PH125.9x Capstone Project Part 2

Banknote Authentication

Jonathan Shiell

This is available on Github at: https://github.com/JonathanShiell/PH125.9x_Capstone-2

Introduction

Introduction to Dataset

The dataset was obtained from the UCI Machine Learning Depository via https://archive.ics.uci.edu/ml/datasets/banknote+authentication and consists of four continuous predictor variables, and one categorical output variable. The four predictor variables correspond to different attributes of the image histogram as described in Gillich and Lohweg (2010). The output variable is categorical with two separate categories, `0` and `1`. A category corresponds to documents printed using a particular method (Intaglio) that is generally only used for bank notes and other, similar (unregistered), financial instruments, whereas the other category corresponds to various forms of non-Intaglio print. The literature is unclear as to which value of y corresponds to Intaglio and which does not, and these will therefore be treated as equal factors. The intention is to develop a model that can accurately classify each and every item in the dataset based on the method of printing.

Variable Name	Variable Description	Variable Attributes
`x1`	Variance of Wavelet Transformed image	Continuous
`x2`	Skewness of Wavelet Transformed image	Continuous
`x3`	Curtosis of Wavelet Transformed image	Continuous
`x4`	Statistical Entropy of image	Continuous
`y`	Class (Printing Type)	Discrete, Integer

Missing Values:

A count of missing values for each variable is provided below:

	x1	x2	x3	x4	y
Missing values	0	0	0	0	0

It can be seen that there are no missing values.

Continuous variables

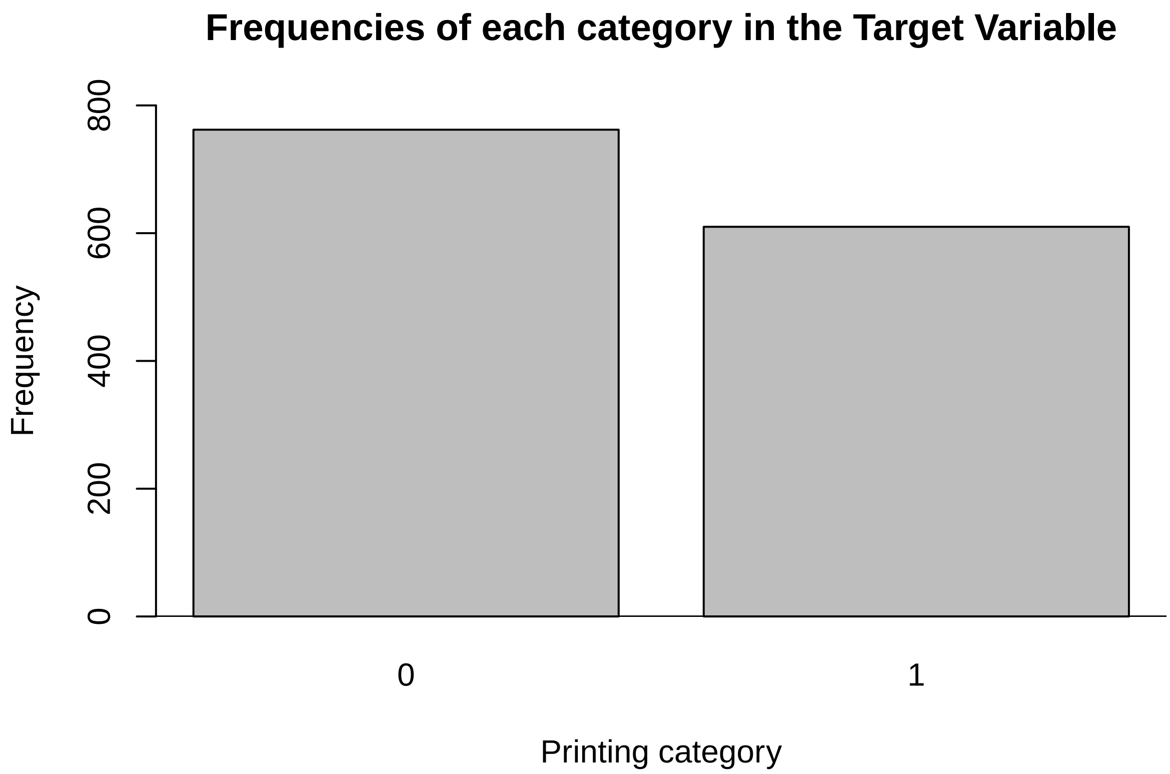
x1	x2	x3	x4
Min. :-7.0421	Min. :-13.773	Min. :-5.2861	Min. :-8.5482
1st Qu.:-1.7730	1st Qu.: -1.708	1st Qu.:-1.5750	1st Qu.:-2.4135
Median : 0.4962	Median : 2.320	Median : 0.6166	Median :-0.5867
Mean : 0.4337	Mean : 1.922	Mean : 1.3976	Mean :-1.1917
3rd Qu.: 2.8215	3rd Qu.: 6.815	3rd Qu.: 3.1793	3rd Qu.: 0.3948
Max. : 6.8248	Max. : 12.952	Max. :17.9274	Max. : 2.4495

Categorical Variables

The target variable y is categorical with values of 0 or 1, whose meaning is not explained in the description provided at the web address above. I will discuss the likely meaning of these variables in the Conclusions section below. The frequencies of each value of y are presented below:

