PREDICTING UNEMPLOYMENT RATE IN AUSTRALIA - GBM AND NEURAL NETWORK

MA5832: Data Mining and Machine Learning

Bienvenido Jr Hiyas

13824819 | 13-Oct-21

Table of Contents

Abstract *2*

Overview of Unemployment Rate in Australia from 1999 to 2020 ..... *3*

Unemployment Rate Dataset *4*

Data Preparation 4

Descriptive Statistics of The Variables 5

Applying Machine Learning *6*

Choosing Gradient Boosting Machines (GBM) 6

GBM Hyperparameters 6

Model Performance and Interpretation 7

Applying Neural Network (NN) *8*

Neural Network Model Structure 8

Neural Network Model Performance and Interpretation 9

Test Set Predictive Performance 10

Impacts of the Number of Hidden Layers on the Model Performance 10

Impacts of the Number of Neurons (Nodes) on the Model Performance 11

Comparison (GBM vs DNN) *11*

Cross-validated accuracy 11

Computational Time to Train Models 12

Interpretability 12

Suggestions *13*

Conclusion *13*

References *14*

APPENDEX *15*

A. Descriptive statistics R Result. 15

B. Plots of Unemployment Rate against Each Features. 16

C. R-Script 16

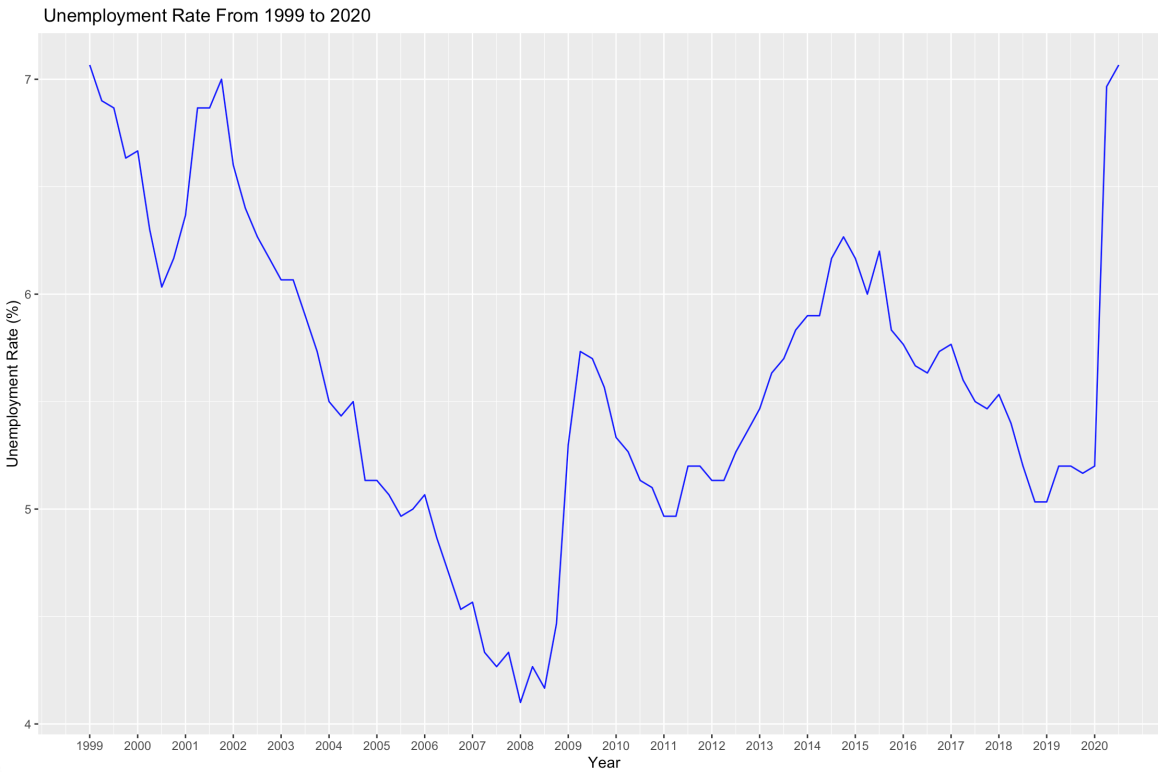
Abstract

The unemployment rate is one of the parameters to gasp and understand the economic growth in a country. The lower unemployment rate means a progressive economy where lots of job from local and foreign investments. Furthermore, it is also an indicator for the labour market conditions in a country. Unemployment rate is susceptible to global financial pandemic. The Australian unemployment rate was seen a lot of ups and downs. From the 1990s recession, the rise of energy and mining in 2003, the global financial crisis in 2007-2008, and the current Covid-19 pandemic. A supervised learning algorithm has been modelled in the unemployment rate of Australia from 1999-2020 to understand the data and provide predictive model. Gradient Boosted Machine (GBM) and neural network was modelled on the data and compared their performance and the results showed that GBM has better results.

Overview of Unemployment Rate in Australia from 1999 to 2020

According to the standard ILO classification, unemployment refers to all people with the age of over 15 years old which doesn't have paid work on the time when the survey is conducted. Those who are unemployed for more than 52 weeks or more are called as long-term unemployment [1]. Unemployment rate (UR) is the conventional measurement for underutilised labour resources in the economy and depends on the economic conditions [2]. Moreover, it usually increases fast during economic downturns and decreased slowly during economic recovery.

The Australian unemployment rate fluctuated between 1999 to 2020. Over the decade of 2000s, the unemployment rate has declined as shown in the graph. A sudden increased can be observed in 2000-2001 and during the Global Financial Crises (GFC) of 2008-2009 [3]. Moreover, the UR has skyrocketed from 2019 to 2020 because of the Covid-19 pandemic.



The decline of unemployment rate in 2004-2008 was associated with the increased of shares in mining and construction industries which in results, plummeted the employment growth. However, during the GFC in 2008, the Australian economy has been hit hard along with the world as seen in the sudden increased of UR in the graph. Although the Australian economy suffered by the GFC, it was doing better than most progressive countries in 2009 [4]. Thus, we can see a swift decreased of UR from 2009. The reason for this includes the prompt policy response from the Australian Government and Reserve Bank of Australia, the substantial benefit from economy trades from Chinese and the perseverance of the Australian banking

system throughout the crisis. The downward trend of UR can be seen 2009 up to 2011. After the decline following the GFC, the UR in Australia rebounded as shown in the graph. This is due to the slower employment growth in 2011 than the preceding years and not strong enough to sustain the growth population [5].

For the next three years, the UR displayed an upward trend, one reason is the increase of the participation of people that are willing to work however could not find a job [6]. Other reason includes the fall in mining investment in 2012-2013 where the terms of trade fell by 9.8% [7]. For the next four years a continued downward trend until 2019 when the COVID-19 pandemic occurred around the world which resulted to lockdown and the UR has skyrocketed from 5.2 to 7.1 per cent in just a year.

Unemployment Rate Dataset

The data, "AUS Data.xlsx", used in the capstone is aggregated and collected from the Australian Bureau of Statistics (ABS). The data is available quarterly from June 1981 to September 2020. The data includes the response variable (unemployment rate) and 7 predictors. Note that some variables are aggregated from monthly to quarterly for the assignment purpose. The UR data used is adjusted seasonally (UR Percentage) for the outcome variable and seven economic indicators as the primary features.

- Trend estimates of the percentage change in chain volume GDP (X1)
- TrendestimatesofthepercentagechangeinchainvolumegeneralgovernmentFCE(X2)
- TrendestimatesofthepercentagechangeinthechainvolumeFCEofallsectors(X3)
- Trend estimates of the percentage change in the ToT index(X4)
- All groups CPI (X5)
- Number of job vacancies (measured in thousands; X6)
- Estimated Resident Population of Australia (measured in millions; X7).

	Y	X1	X2	X3	X4	X5	X6.	X7
NAs	0.00	0.00	0.00	0.00	0.00	0.00	5.00	5.00
Minimum	4.10	-7.00	-4.60	-8.30	-8.10	28.40	26.80	149232.60
Maximum	11.13	3.30	7.50	5.90	13.20	116.60	232.30	253643.07
Mean	6.85	0.72	0.85	0.76	0.35	75.08	111.35	194851.26
Median	6.25	0.70	1.00	0.80	0.30	73.50	99.70	190288.02
Stdev	1.79	0.97	1.67	1.10	2.97	25.02	57.40	29339.76

Data Preparation

The data is time-series data since the observations are obtained through repeated measurement over time (21 years) and consistently measured at equally spaced intervals (per 1 year) [8]. Prior to performing machine learning and neural network models, the data was prepared using pre-processing techniques.

First, there were small numbers of missing values in X6 and X7 and they were imputed using median of each series in the training set and they are inserted in the test set where applicable.

Next, the data was split into training and test set. Since we are dealing with time (Date) splitting randomly is not applicable in this scenario. Instead, we used the data up to the end of 2014 (70% data from 1999-2020) and 2015-2019 for the test set.

The dataset was transformed closely into a supervised learning problem since most machine learning models are unable to handle time series data. First, differencing the outcome variable (Y), X5, X6 and X7 was done on the dataset, this is to remove the presence of the stochastic or deterministic trends that is found in it. Then applying lags initially to each feature and outcome variables. This is to avoid overparameterized model. The number of lags chosen were small enough since unemployment rate usually depends on the economic history. Finally, extracting the quarter and year from the time stamp including the model.

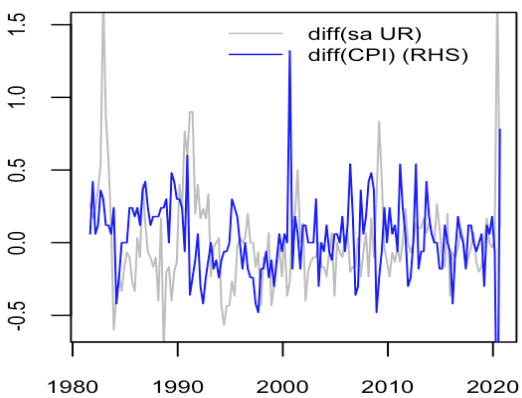
For neural network, using min-max method to create a second normalised data was done since they are sensitive to features on different scales.

