# Predicting Australia's Unemployment using a Support Vector Machine and Vanilla Neural Network

Daniel Majer

24/02/2021

## Abstract

The purpose of this report is to compart the predictive performance of a selected machine learning algorithm with a vanilla Neural Network at a regression task. The regression task involves predicting the Australian Unemployment form a small dataset containing 158 observations and included 7 numeric and one date predictor as well as the numeric response variable. Optimisation techniques were implement on this supervised regression task such as scaling and k fold cross validation to tune each selected algorithm to find the optimal hyper parameters. The metrics used to compare the model performance included, mean squared residual error, R squared and mean absolute error. The final machine learning model used in the final comparison was a support vector machine using a radial kernel which was compared to a three level network with 32 nodes at each hidden layer. Combinations of varying neurons and hidden layers, with the use of K fold cross validation, revealed that 32 nodes in 3 hidden layers produced the most accurate prediction results. However this result may have changed if further time was spent investigating different optimisers available with various combinations of hidden layers and neurons.

Both algorithms produced similar results with observations from June 2018 quarter ranging to December 2020 quarter, However the support vector machine predicted reduced unemployment rates compared to the spike from March 2020 to September whilst the neural network predicted in higher unemployment at this period. However, all evaluation metrics regarding the test data indicated that the support vector machine did a much better job of predicting the unemployment rate than the vanilla neural network. It was expected that the neural network would produce better predictive results due to the complexity of the model however, this may be due to the small amount of observations and features in the supplied dataset in which the simpler model succeeded. Another method that may have improved both results was increase the number of predictors or engineering the variables so that both model could make potential unseen connection. A number of measures could be improve the performance of the neural network included, incorporating back testing or lags on the times series dataset, which would add another level of complexity to the neural network and increase computational demand. Overall, the simple machine learning model won and is more interpretable than the neural network. If this report was repeated and more time was allowed for experimentation of hidden layer and neuron combinations, this may have yielded a different result.

## Overview

Unemployment rate is defined the percentage of people in a labour force who are willing and able to work without employment (Reserve Bank of Australia, 2021). This metric is considered a key indicator of economic performance receives heavy media attention uncertainty (Picardo,2020).

### How Labour Force Data is Collected

The Australia Bureau of Statistics (ABS) is responsible for collecting and reporting Australia's labour market. Each month, the ABS conduct 50,000 labour force about an individual's participation in the labour market (ABS, 2021; Parliament of Australia, 2018). Individuals over the age of 15 are separated into a two classes, "labour force" and "not in the labour force". The "labour force" category classifies individuals as "employed", if they are in a paid job for one hour or more in a week, or "unemployed", if they are not and are actively looking for work. The complementary class includes any individual not in a paid job but are not looking for work for a variety of reasons (study, caring for children etc.). The ABS calculates Australia's unemployment rate using only the labour force, however, recently UR has come under scrutiny due how accurate UR is as an economic indicator (Jericho, 2016). Thus it has reported that underemployment rate is a better measure of a countries economic performance as it measures persons who usually work less than 35 hours per week but prefer to work more hours (ABS, 2021, Jericho, 2016).

### Factors Effecting Unemployment Rate

Unemployment in Australia can be separated into three categories – cyclical, structural and frictional unemployment. In practice, these subsects of unemployment cannot be measured directly but are still taken into account as they provide a layer of understanding about the variety factors affecting Australia's unemployment rate.

Cyclical unemployment (CU) occurs with changes as in economic activity (downturns) over the business cycle. CU is the mercy of the market with business cutting or reduced hiring of staff due to weaker demand for good and services and usually occurs from 1 to 12 months (Chen, 2020; RBA, 2020). This unemployment can also be reduced by governmental policy encouraging job hiring through tax cuts for businesses and indivudals, reduction of interest rates and state and federal infrastructure development (Amadeo, 2020). Structural employment occurs due to the phasing out of previous jobs, including large scale automation of jobs such as job in the manufacturing industry, and workers do not have the skillset to do jobs in the current labour market. This type of unemployment is tends to last the longest because it can take a number of years for workers to develop new skills or move into a different region that matches their skillset. Government tread lightly regarding FU trying to minimise dampening technological innovation maintaining older jobs and preventing businesses form evolving. Frictional unemployment occurs when people move between jobs in the labour market and includes people transitioning into and out of the labour force. This type of unemployment includes seasonal increases and decreases of goods and services is the least serious of the three types (RBA, 2021). This is because it generally short lived compared to the other types may not have substantial effects on wages of inflation (RBA, 2021).

### Unemployment Rate in Recent Times

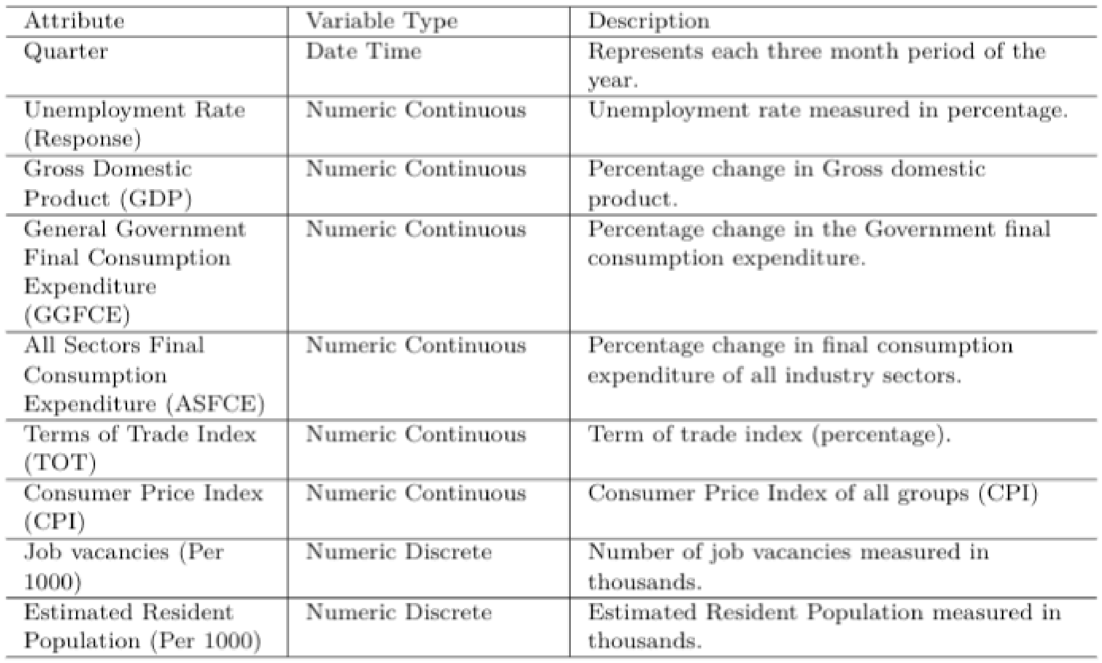
Australia's job market has seen economic highs and lows due to a number of domestic and international economic shifts. These events include the 2008 Australian mining boom as well as downturns due to Global Recessions in the 1990s and 2009 and unprecedented events such as the COVID_19 epidemic. Australia's UR at the most recent quarter, December 2020, reported 6.6% with an underemployment rate of 8.5% and participation rate of 66.2% of Australia's working population. Whilst the UR has reduced by 1.1% from the height of the COVID_19 epidemic which resulted in a UR of 7.5% for June 2020, Australia is far its previous position which is reflected in the underemployment rate. Despite the current underemployment rate, the UR rate for past 3 quarters are uncharacteristic of Australia's employment performance for the last 5 years which until the pandemic was on a 5 year decline. However this UR was still lower compared to the UR during 1990s the where Australia's UR reached record heights of 11.2% in 1992 due to a global recession. Since this historic high, Australia saw extreme economic prosperity in years to come, in which the UR decreased over a 16 year period to an all-time low of 4% in 2008.

### Purpose

The purpose of this report is to apply machine learning (ML) methods on a supplied time series dataset to predict the Australia's unemployment rate. This report will detail and describe the variables used in the dataset, the techniques used to prepare the data, choice and implementation of ML algorithms including a Vanilla Neural Net (VNN), and finally compare and evaluate the selected metrics between the ML algorithm and VNN.

