Assignment 3: Machine Learning

Pranay Meshram (40280938)

Introduction

This assignment will focus on the machine learning models to classify between different types of images such as letters, non-letters, finding out the accuracy of models. This assignment will also put focus on cross-validating the data and recording the accuracy.

Section 1

```
setwd("C:/Users/Asus/Desktop/AI Assignment")

source("./section1_code.r")
```

Now we will first initially install different libraries as per needed.

We install caTools library for use of splitting samples.

```
library(caTools)
```

Then we install InformationValue library for use in ConfusionMatrix

```
library(InformationValue)
```

We install modelr and MLeval library for running our machine learning model.

```
library(modelr)

library(MLeval)
```

And finally, the ROCR library to plot the ROC curve

```
library(ROCR)
```

Once we're set installing the library, we set the seed level to 42 and read all the features from the '40280938_features.csv' file.

```
features <- read.csv(file = './40280938_features.csv', header = TRUE)

nrow(features)
```

```
## [1] 140
```

As we can see, we have 140 rows of data in our features files.

Section 1.1

Now, we have our data of letters from ‘a’ to ‘j’ and then we have the other non-letters. We can see that from rows 81 till 140, all the images are non-letters.

```
features[80,]
```

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_ratio
	<chr>	<int>	<int>	<int>	<int>	<int>	<int>	<dbl>
80	j	8	20	8	0	2	2	0.2142857

1 row | 1-10 of 19 columns

```
features[81,]
```

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_ratio
	<chr>	<int>	<int>	<int>	<int>	<int>	<int>	<dbl>
81	sad	1	20	2	4	1	3	0.7

1 row | 1-10 of 19 columns

We will use this data to classify as letters and non-letters. Rows from 1 to 80 are letters while rows from 80 to 140 are non-leters.

We will create a separate column **isLetter** with two values, 1 and 0. 1 means the image is a letter while 0 means the image is a non-letter.

```
features$isLetter <- sample(100, size = nrow(features), replace = TRUE)

features$isLetter[1:80] = 1
features$isLetter[81:140] = 0
```

We then split the data into a training set and a test set. We put 80% of the data (0.8) in training set and the rest in test set.

```
i_split<- sample.split(row.names(features), 0.8)
train<-features[i_split, ]
test<-features[!i_split, ]

nrow(train)
```

```
## [1] 112
```

```
nrow(test)
```

```
## [1] 28
```

Once we’re done splitting the data, we fit a logistic regression model using the training data. We try to predict if an image is a letter or not based on **nr_pix** and **aspect_ratio** features of the data.

```
logistic <- glm(isLetter ~ nr_pix + aspect_ratio, data = train, family="binomial")

logistic
```

```
##
## Call:  glm(formula = isLetter ~ nr_pix + aspect_ratio, family = "binomial",
##     data = train)
##
## Coefficients:
##  (Intercept)        nr_pix  aspect_ratio
##     -1.11646       0.07976      -1.31375
##
## Degrees of Freedom: 111 Total (i.e. Null);  109 Residual
## Null Deviance:       154.7
## Residual Deviance: 147   AIC: 153
```

Here we an see that there is a very modest change in deviance.

We now predict the probability:

```
probability = predict(logistic, test, type="response")
probability
```

```
##         3         9        18        22        23        24        25        27
## 0.3075766 0.5421868 0.4805194 0.5907182 0.4141176 0.5105085 0.6614199 0.6336151
##        31        32        41        42        43        44        52        57
## 0.8128797 0.8092850 0.5477356 0.6215246 0.6196831 0.5711266 0.6897938 0.5745708
##        59        65        72        74        82        87        88        96
## 0.6111411 0.4016524 0.3826422 0.4653157 0.5104017 0.5649869 0.4766168 0.5576699
##       103       115       134       140
## 0.3186762 0.4673925 0.7034042 0.4209632
```

As we can see, most of the probabilities are very close to 50%. Some are as high as 70% which means our model is pretty good at predicting the possibility if an image belongs to the letter category.

We now report the confusion matrix for threshold of 0.5:

```
confusion_matrix <- confusionMatrix(test$isLetter, probability, threshold = 0.5)
confusion_matrix
```

	0 <int>	**1** <int>
0	4	6
1	4	14

2 rows

Based on this Confusion Matrix, we can make certain notes:

Here,

True Negatives (TN) : 3 False Positives (FP) : 5 True Positives (TP) : 10 False Negatives (FN) : 10 Actual No: 8 Actual Yes: 20 Predicted No: 13 Predicted Yes: 15 Total number of items: 28

Accuracy - We know that accuracy is (TP + TN) / Actual No

```
accuracy <- (10+3)/28
accuracy
```

```
## [1] 0.4642857
```

The accuracy of this model isn't good as it's below 50% but closer to it.

True Positive Rate - We can calculate the true positive rate by using 'sensitivity' function

```
true_positive_rate <- sensitivity(test$isLetter, probability)
true_positive_rate
```

```
## [1] 0.7
```

Our model is good predicting if a letter is actually a letter when it's a letter with positive rate above 50.

False Positive Rate - False positive rate is FP / Actual No

```
false_positive_rate <- (5/8)
false_positive_rate
```

```
## [1] 0.625
```

Our model is good predicting if a non-letter is actually a non-letter when it's a non-letter with positive rate above 50.

Precision - Precision is TP/Predicted Yes

```
precision <- (10/15)
precision
```

```
## [1] 0.6666667
```

Our model is pretty good at being correct in predicting with a decent precision rate of 66%.

Recall - Recall is TP/ (TP + FN)

```
recall <- 10/ (10+10)
recall
```

```
## [1] 0.5
```

Our model has average recall of 50%.

F1-Score - F1 score is (2 * Precision * Recall) / (Precision + Recall)

```
f1_score <- (2*precision*recall)/(precision+recall)
f1_score
```

```
## [1] 0.5714286
```

A poor F1 score is of 0.0 and our F1 score is 0.5 which is on average, providing good accuracy of the our model's performance.