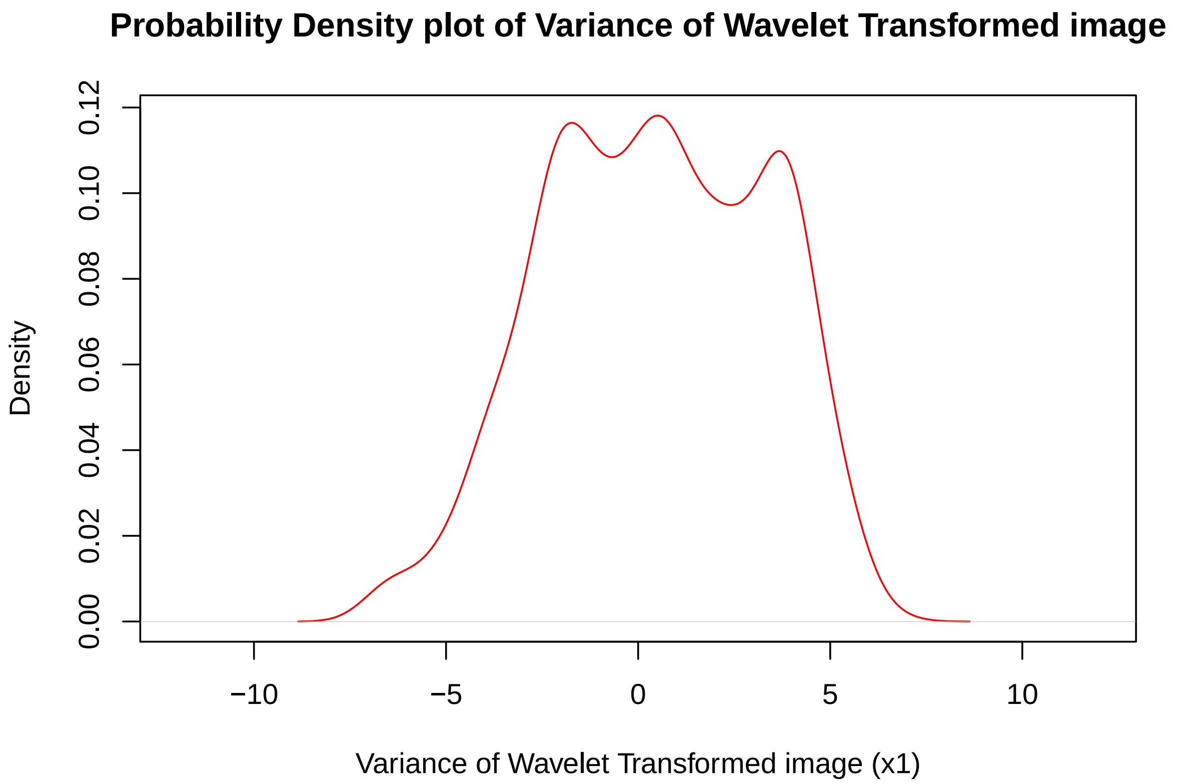
y	Frequency
0	762
1	610

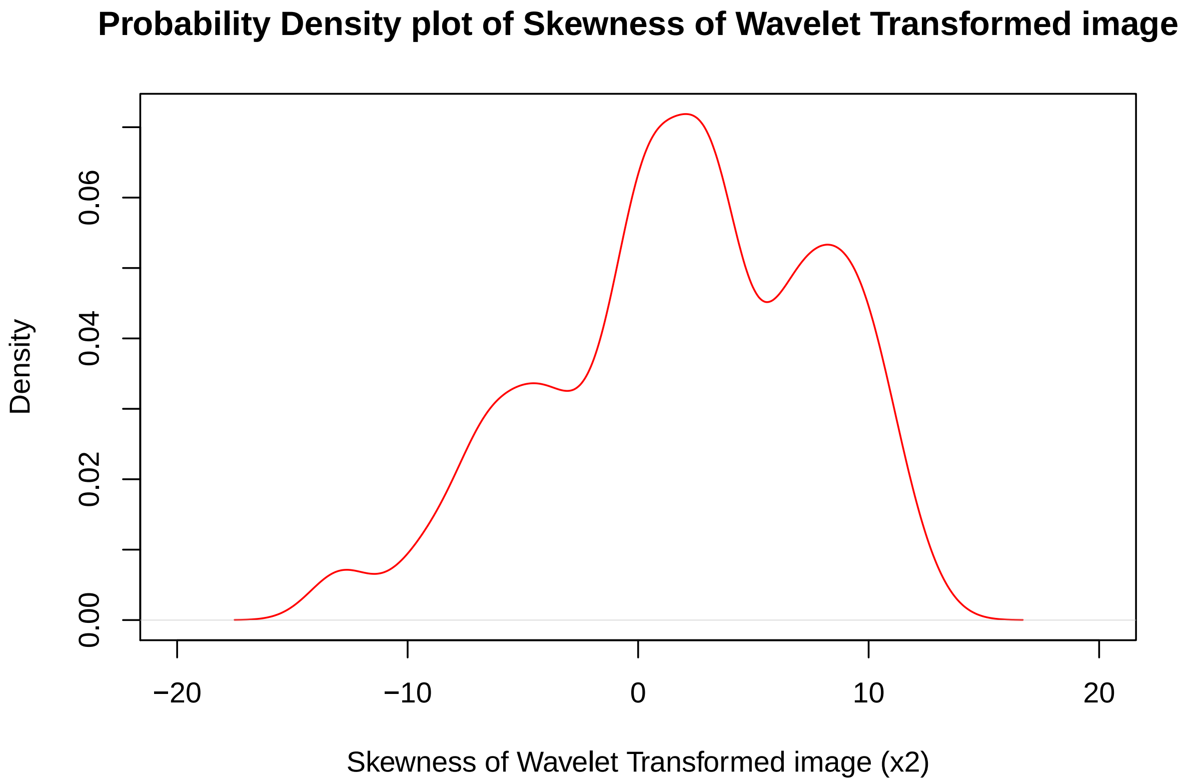
```
barplot(table(banknotes_factor$y),ylim=c(0,800),
        main = 'Frequencies of each category in the Target Variable',
        xlab = 'Printing category',
        ylab = 'Frequency')
abline(0,0)
```



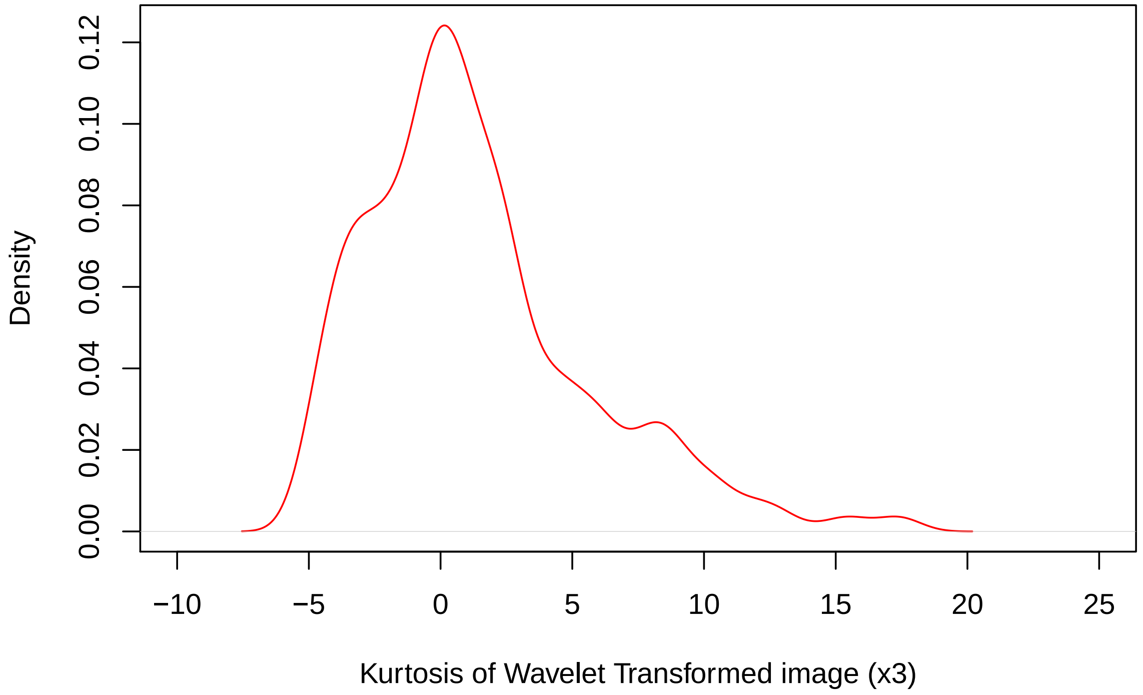
It can be seen that there is an approximately even distribution between the two categories, with slightly more observations in category 0 than category 1.

