Smart Phone Sentiment Analysis and Predictions for Medical App Development

Project background: This project involved working with a government health organization. The overall objective was to develop medical apps to allow aid workers to manage health conditions by facilitating communication with medical professionals elsewhere. However, to limit cost and ensure consistency of training, the agency required the apps be developed for only one cell phone type.

My task was to perform web analysis to determine the prevalence of sentiment toward a short list of devices that were previously determined capable of executing the app's functions. The goal was to select one device by conducting a broad-based sentiment analysis.

The work flow for this project was divided into **two sections**. **The first involved** the identification and data collections. In the initial step of collecting the input web addresses the wet paths file for the month of interest was downloaded from the Common Crawl. Since the project calls for sentiment mining, work was focused on a subset of WARC files that only contain text:WET. Data mining was done Using Amazon's EMR with previously written python mapper and reducer scripts to access and compile web pages from the Common Crawl. An initial exploratory EMR Cluster on one web address was created and run to ensure the job would run smoothly. The data output was then downloaded and using the Anaconda prompt and a concatenate program the data was compiled into a single csv file. Once smooth work flow was confirmed, an additional 255 web addresses were searched by creating a large cluster using the AWS CLI. For this step a .bdf file was created containing the 255 addresses of interest, the addresses were mapped to the AWS S3:// commoncrawl/ folder, and the .bdf folder was then converted to a Json file, a cluster was created on the AWS CLI using Anaconda prompt and the Json file was run in the EMR cluster. The data output files (255 files) of the job was then down loaded and using a concatenate program combined into one csv folder containing over 17,000 observations and attributes describing sentiment toward listed cell phone types. This data set can them be used and predict sentiment to determine the preferred device for our medical app development.

The **Second section** of this work involved the development of machine learning models that predict cell phone sentiment for two preferred cell phones (iphone & Samsung Galaxy) as determined by the health government Agency. For training data sets we have been provided an iphone dataset and a galaxy dataset. These data sets include counts of pertinent words, describing sentiment, for approximately 12,000 observations. The assigned values in the sentiment attribute is representative of the overall sentiment toward the device. These values were assigned by an independent group by read each review rating sentiment. The rating scale is : "0 = very negative, 1 = negative, 2 = somewhat negative, 3 = somewhat positive, 4 = positive, 5 = very positive.

Prediction will be made on the data set that was collected for AWS in the first step of this project.

Begin by looking at iPhone training data set:

 # Set up for parallel processing for faster processing
install.packages("doParallel")
library(doParallel)
find out how many cores are on your machine
detectCores()
create cluster with desired number of cores. dont use them all.
cl <- makeCluster(2)
register cluster
registerDoParallel(cl)
confirm how many cores are now "assigned" to R and Rstudio
getDoParWorkers()
stop Cluster. after performing tasks, stop your cluster
use stopCluster(cl)

 # install packages
library(readr)
library(caret)
library(plotly)
library(corrplot)
library(dplyr)

#upload the data
iphone_smallmatrix <- read.csv("iphone_smallmatrix_labeled_8d.csv")

 # Data at a glance
names(iphone_smallmatrix)

```
[1] "iphone"          "samsunggalaxy"   "sonyxperia"
 [4] "nokialumina"     "htcphone"        "ios"
 [7] "googleandroid"   "iphonecampos"    "samsungcampos"
[10] "sonycampos"      "nokiacampos"     "htccampos"
[13] "iphonecamneg"    "samsungcamneg"   "sonycamneg"
[16] "nokiacamneg"     "htccamneg"       "iphonecamunc"
[19] "samsungcamunc"   "sonycamunc"      "nokiacamunc"
[22] "htccamunc"       "iphonedispos"    "samsungdispos"
[25] "sonydispos"      "nokiadispos"     "htcdispos"
[28] "iphonedisneg"    "samsungdisneg"   "sonydisneg"
[31] "nokiadisneg"     "htcdisneg"       "iphonedisunc"
[34] "samsungdisunc"   "sonydisunc"      "nokiadisunc"
[37] "htcdisunc"       "iphoneperpos"    "samsungperpos"
[40] "sonyperpos"      "nokiaperpos"     "htcperpos"
[43] "iphoneperneg"    "samsungperneg"   "sonyperneg"
[46] "nokiaperneg"     "htcperneg"       "iphoneperunc"
[49] "samsungperunc"   "sonyperunc"      "nokiaperunc"
[52] "htcperunc"       "iosperpos"       "googleperpos"
[55] "iosperneg"       "googleperneg"    "iosperunc"
[58] "googleperunc"    "iphonesentiment"
```

Explanation of the names:

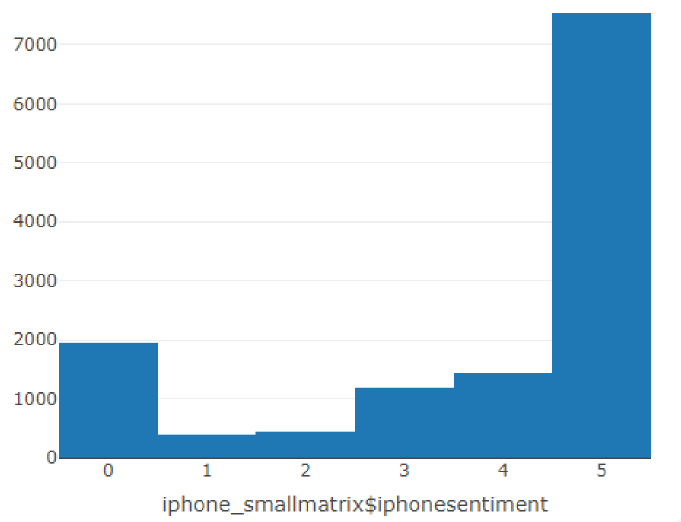
- Iphone, samsunggalaxy, sonyxperia , nokialumina, and htcphone are the five phones on the list that have been determined to be able to perform the app's fuctions.
- ios is the count of mentions of the iphone operating system on the review
- iphonecampos- is the count mentions of positive sentiment towards the iphone camera
- iphonedisunc- is the count of unclear setiment metions toward the iphone display
- iphoneperneg- is the count mentions of negative sentiment towards the iphone performance

 # check data types of attributes
str(iphone_smallmatrix)
str(iphone_smallmatrix$iphonesentiment)

```
str(iphone_smallmatrix$iphonesentiment)
 int [1:12973] 0 0 0 0 0 4 4 0 0 0 ...
```

upon checking the structure of the attributes it is clear that all attributes are integers. For visualization I have only included the structure of the iphonesentiment, which will also be the dependent variable for machine learning and sentiment prediction.

 # to further understand the data, I then plot & study the distribution of the #dependent variable scores.
plot_ly(iphone_smallmatrix, x= ~iphone_smallmatrix$iphonesentiment, type= 'histogram')



Based on the ditrubution plot it is clear that over 50% of the reviews have a positive sentiment toward the iphone.

 # check for missing data and adress if needed
sum(is.na(iphone_smallmatrix))

```
sum(is.na(iphone_smallmatrix))
[1] 0
```

No missing data was found in our data set.