### Data

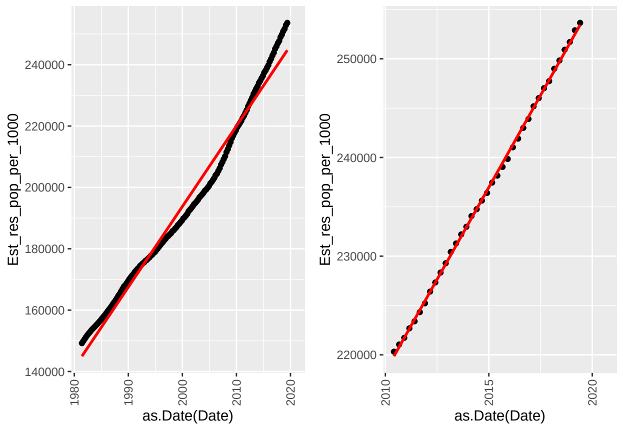
The dataset used in the report was supplied by was aggregated and collection from the ABS (2021) and supplied by James Cook University course Data Mining and Machine Learning. It contained the Australian Unemployment rate for each quarter from June 1981 to September 2020. This timeseries dataset contained 158 observation with one response numeric variable "unemployment rate" and eight predictors, seven numeric and one date variable. The table below displays the metadata about each variable:



| Attribute | Variable Type | Description |
|---|---|---|
| Quarter | Date Time | Represents each three month period of the year. |
| Unemployment Rate (Response) | Numeric Continuous | Unemployment rate measured in percentage. |
| Gross Domestic Product (GDP) | Numeric Continuous | Percentage change in Gross domestic product. |
| General Government Final Consumption Expenditure (GGFCE) | Numeric Continuous | Percentage change in the Government final consumption expenditure. |
| All Sectors Final Consumption Expenditure (ASFCE) | Numeric Continuous | Percentage change in final consumption expenditure of all industry sectors. |
| Terms of Trade Index (TOT) | Numeric Continuous | Term of trade index (percentage). |
| Consumer Price Index (CPI) | Numeric Continuous | Consumer Price Index of all groups (CPI) |
| Job vacancies (Per 1000) | Numeric Discrete | Number of job vacancies measured in thousands. |
| Estimated Resident Population (Per 1000) | Numeric Discrete | Estimated Resident Population measured in thousands. |

## Data Pre-processing

**Data Imputation**: When the data was loaded into R, NAs were observed in the JV and ERP variables. Thus before exploratory analysis was conducted on the dataset, the data was imputed using linear regression and polynomial regression. A scatter plot of ERP and quarter revealed that the slight linear relationship from 1981 to 2020. However a stronger linear relationship was observed from 2010 to 2020 so linear regression was used to calculate the missing ERP values (Noor, 2014).



Thus NA values were replaced using lm() function, set.seed set to 123 for reproducible results. Before running the function, the Date variable was replaced an index from 1 to 37 to calculate an accurate linear equation. The function returned the formula $y = 932.7x + 218930.5$. This equation was used over a for loop to calculate the 5 missing ERP values, which were then added to the original dataframe. A scatter plot of the resulting graph with the calculated values can be seen appendix below.

The mice() function was utilised to calculated the 5 NA values in the JV variable. This function uses built-in imputation models, such as predictive mean matching and normal prediction methods, to calculate continuous variables. This function was discovered after linear interpolation implemented to calculate the ERP values, in hindsight the function could be used to calculate missing NA values in multiple continuous variables. This function used method = “norm.predict”, m = 3 (which repeated the predictions 3 times) and seed = 123 (Swalin, 2018). Finally the predicted results were added to the raw data using the complete() function.

**Numeric Variable Distribution**: Distributions the numeric predictors were inspected using a series of boxplots (See Appendix). This graphic showcases that some variables different substantially with their numeric range and also that some variables are subject to outliers. This helped deiced the choice of ML algorithms, in which the ML algorithm needed to flexible enough to deal with outlier values. No outliers were omitted from the dataset in order to maintaining integrity of the small dataset. However, due to this but it was decided that normalisation was necessary before implementing the model.

| | max | min | range | medians | means | std |
|---|---|---|---|---|---|---|
| Unemployment Rate (Response) | 11.133 | 4.1 | 7.033 | 6.25 | 6.852 | 1.789 |
| Gross Domestic Product (GDP) | 3.300 | -7.0 | 10.300 | 0.70 | 0.725 | 0.971 |
| General Government Final Consumption Expenditure (GGFCE) | 7.500 | -4.6 | 12.100 | 1.00 | 0.853 | 1.671 |
| All Sectors Final Consumption Expenditure (ASFCE) | 5.900 | -8.3 | 14.200 | 0.80 | 0.762 | 1.097 |
| Terms of Trade Index (TOT) | 13.200 | -8.1 | 21.300 | 0.30 | 0.346 | 2.973 |
| Consumer Price Index (CPI) | 116.600 | 28.4 | 88.200 | 73.50 | 75.080 | 25.018 |
| Job vacancies (Per 1000) | 232.300 | 26.8 | 205.500 | 102.50 | 112.556 | 56.902 |
| Estimated Resident Population (Per 1000) | 258103.900 | 149232.6 | 108871.300 | 191831.08 | 196793.894 | 30816.831 |

**Separating Training and Testing Datasets**: The dataset was partitioned into its training and testing sets before they were scaled. This group were separated by Date in which the training data included 147 records prior to March 2018 whilst the testing dataset consisted of 11 after (and including) March 2018. As the task specified the date each group needed to be separated into, set.seed() was not needed to reproduce these results.

**Normalization**: The scale() function was implemented on the numeric predictors for training and test dataset. This was achieved by calculating the means and standard deviations of each numeric predictor for the training dataset which were used inside the scale() function. The rationale behind z score normalization before implementing teh predictors into the selected ML and NN algorithms was because for the following reasons:

- standardisation helps improve the numerical stability a model.
- standardization may speed up the training process.
- standardization gives ‘equal’ considerations for each feature.
- standardization can help surpress the effect of outliers.

However for tree based models, the literature indicated normalisation did not improve or effect this type of ML algorithms because these algorithms seek the best split for each feature. Therefore the scaled data was used anyway on both tested MLs out of convience (Zhang,2019). NN on the other hand should theoretically not be improved by standardisation however, empirical evidence indicates that normalization is beneficial in terms of accuracy and also improve training time (Zhang, 2019).

## Machine Learning Algorithms

For the purposes of this assessment, two machine learning algorithms were selected to predictive the unemployment rate in which the better performing algorithm would be selected for the final comparison with the vanilla neural network (VNN). Both algorithms would be tuned to find their respective optimal hyper parameters. The two selected ML algorithms included, Random Forest (RF) and Support Vector Machines (SVM).

The RF model was selected because the literature stated is a flexibly tree based algorithm that can effectively perform a regression analysis on data that contains Date Time variables as well as outliers. The second ML algorithm that was selected based off the literature was SVM because it can effectively find the optimal boundary between possible outputs. This algorithm is also effective when the response variable is non-linear in which SVM's polynomial or radial kernel can be selected to calculate the boundary for the response variable (KDnugget, 2019). The scaled data was also used to train the SVM model becuase an article from Hoake (2010) stated that both non-linear kernals preformed worse with the presence of outliers when sacling was not performed.

Both attempts would be trained and tuned using repeated cross validation using the caret package. To ensure a validity between model comparison and the final model choice, a standard appraoch was taken when tuning the model. However, whilst a standard approach was taken, the number and repeats for crossvalidation were reduced with computation demand in mind. Thus, the results below may have differ if these metrics were increased.

The trainControl() function was kept the same when training all three models. The metrics used for this model included:

**Train Control**

1. Method - determines the sampling/validation method uses in the algorithm. "repeatedcv" means repeated cross validation.
2. Number – number of folds for cross validation. Number was set to 3 to reduce computation time.
3. Repeats - the number of repetitions during cross validation. Was set to 1 to reduce computation time.
4. summaryFunction - evaluates the summary of the model. Set to 'defaultSummary' because of the continuous response variable.

All models used the expand grid function to help find the optimal hyper parameters.

**Expand Grid for RF**

The metric used for tuning the RF model was "mtry". This parameter was test from 1 to 8 (the number of numeric predictors in the dataset) and indicates the number of variables randomly sampled as candidates at each split. The "mtry" was used becasue it is the only parameter available in caret for tuning in order to determine the final accuracy adn as such must be found empirically (Browlee, 2016).