Density Plots of Continuous Variables

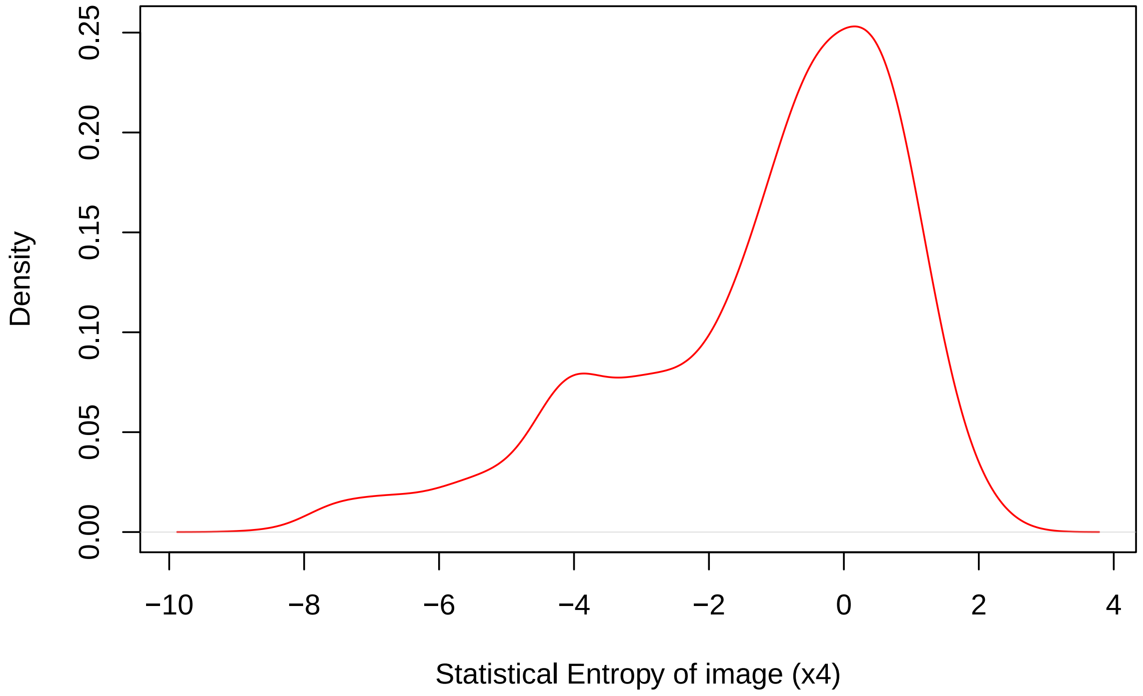




Probability Density plot of Kurtosis of Wavelet Transformed image



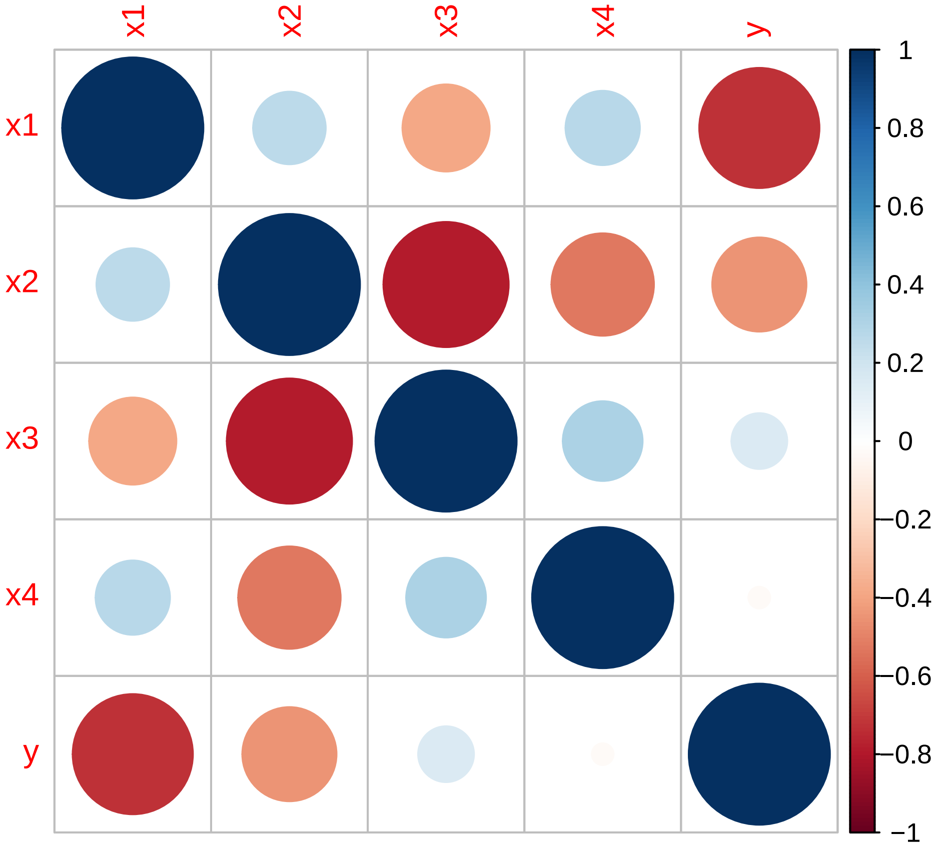
Probability Density plot of Statistical Entropy of image



Correlation between Independent Variables

The correlations (by Pearson's method) between the variables are as follows:

	x1	x2	x3	x4	y
x1	1.000	0.264	-0.381	0.277	-0.725
x2	0.264	1.000	-0.787	-0.526	-0.445
x3	-0.381	-0.787	1.000	0.319	0.156
x4	0.277	-0.526	0.319	1.000	-0.023
y	-0.725	-0.445	0.156	-0.023	1.000



Statistical Distances between Continuous Variables

The statistical distances between the scaled continuous variables are as follows:

	x1	x2	x3	x4
Variance of Wavelet Transformed image (x1)	0.00	44.92	61.53	44.53
Skewness of Wavelet Transformed image (x2)	44.92	0.00	70.00	64.69
Curtosis of Wavelet Transformed image (x3)	61.53	70.00	0.00	43.22
Statistical Entropy of image (x4)	44.53	64.69	43.22	0.00

Introduction to Methods

Model Training Algorithms

The *k*-Nearest Neighbours Algorithm

The *k*-nearest neighbours (*k*NN) algorithm measures the distance between different points. This distance is normally determined by the Euclidean method, which determines the shortest possible distance between two points. The *k*NN algorithm computes the distance between a point and the nearest *k* other points. It then allocates an expected class based on the classes of its neighbours. The value of *k* is a tuning parameter that is specified in the call to the algorithm. The `caret` package in R enables a comparison of multiple values of *k* from a single call to `caret::train()`, into which the argument `method = 'knn'` is passed.

Determination of euclidean distance

Euclidean distance is determined by the use of scaled data, after scaling the data so as to make each X vector equivalent.

		X_1	X_2	X_3	$\dots$	X_n
Observation 1	Obs_1	x_{1_1}	x_{2_1}	x_{3_1}	$\dots$	x_{n_1}
Observation 2	Obs_2	x_{1_2}	x_{2_2}	x_{3_2}	$\dots$	x_{n_2}
Observation 3	Obs_3	x_{1_3}	x_{2_3}	x_{3_3}	$\dots$	x_{n_3}
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
Observation n	Obs_n	x_{1_n}	x_{2_n}	x_{3_n}	$\dots$	x_{n_n}

Viewing each observation as a row-wise vector, then vector arithmetic may be used as follows to determine the distance between observations, e.g between Observation 1 and Observation 2:

$$\sqrt{\frac{\sum(Obs_1 - Obs_2)^2}{n_x}}$$

where n_x is the number of predictors.This may be described point-by-point as follows:

$$\sqrt{\frac{(x_{1_1} - x_{1_2})^2 + (x_{2_1} - x_{2_2})^2 + (x_{3_1} - x_{3_2})^2 + ... + (x_{n_1} - x_{n_2})^2}{n_x}}$$

Or, e.g. between Observation 1 and Observation n (described only in vector form):

$$\sqrt{\frac{\sum(Obs_1 - Obs_{n_{Obs}})^2}{n_x}}$$

where n_x is the number of predictors, and n_{Obs} represents the nth observation. These calculations are generally performed by the algorithm that invokes them. ### Support Vector Machines

A support vector machine (SVM) is an algorithm that seeks to classify by determining the best location of a boundary between classes, based on its tuning parameter, which corresponds to a geometric vector. The classical form of a SVM applies a linear border between points in two dimensions, both of which are predictors; this is extended for more than two dimensions so as to form a hyperplane (an object with $d - 1$ dimensions in d-dimensional statistical space) that forms a boundary between classes. In the case of SVM algorithms, the d dimensions are normally the same as the number of predictors. However, other forms of hyperplane may be used. The determination of support vector properties is normally performed using kernels that seek to determine the clearest boundary possible given the specified parameters, generally maximising the Euclidean distance from the nearest points perpendicular to the boundary vector.

