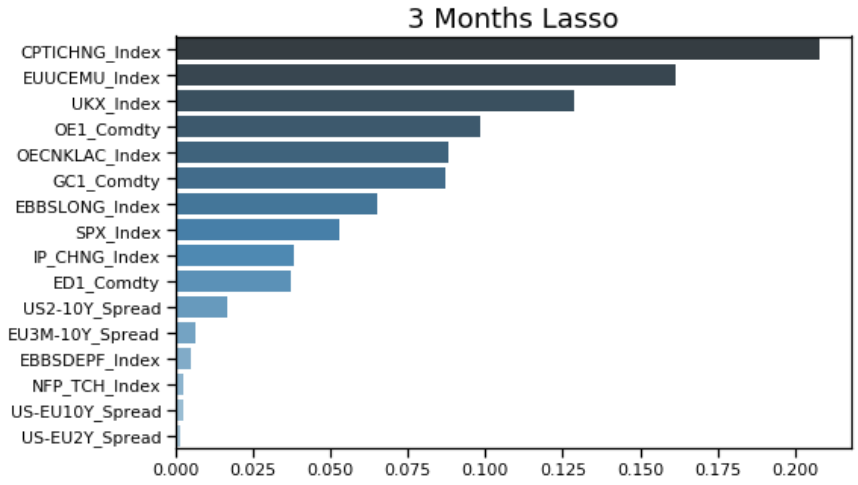
(b)

Figure 6: Lasso Hyper-Parameter Tuning.

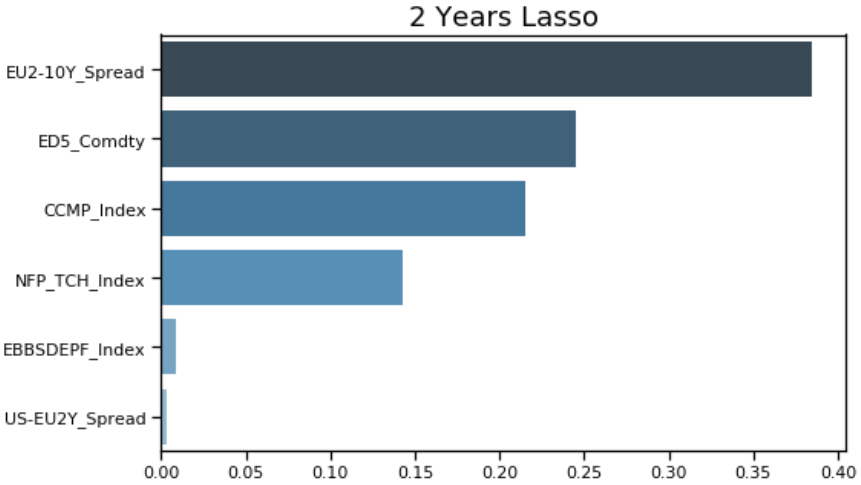
Furthermore, in order to find the variables that the ANN model considered important for the prediction, the Permutation Feature Importance Measurement is used. Variable importances within a model are normalized to sum to one so that relative importances for that particular model are more interpretable.

Figure 7 and 8 report the most relevant variables for Lasso and ANN models for the two predicting targets. First of all, it can be seen that most relevant features of Lasso are not common for both targets. Table 7 also shows that yield spreads are important for yield return prediction. Likewise, other macroeconomic indicators related to economic activity, commonly used for forecasting purposes. Moreover, features related to the European Central Bank Balance Sheet were also useful such as its total assets as well as its Long Term Refinancing Operations.

Secondly, it can be seen that Lasso and ANN models consider different variables as relevant for predicting the percentage changes. The figures show that Lasso focuses more on specific group variables including Economic Indicators, Equities, Commodities, Government Bond Spreads and Interest Rates. Conversely, ANN models are more democratic, drawing predictive information from almost all 8 main categories of macroeconomic time series. These differences occur mainly due to the fact that Lasso only considers linear relationships between the variables while Neural Network captures non-linear patterns in the data.

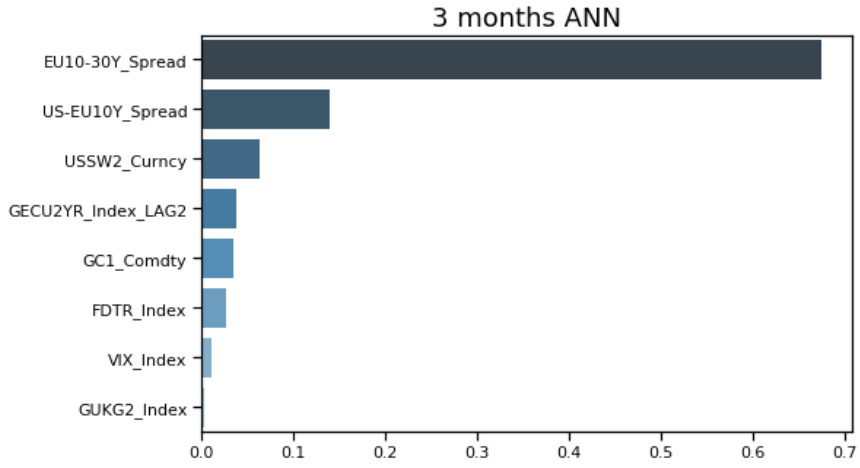


(a)

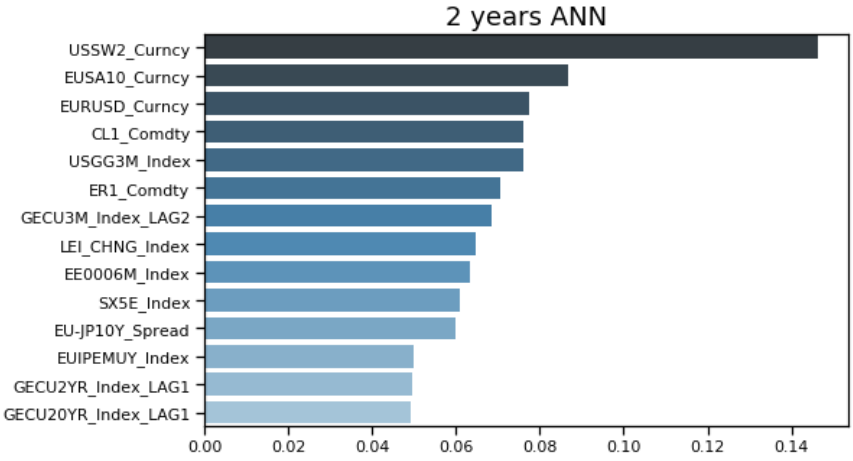


(b)

Figure 7: Lasso Variable Selection.



(a)

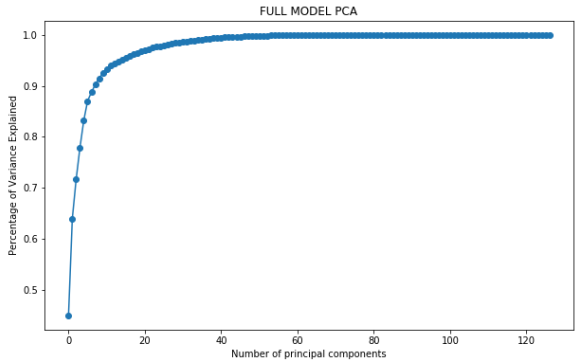


(b)

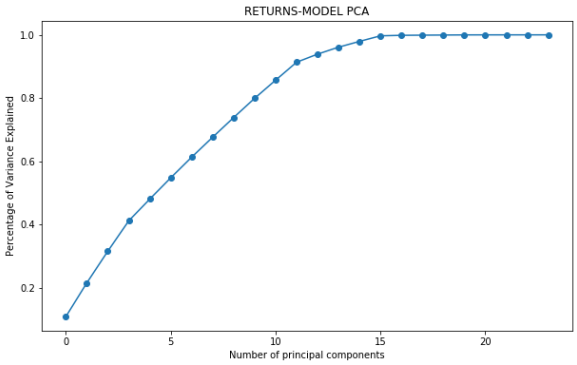
Figure 8: Neural Network Variable Importance.

5.2 Principal Components Analysis

Figures 9 (a), (b) and (c) illustrate the cumulative variance explained by the Standard Principal Components. It can be noted that for the model considering only the Interest Rates as input data, the first 12 principal components explain 91.33% of the variability. On the other hand, for the model with both Interest Rates and Macroeconomic data the first 7 principal components explain 88.72% of the variability. Those first principal components are used to reduce the dimension of input space in the Artificial Neural Network models.



