CSC3060 AIDA – Assignment 3

Adam Coyle

40178464

December 2019 – January 2020

Introduction

The purpose of this assignment is to use machine learning in order to build classifiers for image data, using data from the previous assignment as well as a large sample of unseen data. This report will detail the observations and accuracy of various predictor models such as logistic regression, k-nearest neighbour and decision trees using predetermined features from the data.

Section 1 – Working with Logistic Regression Models

The primary focus of this section is to use logistic regression on the feature data from assignment 2 to build classifiers for living and non-living objects. From the results of the fitted models, we will be looking at the accuracy over the data to determine which individual or combined features might be useful in correctly classifying a given object.

Section 1.1 – Logistic Regression using the Verticalness feature

Upon loading the feature data into the script, I discovered that for the purpose of this task, I needed to classify each of the observations as either "living" or "non-living". So, I wrote a function that iterated through each observation, evaluating the value of the 'label' column. If the value of the label was one of the living things (banana, cherry, flower, pear) then the classification for that observation would be 1. Likewise, for those observations which had labels belonging to non-living things (envelope, golfclub, pencil, wineglass), the classification would be 0. These classifications were made under the assumption that living things were represented by a value of 1 and non-living things by a value of 0.

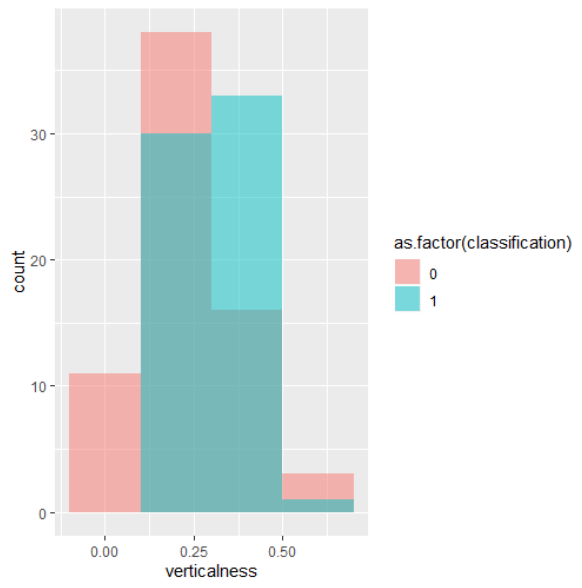
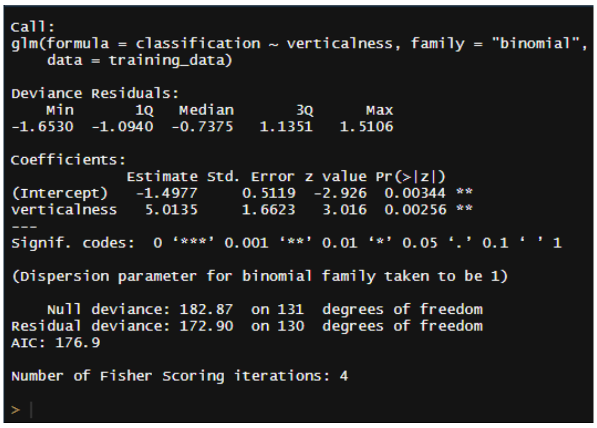


Figure 1 - A histogram of the verticalness feature, with non-living things coloured red and living things coloured blue

With all 160 observations classified as living or non-living, I was able to start building the model. I shuffled a sample of the original dataset and assigned the first 80% to the training dataset and the remaining 20% to the test dataset. The histogram for the 'verticalness' feature shows a significant overlap for both classifications between the values 0.1 and 0.5 (roughly). From a simple assessment of these results one might already suggest that the 'verticalness' feature is not a good predictor of whether a doodle belongs to either the living or non-living category.

The results of the model show significant p values ($p < 0.01$) for both the Intercept and the verticalness coefficients.



```
Call:
glm(formula = classification ~ verticalness, family = "binomial",
    data = training_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6530  -1.0940  -0.7375   1.1351   1.5106

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -1.4977     0.5119  -2.926  0.00344 **
verticalness   5.0135     1.6623   3.016  0.00256 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 182.87  on 131  degrees of freedom
Residual deviance: 172.90  on 130  degrees of freedom
AIC: 176.9

Number of Fisher Scoring iterations: 4

> |
```

Figure 3 - The results table of the logistic regression

The model was built with the training dataset and the results were close to what I had expected. The model could never be totally certain of a doodle’s classification for a given verticalness value, as Figure 2 shows.

