Data Visualisation Final Project

Name: Anh Duc Le

Registration Number: R00183696

Table of Contents

A. Exploratory Data Analysis

The first step of this report is to implement Exploratory Data Analysis on the given datasets. It includes the basic cleaning for the data (A.1), exploratory data analysis (A.2) and relevant pre-processing steps (A.3) which are required for building classification models in the following steps.

A.1. Basic Cleaning

There are two datasets including training set and scoring one. Appendix 1 lists all the names of the columns in the training set with 79 columns; meanwhile, with 77 columns, the scoring set has all of the training set's columns but Eng_Class (label variable) or Outcome.

The basic data cleaning steps are listed below:

- Outcome is removed since it exhibits the same pattern of values with Eng_Class.
- Antennafilename1, 2 are deleted as required.
- RFDBiD, which is the unique ID number for each radio mast, is removed since there is no duplication in both datasets.
- The five columns, DispersivefademargindB1, DispersivefademargindB2, MiscellaneouslossdB1, MiscellaneouslossdB2 & Passivegain2dB, were originally classified as characters. After examination, those columns, which contain only numeric values, are converted into numeric columns for both datasets.
- For columns whose values are characters, the white space in values' characters are changed into underscore "_". For example, in the column Antennamodel1, the value "VHLP4-13 7049B (TR)" is converted into "VHLP4-13_7049B_(TR)". This step is to enhance clarity for naming dummy variables in the following step.

- All of the columns whose values are characters are converted into factors for both datasets.
- Checking missing values, for training set, there are 9 missing values in column DqQ_R2 and 6 missing values in Fullmint1; for scoring set, there is no missing value. Since those missing values account for small fraction of the training set of 2186 rows, observations associated with such missing values are removed.

After basic cleaning, the preliminary summary of the training set and scoring one is provided in the Table 1.

	Training set	**Scoring set**
No. Rows	2173	936
Complete Rows	2173	936
No. Columns	75	74
No. Numeric Columns	63	63
No. Factor Columns	12	11

Table 1. Dataset summary

A.2. Exploratory Data Analysis

This section is to implement exploratory data analysis only on the training set without considering the scoring set. The assumption here is the unbiased random sampling during the data collection process for both datasets. Further investigation might be needed to check whether there is a similarity or difference for pair-wise variables on both datasets.

A.2.1. Univariate Analysis

A.2.1.1. Label variable Eng_Class

The bar plot in Figure 1 summarises 2-class label variables including "okay" with 1901 observations and "under" with 272 observations. To a certain extent, it exhibits the class imbalance in which "okay" and "under" accounts for 87.48% and 12.52%

respectively. Accordingly, dealing with the class imbalance will be implemented when building classification models in the following step.

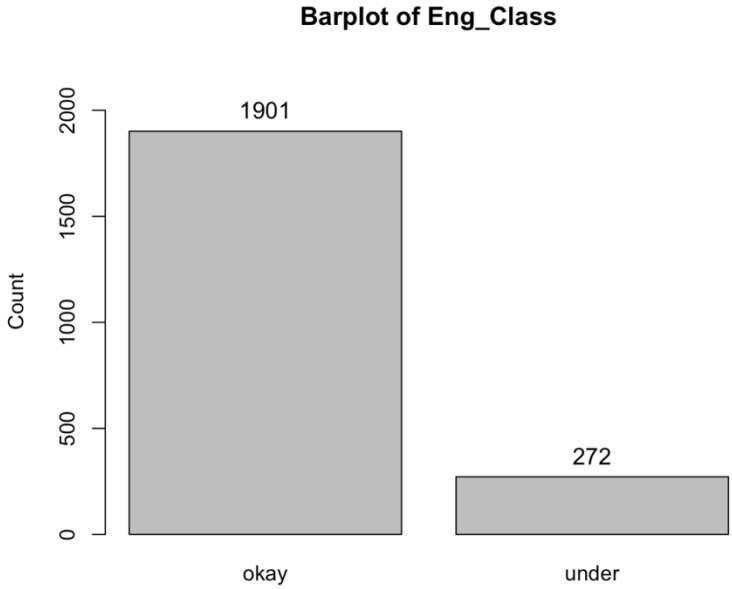


Figure 1: Bar Plot of Eng_Class

A.2.1.2. Numeric Variables

Figure 2 demonstrates the distributions of each numeric variables in the histograms. The distributions express different modalities including unimodal, multimodal and uniform distributions. In terms of skewness, the majority of variables have positively skewed distributions, some of the variables have negatively skewed distribution and a few seem to exhibit normal distributions. Besides, Numeric variables do not share a common scale. Due to the varied differences in distributions and scales, the numeric variables will be processed to achieve a commonality among those variables so that classification models could benefit from this feature.

A.2.1.3. Categorical Variables

Figure 3 demonstrates the distribution of levels/classes for each of categorical variables. Some of variables possess a large number of classes such as Antennamodel1 (50 classes), Antennamodel2 (49 classes), Radiofilename1 (88 classes), Radiofilename2 (88 classes), Radiomodel1 (152 classes) & Radiomodel2 (152 classes). The categorical variables will be processed to create corresponding dummy variables in the following step.

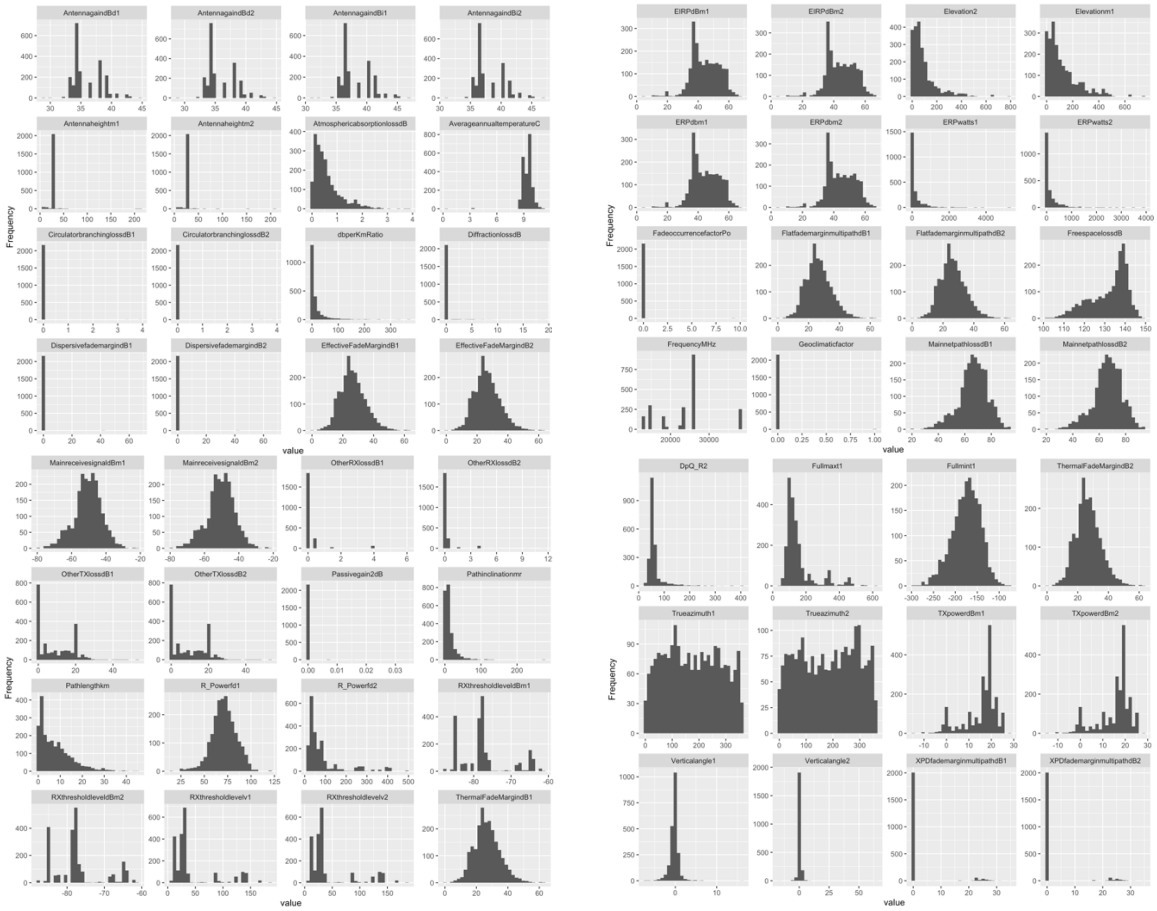


Figure 2: Histogram of Numeric Variables

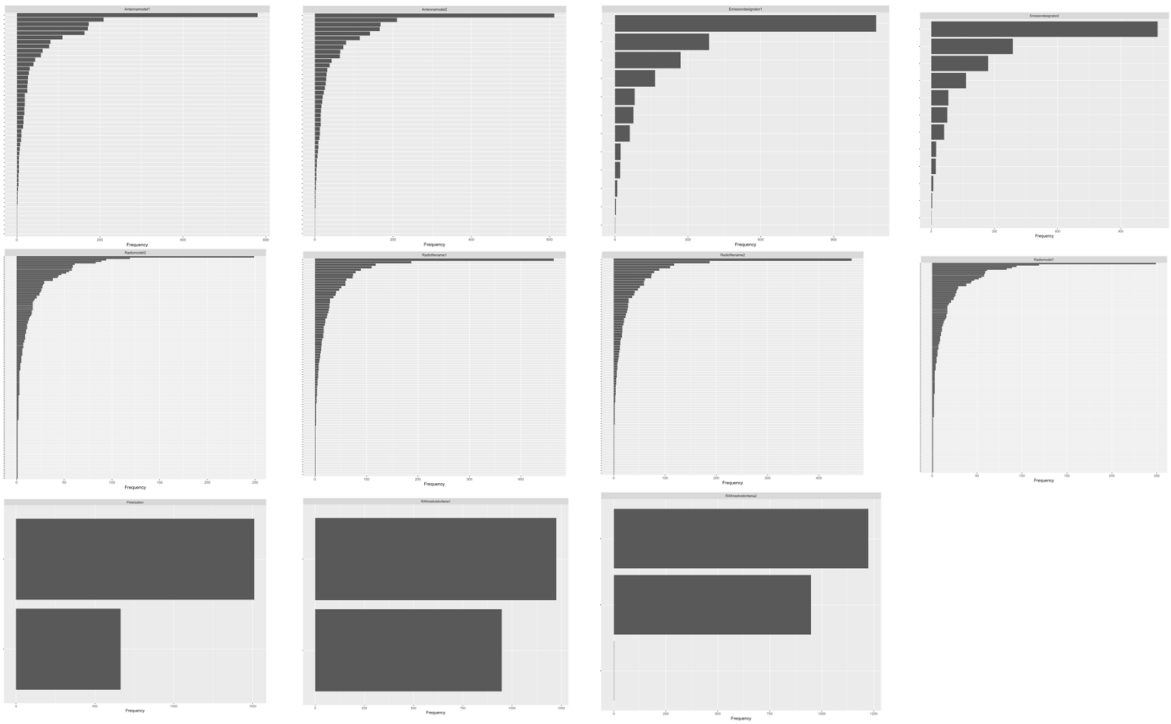


Figure 3: Bar Plot of Categorical Variables (without classes' names)

A.2.2. Bivariate Analysis

A.2.2.1. Between numeric variables

Figure 4 shows the Pearson's correlations between numeric variables. Most of the variables show no or weak correlations with others. Meanwhile, there are some highly correlated pairwise variables in dark blue and dark red. The multicollinearity is thus present in the dataset. Removing variables causing multicollinearity will be implemented in the following step.

