**CSC3060 AIDA – Assignment 2**

Fionn McConville

40156103

11/01/19

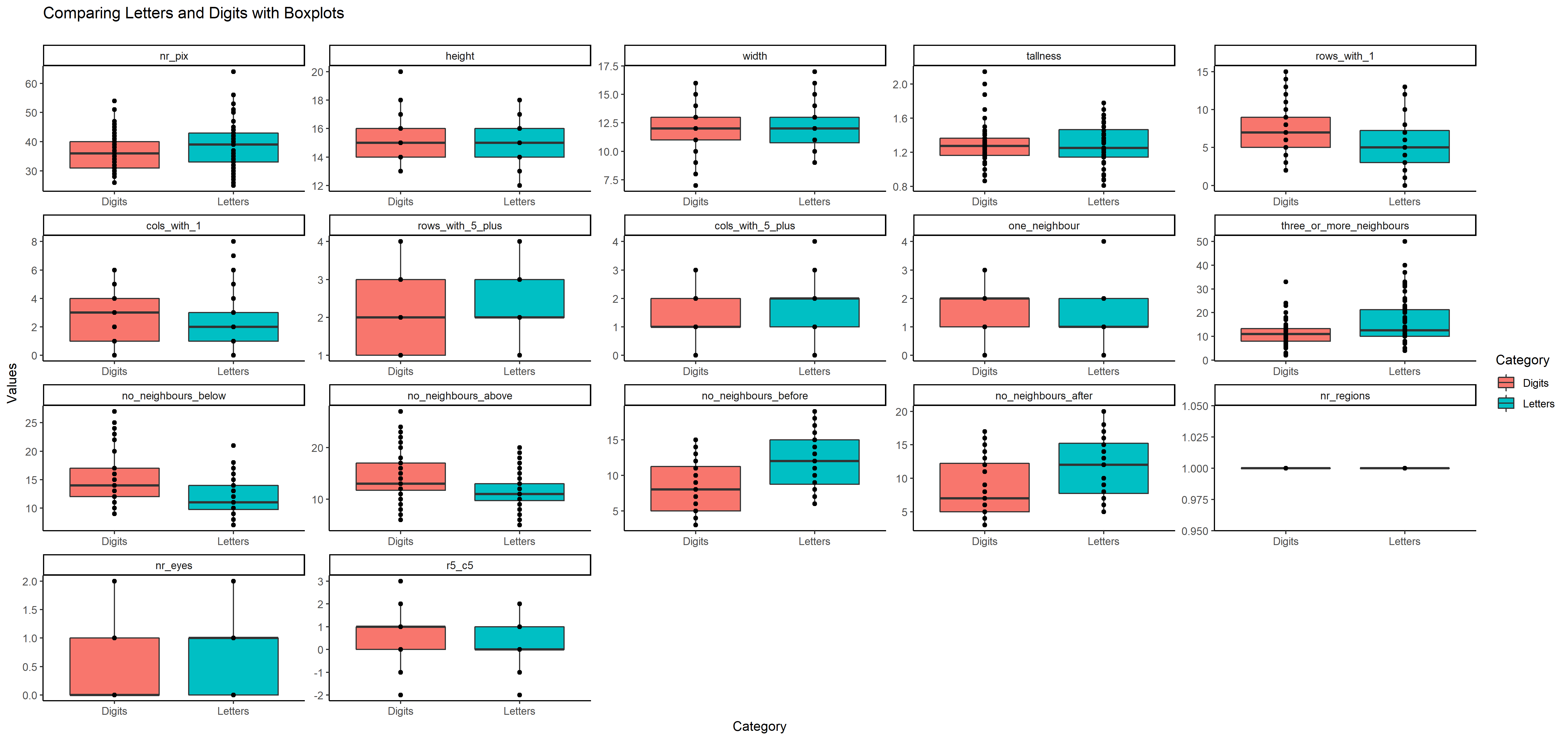
# Introduction

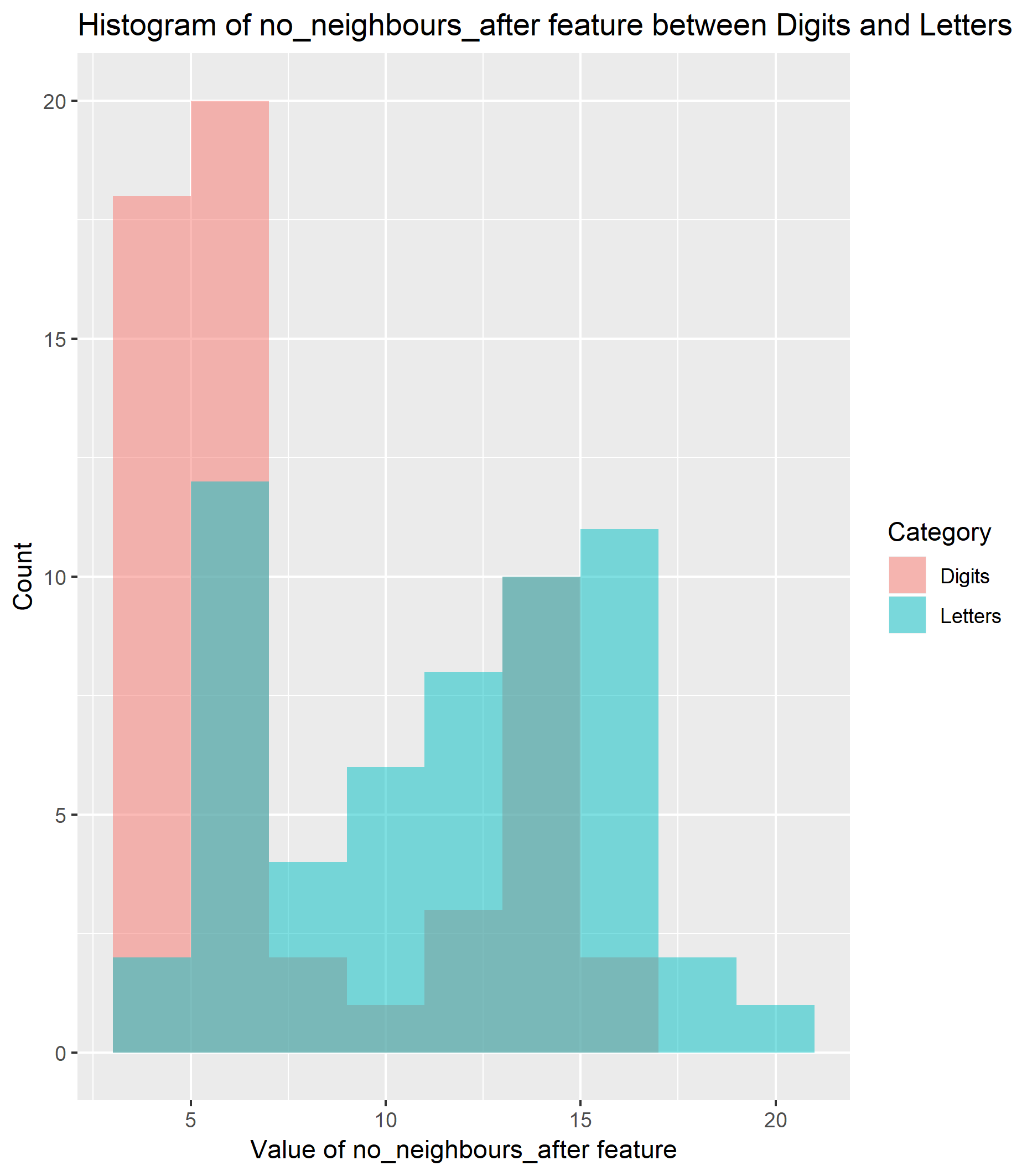
In this assignment I will be use machine learning algorithms to predict class labels based on the feature data I engineered in assignment 1 for section 1 and for section 2 and 3 I will use new training data to train my classification algorithms. I will experiment with various machine learning classification techniques and through my experience with them I’ll pick the best algorithm I can for the classification of the test data in section 4