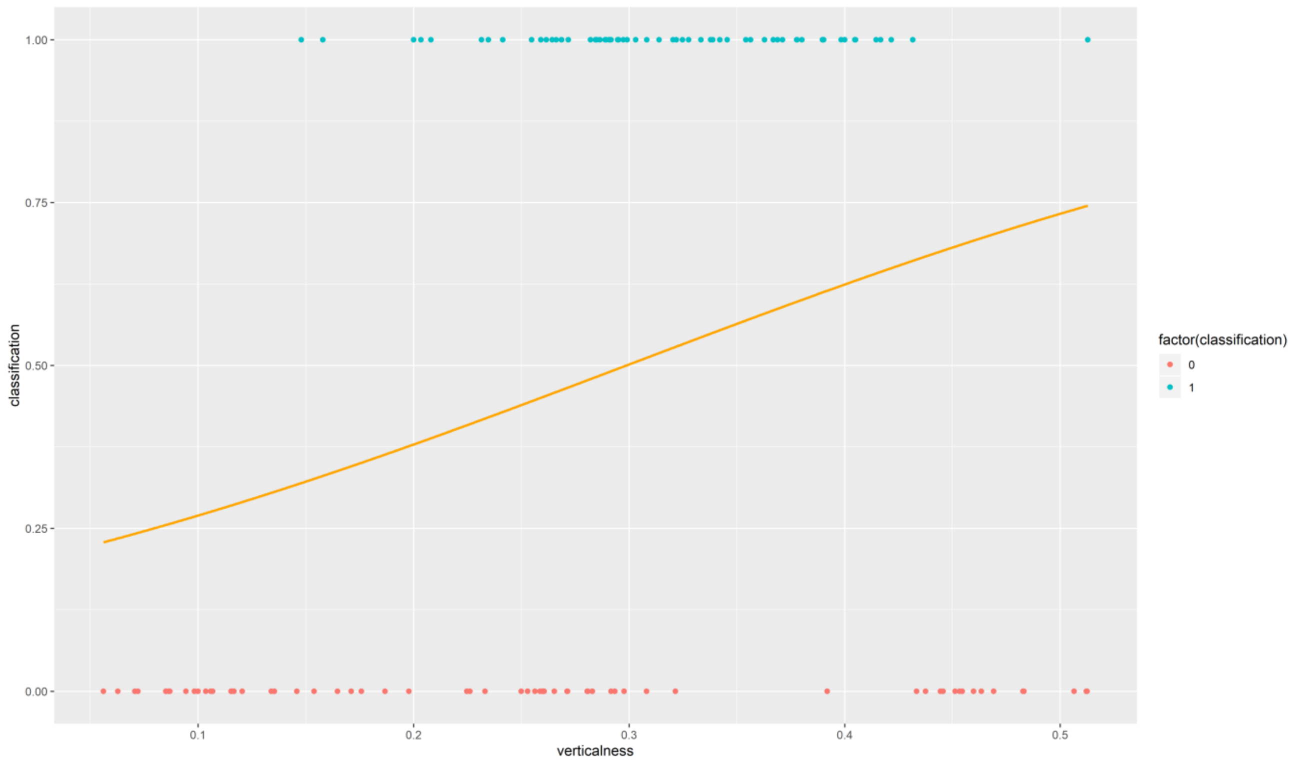


Figure 2 - The fitted curve for the model

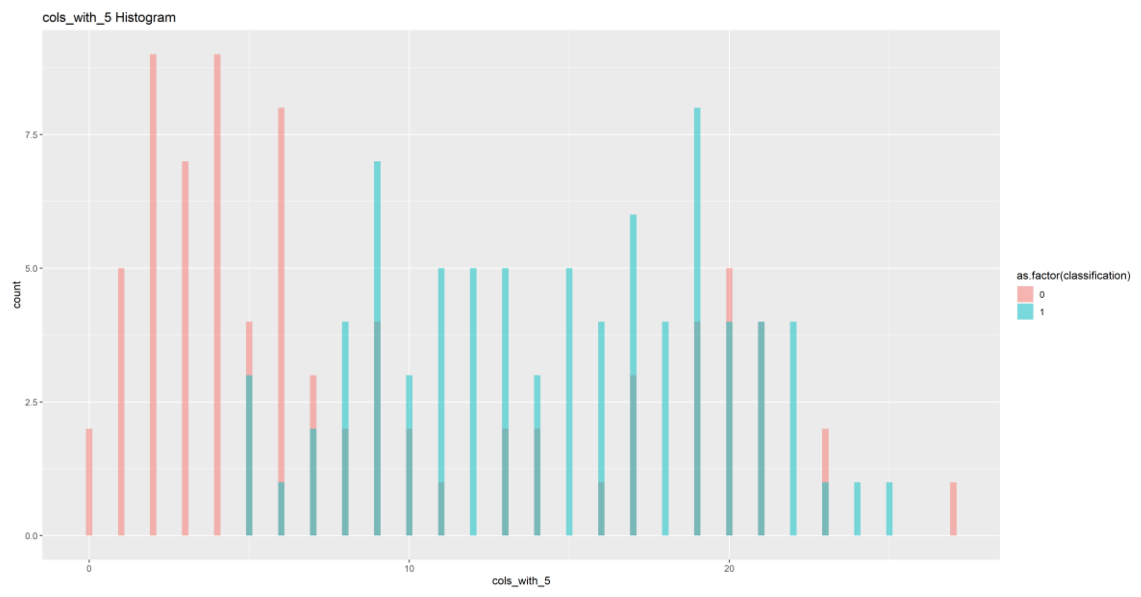
Section 1.2 – Building a classifier using the model from 1.1