(a)



(b)

Cumulative Variability Explained (%) PCA

# Factors	Interest Rates and Macroeconomic Variables	# Factors	Interest Rates
PCA_1	44.94	PCA_1	10.81
PCA_4	77.88	PCA_5	48.01
PCA_7	88.72	PCA_10	79.92
PCA_86	100.00	PCA_21	100.00

(c)

Figure 9: Explained Variance of the factors extracted from a Standard Principal Component Analysis (PCA).

5.3 Model Performance

5.3.1 Regression Results

Analysis for the models with only Interest Rates as Inputs:

Table 10 (a) illustrates the out-of-sample MSE and R^2_{oos} for the Penalized Linear Regressions which take as input the percentage changes on yields together with their lagged values. It can be seen that all models perform poorly out-of-sample with negative R^2_{oos} across both maturities and this is somehow expected showing the motivation to explore more complex and non-linear models.

Table 10 (b) represents the out-of-sample MSE and R^2_{oos} when introducing Neural Network models and Random Forest. It can be seen that Random Forest performs poorly across all maturities with significantly negative R^2_{oos} . On the other hand, Neural Networks outperform the benchmark models (Penalize Linear Regressions) with positive R^2_{oos} across both maturities.

Interestingly, looking at Table 10(c), the fact that yields enter the model as a linear combination (using PCA) slightly reduces the out-of-sample performance.

			Mean Squared Prediction Error		R^2_{oos}	
Panel A: Simple and Penalized Linear Regressions	Optimal alpha 3-Month	Optimal alpha 2-Year	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
Lasso	0.00005	0.00021	0.00092	0.00092	-0.00028	-0.0001
Elastic Net	0.00042	0.0011	0.00104	0.00086	-0.000267	-0.00285
Ridge	1107.104	8265.250	0.00104	0.00087	-0.00021	-0.00525
PLS Regression	2 components	2 components	0.00105	0.00095	-0.01284	-0.10101

(a)

	Mean Squared Prediction Error		R^2_{oos}	
Panel B: Neural Networks and Random Forest	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
Random Forest	0.003	0.0012	-0.00235	-0.28046
ANN 4 layers	0.0007	0.00001	**0.00166**	**0.0011**
ANN 2 layers	0.0007	0.0007	0.00001	0.00131

(b)

	Mean Squared Prediction Error		R^2_{oos}	
Panel C: Neural Networks (with Dimension Reduction)	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
ANN 4 layers + PCA (12 Components)	0.0009	0.0007	0.00084	0.0009
ANN 2 layers + PCA (12 Components)	0.0009	0.0007	**0.00107**	**0.001**

(c)

Figure 10: Out-of-Sample Results with only Interest Rates data.

Analysis for the models with Macroeconomic Variables and Interest Rates as Inputs:

Table 11 illustrates the performance of different models when adding Macroeconomic Variables. More specifically, Table 11(a) shows the out-of-sample MSE and R^2_{oos} for the linear penalized Linear Regressions. It can be seen that the majority of the models still perform poorly out-of-sample with negative R^2_{oos} across both maturities. It can be noticed though that Elastic Net substantially outperforms both Lasso and Ridge Regression with a R^2_{oos} that turns from negative to positive for the yield with three-month maturity.

Table 11(b) illustrates the out-of-sample MSE and R^2_{oos} when introducing non-linear models in the prediction with different number of hidden layers and non-linear activation functions as well as Random Forest Regression model. First of all, it can be seen that there is a general improvement in the predictive R^2_{oos} when introducing non-linear models. For the 3-month yield maturity, a Neural network with 4 hidden layers can improve the overall predictability with $R^2_{oos} = 0.00085$ while for the yield with 2-year maturity a Neural Network with 2 hidden layers gives the best result with $R^2_{oos} = 0.00022$ compared to the other network architectures. Surprisingly, the MSE for both targets is not improved compared to what is obtained for Elastic Net and Lasso models. It can be also noted that the number of hidden layers can affect crucially the performance of the model since for the case of 3-month yield maturity, as the number of hidden layers increases, we move from negative to positive R^2_{oos}.

Surprisingly, Random Forest improved its performance substantially over the larger dataset for the 2-year yield maturity by turning its R^2_{oos} from negative to positive with value 0.00165 and in fact, it outperforms Neural Networks.

Table 11(c) represents the results obtained by applying Variable Selection (Lasso) and Dimensional Reduction techniques (PCA). Surprising, by first introducing variable selection to reduce the number of predictors that act as inputs in the input layer of ANN, the model performance significantly improves. As it can be seen, the R^2_{oos} increased to 0.00310 for the 3-month yield maturity while for the 2-year yield, its R^2_{oos} increased to 0.00180. In comparison with Lasso, when introducing the Principal Components as inputs in the input layer of ANN, there was no improvement in the model performance.

Table 10(a) and 11(a) show that even though PLS is a more complex regression, it massively underperforms out-of-sample as proven by the large and negative R^2_{oos} .

Surprisingly, even though there was an improvement in model performances when including macroeconomic variables, the difference was relatively small. One reason for low improvement is the signal-to-noise ratio which will always be pulled towards zero because financial markets are extremely noisy [Bryan Kelly and Moskowitz].

			Mean Squared Prediction Error		R^2_{oos}	
Panel A: Penalized Linear Regressions	Optimal alpha 3-Month	Optimal alpha 2-Year	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
Lasso	0.0002	0.0006	0.00104	0.00086	-0.00039	-0.00252
Elastic Net	0.00040	0.0011	0.00104	0.00086	**0.00028**	-0.00285
Ridge	0.00490	8265.2	0.03208	0.00087	-0.00121	-0.00525
PLS Regression	11 components	2 components	0.001	0.00080	-0.09631	-0.10471

(a)

	Mean Squared Prediction Error		R^2_{oos}	
Panel B: Neural Networks and Random Forest	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
Random Forest	0.0028	0.0009	-1.6603	**0.00165**
ANN 3 layers	0.0010	0.0009	**0.00085**	0.00005
ANN 4 layers	0.0012	0.0009	-0.16214	0.00005
ANN 2 layers	0.0013	0.0009	-0.25315	**0.00022**

(b)

	Mean Squared Prediction Error		R^2_{oos}	
Panel C: Neural Networks	rx^{3m}_{t+1}	rx^{2y}_{t+1}	rx^{3m}_{t+1}	rx^{2y}_{t+1}
Lasso – ANN – 2 layers	0.001	0.0009	0.0018	**0.00180**
Lasso – ANN – 3 layers	0.001	0.0009	**0.00311**	0.00005
Lasso – ANN – 5 layers	0.001	0.0009	0.00108	0.00001
Lasso – ANN – 4 layers	0.001	0.0009	0.00022	0.00001
ANN 4 layers + PCA (7 Components)	0.001	0.0009	0.00087	0.00025
ANN 2 layers + PCA (7 Components)	0.001	0.0009	-0.00105	0.00022

(c)

Figure 11: Out-of-Sample Results with Macroeconomic Variables and Interest Rates.

5.3.2 Classification Results

Table 12 presents the accuracy obtained by Neural Network models with both interest rates and macroeconomic variables as input data in the input layer. It can be seen that out of the 6 models, only the 2-layer ANN model and the 4-layer ANN model (with dimension reduction) for the 2-year maturity target managed to outperform random guessing with 51.50% and 53.81% accuracy respectively.

Table 13 illustrates the accuracy obtained by Neural Network models which take as input the percentage changes in yields together with their lagged values. Surprisingly, both the 2-layer and 4-layer models outperform random guessing across all maturities with maximum accuracy 60.01% for the 3-month maturity target and 58.37% accuracy for the 2-year maturity target. The above results demonstrate that the two deep learning architectures are able to extract valuable information from the past yield percentage changes. Surprisingly, by introducing PCA technique, the accuracy of the models is reduced quite significantly.

	Accuracy (%)	
Panel A: Neural Networks (2 classes)	rx_{t+1}^{3m}	rx_{t+1}^{2y}
ANN 4 layers	50.49	49.40
ANN 2 layers	49.40	51.50

(a)

	Accuracy (%)	
Panel B: Neural Networks(Dimension reduction): 2 classes	rx_{t+1}^{3m}	rx_{t+1}^{2y}
ANN – Lasso 4 layers	50.00	48.01
ANN - Lasso 2 layers	49.40	50.48
ANN 4 layers + PCA (7 Components)	49.40	**53.81**
ANN 2 layers + PCA (7 Components)	49.40	50.48

(b)

Figure 12: Out-of-Sample Classification Results with Macroeconomic Variables and Interest Rates.