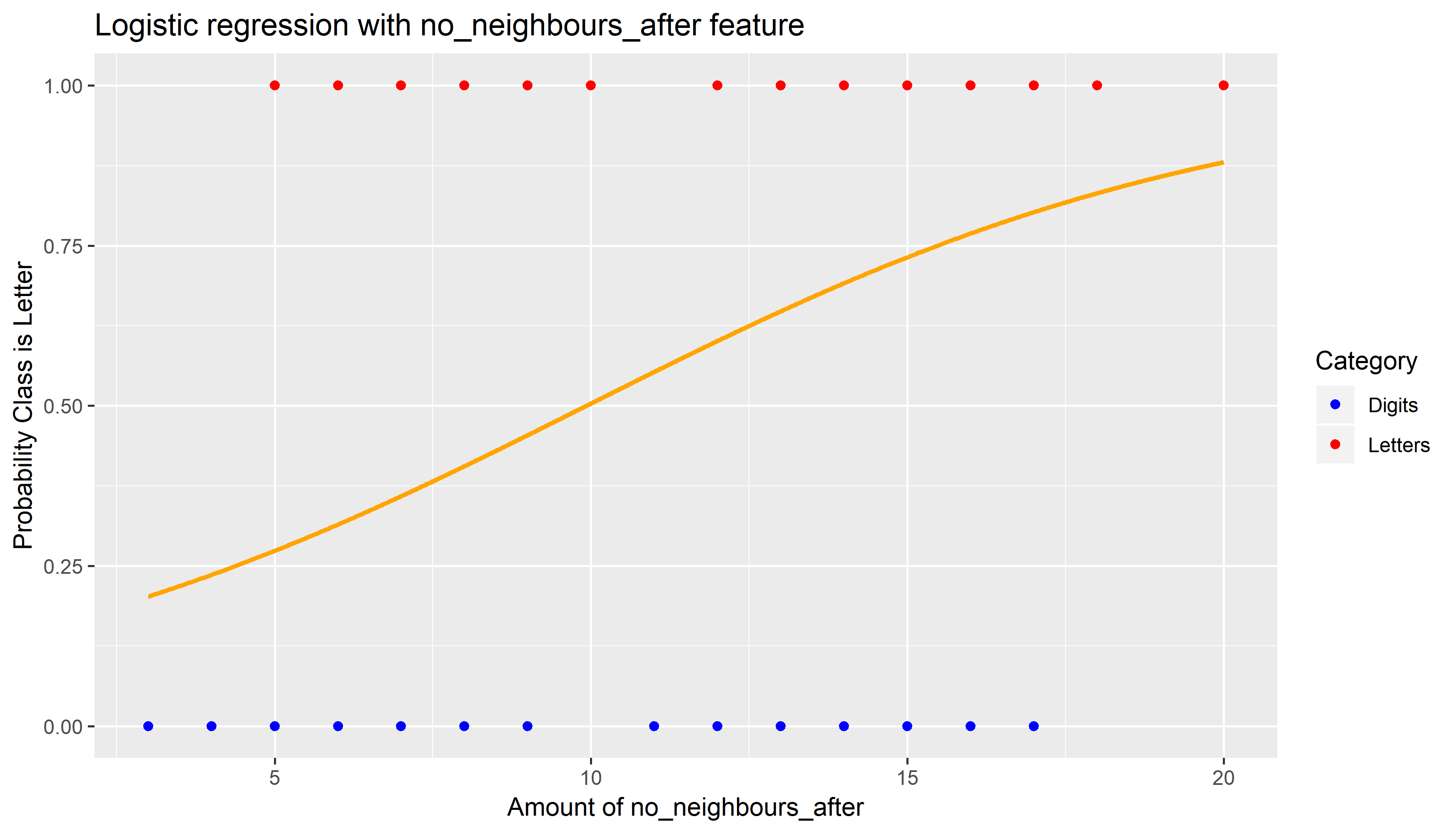
# Section 1 – Logistic Regression Classifier

To begin with in section 1; I will fit a logistic regression classifier to my original feature data, the task here is to distinguish between Digits and Letters so only the digits and letters feature data in my dataset are needed for training this algorithm, (112 training items in total).