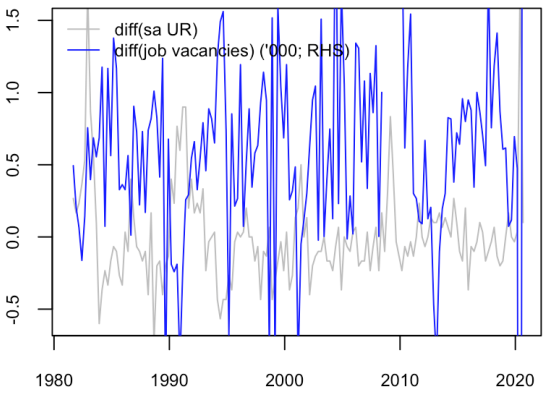
Descriptive Statistics of The Variables

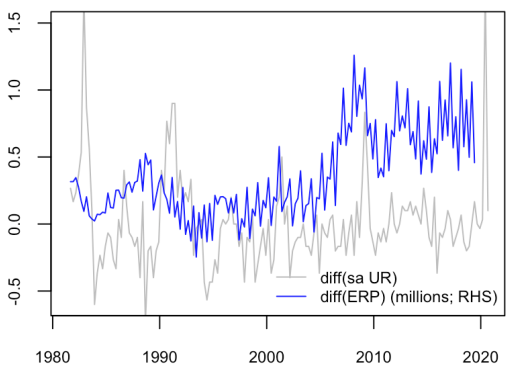
Descriptive statistics are used to summarise and describe a variable or variables for a sample of data [9]. After data preparation and processing, below is the descriptive statistics of the entire dataset. dfSummary() function has been used to summarise each features of the dataset. As shown below, the min, max, range, and mean were given on each feature. We can also see the number distinct values of each feature as well as the number of missing values. We can see those features job vacancies (X6) and resident population (X7) have five missing values or NA's. We can also see the data distribution of each feature in the graph provided.
The Descriptive Statistics R result can be found in Appendix A.

Plotting the data relationship between the unemployment rate against each original feature would give us a clear understanding on their relationship. The plot shows that most features doesn't follow the UR data trend. See Appendix B.

CPI, job vacancies and ERP seems to have no clear relationship with the UR, however, taking the first difference of the features UR, CPI, job vacancies and ERP we can see that these features would somehow follow the pattern of the UR.







Applying Machine Learning

Choosing Gradient Boosting Machines (GBM)

The three supervised machine learning techniques that were discuses in this subject were the support vector machines (SVM), random forest and gradient boosted machines (GBM). These techniques demonstrated high predictive accuracy. SVM's are more computationally intensive than random forest as it requires more data pre-processing to normalise missing values and is hard to interpret. However, SVM uses kernels to increase the dimensionality of large data feature space.

Both random forest and GBM are tree-based methods, their biggest advantage over SVM is that the less assumptions about the distribution of the data. While GBM is based on a weak learner, random forest makes many deep decorrelated trees by using subset of all available features or variables in each split. However, GBM is prone to overfitting when huge trees are used. This can be solved by using cross-validation.

The algorithm that is chosen for the UR data is the GBM. This is chosen over SVM mainly for the interpretation of the model. It was also chosen against random forest because of the sequential approach to tree building that minimizes the bias and the variance of the model.

GBM Hyperparameters

The GBM hyperparameters should be carefully tuned to optimise its predictive power. There are five main hyperparameters for GBM. First is the number of trees iteration or *n.trees*. Over fitting will happen if the *n.trees* is too large. Five-fold cross-validation us used during tuning process to find the final value on *n.tree*. Second is the learning rate or *shrinkage*. The smaller its value, the more robust it makes to the model and increases its ability to generalise, however, it requires more trees. It's typical values range between 0.001 and 0.3. Third is the tree depth or *interaction.depth* that controls the depth of each trees. The higher its value can capture distinctive patters in the data but risks of overfitting, the lower the value can become computationally efficient but needs more trees. Its typical value ranges from three to eight but some uses one. Forth is *the n.minobsinnode* or the minimum number of observations in terminal nodes. It controls the tree complexity and values range from 5-15. The higher its value

the better it against overfitting. Lastly is the *bag.fraction* or the subsample for training observations to build the next tree that introduces randomness into the algorithm with a typical values range between 0.5 and 0.8.

Model Performance and Interpretation

There are two model configurations of GBM model that were trained from the dataset. These are the following:

1. The first difference of Y, X5, X6, X7 with the year and quarter from timestamp.
2. The first model above with additional three lags of the differenced outcome variable.
3. The second model above with additional single lag of each feature.

These models are compared to base model which is the naïve model. The naïve model can be used as a baseline for the performance of the GBM algorithm and use the current time step as the predicted outcome for the next time step. The models are then assessed using three performance metrics which are the mean absolute error (MAE), the mean absolute percentage error (MAPE) and the mean squared error (MSE). The MSE was used instead of the square root of the mean squared error for consistency in comparison since the results will compared against the neural network in the later part of this report.

Shown in the R result below is the summary of the performance metrics for three models mentioned above. The optimal GBM was achieved for each configuration of features as well as the baseline naïve model.

```
|model   |    MSE   |    MAE|   MAPE|train_time  |
|:-------       |------:    |------:    |------:     |:-----------     |
|model 1 | 0.0242| 0.1189| 1.7468|557.05 secs |
|model 2 | 0.0349| 0.1399| 1.9902|262.86 secs |
|model 3 | 0.0476| 0.1549| 2.1662|252.34 secs |
|naive      | 0.1067| 0.2269| 3.1872|2.59 secs      |
```

As shown above, all three model configurations performs better than the baseline mode (naïve) across all three metrics (MSE, MAE, MAPE) except for their training time which clearly indicates that naïve performs training faster than all of the configurations. Furthermore, model 1 and model 2 performs close to each other but the latter clearly outperforms all the configurations as well as the baseline model.

The R results below shows the relative influence of the top 10 features in each model configuration. In GBM the default method for computing variable importance is with relative influence [10] and as shown below, X1 or the percentage of GDP is the most important feature in predicting the unemployment rate for all three models. Excluding the differenced and lagged features, we can notice that the least important features are X2 or FCE and X4 or CPI.

```
|mod1_feature | mod1_value|mod2_feature | mod2_value|mod3_feature | mod3_value|
|:------------ |----------: |:------------ |----------: |:------------ |----------: |
|X1 | 27.674138 |X1 | 35.374277 |X1_1 | 40.258389|
|year | 15.594263 |diffY_2 | 17.990740 |X1 | 13.198002|
|diff_X6 | 12.984701 |X4 | 11.052396 |diffY_2 | 9.334756|
|X4 | 12.743871 |diff_X6 | 8.173181 |X6_1 | 7.045339|
|X2 | 9.815349 |year | 5.568274 |diff_X6 | 6.638526|
|X3 | 7.186609 |X2 | 4.680820 |diff_X7 | 4.437461|
|diff_X7 | 7.038964 |diffY_3 | 4.477022 |X4 | 2.936962|
|diff_X5 | 5.095902 |X3 | 4.304463 |year | 2.883159|
|quarter | 1.866202 |diff_X7 | 3.402324 |X2_1 | 2.159212|
```

(a) **Test Set Performance**

Testing the GBM model's performance to the test set from the UR (from March 2018 to December 2020). Model 1 was used in test set performance of the model since it was the best model. As shown in the R result below GBM model greatly outperformed the naïve base model indicating that GMB did its job in modelling the UR dataset.

```
> knitr::kable(gbm_test_perf)
|model | MSE | MAE| MAPE|
|:----- |------: |-----: |------: |
|GBM | 0.0499| 0.137| 2.2380 |
|naive | 0.3247| 0.260| 4.0892 |
```

Applying Neural Network (NN)

Neural Network Model Structure

Machine Learning uses advanced algorithms that parse data, learns from it, and use those learnings to discover meaningful patterns of interest [11]. Whereas a Neural Network consists of an assortment of algorithms used in Machine Learning for data modelling using graphs of neurons. While a Machine Learning model makes decisions according to what it has learned from the data, a Neural Network arranges algorithms in a fashion that it can make accurate decisions by itself. Thus, although Machine Learning models can learn from data, in the initial stages, they may require some human intervention.

In neural network data will be passing through interconnected layers of nodes, classifying characteristics and information of a layer before passing the results on to other nodes in subsequent layers [11]. All deep learning neural networks were structured on feed-forward deep neural network or the multi-layer perceptron architecture. They consist of an input layer, hidden layers that that corresponds to the input features and output feature or variable and the output layer for model prediction. Each layer is connected.

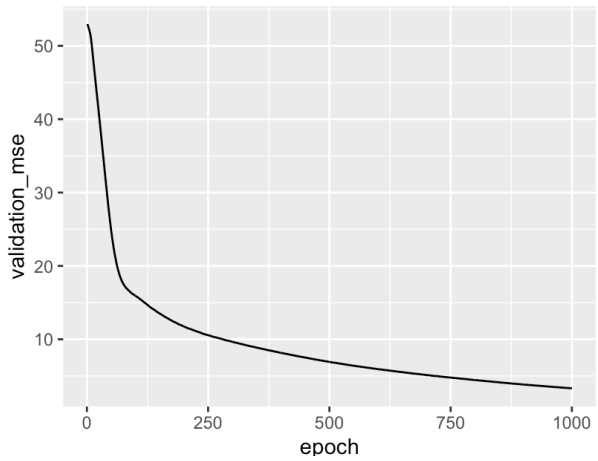
The capacity of the model to learn the different features of the data is determined by the number of the hidden layers and nodes within those hidden layers, thus affects the training time of the model. A common rule to overcome overfitting is to start with a number

somewhere between the size of the input layer (number of features) and the output layer. Since we only have a small size of data set (158 observations) we will start at two hidden layers and seven nodes in each layer. Then linear function was used for the output layer as the activation function for each layer. This was chosen this is the most common to use as a rectified linear unit function for the hidden layers.