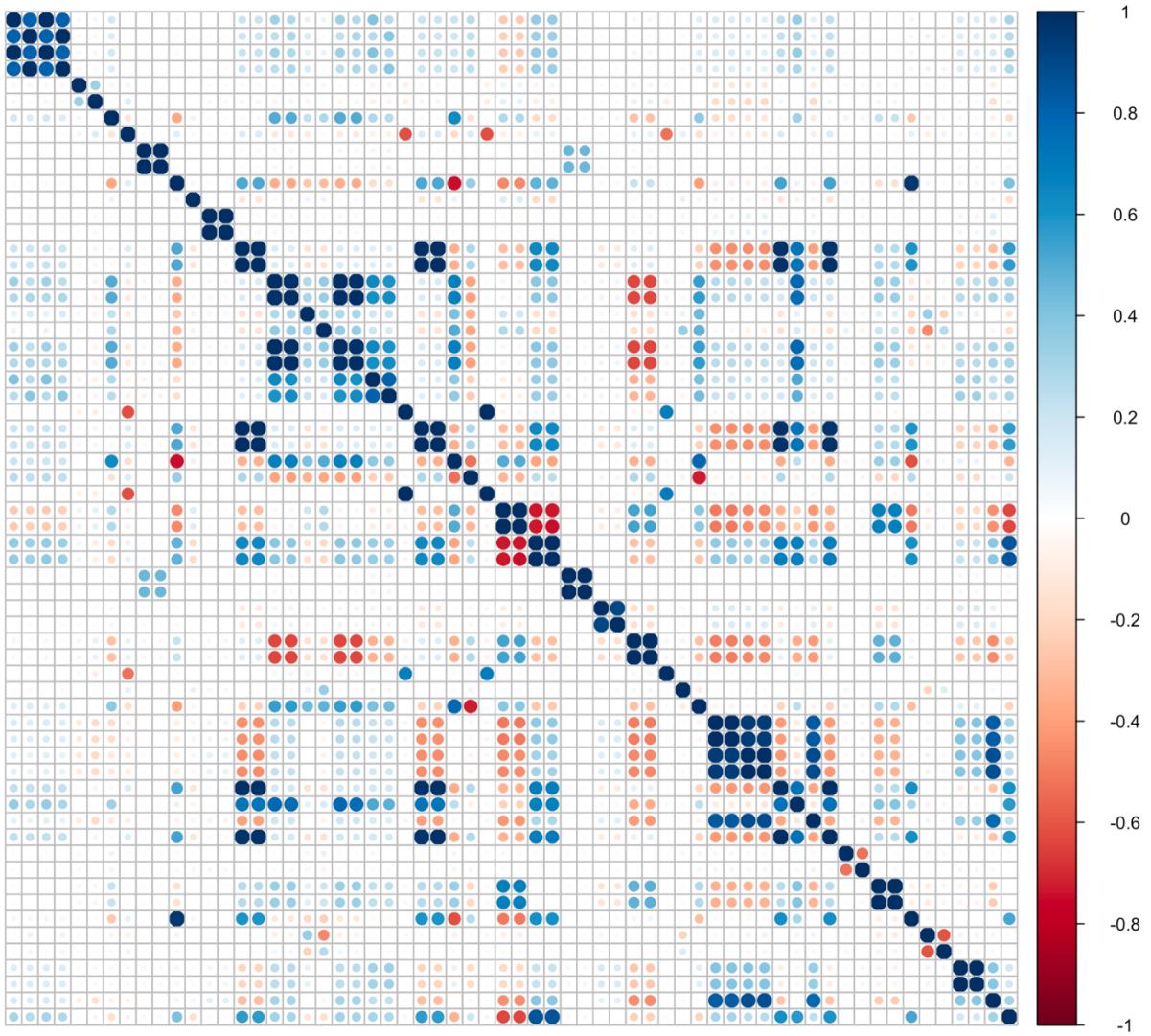


Figure 4: Pearson's Correlation among 63 numeric variables

A.2.2.1. Between categorical variables

Using chi-square test to identify any association between categorical variables, the bigger values are shown in Figure 5, the stronger associations between categorical variables are. With regards to label variable Eng_Class, there are strong associations with Radimodel1 and Radiomodel2. Emissiondesignator1 & Emissiondesignator2 also have a moderate association with Eng_Class.

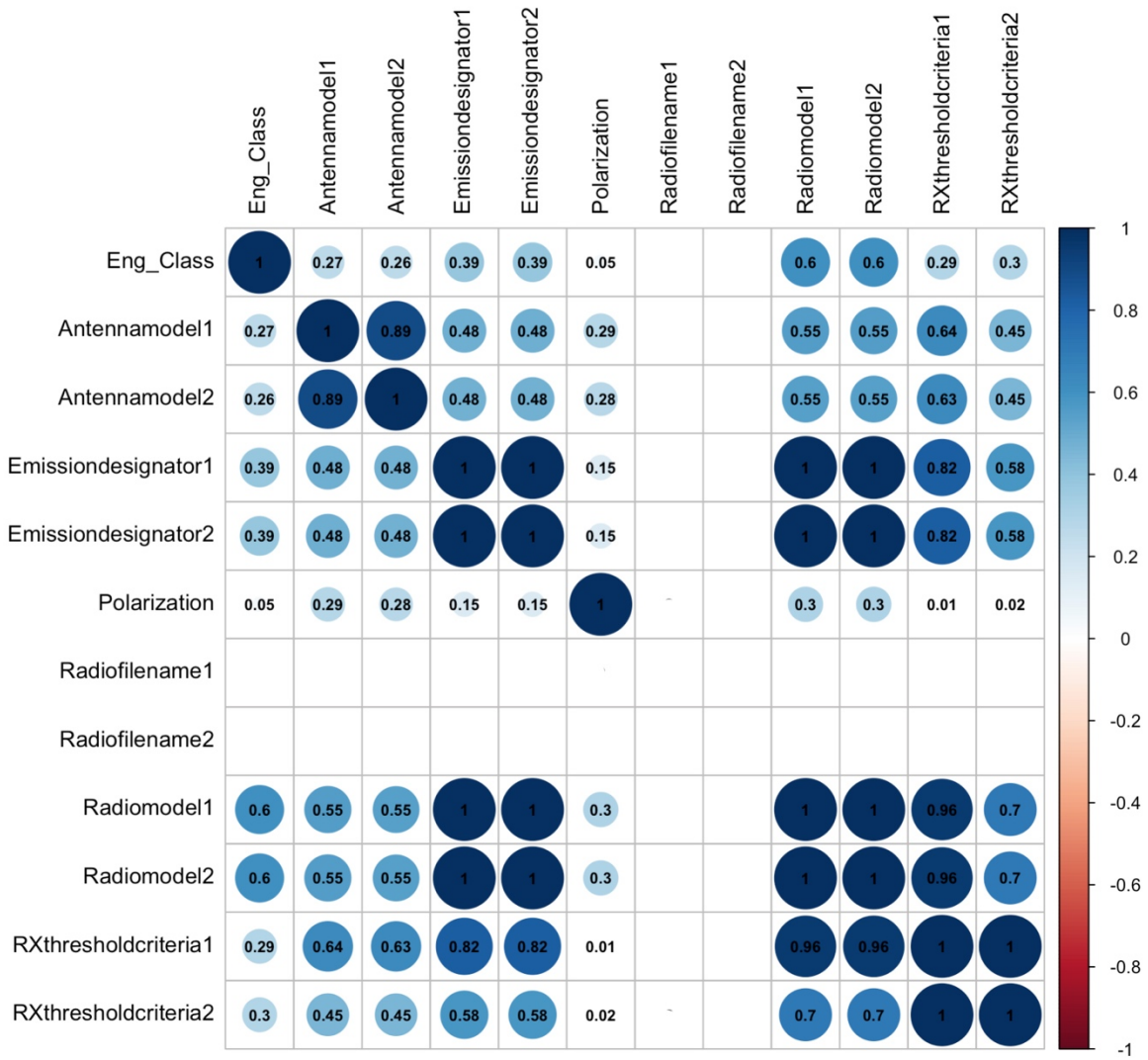


Figure 5: Chi-squared coefficients among 12 categorical variables

A.2.2.3. Between Eng_Class with numeric variables

Using analysis of variance (ANOVA) to identify any variation between groups in a numeric variable, i.e. the relationship between a numeric variable and a categorical variable, the bigger the intraclass coefficients are shown Figure 6, the stronger relationships between categorical variable Eng_Class and numeric variables are. According to Koo & Li (2016), the relationship is:

- Poor if the intraclass coefficient is below 0.5
- Moderate if the intraclass coefficient is between 0.5 & 0.75
- Good if the intraclass coefficient is between 0.75 & 0.9
- Excellent if the intraclass coefficient is above 0.9

In this case, Eng_Class has moderate relationships with 06 numeric variables including ThermalFadeMargindB1, ThermalFadeMargindB2, EffectiveFadeMarginB1, FlatfademarginmultipathB1, EffectiveFadeMarginB2 & FlatfademarginmultipathB2.

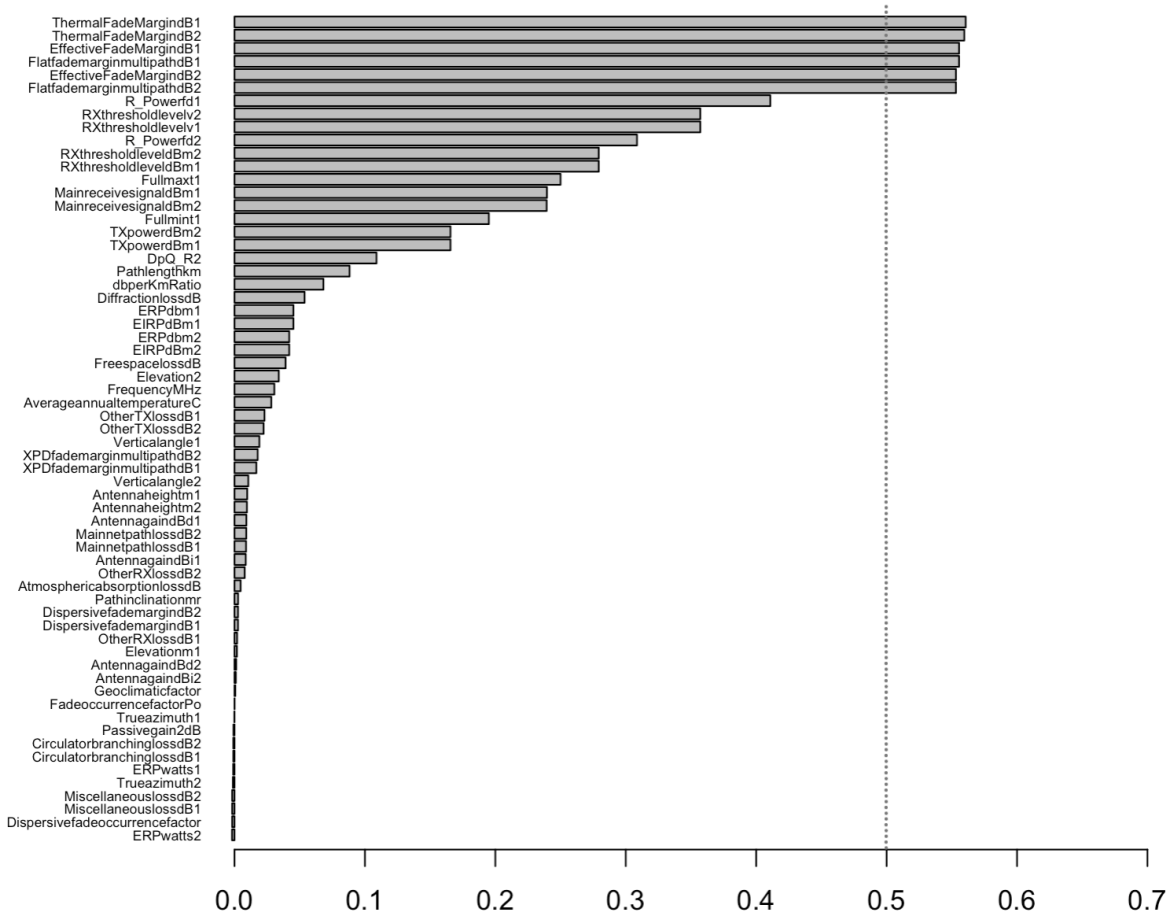


Figure 6: Intraclass correlation coefficients between Eng_Class with 63 numeric variables

A.3. Pre-processing Data

A.3.1. Creating dummy variables

```
library(caret)
# df1 (training set)
df1_dummies <- dummyVars(Eng_Class ~., data = df1)
df1_dummies <- predict(df1_dummies, newdata = df1)
df1_dummies <- as.data.frame(df1_dummies)
dim(df1_dummies)
df1_dummies$Eng_Class <- df1$Eng_Class
dim(df1_dummies)
# df2 (scoring set)
df2$Eng_Class <- c(0,1)                       # fake column
df2_dummies <- dummyVars(Eng_Class~., data = df2)
df2_dummies <- predict(df2_dummies, newdata = df2)
df2_dummies <- as.data.frame(df2_dummies)
dim(df2_dummies)
```

All the categorical variables apart from Eng_Class are turned into dummy variables in both training set and scoring set.