## Section 1.1: Logistic regression Classifier on my most significant feature

According to my statistical analysis in section 3.8 of assignment 1 the most significant feature I had for distinguishing between digits and letters was the no\_neighbours\_after feature, in most of the other features the letters and digits datasets were quite similar, below is a boxplot of the data to demonstrate this:

Below is a histogram to more specifically show the difference for the feature no\_neighbours\_after for the two symbol groups:

Whilst the difference in the feature data for the two groups isn’t clear cut and clean, it is the best discriminant I have at my disposal. Using this feature and only this feature I fitted a logistic regression classifier to my feature data and obtained an accuracy of approximately 70% on the model’s predictions with a decision boundary probability of 0.5 (p>0.5 is classed as Letter). Below is a logistic regression curve of the prediction probabilities of the model – where the curve is the estimated probability of the symbol being a letter and the points on thegraph are the actual feature data of my original dataset of symbols: 

# This logistic classifier, at no point, is definitive in terms of its estimated probabilities, but this would be expected as there isn’t a clear enough threshold for the no\_neighbours\_after feature on the letters and digits datasets as we can see via my earlier histogram, still at an accuracy of 70% predictions with just one feature, the results are promising.

## Section 1.2: Logistic regression Classifier on the first eight features

Using the same method as before I fit a glm() model to my data, only this time I used the first eight features. The results I obtained were surprising – the accuracy for this model was less than the previous one by 9% as it had roughly a 61% accuracy compared to the most significant discriminator feature which had 70% accuracy on predictions on my original data. Evidently, this is because the first ten features were not especially significant in terms of classifying letters and digits and so even a combination of these features was still not as useful as the most significant feature of no\_neighbours\_after.

## Section 1.3: 7-fold Cross Validating my models

Using the traincontrol() and train() methods of the caret package in R, I cross validated the models and found no change in model accuracy for either model. This is because in logistic regression there are no hyper parameters to tune over, the accuracy estimates are always given by maximum likelihood and so it is only one set of estimates per fold and not over many different hyperparameter values. The accuracy estimates of my models will be valid for new items drawn from the same population as my training dataset as I used this dataset to train my model, the accuracy may not be the exact same if I were to use a test dataset from a different population though.

## Section 1.4: Significance of these models

By using the binomial distribution, I can demonstrate how significant these models are in predicting either letter or digit compared to a model that responds “digit” all the time – (this model would be right 50% of the time).

Using the pbinom() function I found that the p value for the most significant feature model as 8.178429e-06 and the p value for the eight feature model as 0.008882891. This shows that these models are indeed significantly much more accurate when compared to a model that is 50% accurate.

## Section 1.5: Confusion matrix for both models

Below is the confusion matrix for the best feature logistic regression classifier model:

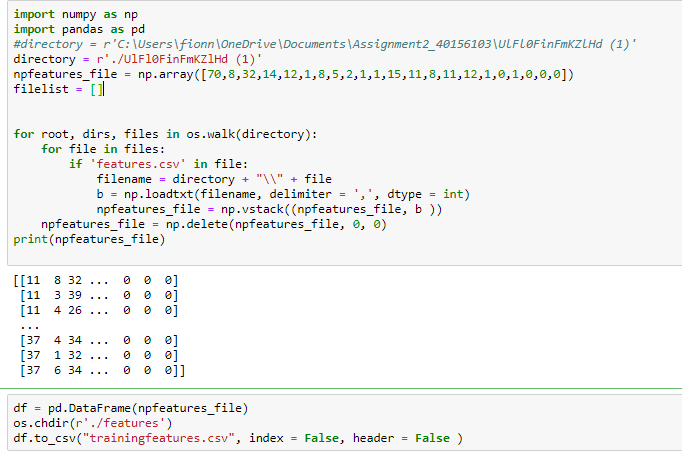
Where it incorrectly classifies a digit as a letter 18 times and it incorrectly classifies a letter as a digit 16 times.

Likewise, below is the confusion matrix for the first 8 features logistic regression classifier model:

Where it incorrectly classifies a digit as a letter 21 times and it incorrectly classifies a letter as a digit 23 times.