Then MSE was used as the loss function and ADAM (adaptive moment estimation was used as the optimiser. This is to improve the accuracy of the model across a specified number (iterations or epoch), the neural networks require a loss function. This is done by adjusting weights across the node connections. Finally, determining the overfitting of the data was done by using a five-fold cross-validation. This is to provide more stable performance metrics.

Neural Network Model Performance and Interpretation

Neural Network arranges algorithms in a fashion that it can make accurate decisions by itself and do not require human intervention. Moreover, it is capable of learning through their own errors [11]. Thus, only a single model was trained. The model was then trained using the seven original features to predict the UR with an additional variable (time stamp) since the UR data is a time series dataset.



Plotting the validation scores as shown above shows that the average value of each metric in the five cross-validation folds were still declining at 1000 epochs. Even so, the number of epochs was not increased at the 1000 epochs point to minimise the overfitting of the model.

To improve training performance, different set of the model parameters were experimented. This includes changing the learning rate, batch size and dropout layer. Changing the number of nodes in each hidden layer has made slight improvements on the result. Because so few samples are available, we will be using a very small network with two hidden layers, each with 16 units. In general, the less training data you have, the worse overfitting will be, and using a small network is one way to mitigate overfitting.

```
|model |    MSE|    MAE|   MAPE|train_time  |
|:-----|------:|------:|------:|:-----------|
|DNN   | 0.3914| 0.4626| 6.7174|388.76 secs |
|naive | 0.1067| 0.2269| 3.1872|2.59 secs   |
```

The R result above shows the summary performance metrics for neural network and the naïve model on the training data. As shown above, the DNN model was out by 46 % average points compared to the naïve model of only 22 % average. We could also see that the naïve model has better MSE results of 0.11 compared to 0.4 of the DNN. Hence, we can say that the neural network performed worse than the naïve model. One issue could be that the hyperparameters were not tuned to their optimal levels.

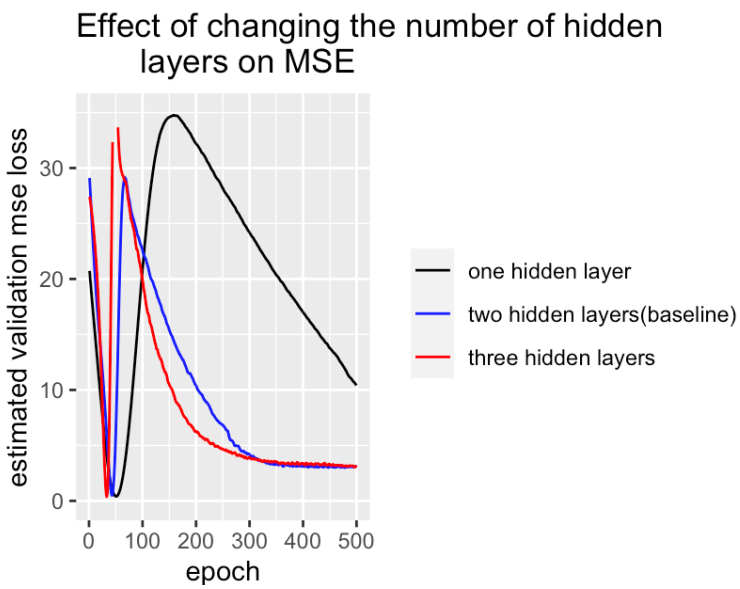
Test Set Predictive Performance

Test set with observations from March 2018 to December 2020 was performed on the NN model and as shown in the R result below, It still performs worse than the naïve model. It was 65% average points out than the naïve model with only 26% average points out.

```
|model  |    MSE  |    MAE|    MAPE|
|:-----     |------:     |------:     |-------:|
|DNN     | 0.8257| 0.6582| 11.1206|
|naive    | 0.3247| 0.2600|  4.0892|
```

Impacts of the Number of Hidden Layers on the Model Performance

The number of hidden layers were varied in the model to check its importance to the model. Since we only have few samples of the dataset, the variations are from the baseline of 2 layers, 1 layer and then 3 layers. Other parameters like the number of nodes were maintained at 16 nodes and the optimizer, batch size and activation function were also kept the same. To reduced computational time, the batch size was changed to 500, the cross-validation was replaced to pseudo-validation split of 1/3.



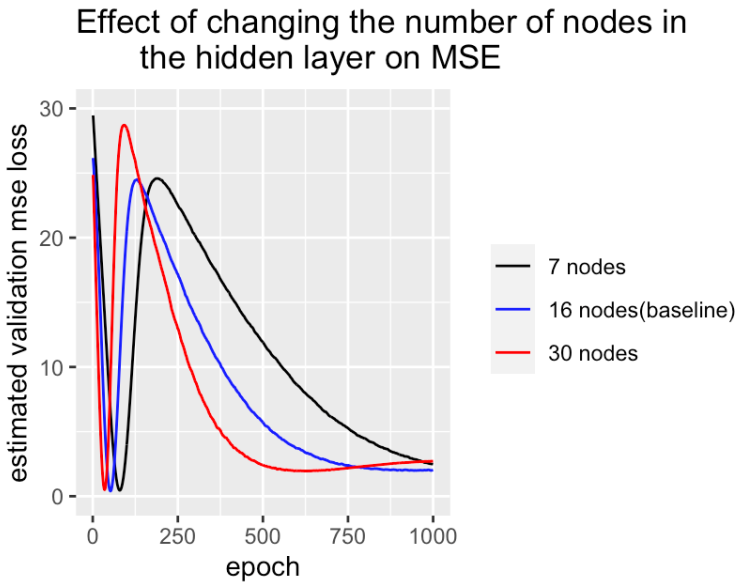
MSE
One Hidden Layer
[1] 5.522172
Two Hidden Layers
[1] 1.625367
Three Hidden Layers
[1] 1.690059

MAE
One Hidden Layer
[1] 2.004086
Two Hidden Layers
[1] 0.9810586
Three Hidden Layers
[1] 1.008335

The graph above shows the effects of changing the hidden layers. As indicated above, our model overfit much earlier on two and three layers than one hidden layer. In terms of MSE, the higher the number of hidden layers will decrease the MSE result, however, our baseline (Two hidden layers) still better than three hidden layers result. This is also the same in the MAE result. With regards to training time, no significant difference was observed.

Impacts of the Number of Neurons (Nodes) on the Model Performance

The number of nodes in the hidden layer were also varied to check the performance of the NN model. For simpler comparison, only one layer is experimented with different number of nodes (7, 16 and 30 nodes) where 16 nodes were the baseline. Other parameters were kept the same except for the number of epochs = 1000 to have a better visualisation of the graph



MSE
16 Nodes
[1] 1.413602
7 Nodes
[1] 1.923957
30 Nodes
[1] 1.509675

MAE
16 Nodes
[1] 0.9037021
7 Nodes
[1] 1.1087
30 Nodes
[1] 0.9272943

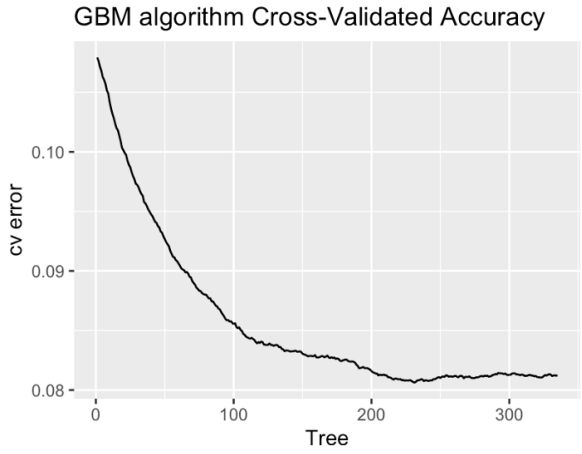
As shown in the graph above, the higher the number of nodes (16 and 30 nodes) overfits the data faster than the lower nodes (7 nodes). In terms of MSE and MAE, it seems the higher the number of nodes the higher the lower their results, thus the better performance. However, the baseline (16 nodes) performs best that the others. With regards to training time, no significant difference was observed.

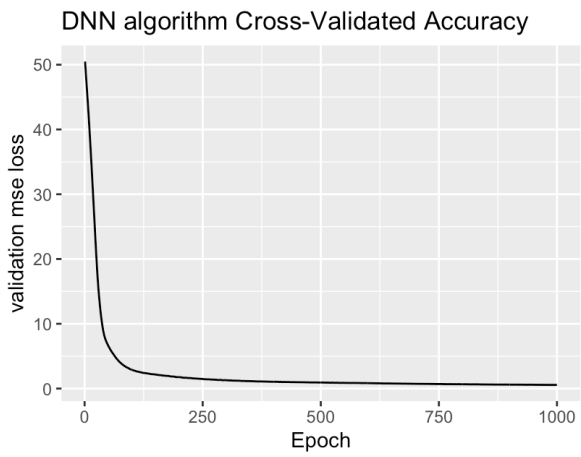
Comparison (GBM vs DNN)

Cross-validated accuracy

The cross validation used for both GBM and NN is 5-fold. This approach is preferred for small size datasets as it enables much more data to be retained during training. It also provides reliable performance results. One disadvantages of using cross-validation is that it increases training time since the algorithm will run many times on the dataset.

Squared error was used as the metric to assess the cross-validation accuracy. Using this across the number of trees for GBM and the number of epochs in DNN will show the accuracy of both GBM and DNN. As shown in the graph below, GBM has better fit to the mode.