A.3.2. Removing zero & near-zero variance variables

```
nzv <- nearZeroVar(df1_dummies)
df1_nzv <- df1_dummies[,-nzv]
dim(df1_nzv)
str(df1_nzv)
```

This step is to remove zero and near-zero variance variables which are likely to be redundant for classification models.

A.3.3. Fixing multicollinearity

```
cor_matrix <- cor(df1_nzv[, -c(89)])    # removing Eng_Class from the correlation matrix
hist(cor_matrix[upper.tri(cor_matrix)])
# visualising different cutoff with no. variables deleted
x <- seq(0,1, by=0.001)
y <- c()
for(i in x){
  y <- c(y, length(findCorrelation(cor_matrix, cutoff = i)))      # this takes 20 seconds
}
cut = 0.982
plot(x,y, type = 'l')
abline(v = cut, col = 'red')
abline(h = length(findCorrelation(cor_matrix, cutoff = cut)), col = 'red')
# filtering highly correlated variables
highly_cor_vars <- findCorrelation(cor_matrix, cutoff = cut)
df1_fil_cor <- df1_nzv[,-highly_cor_vars]
dim(df1_nzv)
dim(df1_fil_cor)
```

This step is to find highly-correlated variables, then deleting them to fix multicollinearity problem. Figure 7 shows that how the choice of cut-off decides the number of deleted variables. The plot in Figure 7 reflects a gradual decrease of the number

of deleted variables as the cut-off level increases. Interestingly, at the cut-off 0.982, there is a sudden decrease by 32 deleted variables. Although there is no clear guide of how to choose a cut-off level to effectively fix multicollinearity problem, the cut-off 0.982 is chosen to avoid the loss of necessary variables but also to remove the abnormal group of high-correlated variables. Later, PCA is applied to create non-correlated variables which are then used for building classification models.

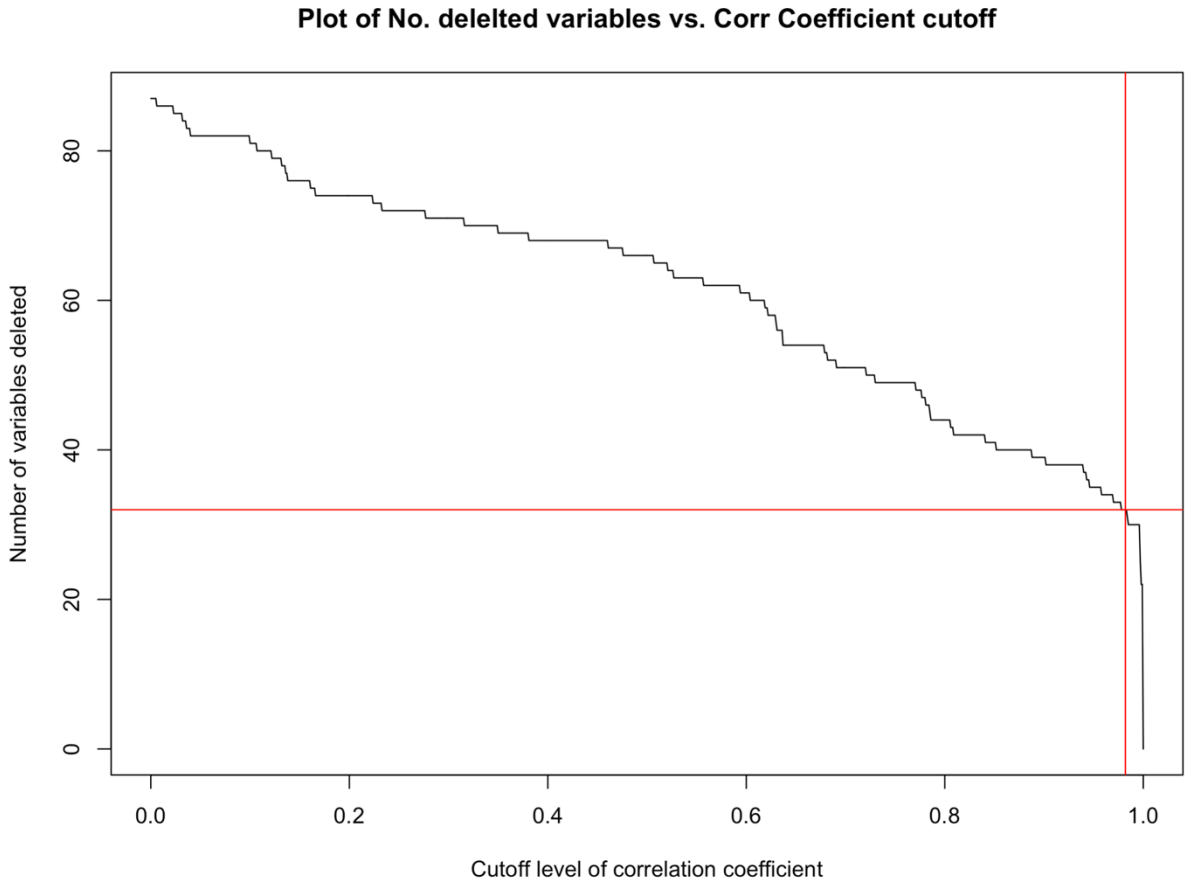


Figure 7: Plot of No. Deleted Variables vs. Choice of Correlation Coefficient Cut-off

A.3.4. Fixing linear dependencies

```
272 comboInfo <- findLinearCombos(df1_fil_cor[,-c(57)])    # removing Eng_Class from the dataset
273 comboInfo
```

Up until this point, there is no variables causing linear dependencies in the dataset.

A.3.5. Subsetting relevant columns for scoring set

```
needed_col1 <- colnames(df1_fil_cor[,-c(57)])    # removing Eng_Class from training set
length(needed_col1)
needed_col2 = c()
for(i in 1:ncol(df2_dummies)){
  if(colnames(df2_dummies[i]) %in% needed_col1){
    needed_col2 = c(needed_col2, i)
  }
}
df2_fil_cor <- df2_dummies[,needed_col2]
dim(df2_fil_cor)
```

This step is to select columns for the scoring set such that columns in the scoring set have to be similar to the columns in the training set.

A.3.6. Scaling dataset

```
preProcValues <- preProcess(df1_fil_cor[,-c(57)],
                            method = c("center", "scale", "BoxCox", "pca"))
df1_trans <- predict(preProcValues, df1_fil_cor[,-c(57)])
df1_trans$Eng_Class <- df1_fil_cor$Eng_Class
df2_trans <- predict(preProcValues, df2_fil_cor)
```

This step is to scale both datasets. The methods are:

- "center": subtract mean from values
- "scale": divide value by standard deviation
- "BoxCox": transform non-normal variables into a normal shape
- "pca": transform data into principal components

This step is finally creating a (training) dataframe with 29 columns of which 28 columns/variables are principal components that are uncorrelated with each other. In another word, the set of the predicting variables in the training set is a 28-dimensional space. Table 2 shows how the datasets are transformed after each of the pre-processing steps.

A.3.7. Splitting and subsampling the dataset

Due to the class imbalance, the training set is subsampled to create more class-balanced dataset so that classification models will not fall into the trap of falsely achieving high accuracy. However, the subsampling process is only applied for classification models' training set so that classification models' test set, which might be as class-imbalanced as the scoring set, can be accurately evaluated. This objective can be only achieved by the unbiased random sampling during the data collection process for both the training set and

scoring set as our underlying assumption (Section A.2). Additionally, a validation set, which is used to select optimal classification models' cut-off levels, is not subsampled either. The validation set's class imbalance will somehow mirror the test set's and scoring set's. Figure 8 shows how splitting and subsampling the training set is implemented.

The subsampling SMOTE method is used in this case.

	Training set	**Scoring set**
No. Rows	2173	936
No. Columns before pre-processing	75	74
No. Columns after making dummy variables (A.3.1)	674	574
No. Columns after removing near-zero variance variables (A.3.2)	89	
No. Columns after removing highly-correlated variables (A.3.3)	57	
No. Columns after fixing linear dependencies (A.3.4)	57	
Subsetting relevant columns for scoring data set (A.3.5)		56
Scaling data and applying principal components (A.3.6)	29	28

Table 2. Transforming data

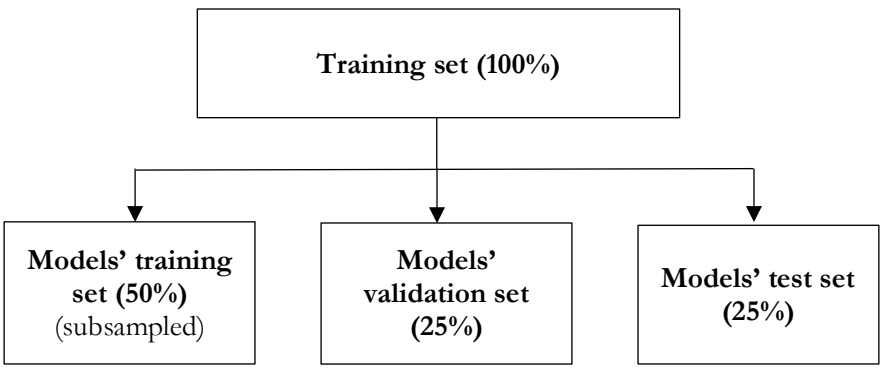


Figure 8. Training/Validation/Testing Splits

B. Building classification models

In this section, Random Forest (B.1) and Gradient Boosting Machine (B.2) are chosen to build classification models. The training set (subsampled) is used for training models and tuning hyperparameters implemented by grid searches supported by the caret package. The criterion of tuning hyperparameters is area under the curve (AUC) which reveals the models' power of classifying binary classes. Next, the effect of the class imbalance is evaluated on the validation set (non-subsampled) so that optimal cut-off levels are chosen to mitigate such effect and to illustrate how different cut-off levels can suggest different strategies for business owners for their future plans. Finally, the test set (non-subsampled) is used to evaluate models' accuracy, specificity, sensitivity and area under the curve (AUC).

After that, based on the models' performance, a suggestion regarding the best classification model will be made at the end.

B.1. Random Forest

B.1.1. Building model & Tuning hyperparameters on training set

The hyperparameter for random forest is mtry which is the number of variables randomly sampled as candidates at each split. Grid search suggests mtry = 6 which proposes the best random forest model in terms of AUC (Figure 9).