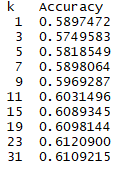
# Section 2

For this section I was provided with a new, and much larger set of data, (8820 training items in total), on which to train my models. To begin with I combined all the feature data into one csv file for convenience; I did this with python code much like the code I used in the first assignment to combine the feature data into one file. Below is a screenshot of said code:



## Section 2.1: Performing KNN on the training data

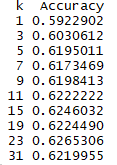
In this section I was tasked with performing a KNN classifier with respect to the 21 symbol categories with different values of K ( k = {1,3,5,7,9,11,15,19,23,31} to be exact) on the training set using the first ten features in my feature data, for this I used the caret train() method which gave an equal accuracy as my manual KNN classifier gave which is encouraging ( see R code for details), within the method I was able to center and scale my feature data - which is necessary for the KNN algorithm to work as it uses the eucledian distanc between the predictor variables to determine neighbours. Below are the accuracies of this model for the different values of K:



The highest accuracy for this model was K = 23.

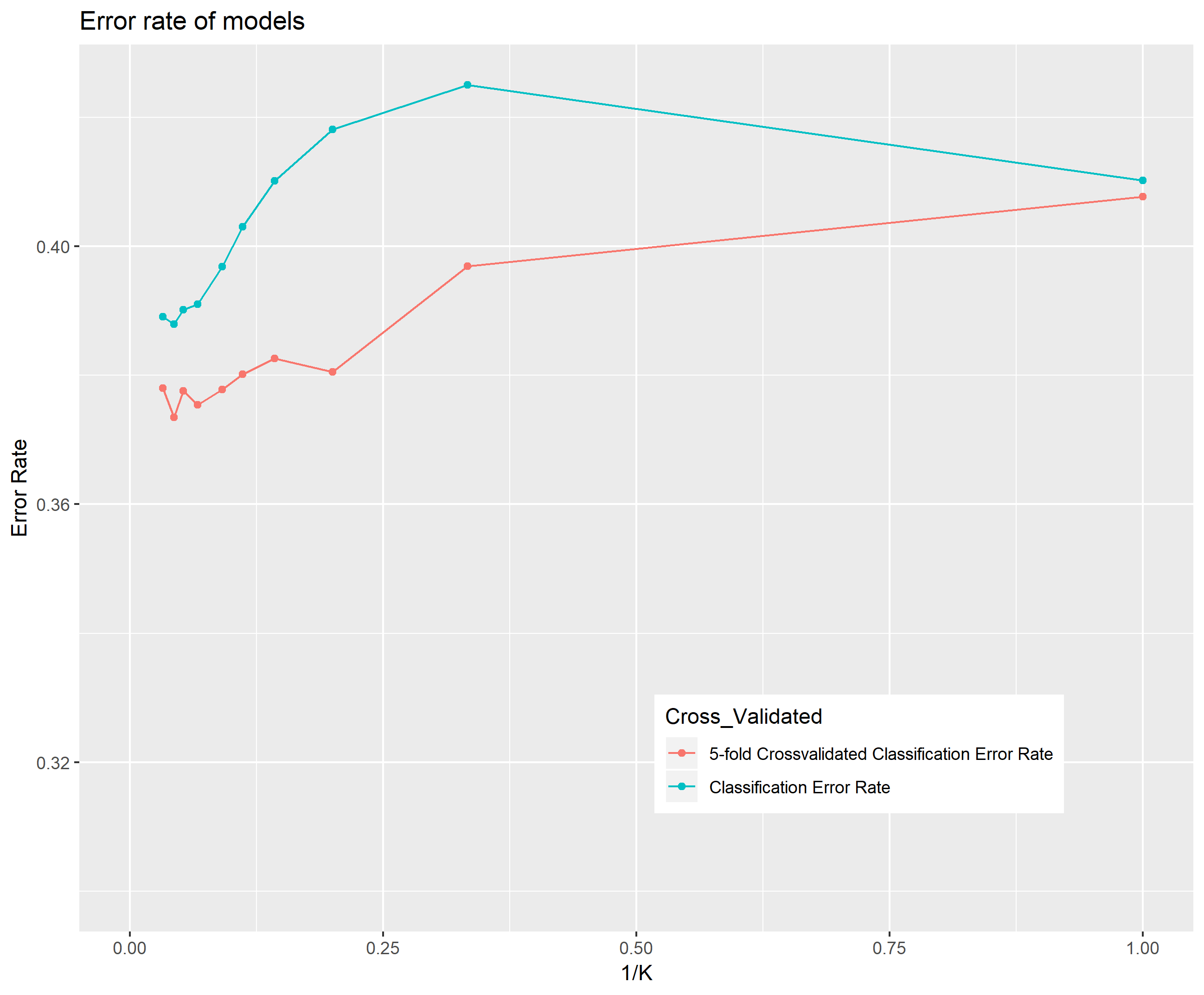
## Section 2.2: Performing 5-fold Cross-Validated KNN on the training data