Panel A: Neural Networks (2 classes)	Accuracy (%)	
	rx_{t+1}^{3m}	rx_{t+1}^{2y}
ANN 4 layers	**58.37**	**60.01**
ANN 2 layers	55.02	56.80

(a)

Panel B: Neural Networks (2 classes)	Accuracy (%)	
	rx_{t+1}^{3m}	rx_{t+1}^{2y}
ANN 4 layers + PCA (12 components)	50.94	50.63
ANN 2 layers + PCA (12 components)	52.82	50.73

(b)

Figure 13: Out-of-Sample Classification Results with only Interest Rates.

6 Conclusion and Future Recommendations

From the discussion above, it can be concluded that the variable importance of Lasso and Neural Networks differ quite significantly with Neural Network selecting from a wider number of categories. In addition, this paper illustrates that non-linear models (especially Neural Networks) can lead to more accurate out-of-sample predictions since they significantly outperform the linear-regression models. Moreover, substantial gains are possible by adding macro-economic information which are not relevant to the term structure of interest rates. To add on that, it is also shown that by introducing the important variables of Lasso as inputs in the input layer of Neural Network, this can offer some improvement in the predictive power of Neural Networks while data compression through PCA technique had no effect on the final outcome. Moreover, predicting the sign of interest rate percentage changes with only the term structure of interest rates was enough to outperform random guessing in a 2-class problem.

In summary, it is shown that multi-layer Neural Networks can provide promising results for the prediction of interest rate percentage changes. With respect to future work, one interesting direction will be to further explore multi-layer Neural Networks with perhaps small number of input variables since given the fact that the full dataset is consisted of 128 variables, models with such input dataset contain low noise-to-signal ratio. To add on that, recent advances in computing power allow the exploration of predictive power in more advanced deep learning models using High Frequency data.

References

M. D. Bauer. Bridging the Gap: Forecasting Interest Rates with Macro Trends. *FRBSF Economic Letter*, 2017. pages 3

B. M. Bianchi, D. and A. Tamoni. *Bond Risk Premia with Machine Learning*. 2019. pages 4

Bloomberg. Bloomberg professional database, 2017. Subscription Service. pages 8

R. I. Bryan Kelly and T. Moskowitz. Can Machines Learn Finance? *Working Paper*, page 18. pages 15

S. Ganguli and J. Dunnmon. Machine Learning for Better Models for Predicting Bond Prices. *arXiv e-prints*, Mar 2017. pages 3

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. pages 5

S. Gu, B. T. Kelly, and D. Xiu. Empirical Asset Pricing via Machine Learning. Swiss Finance Institute Research Paper Series 18-71, Swiss Finance Institute, Nov. 2018. URL `https://ideas.repec.org/p/chf/rpseri/rp1871.html`. pages 3, 9

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370. pages 4, 7

S. C. Ludvigson and S. Ng. Macro Factors in Bond Risk Premia. *The Review of Financial Studies*, 22(12):5027–5067, 10 2009. pages 8

T. Masters. Practical neural network recipes in c++, 1993. ISSN 0124790402. pages 5

M. Nunes, E. Gerding, F. Mcgroarty, and M. Niranjan. A comparison of multitask and single task learning with artificial neural networks for yield curve forecasting. *Expert Systems with Applications*, 119, 11 2018. pages 3

J. H. Stock and M. W. Watson. Macroeconomic forecasting using diffusion indexes. *Journal of Business & Economic Statistics*, 20(2):147–162, 2002. pages 8

7 Appendix

7.1 Ridge, Lasso and Elastic Net

The objective function of Lasso, Ridge and Elastic Net can be expressed as

$$L(\theta, ;) = L(\theta) + E(\theta, ;) \tag{1}$$

where $E(\theta, ;)$ represents the penalty terms which have the form:

$$\begin{cases} \lambda \sum_{j=1}^{p} \beta_j^2 & \text{Ridge Regression} \\ \lambda \sum_{j=1}^{p} |\beta_j| & \text{Lasso} \\ \mu\lambda \sum_{j=1}^{p} \beta_j^2 + \frac{\lambda(1-\mu)}{2} \sum_{j=1}^{p} |\beta_j| & \text{Elastic Net} \end{cases}$$

Full Dataset:

Groups	Subgroup	Ticker	Name
Interest Rates			
Rates	ON	EE00O/N_Index	EUR Libor Overnight
	1M	EE0001M_Index	EUR Libor 1 month
	3M	EE0003M_Index	EUR Libor 3 month
	6M	EE0006M_Index	EUR Libor 6 month
	12M	EE0012M_Index	EUR Libor 12 month
FED	Fed Funds	FDTR	Federal Funds Target Rate
	90day Euro$ Fut	ED1_Comdty	Generic 1st 'ED' Future
		ED2_Comdty	Generic 2nd 'ED' Future
		ED3_Comdty	Generic 3rd 'ED' Future
		ED4_Comdty	Generic 4th 'ED' Future
		ED5_Comdty	Generic 5th 'ED' Future
ECB	ECB refi rate		European Central Bank Refi-nancing Rate
		ER1_Comdty	Generic 1st 'ER' Future
		ER2_Comdty	Generic 2nd 'ER' Future
		ER3_Comdty	Generic 3rd 'ER' Future
		ER4_Comdty	Generic 4th 'ER' Future
		ER5_Comdty	Generic 5th 'ER' Future
Yields			
Europe	3M	GGEU3M	Euro Generic Govt 3 Month Yield
	2Y	GGEU2Y	Euro Generic Govt 2 Year Yield
	5Y	GGEU5Y	Euro Generic Govt 5 Year Yield
	10Y	GGEU10Y	Euro Generic Govt 10 Year Yield
	20Y	GECU20YR	Euro Generic Govt 20 Year Yield
	30Y	GGEU30Y	Euro Generic Govt 30 Year Yield
US	3M	USGG3M_Index	US Generic Govt 3 Month Yield

	2Y	USGG2YR_Index	US Generic Govt 2 Year Yield
	5Y	USGG5YR_Index	US Generic Govt 5 Year Yield
	10Y	USGG10YR_Index	US Generic Govt 10 Year Yield
	30Y	USGG30YR_Index'	US Generic Govt 30 Year Yield
Japan	3M	GGJP3M	JP Generic Govt 3 Month Yield
	2Y	GGJP2Y	JP Generic Govt 2 Year Yield
	5Y	GGJP5Y	JP Generic Govt 5 Year Yield
	10Y	GGJP10Y	JP Generic Govt 10 Year Yield
	30Y	GGJP30Y	JP Generic Govt 30 Year Yield
UK	3M	GGUK3M	UK Generic Govt 3 Month Yield
	2Y	GGUK2Y	UK Generic Govt 2 Year Yield
	5Y	GGUK5Y	UK Generic Govt 5 Year Yield
	10Y	GGUK10Y	UK Generic Govt 10 Year Yield
	30Y	GGUK30Y	UK Generic Govt 30 Year Yield
Equity			
		INDU	Dow Jones Industrial Aver-age
		SPX	S&P 500 Index
		CCMP	NASDAQ Composite Index
		SX5E	Euro Stoxx 50
		SXXP	Stoxx Europe 600
		SX7E	Euro Stoxx Banks
		UKX	FTSE 100 Index
		TPX	TOPIX Index (Tokyo)
		NKY	NIKKEI 225
		HSI	HANG SENG Index
Currencies			
		EURUSD_Curncy	EUR-USD X-Rate
		EURJPY_Curncy	EUR-JPY X-Rate

	EURGBP_Curncy	EUR-GBP X-Rate
	EURCHF_Curncy	EUR-CHF X-Rate
Economic Indicators		
Global		
	MEPRGLEI_Index	Global 0ECD – Leading Economic Indicator
US	GDP_CQOQ_Index	US Chained 2009 Dollars
	CPI_XYOY_Index	US CPI Urban Consumers Less F&E
	NAPMPMI_Index	ISM Manufacturing PMI
	IP_CHNG_Index	US Industrial Production
	CPTICHNG_Index	US Capacity Utilization %
	USURTOT_Index	U-3 US Unemployment Rate Total
	NFP_TCH_Index	US Employees on Nonfarm Payroll
	INJCJC_INdex	US Initial Jobless Claims SA
	LEI_CHNG_Index	Conference Board US Leading Index
	OEUSKLAC_Index	USA OECD Leading Indicators CLI Amplitude Adjusted SA
Europe		
	EUGNEMUY_Index	Euro Area Gross Domestic Product
	CPEXEMUY_Index	Eurostat Eurozone Core MUICP
	EUIPEMUY_Index	Eurostat Industrial Production
	EUUCEMU_Index	Europ. Commission Capacity Utilization
	UMRTEMU_Index	Eurostat Unemployment Eurozone
	ECMSM1Y_Index	1Y ECM Money Supply
	ECMSM2Y_Index	2Y ECM Money Supply
	ECMSM3YY_Index	3Y ECM Money Supply
China		
	OECNKLAC_Index	China OECD Leading Indicators CLI Amplitude Adjusted SA

