**CS 674: Advanced Language Technologies** Fall 2007

# Lecture 2 — August 31, 2007

*Prof. Lillian Lee* Scribes: *Cristian Danescu Niculescu-Mizil*
*Vasumathi Raman*

## 1 Overview

We have already introduced the **classic model of ad-hoc information retrieval**, in which the goal is to ideally rank (some) documents in the corpus or document collection $C$ by relevance to the query $q$ expressing the user's information needs. We discussed per-query quality measures including **precision-at-k** and **average precision**, and introduced the **Vector Space Model** (VSM), wherein we encountered the **term-weighting problem**. As three points of consensus on this problem, we defined **inverse document frequency** ($idf(j)$) and **term frequency** ($tf_d(j)$), and stated that some **normalization** $norm(d)$ is necessary.

Now we deal with normalization and why it matters.

## 2 Review

### 2.1 Ad Hoc IR

We are working within an online, "Google$^{\text{TM}}$-esque" or ad-hoc (from Latin, "for this purpose") retrieval system, where we rank $d \in C$ by relevance to $q$, given any query $q$. This is not a learning problem (as standardly construed, i.e. in the supervised learning sense). In general (although there are exceptions), there are no labeled (relevant vs. irrelevant) training examples for the $q$ at hand, unlike in standard supervised learning problems.

*Aside:* The field of information retrieval (IR) has tended to be much less concerned about overfitting a particular data set or corpus. Rather, we take a single model and apply it to different corpora, so the parameter values are not necessarily expected to generalize from one corpus to the next. This differs from the "usual" formulation of supervised machine learning (ML), where one expects to generalize observations and specific parameter settings from a training corpus and modify an initial model. For example, Google$^{\text{TM}}$ has only one dataset (the Web), so overfitting to the corpus is not an issue (although it may still matter when making generalizations on the internal structure of the corpus).

### 2.2 Term-Weighting Consensus

Recall that the **term weight** $d[j] \approx \frac{tf_d(j) * idf(j)}{norm(d)}$ measures how much term $v_j$ is a good characterizer of document $d$. We now claim that $norm(d)$ is more interesting than might be apparent.
We will work towards Pivoted Document-Length Normalization, as proposed by Singhal, Buckley and Mitra [SBM96]. The authors showed, to the surprise of some, that normalization matters

significantly. This paper is a nice example of empirically-driven research, as there is no particular theoretic approach that they took to achieve their results.

## 3 Why We Need Normalization

Recall the **match function** (with vector representation of $q$ and $d$):

$$match\_function_q(d) = \vec{q} \cdot \vec{d} = \sum_{j=1}^{m} q[j] * d[j] = \sum_{j=1}^{m} q[j] * \frac{tf_d(j) * idf(j)}{norm(d)} \quad (1)$$

Document length normalization of term weights is used to remove the advantage that long documents have in retrieval over the short documents. Two main reasons that necessitate the use of normalization in term weights are:

(a) **Higher term frequencies ($tf_d$)**– Long documents usually use the same terms repeatedly. As a result, the term frequency factors may be large for long documents, increasing query-document similarity as measured by our match function above. But it is possible for longer documents to have more occurrences of query terms than shorter ones without being more relevant. For example, the Encyclopaedia Britannica probably contains more occurrences of the word "car" than most auto-manuals, but we would all agree that the latter are more relevant to a query about cars.

(b) **More Non-Zero Term Frequencies** – Long documents also have numerous different terms. This increases the number of matches between a query and a long document, again increasing the query-document similarity as measured by the inner product, and the probability of retrieval of long documents over shorter ones. Short documents are also biased against if $q$ is too long or too specific, since they may not contain all the words in the query. An extreme example would be a relevant document that is shorter than the query itself.

The above two factors seem to imply that long documents have an unfair advantage. Document-length normalization is a way of penalizing the term weights for a document in accordance with its length.

