Assignment 3: Machine Learning

Eunan Diamond (40293751)

Introduction

The goals of this assignment is to use features developed in assignment 2 to do classification and machine learning using r code and then use my knowledge to interepet the results that my calculations in r code give.

```
set.seed(42)
library(rlang)
library(class)
library(knitr)
library("rstudioapi")
library(MASS)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
search()
```

```
##  [1] ".GlobalEnv"         "package:caret"      "package:lattice"
##  [4] "package:ggplot2"    "package:MASS"       "package:rstudioapi"
##  [7] "package:knitr"      "package:class"      "package:rlang"
## [10] "package:stats"      "package:graphics"   "package:grDevices"
## [13] "package:utils"      "package:datasets"   "package:methods"
## [16] "Autoloads"          "package:base"
```

```
setwd(dirname(getActiveDocumentContext()$path))
getwd()
```

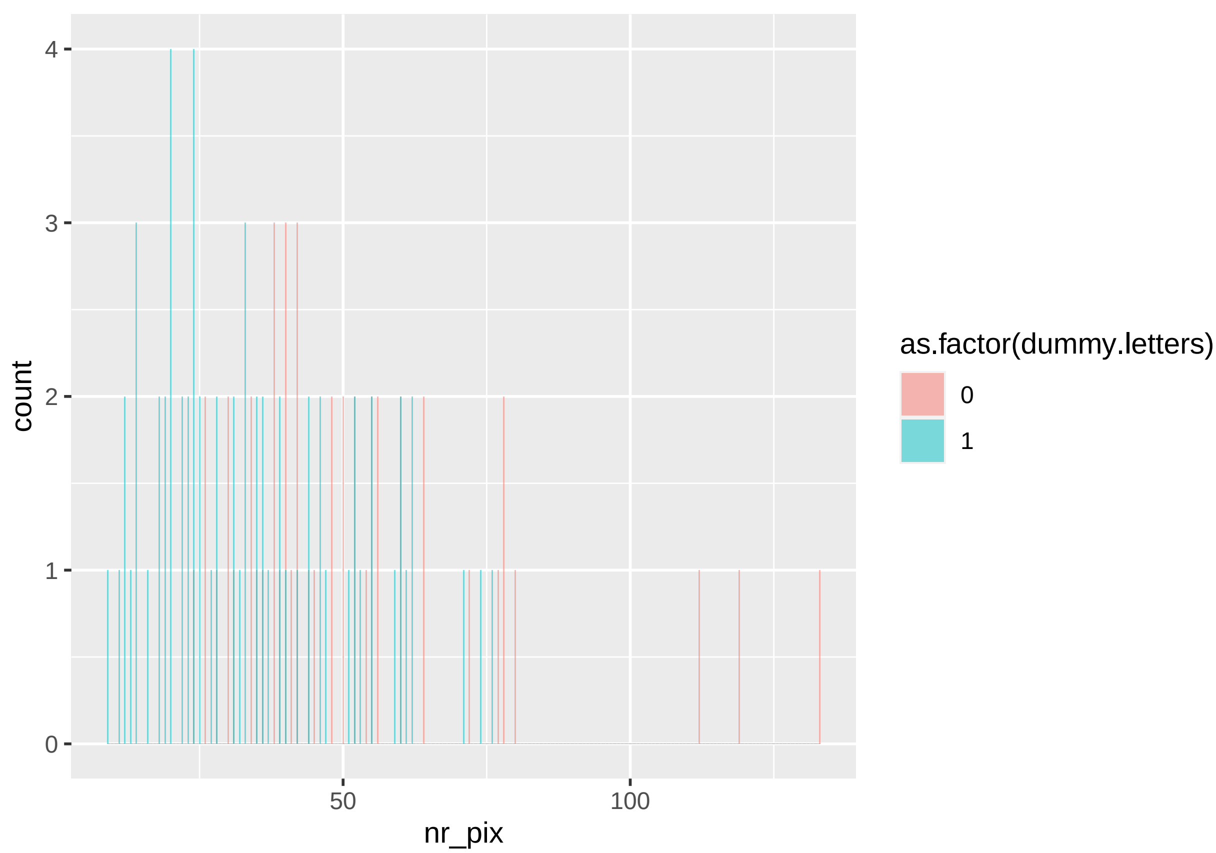
```
## [1] "C:/Users/eunan/Documents/assignment3_40293751"
```

Section 1

```
csvF<-read.csv(file ="40293751_features.csv" ,header = TRUE)

  #Add dummy value 1 for letters, so it discriminates between letters and non letters
  csvF$dummy.letters<-0
  print(csvF$dummy.letters)
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
csvF[1:80,19]<-1

#Shuffle the rows, so the first 80% of the shuffled data could be training data making it random
#and last 20% test data
features_shuffled<-csvF[sample(nrow(csvF)),]

#first 80% will be used as training data, last 20% test data
trainingdata=features_shuffled[1:112,]
testData=features_shuffled[113:140,]

plt <- ggplot(trainingdata, aes(x=nr_pix, fill=as.factor(dummy.letters))) +
  geom_histogram(binwidth=.2, alpha=.5, position='identity')
plot(plt)
```



Section 1.1

```
glmFit<-glm(dummy.letters ~ nr_pix+aspect_ratio,
              data = trainingdata,
              family = 'binomial')

  print(summary(glmFit))
```

```
##
## Call:
## glm(formula = dummy.letters ~ nr_pix + aspect_ratio, family = "binomial",
##     data = trainingdata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0340  -0.9431   0.4928   0.8376   2.0126
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   3.24582    0.67767   4.790 1.67e-06 ***
## nr_pix       -0.03493    0.01202  -2.907  0.00365 **
## aspect_ratio -2.12046    0.66999  -3.165  0.00155 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 152.36  on 111  degrees of freedom
## Residual deviance: 122.36  on 109  degrees of freedom
## AIC: 128.36
##
## Number of Fisher Scoring iterations: 4
```

```
dataframematrix <- matrix(nrow=28,ncol=2)
dataframematrix[,1] <- testData$nr_pix
dataframematrix[,2] <- testData$aspect_ratio

newdata <-as.data.frame(dataframematrix)
colnames(newdata) = c("nr_pix","aspect_ratio")
predicted = predict(glmFit,newdata,type="response")
print(testData)
```

```
##      label index nr_pix rows_with_1 cols_with_1 rows_with_3p cols_with_3p
## 126 xclaim     6     28           0           0            0            2
## 112 smiley    12     38           2           0            4            7
## 91     sad    11     60           0           1           10            9
## 132 xclaim    12     62           0           0           14            5
## 82     sad     2     50           0           2            7           10
## 11       b     3     50           0           0            8            6
## 61       h     5     43           0           0            9            3
## 37       e     5     24           2           0            2            8
## 106 smiley     6     54           0           0            8            8
## 135 xclaim    15     58           0           0           12            4
## 59       h     3     43           0           0            7            6
## 70       i     6     20           0           0            0            2
## 28       d     4     53           0           0           12            7
## 62       h     6     49           0           0            9            4
## 76       j     4     13           8           2            1            1
## 134 xclaim    14     56           0           0           12            4
## 123 xclaim     3     65           0           0           13            5
## 94     sad    14     46           0           4            5           10
## 56       g     8     16          11           3            1            1
```

```
## 7        a     7     80           0           0          12           9
## 23       c     7     54           0           0           8           9
## 12       b     4     69           0           0          12           8
## 45       f     5     51           0           0          15           4
## 81     sad     1     32           0           5           5           9
## 60       h     4     22           5           5           2           2
## 46       f     6     55           0           0          15           5
## 103 smiley     3     54           0           0           7          10
## 85     sad     5     54           0           0           8           8
##     aspect_ratio neigh_1 no_neigh_above no_neigh_below no_neigh_left
## 126   0.07142857       0              4              4            14
## 112   1.10000000       2             17             17             8
## 91    0.92857143       0             19             19            14
## 132   0.28571429       0              8              8            14
## 82    1.08333333       2             20             22             9
## 11    0.42857143       0              2              4            16
## 61    0.58333333       1              5              4            18
## 37    0.87500000       1              9             10             3
## 106   0.91666667       2             18             16            14
## 135   0.17647059       0              2              4            15
## 59    0.77777778       0              6              4            14
## 70    0.09090909       2              4              4            10
## 28    0.40000000       0              1              3            14
## 62    0.66666667       0              5              5            18
## 76    0.30000000       1              3              4             8
## 134   0.18750000       0              2              6            15
## 123   0.28571429       0             10             10            13
## 94    1.08333333       2             20             18            10
## 56    0.33333333       2              1              2             9
## 7     0.72727273       0              9             12             7
## 23    0.80000000       0             13             13            13
## 12    0.43750000       0              4              6            19
## 45    0.42857143       0              3              4            11
## 81    1.30000000       2             21             19            10
## 60    0.54545455       3              4              3            16
## 46    0.50000000       0              5              6            11
## 103   1.08333333       0             18             16             7
## 85    0.84615385       2             16             18            14
##     no_neigh_right no_neigh_horiz no_neigh_vert connected_areas eyes custom
## 126             14              0             0               2    0      2
## 112              8              4             0               4    0     12
## 91              14              2             4               4    0     10
## 132             14              0             0               2    0      5
## 82               9              2             4               4    0     12
## 11              10              0             2               1    1      5
## 61              16              6             0               1    0      7
## 37               3             10            19               1    1      8
## 106             14              0             6               4    0      8
## 135             15              0             0               2    0      4
## 59              11              0             0               1    0      8
## 70              10              0             2               2    0      2
## 28              18              2             1               1    1      4
## 62              15              0             0               1    0      7
## 76               9              8             3               2    0      3
```

```
## 134             15              0              0              2    0      4
## 123             13              0              0              2    0      5
## 94              10              4             12              4    0     14
## 56               9             13              5              1    0      3
## 7               11              0              0              1    1      9
## 23              11              0              0              1    0      8
## 12              17              0              1              1    1      7
## 45              12              1              0              1    0      5
## 81              10              4             12              4    0     14
## 60              14             17              5              1    0      4
## 46              12              1              0              1    0      7
## 103              7              0              2              4    0     12
## 85              14              0              6              4    0      8
##     dummy.letters
## 126             0
## 112             0
## 91              0
## 132             0
## 82              0
## 11              1
## 61              1
## 37              1
## 106             0
## 135             0
## 59              1
## 70              1
## 28              1
## 62              1
## 76              1
## 134             0
## 123             0
## 94              0
## 56              1
## 7               1
## 23              1
## 12              1
## 45              1
## 81              0
## 60              1
## 46              1
## 103             0
## 85              0
```

```
  print(predicted)
```

```
##         1         2         3         4         5         6         7         8
## 0.8924739 0.3979300 0.3059563 0.6163761 0.3104665 0.6434696 0.6240590 0.6346115
##         9        10        11        12        13        14        15        16
## 0.3579535 0.6996368 0.5236078 0.9132842 0.6332637 0.5300927 0.8961877 0.7093121
##        17        18        19        20        21        22        23        24
## 0.5913138 0.3411414 0.8786906 0.2514516 0.4165698 0.4769704 0.6354161 0.3478286
##        25        26        27        28
## 0.7892986 0.5657015 0.2813718 0.3929955
```

```
  testData[["predicted_val"]] = predict(glmFit, testData, type="response")
  testData[["predicted_class"]] = 0
  testData[["predicted_class"]][testData[["predicted_val"]] > 0.5] = 1

  correct_items = testData[["predicted_class"]] == testData[["dummy.letters"]]
   correct_items
```

```
##  [1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE
```

```
  nrow(testData[correct_items,])/nrow(testData)
```

```
## [1] 0.7142857
```

```
  PredictedData <- as.factor(testData[["predicted_class"]])
  PredictedData
```

```
##  [1] 1 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 1 1 0 0
## Levels: 0 1
```

```
  ActualData <- as.factor(testData[["dummy.letters"]])
  ActualData
```

```
##  [1] 0 0 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 0 0
## Levels: 0 1
```

```
  confMatrix <- confusionMatrix(PredictedData,ActualData)
  confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  8  3
##          1  5 12
##
##                Accuracy : 0.7143
##                  95% CI : (0.5133, 0.8678)
##     No Information Rate : 0.5357
##     P-Value [Acc > NIR] : 0.04232
##
##                   Kappa : 0.4197
##
##  Mcnemar's Test P-Value : 0.72367
##
##             Sensitivity : 0.6154
##             Specificity : 0.8000
##          Pos Pred Value : 0.7273
##          Neg Pred Value : 0.7059
```

```
##              Prevalence : 0.4643
##          Detection Rate : 0.2857
##    Detection Prevalence : 0.3929
##       Balanced Accuracy : 0.7077
##
##        'Positive' Class : 0
##
```

```
  confMatrix <- confusionMatrix(PredictedData,ActualData,mode = "prec_recall")
  confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  8  3
##          1  5 12
##
##                Accuracy : 0.7143
##                  95% CI : (0.5133, 0.8678)
##     No Information Rate : 0.5357
##     P-Value [Acc > NIR] : 0.04232
##
##                   Kappa : 0.4197
##
##  Mcnemar's Test P-Value : 0.72367
##
##               Precision : 0.7273
##                  Recall : 0.6154
##                      F1 : 0.6667
##              Prevalence : 0.4643
##          Detection Rate : 0.2857
##    Detection Prevalence : 0.3929
##       Balanced Accuracy : 0.7077
##
##        'Positive' Class : 0
##
```