**Expand Grid for SVM**

The SVM model used two parameters when tuning the model which included, "sigma" and "Cost". The parameters and explainations for each parameter are:

1. Sigma (gamma) – controls the influence of individual training instances. The higher the value, the more consideration the algorithm will have for all data point to calculate the hyperplane. For this parameter, sigma values were set fro 0.1 to 1 which a 0.1 increments to determine the best value.
2. C (cost) – is the regularisation parameter relates to the support vectors on all dimensions of the hyperplane. This parameter indicates the number of misclassifications that are allowed. For this parameter, costs were set from 1 to 10 using the seq() function to determine the best cost.

The code for all tuned models can be found in the Appendix. To save time duing whilst knitting the Markdown document, each model was saved using the saveRDS() function and read back into R using readRDS().

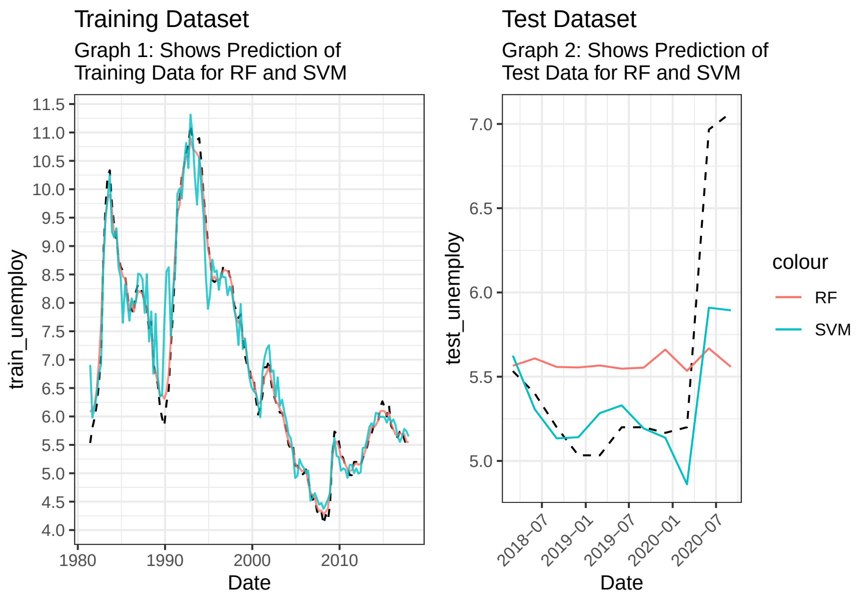
**Metrics For Comparing Model Performance** The evaluation metrics used to compared model performance on the training data was Residual Mean Squared Error (RMSE), Rsquared and Mean Absolute Error (MAE). RMSE was used because it is a good measure of how accurately the model predicts the response and is a good criterion for when assessing model prediction (Grace-Martin, 2019). Rsquared is a useful metric when identifying goodness of fit of the model in which a high metric is optimal (Grace-Martin, 2019). MAE is also a useful as it measures the average magnitude of the errors in a set of predictions, without considering their direction. For both RSME and MAE, the smaller the results the better the model prediction performance. However, these metrics must not be taken at face value in which both models needed be visualised against the training and test data to identify if a model has been overfitted (Janet Wesner 2016).

| | RMSE | Rsquared | MAE |
|---|---|---|---|
| RF | 0.5259019 | 0.9183796 | 0.3534234 |
| SVM | 0.8450682 | 0.7920267 | 0.6043548 |

The training evaluation metrics, in the table above, revealed that the RF model yielded the lowest RMSE and MAE and highest Rsquared value of the three tuned models (See Table Below). This indicates it performed the best in training and predicting the training dataset compared to the SVM model. The graph below of the predicated training dataset also showcases this model did an excellent job predicting the training unemployment rate from the numeric predictors. However, as stated earlier this initial result must be evaluated further using the test data to determine if the model overfitted to the training data.

The graph below displays the predicited training results each model compared to actual unemployement rate. The graph show teh teh RF model more closely followed the traiing dataset compared to the SVM. This seems to indicated that the RF model may have overfitted to the training data.

To evaluate the performance of the tuned models, aforementioned evaluation metrics were calculated using user-defined function in R (see Appendix). These functions were used to compare the generated test predictions with the actual test unemployment rate. This was done to determine the best predictive regression model and see if the RF model had overfitted to the training data.



The plot of the test predictions (coloured lines) against the actual test unemployment rate (dotted line) revealed that the SVM model performed the best out of the two tuned models. This graph also indicated that the RF model had overfit the training data. This is also clearly reflected in the evaluation metrics (See table below) in which the RF model displayed higher test RMSE and MAE values compared to the calculated training values. When RMSE and MAE are higher than then their equivalent training metrics indicates the model is subject to overfitting (Janet Wesner 2016). This table also displayed that SVM had the best predictive performance as it had a much lower RMSE and MAE than the RF model.

| | RMSE | RSquare | MAE |
|---|---|---|---|
| RF | 0.6935941 | 0.0379605 | 0.0924807 |
| SVM | 0.4973538 | 0.5053329 | 0.0483953 |

## Neural Network

A VNN, also known as an “artificial neural network” is a subset of ML, useful for classification and regression tasks. This subset of ML was inspired by the pioneering paper written by neurophysiologist and mathematician McHullock & Pitts that described the functioning of neurons in the human brain (IMB, 2020; JCU Notes, 2020). VNNs are comprised of node layers, containing an input layer (where the dataset is input and dimension are specified), one or more hidden layers, and an output layer.

Each node or neuron connects to another and has an associated weight and threshold. If the output of any individual neuron is above the specified threshold value, that node is activated, sending data to the next layer of the network, otherwise no data is passed along to the next layer of the network. The VNN for this assessment was built using the Keras, reticulate and tensorflow libraries. These library were download into R along with conda and the correct version of python 3.6 to run Keras and tensorflow.

The scaled training and test data was used in the VNN to ensure that the model would not be affected by wide ranging values in the numeric predictors and to reduce the effect of outlier. The quarter column was removed and both training and testing data frames because these structures were converted to matrices. The training and test unemployment rate columns were saved in respective vectors called training_label and testing_label as it was necessary for evaluating the model.

The dimensions of the dataset or input_shape of the model was set to the number of predictor columns (7) in the training matrix (excluding date). Hastie et al. (2017) stated that a typical network contains between 5 to 100 neurons per layer in which it is better to have more than less as lighter networks tend to have a reduce flexibility with complex dataset. For this reason a common VNN structure is 3 layer of 64 neurons per lay. However, this comment was made with complex dataset in mind and because the supplied dataset contains 7 numeric predictors. Dimensionality of a dataset is the main indicator when using a VNN and for datasets with small amounts of features, traditional shallow ML approaches tend to perform just as well, if not better, and are more efficient (AFIT, 2020). Thus for this reason the input layer and two hidden layer contained 32 units (neurons) in the lay_dense() function when building the model. This was done because 32 units was also a common structure observed when researching various VNN structures. The output layer as per convention contained one neuron.

Each individual node in the neuron passes information to the next using its own activation function to calculated the “weight sum” which is comprised of input data, weights, a bias (or threshold) and an output (IMB,2020). The formula for each node depends on the activation function specified in which popular activation functions include, “sigmoid” and “relu” (rectified linear units) (Sharma, 2017). The activation function used in the VNN was “relu” becuase this a common choice for regression tasks (AFIT, 2020). To reduce the bias_regulariser parameter was set to regularizer_l2(0.01) in order to apply penalty on each layer bias in the VNN during optimization (Keras, 2021).

The NN is optimised through the use of a specified cost function which ensures correctness of fit for any given observation (IBM, 2020). Each cost function minimises loss through gradient descent which calculates the slope of the loss function and shifts the weight and biases according to the current slope to a lower point (Wang, 2018). A popular cost function used in this VNN and is Mean Squared Error (MSE) this was also set as the recorded metric to evaluate the VNN’s performance (IBM,2020). The optimiser function also used in conjunction with the loss function was optimizer_rmsprop with a learning rate of 0.03. The conventional learning rate for this parameter is 0.01 but it was increased to 0.03 due to experimentation and examples of this parameter implemented online (AFIT,2020; stackoverflow, 2021).

