Sergi Solís Valdivieso u172769 - Hong-ming Xiang Vico u172801

IRWA Project Part III: Ranking

Statement

Ranking

Ranking score:
Given a query, we want to get the top-20 documents related to the query.

GOAL:
Find all the documents that contain all the words in the query and sort them by their relevance with regard to the query.

SCORE:
1. You're asked to provide 2 different ways of ranking:

    a. **TF-IDF + cosine similarity:** Classical scoring, we have also seen during the practical labs
    b. **Your-Score + cosine similarity**: Here the task is to create a new score, and it's up to you to create a new one.
    c. **BM25**

Explain how the ranking differs when using TF-IDF and BM25 and think about the pros and cons of using them.
Regarding your own score, justify the choice of the score (pros and cons). The only constraint you have is that the score needs to involve the tweets information regarding the popularity over the social network (number of likes, number of tweets, number of comments, etc...)

2. Return a top-20 list of documents for each of the 5 queries, using word2vec + cosine similarity.

    To use the word2vec, you need to generate the tweet representation, which here is expressed as a unique vector of the same dimension of the words, generated as the average of the vectors representing the words included in the tweet:
    Ex. "I won the election"
    Having a vector (generated through word2vec) representing each word, e.g. I=v_1, won = v_2, the = v_3, election = v_4, all of the same number of dimensions n, it is possible to represent the text above and generating a unique representation, by averaging v_1, v_2, v_3, v_4. The result will be a new vector v, of the same dimension n representing the text "I won the election". Since it's a tweet in our case we will talk about tweet2vec.

3. Can you imagine a better representation than word2vec? Justify your answer. (**HINT** - what about Doc2vec? Sentence2vec? Which are the pros and cons?)

GitHub Repository

All the code and resources for the project will be submitted to the following repository:
https://github.com/homexiang3/IRWA-2022-u172769-u172801
The repository TAG is:

IRWA-2022-u172769-u172801-part-3

Code development

Previous work considerations

This is the third d part of the IRWA 2022 project so we skip some previous code documentation provided in the first and second report such as Google Drive connection, how to load JSON data and how we preprocess the data.

For further details visit:
https://github.com/homexiang3/IRWA-2022-u172769-u172801/blob/main/Project/P1/IRWA-2022-u172769-u172801-part-1.pdf
https://github.com/homexiang3/IRWA-2022-u172769-u172801/blob/main/Project/P2/IRWA-2022-u172769-u172801-part-2.pdf

Indexing

Create Index

We use a global create index function similar to the provided in the last project part (computes index, tweet_index, idf, df and tf), but since we need to compute BM25 and our own score taking into account likes and retweets we will also store and return 4 new variables:

- avg_tweet_length = stores the average length of tweets in the corpus
- avg_tweet_likes = stores the average number of likes per tweet in the corpus
- avg_tweet_rt = stores the average number of retweets per tweet in the corpus
- tweet_features = dictionary that given the tweet id returns a list with 3 values [length, likes, retweets] of this tweet (this will be useful when we want to compute BM25 and own score)

```
# Create index function
def create_global_index(tweets,num_tweets):
    index = defaultdict(list)
    tweet_index = {}  # dictionary to map tweet id with index in tweets list
    counter = 0 # keep track of index inside tweets
    tf = defaultdict(list)  # term frequencies of terms in documents (documents in the same order as in the main index)
    df = defaultdict(int)  # document frequencies of terms in the corpus
    idf = defaultdict(float) # inverse document frequency for each term
    tweet_features = defaultdict(list) #length likes retweets of each tweet (in this way we don't need to pass tweets in rank docs function)

    avg_tweet_length = 0 #store average length of tweets
```

```
    avg_tweet_likes = 0 #store average likes of tweets
    avg_tweet_rt = 0 #store average rt of tweets

    for t in tweets:  # for all tweets

        tweet_id = t.id
        terms = preprocess(t.tweet) #preprocess tweet and return list of terms
        tweet_index[tweet_id] = counter # Save original tweets position with tweet id to recover all the
information
        current_page_index = {}

        tweet_features[tweet_id] = [len(terms),t.likes, t.retweets] #Store useful features
        avg_tweet_length += len(terms) # sum length of actual tweet
        avg_tweet_likes += t.likes # sum likes of actual tweet
        avg_tweet_rt += t.retweets # sum rt of actual tweet

        counter = counter + 1 # Move to next tweets position

        for position, term in enumerate(terms):
            try:
                # if the term is already in the dict append the position to the corresponding list
                current_page_index[term][1].append(position)
            except:
                # Add the new term as dict key and initialize the array of positions and add the position
                current_page_index[term] = [tweet_id, array('I', [position])]  #'I' indicates unsigned int
(int in Python)

        # normalize term frequencies
        # Compute the denominator to normalize term frequencies (formula 2 above)
        # norm is the same for all terms of a document.
        norm = 0
        for term, posting in current_page_index.items():
            # posting will contain the list of positions for current term in current document.
            # posting ==> [current_doc, [list of positions]]
            # you can use it to infer the frequency of current term.
            norm += len(posting[1]) ** 2
        norm = math.sqrt(norm)

        #calculate the tf(dividing the term frequency by the above computed norm) and df weights
        for term, posting in current_page_index.items():
            # append the tf for current term (tf = term frequency in current doc/norm)
            tf[term].append(np.round(len(posting[1]) / norm, 4)) ## SEE formula (1) above
            #increment the document frequency of current term (number of documents containing the current
term)
            df[term] += 1 # increment DF for current term

        # Compute IDF
        for term in df:
            idf[term] = np.round(np.log(float(num_tweets / df[term])), 4)

        #merge the current page index with the main index
        for term_page, posting_page in current_page_index.items():
            index[term_page].append(posting_page)

    avg_tweet_length = avg_tweet_length/num_tweets #Average length of all tweets
    avg_tweet_likes = avg_tweet_likes/num_tweets #Average likes of all tweets
    avg_tweet_rt = avg_tweet_rt/num_tweets #Average rt of all tweets

    return index, tf, df, idf, tweet_index, avg_tweet_length, avg_tweet_likes, avg_tweet_rt,
tweet_features
```

Then we just need to run the function to create all the mentioned variables

```
# Create tfidf index with all the tweets WARNING: THIS PROCESS MAY TAKE LONG TIME (we run it for 4 min)
num_tweets= len(tweets)
index, tf, df, idf, tweet_index, avg_tweet_length, avg_tweet_likes, avg_tweet_rt, tweet_features =
create_global_index(tweets, num_tweets)
```

Ranking

TF-IDF

This function remains the same as in the previous project part, first of all, compute the normalized query vector applying tf-idf and then compute the doc vectors for each doc that has at least one of the terms doing again tf-idf (notice that tf is already normalized in index creation function). Finally computes for each doc vector the cosine similarity by doing the dot product between the doc vector and the query vector and ranks the results in descending order, then returns the docs and the scores.

```
def rank_tfidf_documents(terms, docs, index, idf, tf, title_index):

    # I'm interested only on the element of the docVector corresponding to the query terms
    # The remaining elements would became 0 when multiplied to the query_vector
    doc_vectors = defaultdict(lambda: [0] * len(terms)) # I call doc_vectors[k] for a nonexistent key k,
the key-value pair (k,[0]*len(terms)) will be automatically added to the dictionary
    query_vector = [0] * len(terms)

    # compute the norm for the query tf
    query_terms_count = collections.Counter(terms)  # get the frequency of each term in the query.

    query_norm = la.norm(list(query_terms_count.values()))

    for termIndex, term in enumerate(terms):  #termIndex is the index of the term in the query
        if term not in index:
            continue
        # query_vector[termIndex]=idf[term]  # original
        ## Compute tf*idf(normalize TF as done with documents)
        query_vector[termIndex] = query_terms_count[term] / query_norm * idf[term]

        # Generate doc_vectors for matching docs
        for doc_index, (doc, postings) in enumerate(index[term]):
            if doc in docs:
                doc_vectors[doc][termIndex] = tf[term][doc_index] * idf[term]

    # Calculate the score of each doc
    # compute the cosine similarity between queyVector and each docVector:

    doc_scores = [[np.dot(curDocVec, query_vector), doc] for doc, curDocVec in doc_vectors.items()]
    doc_scores.sort(reverse=True)
    result_docs = [x[1] for x in doc_scores]
    result_rank = [x[0] for x in doc_scores] #get rank value
    #print document titles instead if document id's
    #result_docs=[ title_index[x] for x in result_docs ]
    if len(result_docs) == 0:
        print("No results found, try again")
        query = input()
        docs = search_tweets(query, index,0)
    return result_docs, result_rank
```