Feature Selection

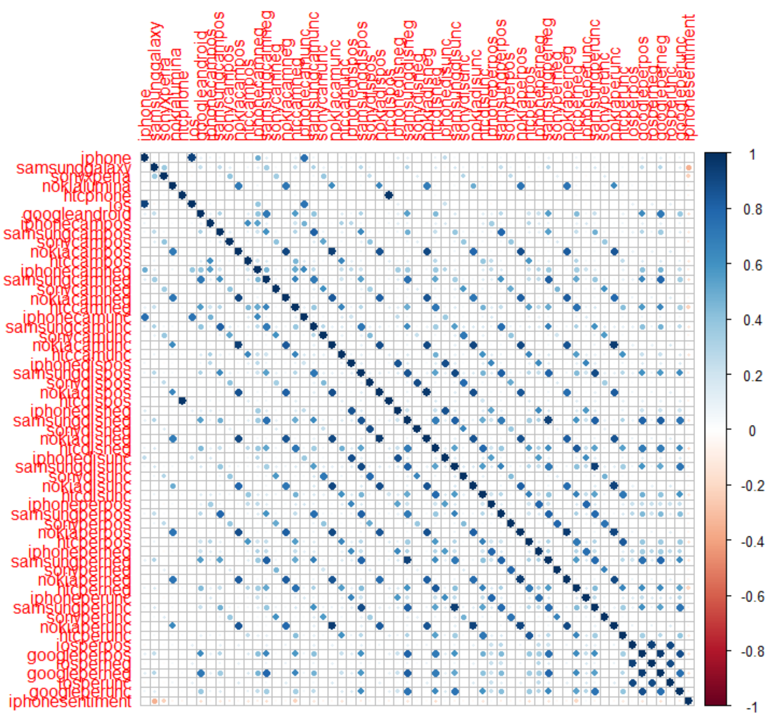
For feature selection we will look at correlation, near zero variance, and recursive feature elimination. For each approach we will create a new data set to test our machine learning models on to determine The best one.

<u>Correlation</u>: even though correlation does not equal causation, a high correlation between an independent variable and dependent variable may cause machine learning model overfitting. Highly correlated features should be removed.

Check <u>for near Zero Variance</u>: Zero or zero variance in variables shows a constant or near constant Predictors across all the observations. This type of constant predictor does not provide much information and may also blog down some of the machine learning algorithms. Attributes with Zero or near Zero variance will be removed.

<u>Recursive feature elimination (RFE)</u>: RFE is a form of automated feature selection. Caret's [rfe() function](#) with random forest will try every combination of feature subsets and return a final list of recommended features. The function accomplishes this by building models to implement backwards selection of attributes based on attribute importance ranking. The least important features are removed and a final list of attributes is recommended.

 # study correlation of between features and dependent variable
library(corrplot)
icorr_data <- cor(iphone_smallmatrix)
corrplot(icorr_data)



We can see from the correlation plot that there no high correlation between iPhone sentiment and the independent variable. Thus no need to remove attributes based on correlation.

based on correlation data no high correlation thus no features will be removed
create a new data frame from the iphone_smallmatrix data frame.
in this case, this new data frame retains all the feature for "out of the box modeling"

iphoneCor <- iphone_smallmatrix
# to remove a feature if it were needed we would use
# iphoneCor$featuretoremove <- Null

 # search and remove near zero variance features. create new data set without nearzero variance
 # using nearZeroVar with saveMetrics =True returns an object containing
at a table: with frequency ration, % unique, zero variance and
near zero variance. it allow us to see the attributes with zero variance.

nzvMetrics <- nearZeroVar(iphone_smallmatrix, saveMetrics = TRUE)
nzvMetrics

```
                 freqRatio percentUnique zeroVar   nzv
iphone            5.041322    0.20812457   FALSE FALSE
samsunggalaxy    14.127336    0.05395822   FALSE FALSE
sonyxperia       44.170732    0.03854159   FALSE  TRUE
nokialumina     497.884615    0.02312495   FALSE  TRUE
htcphone         11.439614    0.06937486   FALSE FALSE
ios              27.735294    0.04624990   FALSE  TRUE
googleandroid    61.247573    0.04624990   FALSE  TRUE
iphonecampos     10.524697    0.23124952   FALSE FALSE
samsungcampos    93.625000    0.08479149   FALSE  TRUE
sonycampos      348.729730    0.05395822   FALSE  TRUE
nokiacampos    1850.142857    0.08479149   FALSE  TRUE
htccampos        79.272152    0.16958298   FALSE  TRUE
iphonecamneg     19.517529    0.13104139   FALSE  TRUE
samsungcamneg   100.132812    0.06937486   FALSE  TRUE
sonycamneg     1851.285714    0.04624990   FALSE  TRUE
nokiacamneg    2158.833333    0.06166654   FALSE  TRUE
htccamneg        93.444444    0.11562476   FALSE  TRUE
iphonecamunc     16.764205    0.16187466   FALSE FALSE
samsungcamunc    74.308140    0.06937486   FALSE  TRUE
sonycamunc      588.318182    0.03854159   FALSE  TRUE
nokiacamunc    2591.200000    0.05395822   FALSE  TRUE
htccamunc        50.548000    0.12333308   FALSE  TRUE
iphonedispos      6.792440    0.24666615   FALSE FALSE
samsungdispos    97.061069    0.13104139   FALSE  TRUE
sonydispos      331.076923    0.06937486   FALSE  TRUE
nokiadispos    1438.777778    0.09249981   FALSE  TRUE
htcdispos        64.694301    0.20041625   FALSE  TRUE
iphonedisneg     10.084428    0.18499961   FALSE FALSE
samsungdisneg    99.155039    0.10791644   FALSE  TRUE
sonydisneg     2159.333333    0.06937486   FALSE  TRUE
nokiadisneg    1850.142857    0.08479149   FALSE  TRUE
htcdisneg        88.492958    0.14645803   FALSE  TRUE
iphonedisunc     11.471875    0.20812457   FALSE FALSE
samsungdisunc    74.255814    0.09249981   FALSE  TRUE
sonydisunc      719.222222    0.05395822   FALSE  TRUE
nokiadisunc    1619.375000    0.04624990   FALSE  TRUE
htcdisunc        50.590361    0.13874971   FALSE  TRUE
iphoneperpos      9.297834    0.19270793   FALSE FALSE
samsungperpos    94.200000    0.10791644   FALSE  TRUE
```

```
sonyperpos        416.870968    0.06166654   FALSE  TRUE
nokiaperpos      2158.000000    0.08479149   FALSE  TRUE
htcperpos          74.279762    0.19270793   FALSE  TRUE
iphoneperneg       11.054137    0.16958298   FALSE FALSE
samsungperneg     101.650794    0.10020812   FALSE  TRUE
sonyperneg       2159.666667    0.07708317   FALSE  TRUE
nokiaperneg      3237.250000    0.09249981   FALSE  TRUE
htcperneg          94.428571    0.15416635   FALSE  TRUE
iphoneperunc       13.018349    0.12333308   FALSE FALSE
samsungperunc      86.500000    0.09249981   FALSE  TRUE
sonyperunc       3240.250000    0.04624990   FALSE  TRUE
nokiaperunc      1850.428571    0.06937486   FALSE  TRUE
htcperunc          50.055556    0.15416635   FALSE  TRUE
iosperpos         153.373494    0.09249981   FALSE  TRUE
googleperpos       98.592308    0.06937486   FALSE  TRUE
iosperneg         141.744444    0.09249981   FALSE  TRUE
googleperneg       99.403101    0.08479149   FALSE  TRUE
iosperunc         135.893617    0.07708317   FALSE  TRUE
googleperunc       96.443609    0.07708317   FALSE  TRUE
iphonesentiment     3.843017    0.04624990   FALSE FALSE
```

