Comparing Machine Learning and Artificial Neural Networks for Predicting the Australian Unemployment Rate

Blake Wang

October 2023

Abstract

The Australian unemployment rate is intricately linked to several economic factors, with recession and inflation serving as significant determinants. This report investigates several historical economic indicators from December 1980 to March 2023, aiming to shed light on the implications of these indicator fluctuations in predicting future unemployment rates.

The primary objective of this project is to assess the predictive performance of various statistical methods for the Australian unemployment rate from the March 2021 to March 2023. It seeks to identify reliable techniques for forecasting unemployment rates by considering the impact of economic indicators and uncertainties introduced by the Covid-19 Pandemic.

This study used conventional Machine Learning (ML) methods and an Artificial Neural Network (ANN) to predict the Australian unemployment rate. Our analysis found that several ML methods still outperformed ANN for unemployment rate prediction according to currently available dataset. Furthermore, there is an extended range of potential for ANN, and their optimal performance necessitates a more extensive dataset.

The findings of this study underscore the vital role of data-driven predictive models in understanding and preparing for fluctuations in the Australian unemployment rate. These insights can inform policy decisions and strategies to mitigate the impact of economic downturns on the labour market, ultimately benefiting society. However, according to the accuracy of our prediction, predicting the Australian unemployment rate remains a challenging task, especially during periods with more uncertainties, such as the Covid-19 Pandemic. To enhance the prediction accuracy, it is recommended to consider additional features, remove outdated historical data, and explore other advanced methods.

1. Introduction

The Australian unemployment rate is profoundly influenced by recession and high inflation (Reserve Band of Australia). During the recession of 1982-1983, there was a notable spike in unemployment, in which the unemployment rate exceeded 10% in certain quarters. This trend began to subside towards the end of 1983. Throughout the rest of the 1980s, the unemployment rate gradually decreased. However, in the early 1990s, Australia faced another recession, which led to a rapid increase in the unemployment rate. This rate remained over 10% for more than two years and reached an all-time high of 11.22% in December 1992. Subsequently, the unemployment rate followed a downward trajectory until the 2008 global economic crisis. During this crisis, there was a significant increase in unemployment rates over the two quarters. Nevertheless, the Australian labour market demonstrated resilience during the economic crisis. The rate remained relatively stable, typically ranging from 5 to 6%.

During the COVID-19 pandemic, Australia implemented strict border controls and lockdown measures. Several industries, particularly those reliant on travel, have been severely affected, leading to expectations of a substantial rise in the unemployment rate. The first two quarters following the imposition of the COVID-19 restrictions witnessed a surge in unemployment. However, the unemployment rate began to drop in the subsequent quarter. Several factors may have contributed to this ease, including stringent border controls, lack of immigration, and proactive measures by the Australian government, such as the implementation of programs like Job Keeper and Job Seeker (Munawar et al., 2021).

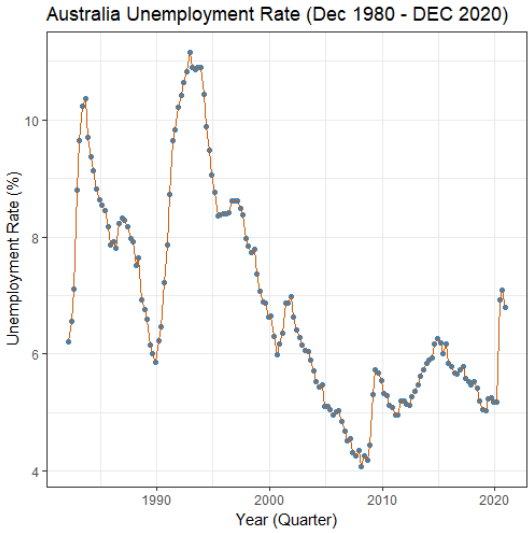


Figure 1 Australian Unemployment Rate 1980 - December 2020 (Australian Bureau of Statistics, 2023)

It's important to note that increases in the unemployment rate do not typically occur suddenly but are often preceded by identifiable signals, such as recessions, inflation (Reserve Band of Australia). Therefore, we can use these economic factors to predict the unemployment rate. Accurate predictions of the unemployment rate can serve as guidelines for governments to apply or stop interventions to reduce potential societal issues and balance spending (Reinhart & Rogoff, 2009).

Therefore, this project aims to evaluate the prediction performance of conventional Machine Learning (ML) methods and Artificial Neural Network (ANN) on the Australian unemployment rate from March 2021 to March 2023. This project is using R Studio 2023.09 (Posit team, 2023) for performing all data analysis tasks.

2. Data Pre-processing

The data were gathered quarterly from December 1982 to March 2023 by the Australian Bureau of Statistics (ABS). A total of 166 observations were included in the dataset. The dataset includes the unemployment rate as the response variable and seven predictor variables. An overview of the data is presented in *Table 1*.

Table 1: Overviews of dataset

Variable Name	Description	Data type	Variable Type	Range
Y	unemployment rate (%)	Numeric	Continuous	3.473 ~ 11.149
X1	change in Gross domestic product per Capita (%)	Numeric	Continuous	-6.8 ~ 3.8
X2	Change in Government final consumption expenditure (%)	Numeric	Continuous	-11.4 ~ 14.9
X3	change in final consumption expenditure of all industry sector (%)	Numeric	Continuous	-8.2 ~ 6
X4	Term of trade index (%)	Numeric	Continuous	-9.7 ~ 11.3
X5	Consumer Price Index of all groups (CPI)	Numeric	Continuous	30.8 ~ 133.70
X6	Number of job vacancies measured in thousands	Numeric	Continuous	28.4~ 454.2
X7	Estimated Resident Population in thousands.	Numeric	Continuous	15122 ~ 26268
Year_Q	Quarterly time line (March 1982 – March 2023)	Date	Ordinal	

Missing data imputation

The dataset contained missing data, as shown in *Table 2*. Specifically, the Number of job vacancies (X6) was missing during the 2008 economic crisis. Determining whether these missing data are "Missing at Random" (MAR) or "Not Missing at Random" (NMAR) is challenging. In this project, we assumed MAR because some recession periods were available, such as and 1982-1982, 1991-1993 which more profoundly affected the unemployment rate.

Table 2 Missing Data in the dataset

	Year_Q	Y	X1	X2	X3	X4	X5	X6	X7
107	2008-09-01	4.185270	0.1	3.2	0.1	6.1	92.7	NA	21366.0
108	2008-12-01	4.441308	-0.9	0.9	0.2	-2.2	92.4	NA	21475.6
109	2009-03-01	5.307405	0.4	-1.1	0.3	-5.0	92.5	NA	21601.7
110	2009-06-01	5.728484	0.2	1.3	1.4	-9.7	92.9	NA	21691.7
111	2009-09-01	5.683525	-0.1	-0.5	0.3	-1.8	93.8	NA	21788.1
165	2023-03-01	3.581322	-0.2	0.7	0.2	2.8	132.6	442.2	NA
166	2023-06-01	3.576919	NA	NA	NA	NA	133.7	432.2	NA

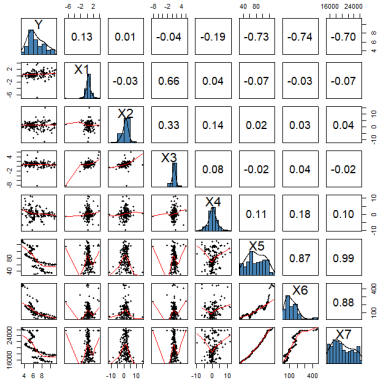


Figure 2: pairs.panels Plot of original dataset

Furthermore, from *Figure 2*, we identified a strong relationship between X6 and X5 with correlation coefficient $r = 0.87$. Thus, predictive imputation is a suitable approach to address the missing values in X6. Additionally, to capture nonlinear relationships, we employ multiple imputation methods using the *mice()* function to test different predictive algorithms.

To evaluate the quality of imputation, we use Support Vector Machines (SVM) on the training data. Visualisation tools were used to assess the naturalness of the imputed values.

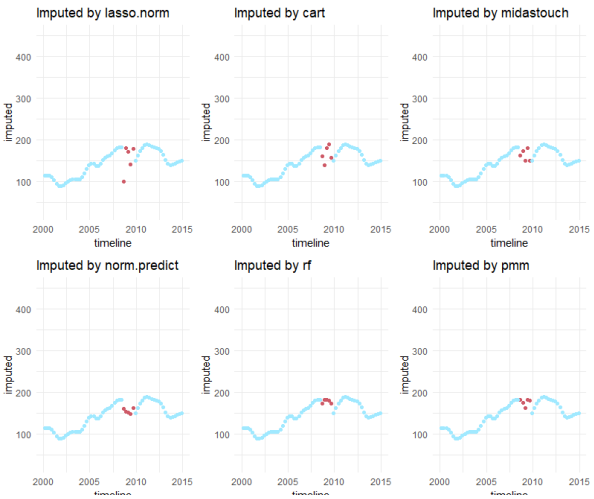


Figure 3: Imputation result on X6 using different method

The results in *Figure 3* indicate that linear predictive (*norm.predict*) and random forest tree (*rf*) imputation resulted in a smoother curve. We compared the performance results using SVM, as shown in *Table 3*. Linear predictive imputation performed relatively better. Consequently, we decided to employ linear predictive imputation to fill in the missing data for X6.

Table 3: Mice imputation result validate on SVM

Imputation Methods	parameter in mice package	MSE in SVM
Lasso linear regression	lasso.norm	2.2569
Classification and regression trees	cart	2.6737
Weighted predictive mean matching	midastouch	2.5733
Linear regression, predicted value	norm.predict	2.5228
Random forest imputations	rf	2.7311
Predictive mean matching	pmm	2.715

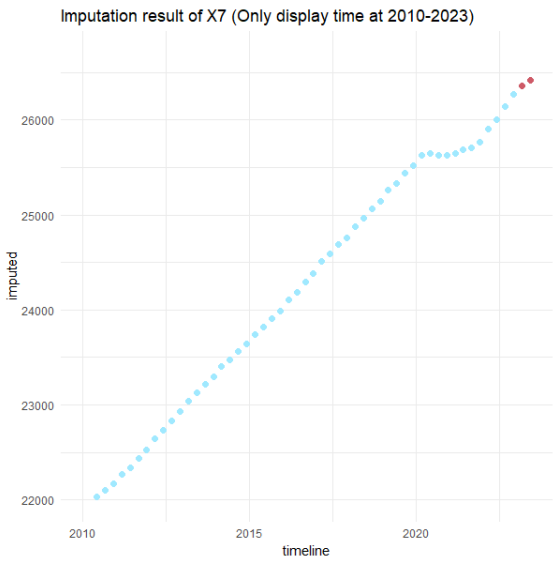


Figure 4: Imputation results of X7 (Timeframe display 2010-2023)

Figure 3 shows that the growth of the Australian population had nearly linear with the timeline in the last decade. However, during the COVID-19 pandemic, when Australia closed its borders, population growth was nearly halted. This suggests that, if we use the entire dataset to predict the last two missing values of X7, the predictions could be either too high or too low.

