IRTM
Wi 20/21
Assigment 3

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

Task 1

Subtask 1

$$w_{t,d} = (1 + \log tf_{t,d}) * \log \frac{N}{df_t}$$

- $w_{pens,d_1} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$
- $w_{pens,d_2} =$ not in the doc
- $w_{pens,d_3} =$ not in the doc
- $w_{write,d_1} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{write,d_2} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{write,d_3} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{on,d_1} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{on,d_2} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{on,d_3} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{paper,d_1} = (1 + \log 2) * \log \frac{3}{3} = 0$
- $w_{paper,d_2} =$ not in the doc
- $w_{paper,d_3} = (1 + \log 1) * \log \frac{3}{3} = 0$
- $w_{pencils,d_1} =$ not in the doc
- $w_{pencils,d_2} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$
- $w_{pencils,d_3} =$ not in the doc
- $w_{envelope,d_1} =$ not in the doc
- $w_{envelope,d_2} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$
- $w_{envelope,d_3} =$ not in the doc
- $w_{ballpens,d_1} =$ not in the doc
- $w_{ballpens,d_2} =$ not in the doc
- $w_{ballpens,d_3} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

Terms	d_1	d_2	d_3
pens	1.1	0.0	0.0
write	0.0	0.0	0.0
on	0.0	0.0	0.0
paper	0.0	0.0	0.0
pencils	0.0	1.1	0.0
envelope	0.0	1.1	0.0
ballpens	0.0	0.0	1.1

$\overrightarrow{d_1} = (1.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)$
$\overrightarrow{d_2} = (0.0, 0.0, 0.0, 0.0, 1.1, 1.1, 0.0)$
$\overrightarrow{d_3} = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.1)$

Subtask 2

$w_{ballpens,q} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$
$w_{envelope,q} = (1 + \log 1) * \log \frac{3}{1} \approx 1,1$

$$\overrightarrow{q} = (0.0, 0.0, 0.0, 0.0, 0.0, 1.1, 1.1)$$

$$SIM(\overrightarrow{q}, \overrightarrow{d_1}) = \frac{\overrightarrow{q} * \overrightarrow{d_1}}{|\overrightarrow{q}| * |\overrightarrow{d_1}|} = 0$$
$$SIM(\overrightarrow{q}, \overrightarrow{d_2}) = \frac{\overrightarrow{q} * \overrightarrow{d_2}}{|\overrightarrow{q}| * |\overrightarrow{d_2}|} = \frac{1.21}{49} \approx 0.025$$
$$SIM(\overrightarrow{q}, \overrightarrow{d_3}) = \frac{\overrightarrow{q} * \overrightarrow{d_3}}{|\overrightarrow{q}| * |\overrightarrow{d_3}|} = \frac{1.21}{49} \approx 0.025$$

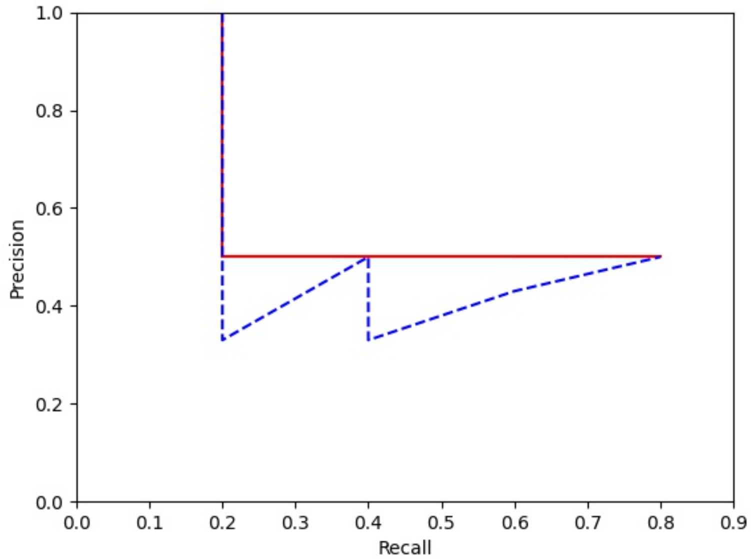
Rank	**Doc**	**SIM**
1	2	0.025
1	3	0.025
2	1	0

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

Task 3

- $Precision = \frac{TPs}{TPs+FPs}$
- $Recall = \frac{TPs}{TPs+FNs}$

k	Result Set	Precision	Recall
1	127	1.0	0.2
2	127, 9	0.5	0.2
3	127, 9, 10	0.33	0.2
4	127, 9, 10, 2	0.5	0.4
5	127, 9, 10, 2, 35	0.4	0.4
6	127, 9, 10, 2, 35, 32	0.33	0.4
7	127, 9, 10, 2, 35, 32, 41	0.43	0.6
8	127, 9, 10, 2, 35, 32, 41, 64	0.5	0.8



Task 4

An advantage of using IDF over stop word list is that you don't have a fixed set of words that maybe must be expanded in the future. Expanding a stop word list will change all the weights as a consequence. Due to the Zipf's law if a term is contained in all the documents is most likely to be a stop word. If the IDF value of a term is 0(occurs in all documents) than will have the same effects as if it was included in the stop word lists.
Advantage of using stops words lists are that we could prevent the calculation of the IDF for stops and we can be more precise not including words that maybe occurs in every document but aren't stops.

