CYO Project

Umer Abid Ali

Contents

The following is the documentation of the final project in the course series HarvardX's Data Science: Professional Certificate. The project was labelled the 'Choose Your Own' Project thereby implying that, for this assignment, the student/learner would have optimum independence and freedom in terms of choosing their datasets to work with and modelling techniques to use.

Overview

When it came to choosing my dataset, although there were many that caught my eye, I decided to opt for the well-documented **Banknote Authentication** dataset available on most popular dataset archives such as UCI and Kaggle. The information present in this dataset was derived from the images of a set of banknotes, some of which were genuine (real) whilst the others were specimens (fake). The images were captured using an industrial camera whose main purpose was print inspection. Each image initially had a resolution of **400 x 400** pixels. The images were then gray-scaled and approximately had a resolution of **660** dpi (dots-per-inch) each. Finally, the **Wavelet Transform Tool** was used to extract the features of each image and ultimately each banknote. The features extracted were the following:

- **Variance** - This is the obtained variance of the wavelet transformed image. It is a **continuous** value usually ranging between $-8 \leq Var \leq 8$
- **Skewness** - This a measure of the lack of symmetry of the transformed image. It is a **continuous** value usually ranging between $-14 \leq S \leq 14$
- **Kurtosis** - This is a measure of the obtained kurtosis of the transformed image. It is a **continuous** value usually ranging between $-6 \leq K \leq 18$
- **Entropy** - This is a measure of the entropy obtained from the transformed image. It is also a **continuous** value usually ranging between $-9 \leq E \leq 3$

Along with the above mentioned list of features, the dataset also includes the **Class** of each banknote.

- **Class** - This is the outcome variable (the variable to be predicted) and is a **factor** having the levels *0* and *1* with 0 representing a **forged** banknote and 1 representing a **genuine** one

The goal of this project is to employ different machine learning techniques and algorithms in order to predict the *Class* of a banknote given its features, and compare their performance. As can be observed from the above information on the dataset, this is primarily a classification problem since the outcome variable is a factor/class and not a continuous value. However, that's not to say that it can't be approached as a regression problem as well.

Methods and Analysis

The following section covers the different stages and processes involved in the preparation and exploration of the data followed by an account on the different modelling techniques used to make predictions.

Data Preparation

This stage involved the three main steps of downloading, reading and cleaning the data and prepping it for different analyses. To download the dataset, I used the version available on the **UCI Machine Learning Repository**. The dataset downloaded as a text file (.txt) by default with each line representing one instance/observation and the values of each variable separated by a *comma (,)*. Therefore, I figured the *read.table()* function available in base R would be most suitable for reading it. The following piece of code was used to do so:

```
#Creating a temporary directory to store the downloaded file
temp <- tempfile()
#Storing the URL of the data file in a variable
url <- "http://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt
#Downloading the file and storing it in the temporary directory
download.file(url, destfile = temp)
#Reading the file into a data.frame in R
banknote_tab <- read.table(temp, header = FALSE, sep = ",")
#Deleting temporary variables
rm(temp, url)
```

Once downloaded and loaded into R, the dataset had the following structure:

```
## 'data.frame':    1372 obs. of  5 variables:
##  $ V1: num  3.622 4.546 3.866 3.457 0.329 ...
##  $ V2: num  8.67 8.17 -2.64 9.52 -4.46 ...
##  $ V3: num  -2.81 -2.46 1.92 -4.01 4.57 ...
##  $ V4: num  -0.447 -1.462 0.106 -3.594 -0.989 ...
##  $ V5: int  0 0 0 0 0 0 0 0 0 0 ...
```

As can be seen from the above console output, there were some significant changes required before the dataset could be put to work, namely:

- Giving each column a **name**. As of now it is not recognizable which column represents which feature
- Defining the outcome variable (V5) as a **factor**

In order to figure out the names of each column, I referred to the dataset information provided on the UCI website and came up with the following:

```
#Defining column names
names(banknote_tab) <- c("Variance", "Skewness", "Kurtosis", "Entropy", "Class")
```

The next step was to define the *Class* column as a factor:

```
#Redefining 'Class' as a factor
banknote_tab <- banknote_tab %>% mutate(Class = factor(Class))
```

The dataset was now almost fully prepped and ready to be worked with, just lacking a check on the presence of missing values or **NAs**.

```
#Searching for NAs
sum(is.na(banknote_tab))
```

```
## [1] 0
```

Now that it was ensured that the dataset was clean and complete, it was time to begin exploratory data analysis.

Exploratory Data Analysis

Exploratory data analysis of the dataset involved visualizing and quantifying the distributions of the different features present in it. In order to gain a general overview of the distribution of each feature, I used the *summary()* function available in R which yielded the following results:

```
summary(banknote_tab)
```

```
##     Variance          Skewness          Kurtosis          Entropy
##  Min.   :-7.0421   Min.   :-13.773   Min.   :-5.2861   Min.   :-8.5482
##  1st Qu.:-1.7730   1st Qu.: -1.708   1st Qu.:-1.5750   1st Qu.:-2.4135
##  Median : 0.4962   Median :  2.320   Median : 0.6166   Median :-0.5867
##  Mean   : 0.4337   Mean   :  1.922   Mean   : 1.3976   Mean   :-1.1917
##  3rd Qu.: 2.8215   3rd Qu.:  6.815   3rd Qu.: 3.1793   3rd Qu.: 0.3948
##  Max.   : 6.8248   Max.   : 12.952   Max.   :17.9274   Max.   : 2.4495
##  Class
##  0:762
##  1:610
##
##
##
##
```

As can be observed from the above output, all 4 features generally have distributions centered around the $-1 \leq \mu \leq 1$ interval but however have variable ranges. As for the outcome variable (Class), it can be seen that the prevalent class is 0 thereby implying that majority of the banknotes investigated were specimens. However, it must be noted that the margin of prevalence is not huge (only 152 more forged notes than genuine ones).