The code above shows the logistic regression predictions for number of pixels and aspect ratio for classification to check if an image belongs to letter or not. The training data (80% of the total images) was used to train the model using the glm function which fitted the logistic regression model. The predict function then used the test data to check the probability if they are a letter or not. Out of the 28 test data images 14 of them are letters. The table shows which ones were predicted letters or not, if there probability of being a letter based of number of pixels and aspect ratio was greater than 0.5. The correct_items shows which of the test data were predicted correctly. 21 out of 28 were predicted correctly that they were not letter or not. This shows that number of pixels and aspect ratio together are two good features to use to predict if the image is a letter or not. The most confident image of the test data to be an image was the first image with 0.8767535, which was indeed a letter it is a j. The least confident was the 3rd image with it only predicting 0.1273884 for it to be a letter it was as expected not a letter and was a smiley image. To add the logistic regression model tells us that the smaller the aspect ratio the more chance it is a letter with a estimate value of -2.12046, the estimate value, and the same for number pixels, with a negative value also with -0.03493, this tells us the lower the number of black pixels and aspect ratio the bigger chance it is a letter.

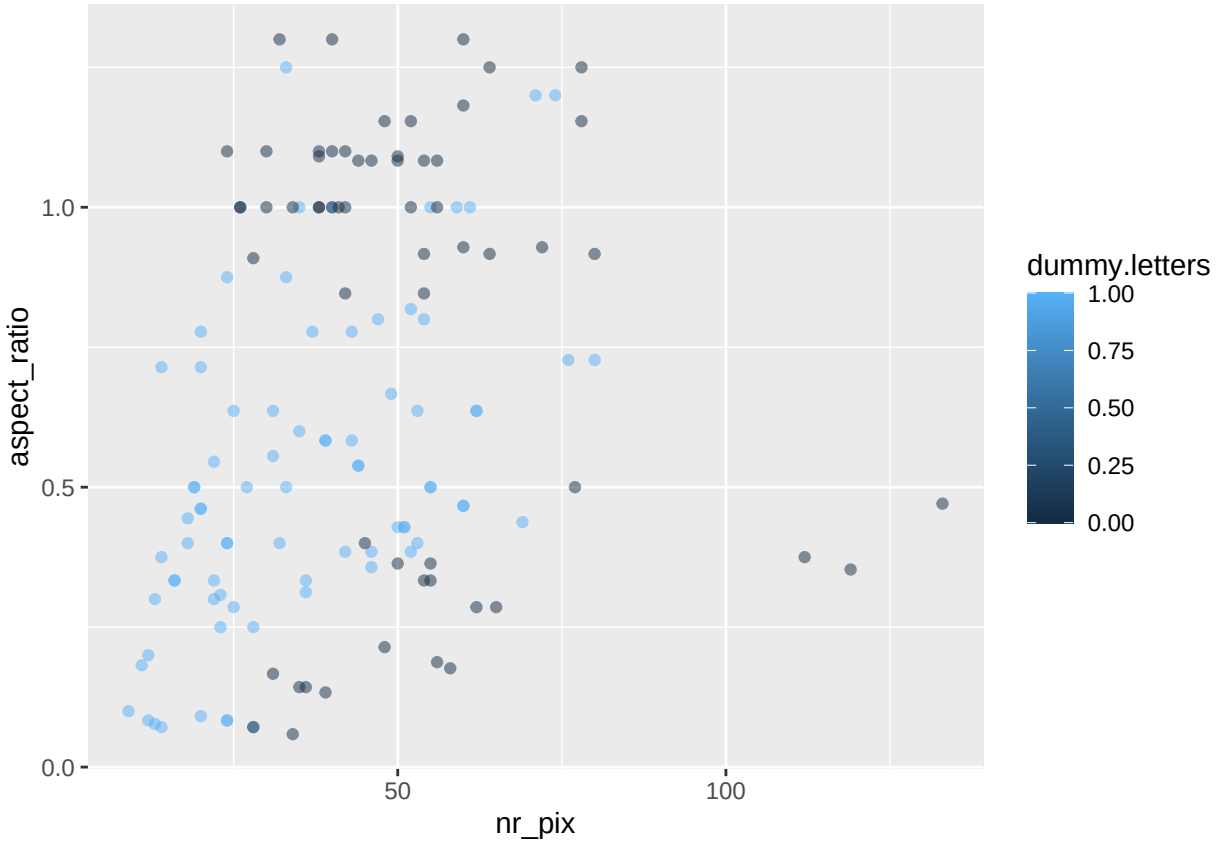
To continue, Using this logistic regression model I got the confusion matrix which gave me the the accuracy,true positive rate (sensitivity) and specificity which take 1 away from it can give false positive rate.

Also using mode ="prec_recall" on confusion matrix gives precision recall and f1.The accuracy of the model is 0.7143, this shows that this model is mostly reliable, a lot better than chance (50%) but not completely and can be improved on with more features . As the classes are nearly balanced(13 to non letters and 15 to letters) this means accuracy is a good measurement if it is a good model or not. The true positive rate for this model is 0.6154 , this tells us that 61.54% of it correctly predicted that a image which is a letter as a letter. This is a nearly 10% lower than the accuracy, although it is still better than chance, it shows it is not the best at predicting that a image which is a letter is actually a letter . The false positive rate can be calculated by taking 0.8000 (which is the specificity or true negative rate) away from 1, which gives 1 - 0.8000 = 0.2, this tells us for time that the model predicts that it is not a letter it is wrong 0.2% of the time. This tells us it is more accurate when it predicts for the class that isn't a letter. The confusion matrix also shows that the precision is 0.7273. The precision is the total number of images predicted correctly divided by total number of images predicted as a letter. This can be seen as a more accurate measurement of accuracy in classication as the total number of the two different classes may not be the same, as is the case here (15 to 13). the precision is just greater than the accuracy and therefore our model may be slightly more reliable than we think. The recall is the of images that are letters that were predicted correctly divided by the total number of images that are letters, here the recall is 0.6154, this tells us that the the for all images that are letters 61.54% of them were predicted correctly, this is same as true positive rate, as it is telling us the same thing. The F1 is the harmonic mean between precision and recall. so basically it combines the recall and precision into one metric. Here the f1 score is 0.6667, this basically tells us there is quite a difference between recall and precision, as the F1 score is over 5% away from precision and over 6% away for recall, this tells us that this may not be the most accuare model and there can be improvements. To add as the test data has 15 letters and 13 non letter images it tells us that F1 score is the the best metric to tell if it is a good classifer or not as accuracy is the best suited when the 2 classes are not equal, therefore 0.6667 tells us that the model is better than chance but there can be improvements to make it a more reliable model.

Section 1.2

```
kfoldsk = 5
```

```
ggplot(csvF, aes(x=nr_pix, y=aspect_ratio, color=dummy.letters)) +
  geom_point(alpha=.5, position='identity')
```




```
train_control<-trainControl(method="cv",number=kfoldsk,savePredictions = T, classProbs = TRUE)
csvF$is.letter <- "no"
csvF$is.letter
```

```
##   [1] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [16] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [31] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [46] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [61] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [76] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
##  [91] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
## [106] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
## [121] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"
## [136] "no" "no" "no" "no" "no"
```

```
csvF$is.letter[csvF$dummy.letters == 1] <- "yes"
csvF$is.letter
```

```
##   [1] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [13] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [25] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [37] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [49] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [61] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes"
##  [73] "yes" "yes" "yes" "yes" "yes" "yes" "yes" "yes" "no"  "no"  "no"  "no"
```

```
##  [85] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
##  [97] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
## [109] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
## [121] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
## [133] "no"  "no"  "no"  "no"  "no"  "no"  "no"  "no"
```

```
model<-train(is.letter~nr_pix+aspect_ratio, data = csvF,trControl=train_control,method="glm",family = "
print(model)
```

```
## Generalized Linear Model
##
## 140 samples
##   2 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 112, 112, 112, 112, 112
## Resampling results:
##
##   Accuracy   Kappa
##   0.7071429  0.3988596
```

```
model$results
```

```
##   parameter  Accuracy     Kappa AccuracySD   KappaSD
## 1      none 0.7071429 0.3988596 0.09244414 0.1874869
```

```
summary(model)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0775  -0.9469   0.4885   0.9308   2.0038
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   3.16025    0.62295   5.073 3.92e-07 ***
## nr_pix       -0.03501    0.01092  -3.207 0.001343 **
## aspect_ratio -2.02742    0.58320  -3.476 0.000508 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 191.21  on 139  degrees of freedom
## Residual deviance: 157.46  on 137  degrees of freedom
## AIC: 163.46
##
## Number of Fisher Scoring iterations: 4
```