I intend to use a SVM variant known as radial support vector machines; these apply a hyperplane that exhibits radial behaviour rather than a hyperplane that forms a linear border. The behaviour is considered to be radial in that the position and shape of the hyperplane depend on the Euclidean distance to a point that represents the centre of the hyperplane object, as well as parameters specified. The border, when viewed in two dimensions, is generally radial. This central point may or may not correspond to actual data points. This means that a particular item may be classified differently to its nearest neighbours; however this may or may not reflect the reality of the situation from which the data is derived. This may be more appropriate if a sharp boundary is necessary to accurately describe the situation that is being modelled and many points lie close to the boundary. This is particularly helpful in detecting isolated clusters. This will be called by passing the argument `method='svmRadial'` into `caret::train()`.

Radial SVMs in `caret::train()` normally take two parameters, `sigma` (σ) and `C` (C). C represents the penalty cost of an incorrect classification; higher levels of C are therefore likely to result in a more complex model. However, there is the risk of over-training with an excessively high value of C. The parameter `sigma`, also described as gamma (γ) in other SVM algorithm implementations, relates to the smoothness of boundaries. Very low values of σ or γ result in smoother boundary vectors whereas high values of σ or γ may result in over-training if the values are too complex. When the `trainLength` argument is used in `caret::train()`, an optimal single value of `sigma` is chosen and a number of values of C (the number being specified by the integer passed into `trainlength =`) are compared, each value of `C` is generally twice the previous value.

Prevention of Over-training and Bootstrap Aggregation

One possible complication in model development is that of over-training. This may arise when a model fits the data that it is built on, but does not fit equivalent data that it would be expected to fit as accurately as would be expected. In effect, the model becomes trained to statistical noise (variation in the data that is not due to the predictors) in the dataset rather than the variation in the dataset that is explained by the predictors.

One way to overcome this is to split the dataset into training and test sets (both of which are proper subsets of the dataset). However, this has the unfortunate side-effect of reducing the size of the dataset and therefore rendering it more prone to outlier effects and noise than a larger dataset. I now intend to discuss another method, readily available in `caret`, by which this problem may be overcome.

An alternative approach is repeated random sampling, with replacement, to provide a training set of the same size as the original dataset. The effect is to produce datasets whose unique items are a proper subset of the dataset, and are of the same overall size of the subset. This is known as bootstrap aggregation (sometimes shortened to 'bagging'). The disadvantage is that some elements may be repeated more than others; however this may be overcome by repeating the process and producing an ensemble model. This may be invoked by passing the argument `method = 'boot'` to `caret::trainControl()`, The number of repetitions required is specified in the argument `number`. An example, where the required number of repetitions is already stored as the variable `reps`, is as follows:

```
trainControl(method = 'boot', number = reps)
```

As for cross-validation described above, the `trainControl` object produced is then passed to `caret::train()` as the argument `trControl`. It may be assigned to a variable if required.The argument `number` specifies the number of times that the sample-and-train process is to be repeated in order to provide the final model. All repetitions are included. This has the advantage of creating many datasets that are similar but not identical to the training dataset. These enable the model to be repeatedly trained in slightly different circumstances. It is considered good practice to specify a high number of repeats, so as to provide repeated and consistent coverage of the entire dataset. It is standard practice to use 100 or more repeats, so as to minimise the undesirable effect of feature replication described above.

The Confusion Matrix

The Confusion Matrix and Associated Statistics

One way to consider the performance of a categorical predictive model is via the

The confusion matrix is a 2x2 matrix, separated row-wise by the predicted outcome and column-wise by the observed outcome. Where both the predicted and observed outcomes are classed as 'negative' and 'positive', the table may be written as follows:

	Observed	
Expected	**Negative**	**Positive**
Negative	TN (True Negative)	FN (False Negative)
Positive	FP (False Positive)	TP (True Positive)

The accuracy of a test is the sum of true positives and true negatives over the total number of entries n (where $n = TN + FP + FN + TP$), expressed as a probability fraction:

$$Accuracy = \frac{TN + TP}{n}$$

The `train` function in `caret` returns an object whose attribute `results` contains an overall accuracy as described above, and a value of Cohen's Kappa (`kappa`) ; it is a continuous value between 0 and 1 and a higher value of `kappa` means that the result is less likely to be due to chance; such that 0 means that a result is as close to being due to chance as possible, and 1 means that a result is as far as possible from being due to chance. The `confusionMatrix` function returns a number of other outputs of interest such as`Sensitivity` (the probability of a known positive case yielding a positive prediction or test result) and `Specificity` (probability of a known negative case yielding a negative prediction or test result).

Aim of Project

In this project, the aim is to develop a model that has an accuracy of exactly 1, such that there are a total of zero False Positives and zero false negatives (i.e. $FN + FP = 0$, and $TN + TP = n$).

Methods

Methods

Packages Used

- tidyverse (including the readr, ggplot2 and dplyr libraries.)
- Additionally, gridExtra was used to group selected ggplot outputs.
- knitr was used for the preparation of the report from the .rmd file, including calls to the kable function.
- caret including the train , predict and confusionMatrix functions.
- corrplot was used to visualise correlation matrices.

Additionally, the modules knitr and kableExtra were loaded in order to prepare this document and a call was made to `dslabs::ds_theme_set()` in order to set the style of plots produced by ggplot2.

Acquisition and Preparation of Data

The data was acquired as described in 'Introduction to Dataset' above, using `readr::read_csv`. The column names described in the introduction were introduced at this stage. The resultant was saved in memory as banknotes. A version was prepared where the dependent variable y was changed in type to a factor using a call to dplyr using `banknotes %>% mutate(y = as.factor(y))`; this object was saved to memory as banknotes_factor.

Exploratory Analysis of Data

Overall Exploratory Analysis

The data was checked for missing values, using the is.na function and adding the totals using colSums. Descriptive statistics of the continuous, predictor variables were prepared using the summary function. Probability density plots of these variables were also made. A frequency table was prepared for the dependent variable y. Correlations between the independent variables were determined using the cor function, and displayed both as a correlation matrix (rounded) and visualised using `corrplot::corrplot`. The statistical distance between the scaled continuous variables was also determined.

Additional Analysis of Factorised Data

As part of the full analysis, further analysis was performed on the independent variables, being grouped by the observed value of associated dependent variable y for that observation. This included the production of factorised scatter plots between independent variables, to see if cluster-based supervised machine learning algorithms were likely to be suitable.

Development and Comparison of Models

Models were developed using *k*NN and radial SVM algorithms, using the bootstrap resampling method with 100 iterations in the first instance and compared for accuracy, with a target of exactly 1 (i.e. 100%). The most accurate model was then tested further with 1000 bootstrap iterations, and all other parameters the same. Variables were extracted from the best output and the model was finalised. An additional R script was prepared so that the final model could be easily reproduced.

Results of Analysis

Further Exploratory Analysis

Let us consider if there are clusters of each class in the dataset. This will help to consider the appropriateness of using grouping-based classification algorithms.