 # nearzeroVar () with saveMetrics = FALSE retruns a vector which can then
 # be used to easily and quckily remove the zero variance features.

```
nvz <- nearZeroVar(iphone_smallmatrix, saveMetrics = FALSE)
nvz
```

```
[1]   3  4  6  7  9 10 11 12 13 14 15 16 17 19 20 21 22 24 25 26 27 29 30
[24] 31 32 34 35 36 37 39 40 41 42 44 45 46 47 49 50 51 52 53 54 55 56 57
[47] 58
```

 # remove nearzero variace features and create new dataset

```
iphoneNZV <- iphone_smallmatrix[,-nvz]
str(iphoneNZV)
```

Recursive Feature Elimination:

Recursive Feature Elimination: Here I Apply random forest functions to build the RFE model,
and cross validation to avoid overfitting

 # sample data before using RFE.

```
set.seed(123)
iphoneSample <- iphone_smallmatrix[sample(1:nrow(iphone_smallmatrix), 1000,replace = FALSE),]
```

 # set up rfeControl with randomforest functions, repeated cross validation and no updates

```
ctrl <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   repeats = 5,
                   verbose = FALSE)
```

 # Use rfe sampled data and omit the dependent variable (attribute 59, iphonesentiment)

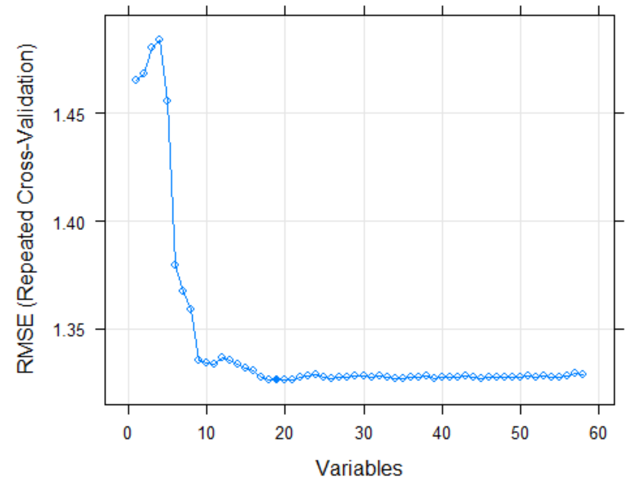
```
rfeResults <- rfe(iphoneSample[,1:58],
                 iphoneSample$iphonesentiment,
                 sizes = (1:58),  rfeControl = ctrl)
```

#get the results

rfeResults

```
Recursive feature selection
Outer resampling method: Cross-Validated (10 fold, repeated 5 times)
Resampling performance over subset size:

 Variables  RMSE Rsquared    MAE  RMSESD RsquaredSD   MAESD Selected
         1 1.465   0.3125 1.1029 0.10778    0.10233 0.07811         
         2 1.468   0.3209 1.1396 0.09954    0.10063 0.06861         
         3 1.480   0.3165 1.1582 0.09438    0.09644 0.06490         
         4 1.484   0.3208 1.1641 0.09356    0.09943 0.06499         
         5 1.456   0.3544 1.1438 0.09178    0.10008 0.06269         
         6 1.380   0.3914 1.0207 0.10184    0.09793 0.06701         
         7 1.368   0.4023 1.0047 0.10361    0.09790 0.06596         
         8 1.359   0.4108 0.9989 0.10481    0.09849 0.06527         
         9 1.336   0.4275 0.9427 0.11560    0.10242 0.07301         
        10 1.334   0.4285 0.9418 0.11219    0.09944 0.06999         
        11 1.334   0.4294 0.9432 0.11624    0.10228 0.07394         
        12 1.337   0.4266 0.9236 0.11800    0.10212 0.07682         
        13 1.335   0.4280 0.9265 0.11851    0.10235 0.07546         
        14 1.334   0.4289 0.9288 0.11791    0.10279 0.07622         
        15 1.332   0.4310 0.9146 0.11959    0.10264 0.07770         
        16 1.331   0.4318 0.9175 0.11699    0.10140 0.07650         
        17 1.327   0.4346 0.9201 0.11624    0.09956 0.07526         
        18 1.326   0.4356 0.9095 0.11788    0.10101 0.07706         
        19 1.326   0.4357 0.9141 0.11725    0.10001 0.07633        *
        20 1.326   0.4354 0.9171 0.11677    0.09974 0.07571         
        21 1.327   0.4353 0.9100 0.11688    0.09934 0.07669         
        22 1.328   0.4344 0.9139 0.11642    0.09921 0.07580         
        23 1.328   0.4341 0.9174 0.11577    0.09896 0.07544         
        24 1.329   0.4338 0.9115 0.11774    0.10038 0.07694         
        25 1.328   0.4344 0.9129 0.11655    0.10000 0.07678         
        26 1.327   0.4350 0.9152 0.11598    0.09915 0.07591         
        27 1.327   0.4348 0.9094 0.11732    0.10013 0.07667         
        28 1.328   0.4345 0.9118 0.11632    0.09931 0.07606         
        29 1.328   0.4341 0.9144 0.11565    0.09912 0.07616         
        30 1.328   0.4344 0.9098 0.11662    0.09968 0.07680         
        31 1.328   0.4344 0.9119 0.11642    0.09978 0.07640         
        32 1.328   0.4341 0.9145 0.11681    0.10004 0.07709         
        33 1.328   0.4344 0.9096 0.11728    0.10005 0.07772         
        34 1.327   0.4350 0.9104 0.11631    0.09928 0.07590         
        35 1.327   0.4350 0.9128 0.11582    0.09919 0.07652         
        36 1.328   0.4347 0.9092 0.11735    0.09993 0.07797         
        37 1.328   0.4346 0.9111 0.11719    0.09974 0.07706         
        38 1.328   0.4341 0.9129 0.11686    0.10025 0.07747         
        39 1.327   0.4349 0.9090 0.11735    0.10007 0.07719         
        40 1.327   0.4347 0.9104 0.11585    0.09926 0.07656         
        41 1.327   0.4347 0.9126 0.11736    0.10008 0.07733         
        42 1.328   0.4347 0.9086 0.11667    0.09928 0.07725         
        43 1.328   0.4342 0.9104 0.11637    0.09966 0.07712         
        44 1.327   0.4347 0.9114 0.11646    0.09971 0.07777         
        45 1.327   0.4354 0.9081 0.11769    0.09995 0.07786         
        46 1.328   0.4345 0.9103 0.11682    0.09984 0.07748         
        47 1.328   0.4345 0.9118 0.11698    0.09998 0.07785         
        48 1.328   0.4347 0.9090 0.11713    0.09983 0.07756         
        49 1.328   0.4345 0.9104 0.11636    0.09945 0.07731         
        50 1.328   0.4345 0.9121 0.11695    0.09978 0.07763         
        51 1.328   0.4341 0.9102 0.11676    0.09943 0.07797         
        52 1.328   0.4345 0.9105 0.11740    0.10010 0.07803         
        53 1.328   0.4343 0.9122 0.11676    0.09975 0.07714         
        54 1.328   0.4344 0.9091 0.11704    0.09981 0.07735         
        55 1.328   0.4343 0.9108 0.11700    0.09965 0.07778         
        56 1.328   0.4341 0.9112 0.11811    0.10062 0.07889         
        57 1.329   0.4332 0.9090 0.11745    0.09983 0.07946         
        58 1.329   0.4336 0.9088 0.11771    0.10066 0.07903         

The top 5 variables (out of 19):
   iphone, googleandroid, iphonedispos, iphonedisneg, samsunggalaxy
```