Furthermore, the cross-validation accuracy is much better for GBM compared to DNN where the minimum error value for GBM is 0.08 while DNN has 0.56.

```
GBM
> min(gbm1$cv.error)
[1] 0.08063423
DNN
> min(nn_cv$validation_mse)
[1] 0.5601009
```

Computational Time to Train Models

The R result below shows that training time for neural network (DNN) is much longer compared to machine learning (GBM). The baseline (naïve) as the fastest among the three. However, we can see that the tuning time for the GBM also took a longer time to finished. This also depends on the number of observations in the training set, input data and the tuning grid as more tuning parameters and more values to search across will increase the number of tuning time. However, on this UR dataset, the total time of the machine learning (GBM) is worthwhile since it performs way better than the neural network (DNN).

```
|model |train_time          |tune_time |total_time   |
|:-----|:-----------        |:---------|:-----------  |
|GBM   |27.51 secs          |3.63 mins |245.06 secs |
|DNN   |142.52 secs         |NA mins   |NA secs      |
|naive |3.07 secs           |NA mins   |3.07 secs    |
```

Interpretability

Interpretability is about the extent to which a cause and effect can be observed within a system, or it is the extent to which you can *predict* what is going to happen, given a change in input or algorithmic parameters [12]. In this case, GBM has more interpretability than DNN since GBM has the variable or feature importance. This feature gives us the most important

features or variables in the dataset that could affect the outcome or the output variable, which in this case is the unemployment rate. However, the MSE, MAE, MAPE and squared error for both GBM and DNN is an important tool for prediction performance of the models. Finally, the economic data is also important as we are able to understand the relationship and interactions of the variable to produce accurate forecasts as economic indicators to make policy decisions.

Suggestions

There are various areas where the UR data prediction and modelling can be improved. First to have additional features like the causes of unemployment rate, age, education, salary wage growth and price index etc. Secondly, we should develop easier way to handle time-series data like UR dataset as we need to perform pre-processing and normalization for neural network algorithms. ARIMA and VAR methods could be used. Finally, instead of using basic deep neural network as used in the UR dataset, we could use the Long short term memory networks which works better for time-series datasets [13].

Conclusion

Using the first differences of the output variable and the features X5-X7 to remove the presence of the stochastic trends of the dataset was the best model configuration for the GBM for both training and test set. Furthermore, the neural network has performed poorly on the UR dataset compared to machine learning (GBM) and naïve model even changing its parameters like the number of hidden layers and nodes. This is probably due to the ability of neural network to completely automate the feature engineering required for machine learning. Although, tuning the hyperparameters of the machine learning (GBM) takes time, and even longer than the neural network, it is still worthwhile since GBM still performs way better than neural network in this UR dataset. Finally, the result in all the experiments and comparisons show that machine learning (GBM) has performed better than neural network.

References

[1] Loundes, J. (1997). A Brief Overview of Unemployment in Australia. Retrieved online from https://library.bsl.org.au/jspui/bitstream/1/241/1/Melbourne%20Institute%20Working%20Paper%20Series%2024%20%2097.pdf

[2] ABS, (2010). 1370.0 – Measures of Australia's Progress, 2010. Retrieved online form https://www.abs.gov.au/ausstats/abs@.nsf/Lookup/by%20Subject/1370.0~2010~Chapter~Unemployment%20(4.3.2)

[3] Borland, J. (2011). The Australian Labour Market in the 2000s: The Quit Decade. Retrieved online from https://www.rba.gov.au/publications/confs/2011/borland.html

[4] Infrastructure, Competition and Consumer Division (2009). Australia's response to the global financial crisis. Retrieved online from https://treasury.gov.au/speech/australias-response-to-the-global-financial-crisis#:~:text=The%20Global%20Financial%20Crisis%20and%20Australia%20As%20for,mid-September%202008%2C%20with%20the%20collapse%20of%20Lehman%20Brothers.

[5] Borland, J. (2012). Slow road to jobs growth: the true picture of unemployment in Australia. Retrieved online from https://theconversation.com/slow-road-to-jobs-growth-the-true-picture-of-unemployment-in-australia-5077

[6] Hurst, D. (2014). Australia's jobless rate hits highest level in more than a decade. Retrieved online from https://www.theguardian.com/business/2014/aug/07/australias-jobless-rate-hits-highest-level-in-more-than-a-decade

[7] Jericho, G. (2014). Unemployment rate same as 10 years ago, but what lies ahead is a worry. Retrieved online from https://www.theguardian.com/business/grogonomics/2014/feb/17/unemployment-rate-same-10-years-ago-but-what-ahead-worry

[8] Australian Bureau of Statistics. Statistical Language – Time Series data. Retrieved from https://www.abs.gov.au/websitedbs/D3310114.nsf/home/statistical+language+-+time+series+data

[9] Curtin University, (2021). What are descriptive statistics? Retrieved from https://libguides.library.curtin.edu.au/uniskills/numeracy-skills/statistics/descriptive

[10] Gradient Boosting Machines. UC Business Analytics R Programming Guide. Retrieved from http://uc-r.github.io/gbm_regression

[11] Goyal, K. (2020). Machine Learning vs Neural Networks: What is the Difference? Retrieved from https://www.upgrad.com/blog/machine-learning-vs-neural-networks/

[12] Gall, R. (2018). Machine Learning Explainability vs Interpretability: Two concepts that could help restore trust in AI. Retrieved from

https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html

[13] Brownlee, J. (2017). Time Series Forecasting with the Long Short-term Memory Network in Python. Retrieved from https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