Summaries, factorised by output variable

for y= 0

x1	x2	x3	x4
Min. :-4.2859	Min. :-6.9321	Min. :-4.9417	Min. :-8.5482
1st Qu.: 0.8833	1st Qu.: 0.4501	1st Qu.:-1.7097	1st Qu.:-2.2283
Median : 2.5531	Median : 5.6688	Median : 0.7006	Median :-0.5524
Mean : 2.2767	Mean : 4.2566	Mean : 0.7967	Mean :-1.1476
3rd Qu.: 3.8845	3rd Qu.: 8.6920	3rd Qu.: 2.6529	3rd Qu.: 0.4233
Max. : 6.8248	Max. :12.9516	Max. : 8.8294	Max. : 2.4495

for y= 1

x1	x2	x3	x4
Min. :-7.0421	Min. :-13.7731	Min. :-5.2861	Min. :-7.5887
1st Qu.:-3.0614	1st Qu.: -5.8100	1st Qu.:-1.3575	1st Qu.:-2.4584
Median :-1.8061	Median : 0.1728	Median : 0.3737	Median :-0.6616
Mean :-1.8684	Mean : -0.9936	Mean : 2.1483	Mean :-1.2466
3rd Qu.:-0.5418	3rd Qu.: 3.1893	3rd Qu.: 5.6264	3rd Qu.: 0.3418
Max. : 2.3917	Max. : 9.6014	Max. :17.9274	Max. : 2.1353

Correlation Matrices, factorised by output variable

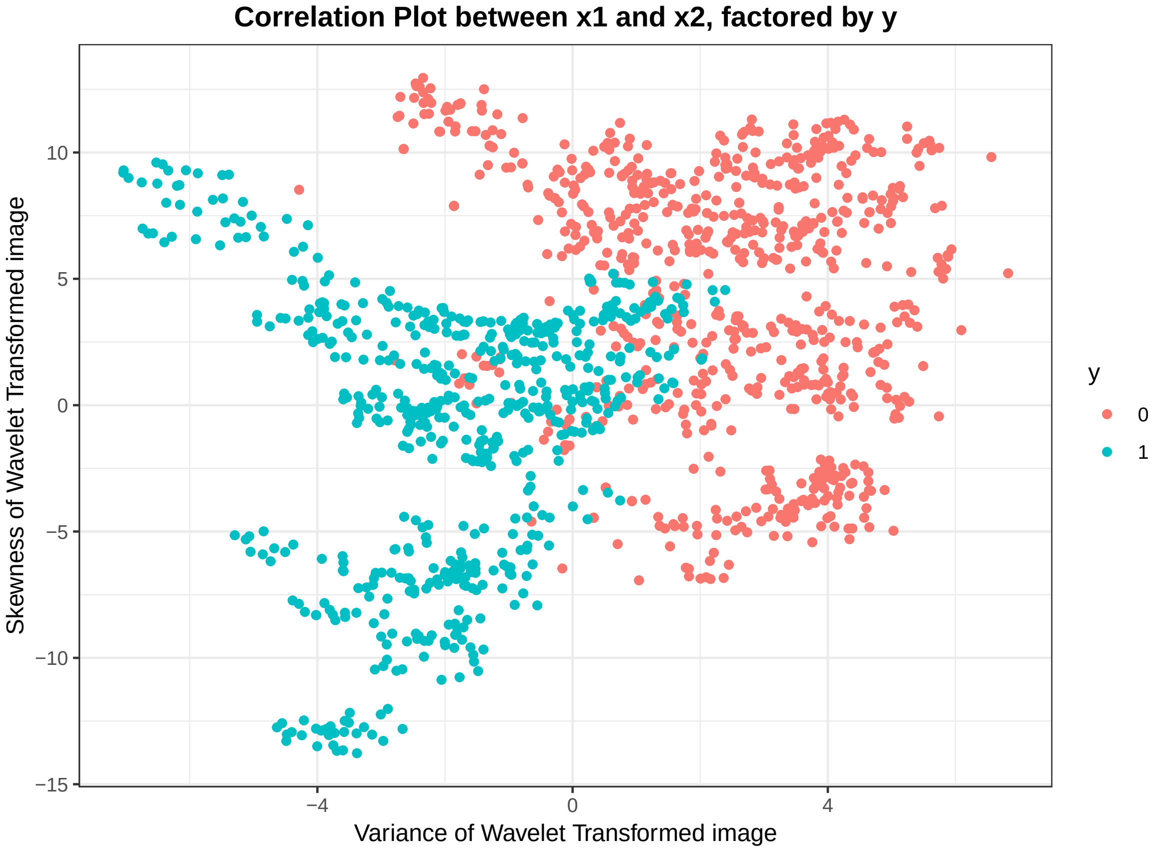
For `y = 0`

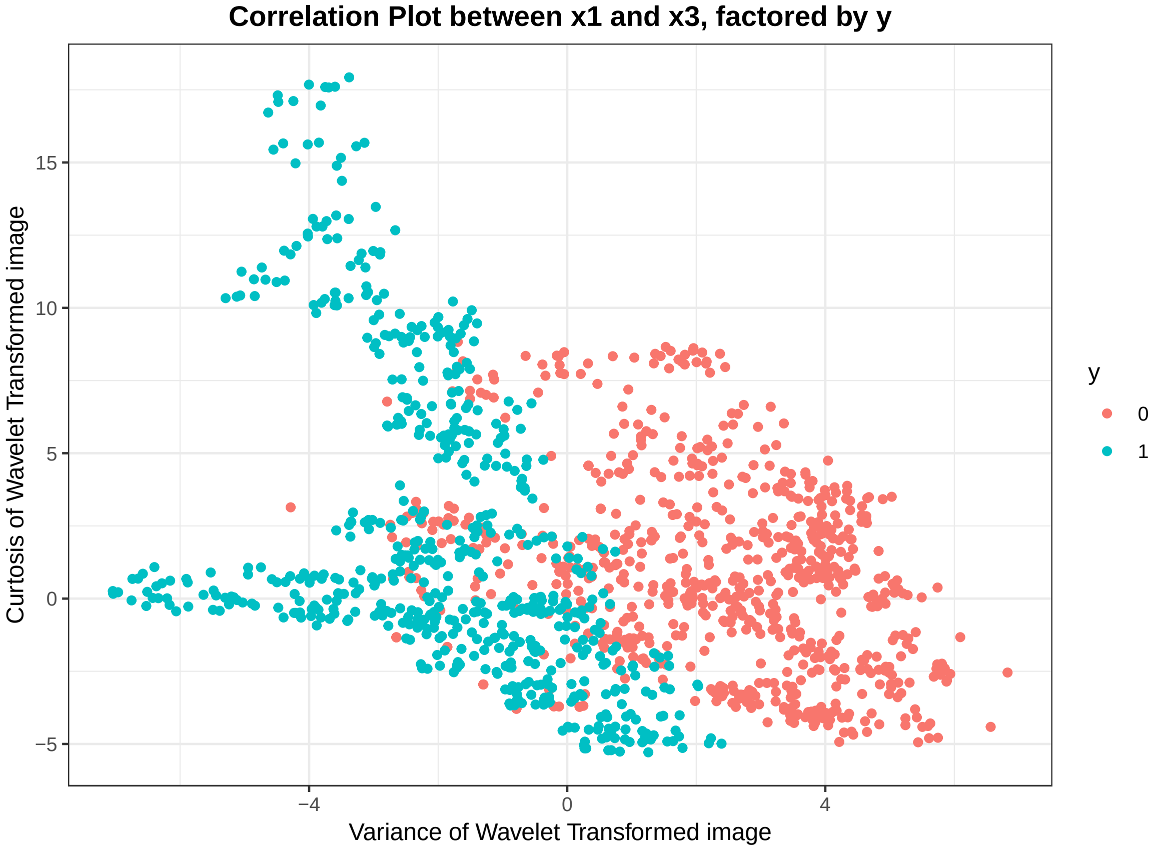
	x1	x2	x3	x4
x1	1.000	-0.227	-0.330	0.416
x2	-0.227	1.000	-0.751	-0.674
x3	-0.330	-0.751	1.000	0.415
x4	0.416	-0.674	0.415	1.000