# Plot rfeResults results**
plot(rfeResults, type=c("g","o"))



The resulting table and plot display each subset and its statistical values. An asterisk denotes the number of features that is judged the most optimal from RFE. Both the table and the plot show the optimal subset is the one that has the lowest RMSE. In this case it consists of 19 variables.

 # create a new data set with the rfe recommended features
iphoneRFE <- iphone_smallmatrix[,predictors(rfeResults)]
 # add the dependent varialbe to iphoneRFE
iphoneRFE$iphonesentiment <- iphone_smallmatrix$iphonesentiment
str(iphoneRFE)

Preprocessing:

Since this is a classification problem our dependent varialble will be converted into a factor. This will be done to all the data sets created via feature selection and the original data set. (in this case, no features were removed from the original data set due to correlation. Thus, since iphone_smallmatrix and iphoncorr data set are the same only one will be used for modeling.

iphone_smallmatrix$iphonesentiment <- as.factor(iphone_smallmatrix$iphonesentiment)
iphoneCor$iphonesentiment<- as.factor(iphoneCor$iphonesentiment)
iphoneNZV$iphonesentiment <- as.factor(iphoneNZV$iphonesentiment)
iphoneRFE$iphonesentiment <- as.factor(iphoneRFE$iphonesentiment)

Model Development and Evaluation

For Model Development in this project now data sampling will be used. The small matrix data set was classified by hand, thus all observations will be used for training and testing. Four classification machine learning algorithms(C5.0, Random Forest, KKNN, and SVM) using the original data set containing all of the features. Since the goal is to find the best combination of algorithm and feature selection method for predicting sentiment, the best model based on accuracy and Kappa will be selected to model on all feature selection data sets. Kappa and accuracy parameters will be compared and the best combination will be chosen to predict sentiment on iPhone in the large data set prepared in step one of this project.

 # create data partition and training and testing sets
set.seed(123)
iphoneinTrain <- createDataPartition(y = iphone_smallmatrix$iphonesentiment,p=0.70, list = FALSE)
str(iphoneinTrain)
iphonetraining <- iphone_smallmatrix [iphoneinTrain,]
iphonetesting <- iphone_smallmatrix [-iphoneinTrain,]
nrow(iphonetraining)
nrow(iphonetesting)

#create fit control
iphonefitControl <- trainControl(method= "repeatedcv", number=10,repeats = 1)

create C5.0 model for classification iphone_smallmatrix data frame
library(C50)
iphone_smallmatrix_C5.0model<- train (iphonesentiment~.,data = iphonetraining, method = "C5.0", trControl=iphonefitControl)
iphone_smallmatrix_C5.0model
varImp(iphone_smallmatrix_C5.0model)
# predicting from iphone testing
iphone_small_C5.0_predictions <- predict(iphone_smallmatrix_C5.0model, iphonetesting)

 # create random forest model for classification iphone_smallmatrix
iphone_smallmatrix_RFmodel<- train (iphonesentiment~.,data = iphonetraining, method = "rf", trControl=iphonefitControl)
iphone_smallmatrix_RFmodel
# predicting from iphone testing
iphone_small_RF_predictions <- predict(iphone_smallmatrix_RFmodel, iphonetesting)

#create kkNN modle for classification of iphone_smallmatrix
iphone_smallmatrix_KKNNmodel<- train (iphonesentiment~.,data = iphonetraining, method = "kknn", trControl=iphonefitControl)
iphone_smallmatrix_KKNNmodel
# predicting from iphone testing
iphone_small_KKNN_predictions <- predict(iphone_smallmatrix_KKNNmodel, iphonetesting)

create SVM model for classification of iphone_smallmatrix data frame
library(e1071)
iphone_smallmatrix_SVMmodel<- train (iphonesentiment~.,data = iphonetraining, method = "svmLinear2", trControl=iphonefitControl)
iphone_smallmatrix_SVMmodel
# predicting from iphone testing
iphone_small_SVM_predictions <- predict(iphone_smallmatrix_SVMmodel, iphonetesting)

Evaluating the models
To evaluate the models I firs use resamples to determine and obtain model metrics from the training (iphonetraining) data partion. Second I will use Postresample to compare the prediction to the groundtruth in the testing (iphonetesting) data partition.

Accuracy describes the number of instances that were classified correctly. kappa describes a comparison between an observed accuracy and an expected accuracy. Expected Accuracy is the accuracy that any random classifier would be expected to achieve. Expected Accuracy is directly related to the number of instances of each class combined with the number of instances that the machine learning classifier agreed with to be ground truth. Overall, Kappa Score is less misleading than simply using accuracy.

#Obtaining accuracy and kappa summary of the for iphone_small matrix models using (resamples)

iphone_smallmatrix_ModelData <- resamples(list(C50 = iphone_smallmatrix_C5.0model, RF = iphone_smallmatrix_RFmodel, kknn = iphone_smallmatrix_KKNNmodel, SVM2 = iphone_smallmatrix_SVMmodel))
summary(iphone_smallmatrix_ModelData)