Volatility			
		VIX_Index	CBOE SPX Volatility Index
		V2X_Index	VSTOXX Index
		VXN_Index	CBOE Nasdaq Market Volatility Index
Commodities			
		CO1_comdty	Crude Oil Future
		NG1_Comdty	NAT GAS
		GC1_Comdty	GOLD
		SI1_Comdty	SILVER
		HG1_Comdty	Copper Futures
		CRY_Index	Commodity Research Bureau
Bond Futures			
		DU1_Comdty	Generic 1st ‘DU’ Future
		OE1_Comdty	Generic 1st ‘OE’ Future
		RX1_Comdty	Generic 1st ‘RX’ Future
		UB1_Comdty	Generic 1st ‘UB’ Future
		CL1_Comdty	Generic 1st 'CL' Future
		C_1_Comdty	Generic 1st 'C ' Future
Swap Rates			
Europe			
	2 Y	EUSA2_Curncy	EUR Swap rate Annual 2Y
	5 Y	EUSA5_Curncy	EUR Swap rate Annual 5Y
	10 Y	EUSA10_Curncy	EUR Swap rate Annual 10Y
	30 Y	EUSA30_Curncy	EUR Swap rate Annual 30Y
US			
	2 Y	USSW2_Curncy	USD Swap Rate Semi-Annual 2Y
	5 Y	USSW5_Curncy	USD Swap Rate Semi-Annual 5Y
	10 Y	USSW10_Curncy	USD Swap Rate Semi- Annual 10Y
	30 Y	USSW30_Curncy	USD Swap Rate Semi-Annual 30Y

European Balance Sheets			
		EBBSTOTA_Index	ECB Balance Sheet All Asets
		EBBSDEPF_Index	ECB Balance Sheet Deposit Facility
		EBBSLONG_Index	ECB Balance Sheet Long Term Refinancing Operations
		EBBSA050_Index	ECB Balance Sheet Lending to Euro Area Credit Institutions Denominated in EUR
Government Bond Spreads			
Inter-markets	2Y	US-EU2Y_Spread	US-Europe Govt 2 Year Spread
		EU-JP2Y_Spread	Europe-Japan Govt 2 Year Spread
		US-JP2Y_Spread	US-Japan Govt 2 Year Spread
		US-UK2Y_Spread	US-UK Govt 2 Year
	10Y	EU-UK10_Spread	US-Europe Govt 10 Year Spread
		EU-JP10Y_Spread	Europe-Japan Govt 10 Year Spread
		US-JP10Y_Spread	US-Japan Govt 10 Year Spread
		EU-UK10Y_Spread	EU-UK Govt 10 Year
Intra-market			
	03M10Y	EU3M-10Y_Spread	Europe Govt 3 M-10 Year Spread
		US3M-10Y_Spread	US Govt 3 M-10 Year Spread
	2Y10Y	EU2-10Y_Spread	Europe Govt 2 Y -10 Year Spread
		US2-10Y_Spread	US Govt 2 Y-10 Year Spread
	10Y30Y	EU10-30Y_Spread	Europe Govt 10 Y-30 Year Spread
		US10-30Y_Spread	US Govt 10 Y-30 Year Spread
Lagged Returns			
		Calculated	Past 3 lagged returns of Euro Generic Govt 3 Month Yield
		Calculated	Past 3 lagged returns of Euro Generic Govt 5 Month Yield
		Calculated	Past 3 lagged returns of Euro Generic Govt 2 Year Yield
		Calculated	Past 3 lagged returns of Euro Generic Govt 5 Year Yield

	Calculated	Past 3 lagged returns of Euro Generic Govt 10 Year Yield
	Calculated	Past 3 lagged returns of Euro Generic Govt 20 Year Yield
	Calculated	Past 3 lagged returns of Euro Generic Govt 30 Year Yield

Coding:

```
## Importing Libraries

from keras.models import Sequential
from keras.layers import  Dense
import pandas as pd, numpy as np, datetime, math, warnings, random
import matplotlib.pyplot as plt
from tqdm import tqdm
from itertools import product
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error as mse, r2_score
from keras.regularizers import l1, l2 ,l1_l2
from keras.callbacks import EarlyStopping
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
import itertools
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.cross_decomposition import PLSRegression
from sklearn.decomposition import PCA

# Insert Data #

raw_data = pd.read_csv('DATA_new.csv', low_memory=False) # Import data
raw_data.head() # Preview data

raw_data.columns = raw_data.columns.str.replace(' ', '_')

raw_data['Dates'] = pd.to_datetime(raw_data["Dates"]).dt.strftime('%Y-%m-%d')

raw_data = raw_data.set_index('Dates')

data = raw_data.dropna() # Drop rows with NaN

data.head()

# Collect the EURO yields data #

df = raw_data[['GECU3M_Index',
 'GECU2YR_Index',
 'GECU5YR_Index',
```

```
 'GECU10YR_Index',
 'GECU20YR_Index',
 'GECU30YR_Index']]


### ADD LAGGED VALUES OF THE TARGETS ###

columns = data.columns.tolist()

for i in df:

  data.insert(# Add 1 day past rate,
              loc = len(data.columns),
              column = str(i) + '_LAG1',
              value = df[i].shift(1))

  data.insert(# Add 2 day past rate,
              loc = len(data.columns),
              column = str(i) +'_LAG2',
             value = df[i].shift(2))

  data.insert(# Add 3 day past rate,
             loc = len(data.columns),
              column = str(i)+'_LAG3',
              value = df[i].shift(3))

data = data.replace([np.inf, -np.inf], np.nan)
data = data.dropna() # Drop rows with NaN

# Create percentage changes of all yields and interest rates columns

list1 = (data.columns).tolist()
L=[]
for i in list1:
  if any((data[i] < 30) == True):
    data[i] = data[i].pct_change()
  else:
    L.append(i)

data = data.replace([np.inf, -np.inf], np.nan)
data = data.dropna() # Drop rows with NaN

#Split data into train, validation and test set

nrow = data.index.shape[0] # Get number of row
dates = data.index.unique()
train = data[data.index < dates[math.floor(dates.size*0.4)]] # First 40% of data -> train
validate = data[(data.index >= dates[math.floor(dates.size*0.4)]) &  # 40-80% of data -
> validate
                (data.index < dates[math.floor(dates.size*0.8)])]
test = data[(data.index>= dates[math.floor(dates.size*0.8)])] # 80-100% of data -> test

## standardize the data ##

       for i in L:
           train[i] = (train[i]-train[i].mean())/(train[i].std())
           validate[i] = (validate[i]-validate[i].mean())/(validate[i].std())
           test[i] = (test[i]-test[i].mean())/(test[i].std())



       # Creating a function that calculates the out of sample R^2

       def my_R_oos_sq(y_test, y_pred):
           Er_squared = (y_test - y_pred)**2