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

```
## [1] 0.7071429
```

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

```
## Confusion Matrix and Statistics
##
##
##         FALSE TRUE
##   FALSE    38   19
##   TRUE     22   61
##
##                Accuracy : 0.7071
##                  95% CI : (0.6243, 0.7809)
##     No Information Rate : 0.5714
##     P-Value [Acc > NIR] : 0.0006494
##
##                   Kappa : 0.3983
##
##  Mcnemar's Test P-Value : 0.7547764
##
##             Sensitivity : 0.6333
##             Specificity : 0.7625
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.7349
##              Prevalence : 0.4286
##          Detection Rate : 0.2714
##    Detection Prevalence : 0.4071
##       Balanced Accuracy : 0.6979
##
##        'Positive' Class : FALSE
##
```

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

```
## Confusion Matrix and Statistics
##
##
##         FALSE TRUE
##   FALSE    38   19
##   TRUE     22   61
##
##                Accuracy : 0.7071
##                  95% CI : (0.6243, 0.7809)
##     No Information Rate : 0.5714
##     P-Value [Acc > NIR] : 0.0006494
##
```

```
##                   Kappa : 0.3983
##
##  Mcnemar's Test P-Value : 0.7547764
##
##               Precision : 0.6667
##                  Recall : 0.6333
##                      F1 : 0.6496
##              Prevalence : 0.4286
##          Detection Rate : 0.2714
##    Detection Prevalence : 0.4071
##       Balanced Accuracy : 0.6979
##
##        'Positive' Class : FALSE
##
```

In this part I used 5 fold cross validation to do the same analysis as in 1.1. Doing cross validation can make us better understand what's going on as instead of just training 1 model like 1.1 code does this trains 5 models , and then it calculates the model based of all 5, Therefore it is maximizing the amount of data used to train the model, every part of the data is used as testing data once, therefore this more accurately predicts how useful these features are in predicting if it's a letter or not for unseen data. Cross-Validation is a technique used in model selection to better estimate the test error of a predictive model. Therefore it gives us a more true accuracy than using the whole data as training and test data. As expected it gives us different values as the model used for 1.1. It would appear that themodel is over fitted, as the accuracy (0.7071) is down from the 0.7143 from the first model. This tells us that data performs better on training data has declined on 5 fold cv where there is data not seen on training, this is the case here and has been affected by over fitting, the true positive rate however has increased from 0.6154 in the first model to 0.6333, Which does not suit this trend. The false positive rate has increased, as specificity has decreased to 0.7625, telling us that this model is more accurate when predicting that it is not a letter the same as the model, but this has decreased compared to the first model. To add the precision and recall has decreased and as expected the f1 score has too. The recall has fell the most from 0.7272 to 0.667. a over 5% fall is shows further supports that this model has been overfitted. These results tell us has memorized the training data instead more than learning the relationships between features and the classes (images or not images). Although the falls in accuracies, tpr f1 score etc are not that big (mostly less than 5%) it does show that this model has been slightly overfitted.

Section 1.3

```
#1.3
library(MLeval)
res <- evalm(model)
```

```
## ***MLeval: Machine Learning Model Evaluation***
```

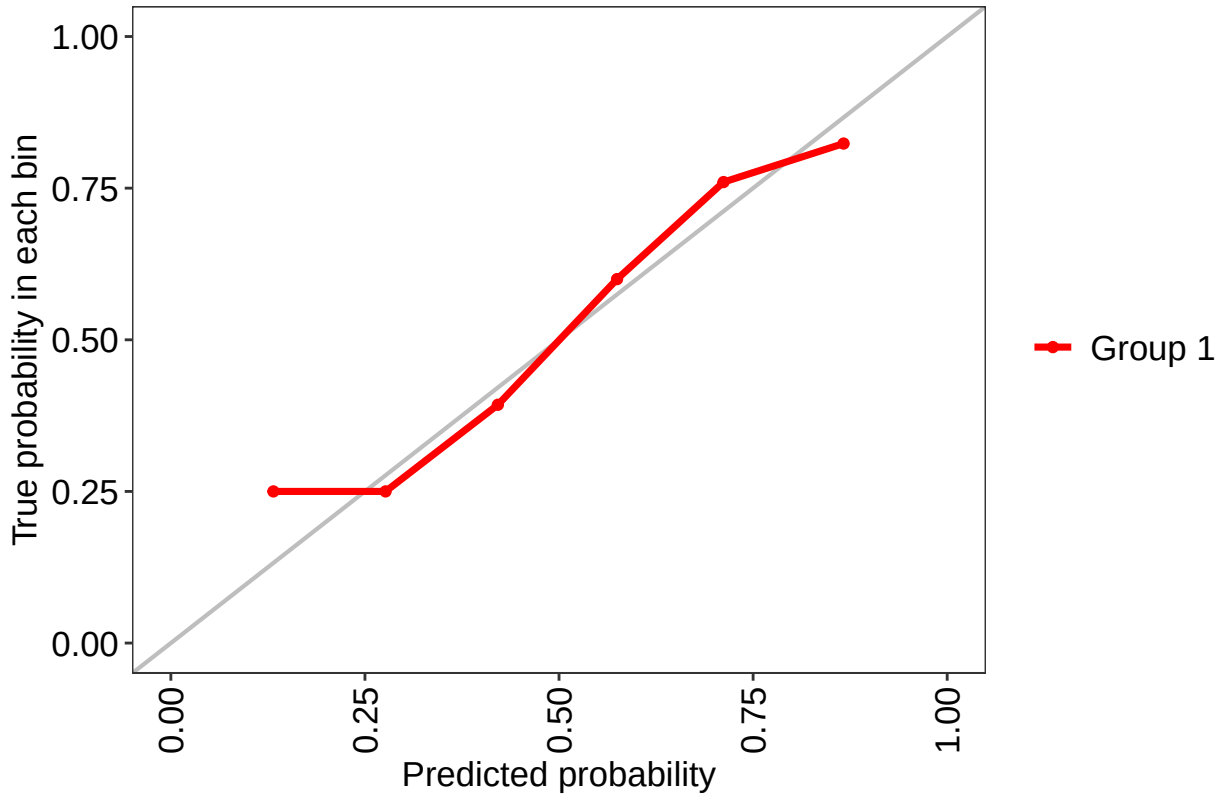
```
## Input: caret train function object
```

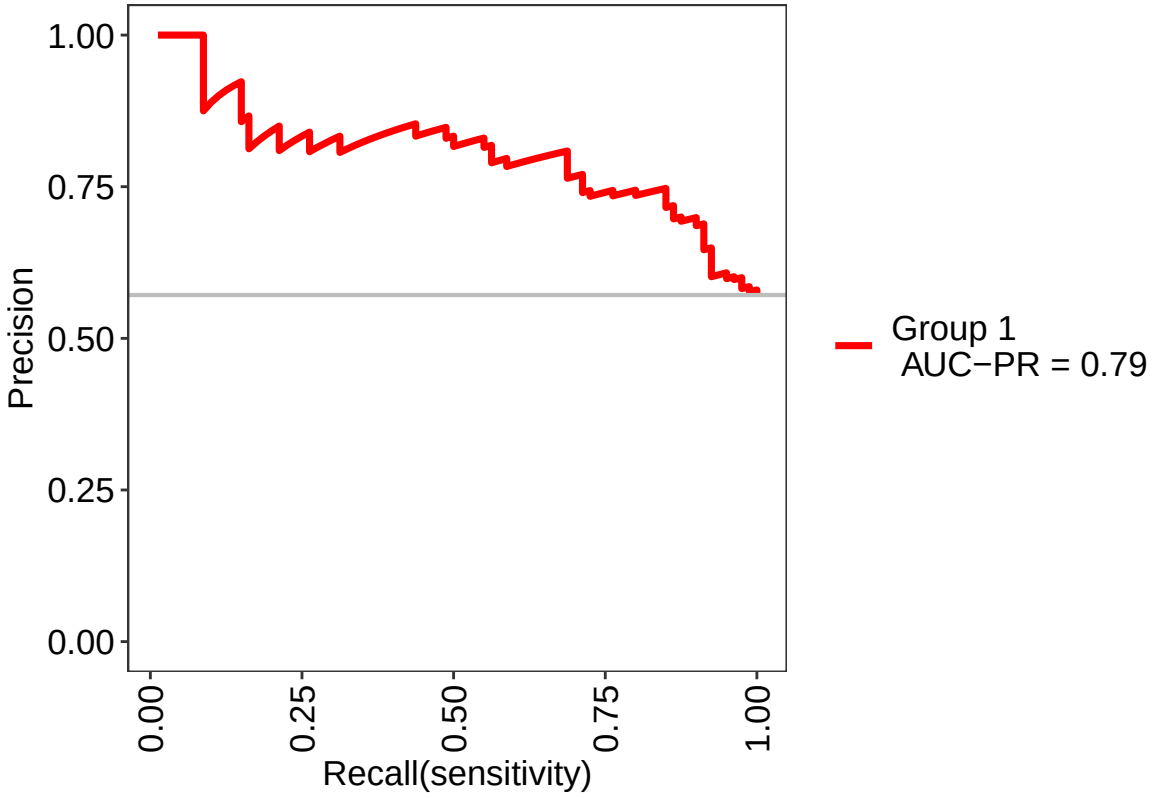
```
## Not averaging probs.
```

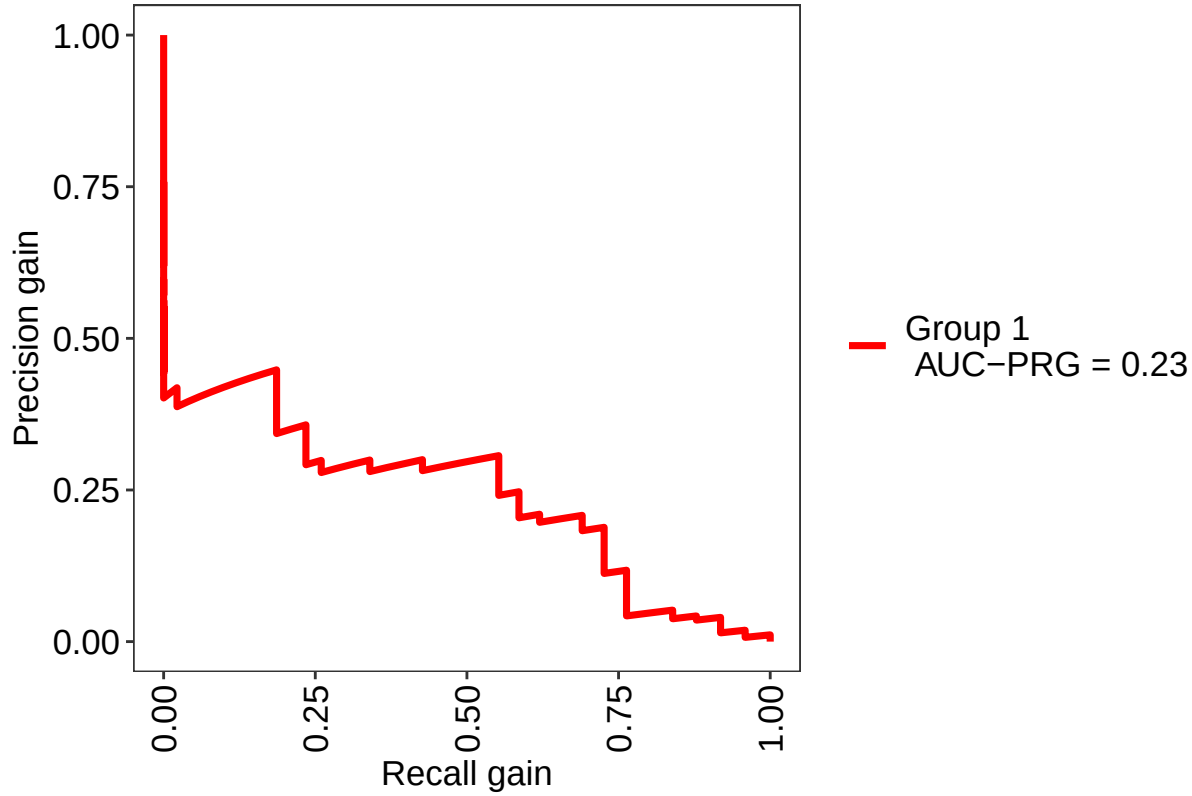
```
## Group 1 type: cv
```

```
## Observations: 140
```

```
## Number of groups: 1

## Observations per group: 140

## Positive: yes

## Negative: no

## Group: Group 1

## Positive: 80

## Negative: 60

## ***Performance Metrics***
```