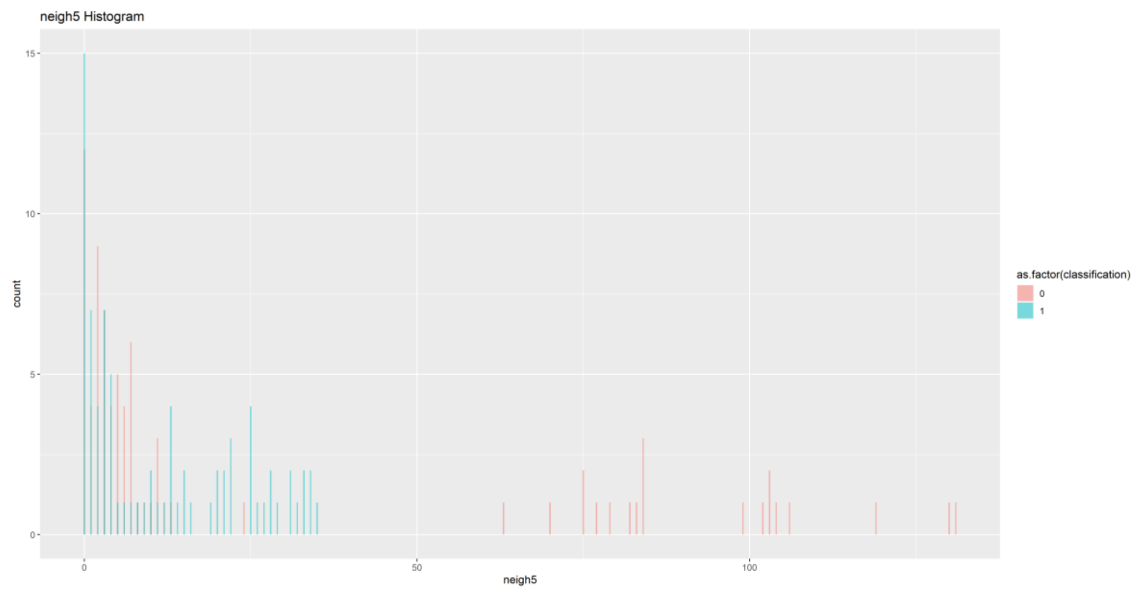
Finding a suitable cut-off value for p to provide the best accuracy for the model was difficult, since there was no clear distinction between living and non-living for a given verticalness value (except for the range 0.5 ~ 0.14, to which the model was still not certain of the classification). Rather than guess the best p-value, I decided to check all possible values of p from 0.01 to 0.99 (inclusive) in increments of 0.01. The table of results (see ‘1.2_p_accuracy.csv’ for full table) showed that several p values shared the highest accuracy of 67.9%. What was most interesting here was that most of the p-values that obtained this accuracy were all consecutive (0.36 – 0.43 inclusive) except for 0.29, which was an outlier. That said, I would use a p cut-off of 0.4 which would obtain said 67.9% accuracy. I would like to think that the model would perform similarly for any future doodle samples if they were drawn similarly to the sample used here, otherwise there would be a reasonable doubt that the code used to calculate the feature data might output different values for the verticalness and therefore result in a very different accuracy for this model.

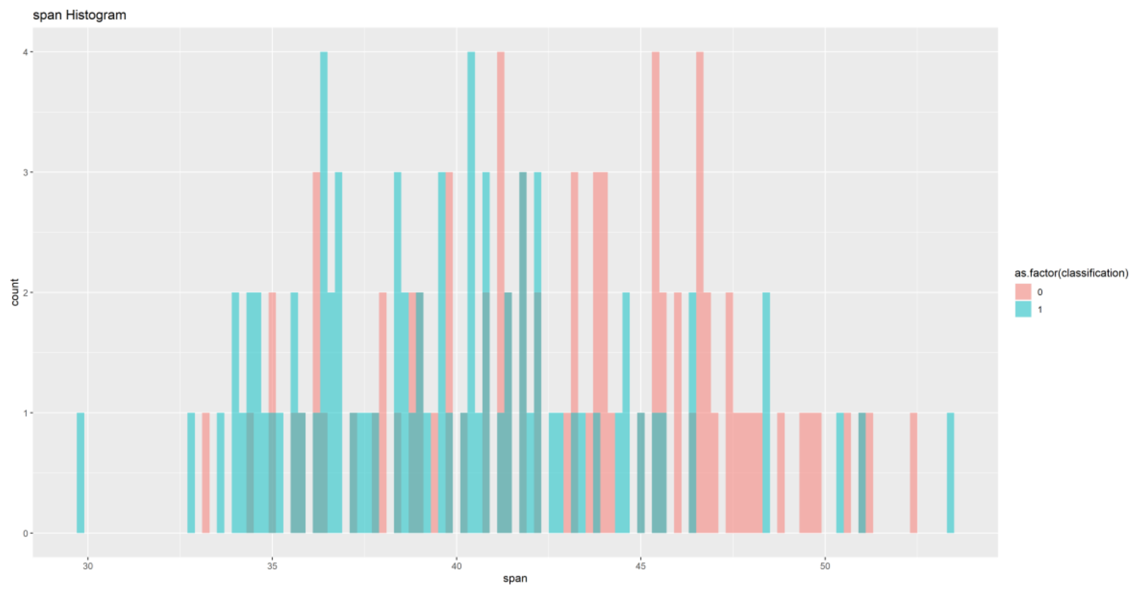
Section 1.3 – Custom Classifier using 3 features

I felt that assessing the feature data from assignment 2 (sample statistics, histograms etc.) would be detrimental to finding the absolute best three features on which to build a classifier, since I had serious doubts about being able to interpret such results in order to build a classifier with the highest accuracy. Initial attempts at this task (choosing the values that I thought were best) showed somewhat decent classification accuracy. Initially I chose to use the three features which had the least skew in the data, but after seeing the results fall in the range 60% ~ 70%, I began to wonder how would find the features with the highest accuracy.