The difference for this section was that I had to perform the classifier with a 5 fold cross validation, again I used the caret train() method only this time I used the traincontrol() method to specify that I wanted a 5-fold cv as a resampling method. Below are the results from this:



The highest accuracy for this model was K = 23. In this case the accuracy of 0.6265306 is the average accuracy when k= 23 across 5 resampled folds of the data.

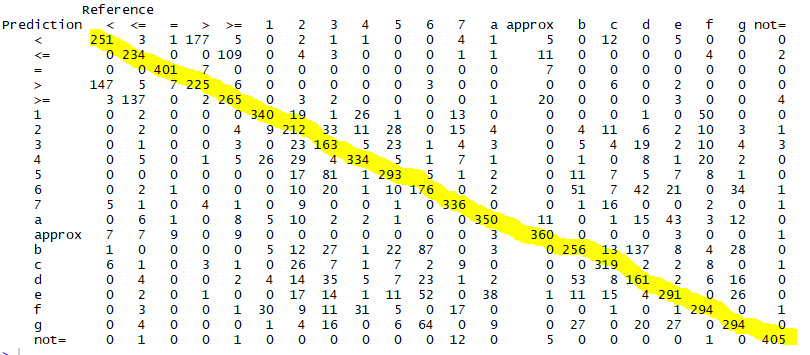
The benefit from performing a cross validation on this model is that the accuracies from the cross validated model more closely resemble the accuracies you would expect when you use this model to classify new test data and thus training a model using cross validation is more likely to give an optimum value of K to be used for the final classification model. This is because by using 5 folds of the data instead of one set of the data like before; there is less of a chance of overfitting the model to the training data. In this instance both models returned the optimal value of K to be 23 but we may not be as lucky on models with less data.

Following is a graph detailing the classification error rate for each value of 1/K:

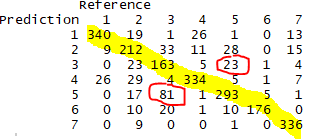
The graph above shows that, for the most part, the error rate of the prediction decreases as K increases, the cross-validated model error rate is lower than the normal model and that the error rate for both models is at its lowest when k is equal to 23.

## Section 2.3: Confusion matrix for 5-fold Cross-Validated KNN Classifier

Below is the overall confusion matrix for classifying symbol categories using my KNN model from section 2.2:



The yellow line highlights the correct predictions, the rest of the numbers tell us how many times each symbol was confused for each other symbol. For example; the digit “4” was correctly predicted by this model 334 times, but it was incorrectly predicted as an “1” 26 times. Below is the more precise confusion matrix of just digits:

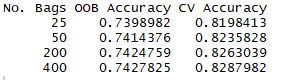
Again, highlighted in yellow is the count of the correct predictions. Circled in red is the most confusable pair of digits in the training data which is 3 and 5; which have combined wrong predictions of 104 between them, (as in 3 is incorrectly predicted as 5 eighty-one times and 5 is incorrectly predicted as 3 twenty-three times.