BM25

For BM25 we don't need to compute query vectors, but we must create for each doc that contains at least one term a doc vector that applies the BM25 function seen in class:

$$BM25 = \sum_{t \in q} \log\left[\frac{N}{df(t)}\right] \cdot \frac{(k_1+1)\cdot tf(t,d)}{k_1 \cdot \left[(1-b)+b\cdot\frac{dl(d)}{dl_{avg}}\right] + tf(t,d)}$$

Similarly as what we have done in tf-idf rank we take each doc tf by doing tf[term][doc_index], then to scale the document length we use the tweet_features variable where “doc” indicate the tweet id and [0] is the index referring to length, we also use avg_tweet_length to perform the operation.

Finally, we need to do a sum for each doc vector corresponding to each query term to perform the final score and rank them in descending order.

```
def rank_BM25_documents(terms, docs, index, idf, tf, title_index, avg_tweet_length, k1 = 1.2, b = 0.75):

    # I'm interested only on the element of the docVector corresponding to the query terms
    # The remaining elements would became 0 when multiplied to the query_vector
    doc_vectors = defaultdict(lambda: [0] * len(terms)) # I call doc_vectors[k] for a nonexistent key k,
the key-value pair (k,[0]*len(terms)) will be automatically added to the dictionary

    for termIndex, term in enumerate(terms):  #termIndex is the index of the term in the query
        if term not in index:
            continue

        # Generate BM25 Score
        for doc_index, (doc, postings) in enumerate(index[term]):
            if doc in docs:
              doc_vectors[doc][termIndex] = (tf[term][doc_index] * idf[term] * (k1 + 1)) /
k1*((1-b)+b*(tweet_features[doc][0]/avg_tweet_length)) + tf[term][doc_index]


    # Calculate the score of each doc
    # compute the sum between BM25 scores for each query term

    doc_scores = [[np.sum(curDocVec), doc] for doc, curDocVec in doc_vectors.items()]
    doc_scores.sort(reverse=True)
    result_docs = [x[1] for x in doc_scores]
    result_rank = [x[0] for x in doc_scores] #get rank value

    if len(result_docs) == 0:
        print("No results found, try again")
        query = input()
        docs = search_tweets(query, index, 1)
    return result_docs, result_rank
```

OwnScore

For our OwnScore function we decided to enhance tf-idf + cosine similarity adding an additional weighting function based on the number of likes and retweets. We decided to not use comments because we think that it is easier to bias the number of comments in a tweet.

The function receives by parameter the regular cosine similarity, the doc (tweet_id), the average likes, the average retweets, tweet features (dictionary that stores for each tweet id the length, likes, and retweets) and finally the like and rt parameters to model the function, in this case, we decided to keep it simple and put 1 as default.

Then, for likes and rt compute the weighting factor, to perform this new weight we multiply the corresponding parameter and the log2 of the likes/retweets of the doc divided by the average plus 1. In this way, we smooth the tweets with a lot of likes or retweets (with the logarithmic function) and in case we don’t have likes or retweets the factor returns 0.

Finally, we just multiply our standard cosine similarity by the sum of each of the factors plus 1, so in case we don't have either likes or retweets the result will be the standard cosine similarity and factors only add weight to the score.

```
def popularity_score (cos_similarity, doc, avg_tweet_likes, avg_tweet_rt, tweet_features, like_param = 1,
rt_param = 1):
  #Like/rt param to assign weights to like or rt
  like_factor = like_param*(math.log2((tweet_features[doc][1]/ avg_tweet_likes)+1))
  rt_factor = rt_param*(math.log2((tweet_features[doc][2]/ avg_tweet_rt)+1))

  popularity_score = cos_similarity * (like_factor + rt_factor + 1)
  return popularity_score
```