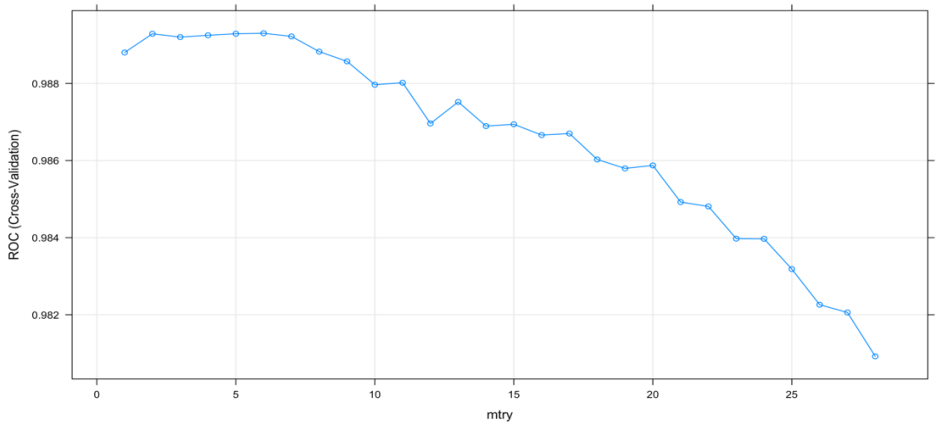


Figure 9. Random Forest's Grid Search

With the chosen mtry, the random forest model is built with the summary as follows

```
Random Forest

952 samples
 28 predictor
  2 classes: 'okay', 'under'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 762, 761, 761, 761, 763
Resampling results:

  ROC        Sens       Spec
  0.9895891  0.9466701  0.9338452

Tuning parameter 'mtry' was held constant at a value of 6
```

Variable (principal component) importance is summarised as follows

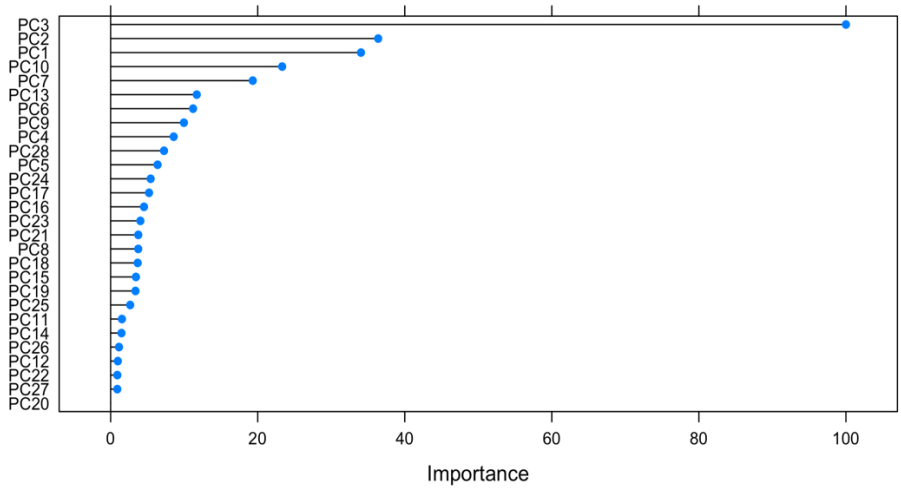


Figure 10. Variable Importance (Random Forest)

B.1.2. Examining Class Imbalance & Choosing Cutoff on validation set

Firstly, examining class probabilities predicted by the model, Figure 11 shows that the probabilities are polarised to two ends in which the model is more "confident" in predicting classes when probabilities are close to zero or one as predicted as under or okay respectively. Calibrating predicted probabilities, which is not within the scope of this project, could be taken into consideration for future task. Figure 12 reveals how different cut-off levels result in varied results of accuracy, sensitivity and specificity. The class imbalance in the validation set is exposed clearly by the plot showing how accuracy is

consistently high for big values of cut-offs. Even the model misclassifies all of the minority class, i.e. under, the accuracy is even at 83%. As the requirement of the business owner Daniel for correctly classifying the class under, considering specificity, which is the true negative rate, is a must. At the cut-off 0.35, the model is able to balance accuracy, sensitivity and specificity as shown in the plot. To this point, the cut-off 0.35 is selected for the model. However, changing the cut-off will be necessary if the business owner Daniel specifies particular levels of accuracy, sensitivity or specificity.

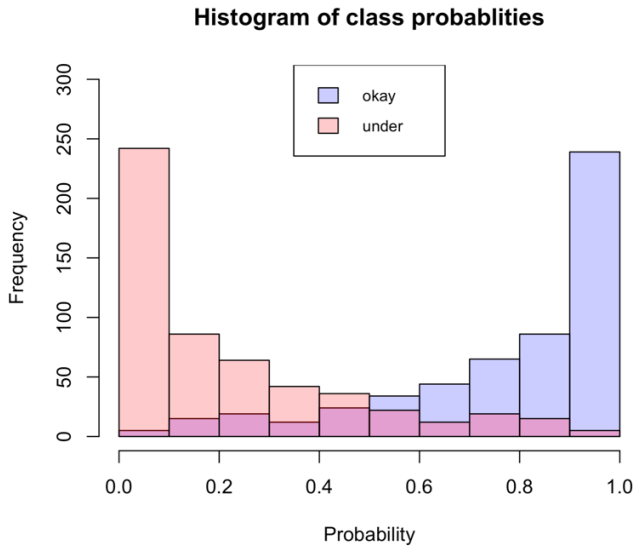


Figure 11. Class Probabilities (Random Forest)

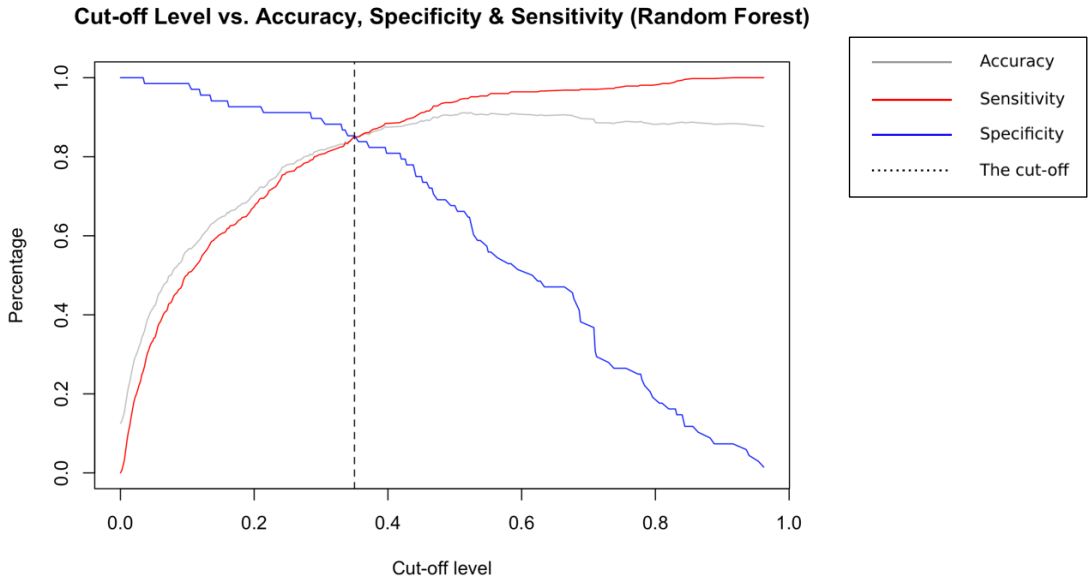


Figure 12. Cut-offs vs. Accuracy, Specificity & Sensitivity (Random Forest)

B.1.3. Evaluating model on test set

Table 3 summarises key metrics evaluating the model's performance. Area Under the Curve (AUC) is the fixed value regardless of different values of cut-offs. This value indicates the power of classifying/distinguishing under with okay. At the cut-off 0.35, the model yields the following Accuracy, Specificity and Sensitivity. A comparison between random forest and gradient boosting machine will be made in the following section.

	AUC	Accuracy	Specificity	Sensitivity
Random Forest	0.9362	0.8566	0.8824	0.8529

Table 3. Random Forest Performance (Test set)

B.2. Gradient Boosting Machine (GBM)

B.2.1. Building model & Tuning hyperparameters on training set

The hyperparameters for GBM are as follows:

- n.trees: Number of trees (the number of gradient boosting iteration) i.e. N. Increasing N reduces the error on training set, but setting it too high may lead to over-fitting.
- interaction.depth (Maximum nodes per tree) - number of splits it has to perform on a tree (starting from a single node).
- shrinkage (Learning Rate) – It is considered as a learning rate.
- n.minobsinnode - the minimum number of observations in trees' terminal nodes.

The grid search suggests the best hyperparameters as follows: n.trees = 200, interaction.depth = 10, shrinkage = 0.1, n.minobsinnode = 8 (Figure 10).

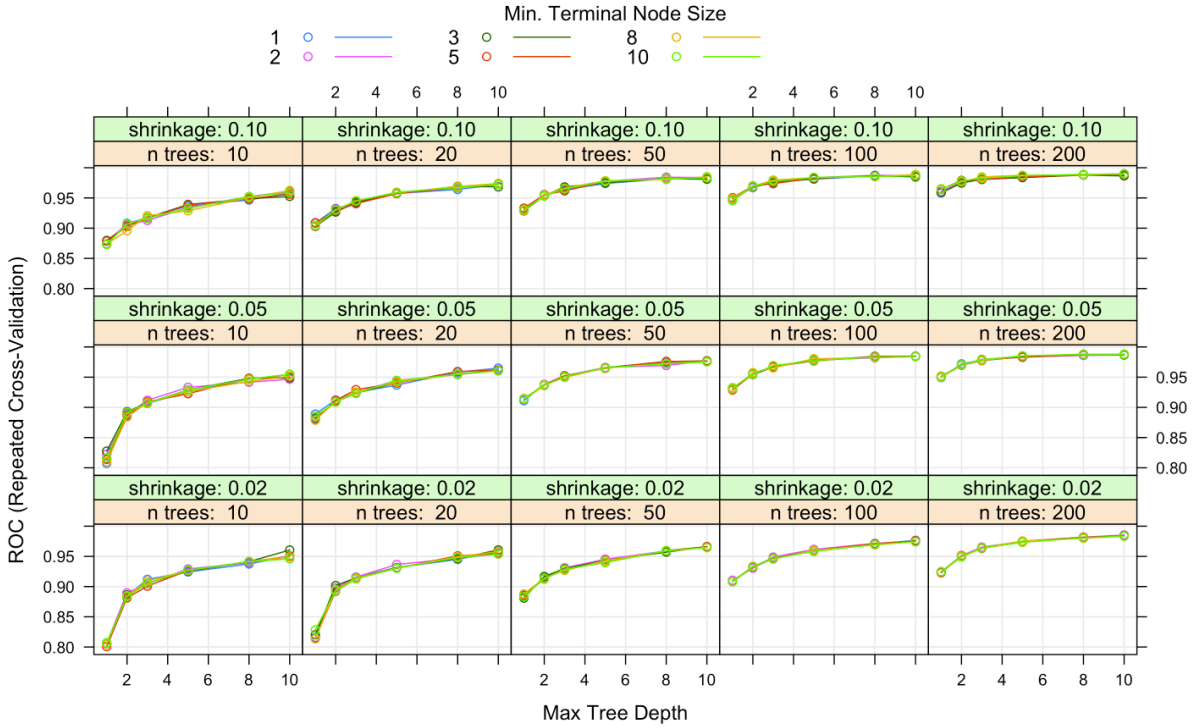


Figure 13. GBM's Grid Search

With the chosen set of hyperparameters, the GBM model is built with the summary as follows:

```
Stochastic Gradient Boosting

952 samples
 28 predictor
  2 classes: 'okay', 'under'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 5 times)
Summary of sample sizes: 762, 761, 761, 761, 763, 762, ...
Resampling results:

  ROC        Sens       Spec
  0.9897727  0.9414917  0.9593436

Tuning parameter 'n.trees' was held constant at a value of 200
Tuning
 parameter 'shrinkage' was held constant at a value of 0.1
Tuning parameter
 'n.minobsinnode' was held constant at a value of 8
```