Batch size determines the number of sample used at a time to estimate the error gradient and the choice of this parameter can substantially affect the VNN. The batch sizes used in the VNN was 8 because of the small amount of features and observations in that dataset and to reduce computational stress when training the model. This was also done because it was observed in an example in the JCU Notes (JCU Notes, 2020).

```r
build_model <- function() {
  model = keras_model_sequential() %>%
    # First hidden layer
    layer_dense(units= 32, bias_regularizer = regularizer_l2(0.01),
                input_shape=c(ncol(NN_training_matrix))) %>%
    # Second hidden layer
    layer_dense(units=32, bias_regularizer = regularizer_l2(0.01), activation = "relu") %>%
    # Third hidden layer
    layer_dense(units=32, bias_regularizer = regularizer_l2(0.01), activation = "relu") %>%
    # Outer layer
    layer_dense(units=1)

  model %>% compile(
    loss = "mse",
    optimizer = optimizer_rmsprop(lr = 0.003),
    metrics = list("mean_absolute_error")
  )
}
```

K folds cross validation (CV) was used, k = 5 and epoch = 300, to determine the optimal epoch level for the final model. During CV epoch is usually higher (500-1000) however, this was reduced with processing time and the small dataset in mind. This is an important step as using an nonoptimal epoch value can lead to over or under fitting issues.

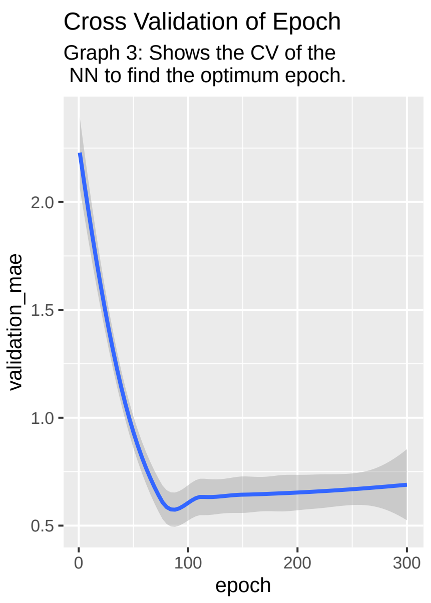
```r
tic = proc.time()[3]
# Cross Validation was adapted fomr classification notes sourced from JCU
k <- 4
indices <- sample(1:nrow(NN_training_matrix))
folds <- cut(indices, breaks = k, labels = FALSE)
num_epochs <- 300
all_mae_histories <- NULL
tf$random$set_seed(123)
for (i in 1:k) {
  #cat("processing fold #", i, "\n")
  # Prepare the validation data: data from partition # k
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- NN_training_matrix[val_indices,]
  val_targets <- NN_training_labels[val_indices]
  # Prepare the training data: data from all other partitions
  partial_train_data <- NN_training_matrix[-val_indices,]
  partial_train_targets <- NN_training_labels[-val_indices]
  # Build the Keras model (already compiled)
  model1 <- build_model()
  # Train the model (in silent mode, verbose=0)
  history <- model1 %>% fit(partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 8, verbose = 0
  )
  mae_history <- history$metrics$val_mean_absolute_error
  all_mae_histories <- rbind(all_mae_histories, mae_history)
}
```

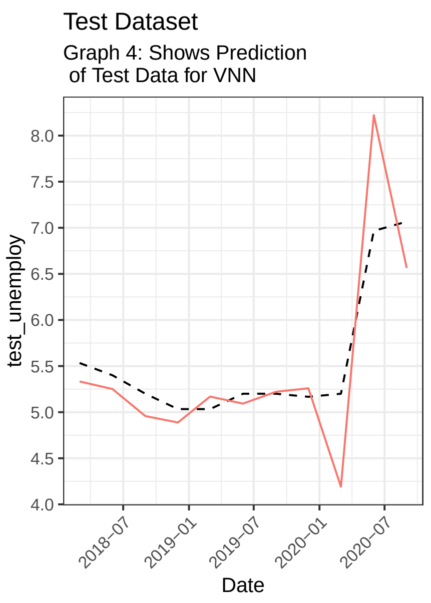
```r
toc_nn_train=proc.time()[3] - tic
average_mae_history <- data.frame(
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories, 2, mean)
)
optimum_epoch = which.min(average_mae_history$validation_mae)

#write.csv(average_mae_history, "average_mae_history.csv")
```

Set.seed was initialised using the set.seed() function in tensorflow library, however an error occurred when building the optimum VNN where set.seed() had not effect on the VNN and when the model was run again it resulted in different loss, MAE and prediction results. Luckily, the optimum VNN was saved using the save_model_tf() and loaded into R using load_model_tf().

The Graph 3 (see below) shows that MAE begins to reduce around 80. However, this is not the lowest MAE which was located at 230 using the which.mean() function and was used to train the final model. The final model was run 10 times and the test predictions, each iterations loss and MAE were stored in 10x11 matrix and two numeric vectors respectively. This was done in order to get the average predicted test unemployment values, and average training loss and MAE. The unemployment predictions were saved in a CSV along with the actual employment rate and loaded back into R to reduce processing time, however, the generated Loss and MAEs were not saved when the model was overridden. Once this was obtained, the aforementioned evaluation metrics, RMSE, Rsquared and MAE were calculated with the respective defined functions for further inspect the VNN performance.





Graph 4 displays the average predicted test data using the optimum epoch parameter. This graphic indicates that the model did a good job predicting the test data unemployment rate. However, the training MAE from the loaded optimised model was 0.4168. This is was much higher than the calculate MAE from the average predicted unemployment rate for the test data (repeated 10 and averaged). The calculated metrics from the average predicted data all indicated that repeating and averaging the test scored yielded much higher results than just one iteration. All 3 evaluation metrics are in the table below.

| Metrics | Values |
|---|---:|
| RMSE | 0.5245565 |
| RSquare | 0.4497415 |
| MAE | 0.0594625 |

**Effect of Hidden Layers on Model Prediction**

To see the effect of Hidden layer, the model was build with only one layer with 32 neurons. All other parameters were kept the same. The same CV method to find the optimum epochs was followed and can be seen in the table below. This code was commented out in the appendix to reduce computational time. As stated previously, the current literature surrounding hidden layers is defeated heavily and thus most advise that experimentation in needed to see the optimum amount layer and neuron combinations (Hastie et al.,2017). However one thing is clear in the literature that complexity of the VNN is heavily dependent on the amount of features contained in the dataset. Thus, in this aspect of the report the VNN was reduced to one layer to see what impact this would have on the prediction power of the model.

The table below shows that reducing the number of hidden layers from three to one substantially worsened the VNN. However, despite large RMSE and Rsquare value, the MAE is much lower than the other two metrics. This is because RMSE and Rsquare are very sensitive to large prediction errors unlike MAE which is why most investigations use this metric the main one to determine the overall success of regression models (Wesner, 2016). When the predicted results were plotted, the plot shows that the first 8 unemployment rates were predicted with high accuracy, however as the unemployment rate increase in 2020, the one layer VNN spiked to 14%. This value greatly affected the RMSE and RSquare value.

| | Three Hidden Layer | One Hidden Layer |
|---|---:|---:|
| RMSE | 0.5245565 | 2.173331 |
| RSquare | 0.4497415 | -8.445695 |
| MAE | 0.0594625 | 0.128004 |

**Reducing the Amount of Neurons per Layer**

Like the previous test, the amount of neurons per layer was also tested to see the predictive power other VNN. The Neurons in each layer was already quite low compared to usual conventions located between 5-100 neurons (Hastie et al., 2017). However, this was substantially reduced to seven per layer, the number of predictive features in the dataset. Before this test was run, it was hypothesised that a lighter VNN with similar parameters to the optimal model would yeild similar results because of the small amount of features and observations in the dataset. This hypothesis ended up being correct as the evaluation metric, compared to the reduced layer are much lower, thus indicating the model did a better job of predicting the test response variable compared to the reduced layers VNN. However, the predicted test unemployment rate, at the 2020 unemployment increase to 10% which in turn affected the RMSE value.

| | 32 Neurons Per Layer | 7 Neurons Per Layer |
|---|---:|---:|
| RMSE | 0.5245565 | 1.058951 |
| RSquare | 0.4497415 | -1.242508 |
| MAE | 0.0594625 | 0.105859 |

**Comparison and Suggestions**

There are many differences between the SVM and VNN algorithms that affect their predictive power in regression tasks. Firstly, SVM are simply much easier to use and the caret packages has inbuilt features which can help the algorithm deal with data containing missing values or is unscaled (KDnugget, 2017). Another benefit of SVM is that it was able to include the quarter variable in the predictive model, which would be essential during timeseries forecasting. Another benefit of the similar algorithm is that it can use both numerical and categorical data without the need for extreme feature transformation or complex architecture (like transforming data into tensors). VNNs, on the other hand, require substantially more pre-processing in which a common practice is omitting missing values if they cannot effectively be imputed. This human-based decision which can greatly affect the integrity of the network whilst also contributing to model bias.

