# Machine learning: decision trees, random forests and artificial neural networks

By A Smith

April 2020

## 1 Abstract

In this project, we will discuss two machine learning techniques: *random forests* and *artificial neural networks* and use them to perform classification tasks. The project begins by examining decision trees, which form the basis of random forests, and applying these to 2 datasets. We improve on the classifications performed by decision trees by implementing a more sophisticated method on one particular dataset: random forests. We then assess the effectiveness of this method. Finally, we discuss an alternative machine learning technique: artificial neural networks, and again assess how well this model performs a classification task. Afterwards, we discuss the relative advantages and disadvantages of each method and discuss how a statistician may decide which machine learning method to implement for a given classification problem on a dataset.

## 2 Introduction to machine learning

Machine learning is one of the most rapidly expanding sectors of data science. Recent research has resulted in several new fields within this subject and some already established methods have been radically expanded on. Machine learning can be defined as 'the study of algorithms and techniques for automating solutions to complex problems that are hard to program using conventional programming techniques'[1]. From this definition, it is clear to see why such techniques and algorithms are so desirable. Given the increasing complexity of modern datasets, using computers and machinery to automate solutions allows us to solve increasingly complex problems quickly and effectively.

Machine learning can be divided into two broad categories: *unsupervised* and *supervised*.

*Unsupervised methods*: unsupervised machine learning methods are not guided by pre-existing labels or classifications of the data. In this method, we allow the

model to work independently, without our supervision. Common unsupervised machine learning techniques include *principal component analysis* (PCA) and *clustering*. PCA methods summarise the most important components in multivariate data based on analysis, whereas clustering techniques group data based on similarity. These methods do not require the user's supervision.

*Supervised methods*: supervised machine learning methods require us to create a model based on known outcomes or expectations. Supervised learning algorithms typically analyse some training data to produce a function which can be used to classify unseen examples. The most common types of supervised learning techniques include *artificial neural networks*, *support-vector machines* and *random forests*.

# 3 Decision trees

Decision trees work by partitioning datasets into smaller subsets based on some criteria. Decision trees may take continuous variables or discrete variables. When sorting continuous variables, the trees are called *regression trees*, whereas for discrete or categorical variables, the trees are called *classification trees*. We now examine how to create decision trees in *R*. The *R* package *rpart* allows us to create decision trees based on a dataset. This package is named for the technique it uses: *Recursive Partitioning And Regression Trees* (RPART)[2].

## 3.1 Decision trees on a dataset: *iris*

For a simple demonstration of how we may use deision trees, we examine the dataset from Ronald Fishers' 1936 paper: *The Use Of Multiple Measurements In Taxonomic Problems* [3]. The dataset contains 150 samples of iris flowers and 5 variables: *sepal length*, *sepal width*, *petal length*, *petal width* and *species.* We note that all measurements are taken in centimetres (cm). We begin by plotting the data. Seeing as we have multivariate data, we create a plot of the first two principal components:

```
fviz_pca_ind(iris.pca,geom.ind="point", col.ind=iris$Species,
addEllipses = TRUE, xlab="Principal component 1",
ylab="Principal Component 2", title="Plot of first two principal
components on Iris dataset")
```

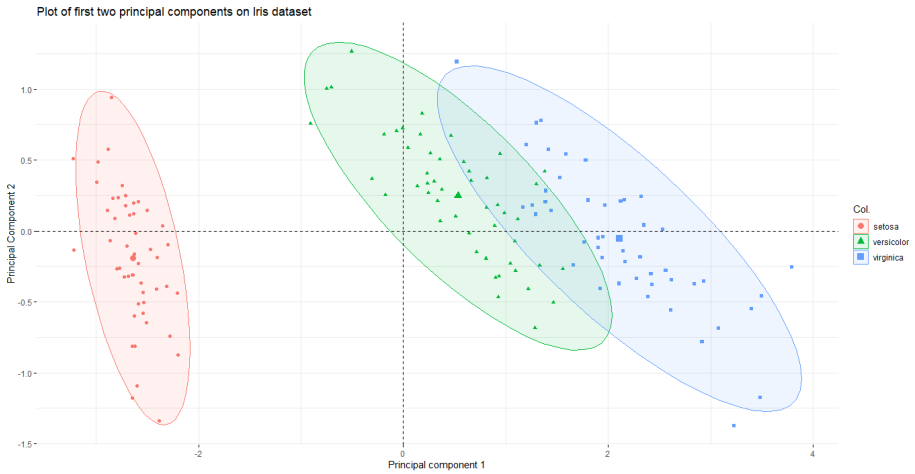


Figure 1: *It is clear to see that this data can be partitioned into 3 clusters. Note that there is an overlap between the observations of the 'versicolor' and 'virginica' species, however the 'setosa' species observations are clustered a large distance away from the other species.*

We also create a plot of two variables: *petal length* and *petal width*. We see the plot below:

```
plot(iris$Petal.Length, iris$Petal.Width,
col=(bg = c("red", "blue", "green")[unclass(iris$Species)]),
main="Plot of Iris dataset",xlab="Petal Length",
ylab="Petal Width",pch=16)
legend("topleft", legend=levels(iris$Species),
col=c("red", "blue", "green"),pch=16)
```

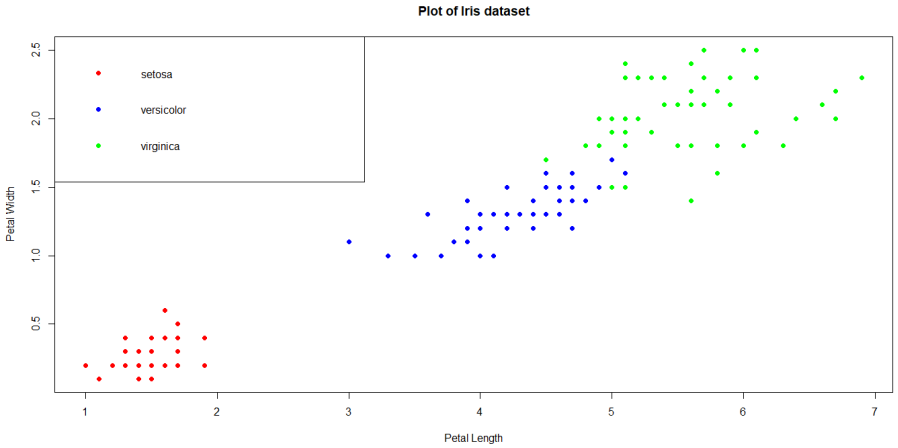


Figure 2: *From the above plot, we see how a decision tree may be used to classify new observations.*

We imagine that any new observation with a petal length of less than 2.45 will belong to the *setosa* species. Any new observation with a petal length greater than or equal to 2.45 will belong to either the *versicolor* or *virginica* species. We also see that any observation with a petal length greater than or equal to 2.45 but a petal width of less than 1.75 will probably belong to the *versicolor* species. Finally, an observation with a petal length greater than 2.45 and a petal width of greater than 1.75 will probably belong to the *virginica* species.

