## Challenge Closer



Thank you for making the time to come to interview with the Data Science team at Closer. We hope you find the process both challenging and rewarding.

The challenge is designed to allow you to show your technical capabilities. We would like you to demonstrate that you can frame and solve real-world problems involving hidden patterns and lots of data with a clear understanding of the statistical, machine learning, software development and big data issues arising.

The challenge comprises an NLP project.

You will receive detailed information of how to get the necessary data for each problem in a separate email.

As deliverables, we expect you to submit the code you used to solve the problems. You can use any open source language in which you feel comfortable with, but we strongly encourage the usage of Python. Besides the submission of the code, we expect that you take us through your solution and recommendations (mentioning both achievements and difficulties) in a small discussion (we discourage PowerPoint slides usage, but you may of course bring written notes). You can expect that, during your presentation, we will ask you questions about your approach. Additionally, we may ask you to reconsider your approach given updated information or alternative scenarios. We will also assess the data structures that you used in both exercises.

You should assume that we are familiar with the cases material.

Some advice:

* The problem is not trivial: if your answer is trivial, you may have misunderstood the problem.
* Make sure your answer demonstrates clear and structured thinking. How you approach the problem is just as important as the answers you get.
* If you have any questions, please email them in advance.

### Exercise 2

ACME, the very same company of Exercise 1, created a unit of root-cause analysts that verify and study each situation and identify the root-cause of each complaint. Moreover, similarly to the situation experienced in Exercise 1, its analysts are taking too long to analyse and identify the cause of each situation. Therefore, ACME contacted for you to build a support tool that will help its analysts in making a more efficient job.

For this exercise, you are provided another dataset: **complaints\_data.csv**. Its fields are self-explanatory and you shall understand their meaning easily.

In the mentioned dataset, you are provided with the corpus of the client’s complaint, and the issue (and sub-issue) that the analysts already classified.

The goal that ACME wants to achieve is to have a tool that, given a non-classified complaint, it provides the following:

* Possible root-cause of the complaint
* Similar complaints

Additionally, ACME also wants a second opinion in the issue (and sub-issue) classification for the already classified complaints.

You should consider that there are two root-causes for each complaint: an apparent one and a real one. Therefore, the real root-cause analysis might be hidden in the corpus of each complaint. If you think necessary, you can use additional datasets to enrich the data.

**Hint:** tackle different dimensionality reduction techniques to achieve better results.

SVM, Naïve Bayes equations, Accuracy v precision v recall, sensitivity v specificity, AUC, PCA, Neural Network algorithm, Python libraries, SMOTE

Challenges: learning the tools, lack of text classification in R, custom stopwods: need industry understanding, need industry understanding: to clean issue column, steep learning curve, accuracy vs time/processing power

#Initialise

#TM package require for text manipulation and generating Document Term Matrix

require(tm)

#dplyr for manipulating data frames

require(dplyr)

#caret for partitioning data into test and train sets

require(caret)

#e1071 for using SVM and Naivebaye's algorithms used for supervised learning

require(e1071)

#topicmodels contains LDA algorithm for unsupervised learning

require(topicmodels)

#Used to extract data from LDA model

require(tidytext)

#Set seed manually

set.seed(475)

#-----------------

#Read data

#-----------------

#Extract the aforementioned columns and create a new data frame

corpus<-data.frame(complaint=data$Consumer.complaint.narrative, issue\_tv=data$Issue)

#Coerce columns into characers

corpus$complaint<-as.character(corpus$complaint)

corpus$issue\_tv<-as.character(corpus$issue\_tv)

#-----------------

#Balance data

#-----------------

#Count complainst by issue

countbyissue<-aggregate(corpus$issue\_tv, by=list(corpus$issue\_tv), length)

#Give data frame meaniful names

colnames(countbyissue)<-c("issue","count")

#Order data in descending order

countbyissue<-countbyissue[order(-countbyissue$count),]

#Create a bar plot

barplot(countbyissue$count, main = "Count of Each Issue", col="blue")

sum(countbyissue$count[1:100])/sum(countbyissue$count)

countbyissue$count[100]

#Create a lookup table for issues are below the 100th issue

lookup<-as.character(countbyissue$issue[101:161])

#Loop through those issues and replace with "other" as issue

for(i in 1: nrow(corpus)){

if(corpus$issue\_tv[i] %in% lookup){

corpus$issue\_tv[i]<-"other"