*Aside:* A completely different approach involves defining the term frequency factor $tf_d$ to compensate for the above trends. Singhal et al [SAB+99] used a logarithmic function to "squash" the $tf_d$s. Conceptually, the modified term frequency factor here was first defined as ln(# of occurrences of $v_j \in d) + 1$, with a special case to handle zero occurrences namely, $tf_d(j)$ is defined as 0 if $\#(v_j \in d) = 0$. As can be seen in Fig. 1, this causes differences in the number of term occurrences to matter more when the counts are low than when they are high. The addition of 1 to the ln function distinguishes a 0-count from a 1-count. Further "squashing" is possible by using ln(ln(# of occurrences of $v_j \in d) + 1) + 1$. The addition of 1 here again distinguishes a 0-count from a 1-count. The effect can again be seen in Fig. 1.

Using such a log function to compress term frequencies compensates only for (a). We still need normalization to deal with (b).

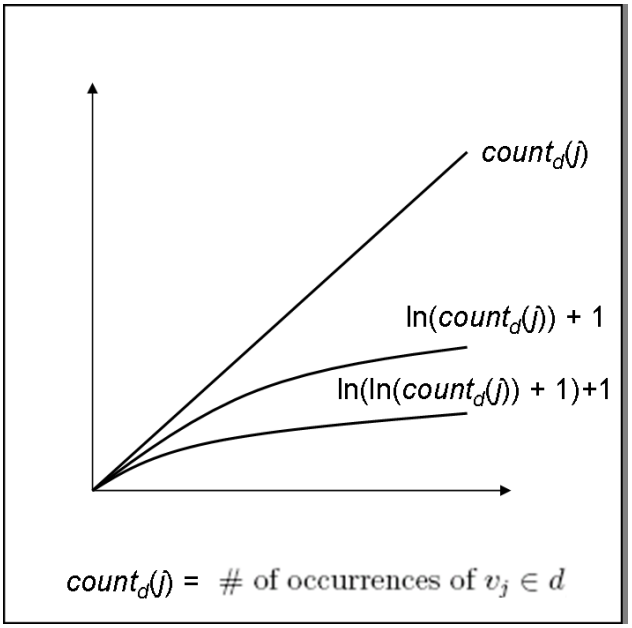


Figure 1: "Squashing" the original term frequencies [SAB+99]

### 3.1 "Contrarian" Argument

It is important to point out that one might argue that the choice of $norm(d)$ is irrelevant for a different, possibly better match function. In particular, let the match function $match\_function_q(d) = \cos(\angle \overrightarrow{q}, \overrightarrow{d})$.

To aid our discussion, define the "raw-tf-idf" (non-normalized) vector $\overrightarrow{rtf}_d$ as follows:

$$\overrightarrow{rtf}_d = \begin{pmatrix} . \\ . \\ tf_d(j) * idf(j) \\ . \\ . \\ . \end{pmatrix}$$

Then, for non-degenerate $d$, all we need to measure the match is the angle $\theta$ between $\overrightarrow{q}$ and $\overrightarrow{rtf}$, as shown in Fig. 2. This is because vectors of any length with the same orientation will have the same match function, precluding any necessity for normalization.

Why, then, is normalization important? As showed empirically by [SBM96], normalization matters and can help us improve results. So we should use a match function sensitive to its effects (i.e. the inner product version).

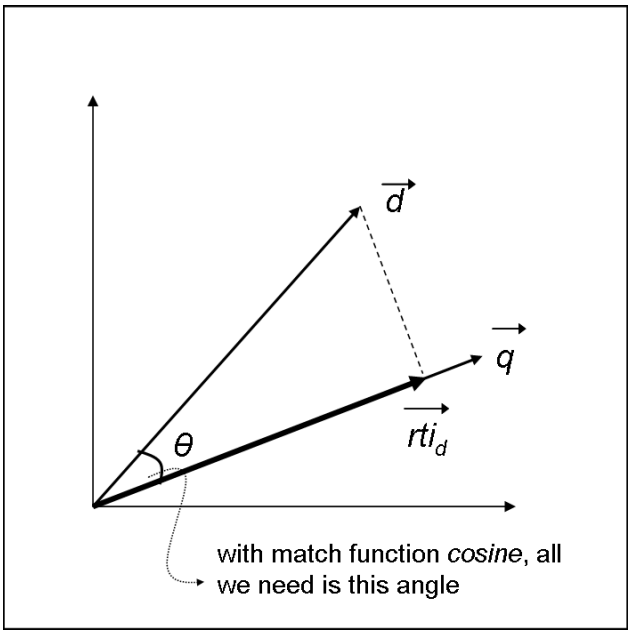


Figure 2: Contra-argument to normalization

# 4 Options for Normalization

With this motivation in mind, we reconsider the matching function mentioned above:

$$match\_function_q(d) = \vec{q} \cdot \vec{d} = \sum_{j=1}^{m} q[j] * \frac{tf_d(j) * idf(j)}{norm(d)} \tag{2}$$

and try to define a suitable normalization function $norm(d)$ that will abate concerns about the biases introduced by document length (from Section 3).

