Problem 1

We propose the following hands-on exercise in order to help you gain a better feeling for the problems that normalization tries to solve, and to see how different possible normalization functions tackle these problems. As discussed in the lecture notes, when using the un-normalized matching function there is a concern that longer documents have an unfair advantage over the shorter ones. This concern is raised by the fact that longer documents will have:

(a) bigger term frequencies

(b) more non-zero term frequencies

Study the documents D1–D6 and the query q below.

D1	milk and cat
D2	cat and cat cat milk cat milk milk cat milk cat (= 6×cat 4×milk 1×and)
D3	potatoes 3×oil 3×tomatoes 3×bread cat milk
D4	and oil and potatoes and tomatoes and bread and milk
D5	7×cat oil bread and
D6	7×cat 7×oil 7×bread 7×and
q	milk and cat

For the purposes of this exercise, we assume that documents D1–D6 were extracted from a corpus in which the inverse document frequency of each term v_j in D1–D6 is:

term (v_j)	cat	milk	and	potatoes	oil	tomatoes	bread
idf_j	1	1	0.1	1	1	1	1

I. Estimate the ranking of the documents based on relative relevance to the query q of the documents based on your intuition. Which documents are susceptible to exhibiting which of the "unfair" biases (a) and (b) with respect to query q?

II. Let q[j] be the number of occurrences of the term v_j in the query q. Define the "raw-tf-idf" function of document d, denoted $\vec{rti}_d$, to be the vector such that $\vec{rti}_d[j] = (\cdots, tf_d(j) * idf_j, \cdots)^T$, where $tf_d(j) = \#$ occurrences of term v_j in d, and use this to define the un-normalized match function as follows:

$$match_q(d) = \sum_{j=1}^{m} q[j] * rti_d[j] \tag{8}$$

Calculate this match function for all documents and compare the numerical results with your estimated ranking.

III. Now we try to compensate for the aforementioned biases. Since these biases favor longer documents, a penalty based on the length of each document seems like a reasonable attempt. Define the normalization function $norm(d)$ to be the number of tokens (i.e number of words, distinct or not) in d and compute the corresponding matching function:

$$match_q^{\#tokens}(d) = \frac{\sum_{j=1}^{m} q[j] * rti_d(j)}{norm_{\#tokens}(d)} \tag{9}$$

What criticisms can you make of this simplistic approach? Try defining $norm_{\#types}(d)$ to be the number of types (distinct terms) in d. Compute the corresponding match function $match_q^{\#types}(d)$ and discuss any relevant differences.

IV. Consider another alternative to the norm function:

$$norm_{maxtf}(d) = \max_j rti_d(j) \tag{10}$$

Which of the biases does this norm function ameliorate? Moreover, does this function discriminate between D5 and D6? Explain why.

We claim that D5 is much more relevant than D6: while D5 focuses on "cat" and just mentions "oil" and "bread", D6 is a more general document, without a special focus on "cat". Define another (simple) norm function that attacks the same bias as (10) but clearly discriminates between D5 and D6.

V. Next, try the classic L_2 normalization (also called cosine normalization):

$$norm(d) = \sqrt{\sum_{j=1}^{m} \vec{rti_d}^{\,2}} \tag{11}$$

Convince yourself that it does in fact ameliorate both (a) and (b).

There is however a relatively deeper observation to be made. We claim that D1 and D2 will obtain the best scores out of all the documents. While this might seem reasonable at first glance, is this what we really want when we submit a query to, let's say, a search engine? To retrieve the query itself or the document that just repeats the query? We know the query! Explain formally why L_2 has this behavior.

Solutions:

I. D1 and D2 are of about the same relevance – both of them contain all three query terms and no others. D5 comes next because it talks a lot about cats, but no milk, and has a few other random terms. D6 comes next, because it talks a lot about cats, but equally lots about oil, bread etc. Then comes D4, which has one occurrence of milk, and doesn't have that many other words. Last is D3, which seems to be talking chiefly about other things and just casually mentions cats and milk.

The relative relevance of the documents thus seems to be D1≈D2>D5>D6>D4>D3.

We would expect D3 to exhibit bias (b), and D4, D6 to exhibit some level of bias (a).