Section 1.2

Here we load the caret library first for 'trainControl' method to work:

```
library(caret)
```

Now we are going to perform k-fold cross validation on the previous model. We divide dataset into k groups and then repeat the process k times with each set.

We first perform the cross validation here:

```
ctrl <- trainControl(method = "cv", number = 5, savePredictions = T)
```

Then we train the model same as how we did in 1.1. However, this time we train all the 140 items instead of just the training data. There's no seperate test data.

```
model <- train(isLetter ~ nr_pix + aspect_ratio, data = features, method = "lm", trControl =
ctrl)

model
```

```
## Linear Regression
##
## 140 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 112, 112, 112, 112, 112
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   0.484755  0.05755864  0.464172
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Here we can make a few note from the observations.

RMSE is pretty low, which is good for the model as lower the RMSE, the more closely can our model predict actual observations.

RSquared is pretty low, which leads to inaccurate predictions as higher the R-Squared, more closely a model predicts actual observations.

MAE is pretty low which is good for prediction, as lower the MAE, more closely the model predicts actual observations.

Now we predict the probabilities:

```
probability_cross_validated_data <- predict(model, features)
```

We then make a confusion matrix. Here, we'll have to unload the caret library first for 'confusionMatrix' function to work as some methods of both the libraries are same in name and causees errors.

```
detach("package:caret", unload = TRUE)

confusionMatrix(features$isLetter, probability_cross_validated_data, threshold = 0.5)
```

	0 <int>	1 <int>
0	20	18
1	40	62

2 rows

True Negatives (TN) : 20 False Positives (FP) : 18 True Positives (TP) : 62 False Negatives (FN) : 40 Actual No: 38 Actual Yes: 102 Predicted No: 60 Predicted Yes: 80 Total number of items: 140

Accuracy

```
accuracy_cross_validated_data <- (62+20)/140
accuracy_cross_validated_data
```

```
## [1] 0.5857143
```

Our data is pretty accurate in this cross-validated model, as expected because of cross validation.

True Positive Rate

```
true_positive_rate_cross_validated_data <- sensitivity(features$isLetter, probability_cross_v
alidated_data)

true_positive_rate_cross_validated_data
```

```
## [1] 0.775
```

False Positive Rate

```
false_positive_rate_cross_validated_data <- (18/38)
false_positive_rate_cross_validated_data
```

```
## [1] 0.4736842
```

Precision

```
precision_cross_validated_data <- (62/80)
precision_cross_validated_data
```

```
## [1] 0.775
```

Recall

```
recall_cross_validated_data <- 62/ (62+40)
recall_cross_validated_data
```

```
## [1] 0.6078431
```

F1-Score

```
f1_score_cross_validated_data <- (2*precision_cross_validated_data*recall_cross_validated_dat
a)/(precision_cross_validated_data+recall_cross_validated_data)