Visualizing the above mentioned distributions in the form of smooth density plots using the *ggplot* package yielded the following results:

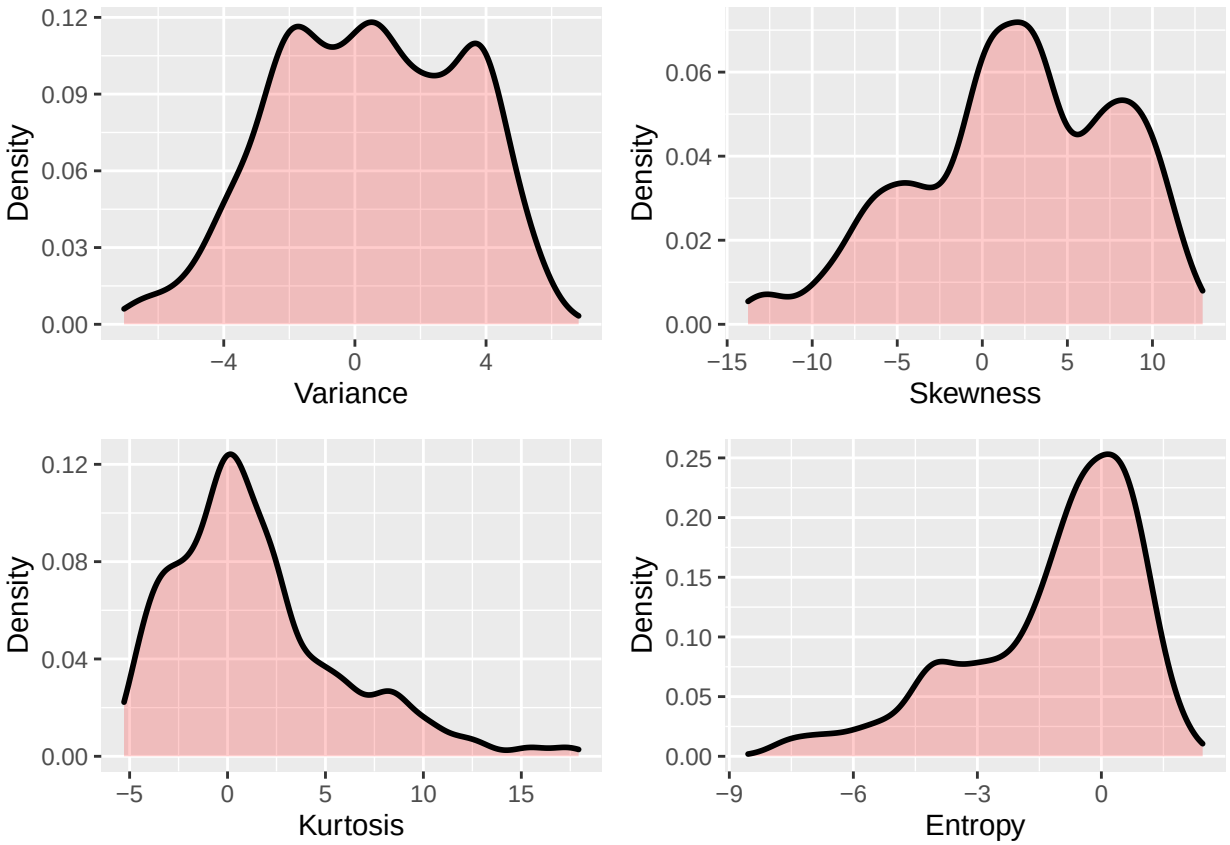

Figure 1: Smooth density plots visualizing distributions of each feature

As can be seen from the above plots, almost none of the 4 features have distributions approximated by the normal distribution. Both *Variance* and *Skewness* have multiple peaks whereas *Kurtosis* and *Entropy* do have a single peak at 0 but however do not have their distribution symmetric about that peak.

With the distributions quantified and visualized, what remained was exploring the correlations, if any, between the features as this information might come in handy when making predictions. The following *corrplot* or *correlation-plot* provides a summary of the different correlations computed between the 4 features:

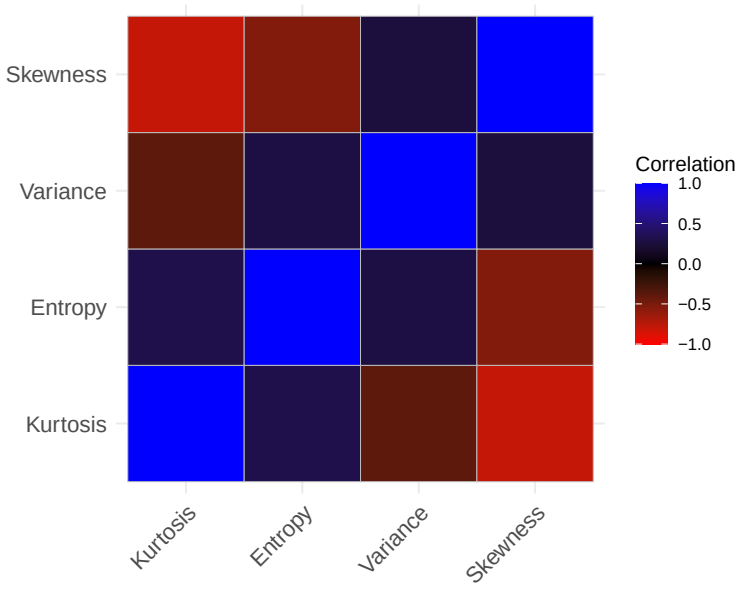

Figure 2: Corrplot visualizing the computed correlations between the 4 features

As can be seen from the above plot, most of the correlation coefficients are in the $-0.5 \leq r \leq 0.5$ range thereby implying to be weak. One significant correlation, however, is the one between the *Kurtosis* and *Skewness* of a Wavelet Transformed image of a banknote. Since this correlation is negative, it implies that as the value of *Kurtosis* increases, the *Skewness* of the image decreases and vice versa.

Modelling Techniques

As this dataset is primarily a classification problem which can also be approached with a regression-based model, there were many popular modelling techniques that could be employed to make predictions. I decided to use the following 4 prediction algorithms and compared their respective performances:

- Logistic Regression Model
- K-Nearest-Neighbors Model (KNN)
- Decision Tree Model
- Random Forest Model

However, before work could begin on building and testing models, the dataset first needed to be subsetted into *training* and *test* sets respectively. When determining the ratio by which I would split my data, I decided to opt for an 80-20 split where 80% of the data was taken as the training set (named *edx* in code) and 20% as the final hold-out test set (named *validation* in code). The reasoning behind choosing this ratio was that, as I was going to make use of *k-fold cross validation*, a technique whereby, in each iteration of the execution of a model while training, a random selection of the available training data is subsetted and used as a *dummy test set*. This is done to test different values for the tuning parameters of the model and to determine the best suited value for said parameters when applied to the available data. Therefore, in order to ensure that there was enough data left for training after the random subsetting was done, the overall training data had to be a substantial proportion of the total data, thus making an 80-20 split appropriate. The following code was used to generate an index that would subset 20% of the data for the *validation* set.

```
#Setting the seed to 1 in order to enable reproduction of datasets
set.seed(1, sample.kind = "Rounding")
#Defining an Index to subset 20% of the data
validation_index <- createDataPartition(banknote_tab$Class, p = 0.2, list = FALSE)
#Defining the training set, 'edx', as 80% of the data
edx <- banknote_tab %>% slice(-validation_index)
#Defining the test set, 'validation', as 20% of the data
validation <- banknote_tab %>% slice(validation_index)
```

The training and test sets were now ready. However, since majority of the models were made and trained using the **caret** package, the training set needed to be split into two different objects in order to make it suitable for use with the *train()* function, namely:

- A **matrix** of the 4 features
- A **vector** of all the outcomes (0 or 1)

The following is the code used to make the above mentioned objects:

```
#Excluding the Class (outcome) column and converting to matrix
edx_features <- as.matrix(edx %>% select(-Class))
#Extracting the Class column of the training set
edx_outcome <- edx$Class
```

With the training set now ready to be used, the time had come to start building models. However, before we could begin doing so, a metric had to be decided that would be used to compare the performances of the models. Since this is a classification problem, the most suitable metric would be to compute the **Accuracy** of each model i.e. the proportion of outcomes correctly predicted. In code, this would be computed as part of the *confusion matrix* of each model. In mathematical notation, the formula for accuracy is as follows:

$$Accuracy = \frac{Number\ of\ correctly\ predicted\ outcomes\ (\widehat{y} = Y)}{Total\ number\ of\ predictions\ made\ (N)}$$

Where each symbol has the following definition:

- $\widehat{y}$ - This is the predicted class of a given banknote
- Y - This is the actual class of a given banknote
- N - This is the total number of predictions made

Logistic Regression Model

This is the only regression-based modelling approach that I used to make predictions. The model would compute the probability of a banknote being genuine (outcome of 1) given its features. If this probability was greater than 50% (0.5), then the banknote would be classified as genuine, otherwise it would be classified as a specimen (outcome of 0). Mathematically, this probability would be denoted in the following way:

$$\widehat{p} = P(C = 1 \mid X)$$

Where each symbol has the following definition:

- $\widehat{p}$ - The probability computed by the model
- C - The class whose probability is computed, in this case, the class is always 1
- X - This symbol represents the values of the 4 features that are available of the banknote image

For $0 < \widehat{p} < 0.5$, the predicted class was 0 and for $0.5 < \widehat{p} < 1$, the predicted class was 1. In order to build the model, I used the *glm()* function available in base R in the following way:

```
#Creating a logsitic regression model using the glm function
#The argument 'family = binomial(link = "logit")'
#Specifies that a logistic regression model is to be created
glm_model <- glm(Class ~ ., data = edx, family = binomial(link = "logit"))
```

A summary of the computed regression coefficients of the model is given below:

```
## (Intercept)    Variance    Skewness    Kurtosis     Entropy
##   6.8806300  -7.0941929  -3.8107970  -4.8246944  -0.5429228
```