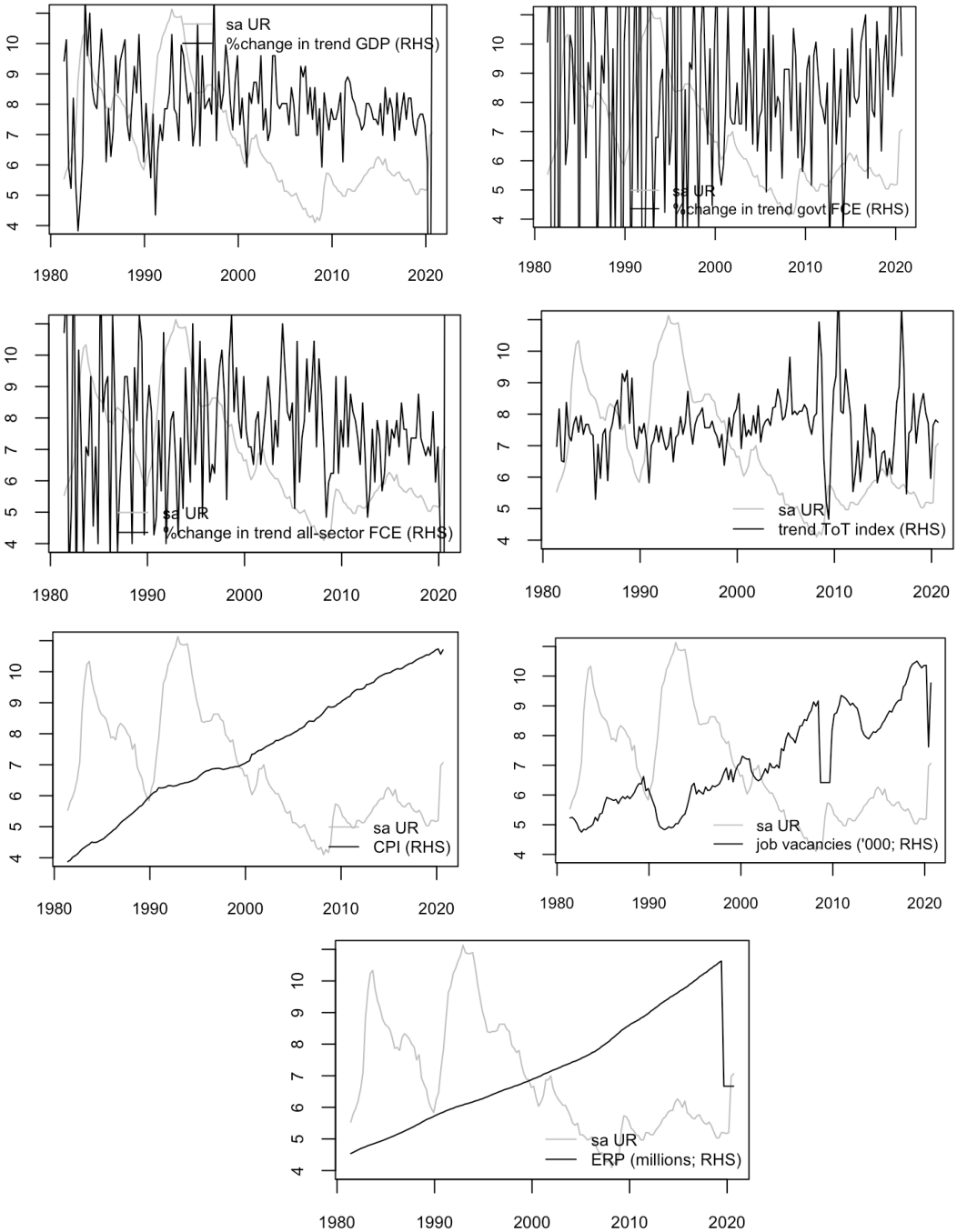
APPENDEX

A. Descriptive statistics R Result.

```
---------------------------------------------------------------------------------------------------------------
No   Variable    Stats / Values                  Freqs (% of Valid)    Graph                 Valid      Missing
---- ----------- ------------------------------- --------------------- --------------------- ---------- ---------
1    date        min : 1981-06-01                158 distinct values     : : : : : :         158        0
     [Date]      med : 2001-01-15                                        : : : : : :         (100.0%)   (0.0%)
                 max : 2020-09-01                                      : : : : : : : :
                 range : 39y 3m 0d                                     : : : : : : : :
                                                                       : : : : : : : :

2    Y           Mean (sd) : 6.9 (1.8)           102 distinct values     :                   158        0
     [numeric]   min < med < max:                                        :                   (100.0%)   (0.0%)
                 4.1 < 6.2 < 11.1                                        : :
                 IQR (CV) : 2.8 (0.3)                                  . : : . :
                                                                       : : : : : : :

3    X1          Mean (sd) : 0.7 (1)             39 distinct values                :         158        0
     [numeric]   min < med < max:                                                  :         (100.0%)   (0.0%)
                 -7 < 0.7 < 3.3                                                    :
                 IQR (CV) : 0.7 (1.3)                                            . : .
                                                                                 : : : .

4    X2          Mean (sd) : 0.9 (1.7)           58 distinct values          :               158        0
     [numeric]   min < med < max:                                            :               (100.0%)   (0.0%)
                 -4.6 < 1 < 7.5                                            . : :
                 IQR (CV) : 1.5 (2)                                        : : :
                                                                         . : : : : .

5    X3          Mean (sd) : 0.8 (1.1)           36 distinct values            :             158        0
     [numeric]   min < med < max:                                              :             (100.0%)   (0.0%)
                 -8.3 < 0.8 < 5.9                                              :
                 IQR (CV) : 0.8 (1.4)                                          :
                                                                           : :

6    X4          Mean (sd) : 0.3 (3)             80 distinct values        : .               158        0
     [numeric]   min < med < max:                                          : :               (100.0%)   (0.0%)
                 -8.1 < 0.3 < 13.2                                         : :
                 IQR (CV) : 2.8 (8.6)                                      : :
                                                                       . : : : : : .

7    X5          Mean (sd) : 75.1 (25)           148 distinct values       : :         :     158        0
     [numeric]   min < med < max:                                          : : :     : :     (100.0%)   (0.0%)
                 28.4 < 73.5 < 116.6                                   : .   : : : : : : :
                 IQR (CV) : 37.8 (0.3)                                 : : : : : : : : : :
                                                                       : : : : : : : : : :

8    X6          Mean (sd) : 111.4 (57.4)        149 distinct values   :   :                 153        5
     [numeric]   min < med < max:                                      :   : :       .       (96.8%)    (3.2%)
                 26.8 < 99.7 < 232.3                                   : : : :   : . :
                 IQR (CV) : 93.9 (0.5)                                 : : : : . : : :   .
                                                                       : : : : : : : :   .

9    X7          Mean (sd) : 194851.3 (29339.8)  153 distinct values  :   : : .            153        5
     [numeric]   min < med < max:                                      : : : : :           (96.8%)    (3.2%)
                 149232.6 < 190288 < 253643.1                          : : : : : : : . . : .
                 IQR (CV) : 46958.6 (0.2)                              : : : : : : : : : :
                                                                       : : : : : : : : : :
---------------------------------------------------------------------------------------------------------------
```

(note that screenshot is taken instead of text since using text destroys the format of the result)

