PREDICTING UNEMPLOYMENT RATES USING SUPPORT-VECTOR MACHINES AND FEED-FORWARD NEURAL NETWORKS

Dhruvisha Gosai

Executive Summary

Machine learning has gained enormous popularity in last few decades due to how efficiently and elegantly it could provide solutions using big data. From 1957 when Frank Rosenblatt designed the first neural network for computers which simulated the thought processes of a human brain (Marr, 2016) to creating deep learning algorithms to comprehend speech-to-text; when these algorithms implemented correctly, they can help anticipate global crisis and assist in creating policies to overcome those catastrophes.

One such issue of interest is unemployment rates in Australia between Mar-2018 and Sep-2020. This period is critical to building a model as none of the global economies predicted such downfall in national GDP and employment rates.

This paper aims to create SVM and FNN algorithms to try predict unemployment rates with minimum error rate and create opportunity for further research into implementing ML techniques into macro-economic field.

Australian Unemployment Rate

Economic progression of a country primarily depends on measures of its macro-economic factors such as GDP, labour market conditions, consumer consumption behaviour, etc. When one of these indicators is negatively impacted, it creates a waterfall effect often resulting in increased unemployment, reduction in incomes, and lower consumption rate (Bank of England, 2021).

One such instance of economic downturn in Australia was in 1990-91 recession, where country's economic growth weakened, resulting in increased unemployment rates that lasted for years to follow. At an all-time low of 11% (in 1992), almost 1.7 million were unemployed jobseekers (Kryger, 1993).

Although unemployment rates dropped significantly since the 1990s potentially due to better fiscal policies, the average duration of unemployment has kept increasing since 2010 reflected a reducing rate at which jobseekers either gain employment or leave the labour force (Cassidy, Chan, Gao, & Penrose, 2020). This trend of increased unemployment rate was especially prominent since the onset of the COVID-19 pandemic where lockdowns have extended beyond 4 weeks (Australian Bureau of Statistics, 2021).

In addition to mental and physical stress individuals go through during the period of unemployment, it also impacts national budget and reduces government spending on fiscal policies, causing lower economic growth. Therefore, it is essential to be able to predict unemployment rates as accurately as possible.

Figure 1: Unemployment rate by quarter

This creates a foundation to create and implement machine learning algorithms for predicting the

Australian unemployment rates between March 2018 and September 2020.

Data

Data Import

Data for this research was sourced from Australian Bureau of Statistics between the periods of Jun-1981 and Sep-2020 for macro-economic factors for each quarter.

This data was imported in R using read_excel() from tidyverse package along with specific col_types of date for quarter and numeric for the rest. This created "NA" for the character column headings in the source dataset which were then dropped. Loaded data consisted of the 158 observations and 9 columns as below –

Field Name	Derived	Description	Format
Quarter		Each quarter as time period	Date
Unemployment		% unemployment of labor force	Numeric
GDP		% change in Gross domestic product	Numeric
GFCE		% Government final consumption expenditure	Numeric
FCE		% final consumption expenditure of all industry sectors	Numeric
trade_index		% term of trade index	Numeric
CPI		Consumer Price Index of all groups	Numeric
vacancy		Number of job vacancies - measured in thousands	Numeric
population		Estimated Resident Population - measured in thousands	Numeric

```
# Import data from excel
aus_data <- read_excel("D:/Dhru Folder/JCU - Master of Data science/MA5832 - Data Mining and Machine Learning/Assignment 3/AUS_Data.xlsx",
                       col_types = c("date","numeric","numeric","numeric","numeric","numeric","numeric","numeric"))
aus_data = aus_data[-1,]  # drop 1st row

#Adding logical column names
colnames(aus_data) <- c("quarter","unemployment","GDP","GFCE","FCE","trade_index","CPI","vacancy","population")
aus_data$quarter <- as.Date(aus_data$quarter) #Converting default POSIXct quarter to as.date
```

Data Cleaning

Visualisation using boxplot() was performed on all numeric variables for their standardised values. This helped identify two significant outliers for Jun-20 – GDP (-7%) and FCE (-8.3%). A potential reason for this being the onset of pandemic resulting in lockdowns which reduced the overall consumption expenditure.

Although these values were outliers, they weren't excluded from the dataset as these figures weren't aren't expected to be one-time anomalies and therefore need to be accounted for to avoid the potential issue of overfitting.

```
# Data spread
boxplot(scale(aus_data[,2:9]),
        col = "darkolivegreen3",
        ylab = "Standardised Value",
        las = 1,
        main = "Boxplot for data spread")
out <- round(boxplot.stats(scale(aus_data[,2:9]))$out,2)
out <- out[out < -5]
mtext(paste("Outliers: ", paste(out, collapse = ", ")))
```

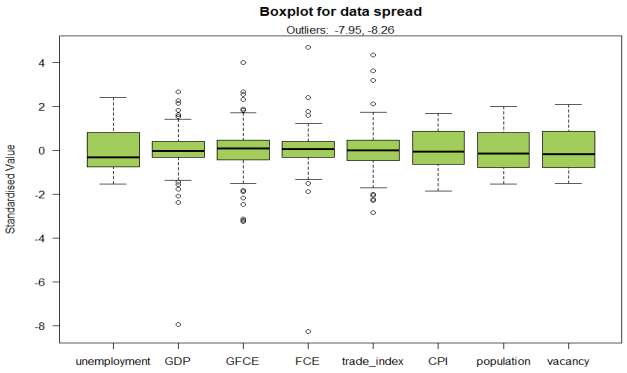

Figure 2: Boxplot for data spread

Missing value treatment

When used sum(is.na()), there were 10 missing values in the dataset – 5 for "population" and 5 for "vacancy".

