Project_final

First read in all text and clean and translate/map to a common repersentation. I don't know the mapping of the arabic words to the transliterated ones but the library is able to shift back and forth.This block reads all text in, removes the stop words, and stems and transliterates the text to latin characters and then pputs it back into the data frame

I first ran this analysis in python which has diffrent cleaning and stemming libraries, so if the report differs from the results in some ways that may be the root. For functional purpouses it should be the same.

```
library(knitr)
master = read.csv('data.csv', stringsAsFactors = FALSE)
master = as.data.frame(master)
typeof(master)
```

[1] "list"

```
library(arabicStemR)
```

```
##
## Attaching package: 'arabicStemR'
```

```
## The following object is masked from 'package:graphics':
##
##     stem
```

```
wrap = function(chr){
  return(as.character(removeStopWords(chr)[1]))
}
master$transliterated = removePunctuation(cleanChars(transliterate(sapply(master$Text, wrap))))
kable(master[1:5,])
```

X	Text	Positive.	transliterated
0	أبذل الجهد و العرق و توكل على الله و اطمئن فرزقك مضمون	1	abil aljhd al3rQ twkl allh aTm5n frzQk mDmwn

X	Text	Positive.	transliterated
1	احُبك ربي	1	a7bk rby
2	اللهم عفوك و رضاك و الجنه	1	allhm 3fwk rDak aljnh
3	الماء هى الحياة	1	almaq hA al7ya0
4	بالفعل لازم كل واحد بيتكلم عن الدين اقل مافيها يلتزم باقل مبادئه عشان يكون قدوة مش بيخلي الناس تقول هو ده الدين	1	balf3l lazm kl wa7d bytklm aldyn mafyha yltzm baQl mbad5h 3Wan ykwn Qdw0 mW yyKly alnas tQwl dh aldyn
I fir	st ran this analysis in python which has diffrent cleaning and stemming libraries, so if the report differs from t	he results i	n some ways that may be the root. For functional purpouses it should be the same.

Spliting data in to train and test sets, building word frequency tables for the positive and negitive words and then building training and testing data to feed into naive bayes.

```
library(keras)
library(stringr)

word_counts <- as.data.frame(table(unlist( strsplit(master$transliterated, "\ ") )))
word_counts$id = seq(1,nrow(word_counts))

get_int_rep = function(string){
  int_rep = c()
  for (x in unlist(strsplit(string, ' '))){
    if ((x %in% unlist(word_counts['Var1']))){
      row = word_counts[word_counts['Var1'] == x,]
      int_rep = append(int_rep, row$id)
    }
  }
  if(length(int_rep) > 1){return(array(pad_sequences(list(int_rep), maxlen = 180)))}
  else{return(array(pad_sequences(list(0), maxlen = 180)))}
}

int_reps = c()
for (x in master$transliterated){
  int_reps = append(int_reps, get_int_rep(x))
}

t = array_reshape(int_reps, c(nrow(master), 180))
master$int_rep = t

smp_size <- floor(0.75 * nrow(master))
train_ind <- sample(seq_len(nrow(master)), size = smp_size)
train <- master[train_ind, ]
test <- master[-train_ind, ]

pos_words = train[train$Positive. == 1,]
neg_words = train[train$Positive. == 0,]

pos_word_tbl = as.data.frame(table(unlist( strsplit(pos_words$transliterated, "\ ") )))
neg_word_tbl = as.data.frame(table(unlist( strsplit(neg_words$transliterated, "\ ") )))

total_words = sum(word_counts$Freq)

word_counts$log_frequency = log(word_counts$Freq/total_words)
```

```
pos_word_tbl$log_frequency = log(pos_word_tbl$Freq/total_words)
neg_word_tbl$log_frequency = log(neg_word_tbl$Freq/total_words)

kable(word_counts[1:5,], caption = 'Word frequency Tables')
```

Word frequency Tables

Var1	Freq	id	log_frequency
0	1	1	-9.537123
1	3	2	-8.438511
10	3	3	-8.438511
١٠٠	2	4	-8.843976
100	4	5	-8.150829

```
kable(pos_word_tbl[1:5,], caption =  'Positive words and thier frequency')
```

Positive words and thier frequency

Var1	Freq	log_frequency
1	1	-9.537123
3	5	-7.927685
3afyt	1	-9.537123
3afytk	1	-9.537123
3almwasah	1	-9.537123

Defining slightly modifyed naive bayes function

```
naive_b = function(string){
  prob_p = 1
  prob_n = 1
  multiplier = .5
  for (x in unlist(strsplit(string, ' '))){
    if (x %in% unlist(pos_word_tbl['Var1'])){
      freq = pos_word_tbl[pos_word_tbl['Var1'] == x,]
      prob_p = prob_p + freq$log_frequency
    }
    if (x %in% unlist(neg_word_tbl['Var1'])){
      freq = neg_word_tbl[neg_word_tbl['Var1'] == x,]
      prob_n = prob_n + freq$log_frequency
    }
  }
  if (prob_n > prob_p){return(1)}
  else{return(0)}
}
```

Running the naive bayes on training and test data

```
train$guesses = sapply(as.list(train$transliterated), naive_b)
test$guesses = sapply(as.list(test$transliterated), naive_b)
```