One main benefit of using SVM over VNN was the reduced computational time that the algorithm recorded. This occurred because the algorithm has fewer parameters compared to the VNN which reduced the processing time. SVM took 15.47 seconds to train and tune to optimal parameters, as opposed to the VNN which took 89.61 seconds to train and 121.08 seconds to run the test data. Despite this increased computer processing time, the comparison of the test evaluation metric revealed that SVM performed marginally better than the VNN. This can be explained by the small dataset in which if more observations and features were added, there would have been a greater difference between the two algorithm in which it would be expected that VNN would have a greater predictive power.

Neural Networks have a greater ability to deal with highly dimensional and complex data due to their complex procedure and nature. The large amount of parameters means that these model have the ability to be tuned and perfected using a variety of combinations of Neurons and hidden layers (AFIT, 2020; IMB, 2021). Not only is this one of the main benefits of VNN but also back propagation allows this model to be able to learn from its own mistakes. However as stated previous in this report, VNNs are dependent on the data used to train them, and thus the small dataset provided was small enough and simple enough to be adequately analysed using the simpler SVM algorithm (AFIT, 2020; Hoakes, 2010). One recorded decision that may have effect the results of the VNN was keeping the outliers in the dataset, which may have affected the predictive power of the dataset.

One problem associated with VNNs is that this branch of ML is often viewed as a "black box" method in which the collected results are hard to communicate to stakeholders (Fan et al., 2020; Maroto, 2017). VNNs thrive in situations involving non-tabulated data, such as image processing tasks such as identifying handwritten number, Natural Language Processing speech recognition and facial recognition software (Cheung, 2020). However, these tasks are very complex where understanding the relationships between features is not as important. Thus the implementation of VNNs should be considered case by case only if it is necessary to complete the task. For this problem, the simpler SVM method is better because each numeric predictor relative importance can be evaluated which may be useful for interested parties hoping to gain a greater understanding economic indicators affecting Australia's unemployment rate rather than prediction task altogether. The fact that the optimised VNN performed marginally lower than the SVM indicates that that individuals should consider simpler ML approaches before implementing a VNN due to the time spent building the model and the increases computational time.

| | Neural Network | SVM Radial |
|---|---:|---:|
| RMSE | 0.5245565 | 0.4973538 |
| RSquare | 0.4497415 | 0.5053329 |
| MAE | 0.0594625 | 0.0483953 |

While it was expected that the VNN would yield the best results due to the numerous amount of parameters, it performed slight worse than the SVM model at the given regression task. This is reflected in the evaluation metrics, however, this result have been the result of a few outlier observed in the generated boxplot in the Indices. These were included to maintain data integrety but step could have been take to reduce the effect these outlies had on the VNN. The VNN could have also been improved through the implementation of further model improvements.

If a greater number of predictor variables and observations were implemented into the model, this may have improve prediciton power of the VNN. As only 158 observations were included in the dataset this meant that the simpler SVM models performed better than the VNN. It is also possible that better predictors of unemployment could have been selected and added to the dataset to predict unemployment rate however further research in to the economic indicators surrounding unemployment is necessary. The time series aspect of the data also possess a number of challenges for each model, given the sequential nature of each observation (Teles et al., 2006; Ismail Fawaz et al., 2019). For this task each quarter was disregarded which greatly affected the forecasting performand of the VNN. Various methods could have also been used to improve the forecasting of neural networks, such as Back Testing, which is similar to k-fold cross validation used in this task (Osipenko, 2020). Another such method to improve the network is incorportating lags, the time between two time series that are being correlated, into the model (Teles et al., 2006). This complex method has beeen reported to improve prediction performance in regression tasks however, more research is needed in this areas. Furthermore, a more experimentation of hidden layers and neruons per layer could have also affected the recorded results. Overall, VNN did well predicting the model but due to the small dataset, the simpler SVM algorithm yeilded the best unemployment results.

## References

AFIT. (2020). Feedforward Deep Learning Models. Data Science Lab R Programming Guide. Retrieved from: https://afit-r.github.io/feedforward_DNN#ff

Amadeo K. (2020). Four real world ways to create more jobs. the balance. Retrieved from: https://www.thebalance.com/job-creation-ideas-4-ways-that-work-best-3305521

Australia Bureau of Statistics. (2021). Survey Participant Information - Monthly Population. Retrieved from: https://www.abs.gov.au/websitedbs/D3310114.nsf/home/survey+participant+information+-+Monthly+Population+Survey

Australian Bureau of Statistics. (2001). Underutilised Labour: Unemployment trends and patterns. Retrieved from: https://www.abs.gov.au/AUSSTATS/abs@.nsf/2f762f95845417aeca25706c00834efa/855e6f87080d2e1aca2570ec000c8e5f!OpenDocument#:~:text=Over%20this%20period%2C%20the%20number,increased%20fr

Brownlee, J. (2016). Tune Machine Learning Algorithms in R (random forest case study). Machine Learning Mastery. Retrieved from: https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

Cheung, K. (2020). 10 Use Cases of Neural Networks in Business. Algorithm-X Lab. Retrieved 17 June 2020, from https://algorithmxlab.com/blog/10-use-cases-neural-networks/#Developing_Personalised_Treatment_Plans.

Didugu, C. (2020). Ultimate Guide to Input shape and Model Complexity in Neural Networks. toward data science. Retrieved from: https://towardsdatascience.com/ultimate-guide-to-input-shape-and-model-complexity-in-neural-networks-ae665c728f4b

Fan, F., Xiong, J., Li, M. & Wang, G. (2020). On Interpretability of Artificial Neural Networks: A Survey. Cornell University. Retreived from: https://arxiv.org/abs/2001.02522

Grace-Martin, K. (2019). Assessing the Fit of Regression Models. The Analysis Factory. Retrieved from: https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/#:~:text=The%20RMSE%20is%20the%20square,

Hastie, T., Friedman, J., & Tisbshirani, R. (2017). The Elements of statistical learning. Springer.

Hoak, J. (2010). The Effects of Outliers on Support Vector Machines. Portland State University. Retrieved from: http://www.common-index.com/files/AML_Project.pdf

IMB. (2021). Neural Networks. IBM Cloud Learn Hub. Retrieved from: https://www.ibm.com/cloud/learn/neural-networks

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. (2019). Deep learning for time series classification: a review. Data Mining and Knowledge Discovery, 33(4), 917–963. https://doi.org/10.1007/s10618-019-00619-1

Jericho G. (2016). Why unemployment is no longer the best indicator of the economy's health. The Guardian. Retrieved from: https://www.theguardian.com/business/grogonomics/2016/aug/22/why-unemployment-is-no-longer-the-best-indicator-of-the-economys-health

Keras. (2021). The Model class. Keras API reference. Retrieved from: https://keras.io/api/models/model/

KDnugget. (2017). What is Support Vector Machines, and why would I use it? Retrieved from: https://www.kdnuggets.com/2017/02/yhat-support-vector-machine.html#:~:text=SVM%20is%20a%20supervised%20machine,boun

Maroto, M. (2017). Neural Networks = Black Box?. toward data science. Retrieved from: https://towardsdatascience.com/neural-networks-black-box-b20723f9a417

Noor, M. (2014). Comparison of Linear Interpolation Method and Mean Method to Replace the Missing Values in Environmental Data Set. Materials Science Forum. 803. 278-281. 10.4028/www.scientific.net/MSF.803.278.
Picardo E. (2021). How unemployment affects everybody. Investotopia. Retrieved from: https://www.investopedia.com/articles/economics/10/unemployment-rate-get-real.asp

Parliament of Australia. (2018). Unemployment statistics: a quick guide. Retrieved from: https://www.aph.gov.au/About_Parliament/Parliamentary_Departments/Parliamentary_Library/pubs/rp/rp1718/Quick_Guides/Unemployment

Reserve Bank of Australia. (2021). Types of Unemployment. Retrived from: https://www.rba.gov.au/education/resources/explainers/unemployment-its-measurement-and-types.html#:~:text=The%20size%20of%20the%20labo

Sharma, A. V. (2017). Understanding Activation Functions in Neural Networks. Medium. Retrieved from: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

StackExchange. (2020). How to choose hidden layer and nodes in a feedfoward neural network?. Cross Validate. Retreived from: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

Swalin A. (2018). How to Handle Missing Data. toward data science. Retrieved from: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4

Teles, L. O., Vasconcelos, V., Pereira, E., & Saker, M. (2006). Time series forecasting of cyanobacteria blooms in the Crestuma Reservoir (Douro River, Portugal) using artificial neural networks. Environmental Management, 38(2), 227-237.