Population: Given the missing values for "population" were from Sep-19 to Sep-20, a subset of dataset was created for quarters from 2010 onwards. This selection saw no significant movement in the volumes with a linear trend. Hence, extrapolation using linear regression with *quarter* as the predictor and *population* as the response variable was used to generate predicted values for "NA". This extrapolation method was tested using a ggplot() with geom_line() for actual vs. predicted which showed the fit of regression line was with 99.94% accuracy (derived from R-squared value).

```
# Missing population records - extrapolate
# ----------------------------------------
aus_pop <- aus_data[c("quarter","population")] %>%
    na.omit(aus_data) %>%
    filter(year(quarter) > 2010) # Selecting records from 2010 only to have accurate extrapolation

aus_pop$pred1 <- predict(lm(population ~ quarter, data=
```

```
aus_pop)) # Using lm as population growth is assumed to be linear
```

Vacancy: On the contrary to "population", "vacancy" had missing values between Sep-18 and Sep-19 with no linear trend with consistent fluctuations. This meant that imputation would be the best choice. To get the most accurate values to impute, data was subset to be between 2004 and 2016. Using na.approx() from package zoo, imputed values were added back to full dataset and visualised using ggplot() with geom_line() to confirm if the imputation values were as expected.

```
# Missing vacancy records - Imputation/interpolation
# ----------------------------------
aus_vacancy <- aus_data[c("quarter","vacancy")] %>%
    filter(year(quarter) > 2004 & year(quarter) < 2016)
# Selecting records between 2005 and 2015 to have accurate imputation

aus_vacancy$vacancy <- zoo::na.approx(aus_vacancy$vacancy)
```

Descriptive statistics and data visualisation

Following to preliminary data cleaning and missing value treatments, describe() from psych package and ggpairs from GGally package were used to comprehend the data composition and its spread.

From descriptive statistics in figure 3, it was evident that 4 variables were negatively skewed and "population" and "vacancy" were measured in thousands whereas rest of the variables were recorded in percentages. Based on this, data was standardised using "scale" and "center" when creating models.

```
describe(aus_data)
```

```
> describe(aus_data)
             vars   n       mean         sd     median
quarter         1 158        NaN         NA         NA
unemployment    2 158       6.85       1.79       6.25
GDP             3 158       0.72       0.97       0.70
GFCE            4 158       0.85       1.67       1.00
FCE             5 158       0.76       1.10       0.80
trade_index     6 158       0.35       2.97       0.30
CPI             7 158      75.08      25.02      73.50
population      8 158 196795.80 30820.55 191831.08
vacancy         9 158     113.10      57.33     102.50
             trimmed       mad       min       max
quarter          NaN        NA       Inf      -Inf
unemployment    6.69      1.66       4.1     11.13
GDP             0.76      0.59      -7.0      3.30
GFCE            0.88      1.11      -4.6      7.50
FCE             0.79      0.59      -8.3      5.90
trade_index     0.28      2.08      -8.1     13.20
CPI            75.41     28.69      28.4     116.60
population 195465.11 35487.96 149232.6 258180.31
vacancy        110.75     68.35      26.8     232.30
               range  skew kurtosis       se
quarter          -Inf    NA       NA      NA
unemployment     7.03  0.67    -0.56    0.14
GDP             10.30 -3.00    23.85    0.08
GFCE            12.10 -0.12     2.60    0.13
FCE             14.20 -2.84    30.18    0.09
trade_index     21.30  0.67     3.02    0.24
CPI             88.20 -0.03    -1.09    1.99
population  108947.71  0.35    -1.03 2451.95
vacancy        205.50  0.30    -1.05    4.56
```

Figure 3: Describtive statistics for dataset

To be able to predict unemployment rates most accurately, spread of the data needed to be understood which was done via GGally::ggpairs() as seen in figure 4. It was noted that with 3 of 8 predictive variables heavily correlated (above 0.9), a model would become vulnerable towards a very small change in the data (Raj, 2019) unless used a non-linear model.

```
gg <- GGally::ggpairs(aus_data[,2:9])
gg
```

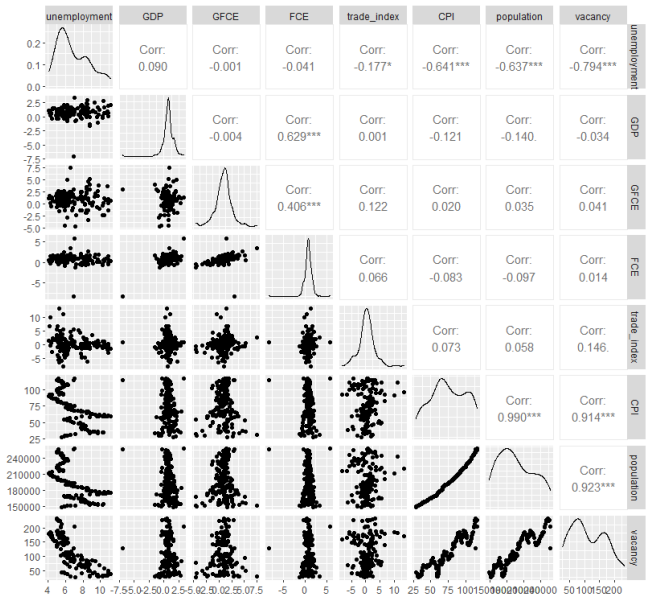

Figure 4: Variable plot for data spread

Hence the model choice for this analysis were SVM using radial basis function (RBF) kernel and feed-forward neural networks (FNN) which is comprised of non-linear activation functions.

Preceding to creating the model, data was partitioned into test and training split where "train" consisted of all the observations prior to Mar-18 quarter totaling to 147 records and "test" with 11 observations between Mar-18 and Sep-20.

```
# Split the data
set.seed(7789)
train <- aus_data %>%
    filter(quarter < "2018-03-01")
test  <- aus_data %>%
    filter(quarter >= "2018-03-01")
```

Analysis and Investigation

All analysis and statistical procedures were performed in R version 1.4.1717 with a list of libraries referenced in Appendix (RStudio Team, 2021).

Algorithm 1: Support Vector Machines (SVM)

To predict the unemployment rate for the macro-economic variables with non-linear spread, SVM regression was used. This algorithm allows the presence of non-linearity in data providing a proficient prediction model (Raj A. , 2020). To address the non-linearity, a more generalized form of kernel was selected – RBF kernel, as it performs similar to Gaussian distribution (Sreenivasa, 2020). It calculates Euclidean distance between the two points.