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

Programming Task

Subtask 1

The ranking method implemented is based on tf-matching-score with log frequency weighting. Given a query the method **tf_matching_scores** will output a dictionary with the tf-matching-score for all relevant documents. A *relevant* document contains at least one term of the query at least one time.

In the implementation we iterate through the documents to calculate the log frequency weights for each term in the document ($W_{t,d}$) and then we calculate the tf-matching-score for each document. We use a log tf because the relevance does not increase proportionally with term frequency. After that we rank the results with the method **ranking_table** which ranks the results and print a ranking table to the terminal.

Code

```python
#...

def tf_matching_scores(self, query: str) ->dict:
"""Calculate tf-matching-score for all documents given the query.

Args:
    query (str): Query against which we calculate the document's score

Returns:
    dict: For each document the tf-matching-score
"""
query = query.lower()
query = query.split()

docs_weights = {}
tf_matching_scores = {}

# retrive the news title and text to calculate tf-weights
with open( self.filename, 'r' ) as file:
    reader = csv.reader(file, delimiter = '\t')

    #iterate through each row of the table
    for row in reader:
        # skip table header
        if( row[0] == 'id' ): continue

        #(doc_id, url, pub_date, title, news_text) = row
        doc_ID = int(row[0])
        news_title = row[-2]
        news_text = row[-1]

        docs_weights.update({doc_ID: []})

        tokenizer = nltk.RegexpTokenizer(r"\w+")
        normalized_news_title = tokenizer.tokenize(news_title.lower())
        normalized_news_text = tokenizer.tokenize(news_text.lower())
```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

```

            news_terms = normalized_news_title + normalized_news_text

            for term in query:

                if( term in news_terms ):
                    log_tf = math.log(dict(Counter(news_terms))[term])
                    docs_weights[doc_ID].append(1 + log_tf)
                else:
                    docs_weights[doc_ID].append(0)

            # delete all documents that are not relevant to the query
            if( docs_weights[doc_ID] == [0,0] ):
                docs_weights.pop(doc_ID)

            #calculate tf-matching-scores
            for doc_ID in docs_weights:
                tf_matching_scores.update( {doc_ID: sum(docs_weights[doc_ID])} )

    return tf_matching_scores

def ranking_table(self, scores: dict):
    """Print a ranking table given a dictionary of scores

    Args:
        scores (dict): Datastructure that contains all scores
    """
    sorted_dict = sorted(scores.items(), key=itemgetter(1), reverse=True)
    ranked_list = []
    rank = 0
    old_score = 0

    for items in sorted_dict:
        doc_ID = items[0]
        score = items[-1]

        # don't rank if the score is zero
        if( score == 0 ): continue

        if( old_score == score):
            ranked_list.append((rank, doc_ID, score))
        else:
            rank += 1
            ranked_list.append((rank, doc_ID, score))
            old_score = score

    # print ranking table
    #print only top 10
    print("Rank \t", "Doc \t", "Score")
    print("-"*30)
    counter = 0
    for row in ranked_list:
        if ( counter > 9 ): break
        rank, doc_ID, score = row
        print(rank, "\t", doc_ID, "\t", round(score, 5))
        counter += 1
```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

```python



if __name__ == "__main__":
filename = 'assignment3/code/postillon.csv'
search = Search(filename=filename)

print("\n"*3)
print("1. Query: olympische sportbund")
scores = search.tf_matching_scores("olympische sportbund")
search.ranking_table(scores)

print("\n"*3)
print("2. Query: rot wein")
scores = search.tf_matching_scores("rot wein")
search.ranking_table(scores)

print("\n"*3)
print("3. Query: kinder sind faul")
scores = search.tf_matching_scores("kinder sind faul")
search.ranking_table(scores)
```

OUTPUT:

```
    1. Query: olympische sportbund
Rank      Doc      Score
------------------------------
1         4194     2.69315
2         276      2.60944
3         1632     2.09861
4         3925     2.0
4         4759     2.0
5         2018     1.69315
5         2146     1.69315
5         4990     1.69315
6         228      1.0
6         548      1.0




2. Query: rot wein
Rank      Doc      Score
------------------------------
1         165      2.38629
2         1063     1.69315
2         1210     1.69315
2         1872     1.69315
2         2212     1.69315
2         2332     1.69315
2         2837     1.69315
2         2978     1.69315
```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - st145113@stud.uni-stuttgart.de

```
2          3712      1.69315
3          150       1.0




3. Query: kinder sind faul
Rank       Doc       Score
------------------------------
1          3763      5.4012
2          2789      5.07944
3          1827      4.89037
4          583       4.77259
5          4228      4.60944
6          1494      4.19722
6          2658      4.19722
7          2011      4.07944
8          150       3.94591
8          4887      3.94591
```