Evaluation of results Note: The accuracy of this test is suspiciously good. I'm not sure if this is a flux or if the transliteration function is much more complex than my solution, but I used the accruacies from my original code in python in the report because I know the mapping function and am much more confident in the reproducability of my solution. If I had more time I would be able ot verify the library's transliteration function and these results but until this I'll only report on what I know to be accurate.

```
train_accruacy = sum(ifelse(train$Positive.==train$guesses,1,0)) / nrow(train)
test_accuracy = sum(ifelse(test$Positive.==test$guesses,1,0)) / nrow(test)
print('Train accuracy for modified naive bayes is :')
```

```
## [1] "Train accuracy for modified naive bayes is :"
```

```
train_accruacy
```

```
## [1] 0.9807692
```

```
print('Test accuracy for modified naive bayes is :')
```

```
## [1] "Test accuracy for modified naive bayes is :"
```

```
test_accuracy
```

```
## [1] 0.8031809
```

This chunk implements the RNN. I added a dropout layer to combat overfitting but it doesn't help as much as I would like. I belive the encoding is less consistent with the R version of this code because when I build a RNN with the same architechure, the python version outperformed by almost 20% and was far less volitile.

```
library(keras)
library(tensorflow)
#install_tensorflow(version =  '1.12') #recent version is broken...?
embedding_size = 32
vocab_size = nrow(word_counts)
max_words = 180

model = keras_model_sequential()
model %>%
  layer_embedding(vocab_size, 64, input_length = max_words, name = 'embedding') %>%
  bidirectional(layer_lstm(units =64, name = 'reccurent_layer')) %>%
  layer_dropout(.5) %>%
  layer_dense(1, activation = 'sigmoid', name = 'output')

model %>% compile(
  optimizer = 'adam',
  loss = 'binary_crossentropy',
  metrics = c('accuracy')
)
summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## embedding (Embedding)            (None, 180, 64)               426368
## ___________________________________________________________________________
## bidirectional (Bidirectional)    (None, 128)                   66048
## ___________________________________________________________________________
## dropout (Dropout)                (None, 128)                   0
## ___________________________________________________________________________
## output (Dense)                   (None, 1)                     129
## ===========================================================================
## Total params: 492,545
## Trainable params: 492,545
## Non-trainable params: 0
## ___________________________________________________________________________
```

```
X_valid = train$int_rep
y_val = train$Positive.
X_train2 = test$int_rep
y_train2 = test$Positive.

history <- model %>% fit(
  X_train2, y_train2,
  epochs = 35, batch_size =64,
  validation_split = 0.2
)
```

This chunk uses a hashmap to find a euclidian distance be occurance of individual words. Hashmap is used because its crazy fat compared to whatever R lists and columns are made out of. For more details and exploratory data analysis check the report and the python document.

```
#install.packages('hashmap')
library(hashmap)
word_c = hashmap(as.character(word_counts$Var1), word_counts$id )
find_location = function(string){
  map = hashmap(seq(1, nrow(word_counts)), numeric(nrow(word_counts)))
  for (x in unlist(strsplit(string, ' '))){
      x = as.character(x)
      if (word_c$has_key(x) == TRUE){
        id = word_c[[x]]
        map[[id]] = map[[id]] + 1
    }
  }
  return(c(map$values()))
}

data = c()
num = 1
for (x in master$transliterated){
  data = append(data, find_location(x))
}

data2 = array_reshape(data, dim =c(2011,length(find_location(master$transliterated[1]))))

kable(master[1:5,], caption = 'master with grouping variable')
```

```
## Warning in `[<-.data.frame`(`*tmp*`, , isn, value = structure(list(X =
## structure(c("0", : provided 182 variables to replace 3 variables
```

master with grouping variable

X	Text	Positive.	transliterated	int_rep
0	أبذل الجهد و العرق و توكل على الله و اطمئن فرزقك مضمون	1	abil aljhd al3rQ twkl allh aTm5n frzQk mDmwn	0
1	احُبك ربي	1	a7bk rby	0
2	اللهم عفوك و رضاك و الجنه	1	allhm 3fwk rDak aljnh	0
3	الماء هى الحياة	1	almaq hA al7ya0	0

X	Text	Positive.	transliterated	int_rep
4	بالفعل لازم كل واحد بيتكلم عن الدين اقل مافيها يلتزم باقل مبادئه عشان يكون قدوة مش بيخلي الناس تقول هو ده الدين	1	balf3l lazm kl wa7d bytklm aldyn mafyha yltzm baQl mbad5h 3Wan ykwn Qdw0 mW yyKly alnas tQwl dh aldyn	0

This chunk calculated the k-means groupings and appends it to the master dataframe which associates it with a tweet. I ran this many times and looked through the data manually but the code takes up a lot of spcae so this is just one itteration.

```
kmean = kmeans(data2, 5)
master$group_5 = kmean$cluster
kable(master[1:5,], caption = 'master with grouping variable')
```

```
## Warning in `[<-.data.frame`(`*tmp*`, , isn, value = structure(list(X =
## structure(c("0", : provided 183 variables to replace 4 variables
```

master with grouping variable

X	Text	Positive.	transliterated	int_rep	group_5
0	أبذل الجهد و العرق و توكل على الله و اطمئن فرزقك مضمون	1	abil aljhd al3rQ twkl allh aTm5n frzQk mDmwn	0	0
1	احُبك ربي	1	a7bk rby	0	0
2	اللهم عفوك و رضاك و الجنه	1	allhm 3fwk rDak aljnh	0	0
3	الماء هى الحياة	1	almaq hA al7ya0	0	0
4	بالفعل لازم كل واحد بيتكلم عن الدين اقل مافيها يلتزم باقل مبادئه عشان يكون قدوة مش بيخلي الناس تقول هو ده الدين	1	balf3l lazm kl wa7d bytklm aldyn mafyha yltzm baQl mbad5h 3Wan ykwn Qdw0 mW yyKly alnas tQwl dh aldyn	0	0