Assignment 8: Sentiment Analyzer

In this assignment you will construct your own sentiment analyzer.

1 Word Based Sentiment Analyzer

Write a function that takes a tweet as input and throws back its sentiment. To this end, the function needs to 1) split the tweet into individual words devoid of all punctuation 2) extract any valence terms from this **LIST** found among the words, and 3) calculate the valence of the tweet by averaging the corresponding valence values. Step 2) is best achieved using `terms %in% words`. Regarding step 1) I recommend implementing the procedure from Assignment 7 to split sequences of Emojis. This has the positive side-effect that the Emojis reduce to the smaller set of single-byte Emojis.

```
# ------- sentiment analyzer

# load my lexicon
valence_lexicon = read.table(paste0(my_path,'vaderSent.txt'), header=T, sep='\t')

# define my sentiment analyzer
# try to set min as high as possible - will lead to more robust results
my_sentiment_analyzer = function(tweet, min = 5, method = 'mean',
                                 lexicon = valence_lexicon){

  # remove punctuation
  tweet = stri_replace_all_regex(tweet,'[:punct:]','')

  # split tweet
  words = stri_split_regex(tweet,' ')[[1]]

  # remove empty strings
  words = words[words != '']

  # get table of words
  word_table = table(words)

  # get indices
  matches = lexicon[which(lexicon$term %in% names(word_table)),]

  # expand by occurences
  values = rep(matches$sentiment, word_table[matches$term])

  # return if length(values) >= min
  if(length(values) >= min){

    # check method
    if(method == 'mean'){

      # method mean
      sentiment = mean(values)

    } else {
      # method proportion
      sentiment = sum(values > 0) / sum(values != 0)
    }

  } else {

    # if smaller return NA
    sentiment = NA
  }

  # return result
  return(sentiment)
}
```

2 Measure Emoji Valence

Include all or some Emojis in the list of valence terms. Two to ways to achieve this:

2.1a Context Route

The context route shares similarities to the co-occurence approach applied in past assignments. That is, you can infer the valence of an emoji by assessing the words it co-occurs with. The easiest way to do this is to derive valence using the Sentiment Analyzer from the section above for the tweets the Emoji occurs in and averaging the results. A slightly different and possibly better approach would be to determine the proportion of tweets that are positive for each Emoji. Note, however, that in the latter case the proportions need to be scaled to the range of -4 and 4 to match the range in the valence term list.

2.1b Similarity Route

The similarity route is a sophisticated version of the context route. That is, rather than defining the context as the collection of tweets the Emoji occurs in, we can define the context using the similarities learned from the tweets using, e.g., `word2vec`. The best way to do this, is to train the model again for all occuring Emojis and all occuring valence terms and then compute the valence of the Emojis as the sum of valences of the terms weighted by the cosine similarities between the terms and the respective Emoji. Note, however, that the cosines need to be normalized so that they sum up to 1 to ensure again a valence range of -4 and 4.

2.2 Expand Valence Terms

After completing either the Context or the Similarity Route add the full set of Emojis (or only a subset, in which case the extreme ones should be preferred) to the term list.