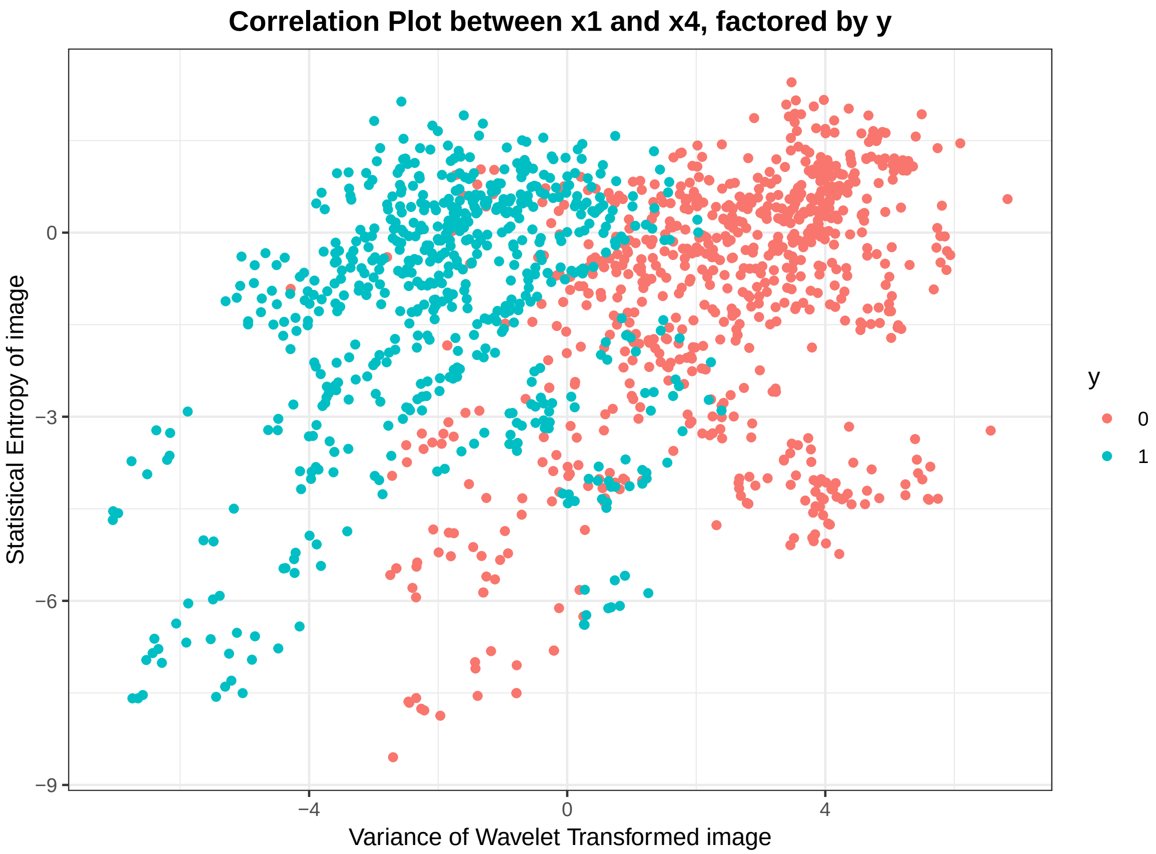
For `y = 1`

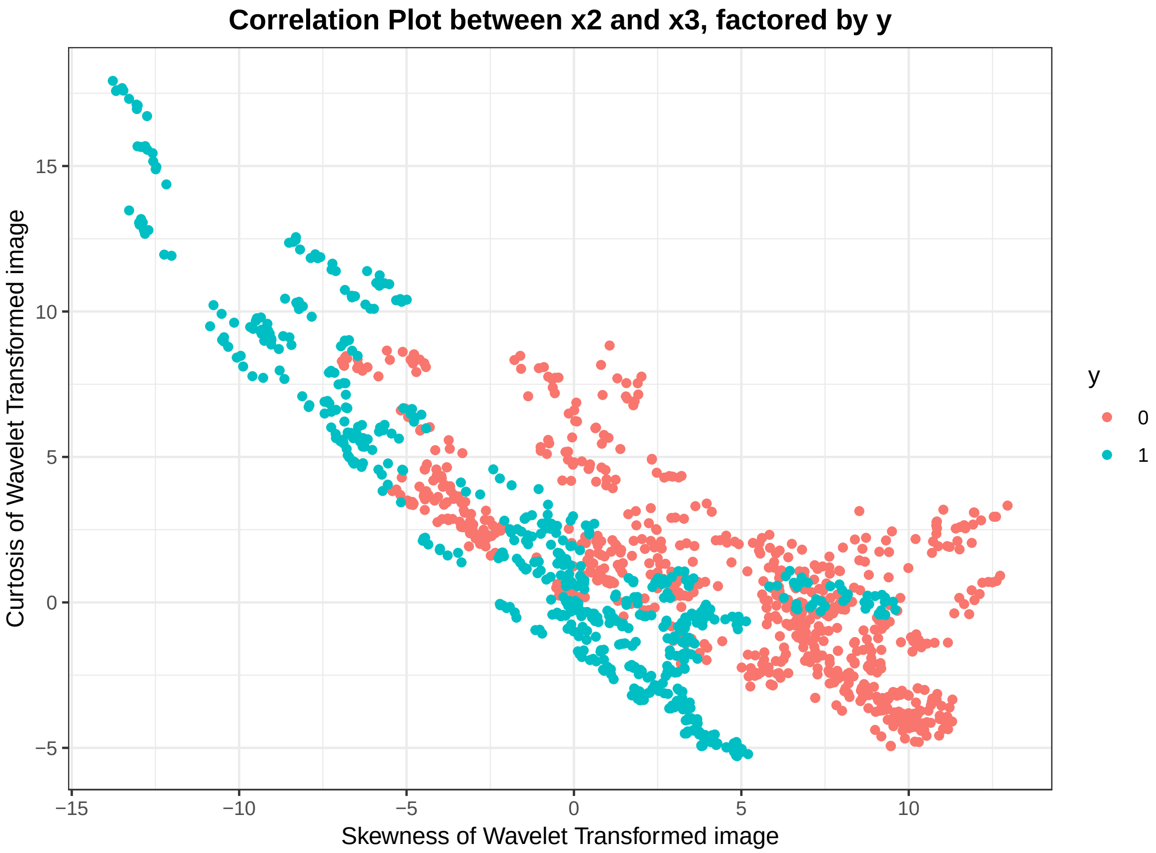
	x1	x2	x3	x4
x1	1.000	0.074	-0.474	0.324
x2	0.074	1.000	-0.888	-0.509
x3	-0.474	-0.888	1.000	0.276
x4	0.324	-0.509	0.276	1.000