As shown in *Figure 4*, in the post-COVID period, population growth returned to a more typical pattern. To impute the missing data in X7, we simply used the data from the last five quarters to predict the missing values. The results (red points) of this imputation were reasonable and realistic.

Imputing the last row is a bit challenging since it contains four different missing values. It is difficult to apply predictive imputation owing to the lack of

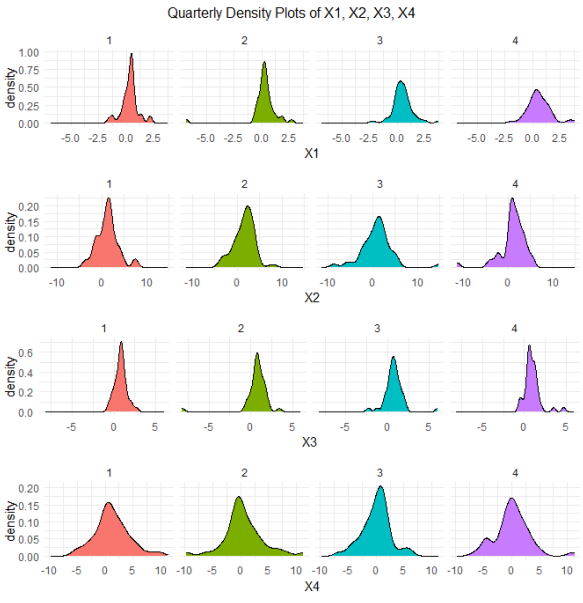


Figure 5 Density plots of X1, X2, X3, X4 by quarters

sufficient predictors. To address this, we assumed that X1, X2, X3, X4 are related to seasonal factors. As *Figure 5* shows, we noticed that the data followed a normal distribution, with some of it being right-skewed. Therefore, we assumed that the missing data were missing at complete random (MACR). In addition, there is only one missing value in each column. We simply applied the median and mean to impute these missing data. We applied the mean of the second quarter for non-skew data X2 and X4, and the median for skew data X1 and X3. Furthermore, we understand that there is not a best imputation method, and performance validation of different machine learning methods might yield different results. These methods we chose only specify on this application.

Finally, after imputation, we split the dataset into two sets: December 2021 – March 2023 as the test set and the remaining data as the training set. The training set had 156 observations, and the test set had 10 observations.

3. Machine Learning

In this section, we explored various machine learning methods for predicting the Australian unemployment rate from December 2021 to March 2023. To validate these methods, we conducted the prediction on both the training and test datasets and compared the Mean Square Error (MSE) of the results.

3.1. Regularization Methods

We will use *glmnet* package to apply two types of regularisation methods: L1 (Ridge) and L2 (Lasso). Lasso and Ridge regression introduce a penalty term add into the residual sum of squares (RSS), and this penalty is determined by the coefficients of the model. The key difference between Lasso and Ridge lies in how this penalty is computed. Ridge uses the square of the coefficients, whereas Lasso uses the absolute values of the coefficients. As the regularization parameter (λ) decreases, the influence of small coefficients on the model decreases. In Lasso, this reduction can push small coefficients to zero. When λ is sufficiently small, some coefficients can become exactly zero, effectively excluding corresponding features from the model. However, this may reduce predictive power when the dataset contains multicollinearity issues. Ridge, on the other hand, can reduce the impact of multicollinearity without necessarily eliminating any features. Given that we have only seven predictors, and dropping any of them might reduce the predictive power, we expect that Ridge will perform better than Lasso in this dataset. Even so, a performance comparison is always practical, process, and results, as shown in *Tables 4-5, Figure 6.*

Table 4: Best model of Lasso and Ridge regression

Methods	Hyperparameter	Method to get hyperparameter
Lasso	lambda = 0.0004690595	Cross validation -> cv.glmnet()
Ridge	lambda = 0.1399508	Cross validation -> cv.glmnet()

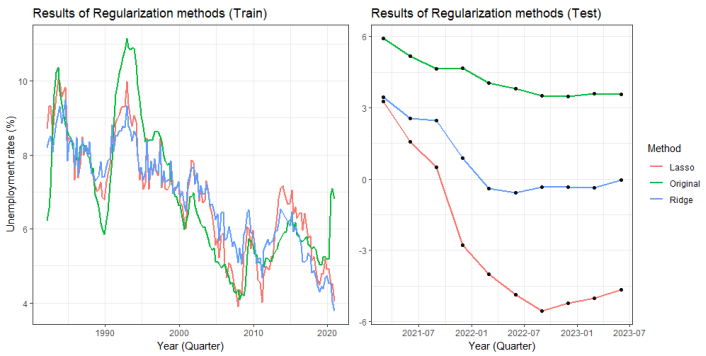


Figure 6: Prediction Results of Lasso and Ridge Regression

Table 5: Performance of Lasso and Ridge Regression

	MSE (Test)	MSE (Train)
Lasso	53.13	0.845
Ridge	12.867	1.101

The results demonstrate that Ridge outperformed Lasso as expected, with a mean squared error (MSE) of 12.86. This suggests that, despite the presence of multicollinearity among predictors, retaining these predictors results in better predictions. *Figure 6* (left) shows that both methods struggled to capture the rapid increase in the unemployment rate during the COVID-19 pandemic. Therefore, these two methods may not be suitable for in this dataset.

3.2 Tree-based method

In this section, we assessed the performance of the tree-based machine learning methods. These methods included Classification and Regression Trees (CARTs), Random Forest (RF), and Boosting methods.

Classification and Regression Trees

CARTs are valuable tools for decision-making tasks when the relationship between the input features and the target variable can be effectively represented using a tree structure. It is interpretable and can handle both the regression and classification problems. However, in our dataset, there is no clear nonlinear decision boundary, and CARTs might not perform well.

Random Forest and Bagging

Bagging creates multiple subsets of training data through bootstrapping and each subset contains the same variables of the dataset. RF performs the same way, but it performs a feature selection. Therefore, RF can avoid overfitting in datasets containing multicollinearity variables. In our dataset, we have identified some strong correlations within predictors. Therefore, in comparison with Bagging, RF is more suitable for this project.

Boosting

This project we will apply Gradient Boosting (GB) on two different package *gbm* and *xgboost*. GB evaluates the weight of trees and generally performs well in scenarios with complex relationships and feature interactions. Extreme Gradient Boosting (XGB) offers more hyperparameters for fine-tuning. We evaluated both methods to determine which method achieved better performance on this dataset.

To construct the best-suited model for each method, we performed cross-validation and grid searching to identify hyperparameters. In addition, although scale differences exist in our dataset, tree-based

methods generally do not require scaling. *Table 6* shows the results of the best-suited hyperparameters, and the method used to search for these hyperparameters.

Table 6: Hyperparameters setting of Tree-based Method

Methods	Package	Function	Hyperparameter	Method to get Hyperparameter
CART	tree	tree()	best prune = 5	Cross validation -> cv.tree()
Random Forest	randomForest	randomForest()	mtry = 4, tree = 35	Grid searching + compare oob error
Gradient Boosting	gbm	gbm()	ntree=200, interaction.depth=1, shrinkage=0.1	Grid searching -> train(...method= "gbm")
Xgboost	xgboost	xgboost()	gamma = 0.1, eta = 0.1, max_depth = 3	Grid searching -> train(...method= "xgbTree")

The results (*Figure 7*) indicate that while some methods closely capture the curve of the original data when applied to the training dataset, their accuracy significantly decreases when evaluated on the test dataset. This suggests a potential issue of overfitting.

Table 7 shows that CART achieved the lowest MSE but consistently produced the same prediction. Therefore, GB performed the best among tree-based methods.

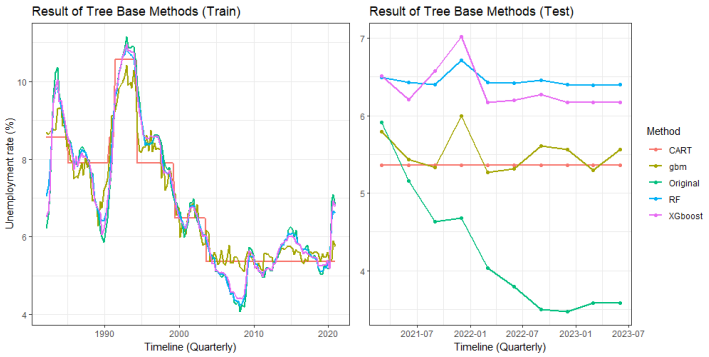


Figure 7: Prediction Results of Tree-based Methods

Table 7: Performance of Tree-based Methods

Method	MSE (Test)	MSE (Train)
RF	5.508	0.034
CART	1.901	0.559
GB (gbm)	2.182	0.391
XGB (Xgboost)	4.948	0.026

3.3 Support Vector Machine

In this section, we analysed the performance using a Support Vector Machine (SVM), which is a potent machine learning algorithm suitable for various problem types, including both linear and nonlinear regression and classification tasks. To find a suitable kernel and hyperparameters for our dataset, we used grid searching techniques. It is worth noting that we set the scale as True in both the tuning and training processes, and there is a large-scale difference between some variables. The grid search results, and other model training settings are listed in *Table 8*, and the prediction results are shown in *Figure 8*, *Table 9*.

Table 8: Best Tuning hyperparameters and presetting of SVM

Methods	Package	Function	Hyperparameter	Method to get hyperparameter
Radial	e1071	svm(…scale =TRUE)	cost=10, gamma=0.09	Grid searching -> tune(… scale = TRUE)
Linear			cost=0.5	
Polynomial			degree = 3, cost = 0.1, gamma=0.1	

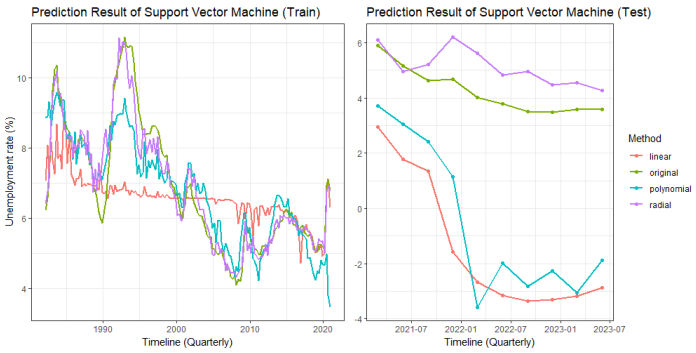


Figure 8: Prediction Results of SVM (seed: 123)

Table 9 Performance of SVM (seed: 123)

Method	MSE - Test	MSE -Train
Radial	1.096	0.223
Linear	35.006	0.936
Polynomial	199.446	1.871

The result indicates that the linear and polynomial SVMs can be disregarded, whereas the radial kernel shows good performance in both the training and test sets.

