**Text Classification using Naïve Bayesian Technique**

Naïve Bayesian works on Bayes theorem of probability to predict the class of unknown data set. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature

#Setting the directory location

setwd("C:\\ISQS6337\\L01")

# Reading the data and assigning it to a variable “Edata”. Here, I have converted the .xlsx file to .csv

Edata<-read.csv("EnggE.csv")

#View the data

View(Edata)

#Trying to know more about the dataset

str(Edata)

# The Edata type variable is a character vector. Since this is categorical, it would be better to convert it into factor variable

Edata$Type <- factor(Edata$Type)

str(Edata$Type)

table(Edata$Type)

prop.table(table(Edata$Type))

# Edata messages are strings of text composed of words, spaces, numbers and punctuations We have to remove numbers, punctuations and handle uninteresting words like and, but and or; and break apart sentences into individual words. tm package can help in this

library(tm)

# The first step in processing text data involved creating a corpus, which is a collection of text documents In order to create a corpous we will use VCorpus() function in the tm package. VCorpus() refers to volatile as it is stored in memory as opposed to being stored in the disk. PCorpus() function can be used to access permanent corpus stored in the database.

Edata\_corpus <- VCorpus(VectorSource(Edata$Text))

# By printing the corpus, we see that it contains documents for each of the Edata messages in the training data

print(Edata\_corpus)

# To view the actual message text, you can use the as.character() function

as.character(Edata\_corpus[[1]])

as.character(Edata\_corpus[[4]])

# To view multiple documents, we need to use as.character() on several items in the corpous. To do so, we will use lapply() function

lapply(Edata\_corpus[1:5], as.character)

# The corpus contains text messages. In order to perform our analysis, we will need to divide these messages into individual words. But first we need to clean the texts, standardize the words, by removing punctuations & other unwanted characters. The tm\_map() function provides a method to apply a transformation to the tm corpus. We start with standardizing the messages to use only lower case

Edata\_corpus\_clean <- tm\_map(Edata\_corpus, content\_transformer(tolower))

# Check if the tolower() function worked

as.character(Edata\_corpus[[4]])

as.character(Edata\_corpus\_clean[[4]])

# lets remove the numbers from the Edata messages

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, removeNumbers)

as.character(Edata\_corpus[[4]])

as.character(Edata\_corpus\_clean[[4]])

# Our next task is to remove filler words such as to, and, but, and, or from our Dataset. These terms are known as STOP WORDS and are typically removed prior to text mining. We use the stopwords() function.

stopwords()

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, removeWords, stopwords())

mystopwords <- c("canât","Itâs")

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, removeWords, mystopwords)

as.character(Edata\_corpus[[4]])

as.character(Edata\_corpus\_clean[[4]])

# Next step to remove the punctuations

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, removePunctuation)

as.character(Edata\_corpus[[4]])

as.character(Edata\_corpus\_clean[[4]])

# Another common standardization for text data involves reducing words to their root form in a process called STEMMING. For example words like learned, learning, learnt get transformed to learn. The tm package provides STEMMING functionality via integration with SnowballC package

library("SnowballC")

# See how it works

wordStem(c("learning", "learn", "learns", "learned"))

wordStem(c("work", "working", "worked"))

# but for words like below won’t change

wordStem(c("go", "going", "gone", "went"))

# Now applying STEMMING to our courpus

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, stemDocument)

# After removing numbers, stop words, punctuations and performing stemming, now we remove whitespaces

Edata\_corpus\_clean <- tm\_map(Edata\_corpus\_clean, stripWhitespace)

# Splitting text documents into words .Now that data is processed, the final step is to split the messages into individual components through a process called TOKENIZATION. A token is a single element of a text string

# The DocumentTermMatrix() function in the tm package will take a corpus and create a data structure called Document Term Matrix (DTM) in which rows indicate documents (Edata messages) and columns indicate words .Creating a DTM sparse matrix, given a tm corpus:Next we create a Term Document Matrix (TDM) which reflects the number of times each word in the corpus is found in each of the documents.

