Assignment 4

Rik Vegter (s3147495)
Jordi Timmermans (s3870758)
Hari Vidharth (s4031180)
Ayça Avcı (s4505972)

Group 12

October 15, 2020

Part A

(a)

To formulate query vector $\vec{q}$, we iterated through the terms of the inverted index I. Whenever there was a match between any word from the query ("Human Computer Interaction") and the inverted index, we assigned the value 1 to the $\vec{q}$ vector at the index of the match. What we added to *create_query_vector*$(self, q)$ method can be found below:

```
for t in q:
    for index, pair in enumerate(self.I.terms):
        if pair[0] == t:
            ret_q[index] = 1.0
```

The resulting vector is req_q = [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], where the first 1 indicates computer and the second indicates human. Interaction is not part of the terms and therefore it is not added to the query vector.

To fold a our vector into a semantic space, we used the formula below:

$$NewQueryVector = \vec{q} \cdot T \cdot S^{-1} \tag{1}$$

To implement that formula, the following lines added to the *fold_in_query*$(self, q)$ method:

```
1  q_head = [0.0 for i in range(rank)]
2
3  for i in range(rank):
4      for j in range(no_terms):
5          fol_q[i] += q[j] * self.T[j][i]
6
7  for i in range(rank):
8      q_head[i] += fol_q[i] * self.S_inv[i]
9
10 return q_head
```

In first line, we created a vector called q_head consisting of 0's with a length equal to the rank of the S matrix. This will be a placeholder for our new query vector. Between line 3 and 5, we basically performed matrix multiplication to fold our query vector into semantic space ($\vec{q} \cdot T$). In lines 7 and 8, we scale our folded in vector with S, resulting in our scaled query vector *q_head*.

The folding in and scaling resulted in the new query vector:
$q\_head \approx [-0.138, 0.028, -0.053, 0.614, -0.142, -0.456, 0.261, 0.003, -0.236]$.

(b)

In order to compare the folded in query vector with the documents in semantic space we used the cosine operator:

$$cos(\Theta) = \frac{\vec{q} \cdot D_i^T}{\|\vec{q}\| * \|D_i^T\|} \tag{2}$$

Where $\vec{q}$ is the folded in query vector and D_i^T is the document vector (i^{th} column of the D^T matrix).
We first scaled the arrays with S, the diagonal matrix and we calculated the magnitude of the reduced query and document vectors (reduced to *max_dimension* indicating the maximum number of dimensions taken in to account) by the following code:

```
    m_q = np.linalg.norm(self.Dt[:, doc][0:self.max_dimension] * self.S[0:self.max_dimension])
    m_d = np.linalg.norm(q[0:self.max_dimension] * self.S[0:self.max_dimension])
```

These magnitudes are the denominator of the cosine comparison operator.
We then computed the numerator of the cosine comparison operator in a for loop using:

```
    for i in range(self.max_dimension):
        calc += q[i] * self.S_sq[i] * self.Dt[i, doc]
```

Here we implemented a shortcut by using the square of the S matrix instead of 2 times the S matrix. This can be done because the S matrix is represented as an array. When we would not have the shortcut we would use:

```
    for i in range(self.max_dimension):
        calc += q[i] * self.S[i] * self.Dt[i, doc] * self.S[i]
```

The two $self.S[i]$'s here are implemented as the square of them in the code with the shortcut.
After this we divided the numerator by the denominator (we scale our values between -1 and 1:

```
    cos_with_doc = calc / (m_q * m_d)
    return cos_with_doc
```

With this code we compared "Human Computer Interaction" with every document. This resulted in the following similarity values:
C1: 0.998
C2: 0.937
C3: 0.998
C4: 0.987
C5: 0.908
M1: -0.124
M2: -0.106
M3: -0.099
M4: 0.050

(c)

In order to compare how similar the terms are, we compared the reduced (the same way as in b) terms with every other term. Just as in b, we computed the magnitudes of the vectors (here $self.T[t1 :]$ and $self.T[t1 :]$ are the 2 term vectors we compare):

```
    m_t1 = np.linalg.norm(self.T[t1, :][0:self.max_dimension] * self.S[0:self.max_dimension])
    m_t2 = np.linalg.norm(self.T[t2, :][0:self.max_dimension] * self.S[0:self.max_dimension])
```

We then calculate the numerator of the cosine operator using the same shortcut (S^2 instead of multiplying by S twice):

```
    for i in range(self.max_dimension):
        calc += self.T[t1, i] * self.S_sq[i] * self.T[t2, i]
```