Trading Economics. (2021). Australia Unemployment Rate. Retrieved from: https://tradingeconomics.com/australia/unemployment-rate#:~:text=Unemployment%20Rate%20in%20Australia%20averaged,percent%20in%20Febr
%20data%2C%20historical,updated%20on%20February%20of%202021.

Wang, C. (2018). Finding the Cost Function of Neural Networks. toward data science. Retrieved from: https://towardsdatascience.com/step-by-step-the-math-behind-neural-networks-490dc1f3cfd9

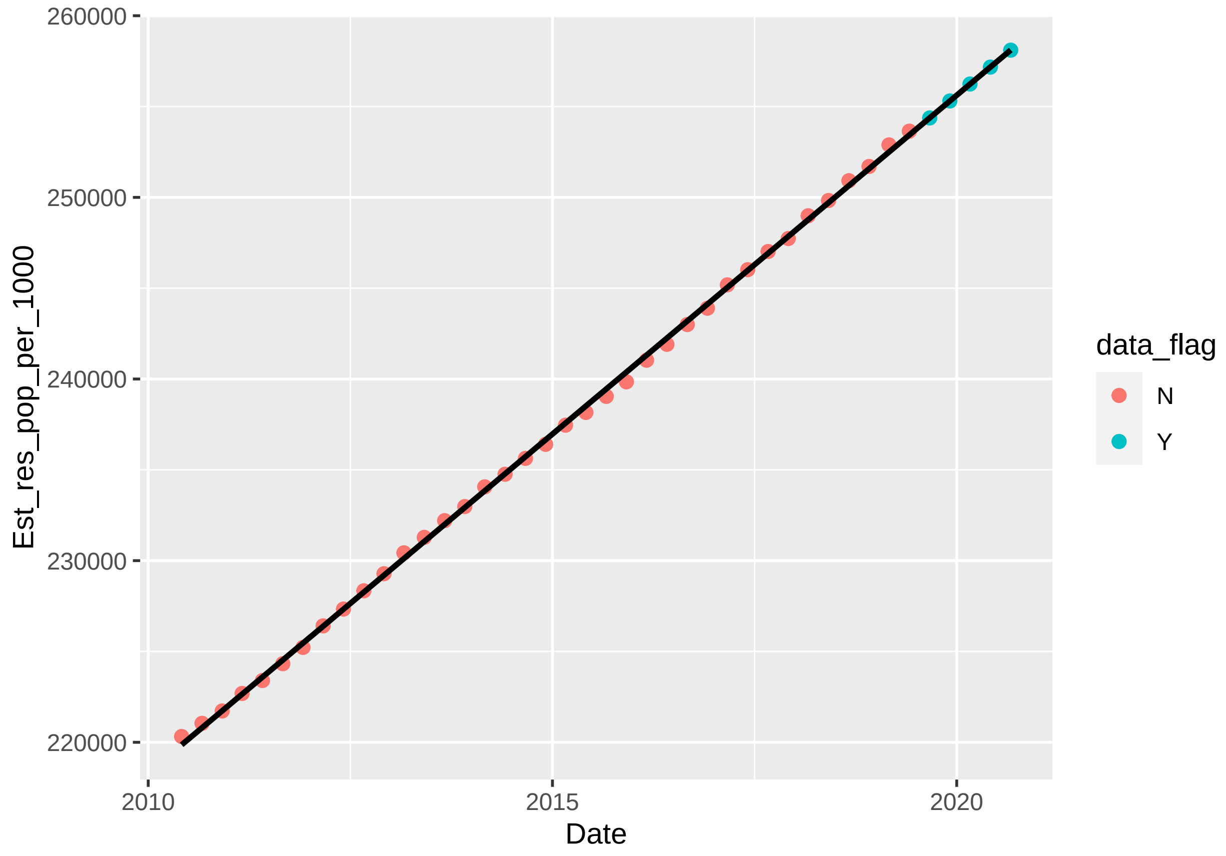
Wesner, J. (2016). MAE and RMSE — Which Metric is Better?. Medium. Retrieved from: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

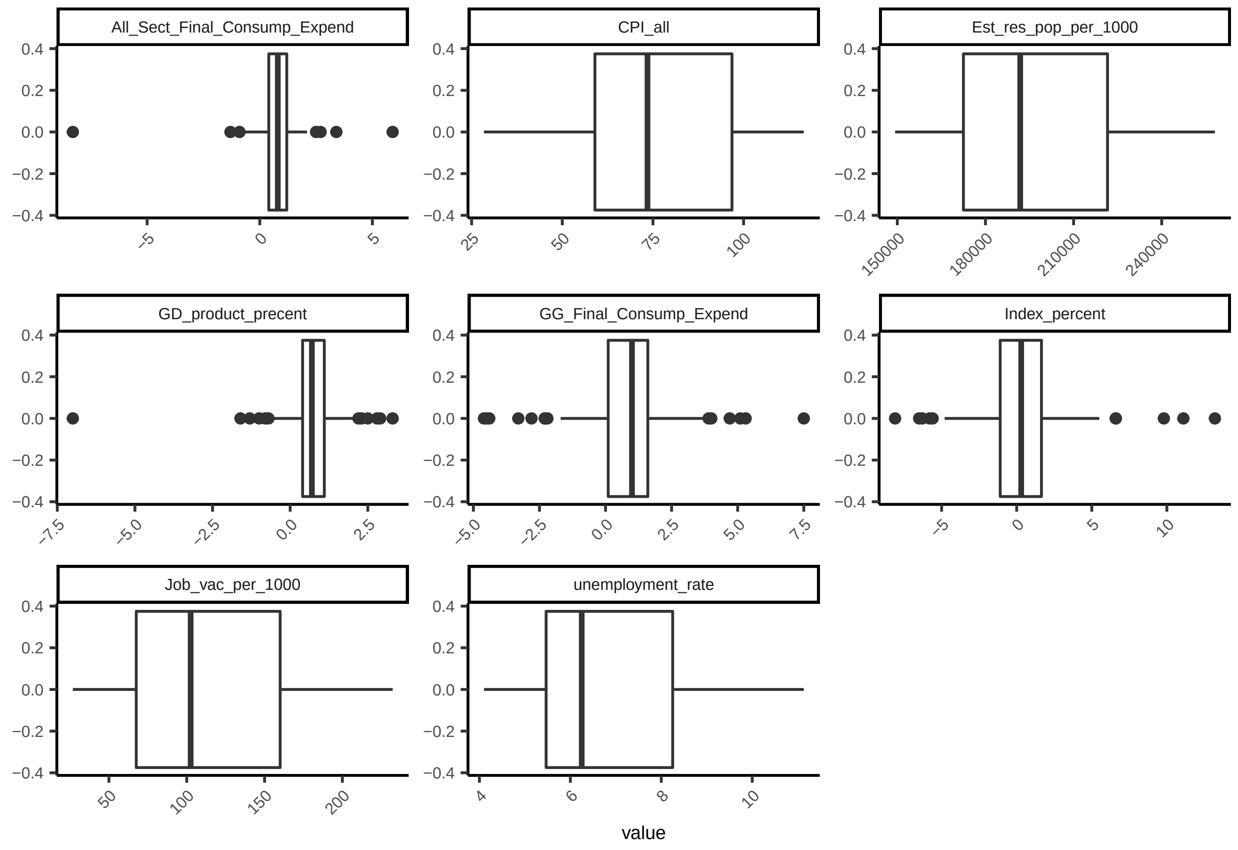
Zhang Z. (2019). Understand Data Normalization in Machine Learning. Retrieved from: https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0