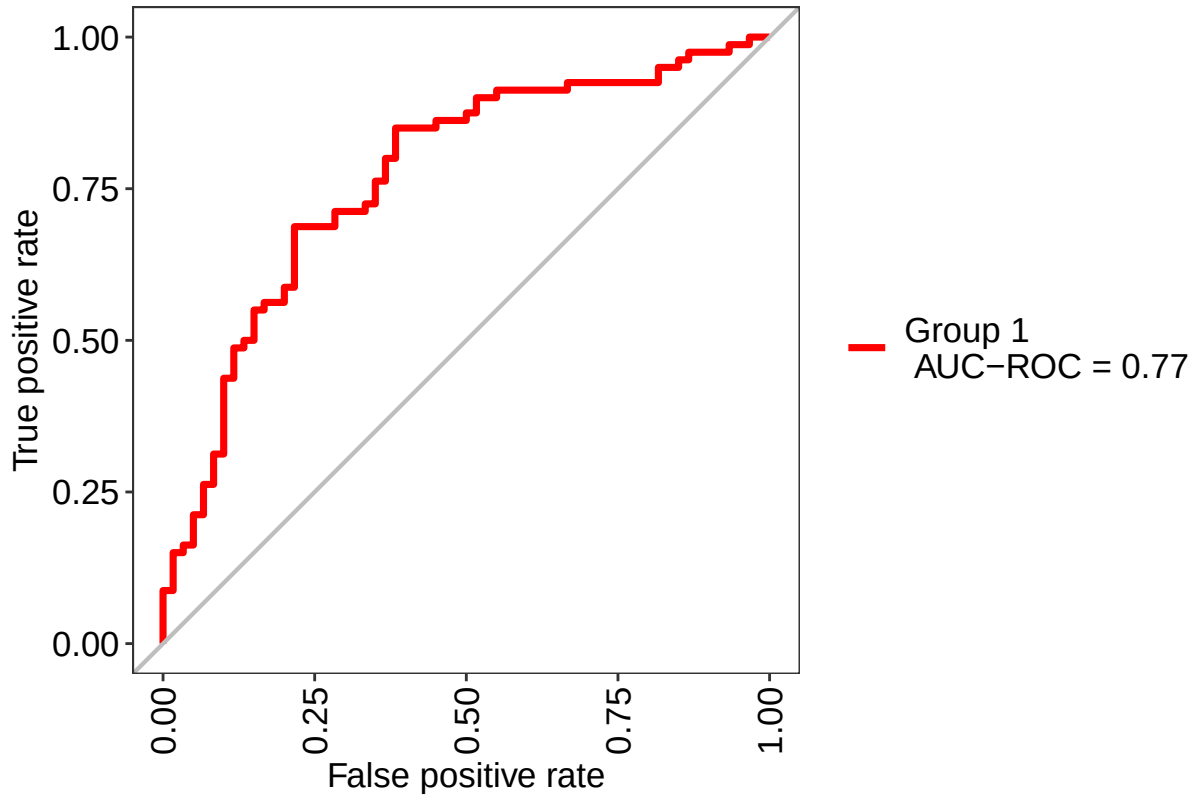



1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
Group 1
AUC−PR = 0.79

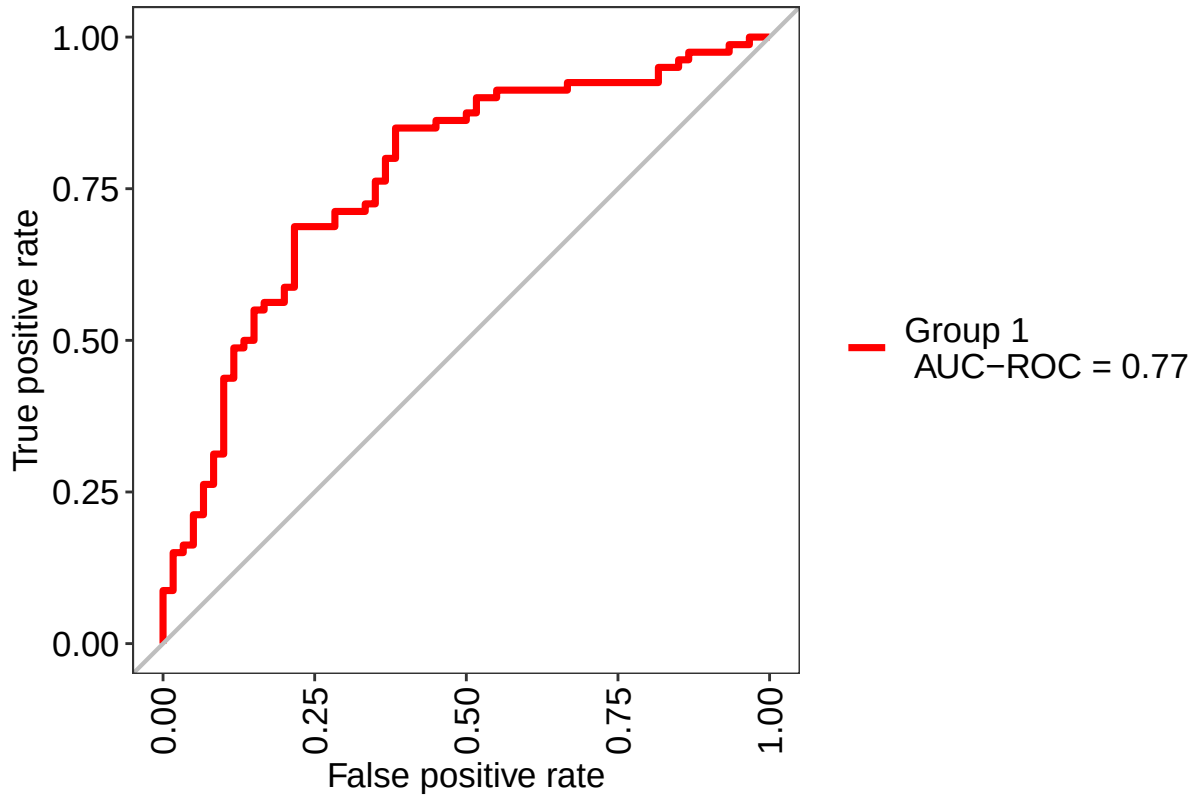



```
## Group 1 Optimal Informedness = 0.470833333333333
```

```
## Group 1 AUC-ROC = 0.77
```


1.00
0.75
0.50
0.25
0.00
True positive rate
0.00
0.25
0.50
0.75
1.00
False positive rate
Group 1
AUC−ROC = 0.77

```
res$roc
```



Above there is the ROC curce for the classifier. The roc curve show with the line in the middle what a useless test would be (50% tpr 50% fpr) means it's just by chance therefore not useful. Classifiers that give curves closer to the top-left corner indicate a better performance. This curve shows a somewhat reliable classifier as it is mostly on towards left hand side of the graph but for very small values and big values of tpr and fpr it is towards the line in the middle and even sometimes worse which tells us it is no better than random guessing here. Towards the middle of the graph and for middle values the classifier could be seen as useful and for these values is a lot better than random guessing , this tells us that our classifier could be useful to predict whether an image is a letter or not. This is backed up by the AUC-ROC, as the bigger the area undernath the roc curve the more reliable the classifier in distinguishing between two classifiers, between 0 and 1, here we got 0.77 which tells us that this model, for the most part is very useful as 77% is a high score, the AUC score is very important as it is a good metric to summarise the ROC score and the overall model, therefore in conclusion this is a good model. In 1.1 In the absence of cross-validation, it's possible that the model becomes biased by the data split

Section 2

Section 2.1

```
set.seed(42)
csvF<-read.csv(file ="40293751_features.csv" ,header = TRUE)
#kNeighbourMat<-Matrix(csvF[1:8,3])

kNeighbourMat=csvF[csvF$label %in% c('a','j','sad','smiley','xclaim'),]
kNeighbourMat$dummy.class[kNeighbourMat$label %in% c('a','j')]<-"letter"
```

```
kNeighbourMat$dummy.class[kNeighbourMat$label=="sad"]<-"sad"
kNeighbourMat$dummy.class[kNeighbourMat$label=="smiley"]<-"smiley"
kNeighbourMat$dummy.class[kNeighbourMat$label=="xclaim"]<-"xclaim"
train.X = cbind(kNeighbourMat$no_neigh_right,kNeighbourMat$aspect_ratio,kNeighbourMat$no_neigh_above,kN

ks = c(1,3,5,7,9,11,13)
accuracies = c()
for (kk in ks){

  if (kk / 2 !=0)

    {
    print(kk)
    knn1=knn(train.X,train.X,kNeighbourMat$dummy.class,k=kk)
  print(table(knn1,kNeighbourMat$dummy.class))
  accuracies = cbind(accuracies, mean(knn1==kNeighbourMat$dummy.class))
  }
}
```

```
## [1] 1
##
## knn1     letter sad smiley xclaim
##   letter     16   0      0      0
##   sad         0  20      0      0
##   smiley      0   0     20      0
##   xclaim      0   0      0     20
## [1] 3
##
## knn1     letter sad smiley xclaim
##   letter     16   0      0      1
##   sad         0  17      5      2
##   smiley      0   3     15      0
##   xclaim      0   0      0     17
## [1] 5
##
## knn1     letter sad smiley xclaim
##   letter     16   0      0      2
##   sad         0  15      4      2
##   smiley      0   5     16      0
##   xclaim      0   0      0     16
## [1] 7
##
## knn1     letter sad smiley xclaim
##   letter     16   0      0      3
##   sad         0  15      8      1
##   smiley      0   5     12      0
##   xclaim      0   0      0     16
## [1] 9
##
## knn1     letter sad smiley xclaim
##   letter     16   0      0      3
##   sad         0  15      7      1
##   smiley      0   5     13      1
```

```
##   xclaim      0   0      0     15
## [1] 11
##
## knn1     letter sad smiley xclaim
##   letter     14   0      0      3
##   sad         0  15      9      2
##   smiley      0   5     11      0
##   xclaim      2   0      0     15
## [1] 13
##
## knn1     letter sad smiley xclaim
##   letter     14   0      0      3
##   sad         0  14      9      2
##   smiley      0   5     10      0
##   xclaim      2   1      1     15
```

```
accuracies
```

```
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,]    1 0.8552632 0.8289474 0.7763158 0.7763158 0.7236842 0.6973684
```

For 2.1 I perform 4 way knn classification, for letter (only a or j), sad, smiley or xclaim using 4 features, which were no_neigh_right,aspect_ratio, no_neigh_above and connected_areas. I chose these features because in assignment 2 these were some of the best features to distinguish between a letter and non letter, and to add these features have similarities between each class e.g. aspect ratio the letters have low values for, while sad is mostly just above 1, xclaim very low values below 0.4 and smiley similar to sad but got more values below 1 and still a lot higher than xclaim and the letters. To distingish between each class, I have set up a dummy.class feature which says in english what class it belongs out of from the 4. I then performed a for loop, taking every number from 1 to 13 and doing if statement to check if it dividing this number by 2 doesn't give 0, if It doesn't it is a odd number and is needed for analysis. I then pass my data frame with the 4 features into knn function for both training and test and the dummy.class field and the value of k. The accuracies of each knn would be number of images (76) divided by total number that are correct classification. Above you can see the knn score for each odd value between 1 and 13, as you can see the bigger the k value the worse the accuracy gets, with 1 being a perfect accuracy, getting each classification right. As the k value increases, the training error also increases, this is increase bias because it has to consider more neighbours therefore, the model becomes more complex, as there is no testing data, when k=1 the training data becomes equal to testing data set, therefore 100% correct , as the model becomes more complex it doesn't do this therefore the accuracies decrease as the k value increases.

