CSC2062 AIDA – Assignment 3

James Cassidy

40267110

30/4/21

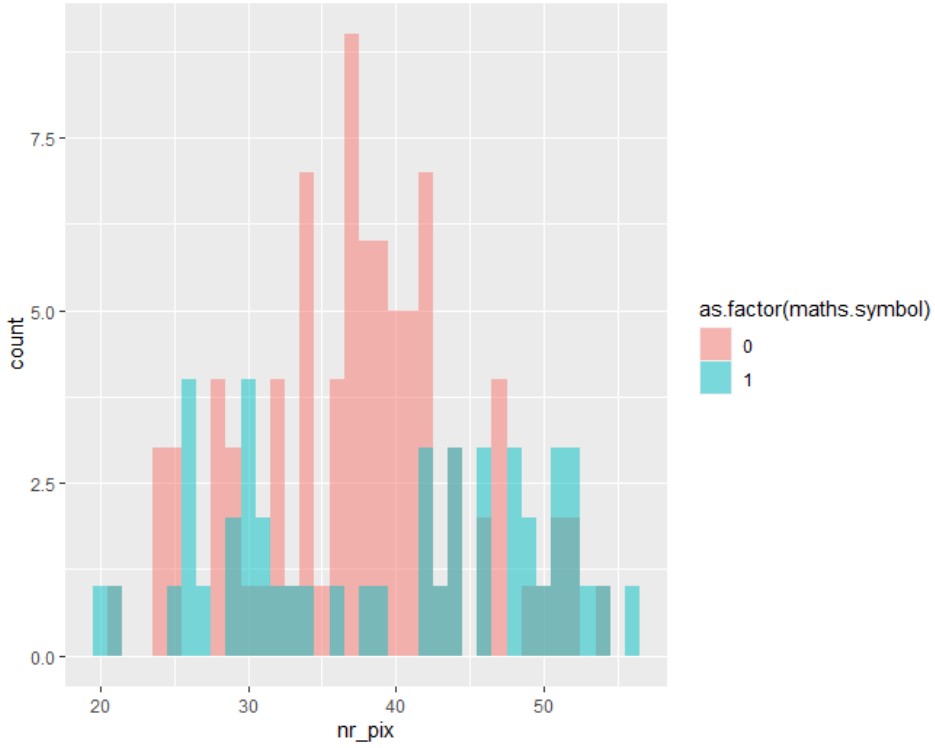
Introduction

In this assignment, I will use the features I developed in my previous assignment 2 to solve classification problems using machine learning techniques. These classifiers will be fitted to my image data to use specific models to predict unseen images.

Section 1

First, I will fit a single logistic regression model to predict the probability of a random image belonging to a maths symbol. Loading in the dataframe containing the results from my previous 168 images. As I need to predict what symbols in my dataset are mathematic symbols or not, I added another row to my features_df called 'maths.symbol'. If the feature is a mathematic symbol, this will be represented by a 1 in the math.symbol column. If not, it will be represented by a 0. This was done to be able to visualise the symbols using ggplot

The library ggplot2 was used to help visualise the dataframe. As we can without using any single logistic models that nr_pix is not a very good predictor when it comes what image has a maths symbol. The maths symbols are represented in the blue.



Now I will apply the model to the training data. In R I have utilized the glm function. This is Generalised Linear Model. Here we will predict the maths symbols using the nr_pix as my predictor and from my dataframe. This will be stored in glmfit.

When using the summary$coef function on glmfit we are presented with the following:

```
> summary(glmfit)$coef
             Estimate Std. Error   z value   Pr(>|z|)
(Intercept) -1.63211236 0.78952568 -2.067206 0.03871473
nr_pix       0.02450825 0.02001317  1.224606 0.22072362
>
```