# Section 3: Decision Tree Classification

## Section 3.1: Performing classification using bagged decision trees; comparing 5-fold cross validation accuracy and Out-Of-Bag error:

Again, in this section, my response class was the 21 label categories of my data. This time I resorted to using bagged decision trees as a method of classification, I repeated the decision tree process using the number of bags = {25, 50, 500, 400}. This section proved quite computationally expensive as each subsection took a lot of time to process my decision trees as there was likely many levels to the trees, owing to the large number of training items and many predictor variables. Unlike previously, I chose to use all the features I created as predictor variables for the decision trees as I felt that more predictor variable information could lead to a better accuracy on my data and I felt that decision trees could harness this well. This possibly made my model susceptible to overfitting to my training data. The bagging of the decision trees may have curtailed this issue somewhat however as it reduced the variance and made my model more robust – the number of bags used in my model were large, (as we see later I used as much as 400 bags in the model) and so an individual decision tree produced by the model may have been high in variance and low in bias – (these trees aren’t pruned and grown deep, as there are a large amount of predictor variables and 21 different classifications to fit) - but by averaging all of these individual trees will give us a reduced variance and a model that is more appropriate for a new, independent set of test data.

I was tasked with comparing two methods of accuracy error estimation with my models – 5-fold cross validation error and out of bag error of my models. I used these resampling methods for each different number of bags in a bagged decision tree and below are the results from this:



The accuracy estimation of the out of bag error for these models is more pessimistic than the cross-validation error, this is likely because the cross-validation method uses a larger set of data to train the model for each decision tree, (as this is 5-fold, the training data for the model is 80% whilst the validation data is 20%). The bagged decision tree uses approximately 2/3rds of the observations to train its model and the remaining 1/3rd of the observations are the OOB (out-of-bag) observations which are used to validate the model created by using the in-bag observations and calculate the OOB error. As a result; the cross-validated model takes advantage of more information to fit a better model. The advantage of an out of bag error is that it takes less time to process than the cross-validation error – this is especially useful when the training data is large, however I believe the models created using 5-fold cross validation are more useful in this case as they make use of more of the training data and so have a more complete final model and also the training data is not too large that we can’t use cross validation.

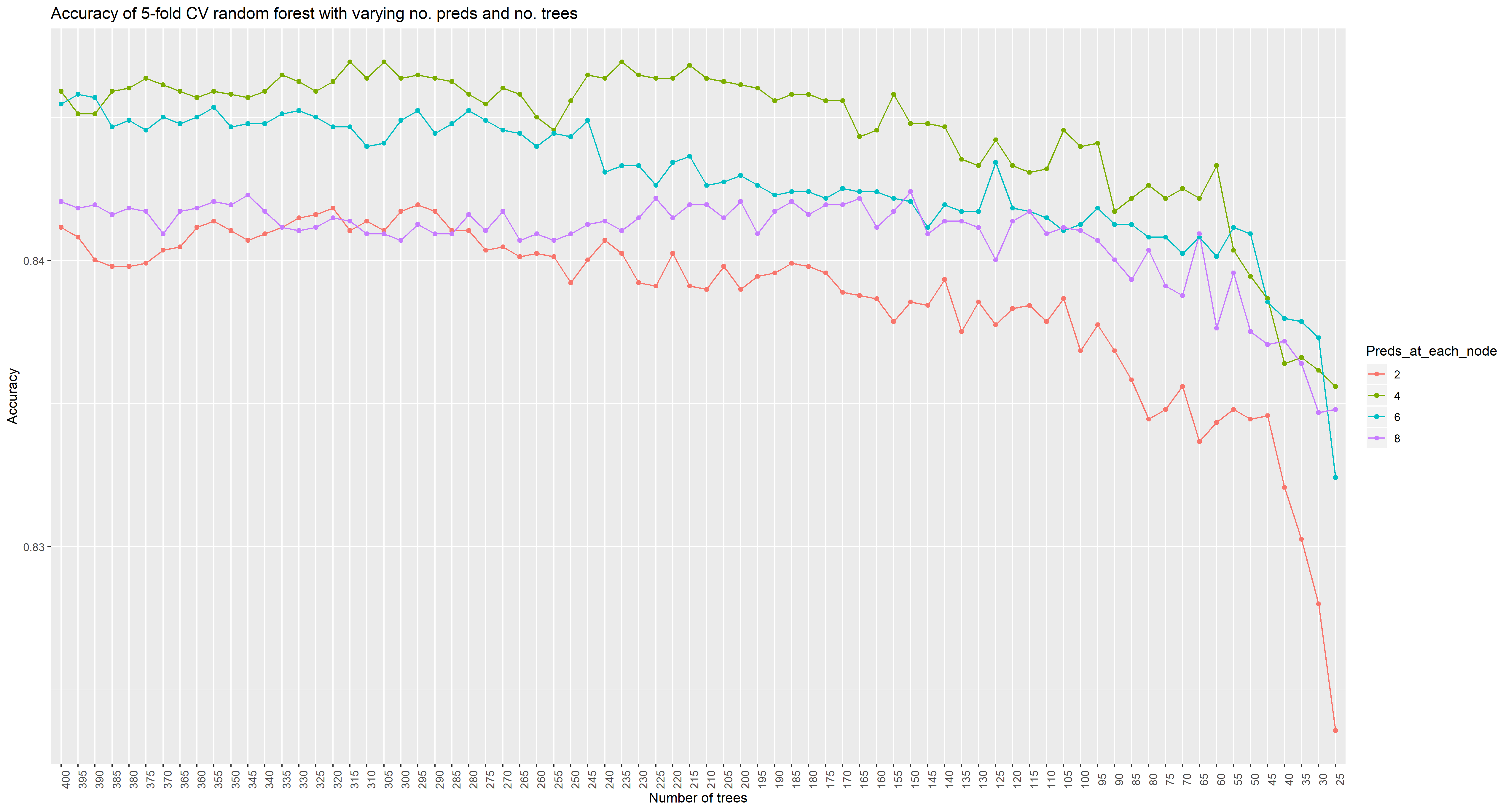
## Section 3.2: Performing classification with random forests using 5-fold cross-validation. Finding the optimal hyper parameters to find the best accuracy:

Random forest models are an improvement over the bagged tree by way of a small tweak that decorrelates the trees – this meaning that the trees built by way of random forest building are quite different from each other in comparison to the trees grown by the normal bagged tree method.

The normal bagged tree method is a greedy algorithm, in that it selects the predictor at each node that gives the maximum information gain, by using the random forest method we no longer can use all of the predictors at each node and our choice of predictors is limited – (the number of predictors used at each node in this experiment was tested p = {2, 4, 6, 8}). Limiting the model’s choice of predictors gives a better accuracy model along with a better reduction in variance as each tree grown by the model will be more different from each other and hence a random forest tree is more reliable. Without limiting the number of predictors most if not all of the trees will start with the same split as they’ll likely all use the predictor that gives the biggest split (largest average entropy) – with random forest a lot of the trees will be limited in their choice of predictor variables and so they will not be able to use this initial predictor as a split.

As such, we should expect better accuracy with this random forest method than the normal bagged decision tree previously used as there is a more robust and less overfitted final model. In this section I was tasked with creating multiple random forest solutions using the number of trees between 25 and 400 (with increments of 5) and varying the number of predictors at each node for each number of trees model with p = {2, 4, 6, 8]. Ultimately this produced a list of decision tree models and I created code to search through this list of models and return the optimal combination of these two hyper parameters that gave the best cross validated accuracy. Below is the result of this search:



The random forest seemed to perform at it’s best along all variations of number of trees when the number of predictors considered at each node was limited to 4, the graph below shows that while at first improving the size of the number of trees improves the model relatively well; after approximately 150 trees there is diminishing returns in terms of model accuracy as the change in accuracy is very slight despite the probable increased computation time it causes. This construction of multiple random forests was very computationally expensive, taking more than 2 hours to complete. The range in accuracies ended up being between 0.823 – 0.846 but the vast majority of the random forest solutions were in the 0.84 -0.845 range – a quite insignificant variation which means that this information gathered will not be conclusive when creating a final model with optimal hyper parameters to use on test data, with maybe the exception of limiting the number of predictors to 4 as it’s consistently proven as the best value by the graph below:

# Section 4: Final model predicting test items

For my final model I chose to use a random forest algorithm with the optimal tuning parameters of number of trees = 235 and number of predictors at each node at 4. I am confident the number of predictors at each node is optimal, as for the number of trees in the model I am sure that my chosen value of 235 will be close enough to the optimal accuracy I can achieve on the test data. Like previously, I used python cod to merge all of the feature files into one overall file for convenience.