B. Plots of Unemployment Rate against Each Features.



C. R-Script

```
library(readxl)
library(tidyverse)
library(scales)
install.packages("fBasics")
library(fBasics)
```

```
library(lubridate)
install.packages("summarytools")
library(summarytools)
install.packages("gbm")
library(gbm)
install.packages("vip")
library(vip)
library(keras)
install_keras()
library(caret)
library(ggplot2)

# data import
data <- read_excel("AUS_Data.xlsx", skip = 2, col_names = FALSE)
col_names <- data.frame(date = "date", "Y", "X1", "X2", "X3", "X4", "X5", "X6", "X7")
colnames(data) <- col_names
data <- mutate_if(data, is.character, as.numeric)
data$date <- as.Date(data$date, format = "%Y-%m-%d")
# plotting Unemployment Rate and date
startDate = as.Date("1999-03-01", format = "%Y-%m-%d")
data %>%
  subset(date >= "1999-03-01") %>%
  ggplot(aes(date, Y)) +
  geom_line(colour = "blue") +
  scale_x_date(breaks = function(x) seq.Date(from = startDate,
                                    to = max(data$date), by = "1 years"),
               labels = date_format("%Y", tz = "AEST")) +
  labs(title = " Unemployment Rate From 1999 to 2020",
       x = "Year", y = "Unemployment Rate (%)")

#Dataset
summary(data)
sum(is.na(data))
data_nd <- data %>% dplyr::select(-date)
sum_tbl <- round(basicStats(data_nd)[c("NAs","Minimum", "Maximum", "Mean", "Median",
"Stdev"),], digits = 2)
knitr::kable(sum_tbl)


dfSummary(data)

# traning data < 2015
train <- data %>%
  subset(date < "2015-03-01")
colSums(is.na(train))

X6_med <- median(train$X6, na.rm = TRUE)
```

```
data$X6 <- replace_na(data$X6, X6_med)

## test data > 2015
test <- data %>%
  subset(date >= "2015-03-01")
colSums(is.na(test))
X7_med <- median(train$X7, na.rm = TRUE)
data$X7 <- replace_na(data$X7,X7_med)

#Year and quarter
year <- year(as.POSIXct(data$date))
quarter <- quarter(as.POSIXct(data$date))

#First difference
diff_Y <- c(NA, diff(data$Y))
diff_X5 <- c(NA, diff(data$X5))
diff_X6 <- c(NA, diff(data$X6))
diff_X7 <- c(NA, diff(data$X7))

# creating lags of all variables as additional features
diffY_2 <- dplyr::lag(diff_Y, n = 2)
diffY_3 <- dplyr::lag(diff_Y, n = 3)
diffY_4 <- dplyr::lag(diff_Y, n = 4)
X1_1 <- dplyr::lag(data$X1)
X2_1 <- dplyr::lag(data$X2)
X3_1 <- dplyr::lag(data$X3)
X4_1 <- dplyr::lag(data$X4)
X5_1 <- dplyr::lag(diff_X5)
X6_1 <- dplyr::lag(diff_X6)
X7_1 <- dplyr::lag(diff_X7)

#combining original data with 1st difference and lags variables
new_features <- data.frame(diff_Y = diff_Y, diffY_2 = diffY_2, diffY_3 = diffY_3, diffY_4 =
diffY_4,
                    X1_1 = X1_1, X2_1 = X2_1, X3_1 = X3_1, X4_1 = X4_1, X5_1 = X5_1, X6_1 =
X6_1, X7_1 = X7_1,
                    diff_X5 = diff_X5, diff_X6 = diff_X6, diff_X7 = diff_X7,
                    year, quarter)
data_orig <- data
data <- cbind(data_orig, new_features)

#Train and set data
train <- data %>%
  subset(date < "2015-03-01")
test <- data %>%
  subset(date >= "2015-03-01")
```

```
# plotting the features against the UR
#%change in trend GDP (RHS)
with(data, plot(date, Y, type = "l", col = "gray", ylim = c(4, 11),
            xlab = NA, ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X1, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(-1.5, 2.5)))
legend("topright",
      legend = c("sa UR", "%change in trend GDP (RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#"%change in trend govt FCE (RHS)
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X2, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(-1, 2)))
legend("bottomright",
      legend = c("sa UR", "%change in trend govt FCE (RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#%change in trend all-sector FCE (RHS)
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA, ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X3, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(-0.5, 2)))
legend("bottomright",
      legend = c("sa UR", "%change in trend all-sector FCE (RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#trend ToT index (RHS)
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X4, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(-10, 10)))
legend("bottomright",
      legend = c("sa UR", "trend ToT index (RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#CPI (RHS)
```

```
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X5, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(30, 120)))
legend("bottomright",
      legend = c("sa UR", "CPI (RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#job vacancies ('000; RHS)
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X6, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(0, 250)))
legend("bottomright",
      legend = c("sa UR", "job vacancies ('000; RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

#ERP (millions; RHS)
data$X7 <- data$X7
with(data, plot(date, Y, type = "l", col = "grey", ylim = c(4, 11),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, X7/10000, type = "l", col = "black", axes = F,
            xlab = NA, ylab = NA, ylim = c(14, 26)))
legend("bottomright",
      legend = c("sa UR", "ERP (millions; RHS)"),
      lty = c(1,1), col = c("grey", "black"), cex = 0.75,
      bty = "n")

# Taking First Differences of UR< CPI, job vacancies and ERP to remove the trend in each series
#
with(data, plot(date, c(NA,diff(Y)), type = "l", col = "grey", ylim = c(-0.6, 1.5),
            xlab = NA, ylab = waiver(), cex.axis = 0.75))
legend("topright",
      legend = "diff(sa UR)",
      lty = 1, col = "grey", cex = 0.75,
      bty = "n")
with(data, plot(date, c(NA,diff(Y)), type = "l", col = "grey", ylim = c(-0.6, 1.5),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, c(NA,diff(X5)), type = "l", col = "blue", axes = F,
            xlab = NA, ylab = NA, ylim = c(-0.5, 3.0)))
legend("topright",
```

```
      legend = c("diff(sa UR)", "diff(CPI) (RHS)"),
      lty = c(1,1), col = c("grey", "blue"), cex = 0.75,
      bty = "n")
with(data, plot(date, c(NA,diff(Y)), type = "l", col = "grey", ylim = c(-0.6, 1.5),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, c(NA,diff(X6)), type = "l", col = "blue", axes = F,
            xlab = NA, ylab = NA, ylim = c(-12, 12)))
legend("topleft",
      legend = c("diff(sa UR)", "diff(job vacancies) ('000; RHS)"),
      lty = c(1,1), col = c("grey", "blue"), cex = 0.75,
      bty = "n")
data$X7 <- data$X7
with(data, plot(date, c(NA,diff(Y)), type = "l", col = "grey", ylim = c(-0.6, 1.5),
            xlab = NA,  ylab = waiver(), cex.axis = 0.75))
par(new = T)
with(data, plot(date, c(NA,diff(X7/10000)), type = "l", col = "blue", axes = F,
            xlab = NA, ylab = NA, ylim = c(0, 0.15)))
legend("bottomright",
      legend = c("diff(sa UR)", "diff(ERP) (millions; RHS)"),
      lty = c(1,1), col = c("grey", "blue"), cex = 0.75,
      bty = "n")


#Training the Model using Naive  Model (Baseline)
start_naive <- Sys.time()
naive_pred <- dplyr::lag(train$Y)
naive_df <- data.frame(date = train$date,
                    Y = train$Y,
                    Y_pred = naive_pred)
end_naive <- Sys.time()
time_naive <- end_naive - start_naive
time_naive

# Applying GBM
################################                    Model                    1
#########################################
#The first difference of Y, X5, X6, X7 with the year and quarter
model1_train <- train %>%
  select(date, diff_Y, X1:X4, diff_X5:diff_X7, year:quarter)
model1_train <- model1_train[2:nrow(model1_train),]
is.na(model1_train)

start_gbm1 <- Sys.time()
set.seed(2020)
model1_gbm <- gbm(diff_Y ~.-date, data = model1_train,
            distribution = "gaussian",
```

```
            n.trees = 2000,       # set to an arbitrarily high number
            shrinkage = 0.1,      # default
            interaction.depth = 3, # default is 1
            n.minobsinnode = 10,   # default
            bag.fraction = 0.5,    # default
            cv.folds = 5)
model1_gbm
end_naive <- Sys.time()
RMSE_loc <- which.min(model1_gbm$cv.error)
RMSE <- sqrt(min(model1_gbm$cv.error))
gbm.perf(model1_gbm, method = "cv", plot.it = FALSE)

# tuning the learning rate
start_tunegbm1 <- Sys.time()
gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)

for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
      diff_Y ~.-date, data = model1_train,
      distribution = "gaussian",
      n.trees = 1000, # set somewhere above the optimal number indicated by the first run
      shrinkage = gbm_grid$shrinkage[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      bag.fraction = 0.5,
      cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
  gbm_grid$time[i] <- train_time[["elapsed"]]

}
gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
#result
#shrinkage      RMSE trees  time
#1     0.010 0.2850717   335 3.599
```

```
# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 335, #from tuned trees
  shrinkage = 0.010,#from tuning
  interaction.depth = c(1, 3, 5, 7, 8),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))

gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode, bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = model1_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}
gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)
hyper_params1 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
end_tunegbm1 <- Sys.time()

# training the final model
n.trees <- hyper_params1[[1,1]]
shrinkage <- hyper_params1[[1,2]]
interaction.depth <- hyper_params1[[1,3]]
n.minobsinnode <- hyper_params1[[1,4]]
bag.fraction <- hyper_params1[[1,5]]
set.seed(2020)
gbm1 <- gbm(diff_Y ~.-date, data = model1_train,
            distribution = "gaussian",
            n.trees = n.trees,
```

```
          shrinkage = shrinkage,
          interaction.depth = interaction.depth,
          n.minobsinnode = n.minobsinnode,
          bag.fraction = bag.fraction,
          cv.folds = 5)
gbm1
summary(gbm1, plot = FALSE) # produces feature importance output
end_gbm1 <- Sys.time()
totaltime_gbm1 <- end_gbm1 - start_gbm1
tunetime_gbm1 <- end_tunegbm1 - start_tunegbm1
traintime_gbm1 <- totaltime_gbm1 - tunetime_gbm1


################################## Model 2
########################################
#The first model above with additional three lags of the differenced outcome variable

model2_train <- train %>%
  select(date, diff_Y, diffY_2:diffY_4, X1:X4, diff_X5:diff_X7, year:quarter)
model2_train <- model2_train[6:nrow(model2_train),]
# training a basic model
start_gbm2 <- Sys.time()
set.seed(2020)
Model2_gbm <- gbm(diff_Y ~.-date, data = model2_train,
                  distribution = "gaussian",
                  n.trees = 2000,       # set to an arbitrarily high number
                  shrinkage = 0.1,      # default
                  interaction.depth = 3, # default is 1
                  n.minobsinnode = 10,   # default
                  bag.fraction = 0.5,    # default
                  cv.folds = 5)
Model2_gbm
RMSE_loc <- which.min(Model2_gbm$cv.error)
RMSE <- sqrt(min(Model2_gbm$cv.error))
gbm.perf(Model2_gbm, method = "cv", plot.it = FALSE)

# tuning the learning rate
start_tunegbm2 <- Sys.time()
gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)
for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
```

```
    diff_Y ~.-date, data = model2_train,
    distribution = "gaussian",
    n.trees = 1000, # set somewhere above the optimal number indicated by the first run
    shrinkage = gbm_grid$shrinkage[i],
    interaction.depth = 3,
    n.minobsinnode = 10,
    bag.fraction = 0.5,
    cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
  gbm_grid$time[i] <- train_time[["elapsed"]]

}
gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