```

```
            SE = 0
            for i in range(len(Er_squared)):
                SE = SE + (float(Er_squared[i]))

            SR = 0
            for j in range(len(y_test)):
                SR = SR + (float(y_test[j]))**2

            R_oos_sq = 1 - (SE/SR)
            return R_oos_sq

        # use Lasso with GECU3M_Index respond variable

        # create a function to train Lasso model

        def trainLasso(alpha): # Define a function to train lasso regres-
    sion model on training data
            x_train = train.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_train = train['GECU3M_Index'] # Response variabl
            lassoreg = Lasso(alpha = alpha) # Initialise lasso model
            fit = lassoreg.fit(X = x_train, y = y_train)
            return fit

        # create a function to validate Lasso model with MSE

        def validateLassoMSE(alpha):
            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory variable
            y_validate = validate['GECU3M_Index'] # Response variable
            lassoreg = trainLasso(alpha) # Train model using specific alpha
            y_pred = lassoreg.predict(x_validate) # Use model to predict on valida-
    tion set
            return (mse(y_true = y_validate, y_pred = y_pred)) # Calculate and return mean-
    squared-error

        # create a function to validate Lasso model with R2

        def validateLassoR2(alpha):
            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory varia-
    bles
            y_validate = validate['GECU3M_Index'] # Response variable
            lassoreg = trainLasso(alpha) # Train model using specific alpha
            y_pred = lassoreg.predict(x_validate) # Use model to predict on valida-
    tion set
            return (my_R_oos_sq(y_validate, y_pred)) # Calculate and r-square

        # Choosing best parameter (alpha)

        fig = plt.figure()
        alpha = np.arange(0.0001, 0.1, 0.001) # Range of alpha (hyperparame-
    ter) to be tuned
        T = np.vectorize(validateLassoMSE)(alpha) # Validate and calculate MSE on valida-
    tion set
        plt.plot(alpha, T, '-', alpha, T, 'b') # Plot
        fig.suptitle('Optimal Alpha', fontsize=13)
        plt.axvline(x = 0.002,color ='r')
        plt.xlabel('alpha')
        plt.ylabel('MSE')
        plt.show()
        fig.savefig('{}Lasso_alpha.jpg'.format('Applied_Project'))


        opt_alpha = 0 # Variable to keep optimal alpha
        lasso_r2 = -10 # Variable to keep maximum r2
        for i in tqdm(np.arange(0.0001, 0.070, 0.001)):
            temp_r2 = validateLassoR2(i)
            if(temp_r2 > lasso_r2):
```

```
                lasso_r2 = temp_r2
                opt_alpha = i

        print("Optimal alpha: " + str(round(opt_alpha, 5)))

        ##Lasoo Variable Importance

        lassoreg = trainLasso(alpha = opt_alpha) # Get model for best alpha (alpha = 1.0)
        coef = pd.DataFrame({"Variable": train.drop(['GECU3M_Index' ], axis = 1).columns, "Coefficient" : lassoreg.coef_}) # Coef. dataframe


        features = coef[coef['Coefficient'] != 0] # Store list of important variables
        variables = features['Variable'].values.tolist()
        variables.insert(0,'GECU3M_Index')
        variables

        #Performance on Testing set

        #create functions to test the performance of the model on test set

        def testLassoMSE(alpha):
            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_test = test['GECU3M_Index'] # Response variable
            lassoreg = trainLasso(alpha) # Train model using specific alpha
            y_pred = lassoreg.predict(x_test) # Use model to predict on test set
            return (mse(y_true = y_test, y_pred = y_pred)) # Calculate and return mean-squared-error

        def testLassoR2(alpha):
            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_test = test['GECU3M_Index'] # Response variable
            lassoreg = trainLasso(alpha) # Train model using specific alpha
            y_pred = lassoreg.predict(x_test) # Use model to predict on test set
            return ((my_R_oos_sq(y_test, y_pred))) # Calculate and r-square


        lasso_mse = testLassoMSE(opt_alpha)
        lasso_r2 = testLassoR2(opt_alpha)
        print('Maximum R-square: ' + str(round(lasso_r2, 5)) + " (alpha = " + str(opt_alpha) + ")")
        print('Minimum MSE: ' + str(round(lasso_mse, 5)) + " (alpha = " + str(opt_alpha) + ")")

        # Set up Neural Network


        ### training of Neural Networks ###

        def trainNN(train, nnmodel, arch, l, opt, init):
            x_train = train.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_train = train['GECU3M_Index'] # Response variable
            n_cols = x_train.shape[1] # Number of columns in training data

            # Setup model architecture
            # First layer
            nnmodel.add(Dense(arch[0],
                              activation = 'relu',
                              input_shape = (n_cols,),
                              activity_regularizer = l1(l),
                              init = init))
            # 2nd to pre-last layer
            for i in range(1, len(arch)-1):
                nnmodel.add(Dense(arch[i], activation = 'relu', activity_regularizer = l1(l)))
            # Last layer
```

```
            nnmodel.add(Dense(arch[len(arch)-1], activation = 'sigmoid', activity_regularizer = l1(l)))

            # Compile model using MSE as a measure of model performance
            nnmodel.compile(optimizer = opt, loss='mean_squared_error')

            # Set early stopping monitor so the model stops training when it won't improve anymore
            early_stopping_monitor = EarlyStopping(patience = 3)

            # Prepare validation set
            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_validate = validate['GECU3M_Index'] # Response variable

            # Train model
            nnmodel.fit(x_train,
                        y_train,
                        validation_split = 0.2,
                        epochs = 300,
                        callbacks = [early_stopping_monitor])

        #create function for the validation of the model

        def validateNN(validate, nnmodel):
            x_validate = np.asarray(validate.drop(['GECU3M_Index'], axis = 1)) # Explanatory variables
            y_validate = validate['GECU3M_Index'] # Response variable
            y_pred = [np.array(nnmodel.predict(x_validate))] # Use model to predict on testing set
            y_pred =  np.array(y_pred).ravel()
            return (mse(y_true = y_validate, y_pred = y_pred), (my_R_oos_sq(y_validate, y_pred))) # Calculate and return mean-squared-error and r-square

        # Define a function to train and validate model on given parameter

        def trainValidateNN(arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency
            nnmodel = Sequential() # Initiate sequential model
            trainNN(train, nnmodel, arch, l, opt, init) # Train it
            return validateNN(validate, nnmodel) # Validate it

        # Hyper-parameter Tuning

        #find the parameters which optimize the results on the validation set

        arch = [[64,32,16,8,1], [15,6,1],[4,2,1]]#,[32,8,4,1]] # Setup network architecture
        l = [0.001,0.0001, 0.00001] # Set l1 penalised value
        opt = ['adam','nadam', 'rmsprop'] # Set up optimization method
        init = ['glorot_uniform', 'normal', 'uniform'] # Set up init method

        resultNN = pd.DataFrame()
        resultNN = resultNN.append(pd.DataFrame({'ARCH' : 'Lasso',
                                                 'L1' : 'Lasso',
                                                 'OPT' : 'Lasso',
                                                 'init' : 'Lasso',
                                                 'mse' : [lasso_mse],
                                                 'r2' : [lasso_r2]}), ignore_index = True)
        for i in tqdm(itertools.product(arch, l, opt, init)):
            res_mse, res_r2 = trainValidateNN(i[0], i[1], i[2], i[3])
            resultNN = resultNN.append(pd.DataFrame({'ARCH' : [i[0]],
                                                     'L1' : i[1],
                                                     'OPT' : i[2],
                                                     'init' : i[3],
                                                     'mse' : [res_mse],
```

```
                                                'r2' : [res_r2]}),
                                    ignore_index = True)

        resultNN.to_csv('resultNN.csv', sep = ',', index = False)
        resultNN = pd.read_csv('resultNN.csv', sep = ',')
        resultNN.sort_values(by=['r2'], ascending = False)

        # Performance on Testing set

        merge_validate = pd.concat([train,validate])

        nnmodel = Sequential() # Initiate sequential model

        # Define a function to test model with specific parameters on testing set

        def testNN(test, arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency
            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_test = np.asarray(test['GECU3M_Index']) # Response variable
            trainNN(merge_validate, nnmodel, arch, l, opt, init) # Train it
            y_pred = [np.array(nnmodel.predict(x_test))] # Use model to predict on test-
    ing set
            y_pred =  np.array(y_pred).ravel()
            re-
    turn (mse(y_true = y_test, y_pred = y_pred), (my_R_oos_sq(y_test, y_pred)))

        test_mse_1, test_r2_1 = testNN(test, [64, 32, 16, 8, 1] ,1e-05, 'adam', 'nor-
    mal') # Test 1st best model on testing set
        test_mse_2, test_r2_2 = testNN(test,[4, 2, 1] ,0.001  ,'rmsprop'  ,'glorot_uni-
    form' ) # Test 2nd best model on testing set


        print("Test MSE (1st best model): " + str(round(test_mse_1, 4)) + " R2: " + str(rou
    nd(test_r2_1, 5)))