For the ranking part we have the same function that in TF-IDF, but instead of just computing *np.dot*, we use our popularity_score function to enhance the ranked tweets by popularity.

```
def rank_ownscore_documents(terms, docs, index, idf, tf, avg_tweet_likes, avg_tweet_rt, tweet_features):

    # I'm interested only on the element of the docVector corresponding to the query terms
    # The remaining elements would became 0 when multiplied to the query_vector
    doc_vectors = defaultdict(lambda: [0] * len(terms)) # I call doc_vectors[k] for a nonexistent key k,
the key-value pair (k,[0]*len(terms)) will be automatically added to the dictionary
    query_vector = [0] * len(terms)

    # compute the norm for the query tf
    query_terms_count = collections.Counter(terms)  # get the frequency of each term in the query.

    query_norm = la.norm(list(query_terms_count.values()))

    for termIndex, term in enumerate(terms):  #termIndex is the index of the term in the query
        if term not in index:
            continue
        # query_vector[termIndex]=idf[term]  # original
        ## Compute tf*idf(normalize TF as done with documents)
        query_vector[termIndex] = query_terms_count[term] / query_norm * idf[term]

        # Generate doc_vectors for matching docs
        for doc_index, (doc, postings) in enumerate(index[term]):
            if doc in docs:
                doc_vectors[doc][termIndex] = tf[term][doc_index] * idf[term]

    # Calculate the score of each doc
    # compute the cosine similarity between queyVector and each docVector:

    doc_scores = [[popularity_score(np.dot(curDocVec,
query_vector),doc,avg_tweet_likes,avg_tweet_rt,tweet_features), doc] for doc, curDocVec in
doc_vectors.items()]
    doc_scores.sort(reverse=True)
    result_docs = [x[1] for x in doc_scores]
    result_rank = [x[0] for x in doc_scores] #get rank value
    #print document titles instead if document id's
    #result_docs=[ title_index[x] for x in result_docs ]
    if len(result_docs) == 0:
        print("No results found, try again")
        query = input()
        docs = search_tweets(query, index,2)
    return result_docs, result_rank
```

Search Function

For the search function we decided to make it simple and implement a unique search function with the rank method requested as a parameter. First of all, creates the list of docs with at least one query term and then using the rank_method decides which rank is used to perform the operation.

```
def search_tweets(query, index, rank_method=0):
    query = preprocess(query)#create list of query terms (each term is preprocessed to match terms in
index)
    docs = set()
    for term in query:
        try:
            # store in term_docs the ids of the docs that contain "term"
            term_docs = [posting[0] for posting in index[term]]

            # docs = docs Union term_docs
            docs |= set(term_docs)
        except:
            #term is not in index
            pass
    docs = list(docs)

    if rank_method == 0:
        ranked_docs, ranked_score = rank_tfidf_documents(query, docs, index, idf, tf, tweet_index)#TFIDF

    elif rank_method == 1:
        ranked_docs, ranked_score = rank_BM25_documents(query, docs, index, idf, tf,
tweet_index,avg_tweet_length)#BM25

    else:
        ranked_docs, ranked_score = rank_ownscore_documents(query, docs, index, idf, tf,
tweet_index,avg_tweet_length, avg_tweet_likes, avg_tweet_rt,)#OwnScore

    return ranked_docs, ranked_score
```

Display Function

Similarly as in the Search function, we create a function to visualize the query, where rank_method is requested to perform the search and rank methods and also you can specify the number of retrieved tweets (by default is 20). Then we process the retrieved docs to take the information from the original tweet (using tweet_index) and visualize it using tabulate package.

```
# Define query to visualize - top 20 ranked tweets displayed
# example used in our report: "Hurricane Ian in Florida"
def query_visualizer(rank_method,top=20):
  print("Insert your query (i.e.: 'hurricane ian'):\n")
  query = input()
  ranked_docs, ranked_score = search_tweets(query, index, rank_method)
  visualization_tweets = []
  #create table headers
  headers = ['DOC_ID','ID','TWEET','USERNAME','DATE','HASHTAGS','LIKES', 'RETWEETS', 'URL']
  print("\n======================\nSample of {} results out of {} for the searched query:\n".format(top,
len(ranked_docs)))
  #create table of tweets for each match
  for d_id in ranked_docs[:top]:
      t = tweet_index[d_id]
      visualization_tweets.append(tweets[t])
  #print ranked score
```

```
print("Ranked Scores:", ranked_score[:top])
#print table
print(tabulate(visualization_tweets, headers=headers, tablefmt='grid'))
```

Results

We use the same query “Hurricane Ian in Florida” to test our ranking functions and retrieve top-20 tweets.