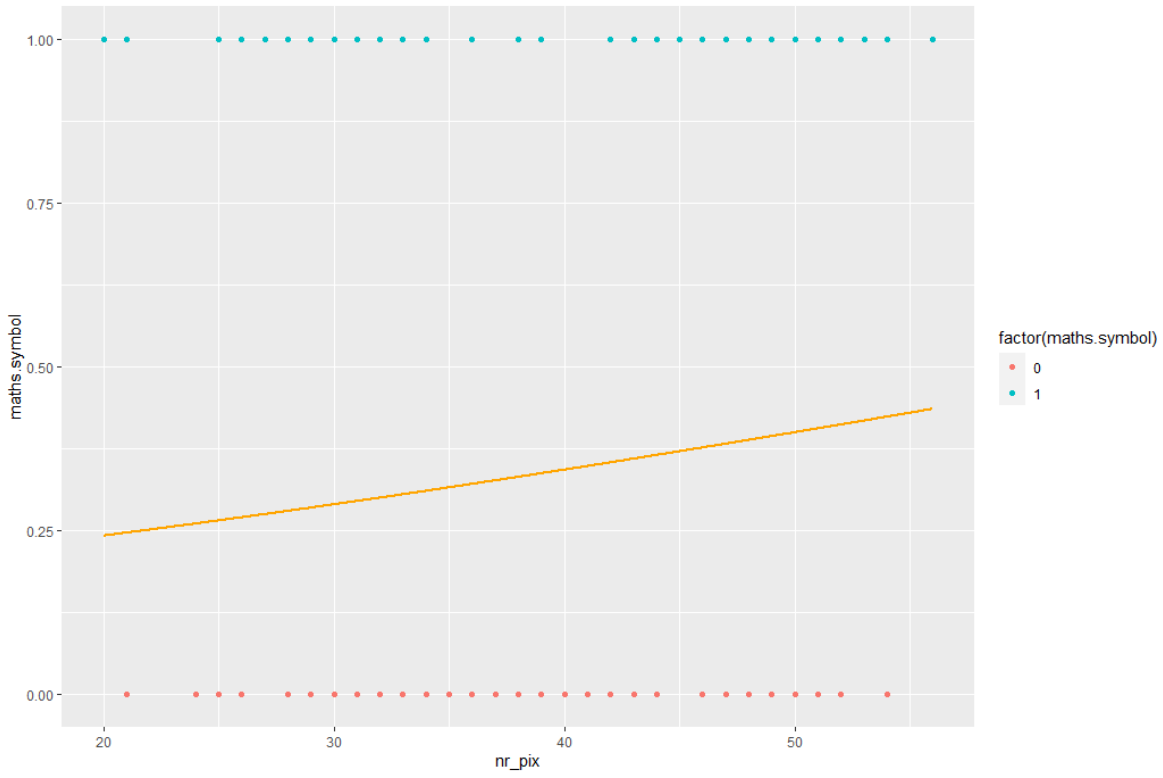
When looking at the p-value we are presented with 0.22072362. This is not a significant p-value and this was to be expected from the previous histogram from earlier. As nr_pix increases, the probability that it is a maths symbol also increases because it is positive. But because it is only so small, when nr_pix increases, the probability that it is a maths symbol increases to next to insignificant amount. Therefore, this model is not a good predictor of what is a maths symbol. When I plot a fitted curve, this will become a lot more noticeable.

I have plotted nr_pix on the x-axis and maths.symbol on the y-axis.

The estimate for nr_pix is 0.02450825 with the intercept being -1.63211236. When we find the exponent of the coefficient of nr_pix we find that it is 1.024813. So, if the number of pixels in a picture go up by 1 then the chances that it is a maths symbol go up by 2.4813%. This can explain the shallow orange curve in the graph.

The z-score for nr_pix is 1.224606 and intercept z-score of -2.067206. This would indicate that a value is over one standard deviation from the mean of nr_pix. As it is positive it indicates that it is higher than the mean.

The p-value of nr_pix stands at 0.22072362 with the p-value of the intercept being 0.3871473. This is quite high for a p-value. This is not statistically significant, but as it is so large, I should proceed to use it with more caution. When running the summary statistic, we see that the p-value of the intercept is 0.0382. This can be seen as significant due to the '*'.



Section 1.2

In this section of the assignemt I will use the same model again to predict whether an image is a maths symbol or not. I will evaluate this model using k fold cross validation over all 168 images.

K Fold cross validation is used as it generally results in a less biased models compared to others without it. This is because every piece of data from the dataset has a chance of appearing in the training and test sets that have been created. This is especially useful in my case as I am only using 168 images, a relatively small dataset. If I had reduced the training data, I could lose important trends within my dataset which would increase the likelihood of bias.

5 fold cross validation will involve that dataset getting divided into 5 equal parts, while using 4 of those samples to validate the 5th sample. The error estimation will then be averaged over to get an ovrall average effectiveness of my model. It will also help to prevent overfitting in my model.

Firstly, I converted the 0 in my test and training dataframe to 'no' and 1 to 'yes'. These values where then converted into type factor. This was done to ensure compatabiltiy with the confusion matrix. The confusion matrix is what I will use to calulate the accuracy, true positive rate, false positive rate, precision, recall and F1-score.

```
#Relabel values (1 = yes, 0 = no)
#1 = Maths Symbol, 0 = Not Maths Symbol

df$maths.symbol[df$maths.symbol==1] <- "yes"
df$maths.symbol[df$maths.symbol==0] <- "no"
```

I have added the libray caret to my r code in orfer to use the trainControl function. When setting up cross validation, I have created the variable ctrlspecs. From here I have used trainControl method. In this method, the first argument used is method="cv" meaning cross valiidation, followed by number = 5. This is the number of folds. I have then specified savePredictions = "TRUE" in order for all the predictions that the model makes are saved. Lastly I have set the classProbs = "TRUE" in order to save the class probabilities.