Variable (principal component) importance is summarised as follows

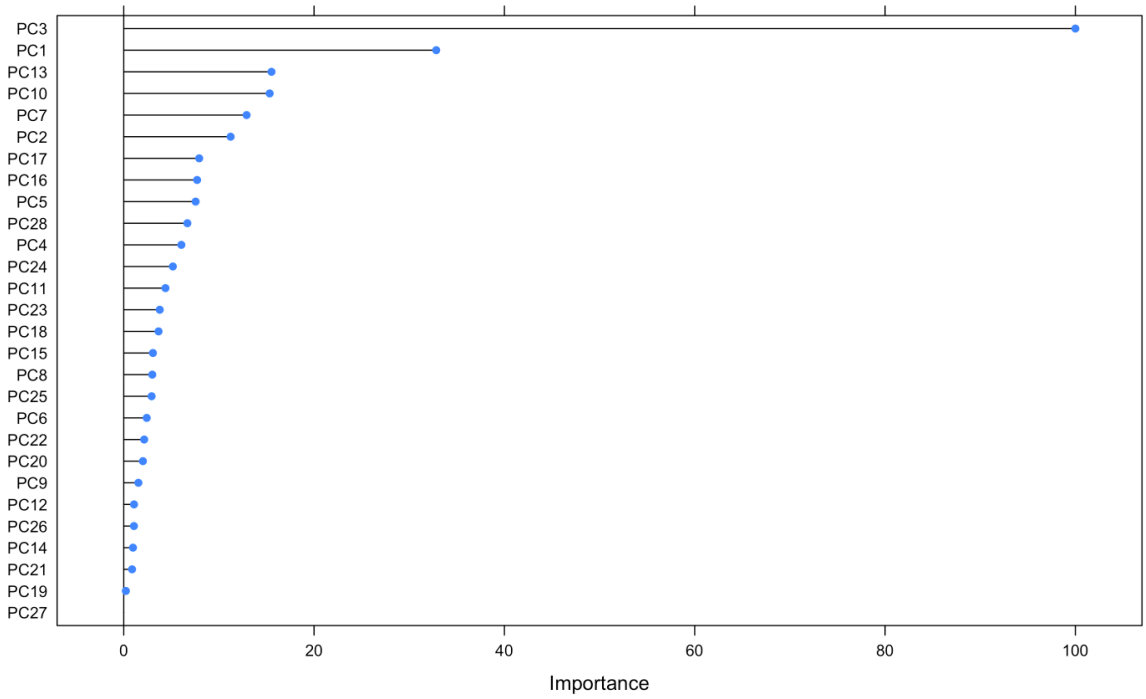


Figure 14. Variable (Principal Component) Importance

B.1.2. Examining Class Imbalance & Choosing Cutoff on validation set

Firstly, examining class probabilities predicted by the model, Figure 15 shows that the probabilities are extremely polarised to two ends in which the model is quite "confident" in predicting classes when probabilities are close to zero or one as predicted as under or okay respectively. The polarised predicted probabilities also clearly affect how accuracy, sensitivity and specificity are determined by cut-offs. Towards two ends, such metrics change significantly as cut-offs change. Therefore, calibrating predicted probabilities, which is not within the scope of this project, could be taken into consideration for future task.

Figure 16 reveals how different cut-off levels result in varied results of accuracy, sensitivity and specificity. Similarly, the class imbalance in the validation set is exposed clearly by the plot showing how accuracy is consistently high for big values of cut-offs. Even the model misclassifies all of the minority class, i.e. under, the accuracy is even at 83%. As the requirement of the business owner Daniel for correctly classifying the class

under, considering specificity, which is the true negative rate, is a must. At the cut-off 0.145, the model is able to balance accuracy, sensitivity and specificity as shown in the plot. To this point, the cut-off 0.145 is selected for the model. However, changing the cut-off will be necessary if the business owner Daniel specifies particular levels of accuracy, sensitivity or specificity.

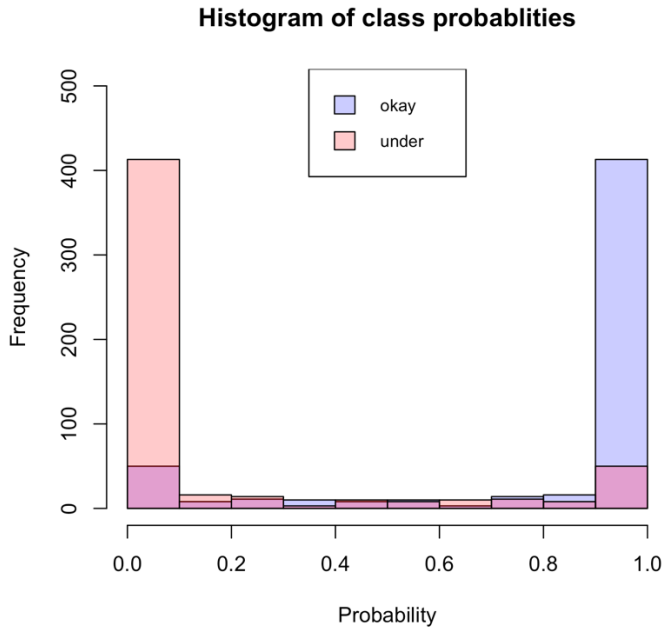


Figure 15. Class Probabilities (GBM)

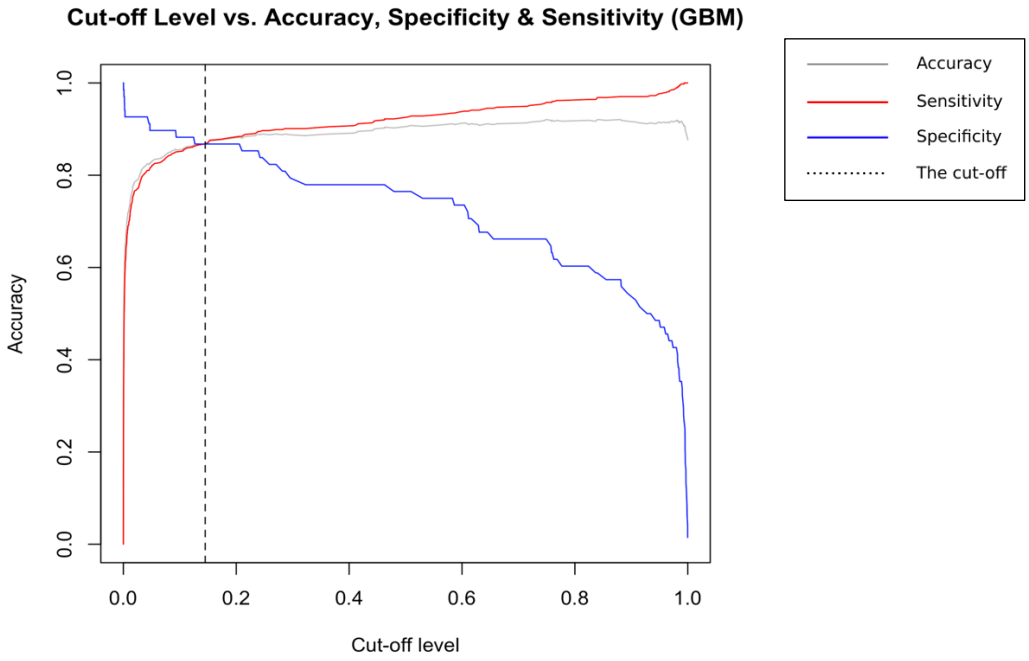


Figure 16. Cut-offs vs. Accuracy, Specificity & Sensitivity (GBM)

B.1.3. Evaluating model on test set

Table 4 summarises key metrics evaluating the model's performance. Area Under the Curve (AUC) is the fixed value regardless of different values of cut-offs. This value indicates the power of classifying/distinguishing under with okay. At the cut-off 0.145, the model yields the following Accuracy, Specificity and Sensitivity. A comparison between random forest and gradient boosting machine will be made in the following section.

	AUC	Accuracy	Specificity	Sensitivity
GBM	0.9445	0.8805	0.8824	0.8803

Table 4. GBM Performance (Test set)

B.3. The best model

Table 5 shows a comparison between Random Forest and GBM models. Most of the metrics suggest GBM as the better model. Interestingly, at the cut-off 0.35 for Random Forest and the cut-off 0.145 for GBM, the specificity values are the same for both models. Besides, Figure 17 also shows the bigger area under the curve for GBM. Taken together, the chosen model is GBM.

	AUC	Accuracy	Specificity	Sensitivity
Random Forest	0.9362	0.8566	0.8824	0.8529
GBM	0.9445	0.8805	0.8824	0.8803

Table 5. Random Forest & GBM Performance (Test set)

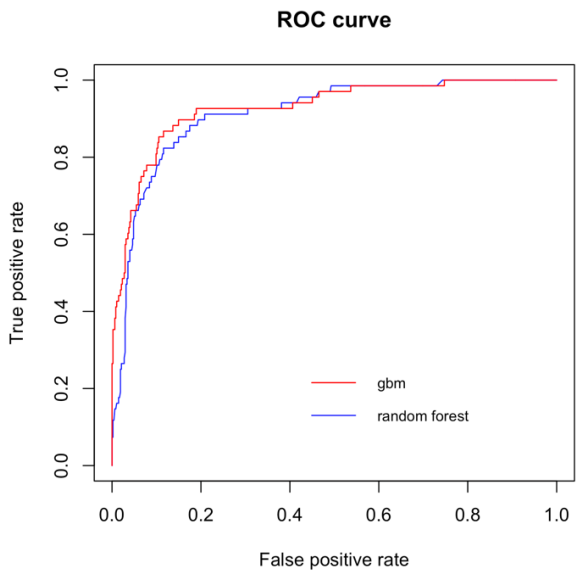


Figure 17. ROC curve

C. Feature Selection

This section is to perform feature selection for the chosen model GBM from the training data which was already pre-processed in section A.3. The objective is to find a subset of variables contributing most to the GBM model by deleting irrelevant features. which might decrease the model's performance. The training data has 952 observations/rows, 1 label column Eng_Class and 28 features/columns namely PC1, PC2,…, PC28 as the original dataset was PCA transformed. The feature selection methods are recursive feature elimination in caret package (C.1) and Boruta feature selection in Boruta package (C.2). Both are wrapper methods which involve adding and/or removing features to find the optimal combination that maximises model performance.

C.1. Recursive Feature Elimination

Recursive feature elimination a backward selection method by building a model on the entire set of features and computing an importance score for each feature. The least important feature(s) are then removed, the model is re-built, and importance scores are computed again. The caret package is used to perform feature selection. The code is shown below.

```
fiveStats <- function(...) c(twoClassSummary(...), defaultSummary(...))
ctrl <- rfeControl(method = "repeatedcv",
                   repeats = 5,
                   saveDetails = TRUE,
                   functions = caretFuncs,
                   returnResamp = "final")
ctrl$functions$summary <- fiveStats
cvCtrl <- trainControl(method = "cv",
                       verboseIter = FALSE,
                       classProbs = TRUE,
                       allowParallel = TRUE)
set.seed(696)
gbmRFE <- rfe(Eng_Class ~., data = training,
              sizes = 1:28,
              rfeControl= ctrl,
              metric = "ROC",
              method = "gbm",
              tuneLength = 10,
              trControl = cvCtrl)
gbmRFE
```

However, execution took very long time. Due to the time constraint of this project, the code was not completely executed. It prompts future revisiting of the code.

C.2. Boruta Feature Selection

Boruta Feature Selection is a wrapper algorithm built around the random forest classification algorithm which is relatively quick by running without tuning of parameters. How Boruta works is as follows:

- Firstly, it adds randomness to the given data set by creating shuffled copies of all features (which are called shadow features).
- Then, it trains a random forest classifier on the extended data set and applies a feature importance measure to evaluate the importance of each feature where higher means more important.
- At every iteration, it checks whether a real feature has a higher importance than the best of its shadow features (i.e. whether the feature has a higher Z score than the maximum Z score of its shadow features) and constantly removes features which are deemed highly unimportant.
- Finally, the algorithm stops either when all features gets confirmed or rejected or it reaches a specified limit of random forest runs.

Figure 18 and Figure 19 shows the result of Boruta feature selection on the training set in which all of the features acquired higher importance scores than the shadow variables. It means that the GBM model was already built on the training set having important features that further feature elimination is not necessary.

Besides, the result of Boruta feature selection provokes speculations on the pre-processing step A.3. Possible scenarios are as follows:

- The pre-processing might have done perfectly that only important features are retained.
- The pre-processing might have "overkilled" other important features which are not presented in this step.
- PCA transformation might have combined unimportant features into the important ones to create principle components that are now essential to the model.