3.4 Summary of Machine Learning Methods

Based on our results, the optimal choice of the ML model is between GB and SVM using the radial kernel. SVM demonstrated better performance on the test set. *Figure 9* shows that it has 115 support vectors, and considering the dataset size, the model is likely to be robust with new data. However, GB has the advantage of being more interpretable. *Figure 10* provides the ranking of relative influence, showing that the number of job vacancies (X6) and Consumer Price Index (CPI) (X7) have a significant influence on the prediction. Consequently, when selecting between these two methods, it is crucial to consider their real-world applications.

```
Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.09
    epsilon:  0.1


Number of Support Vectors:  115
```

Figure 9: Summary of SVM

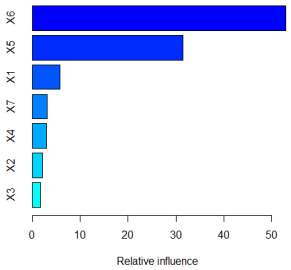


Figure 10: Summary of Gradient Boosting

4. Neural Network

In this section, we assessed the performance of ANN on the dataset. Given the relatively small size of our dataset, we initially employed the *neuralnet* package (Fritsch, 2019) to construct a simple Vanilla Neural Network (VNN). However, the results obtained from the *neuralnet* package were not as promising as those obtained using the machine learning methods. To assess better hyperparameter tuning and layer settings. We used Keras (Chollet & et al., 2015) and TensorFlow (Abadi et al., 2015), which were connected to Python 3.11.

Keras offers great flexibility in configuring various hyperparameters, layers, and optimisers. However, this flexibility has a potential drawback because the process of finding the optimal combination and tuning these parameters can be time-consuming. We searched for the optimal model by discovering the performance change when tuning each hyperparameter. *Table 10* lists some of the important hyperparameters, layer configurations, and tuning choices. We also scaled the dataset to ensure the performance.

Table 10: ANN Hyperparameters

Hyperparameter	Descriptions	Tuning setting
Activation function	Because this is a regression problem, we adopt the Rectified Linear Unit (ReLU) as the below Equation show.	$f(x) = \max(0, x)$
Optimizer	In our project, we will primarily focus on evaluating two different optimizers: RMSprop and Adam. RMSprop is a gradient-based optimization technique that incorporates an adaptive learning rate (Hinton et al., 2012). Adam, on the other hand, is a first-order gradient-based algorithm designed for handling stochastic objective functions (Kingma & Ba, 2014).	RMSprop and Adam
Dropout rate	We introduce two dropout layers into on layer 1 and 2, our network. A dropout layer randomly deactivates a percentage of activations, which can be advantageous in mitigating overfitting issues,	0.3, 0.4
Batch size	Determine the number of samples used for each gradient update.	10, 20, 40
Numbers of layers	We are going to test performance adding layers from 1 to 5	1, 2, 3, 4, 5
numbers of units in the hidden layers (Neurons)	Determining the optimal number of units can be challenging, but in this case, we will explore the change units in the first layer.	Layer 1 -> 16 , seq(32, 64, by = 4) Layer 2 -> seq(16, 64, by =8) Layer 3 -> 16 units Layer 4 -> 8 units
the number of epochs	which determines the iterations for training the model. We will set this to 100.	100

It is important to acknowledge that there is an extensive range of hyperparameter settings, making it impossible to test all of them. In this project, our aim was to find a relatively effective ANN model, although it may not be the best. After presetting, we used *tuning_run()* in the *tfrun* package for hyperparameter tuning. *Table 11-13* list some examples of the tuning results.

Table 11 Layer change testing, all setting shows in Appendix 1

Layer	Training Loss	Test Loss	Sum of Loss
1	2.4769	7.0534	9.5303
2	4.5619	6.0386	10.6005
3	1.7003	4.7876	6.4879
4	1.1872	3.6722	4.8594
5	1.1071	5.8898	6.9969

Table 12: Neurons Tuning, all setting in Appendix 2

Neurons of Layer 1	Training Loss	Test Loss	Sum of Loss
16	0.9862	5.3609	6.3471
32	0.973	5.3555	6.3285
36	1.0572	4.8939	5.9511
40	0.9845	5.1039	6.0884
44	0.9182	6.1945	7.1127
48	0.9582	5.3039	6.2621
52	0.9737	4.3821	5.3558
56	1.0453	5.5748	6.6201
60	1.1395	4.8432	5.9827
64	1.1633	5.5429	6.7062

Table 13: Neurons Tuning, all setting in Appendix 3

Neurons of Layer 2	Training Loss	Test Loss	Sum of Loss
16	0.9689	4.0778	5.0467
24	0.9212	5.3799	6.3011
32	1.1037	6.465	7.5687
36	1.1064	6.5456	7.652
48	0.7265	5.2211	5.9476
56	1.2161	7.1936	8.4097
64	1.0151	6.3515	7.3666

After tuning all the parameters, we found that increased layers can potentially obtain better results; however, if the layer is too high, it tends to overfit. In addition, fewer neurones and a higher dropout rate are less prone to overfitting. We also discovered that simultaneously achieving low training and test losses is difficult. It is necessary to determine a trade-off between the training and test losses. We identified the optimal model by the minimal sum of the training and test losses, as highlighted in *Tables 11-13*. We selected the best-tuned models for the performance analysis, as shown in Table 14. *Finally,* we used these models to make predictions for both the training and test sets, as shown in *Figure 13*.

Table 14: The best model setting and performance

Optimizer	Test Loss	Training Loss	dropout1	dropout2	flag_batch	flag_unit1	flag_unit2	epochs
Adam	3.6722	1.1872	0.3	0.3	10	26	10	100

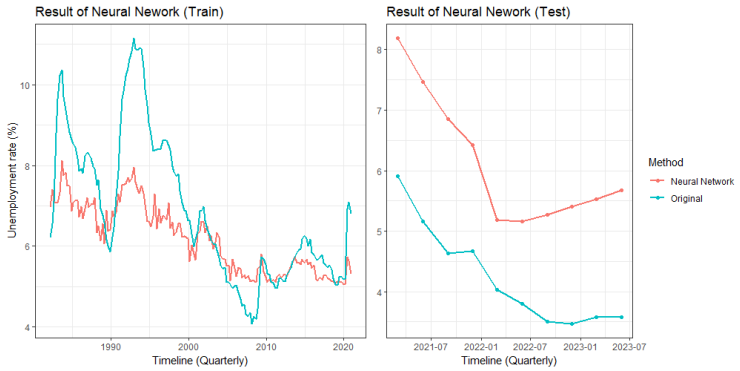


Figure 13: Result with best trade-off points (test loss = 3.6722)

According to the results, in the training predictions, most of the time, the model underestimated unemployment. Meanwhile, in the test predictions, the model overestimated this rate. Therefore, the model is not robust for predicting the unemployment rate. This may be due to the following reasons.

- Inadequate Data Size: ANN performance is highly dependent on the quantity of data. The dataset might be too small for this black-box method, where the model construction process is unknown, and adding more observations and features could potentially enhance the performance.
- Inappropriate Hyperparameter Selection: The choice of hyperparameters for our dataset may be suboptimal. Further analyses can be conducted using different hyperparameter settings.
- Unsuitable Model Architecture: Our model architecture may not be suitable for the dataset.

4. Comparison and Suggestion

Cross Validation

In this section, we performed a comparison between our selected ML models (GB and SVM) and ANN by using k-fold cross-validation, as this is a time series dataset, to keep each fold representative, we divide data by a series of quarter. This method includes using future data to predict past data. The purpose was to evaluate the performance of the selected methods on the dataset. This process is illustrated in *Figure 14*.

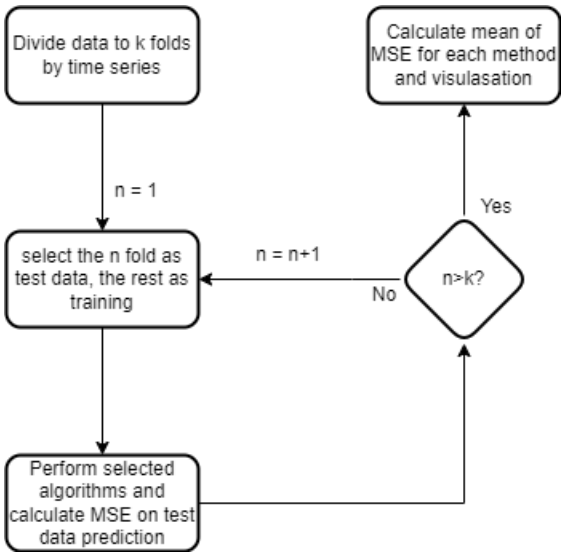


Figure 14 Process of k-fold cross validation

This process was conducted on both the training and full datasets. In the full dataset, we removed the last row which had previously imputed four data points; this not only kept the number of observations on each fold equal but also reduced the potential bias by the imputation. This does not affect our cross-validation results.

Additionally, we applied the same optimal hyperparameters discovered in the previous sections to the ML and ANN training processes. Although different datasets might have different optimal hyperparameters, this rule helps us produce consistent results and fair comparisons. *Figure 15-16*, Table

Table 15: Cross validation results

	12-folds on training data (mean MSE)	15-folds on full data (mean MSE)
Neural Network	6.07	7.25
SVM	1.06	0.974
Gradient Boosting (gbm)	1.03	0.697

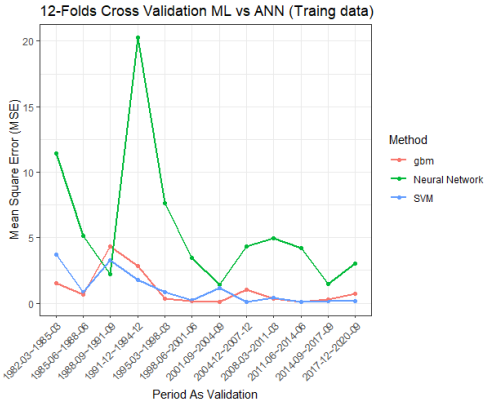


Figure 15: k-folds cross-validation on training dataset

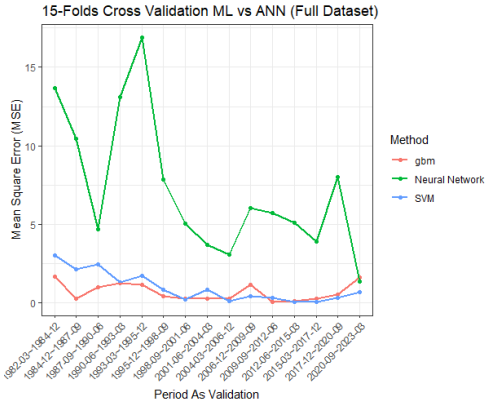


Figure 16: k-folds cross-validation on full dataset

According to these results, the Neural Network exhibited the worst performance in both sections. The boosting method outperformed SVM in the periods before 2004, while SVM had better prediction of more recent periods, which consistently produced a low MSE. We also recorded the training time for

each model, as shown in Table 13. GB ran the fastest, but the SVM is only slightly behind. And its time consuming to train a Neural Network model.