f1_score_cross_validated_data
```

```
## [1] 0.6813187
```

Our model provides good accuracy, more than the model without cross-validated accuracy. It leads to increase in every rate that we previously calculated in the previous model, which makes sense as k-fold cross validation would increase the accuracy of the model.

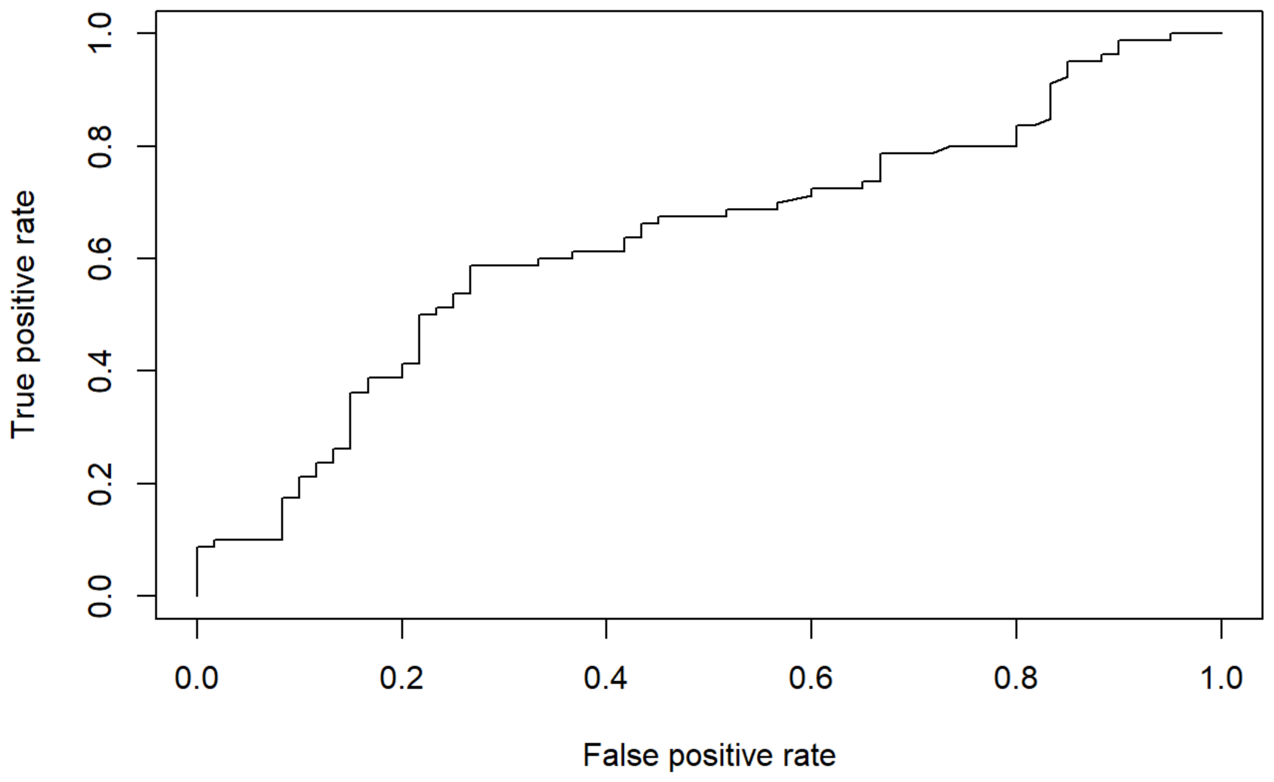
Section 1.3

Now we plot the ROC curve for Section 1.2.

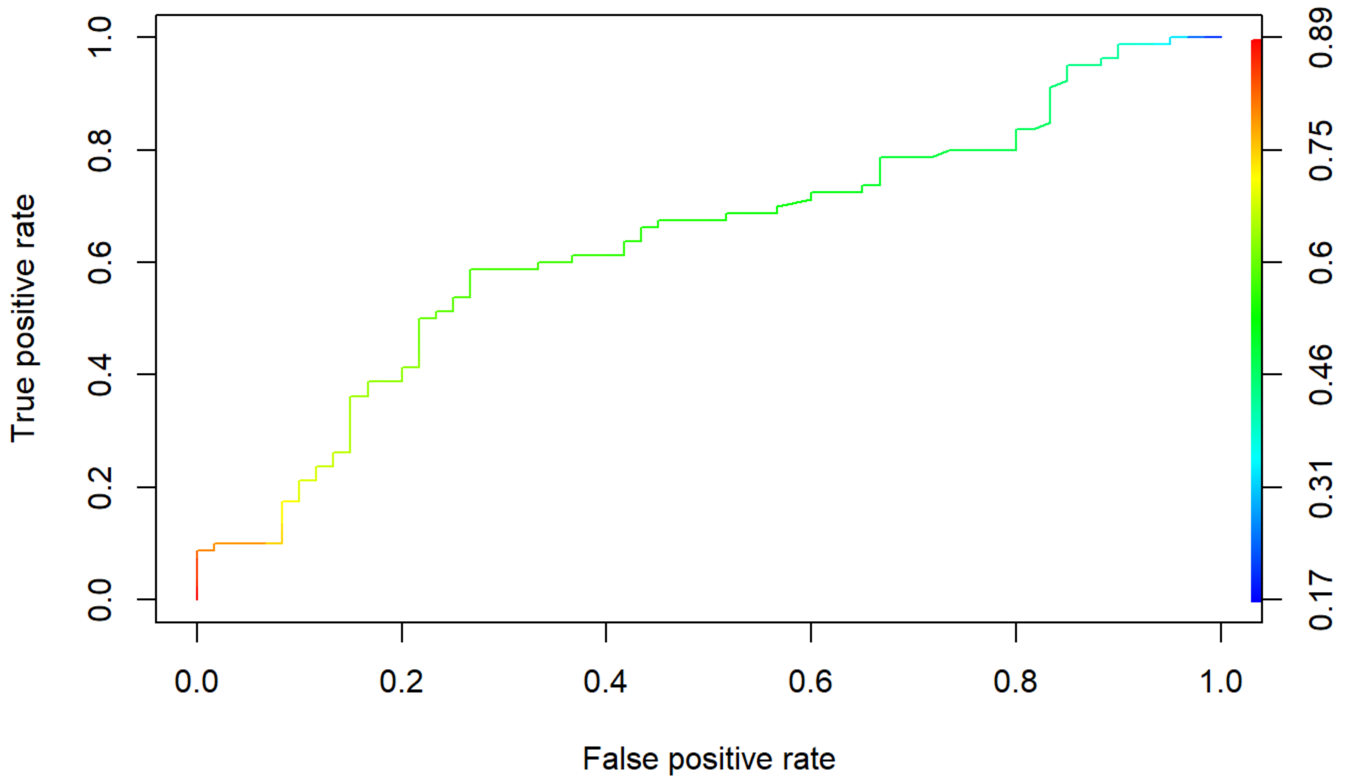
```
rocr_pred = prediction(probability_cross_validated_data, features$isLetter)

rocr_performance <- performance(rocr_pred, "tpr", "fpr")

plot(rocr_performance)
```



```
plot(rocr_performance, colorize = TRUE)
```



This is a decent classifier, not perfect. It has chance level specificty and sensitivity but it’s not completely straight and with high false positive rate, the true positive rate increases as well.

At threshold, we will not collect any poor case however we ill correctly label good care cases. At the end, we will catch all poor care cases and incorrectly label all good care cases as poor cases.

However since the line is not straight, we can say that our model is not a random classifier. It is a decent classifier but not perfect

Section 2

We first set the working directory and the source code.

```
setwd("C:/Users/Asus/Desktop/AI Assignment")

source("./section2_code.r")
```

We first install the library to use knn model and then we import the data as features

```
library(class)

features <-  read.csv(file = './40280938_features.csv', header = TRUE)
```

Now, we only choose a, j and no_letters as per the assignment’s specification. We know that a letters are till 1st to 8th row, j from 73rd to 80th and non-letters from 81st to 140th row.

```
a_letters <- features[1:8,]
j_letters <- features[73:80,]
all_non_letters <- features[81:140,]
```

Once done above, we add all the above data in a separate new dataframe. This will be used throughout the second section.

```
new_features <- rbind(a_letters, j_letters, all_non_letters)
```

Since we'll be using 'label' for KNN classification, we make 'label' column as a categorical factor. a' and 'j' are classified as 'letter' while 'sad', 'smiley' and 'xclaim' are classified separetly for four-way classification.