We then divide the numerator by the denominator:

```
    cos_with_terms = calc / (m_t1 * m_t2)
    return cos_with_terms
```

This code resulted in the similarity measures in appendix A.
We can see that in (b), the documents starting with C result in the highest similarities. C1 for example consists of "human interface computer", which contains words from the query and thus has high similarity. We can also see that C3 for example contains "EPS user interface system". This contains no word from the query. In the term-term comparison, we can see that EPS and interface are quite similar to the computer. We can also see that human is quite similar to eps, user, interface and system. This term similarity results in a document similarity as well.
We can also see that trees, graph and minors are "separated" from the rest of the terms similarity wise, resulting in a low similarity value for the documents M1, M2, M3 (since these terms occur in these documents). M4 contains "graph minors survey" where the survey is a little bit similar to human and quite similar to a computer, resulting in a little bit of a higher similarity compared to the rest of the M documents.

Part B - Term Weightings

The data in Figure 1 represents the term by document matrix for our dataset, each entry in the matrix corresponds to the number of times a term appears in a document.

$$\begin{pmatrix}
Term\backslash Document & c1 & c2 & c3 & c4 & c5 & m1 & m2 & m3 & m4 \\
computer & 1.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.0 \\
eps & 0.00 & 0.00 & 1.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
graph & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 \\
human & 1.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
interface & 1.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
minors & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 1.00 \\
response & 0.00 & 1.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
survey & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\
system & 0.00 & 1.00 & 1.00 & 2.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
time & 0.00 & 1.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
trees & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 1.00 & 1.00 & 0.00 \\
user & 0.00 & 1.00 & 1.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00
\end{pmatrix}$$

Figure 1: Term by Document Matrix, counting the number of term occurrences in a document

TF-IDF

In this part of the assignment Term-Frequency - Inverse Document Frequency is implemented. This method can calculate how important a word is inside of a document relative to the same word in other documents. This method is meant to make words that occur very often more proportional. One of the downsides of using the TF-IDF weighting schema is that text that is very long will become more relevant. The reason behind this is that the IDF-TF model makes terms that occur very often relevant, even though it increases proportionally. To apply the term-frequency weighting we simply follow the formula described in the assignment. Our implementation uses a simple loop over the entire matrix. Per element in the matrix, we multiply the element by the natural logarithm of the number of documents divided by the number of documents the word occurs in. The answer becomes the new element in the transformed matrix. The results of the transformed matrix can be seen in Figure 2. This function is implemented in *tf_weighting.py* which is on github.

$Term \backslash Document$	$c1$	$c2$	$c3$	$c4$	$c5$	$m1$	$m2$	$m3$	$m4$
$computer$	1.5041	1.5041	0.00	0.00	0.00	0.00	0.00	0.00	0.0
eps	0.00	0.00	1.5041	1.5041	0.00	0.00	0.00	0.00	0.00
$graph$	0.00	0.00	0.00	0.00	0.00	0.00	1.0986	1.0986	1.0986
$human$	1.5041	0.00	0.00	1.5041	0.00	0.00	0.00	0.00	0.00
$interface$	1.5041	0.00	1.5041	0.00	0.00	0.00	0.00	0.00	0.00
$minors$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.5041	1.5041
$response$	0.00	1.5041	0.00	0.00	1.5041	0.00	0.00	0.00	0.00
$survey$	0.00	1.5041	0.00	0.00	0.00	0.00	0.00	0.00	1.5041
$system$	0.00	1.0986	1.0986	2.1972	0.00	0.00	0.00	0.00	0.00
$time$	0.00	1.5041	0.00	0.00	1.5041	0.00	0.00	0.00	0.00
$trees$	0.00	0.00	0.00	0.00	0.00	1.0986	1.0986	1.0986	0.00
$user$	0.00	1.0986	1.0986	0.00	1.0986	0.00	0.00	0.00	0.00

Figure 2: TF-IDF transformation model

Log-Entropy

Another weighting method is Log-Entropy weighting. This scheme is different from TF-IDF in a sense that it takes the distribution of terms into account. This means that words that occur very often in a text will get less weight when using LE weighting than with TF-IDF. When applying the Log-Entropy model on the same matrix as we did with the TF-IDF model we get the transformed matrix shown in Figure 3
. The code for this function is in *le_weighting.py*.