Table 16: Process time on 15-fold Cross validation

	Total time in second (15 iterations)	Each Iteration (mean)
Neural Network	277.58	18.5
SVM	0.152	0.01
GB (gbm)	0.131	0.008

In conclusion, of all methods we discuss on this subject, both SVM and GB are applicable to do predictions on this dataset. The SVM model might better predict near-future data because it contains relatively large support vectors (113). GB might perform better predictions during recessions, because its decision is dominated by the number of job vacancies and CPI. ANN have considerable potential. However, this requires time and more data to be discovered. Our data are too small for training a good ANN model.

Suggestion

In summary, accurately predicting the unemployment rate using the current dataset remains challenging. In particular, the periods during the Covid-19 restriction and post Covid-19 boarder reopening introduced numerous uncertainties. Several strategies can be employed to enhance the prediction of Australia's unemployment rate, particularly in the face of unforeseeable events such as recession.

Introducing more features to the dataset

Incorporate real-time data sources, such as job postings and hiring trends. And pay attention to a wide range of economic indicators beyond traditional ones, including consumer confidence, business sentiment, and trade data. These indicators might provide insights into the overall health of the job market and provide help in reacting quickly to changes in the job market (Li et al., 2014). Furthermore, considering temporal factors such school graduation, harvest, and holiday season, as they might has somewhat connection to the job market.

Remove outdated and unrepresentative historical data

As our analysis shows, unemployment in the 1980s and 1990s fluctuated greatly, and it might not represent the current job market. Removing these periods may reduce data noise and improve the data quality.

Consider other algorithms

Explore advanced deep learning techniques which can handle time series data such as Recurrent Neural Networks, Long Short-Term Memory Networks, Autoregressive Fractionally Integrated moving Average (ARFIMA) and Convolutional Neural Networks (CNNs), which has been proved that can well deal with time series forecasting (Borovykh et al., 2017; Hewamalage et al., 2021; Li et al., 2014; Malhotra et al., 2015), so they may perform better in predicting unforeseen events.

Reference

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., . . . Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. In.

Australian Bureau of Statistics. (2023). *Labour Force, Australia*. https://www.abs.gov.au/statistics/labour/employment-and-unemployment/labour-force-australia/latest-release

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*.

Chollet, F., & et al. (2015). Keras. In.

Fritsch, S. G., Frauke (2019). *neuralnet: Training of Neural Networks*. In (Version 1.44.2) https://cran.r-project.org/web/packages/neuralnet/index.html

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, *37*(1), 388-427.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8), 2.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, Z., Xu, W., Zhang, L., & Lau, R. Y. (2014). An ontology-based Web mining method for unemployment rate prediction. *Decision Support Systems*, *66*, 114-122.

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long Short Term Memory Networks for Anomaly Detection in Time Series. Esann,

Munawar, H. S., Khan, S. I., Ullah, F., Kouzani, A. Z., & Mahmud, M. A. P. (2021). Effects of COVID-19 on the Australian Economy: Insights into the Mobility and Unemployment Rates in Education and Tourism Sectors. *Sustainability*, *13*(20), 11300. https://www.mdpi.com/2071-1050/13/20/11300

Posit team. (2023). RStudio: Integrated Development Environment for R. Posit Software. In *Posit Software*. PBC, Boston, MA.

Reinhart, C. M., & Rogoff, K. S. (2009). The aftermath of financial crises. *American Economic Review*, *99*(2), 466-472.

Reserve Band of Australia. *Recession*. R. B. o. Australia. https://www.rba.gov.au/education/resources/explainers/recession.html#:~:text=Given%20the%20costs%20of%20high,demand%20were%20compounded%20by%20drought.

Appendix 1: Layer 1 compare with Layer 4

Run		Run	
context	local	context	local
script	a3_nn_adam.R	script	a3_nn_adam.R
started	2023-10-24 11:57:22 GMT	started	2023-10-24 12:42:22 GMT
time	00:01:15	time	00:01:19
Metrics		**Metrics**	
loss	2.4769	loss	1.1872
val_loss	7.0534	val_loss	3.6722
Evaluation		**Evaluation**	
eval_	7.0534	eval_	3.6722

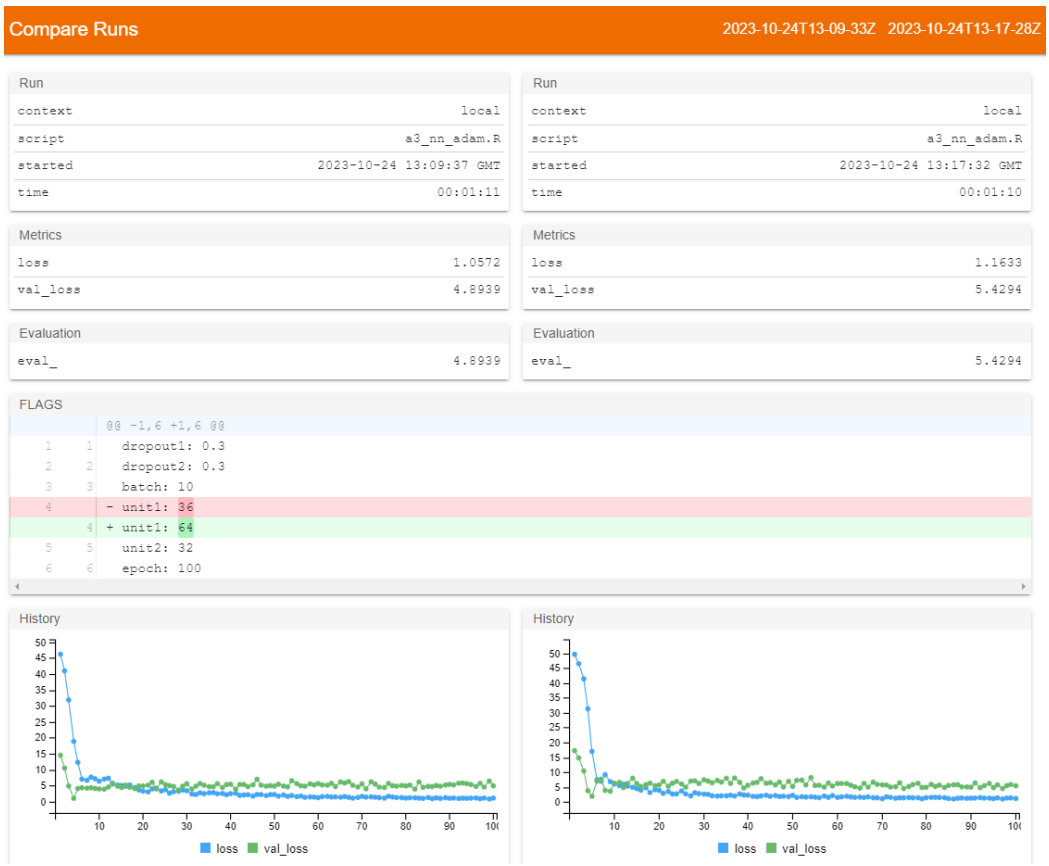
a3_nn_adam.R

```
@@ -42,10 +42,12 @@
  model %>%
    layer_dense(units = FLAGS$unit1, activation = 'relu',
                input_shape = ncol(scale.x.tr)) %>%
    layer_dropout(rate = FLAGS$dropout1) %>%
-   #layer_dense(units = FLAGS$unit2, activation = 'relu') %>%
+   layer_dense(units = FLAGS$unit2, activation = 'relu') %>%
-   #layer_dropout(rate = FLAGS$dropout2) %>%
+   layer_dropout(rate = FLAGS$dropout2) %>%
+   layer_dense(units = 16, activation = 'relu') %>%
+   layer_dense(units = 8, activation = 'relu') %>%
    layer_dense(units = 1)


# Set up the EarlyStopping callback -----------------------------------------
```

History

Appendix 2: Neurons 64 compare with Neurons 32 in Layer 1



Compare Runs 2023-10-24T13-09-33Z 2023-10-24T13-17-28Z

Run		Run	
context	local	context	local
script	a3_nn_adam.R	script	a3_nn_adam.R
started	2023-10-24 13:09:37 GMT	started	2023-10-24 13:17:32 GMT
time	00:01:11	time	00:01:10
Metrics		**Metrics**	
loss	1.0572	loss	1.1633
val_loss	4.8939	val_loss	5.4294
Evaluation		**Evaluation**	
eval_	4.8939	eval_	5.4294

FLAGS

```
@@ -1,6 +1,6 @@
  dropout1: 0.3
  dropout2: 0.3
  batch: 10
- unit1: 36
+ unit1: 64
  unit2: 32
  epoch: 100
```

History

Appendix 3: Neuron 64 compare with Neurons 16 in layer 2

Compare Runs 2023-10-24T13-43-18Z 2023-10-24T13-50-20Z

Run	
context	local
script	a3_nn_adam.R
started	2023-10-24 13:43:32 GMT
time	00:01:14

Metrics	
loss	0.9689
val_loss	4.0778

Evaluation	
eval_	4.0778

Run	
context	local
script	a3_nn_adam.R
started	2023-10-24 13:50:24 GMT
time	00:01:12

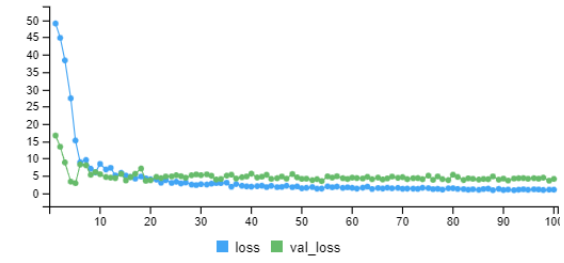
Metrics	
loss	1.0151
val_loss	6.3515

Evaluation	
eval_	6.3515

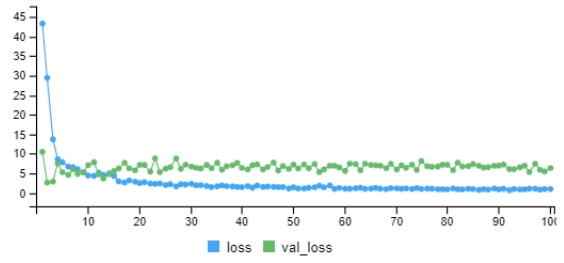
FLAGS

```
@@ -1,6 +1,6 @@
   dropout1: 0.3
   dropout2: 0.3
   batch: 10
   unit1: 52
- unit2: 16
+ unit2: 64
   epoch: 100
```