To create SVM model, caret package was used on the training dataset with method of "svmRadial" and in-built pre-processing function to standardize the data using "center" and "scale". With such a small sample size, cross-validation was required to avoid overfitting so train control parameters were set to run 10-fold cross-validations repeated 3 times, saving the predictions for each run.

```
# ALGORITHM 1: SVM
set.seed(7789)

# Specify training control cross validation parameters
train_control <- trainControl(method="repeatedcv",
                              number=10,
                              repeats=3,
                              savePredictions=TRUE)
train_svm <- as.matrix(train[,2:9]) # Caret for svm req
uires input of matrix dataset form

svm_start <- Sys.time() # Start time of the model
set.seed(7789)
SVM_caret <- caret::train(unemployment ~ .,
                          data = train_svm,
                          method = "svmRadial",
                          trControl = train_control,
                          preProcess = c("center","scal
e"),
                          tuneLength = 10)
print(SVM_caret)

svm_end <- Sys.time()  # End time of the model
print(svm_end - svm_start) #Time taken to run model

SVM_results <- data.frame(SVM_caret$results) # Add resu
lts into a dataframe to plot
# GGPlot for best C value against Rsquared
ggplot(aes(x = C, y=Rsquared), data=SVM_results) +
    geom_line() +
    geom_text(aes(label=C), color="red")+
    ggtitle(" SVM Model performance for each cost where
sigma = 0.2241661")+
    scale_y_continuous(limits=c(0.68, 0.83))+
    theme_bw()

SVM_caret$bestTune # maximizes model accuracy - Use the
se C and sigma

SVM_Grid <- expand.grid(.C = SVM_caret$bestTune$C,.sigm
a=SVM_caret$bestTune$sigma) # Results from grid search
tuning

set.seed(7789)
SVM_tuned.2 <- caret::train(unemployment ~ .,
                            data = train_svm,
                            method = "svmRadial",
                            tuneGrid = SVM_Grid,
                            trControl = train_control,
                            preProcess = c("center","sc
ale"),
                            tuneLength = 10)

print(SVM_tuned.2) #R_squared = 0.8003504
```

This provided with resampling results across tuning parameters along with Root Mean Squared Error (RMSE), R-Squared value and Mean Average Error (MAE) against it. Plotting the performance metric of R-squared against the cost function "C", at (C) = 2 and sigma value of 0.224, R-squared value of 0.8065 revealed that 80.65% of the variance in the unemployment rates could be explained by the independent variables used to fit the SVM model. A plot with SVM predicted values against the actual unemployment rates showed significantly low variance.

```
pred_SVM <- predict(SVM_tuned.2,train_svm) # Predict to
assess model performance

# GGPlot for tuned svm model
ggplot(train, aes(x = quarter, y=unemployment)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = pred_SVM), color="red")+ #untuned
model
    ggtitle(" SVM Model performance (Tuned) - Training
data")+
    scale_y_continuous(limits=c(3, 12))+
    theme_bw()
```

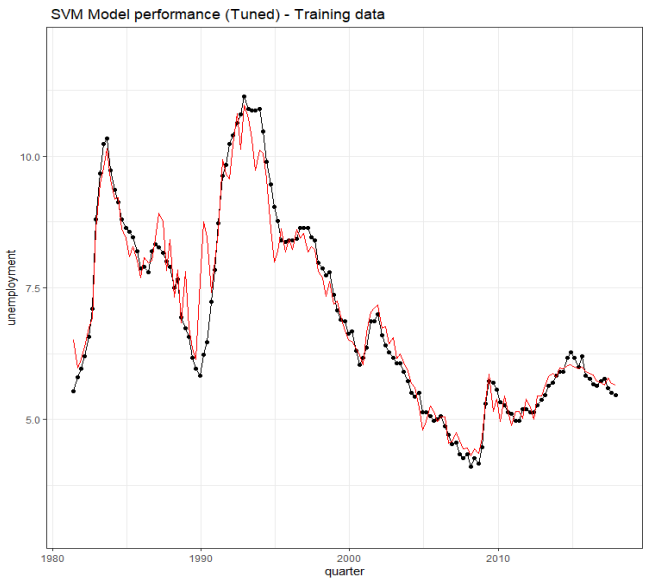

Figure 5: SVM model performance on Training data

But to test how the model performed against the test data, predict() function was used with "test" dataset and plotted for the period of Mar-18 to Sep-20 as seen in figure 5. It was noticeable how

accurately the model predicted the spike in unemployment rates due to the pandemic.

```
test_svm <- as.matrix(test[,2:9])# Caret for svm requir
es input of matrix dataset form
pred_SVM_test <- predict(SVM_tuned.2,test_svm) # Predic
t to assess model performance

res <- caret::postResample(pred_SVM_test,test$unemploym
ent) #RMSE - root of mean squared error
rsquare <- res[2] # R-square helps in assessing model p
erformance on test
print(rsquare) #0.8435274

# GGPlot for tuned svm model
ggplot(test, aes(x = quarter, y=unemployment)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = pred_SVM_test), color="red")+ #un
tuned model
    ggtitle(" SVM Model performance (Tuned) - Testing d
ata")+
    scale_y_continuous(limits=c(3, 12))+
    theme_bw()
```

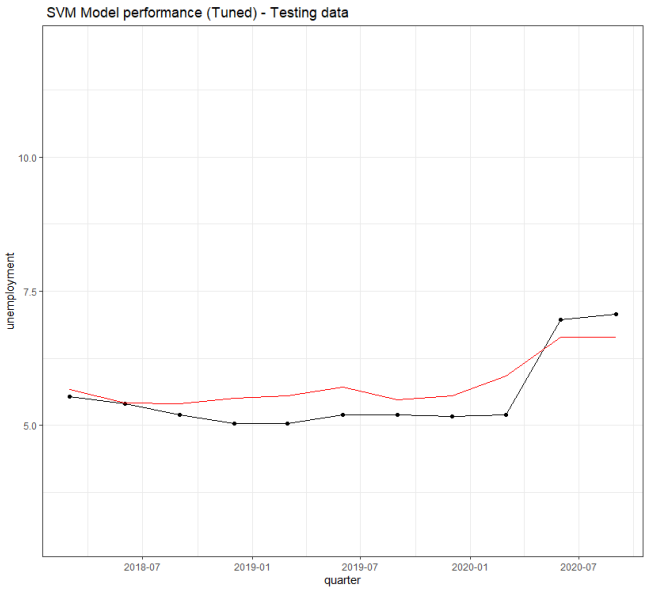

Figure 6: SVM model performance on Test data

When calculating the R-squared value using caret::postResample() function for "test" data, value of 0.87 indicated that 87% of the movement in unemployment rates were explained by the explanatory variables.