TF-IDF Rank

DOC_ID	ID	TWEET
doc_3373	1575864607578927110	After slamming Florida, Hurricane Ian barrels toward South Carolina #Hurricane #Florida #Orlando #IAN #HurricaneIan #Florida #floridahurricane #Ian2022 #PineIsland #hurricaneian #ian #stormsurge #BREAKING #Viral #Cuba #rain #Strom #climate #tornado https://t.co/tQKWZ7FJk4
doc_111	1575916596693913600	Florida’s Long Road To Recovery #florida #hurricaneian #hurricane #ian https://t.co/zcE3Yv79bg
doc_634	1575910361298968576	Hurricane IAN #Ian #HurricaneIan #HurricanIan #Huracan #HuracanIan #Hurricane https://t.co/HbllO4Q3vB
doc_746	1575909158746038277	Yesterday, Hurricane Ian made landfall near Cayo Costa, Fla., as a Category 4 hurricane, bringing record wind and storm surge #hurricaneian #florida
doc_63	1575917184655974402	Hurricane Ian's toll on Florida's Lee County #hurricaneian #floridaflood2022 #Leecountyflood #Ian https://t.co/4J5J86bDcu
doc_1963	1575885709734543360	The Latest on Hurricane Ian: https://t.co/nk6l0YkIH0 #hurricaneian #ian #severewx #kprc2 #click2houston #sc #florida
doc_2943	1575870035016314882	67-Year-Old Man Died While Trying To Escape Flood Waters Of Ian In Florida #florida #hurricaneian #ian #hurricane #TropicalSto https://t.co/3LJS00C7rU
doc_2637	1575873925342326785	Repost from @GMA Images show Hurricane Ian’s destruction after making landfall on Florida’s west coast as a Category 4 hurricane. #hurricaneian #hurricane #weather #florida #news https://t.co/FuV5I5Pp37
doc_3757	1575859391491973120	"Only in Florida." A woman and a dog running through a flooded park in Orlando, Florida, despite high waters following Hurrica
doc_2877	1575870720932208640	Hurricane Ian From Cuba through Florida to South Carolina. #HurricaneIan https://t.co/7TvpUjrtlW

BM25 Rank

DOC_ID	ID	TWEET
doc_3373	1575864607578927110	After slamming Florida, Hurricane Ian barrels toward South Carolina #Hurricane #Florida #Orlando #IAN #HurricaneIan #Florida #floridahurricane #Ian2022 #PineIsland #hurricaneian #ian #stormsurge #BREAKING #Viral #Cuba #rain #Strom #climate #tornado https://t.co/tQKWZ7FJk4
doc_746	1575909158746038277	Yesterday, Hurricane Ian made landfall near Cayo Costa, Fla., as a Category 4 hurricane, bringing record wind and storm surge #hurricaneian #florida
doc_3757	1575859391491973120	"Only in Florida." A woman and a dog running through a flooded park in Orlando, Florida, despite high waters following Hurrica
doc_2637	1575873925342326785	Repost from @GMA Images show Hurricane Ian’s destruction after making landfall on Florida’s west coast as a Category 4 hurricane. #hurricaneian #hurricane #weather #florida #news https://t.co/FuV5I5Pp37
doc_3596	1575861871994871809	Watching a storm come in a few years ago in South Florida. I always worried about my East Coast friends and hurricanes - never
doc_1830	1575888736000147457	Hurricane Ian update East #orlando … waters finally going down . Streets are impassible all around me. #Hurricane #hurricaneIa
doc_1922	1575886742363594752	Hurricane Ian update in East #orlando … waters finally going down . Streets are impassible all around me. #Hurricane #hurrica
doc_3894	1575857686163652608	Our hearts go out to those in Florida suffering the terrible consequences of #Hurricane #Ian. Here are some tips for those wi
doc_419	1575912910689165312	#Trump desperate for #cash sending out #donation requests in #Florida before #HurricaneIan #Hurricane makes #Landfall.. #globa #DonaldTrump is raising money off a "#ransacking" in Florida – but it has nothing to do with Hurricane Ian https://t.co/XaSr32
doc_2401	1575876964833001475	FL - CATASTROPHE BULLETIN - #HurricaneIan Hurricane Ian made landfall in Florida on September 28. Coverage and other related issues are discussed in this bulletin to as
doc_1352	1575900805336485888	JUST IN: Prime Minister @YairLapid extended his support and encouragement to the people of Florida after Hurricane Ian ravaged
doc_3524	1575863012137013248	After slamming Florida, Hurricane Ian barrels toward South Carolina #Ian #SouthCarolina #HurricaneIan #Hurricane #Storm #Climate #Viral #Rain #Tornado #TropicsWx #SCwx https://t.co/1XYmsW7IYW
doc_74	1575917111331151872	😱🌪️Timelapse shows devastating storm surge from Hurricane Ian in Fort Myers, Florida https://t.co/UG72qvxNCF