Now I will specify the logestic regression model that I have used earlier in Section 1.1. This will again use nr_pix as a predictor for predicting if an image is a maths symbol or not. Method used is glm as this is the family of analysis that logestic regression lives within. The family used is binomal as we are essentialy working with 1 and 0. trControl will equal crtlspecs that was created earlier as we will train the model using cross validation.

```
#Specify the training method used and number of folds (5 fold Cross Validation)
ctrlspecs <- trainControl(method="cv", number=5,
                          savePredictions=TRUE,
                          classProbs = TRUE)

set.seed(42)

model <- train(maths.symbol~nr_pix, data=df,
               trControl=ctrlspecs, method="glm", family="binomial")

mean(model$pred$pred==model$pred$obs)
```

When we print the model, we see that there is 1 predictor which is nr_pix with 2 classes 'yes' and 'no'. We see an accuracy score of 0.6666667. This is expected as two thirds of the dataset is not a maths symbol.

I can now go onto create a confusion matrix using a decision threshold of 0.5:

```
# Threshold of 0.5 threw an error when trying to run so 0.4 was used

cm = confusionMatrix(table(model$pred$yes >= 0.5, model$pred$obs == "yes"),
                     mode = "prec_recall")
cm


# Threshold of 0.4 used as alternative to 0.5

cm = confusionMatrix(table(model$pred$yes >= 0.4, model$pred$obs == "yes"),
                     mode = "prec_recall")
cm
#True Positive Rate
cm$byClass["Sensitivity"]

#True Negative Rate
cm$byClass["Specificity"]
```

Upon running this confusion matrix, I was greeted with an error:

```
> cm = confusionMatrix(table(model$pred$yes >= 0.5, model$pred$obs == "yes"),
+                      mode = "prec_recall")
Error in !all.equal(nrow(data), ncol(data)) : invalid argument type
> cm
function (x)
2.54 * x
<bytecode: 0x000001a88c91ea08>
<environment: namespace:grDevices>
```

Trouble shooting this issue I was unable to get the decision threshold of 0.5 to work. To work around this is used the decision threshold of 0.4. This ran without error and was one of the only values that yielded no errors. This confusion matrix will use the model to predict that probability of picking yes from the dataframe with a decision threshold of 0.4. Mode = “prec_recall” will allow me to retrieve the precsion, recall and F1 score of this confusion matrix. Once it ran, I was greeted with these values:

```
Confusion Matrix and Statistics


       FALSE TRUE
  FALSE   106   48
  TRUE      6    8

               Accuracy : 0.6786
                 95% CI : (0.6023, 0.7484)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : 0.4064

                  Kappa : 0.1099

 Mcnemar's Test P-Value : 2.414e-08

              Precision : 0.6883
                 Recall : 0.9464
                     F1 : 0.7970
             Prevalence : 0.6667
         Detection Rate : 0.6310
   Detection Prevalence : 0.9167
      Balanced Accuracy : 0.5446

       'Positive' Class : FALSE
```

We see that the true positive value was 106, the false positive was 48, the false negative was 6 and the true negative was 8. We see that it correctly predicted that 106 ‘no’ values and 48 ‘yes’ values from the data frame. It also predicted 6 ‘yes’ values when they were in fact ‘no’ and also predicted 8 ‘no’ values when they were ‘yes’.

To find the true positive rate, also known as Sensitivity, I used cm$byClass["Sensitivity"]. This returned a value of 0.9464286.

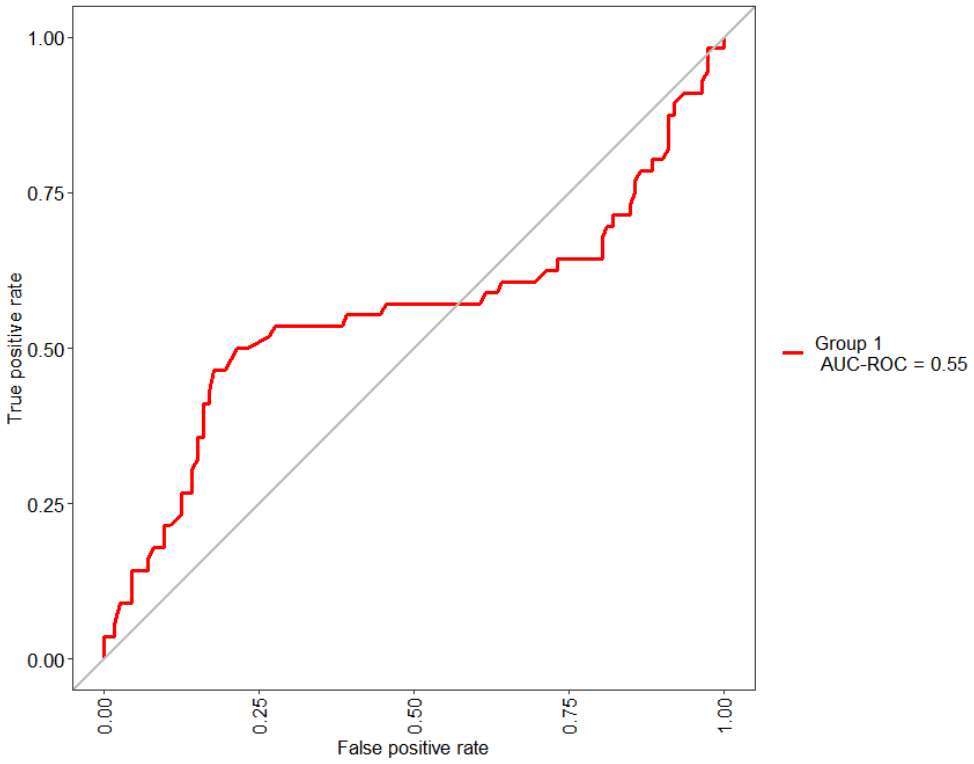
To find the True Negative Rate, also known as the Specificity, I used cm$byClass["Specificity"]. This returned a value of 0.1428571.