Therefore, the pre-processing step might be revisited in the future so that models could be built on better datasets. Finally, the result of Boruta Feature Selection suggests that the GBM model built in the previous step will be used for making predictions on the scoring set in section F.

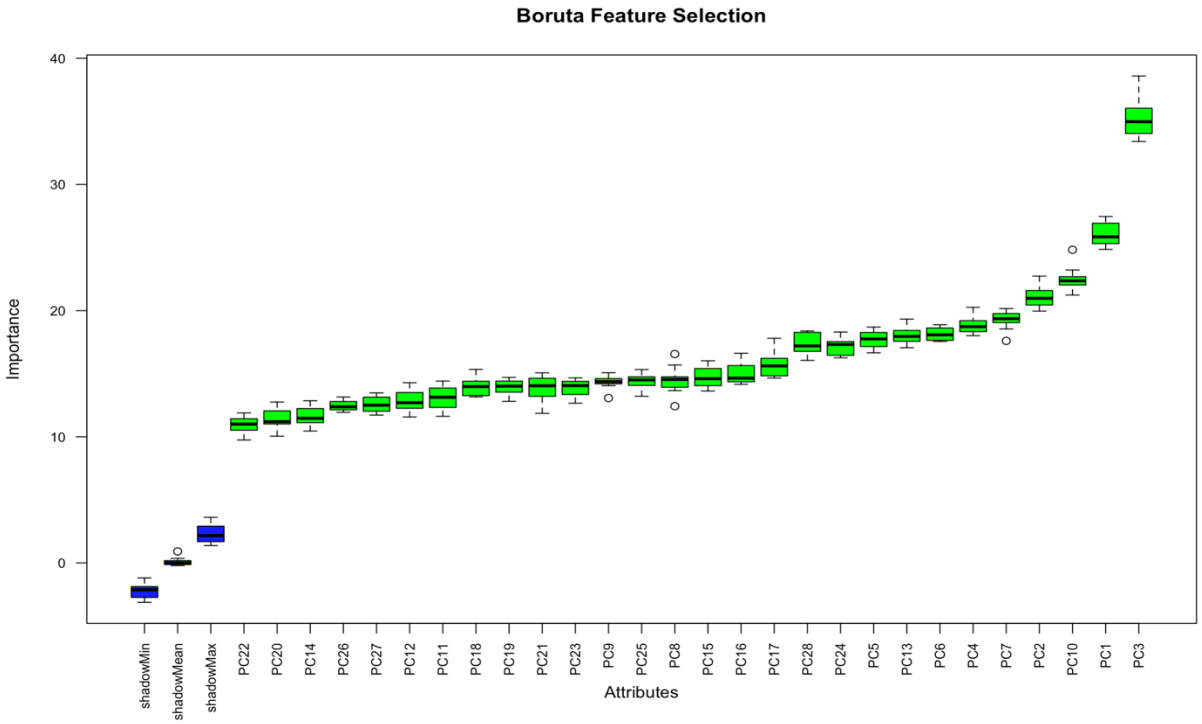


Figure 18. Boruta Feature Selection

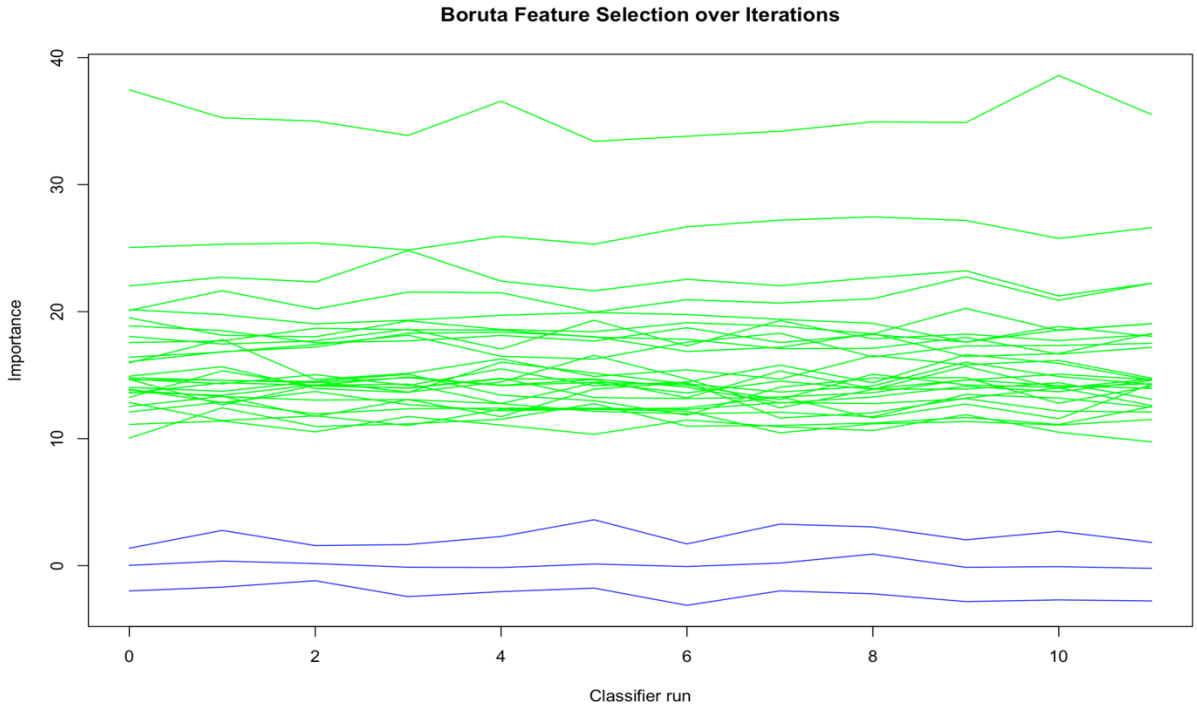


Figure 19. Boruta Feature Selection over Iterations

D. How GBM works

This section is to provide the business owner Daniel a simple explanation of how GBM works. The purpose is to help Daniel or someone without relevant knowledge of machine learning adequately understand the algorithm in the simplest sense. Thus, this section is constructed as follows:

D.1. Supervised Machine Learning

Machine Learning is the modern statistical technique, being developed from the traditional statistical field and considerably benefited from the advancement of computational power. Machine Learning involves making statistical inference from existing data, a process called training, in order to make decisions about new data. There are many types of Machine Learning algorithms in which practitioners have to reply on type of data they have and type of task or problem they are intended to solve to choose (an) appropriate Machine Learning algorithm(s).

With the given dataset and the task of binarily classifying whether a radio mast is under or okay, supervised machine learning algorithms, which try to predict the results of future radio masts from other relevant factors, are used. In our case, two supervised machine learning algorithms, i.e. Random Forest and GBM, were chosen to perform such task. In the simplest sense, how to single out and use the best algorithm depends on the algorithm's performance of correctly predicting the results. After experimenting and evaluating two algorithms' performance, GBM was finally chosen.

D.2. What is GBM?

Gradient Boosting Machine (GBM) is an ensemble method in which weak prediction models being built sequentially learn the mistakes of the previous prediction models. At first, a single weak prediction model is a decision tree (Figure 20). Number of layers and number of leaves (outcomes) in the decision tree are carefully chosen. The simplicity of such weak models conveniently leads to prediction errors which are focused by the subsequent models trying to minimise the errors or the loss function. Number of trees and how fast weak models minimise prediction errors are also thoroughly decided.

The final prediction is the weighted sum of the predictions made by all of the weak tree models (Figure 21) (He et al., 2019).

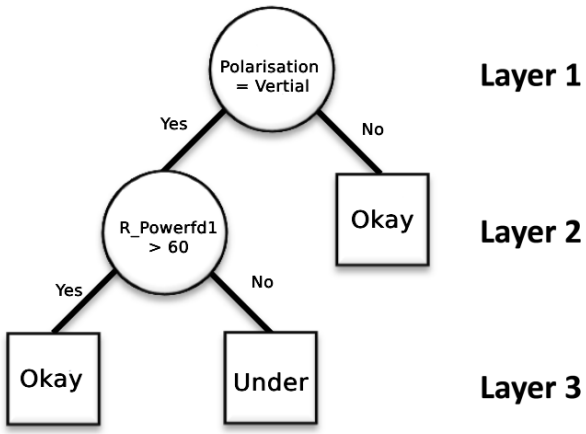


Figure 20 A single weak prediction model – Decision Tree

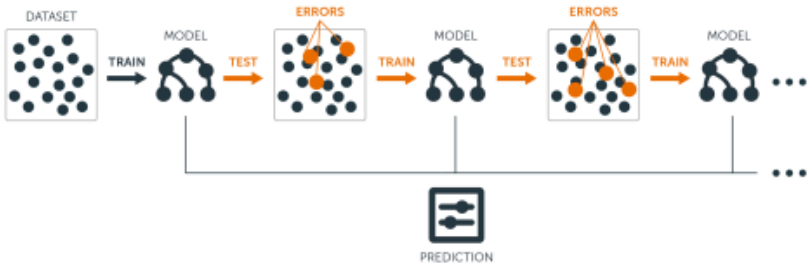


Figure 21. How GBM works

D.3. The Loss function

Loss functions are the very heart of the machine learning algorithms. A loss function can tell how decisions made by machine learning algorithms are associated with some costs. The main purpose is to reduce such costs. In our case of binary classification, the loss function incorporated with GBM is the cross entropy which measures the impurity or heterogeneity of the elements in a set. The cross entropy for binary classification is formulised as follows:

$$CE = -\frac{1}{M}\sum_{i=1}^{M}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

Where M is the number of observations in the training dataset, $\hat{p}$ is the probability of positive class okay with label $y = 1$. Simplifying the formula, we have:

$$CE = -\frac{1}{M}\sum_{i=1}^{M}[\log(\hat{p}_i)]$$

The GBM algorithm tries to minimise the loss function CE. The classification model that predicts perfect probabilities has a cross entropy of 0.0 (Khan, 2019).

E. Different scenarios for label variable

F. Predictions for scoring set

This section is to apply both GBM model and random forest one on the scoring data set of 936 observations.