## Appendix

```r
#visualising new values
raw_data %>%
  filter(as.Date(Date)>"2010-03-01") %>%
  mutate(index = seq(1, 42, 1),
         data_flag = as.factor(ifelse(index>37, "Y", "N"))) %>%
  select(Date, index,Est_res_pop_per_1000, data_flag) %>%
  ggplot(aes(x = Date, y = Est_res_pop_per_1000, color = data_flag))+
  geom_point(size = 2)+
  geom_smooth(method = "lm", se = F, color = "black")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



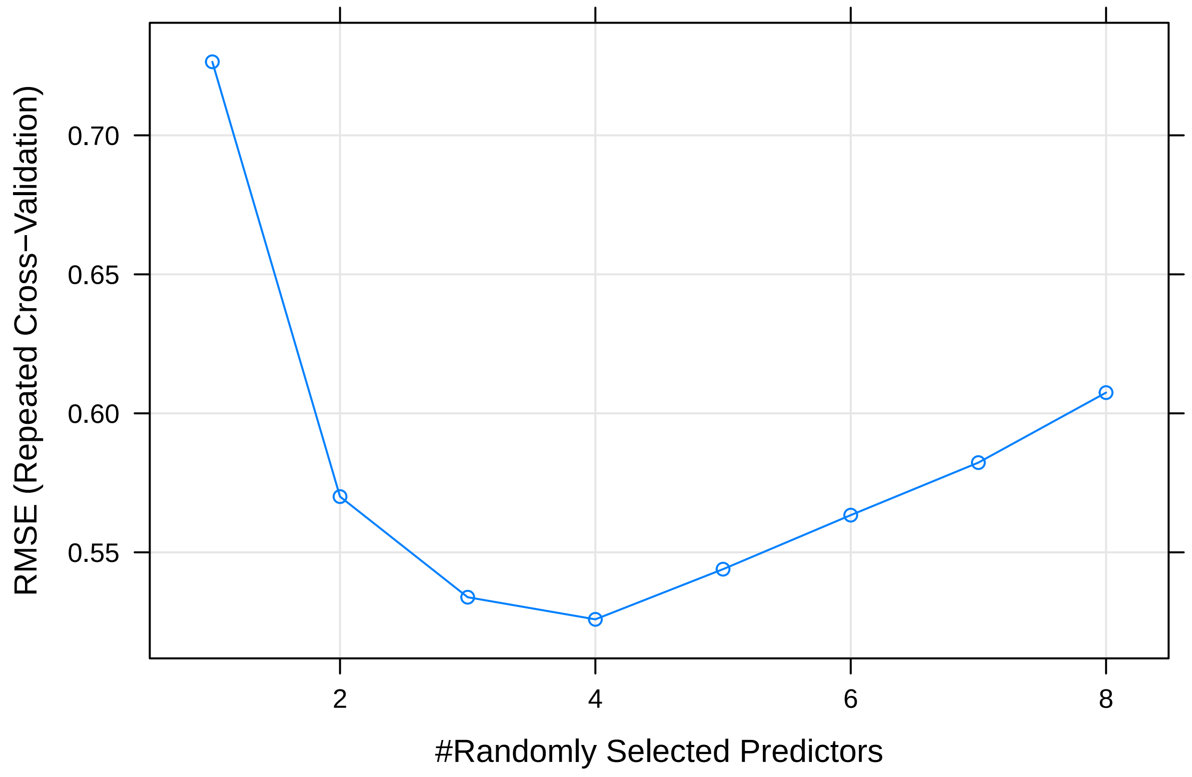


```
#tuning RF
# library(caret)
# fit_control_rf <- trainControl(method = "repeatedcv",
#                                number = 3,
#                                repeats = 1,
#                                summaryFunction = defaultSummary)
#
# rf_grid <- expand.grid(.mtry =c(1:8))
#
# metrics <- "RSME"
#
# set.seed(123)
#
# rf_fit <-train(unemployment_rate ~ .,
#                   data = scaled_training_data,
#                   method = "rf",
#                   trControl = fit_control_rf,
#                   # provide a grid of parameters
#                   tuneGrid = rf_grid)
#
rf_fit
```

```
## Random Forest
##
## 147 samples
##   8 predictor
```

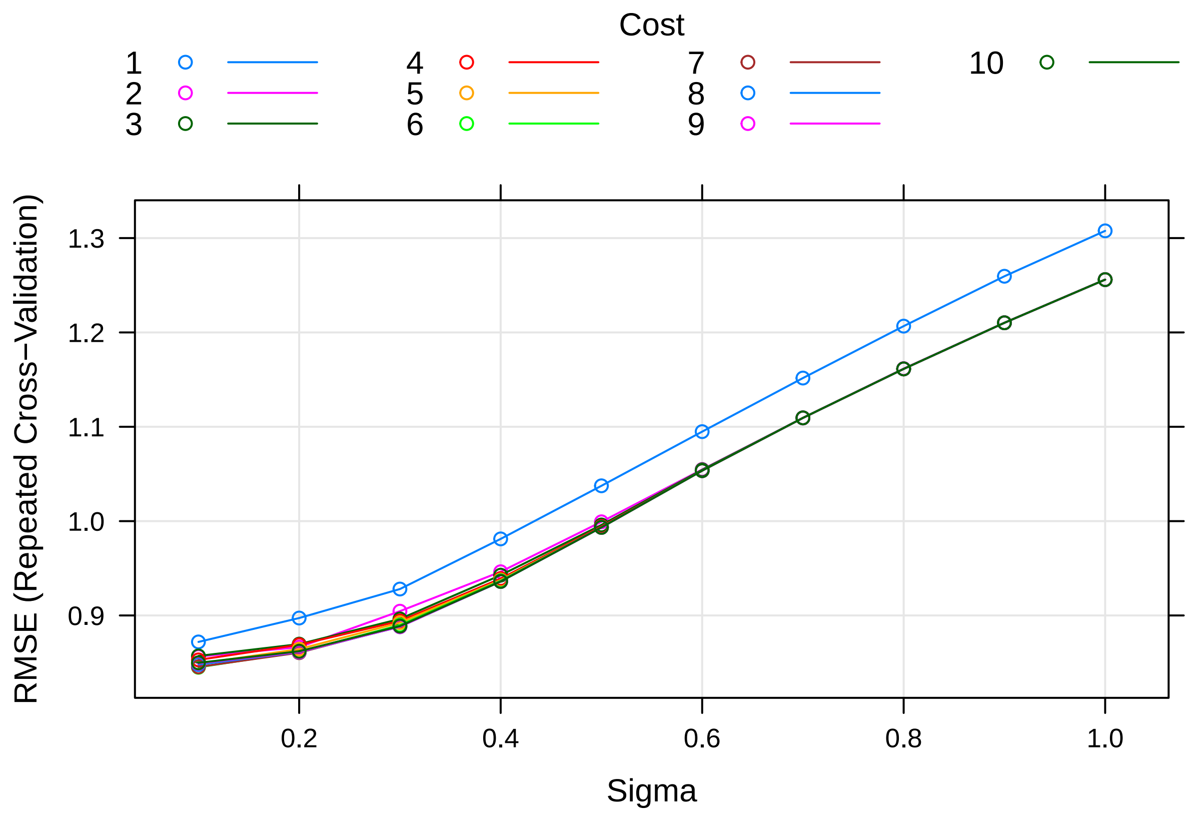
```
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 1 times)
## Summary of sample sizes: 99, 99, 96
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##   1     0.7264419  0.8503743  0.4976370
##   2     0.5700571  0.9067731  0.3874321
##   3     0.5338627  0.9168311  0.3547588
##   4     0.5259019  0.9183796  0.3534234
##   5     0.5439465  0.9124504  0.3653481
##   6     0.5633794  0.9060414  0.3769507
##   7     0.5822802  0.8989534  0.3889521
##   8     0.6074584  0.8896880  0.4106274
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 4.
```