With a high sensitivity value, there are very few negative results as seen. The low specificity also tells us that there are a lot of false positive results (48).

The F1 score was 0.7970, Recall was 0.9464 and Precision was 0.6883. These would likely be far different had the decision threshold of 0.5 worked.

Section 1.3

From the ROC curve plotted, I can see the AUC - ROC is 0.55. This tells me the predictor of nr_pix makes random guesses.



Section 2

In this section I will be performing K Nearest Neighbour classification with all odd values of k between 1 and 25 on my dataset. KNN is a supervised learning algorithm that uses LABEL input data to predict the output of data points. It checks how similar a data point is to its neighbour ad classifies the data point into the class it is most similar with. I will use the first 6 features of my features data frame to calculate the accuracy using KNN. With the data frame imported and the first 6 features selected, I then add a new column called 'type' and add the values 'Letter', 'Digit' and 'Symbol' to the corresponding value in LABEL. These will be my three classifiers. Once this has been done, I will remove the LABEL from the dataframe. The 6 columns used for predictors will be set to numeric. The factor will allow me to calculate the accuracy using K Nearest Neighbour. 'Type' is not a predictor so it will not be used. After this has been applied, I must normalise the data frame so that the output remains unbiased.

```
#Normalize function used so KNN can be performed on df
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

#Apply Normalize function to each column apart from LABEL
df.norm <- as.data.frame(lapply(df[,2:7], normalize))
head(df.norm)
```

I will then shuffle the dataframe and split the dataframe into training and testing datasets. I will also create a matching training and testing dataset with the LABEL column used. This is for the confusion matrix.

```
# Training dataframe will use all 168 features

train_df = df_shuffled[1:168,]
test_df = df_shuffled[1:168,]


df_train_target <- df[1:168, 1]
df_test_target <- df[1:168,1]
```

Now that I have cleaned up my data frame, I am able to apply K nearest neighbour. To do this more efficiently I have created a loop that calculates the accuracy of the KNN model ranging from 1 to 25 with only the odd numbers being selected. This was done creating a modulus function to only use numbers divided by 2 that leave a remainder. This odd number function was then applied to the for loop. As it is only applying the odd K values, the even values will return as NA. These values will be omitted so they do not interfere with the graph. When the loop runs, we can then check what the most accurate K value is.

```
107 set.seed(42)
108 i=1
109 k.trainingsets=1
110 #function that will only pick odd numbers
111 oddnumbers <- function(x) x[ x %% 2 == 1 ]
112
113 for (i in oddnumbers(1:25)){
114   knn.mod <- knn(train=train_df, test=test_df, cl=df_train_target, k=i)
115   k.trainingsets[i] <- 100 * sum(df_test_target == knn.mod)/NROW(df_test_target)
116   k=i
117   cat(k,'=',k.trainingsets[i],'')
118 }
119
120 k.trainingsets
121
122 #Remove NA values from k.trainingsets has it will interfere with graph
123 k.trainingsets <- na.omit(k.trainingsets)
```

This accuracy function was then applied to all the confusion matrixes. This then give me a percentage to see how accurate each model was:

- k1 = 97.02381
- k3 = 35.11905
- k5 = 29.16667
- k7 = 20.23810
- k9 = 23.80952
- k11 = 16.07143
- k13 = 19.04762
- k15 = 17.85714
- k17 = 15.47619
- k19 = 16.07143
- k21 = 17.26190
- k23 = 15.47619
- k25 = 15.47619

You can see that the model is most accurate when K = 1. As I am calculating the accuracy of all 168 images with the training data frame, when K = 1 I get close to 100% as the values have already been seen and a rough boundary is formed when I use k = 1. Therefore, I see such a steep decline between k = 1 to k = 3 from 97.02381% to 35.11905%. Had I used an unseen training dataset I would likely see a much lower value for k = 1. I can also see that the model is least accurate when k = 23 and 25 at 15.47619%.

When we create a graph for this model, we from around K = 13 that the accuracy to K = 25 stays somewhat similar, all K values inclusive between this being within a 4%.