Sergi Solís Valdivieso u172769 - Hong-ming Xiang Vico u172801

OwnScore Rank

```
Hurricane Ian in Florida

======================
Sample of 20 results out of 1652 for the searched query:

Ranked Scores: [14.227426089432337, 9.941696797034025, 9.01657435869609, 8.902993896791122, 7.656569598775833, 7.656336372859996, 7.499931669527972, 7.107061
```

DOC_ID	ID	TWEET
doc_3636	1575861142211235842	Hurricane Ian heads towards South Carolina after striking Florida. The death toll in Florida has risen to 21. We are praying from Israel for all those affected. Our hearts are with you. #Hu
doc_1860	1575887919872475140	#Charleston #Hurricane #IAN #SouthCarolina #Ian #TropicalStormIan #HurricaneIan #MyrtleBeach #Savannah #NorthCarolina #OuterBanks #Augusta #Wilmington #Jacksonville Market and Church Streets https://t.co/V6NxnQjBh9
doc_1335	1575901046869671936	WATCH: After ravaging Florida, a strengthened Hurricane Ian has made landfall in South Carolina and is predicted to enter
doc_1125	1575904184590663680	WHAT'S THIS? Squirrel inspecting the damage in his tree in Venice, Florida after Hurricane Ian. (via Catherin Richardson)
doc_1586	1575894492443336714	House Democrats are saddened and mourn the loss of life in Florida from #HurricaneIan. We pray for the safety of all in Ia We are moving swiftly for those affected – with immediate relief and resources, as well as long term help to rebuild their
doc_3757	1575859391491973120	"Only in Florida." A woman and a dog running through a flooded park in Orlando, Florida, despite high waters following Hur
doc_463	1575912351152283648	#Charleston streets #flooded with inches of standing water #Charleston #Hurricane #IAN #SouthCarolina #Ian #TropicalStormIan #HurricaneIan #MyrtleBeach #Savannah #NorthCarolina #OuterBanks #Augusta #Wilmington #Jacksonville
doc_3072	1575868577491869697	Help Reach Out America serve Florida residents impacted by Hurricane Ian. #HurricaneIan https://t.co/hWLiDex5EW
doc_3373	1575864607578927110	After slamming Florida, Hurricane Ian barrels toward South Carolina #Hurricane #Florida #Orlando #IAN #HurricaneIan #Florida #floridahurricane #Ian2022 #PineIsland #hurricaneian #ian #storms #BREAKING #Viral #Cuba #rain #Strom #climate #tornado https://t.co/tQKWZ7FJk4
doc_1950	1575885951057829890	Wow! You can start to see #flooding from #HurricaneIan on one of the Charleston, #SouthCarolina livecams right now. Lets h
doc_1638	1575893575937581069	JUST IN: Hurricane Ian could be the costliest Florida storm since Hurricane Andrew in 1992 as insurers brace for a hit of
doc_819	1575908495248920577	WATCH: Hurricane Ian pushes sea water to the coast of South Carolina. #HurricaneIan #SouthCarolina #Environment https://t.
doc_426	1575912824853127168	I have no idea how this toy playhouse survived Hurricane Ian, (found sitting like this), but I’m booking a room there if a