}

}

#Sample 151 complaints from each issue

corpus <- corpus %>% group\_by(issue\_tv) %>% sample\_n(151)

#Convertfrom dplyr to data frame

corpus <- ungroup(corpus) %>% as.data.frame()

#Count complainst by issue

countbyissue<-aggregate(corpus$issue\_tv, by=list(corpus$issue\_tv), length)

#Give data frame meaniful names

colnames(countbyissue)<-c("issue","count")

#Order data in descending order

countbyissue<-countbyissue[order(-countbyissue$count),]

#Create a bar plot

barplot(countbyissue$count, main = "Count of Each Issue(after balancing)", col="blue")

#-----------------

#Clean text: stopwords, tolower, punctuation, XXXX, \n, stemming

#-----------------

#Convert letters to lower case

corpus$complaint<-tolower(corpus$complaint)

#Remove Numbers

corpus$complaint<-removeNumbers(corpus$complaint)

#Remove Punctuation

corpus$complaint<-removePunctuation(corpus$complaint)

#Create a function that will take a pattern and remove it

findremove<- function (pattern, object) { gsub(pattern, "", object)}

#Remove xxxx

corpus$complaint<-findremove("xxxx", corpus$complaint)

#Remove xx

corpus$complaint<-findremove("xx", corpus$complaint)

#Remove \n

corpus$complaint<-findremove("\n", corpus$complaint)

#Remove \t

corpus$complaint<-findremove("\t", corpus$complaint)

#Strip extra white spaces

corpus$complaint<-stripWhitespace(corpus$complaint)

#Convert complaints into a Corpus using tm package

complaintCorpus<-Corpus(VectorSource(corpus$complaint))

#Remove Stopwords

complaintCorpus<-tm\_map(complaintCorpus, removeWords, stopwords())

#Stemming words

#corpus$complaint<-stemDocument(corpus$complaint)

#-----------------

#Feature creation: DTM

#-----------------

#Create DTM using TM

dtm<-DocumentTermMatrix(complaintCorpus)

#View the DTM

inspect(dtm)

#Sum each term across all documents (i.e column sum)

cumulativeAllTerms<-colSums(as.matrix(dtm))

#Sort in descending order

cumulativeAllTerms<-cumulativeAllTerms[order(-cumulativeAllTerms)]

#Show top 100 terms

head(cumulativeAllTerms, 100)

#Create my list of stopwords

otherstopwords<-c("told","called","back","can","will","get","said","never","also",

"even","just","know","another","like","want","went","please","take",

"however","going","see","got","several","able")

#Remove custom stopwords

complaintCorpus<-tm\_map(complaintCorpus, removeWords, otherstopwords)

#Create a new DTM

dtm<-DocumentTermMatrix(complaintCorpus)

#Find terms that appear 10 times or more

freqterms<-findFreqTerms(dtm, lowfreq = 10)

#Limit DTM to contain terms that appear >= 10

dtm<-DocumentTermMatrix(complaintCorpus, list(dictionary=freqterms))

inspect(dtm)

#Sum count of each term across all documents

cumulativeAllTerms<-colSums(as.matrix(dtm))

#Sort in descending order and take top 30 terms

Top30<-head(cumulativeAllTerms[order(-cumulativeAllTerms)], 30)

#Convert to data frame

Top30<-data.frame(term=names(Top30), count=Top30)

Top30<-Top30[order(-Top30$count),]

#Plot

barplot(rev(Top30$count), horiz = T, names.arg = Top30$term, las=2, col="blue", main="Most Frequent 30 Terms")

#-----------------

#Create train and test sets

#-----------------

#Convert issue/target variable to factor, in order to conserve levels in case some categories don't appear in one of the set (highly unlikely since data is balanced)

corpus$issue\_tv<-as.factor(corpus$issue\_tv)

#Create an index with 75% split based on issue value in raw data

inTrain<-createDataPartition(corpus$issue\_tv,p=0.75,list=FALSE)

#Subset raw data with index

train<-corpus[inTrain,]

#Subset raw data with NOT index

test<-corpus[-inTrain,]

#Subset cleaned corpus for training & test sets

corpustrain<-complaintCorpus[inTrain]

corpustest<-complaintCorpus[-inTrain]

#Create DTM based on subsetted cleaned corpus

dtmtrain<-DocumentTermMatrix(corpustrain, list(dictionary=freqterms))

dtmtest<-DocumentTermMatrix(corpustest, list(dictionary=freqterms))