Introduction

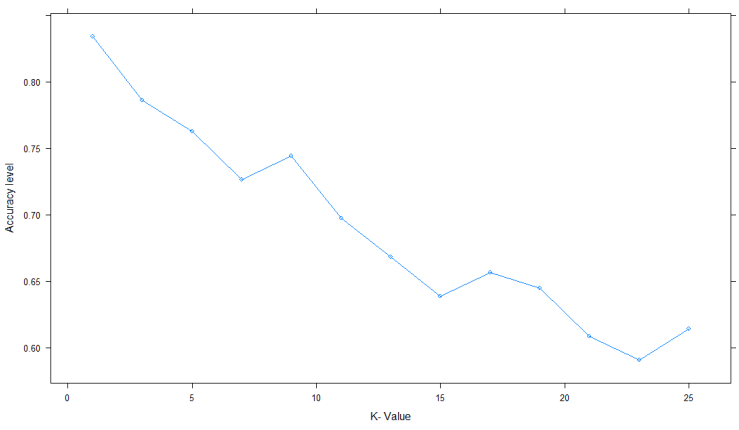
I will now k nearest neighbour on the first 6 features of my dataframe with all odd values of k between 1 and 25 but using 5-fold cross validation. K Fold cross validation is used as it generally results in a less biased models compared to others without it. This is because every piece of data from the dataset has a chance of appearing in the training and test sets that have been created. This is especially usual in my case as I am only using 168 images, a relatively small dataset.

First, I have set up my trainControl method to using 5-fold cross validation. This will be stored in the variable trControl. Instead of calculating each k value separately, I have created a variable called k_values that stores all odd k values between 1 to 25. This will be used in tuneGrid of the train function.

```
141  #Section 2.2 - KNN and 5-Cross Validation
142
143  trControl <- trainControl(method  = "cv",
144                            number  = 5,
145                            savePredictions = TRUE)
146
147  # K values 1 - 25 (odd numbers inclusive)
148  set.seed(42)
149  k_values <- seq(from=1, to=25, by=2)
150
151
152  fit_all_k <- train(type ~ .,
153                     method     = "knn",
154                     tuneGrid   = expand.grid(k = k_values),
155                     trControl  = trControl,
156                     metric     = "Accuracy",
157                     data       = new_df)
158  fit_all_k
159
```

Fit_all_k is the variable used to store the model. Once we run the model, we see immediately that K = 1 is still the most accurate K nearest neighbour. However, the accuracy drop off is not as steep now using 5-fold cross validation. We can plot this data on a graph to give us a clearer picture to how the accuracy drops off as k increases.

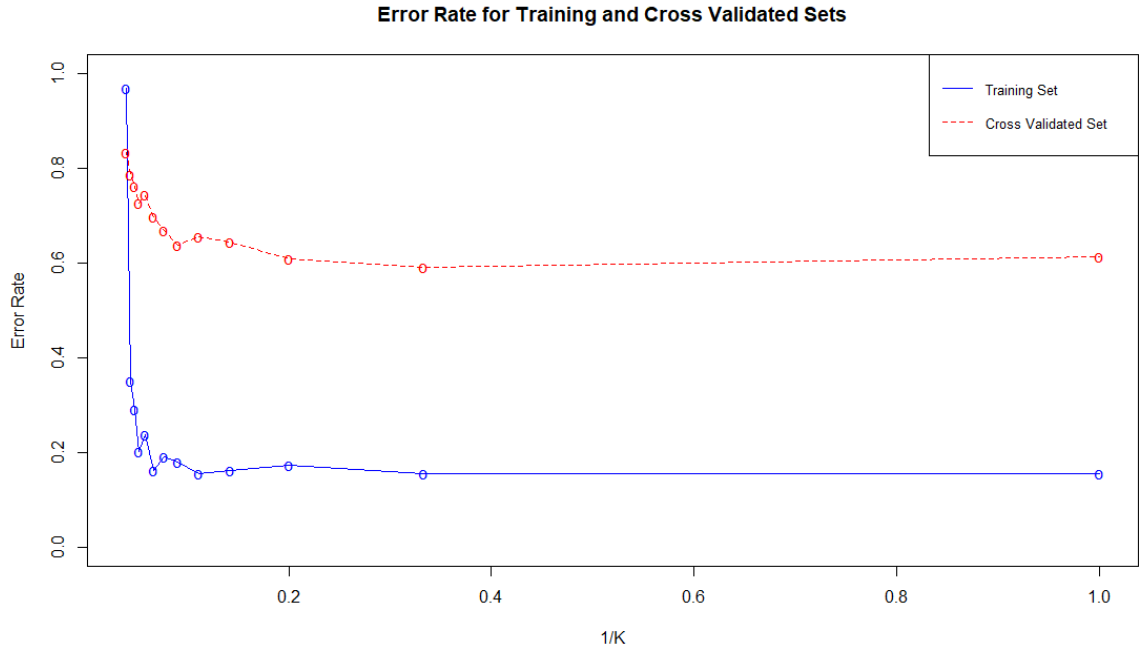
```
k-Nearest Neighbors

168 samples
  6 predictor
  3 classes: 'Digit', 'Letter', 'Number'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 135, 135, 135, 134, 133
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.8343112  0.7517056
   3  0.7863713  0.6793623
   5  0.7626636  0.6437748
   7  0.7261319  0.5887336
   9  0.7441355  0.6159073
  11  0.6972142  0.5451283
  13  0.6679399  0.5010573
  15  0.6383499  0.4568469
  17  0.6565215  0.4838669
  19  0.6445786  0.4659677
  21  0.6082047  0.4112725
  23  0.5903794  0.3845878
  25  0.6139292  0.4205756
```



From what my model has told me, there is a gradual drop off in accuarcy as K increases, but there is no steep decline in accuracy. The model is least accurate at 59.1% at k = 23 but this is still a fair accuracy value when comapred to 15.47% when k=23 was used without 5 fold cross validation. The model is again most accuate at k=1 but this time it is 83.4%, not as high as it was when cross validation wasn't used for k=1.