For the purposes of analyses to follow, we briefly discuss here the geometric interpretation of the match function. The inner product between the query vector and the document vector can be rewritten as:

$$\vec{q} \cdot \vec{d} = \|\vec{q}\|_2 \, proj(\vec{d}, \vec{q}) \tag{3}$$

where $proj(\vec{d}, \vec{q})$ is the signed length of the projection of $\vec{d}$ on $\vec{q}$.

Given a fixed query $q$, this is equal under ranking to $proj(\vec{d}, \vec{q})$. Therefore, the the matching value of a document $d$ with respect to $q$ is directly proportional to the length of the projection of the document vector $\vec{d}$ on the query vector $\vec{q}$, as illustrated in Fig. 3.

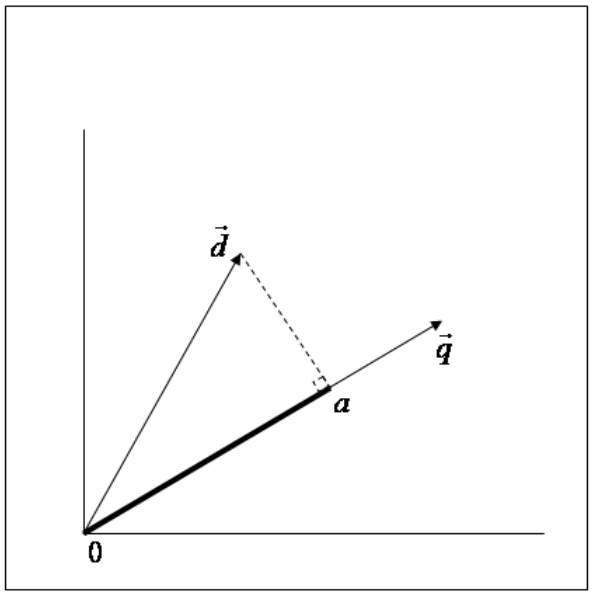


Figure 3: The matching value of $d$ with respect to $q$ is directly proportional to the projection of $\vec{d}$ on $\vec{q}$ (the signed length of the segment $[0, a]$)

### 4.1 $L_1$ Normalization

An apparently straightforward option for normalization is the $L_1$ norm, which sets:

$$norm(d) = \sum_{j=1}^{m} r\vec{t}i_d \tag{4}$$

Note that if we define $tf_d(j)$ to be the number of times term $v_j$ appears in $d$, and if for all terms $v_j$ we consider $idf_j = 1$, then $norm(d)$ is exactly the length of $d$ in words, penalizing the exact cause of the biases that we want to eliminate. Also, this normalization corresponds to "probability normalization", a technique commonly employed in other situations. While all this is intuitively appealing, it is practically unsuitable as shown by the geometric interpretation in Fig. 4.

Fig. 4 depicts the square region whose boundary is inhabited by all the $L_1$-normalized document vectors. Therefore, when searching for a document having the best match with a given query $q$, we have to look along the boundaries of this square. As illustrated, for the chosen query vector $\vec{q}$, the documents with the best possible match to the query are the ones for which the normalized vector points to $(0, 1)$, i.e. those documents containing only one term – the most frequent one in the query. What this means is that the $L_1$ norm amounts to reducing the query to one term, and finding the document in which this term is most frequent. This is clearly not a good solution to our problems.