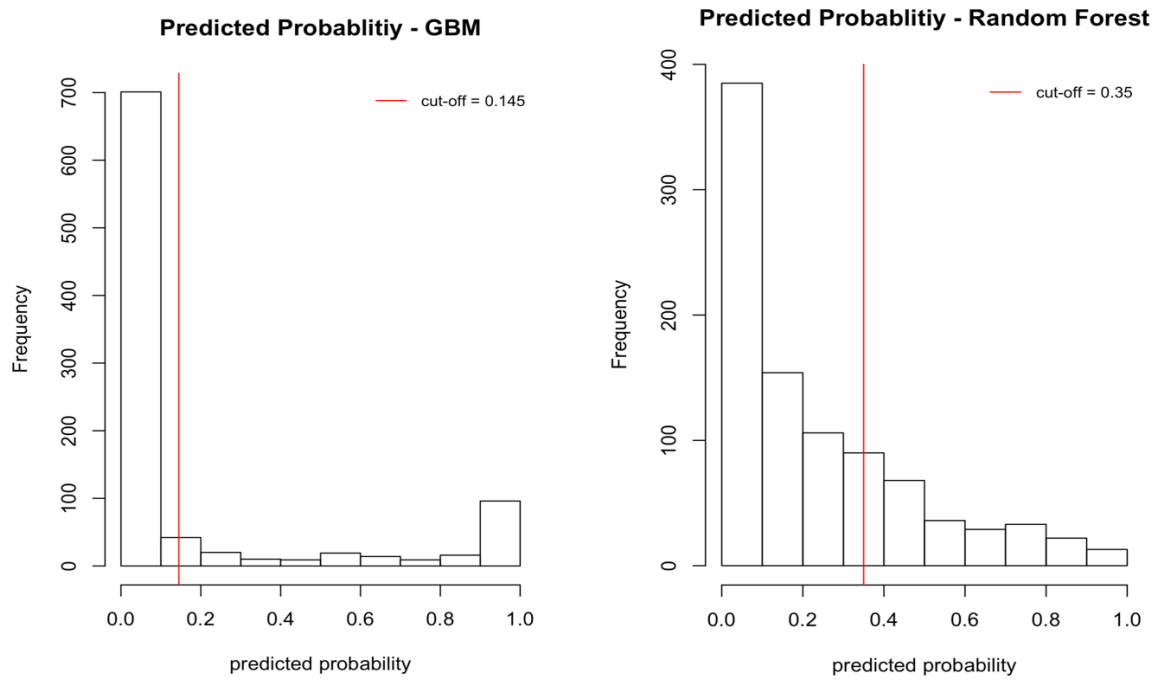


Figure 22. Predicted Probabilities

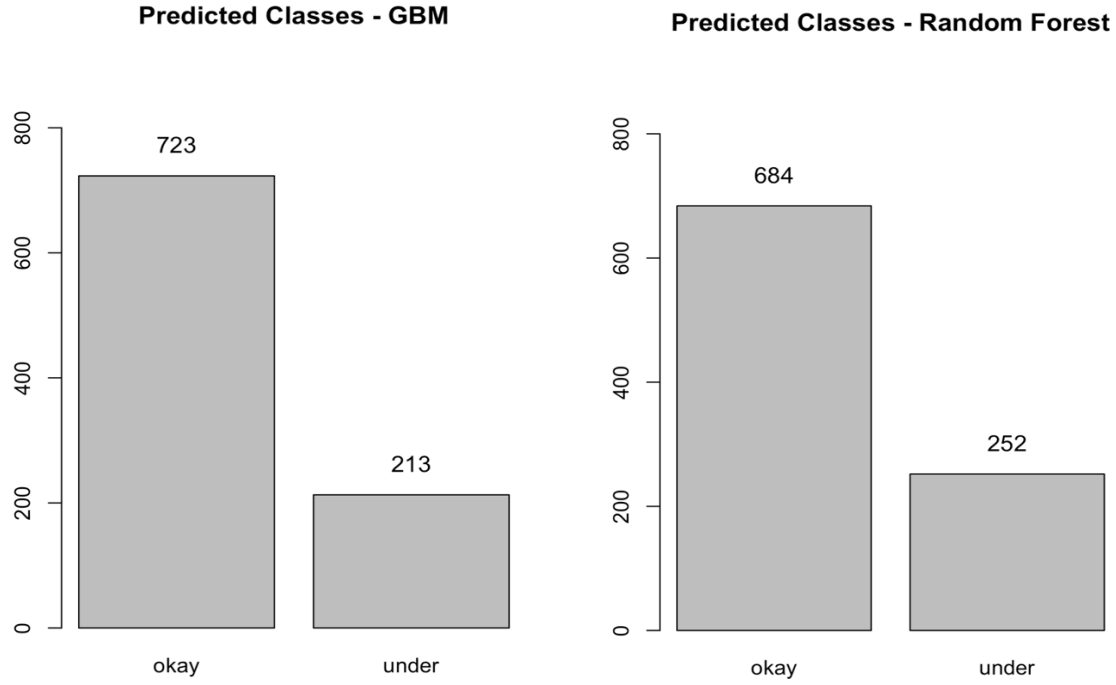


Figure 23. Predicted Classes

Figure 22 and Figure 23 shows the predicted probabilities and predicted classes that GBM and random forest models make. The class prediction of under in GBM model is 213 / 936 = 22.76%. The class prediction of under in Random Forest model is 252 / 936 = 26.92%. Both class predictions are higher than the under ratio in the training set with 12.52% (section A.2.1.1). It might suggest that both models' predictions are quite "conservative" toward the class under. This conclusion is only valid if and only if unbiased random sampling was conducted.

If the business owner Daniel wants to have "safe" predictions for class under, Random Forest model should be chosen. If Daniel wants to have more "balanced" predictions, GBM model will the choice.

Further changes will be made to improve models' performance.

G. Deep Learning Practices

In this section, the report will provide a full written explanation for the following questions:

(G.1.) How does convolution layers and max pooling actually work in CNNs? Give also a broad overview understanding of what both layers do in deep CNN for image classification.

(G.2.) Describe in detail a deep learning methodology that are used in time series analysis.

G.1. CNN for Image Classification

Convolutional Neural Network (CNN) is a feedforward neural network in which information flow takes place in one direction from an input layer to multiple hidden layers and finally an output layer. The hidden layers of a CNN commonly include convolutional layers, pooling layers and fully connected layers. Notably, convolutional and pooling layers are the most important features distinguishing CNN from other neural networks (Stanford University, 2020). This report will provide an overview of how convolutional layers and max pooling work in a CNN in the context of image classification.

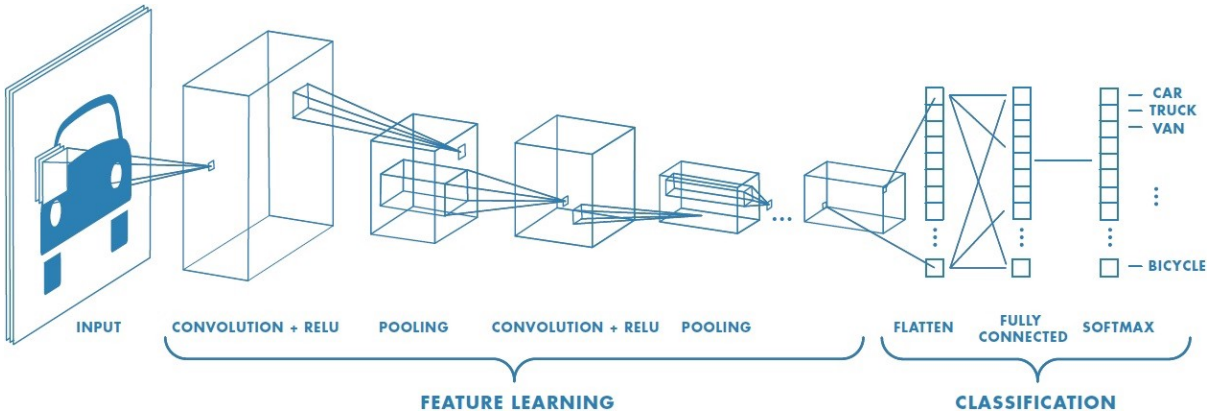
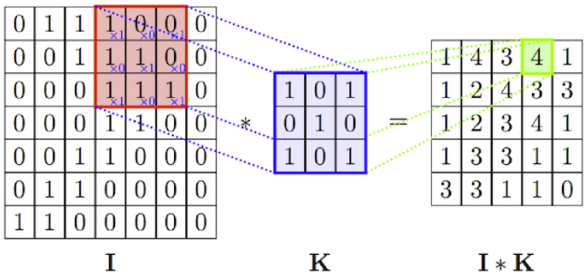


Figure 24: CNN architecture

The convolutional layers extract different features of the input. The first convolutional layer extracts low-level features like edges, lines and corners. High-level layers extract high-level features. Image feature extraction is done by the convolution kernel which is described in equation form as below, where K is the convolution kernel, I

is the image data in the for of matrix, h is the height of the image, and w is the width of image (Murphy, 2018).



$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

A complete convolutional layer includes a number of kernels transforming an input image into multiple output images which are then fed into an activation function to produce feature maps as the output for the subsequent layers. The most common activation functions are sigmoid function, hyperbolic tangent function and ReLU function. The convolutional layers can be customised by specifying the feature extractions, i.e. number of kernels and biases to be convolved with the input image, the size of the kernels, and the stride which is the number of pixels the kernel is slid along each time it is moved across the image. The following examples show how different kernels can transform the input image into different outputs for the purpose of edge detection in the convolutional layers (Rawat & Wang, 2017) (Figure 25)

The pooling layer, which comes after the convolutional layer, is to reduce the resolution of the feature maps to reduce the number of parameters to be computed but to achieve invariance to translations in shape, size and scale. Two common types of pooling are average pooling and max pooling. The max pooling is to select the largest element within a fixed-sized receptive field such at

$$Y_{kij} = \max_{(p,q)\in\mathcal{R}_{ij}} x_{kpq}$$

Where Y_{kij} is the output of the pooling operations associated with the k-th feature map, x_{kpq} is the element at location (p, q) contained by the pooling region $\mathcal{R}_{ij}$, a receptive field around the position (i, j) (Yu et al., 2014). The difference between max pooling and

average one is illustrated in Figure 26 The max pooling selects the max value of each receptive field; meanwhile, the average pooling averages all values in each receptive field (Rawat & Wang, 2017).

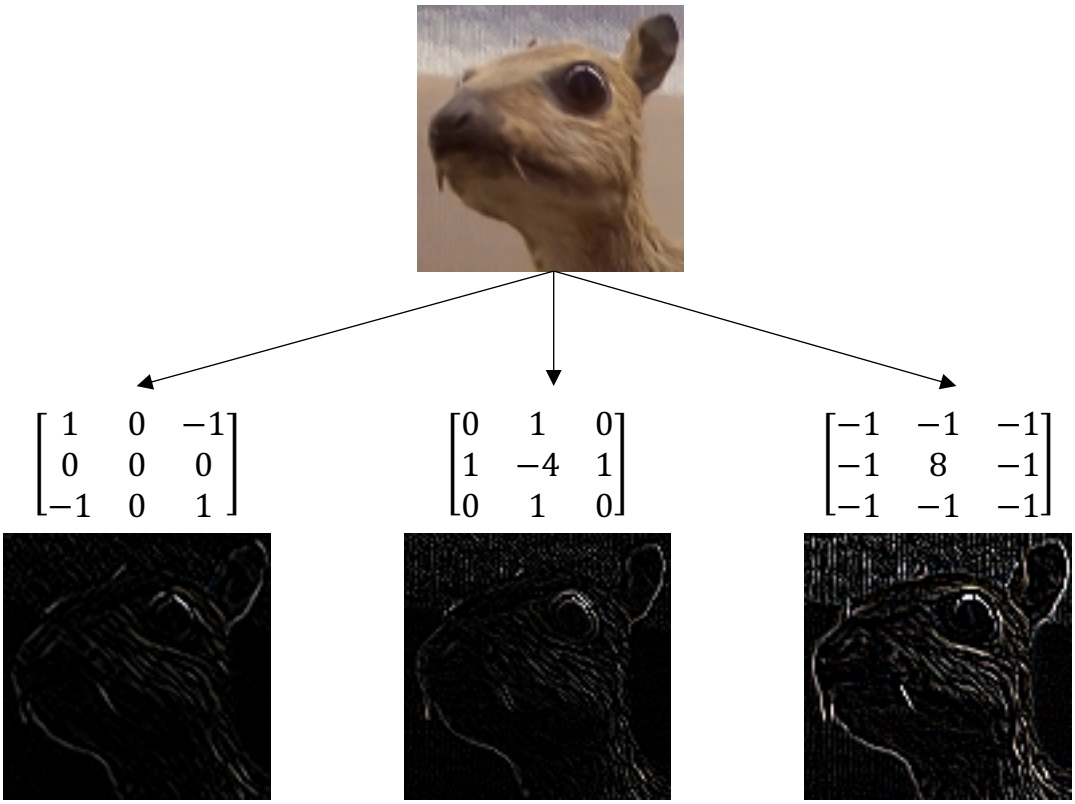


Figure 25. Convolution Kernels

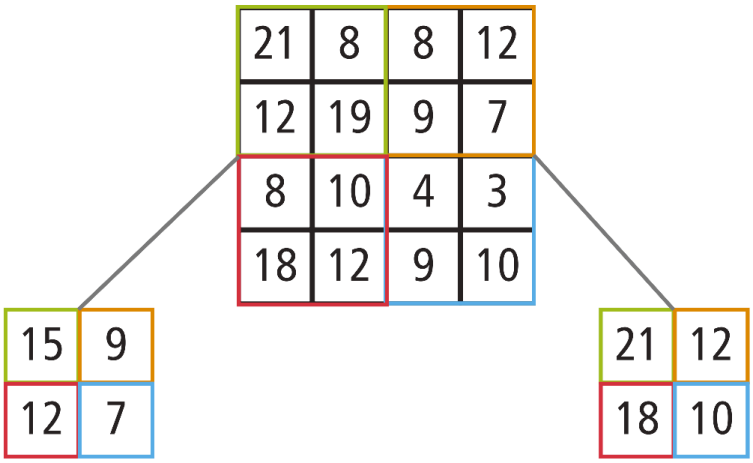


Figure 26: Average Pooling (left) vs. Max Pooling (right)

G.2. Deep learning in time series analysis

Time series analysis is a statistical technique dealing with time series data to explore characteristics, to detect trend and to produce future forecast associated with the time series. A number of approaches dealing with the time series include traditional statistical methods such as parametric autoregressive (AR) or autoregressive models with integrated moving average (ARIMA) and modern statistical ones such as machine learning or deep learning. Singling out one of the most well-acclaimed deep learning techniques, this report will focus on explaining how Long Short-term Memory (LSTM) is used for time series forecast.