History



History



Appendix 4: R Code

Libraries

```
library(ggplot2)       # Visualisation
library(gridExtra)     # Visualisation
library(lubridate)     # time data
library(dplyr)         # data preprocess
library(tidyverse)     # For data wrangling
library(mice)          # Missing data imputation
library(glmnet)        # Lasso and Ridge
library(e1071)         # Support vector machine
library(tree)          # Classification and Regression Tree
library(randomForest)  # Random Forest and Bagging
library(gbm)           # Boosting with Gradient Boosting
library(xgboost)       # X Gradient Boosting
library(caret)         # Cross validation tuning, train fucntion
```

Data Pre-processing

```
# Read Data
aus.raw <- read.csv("aus_data_2023.csv",
                    header = TRUE)

## ----------------------Clean Data-----------------------------
# Convert Date to Datetype
aus.raw$Year_Q <- gsub("_", "-", aus.raw$Year_Q) # Replace "_" wi
th "-"
aus.raw$Year_Q <- toupper(aus.raw$Year_Q)        # Convert letter
 to upper
aus.raw$Year_Q <- as.Date(paste0(aus.raw$Year_Q, "-01"), # Add "-
01" in the end
                          format="%Y-%b-%d")     # Let R know our
 date format

# Arrange dataset by date
aus.raw <- aus.raw %>% arrange(Year_Q)

# Understanding the missing value
row.missing <- aus.raw[apply(is.na(aus.raw), 1, any),]
knitr::kable(row.missing)

ggplot(aus.raw[-(157:166),], aes(x = Year_Q, y = Y)) +
  geom_point(col = "steelblue") +
  geom_line(col= "#CD5C08") +
  labs(title="Australia Unemployment Rate (Dec 1980 - DEC 2020)",
       x = "Year (Quarter)",
       y = "Unemployment Rate (%)") +
  theme_bw()

summary(aus.raw)
psych::pairs.panels(aus.raw[,-1], hist.col = "steelblue",rug =F,
ellipses=F)
```

Imputation X7

```
# data is Miss at Random (MAR)

## ----------------------- Impute X7 ---------------------------
lm.X7 <- lm(data = aus.raw[(160:164),],  # remove affect by Covid
-19, we impute data by using the CIP from the last 5 quarters  15
4:160
            X7 ~ X5)

aus.temp <- aus.raw
# Impute the last two value of X7
aus.raw$X7[165:166] <- predict(lm.X7,
                               aus.raw[is.na(aus.raw$X7), ]
                               )

start_date <- as.Date("2010-01-03")  # Replace with your desired
start date
end_date <- as.Date("2023-06-08")    # Replace with your desired
end date
df <- data.frame(imputed = aus.raw$X7, timeline = aus.raw$Year_Q,
  original = aus.temp$X7)
ggplot(df, aes(x = timeline))+
                          geom_point(aes(y=imputed), col="#CE5A67
", size=2) +
                          geom_point(aes(y=original), col="#A0E9F
F", size=2) +
                          scale_x_date(limits = c(start_date, end
_date)) +
                          scale_y_continuous(limits = c(22000, 26
700))+
                          theme_minimal() +
                          labs(title = "Imputation result of X7
(Only display time at 2010-2023)")
## ----------------------- Impute X6 ---------------------------
aus.temp <- aus.raw[-166,] # Drop the last row and deal with the
X6 missing data and assign to a temporary dataset

# ---------------- Using mice to impute X6-----------------------
library(mice)
#library(VIN)
#library(missForest)

# Understanding the missing value
row.missing <- aus.temp[apply(is.na(aus.temp), 1, any),]
row.missing
knitr::kable(row.missing)
md.pattern(aus.temp,
           plot = TRUE,
           rotate.names = FALSE) # Find where is the missing data

# Imputation validation using SVM
methods = c("pmm", "rf", "norm.predict",
            "lasso.select.norm", "midastouch",
            "cart", "lasso.norm")
fig.list <- list()
# Set the x-axis limits to display a certain date range
start_date <- as.Date("2000-01-03")  # Replace with your desired
start date
end_date <- as.Date("2015-01-08")    # Replace with your desired
end date
for (method in methods){
  mse = 0
  for (i in 1:5){
  imputed_data <- mice(aus.temp[, -(1:2)],  # Ignore the Year_Q
                       m = 5,
                       method = method,
                       printFlag = FALSE, # Stop printing process
                       seed = i*200.        # Random seed change
                       )
  x1 <- cbind(Y = aus.temp$Y, complete(imputed_data, 5))
  ## -------------Split data--------------
  tr <- x1[1:156, ]
  ts <- x1[157:165,]
  svm.impt <- svm(Y ~ .,
                  data = tr,
                  kernel = "radial",
                  gamma = 0.1,
                  cost = 5,
                  decision.values = T,   # here we can find fit v
alue to find the ROC Curves
                  scale = T
                  )
  pred.svm <- predict(svm.impt,
                      newdata = ts)
  mse <- mse + mean((ts$Y - pred.svm)^2)
  }
  df <- data.frame(imputed = x1$X6, timeline = aus.raw$Year_Q[-16
6],  original = aus.raw$X6[-166])
  fig.list[[method]] <- ggplot(df, aes(x = timeline))+
                        geom_point(aes(y=imputed), col="#CE5A67
") +
                        geom_point(aes(y=original), col="#A0E9F
F") +
                        scale_x_date(limits = c(start_date, end
_date)) +
                        theme_minimal() +
                        labs(title = paste("Imputed by", metho
d, sep = " "))
  print(paste("mse", method, mse/5, sep = ":"))
}
# Figure Imputation 1
gridExtra::grid.arrange(fig.list$lasso.norm, fig.list$cart, fig.l
ist$midastouch,
                        fig.list$norm.predict, fig.list$rf, fig.l
ist$pmm, ncol=3)
```

Imputations

```
imputed_data <- mice(aus.temp[, -(2)], m = 5,
                     method = "norm.predict",
```

```
                    printFlag = FALSE,
                    seed = 123)
aus.temp <- complete(imputed_data, 5)

## --------- Imputation Row 166----------------
# Before we perfrom imputation we plot the histogram of X1-X4
quarter_data <- aus.raw %>%
  group_by(quarter = lubridate::quarter(Year_Q)) %>%
  mutate(quarter = as.factor(quarter))

## Density plot by quarter
x1.q <- ggplot(quarter_data) +
         geom_density(aes(x = X1, fill = quarter) ) +
         facet_grid(.~quarter) +
          theme_minimal() +
         theme(legend.position = "none")

x2.q <- ggplot(quarter_data) +
  geom_density(aes(x = X2, fill = quarter), ) +
  facet_grid(.~quarter) +
   theme_minimal()+
  theme(legend.position = "none")

x3.q <- ggplot(quarter_data) +
  geom_density(aes(x = X3, fill = quarter), ) +
  facet_grid(.~quarter)+
  theme_minimal()+
  theme(legend.position = "none")

x4.q <- ggplot(quarter_data) +
  geom_density(aes(x = X4, fill = quarter), ) +
  facet_grid(.~quarter)+
    theme_minimal()+
  theme(legend.position = "none")

gridExtra::grid.arrange(x1.q, x2.q, x3.q, x4.q, nrow=4, top=gri
d::textGrob("Quarterly Density Plots of X1, X2, X3, X4"))

row.166 <- aus.raw[166,]
row.166
row.166$X1 <- round(median(aus.temp$X1[113:153]),1)
row.166$X2 <- round(mean(aus.temp$X2[77:153]),1)
row.166$X3 <- round(median(aus.temp$X3[77:153]),1)
row.166$X4 <- round(mean(aus.temp$X4), 1)

# assign the a clean data
aus <- aus.raw

# Replace as Imputed data X6
aus[1:165, ]$X6 <- aus.temp$X6
# Replace as Imputed data X7
aus[1:165, ]$X7 <- aus.temp$X7
# Replace as Imputed data row 166
aus[166,] <- row.166
#write.csv(aus, file = "aus.csv", row.names = F)
rm(x1, x1.q, x2.q, x3.q, x4.q, df, aus.temp, fig.list, imputed_da
ta, row.missing,
   svm.impt, tr, ts, end_date, i, method, methods, mse, pred.svm,
 start_date, lm.X7, quarter_data, row.166)
```

Partition data

```
## -------------Training data--------------
aus.tr <- aus %>%
  filter(Year_Q < as.Date("2021-01-01")) %>%
  select(-Year_Q)

## ------------Test data---------------
aus.ts <- aus %>%
  filter(Year_Q > as.Date("2020-12-31"))%>%
  select(-Year_Q)
## -----------Test Year-----------------
Year_Q.ts <- aus %>%
  filter(Year_Q > as.Date("2020-12-31"))%>%
  select(Year_Q)
```

Machine learning

Ridge and Lasso

Function

```
# Build a new dataset with dummy variables
aus.tr.x <- model.matrix(Y ~ .,aus.tr)[,-1] # first column is int
ercept
aus.ts.x <- model.matrix(Y ~ .,aus.ts)[,-1] # first column is int
ercept

## ----------------------Lasso Function-------------------------
-----
las.fn <- function(train.x,        # *Input train data
                   train.y,
                   test.x,          # *Input test data
                   test.y
                  )
  {
  # Cross validation to find the best lambda
  cv.las <- cv.glmnet(x = train.x,
                      y = train.y,  # Response
                      alpha = 1,           # 1 for lasso
                      standardize = T)  # does a 10-fold cross va
lidation by default
  plot(cv.las)

  # The best lambda
  bestlam <- cv.las$lambda.min
  print(bestlam)
  # The Best Lasso Regression Model
  las.best <- glmnet(x = train.x,
                     y = train.y,
                     alpha = 1,
                     lambda = bestlam,
                     standardize = T)
  # Get Predictions
  # Test data
  pred.las <- predict(las.best,
                      s = bestlam,
                      newx = test.x,
                      type = "response"     # Predict the respons
es
                      )

  # Train data
  pred.las.train <- predict(las.best,
                      s = bestlam,
                      newx = train.x,
                      type = "response"     # Predict the respons
es
                      )
  results <- list("pred.test" = pred.las,
                  "mse.test" = mean((test.y - pred.las)^2),   # T
est data performance
                  "pred.train" =pred.las.train,
                  "mse.train" = mean((train.y - pred.las.train)^
2) # Train data performance
                  )
  return(results)
  }

## ----------------------Ridge Function-------------------------
------
rid.fn <- function(train.x,        # *Input train data predictors
                   train.y,
                   test.x,         # *Input test data predictors
                   test.y
                  ){

  # Cross validation to find the best lambda
  cv.rig <- cv.glmnet(x = train.x,
                      y = train.y,  # Response
                      alpha = 0,     # 0 for Ridge
                      standardize = T)  # does a 10-fold cross va
lidation by default

  plot(cv.rig)

  # The best lambda
  bestlam <- cv.rig$lambda.min
  print(bestlam)
```

```
  # The Best Lasso Regression Model
  rig.best <- glmnet(x = train.x,
                     y = train.y,
                     alpha = 0,
                     lambda = bestlam,
                     standardize = T)

  # Get Predictions
  # Test data
  pred.rig <- predict(rig.best,
                      newx = test.x,
                      type = "response"     # Predict the respons
es
                      )
  # Train data
  pred.rig.train <- predict(rig.best,
                      newx = train.x,
                      type = "response"     # Predict the respons
es
                      )
  results <- list("pred.test" = pred.rig,
                  "mse.test" = mean((test.y - pred.rig)^2),
                  "pred.train" = pred.rig.train,
                  "mse.train" = mean((train.y - pred.rig.train)^
2)  # Train performance
                  )
  return(results)
}
```