Section 2.2

```
kfoldsk=5
kNeighbourMat=kNeighbourMat

kNeighbourMat<-kNeighbourMat[sample(nrow(kNeighbourMat)),]
kNeighbourMat$folds<- cut(seq(1,nrow(kNeighbourMat)),breaks=kfoldsk,labels=FALSE)

kNeighbourMat<-kNeighbourMat[sample(nrow(kNeighbourMat)),]
kNeighbourMat$folds <- cut(seq(1,nrow(kNeighbourMat)),breaks=kfoldsk,labels=FALSE)
train_control <- trainControl(method="cv", number=kfoldsk)
```

```
tune_grid <- expand.grid(k = ((1:7)*2) -1)


# train the model

model <- train(dummy.class~no_neigh_right+aspect_ratio+no_neigh_above+connected_areas, data=kNeighbourMa
               trControl=train_control, tuneGrid=tune_grid, method="knn")
# summarize results
print(model)
```

```
## k-Nearest Neighbors
##
## 76 samples
##  4 predictor
##  4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 60, 61, 61, 61, 61
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.7091667  0.6122930
##    3  0.6975000  0.5964004
##    5  0.6175000  0.4916618
##    7  0.6308333  0.5071905
##    9  0.6441667  0.5257873
##   11  0.6308333  0.5079314
##   13  0.6958333  0.5947355
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

For 2.2 I performed k-nearest-neighbour classification for all odd values between 1 and 13 using 5 fold cross validation for the 4 classes used in 2.1. This would give a better indication on how this data can be fitted as 5 fold cross validation gives us a better indication on how the data is trained and used on unseen data, unlike in 2.1. As expected the accuaries are different from 2.1 and have all decreased from the knn which used the same data as training and test data. Again the best value of K is 1, with accuracy 0.7091667, there is a similar pattern as seen in 2.1, with the higher the k value the lower the accuracy, this is because as the k value becomes bigger the model becomes too generalized and fails to accurately predict data points in both train and test sets, this is known as underfitting (except from k=13), the lower values have also decreased massively on accuracy because of overfitting, e.g. k=1 had accuracy of 1 for 2.1 but it is 0.7091667, this is because the model is too specific and fails to be to generalize,The model accomplishes a high accuracy on train set but will be a poor predictor on new, previously unseen data points, based off our accuracies this is shown to be true. (rest of analysis at 2.4 with graphs).

Section 2.3

```
tune_grid<- expand.grid(k = 1)#best value of k
newModel<-train(dummy.class~no_neigh_right+aspect_ratio+no_neigh_above+connected_areas, data=kNeighbourM
                trControl=train_control, tuneGrid=tune_grid, method="knn")
```

```
knnPredict<-predict(newModel,newdata=kNeighbourMat)
ActualData <- as.factor(kNeighbourMat[["dummy.class"]])
cm=confusionMatrix(knnPredict,ActualData)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction letter sad smiley xclaim
##     letter     16   0      0      0
##     sad         0  20      0      0
##     smiley      0   0     20      0
##     xclaim      0   0      0     20
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9526, 1)
##     No Information Rate : 0.2632
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: letter Class: sad Class: smiley Class: xclaim
## Sensitivity                 1.0000     1.0000        1.0000        1.0000
## Specificity                 1.0000     1.0000        1.0000        1.0000
## Pos Pred Value              1.0000     1.0000        1.0000        1.0000
## Neg Pred Value              1.0000     1.0000        1.0000        1.0000
## Prevalence                  0.2105     0.2632        0.2632        0.2632
## Detection Rate              0.2105     0.2632        0.2632        0.2632
## Detection Prevalence        0.2105     0.2632        0.2632        0.2632
## Balanced Accuracy           1.0000     1.0000        1.0000        1.0000
```

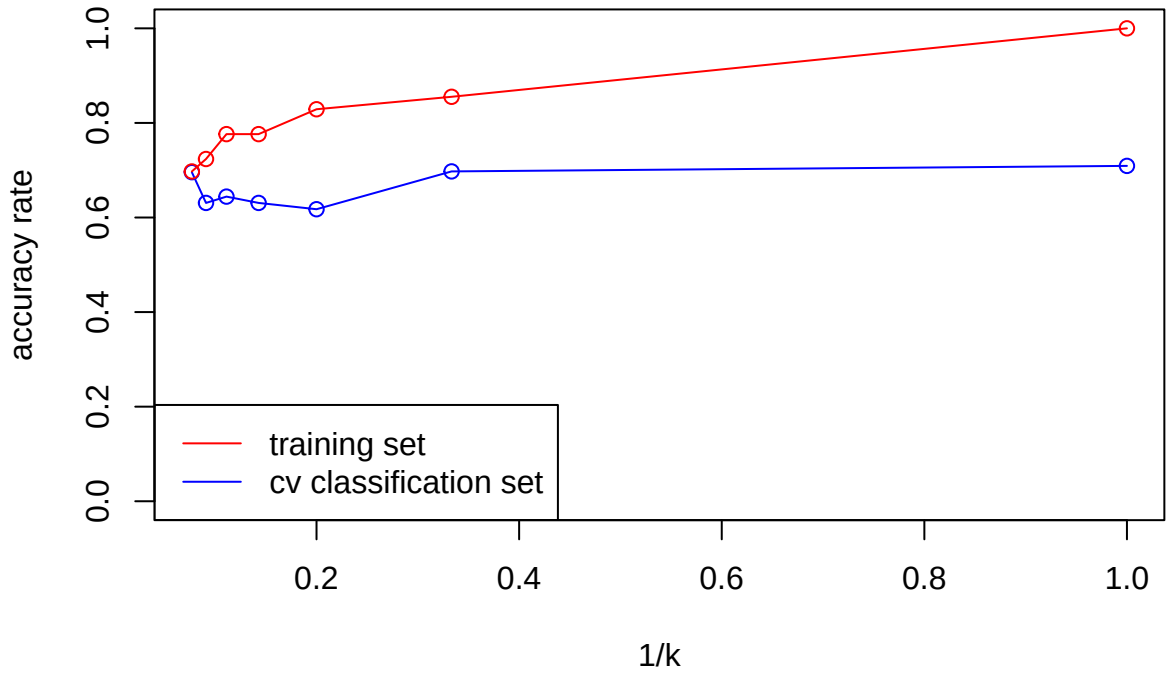
```
tune_grid<- expand.grid(k = 3)# next best value of k
newModel<-train(dummy.class~no_neigh_right+aspect_ratio+no_neigh_above+connected_areas, data=kNeighbourl
                trControl=train_control, tuneGrid=tune_grid, method="knn")
knnPredict<-predict(newModel,newdata=kNeighbourMat)
ActualData <- as.factor(kNeighbourMat[["dummy.class"]])
cm=confusionMatrix(knnPredict,ActualData)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction letter sad smiley xclaim
##     letter     16   0      0      1
##     sad         0  17      5      2
##     smiley      0   3     15      0
##     xclaim      0   0      0     17
```

```
##
## Overall Statistics
##
##                Accuracy : 0.8553
##                  95% CI : (0.7558, 0.9255)
##     No Information Rate : 0.2632
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8067
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: letter Class: sad Class: smiley Class: xclaim
## Sensitivity                 1.0000     0.8500        0.7500        0.8500
## Specificity                 0.9833     0.8750        0.9464        1.0000
## Pos Pred Value              0.9412     0.7083        0.8333        1.0000
## Neg Pred Value              1.0000     0.9423        0.9138        0.9492
## Prevalence                  0.2105     0.2632        0.2632        0.2632
## Detection Rate              0.2105     0.2237        0.1974        0.2237
## Detection Prevalence        0.2237     0.3158        0.2368        0.2237
## Balanced Accuracy           0.9917     0.8625        0.8482        0.9250
```

For 2.3 I used the best value of k from 2.2 (which was 1.1), trained the model again and then used the confusionMatrix function to calculate its confusion matrix. As it predicted all the images to it's correct classes I will instead use the next best k value which was 3 with 0.6975000. The best class in terms of predictions here was letters with 16 out of 16 predicted correctly. The worst with k=3 was smiley with only 75% of the actual smiley images predicted correctly, the other 5 were predicted as sad faces, which isn't too shocking as there is similarities between both of them. this makes smiley and sad faces the most difficult to discriminate between as of course a smiley and sad face have similar features as only difference is the mouth, meaning that the feautures used for this model are more than likley very similar for both of these images, the predictions for the sad faces also back this up, although 17 out of 20 were predicted correctly, the 3 that weren't were smiley faces, therfore backing up our point that these 2 classes are the hardest to discriminate.