```
levels = c("a","j", "sad", "smiley", "xclaim")

labels = c("letter","letter","sad", "smiley", "xclaim" )


new_features$label <- factor(new_features$label,
                        levels = levels, labels = labels)


str(new_features)
```

```
## 'data.frame':    76 obs. of  18 variables:
##  $ label          : Factor w/ 4 levels "letter","sad",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ index          : int  1 2 3 4 5 6 7 8 1 2 ...
##  $ nr_pix         : int  27 28 25 32 31 38 32 26 17 15 ...
##  $ rows_with_1    : int  0 0 0 1 0 1 0 2 8 8 ...
##  $ cols_with_1    : int  1 0 1 0 2 2 1 0 0 1 ...
##  $ rows_with_3p   : int  7 8 6 8 6 10 9 7 2 1 ...
##  $ cols_with_3p   : int  5 6 6 6 4 6 6 3 2 2 ...
##  $ aspect_ratio   : num  1.14 1.29 1.29 0.9 1.25 ...
##  $ neigh_1        : int  0 0 1 1 1 1 0 2 3 2 ...
##  $ no_neigh_above : int  5 4 4 4 8 6 5 5 4 3 ...
##  $ no_neigh_below : int  5 4 6 6 9 6 7 4 5 4 ...
##  $ no_neigh_left  : int  8 7 7 14 6 10 10 8 7 12 ...
##  $ no_neigh_right : int  8 8 7 12 7 11 8 10 9 12 ...
##  $ no_neigh_horiz : int  8 13 14 20 9 17 15 15 9 12 ...
##  $ no_neigh_vert  : int  7 6 9 10 12 13 11 7 5 2 ...
##  $ connected_areas: int  1 1 1 1 1 1 1 1 2 2 ...
##  $ eyes           : int  1 1 1 1 1 1 1 1 0 0 ...
##  $ custom         : num  0.9907 0.3334 0.0771 0.0138 0.6408 ...
```

We then set the seed to 42

```
set.seed(42)
```

We print out the summary and sample head first:

```
summary(new_features)
```

```
##     label        index            nr_pix       rows_with_1      cols_with_1
##  letter:16   Min.   : 1.000   Min.   :10.00   Min.   : 0.000   Min.   :0.000
##  sad   :20   1st Qu.: 4.000   1st Qu.:18.00   1st Qu.: 1.000   1st Qu.:0.000
##  smiley:20   Median : 8.000   Median :25.00   Median : 2.000   Median :2.000
##  xclaim:20   Mean   : 9.237   Mean   :24.25   Mean   : 3.342   Mean   :2.013
##              3rd Qu.:14.000   3rd Qu.:29.00   3rd Qu.: 6.000   3rd Qu.:4.000
##              Max.   :20.000   Max.   :42.00   Max.   :11.000   Max.   :6.000
##   rows_with_3p     cols_with_3p    aspect_ratio       neigh_1
##  Min.   : 0.000   Min.   :1.000   Min.   :0.0000   Min.   :0.000
##  1st Qu.: 2.000   1st Qu.:2.000   1st Qu.:0.2024   1st Qu.:2.000
##  Median : 3.000   Median :4.000   Median :0.7386   Median :2.500
##  Mean   : 3.329   Mean   :3.632   Mean   :0.6511   Mean   :3.671
##  3rd Qu.: 5.000   3rd Qu.:5.000   3rd Qu.:0.9183   3rd Qu.:6.000
##  Max.   :10.000   Max.   :8.000   Max.   :1.5000   Max.   :8.000
##  no_neigh_above   no_neigh_below   no_neigh_left   no_neigh_right
##  Min.   : 2.000   Min.   : 2.000   Min.   : 6.00   Min.   : 7.00
##  1st Qu.: 4.000   1st Qu.: 4.000   1st Qu.:10.00   1st Qu.:10.00
##  Median : 7.000   Median : 7.000   Median :12.00   Median :12.00
##  Mean   : 6.776   Mean   : 6.961   Mean   :12.55   Mean   :12.63
##  3rd Qu.: 9.000   3rd Qu.: 9.000   3rd Qu.:14.00   3rd Qu.:14.25
##  Max.   :14.000   Max.   :12.000   Max.   :27.00   Max.   :28.00
##  no_neigh_horiz  no_neigh_vert    connected_areas      eyes
##  Min.   : 0.00   Min.   : 0.000   Min.   :1.000   Min.   :0.0000
##  1st Qu.: 9.00   1st Qu.: 1.000   1st Qu.:2.000   1st Qu.:0.0000
##  Median :11.00   Median : 5.000   Median :4.000   Median :0.0000
##  Mean   :12.08   Mean   : 4.908   Mean   :2.947   Mean   :0.1053
##  3rd Qu.:15.50   3rd Qu.: 7.000   3rd Qu.:4.000   3rd Qu.:0.0000
##  Max.   :31.00   Max.   :13.000   Max.   :4.000   Max.   :1.0000
##      custom
##  Min.   :0.007946
##  1st Qu.:0.309395
##  Median :0.510595
##  Mean   :0.499540
##  3rd Qu.:0.721870
##  Max.   :0.990674
```

```
head(new_features, 10)
```

	label <fct>	ind... <int>	nr_pix <int>	rows_with_1 <int>	cols_with_1 <int>	rows_with_3p <int>	cols_with_3p <int>	aspect_rati <dbl
1	letter	1	27	0	1	7	5	1.142857
2	letter	2	28	0	0	8	6	1.285714
3	letter	3	25	0	1	6	6	1.285714
4	letter	4	32	1	0	8	6	0.900000

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_rati
	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<dbl
5	letter	5	31	0	2	6	4	1.250000
6	letter	6	38	1	2	10	6	1.200000
7	letter	7	32	0	1	9	6	1.375000
8	letter	8	26	2	0	7	3	1.000000
73	letter	1	17	8	0	2	2	0.250000
74	letter	2	15	8	1	1	2	0.166666

1-10 of 10 rows | 1-10 of 19 columns

Afterwards we shuffle the data.