I discovered the combn() function could return all the unique subsets of a given vector, which in the case of the feature data, was 1140. From there it was a similar model building procedure as before, except I built models for all 1140 feature combinations using 5-fold cross-validation and testing every p value between 0.01 and 0.99 for accuracy. After some 20 – 30 minutes, the code finished running and I had a table of ~11400 entries (see '1.3_combn_table.csv' for full results). The table showed 8 entries tied for the highest accuracy (96.25%) using the features 'span', 'cols_with_5' and 'neigh5' for a range of p cut-off values (0.35 – 0.42). Overall the three features hold the top 35 entries for highest accuracy, with entries below containing both 'cols_with_5' and 'neigh5'. Based on the results, I decided to use these three features with a p cut-off value of 0.4 to get an accuracy of 96.25%.







Section 1.4 – Comparing the Model from 1.3 to a random model

Before even investigating the accuracy of a random model, I believe I could confidently say that such a model would be inferior to the model used in 1.3. Given that each observation has a 50% probability of being correct, I would expect a random model to have a very slim chance of attaining an accuracy greater than 96.25%. Figure 4 shows the probability density for the random model. The blue line represents the number of correct predictions needed to be considered more accurate than the model in 1.3 (154 to be exact). The probability of such an occurrence is *basically* zero, and we can disregard such a model in comparison to the one created in 1.3 since it would otherwise be impossible for a random model to have a 96% success rate with only a probability of success for a given trial being 50%.

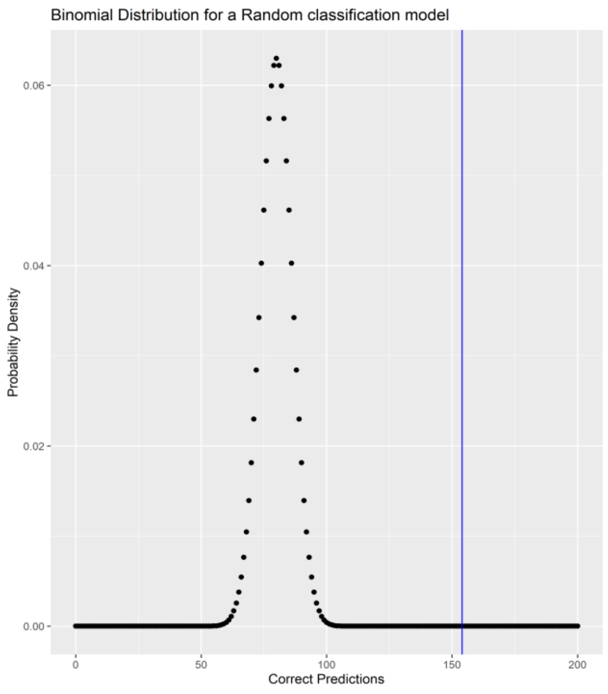


Figure 4 - The binomial distribution of a random classification model.

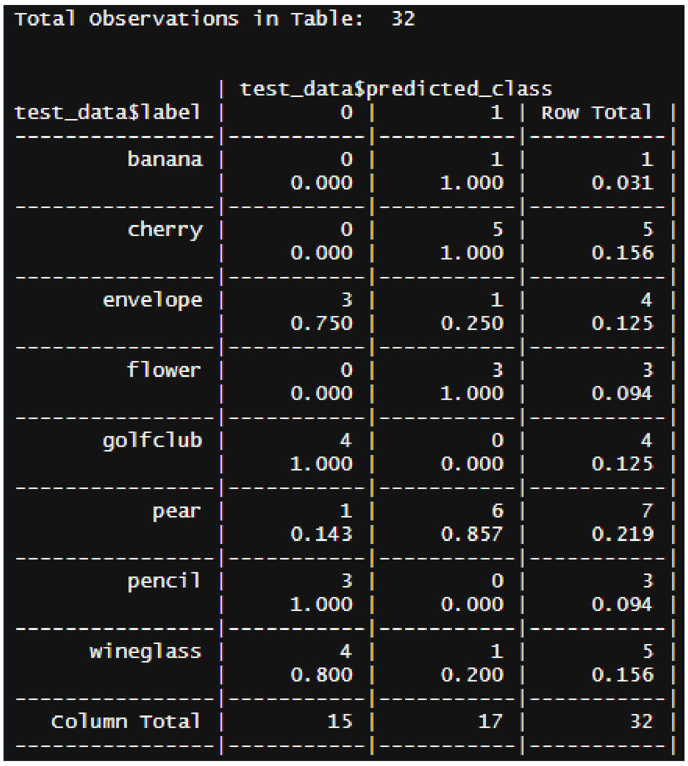
Section 1.5 – Analysing the pattern of errors in the 1.3 model

I decided to go back and gather the data whilst the code for 1.3 was executing, as the results wouldn't be different since the sample was unchanged at this point, and I thought it would be more performant than building the same model a second time. The following table shows the frequency of misclassifications made by the model in 1.3. It is a culmination of the predictions made over each validation fold.