```
#2.4
x1<-model$results$k
y1<-model$results$Accuracy
y2<-accuracies

ks<-c(1,1/3,1/5,1/7,1/9,1/11,1/13)
plot(ks,y1,type="o",ylim=c(0,1),col="blue",ylab="accuracy rate",xlab="1/k")
lines(ks,type="o",y2,col="red")
legend(x = "bottomleft",
       legend = c("training set", "cv classification set"),lty = c(1, 1),col=c("red","blue"))
```



The graph above shows the comparison between accuracy rate over training data (from 2.1) and cv classication (from 2.2). It shows that no accuries are bigger for cross validation, but there values better reflect the dataset because there is less bias and is used on unseen data unlike training set.

This graph shows what I described in 2.2, that the high k values (ones on the left side of graph are underfitted), as being top left in the graph shows underfitting and lower values overfitted as k values with top right plots in graph show overfitting.

Section 3

Section 3.1

```
set.seed(42)
allFeats<-read.delim(file ="all_features.csv" ,header = F)

allFeats <- allFeats[sample(nrow(allFeats)),]
allFeats$folds <- cut(seq(1,nrow(allFeats)),breaks=kfoldsk,labels=FALSE)
allFeats$folds
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

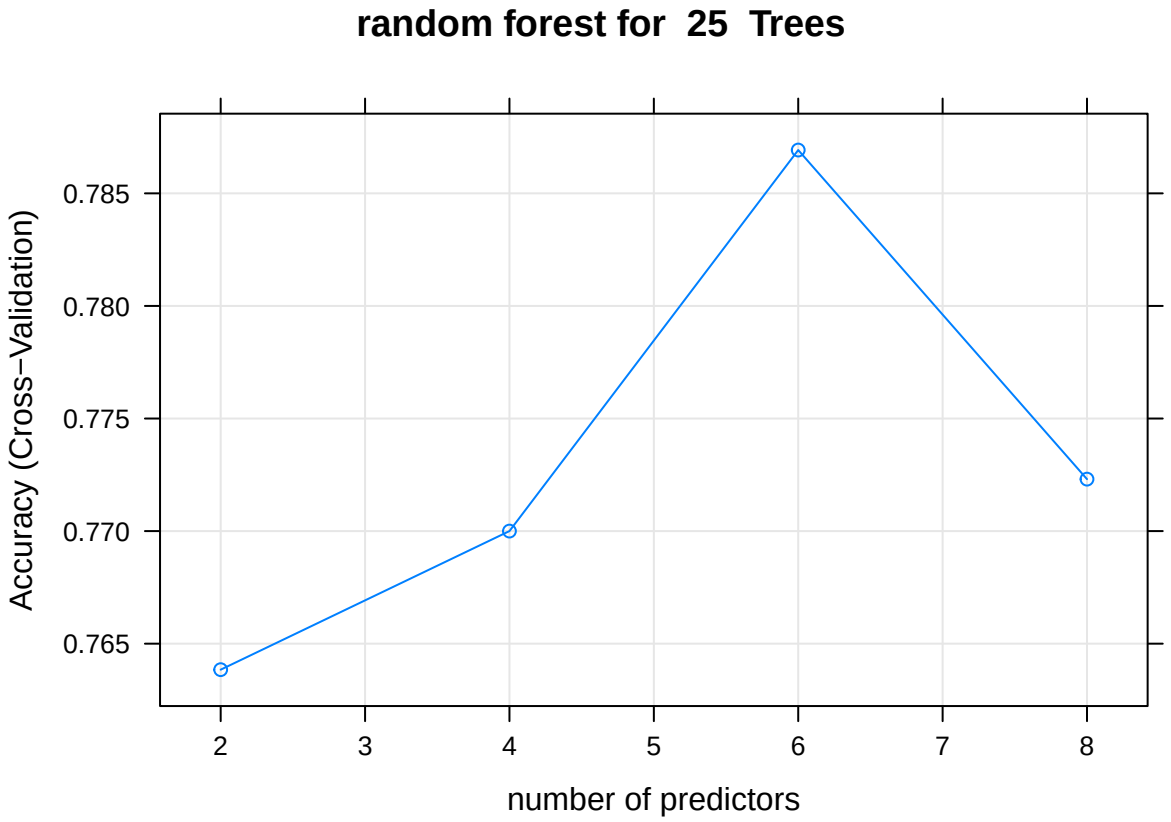
```
##  [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [260] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [445] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [482] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [519] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [556] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [593] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [630] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [667] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [704] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [741] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [778] 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [815] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [852] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [889] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [926] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [963] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1000] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1037] 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1074] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1111] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1148] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1185] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1222] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1259] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [1296] 5 5 5 5 5
```

```
train_control<-trainControl(method = "cv",number = 5,search = 'grid')
tune_grid <-expand.grid( .mtry=c(2,4,6,8))

x=25
results<-matrix(,ncol = 4,byrow = T)



model<-train(V1 ~V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+V16+V17+V18,data=allFeats,method="rf",trC
print(model)
```

```
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa
##   2     0.7638462  0.7441667
##   4     0.7700000  0.7508333
##   6     0.7869231  0.7691667
##   8     0.7723077  0.7533333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

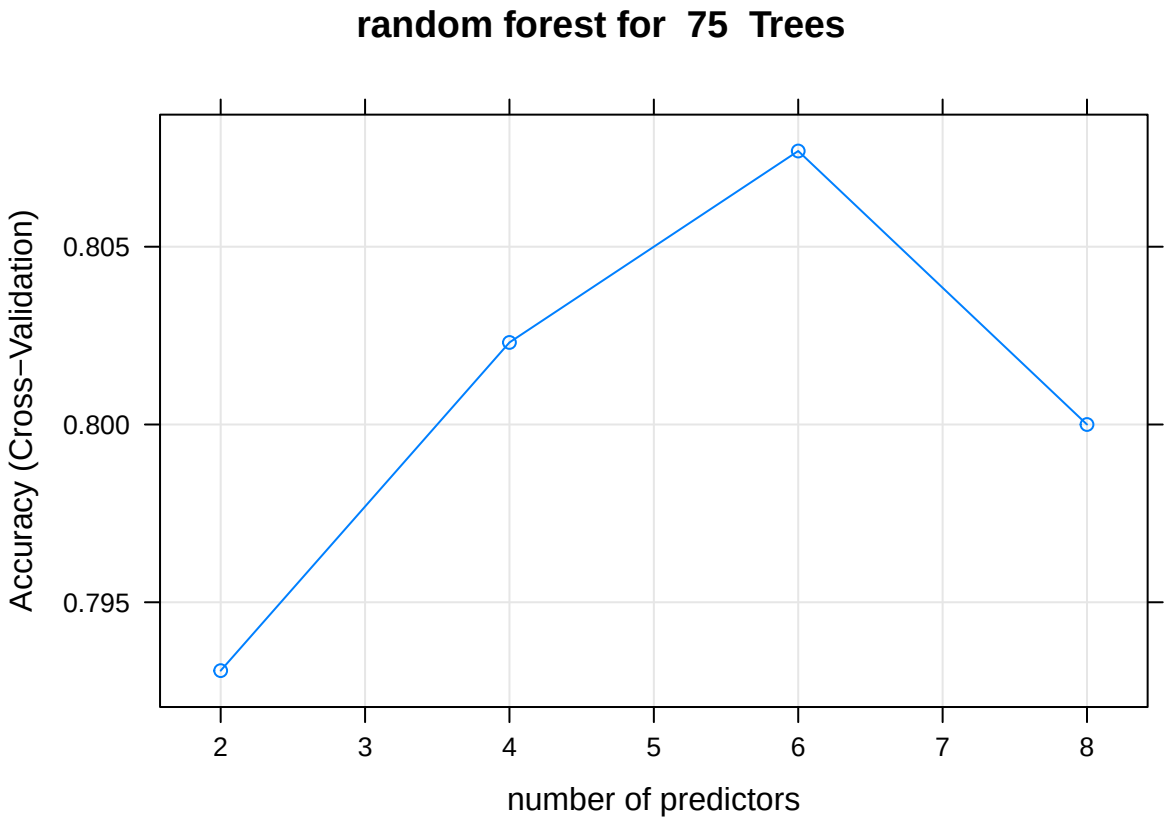
```
results<-rbind(results,model$results$Accuracy)

plot((model),main=paste("random forest for ",x," Trees"),xlab="number of predictors")
```




```
for (i in 1:7)
{
  x<-x+50
  print(x)
  model<-train(V1 ~V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+V16+V17+V18,data=allFeats,method="rf",t
  print(model)
  print(plot((model),main=paste("random forest for ",x," Trees"),xlab="number of predictors"))
  results<-rbind(results,model$results$Accuracy)

}
```

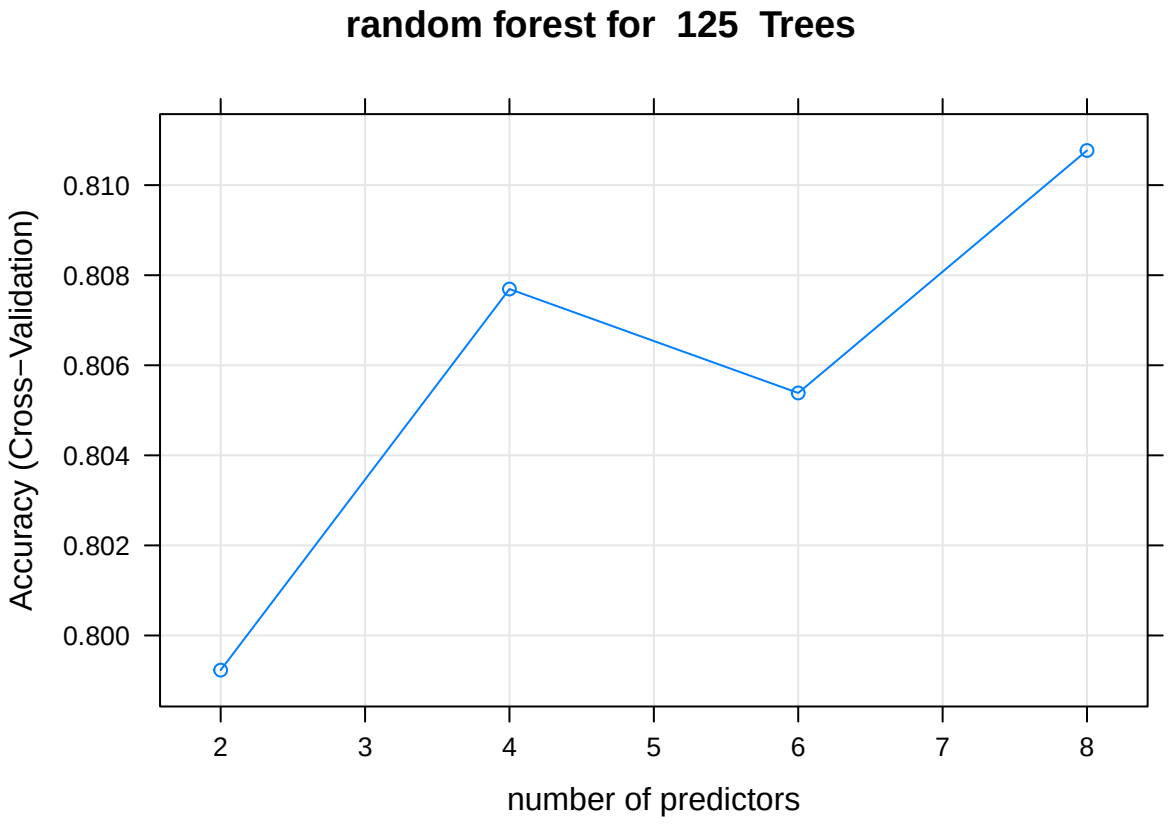
```
## [1] 75
```

```
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7930769  0.7758333
##   4     0.8023077  0.7858333
##   6     0.8076923  0.7916667
##   8     0.8000000  0.7833333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```