        print("Test MSE (2nd best model): " + str(round(test_mse_2, 4)) + " R2: " + str(rou
    nd(test_r2_2, 5)))


        # Neural Networks Permutation Feature Importance Measurement

        variable_list = data.drop(['GECU3M_Index'],axis=1).columns.values
        benchmark_r2 = resultNN.sort_values(by=['r2'], ascend-
    ing = False).iloc[0,5] # Get r2 benchmark
        print("Benchmark R-squared: " + str(round(benchmark_r2, 5)))

        # Define a function to train and validate model on given parameter

        def trainValidateSenNN(trainSen, validateSen, arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency
            nnmodel = Sequential() # Initiate sequential model
            trainNN(trainSen, nnmodel, arch, l, opt, init) # Train it
            return validateNN(validateSen, nnmodel) # Validate it

        var_imp = pd.DataFrame()
        for i in tqdm(variable_list):
            trainSen = train.drop([i], axis = 1) # Drop single feature from training set
            validateSen = validate.drop([i], axis = 1) # Drop single feature from validat-
    ing set
            res_mse, res_r2 = trainValidateSenNN(trainSen, vali-
    dateSen, [15, 6, 1], 0.0001, 'rmsprop', 'glorot_uniform') # Train and get perfor-
    mance on validation set
            var_imp = var_imp.append(pd.DataFrame({'Variable' : i,
                                                   'R2_Diff' : [benchmark_r2 - res_r2]}), ig-
    nore_index = True)
        [ ]
        var_imp.to_csv('sensitivity.csv', sep = ',', index = False)
```

```
        var_imp = pd.read_csv('sensitivity.csv', sep = ',')
        var_imp.sort_values(by=['R2_Diff'], ascending = False)

        # Feature Selection with Lasso as input in Neural Networks

        train_Lasso = train[variables]
        validate_Lasso = validate[variables]
        test_Lasso = test[variables]

        # Define a function to train and validate model on given parameter

        def trainValidateLassoNN(arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency
            nnmodel = Sequential() # Initiate sequential model
            trainNN(train_Lasso, nnmodel, arch, l, opt, init) # Train it
            return validateNN(validate_Lasso, nnmodel) # Validate it


        arch = [[15, 6, 1],[64,32,16,8,4,1],[32, 16,8, 4, 1]] # Setup network architec-
    ture
        l = [0.0001, 0.00001,0.000001] # Set l1 penalised value
        opt = ['adam','nadam', 'rmsprop'] # Set up optimization method
        init = ['glorot_uniform', 'normal', 'uniform'] # Set up init method
        resultLassoNN = pd.DataFrame()
        resultLassoNN = resultLassoNN.append(pd.DataFrame({'ARCH' : 'Lasso',
                                                'L1' : 'Lasso',
                                                'OPT' : 'Lasso',
                                                'init' : 'Lasso',
                                                'mse' : [lasso_mse],
                                                'r2' : [lasso_r2]}), ignore_index = True)
        for i in tqdm(itertools.product(arch, l, opt, init)):
            res_mse, res_r2 = trainValidateLassoNN(i[0], i[1], i[2], i[3])
            resultLassoNN = resultLassoNN.append(pd.DataFrame({'ARCH' : [i[0]],
                                                    'L1' : i[1],
                                                    'OPT' : i[2],
                                                    'init' : i[3],
                                                    'mse' : [res_mse],
                                                    'r2' : [res_r2]}),
                                        ignore_index = True)
        # Hyperparameter Tuning

        resultLassoNN.to_csv('resultLassoNN.csv', sep = ',', index = False)
        resultLassoNN = pd.read_csv('resultLassoNN.csv', sep = ',')
        resultLassoNN.sort_values(by=['r2'], ascending = False)

        merge_val = pd.concat([train_Lasso,validate_Lasso])

        #Performance on Testing set

        # Define a function to test model with specific parameters on testing set

        def testLassoNN(test_Lasso, arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency0
            x_test = test_Lasso.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_test = np.asarray(test_Lasso['GECU3M_Index'])# Response variable
            nnmodel = Sequential() # Initiate sequential model
            trainNN(merge_val, nnmodel, arch, l, opt, init) # Train it
            y_pred = [np.array(nnmodel.predict(x_test))] # Use model to predict on test-
    ing set
            y_pred =  np.array(y_pred).ravel()
            return (mse(y_true = y_test, y_pred = y_pred), my_R_oos_sq(y_test, y_pred))

        test_lasso_mse_1, test_lasso_r2_1 = testLas-
    soNN(test_Lasso,[64, 32, 16, 8, 4, 1]  ,1e-05  ,'nadam', 'glorot_uni-
    form'  ) # Test 1st best model on testing set
```

```
        test_lasso_mse_2, test_lasso_r2_2 = testLasoNN(test_Lasso,[64, 32, 16, 8, 4, 1]  ,1e-05, 'adam', 'uniform' ) # Test 2nd best model on testing set


        print("Test MSE (1st best model): " + str(round(test_lasso_mse_1, 4)) + " R2: " + str(round(test_lasso_r2_1, 5)))
        print("Test MSE: (2nd best model)" + str(round(test_lasso_mse_2, 4)) + " R2: " + str(round(test_lasso_r2_2, 5)))

        # Classification Problem with full dataset

        # Define target values/classes

        targets = ['GECU3M_Index']

        num_classes = 2
        model_targets_train = pd.DataFrame(data = train['GECU3M_Index'].values, index = train.index, columns = targets)
        model_targets_validate = pd.DataFrame(data = validate['GECU3M_Index'].values, index = validate.index, columns = targets)
        model_targets_test = pd.DataFrame(data = test['GECU3M_Index'].values, index = test.index, columns = targets)


        def number_classes(L):
          return L


        if number_classes(num_classes) == 2:
          model_targets_train[model_targets_train['GECU3M_Index'] <= 0] = -1
          model_targets_train[model_targets_train['GECU3M_Index'] >= 0] = 0

          model_targets_validate[model_targets_validate['GECU3M_Index'] <= 0] = -1
          model_targets_validate[model_targets_validate['GECU3M_Index'] >= 0] = 0

          model_targets_test[model_targets_test['GECU3M_Index'] <= 0] = -1
          model_targets_test[model_targets_test['GECU3M_Index'] >= 0] = 0


        model_targets_train['GECU3M_Index'] = model_targets_train['GECU3M_Index'].astype(int)
        model_targets_validate['GECU3M_Index'] = model_targets_validate['GECU3M_Index'].astype(int)
        model_targets_test['GECU3M_Index'] = model_targets_test['GECU3M_Index'].astype(int)


        y_train = pd.DataFrame(to_categorical(model_targets_train-model_targets_train.min(),number_classes(num_classes)).astype(int))
        y_test = pd.DataFrame( to_categorical(model_targets_test-model_targets_test.min(),number_classes(num_classes)).astype(int))
        y_validate = pd.DataFrame( to_categorical(model_targets_validate - model_targets_validate.min(),number_classes(num_classes)).astype(int))


        ### Neural Networks ###

        warnings.filterwarnings('ignore') #ignore warning
        def classtrainNN(train,y_train, nnmodel, arch, l, opt, init):
            x_train = train.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            n_cols = x_train.shape[1] # Number of columns in training data

            # Setup model architecture
```

```
            # First layer
            nnmodel.add(Dense(arch[0],
                              activation = 'relu',
                              input_shape = (n_cols,),
                              activity_regularizer = l1(l),
                              init = init))
            # 2nd to pre-last layer
            for i in range(1, len(arch)-1):
                nnmodel.add(Dense(arch[i], activation = 'relu', activity_regular-
    izer = l1(l)))
            # Last layer
            nnmodel.add(Dense(number_classes(num_classes), activation = 'sigmoid', activ-
    ity_regularizer = l1(l)))

            # Compile model using MSE as a measure of model performance
            nnmodel.compile(optimizer = opt, loss='categorical_crossentropy', metrics=['ac-
    curacy'])

            # Set early stopping monitor so the model stops training when it won't im-
    prove anymore
            early_stopping_monitor = EarlyStopping(patience = 3)

            # Prepare validation set
            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory varia-
    bles


            # Train model
            nnmodel.fit(x_train,
                        y_train,
                        validation_split = 0.2,
                        epochs = 300,
                        callbacks = [early_stopping_monitor])


        def multi_class(L):
          for i in range(L.shape[0]):
            for j in range(L.shape[1]):
              if L[i,j] == L[i,:].max():
                L[i,j] = 1
              else:
                L[i,j] = 0
          return L
        [ ]
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        import matplotlib.pyplot as plt
        import seaborn as sns
        from mlxtend.evaluate import confusion_matrix
        from mlxtend.plotting import plot_confusion_matrix


        def classvalidateNN(validate,y_validate, nnmodel):
            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory varia-
    bles
            y_pred = nnmodel.predict(x_validate) # Use model to predict on validation set
            y_pred = pd.DataFrame(multi_class(y_pred).astype(int))
            y_pred
            acc = accuracy_score(y_validate, y_pred)
            cm = confusion_matrix(y_validate.as_matrix().argmax(axis=1), y_pred.as_ma-
    trix().argmax(axis=1),binary=False)
            fig, ax = plot_confusion_matrix(conf_mat=cm)
            plt.show()

            return acc # Calculate and return mean-squared-error and r-square

