Homework 9

Prepare your answers as a **single PDF file**.
Group work: You may work in groups of 1-3. Include all group member names in the PDF file. Only **one person** in the group should submit to Canvas.
Due: check on Canvas.

1. Consider the following short documents:

    Document 1: *how do you do*
    Document 2: *how are you*
    Document 3: *how do you feel*

This problem is meant to be done by hand though you can also use R.

a) Show the Document-Term matrix weighted by Term-frequency (Tf). This is partially complete:

	Document 1	Document 2	Document 3
how	0.25	0.333	0.25
do	0.5	0	0.25
you	0.25	0.33	0.25
are	0	0.33	0
feel	0	0	0.25

b) What is the inverse document frequency (Idf) of each word?

		Idf
how	3	log(3/3) = 0
do	2	log(3/2) = 0.59
you	3	log(3/3) = 0
are	1	log(3/1) = 1.59
feel	1	log(3/1) = 1.59

c) Show the Document-Term matrix weighted by Tf-Idf for this dataset.

	Document 1	Document 2	Document 3
how	0	0	0

do	0.295	0	0.147
you	0	0	0
are	0	0.524	0
feel	0	0	0.397

d) Which word is the most “important” for comparing these documents based on Tf-idf?
The “are” word has the maximum value

2. Download the CSV file on Canvas called `airline_tweets_sentiment_utf8.csv` that contains tweets about airlines[1]. The goal is to create a **word cloud** from the most frequent words in the “tweet” column after some pre-processing. Write R code to do the following tasks (please refer to the text processing R code used in class on Canvas). You must use the `tidytext` and `wordcloud` packages (and not any other text processing package such as `tm`).

a) Load the given CSV file. [code]
```
tweets <- read_csv("~/Downloads/airline_tweets_sentiment_utf8.csv", col_types = "cfcc")
```

b) What is the full text of the first tweet?
```
tweets$tweet[1]
[1] "@united yes. We waited in line for almost an hour to do so. Some passengers just left not wanting to wait past 1am."
```

c) Convert your data to the “tidy” format, i.e., one word per row. (Hint: use unnest_tokens.) How many words are there? [code, number of words]
```
tidy_tweets <- tweets %>% unnest_tokens(word, tweet)
nrow(tidy_tweets)
[1] 261348
```

d) Remove stop words from the tidy dataset. How many words are there? [code, number of words]
```
tidy_tweets <- tweets %>% unnest_tokens(word, tweet) %>% anti_join(stop_words)
nrow(tidy_tweets)
[1] 120335
```

e) Calculate the word count for each word and sort them with the most frequent words first. How many unique words are there? What are the 10 most frequent words with counts? [code, list of 10 word-counts]
```
word_counts <- tidy_tweets %>% arrange(desc(n)) %>% print(head(10))

word
  <chr>
```

[1] Modified from https://www.kaggle.com/crowdflower/twitter-airline-sentiment

```
 1 united
 2 flight
 3 usairw…
 4 americ…
 5 southw…
 6 jetblue
 7 t.co
 8 http
 9 cancel…
10 service

nrow(word_counts)
[1] 14934
```

f) Create a word cloud of the 50 most frequent words. (Hint: use the max.words parameter to wordcloud()). [code, picture of word cloud]

```
wordcloud(words = word_counts$word, freq = word_counts$n, max.words = 50,
scale=c(3,0.5))
```

3. Define a function, cossim(A,B), that takes two vectors as input and computes their cosine similarity:

$$A.norm = \frac{A}{||A||} = \frac{A}{\sqrt{sum(A_i^2)}}$$

$$B.norm = \frac{B}{||B||} = \frac{B}{\sqrt{sum(B_i^2)}}$$

$$cossim(A, B) = sum(A.norm * B.norm)$$

Note: your function does not need any loops as you can use the vectorized operators in R. Test your function for correctness. For instance, the cosine similarity of any vector to itself is 1 and cosine similarity between vectors (1,2,3) and (0,2,5) should be 0.9429542

a. Give code

```
cossim <- function(A, B) {
    A.norm <- sqrt(sum(A^2))
    B.norm <- sqrt(sum(B^2))
    return(sum(A * B) / (A.norm * B.norm))
}
```

b. Output of cossim(c(1,2,3), c(0,2,5))

```
cossim(c(1, 2, 3), c(0, 2, 5))
[1] 0.9429542
```

4. Use the same CSV file from Problem 2. Each tweet also has an associated "sentiment" - whether the expressed opinion in the tweet is positive, negative, or neutral. The goal is to use this data to **predict** the sentiment of the first tweet[2].