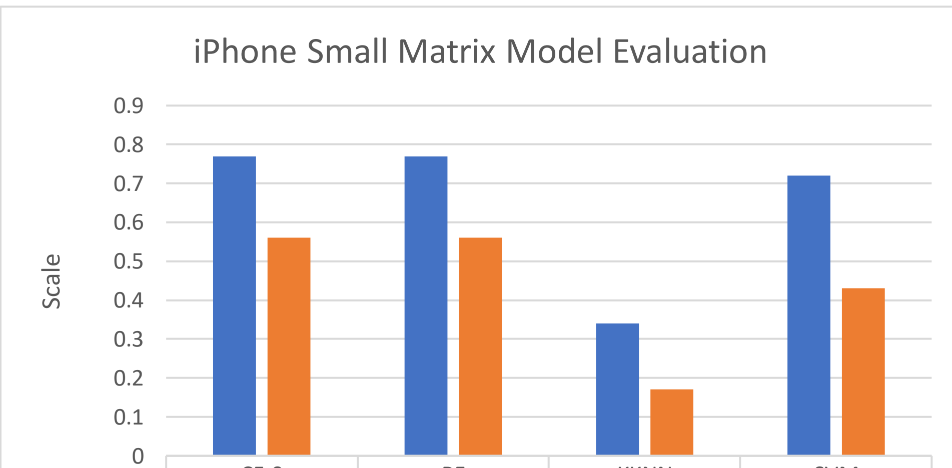
```
Call:
summary.resamples(object = iphone_smallmatrix_ModelData)

Models: C50, RF, kknn, SVM2 
Number of resamples: 10 

Accuracy 
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
C50  0.7577093 0.7663268 0.7713034 0.7715533 0.7778702 0.7850055    0
RF   0.7546755 0.7686389 0.7744772 0.7731025 0.7769824 0.7913907    0
kknn 0.3171806 0.3275526 0.3309512 0.3313906 0.3382941 0.3428886    0
SVM2 0.6938326 0.6979371 0.7077627 0.7079155 0.7155428 0.7282728    0

Kappa 
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
C50  0.5241782 0.5422098 0.5578319 0.5561141 0.5685017 0.5863730    0
RF   0.5257395 0.5535973 0.5658198 0.5627197 0.5693608 0.6015349    0
kknn 0.1395009 0.1586956 0.1626841 0.1621276 0.1696927 0.1761262    0
SVM2 0.3723279 0.3848960 0.4056711 0.4077320 0.4245260 0.4648812    0
```

From using “resamples” function we can see that the model with the highest accuracy and kappa values, during model training, is the random forest algorithm. However, the C5.0 model is also with very similar values.

To study which model is best suited for predictions we will test the predictions against the ground truth on the testing data partition (iphonetesting) we created. This is done using the “postresample” function.

 # evaluate models using postresample to compare predictions to ground truth
 # in testing (iphonetesting) datapartition.

#evaluate C5.0
postResample(iphone_small_C5.0_predictions, iphonetesting$iphonesentiment)
#evaluate RF
postResample(iphone_small_RF_predictions, iphonetesting$iphonesentiment)
#evaluate KKNN
postResample(iphone_small_KKNN_predictions, iphonetesting$iphonesentiment)
#evaluate SVM
postResample(iphone_small_SVM_predictions, iphonetesting$iphonesentiment)



	C5.0	RF	KKNN	SVM
Acurracy	0.77	0.77	0.34	0.72
Kappa	0.56	0.56	0.17	0.43

Upon evaluation of the predictions and ground truth values we can see that model that performed the best are the C5.0 and Random Forest algorithms. For time saving interest I will use the C5.0 machine learning algorithm to test and determine the best feature selection strategy for our classification sentiment prediction strategy.

The C5.0 model has been selected to be used on the feature selction data sets. This model has already been run ont the iphone small matrix data set. So I will only train it using the NZV and RFE feature selection data sets.

#iphnoneNZV feature selection data set

```
iphone_NZVinTrain <- createDataPartition( y = iphoneNZV$iphonesentiment,p=0.70, list =  FALSE)
str(iphone_NZVinTrain)
iphone_NZVtraining <- iphoneNZV [iphone_NZVinTrain,]
iphone_NZVtesting <- iphoneNZV [-iphone_NZVinTrain,]
nrow(iphone_NZVtraining)
nrow(iphone_NZVtesting)
```

#create fit control

```
iphone_NZVfitControl <- trainControl(method= "repeatedcv", number=10,repeats = 1)
```

create C5.0 model for classification iphone_smallmatrix data frame

```
library(C50)
iphone_NZV_C5.0model<- train (iphonesentiment~.,data = iphone_NZVtraining, method = "C5.0",
trControl=iphone_NZVfitControl)
iphone_NZV_C5.0model
varImp(iphone_NZV_C5.0model)
```

 # predicting from iphone testing

```
iphone_NZV_C5.0_predictions <- predict(iphone_NZV_C5.0model, iphone_NZVtesting)
```

#iphoneRFE feature selection data set

```
iphone_RFEinTrain <- createDataPartition( y = iphoneRFE$iphonesentiment,p=0.70, list =  FALSE)
str(iphone_RFEinTrain)
iphone_RFEtraining <- iphoneRFE [iphone_RFEinTrain,]
iphone_RFEtesting <- iphoneRFE[-iphone_RFEinTrain,]
nrow(iphone_RFEtraining)
nrow(iphone_RFEtesting)
```

#create fit control

```
iphone_RFEfitControl <- trainControl(method= "repeatedcv", number=10,repeats = 1)
```

create C5.0 model for classification iphone_smallmatrix data frame

```
iphone_RFE_C5.0model<- train (iphonesentiment~.,data = iphone_RFEtraining, method = "C5.0",
trControl=iphone_RFEfitControl)
iphone_RFE_C5.0model
varImp(iphone_RFE_C5.0model)
```

 # predicting from iphone testing

```
iphone_RFE_C5.0_predictions <- predict(iphone_RFE_C5.0model, iphone_RFEtesting)
```

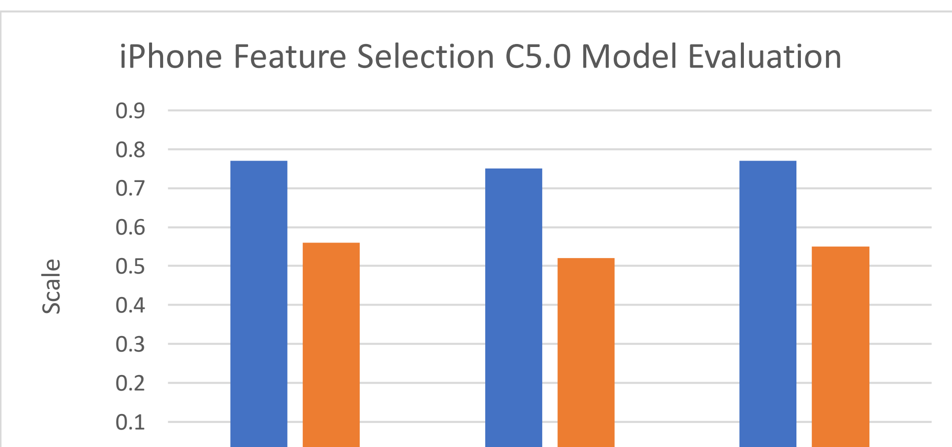
#evaluate C5.0 models with different feature selection data sets based on the training data set using resamples
##Obtaining accuracy and kappa summary of the for C5.0 models using (resamples)

iphone_C5.0_ModelData <- resamples(list(C50_iS = iphone_smallmatrix_C5.0model, C5.0_NZV = iphone_NZV_C5.0model, C5.0_RFE = iphone_RFE_C5.0model))
summary(iphone_C5.0_ModelData)

```
Call:
summary.resamples(object = iphone_C5.0_ModelData)

Models: C50_iS, C5.0_NZV, C5.0_RFE
Number of resamples: 10

Accuracy
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
C50_iS   0.7577093 0.7663268 0.7713034 0.7715533 0.7778702 0.7850055    0
C5.0_NZV 0.7431092 0.7529578 0.7557756 0.7572416 0.7613672 0.7750827    0
C5.0_RFE 0.7588106 0.7685757 0.7763073 0.7740807 0.7798843 0.7843784    0

Kappa
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
C50_iS   0.5241782 0.5422098 0.5578319 0.5561141 0.5685017 0.5863730    0
C5.0_NZV 0.4937573 0.5112421 0.5204902 0.5222698 0.5315565 0.5598355    0
C5.0_RFE 0.5278570 0.5512829 0.5660371 0.5620452 0.5735504 0.5841558    0
```