From the above information, We begin to see how a decision tree would help us classify new observations to an appropriate species. We create this decision tree using *R*:

```
install.packages("rpart")
library(rpart)
decisiontree<-rpart(Species~.,data=iris)
par(xpd=NA)
plot(decisiontree, uniform=TRUE,
main="Classification Of Species for Iris")
text(decisiontree,digits=3)
```

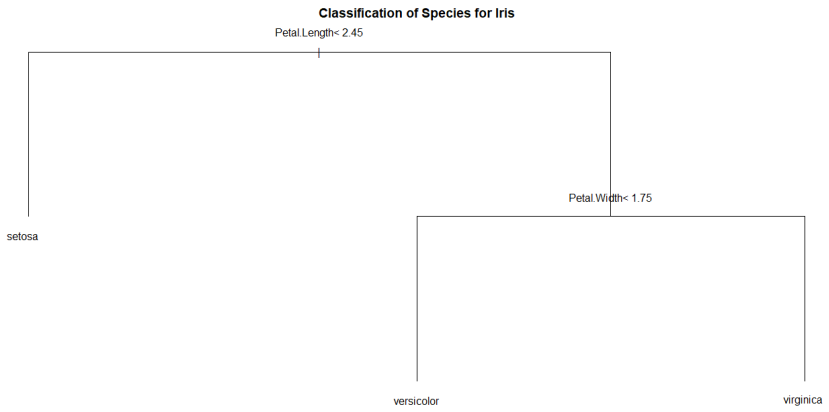


Figure 3: *Here we see a simple decision tree which we may use to classify new observations.*

In the above plot, we see graphically how we may allocate new individuals to a species. Any new observations with petal length less than 2.45 are assigned to the left-hand side of the tree into the *setosa* category. New observations with petal length greater than or equal to 2.45 will be allocated to either the *versicolor* or *virginica* group depending on their petal width.

The above decision tree also allows us to effectively see how the decision tree partitions the data. The data is initially partitioned into two groups based on one measurement and one side of the tree is then partitioned into a further two groups based on a second measurement.

We may also print the model to examine exactly how many elements are allocated to each classification:

```
print(decisiontree,digits=3)
```

```
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.3333 0.3333 0.3333)
  2) Petal.Length< 2.45 50   0 setosa (1.0000 0.0000 0.0000) *
  3) Petal.Length>=2.45 100  50 versicolor (0.0000 0.5000 0.5000)
    6) Petal.Width< 1.75 54   5 versicolor (0.0000 0.9074 0.0926) *
    7) Petal.Width>=1.75 46   1 virginica (0.0000 0.0217 0.9783) *
```

Figure 4: *The print function allows us to examine exactly how many elements are allocated to each species.*

The output begins its analysis by partitioning the data using the variable with the highest association with the response variable. $R$ continues to partition the data by the variable with the next highest association with the response variable until some stopping criterion is met. Our trees grow from a root, with the algorithm deciding, at each node, the most effective way of partitioning the data further. From the above output, we can assess the performance of our decision tree. In our data, the variable with the highest association with species is petal length, which is why $R$ begins by partitioning the data based on this measurement.

The output tells us that, for the 150 total observations, partitioning the data by petal length is the decision which has the largest association with the response variable. When the tree partitions the data by this variable, we see that all 50 elements of the *setosa* species are partitioned into the $Petal.Length < 2.45$ category. Conversely, all 100 elements from the other species are categorised into the $Petal.Length \geq 2.45$ category. This means that our first decision node splits the dataset into elements of the *setosa* species and those not in the *setosa* species. Next, $R$ partitions the observations by petal width. We see that 54 elements are partitioned into the $Petal.Width < 1.75$ category. 49 of these observations belong to the *versicolor* species. This means that 5 elements from species which are not *versicolor* are classified into the $Petal.Width < 1.75$ category. $R$ provides us with a probability that an observation with a petal width less than 1.75 belongs to the *versicolor* species. This probability is 0.9074. Similarly, $R$ informs us that 46 observations belong to the $Petal.Width \geq 1.75$ category, all but 1 of these observations belong to the *virginica* species. $R$ also provides us with the probability that an observation with a petal width greater than or equal to 1.75 belongs to the *virginica* species. This probability is 0.9783.

A simpler way to examine the classifications and misclassifications of our data is to create a *confusion matrix*. This matrix shows the number of each observation that were correctly identified and the nature of any misidentified observations:

```
p<-predict(decisiontree,type="class")
```

```
confusionMatrix(p,iris$Species)
```

```
            Reference
Prediction   setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         5
  virginica       0          1        45
```

Figure 5: *The columns in this matrix correspond to the number of each observation and the rows correspond to how the observations were classified by our model. The elements in the main diagonal correspond to correct predictions.*

From the above matrix, we see that 1 observation from the *versicolor* species that was incorrectly identified as a *virginica* species. Similarly, 5 observations from the *virginica* species were misclassified as *versicolor*. We can assess our model's performance and see where our model fails. We may calculate the effectiveness of our model by calculating:

$$\textit{Effectiveness(\%)} = \frac{\textit{Correctly classified observations}}{\textit{All observations}} \times 100$$

For our *iris* model, the effectiveness is 96%.

One important consideration when creating our decision tree is choosing where our tree should split points.

For a decision tree with a continuous variable (regression tree), we define our cut-off point as the point where the *residual sum of squared error* (RSS) is minimised across the samples within the subpartition[4]. However, for a decision tree with categorical or discrete variables (classification trees), the cut-off point is defined as the point where the partitioning will ensure that the populations in the subpartitions are as homogenous as possible. For classification trees, we may calculate a measure of this homogeneity using either the *Gini* index or the *entropy*. We define the Gini index as:

$$G = \sum_{k=1} p_{mk}(1 - p_{mk})$$

Where $p_{mk}$ represents the proportion of observations in the $m$th region that are from the $k$th class[5]. We may also use the entropy:

$$E = -\sum_{k=1} p_{mk} \log_2(1 - p_{mk})$$

Both the Gini index and entropy take values between 0 and 1, where 0 represents absolute homogeneity (or node purity), and 1 represents absolute heterogeneity (or node impurity). Since $R$ uses the Gini index as its default (and entropy is slightly slower due to the calculation of logarithms)[6], we favour using the Gini index to calculate where the tree should split.

We next examine how to use our decision tree to predict the species of new observations. Suppose we take a further 6 samples with these values:

| New Sample | Sepal Length | Sepal Width | Petal Length | Petal Width |
|---|---|---|---|---|
| 1 | 5.0 | 3.2 | 1.4 | 0.2 |
| 2 | 5.2 | 3.5 | 1.5 | 0.2 |
| 3 | 6.7 | 3.2 | 4.6 | 1.5 |
| 4 | 5.4 | 2.7 | 3.6 | 1.2 |
| 5 | 6.1 | 3.0 | 5.6 | 4.4 |
| 6 | 6.1 | 3.2 | 5.3 | 2.1 |

Figure 6: *Here we see a table of our new samples for classification.*

We construct these new samples into a data frame in $R$ and use our decision tree to predict their species:

```
newsamples<-data.frame(Sepal.Length=c(5.0,5.2,6.7,5.4,6.1,6.1),
Sepal.Width=c(3.2,3.5,3.2,2.7,3.0,3.2),
Petal.Length=c(1.4,1.5,4.6,3.6,5.6,5.3),
Petal.Width=c(0.2,0.2,1.5,1.2,4.4,2.1))
Predicted.Species<- decisiontree %>% predict(newsamples,"class")

cbind(newsamples,Predicted.Species)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Predicted.Species
1          5.0         3.2          1.4         0.2            setosa
2          5.2         3.5          1.5         0.2            setosa
3          6.7         3.2          4.6         1.5        versicolor
4          5.4         2.7          3.6         1.2        versicolor
5          6.1         3.0          5.6         4.4         virginica
6          6.1         3.2          5.3         2.1         virginica
```

Figure 7: *The final column gives us the predicted species for each observation based on our decision tree.*

## 3.2 Decision trees on a different dataset: *banknotes*

We now explore how decision trees can be used on a larger dataset. For our analysis, we will use the *banknote authentication* dataset, available through the