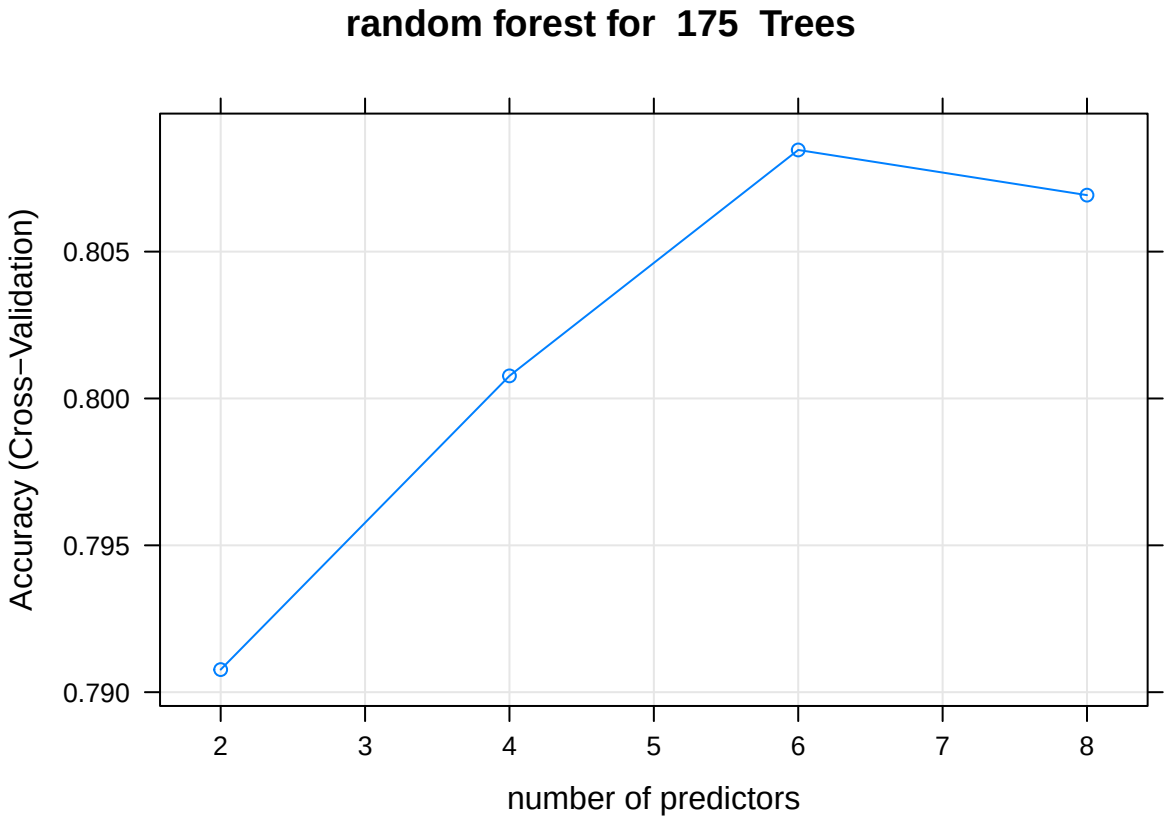

```
## [1] 125
## Random Forest
##
## 1300 samples
##   16 predictor
```

```
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7992308  0.7825000
##   4     0.8076923  0.7916667
##   6     0.8053846  0.7891667
##   8     0.8107692  0.7950000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```



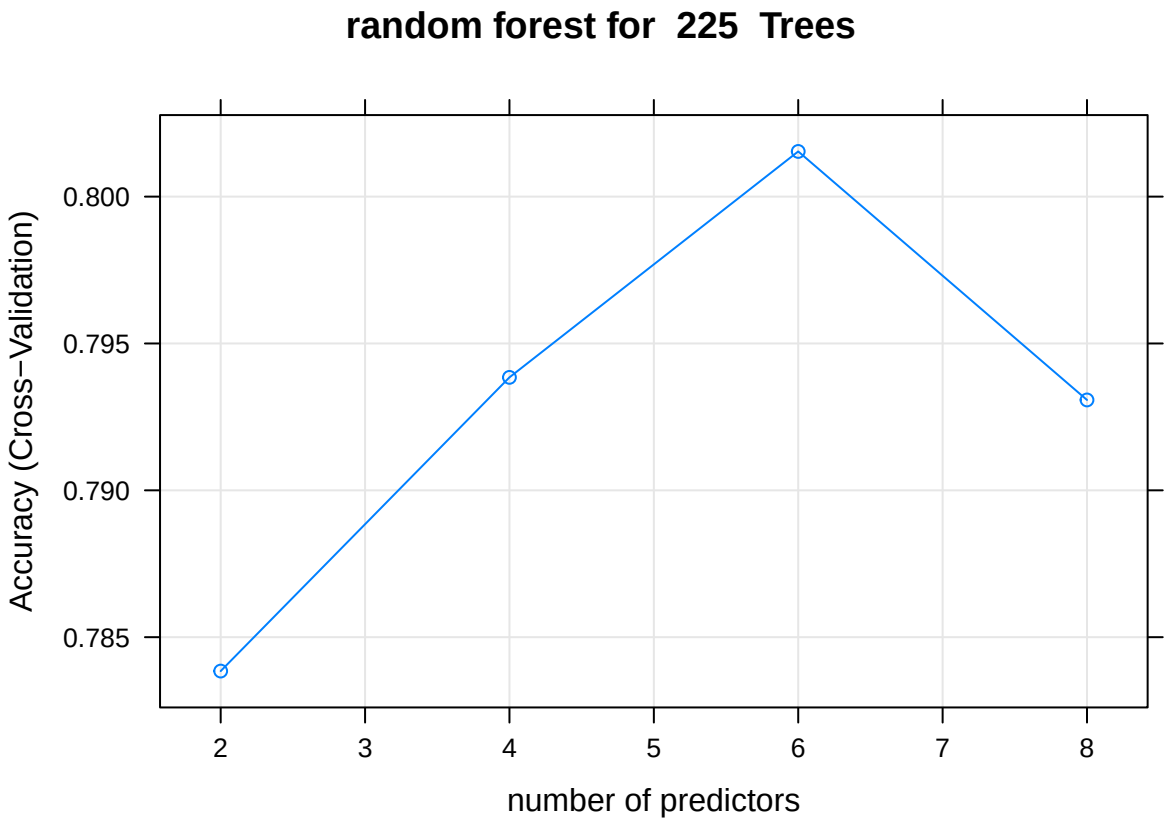

```
## [1] 175
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
```

```
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7907692  0.7733333
##   4     0.8007692  0.7841667
##   6     0.8084615  0.7925000
##   8     0.8069231  0.7908333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```



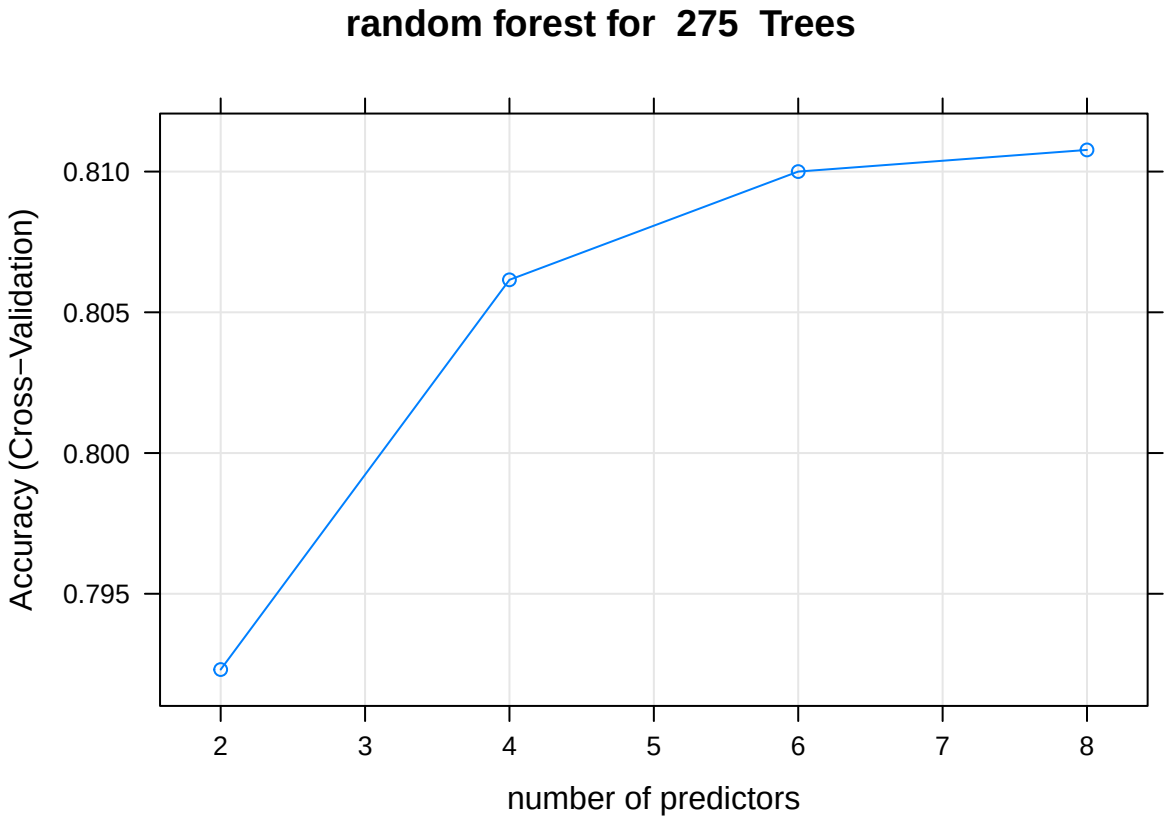

```
## [1] 225
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
```

```
##   2     0.7838462  0.7658333
##   4     0.7938462  0.7766667
##   6     0.8015385  0.7850000
##   8     0.7930769  0.7758333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

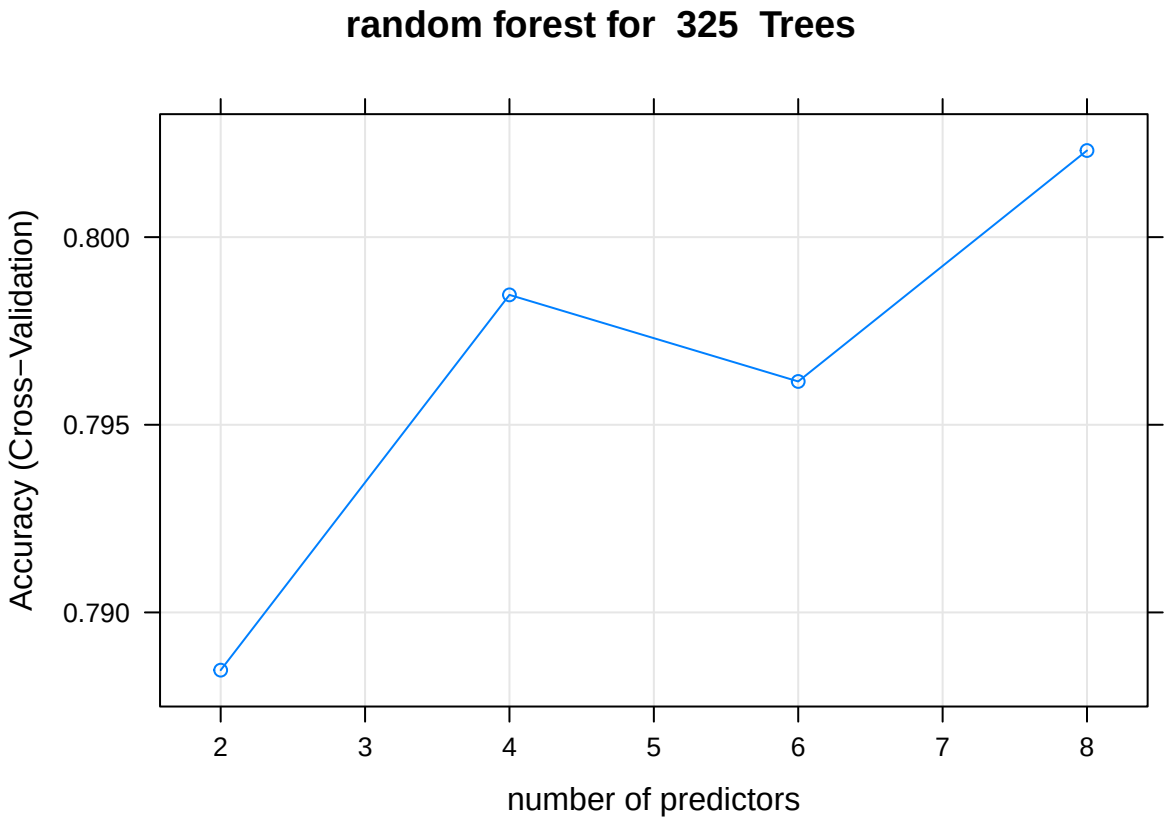