Discussion

Looking at the results we can see that BM25 and TF-IDF share some results but overall in our own judgment BM25 performs a better ranking, we believe that this is because TF-IDF only takes into account the term frequency and the document frequency, giving results with more term saturation (see the third result of TF-IDF for example). On the other hand, BM25 tries to take into account document length and term saturation and we can see how the results overall have a better relevance since all the terms of the query are more present in BM25 tweets. Moreover, for our own score, we have a completely different set of top tweets since in the previous ones there are a big percentage of tweets with a low number of likes and retweets, but with the results, we can easily check that our retweet and like enhancement performs correctly since “popular” tweets following the query requested information are displayed. (Remember that we have assigned the same weight for like and rt parameters, you can test how the retrieved tweets change in function of likes - rt changing this feature)

Word2vec

For this part we decided to use the Word2Vec package from gensim to create our model. For further information visit https://radimrehurek.com/gensim/models/word2vec.html

Functions

First of all we need to tokenize the content of our tweets to train our word2vec model, to do this we just append in a list the content of each tweet preprocessed and then initialize and save our word2vec model using the tokenized_tweets corpus. We decided to use a min_count = 1 to avoid problems when finding words in the vocabulary.

```
tokenized_tweets = []
for t in tweets:
  terms = preprocess(t.tweet)
  tokenized_tweets.append(terms)

model = Word2Vec(sentences=tokenized_tweets, window=5, min_count=1, workers=4)
model.save("word2vec.model")
```

We define our cosine similarity function, that given two vectors return de dot product normalized, to archive this we just need to use *np.dot* function and *np.linalg.norm* function.

```
def custom_cosine_similarity(v1, v2):
  return np.dot(v1,v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
```

For our search and rank function we proceed as always first, we create our set of docs that contain at least one term of the query but now we also store the sum of word2vec values of each query term in a variable named word2vec_query, then when we finish storing all docs we perform the average of this vector.

For the rank part, we proceed in a similar way, for each doc, we tokenize the content of the tweet and perform the sum of word2vec values of each tweet, finally, we perform the average of the tweet vector and we store it in a dictionary that given tweet id as key, stores as value the resulting tweet2vec.
Finally, for each key in our dictionary, we perform our cosine similarity between the word2vec_query and each tweet2vec vector and we sort the results in descending order.

```
def searchrank_tweet2vec(query, index):
    query = preprocess(query)#create list of query terms (each term is preprocessed to match terms in
index)
    docs = set()
    word2vec_query = 0
    for term in query:
        try:
            # store in term_docs the ids of the docs that contain "term"
            term_docs = [posting[0] for posting in index[term]]

            # docs = docs Union term_docs
            docs |= set(term_docs)

            # Store value of term
            word2vec_query += model.wv[term]
        except:
            #term is not in index
            pass

    word2vec_query = word2vec_query / len(query)
    docs = list(docs)
```

```
    #create doc_vectors dict to store word2vec tweets with tweet id as key
    doc_vectors = defaultdict(lambda: [0])

    # for each tweet, serialize data and add vector of each word in dict, then average dividing by len of
terms
    for doc in docs:
        i = tweet_index[doc]
        tweet_text = preprocess(tweets[i].tweet)
        for term in tweet_text:
          doc_vectors[doc] +=  model.wv[term]
        doc_vectors[doc] = doc_vectors[doc] / len(tweet_text)

    # Apply custom cosine similarity between tweet vector and query vector and sort results
    doc_scores = [[custom_cosine_similarity(word2vec_t, word2vec_query), doc] for doc, word2vec_t in
doc_vectors.items()]
    doc_scores.sort(reverse=True)
    result_docs = [x[1] for x in doc_scores]
    result_rank = [x[0] for x in doc_scores] #get rank value

    if len(result_docs) == 0:
        print("No results found, try again")
        query = input()
        docs = search_tweets(query, index,0)

    return result_docs, result_rank
```

We have an identical version of Display Function to display our top 20 retrieved tweets using tweet2vec rank function.

```
# Define query to visualize - top 20 ranked tweets displayed
# example used in our report: "Hurricane Ian in Florida"
def visualize_tweet2vec(top=20):
  print("Insert your query (i.e.: 'hurricane ian'):\n")
  query = input()
  ranked_docs, ranked_score = searchrank_tweet2vec(query, index)
  visualization_tweets = []
  #create table headers
  headers = ['DOC_ID','ID','TWEET','USERNAME','DATE','HASHTAGS','LIKES', 'RETWEETS', 'URL']
  print("\n======================\nSample of {} results out of {} for the searched query:\n".format(top,
len(ranked_docs)))
  #create table of tweets for each match
  for d_id in ranked_docs[:top]:
      t = tweet_index[d_id]
      visualization_tweets.append(tweets[t])
  #print ranked score
  print("Ranked Scores:", ranked_score[:top])
  #print table
  print(tabulate(visualization_tweets, headers=headers, tablefmt='grid'))
```