```

```
        # Define a function to train and validate model on given parameter

        def classtrainValidateNN(arch, l, opt, init):
            random.seed(a = 0) # Set seed for consistency
            nnmodel = Sequential() # Initiate sequential model
            classtrainNN(train,y_train, nnmodel, arch, l, opt, init) # Train it
            return classvalidateNN(validate,y_validate, nnmodel) # Vali-
    date it

        arch = [[15,6,1],[4,2,1]]#,[64,32,16,8,4,1]] # Setup network architecture
        l = [0.001,0.0001, 0.00001] # Set l1 penalised value
        opt = ['adam','nadam', 'rmsprop'] # Set up optimization method
        init = ['glorot_uniform', 'normal', 'uniform'] # Set up init method

        resultNN = pd.DataFrame()
        resultNN = resultNN.append(pd.DataFrame({'ARCH' : 'Lasso',
                                                 'L1' : 'Lasso',
                                                 'OPT' : 'Lasso',
                                                 'init' : 'Lasso',
                                                 'mse' : [lasso_mse],
                                                 'r2' : [lasso_r2],
                                                 'acc': 0}), ignore_index = True)
        for i in tqdm(itertools.product(arch, l, opt, init)):
            acc = classtrainValidateNN(i[0], i[1], i[2], i[3])
            resultNN = resultNN.append(pd.DataFrame({'ARCH' : [i[0]],
                                                     'L1' : i[1],
                                                     'OPT' : i[2],
                                                     'init' : i[3],
                                                     'mse' : 0,
                                                     'r2' : 0,
                                                     'acc' : [acc]}),
                                       ignore_index = True)

        resultNN.to_csv('resultNN_class2.csv', sep = ',', index = False)
        resultNN_class2 = pd.read_csv('resultNN_class2.csv', sep = ',')
        resultNN_class2.sort_values(by=['acc'], ascending = False).head()

        nnmodel = Sequential() # Initiate sequential model
        def classtestNN(test,y_test, arch, l, opt, init): # Define a func-
    tion to test model with specific parameters on testing set
            random.seed(a = 9) # Set seed for consistency
            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            classtrainNN(validate,y_validate, nnmodel, arch, l, opt, init) # Train it
            y_pred = nnmodel.predict(x_test) # Use model to predict on testing set
            y_pred = pd.DataFrame(multi_class(y_pred).astype(int))
            acc = accuracy_score(y_test, y_pred)
            cm = confusion_matrix(y_test.as_matrix().argmax(axis=1), y_pred.as_ma-
    trix().argmax(axis=1),binary=False)
            fig, ax = plot_confusion_matrix(conf_mat=cm)
            plt.show()

            return acc # Calculate and return mean-squared-error and r-square

        # class 2 test full model

        acc_1 = classtestNN(test,y_test,[64, 32, 16, 8, 4, 1,], 0.001,  'adam', 'glo-
    rot_uniform') # Test 1st best model on testing set
        acc_2 = classtestNN(test,y_test,[64, 32, 16, 8, 4, 1],  0.001,  'adam'  ,'nor-
    mal' ) # Test 2nd best model on testing set

        print("Test Acc (1st best model): " + str(round(acc_1, 4)))
        print("Test Acc (2nd best model): " + str(round(acc_2, 4)))

        # Classification using Lasso variables

        train_Lasso = train[variables]
```

```
        validate_Lasso = validate[variables]
        test_Lasso = test[variables]

        def classvalidateLassoNN(validate_Lasso,y_validate, nnmodel):
            x_validate = validate_Lasso.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_pred = nnmodel.predict(x_validate) # Use model to predict on validation set
            y_pred = pd.DataFrame(multi_class(y_pred).astype(int))
            acc = accuracy_score(y_validate, y_pred)
            cm = confusion_matrix(y_validate.as_matrix().argmax(axis=1), y_pred.as_matrix().argmax(axis=1),binary=False)
            fig, ax = plot_confusion_matrix(conf_mat=cm)
            plt.show()

            return acc # Calculate and return mean-squared-error and r-square

        # Define a function to train and validate model on given parameter

        def classtrainValidateLassoNN(arch, l, opt, init):
            random.seed(a = 0) # Set seed for consistency
            nnmodel = Sequential() # Initiate sequential model
            classtrainNN(train_Lasso,y_train, nnmodel, arch, l, opt, init) # Train it
            return classvalidateNN(validate_Lasso,y_validate, nnmodel) # Validate it

        arch = [[64,32,16,8,4,1],[32,16,8,4,1], [15,6,1]] # Setup network architecture
        l = [0.001,0.0001, 0.00001] # Set l1 penalised value
        opt = ['adam','nadam', 'rmsprop'] # Set up optimization method
        init = ['glorot_uniform', 'normal', 'uniform'] # Set up init method

        resultNN = pd.DataFrame()
        resultNN = resultNN.append(pd.DataFrame({'ARCH' : 'Lasso',
                                                 'L1' : 'Lasso',
                                                 'OPT' : 'Lasso',
                                                 'init' : 'Lasso',
                                                 'mse' : [lasso_mse],
                                                 'r2' : [lasso_r2],
                                                 'acc': 0}), ignore_index = True)
        for i in tqdm(itertools.product(arch, l, opt, init)):
            acc = classtrainValidateLassoNN(i[0], i[1], i[2], i[3])
            resultNN = resultNN.append(pd.DataFrame({'ARCH' : [i[0]],
                                                     'L1' : i[1],
                                                     'OPT' : i[2],
                                                     'init' : i[3],
                                                     'mse' : 0,
                                                     'r2' : 0,
                                                     'acc' : [acc]}),
                                       ignore_index = True)

        resultNN.to_csv('resultLassoNN_class2.csv', sep = ',', index = False)
        resultNN_class2 = pd.read_csv('resultLassoNN_class2.csv', sep = ',')
        resultNN_class2.sort_values(by=['acc'], ascending = False).head()


        merge_val = pd.concat([train_Lasso,validate_Lasso])

        nnmodel = Sequential() # Initiate sequential model
        # Define a function to test model with specific parameters on testing set

        def classtest_LassoNN(test,y_test, arch, l, opt, init):
            random.seed(a = 9) # Set seed for consistency
            x_test = test_Lasso.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            classtrainNN(merge_val,y_val, nnmodel, arch, l, opt, init) # Train it
            y_pred = nnmodel.predict(x_test) # Use model to predict on testing set
            y_pred = pd.DataFrame(multi_class(y_pred).astype(int))
            acc = accuracy_score(y_test, y_pred)
```

```
644.            cm = confusion_matrix(y_test.as_matrix().argmax(axis=1), y_pred.as_ma-
    trix().argmax(axis=1),binary=False)
645.            fig, ax = plot_confusion_matrix(conf_mat=cm)
646.            plt.show()
647.
648.            return acc # Calculate and return mean-squared-error and r-square
649.        # Class 2 ANN lasso results
650.        acc_1 = classtest_LassoNN(test_Lasso,y_test,[4, 2, 1] , 1e-05 , 'nadam','nor-
    mal') # Test 2nd best model on testing set
651.        acc_2 = classtest_LassoNN(test_Lasso,y_test,[15,6,1], 1e-05, 'nadam','normal')
652.
653.        print("Test Acc (1st best model): " + str(round(acc_1, 4)))
654.        print("Test Acc (2nd best model): " + str(round(acc_2, 4)))
655.
656.
657.        # PLS Regression MODEL
658.
659.        def trainPLS(n_components): # Define a function to train lasso regres-
    sion model on training data
660.            x_train = train.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
661.            y_train = train['GECU3M_Index'] # Response variabl
662.            PLSreg = PLSRegression(n_components) # Initialise lasso model
663.            fit = PLSreg.fit(scale(x_train),y_train)
664.            return fit # Train model on training data
665.
666.        def validatePLSMSE(n_components):
667.            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory variable
668.            y_validate = validate['GECU3M_Index'] # Response variable
669.            PLSreg = trainPLS(n_components = n_components) # Train model using specific al-
    pha
670.            y_pred = PLSreg.predict(scale(x_validate)) # Use model to predict on valida-
    tion set
671.            return (mse(y_true = y_validate, y_pred = y_pred)) # Calculate and return mean-
    squared-error
672.
673.        def validatePLSR2(n_components):
674.            x_validate = validate.drop(['GECU3M_Index'], axis = 1) # Explanatory varia-
    bles
675.            y_validate = validate['GECU3M_Index'] # Response variable
676.            PLSreg = trainPLS(n_components = n_components) # Train model using specific al-
    pha
677.            y_pred = PLSreg.predict(scale(x_validate)) # Use model to predict on valida-
    tion set
678.            return (r2_score(y_validate, y_pred)) # Calculate and r-square
679.
680.        ###Choosing best parameter (Number of components)
681.
682.        opt_components = 0 # Variable to keep optimal number of components
683.        PLS_r2 = -10 # Variable to keep maximum r2
684.        for i in tqdm(np.arange(2,30,1)):
685.            temp_r2 = validatePLSR2(i)
686.            if(temp_r2 > PLS_r2):
687.                PLS_r2 = temp_r2
688.                opt_components = i
689.
690.        print("Optimal alpha: " + str(round(opt_components, 5)))
691.
692.        ###Performance on Testing set
693.
694.        def testPLSMSE(n_components):
695.            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
696.            y_test = test['GECU3M_Index'] # Response variable
697.            PLSreg = trainPLS(n_components = n_components)# Train model using specific al-
    pha
698.            y_pred = PLSreg.predict(scale(x_test)) # Use model to predict on test se
```

```
            return (mse(y_true = y_test, y_pred = y_pred)) # Calculate and return mean-squared-error

        def testPLSR2(n_components):
            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_test = test['GECU3M_Index'] # Response variable
            PLSreg = trainPLS(n_components = n_components)# Train model using specific alpha
            y_pred = PLSreg.predict(scale(x_test)) # Use model to predict on test set
            return (r2_score(y_test, y_pred)) # Calculate and r-square


        PLS_mse = testPLSMSE(opt_components)
        PLS_r2 = testPLSR2(opt_components)
        print('Maximum R-square: ' + str(round(PLS_r2, 5)) + " (opt_components = " + str(opt_components) + ")")
        print('Minimum MSE: ' + str(PLS_mse) + " (opt_components = " + str(opt_components) + ")")


        ### Random Forest###

        warnings.filterwarnings('ignore') #ignore warning
        def trainForest(train,n_estimators, max_features,max_depth, min_samples_split,min_samples_leaf,bootstrap):
            x_train = train.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
            y_train = train['GECU3M_Index'] # Response variable
            n_cols = x_train.shape[1] # Number of columns in training data

            rfreg = RandomForestRegressor(random_state=42,bootstrap = bootstrap, max_depth= max_depth, max_features= max_features, min_samples_leaf=  min_samples_leaf,
            min_samples_split= min_samples_split, n_estimators= n_estimators)

            # Train model
            fit = rfreg.fit(x_train, y_train)

            return fit

        def validateforest(validate,n_estimators, max_features,max_depth, min_samples_split,min_samples_leaf,bootstrap):
            x_validate = np.asarray(validate.drop(['GECU3M_Index'], axis = 1)) # Explanatory variables
            y_validate = validate['GECU3M_Index'] # Response variable
            rfreg = trainForest(train,n_estimators, max_features,max_depth, min_samples_split,min_samples_leaf,bootstrap)
            y_pred = [np.array(rfreg.predict(x_validate))] # Use model to predict on testing set
            y_pred =  np.array(y_pred).ravel()
            return (mse(y_true = y_validate, y_pred = y_pred), (my_R_oos_sq(y_validate, y_pred))) # Calculate and return mean-squared-error and r-square

        # Hyper-parameter tuning
        n_estimators = [30,50,100]
        # Number of features to consider at every split
        max_features = ['auto','sqrt']
        # Maximum number of levels in tree
        max_depth = [3,5,10]
        #max_depth.append(None)
        # Minimum number of samples required to split a node
        min_samples_split = [2, 5,7]
        # Minimum number of samples required at each leaf node
        min_samples_leaf = [1, 2,5]
        # Method of selecting samples for training each tree
        bootstrap = [False, True]