Table 1 – The results for the model prediction over the validation sets for each fold

label	0 (non-living)	1 (living)	total
banana	0	20	20
cherry	0	20	20
envelope	17	3	20
flower	0	20	20
golfclub	20	0	20
pear	1	19	20
pencil	20	0	20
wineglass	18	2	20

The results table above shows clearly the impressive accuracy of the model; all but 1 observation of the pear doodles from the entire sample of living things, was classified incorrectly. This is in comparison to the 5 misclassifications among the non-living objects (2 wineglasses and 3 envelopes were classed as living). Without going into detail fully examining the validation samples in each fold, I think it would be accurate to put the misclassifications down to the distribution of objects in each fold sample, as the cross-tables printed to the console revealed a relatively large imbalance some samples. In the first fold for example, observations of pear objects accounted for 22% of that fold's observations, in comparison to what should be around 12.5%.



```
Total Observations in Table:  32

               | test_data$predicted_class
test_data$label |         0 |         1 | Row Total |
---------------|-----------|-----------|-----------|
        banana |         0 |         1 |         1 |
               |     0.000 |     1.000 |     0.031 |
---------------|-----------|-----------|-----------|
        cherry |         0 |         5 |         5 |
               |     0.000 |     1.000 |     0.156 |
---------------|-----------|-----------|-----------|
      envelope |         3 |         1 |         4 |
               |     0.750 |     0.250 |     0.125 |
---------------|-----------|-----------|-----------|
        flower |         0 |         3 |         3 |
               |     0.000 |     1.000 |     0.094 |
---------------|-----------|-----------|-----------|
      golfclub |         4 |         0 |         4 |
               |     1.000 |     0.000 |     0.125 |
---------------|-----------|-----------|-----------|
          pear |         1 |         6 |         7 |
               |     0.143 |     0.857 |     0.219 |
---------------|-----------|-----------|-----------|
        pencil |         3 |         0 |         3 |
               |     1.000 |     0.000 |     0.094 |
---------------|-----------|-----------|-----------|
     wineglass |         4 |         1 |         5 |
               |     0.800 |     0.200 |     0.156 |
---------------|-----------|-----------|-----------|
  Column Total |        15 |        17 |        32 |
---------------|-----------|-----------|-----------|
```

Figure 5 - The cross table for the first fold of validation items and the model's predictions.

Determining which features might improve the accuracy of the model in 1.3 was interesting, as I initially believed that the accuracy of 96%, I observed in 1.3 was as high as it might go. However, I realised that the addition of a fourth feature to the model could yield an improvement, but as I experienced in 1.3, finding a good fourth feature on the basis of visual assessment was difficult, since I would need to find a feature which had distinct values for living and non-living objects. The height feature, for example, showed strong visual variation between the groups, as well as a near normal distribution, so I would have thought it to be the likely fourth candidate.

Although not explicitly required in the task, further model testing with a fourth feature from the remaining ones showed that height did in fact yield the highest improvement in accuracy (up to 97.5%).

Section 2 – Working with KNN classifiers

In this section I will be investigating the accuracy of models built using k-nearest neighbour to classify individual images. I will be evaluating the difference between models built with and without 5-fold cross validation, to assess the effect that overfitted data might have on the accuracy of the model.

Section 2.1 – KNN classification for all odd values 1-59

Using the supplied training data of 4000 items, I decided to create training and test sets using an 80/20 split. I found that the quickest way to get odd k values between 1 and 59 was to iterate through every number in that range and build a model when the value modulo 2 resulted in a remainder of 1 (i.e. an odd number). I stored the accuracy and complimentary error rates for each value of k in a table.

Initially I had made the mistake of testing the model's accuracy over the full sample of 4000 items, which yielded accuracies between 24% and 26%, and believed it was an indicator of how poor the features were when combined. However, I eventually realised that I had to test the accuracy over the labels for the test data I created for the model, since those labels were unseen by the model. Once this mistake was rectified, I saw accuracies between 70% and 77.5%, which made much more sense given that the model included the best predictor features (for logistic regression) from 1.3 and 1.5. I had the results of each iteration appended to a data frame for future reference.

Additionally, I decided to sample training and test datasets only once before iterating through the values for k, since I thought it would be better to compare the accuracy for different values of k over the same sample.

Section 2.2 – 5-fold Cross-Validation KNN for all odd values 1-59

In my initial attempts to complete this section, I made the same mistake as in 2.1, which was testing model accuracy over the full sample, resulting in the same terrible accuracies. When fixed however, the results showed a slight improvement in the range of accuracies, that was 70.7% up to 78.5% (3 sig. figures).