Algorithm 2: Artificial Neural Networks (FNN)

A more advanced form of machine learning model used in unemployment rate prediction was neural networks where the network learns something simple at initial layer and sends that information to the next layer (Murnane, 2016). The aim here was to create a multi-layered neural network where macro-economic variables were inputted and recurrently transformed by the layer's weights to produce prediction of unemployment rate.

To find the accurate values of the layer weights, loss function MSE was used which took predictions of the network and actual values to compute loss score, capturing the network performance. An additional requirement of FNN is to have an optimizer that adjusts weights within the layers using a backpropagation algorithm.

Because the data only consists of 147 records for the train and 11 for test, and two of those variables on a different scale, data had to be scaled – similar to SVM. Normalization was performed using scaler_standard() function and only two hidden layers were created with 113 and 65 units each to mitigate overfitting in such small training sample.

```
# ALGORITHM 2: Neural Networks
ann_time <- Sys.time()

# Define hyper parameters
# -----------------------
verbose = 1
validation = 0.15  # validation split (% of dataset to
be withheld for validation)
epoch = 50         # iterations of dataset

train_ann <- train[,2:9]
test_ann  <- test[,2:9]

# Scaling using python
# --------------------
spec <- feature_spec(train_ann, unemployment ~ . ) %>%
  step_numeric_column(all_numeric(), normalizer_fn = sc
aler_standard()) %>%
  fit()
spec
```

FNN was created using Keras and Tensorflow libraries. To build a full-connected network on Keras, an input layer was specified using layer_input_from_dataset(). Followed by a nonlinear activation function for the hidden layers. Rectified linear activation function (ReLu) was used to create two hidden layers. There was no requirement of last-layer activation as the output was expected to be with arbitrary values of unemployment rates.

For this type of output, mean squared error (MSE) loss function would provide the difference between the predictions and the targets. During the training, metrics such as mean absolute error (MAE) - which is the absolute difference between the predicted and the actuals, MSE and mean absolute percentage error (MAPE) that measures prediction accuracy of forecasting.

Setting the hyperparameters with verbose=1 to get visibility of the historic model data, validation split of 15% where that % of data would be withheld for validation, epoch was set to 50 for the number of iterations, and the optimizer set for "RMSprop" which is a gradient-based optimizer used in this instance as it is better at changing direction by dividing the learning rate by an exponentially decaying average of squared gradients (Ruder, 2016).

```
# set the seeds
set.seed(7789)
set_random_seed(7789)
build_model <- function() {
  input <- layer_input_from_dataset(train_ann)
  output <- input %>%
    layer_dense_features(dense_features(spec))   %>%
    layer_dense(units = 113, activation = "relu",
                input_shape = dim(train_ann)[[2]]) %>%
    layer_dense(units = 65, activation = "relu") %>%
    layer_dense(units = 1)

  model <- keras_model(input, output)

  model %>%
    compile(
      loss = "mse",
      optimizer = "rmsprop",
      metrics = list("mae","mape","mse")
      )
}

model <- build_model()

print_dot_callback <- callback_lambda(  on_epoch_end =
function(epoch, logs) {
  if (epoch %% 80 == 0) cat("\n")
  cat(".") } )
```

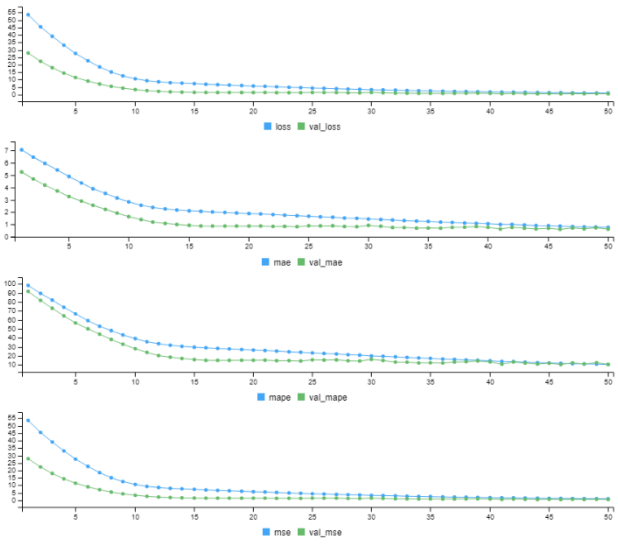

Figure 7: FNN epoch selection and model performance

Provided plot enables the visualization of training and validation metrics using epoch. When MAE stopped improving after 49 epochs, optimum epoch selection could have been 40.

However, with MAE of 3.42, and MAPE set at 52.64%, model performance seemed poor. Predicted values were calculated for training and test data to have them plotted in "red" against the actuals to visualize the model performance.

```
train_predictions <- model %>% predict(train_ann)
train_predictions[,1]
paste0("MAE: " , round(mae,2))
paste0("MAPE: ", round(mape,2),"%")
paste0("MSE: " , round(mse,2))

ggplot(train, aes(x = quarter, y=unemployment)) +
  geom_line() +
  geom_point() +
  geom_line(aes(y = train_predictions), color="red")+ #
model output
  ggtitle(" ANN Model performance - Train data")+
  # scale_y_continuous(limits=c(3, 12))+
  theme_bw()
```

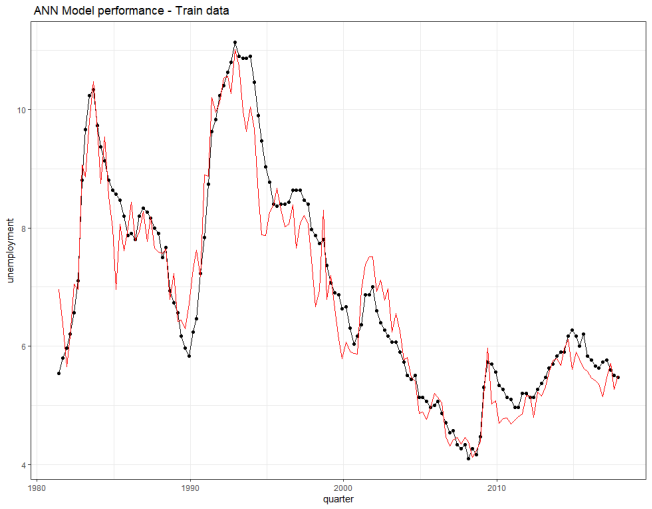

Figure 8: FNN model performance on Training data

When plotted for test data, up until Jan-20, the predictions seemed more or less aligned to the actuals be when predicting the unemployment rates during unprecedented COVID-19 pandemic, model performed very poorly. This could issue could be overcome by introduction of additional hidden layers to and increasing the cross-validation folds.

```
test_predictions <- model %>% predict(test_ann)
test_predictions[,1]
paste0("MAE: ", round(mae,2))
paste0("MAPE: ", round(mape,2),"%")
paste0("MSE: ", round(mse,2))

ggplot(test, aes(x = quarter, y=unemployment)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = test_predictions), color="red")+
#model output
    ggtitle(" ANN Model performance - Test data")+
    # scale_y_continuous(limits=c(3, 12))+
    theme_bw()
```

```
ann_end <- Sys.time()
ann_end - ann_time # Total run-time of 9.52 seconds
```