#  shrinkage      RMSE trees  time
#1     0.001 0.2907004  1000 3.65

# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 24, #from tuning
  shrinkage = 0.1, #from tuning
  interaction.depth = c(1, 3, 5, 7, 8),
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))
gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode, bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = model2_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}
gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
```

```
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)
hyper_params2 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
end_tunegbm2 <- Sys.time()
# training the final model
n.trees <- hyper_params2[[1,1]]
shrinkage <- hyper_params2[[1,2]]
interaction.depth <- hyper_params2[[1,3]]
n.minobsinnode <- hyper_params2[[1,4]]
bag.fraction <- hyper_params2[[1,5]]
set.seed(2020)
gbm2 <- gbm(diff_Y ~.-date, data = model2_train,
            distribution = "gaussian",
            n.trees = n.trees,
            shrinkage = shrinkage,
            interaction.depth = interaction.depth,
            n.minobsinnode = n.minobsinnode,
            bag.fraction = bag.fraction,
            cv.folds = 5)
gbm2
summary(gbm2, plot = FALSE) # produces feature importance output
end_gbm2 <- Sys.time()
totaltime_gbm2 <- end_gbm2 - start_gbm2
tunetime_gbm2 <- end_tunegbm2 - start_tunegbm2
traintime_gbm2 <- totaltime_gbm2 - tunetime_gbm2
totaltime_gbm2
tunetime_gbm2
traintime_gbm2

##################################                    Model                    3
########################################
#The second model above with additional single lag of each feature.

model3_train <- train %>%
  select(date, diff_Y, diffY_2:diffY_4, X1:X4, diff_X5:diff_X7, X1_1:X7_1, year:quarter)
model3_train <- model3_train[6:nrow(model3_train),]
# training a basic model
start_gbm3 <- Sys.time()
set.seed(2020)
model3_gbm <- gbm(diff_Y ~.-date, data = model3_train,
              distribution = "gaussian",
              n.trees = 2000,       # set to an arbitrarily high number
```

```
            shrinkage = 0.1,       # default
            interaction.depth = 3, # default is 1
            n.minobsinnode = 10,   # default
            bag.fraction = 0.5,    # default
            cv.folds = 5)
model3_gbm
RMSE_loc <- which.min(model3_gbm$cv.error)
RMSE <- sqrt(min(model3_gbm$cv.error))
gbm.perf(model3_gbm, method = "cv", plot.it = FALSE)
# tuning the learning rate
start_tunegbm3 <- Sys.time()
gbm_grid <- expand.grid(
  shrinkage = c(0.001, 0.005, 0.01, 0.05, 0.1, 0.3),
  RMSE = NA,
  trees = NA,
  time = NA)
for(i in seq_len(nrow(gbm_grid))) {
  set.seed(2020)
  train_time <- system.time({
    gbm_tune <- gbm(
      diff_Y ~.-date, data = model3_train,
      distribution = "gaussian",
      n.trees = 1000, # set somewhere above the optimal number indicated by the first run
      shrinkage = gbm_grid$shrinkage[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      bag.fraction = 0.5,
      cv.folds = 5)
  })

  gbm_grid$RMSE[i] <- sqrt(min(gbm_tune$cv.error))
  gbm_grid$trees[i] <- which.min(gbm_tune$cv.error)
  gbm_grid$time[i] <- train_time[["elapsed"]]

}
gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
# result
#shrinkage   RMSE trees   time
#1     0.300 0.2623402    12 3.583

# tuning the remaining parameters
gbm_grid <- expand.grid(
  n.trees = 12, #from tuning
  shrinkage = 0.300, #from tuning
  interaction.depth = c(1, 3, 5, 7, 8),
```

```
  n.minobsinnode = c(5, 10, 15),
  bag.fraction = c(0.5, 0.65, 0.8))
gbm_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode, bag.fraction) {
  set.seed(2020)
  gbm_tune <- gbm(
    diff_Y ~.-date, data = model3_train,
    distribution = "gaussian",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth =  interaction.depth,
    n.minobsinnode = n.minobsinnode,
    bag.fraction = bag.fraction,
    cv.folds = 5
  )
  sqrt(min(gbm_tune$cv.error))
}
gbm_grid$RMSE <- purrr::pmap_dbl(
  gbm_grid,
  ~ gbm_fit(
    n.trees = ..1,
    shrinkage =  ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4,
    bag.fraction = ..5
  )
)
hyper_params3 <- gbm_grid %>%
  dplyr::arrange(RMSE) %>%
  head(20)
end_tunegbm3 <- Sys.time()
# training the final model
n.trees <- hyper_params3[[1,1]]
shrinkage <- hyper_params3[[1,2]]
interaction.depth <- hyper_params3[[1,3]]
n.minobsinnode <- hyper_params3[[1,4]]
bag.fraction <- hyper_params3[[1,5]]
set.seed(2020)
gbm3 <- gbm(diff_Y ~.-date, data = model3_train,
            distribution = "gaussian",
            n.trees = n.trees,
            shrinkage = shrinkage,
            interaction.depth = interaction.depth,
            n.minobsinnode = n.minobsinnode,
            bag.fraction = bag.fraction,
            cv.folds = 5)
gbm3
end_gbm3 <- Sys.time()
```

```
totaltime_gbm3 <- end_gbm3 - start_gbm3
tunetime_gbm3 <- end_tunegbm3 - start_tunegbm3
traintime_gbm3 <- totaltime_gbm3 - tunetime_gbm3



# reverse engineering the transformation for training set performance
train_gbm <- data.frame(date = train$date,
                        Y = train$Y,
                        naive_pred = naive_df$Y_pred,
                        diff_Y = train$diff_Y,
                        diff1_pred = c(NA, gbm1$cv.fitted),
                        diff2_pred = c(rep(NA, 5), gbm2$cv.fitted),
                        diff3_pred = c(rep(NA, 5), gbm3$cv.fitted),
                        gbm1_pred = c(NA, gbm1$cv.fitted) + train$Y,
                        gbm2_pred = c(rep(NA, 5), gbm2$cv.fitted) + train$Y,
                        gbm3_pred = c(rep(NA, 5), gbm3$cv.fitted) + train$Y)

naive_error <- train_gbm$Y[2:135] - train_gbm$naive_pred[2:135]
MSE_naive <- mean(naive_error^2)
MAE_naive <- mean(abs(naive_error))
MAPE_naive <- mean(abs(naive_error / train_gbm$Y[2:135]) * 100)

gbm1_error <- train_gbm$Y[2:135] - train_gbm$gbm1_pred[2:135]
MSE_gbm1 <- mean(gbm1_error^2)
MAE_gbm1 <- mean(abs(gbm1_error))
MAPE_gbm1 <- mean(abs(gbm1_error / train_gbm$Y[2:135]) * 100)

gbm2_error <- train_gbm$Y[6:135] - train_gbm$gbm2_pred[6:135]
MSE_gbm2 <- mean(gbm2_error^2)
MAE_gbm2 <- mean(abs(gbm2_error))
MAPE_gbm2 <- mean(abs(gbm2_error / train_gbm$Y[6:135]) * 100)

gbm3_error <- train_gbm$Y[6:135] - train_gbm$gbm3_pred[6:135]
MSE_gbm3 <- mean(gbm3_error^2)
MAE_gbm3 <- mean(abs(gbm3_error))
MAPE_gbm3 <- mean(abs(gbm3_error / train_gbm$Y[6:135]) * 100)

gbm_train_perf <- data.frame(model = c("model 1", "model 2", "model 3", "naive"),
                             MSE = round(c(MSE_gbm1, MSE_gbm2, MSE_gbm3, MSE_naive), 4),
                             MAE = round(c(MAE_gbm1, MAE_gbm2, MAE_gbm3, MAE_naive), 4),
                             MAPE = round(c(MAPE_gbm1, MAPE_gbm2, MAPE_gbm3, MAPE_naive), 4),
                             train_time = round(c(totaltime_gbm1, totaltime_gbm2, totaltime_gbm3,
time_naive), 2))
knitr::kable(gbm_train_perf)
```

```
#Feature Importance
rel_inf1 <- data.frame(gbm1_feature = head(vi(gbm1)[,1], 9),
                       gbm1_value = head(vi(gbm1)[,2], 9))
rel_inf2 <- data.frame(gbm1_feature = head(vi(gbm2)[,1], 9),
                       gbm1_value = head(vi(gbm2)[,2], 9))
rel_inf3 <- data.frame(gbm1_feature = head(vi(gbm3)[,1], 9),
                       gbm1_value = head(vi(gbm3)[,2], 9))
gbm_vi <- data.frame(rel_inf1, rel_inf2, rel_inf3)
colnames(gbm_vi) <- c("mod1_feature", "mod1_value", "mod2_feature", "mod2_value",
"mod3_feature", "mod3_value")
knitr::kable(gbm_vi)

summary(gbm1, plot = TRUE) # produces feature importance output
summary(gbm2, plot = TRUE) # produces feature importance output
summary(gbm3, plot = TRUE) # produces feature importance output

## Test set predictive performance
test <- data %>%
  subset(date >= "2018-03-01")

diff_test <- test %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year:quarter)
gbm_fc <- predict(gbm1, newdata = diff_test, n.trees = hyper_params1[[1,1]]) #using model 1
parameters
# naive persistence model forecasts
naive_fc <- dplyr::lag(test$Y)
naive_tdf <- data.frame(date = test$date,
                        Y = test$Y,
                        Y_fc = naive_fc)
test_gbm <- data.frame(date = test$date,
                       Y = test$Y,
                       naive_fc = naive_tdf$Y_fc,
                       diff_Y = test$diff_Y,
                       diffgbm_fc = gbm_fc,
                       gbm_fc = gbm_fc + test$Y)

naive_terror <- test_gbm$Y[2:11] - test_gbm$naive_fc[2:11]
tMSE_naive <- mean(naive_terror^2)
tMAE_naive <- mean(abs(naive_terror))
tMAPE_naive <- mean(abs(naive_terror / test_gbm$Y[2:11]) * 100)

gbm_terror <- test_gbm$Y - test_gbm$gbm_fc
tMSE_gbm <- mean(gbm_terror^2)
tMAE_gbm <- mean(abs(gbm_terror))
tMAPE_gbm <- mean(abs(gbm_terror / test_gbm$Y) * 100)

gbm_test_perf <- data.frame(model = c("GBM", "naive"),
```

```
                    MSE = round(c(tMSE_gbm, tMSE_naive), 4),
                    MAE = round(c(tMAE_gbm, tMAE_naive), 4),
                    MAPE = round(c(tMAPE_gbm, tMAPE_naive), 4))
knitr::kable(gbm_test_perf)

##################################### Applying Neural
Network#######################
#library(keras)
#install_keras()
#library(caret)

# Feature-wise normalisation using min-max method fore neural network.
nn_train1 <- train %>%
  select(Y, X1:X7, year, quarter)
nn_train2 <- train %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year, quarter)
nn_train2 <- nn_train2[2:nrow(nn_train2),]

test <- data %>%
  subset(date >= "2018-03-01")
nn_test1 <- test %>%
  select(Y, X1:X7, year, quarter)
nn_test2 <- test %>%
  select(diff_Y, X1:X4, diff_X5:diff_X7, year, quarter)

train_stats1 <- preProcess(nn_train1[2:10], method = "range")
norm_train1 <- predict(train_stats1, nn_train1[2:10])
norm_test1 <- predict(train_stats1, nn_test1[2:10])
train_stats2 <- preProcess(nn_train2[2:10], method = "range")
norm_train2 <- predict(train_stats2, nn_train2[2:10])
norm_test2 <- predict(train_stats2, nn_test2[2:10])

## Model: Baseline plus time stamp variables
train1_data <- as.matrix(norm_train1)
train1_y <- as.matrix(nn_train1$Y)
build_model <- function() {
  nn1 <- keras_model_sequential() %>%
    layer_dense(units = 16, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
    layer_dense(units = 7, activation = "relu") %>%
    layer_dense(units = 1)
  nn1 %>% compile(
    optimizer = "adam",
    loss = "mse",
    metrics = c("mae", "mape", "mse"))
}
# model training with k-fold cross-validation
```

```
start_nn1 <- Sys.time()
k <- 5
indices <- sample(1:nrow(train1_data))
folds <- cut(indices, breaks = k, labels = FALSE)
num_epochs <- 1000
all_mae_histories <- NULL
all_mape_histories <- NULL
all_mse_histories <- NULL
for (i in 1:k) {
  cat("processing fold #", i, "\n")

  # preparing validation data: data from partition k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train1_data[val_indices,]
  val_targets <- train1_y[val_indices]

  # preparing validation data: data from all other partitions
  partial_train_data <- train1_data[-val_indices,]
  partial_train_targets <- train1_y[-val_indices]

  # building model
  nn1 <- build_model()

  # training model
  history <- nn1 %>% fit (
    train1_data, train1_y,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 32, verbose = FALSE,
  )
  mae_history <- history$metrics$val_mae
  mape_history <- history$metrics$val_mape
  mse_history <- history$metrics$val_mse

  all_mae_histories <- rbind(all_mae_histories, mae_history)
  all_mape_histories <- rbind(all_mape_histories, mape_history)
  all_mse_histories <- rbind(all_mse_histories, mse_history)
}
plot(history)
# averaging per-epoch metric scores for all folds
average_mae_history <- data.frame(