Proceeding to compute probabilities using the model, I used the *predict()* function also available in base R with the argument *'type = "response" '* in order to specify that *probabilities* are to be computed and not *outcomes*. The following is the code used to do so:

```
#Computing the probability of each banknote in the validation set being genuine
probs <- predict(glm_model, newdata = validation, type = "response")
```

The plot below visualizes these probabilities:

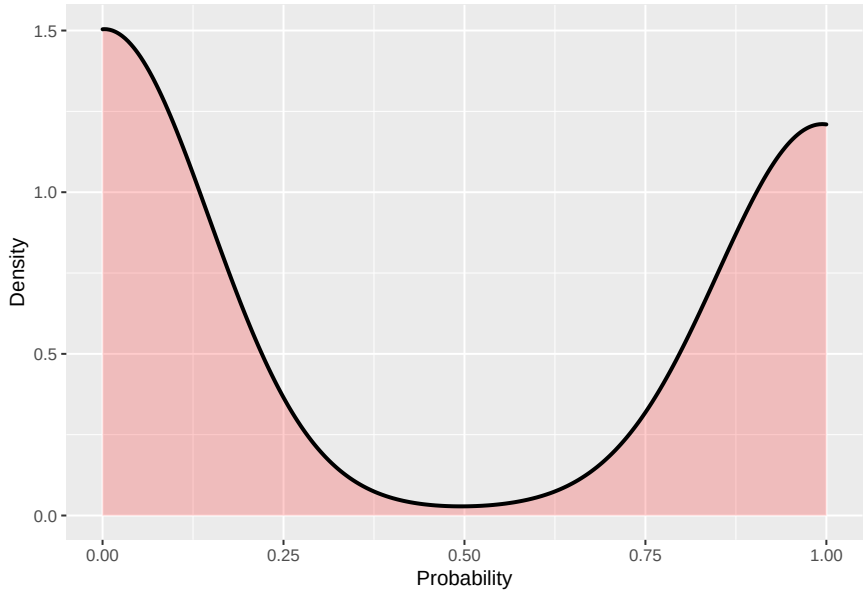

Figure 3: Smooth Density plot showing the distribution of computed probabilities

As can be seen from the plot above, the probabilities are highly concentrated at the extreme values of 0 and 1 and are more prevalent on the left side of the axis i.e. towards 0. Thereby suggesting that the prevalent class in the predictions is going to be 0.

The next step was to use these probabilities to predict the class of each banknote in the validation set. A simple *ifelse* statement was used to achieve this:

```
#Predicting the class of each banknote in the validation set
#Banknote is predicted to be genuine (1) if probability is greater than 0.5
#Otherwise, it is predicted to be a specimen (0)
glm_preds <- ifelse(probs > 0.5, 1, 0) %>% factor()
```

K-Nearest-Neighbors (KNN) Model

The **KNN Model** generally has a use-case in tackling classification and regression problems and is therefore an appropriate means of making predictions on this dataset. The basic idea regarding the functionality of the model is that, in the case of a classification problem, for a given data point of an unknown category, the model looks for the k nearest *neighbors* or data points based on euclidean distance, where k is a tuning parameter. It then determines the prevalent category among them. This category or class is then predicted as the category or class of the unknown data point.

In the context of this dataset, for a given banknote of unknown class, the model would examine the k nearest banknotes based on the values of the 4 features and would predict the banknote in question to be of the prevalent class amongst its *neighbors*

In order to create and optimize this model, I used the *train()* function available in the *caret* package in the following way:

```
#Creating and training the KNN Model
#The 'tuneGrid' argument is used to define a set of values for K
#Which is a tuning parameter and needs to be optimized
knn_model <- train(edx_features, edx_outcome,
                   method = "knn",
                   tuneGrid = data.frame(k = seq(2, 10, 2)))
```

The plot below shows how the performance of the model varied with different values of k.

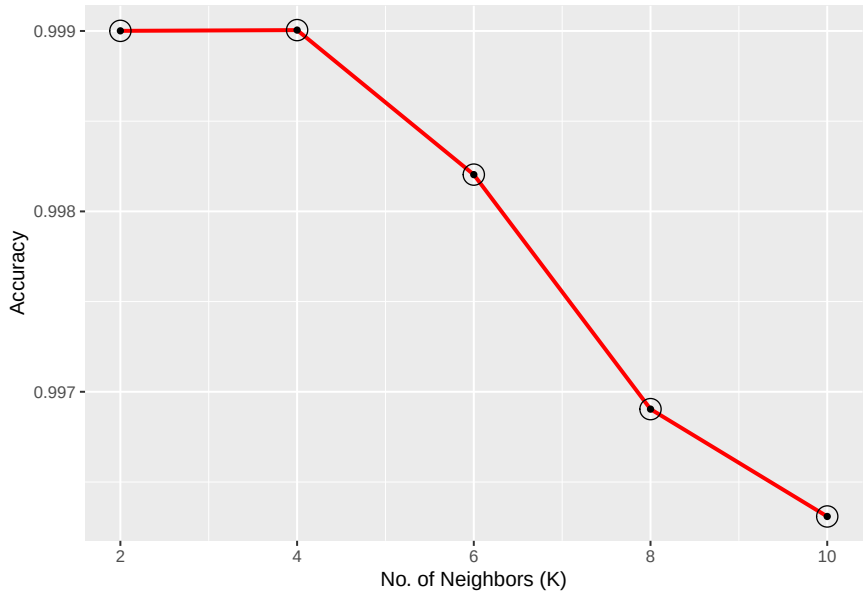

Figure 4: Line graph showing how model accuracy varies with increasing K

As can be seen from the plot, the optimal value for K is 4 as the accuracy after cross-validation is maximized at this value. Therefore, it was set to 4 when using the model to make predictions on the validation set. The predictions were again made using the *predict()* function available in base R:

```
#Making predictions on the validation set with the KNN Model
knn_preds <- predict(knn_model, newdata = validation)
```