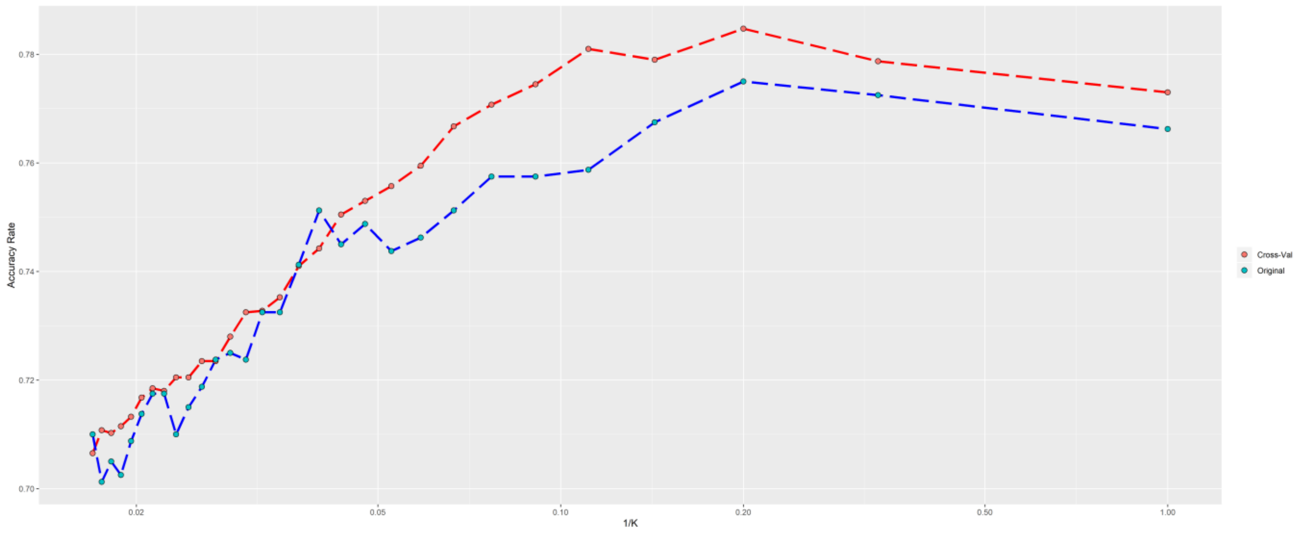


Figure 6 - The cross-validated accuracies and the sample accuracies for 1/k.

Similarly to my procedure in 2.1, I used the same sample for each cross-validated test, with the sample being cut into folds once before the iteration, such that Figure 6 above shows how the accuracy changes for different k values over the **same** sample.

When it came to plot the accuracies against 1/k, it took a lot longer than I had thought. Despite having the results for both methods tabled against k and 1/k, the actual plot looked terrible. By default, the plotted graphs were scaled realistically, such that the units used to space the values of 1/k were constant. This led to a situation where most of the values for 1/k were too coalesced to make any meaningful interpretation of the results. After much trial, error and frustration I discovered the solution was to scale the x axis logarithmically.

The graph shows a slight contrast in how the accuracies change for each method as the number of k neighbours decreases (i.e. 1/k increases). The method that used a random sample initially displayed stronger fluctuations in accuracy as the value of 1/k changed. However, as the number of k-neighbours decreased, both modelling methods showed a trend emerging with a peak accuracy around 0.2 (5 neighbours), and then showed a slight decline until reaching k = 1. Once the number of neighbours considered drops below 25, the cross-validated method becomes the dominant method with higher accuracies until reach k = 1.

Section 2.3

Using the value k = 5 to build the knn model, I iterated through each fold and stored the cross tabulated results for each fold's test dataset into one large table that gives an overview of how often the model confused a given doodle with another.

Table 2 - Overall Contingency Table for the KNN model. The columns are the model predictions and the rows are the actual test data labels.

label	banana	cherry	envelope	flower	golfclub	pear	pencil	wineglass	total
banana	348	23	1	15	14	23	66	10	500
cherry	22	361	1	1	8	95	4	8	500
envelope	1	0	492	6	0	0	1	0	500
flower	2	1	5	486	0	3	3	0	500
golfclub	40	10	0	0	348	8	65	29	500
pear	23	129	0	1	3	322	3	19	500
pencil	90	2	1	12	17	5	367	6	500
wineglass	19	13	0	4	22	16	10	416	500

As Table 2 shows, the model had varying accuracy for any given doodle, the exact same accuracy for banana and golfclub doodles being of some significance. What is more interesting about these doodles was that they were both confused for pencils roughly the same amount of times (66 and 65 respectively) across all 5 folds. Another interesting observation is the high confusion rate between pear and cherry doodles (that is, they are mistaken for each other), which is a little unsurprising if you were to look at the image data since you would notice that some instances of pears strongly resemble cherries and vice versa.

The model was most accurate in identifying envelope doodles (98.4% accuracy) with flowers coming in close second (97.2). This does not surprise me given their unique shapes relative to the other doodles which often have similar shapes to each other (i.e. pears and cherries or pencils, golfclubs and bananas).