*University of California, Irvine's* dataset repository[7]. This dataset consists of measurements taken on genuine and counterfeit banknotes. The data is taken from photographs of the banknotes and consists of 4 measurements based on these images: *variance of wavelet transformed image*, *skewness of wavelet transformed image*, *kurtosis of wavelet transformed image* and *entropy of image*. The final variable is a Bernoulli variable (0 = *genuine*, 1 = *counterfeit*). The dataset consists of 1372 observations, 762 of which are on genuine banknotes and 610 of which are counterfeit. To simplify the plotting of the principal components, I altered the values for the authenticity variable from 0 and 1 to *genuine* and *counterfeit* respectively.

```
head(banknote.data)
```

| | Variance | Skewness | Kurtosis | Entropy | Authentic |
|---|---|---|---|---|---|
| 1 | 3.62160 | 8.6661 | -2.8073 | -0.44699 | Genuine |
| 2 | 4.54590 | 8.1674 | -2.4586 | -1.46210 | Genuine |
| 3 | 3.86600 | -2.6383 | 1.9242 | 0.10645 | Genuine |
| 4 | 3.45660 | 9.5228 | -4.0112 | -3.59440 | Genuine |
| 5 | 0.32924 | -4.4552 | 4.5718 | -0.98880 | Genuine |
| 6 | 4.36840 | 9.6718 | -3.9606 | -3.16250 | Genuine |

Figure 8: *The first 6 observations from our banknotes dataset.*

We plot the first two principal components to give us an idea of the structure of the data:

```
banknotes.pca <- prcomp(banknote.data[,-5])
fviz_pca_ind(banknotes.pca,geom.ind="point",
col.ind=banknote.data$Authentic,addEllipses = TRUE,
xlab="Principal component 1",ylab="Principal Component 2",
title="Plot of first two principal components on banknotes dataset")
```

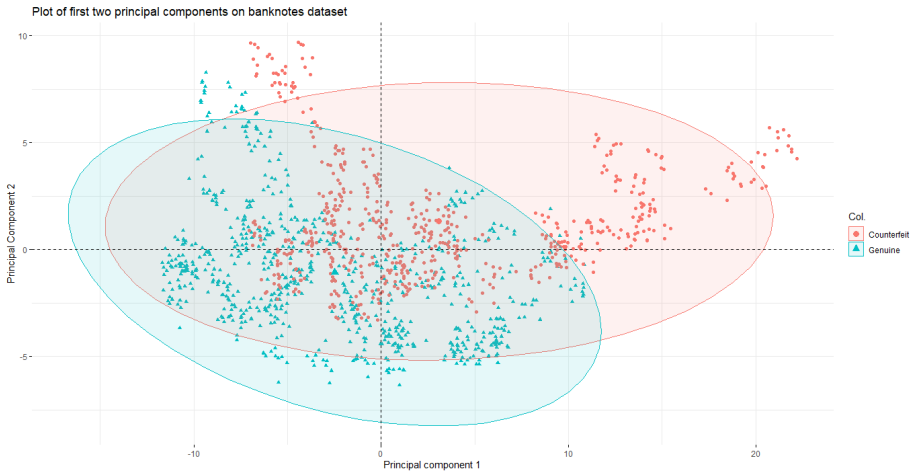


Figure 9: *A plot of the first two principal components from the banknotes dataset with the data clustered by the authenticity of the banknote. We see a significant overlap between the two clusters which implies that a more complicated decision tree may be required to categorise the data.*

We partition our data into a training set containing 75% of the datapoints and a test set which contains 25% of our datapoints. As the names suggest, the training set helps to build the predictive model and the test set helps us evaluate the effectiveness of the model.

```
set.seed(999)
samplesize <- sample.int(n = nrow(banknote.data),
size = floor(.75*nrow(banknote.data)), replace =F)
trainingset <- banknote.data[samplesize, ]
testset  <- banknote.data[-samplesize, ]
```

We create a decision tree based on our training set from the *banknotes* dataset:

```
decisiontreebank<-rpart(Authentic~.,trainingset)
par(xpd=NA)
plot(decisiontreebank,
main="Classification Of Authenticy for Banknotes")
text(decisiontreebank,cex=0.7)
```

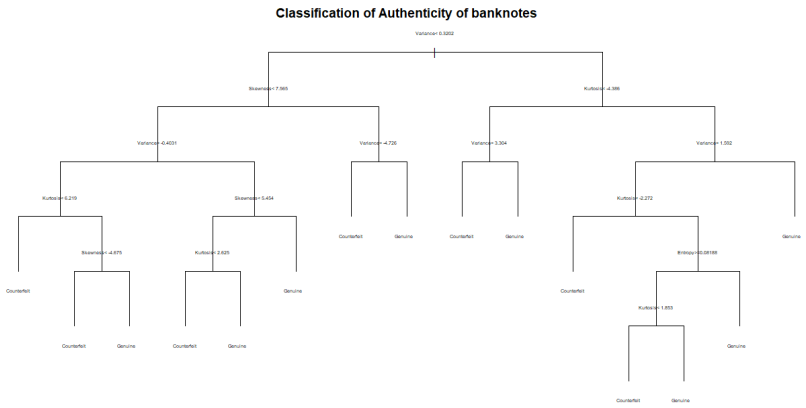


Figure 10: *The decision tree for counterfeit banknote detection. This tree is much more intricate than the decision tree for iris species.*

We test the effectiveness of our decision tree by applying it to the test set:

```
pbank<-decisiontreebank %>% predict(testset,type="class")
confusionMatrix(pbank,testset$Authentic)
```

```
            Reference
Prediction    Counterfeit Genuine
  Counterfeit         155      12
  Genuine               5     171
```

Figure 11: *Our decision tree has an accuracy of around 95%: a very good performance. It did, however, misclassify 5 counterfeit banknotes as genuine, and 12 genuine banknotes as counterfeit.*

We may wonder what the optimal size of the decision tree is. A tree with too few branches may be inadequate in capturing the important information about the dataset. A tree with too many branches, however, risks overfitting the data and may therefore give poor predictions for new datapoints. One technique which reduces the size of the decision tree by removing branches which do not perform effective partitioning is called *pruning*. Pruning aims to reduce the complexity of the full decision tree while still performing comparatively well. If a partitioning is not particularly effective in improving the model, we may wish to discard this partitioning. The *rpart* package provides a complexity parameter, which penalises the tree for being overcomplicated. The *complexity parameter* takes default value 0.01 and is inversely proportional to the size of the tree. We wish to take a value for this parameter which maximises cross-validation accuracy[8].

To examine whether we should prune our tree, we assess the complexity parameter against the *RMSE* (root mean square error). If pruning our tree reduces the error in our model, then we should do it.