Write R code to do the following tasks:

a) Repeat the steps in Problem 2a-d, i.e., calculate the word counts but keep the tweet_id column too. [code]
   - Note: `tweet_id` is a long integer but `read_csv()` will read it as a double by default. Hence, give the `col_types="cfcc"` parameter to `csv_read()` which forces the tweet_id column to be a character/string type. Make sure the `tweet_id` of the first tweet is "`567591480085463000`"

```
tweets <- read_csv("~/Downloads/airline_tweets_sentiment_utf8.csv",
col_types = "cfcc")
tidy_tweets <- tweets %>% unnest_tokens(word, tweet) %>%
anti_join(stop_words)
word_counts <- tidy_tweets %>% count(word, tweet_id) %>%
arrange(desc(n))
```

b) Calculate the tf-idf weight for each word and tweet_id. [code]

```
tf_idf <- word_counts %>% bind_tf_idf(word, tweet_id, n)
```

[2] The learning goal is to get familiar with fundamental text processing steps; so do not install and run any sentiment analysis library or function for this question.

c) Transform the “long” tf-idf data into a row-column table format with words along rows and tweet_ids along columns. How many unique words are present? [code]

```
mydata <- tf_idf %>% pivot_wider(names_from = tweet_id, values_from = tf_idf, values_fill = 0)
```

d) Predict the sentiment of the first tweet (with tweet_id = “`567591480085463000`”) using the 1-Nearest Neighbor approach with cosine similarity.
   - Write code to compare the tf-idf vector of the first tweet to that of the remaining tweets using the cossim function from Problem 3. The sentiment of the tweet with the **highest** similarity will then be the predicted sentiment. [code]
     - Hints: the first column in your tf-idf table may not correspond to the original first tweet. The “first” tweet is that with tweet_id = “`567591480085463000`”.
     - to get the names of columns, use `names(mytable)`
     - to get a column of a data.frame/tibble as a vector use [[ ]]. For example, `mytable[[2]]` returns the 2nd column as a vector.

```
query_tweet <- "567591480085463000"
columns_to_compare <- names(mydata)[6:ncol(mydata)]
result_df <- tibble(column_name = columns_to_compare, similarity = map_dbl(mydata[columns_to_compare], ~cossim(mydata[[query_tweet]], .)))
result_df <- result_df %>% filter(column_name != query_tweet)
```

   - What is the tweet_id of the most similar tweet(s)? What is the text of this tweet(s)? What is its sentiment (this is the predicted sentiment)?

```
most_similar_tweet <- result_df %>% filter(similarity == max(similarity)) %>%
# (The similarity was 0.208 for the most similar tweet)
select(column_name)
tweets %>% filter(tweet_id == most_similar_tweet$column_name)
```

```
tweet_id          airline_sentiment airline tweet
  <chr>             <fct>             <chr>   <chr>
1 569683647990599000 negative          United  @united I'm really glad I just waited on the phone for over an h…
```

   - Does the predicted sentiment match its known sentiment (from row 1)?
     YES, this is the predicted sentiment

e) Is this bag-of-words approach in general a good way to predict sentiment in tweets? Why or why not? Answer in 2-3 sentences.

The bag-of-words approach, while effective in capturing keyword frequencies, may struggle with nuances like context and sarcasm in tweets. Additionally, it overlooks word order, a crucial aspect in natural language. More advanced techniques, such as utilizing word embeddings or deep learning models, could better grasp the subtleties of sentiment in tweets by considering semantic relationships and context