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories, 2, mean)
)
average_mape_history <- data.frame(
  epoch = seq(1:ncol(all_mape_histories)),
  validation_mape = apply(all_mape_histories, 2, mean)
)
```

```
average_mse_history <- data.frame(
  epoch = seq(1:ncol(all_mse_histories)),
  validation_mse = apply(all_mse_histories, 2, mean)
)
# plotting validation scores
#library(ggplot2)
mse_plot1 <- average_mse_history %>%
  ggplot(aes(x = epoch, y = validation_mse)) +
  geom_line()
mse_plot1
mse_min1 <- which.min(average_mse_history$validation_mse)
mse1 <- average_mse_history$validation_mse[mse_min1]
# training the final model
final_nn1 <- build_model()
final_nn1_mod <- fit(final_nn1, train1_data, train1_y, epochs = 1000, batch_size = 32, verbose
= FALSE)
nn1_pred <- predict(final_nn1, train1_data)
end_nn1 <- Sys.time()
time_nn1 <- end_nn1 - start_nn1

# test set performance
test1_data <- as.matrix(norm_test1)
test1_y <- as.matrix(nn_test1$Y)
test_perf <- evaluate(final_nn1, test1_data, test1_y)
test_perf
nn_fc <- predict(final_nn1, test1_data)
tMSE_nn = as.numeric(test_perf[1])
tMAE_nn = as.numeric(test_perf[2])
tMAPE_nn = as.numeric(test_perf[3])

# training set performance
train_nn <- data.frame(date = train$date,
                       Y = train$Y,
                       naive_pred = naive_df$Y_pred,
                       nn1_pred = nn1_pred)
naive_error <- train_nn$Y[2:135] - train_nn$naive_pred[2:135]
MSE_naive <- mean(naive_error^2)
MAE_naive <- mean(abs(naive_error))
MAPE_naive <- mean(abs(naive_error / train_nn$Y[2:135]) * 100)
nn1_error <- train_nn$Y - train_nn$nn1_pred
MSE_nn1 <- mean(nn1_error^2)
MAE_nn1 <- mean(abs(nn1_error))
MAPE_nn1 <- mean(abs(nn1_error / train_nn$Y) * 100)

nn_train_perf <- data.frame(model = c("DNN", "naive"),
                            MSE = round(c(MSE_nn1, MSE_naive), 4),
                            MAE = round(c(MAE_nn1, MAE_naive), 4),
```

```
                    MAPE = round(c(MAPE_nn1, MAPE_naive), 4),
                    train_time = round(c(time_nn1, time_naive), 2))
knitr::kable(nn_train_perf)

#(c)      Test Set Predictive Performance

nn_test_perf <- data.frame(model = c("DNN", "naive"),
                    MSE = round(c(tMSE_nn, tMSE_naive), 4),
                    MAE = round(c(tMAE_nn, tMAE_naive), 4),
                    MAPE = round(c(tMAPE_nn, tMAPE_naive), 4))
knitr::kable(nn_test_perf)

#(d)      Impacts of the Number of Hidden Layers on the Model Performance
# Using 2 hidden layers - Baseline
exp1_start <- Sys.time()
nn_exp1 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
            input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1)
nn_exp1 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))
# model training
history1 <- fit(nn_exp1, train1_data, train1_y, validation_split = 1/3, epochs = 500, batch_size
= 32,
             verbose = FALSE)
hist_plot1 <- plot(history1)
exp1_data <- data.frame(x = c(1:history1$params$epochs), y = history1$metrics$val_loss)
plot_exp1 <- ggplot(exp1_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch1 <- which.min(history1$metrics$val_loss)
exp1_pred <- predict(nn_exp1, train1_data)
exp1_error <- train1_y - exp1_pred
MSE_exp1 <- mean(exp1_error^2)
MAE_exp1 <- mean(abs(exp1_error))
exp1_end <- Sys.time()
exp1_time <- exp1_end - exp1_start

# Using One hidden layer
exp2_start <- Sys.time()
nn_exp2 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
            input_shape = dim(train1_data)[[2]]) %>%
```

```
  layer_dense(units = 1)
nn_exp2 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))
# model training
history2 <- fit(nn_exp2, train1_data, train1_y, validation_split = 1/3, epochs = 500, batch_size
= 32,
              verbose = FALSE)
hist_plot2 <- plot(history2)
exp2_data <- data.frame(x = c(1:history2$params$epochs), y = history2$metrics$val_loss)
plot_exp2 <- ggplot(exp2_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch2 <- which.min(history2$metrics$val_loss)
exp2_pred <- predict(nn_exp2, train1_data)
exp2_error <- train1_y - exp2_pred
MSE_exp2 <- mean(exp2_error^2)
MAE_exp2 <- mean(abs(exp2_error))
exp2_end <- Sys.time()
exp2_time <- exp2_end - exp2_start

# Using Three hidden layers
exp3_start <- Sys.time()
nn_exp3 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1)
nn_exp3 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))
# model training
history3 <- fit(nn_exp3, train1_data, train1_y, validation_split = 1/3, epochs = 500, batch_size
= 32,
              verbose = FALSE)
hist_plot3 <- plot(history3)
exp3_data <- data.frame(x = c(1:history3$params$epochs), y = history3$metrics$val_loss)
plot_exp2 <- ggplot(exp3_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch3 <- which.min(history3$metrics$val_loss)
exp3_pred <- predict(nn_exp3, train1_data)
```

```
exp3_error <- train1_y - exp3_pred
MSE_exp3 <- mean(exp3_error^2)
MAE_exp3 <- mean(abs(exp3_error))
exp3_end <- Sys.time()
exp3_time <- exp3_end - exp3_start

exp_data_metrics <- data.frame(exp2_loss = history2$metrics$val_loss,
                               exp2_mae = history2$metrics$val_mae,
                               exp1_loss = history1$metrics$val_loss,
                               exp1_mae = history1$metrics$val_mae,
                               exp3_loss = history3$metrics$val_loss,
                               exp3_mae = history3$metrics$val_mae)
metric_means <- apply(exp_data_metrics, 2, mean)
exp_data_pred <- data.frame(exp2_MSE = MSE_exp2,
                            exp2_MAE = MAE_exp2,
                            exp1_MSE = MSE_exp1,
                            exp1_MAE = MAE_exp1,
                            exp3_MSE = MSE_exp3,
                            exp3_MAE = MAE_exp3)
library(reshape2)
exp_data <- data.frame(epoch = exp1_data$x,
                       mse_1hd = exp2_data$y,
                       mse_2hd = exp1_data$y,
                       mse_3hd = exp3_data$y)
exp_data <- melt(exp_data, id.vars = "epoch")
exp_data %>%
  ggplot() +
  geom_line(aes(x = epoch, y = value, color = variable)) +
  scale_colour_manual(name = NULL,
                      labels = c("one hidden layer", "two hidden layers(baseline)", "three hidden
layers"),
                      values = c("black", "blue", "red")) +
  scale_y_continuous(limits = c(0, 35), breaks = seq(0, 35, 10)) +
  labs(x = "epoch",
       y = "estimated validation mse loss",
       title = "Effect of changing the number of hidden
       layers on MSE")

#(e)     Impacts of the Number of Neurons (Nodes) on the Model Performance
#Baseline - one hidden layer with 16 nodes
exp4_start <- Sys.time()
nn_exp4 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)
nn_exp4 %>% compile(
  optimizer = "adam",
```

```
  loss = "mse",
  metrics = c("mae"))
# model training
history4 <- fit(nn_exp4, train1_data, train1_y, validation_split = 1/3, epochs = 1000, batch_size
= 32,
              verbose = FALSE)
hist_plot4 <- plot(history4)
exp4_data <- data.frame(x = c(1:history4$params$epochs), y = history4$metrics$val_loss)
plot_exp4 <- ggplot(exp4_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch4 <- which.min(history4$metrics$val_loss)
exp4_pred <- predict(nn_exp4, train1_data)
exp4_error <- train1_y - exp4_pred
MSE_exp4 <- mean(exp4_error^2)
MAE_exp4 <- mean(abs(exp4_error))
exp4_end <- Sys.time()
exp4_time <- exp4_end - exp4_start

# One hidden layer with seven nodes
exp5_start <- Sys.time()
nn_exp5 <- keras_model_sequential() %>%
  layer_dense(units = 7, activation = "relu",
            input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)
nn_exp5 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))
# model training
history5 <- fit(nn_exp5, train1_data, train1_y, validation_split = 1/3, epochs = 1000, batch_size
= 32,
              verbose = FALSE)
hist_plot5 <- plot(history5)
exp5_data <- data.frame(x = c(1:history5$params$epochs), y = history5$metrics$val_loss)
plot_exp5 <- ggplot(exp5_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch5 <- which.min(history5$metrics$val_loss)
exp5_pred <- predict(nn_exp5, train1_data)
exp5_error <- train1_y - exp5_pred
MSE_exp5 <- mean(exp5_error^2)
MAE_exp5 <- mean(abs(exp5_error))
exp5_end <- Sys.time()
exp5_time <- exp5_end - exp5_start
```

```
# One hidden layer with 30 nodes
exp6_start <- Sys.time()
nn_exp6 <- keras_model_sequential() %>%
  layer_dense(units = 30, activation = "relu",
              input_shape = dim(train1_data)[[2]]) %>%
  layer_dense(units = 1)
nn_exp6 %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = c("mae"))
# model training
history6 <- fit(nn_exp6, train1_data, train1_y, validation_split = 1/3, epochs = 1000, batch_size
= 32,
                verbose = FALSE)
hist_plot6 <- plot(history6)
exp6_data <- data.frame(x = c(1:history6$params$epochs), y = history6$metrics$val_loss)
plot_exp6 <- ggplot(exp6_data, aes(x = x, y = y)) +
  geom_line() +
  xlab("epoch") +
  ylab("estimated validation mse loss")
overfit_epoch6 <- which.min(history6$metrics$val_loss)
exp6_pred <- predict(nn_exp6, train1_data)
exp6_error <- train1_y - exp6_pred
MSE_exp6 <- mean(exp6_error^2)
MAE_exp6 <- mean(abs(exp6_error))
exp6_end <- Sys.time()
exp6_time <- exp6_end - exp6_start

exp_data_metrics2 <- data.frame(exp5_loss = history5$metrics$val_loss,
                                exp5_mae = history5$metrics$val_mae,
                                exp4_loss = history4$metrics$val_loss,
                                exp4_mae = history4$metrics$val_mae,
                                exp6_loss = history6$metrics$val_loss,
                                exp6_mae = history6$metrics$val_mae)
metric_means2 <- apply(exp_data_metrics2, 2, mean)
exp_data_pred2 <- data.frame(exp5_MSE = MSE_exp5,
                             exp5_MAE = MAE_exp5,
                             exp4_MSE = MSE_exp4,
                             exp4_MAE = MAE_exp4,
                             exp6_MSE = MSE_exp6,
                             exp6_MAE = MAE_exp6)
exp_data2 <- data.frame(epoch = exp4_data$x,
                        mse_17n = exp5_data$y,
                        mse_7n = exp4_data$y,
                        mse_34n = exp6_data$y)
exp_data2 <- melt(exp_data2, id.vars = "epoch")
```

```
exp_data2 %>%
  ggplot() +
  geom_line(aes(x = epoch, y = value, color = variable)) +
  scale_colour_manual(name = NULL,
                      labels = c( "7 nodes", "16 nodes(baseline)", "30 nodes"),
                      values = c("black", "blue", "red")) +
  scale_y_continuous(limits = c(0, 30), breaks = seq(0, 30, 10)) +
  labs(x = "epoch",
       y = "estimated validation mse loss",
       title = "Effect of changing the number of nodes in
       the hidden layer on MSE")

#COMPARISON
#(a)       Cross-validated accuracy
gbm_cv <- data.frame(num_trees = 1:gbm1$n.trees,
                     value = gbm1$cv.error)
min_gbm <- min(gbm_cv$value)
gbm_cv %>%
  ggplot() +
  geom_line(aes(x = num_trees, y = value)) +
  labs(x = "Tree",
       y = "cv error",
       title = "GBM algorithm Cross-Validated Accuracy")

nn_cv <- average_mse_history
min_cv <- min(nn_cv$validation_mse)
nn_cv %>%
  ggplot() +
  geom_line(aes(x = epoch, y = validation_mse)) +
  labs(x = "Epoch",
       y = "validation mse loss",
       title = "DNN algorithm Cross-Validated Accuracy")

#(b)      Computational Time to Train Models
comp_time <- data.frame(model = c("GBM", "DNN", "naive"),
                        train_time = round(c(traintime_gbm1, time_nn1, time_naive), 2),
                        tune_time = round(c(tunetime_gbm1, "", ""), 2),
                        total_time = round(c(totaltime_gbm1, "", time_naive), 2))
knitr::kable(comp_time)
```