Information Retrieval Assignment 1

October 18, 2021

1 Alexey Shapovalov (20235952)

2 Question 1

- The vector space model can be used to rank D1, D2, D3 with respect to Q.
- Each document and the query need to be vectorized. This can be done using a td-idf based weighting scheme:
  - The documents and query are preprocessed, two examples: removing stopwords, stemming.
  - For each document and query a vector of size `unique_words` is created. `unique_words` are all the unqiue words that appear across all the documents and the query.
  - Each element in the vector is created by a tf-idf calculation.
  - The tf calculation is based on the local term frequency. This calculation can be normalized to penalize longer documents.. the longer a document is the more likely it will have relevant terms.
  - The idf calculation is based on the global term frequency, e.g. terms that appear in all documents are likely less relevant.
- Once each document and the query are vectorized using a tf-idf calculation, the cosine similarity between the query and document vectors can be used as a measure to rank all of the documents i.e. the documents are sorted by the cosine similarity of the vectors.

[1]:
```python
# Note this code is not bug free, it is only designed
# for explanatory purposes and will only work simple examples.
import math
from collections import Counter
from collections import defaultdict
from nltk.corpus import stopwords


def frequencies(ds):
    # Calculate word frequencies
    word_freqs = Counter()
    for d in ds:
        word_freqs.update(d)

    # Calculate document frequencies
    doc_freqs = defaultdict(int)
    for d in ds:
        for w in word_freqs:
            if w in d:
                doc_freqs[w] += 1

    return word_freqs, doc_freqs


def preprocess(sent):
    # Simple removal of stopwords
    return [
        w for w in sent.lower().split(" ")
        if w not in stopwords.words('english')
    ]


def tfidf(t, N, word_freqs, doc_freqs, alpha=0.5):
    # Most occurring word frequency
    tfmax = word_freqs.most_common(1)[0][1]

    # Maximum term normalized tf calculation
    tf = alpha + ((1 - alpha) * (word_freqs[t] / tfmax))

    # Log based idf calculation
    idf = math.log(N / doc_freqs[t])

    return tf * idf


def vectorize(sent, num_docs, word_freqs, doc_freqs):
    # Tf-idf based vectorization
    return [
        tfidf(t, num_docs, word_freqs, doc_freqs)
        if t in sent else 0 for t in word_freqs.keys()
    ]


def cosine_similarity(a, b):
    """
    Cosing similarity between vectors a and b

    @param a: list of number - first vector
    @param q: list of number - second vector
    """

    a_length, b_length, ab = 0, 0, 0
    for i in range(len(a)):
        a_i = a[i]
        b_i = b[i]
        a_length += a_i * a_i
        b_length += b_i * b_i
        ab += a_i * b_i
    return ab / (math.sqrt(a_length) * math.sqrt(b_length))


def rank(ds, q):
    """
    Ranks each sentence in ds with respect to q

    @param ds: list of string - the documents
    @param q: string - the query

    """

    word_freqs, doc_freqs = frequencies(ds)
    num_docs = len(ds)
    q = vectorize(q, num_docs, word_freqs, doc_freqs)
    ds = [
        (d, cosine_similarity(q, vectorize(d, num_docs, word_freqs, doc_freqs)))
        for d in ds
    ]
    return sorted(ds, key=lambda x: x[1], reverse=True)



D1 = "Shipment of gold damaged in a fire"
D2 = "Delivery of silver arrived in a silver truck"
D3 = "Shipment of gold arrived in a truck"

Q = "gold silver truck"

ranked = rank([D1, D2, D3], Q)
print("Sentenced are ranked D2, D3, D1:")
for inx, (sent, similarity) in enumerate(ranked):
    print("{}) '{}', with similarity: {}".format(inx+1, sent, similarity))
```

```
Sentenced are ranked D2, D3, D1:
1) 'Delivery of silver arrived in a silver truck', with similarity:
0.652972569615761
2) 'Shipment of gold arrived in a truck', with similarity: 0.49105084280413136
3) 'Shipment of gold damaged in a fire', with similarity: 0.1348783507687377
```

3 Question 2

To make my answers a little easier to explain I have renamed the variables in question by appending _part1, _part2, _part3 and _part4 to the variable name based on the question part. 1. D1_part1 = Shipment of gold damaged in a fire. Fire. * In this case the expectation is that sim(Q, D1_part1) **should be lower** when compared to the original sim(Q, D1). This is based on axiom 2 that states "Adding a non-query to a document must always decrease the score of a document." Since the additional "Fire" is not part of the query term sim(Q, D1_part1) should decrease.

2. D1_part2 = Shipment of gold damaged in a fire. Fire. Fire.

- Again in this case sim(Q, D1_part2) **should be lower** when compared to the original D1. It should also be **lower than sim(Q, D__part1)** as there is now two additional non-query terms. However, according to axiom 4, the expectation is that the score should be lower than D1_part1 **by a proportion that is smaller** than the decrease of D1_part1 i.e. `(sim(Q, D1) - sim(Q, D1_part1)) > (sim(Q, D1_part1) - sim(Q, D1_part2))`

3. D1_part3 = Shipment of gold damaged in a fire. Gold.

- This time sim(Q, D1_part3) **should be higher** when compared to the original sim(Q, D1). This is based on axiom 1 that states "Adding a query term to a document must always increase the score of a document." Since "Gold" is present in the query term sim(Q, D1_part3) is expected to increase.

4. D1_part4 = Shipment of gold damaged in a fire. Gold. Gold.

- Again in this case sim(Q, D1_part4) **should be higher** when compared to the original D1. It should also be **higher than sim(Q, D__part3)** as there are now two additional terms that are present in the query. However, according to axiom 3, the expectation is that the similarity score should be higher **by a proportion that is smaller** than the increase of D1_part3 i.e. `(sim(Q, D1_part3) - sim(Q, D1)) > (sim(Q, D1_part4) - sim(Q, D1_part3))`

4 Question 3

Since the IR system in question is related to scientific articles, the features of the articles could also be encorporated into the weighting schemes. Two specific features that could be incorporated would be: 1) The amount of other research papers that cite the target paper (document), let this number be denoted by the variable `cited_by`. This is a good indicator of the quality of a paper because if the paper is not cited by any other papers (unless it is very new) than it is likely not a great paper. To encorporate this into the weighting scheme, I would **add** the following calculation to the weighting: `cited_by / max_cited_by_of_year_month`. The division by `max_cited_by_of_year_month` (max citations of any paper in the particular year and month the paper was published in) is to normalize the impact of the new feature (should always make it between 0 and 1). It should also handle the fact that the longer the time the paper is published, the higher chance there is that it will be cited.

2) The amount of other research papers that the target paper (document) references, let this number be denoted by the variable `references`. This is a good indicator of the quality of a paper because if for example, a paper references nothing than it is not well researched and as such likely contains invalid information. The more references the paper has, the more it is researched and thus the better the quality. To encorporate this into the weighting scheme, I

would **add** the following calculation to the weighting: `references / max_references`. The division by `max_references` (max references of any paper) is to normalize the impact of the new feature (should always make it between 0 and 1).