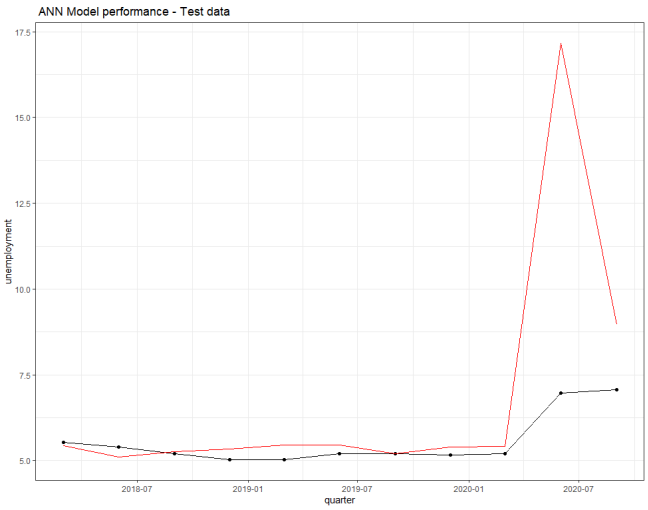

Figure 9: ANN model performance on Test data

Model Comparison and Contrasting

Before finalizing which model predicts unemployment rates more accurately, there are other metrics that need to be investigated.

Cross-Validation accuracy

SVM used 10-fold cross validation repeated 3 times which provided the optimum cost and sigma at the lowest RMSE of 0.8047. On the contrary, epochs of 50 was used for validation to get the optimum tuning parameters and improve predictability.

Computational time to train models

Model run-time was monitored using sys.time() at the start and end of the code and time taken for model to run was calculated by taking the difference of start and end of the model run.

SVM took 3.5 seconds to run end-to-end starting from full model, plots to final model with tuned parameters and test predictions. Whereas FNN too 9.5 seconds to run with multiple hidden layers and performance visualization.

Interpretability

SVM results were intuitive and easy to interpret with r-squared value explaining over 80% of the variance. Whereas FNN had lack of interpretable results with quite a few tuning parameters and higher possibility to causing overfitting problems.

Concluding Remarks

In this research, there were a few limitations where sample size was significantly lower to make accurate predictions, but the created models provided a basis for implementing a machine learning model in government Treasury or policy making department to support with fiscal policies requiring budgeting for unemployment assistance in advance.

Comparing the two models, SVM was crowned as the superior choice with predictions having 87% goodness of fit with predicted trendline very closely representing the test data. In addition to that, model run-time was less than half of FNN, and much easier interpretability to be able to update tuning parameters as and when required.

Finally, this research would provide basis to create hybrid models utilizing economic model metrics of ARIMA and implementing them into the SVM and FNN.

References

Australian Bureau of Statistics. (2021, 08). *Labour Force, Australia*. Retrieved from Australian Bureau of Statistics: https://www.abs.gov.au/statistics/labour/employment-and-unemployment/labour-force-australia/latest-release

Bank of England. (2021). *Bank of England*. Retrieved from Why does economic growth matter?: https://www.bankofengland.co.uk/knowledgebank/why-does-economic-growth-matter

Cassidy, N., Chan, I., Gao, A., & Penrose, G. (2020, 12 10). *Long-term Unemployment in Australia*. Retrieved from Reserve Bank of Australia: https://www.rba.gov.au/publications/bulletin/2020/dec/long-term-unemployment-in-australia.html

Kryger, T. (1993, 12 15). Trends in Unemployment and Underemployment. *Department of the Parliamentary Library*.

Marr, B. (2016, 02 19). *A Short History of Machine Learning -- Every Manager Should Read*. Retrieved from Forbes: https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/?sh=1908e36415e7

Murnane, K. (2016, 04 01). *What Is Deep Learning And How Is It Useful?* Retrieved from Forbes: https://www.forbes.com/sites/kevinmurnane/2016/04/01/what-is-deep-learning-and-how-is-it-useful/?sh=4b9e74ddd547

Raj, A. (2020, 10 03). *Unlocking the True Power of Support Vector Regression*. Retrieved from Towards Data science: https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0

Raj, S. (2019, 08 10). *Effects of Multi-collinearity in Logistic Regression, SVM, Random Forest(RF)*. Retrieved from Medium: https://medium.com/@raj5287/effects-of-multi-collinearity-in-logistic-regression-svm-rf-af6766d91f1b

Reserve bank of Australia. (2021, 05). *5. Economic Outlook*. Retrieved from Reserve bank of Australia: https://www.rba.gov.au/publications/smp/2021/may/economic-outlook.html

RStudio Team. (2021). RStudio: Integrated Development Environment for R. *RStudio, PBC*. Boston, MA: URL http://www.rstudio.com/. Retrieved from RStudio, PBC: http://www.rstudio.com/

Ruder, S. (2016, 01 19). *An overview of gradient descent optimization algorithms*. Retrieved from Ruder: https://ruder.io/optimizing-gradient-descent/

Sreenivasa, S. (2020, 10 12). *Radial Basis Function (RBF) Kernel: The Go-To Kernel*. Retrieved from Towards Data science: https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a