Decision Tree Model

The **Decision Tree Model** is another modelling technique used mainly for classification and regression problems and is therefore applicable on our dataset. The functionality of this model lies in its name, a *Decision Tree*! The model analyzes the given training data and builds a decision tree based on it which is used to classify new data into a category. At each node of the tree is either an outcome (in this case either 0 or 1) or a condition whose fulfillment determines what path the data takes down the tree to the above mentioned outcomes.

The model involves the optimization of one tuning parameter, which is the **Complexity Parameter**, also known as *CP*. This parameter decides the optimal size of the decision tree i.e. if after the splitting of a branch and addition of a new node, the cross-validation error doesn't reduce by a factor of *CP*, then that node and ultimately branch split is considered as unnecessary, thereby preventing the tree from growing further and saving processing time.

Although this model has its own function available in R, I decided to opt for training it using the *caret* package instead for consistency across models. Below is the code used:

```
#Creating and training the Decision Tree Model
#The 'tuneGrid' argument is used to define a set of value to test as CP
#Which is a tuning parameter that needs to be optimized
rpart_model <- train(edx_features, edx_outcome,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.01, 0.05, 0.005)))
```

The plot below illustrates how the cross-validation accuracy varies with different values of *CP*.

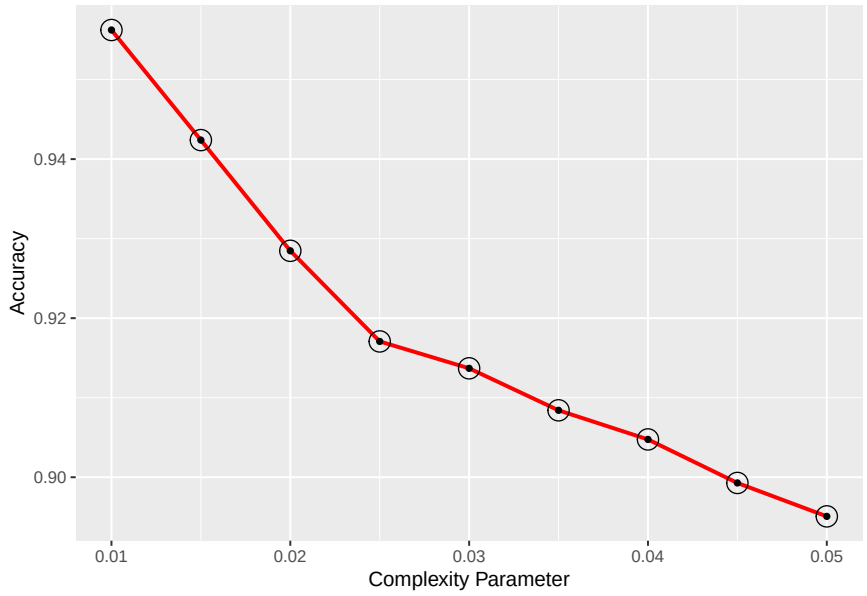

Figure 5: Line graph visualizing the variation of cross-validation accuracy with increasing CP

As can be seen from the above plot, cross-validation accuracy is maximized when the complexity parameter is set to 0.01. Therefore, when making predictions, the final model will have it set at this value in order to gain the best possible results.

The final decision tree created by the model which will be used to make predictions on the validation set is as follows:

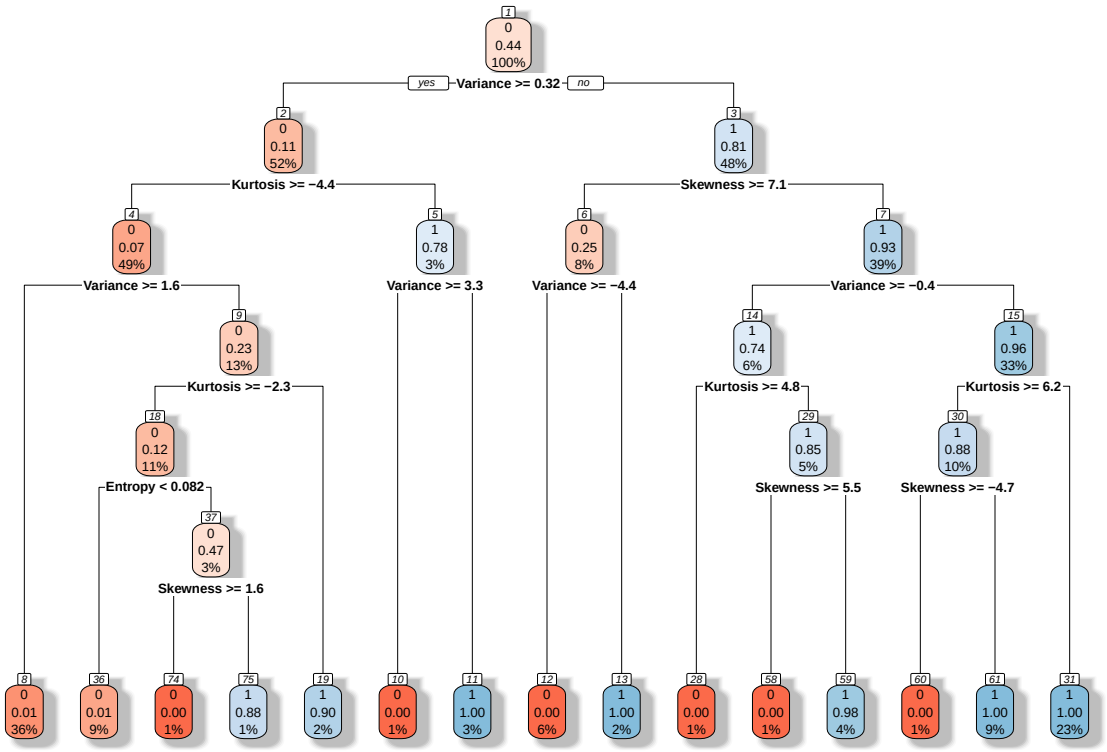

Figure 6: Decision Tree to be used when making predictions

Proceeding to applying the model on the validation set, I used the following piece of code to create a vector that contained the predictions for each banknote in the validation set:

```
#Making predictions on the validation set using the decision tree model
rpart_preds <- predict(rpart_model, newdata = validation)
```

Random Forest Model

The **Random Forest** model is an extension to the *Decision Tree* model discussed above with the main difference being that now, instead of using a single tree to make predictions, we will be using a *forest* of trees in which each tree has been trained using a **random** subset of the training data. When new data is given to the model, the data goes through every single tree in the *forest* and, in the case of a classification problem, the modal class that is predicted across all trees will be the final prediction made. To sum it all up, random forests are an **ensemble** of a group of individual decision trees.

The model involves the use of two tuning parameters which are explained below:

- **ntree** - This parameter represents the number of trees that are to be grown in the forest. Generally, this value ranges from 50 for small datasets to 500 or more for larger datasets. For this model, I kept this value defaulted to 500 throughout training (reasons explained later)
- **mtry** - This parameter represents the number of features that are to be considered at each individual node of every single decision tree. For classification problems, this is often set to the square-root of the total number of features available. However, other values can be tested as well

To build this model, I again stuck with the *caret* package and used the following code:

```
#Creating and training the Random Forest Model
#The argument 'tuneGrid' is used to define a set of values
#Which are to be tested for the parameter mtry
rf_model <- train(edx_features, edx_outcome,
                  method = "rf",
                  tuneGrid = data.frame(mtry = seq(1, 4, 1)))
```

The following plot visualizes how the cross-validation accuracy of the model varies with increasing number of features considered at each node (mtry):

Figure 7: Plot showing how accuracy varies with increasing number of features considered at each node

As can be seen from the above plot, the accuracy maximizes at $mtry = 1$ therefore in the final model, this parameter is set to 1.

As for the optimization of the number of trees, the following plot shows how the **Out-of-the-box error rate (OOB)** varies with increasing number of trees:

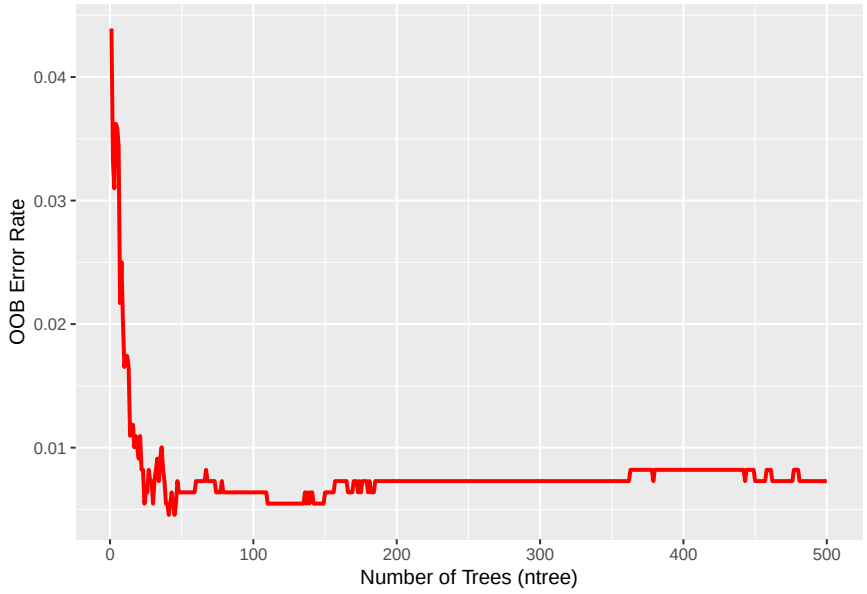

Figure 8: Plot illustrating how OOB Error rate varies with increasing number of trees

As is visible, the OOB error rate starts to stabilize approximately at $ntree = 200$ and doesn't undergo significant variation that point forward. Therefore, 500 trees is an appropriate number for this forest.