```
new_features <- new_features[sample(nrow(new_features)),]

new_features
```

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_ra
	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<db
113	smiley	13	30	3	2	5	5	0.636363
129	xclaim	9	13	9	0	0	1	0.083333
89	sad	9	29	1	5	4	5	0.818181
82	sad	2	25	1	3	3	4	0.636363
140	xclaim	20	10	10	0	0	1	0.000000
111	smiley	11	42	2	1	8	8	0.769230
88	sad	8	26	2	3	5	5	0.800000
101	smiley	1	23	3	4	4	4	0.888888
84	sad	4	32	1	5	3	3	0.714285
90	sad	10	33	1	6	6	4	0.916666

1-10 of 76 rows | 1-10 of 19 columns   Previous **1** 2 3 4 5 6 ... 8 Next

```
head(new_features, 10)
```

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_ra
	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<db
113	smiley	13	30	3	2	5	5	0.636363
129	xclaim	9	13	9	0	0	1	0.083333

	label	ind…	nr_pix	rows_with_1	cols_with_1	rows_with_3p	cols_with_3p	aspect_ra
	<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<db
89	sad	9	29	1	5	4	5	0.818181
82	sad	2	25	1	3	3	4	0.636363
140	xclaim	20	10	10	0	0	1	0.000000
111	smiley	11	42	2	1	8	8	0.769230
88	sad	8	26	2	3	5	5	0.800000
101	smiley	1	23	3	4	4	4	0.888888
84	sad	4	32	1	5	3	3	0.714285
90	sad	10	33	1	6	6	4	0.916666

1-10 of 10 rows | 1-10 of 19 columns

We then have to splice the data into training set and test set.

However here we use the entire data as the training data and there's no separate test set. So we use '1.0' value for size. We randomly select 100% of data.

```
new_features_fold <- sample(1:nrow(new_features),size=nrow(new_features)*1,replace = FALSE) #
random selection of 100% data.
```

We then select the features for classification:

```
fs = c('nr_pix', 'aspect_ratio', 'rows_with_1', 'cols_with_1')
```

We used the above four features as in the previous assignment, those features were very helpful is discriminating between different types of images. They had the most varied values and were properly calculated amongst other features. We need features with higher variability and precision to determine accurate classification.

Section 2.1

We create a vector to store all accuracies in:

```
accs_2_1=c()
```

Then we put all the separate odd values of k from 1 to 13 to go through:

```
knn_values <- c(1, 3, 5, 7, 9, 11, 13)
```

We perform our knn classification (We won't print individual results as they're too many and big to report in the report):

```
for (i in knn_values) {

  train_items_this_fold  = new_features[new_features_fold,]
  validation_items_this_fold = new_features[new_features_fold,]

  # fit knn model on this fold

  predictions = knn(train_items_this_fold[,fs],
                    validation_items_this_fold[,fs],
                    train_items_this_fold$label, k=i)
  predictions

  correct_list = predictions == validation_items_this_fold$label
  nr_correct = nrow(validation_items_this_fold[correct_list,])

  acc_rate = nr_correct/nrow(validation_items_this_fold)
  accs_2_1[i] = acc_rate
  print(acc_rate)
}
```

```
accs_2_1 <- accs_2_1[!is.na(accs_2_1)]

accs_2_1
```

```
## [1] 1.0000000 0.8026316 0.7368421 0.7105263 0.6710526 0.6578947 0.6184211
```

We report the accuracy over the full set of 76 items ran by our knn_classification model by the odd values of k.

We report high accuracy in our model as all the accuracies have been above 60%, thus noting that our model is good in performing four way classification.

Section 2.2

In this section, we run the same model as in Section 2.1 However, we cross-validate it with 5-fold cross validation first.

As usual, we put k values to go through in a variable along with a new vector to store all accuracies of this model in:

```
knn_values <- c(1, 3, 5, 7, 9, 11, 13)

accs_2_2=c()
```

We then perform the model:

```r
library(caret)

for (i in knn_values) {

  # define training control
  train_control <- trainControl(method="cv", number=5)

  tune_grid <- expand.grid(k = i)

  # train the model
  model <- train(label~nr_pix+aspect_ratio+cols_with_1+rows_with_1, data=new_features,
                 trControl=train_control, tuneGrid=tune_grid, method="knn")
  # summarize results
  print(model)

  # Store accuracy rate

  accs_2_2[i] = model$results$Accuracy

}
```

We have to hide the results as the amount of times this model will run is too many.

We print out the accuracy rate after removing NA values that come with adding it to a vector:

```r
accs_2_2 <- accs_2_2[!is.na(accs_2_2)]

accs_2_2
```

```
## [1] 0.6175000 0.5666667 0.6975000 0.5525000 0.5300000 0.5525000 0.5275000
```

Oddly, we can see that accuracy has decreased for each value of k in comparison to classification without cross-validation. The model is not as accurate with cross-validation.

Section 2.3

We can see that the value of k = 1 has the highest accuracy with 66%. So we will use that for this section.

We run the model again but this time, not with different values. With just k = 3, in order to summarize it in the form of a confusion matrix.

```
# define training control
train_control_3 <- trainControl(method="cv", number=5)

tune_grid_3 <- expand.grid(k = 1)

# train the model
model_3 <- train(label~nr_pix+aspect_ratio+cols_with_1+rows_with_1, data=new_features,
               trControl=train_control_3, tuneGrid=tune_grid_3, method="knn")
# summarize results
print(model_3)
```

```
## 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: 61, 61, 61, 61, 60
## Resampling results:
##
##   Accuracy  Kappa
##   0.645     0.5241382
##
## Tuning parameter 'k' was held constant at a value of 1
```

```
confusionMatrix(model_3)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction letter  sad smiley xclaim
##     letter   11.8  0.0    1.3    2.6
##     sad       2.6 15.8   10.5    0.0
##     smiley    2.6 10.5   14.5    1.3
##     xclaim    3.9  0.0    0.0   22.4
##
##  Accuracy (average) : 0.6447
```

Discriminating letter with sad images, sad with exclamatory images, exclamatory with sad and exclamatory with smiley are the most difficulty pairs to discriminate from.

Section 2.4

For this we first install the ggplot library.