II. The ranking by relevance (as measured by the match function in equation (8)) here is D2>D6>D5>D1>D3>D4. This is very different from our estimated (intuitive) relevance ranking. Documents with a higher number of words such as D2, D6 and D5 rank higher, whereas relevant but short document D1 ranks rather low. Indeed, D3 exhibits bias (b), its matching function being almost equal to D1. D6 clearly exhibits bias (a). D4 exhibits (a) too, albeit less obviously.

d	term frequencies (tf_d)							$\overrightarrow{rti_d}$	$match_q(d)$
	cat	*milk*	*and*	*potatoes*	*oil*	*tomatoes*	*bread*		
D1	1	1	1	0	0	0	0	(1,1,0.1,0,0,0,0)	2.1
D2	6	4	1	0	0	0	0	(6,4,0.1,0,0,0,0)	10.1
D3	1	1	0	1	3	3	3	(1,1,0,1,3,3,3)	2.0
D4	0	1	5	1	1	1	1	(0,1,0.5,1,1,1,1)	1.5
D5	7	0	1	0	1	0	1	(7,0,0.1,0,1,0,1)	7.1
D6	7	0	7	0	7	0	7	(7,0,0.7,0,7,0,7)	7.7
q	1	1	1	0	0	0	0	NA	NA

III. The ranking for $match_q^{\#tokens}$ is D2>D4>D5>D1>D6>D3. This approach does not solve (b) above – it does not account for the fact that longer documents have more non-zero term frequencies. This is seen by the fact that D4 ranks higher than D6 and almost as high as D1, although D6 is more relevant (has more occurrences of query terms). This function therefore seems to over-penalize long documents.

d	$norm_{\#tokens}(d)$	$match_q^{\#tokens}(d)$	$norm_{\#types}(d)$	$match_q^{\#types}(d)$
D1	3	0.70	3	0.70
D2	11	0.92	3	3.37
D3	12	0.17	6	0.33
D4	10	0.15	6	0.25
D5	10	0.71	4	1.77
D6	28	0.28	4	1.92

There is another problem with this normalization: a document consisting of just one query term would get a maximum matching value, which is counter-intuitive. This problem is inherited from L_1 normalization, of which the above is just a simplification (we just discard idf_j).

The ranking for $match_q^{\#types}$ is D2>D6>D5>D1>D3>D4. Here we do account for (b), but not for (a), as seen by the fact that D6 (long) ranks higher than D1 (short).

IV. The ranking for $match_q^{maxtf}(d)$ is D1>D2>D4>D6>D5>D3. It compensates somewhat for bias (a), i.e. higher term frequencies in longer documents.

d	$norm_{maxtf}(d)$	$match_q^{maxtf}(d)$
D1	1	2.10
D2	6	1.68
D3	3	0.66
D4	1	1.50
D5	7	1.01
D6	7	1.10

D6 and D5, which contain the same terms (types), are ranked about the same, because they have the same *maxtf* although the number of terms with that *maxtf* is different. D6 is a lot less specific to the query, since it contains high frequencies for all terms whereas the frequency of "cat" in D5 is significantly higher than that of the other terms. Therefore $match_q^{maxtf}(d)$ is not an optimal normalization scheme to fix bias (a).

Define

$$norm_{avgtf}(d) = \sum_{j=1}^{m} \frac{rti_d(j)}{\mathrm{m}} \tag{12}$$

where m is the size of our vocabulary (in our case $m = 7$) and use this instead.

d	$norm_{avgtf}(d)$	$match_q^{avgtf}(d)$
D1	0.30	7.00
D2	1.44	7.01
D3	1.71	1.17
D4	0.79	1.90
D5	1.30	5.46
D6	3.10	2.48

The ranking for $match_q^{avgtf}(d)$ is D2>D1>D5>D6>D4>D3. Therefore $norm_{avgtf}(d)$ overcomes the problem identified above, and ranks D5 above D6, with a significantly higher value of $match_q^{avgtf}(d)$.