```
plot(rf_fit)
```

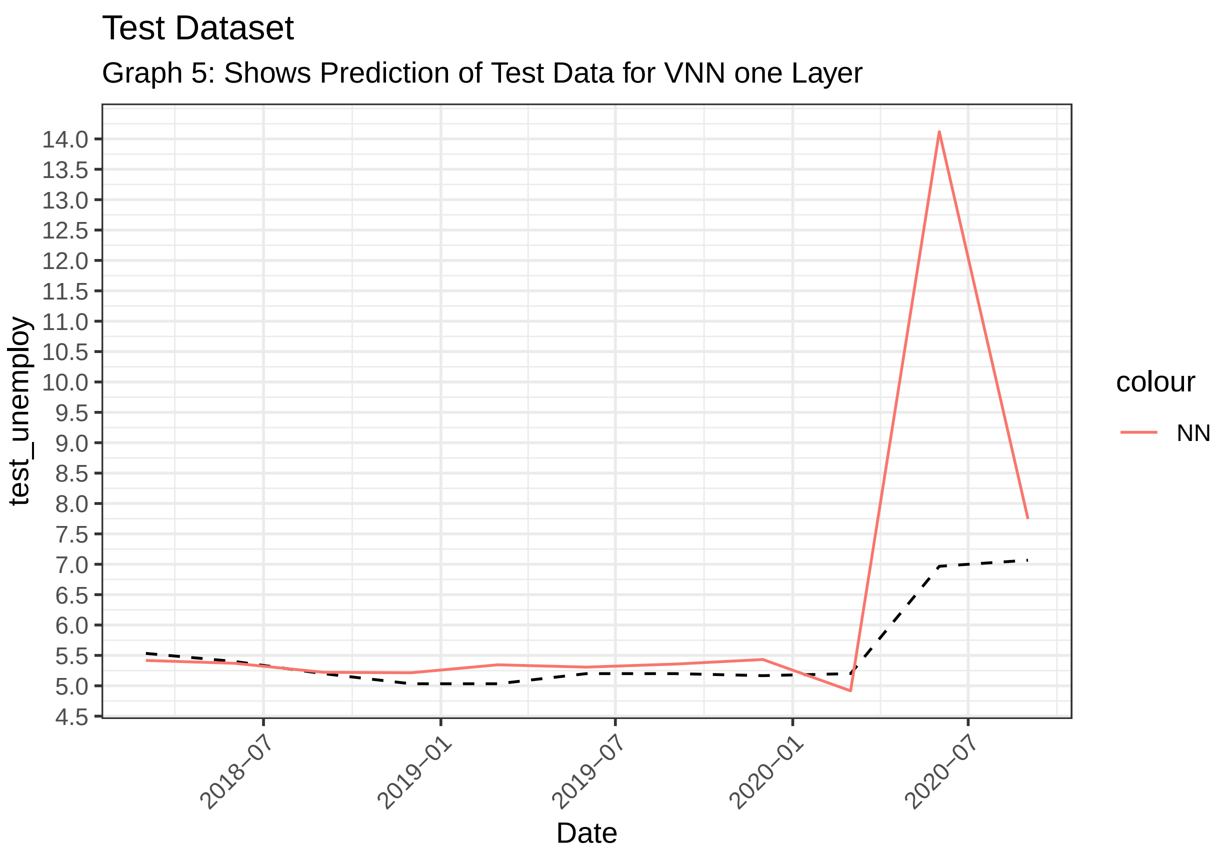


```
# library(caret)
# set.seed(123)
# tic = proc.time()[3]
# train_control <- trainControl(method = "repeatedcv",
#                                number = 3,
#                                repeats = 1,
#                                summaryFunction = defaultSummary)
#
# svm_grid <- expand.grid(sigma = seq(0.1,1,0.1), C=seq(1,10,1))
#
#
# svm_radial_scaled <- train(unemployment_rate ~ .,
#                      data = scaled_training_data,
#                      method = "svmRadial", trControl = train_control, metric = "RMSE",
#                      tuneGrid = svm_grid,
#                      )
# svm_toc = proc.time()[3] - tic
# svm_toc

plot(svm_radial)
```



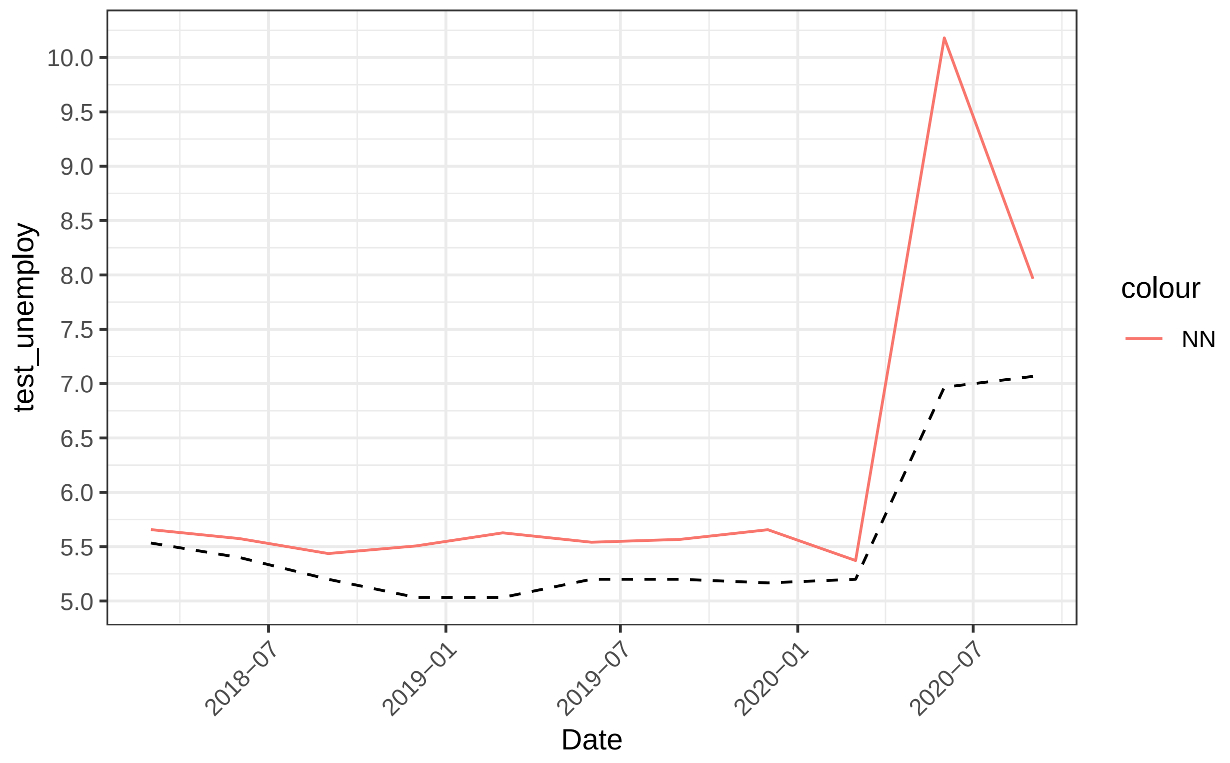
```
NN_one_layer_df %>%
  mutate(Date = as.Date(Date),
         group = as.factor(1)) %>%
  ggplot()+
  geom_line(aes(x = Date, y = test_unemploy, group = group), linetype ="dashed")+
  geom_line(aes(x = Date, y = predict_NN, color = "NN", group = group))+
  scale_y_continuous(breaks = seq(3,14,0.5))+
  theme_bw()+
  labs(title = "Test Dataset", subtitle = "Graph 5: Shows Prediction of Test Data for VNN one Layer")+
  theme(axis.text.x = element_text(angle = 45, hjust=1))
```



```
NN_reduced_neurons_df %>%
  mutate(Date = as.Date(Date),
         group = as.factor(1)) %>%
  ggplot()+
  geom_line(aes(x = Date, y = test_unemploy, group = group), linetype ="dashed")+
  geom_line(aes(x = Date, y = predict_NN, color = "NN", group = group))+
  scale_y_continuous(breaks = seq(3,14,0.5))+
  theme_bw()+
  labs(title = "Test Dataset", subtitle = "Graph 6: Shows Prediction of Test Data for VNN Reduced Neuro
  theme(axis.text.x = element_text(angle = 45, hjust=1))
```

Test Dataset

Graph 6: Shows Prediction of Test Data for VNN Reduced Neurons



```
#function calculating RMSE
RMSE = function(actual, pred){
  sqrt(mean((pred - actual)^2))
}
#Reference: https://stackoverflow.com/questions/26237688/rmse-root-mean-square-deviation-calculation-in

Rsquare <- function(actual, pred){
  rss <- sum((pred - actual) ^ 2)  ## residual sum of squares
  tss <- sum((actual - mean(actual)) ^ 2)  ## total sum of squares
  rsq <- 1 - rss/tss
  return(rsq)
}

#Reference: https://stackoverflow.com/questions/40901445/function-to-calculate-r2-r-squared-in-r

#function calculating MAE
MAE <- function(actual, pred){
  mae <- mean(abs((actual - pred)/actual))
  return (mae)
}

#Reference: https://stackoverflow.com/questions/59499222/how-to-make-a-function-of-mae-and-rae-without-
```

## References