Result

```
set.seed(123)
result.las <- las.fn(train.x = aus.tr.x,
                  test.x = aus.ts.x,
                  train.y = aus.tr$Y,
                  test.y = aus.ts$Y)
result.las


result.rig <- rid.fn(train.x = aus.tr.x,
                  test.x = aus.ts.x,
                  train.y = aus.tr$Y,
                  test.y = aus.ts$Y)
# result of rig
result.rig


res.reg.test <- data.frame(result.las[[1]],
                           result.rig[[1]],
                           aus.ts$Y,
                           Year_Q.ts)
colnames(res.reg.test) <- c("Lasso", "Ridge", "Original", "Year_Q
" )

reg.g1 <- res.reg.test %>% pivot_longer(cols = c("Lasso", "Ridge
", "Original"),
                            names_to = "Method",
                            values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title="Results of Regularization methods (Test)", x = "Yea
r (Quarter)", y = NULL) +
  geom_line(size=1) +
  geom_point(col="#0F0F0F")+
  theme_bw()

res.reg.train <- data.frame(result.las$pred.train,
                            result.rig$pred.train,
                            aus.tr$Y,
                            aus$Year_Q[1:156])
colnames(res.reg.train) <- c("Lasso", "Ridge", "Original", "Year_
Q" )

reg.g2 <- res.reg.train %>% pivot_longer(cols = c("Lasso", "Ridge
", "Original"),
                            names_to = "Method",
                            values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title="Results of Regularization methods (Train)", x = "Ye
ar (Quarter)", y = "Unemployment rates (%)") +
  geom_line(size=1) +
  theme_bw() +
  theme(legend.position = "none")

grid.arrange(reg.g2, reg.g1, ncol=2)
```

Tree base

Functions

Random Forest Function

```
## ----------------------Random Forest Function----------------
---------
rf.fn <- function(train,
                  train.y,
                  test.x,
                  test.y)
  {
  # Out of Bag Error among different mtry values
  mtry.grid <- c(2, 4, 6, 7)
  oob.errors <- data.frame()
  for (mtry in mtry.grid) {
    rf.mod.temp <- randomForest(Y ~ ., data = train,
                                mtry = mtry, ntree = 500)
    mse <- mean(rf.mod.temp$mse)
    oob.errors <- rbind(oob.errors, c(mtry, mse))  # List of the
error rate
  }
  colnames(oob.errors) <- c("mtry", "mse")

  # The minimum OOB error rate
  mtry.best <- oob.errors[which.min(oob.errors$mse),][[1]]
  # Find Optimal Tree
  rf.mod.temp <- randomForest(Y ~ .,
                              data = train,
                              mtry = mtry.best,
                              ntree = 500)


  plot(rf.mod.temp$mse ,
       main = "Random Forest mse (mtry = 4)",
       ylab = "mse",
       xlab = "Number of Tree",
       type = "l", col = "steelblue")

  tree.best <- which.min(rf.mod.temp$mse)
  ## ----------------------Random Forest Optimal Model----------
----------------
  rf.mod <- randomForest (Y ~ .,
                          data = train, # No need for transform i
n tree based methods
                          mtry = mtry.best,         # number of v
ariables
                          ntree = tree.best,       # number of tr
ee
                          importance = TRUE # Consider Variable I
mportance to reduce computing time
                          )

  importance(rf.mod)     # Variable Important
  varImpPlot(rf.mod,
             main = "Summary of Random Forest Model") # Variable
Important plot

  ## ----------------------Random Forest Prediction-----------
----------------
  # Test data
  rf.pred <- predict(rf.mod,
                     newdata = test.x)
  # Train data
  rf.pred.train <- predict(rf.mod,
                     newdata = train[, -1])
  results <- list("pred.test" = rf.pred,
                  "mse.test" = mean((test.y - rf.pred)^2),
                  "pred.train" = rf.pred.train,
                  "mse.train" = mean((train.y - rf.pred.train)^2)
  # Train data performance
                  )
  return(results)
  }
```

CRAT

```
# --------------------------CRAT Function---------------------
---------------
CART.fn <- function(test.x,
                    test.y,
```

```
                    best.prune = 5)
  {
    CRAT.tree <- tree(Y ~ ., data = aus.tr)  # Regression Tree
    plot(CRAT.tree)
    text(CRAT.tree, pretty = 0)
    # Check weather prune a tree can improve the performance
    cv.CRAT.tree <- cv.tree(CRAT.tree)
    plot(cv.CRAT.tree$size, cv.CRAT.tree$dev, type = "b")

    # Prune tree
    prune.CRAT.tree <- prune.tree(CRAT.tree, best = best.prune)
    plot(prune.CRAT.tree)
    text(prune.CRAT.tree, pretty = 0)
    tree.pred <- predict(prune.CRAT.tree,
                         newdata = test.x)

    tree.pred.train <- predict(prune.CRAT.tree,
                               newdata = aus.tr[,-1])

    # Get Result
     results <- list("pred.test" = tree.pred,
                     "mse.test" = mean((test.y - tree.pred)^2),
                     "pred.train" = tree.pred.train,
                     "mse.train" = mean((aus.tr$Y - tree.pred.tra
in)^2)  # Train data performance
                     )
    return(results)
}
```

Boosting gbm

```
## ------------------------Boosting Function -------------------
---------------
gbs.fn <- function(train,
                   test.x,
                   test.y,
                   ntrees = 200){
  #   # Searching Grid
  # boost.grid <- expand.grid(
  #   interaction.depth = c(1, 2, 3),  # Different values of inte
raction depth
  #   n.trees = seq(10, 200, by = 10), # Range of number of trees
 to test
  #   shrinkage = c(0.1, 0.01),  # Learning Rate, the higher the
faster but less precise
  #   n.minobsinnode = 1
  # )

  # Set as 10-fold cross-validation
  # ctrl <- trainControl(method = "cv", number = 10) # 10-fold cr
oss-validation

  # Boosting Cross Validation Process
  # cv.boosting <- train(Y ~., data = train,
  #                      method = "gbm",
  #                      trControl = ctrl,
  #                      tuneGrid = boost.grid,
  #                      verbose = FALSE  # No Show progress
  #                      )

  # Best tune parameter
  # best.tune <- cv.boosting$bestTune
  # print(best.tune)
  boost.mod <- gbm(Y ~ .,
                   data = train,
                   distribution = "gaussian", # Here is a regress
ion problem
                   n.trees = 200,
                   interaction.depth = 1,
                   shrinkage = 0.1)
  bs.pred <- predict(boost.mod, newdata = test.x)
  bs.pred.train <- predict(boost.mod, newdata = train[, -1])
  results <- list("pred.test" = bs.pred,
                  "mse.test" = mean((test.y - bs.pred)^2),
                  "pred.train" = bs.pred.train,
                  "mse.train" = mean((train$Y - bs.pred.train)^2)
  # Train data performance
                  )
  return(results)
}
```

Boosting xgboost

```
## ------------------------boosting xgboost Tuning-------------
----------------------

# Set up cross-validation
param_grid <- expand.grid(
  nrounds = c(100, 200, 300),
  max_depth = c(3, 4, 5, 6),
  eta = c(0.01, 0.1, 0.3, 0.5),
  gamma = c(0.1, 0.4, 0.8),              # You can specify a rang
e of values for gamma if needed
  colsample_bytree = 1,   # You can specify a range of values for
 colsample_bytree if needed
  min_child_weight = 1,   # You can specify a range of values for
 min_child_weight if needed
  subsample = 1           # You can specify a range of values for
 subsample if needed
)

# Set up the train control for k-fold cross-validation
train_control <- trainControl(
  method = "cv", # Cross-validation method
  number = 5,   # Number of folds
  verboseIter = TRUE
)

# Hyperparameter tuning
tuned_model <- train(
  x = aus.tr[,-1],  # Features
  y = aus.tr$Y,          # Target variable
  method = "xgbTree",       # Specify the xgboost model
  trControl = train_control,
  tuneGrid = param_grid,
  metric = "MSE"            # Evaluation metric
)

best_params <- tuned_model$bestTune
best_model <- tuned_model$finalModel

xgb.fn <- function(train.x,
                   train.y,
                   test.x,
                   test.y)
  {
  params <- list(
                  objective = "reg:linear",
                  #alpha = 0.3,  # L1 regularization (optional)
                  #lambda = 2,   # L2 regularization (optional)
                  gamma = 0.1,  # Minimum loss reduction (optiona
l)
                  eta = 0.1,
                  max_depth = 3,
                  colsample_bytree = 1,
                  min_child_weight = 1,
                  subsample = 1
                  )
  dtrain <- xgb.DMatrix(data = as.matrix(train.x),
                        label = train.y)

  xgb_model <- xgboost(params = params, nrounds = 100, data = dtr
ain)

  dtest <- xgb.DMatrix(data = as.matrix(test.x), label = test.y)
  dtrain <- xgb.DMatrix(data = as.matrix(train.x), label = train.
y)
  xgb_pred <- predict(xgb_model, dtest)
  xgb_pred.train <- predict(xgb_model, dtrain)

  results <- list("pred.test" = xgb_pred,
                  "mse.test" = mean((test.y - xgb_pred)^2),
                  "pred.train" = xgb_pred.train,
                  "mse.train" = mean((train.y - xgb_pred.train)^
2)  # Train data performance
                  )
  return(results)
  }
```