V. The ranking for $match_q^{L2}(d)$ is D1>D2>D5>D4>D6>D3. This is almost what we had predicted – as expected, L_2 appears to account for both (a) and (b).

d	$norm_{L2}(d)$	$match_q^{L2}(d)$
D1	1.41	1.49
D2	7.21	1.40
D3	5.47	0.37
D4	2.29	0.66
D5	7.14	0.99
D6	12.14	0.63

As regards the additional observation, an explanation can be found in the geometric interpretation of the L_2-norm (Fig. 5 in the lecture notes): the document with the best match has document vector $\vec{d}$ oriented in the same direction as the query vector $\vec{q}$, with its L_2-normalized vector $\vec{d}^*$ being exactly the query.

Problem 2

Now let's try, as a thought exercise, to consider the problem from a different direction. Normalization was motivated by the bias that the length of the document introduces in the retrieval process; from this perspective, an ideal world would be a world in which all documents are of the same size. So why not try to bring all documents to the same size, without distorting our intuition about the relative relevance of the documents.

- One approach would be to extract for each document a probability distribution over terms and to use that distribution to "complete" the document by generating as many new words as needed to reach the maximum length in the collection. Implement this, apply it to D1–D6 and discuss the results of the un-normalized matching function (8). How did this method perform on problems (a) and (b)?
- Propose a way of improving this approach, with respect to our original bias problems.

Solution:

The longest document in our collection has 28 words, so the generated documents D1′–D6′ (derived from D1–D6) must all have 28 words. The following is the MATLAB code used to "grow" the documents:

```
function newWords=generate(maxLength,docLength,distrVector,nrTerms)
% output: newWords is a vector of length nr of terms, indicating
%           which terms have to be added to the document and how
%           many times in order to 'complete' the document
% input: maxLength is the maximum document length in the
%           collection
%        docLength is the length of the current document
%        distrVector is a vector containing the probability
%           distribution of terms
%        nrTerms is the number of distinct terms in the corpus


%initialization
newWords=zeros(1,nrTerms);
r=rand(1,maxLength-docLength);
distrInterval=zeros(1,nrTerms+1);

%divide the 0,1 interval according to the given distribution
for j=1:nrTerms-1
    distrInterval(j+1)=distrVector(j)+distrInterval(j);
end;

%sample from that distribution
```

```
for  i=1:maxLength-docLength
     [dummy,indV]=find(distrInterval>r(i));
     ind=min(indV);
     newWords(ind-1)=newWords(ind-1)+1;
end;
```

The table below shows the number of additional terms to be added to each document, as generated using this code.

d	terms to be added						
	cat	*milk*	*and*	*potatoes*	*oil*	*tomatoes*	*bread*
D1	13	5	7	0	0	0	0
D2	12	4	1	0	0	0	0
D3	0	3	0	1	1	4	0
D4	0	2	11	2	3	0	0
D5	15	0	2	0	0	0	0
D6	0	0	0	0	0	0	0

By adding the extra terms, we get documents D1′–D6′ as below:

d	term frequencies (tf_d)							$\overrightarrow{rti_d}$	$match_q(d)$
	cat	*milk*	*and*	*potatoes*	*oil*	*tomatoes*	*bread*		
D1′	14	6	8	0	0	0	0	(14,6,0.8,0,0,0,0)	20.8
D2′	18	8	2	0	0	0	0	(18,8,0.2,0,0,0,0)	26.2
D3′	1	4	0	2	4	7	3	(1,4,0,2,4,7,3)	5.0
D4′	0	3	16	3	4	1	1	(0,3,1.6,3,4,1,1)	4.6
D5′	22	0	3	0	1	0	1	(22,0,0.3,0,1,0,1)	22.3
D6′	7	0	7	0	7	0	7	(7,0,0.7,0,7,0,7)	7.7

The ranking for $match_q(d)$ is D2′ >D5′ >D1′ >D6′ >D3′ >D4′. We observe that the longer documents D3, D4 and D6, which we predicted would cause a bias (of either type) receive low scores.

Note that our approach does not tackle (b) either – a term that does not appear in the original document will not appear in the extended document. But this can be remedied by altering the

distribution of terms in order to allow this to happen. One way of doing so is to introduce a certain degree of noise into the distribution of terms. A more informed method would be to add new (missing) terms to the documents based on their distribution in the corpus.

Unfortunately, one major drawback of this approach is that growing all the documents in the corpus to the maximum document length is very expensive.