```

```
753.        resultforest = pd.DataFrame()
754.
755.        for i in tqdm(itertools.product(n_estimators, max_features,max_depth, min_sam-
    ples_split,min_samples_leaf,bootstrap)):
756.            res_mse, res_r2 = validateforest(validate,i[0], i[1], i[2], i[3],i[4],i[5])
757.            resultforest = resultforest.append(pd.DataFrame({'n_estimators' : [i[0]],
758.                                                    'max_features' : i[1],
759.                                                    'max_depth' : i[2],
760.                                                    'min_samples_split' : i[3],
761.                                                    'min_samples_leaf' : i[4],
762.                                                    'bootstrap' : i[5],
763.                                                    'mse' : [res_mse],
764.                                                    'r2' : [res_r2]}),
765.                                       ignore_index = True)
766.
767.        resultforest.sort_values(by=['r2'], ascending = False)
768.
769.        resultforest.to_csv('result_forest_3month.csv', sep = ',', index = False)
770.        resultforest = pd.read_csv('result_forest_3month.csv', sep = ',')
771.        resultforest.sort_values(by=['r2'], ascending = False)
772.
773.        merge_val = pd.concat([train,validate])
774.
775.
776.        ### Performance on Testing set
777.
778.        def testforest(test, n_estimators, max_features,max_depth, min_sam-
    ples_split,min_samples_leaf,bootstrap): # Define a function to test model with specific pa-
    rameters on testing set
779.            random.seed(a = 9) # Set seed for consistency
780.            x_test = test.drop(['GECU3M_Index'], axis = 1) # Explanatory variables
781.            y_test = np.asarray(test['GECU3M_Index']) # Response variable
782.            rfreg = trainForest(merge_val,n_estimators, max_features,max_depth, min_sam-
    ples_split,min_samples_leaf,bootstrap) # Train it
783.            y_pred = [np.array(rfreg.predict(x_test))] # Use model to predict on test-
    ing set
784.            y_pred =  np.array(y_pred).ravel()
785.            re-
    turn (mse(y_true = y_test, y_pred = y_pred), (my_R_oos_sq(y_test, y_pred))) # Calcu-
    late and return mean-squared-error and r-square
786.
787.        test_mse_1, test_r2_1 = testforest(test,200,    'au-
    to', 10, 5   ,5, False   ) # Test 1st best model on testing set
788.        #test_mse_2, test_r2_2 = testforest(test,50,    'au-
    to', 10, 7,  5,  False           ) # Test 2nd best model on testing set
789.
790.
791.        print("Test MSE (1st best model): " + str(round(test_mse_1, 4)) + " R2: " + str(rou
    nd(test_r2_1, 5)))
792.        #print("Test MSE (2nd best model): " + str(round(test_mse_2, 4)) + " R2: " + str(ro
    und(test_r2_2, 5)))
793.
794.
795.        ## Principal Component Analysis
796.
797.        pca = PCA(n_components = 24)
798.
799.        Z = pca.fit_transform(data)
800.        explained_variance = pca.explained_variance_ratio_
801.
802.        # scree plot
803.
804.        plt.figure(figsize=(10,6))
805.        plt.plot(np.cumsum(pca.explained_variance_ratio_),'-o')
806.        plt.xlabel('Number of principal components')
807.        plt.ylabel('Percentage of Variance Explained')
```

```
808.        plt.title('RETURNS-MODEL PCA',loc='center')
809.        plt.savefig('{}RETURNS-MODEL PCA'.format('Applied_Project'))
810.
811.        Var_expl = round(pd.DataFrame(np.cumsum(pca.explained_variance_ratio_), col-
    umns = ["Cumulative Variability Explained (%)"])*100,2)
812.
813.        # Plot the Cumulative Variability Explained
814.        L = []
815.
816.        for i in range(24):
817.            L.append('PCA_'+str(i))
818.        Var_expl.index = L
819.
820.        Var_expl[:15]
821.
822.        label = L[:15]
823.        index = np.arange(len(label))
824.        def plot_bar_x():
825.            # this is for plotting purpose
826.            index = np.arange(len(label))
827.            plt.bar(index, pca.explained_variance_ratio_[:15])
828.            plt.xlabel('Individual', fontsize=10)
829.            plt.ylabel('Explained variance in percent', fontsize=10)
830.            plt.xticks(index, label, fontsize=10, rotation=30)
831.            plt.title('Explained variance by different principal components')
832.            plt.show()
833.        plot_bar_x()
```