For this analysis, we use the *trControl* function to apply 10-fold cross validation:

```
decisiontreebank2<-train(Authentic~.,data=trainingset,
method="rpart",trControl=trainControl("cv",number=10),
tuneLength=10)
decisiontreebank2
```

```
CART

1029 samples
   4 predictor
   2 classes: 'Counterfeit', 'Genuine'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 926, 926, 927, 926, 926, 926, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.00000000  0.9679421  0.9350048
  0.07358025  0.8940796  0.7818940
  0.14716049  0.8493908  0.6970762
  0.22074074  0.8493908  0.6970762
  0.29432099  0.8493908  0.6970762
  0.36790123  0.8493908  0.6970762
  0.44148148  0.8493908  0.6970762
  0.51506173  0.8493908  0.6970762
  0.58864198  0.8493908  0.6970762
  0.66222222  0.7248144  0.3988457

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.
```

Figure 12: *This output demonstrates how increasing the complexity parameter of the model will affect the accuracy.*

We may also examine a plot of how pruning would affect the accuracy of our model:

```
plot(decisiontreebank2)
```

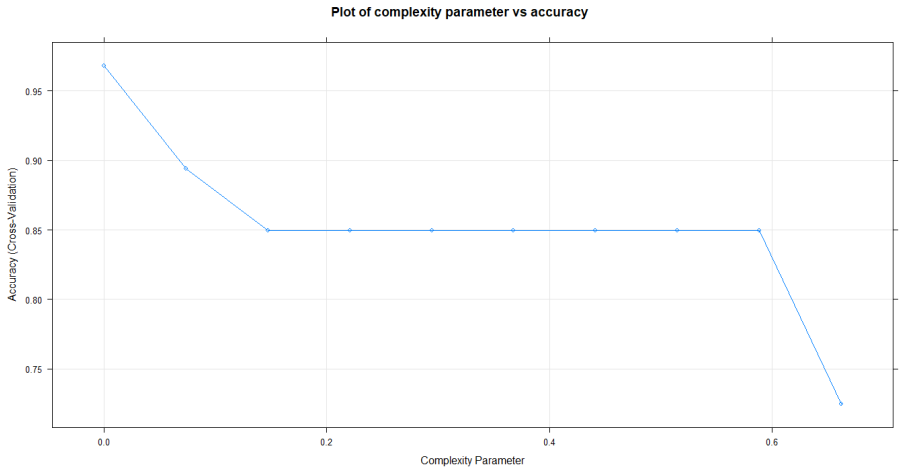


Figure 13: *A plot of the previous output, showing how the cross-validation accuracy of our model decreases as computational complexity increases. We conclude that pruning will cause a sharp decrease in the accuracy of our model.*

*R* will print the recommended pruning:

```
decisiontreebank2$bestTune
```

```
  cp
1  0
```

Figure 14: *This corresponds to no pruning at all.*

## 3.3 Conclusion on decision trees

Here we have discussed how decision trees can be used to categorise variables into subpartitions based on certain characteristics. We have examined plots of some trees on the *iris* dataset to gain an insight into how they achieve the partitioning of our datapoints into categories. We printed the models to examine the statistics behind the partitioning and to see the proportion of correctly and incorrectly categorised variables in each partition. We also created a confusion matrix to analyse where correct and incorrect classifications were made. We also discussed the metrics through which the decision of where to cut the tree was based: the Gini-index and entropy. Upon the creation of 6 new samples for our simple *iris* dataset, we asked our model to categorise these new observations based on the decision tree. Next, we examined a more complex dataset based on the measurements of banknotes to determine which banknotes were genuine and which were counterfeit. After splitting the entire *banknotes* dataset into

a training set (75% of the datapoints) and a test set (25% of the datapoints), we created another, more complex, decision tree to determine the authenticity of banknotes and assessed the effectiveness of our prediction model. We found that our *banknotes* model performed relatively effectively, accurately predicting banknote authenticity with around a 95% accuracy rate. In the next section, we will determine whether using a random forest approach will improve the effectiveness of our *banknotes* dataset predictions.

# 4 Random forests

As the name suggests, random forest methods extend the idea of decision trees by creating a model which combines many decision trees into a 'forest'. Random forest algorithms hedge the instability that we see from decision trees by taking information from an ensemble of trees, instead of a single tree. Single decision trees are often unstable and sensitive to outliers and noise however random forests are more robust to these changes, since they take information from an ensemble of trees. The word 'random' refers to the randomness used by the algorithm in selecting a dataset's observations and variables. The randomness of the algorithm ensures its robustness and protects the model against the overfitting often experienced when using single decision trees.

The random forest algorithm works by applying *bootstrap aggregating* (building multiple decision trees by averaging bootstrapped data from random variables within some training dataset) to the dataset. Replacement is allowed in the bootstrap aggregating, meaning that an observation may be selected multiple times. The algorithm next employs *bootstrap sampling* to the dataset variables at each split so that a random sample of variables are selected as potential split variables. Since the dataset and variables are randomly selected, each tree consists of different observations and variables. We consider an ensemble of these trees to create our random forest. Finally, when using a random forest as a single model, we afford equal weight to each tree in the forest, therefore if the majority of trees predict an outcome, we expect this outcome[9].

## 4.1 Random forests on a dataset: *banknotes*

We can use the *randomForest* function, available from the *randomForest* package in *R* to determine the authenticity of banknotes based on measurements of images. For this analysis, we examine whether implementing the random forest algorithm on our test dataset yields better results than the 95% accuracy scored by the single decision tree approach. We use the *randomForest* function to determine the predicted authenticity of the banknotes in the test set[10]. We set the *data* argument to *trainingset* and allow *R* to assess the importance of predictors by setting the argument *importance = TRUE*. The function also allows us to specify the number of trees to include in our forest. If we do not

specify a number, $R$ will estimate an appropriate number of trees to include based on the complexity of the dataset:

```
rfbank<-randomForest(Authentic~ ., data = trainingset,
importance = TRUE)
```

We may examine the importance of each variable in our random forest using the *importance* function, also available through the *randomForest* package. The output of this command shows the mean decrease in accuracy and mean decrease in the Gini-index for each variable in our dataset:

```
          Counterfeit   Genuine MeanDecreaseAccuracy MeanDecreaseGini
Variance   128.75525 104.38226            141.43505        275.82598
Skewness    89.74369  54.43322             89.75263        124.62361
Kurtosis    62.01821  49.39192             75.14021         78.08410
Entropy     24.19208  18.60292             28.95879         29.72271
```

Figure 15: *The importance function allows us to examine the importance of each variable in creating our model.*

The 'mean decrease accuracy' column gives values corresponding to the average decrease of accuracy when this variable is not included in the model. The output for the *importance* function lists the variables in order from the variable whose exclusion would result in the largest decrease in accuracy to the variable whose exclusion would result in the smallest decrease in accuracy. The function also returns the mean decrease in the node impurity for each variable from splits over that variable[11]. From the above output, we see that the most important variable in determining the outcome of our model is *variance*, followed by *skewness*, *kurtosis* and *entropy*. We now examine our random forest:

```
rfbank
```

```
Call:
 randomForest(formula = Authentic ~ ., data = trainingset, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 1.07%
Confusion matrix:
           Counterfeit Genuine class.error
Counterfeit         456       4 0.008695652
Genuine               7     562 0.012302285
```

Figure 16: *The randomForest function displays the number of trees used in the algorithm and displays the confusion matrix on the training dataset.*

From the above output, we see that the *randomForest* function creates 500 random decision trees on which to base its output. We calculate the effectiveness of our model to be approximately 98.93%. A substantial improvement from our single decision tree which had an accuracy score of around 95%. Having created our random forest model, we test the model using the 25% test set:

```
pbankrf<-rfbank %>% predict(testset,type="class")
```

The above code creates a prediction for the authenticity of the test set, based on the model created by the training set. We examine a subset of the output below:

```
pbankrf
```

```
     3         7         8        12        14        19        22
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
    25        26        27        39        53        55        59
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
    60        65        69        74        82        83        84
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
    88        92       105       109       114       119       122
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
   123       125       126       129       130       131       132
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
   140       141       154       155       158       161       166
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
   168       193       195       197       198       205       207
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
   208       210       212       215       216       217       219
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
   228       233       235       237       241       243       246
Genuine   Genuine   Genuine   Genuine   Genuine   Genuine   Genuine
```

Figure 17: *The output from this command lists all the elements in the test set and gives a prediction of their authenticity.*

The model predicts that 194 of the 343 banknotes in the test set are predicted to be genuine and the remaining 149 banknotes are predicted to be counterfeit. To assess how accurately this reflects the known authenticity of the *banknotes* test set, we again use a confusion matrix:

```
confusionMatrix(pbankrf,testset$Authentic)
```

```
Confusion Matrix and Statistics

            Reference
Prediction    Counterfeit Genuine
  Counterfeit         149       0
  Genuine               1     193
```

Figure 18: *Confusion matrix of our random forest when applied to the test set of the banknotes dataset. This model appears to perform extremely well, with only 1 misclassification out of 343 observations.*

The confusion matrix informs us that 149 banknotes in the test set were correctly identified as counterfeit and 193 banknotes were correctly identified as genuine. The model only misclassified 1 datapoint: a counterfeit banknote which was misidentified as genuine. We calculate the accuracy of this model as approximately 99.7%.