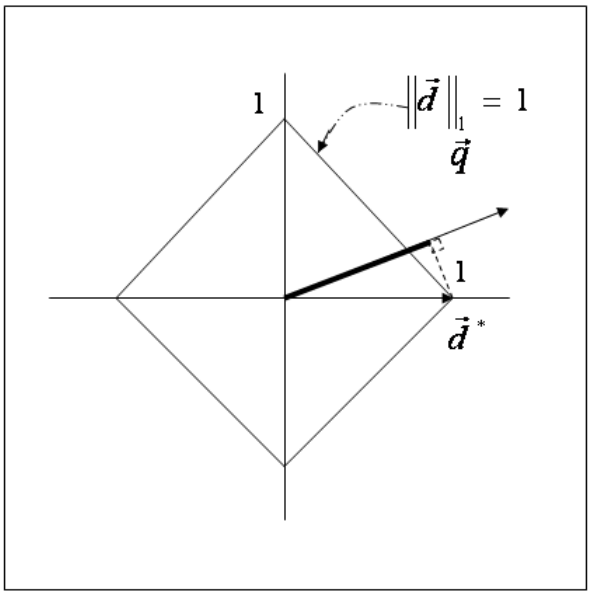


Figure 4: The $L_1$-normalized document vectors lie upon the depicted square. The documents best matching query $q$ are the ones normalized to $\vec{d}^*$.

### 4.2 $L_2$ normalization

A classic choice that is used in practice is the $L_2$ norm, also referred to as **cosine normalization**:

$$norm(d) = \sqrt{\sum_{j=1}^{m} r\vec{t}i_d^{\,2}} \tag{5}$$

The geometric interpretation of the $L_2$ norm is shown in Fig. 5. The $L_2$-normalized document vectors lie on the boundary of a circle centered at the origin; for a query $q$, the best matching document is a scaled version of $\vec{q}$.

Looking at figure 3, from the geometric interpretation of the cosine we have:

$$cos(\angle(\vec{d},\vec{q})) = \frac{proj(\vec{d},\vec{q})}{\left\|\vec{d}\right\|_2} \;\Rightarrow\; proj(\vec{d},\vec{q}) = \left\|\vec{d}\right\|_2 cos(\angle(\vec{d},\vec{q})) \tag{6}$$

and since $\left\|\vec{d}\right\|_2 = 1$:

$$proj(\vec{d},\vec{q}) = \left\|\vec{d}\right\|_2 cos(\angle(\vec{d},\vec{q})) \tag{7}$$

We can thus conclude that, for this matching function, normalizing with $L_2$ is equivalent to directly using $cos(\angle(\vec{d},\vec{q}))$ as the matching function, which seems to support the argument in 3.1 about not bothering with normalization. However, the fact that our matching function allows normalization is an advantage in itself, as shown by [SBM96].

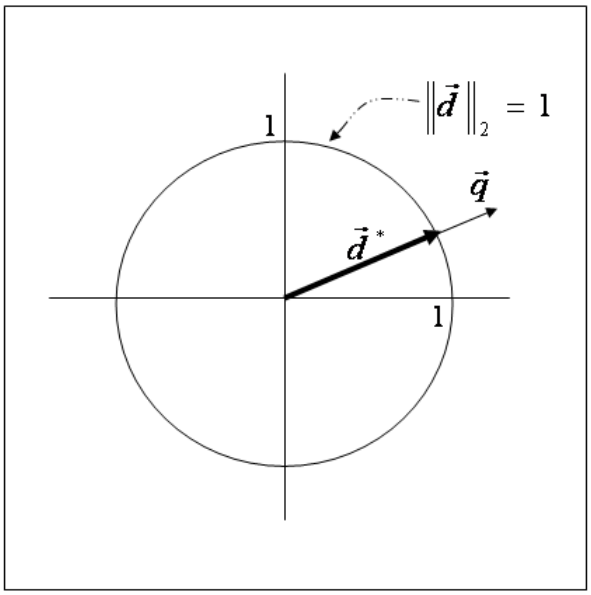


Figure 5: The best matching document has normalized vector $\vec{d^*} = \frac{\vec{q}}{\|\vec{q}\|_2}$

.

## References

[SAB+99] Amit Singhal, Steven P. Abney, Michiel Bacchiani, Michael Collins, Donald Hindle, and Fernando C. N. Pereira. AT&T at TREC-8. In *TREC*, 1999.

[SBM96] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *SIGIR '96: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29, New York, NY, USA, 1996. ACM Press.

## 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.