Appendix

```
# Install packages if not available already
if (!require("car")) install.packages("car")
if (!require("datasets")) install.packages("datasets")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("dplyr")) install.packages("dplyr")
if (!require("tidyverse")) install.packages("tidyverse")
if (!require("qqplotr")) install.packages("qqplotr")
if (!require("ggfortify")) install.packages("ggfortify")
```

```r
if (!require("ggthemes")) install.packages("ggthemes")
if (!require("hrbrthemes")) install.packages("hrbrthemes")
if (!require("ISLR")) install.packages("ISLR")
if (!require("caret")) install.packages("caret")
if (!require("GGally")) install.packages("GGally")
if (!require("knitr")) install.packages("knitr")
if (!require("MASS")) install.packages("MASS")
if (!require("ROCR")) install.packages("ROCR")
if (!require("corrplot")) install.packages("corrplot")
if (!require("ggridges")) install.packages("ggridges")
if (!require("klaR")) install.packages("klaR")
if (!require("psych")) install.packages("psych")
if (!require("yaml")) install.packages("yaml")
if (!require("cluster")) install.packages("cluster")
if (!require("factoextra")) install.packages("factoextra")
if (!require("reshape2")) install.packages("reshape2")
if (!require("broom")) install.packages("broom")
if (!require("aod")) install.packages("aod")
if (!require("ggpubr")) install.packages("ggpubr")
if (!require("gridExtra")) install.packages("gridExtra")
if (!require("fpc")) install.packages("fpc")
if (!require("datarium")) install.packages("datarium")
if (!require("e1071")) install.packages("e1071")
if (!require("glmpath")) install.packages("glmpath")
if (!require("quadprog")) install.packages("quadprog")
if (!require("data.table")) install.packages("data.table")
if (!require("readxl")) install.packages("readxl")
if (!require("randomForest")) install.packages("randomForest")
if (!require("e1071")) install.packages("e1071")
if (!require("rpart")) install.packages("rpart")
if (!require("rpart.plot")) install.packages("rpart.plot")
if (!require("rattle")) install.packages("rattle")
if (!require("gbm")) install.packages("gbm")
if (!require("ranger")) install.packages("ranger")
if (!require("doParallel")) install.packages("doParallel")
if (!require("rminer")) install.packages("rminer")
if (!require("reticulate")) install.packages("reticulate") #ANN
if (!require("tensorflow")) install.packages("tensorflow") #ANN
if (!require("tfdatasets")) install.packages("tfdatasets") #ANN
if (!require("keras")) install.packages("keras") #ANN
if (!require("zoo")) install.packages("zoo") #imputation
if (!require("Metrics")) install.packages("Metrics") #imputation
if (!require("neuralnet")) install.packages("neuralnet") #ANN

# Loading relevant R packages
library(car, warn.conflicts = F, quietly = T)
library(datasets, warn.conflicts = F, quietly = T)
library(ggplot2, warn.conflicts = F, quietly = T)
library(MASS, warn.conflicts = F, quietly = T)
library(dplyr, warn.conflicts = F, quietly = T) #for piping
library(tidyverse, warn.conflicts = F, quietly = T)
library(qqplotr, warn.conflicts = F, quietly = T) # for qq plots
library(ggfortify, warn.conflicts = F, quietly = T) # for visualisations
library(ggthemes, warn.conflicts = F, quietly = T) # for ggplot themes
library(hrbrthemes, warn.conflicts = F, quietly = T) # for ggplot background themes
library(ISLR, warn.conflicts = F, quietly = T) #for data
library(caret, warn.conflicts = F, quietly = T) #for splitting the data
```

```
library(GGally, warn.conflicts = F, quietly = T)
library(knitr, warn.conflicts = F, quietly = T) # to add appendix in the end
library(ROCR, warn.conflicts = F, quietly = T)
library(corrplot, warn.conflicts = F, quietly = T) # Correlation matrix
library(ggridges, warn.conflicts = F, quietly = T)
library(klaR, warn.conflicts = F, quietly = T)
library(psych, warn.conflicts = F, quietly = T) # Visualise
library(yaml, warn.conflicts = F, quietly = T)
library(cluster, warn.conflicts = F, quietly = T)
library(factoextra, warn.conflicts = F, quietly = T)
library(reshape2, warn.conflicts = F, quietly = T)#reshaping data
library(broom, warn.conflicts = F, quietly = T)
library(aod, warn.conflicts = F, quietly = T) # for wald test
library(ggpubr, warn.conflicts = F, quietly = T)
library(gridExtra, warn.conflicts = F, quietly = T)
library(fpc, warn.conflicts = F, quietly = T)
library(datarium, warn.conflicts = F, quietly = T) #to get marketing dataset from data
rium
library(e1071, warn.conflicts = F, quietly = T)    #for svm
library(glmpath, warn.conflicts = F, quietly = T)  #for svm
library(quadprog, warn.conflicts = F, quietly = T)  #for QP
library(data.table, warn.conflicts = F, quietly = T)
library(readxl, warn.conflicts = F, quietly = T) # importing xls file type
library(randomForest, warn.conflicts = F, quietly = T)   #For applying Random Forest
library(e1071, warn.conflicts = F, quietly = T)          #For SVM
library(rpart, warn.conflicts = F, quietly = T)          #For tree models
library(rpart.plot, warn.conflicts = F, quietly = T)     #for plotting tree
library(rattle, warn.conflicts = F, quietly = T)
library(gbm, warn.conflicts = F, quietly = T)
library(ranger, warn.conflicts = F, quietly = T)
library(doParallel, warn.conflicts = F, quietly = T)
library(rminer, warn.conflicts = F, quietly = T)
library(reticulate, warn.conflicts = F, quietly = T) #ANN
library(tensorflow, warn.conflicts = F, quietly = T) #ANN
library(tfdatasets, warn.conflicts = F, quietly = T)
library(keras, warn.conflicts = F, quietly = T) #ANN
library(zoo, warn.conflicts = F, quietly = T) #imputation
library(Metrics, warn.conflicts = F, quietly = T) #ML metrics
library(neuralnet, warn.conflicts = F, quietly = T) #ML metrics

set_random_seed(7789) # For tensorflow and keras


# Import data from excel
aus_data <- read_excel("D:/Uni/MA5832_AdvMachineLearning/Assignment 3/AUS_Data.xlsx",
                       col_types = c("date","numeric","numeric","numeric","numeric","n
umeric","numeric","numeric","numeric"))