Section 3 – Working with Decision Tree Models

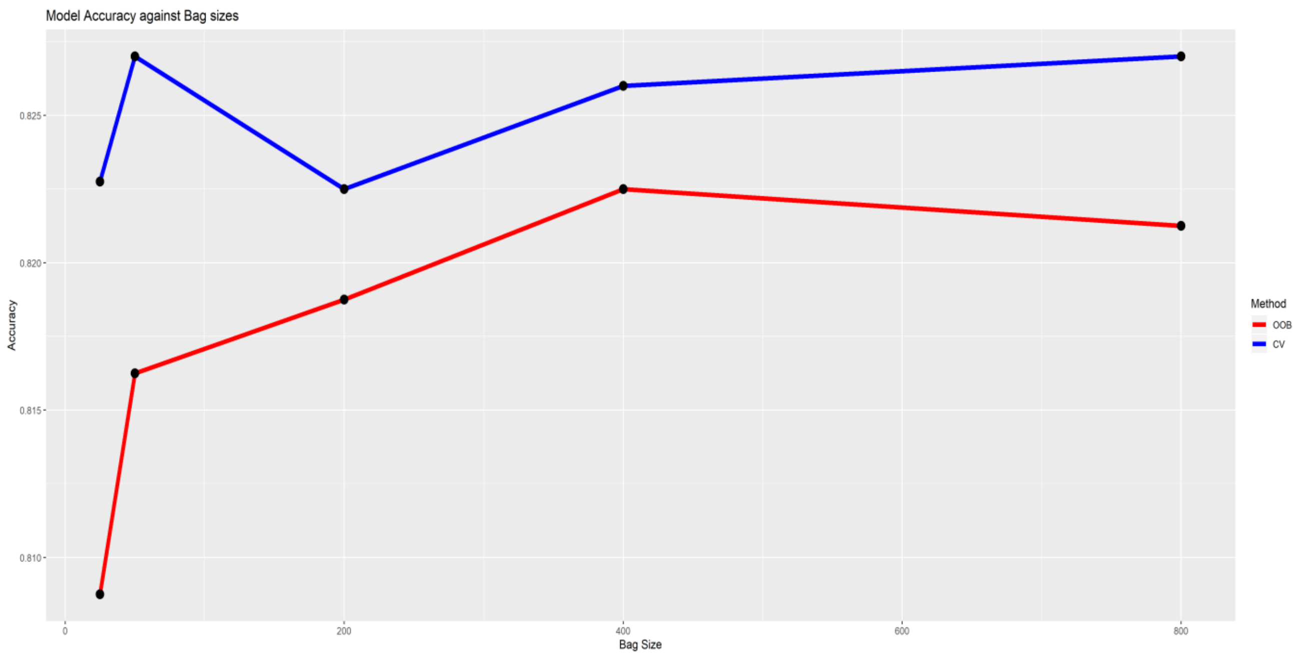
This section will evaluate the accuracy of decision trees and random forests, comparing the accuracies of both 5-fold cross validation and out-of-bag estimation as well as investigating the results of constructing the same random forest 20 times and evaluating the performance of a model once a feature has been removed.

Section 3.1 – Cross-Validated Bagging versus Out-Of-Bag Estimation Bagging

For this subsection, I had to spend a considerable amount of time looking on the web for examples of decision tree bagging in R. The *ipred* library has a built-in bagging function which seemed easy to use but had several parameters which I didn't quite understand. However, given that I would otherwise be unable to write a decent bagging algorithm myself, I went ahead and used the function to build the appropriate models.

bag.size	oob.accuracy	cv.accuracy
25	0.80875	0.82275
50	0.81625	0.827
200	0.81875	0.8225
400	0.8225	0.826
800	0.82125	0.827

The results table showed both models performing well.



The most interesting observation here is the sudden drop in accuracy going from 50 to 200 trees for the cross-validated results. The steady increase from 200 onward would suggest that the high accuracy at nbagg = 50 was possibly due to chance with how each bag drew its sample. I would almost suggest the data was overfit for that parameter, but that wouldn't explain how nbagg = 25 performed worse.

The cross-validated accuracy for a bag size of 200 is the same as the out-of-bag estimate for a bag size of 400. Both methods also experience the same relative jump in accuracy going from 200 replications to 400, before showing diminishing returns for the cross validated results and a decrease for the out-of-bag estimation, despite doubling the number of replications. It would suggest that 400 is the 'sweet spot' for both methods in terms of accuracy to performance.

Overall it is interesting to see that cross-validation outperformed the out-of-bag estimation for all bag sizes. However, I thought I should expect this due to fundamental differences in how each method builds their respective models.

Section 3.2 – Random Forests using various hyperparameters

This section was the most computationally expensive (in terms of execution time) of all sections of the assignment, since I had to fit models using 16 different values for the number of trees, combined with 4 options of predictor and building a model for each of these combinations using 5-fold cross-validation. The results showed very little variance ranging from 82% to 84.5%, which I found to be a little surprising given the broad range of variables used to create each random forest.