When I went about creating the graph for I stored k.trainingsets and fit_all_k in y1 and y2 respectively. I was unable to take the variable kit_all_k and stored it in y2. To rectify this I hard coded the values I calculated from fit_all_k into it. Before this however, I ran my R code muliple times with the seed set to 42 to ensure each cross validated value of k was the same each time. This was the case so I went ahead with this method.

I then created a graph showing the error rate between the cross validated set and the training set with 1/k on the x-axis. As you can see there is a massive difference bewteen both sets. The error rate of the training set drops significantly with 1/23. It hoovers jus under 0.2 for the from 1/15 onwards. We see from the cross validated training set that the drop in error rate is less substantual comapred to the training set. We can see that it hoovers just under 0.6 at 1/3. Overall this graph tells us that when the dataset has been cross validated, the drop off in accuracy is less severe.



Section 3

Random forest is a machone learning algorithm that fits many classifications to random subsets of the input data, in this case the 2100 doodle sets that I will be using, and used the combined resut for the prediction. It also avoids overfitting.

I will be using the libraries randomeForest, mlbench and caret for this section.

Reading in the the large doodle set, I used read.delim as the file was tab spaced. From here I was able to add the column names to their corresponding columns. LABEL will be set as a factor as it will not be used in random forest.

Using the caret package, I am to use the control function which I have named trainControl. I have used the method 'repeatedcv' with a cross validation of 5 and repeat of 3. The reason for using repeatedcv over cv is

mainly due to the ability to repeat. As I am woring with 2100 rows of data a repeated cross validation will ensure I get the most accueate reading and reduce overfitting overall.

```
###Section 3.1

set.seed(42)

#Manual search by create 5 folds and repeat 3 times
control <- trainControl(method = 'repeatedcv',
                        number = 5,
                        repeats = 3,
                        search = 'grid')
```

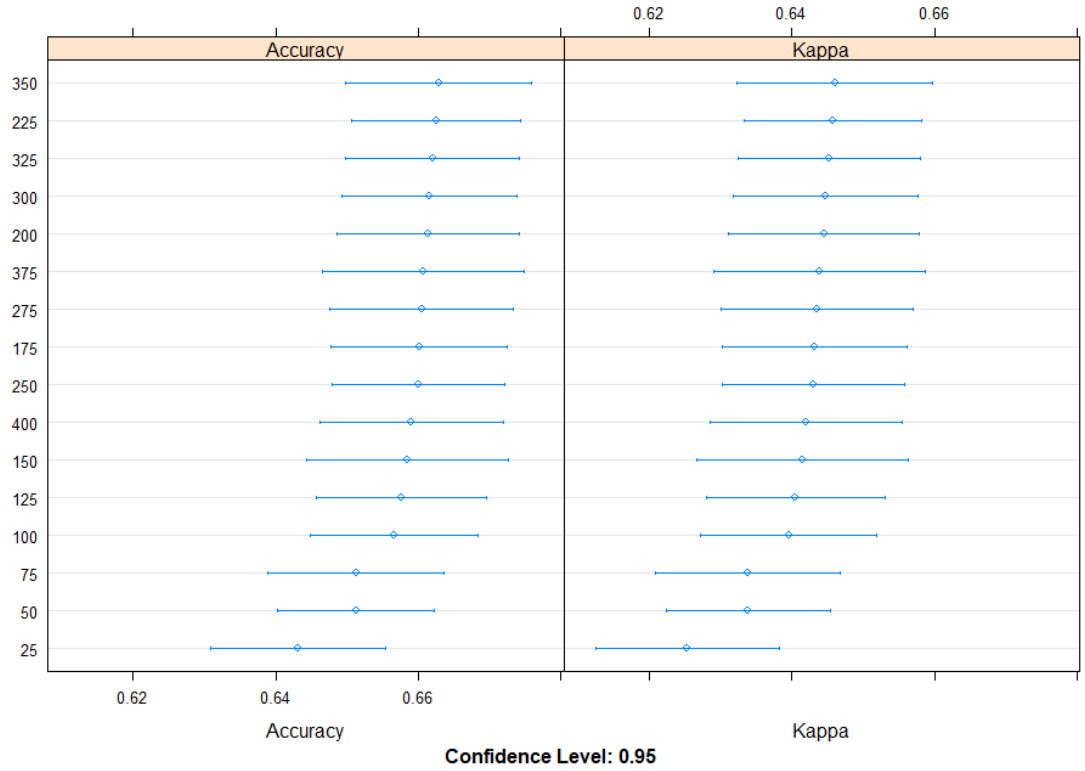
When setting the number of predictors at each node, I have used the expand.grid function and set the predictiors to increment by two, starting from two and making its way up to eight. Mtry is what is used to set the predictor. This will be stored in the variable tunegrid.

```
#Set mtry to be 2 and increment by 2 all the way to 8
tunegrid <- expand.grid(.mtry = seq(from=2, to=8, by=2))
rf_gridsearch <- list()
```