```
## [1] 275
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7923077  0.7750000
##   4     0.8061538  0.7900000
##   6     0.8100000  0.7941667
##   8     0.8107692  0.7950000
```

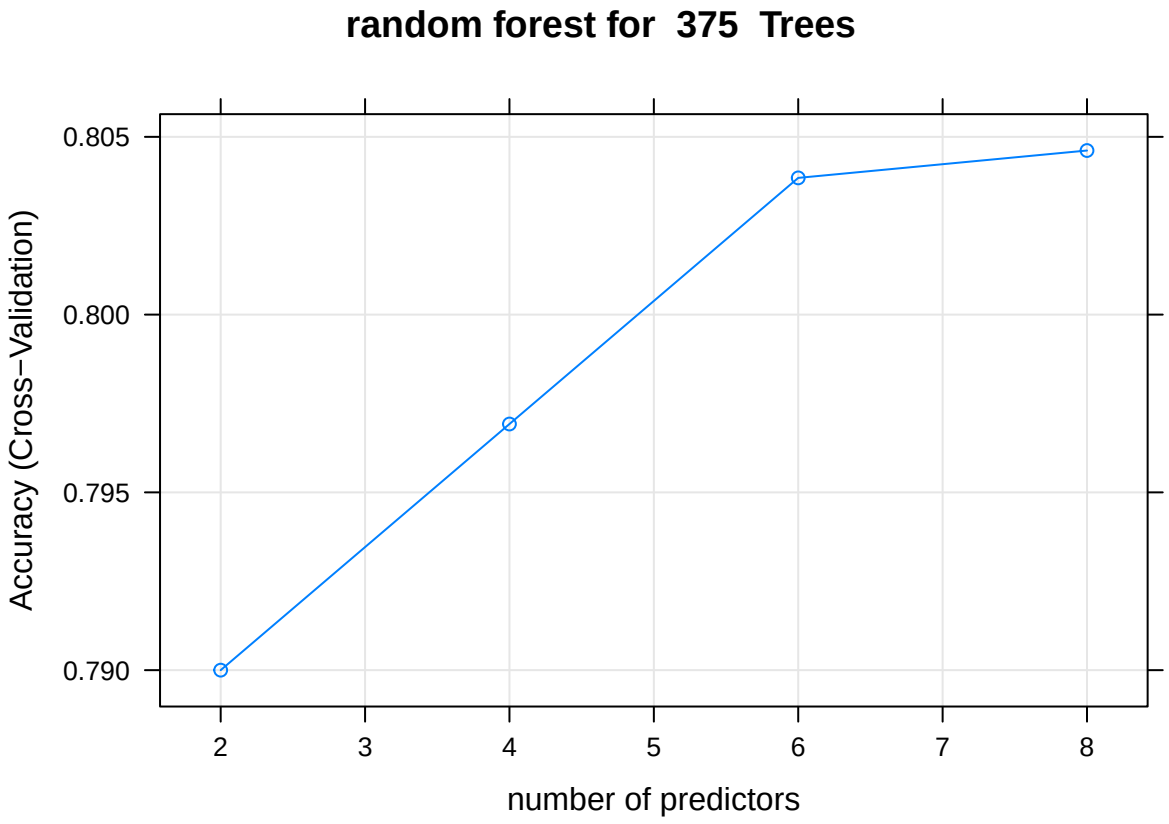
```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```




```
## [1] 325
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7884615  0.7708333
##   4     0.7984615  0.7816667
##   6     0.7961538  0.7791667
##   8     0.8023077  0.7858333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```




```
## [1] 375
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7900000  0.7725000
##   4     0.7969231  0.7800000
##   6     0.8038462  0.7875000
##   8     0.8046154  0.7883333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```




```
colnames(results)<-c(2,4,6,8)
rownames(results)<-c('',25,75,125,175,225,275,325,375)
print(results)
```

```
##             2         4         6         8
##            NA        NA        NA        NA
## 25  0.7638462 0.7700000 0.7869231 0.7723077
## 75  0.7930769 0.8023077 0.8076923 0.8000000
## 125 0.7992308 0.8076923 0.8053846 0.8107692
## 175 0.7907692 0.8007692 0.8084615 0.8069231
## 225 0.7838462 0.7938462 0.8015385 0.7930769
## 275 0.7923077 0.8061538 0.8100000 0.8107692
## 325 0.7884615 0.7984615 0.7961538 0.8023077
## 375 0.7900000 0.7969231 0.8038462 0.8046154
```

For 3.1 I performed random forest classification with 5 fold cv. I first split the allFeautres dataframe into 5 folds, to perform 5 fold cv. I use expand.grid mtry to get number of predictors {2,4,6,8} and then outside loop train the model with the expand grid for 25, then use a loop to do this for 75 to 375 with 50 iterations for each, I then saved the accuracies of each random forest model onto a dataframe as you can see above. To add I have created a scatter graph for each value of k, this is a good visualization for each value of k as it clear shows which number of predictors are best for each k. The best value of k was 125 when it had 8 predictors with 0.8107692 (8 for =275 is also 0.8107692 but on average k=125 has better accuracies on average). This tells us that clearly 125 is the best value of k to predict what symbol the image is or not. On average between all values of k 25 has the worst accuracies, it also has the lowest accuracies when number of predictors = 2 with 0.7638462. A trend for all values of K is that when the number of predictors is 2, it gives

the lowest accuracy, this is because it considers only 2 features meaning it would be harder to differ between letters and non letters with less features considered. And as the number of predictors increases the values tend to increase, this is because the it is easier to tell the difference between images if there is more features considered, however for some nTrees it doesn't, this may be that too many features are considered here and the model becomes overfitted, e.g. 175, number of predictors=6 accuracy is 0.8084615 and when it is 8 accuracy is 0.8069231. To add the best nTrees seem to be in the middle, with them getting, with smaller and larger nTrees having slightly less accruacies, this is probably because too little trees are considered at start, e.g. 25, With a lower number of trees the model is more specific and fails to generalize,therfore this causes overfitting ,and too many at end e.g 325 and 375, this is because when the nTrees value increases the model becomes too generalized and fails to accurately data points in both the training and test sets.This is under fitting, the values in the middle have the best balance of these, therfore giving them the best accuracies. To conclude nTree=125 and No of predictor=8 is the best value for accuracy and will be used for analysis in 3.2

Section 3.2

```
set.seed(42)
accs<-c()
for (i in 1:15)
{
  tune_grid <-expand.grid( .mtry=8)
  model<-train(V1 ~V3+V4+V5+V6+V7+V8+V9+V10+V11+V12+V13+V14+V15+V16+V17+V18,data=allFeats,method="rf",t
  print(model)
accs<-append(accs,model$results$Accuracy)
  #accs<-c(accs+model$results$Accuracy)
}
```

```
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8046154  0.7883333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
```

```
##   Accuracy   Kappa
##   0.7961538  0.7791667
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7838462  0.7658333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7984615  0.7816667
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8007692  0.7841667
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8015385  0.785
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8023077  0.7858333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7930769  0.7758333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7953846  0.7783333
##
```

```
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7992308  0.7825
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8015385  0.785
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8015385  0.785
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
```

```
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7930769  0.7758333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.8115385  0.7958333
##
## Tuning parameter 'mtry' was held constant at a value of 8
## Random Forest
##
## 1300 samples
##   16 predictor
##   13 classes: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1040, 1040, 1040, 1040, 1040
## Resampling results:
##
##   Accuracy   Kappa
##   0.7846154  0.7666667
##
## Tuning parameter 'mtry' was held constant at a value of 8
```

```
print(accs)
```

```
##  [1] 0.8046154 0.7961538 0.7838462 0.7984615 0.8007692 0.8015385 0.8023077
##  [8] 0.7930769 0.7953846 0.7992308 0.8015385 0.8015385 0.7930769 0.8115385
## [15] 0.7846154
```

```
accMean<-mean(accs)
accSd<-sd(accs)
accMean
```

```
## [1] 0.7978462
```

```
accSd
```

```
## [1] 0.007223049
```

```r
t.test(accs,mu = 1/13, alternative = "two.sided")
```

```
##
##  One Sample t-test
##
## data:  accs
## t = 386.56, df = 14, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0.07692308
## 95 percent confidence interval:
##  0.7938462 0.8018461
## sample estimates:
## mean of x
## 0.7978462
```

In this section, I reran the best combination of trees and predictors (nTree=125 and No Of predictors=8) 15 times to check if it performs significantly better than chance. I did this by having a for loop for 15 times performing random forest for nTree=125 and No Of predictors=8 and saving the accuracy into a data frame. I then used mean function to get the mean of the 15 accuracies and sd function to get the standard deviation. The mean was 0.7978462 and the standard deviation was 0.007223049. This standard deviation score tells us that the data is clustered around the mean, which tells us that there is not distance between each model fitting, so even though there is an element of randmominess in each model fitting the difference is so slight and therefore this supports that this model performs better than chance. The mean is 0.7978462, which is down from 0.8107692 that this model got for accuracy in the results for 3.1, This tells us that this model performed slightly better than on average due the the randominess of cross validation and random forest. The t-test results will tell us if it performs better than chance or not. In this t-test we check if the accuracies are equal to the probabilty of chance which is 1/13(13 groups), if the p score is $> 5\%$ the null hypothesis (they are equal stands), if it is less, the alternative hypothesis is true (they are not equal), here the p value is 2.2e-16, which is a very small value and way less than 5%, therefore the alternative hypothesis is true and they differ a lot, as the p value is very small, to add the t value supports this as, the bigger the t value the bigger difference between the groups, the t value is positive and is 386.56, which is a big score which tells us that this model performs way better than random chance . Therefore it is clear based off mean, standard deviation and t test (p and t score) that this model performs significantly better than chance.