# aus_data <- read_excel("D:/Dhru Folder/JCU - Master of Data science/MA5832 - Data Mi
ning and Machine Learning/Assignment 3/AUS_Data.xlsx",
#                        col_types = c("date","numeric","numeric","numeric","numeric",
"numeric","numeric","numeric","numeric"))
aus_data = aus_data[-1,]  # drop 1st row

#Adding logical column names
colnames(aus_data) <- c("quarter","unemployment","GDP","GFCE","FCE","trade_index","CPI
","vacancy","population")
```

```r
aus_data$quarter <- as.Date(aus_data$quarter) #Converting default POSIXct quarter to a
s.date

summary(aus_data)

# Data spread
boxplot(scale(aus_data[,2:9]),
        col = "darkolivegreen3",
        ylab = "Standardised Value",
        las = 1,
        main = "Boxplot for data spread")
out <- round(boxplot.stats(scale(aus_data[,2:9]))$out,2)
out <- out[out < -5]
mtext(paste("Outliers: ", paste(out, collapse = ", ")))

# GGPlot to make sure linear growth of population is met + check prediction accuracy
ggplot(aus_data, aes(x = quarter, y=unemployment)) +
    geom_line(aes(y=unemployment), colour="red") +
    ggtitle("Unemployment rate (in %) in Australia")+
    # scale_y_continuous(limits=c(3, 12))+
    theme_bw()+
    theme(plot.title = element_text(size = 16),
          axis.title = element_text(size = 12, face = "bold"))

# ======================
# Data cleaning
# ======================
sum(is.na(aus_data))

# ------------------------------------------
# Update GDP, FCE outlier values - Imputation/interpolation
# ------------------------------------------

# Delete existing outlier values
# aus_data <- aus_data %>%
#             mutate(GDP = ifelse(quarter=="2020-06-01",NA,GDP),
#                    FCE = ifelse(quarter=="2020-06-01",NA,FCE)) ##Removal of outliers
#
# aus_gdp_fce$GDP <- zoo::na.approx(aus_data$GDP)
# aus_gdp_fce$FCE <- zoo::na.approx(aus_data$FCE)
#
# aus_gdp_fce <- aus_gdp_fce %>%
#                 select(quarter,GDP,FCE)
#
# aus_data <- aus_data %>% left_join(aus_gdp_fce, by="quarter") %>%
#   mutate(GDP = coalesce(GDP.x, GDP.y),
#          FCE = coalesce(FCE.x, FCE.y))%>%
#   dplyr::select(-c("GDP.x","GDP.y","FCE.x","FCE.y" ))

# boxplot(scale(aus_data[,2:9]),
#         col = "darkolivegreen3",
#         ylab = "Standardised Value",
#         las = 1,
#         main = "Boxplot for data spread - post taking outliers out")
# out <- round(boxplot.stats(scale(aus_data[,2:9]))$out,2)
# out <- out[out < -5]
# mtext(paste("Outliers: ", paste(out, collapse = ", ")))
```

```r
# -------------------------------------
# Missing population records - extrapolate
# -------------------------------------
aus_pop <- aus_data[c("quarter","population")] %>%
    na.omit(aus_data) %>%
    filter(year(quarter) > 2010) # Selecting records from 2010 only to have accurate e
xtrapolation

aus_pop$pred1 <- predict(lm(population ~ quarter, data=aus_pop)) # Using lm as populat
ion growth is assumed to be linear

# GGPlot to make sure linear growth of population is met + check prediction accuracy
ggplot(aus_pop, aes(x = quarter, y=population)) +
    geom_line() +
    geom_point() +
    geom_hline(aes(yintercept=0))+
    geom_line(aes(y = pred1), color="red")+
    ggtitle("Population growth against predicted (using lm)")+
    theme_bw()+
    theme(plot.title = element_text(size = 16),
          axis.title = element_text(size = 12, face = "bold"))

# Create new dataset with extrapolation based on model
pred <- data.frame(quarter=seq(as.Date("2011-03-01"), as.Date("2020-09-01"), by = "qua
rter"))

lm_population <- (lm(population ~ quarter, data=aus_pop))
summary(lm_population)

pred$population <- predict(lm_population,newdata=pred) #Add predictions

#Actual vs. predicted
ggplot(aus_pop, aes(x = quarter, y=population)) +
    geom_line() +
    geom_point() +
    geom_line(color="red", data=pred)+
    ggtitle("Extrapolated data using lm()")+
    scale_y_continuous(limits=c(2e+05, 2.7e+05))+
    theme_bw()+
    theme(plot.title = element_text(size = 16),
          axis.title = element_text(size = 12, face = "bold"))

pred <- pred %>% filter(quarter > as.Date("2019-06-01"))

aus_data <- aus_data %>% left_join(pred, by="quarter") %>%
    mutate(population = coalesce(population.x, population.y)) %>%
    dplyr::select(-c("population.x","population.y"))
# ---------------------------------
# Missing vacancy records - Imputation/interpolation
# ---------------------------------
aus_vacancy <- aus_data[c("quarter","vacancy")] %>%
    filter(year(quarter) > 2004 & year(quarter) < 2016) # Selecting records between 20
05 and 2015 to have accurate imputation

aus_vacancy$vacancy <- zoo::na.approx(aus_vacancy$vacancy)
```

```r
aus_data <- aus_data %>% left_join(aus_vacancy, by="quarter") %>%
    mutate(vacancy = coalesce(vacancy.x, vacancy.y))

# GGPlot to make sure linear growth of population is met + check prediction accuracy
ggplot(aus_data, aes(x = quarter, y=vacancy.x)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = vacancy.y), color="red")+
    ggtitle("Vacancy value imputations")+
    theme_bw()+
    theme(plot.title = element_text(size = 16),
          axis.title = element_text(size = 12, face = "bold"))

aus_data <- aus_data %>% dplyr::select(-c("vacancy.x","vacancy.y"))
# ----------------------------------

# ================================
# Data Exploration & Visualisation
# ================================

describe(aus_data)

# Correlation to get relationship between variables
# M <- round(cor(aus_data[,2:9]), 2) # Create the correlation matrix
# corrplot(M,order="hclust",
#          tl.cex = 0.90,
#          addCoef.col ='black',
#          method = "square",
#          type = 'lower',
#          diag = FALSE)# Create corr plot
# title("Correlation plot for numeric variables")

# plot variables to understand the spread
gg <- GGally::ggpairs(aus_data[,2:9])
gg


# ======================
# Data Partition
# ======================

sum(is.na(aus_data)) # Check for missing data
summary(aus_data) # Check summary before running models


# Split the data into 70% train and 30% test
set.seed(7789)
train <- aus_data %>%
    filter(quarter < "2018-03-01")
test  <- aus_data %>%