The confusion matrices for both the single decision tree and random forest methods allow for easy comparison between the two models. When applied to the test set, the single decision tree made 17 misclassifications in total: 12 genuine banknotes which were misclassified as counterfeit and 5 counterfeit banknotes which were misclassified as genuine. As previously mentioned, our random forest model only made 1 single misclassification, identifying 1 counterfeit banknote as genuine. We also compare the single decision tree model's accuracy score of 95% against the random forest model's accuracy score of 99.7%. When employing a random forest model, the rate of misclassification decreases drastically, and the predictions are far more accurate when compared to the observed values. Because of this increase in accuracy, we endorse the use of the random forest model in classifying future observations over the use of single decision trees.

### 4.2 Conclusion on random forests

In this section, we have discussed the random forest method of machine learning, which takes an ensemble of randomly created decision trees to provide a better insight into the data. Random forests mitigate the overfitting problem associated with single decision trees by using information from a large number of trees with random samples and variables. Here we have applied the method of random forests to a large dataset to perform a classification task on the authenticity of banknotes. Using a package available in $R$, we created a random forest of 500 trees to explain the data in our training dataset which represents 75% of the datapoints in our entire dataset. We next applied this model to the remaining 25% of our dataset to measure its effectiveness.

After examining which variables were the most important in increasing our model's accuracy, we again used a confusion matrix to examine the performance of our random forest model. We found that the model outperformed the single decision tree model with an accuracy of approximately 99.7%. From this analysis, we observe that random forests can be an extremely effective method of performing classification tasks on previously unseen data. The effectiveness of the model does, however, depend on the amount of data available from which to build the model. The larger the amount of data from which to base the model, the more effective the model will be. The effectiveness of our model relies, in part, on the large amount of data available in the training dataset. Random forests, like other machine learning algorithms, require large amounts of data to perform effectively. In the next section, we explore a different supervised machine learning technique, artificial neural networks.

## 5 Artificial neural networks

Artificial neural networks are computational models inspired by the biology of the human brain. In human biology, signals travel through synapses which are received by neurons. These neurons are activated if these signals exceed some

threshold. Upon activation, signals may be sent to other neurons as part of a network. In a mathematical setting, these synapses are modelled as inputs and the strength of the signals corresponds to a weight. At the neuron, some function determines whether the neuron should be activated. Finally, we calculate the output of the neuron and send this output to other parts of the artificial neural network[12].

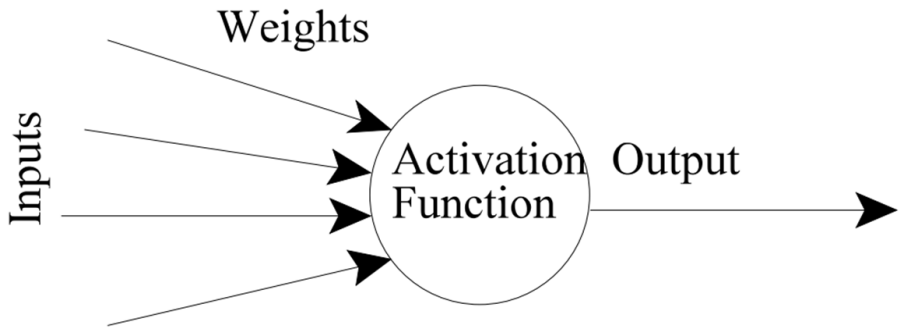


Figure 19: *Diagram of a neuron, showing the inputs, weights, activation function and output*[12].

Clearly, larger weights will result in a stronger input for the activation function and the input values for neurons will vary based on these weights. We may represent each input multiplied by its weight:

$x_1 \rightarrow x_1 * \ w_1$
$x_2 \rightarrow x_2 * \ w_2$
...
$x_n \rightarrow x_n * \ w_n$

Before using our activation function,we sum all the weighted inputs, with some bias parameter $\beta$:

$$\beta + \sum_{i=1}^{n} (x_i * \ w_i)$$

This sum is inputted into the activation function:

$$y = f\left(\beta + \sum_{i=1}^{n} (x_i * \ w_i)\right)$$

This determines whether to activate the neuron and also assigns a value to the output.

We can combine hundreds or thousands of artificial neurons to create an artificial neural network, and we can train this network to perform classification tasks by finding algorithms which adjusts weights within the network to give a desired outcome. The process of training the network is as follows:

1. Initialise weight values.
2. For each data point in some training set:
   (a) Calculate the output with previous weights.
   (b) Update the weight values based on the known outcome.
3. Repeat step 2 until some stopping criterion is met. Generally we stop when every datapoint in the training set has been used or some prespecified number of iterations has been reached[13].

An example of an artificial neural network generated using $R$ can be seen below:

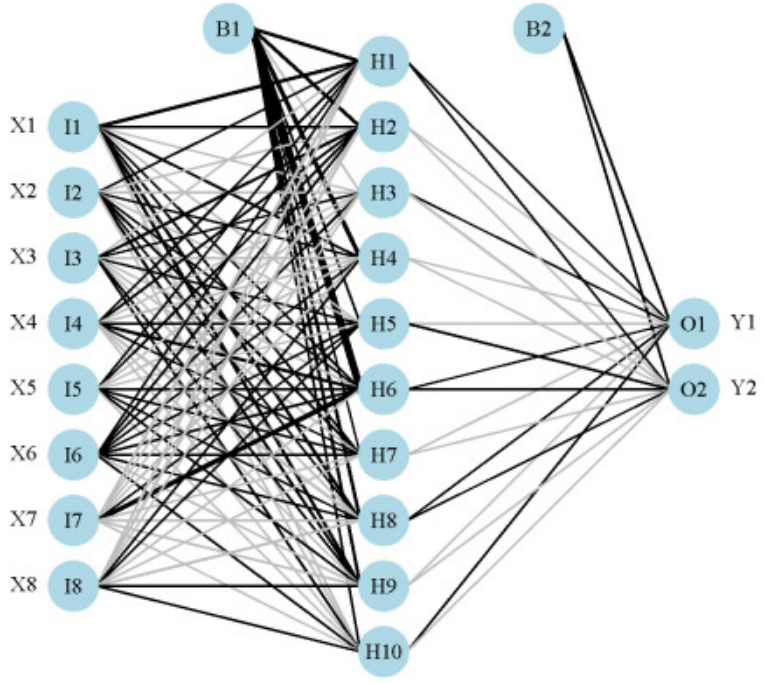


Figure 20: *An example of an artificial neural network. Here the 'I' neurones correspond to the input section of the network, the 'H' neurones correspond to the middle, "hidden" section of the network, the 'O' neurones correspond to the output of the network and the 'B' neurones correspond the biases added at certain parts of the network.*[14]

The input and output sections are specified by the explanatory and response variables respectively. The "hidden" section of the network is not input by the statistician and is usually the most complex component of the artificial neural network. Because of the complexity of the "hidden" section, the artificial neural network method is often referred to as a 'black box' approach, as studying the structure of the network does not provide an insight into the link between the weights and functions within the network.