Time series forecast can be formularised as the prediction of a time series $\{y_1, y_2, \dots\}$ at the time i based on its previous data $y_{i-1}, y_{i-2}, \dots$. If there is a vector $\boldsymbol{x} = \{y_{i-k}, y_{i-k+1}, \dots, y_{i-1}\}$ where $i = \{k, \dots, n\}$, the goal is to find a function $f(\boldsymbol{x})$ so that $\widehat{y_i} = f(\boldsymbol{x})$ is as close to the ground truth y_i as possible (Gamboa, 2017).

LSTM is one of the Recurrent Neural Network (RNN) variations which belong to a bigger family, Artificial Neural Network (ANN). ANN can be divided into two main categories namely RNN and Feed Forward Neural Network (FFNN). The main difference between RNN and FFNN is the neuron connection within the same hidden layer. Figure 27 and Figure 28 illustrate the difference between FFNN and RNN in which the former proposes only one-direction information flow, meanwhile, the latter allows interaction among neurons in the same layer. (Sundermeyer et al., 2013). The output $\widehat{y_t}$ is calculated from not only the input x_t but also the hidden state of the previous moment h_{t-1} (Figure 28). The capability of internally storing historical input information enables CNN to map the historical input data to the final output. In short, RNN is more suitable in handling historical dependencies like time series than FFNN. Unlike the theoretical standpoint of RNN, in practice, the RNN's solutions nevertheless seem less optimistic due to the phenomena called vanishing and exploding gradients (Hua et al., 2019).

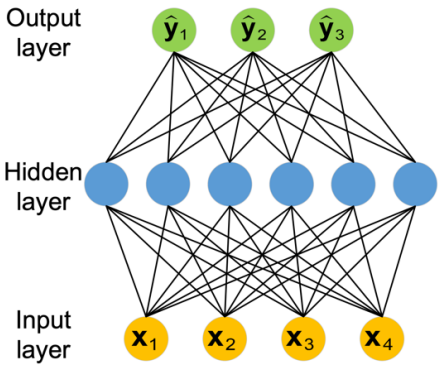


Figure 27. FFNN

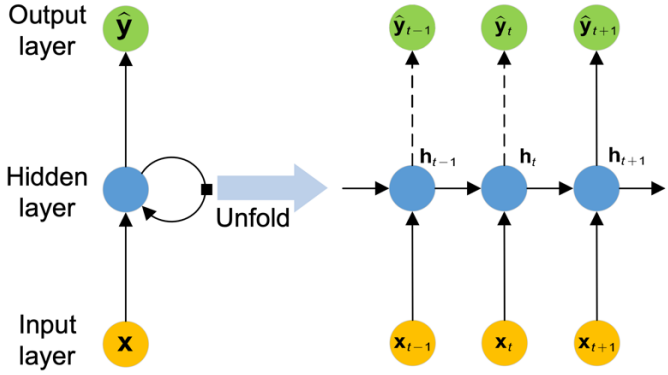


Figure 28. RNN

With similar overall architecture with RNN, LSTM advances its power of capturing historical dependencies by customising the hidden layer neuron as the memory block featuring recurrently connected modules called the memory cell and gates (Figure 29). The memory cell is responsible for remembering the temporal state of the neural network; meanwhile, the gates control the pattern of information flow. There are three main gates consisting of input gates, output gates and forget gates. Input gates decide how much new information is used in the memory cell. Forget gates controls how much information of the previous memory cell still remains in the current memory cell through recurrent connection. Output gates manipulate how much information is used to compute the memory cell's output activation which will flow into the rest of the neural network (Finsveen, 2018).

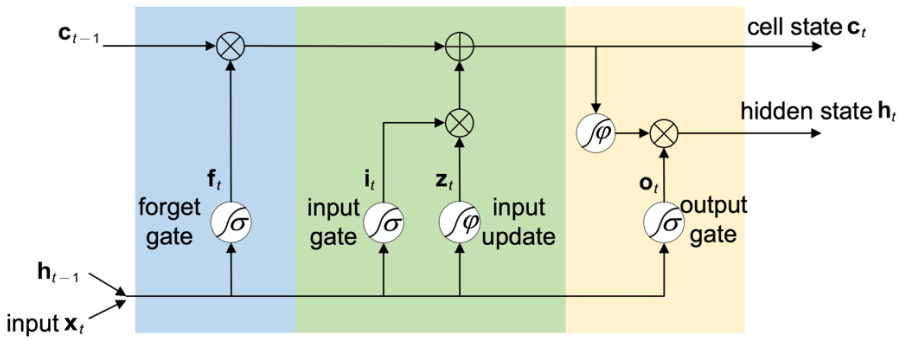


Figure 29: LSTM memory block

How LSTM works is explained in Figure 29 and Table 6 at a particular time t with the given hidden state vector h_{t-1} from the previous memory block and the input vector x_t. In the forget gate, the first step is to calculate how much information f_t is used from the previous memory cell (1) and how much information i_t is used from the input (2). Next, a cell input activation z_t is calculated (3). Then, the cell state vector is updated from c_{t-1} to c_t (4). The last step is to decide what to output which is the output vector of the LSTM unit h_t calculated from the output gate o_t (6). The sophisticated coordination between the memory cell and the gates grants LSMT an outstanding ability to predict time-series with long-term dependences (Hua et al., 2019).

	Where:
	$x_t \in \mathbb{R}^d$: input vector to the LSTM unit
	$f_t \in \mathbb{R}^h$: forget gate's activation vector
	$i_t \in \mathbb{R}^h$: input/update gate's activation vector
	$o_t \in \mathbb{R}^h$: output gate's activation vector
	$h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit
(1) $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$	$z_t \in \mathbb{R}^h$: cell input activation vector
(2) $i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$	$c_t \in \mathbb{R}^h$: cell state vector
(3) $z_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$	$W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times d}$ & $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training.
(4) $c_t = f_t \circ c_{t-1} + i_t \circ z_t$	The superscripts d and h refer to the number of input features and number of hidden units, respectively.
(5) $o_t = \sigma(W_o x_t + U_0 h_{t-1} + b_0)$	The initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product. The subscript t indexes the time step.
(6) $h_t = o_t \circ \tanh(c_t)$	The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$
	The tanh function $\tanh(x) = 2\sigma(2x) - 1$

Table 6. LSTM equations

Appendix

Appendix 1 Column Names

```
[1] "RFDBid"                        "Eng_Class"                     "Antennafilename1"
[4] "Antennafilename2"              "AntennagaindBd1"               "AntennagaindBd2"
[7] "AntennagaindBi1"               "AntennagaindBi2"               "Antennaheightm1"
[10] "Antennaheightm2"              "Antennamodel1"                 "Antennamodel2"
[13]     "AtmosphericabsorptionlossdB"                              "AverageannualtemperatureC"
"CirculatorbranchinglossdB1"
[16] "CirculatorbranchinglossdB2"   "dbperKmRatio"                  "DiffractionlossdB"
[19]    "DispersivefademargindB1"                                   "DispersivefademargindB2"
"Dispersivefadeoccurrencefactor"
[22] "EffectiveFadeMargindB1"       "EffectiveFadeMargindB2"        "EIRPdBm1"
[25] "EIRPdBm2"                     "Elevation2"                    "Elevationm1"
[28] "Emissiondesignator1"          "Emissiondesignator2"           "ERPdbm1"
[31] "ERPdbm2"                      "ERPwatts1"                     "ERPwatts2"
[34]   "FadeoccurrencefactorPo"                                     "FlatfademarginmultipathdB1"
"FlatfademarginmultipathdB2"
[37] "FreespacelossdB"              "FrequencyMHz"                  "Geoclimaticfactor"
[40]    "MainnetpathlossdB1"                                        "MainnetpathlossdB2"
"MainreceivesignaldBm1"
[43]    "MainreceivesignaldBm2"                                     "MiscellaneouslossdB1"
"MiscellaneouslossdB2"
[46] "OtherRXlossdB1"               "OtherRXlossdB2"                "OtherTXlossdB1"
[49] "OtherTXlossdB2"               "Passivegain2dB"                "Pathinclinationmr"
[52] "Pathlengthkm"                 "Polarization"                  "Radiofilename1"
[55] "Radiofilename2"               "Radiomodel1"                   "Radiomodel2"
[58] "RXthresholdcriteria1"         "RXthresholdcriteria2"          "RXthresholdleveldBm1"
[61] "RXthresholdleveldBm2"         "RXthresholdlevelv1"            "RXthresholdlevelv2"
[64] "ThermalFadeMargindB1"         "R_Powerfd1"                    "R_Powerfd2"
[67] "ThermalFadeMargindB2"         "Trueazimuth1"                  "Trueazimuth2"
[70] "TXpowerdBm1"                  "TXpowerdBm2"                   "DpQ_R2"
[73] "Verticalangle1"               "Verticalangle2"                "XPDfademarginmultipathdB1"
[76] "XPDfademarginmultipathdB2"    "Fullmaxt1"                     "Fullmint1"
[79] "Outcome"
```

References

Gamboa, J. (2017). *Deep Learning for Time-Series Analysis*. arXiv:1701.01887v1 [cs.LG].

Sundermeyer, M., Oparin, I., Gauvain, J.-L., Freiberg, B., Schluter, R. & Ney, H., (2013). *Comparison of feedforward and recurrent neural network language models*, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Presented at the ICASSP 2013 - 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Vancouver, BC, Canada, pp. 8430–8434. https://doi.org/10.1109/ICASSP.2013.6639310.

Hua Y., Zhao, Z., Li, R., Chen, X., Liu Z. & Zhang, H. (2019). *Deep Learning with Long Short-Term Memory for Time Series Prediction*. In: IEEE Communications Magazine, 57(6), pp.114-119, DOI: 10.1109/MCOM.2019.1800155.

Finsveen, L., (2018). *Time-series predictions with Recurrent Neural Networks: Studying Recurrent Neural Networks predictions and comparing to state-of-the-art Sequential Monte Carlo methods*. Trondheim: Norwegian University of Science and Technology.

Yu, D., Wang, H., Chen, P., & Wei, Z. (2014). Mixed pooling for convolutional neural networks. In *Proceedings of the 9th International Conference on Rough Sets and Knowledge Technology*. pp.364–375. Berlin: Springer.

Stanford University, (2020). *Convolutional Neural Networks for Visual Recognition* [Online]. Available from: https://cs231n.github.io/convolutional-networks/ [Accessed 14th May 2020].

Murphy, S. (2018). *Telecommunications signal performance classification using Deep Learning techniques*. Cork: Cork Institute of Technology.

Rawat, W. & Wang, Z. (2017). Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*. 29(9), pp.2352-2449.

Koo T. & Li M. (2016). *A Guideline of Selecting and Reporting Intraclass Correlation Coefficients for Reliability Research. Journal of Chiropractic Medicine*. 15 (2): pp.155–63.

He, Z., Lin, D., Lau, T. & Wu, M., (2019*). Gradient Boosting Machine: A Survey*. arXiv:1908.06951 [cs, stat].

Khan, R. (2019). Where did the Binary Cross-Entropy Loss Function come from? [Online]. Available from: https://towardsdatascience.com/where-did-the-binary-cross-entropy-loss-function-come-from-ac3de349a715 [Accessed 16th May 2020].