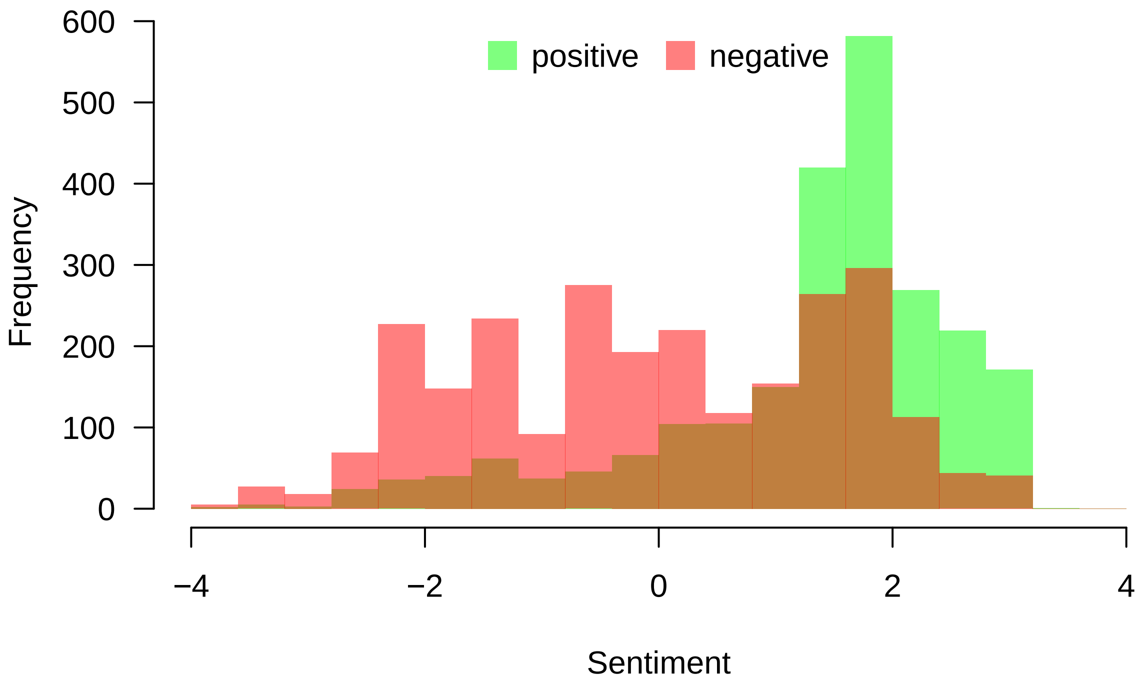
```
# ------- Context route

# my tweets
tweets = unique(my_stream$text)[1:10000]

# get Emojis
utf8 = as.character(emoji_ids$utf8)

# find Emojis
text = paste(tweets, collapse = ' ')
found_emojis = utf8[stri_detect_fixed(text, utf8)]
```

```r
# reduce tweets
emojis = paste(utf8, collapse = '|')
tweets_with_emojis = tweets[stri_detect_regex(tweets, emojis)]

sentiments = c()
method = 'mean'
min = 10 # try to set this as high as possible to avoid spurious results
         #i.e., if you have enought tweets go for, e.g., 100.
for(i in 1:length(found_emojis)){

  # keep track
  # print(i)

  # find tweets with that emojis
  selected_tweets = tweets_with_emojis[stri_detect_fixed(tweets_with_emojis,
                                                         found_emojis[i])]

  # retrieve tweet sentiments
  values = sapply(selected_tweets, my_sentiment_analyzer)

  # calculate sentiment
  if(sum(!is.na(selected_tweets)) >= min){

    if(method == 'mean'){
      sentiment = mean(values, na.rm = T)
    } else {
      sentiment = sum(values > 0, na.rm = T) / sum(values != 0, na.rm = T)
    }
  } else {
    sentiment = NA
  }

  # store sentiments
  sentiments[i] = sentiment
}

# ------- Expand valence terms

# load my lexicon
# valence_lexicon = read.table('vaderSent.txt', header=T, sep='\t')

# define emoji lexicon
emoji_lexicon = data.frame('term'=found_emojis[!is.na(sentiments)],
                           'sentiment'=sentiments[!is.na(sentiments)])

# add emojis to lexicon
expanded_lexicon = rbind(valence_lexicon, emoji_lexicon)
```

3 Evaluate Sentiment Analyzer

Download this **LIST** of classified tweets and use it to evaluate your sentiment analyzer. Specifically, extract the tweets and apply your sentiment analyzer to each of them. Then compare the predictions of your sentiment analyzer with the existing valence classification of positive or negative valence. One way to do this is to compare summary statistics for the predictions for the positie and the predictions for the negative tweets. For instance, one could calculate the means and standard devitions of the prediction and calculate a cohen's d as a measure of sensitivity. Alternatively, you can analyze the number of times your sentiment analyzer predicts the correct polarity, i.e., whether a positive tweet is also assigned an above zero number and vice versa. Make sure that the results demonstrate a successful classification, e.g., by checking the sign and value of d.

```
# get sentiment corpus
corpus = read.table(paste0(my_path,'cleanTwitterSentCorpus.txt'),sep='\t',quote=NULL)

# iterate through corpus
sentiments = c()
for(i in 1:nrow(corpus)){

  # calculate sentiment
  sentiment[i] = my_sentiment_analyzer(corpus[i,2], min = 1, lexicon = expanded_lexicon)

  }

# ------- Plotting sentiments by classification (instead of evaluating d)

# plot sentiments for positive tweets
hist(sentiment[corpus[,1] == 'positive'], col = rgb(0,1,0,alpha=.5),
     border = NA, xlim = c(-4,4), xlab = 'Sentiment', las = 1, main = '',
     breaks = seq(-4,4,.4))

# plot sentiments for negative tweets
hist(sentiment[corpus[,1] == 'negative'], col = rgb(1,0,0,alpha=.5),
     border = NA, xlim = c(-4,4), add = T, breaks = seq(-4,4,.4))

# add legend
legend('top', legend = c('positive', 'negative'), bty = 'n', horiz = T,
       pch = 15, pt.cex = 2, col = c(rgb(0,1,0,alpha=.5), rgb(1,0,0,alpha=.5)))
```



END