## 5.1 Artificial neural networks on a dataset: *banknotes*

To fit an artificial neural network to our dataset, we must install the *R* package *nnet*, which contains the function *nnet*. This function fits a single-hidden-layer artificial neural network to the data. The main arguments which we specify are: *size* (which specifies the number of neurones in the "hidden" section), and

*maxit* (which specifies the maximum number of iterations). Again, we perform the analysis on the test set to assess the performance of this alternative machine learning technique against the rudimentary decision trees method and random forest method. It is worth noting that we may wish to tweak the parameters of size and maxit based on the structure of our data. Increasing the number of neurones in the network increases the speed that the network converges to an optimal solution but at the expense of computational complexity. Similarly, the number of iterations should be set to ensure that enough iterations run for convergence to an optimal solution. The algorithm terminates either after the maximum number of iterations has been reached or upon the return of an optimal fitting criterion[15].

```
banknnet<-nnet(Authentic~.,data=trainingsetbank,size=10,maxit=100)
banknnet
```

```
# weights:  61
initial  value 637.124268
iter  10 value 14.193980
iter  20 value 0.267400
iter  30 value 0.003733
iter  40 value 0.000317
final  value 0.000087
converged
> banknnet
a 4-10-1 network with 61 weights
inputs: Variance Skewness Kurtosis Entropy
output(s): Authentic
options were - entropy fitting
```

Figure 21: *The output of the banknnet command, we see that the algorithm converged after around 50 iterations with the fitting criterion value of 0.000087. The second part of the output outlines the nature of the artificial neural network, including the number of weights, the structure of the neurones and the method of fitting used- here it was 'entropy fitting'.*

We may view a plot of our artificial neural network by installing the *R* package *NeuralNetTools* and using the *plotnet* command in *R*:

```
install.packages("NeuralNetTools")
library(NeuralNetTools)
plotnet(banknnet,circle_cex=3)
```

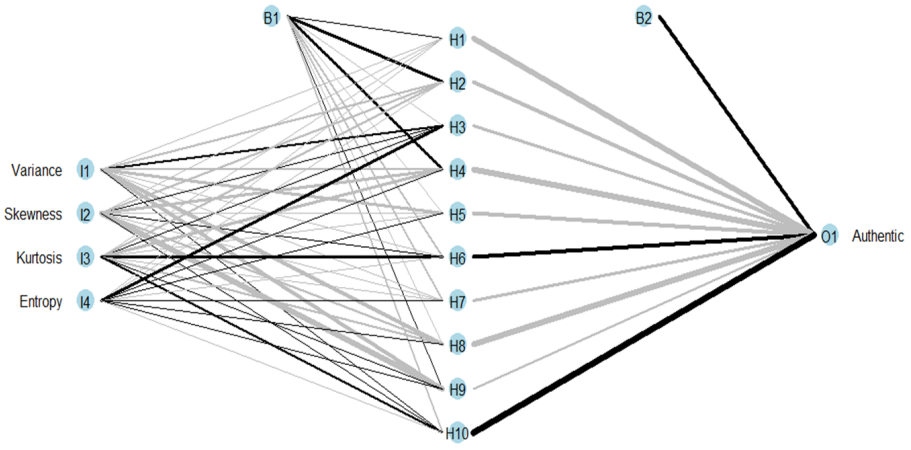


Figure 22: *A visualisation of our banknotes artificial neural network. By default, the plotnet function plots positive weights as black lines and negative weights in grey. Furthermore, the thickness of the lines are proportional to the magnitude of their weights*[16].

Like with random forests, we can use *R* to assess the importance of each explanatory variable in the results of the response variable. Garson (1995) deconstructs model weights to calculate the *relative importance* of each explanatory variable to the response variable[16]. This method identifies the weights of all input neurones through the hidden layer for all explanatory variables. We use the *R* command *garson*, also available through the *NeuralNetTools* package, to assess the importance of our variables:

```
garson(banknnet,bar_plot=F)
```

```
          rel_imp
Variance 0.2961335
Skewness 0.2540467
Kurtosis 0.2684899
Entropy  0.1813300
```

Figure 23: *Values for the relative importance of each explanatory variable.*

We may also create a histogram of the *relative importance* of each explanatory variable:

```
garson(banknnet)
```

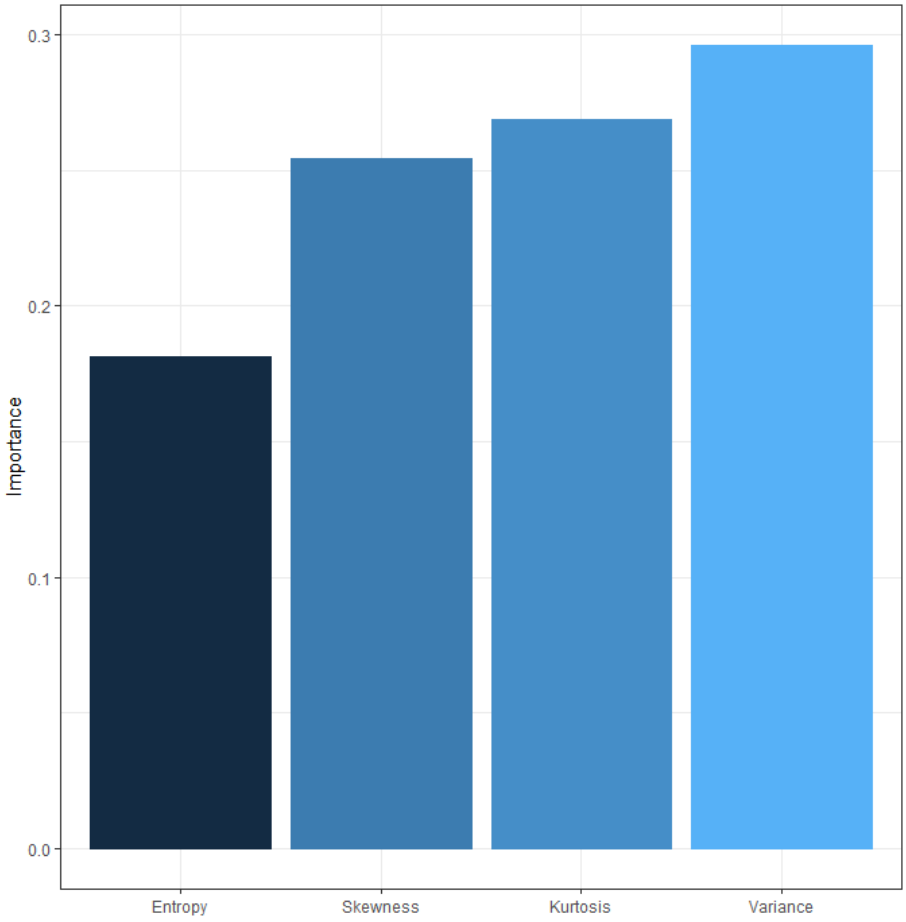


Figure 24: *A histogram which allows us to visualise the importance of each explanatory variable.*

We see that *variance* is the most important explanatory variable in our data, followed by *kurtosis*, *skewness*, and *entropy*. Interestingly, these results are different from the importance measures from the random forest method, which ranked *skewness* as more important than *kurtosis*.