Sergi Solís Valdivieso u172769 - Hong-ming Xiang Vico u172801

Results

For the results we decided to use the same query of our previous rank methods “Hurricane Ian in Florida”.

```
Hurricane Ian in Florida

======================
Sample of 20 results out of 1652 for the searched query:

Ranked Scores: [0.9999840971394219, 0.9999801341499555, 0.9999782395786531, 0.9999750357616188, 0.9999740424371861, 0.9999723491402742, 0.9999709365321743, 0.999
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| DOC_ID   |                 ID | TWEET
+==========+====================+=========================================================================================================================
| doc_634  | 1575910361298968576 | Hurricane IAN #Ian #HurricaneIan #HurricanIan #Huracan #HuracanIan #Hurricane https://t.co/HblIO4Q3vB
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_1933 | 1575886315740336128 | Hurricane Ian and Its Impacts Update: https://t.co/Zfx80G03Sk #NBC29 #VAwx #Ian #HurricaneIan https://t.co/3eBl1QDWoA
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_1963 | 1575885709734543360 | The Latest on Hurricane Ian: https://t.co/nk6l0YkIH0 #hurricaneian #ian #severewx #kprc2 #click2houston #sc #florida
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_3373 | 1575864607578927110 | After slamming Florida, Hurricane Ian barrels toward South Carolina
|          |                    | #Hurricane #Florida #Orlando #IAN #HurricaneIan #Florida #floridahurricane #Ian2022 #PineIsland #hurricaneian #ian #stormsurge
|          |                    | #BREAKING #Viral #Cuba #rain  #Strom #climate #tornado https://t.co/tQKWZ7FJk4
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_63   | 1575917184655974402 | Hurricane Ian's toll on Florida's Lee County
|          |                    |
|          |                    | #hurricaneian #floridaflood2022 #Leecountyflood #Ian https://t.co/4J5J86bDcu
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_807  | 1575908618200924167 | Ian is predicted to spare Atlanta
|          |                    |
|          |                    | #HurricaneIan #Hurricane #Ian #AtlantaNews #ATL https://t.co/Zib8NtmG2k
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_3572 | 1575862334924689408 | Hurricane Ian unearthed bodies in the cemetery!
|          |                    | https://t.co/OijETpAaal #hurricaneian #florida #oaklandcemetery https://t.co/0WckULBKHG
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_2905 | 1575870458964344833 | Hurricane Ian heads towards South Carolina after striking Florida.
|          |                    | The death toll in Florida has risen to 21. We are praying for all those affected. Our hearts are with you. #HurricaneIan https:
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_2022 | 1575884215417462784 | LION NATION!!
|          |                    |
|          |                    | Clear skies in #MiamiGardens. By the grace of God we avoided the path of Ian; praying for those around #Florida and the rest o
|          |                    |
|          |                    | #hurricaneIan #hurricane #Ian🌀 #Ian 🌀 https://t.co/hwAzeGjak8
+----------+--------------------+-------------------------------------------------------------------------------------------------------------------------
| doc_3636 | 1575861142211235842 | Hurricane Ian heads towards South Carolina after striking Florida.
|          |                    |
```

Discussion

Looking at Word2vec results and comparing them to our other ranking methods we can say that the most similar approach (in terms of the list of top-20 results) is TF-IDF + cosine similarity. We believe that this behavior makes sense since word2vec is a word embedding technique that tries to relate words with semantic closeness after scanning a given corpus while tf-idf tries to determine word significance and relevance given a corpus. Moreover, since we just take into account the semantic closeness between the query and the tweet information the results are worse in our opinion.

Regarding discussing better representations of word2vec, we believe that doc2vec and sentence2vec should result in higher relevance results since we are comparing the whole semantic meaning of the document or sentence to the query, and not just averaging words, which for example allow us to compare entire tweets. On the other hand, we think that embedding a whole doc vector or sentence vector should be less efficient.