#Function to convert non-zero values to 1

convert\_counts <- function(x) {

x <- ifelse(x > 0, 1, 0)

}

#Convert non-zero values to 1 in train and test DRM

dtmtrain<- dtmtrain %>% apply(MARGIN=2,convert\_counts)

dtmtest<- dtmtest %>% apply(MARGIN=2,convert\_counts)

#Convert DTM to data frames

dtmtrain<-as.data.frame(dtmtrain)

dtmtest<-as.data.frame(dtmtest)

#Bind target variable to test and train DTMs

dtmtrain<-cbind(issue\_tv=train$issue\_tv,dtmtrain)

dtmtest<-cbind(issue\_tv=test$issue\_tv,dtmtest)

#-----------------

#Training Models - Supervised

#-----------------

#Train a model based on Naive Bayes using e1017 package

fit\_NB<-naiveBayes(dtmtrain, dtmtrain$issue\_tv)

#Predict using Naive Bayes model using the test set

pred\_NB<-predict(fit\_NB, newdata= dtmtest)

#Create a confusion matrix for that model/prediction

conf\_NB<-confusionMatrix(pred\_NB,dtmtest$issue\_tv)

#Extract accuracy

conf\_NB$overall["Accuracy"]

#Train a model based on SVM in the e1017 package

fit\_SVM<-svm(issue\_tv ~ ., data = dtmtrain, scale=FALSE)

#Predict using Naive Bayes model using the test set

pred\_SVM<-predict(fit\_SVM, newdata= dtmtest)

#Create a confusion matrix for that model/prediction

conf\_SVM<-confusionMatrix(pred\_SVM,dtmtest$issue\_tv)

#Extract Accuracy

conf\_SVM$overall["Accuracy"]

#-----------------

#Training Models - unsupervised

#-----------------

#10 topics/categories

k<-10

#Run LDA algorithm

lda<-LDA(dtm, k=10, method = "GIBBS")

#-----------------

#The below code is borrowed from this site: https://www.datacamp.com/community/tutorials/ML-NLP-lyric-analysis

#-------------------

theme\_lyrics <- function(aticks = element\_blank(),

pgminor = element\_blank(),

lt = element\_blank(),

lp = "none")

{

theme(plot.title = element\_text(hjust = 0.5), #center the title

axis.ticks = aticks, #set axis ticks to on or off

panel.grid.minor = pgminor, #turn on or off the minor grid lines

legend.title = lt, #turn on or off the legend title

legend.position = lp) #turn on or off the legend

}

word\_chart <- function(data, input, title) {

data %>%

#set y = 1 to just plot one variable and use word as the label

ggplot(aes(as.factor(row), 1, label = input, fill = factor(topic) )) +

#you want the words, not the points

geom\_point(color = "transparent") +

#make sure the labels don't overlap

geom\_label\_repel(nudge\_x = .2,

direction = "y",

box.padding = 0.1,

segment.color = "transparent",

size = 3) +

facet\_grid(~topic) +

theme\_lyrics() +

theme(axis.text.y = element\_blank(), axis.text.x = element\_blank(),

#axis.title.x = element\_text(size = 9),

panel.grid = element\_blank(), panel.background = element\_blank(),

panel.border = element\_rect("lightgray", fill = NA),

strip.text.x = element\_text(size = 9)) +

labs(x = NULL, y = NULL, title = title) +

#xlab(NULL) + ylab(NULL) +

#ggtitle(title) +

coord\_flip()

}

num\_words <- 10 #number of words to visualize

#create function that accepts the lda model and num word to display

top\_terms\_per\_topic <- function(lda\_model, num\_words) {

#tidy LDA object to get word, topic, and probability (beta)

topics\_tidy <- tidy(lda\_model, matrix = "beta")

top\_terms <- topics\_tidy %>%

group\_by(topic) %>%

arrange(topic, desc(beta)) %>%

#get the top num\_words PER topic

slice(seq\_len(num\_words)) %>%

arrange(topic, beta) %>%

#row is required for the word\_chart() function

mutate(row = row\_number()) %>%

ungroup() %>%

#add the word Topic to the topic labels

mutate(topic = paste("Topic", topic, sep = " "))

#create a title to pass to word\_chart

title <- paste("LDA Top Terms for", k, "Topics")

#call the word\_chart function you built in prep work

word\_chart(top\_terms, top\_terms$term, title)

}

#call the function you just built!

top\_terms\_per\_topic(lda, num\_words)