We may also perform *sensitivity analysis* on our artificial neural network using *Lek's profile* method. This method plots the predicted response value for a range of values of each separate variable[16].

```
lekprofile(banknnet,group_show=T)
```

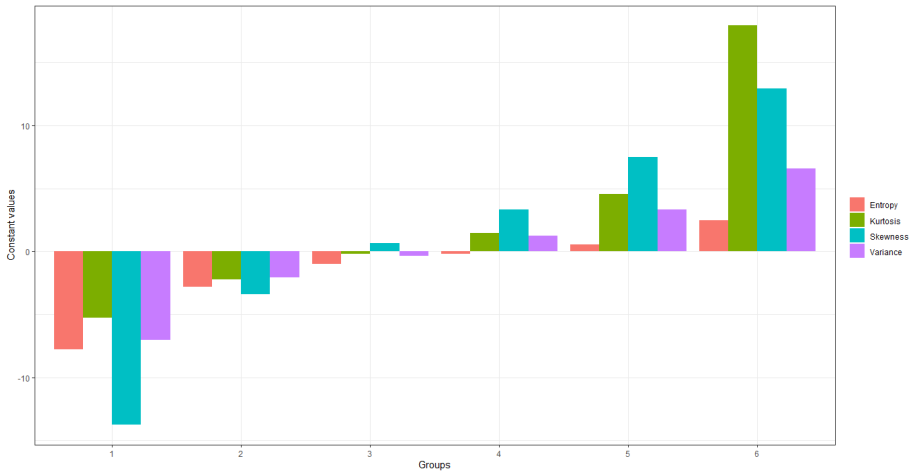


Figure 25: *The groupings for each variable at the 0th, 20th, 40th, 60th, 80th and 100th percentiles.*

```
lekprofile(banknnet)
```

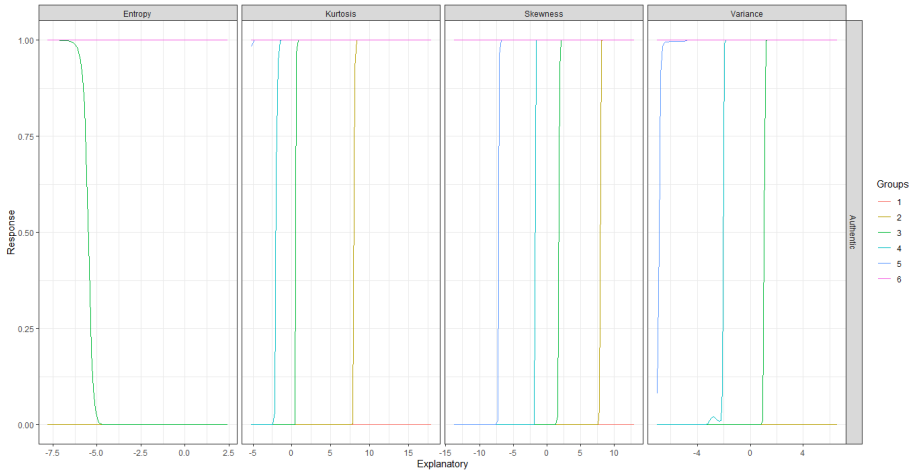


Figure 26: *Results of the Lek's profile method of sensitivity analysis. Again, the 6 groups represent the 0th, 20th, 40th, 60th, 80th and 100th percentiles.*

The *Lek's profile* method allows us to characterise the relationships between explanatory variables within the artificial neural network. Here we see how the response variable changes depending on the explanatory variable percentile being evaluated. We now consider the effectiveness of our artificial neural network model:

```
pbanknnet<-as.factor(banknnet %>% predict(testsetbank,type="class"))
pbanknnet
```

```
 [1] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
 [8] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[15] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[22] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[29] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[36] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[43] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[50] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[57] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[64] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[71] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[78] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[85] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[92] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
[99] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
106] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
113] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
120] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
127] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
134] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
141] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
148] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
155] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
162] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
169] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
176] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
183] Genuine     Genuine     Genuine     Genuine     Genuine     Genuine     Genuine
190] Genuine     Genuine     Genuine     Genuine     Counterfeit Counterfeit Counterfeit
197] Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit
204] Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit
211] Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit
218] Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit Counterfeit
```

Figure 27: *This command returns a list of the predicted authenticity of each banknote in the banknote test dataset we created earlier.*

It is worth noting that, unlike with the *randomForest* function, the ouput of the *nnet* function is a character string since we have categorical data. Therefore, for easier analysis, we convert the output into a factor. We continue our analysis by again creating a confusion matrix to assess the effectiveness of our model at predicting the authenticity of the banknotes:

```
confusionMatrix(pbanknnet,testsetbank$Authentic)
```

```
            Reference
Prediction    Counterfeit Genuine
  Counterfeit         150       0
  Genuine               0     193
```

Figure 28: *The confusion matrix resulting from implementing the artificial neural network method on the banknotes test dataset. This model correctly predicted the authenticity of every banknote.*

The above confusion matrix shows that the artificial neural network method, when applied to the *banknotes* test dataset, performs extremely well. The

method correctly predicted the authenticity of all 343 banknotes in the test set. This model strongly outperforms the decision tree method and slightly outperforms the random forest method, therefore we endorse the use of an artificial neural network approach to the banknote authenticity problem.

### 5.2 Conclusion on artificial neural networks

In this section, we have discussed the artificial neural network approach to classification tasks and used it effectively on a dataset. Artificial neural networks mimic the structure of the brain in human biology by allocating weights to input values and passing these values through some activation function in a neurone. The output of this function then carries on to the next neurone in the network. We have examined the structure of these networks, which consist of the input variables at the beginning of the network, the hidden middle neurones, the bias values which are added into the network at each level, and finally the output variables of interest.

We next created an artificial neural network for our banknotes dataset, using the *R* package *nnet*. Upon creating this artificial neural network, we created a visual representation of the model and assessed the diagnostics to determine which explanatory variables were the most important in the decisions made by the model. Finally, we evaluated the effectiveness of the model by testing its performance on the *banknotes* test dataset. The model correctly predicted the authenticity of every banknote in the set, outperforming both the decision tree and random forest methods. In terms of accuracy, the artificial neural network model performed extremely well, and we would endorse this approach to ensure the most accurate predictions.

## 6 Comparison of machine learning techniques

Now we have discussed both random forests and artificial neural networks in detail, we compare the advantages and disadvantages of each method.

On our dataset, both machine learning techniques performed extremely effectively, scoring a very high accuracy for a classification test on our *banknotes* dataset. To determine which method performed best overall, we run both the random forest and artificial neural network models 10 times, each with a different random sample taken for the training and test datasets:

| Method: | | |
|---|---|---|
| | Random forest | Artificial neural network |
| Run | Accuracy score (%) | Accuracy score (%) |
| 1 | 99.71 | 100 |
| 2 | 99.13 | 100 |
| 3 | 99.71 | 100 |
| 4 | 99.71 | 100 |
| 5 | 99.42 | 100 |
| 6 | 99.42 | 100 |
| 7 | 98.83 | 100 |
| 8 | 98.83 | 100 |
| 9 | 99.42 | 98.83 |
| 10 | 99.42 | 100 |
| | | |
| Mean | 99.36 | 99.88 |

Figure 29: *Comparison table for 10 runs of both methods.*

On average, the artificial neural network method outperforms the random forest model. From our data, we see that the artificial neural network method mostly achieves a 100% accuracy score. The random forest approach, however, made at least 1 misclassification in each run. If we are selecting a model based on performance, without any considerations of robustness or comprehensibility, we would endorse the artificial neural network model, as it gives the best results for our dataset.