Results

```
set.seed(123)

CART.results <- CART.fn(test.x = aus.ts[, -1],
                        test.y = aus.ts$Y)
```

```
rf.result <- rf.fn(train = aus.tr,
                   train.y = aus.tr$Y,
                   test.x = aus.ts[, -1],
                   test.y = aus.ts$Y)

boost.results <- gbs.fn(train = aus.tr,
                        test.x = aus.ts[, -1],
                        test.y = aus.ts$Y,
                        ntrees = 200)

xgb.results <- xgb.fn(train.x = aus.tr[, -1],
                      train.y = aus.tr$Y,
                      test.x = aus.ts[, -1],
                      test.y = aus.ts$Y)
# Prediction Comparison test
tree.df <- data.frame(rf.result[[1]], CART.results[[1]], boost.results[[1]],
                      xgb.results[[1]], aus.ts$Y, Year_Q.ts)

colnames(tree.df) <- c("RF", "CART", "gbm", "XGboost", "Original", "Year_Q")

tree.g1 <- tree.df %>% pivot_longer(cols = c("RF", "Original", "CART", "gbm", "XGboost"),
                           names_to = "Method",
                           values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Result of Tree Base Methods (Test)", x = "Timeline (Quarterly)",
       y = NULL) +
  geom_line(size=1) +
  geom_point()+
  theme_bw()

# Prediction Comparison train
tree.df.train <- data.frame(rf.result[[3]], CART.results[[3]], boost.results[[3]],
                            xgb.results[[3]], aus.tr$Y, aus$Year_Q[1:156])

colnames(tree.df.train) <- c("RF", "CART", "gbm", "XGboost", "Original", "Year_Q")

tree.g2 <- tree.df.train %>% pivot_longer(cols = c("RF", "Original", "CART", "gbm", "XGboost"),
                           names_to = "Method",
                           values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Result of Tree Base Methods (Train)", x = "Timeline (Quarterly)",
       y = "Unemployment rate (%)") +
  geom_line(size=1) +
  #facet_wrap(.~Method) +
  theme_bw() +
  theme(legend.position = "none")
grid.arrange(tree.g2, tree.g1, ncol=2)
# MSE Comparison
tree.mse <- data.frame("method" = c("RF", "CART", "gbm", "XGboost"),
                       "train.mse" = c(rf.result[[4]], CART.results[[4]], boost.results[[4]], xgb.results[[4]]),
                       "test.mse" = c(rf.result[[2]], CART.results[[2]], boost.results[[2]],  xgb.results[[2]])
                       )
knitr::kable(tree.mse)
```

SVM

SVM Tuning

```
## -------------------SVM Validation process-----------

# ------------------Kernel parameter setting--------------------
kernel.ls <- c("radial", "linear", "polynomial")
kernel.para <- list(list(cost = seq(0.1, 10, by=0.1),
                         gamma = seq(0.01, 1, by=0.01)),
                    list(cost = seq(0.1, 10, by=0.1)),
                    list(degree = c(2, 3, 4),
                         cost = c(0.1, 1, by=0.1),
                         gamma = c(0.01, 0.05, 0.1, 0.5)
                         )
                    )
```

```
# ---------Running Find Support Vector Machine Best Model--------
mod.svm <- function(kernel,      # a Vector of kernel name to test
                    parameter,  # a list of kernel tuning parameter
                    data,        # data to test
                    scale = TRUE
                    ){
  start <- Sys.time()
  mod.ls <- list()
  for (j in 1:length(kernel))
    {
    start.1 <- Sys.time()
    tune.out <- tune(svm,
                     Y ~ .,
                     kernel = kernel[j],
                     ranges = kernel.para[[j]],
                     data = data,
                     tunecontrol = tune.control(sampling = "fix"),
                     scale = TRUE    # Depend on the analysis
                     )
        model.name <- paste(kernel[j])
        mod.ls[[model.name]] <- tune.out
        print(paste(
                    kernel[j],
                    "has been completed",
                    sep=" "))

    }
    end.1 <- Sys.time()
        print(end.1 - start.1)
  end <- Sys.time() # Timer end
  print(paste("Total", (end - start), sep=":"))
  return(mod.ls)
}
```

SVM Fucntion

```
svm.fn <- function(train,
                   test.x,
                   test.y,
                   kernel.list,
                   best.radial,
                   best.linear,
                   best.poly){
  results <- list()
  for (i in kernel.list){
    if (i == "radial"){
      svm.full <- svm(Y ~ .,
                      data = train,
                      kernel = i,
                      #gamma = best.radial[[2]],
                      gamma=0.09,
                      #cost = best.radial[[1]],
                      cost=10,
                      decision.values = T,   # here we can find fit value to find the ROC Curves
                      scale = T
                      )
    }
    else if (i == "linear"){
      svm.full <- svm(Y ~ .,
                      data = train,
                      kernel = i,
                      #cost = best.linear[[1]],
                      cost = 0.5,
                      decision.values = T,   # here we can find fit value to find the ROC Curves
                      scale = T
                      )
    }
    else if (i == "polynomial"){
      svm.full <- svm(Y ~ .,
                      data = train,
                      kernel = i,
                      #cost = best.poly[[2]],
                      #degree = best.poly[[1]],
                      #gamma = best.poly[[3]],
                      cost = 0.1,
                      gamma = 0.1,
                      degree = 3,
                      decision.values = T,   # here we can find fit v
```

```
alue to find the ROC Curves
                  scale = T
                  )
    }
    pred.svm.train <- predict(svm.full,
                              newdata = train)

    pred.svm.test <- predict(svm.full,
                              newdata = test.x)

    results[[i]] <- list("pred.test" = pred.svm.test,
                         "mse.test" = mean((test.y - pred.svm.tes
t)^2),
                         "pred.train" = pred.svm.train,
                         "mse.train" = mean((train$Y - pred.svm.t
rain)^2))
    #print(paste(i, mean((aus.ts$Y - pred.svm.test)^2)), sep = ":
")
  }
 return(results)
}
```

Results

```
set.seed(1223)
#tune.result <- mod.svm(kernel.ls, kernel.para, aus.tr)
#tune.result$radial$best.parameters
svm.results <- svm.fn(train = aus.tr,
                      test.x = aus.ts[, -1],
                      test.y = aus.ts$Y,
                      kernel.list = kernel.ls,
                      best.radial = tune.result$radial$best.param
eters,
                      best.linear = tune.result$linear,
                      best.poly = tune.result$polynomial$best.par
ameters)
# Prediction Comparison Test
svm.df.test <- data.frame(svm.results$radial$pred.test,
                      svm.results$polynomial$pred.test,
                      svm.results$linear$pred.test,
                      aus.ts$Y,
                      Year_Q.ts)

colnames(svm.df.test) <- c("radial", "polynomial", "linear", "ori
ginal", "Year_Q")

svm.g1 <- svm.df.test %>% pivot_longer(cols = c("radial", "linear
", "polynomial", "original"),
                               names_to = "Method",
                               values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Prediction Result of Support Vector Machine (Tes
t)", x = "Timeline (Quarterly)",
       y = NULL) +
  geom_line(size=1) +
  geom_point()+
  theme_bw()

# Prediction Comparison train
svm.df.train <- data.frame(svm.results$radial$pred.train,
                      svm.results$polynomial$pred.train,
                      svm.results$linear$pred.train,
                      aus.tr$Y,
                      aus$Year_Q[1:156])

colnames(svm.df.train) <- c("radial", "linear", "polynomial", "or
iginal", "Year_Q")

svm.g2 <- svm.df.train %>% pivot_longer(cols = c("radial", "linea
r", "polynomial", "original"),
                               names_to = "Method",
                               values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Prediction Result of Support Vector Machine (Trai
n)", x = "Timeline (Quarterly)",
       y = "Unemployment rate (%)") +
  geom_line(size=1) +
  theme_bw() +
  theme(legend.position = "none")

# MSE Comparison
svm.mse <- data.frame("method" = c("radial", "linear", "polynomia
l"),
                      "train.mse" = c(svm.results$radial$mse.tra
in, svm.results$linear$mse.train, svm.results$polynomial$mse.trai
n),
                      "test.mse" = c(svm.results$radial$mse.tes
t, svm.results$linear$mse.test, svm.results$polynomial$mse.test)
                      )

knitr::kable(svm.mse)
grid.arrange(svm.g2, svm.g1, ncol=2)
```

Nueral Network

Nueralnet

Function

```
library(neuralnet)
nn.fn <- function(train.x,
                  train.y,
                  test.x,
                  test.y,
                  hidden,
                  threshold = 0.01){
  # Scale features
  tr.s <- data.frame(Y = train.y,
                     scale(train.x))
  ts.s <- data.frame(Y = test.y,
                     scale(test.x,))

  formula <- as.formula(Y ~ .)
  nn <- neuralnet(formula,
                  data=tr.s,
                  hidden=hidden,  # set number of hidden layer
                  linear.output=TRUE,
                  act.fct = "logistic",
                  threshold = threshold, # meaning if the change
of error < threshold the training will stop
                  stepmax = 1e+05
                  )
  nn.results <- compute(nn, ts.s[, -1])
  nn.results.train <- compute(nn, tr.s[, -1])
  results <- list("pred.test" = nn.results$net.result,
                  "mse.test" = mean((test.y - nn.results$net.resu
lt)^2),
                  "pred.train" = nn.results.train$net.result,
                  "mse.train" = mean((train.y - nn.results.train
$net.result)^2) # Train data performance
                  )
  return(results)
}
```

Results

```
hiddens <- list(c(5,2), c(10,5), c(20,10), c(30,10), c(40, 20), c
(60, 20), c(80, 50))
cv.nn.mse <- c()
for (hidden in hiddens){
  nn.result <- nn.fn(train.x = aus.tr[, -1],
                     train.y = aus.tr$Y,
                     test.x = aus.ts[, -1],
                     test.y = aus.ts$Y,
                     hidden = hidden)
  cv.nn.mse <- rbind(cv.nn.mse, nn.result$mse.test)
}
cv.nn.mse
cv.nn.mse <- c()
set.seed(123)
for (i in 1:100){
  nn.result <- nn.fn(train.x = aus.tr[, -1],
                     train.y = aus.tr$Y,
                     test.x = aus.ts[, -1],
                     test.y = aus.ts$Y,
                     hidden = c(20, 20))
  cv.nn.mse <- rbind(cv.nn.mse, nn.result$mse.test)
}
cv.nn.mse[-12, ]
mean(cv.nn.mse[-12, ])
nn.result
```

Keras

Essential library for ANN

```
library(keras)          # Deep Learning framework
library(tensorflow)     # funcitonal toolkit for building nerual n
etwork
```

```r
library(reticulate)    # building connection between R and Python
library(tfruns)        # Tuning neural network
library(tfestimators)  # Visualisation on run
# Python environment:
use_virtualenv("r-reticulate")
# Check if tensorflow is connecting
tensorflow::tf_config()
```