```
library(ggplot2)
```

Then we make a separate dataframe which includes all the accuracies and the k-values:

```
plotting_data_frame <- data.frame(accs_2_1,accs_2_2, knn_values)

plotting_data_frame
```

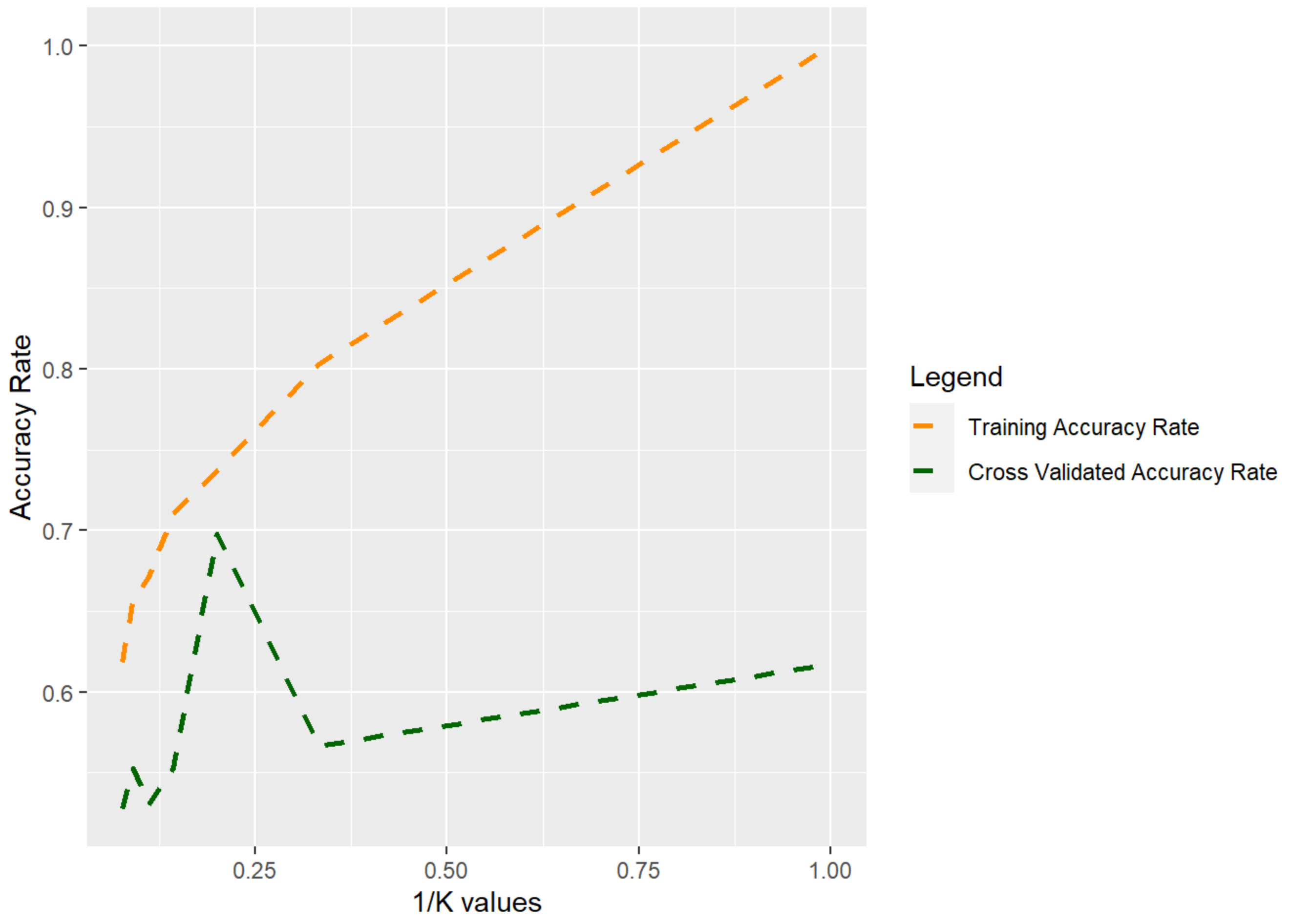
accs_2_1 <dbl>	accs_2_2 <dbl>	knn_values <dbl>
1.0000000	0.6175000	1
0.8026316	0.5666667	3
0.7368421	0.6975000	5
0.7105263	0.5525000	7
0.6710526	0.5300000	9
0.6578947	0.5525000	11
0.6184211	0.5275000	13

7 rows

Then make a graph using ggplot function and plot it:

```
plot_2 <- ggplot()+
  geom_line(data=plotting_data_frame,aes(y=accs_2_1,x= 1/knn_values,colour="Training Accuracy
Rate"),size=1, linetype="dashed" )+
  geom_line(data=plotting_data_frame,aes(y=accs_2_2,x= 1/knn_values,colour="Cross Validated A
ccuracy Rate"),size=1, linetype="dashed") +
  scale_color_manual(name = "Legend", values = c("Training Accuracy Rate" = "darkorange", "Cr
oss Validated Accuracy Rate"= "darkgreen")) + labs(y= "Accuracy Rate", x = "1/K values")

plot_2
```



Here, we have plotted accuracy rate of training data and cross validated data against 1/K. We can see that as 1/K increases, the accuracy rate increases. There was a steady rise before K value (K value = 4 at maximum - in our data, we used odd values of K from 1 to 13) and then decline till just before halfway of maximum k value = 2. Then K value increases, with increase accuracy and declining error rate.

Section 3

```
setwd("C:/Users/Asus/Desktop/AI Assignment")

source("./section3_code.r")
```

We first set the directory listing for CSV files:

```
imagesDir<-"./Images"


imagesDir
```

```
## [1] "./Images"
```

Then we put all the features into a separate variable:

```
allData <- read.table("./Images/all_features.csv")
allData
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
<chr>	<int>	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
a	0	30	0	0	7	6	1.28571429	0	7
a	1	35	0	1	7	6	1.00000000	0	7
a	2	34	0	1	5	5	1.00000000	0	10
a	3	35	0	4	5	4	1.40000000	1	11
a	4	51	3	0	9	6	0.84615385	1	6
a	5	35	1	0	9	6	0.81818182	1	6
a	6	43	2	0	9	5	1.00000000	0	11
a	7	46	7	0	3	11	0.84615385	1	27
a	8	44	0	1	10	6	1.09090909	0	7
a	9	34	0	1	6	4	0.66666667	0	4