$Term \backslash Document$	$c1$	$c2$	$c3$	$c4$	$c5$	$m1$	$m2$	$m3$	$m4$
$computer$	0.4744	0.4744	0.00	0.00	0.00	0.00	0.00	0.00	0.0
eps	0.00	0.00	0.4744	0.4744	0.00	0.00	0.00	0.00	0.00
$graph$	0.00	0.00	0.00	0.00	0.00	0.00	0.3465	0.3465	0.3465
$human$	0.4744	0.00	0.00	0.4744	0.00	0.00	0.00	0.00	0.00
$interface$	0.4744	0.00	0.4744	0.00	0.00	0.00	0.00	0.00	0.00
$minors$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.4744	0.4744
$response$	0.00	0.4744	0.00	0.00	0.4744	0.00	0.00	0.00	0.00
$survey$	0.00	0.4744	0.00	0.00	0.00	0.00	0.00	0.00	0.4744
$system$	0.00	0.3651	0.3651	0.5787	0.00	0.00	0.00	0.00	0.00
$time$	0.00	0.4744	0.00	0.00	0.4744	0.00	0.00	0.00	0.00
$trees$	0.00	0.00	0.00	0.00	0.00	0.3466	0.3466	0.3466	0.00
$user$	0.00	0.3466	0.3466	0.00	0.3466	0.00	0.00	0.00	0.00

Figure 3: Log-Entropy transformation model

A Appendix: Term Similarity Measures

Similarity with computer
computer: 1.000
eps: 0.888
graph: 0.210
human: 0.874
interface: 0.919
minors: 0.226
response: 0.987
survey: 0.793
system: 0.946
time: 0.987
trees: 0.169

user: 1.000

Similarity with eps
computer: 0.888
eps: 1.000
graph: -0.264
human: 1.000
interface: 0.997
minors: -0.248
response: 0.801
survey: 0.423
system: 0.989
time: 0.801
trees: -0.304
user: 0.900

Similarity with graph
computer: 0.210
eps: -0.264
graph: 1.000
human: -0.291
interface: -0.193
minors: 1.000
response: 0.366
survey: 0.762
system: -0.119
time: 0.366
trees: 0.999
user: 0.182

Similarity with human
computer: 0.874
eps: 1.000
graph: -0.291
human: 1.000
interface: 0.995
minors: -0.275
response: 0.784
survey: 0.398
system: 0.985
time: 0.784
trees: -0.330
user: 0.888

Similarity with interface
computer: 0.919
eps: 0.997
graph: -0.193
human: 0.995
interface: 1.000
minors: -0.177
response: 0.842
survey: 0.488
system: 0.997
time: 0.842
trees: -0.234
user: 0.929

Similarity with minors

computer: 0.226
eps: -0.248
graph: 1.000
human: -0.275
interface: -0.177
minors: 1.000
response: 0.381
survey: 0.773
system: -0.102
time: 0.381
trees: 0.998
user: 0.198

Similarity with response
computer: 0.987
eps: 0.801
graph: 0.366
human: 0.784
interface: 0.842
minors: 0.381
response: 1.000
survey: 0.881
system: 0.881
time: 1.000
trees: 0.326
user: 0.982

Similarity with survey
computer: 0.793
eps: 0.423
graph: 0.762
human: 0.398
interface: 0.488
minors: 0.773
response: 0.881
survey: 1.000
system: 0.552
time: 0.881
trees: 0.735
user: 0.775

Similarity with system
computer: 0.946
eps: 0.989
graph: -0.119
human: 0.985
interface: 0.997
minors: -0.102
response: 0.881
survey: 0.552
system: 1.000
time: 0.881
trees: -0.160
user: 0.955

Similarity with time
computer: 0.987
eps: 0.801
graph: 0.366

human: 0.784
interface: 0.842
minors: 0.381
response: 1.000
survey: 0.881
system: 0.881
time: 1.000
trees: 0.326
user: 0.982

Similarity with trees
computer: 0.169
eps: -0.304
graph: 0.999
human: -0.330
interface: -0.234
minors: 0.998
response: 0.326
survey: 0.735
system: -0.160
time: 0.326
trees: 1.000
user: 0.141

Similarity with user
computer: 1.000
eps: 0.900
graph: 0.182
human: 0.888
interface: 0.929
minors: 0.198
response: 0.982
survey: 0.775
system: 0.955
time: 0.982
trees: 0.141
user: 1.000