Evaluate C5.0 models with different feature selection aproaches by using postresample

#evaluate C5.0 with iphone_smallmatrix data set
postResample(iphone_small_C5.0_predictions, iphonetesting$iphonesentiment)
#evaluate C5.0 with iphone_NZV data set
postResample(iphone_NZV_C5.0_predictions, iphonetesting$iphonesentiment)
#evaluate C5.0 with iphoneRFE data set
postResample(iphone_RFE_C5.0_predictions, iphonetesting$iphonesentiment)



	no selction	NZV	RFE
Accuracy	0.77	0.75	0.77
Kappa	0.56	0.52	0.55

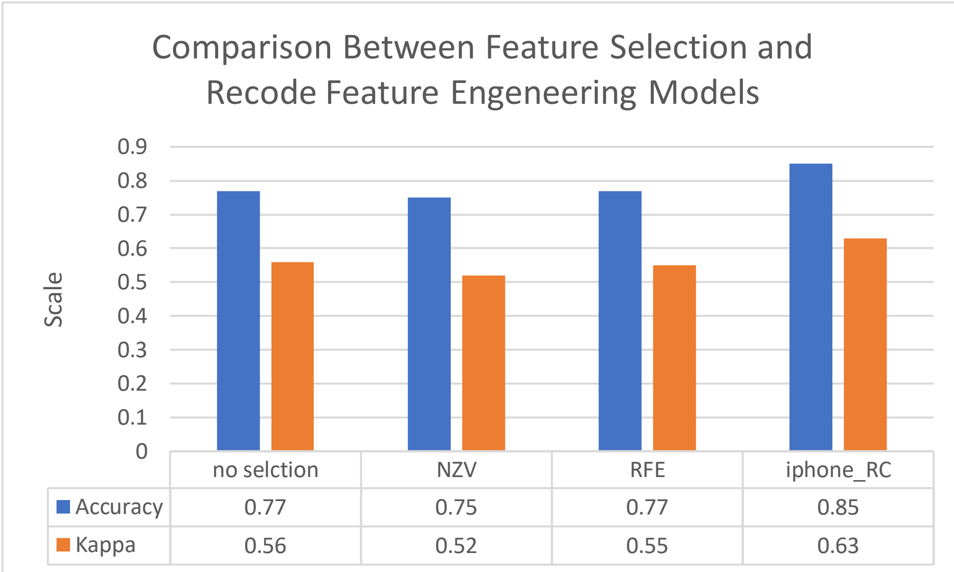
As we can see from the Chart both the no feature selection and the RFE feature selection approaches are compatible and out performed the nonzerovariance approach. However, the accuracy and Kappa values are still not optimal and improvement is needed before we can confidently apply these models to our data set.

Model performance improvement by Performing Feature Engineering.

To improve the C5.0 model we change the number of levels of the dependent variable. From 6 to 4. Thus, level (0) very negative will be combined with level(1) negative and level (6) very positive will be combined with level (5) positive in a new data frame and the model will be tested to see if accuracy and kappa have improved.

```
# feature engineering to improve C5.0 model performance
# combine levels 0 and 1 into 1, and 5 and 6 into 5 using dplyr's recode function
#create a new dataset that will be used for recoding sentiment
iphone_Recode <- iphone_smallmatrix
# recode sentiment to combine factor levels 0 & 1 and 4 & 5
iphone_Recode$iphonesentiment <- recode(iphone_Recode$iphonesentiment, '0' = 1, '1' = 1, '2' = 2, '3'
= 3, '4' = 4, '5' = 4)
# inspect results
summary(iphone_Recode)
str(iphone_Recode)
# make iphonesentiment a factor
iphone_Recode$iphonesentiment <- as.factor(iphone_Recode$iphonesentiment)

# create the C5.0 model on the new data set iphone_Recode

# create data partition and training and testing sets
set.seed(123)
iphone_RC_inTrain <- createDataPartition( y = iphone_Recode$iphonesentiment,p=0.70, list =  FALSE)
str(iphone_RC_inTrain)
iphone_RC_training <- iphone_Recode [iphone_RC_inTrain,]
iphone_RC_testing <- iphone_Recode [-iphone_RC_inTrain,]
nrow(iphone_RC_training)
nrow(iphone_RC_testing)

#create fit control
iphone_RC_fitControl <- trainControl(method= "repeatedcv", number=10,repeats = 1)

## create C5.0 model for classification iphone_smallmatrix data frame
library(C50)
iphone_RC_C5.0model<- train (iphonesentiment~.,data = iphone_RC_training, method = "C5.0",
trControl=iphone_RC_fitControl)
iphone_RC_C5.0model
varImp(iphone_RC_C5.0model)
# predicting from iphone testing
iphone_RC_C5.0_predictions <- predict(iphone_RC_C5.0model, iphone_RC_testing)

# obtain accuracy and kappa metrics for iphone_RC_C5.0 model to determine perfomance
postResample(iphone_RC_C5.0_predictions, iphone_RC_testing$iphonesentiment)
```

Model Name	Accuracy	Kappa
Iphone_RC_5.0model	0.85	0.63



The chart show that application of fearture engeneering by using dplyr's recode function improve the C5.0 model performance for our data. Here Accuracy increased to 0.85 and kappa increased to 0.63. This values represent more accurate model that can be used for predicting iphone sentiment.