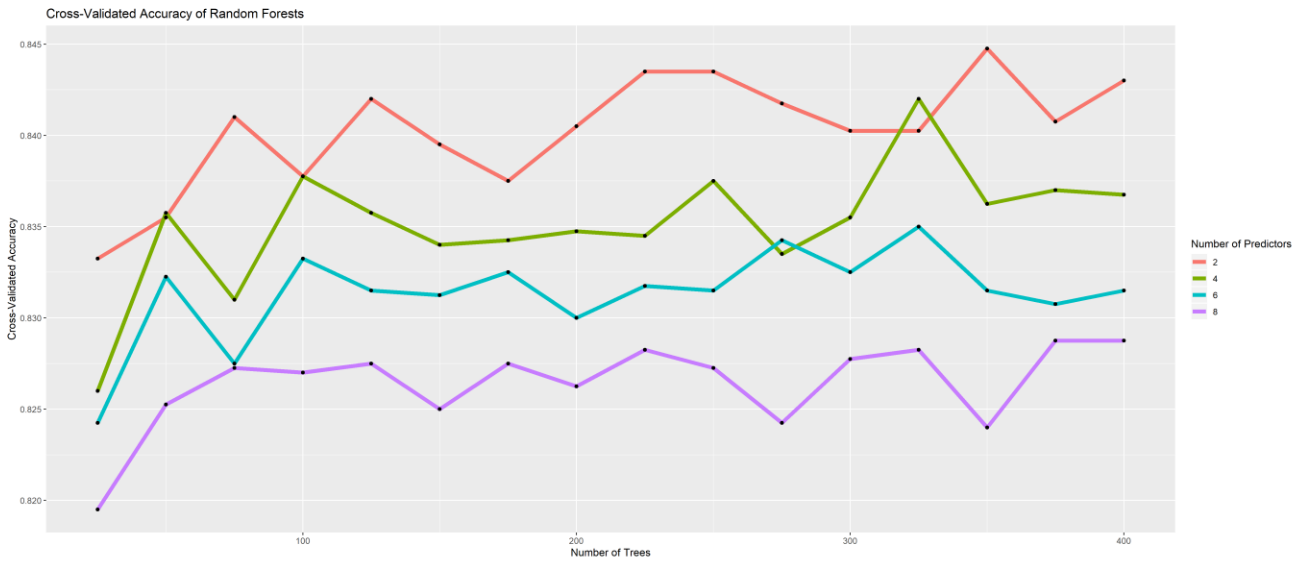


Figure 7 - The accuracy of the random forests using various hyperparameters.

As Figure 7 shows above, as the number of predictors increased, the accuracy of the model continued to worsen. Interestingly, forests built using more predictors experienced more consistent accuracies as the number of trees increased. Take npredictor = 8 for example. It performed at its worst initially (a common theme for all predictors), and then its accuracy stayed around 82.4% - 82.75%, a range of ~ 0.35%.

In contrast, npredictor = 4, even after it's poor start up, had accuracies in the area of 83.125% - 84.2%, a range of ~1.08%. Whilst it would be desirable to have more consistent accuracy rates, this is not always economically feasible given it requires much more execution time for larger npredictors and ntrees. In hindsight, I would have calculated the cost-accuracy ratio for each configuration to find which would be more optimal.

Section 3.3 – Repeated Cross-Validation on the best model from 3.2

Using the table that I constructed in 3.2, I found the entries which had cross-validated accuracies equal to the highest accuracy of the whole table. The highest cross-validated accuracy was obtained using ntrees = 350 and npredictors = 2, for an accuracy of 84.5%. However, as I mentioned in 3.2, a smaller number of predictors had higher accuracies overall, but experienced higher fluctuation as ntrees changed.

The mean of the 20 cross-validated model accuracies was 84%, with a standard deviation of 0.00221 (0.221%?). While a 0.5% difference might not seem like much, keep in mind that models using 8 predictors in 3.2 only had accuracies differing by a max of 0.35%. So, I wonder how this model with 2 predictors would perform if I were to refit it say, 20 or even 100 times?

Section 3.4 – Analysing the Accuracy of the 3.2 model with a feature removed

I found the best way to evaluate which feature should be removed was to check each combination, like previous tasks. I generated each unique 7 feature subset (8 in total) from the original 8 features and built random forests for each using 5-fold cross-validation, with the number of trees and predictors considered equal to the best result obtained in 3.3 (350 and 2 respectively). I then found the difference between the cross-validated accuracy for the given subset and the accuracy obtained using the above parameters.

The results showed that removing any feature reduced the accuracy of the model, with the 'rows_wth_5' feature resulting in the smallest decrease in accuracy (0.8%) to be exact.

removed.feature	cv.accuracy	percent.difference
neigh5	0.77	-0.075
neigh1	0.79	-0.055
cols_with_5	0.834	-0.011
rows_with_5	0.837	-0.008
span	0.835	-0.01
width	0.832	-0.013
height	0.819	-0.026
nr_pix	0.824	-0.021

The results of the 20 cross-validated models showed a mean of 83.3% accuracy. Compared to the accuracy of the model in 3.3, this is a difference of ~ 1.2%. Such a small difference in accuracy might suggest that the feature was not so useful in building the model, since the accuracy difference is rather insignificant. However, the model was restricted to using the best hyperparameter values from 3.3 (ntrees = 350 and npredictors = 2), and so one might ask how the model might perform without the feature for the other hyperparameter values that were used earlier.

Conclusions

As the results have shown, it is possible to classify living and non-living objects using logistic regression using only 3 features to build the model, and it was possible to improve the model by a small percentage by adding a fourth feature. Section 2 demonstrated that knn models could correctly identify doodles about 75% of the time within a range of around 10%. It also identified that such models will often mistake similar (but with differing classifications) doodles for each other quite often, raising the question of how we might improve the accuracy of such models without radically altering the features used to build them. Section 3 demonstrated how using classification decision trees we might also have impressive accuracy rates when identifying doodles. It also showed how changing hyperparameter values might alter the accuracy rates of a model, providing more consistent accuracy rates but at the cost of proportionally increasing performance costs. Overall, I have enjoyed investigating the performance of these different models and working to find the best classification accuracy for the given data.