When dealing with big data, we often have data with a large number of datapoints with high dimensionality. The vastness of the data may result in the inclusion of outliers and extreme values. Because of the complexity of our data, any machine learning model should identify underlying relationships within the data and not necessarily only fit the data as well as possible (which usually results in overfitting). To prevent overfitting, both random forests and artificial neural networks employ different approaches. Random forests allow us to specify the number of decision trees in the forest. Adding more trees adds more protection against overfitting but at the expense of computational complexity. On the other hand, in artificial neural networks the number of neurones in the hidden layer affects the protection against overfitting at the expense of computational complexity. Compared with random forests, which hedge the instability caused by extreme inputs, artificial neural networks are sensitive to the input variables; inputting extreme values may increase the risk of deviations from the model[17].

The models also differ in terms of comprehensibility. As we have discussed, artificial neural networks present a 'black box' problem as they are extremely difficult to interpret. The hidden section of the artificial neural networks presents a complex layer of interactions between neurones and therefore artificial neural networks may not be suitable in situations where we need to understand how predictions have been allocated. Similarly, random forests are also very difficult to interpret, and they involve many complex, randomly created decision trees. Again, adding more trees helps to protect against overfitting, but makes the model more difficult to interpret.

Another important factor to consider when choosing a model is complexity. The training process of artificial neural networks is very computationally expensive as it creates a model over many iterations[17]. Furthermore, artificial neural networks perform best when exposed to an extremely large amount of training data. Random forests, however, often perform well even on a smaller dataset. Because of these considerations, we should consider the structure of our data and the amount of processing power available before deciding between random forests or artificial neural networks.

In studies of the effectiveness of the two methods, random forests outperformed artificial neural networks in 90% of classification tasks using datasets from the *UC Irvine* dataset repository[18]. In most cases, however, the performance of both methods are similar and it has been demonstrated that the performance of artificial neural networks increases with the size of the dataset being examined[17].

# 7 Conclusion

In this project, we have discussed two main machine learning techniques: random forests and artificial neural networks. To effectively demonstrate the basis of random forests, we created a simple decision tree for the *iris* dataset, which is freely available in *R*. Upon creating our simple decision tree model, we used it to classify existing and new observations by species. We next created a decision tree for a dataset which classified banknotes by authenticity and partitioned this dataset into a training and test set. We used the machine learning concept of a *confusion matrix* to assess the performance of the model. The confusion matrix displays how observations were classified by the model, compared with their actual known classification. On both datasets, the decision tree models did not perform especially well, and made several misclassifications, due to their tendency to overfit the data.

Because of this issue, we introduced the random forest unsupervised machine learning technique. Random forests hedge the instability from single decision tree models by taking information from a large number of randomly generated

trees using bootstrap sampling. We also used the $R$ command *importance* to assess the importance of each explanatory variable in the results given by the model. When the random forest method was applied to the *banknotes* dataset, we again used a confusion matrix to assess its effectiveness and saw a vast improvement on the quality of predictions. We therefore endorsed the use of random forests as an effective method for performing classification tasks.

Next, we discussed an alternative machine learning technique, artificial neural networks, which are also often used for classification tasks. The method is based on the structure of the human brain and consists of an input layer, a "hidden" middle layer, and an output layer. We discussed the structure of each neurone in the network, and how these neurones form a decision process. We also examined the *Garson* and *Lek's profile* diagnostics to again assess the most important explanatory variables in relation to the artificial neural network's output variable. Finally, we performed the method on our test data and found that it performed extremely well, usually scoring an accuracy rating of 100%.

In the next section, we discussed the advantages and disadvantages of both machine learning techniques. We examined the merits of each method in relation to performance, comprehensibility, and complexity. We found that, in our own example, the artificial neural network performed best however in general this model may be unsuitable due to its computational complexity and sensitivity to input variables. Conversely, the random forest model, which performed slightly worse in our example, is considered more effective than artificial neural networks on smaller datasets and is more suitable if less processing power is available. We found that, empirically, the random forest model outperformed the artificial neural network for most datasets within the *UC Irvine* datasets repository, however artificial neural networks were more effective for datasets with a larger number of observations. We concluded that a statistician should assess the complexity and structure of the data before deciding which method to use.

Machine learning techniques represent an exciting and innovative new approach to classification tasks and, as we have demonstrated, they can be used to perform these tasks extremely effectively. Furthermore, the ease with which machine learning techniques can be applied to a large number of problems makes them applicable to many areas within mathematics and statistics. Given the rate at which computers are improving, we imagine that machine learning techniques will be used effectively on increasingly complex datasets, with increasingly accurate predictions.

# References


[1] Gopinath Rebala, Ajay Ravi, Sanjay Churiwala (2019) *An Introduction to Machine Learning*, Cham, Switzerland: Springer Nature Switzerland. p. 1.

[2] Terry Therneau, Beth Atkinson, Brian Ripley (2019) *Package 'rpart': Recursive Partitioning and Regression Trees*, Available at: *https://cran.r-project.org/web/packages/rpart/rpart.pdf* (Accessed: March 2020).

[3] Ronald Fisher (1936) *The Use Of Multiple Measurements In Taxonomic Problems*, Annual Eugenics, 7, Part II, Blackwell Publishing, London. p. 179-188.

[4] Alboukadel Kassambara (2017) *Machine Learning Essentials: Practical Guide in R*, STHDA (*http://www.sthda.com*): p. 162.

[5] Karthik Ramasubramanian, Abhishek Singh (2019) *Machine Learning Using R: With Time Series and Industry-Based Use Cases in R*, 2nd edn., New York, NY, USA: Springer Science+Business Media. p. 344.

[6] Karthik Ramasubramanian, Abhishek Singh (2019). p. 345.

[7] Volker Lohweg (2013) *Banknote Authentication Data Set*, Available at: *http://archive.ics.uci.edu/ml/datasets/banknote+authentication* (Accessed: March 2020).

[8] Alboukadel Kassambara (2017). p. 166.

[9] Graham Williams (2011) *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, New York, NY: Springer Science+Business Media.

[10] Leo Breiman, Adele Cutler, Andy Liaw, Matthew Wiener (2018) *Package 'randomForest': Breiman and Cutler's Random Forests for Classification and Regression*, Available at: *https://cran.r-project.org/web/packages/randomForest/randomForest.pdf* (Accessed: March 2020).

[11] Alboukadel Kassambara (2017). p. 175.

[12] Carlos Gershenson (2001) *Artificial Neural Network For Beginners*, University of Sussex: Formal Computational Skills Teaching Package, COGS.

[13] Karthik Ramasubramanian, Abhishek Singh (2019). p. 430.

[14] Marcus W. Beck (2013) *Visualizing neural networks in R – update*, Available at: *https://beckmw.wordpress.com/2013/11/14/visualizing-neural-networks-in-r-update/* (Accessed: April 2020).

[15] Brian Ripley, William Venables (2020) *Package 'nnet': Feed-Forward Neural Networks and Multinomial Log-Linear Models*, Available at: *https://cran.r-project.org/web/packages/nnet/nnet.pdf* (Accessed: April 2020).

[16] Marcus W. Beck (2018) *Package 'NeuralNetTools': Visualization and Analysis Tools for Neural Networks*, Available at: *https://cran.r-project.org/web/packages/NeuralNetTools/NeuralNetTools.pdf* (Accessed: April 2020).

[17] Prof. Dr. Peter Roßbach (2018) *Neural Networks vs. Random Forests – Does it always have to be Deep Learning?*, Available at: *https://blog.frankfurt-school.de/wp-content/uploads/2018/10/Neural-Networks-vs-Random-Forests.pdf* (Accessed: April 2020).

[18] Manuel Fernandez-Delgado, Eva Cernadas, Senen Barro (2014) *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?*, Journal of Machine Learning Research, 15. pp. 3133-3181.