Edata\_dtm <- DocumentTermMatrix(Edata\_corpus\_clean)

str(Edata\_dtm)

#checking the frequency of the words which occurred more than 20 times

findFreqTerms(Edata\_dtm, 20)

# DATA PREPARATION - we will split into train & test data

Edata\_dtm

Edata\_dtm\_train <- Edata\_dtm[1:817,]

Edata\_dtm\_test <- Edata\_dtm[818:1729,]

# To confirm that the subsets are representative of the complete set of Edata

Edata\_train\_labels <- Edata[1:817,]$Type

Edata\_test\_labels <- Edata[818:1729,]$Type

prop.table(table(Edata\_train\_labels))

prop.table(table(Edata\_test\_labels))

# The final step is to transform the sparse matrix into a data structure that Naive Bayesian Classifier can consume

# Currently the sparse matrix includes over many features - there is a feature for every word. Its unlikely that all of these are useful for classification. So we remove those features/words which appear in less than 5 messages We can find the frequent words using findFreqTerms() function in the tm package

Edata\_frequent\_words <- findFreqTerms(Edata\_dtm\_train, 5)

str(Edata\_frequent\_words)

# Filter out the non frequent words from the training & test dataset

Edata\_dtm\_freq\_train <- Edata\_dtm\_train[, Edata\_frequent\_words]

Edata\_dtm\_freq\_test <- Edata\_dtm\_test[, Edata\_frequent\_words]

# Finally - because Naive Bayesian is typically trained on data with CATEGORICAL FEATURES The sparse matrix poses a problem because the cells in the sparse matrix are numeric and measure the number of times a word appears in a message. We need to change this to a categorical variable that simply indicates yes/no depending on whether the word appears at all

# We write a function to convert counts to Yes/No strings

convert\_counts <- function(x){

x <- ifelse(x>0, "Yes", "No")

}

# We need to apply convert\_counts() to each of the coulums in the sparse matrix. The apply() function allows a function to be used on each of the rows & columns in a matrix. It uses MARGIN parameter to specify either rows or columns. MARGIN=1 means rows, MARGIN=2 means columns We are interested in columns

Edata\_train <- apply(Edata\_dtm\_freq\_train, MARGIN=2, convert\_counts)

Edata\_train

Edata\_test <- apply(Edata\_dtm\_freq\_test, MARGIN=2, convert\_counts)

Edata\_test

# Result will be 2 character type matrixes, each with cells indicating "Yes" or "No" for whether the word represented by the column appears at any point in the message represented by the row

# TRAINING THE MODEL

library("e1071")

# To build our model on the Edata\_train:

Edata\_classifier <- naiveBayes(Edata\_train, Edata\_train\_labels)

# EVALUATING MODEL PERFORMANCE

# To evaluate the Edata classifier, we need to test its prediction on unseen messages

# in the test data

# Recall that unseen message features are in a matrix named Edata\_test

# While the class labels (Ethics or Not Ethics) are stored in a vector names Edata\_test\_labels

# Model we built for classification is named Edata\_classifier

# We use predict() function to make predictions

Edata\_test\_pred <- predict(Edata\_classifier, Edata\_test)

# To compare the predictions to the true values, we will use CrossTable() function in the

# gmodels package

library("gmodels")

CrossTable(Edata\_test\_pred, Edata\_test\_labels, prop.chisq = FALSE,

prop.r = FALSE, prop.c=FALSE, dnn=c('predicted', 'actual'))

# IMPROVING MODEL PERFORMANCE

# Read LAPLACE ESTIMATOR

# We will rebuild the model with LAPLACE estimator=1

#consider the worst case where none of the words in the training sample appear in the test

#sentence. In this case, under your model we would

#conclude that the sentence is impossible but it clearly exists creating a contradiction.

#Another extreme example is the test sentence "Alex met Steve." where "met" appears several

#times in the training sample but "Alex" and "Steve" don't. Your model would conclude this

#statement is very likely which is not true

Edata\_classifier\_laplace <- naiveBayes(Edata\_train, Edata\_train\_labels, laplace=1)

system.time(Edata\_test\_pred\_laplace <- predict(Edata\_classifier\_laplace, Edata\_test))

#using this we improved the outcome of 4 messages.

CrossTable(Edata\_test\_pred\_laplace, Edata\_test\_labels, prop.chisq = FALSE,

prop.r=FALSE, prop.c=FALSE, dnn=c('predicted', 'actual'))