Now that the random forest was fully grown, it had to be put to the test with the validation set. The following piece of code was used to do so:

```
#Making predictions on the validation set using the Random Forest Model
rf_preds <- predict(rf_model, newdata = validation)
```

With all 4 models trained and optimized, all that was left was to compare their respective performances when applied to the validation set. A good way to start, however, would be to recap all the models we've built so far. The table below lists all the models made and also summarizes each model's respective tuning parameters (if any) and their values set after optimization:

Model	Tuning Parameters	Optimized Value
Logistic Regression Model	None	-
K-Nearest-Neighbors (KNN) Model	No. of Neighbors (K)	4
Decision Tree Model	Complexity Parameter (CP)	0.01
Random Forest Model	No. of features considered at each node (mtry)	1

Results

The following section compares the performances of the 4 models described above when applied on the validation set (also defined above). There will be three main metrics that will be used to quantify and compare the performances of these models, namely:

- **Overall Accuracy** - This is the ratio of predictions that are correctly made to the total number of predictions made. The greater this value is, the better
- **Sensitivity** - This is the ratio of the number of banknotes that are correctly predicted to be **genuine (class of 1)** to the total number of genuine banknotes. The greater this value is, the better
- **Specificity** - This is the ratio of the number of banknotes that are correctly predicted to be **forged (class of 0)** to the total number of forged banknotes. The greater this value is, the better

The following bar plot visualizes the distributions of predictions made by each model**:

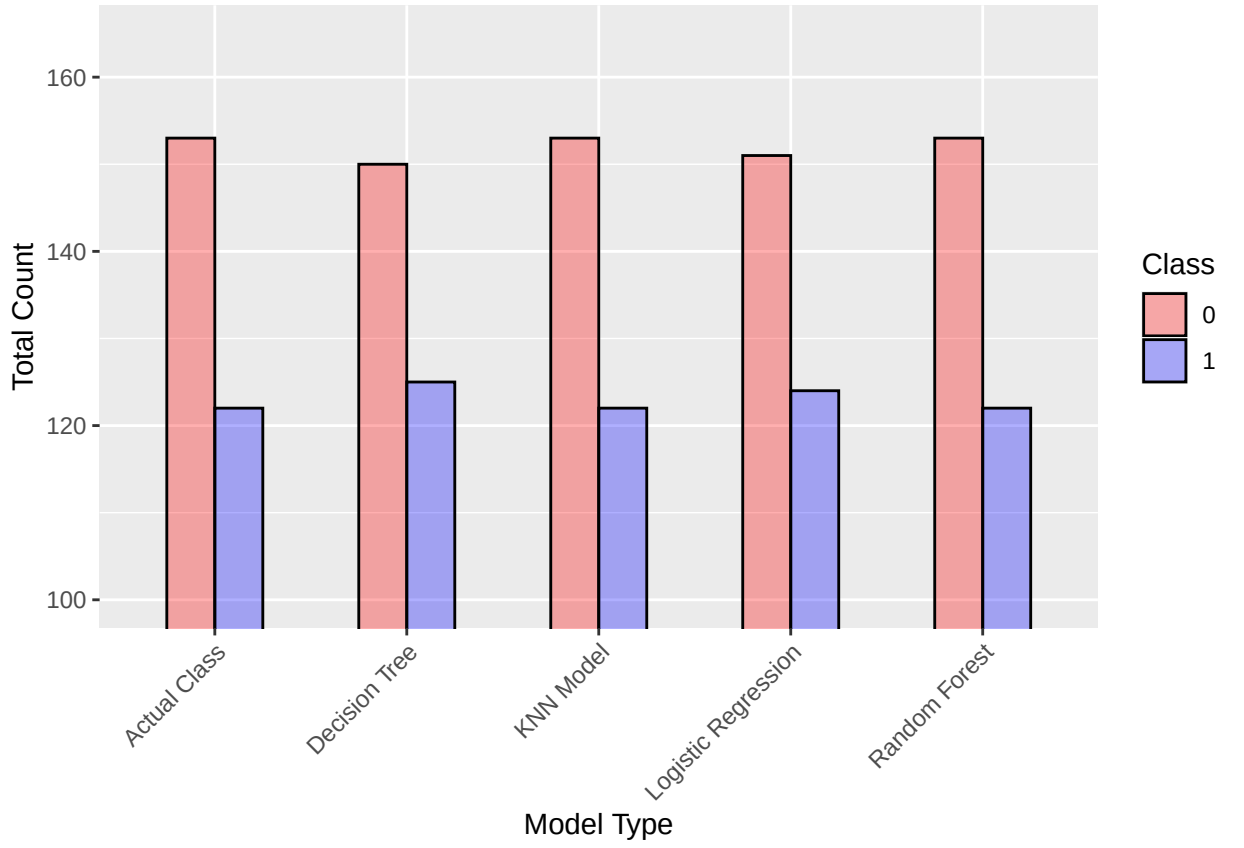

Figure 9: Barplot visualizing the distribution of predictions made across models

As can be seen from the plot on the previous page, almost all 4 models made predictions of distributions more or less the same as that of the actual distribution thereby implying that the models would have similar and high accuracy, sensitivity and specificity. These values are visualized in the series of bar plots below:

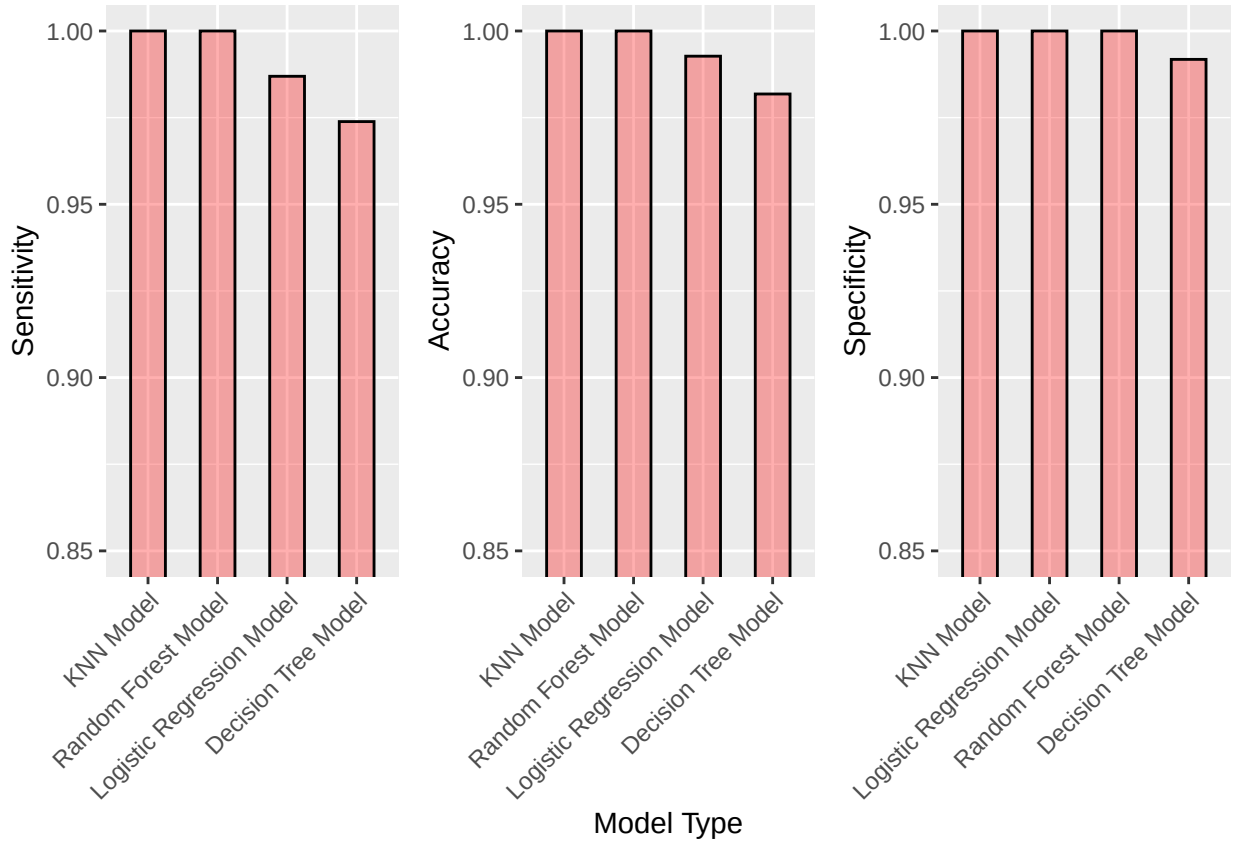

Figure 10: Bar plots comparing the overall performances of the different models

From the plots, it is clear who is the winner here. The **KNN** and **Random Forest** models outperform all other models and yield a 100% accuracy! The **Logistic Regression** model performs second best (based on accuracy) and is followed by the **Decision Tree** model which comes in at 3rd place with an accuracy below 99% .

Coming on to comparing the sensitivity and specificity of each model, it must be noted that, as the *KNN* and *Random Forest* models already yield a perfect 100% accuracy, they will automatically have the best sensitivity and specificity as well (each 1.0). Therefore, the purpose of comparing these metrics is more to determine the best amongst the other 2 models. In terms of sensitivity, the models follow the same rank order as they did when compared on overall accuracy. However, looking at the specificity of each model, the **Logistic Regression** model posts an outstanding specificity of 1.0 (maximum possible) and joins the top 2. The odd one out is, again, the **Decision Tree** model which ranks last in all three metrics.

** Note - Figures 9 and 10 have the y-axis starting at non-zero points for the purpose of better visualizing the differences in values.

Overall, the following table summarizes the performances of each model:

Rank	Model Name	Accuracy	Sensitivity	Specificity
1	KNN Model	1.0000	1.0000	1.0000
1	Random Forest Model	1.0000	1.0000	1.0000
2	Logistic Regression Model	0.9927	1.0000	0.9869
3	Decision Tree Model	0.9818	0.9918	0.9739

Conclusion

To conclude this project, although one can be impressed with the performances of the models, particularly the **KNN** and **Random Forest** models, it must be noted that this report only discusses a handful of algorithms and that there's still a plethora of other and possibly better modelling techniques available that have applications in the the dataset used. However, that's not taking any credit away from these models who outshone their competitors in almost all areas of comparison, particularly in terms of overall accuracy and computational cost. Although, it must be said that the dataset used was fairly small and therefore computational cost may increase as and if it gets larger.

In terms of future work, I think there are many other modelling techniques that could be applied and tested on this dataset alongside of which work can also be done on the expansion of the dataset and increasing the available features which may make it more applicable for use cases in *Random Forests* and *Decision Trees* and other similar modelling techniques.