Fucntion

```r
dnn.fn <- function(train,
                   train.y,
                   test,
                   test.y,
                   FLAGS){
    # Scale training and test date
  scale.x.tr <- scale(model.matrix(Y ~ ., data = train))[,-1]
  scale.x.ts <- scale(model.matrix(Y ~ ., data = test))[,-1]

  # Define Model --------------------------------------------------
  set.seed(123)
  model <- keras_model_sequential()
model %>%
  layer_dense(units = FLAGS$unit1, activation = 'relu',
              input_shape = ncol(scale.x.tr)) %>%
  layer_dropout(rate = FLAGS$dropout1) %>%
  layer_dense(units = FLAGS$unit2, activation = 'relu') %>%
  layer_dropout(rate = FLAGS$dropout2) %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dense(units = 1)

  model %>% compile(loss = "mse",
                    optimizer = optimizer_rmsprop())
  # Training & Evaluation ----------------------------------------
  history <- model %>% fit(
    x = scale.x.tr,
    y = train.y,
    batch_size = FLAGS$batch,
    epochs = FLAGS$epoch,
    verbose = 1,
    validation_data = list(scale.x.ts, test.y),
    #callbacks = list(custom_early_stopping)
  )
  plot(history)
  pred.dnn <- predict(model, scale.x.ts)
  train.dnn <- predict(model, scale.x.tr)
  results <- list("pred.test" = pred.dnn,
                  "mse.test" = mean((test.y - pred.dnn)^2),
                  "pred.train" = train.dnn,
                  "mse.train" = mean((train.y - train.dnn)^2)  # Train data performance
                  )
  return(results)
}
```

Tuning Process

```r
# Run with Rmsprop
runs_Rmsprop <- tuning_run("a3_nn.R", flags = list(
  dropout1 = c(0.3, 0.4),    # Prevent over fitting
  dropout2 = c(0.3, 0.4),
  batch = c(10, 20, 40),
  unit1 = c(52, 78, 104),
  unit2 = c(52, 78, 104)
))
# Run with Adam
runs_Rmsprop <- tuning_run("a3_nn_adam.R", flags = list(
  dropout1 = c(0.3),
  dropout2 = c(0.3),
  batch = c(10),
  unit1 = c(16),
  unit2 = c(32)
))

# Quick Look on Run
latest_run()
view(ls_runs())
compare_runs(ls_runs(c(1, 2)))
copy_run("runs/2023-10-23T14-32-46Z", to = "best_runs")
```

Results

```r
# Run with Rmsprop
set.seed(123)
model.Rmsprop <- training_run("a3_nn.R", flags = list(
  dropout1 = c(0.4),
  dropout2 = c(0.3),
  batch = c(10),
  unit1 = c(52),
  unit2= c(10),
  epoch = 100
))

model.adam <- training_run("a3_nn_adam.R", flags = list(
  dropout1 = c(0.4),
  dropout2 = c(0.4),
  batch = c(40),
  unit1 = c(78),
  unit2= c(10),
  epoch = 100
))


# Scale training and test date
s.tr <- scale(model.matrix(Y ~ ., data = aus.tr))[,-1]
s.ts <- scale(model.matrix(Y ~ ., data = aus.ts))[,-1]

# best balance
model <- load_model_tf("keramodels/3.67222213745117.keras")

# Prediction
pred.dnn <- predict(model, s.ts)
train.dnn <- predict(model, s.tr)

# MSE
mean((pred.dnn - aus.ts$Y)^2)
mean((train.dnn - aus.tr$Y)^2)

# Create dataframe for visualization
dnn.df.train <- data.frame(train.dnn, aus.tr$Y, aus$Year_Q[1:156])
dnn.df.test <- data.frame(pred.dnn, aus.ts$Y,aus$Year_Q[157:166])
colnames(dnn.df.train) <- c("Neural Network", "Original", "Year_Q")
colnames(dnn.df.test) <- c("Neural Network", "Original", "Year_Q")

# Visualization of NN
dnn.g1 <- dnn.df.test %>% pivot_longer(cols = c("Neural Network", "Original"),
                          names_to = "Method",
                          values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Result of Neural Nework (Test)", x = "Timeline (Quarterly)",
       y = NULL) +
  geom_line(aes(group=Method), size=1) +
  geom_point()+
  theme_bw()
dnn.g2 <- dnn.df.train %>% pivot_longer(cols = c("Neural Network", "Original"),
                           names_to = "Method",
                           values_to = "Value") %>%
  ggplot(aes(x = Year_Q, y = Value, col = Method)) +
  labs(title = "Result of Neural Nework (Train)", x = "Timeline (Quarterly)",
       y = "Unemployment rate (%)") +
  geom_line(aes(group=Method), size=1) +
  #facet_wrap(.~Method) +
  theme_bw() +
  theme(legend.position = "none")

grid.arrange(dnn.g2, dnn.g1, ncol=2)

mean((pred.adam-aus.ts$Y)^2)
```

Cross validation

```r
## ------------Create Test Index Funcnion-------------------
## Set the starting point and gap and create a test set and train set
creat.list <- function(start, gap, total, data){
  start <- start
  gap <- gap
  {
  # Create an empty list to store the test index
  index <- list()
```

```
  # Calculate the number of vectors
  num_vectors <- ceiling(total / gap)
  # Create the list of vectors
  for (i in 1:num_vectors) {
    end <- min(start + gap - 1, total)  # Ensure the last element
 doesn't exceed 166
    vector <- seq(start, end)
    index[[i]] <- vector
    start <- end + 1
  }
  ## Creating 10 folds of split dataset according to time series
data
  ts.list <- list()
  tr.list <- list()
  for (i in 1:length(index)){
    ts.list[[i]] <- data[index[[i]],]   # Remove the Year_Q colum
n
    tr.list[[i]] <- data[-index[[i]],]  # Remove the Year_Q colum
n
  }
  full.ls <- list(ts.list, tr.list)
  return(full.ls)
  }
}
## ----------------------Create folds---------------------
f.ls <- creat.list(1,  # Start index
                   11, # Observation, change to 13 for trianing
                   165, # Total observations, change to 156 for t
rianing
                   aus[,-1])
ts.list <- f.ls[[1]]
tr.list <- f.ls[[2]]
## Cross validation on select data set
set.seed(123)

mse.cv <- list(nn = data.frame(),
               bs = data.frame(),
               svm = data.frame())

flags.s <- flags(
  flag_numeric("dropout1", 0.3),
  flag_numeric("dropout2", 0.3),
  flag_integer("batch", 10),
  flag_integer("unit1", 26),
  flag_integer("unit2", 10),
  flag_integer("epoch", 100)
)
time.list <- list(nn = c(),
              svm = c(),
              bs = c())
for (i in 1:length(ts.list)){
  mse.cv.bs <- c()
  mse.cv.nn <- c()
  mse.cv.svm <- c()

  # NN
  start.nn <- Sys.time()
  nn.results <- dnn.fn(train = tr.list[[i]],
                       train.y = tr.list[[i]]$Y,
                       test = ts.list[[i]],
                       test.y = ts.list[[i]]$Y,
                       FLAGS = flags.s)

  mse.cv.nn <- c(mse.cv.nn, nn.results[[2]])
  end.nn <- Sys.time()
  nn.dif <- end.nn - start.nn
  time.list$nn <- c(time.list$nn, nn.dif)
  # SVM
  start.svm <- Sys.time()
  svm.results <- svm.fn(train = tr.list[[i]],
                        test.x = ts.list[[i]][,-1],
                        test.y = ts.list[[i]]$Y,
                        kernel.list = c("radial"))
  mse.cv.svm <- c(mse.cv.svm, svm.results$radial[[2]])
  end.svm <- Sys.time()
  svm.dif <- end.svm - start.svm
  time.list$svm <- c(time.list$svm, svm.dif)

  # Boosting
  start.bs <- Sys.time()
  bs.results <- gbs.fn(train = tr.list[[i]],
                       test.x = ts.list[[i]][, -1],
                       test.y = ts.list[[i]]$Y,
                       ntrees = 200)
  mse.cv.bs <- c(mse.cv.bs, bs.results[[2]])
  end.bs <- Sys.time()
  bs.dif <- end.bs - start.bs
  time.list$bs <- c(time.list$bs, bs.dif)


  mse.cv$bs <- rbind(mse.cv$bs, mse.cv.bs)
  mse.cv$nn <- rbind(mse.cv$nn, mse.cv.nn)
  mse.cv$svm <- rbind(mse.cv$svm, mse.cv.svm)
}
print(paste("Neural Nework: ", sum(time.list$nn), sep=""))
print(paste("SVM: ", sum(time.list$svm), sep=""))
print(paste("Boosting: ", sum(time.list$bs), sep=""))
```

Function

```
## create final report
create.df.fn <- function(end = 165, # 156 for train, 165 for full
                         by = 11,    # 13 for train 11 for full
                         fold = 15   # 12 for train, 15 for full
                         ){
  period.col <- c()
  year.id <- c(seq(1,
                   end,
                   by = by),
               end)
  year.m <- substr(as.character(aus$Year_Q), 1,
                   nchar(as.character(aus$Year_Q)) - 3)
  for (i in 1:fold){
    period.col <- rbind(period.col,
                        paste(year.m[year.id[i]],
                              year.m[year.id[i+1]],sep = "~")
 # Create a name the present period
                        )
  }
  df.nn <- data.frame(Period = period.col, "MSE.test" = mse.cv$n
n)
  colnames(df.nn) <- c("Period", "MSE.test")
  df.svm <- data.frame(Period = period.col, "MSE.test" = mse.cv$s
vm)
  colnames(df.svm) <- c("Period", "MSE.test")
  df.bs <- data.frame(Period = period.col, "MSE.test" = mse.cv$b
s)
  colnames(df.bs) <- c("Period", "MSE.test")

  df.nn$Method <- rep("Neural Network", fold)
  df.svm$Method <- rep("SVM", fold)
  df.bs$Method <- rep("gbm", fold)
  cv.df <- merge(df.nn, df.svm, by = c("Period", "Method", "MSE.t
est") , all = TRUE)
  cv.df <- merge(cv.df, df.bs, by = c("Period", "Method", "MSE.te
st") , all = TRUE)
  return(cv.df)
  }
```

Create Final Report of Cross Validation

```
cv.df <- create.df.fn()

ggplot(cv.df , aes(x = Period, col=Method)) +
  geom_point(aes(y=MSE.test))+
  geom_line(aes(y=MSE.test, group = Method), size =1) +
  labs(title = "15-Folds Cross Validation ML vs ANN (Full Datase
t)", x = "Period As Validation", y = "Mean Square Error (MSE)") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))


cv.df %>% group_by(Method) %>% summarise(mean.mse = mean(MSE.tes
t))
# mean time
# NN
mean(time.list$nn)
# SVM
mean(time.list$svm)
# BS
mean(time.list$bs)
```