1-10 of 1,300 rows | 1-10 of 18 columns Previous **1** 2 3 4 5 6 ... 130 Next

We then set the column names of the data:

```
colnames(allData) <- c("label", "index", "nr_pix", "rows_with_1", "cols_with_1",
                       "rows_with_3p", "cols_with_3p", "aspect_ratio", "neigh_1",
                       "no_neigh_above", "no_neigh_below", "no_neigh_left", "no_neigh_right",
                       "no_neigh_horiz", "no_neigh_vert", "connected_areas", "eyes",
                       "custom")
```

We make ‘label’ as categorical factor as we’ll be classifying our data based on it. All the letters from ‘a’ to ‘j’ classified as letters, and others based on their emoticons.

```
 levels = c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "sad", "smiley", "xclaim")

 labels = c("letter","letter","letter","letter","letter","letter",
          "letter","letter","letter","letter", "sad", "smiley", "xclaim" )



allData$label <- factor(allData$label,
                         levels = levels, labels = labels)

str(allData)
```

```
## 'data.frame':    1300 obs. of  18 variables:
##  $ label          : Factor w/ 4 levels "letter","sad",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ index          : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ nr_pix         : int  30 35 34 35 51 35 43 46 44 34 ...
##  $ rows_with_1    : int  0 0 0 0 3 1 2 7 0 0 ...
##  $ cols_with_1    : int  0 1 1 4 0 0 0 0 1 1 ...
##  $ rows_with_3p   : int  7 7 5 5 9 9 9 3 10 6 ...
##  $ cols_with_3p   : int  6 6 5 4 6 6 5 11 6 4 ...
##  $ aspect_ratio   : num  1.286 1 1 1.4 0.846 ...
##  $ neigh_1        : int  0 0 0 1 1 1 0 1 0 0 ...
##  $ no_neigh_above : int  7 7 10 11 6 6 11 27 7 4 ...
##  $ no_neigh_below : int  6 9 10 12 8 6 12 27 8 6 ...
##  $ no_neigh_left  : int  7 9 12 10 7 13 14 12 12 16 ...
##  $ no_neigh_right : int  6 9 12 11 13 12 14 14 11 15 ...
##  $ no_neigh_horiz : int  6 7 14 16 7 14 15 13 9 15 ...
##  $ no_neigh_vert  : int  6 9 12 18 9 8 16 28 7 7 ...
##  $ connected_areas: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ eyes           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ custom         : int  0 0 0 4 1 2 1 0 0 1 ...
```

here, we choose 80 values of each of the 13 image types as training data and the rest as test data. Also, we have removed the custom feature as it was stated in the assignment specification.

```
training_data <- allData[allData$index<80, 1:17]

test_data <- allData[allData$index >= 80, 1:17]
```

Section 3.1

Here we will perform classification with random forests using 5-fold cross validation.

We first install the libraries needed and set seed:

```
library(caret)
library(randomForest)
set.seed(42)
```

We first do the cross-validation. We perform a grid search of two hyper-parameters for our cross validation.

```
train_control = trainControl(method="cv", number = 5, search = "grid", savePredictions = T)

set.seed(42)

tuningGrid <- expand.grid(mtry=c(2,4,6,8))
```

Now we will use random-forests with number of trees between 25 and 375 at increment of 50, and number of parameters individually at 2, 4, 6 and 8.

```
set.seed(42)

# For number of trees = 25

model_25 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 25)

model_25
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8836538  0.6929488
##   4     0.8846154  0.6973691
##   6     0.8875000  0.7052548
##   8     0.8730769  0.6702333
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

```
model_25$finalModel$ntree
```

```
## [1] 25
```

```
# For number of trees = 75

model_75 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 75)

model_75
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8769231  0.6761849
##   4     0.8826923  0.6931175
##   6     0.8750000  0.6762343
##   8     0.8711538  0.6660914
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

```
model_75$finalModel$ntree
```

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

```
# For number of trees = 125

model_125 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 125)

model_125
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8865385  0.7005126
##   4     0.8750000  0.6703543
##   6     0.8769231  0.6791391
##   8     0.8759615  0.6773404
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_125$finalModel$ntree
```

```
## [1] 125
```

```
# For number of trees = 175

model_175 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 175)

model_175
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8769231  0.6697701
##   4     0.8759615  0.6749948
##   6     0.8730769  0.6698518
##   8     0.8701923  0.6593521
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_175$finalModel$ntree
```

```
## [1] 175
```

```
# For number of trees = 225

model_225 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 225)

model_225
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8855769  0.6952499
##   4     0.8807692  0.6899596
##   6     0.8778846  0.6862747
##   8     0.8798077  0.6886198
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_225$finalModel$ntree
```

```
## [1] 225
```

```
# For number of trees = 275

model_275 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 275)

model_275
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8817308  0.6847308
##   4     0.8817308  0.6862741
##   6     0.8769231  0.6787524
##   8     0.8807692  0.6877542
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_275$finalModel$ntree
```

```
## [1] 275
```

```
# For number of trees = 325

model_325 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree =325)

model_325
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8942308  0.7216262
##   4     0.8836538  0.6978879
##   6     0.8807692  0.6913834
##   8     0.8788462  0.6853691
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_325$finalModel$ntree
```

```
## [1] 325
```

```
# For number of trees = 375

model_375 = train(label~., data=training_data, method="rf", trControl = train_control,
              tuneGrid = tuningGrid, ntree = 375)

model_375
```

```
## Random Forest
##
## 1040 samples
##   16 predictor
##    4 classes: 'letter', 'sad', 'smiley', 'xclaim'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 832, 832, 832, 832, 832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8817308  0.6867935
##   4     0.8788462  0.6807145
##   6     0.8769231  0.6772713
##   8     0.8769231  0.6779904
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
model_375$finalModel$ntree
```