Galaxy Smart Phone:

The data for the second smart phone to consider (galaxy smart phone) is on a separate data set that was also hand classified for sentiment. Thus, this second data set was considered with the same feature selection, preprocessing, model testing, and feature engineering to optimize the selected model as the iPhone data set. The result is that the C5.0 model, with the feature engineering reducing number of levels for the dependent variable, also gave the best result. This report shows the modeling steps for only the optimal model. See chart below for performance comparison for the between the iPhone and galaxy data set models.

```
# upload galaxy small matrix data set
galaxy_smallmatrix <- read.csv("galaxy_smallmatrix_labeled_9d.csv")


# apply feature engeneering with dplyr's recode method
# create new data set that is to be used for recoding the levels of sentiment
galaxyRecode <- galaxy_smallmatrix
#recode the galaxysentiment attribute levels 0&1 and 4&5
galaxyRecode$galaxysentiment <- recode(galaxyRecode$galaxysentiment, '0' = 1, '1' = 1, '2' = 2, '3' = 3,
'4' = 4, '5' = 4)
# inspect results
summary(galaxyRecode)
str(galaxyRecode)
# make galaxysentiment a factor
galaxyRecode$galaxysentiment <- as.factor(galaxyRecode$galaxysentiment)
```

 # prepare to train C5.0 model on galaxyRecode data set

 # create training and testing data sets from galaxyrecode data set
galaxyRC_inTrain <- createDataPartition(y = galaxyRecode$galaxysentiment,p=0.7, list = FALSE)
galaxyRC_training <- galaxyRecode[galaxyRC_inTrain,]
galaxyRC_testing <- galaxyRecode [-galaxyRC_inTrain,]
nrow(galaxyRC_training)
nrow(galaxyRC_testing)

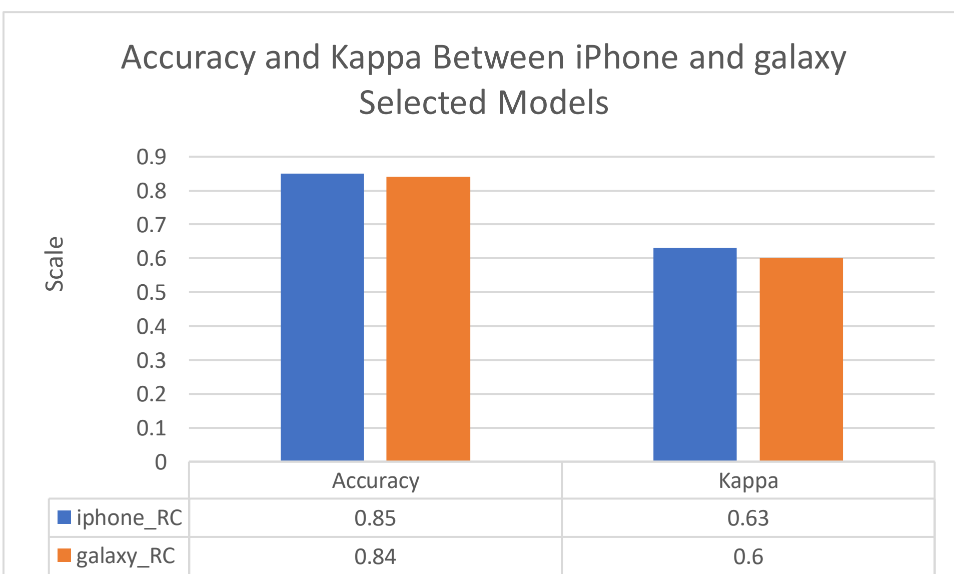
#create fit control
galaxyRC_fitControl <- trainControl(method= "repeatedcv", number=10,repeats = 1)

 # apply and train the C5.0 model to the galaxyRecode data set

create C5.0 model for classification galxyRecode data frame
library(C50)
galaxyRC_C5.0model<- train (galaxysentiment~.,data = galaxyRC_training, method = "C5.0", trControl=galaxyRC_fitControl)
galaxyRC_C5.0model
varImp(galaxyRC_C5.0model)
 # predicting from iphone testing
galaxyRC_C5.0_predictions <- predict(galaxyRC_C5.0model, galaxyRC_testing)

 # obtain accuracy and kappa metrics for galaxyRC_C5.0 model to determine perfomance
postResample(galaxyRC_C5.0_predictions, galaxyRC_testing$galaxysentiment)

Model Name	Accuracy	Kappa
galaxy_RC_5.0model	0.84	0.60



Obtaining other model performance metrics for selected models
 # obtain a confursion matrix and model statistics using the confusionmatrix function

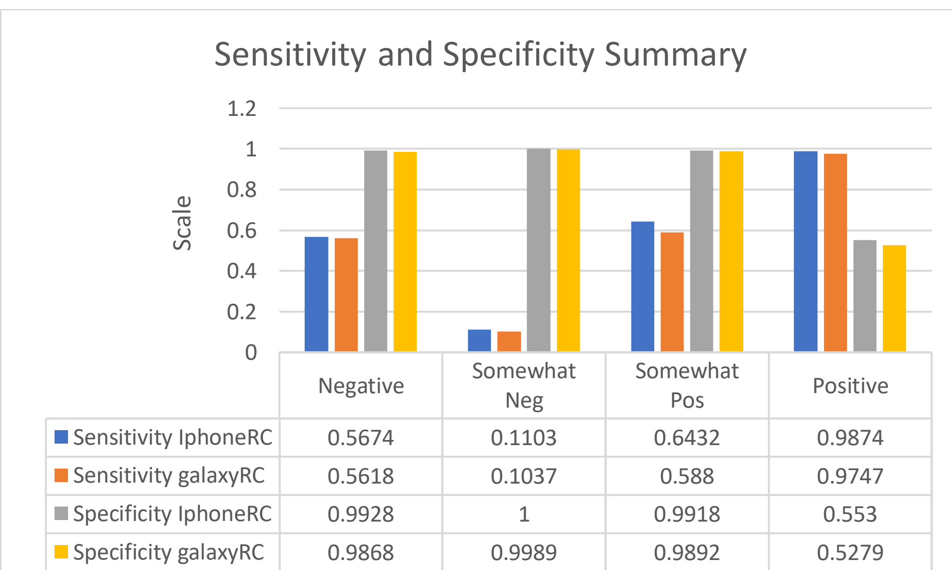
 # iphone_RC_C5.0 _model
matrix_iphone_RC_C5.0 <- confusionMatrix(iphone_RC_C5.0_predictions, iphone_RC_testing$iphonesentiment)
matrix_iphone_RC_C5.0

 # galaxyRC_C5.0_model
matrix_galaxyRC_C5.0 <- confusionMatrix(galaxyRC_C5.0_predictions, galaxyRC_testing$galaxysentiment)
matrix_galaxyRC_C5.0

Summary of statistics for the models that will be used to classify the sentiment for the two selected smart phones

Classification model name	**Accuracy**	**Kappa**	**95% CI**	**P value**	**Balance Accuracy**	
					Neg. cat.	**Pos. cat.**
iPhone_RC_C5.0_model	0.85	0.63	0.8375, 0.8602	<2.2e-16	0.7801	0.7702
galaxyRC_C5.0_modle	0.84	0.6	0.8309, 0.8541	<2.2e-16	0.7843	0.7513

Summary of sensitivity and specificity statistics for the models that will be used to classify sentiment for the two selected smart phones.



Both Sensitivity and Specificity are important parameters in machine learning algorithms. Sensitivity describes the proportion of actual correct positive predictions while specificity describes the proportion of actual correct negative predictions.