    filter(quarter >= "2018-03-01")

dim(train)

# Convert unemployment as.factor
# train$unemployment <- as.factor(train$unemployment)
# test$unemployment  <- as.factor(test$unemployment)
```

```r
# ======================
# ALGORITHM 1: SVM
# ======================
#Support Vector Machine (using radial kernel)
set.seed(7789)

# Specify training control cross validation parameters
train_control <- trainControl(method="repeatedcv",
                              number=10,
                              repeats=3,
                              savePredictions=TRUE)

# SVM model - without tuning
# --------------------------
train_svm <- as.matrix(train[,2:9]) # Caret for svm requires input of matrix dataset f
orm

svm_start <- Sys.time() # Start time of the model
set.seed(7789)
SVM_caret <- caret::train(unemployment ~ .,
                          data = train_svm,
                          method = "svmRadial",
                          trControl = train_control,
                          preProcess = c("center","scale"),
                          tuneLength = 10)

print(SVM_caret)

svm_end <- Sys.time()  # End time of the model
print(svm_end - svm_start) #Time taken to run model

SVM_results <- data.frame(SVM_caret$results) # Add results into a dataframe to plot
# GGPlot for best C value against Rsquared
ggplot(aes(x = C, y=Rsquared), data=SVM_results) +
    geom_line() +
    geom_text(aes(label=C), color="red")+
    ggtitle(" SVM Model performance for each cost where sigma = 0.2241661")+
    scale_y_continuous(limits=c(0.68, 0.83))+
    theme_bw()

SVM_caret$bestTune # maximizes model accuracy - Use these C and sigma

SVM_Grid <- expand.grid(.C = SVM_caret$bestTune$C,.sigma=SVM_caret$bestTune$sigma) # R
esults from grid search tuning

set.seed(7789)
SVM_tuned.2 <- caret::train(unemployment ~ .,
                            data = train_svm,
                            method = "svmRadial",
                            tuneGrid = SVM_Grid,
                            trControl = train_control,
                            preProcess = c("center","scale"),
                            tuneLength = 10)

print(SVM_tuned.2) #R_squared = 0.8003504
pred_SVM <- predict(SVM_tuned.2,train_svm) # Predict to assess model performance
```

```r
# GGPlot for tuned svm model
ggplot(train, aes(x = quarter, y=unemployment)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = pred_SVM), color="red")+ #untuned model
    ggtitle(" SVM Model performance (Tuned) - Training data")+
    scale_y_continuous(limits=c(3, 12))+
    theme_bw()

test_svm <- as.matrix(test[,2:9])# Caret for svm requires input of matrix dataset form

pred_SVM_test <- predict(SVM_tuned.2,test_svm) # Predict to assess model performance
# GGPlot for tuned svm model
ggplot(test, aes(x = quarter, y=unemployment)) +
    geom_line() +
    geom_point() +
    geom_line(aes(y = pred_SVM_test), color="red")+ #untuned model
    ggtitle(" SVM Model performance (Tuned) - Testing data")+
    scale_y_continuous(limits=c(3, 12))+
    theme_bw()

res <- caret::postResample(pred_SVM_test,test$unemployment) #RMSE - root of mean squar
ed error
rsquare <- res[2] # R-square helps in assessing model performance on test
print(rsquare) #0.8435274

# ====================================
# ALGORITHM 2: Neural Networks
# ====================================
# get current time to check duration of the neural network run time at the end
ann_time <- Sys.time()

# Define hyper parameters
# -----------------------
verbose = 1
validation = 0.15  # validation split (% of dataset to be withheld for validation)
epoch = 50       # iterations of dataset

train_ann <- train[,2:9]
test_ann  <- test[,2:9]

# Scaling using python
# --------------------
spec <- feature_spec(train_ann, unemployment ~ . ) %>%
  step_numeric_column(all_numeric(), normalizer_fn = scaler_standard()) %>%
  fit()
spec

# set the seeds
set.seed(7789)
set_random_seed(7789)
build_model <- function() {
  input <- layer_input_from_dataset(train_ann)
  output <- input %>%
    layer_dense_features(dense_features(spec))   %>%
    layer_dense(units = 113, activation = "relu",
```

```
                input_shape = dim(train_ann)[[2]]) %>%
    layer_dense(units = 65, activation = "relu") %>%
    layer_dense(units = 1)

  model <- keras_model(input, output)

  model %>%
    compile(
      loss = "mse",
      optimizer = "rmsprop",
      metrics = list("mae","mape","mse")
      )
}

model <- build_model()

print_dot_callback <- callback_lambda(  on_epoch_end = function(epoch, logs) {
  if (epoch %% 80 == 0) cat("\n")
  cat(".") } )

set.seed(7789)
set_random_seed(7789)
# This is to find optimum epochs
history <- model %>% fit(x = train_ann,
                         y = train_ann$unemployment,
                         epochs = epoch,
                         validation_split = validation,
                         verbose = verbose,
                         callbacks = list(print_dot_callback))

# plot(history)

c(loss,mae,mape,mse) %<-% (model %>% evaluate(test_ann , test_ann$unemployment, verbos
e = 0))
xxx <- (model %>% evaluate(test_ann , test$unemployment, verbose = 0))

train_predictions <- model %>% predict(train_ann)
train_predictions[,1]
paste0("MAE: ", round(mae,2))
paste0("MAPE: ", round(mape,2),"%")
paste0("MSE: ", round(mse,2))

ggplot(train, aes(x = quarter, y=unemployment)) +
  geom_line() +
  geom_point() +
  geom_line(aes(y = train_predictions), color="red")+ #model output
  ggtitle(" ANN Model performance - Train data")+
  # scale_y_continuous(limits=c(3, 12))+
  theme_bw()

test_predictions <- model %>% predict(test_ann)
test_predictions[,1]
paste0("MAE: ", round(mae,2))
paste0("MAPE: ", round(mape,2),"%")
paste0("MSE: ", round(mse,2))

ggplot(test, aes(x = quarter, y=unemployment)) +
```

```
    geom_line() +
    geom_point() +
    geom_line(aes(y = test_predictions), color="red")+ #model output
    ggtitle(" ANN Model performance - Test data")+
    # scale_y_continuous(limits=c(3, 12))+
    theme_bw()


ann_end <- Sys.time()

ann_end - ann_time # Total run-time of 9.52 seconds
```