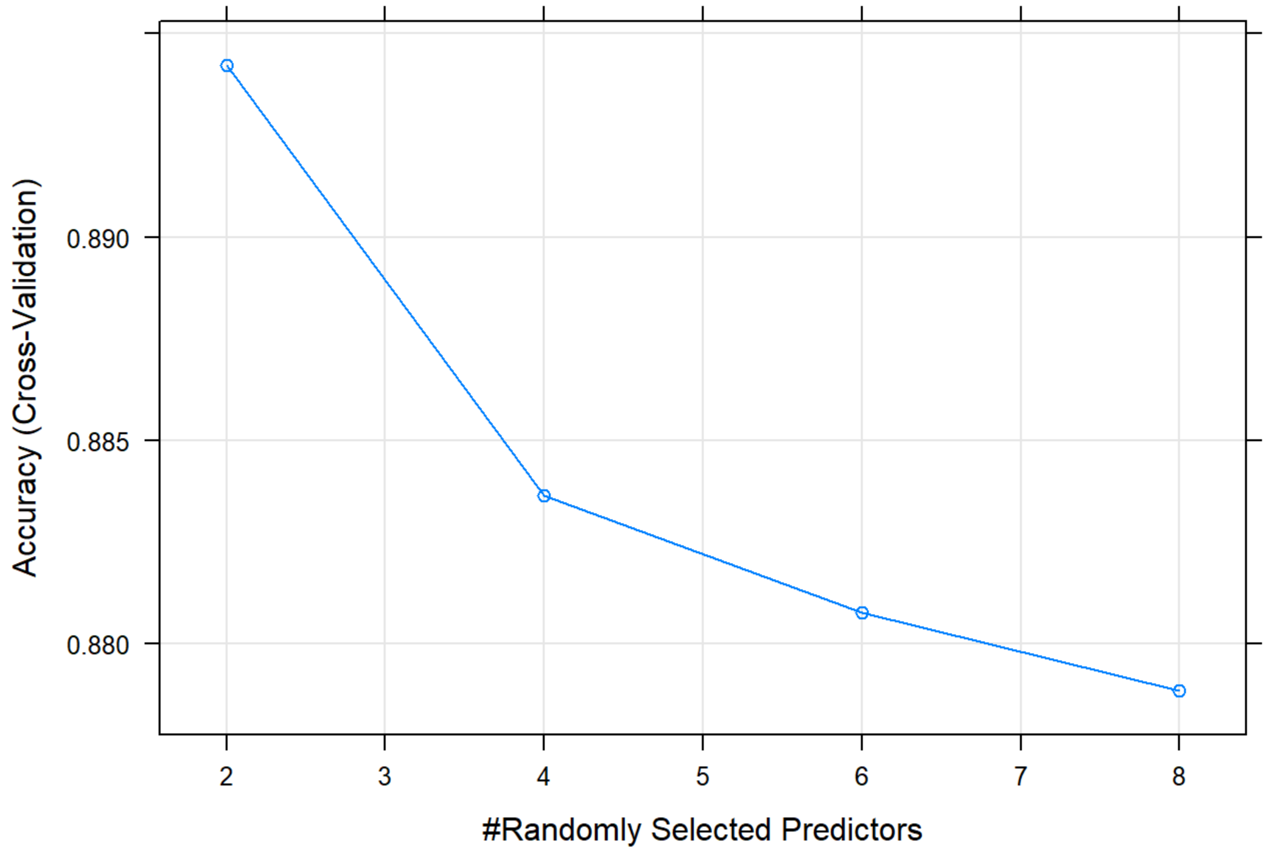
```
## [1] 375
```

Now the model with the highest accuracy is the second last model with number of trees = 325 and predictor number as 2. Hence it’s the best model.

```
model_highest_accuracy = 0.8942308
```

We plot this model!

```
plot.train(model_325)
```



From the graph, it’s visible that the random selected features we chose using cross-validation are 2, 4, 6 and 8.

The best predictor number is 2 for this model and apparently from the graph, the more number of predictor numbers we have, there’s a chance of more redundancy predictor numbers selected as split nodes. Hence that’s the reason for decrease of predictions.

Section 3.2

In this section, we’ll use the best model from the previous section. The model with Nt = 325. We first set the cross-validation to 5-fold.

```
train_control_refitted = trainControl(method="cv", number = 5, search = "grid", savePredictio
ns = T)



tuningGrid_refitted <- expand.grid(mtry=c(2,4,6,8))
```

We then create a vector to store all the accuracies of the model in:

```
all_accuracies <- c()

all_accuracies
```

```
## NULL
```

Now, we refit the model 15 times. That means we run the same model 15 times and note the highest accuracy to the ‘all_accuracies’ vector.

```
for (i in seq(1,15,1)) {

model_325 = train(label~., data=training_data, method="rf", trControl = train_control,
                  tuneGrid = tuningGrid, ntree = 325)

all_accuracies <- append(all_accuracies, max(model_325$results$Accuracy))


}
```

Then we print all the accuracies:

```
print(all_accuracies)
```

```
##  [1] 0.8894231 0.8826923 0.8884615 0.8894231 0.8826923 0.8788462 0.8778846
##  [8] 0.8865385 0.8875000 0.8798077 0.8855769 0.8730769 0.8836538 0.8788462
## [15] 0.8903846
```

We get the mean of all accuracies:

```
mean_of_accuracies <- mean(all_accuracies)
mean_of_accuracies
```

```
## [1] 0.8836538
```

We get the standard deviation of all accuracies:

```
standard_deviation_of_accuracies <- sd(all_accuracies)
standard_deviation_of_accuracies
```

```
## [1] 0.005139639
```

Our standard deviation is significantly low. This means that we’re confident of our estimators of the model.

Now we check if our model performs better than chance.

We perform a one-sample t-test.

```
t.test(all_accuracies)
```

```
##
##  One Sample t-test
##
## data:  all_accuracies
## t = 665.88, df = 14, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.8808076 0.8865001
## sample estimates:
## mean of x
## 0.8836538
```

Since the value of p is very low, we know that our model performed didn’t have any chance in it and performed significantly better than chance.