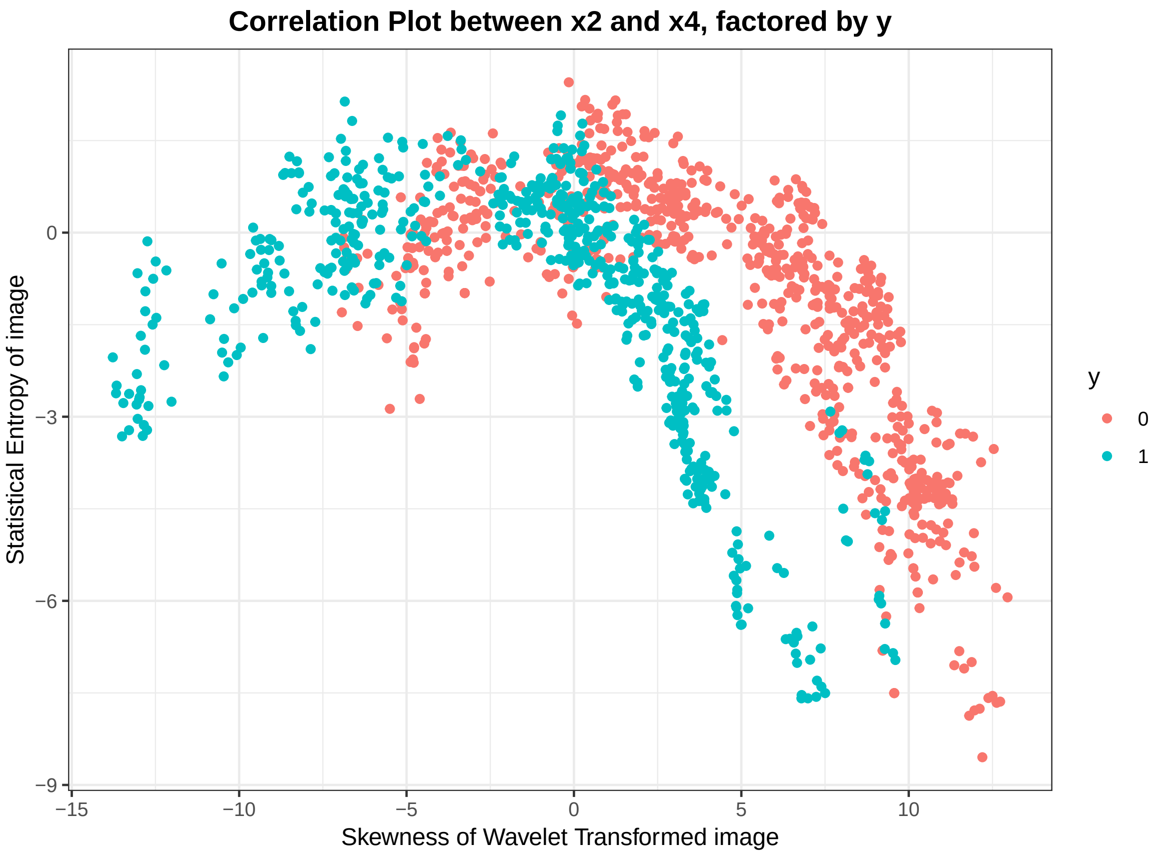
Plots between predictors, factorised by output variable

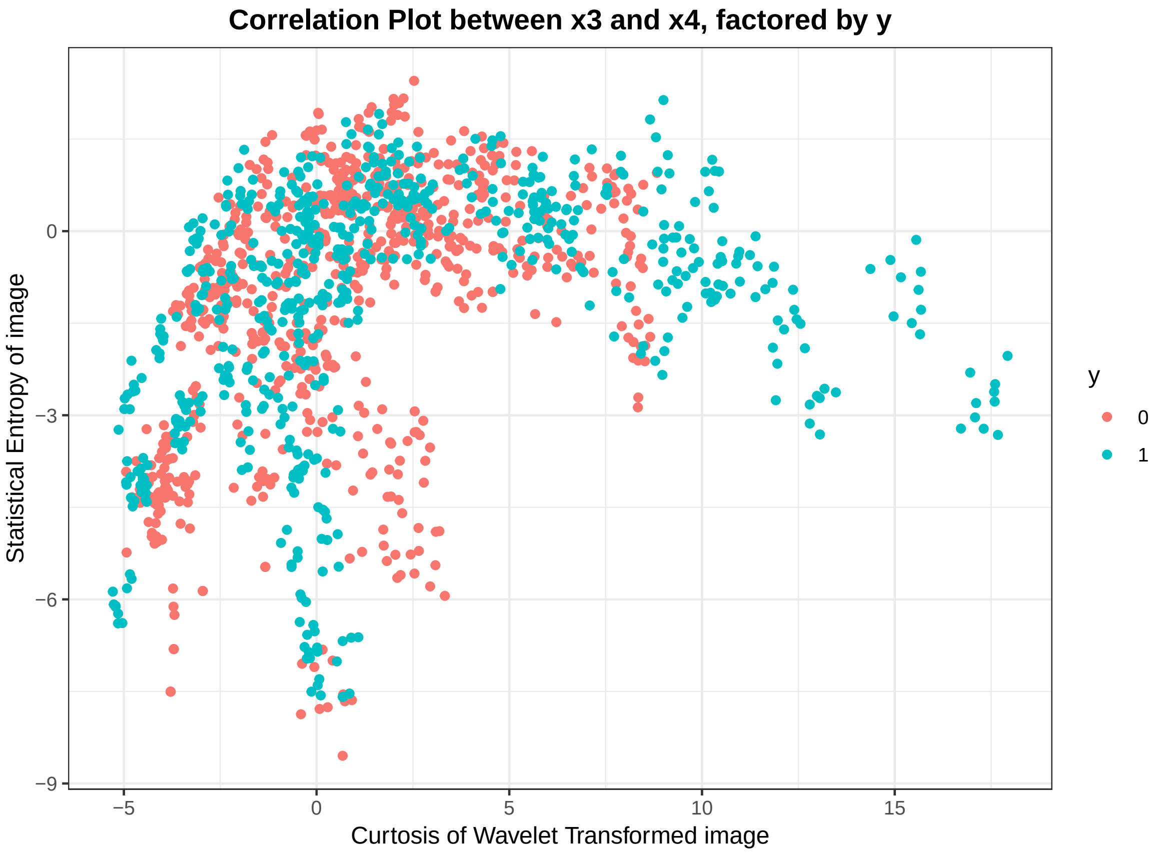





Correlation Plot between x1 and x4, factored by y
Statistical Entropy of image
0
−3
−6
−9
−4
0
4
Variance of Wavelet Transformed image
y
● 0
● 1


Correlation Plot between x2 and x3, factored by y
Curtosis of Wavelet Transformed image
15
10
5
0
−5
−15
−10
−5
0
5
10
Skewness of Wavelet Transformed image
y
● 0
● 1


Correlation Plot between x2 and x4, factored by y


Correlation Plot between x3 and x4, factored by y

k-Nearest Neighbour Models

Let us establish some parameters for the generation of models:

```
params_1 <- trainControl(method = 'boot',
                         number = 100)

params_2 <- trainControl(method = 'boot',
                         number = 1000)
```

`params_1` and `params_2` provide for the use of the bootstrap-aggregation method with 100 and 1000 iterations respectively.