I have used a for loop for the ntree(number of trees), this way the model will loop through the ntree values and find the accuracy for each. To train the model I have removed the Index and zero_bottom_right as these values have no significance in predicting how accurate the model will be. The method used is ‘rf’ for random forest and the metric will be ‘Accuracy’. Tunegrid and control that I created earlier are also used within the train function as well as ntree. The variable rf_gridsearch is where this will all be stored.

```
#train with different ntree parameters, 25 increments up to 400
for (ntree in c(25,50,75,100,125,150,175,200,225,250,275,300,325,350,375,400)){
  set.seed(42)
  modelfit <- train(LABEL~.-Index -zero_bottom_right,
                    data = data,
                    method = 'rf',
                    metric = 'Accuracy',
                    tuneGrid = tunegrid,
                    trControl = control,
                    ntree = ntree)
  key <- toString(ntree)
  rf_gridsearch[[key]] <- modelfit
}
```

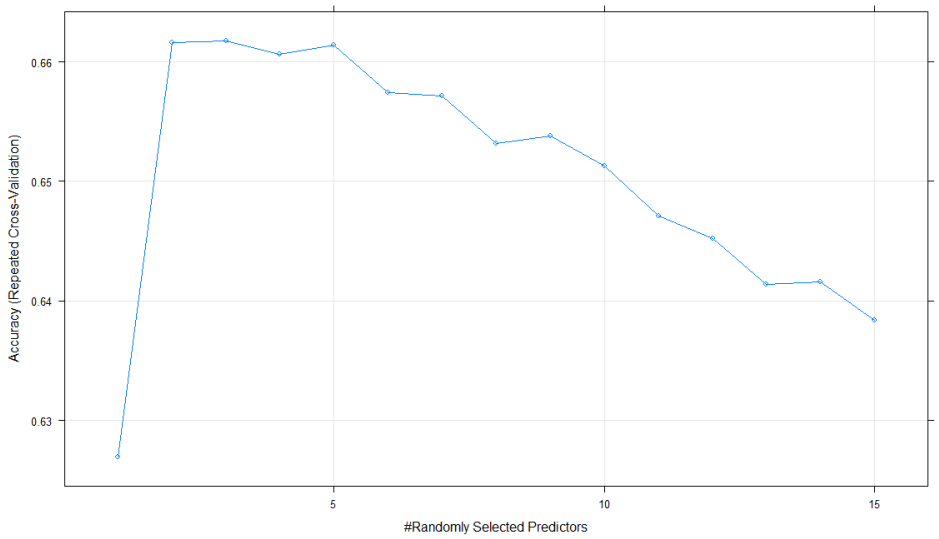
This will take some time to run due to the large doodle set and repeat of 3 used. Once it has finally ran I can use dotplot to plot my results. As you can see from the graph, my model was most accurate when ntree of 350 with accuracy of 66.28670%. This graphs also takes into account the predictors used {2,4,6,8}. There is very little difference bewteen each accuracy for ntree. This may be in part to using repeated cross validation. Without this the range bewteen 1st and 3rd quartile may have been greater.



Section 3.2

As there is an element of randomness in cross validation and random forest, I will implement 15 independent runs and see how the accuracy is affected. To do this I set my mtry value(predictors) to go from 1:15. This will create 15 independent accuracy values. Ntree will be set to 350 as in section 3.1 I found that it give the most accurate readings. The search method within trainControl will be set to 'random'. The train

method will be stored within the variable rf_independent. Once it has ran and been plotted, we can see that when the mtry value is equal to 3, the model is most accurate at 66.17385%. After 3 predictors, we can see that the accuracy of the model decreased slightly, until it is at its least accurate at mtry with a percentage of 63.83979. The least accurate value recorded was at mtry 1 with a percentage of 62.69625. Although the range between the 2 is rather small but because there are 2100 items within the doodle set, over 3 percent change in accuracy can make a very significant difference.



When we run rf_independent$results we are also able to see the mean Standard Deviation and Mean results for each of the 15 runs. At mtry 13 we can see that the standard deviation is the highest at 0.02380087. The lowest standard deviation value is at mtry 3 at 0.01499643. Below is all the Standard Deviation and Means for the accuarcy of this model.