Work with Large Data Matrix Collected via Amazon EMR to Make Prediction on iPhone and galaxy Sentiment

IphoneLarge Matrix

#upload iphone LargeMatrix
iphone_LargeMatrix <- read.csv("iphone_LargeMatrix.csv.csv")

 # Data at a glance
summary(iphone_LargeMatrix)
str(iphone_LargeMatrix)
head(iphone_LargeMatrix)
names(iphone_LargeMatrix)

```
[1] "id"              "iphone"         "samsunggalaxy" "sonyxperia"
 [5] "nokialumina"    "htcphone"       "ios"            "googleandroid"
 [9] "iphonecampos"   "samsungcampos" "sonycampos"     "nokiacampos"
[13] "htccampos"      "iphonecamneg"   "samsungcamneg" "sonycamneg"
[17] "nokiacamneg"    "htccamneg"      "iphonecamunc"   "samsungcamunc"
[21] "sonycamunc"     "nokiacamunc"    "htccamunc"      "iphonedispos"
[25] "samsungdispos" "sonydispos"     "nokiadispos"    "htcdispos"
[29] "iphonedisneg"   "samsungdisneg" "sonydisneg"     "nokiadisneg"
[33] "htcdisneg"      "iphonedisunc"   "samsungdisunc" "sonydisunc"
[37] "nokiadisunc"    "htcdisunc"      "iphoneperpos"   "samsungperpos"
[41] "sonyperpos"     "nokiaperpos"    "htcperpos"      "iphoneperneg"
[45] "samsungperneg" "sonyperneg"     "nokiaperneg"    "htcperneg"
[49] "iphoneperunc"   "samsungperunc" "sonyperunc"     "nokiaperunc"
[53] "htcperunc"      "iosperpos"      "googleperpos"   "iosperneg"
[57] "googleperneg"   "iosperunc"      "googleperunc"
```

From the list of attribute names we can see that the first name “id” is new and is not consistent with the small matrix names.

 # remove “id” attribute from data frame
iphone_LargeMatrix$id <- NULL
 # convert iphone sentiment to a factor
iphone_LargeMatrix$iphonesentiment <- as.factor(iphone_LargeMatrix$iphonesentiment)
str(iphone_LargeMatrix)

 # Make predictions about iphone sentiment in the iphone_LargeMatrix
iphone_LM_C5.0_predictions <- predict(iphone_RC_C5.0model, iphone_LargeMatrix)
summary(iphone_LM_C5.0_predictions)

```
   1    2    3    4
7754  800 1297 7600
```

Work with large data matrix to make predictions regarding galaxysentiment

Galaxy_LargeMatrix

#upload galaxy_LargeMatrix
galaxy_LargeMatrix<- read.csv("galaxy_LargeMatrix.csv.csv")

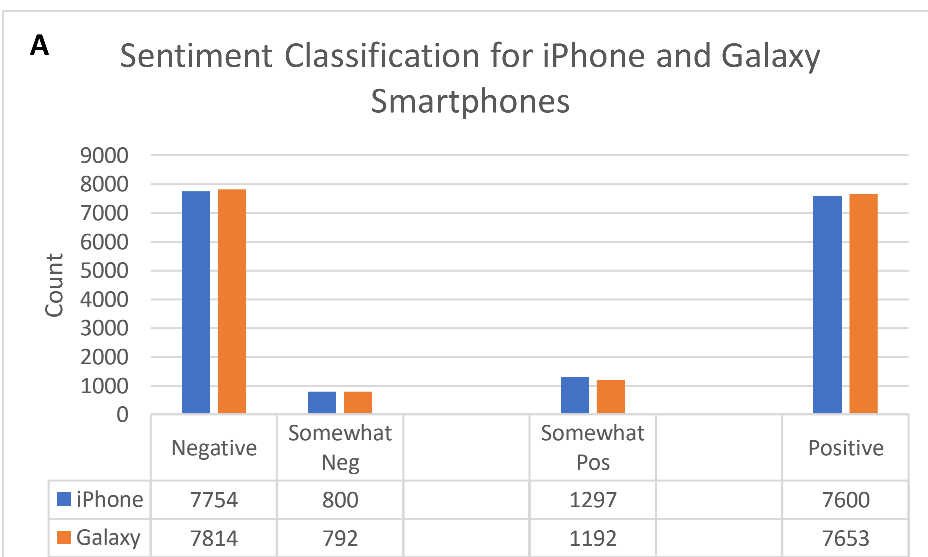
#Data at a glance
summary(galaxy_LargeMatrix)
str(galaxy_LargeMatrix)
names(galaxy_LargeMatrix)

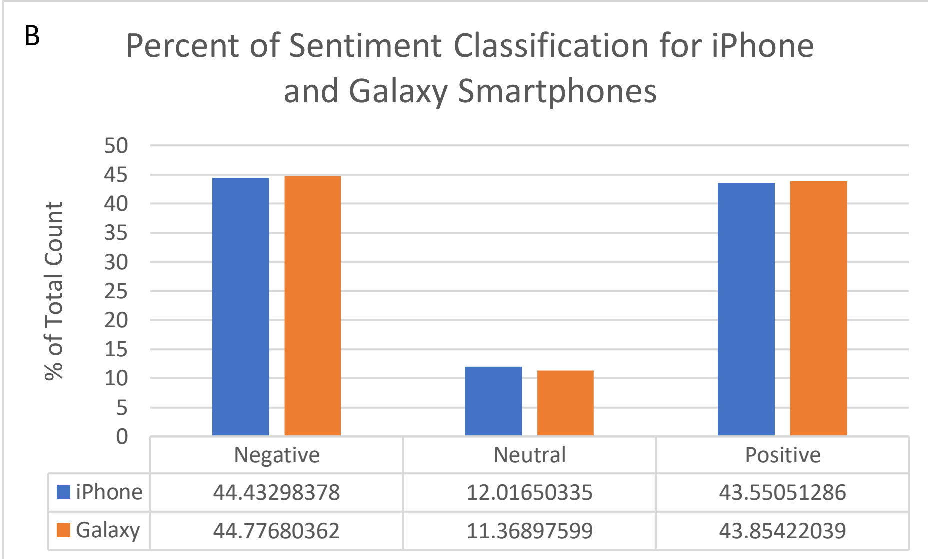
#remove the "id" feature from the data set
galaxy_LargeMatrix$id <- NULL

#convert galaxysentiment to factor
galaxy_LargeMatrix$galaxysentiment <- as.factor(galaxy_LargeMatrix$galaxysentiment)
str(galaxy_LargeMatrix)

 # Make predictions regarding galaxy sentiment on the galaxy Large Matrix
galaxy_LM_C5.0_predictions <- predict(galaxyRC_C5.0model, galaxy_LargeMatrix)
summary(galaxy_LM_C5.0_predictions)

```
   1    2    3    4 
7814  792 1192 7653 
```





The charts show a summary of the predicted classification of the iPhone and Galaxy smart phones listed in the large data matrix collected via Amazons EMR. Predictions were made based on the best fit machine learning algorithm, feature selection, and feature engineering approach. The large data matrix consists of over 17400 observations and 59 features. A. Show the actual count of each category. B. shows the percent of total count sentiment for both iphone and galaxy handsets with negative = 44% and 45 %, Neutral (somewhat neg & somewhat pos combined) = 12% and 11% and positive = 44% and 44% respectively.

In conclusion: it is clear that the sentiment for both iPhone and galaxy handsets are extremely similar. In making a choice for the medical app. One should consider the cost of the unit along with the availability of the unit in the countries of intended use.