kNN Model with 100 Bootstrap Iterations

Let us try a kNN model with 100 Bootstrap iterations, so as to provide resilience against over-training.

```
knn_model_1 <- train(y~.,data=banknotes_factor,method='knn',
                      tuneLength = 20,
                      preProcess = c('center','scale'),
                      trControl = params_1)
knn_model_1$results %>% kable(format = 'markdown')
```

k	Accuracy	Kappa	AccuracySD	KappaSD
5	0.9981375	0.9962257	0.0016558	0.0033584
7	0.9979406	0.9958245	0.0022004	0.0044712
9	0.9973146	0.9945601	0.0033135	0.0067209
11	0.9966882	0.9932940	0.0042992	0.0087059
13	0.9960802	0.9920640	0.0045609	0.0092372
15	0.9951583	0.9901982	0.0051215	0.0103733
17	0.9946349	0.9891390	0.0055271	0.0112022
19	0.9933695	0.9865801	0.0057217	0.0115983
21	0.9925518	0.9849281	0.0054543	0.0110581
23	0.9917570	0.9833309	0.0051185	0.0103591
25	0.9914003	0.9826052	0.0046491	0.0094096
27	0.9909573	0.9817112	0.0044687	0.0090431
29	0.9906639	0.9811210	0.0044880	0.0090824
31	0.9903825	0.9805480	0.0044153	0.0089467
33	0.9900469	0.9798680	0.0043642	0.0088452
35	0.9900667	0.9799090	0.0043227	0.0087535
37	0.9899866	0.9797455	0.0044825	0.0090813
39	0.9895940	0.9789523	0.0046928	0.0095068
41	0.9894525	0.9786653	0.0047565	0.0096357
43	0.9892523	0.9782609	0.0044363	0.0089866

Let us consider the confusion matrix for the best outcome of kNN with 100 bootstrap iterations.

```r
yhat_knn_1 <- predict(knn_model_1,banknotes_factor)

confusionMatrix(yhat_knn_1,banknotes_factor$y,positive='1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 760   0
         1   2 610

               Accuracy : 0.9985
                 95% CI : (0.9947, 0.9998)
    No Information Rate : 0.5554
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.997

 Mcnemar's Test P-Value : 0.4795

            Sensitivity : 1.0000
            Specificity : 0.9974
         Pos Pred Value : 0.9967
         Neg Pred Value : 1.0000
             Prevalence : 0.4446
         Detection Rate : 0.4446
   Detection Prevalence : 0.4461
      Balanced Accuracy : 0.9987

       'Positive' Class : 1
```

We see that this performance is unsatisfactory at 100 bootstrap-aggregation interations; we will therefore consider another machine learning algorithm.

Radial Support Vector Machines

SVM Model 1 (Bootstrap Aggregation, 100 repetitions)

```
svm_model_1 <- train(y~.,data=banknotes_factor,method='svmRadial',
                       preProcess = c('center','scale'),
                       trainParams = params_1,
                       tuneLength = 10)
svm_model_1$results %>% kable(format = 'markdown')
```

sigma	C	Accuracy	Kappa	AccuracySD	KappaSD
0.4521052	0.25	0.9974614	0.9948357	0.0029463	0.0060145
0.4521052	0.50	0.9996744	0.9993433	0.0009629	0.0019369
0.4521052	1.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	2.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	4.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	8.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	16.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	32.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	64.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4521052	128.00	1.0000000	1.0000000	0.0000000	0.0000000

SVM Model 2 (Bootstrap Aggregation, 1000 repetitions)

Let us repeat this with 1000 iterations of the bootstrap resampling-aggregation process.

```
svm_model_2 <- train(y~.,data=banknotes_factor,method='svmRadial',
                     preProcess = c('center','scale'),
                     trainParams = params_2,
                     tuneLength = 10)
svm_model_2$results %>% kable(format = 'markdown')
```

sigma	C	Accuracy	Kappa	AccuracySD	KappaSD
0.4972677	0.25	0.9977766	0.9954811	0.0026942	0.0054850
0.4972677	0.50	0.9996789	0.9993465	0.0012538	0.0025566
0.4972677	1.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	2.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	4.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	8.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	16.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	32.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	64.00	1.0000000	1.0000000	0.0000000	0.0000000
0.4972677	128.00	1.0000000	1.0000000	0.0000000	0.0000000

Let us finalise the model with a value for the parameter C that is sufficiently high to achieve complete accuracy, but not unduly high that it would lead to over-training. The value of model complexity C is to be high enough to capture all the required complexity, but not produce an overly complex model that is trained to noise. As more than one value of C yields full accuracy, the second-lowest such vaule will be used. Let us consider this section of the model results:

	sigma	C	Accuracy	Kappa	AccuracySD	KappaSD
3	0.4972677	1	1	1	0	0
4	0.4972677	2	1	1	0	0

Let us examine the data frame of parameters that will be passed into the final model.

```
best_values <- svm_model_2$results[first_C_index+1,1:2]
rownames(best_values) <- NULL
best_values %>% kable(format = 'markdown')
```

sigma	C
0.4972677	2

Let us now produce a finalised model and consider its output:

```
svm_model_final <- train(y~.,data=banknotes_factor,method='svmRadial',
                       preProcess = c('center','scale'),
                       trainParams = params_2,
                       tuneGrid = best_values)
svm_model_final$results %>% kable(format = 'markdown')
```

sigma	C	Accuracy	Kappa	AccuracySD	KappaSD
0.4972677	2	1	1	0	0

Let us also consider its confusion matrix and associated statistics:

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 762   0
         1   0 610

               Accuracy : 1
                 95% CI : (0.9973, 1)
    No Information Rate : 0.5554
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.4446
         Detection Rate : 0.4446
   Detection Prevalence : 0.4446
      Balanced Accuracy : 1.0000

       'Positive' Class : 1
```

We can see that the aim of an accuracy of 1 (i.e 100%) has been achieved.

Conclusion

The further analysis of the independent variables, separated by factor, shows significant grouping and therefore the most appropriate machine learning algorithm is a supervised one that allows for grouping around certain points. In particular, the radial SVM algorithm offers the ability to group items in tightly defined groups so as to overcome issues caused by sharp boundaries.

In the variety of models tested, radial SVM offers the best performance over 100 bootstrap iterations with very similar performance over 1,000 iterations, achieving an accuracy of exactly 1, i.e. 100%. The advantage of radial SVM measures over *k*NN algorithms is that the former seeks to define specific boundaries, whereas the latter seeks to classify by the class of the neighbours of an object. Therefore, SVM is able to deal accurately and consistently with cases where an observation is in one class but its neighbours are in another class. Therefore it is possible to develop a model that can return predictions with complete accuracy in situations that feature a sharp boundary.

Also, we can be confident that over-training is unlikely to be an issue. This is achieved by the use of bootstrap aggregation, providing multiple training sets from the base dataset and confirming its accuracy against the original dataset. This seeks to remove noise by using multiple selective iterations.

Consideration of The Output Variable

There seems to be some uncertainty in the dataset description regarding the identity of the output variable factors, and this remains unclear at this time. However, some non-Intaglio prints produced histograms that have high kurtosis (Gillich & Lohweg (2010)) and the independent variable `x3` ('Curtosis of Wavelet Transformed image') does appear to be highest for some of those in category 1, but not category 0. It would therefore be necessary to test the model against data where the `y` category corresponds to known Intaglio-printed objects; this would confirm which factor in the training set corresponds to Intaglio print. Also other methods may need to be deployed to give a definitive authentication, as this can only confirm the method of printing; the Intaglio method is difficult to reproduce but the dependence on it may form a point of failure that requires other measures to overcome.

References and Bibliography

Gillich, E. and Lohweg, V. (2010) *Banknote Authentication*, Conference Paper via: https://www.researchgate.net/publication/266673146

Hsu, C.-W. Chang, C.-C. Lin, C.-J. (2010) *A Practical Guide to Support Vector Classifcation* (2010 edition)

Irizarry, R. 2019. *Introduction to Data Science* as published at https://rafalab.github.io/dsbook/ and linked pages on the same website.

Kuhn, M. *The caret Package* at https://topepo.github.io/caret/

Websites consulted: http://www.svms.org/

Data obtained from: https://archive.ics.uci.edu/ml/datasets/banknote+authentication