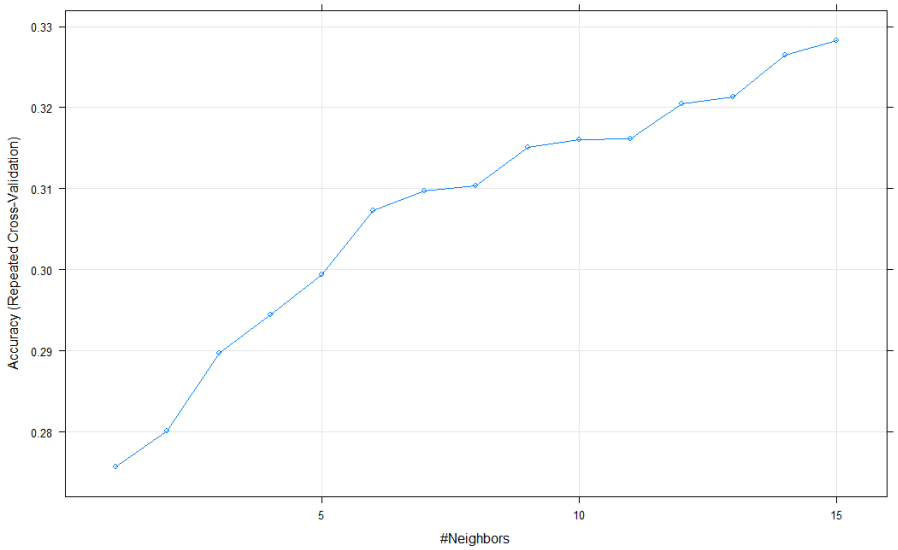
```
> rf_independent$results
   mtry  Accuracy     Kappa AccuracySD    KappaSD
1     1 0.6269625 0.6083105 0.02283001 0.02397182
2     2 0.6615831 0.6446619 0.01811204 0.01901720
3     3 0.6617385 0.6448247 0.01499643 0.01574679
4     4 0.6606296 0.6436601 0.01688361 0.01772818
5     5 0.6614233 0.6444932 0.01720090 0.01806140
6     6 0.6574512 0.6403229 0.01762200 0.01850357
7     7 0.6571338 0.6399893 0.02082645 0.02186838
8     8 0.6531632 0.6358201 0.01590461 0.01670102
9     9 0.6537993 0.6364878 0.01509596 0.01585185
10   10 0.6512570 0.6338188 0.02000609 0.02100717
11   11 0.6471281 0.6294830 0.01831188 0.01922840
12   12 0.6452237 0.6274838 0.01875611 0.01969451
13   13 0.6414108 0.6234796 0.02380087 0.02499194
14   14 0.6415729 0.6236502 0.01693354 0.01778070
15   15 0.6383979 0.6203161 0.02097989 0.02202943
>
```

Section 3.3

In this section I will create 2 models, a KNN model and a random forest model. Each model will use 5 fold cross validation in order to be as accurate as it can be. From section 3.2 I found that 3 predictors yielded th most accurate results from my random forest model. For this sub section I will use 3 predictors in each model. They will be height, width and rows_with_3p. Height was picked as it finds the distance bewteen the top most pixel and the bottom most pixel. This can be useful in discrimating between the different symbols e.g, all 'c's may have around the same height while 'a' will have a very different height compared to 'c' but all 'a's. This applies to the other 19 symbols in the doodle set. That's why I have choosen this variable. This also applies to width. I also picked rows_with_3p. This was my third predictor as '<' may not contain a row of black pixels that is 3 or more whereas 'e' would contain more. This feature is useful to discriminate between symbols. These will be used for my model.

```
#Section 3.3 Best Model Random Forest KNN

controlKNN <- trainControl(method='repeatedcv',
                           number=5,
                           repeats = 3,
                           savePredictions = TRUE)

#KNN
set.seed(42)
k_values <- seq(from=1, to=15, by=1)


knn_section3 <- train(LABEL ~ height + width + rows_with_3p,
                      method     = "knn",
                      tuneGrid   = expand.grid(k = k_values),
                      trControl  = controlKNN,
                      metric     = "Accuracy",
                      data       = data)

knn_section3

plot(knn_section3)
```

My K-nearest neighbour k value will range 1 to 15 inclusive. From the plot below we can see it is far less accurate comapred to the random forest models from earlier in the section. Unfortunately, from the 3 randomly selected predictors in section 3.2 we saw the accuracy around 66% compared to just under 33% in the KNN model when k=15. However, the model did get more accurate as K increased. Therefore the best model to use would be the random forest model utilised in section 3.2.



Had this model been done again, I would utilise a 10 fold cross validation with 5 repeats as I am working with far more data in this section comapred to the previous 2. Or perhaps use more predictors but as the random forest model was most accurate with 3 predictors I decided against this for section 3.3.

Conclusion

In completing this assignment, I have found that using cross validation with any machine learning algorithms will result in more accurate readings of a dataset. Using such algorithms such as logistic regression, K Nearest Neighbour and Random Forest I have found that each has their benefits and weaknesses when analysing different data. I have become more proficient in using these machine learning algorithms.