CHAPTER 4

Text Mining 1: Indexing and Querying Text

1. Motivation

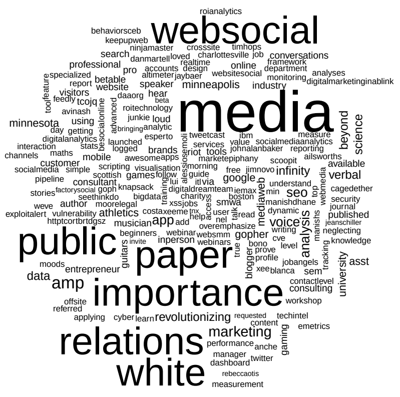
Finding the public opinion. The social Web contains the opinions of a large sample of the population. We can use messages written by the public to examine public opinion.

To examine opinion, we must examine the text and be able to compute:

- the relationship between words (dog is more similar to puppy than rocket)
- the relationship between words and documents (a document is related to the word puppy)
- the relationship between documents (document A is more similar to document B than C)

Example using word similarities: Geek vs Nerd

Digging through the piles of text. Given ten documents, we could read them and record the relationships between words and documents.



Given a million documents, we have to automate the process.

2. Document Models

The need for a document model. Given a set of documents, we are usually faced with the problem of searching through them, to either:

- Find the document that is most similar to a query (Google)
- Find the document that is most similar to another document
- Find words that are most similar to another word

Which tweet is most similar to "Hero memories" or "Game"? @Richmond_FC Final siren, the Tiges win by 41 points at the MCG! #gotiges #yellowandblack @SJGames #hook The deities inform the heroes that one of the PCs will soon die, to be replaced by another version who will retain his memories. -sjm

How can we compute the similarity over millions of tweets?

Bag of words (independent) models. The most common form of document representation is as a *bag of words*, meaning that each document is represented as a set of unique words and an associated weight (the order of the words is ignored, just as if we tipped the document words into a bag):

	word 1	word 2	...	word M
$\vec{d}$	$w_{d,1}$	$w_{d,2}$	...	$w_{d,M}$

where $w_{d,t}$ is the weight of word t in document d.

The weight $w_{d,t}$ is a score showing how well the term t represents document d. The weight can be positive, zero or negative. The greater the weight, the more likely that the term represents the document.

Example: Document representation. Let's use $w_{d,t} = f_{d,t}$ the frequency of term t in document d. Our document set:

1. Social Web analytics is the best!
2. Social Web analytics is the greatest unit.
3. The best Web unit is Social Web analytics.

Construct the frequency table:

	Social	Web	analytics	is	the	best	greatest	unit
$\vec{d}_1$	1	1	1	1	1	1	0	0
$\vec{d}_2$	1	1	1	1	1	0	1	1
$\vec{d}_3$	1	2	1	1	1	1	0	1

Note that each row of the frequency table is an eight dimensional vector ($M = 8$).

Problem: Document Index.

Problem. Construct the frequency table of the following four documents:

- One one was a race horse
- Two two was one too
- One one won one race
- Two two won one too

Types of Document Models. There are three main document models:

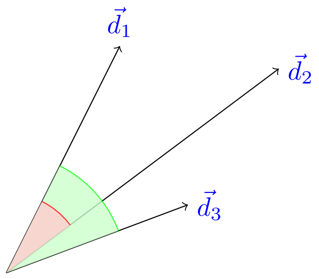
- Vector Space Model
- Probabilistic Model
- Language Model

Each of these models treats the documents as a *bag of words*, meaning that order of the words is lost once the model is computed.

We will examine the Vector Space Model and Language Model.

Vector Space Model.

Vector Space Model. Each document is treated as a vector in an M dimensional vector space, where M is the number of unique terms in the document collection. Each element of the document vector is associated to a particular term, the value of the element is the frequency of the term in the document.



Document vectors. A document vector $\vec{d}_i$ is an ordered set of term weights $w_{d,t}$, where the order depends on the words in the document collection.

To compare document vectors, we examine the angle between the vectors, which is inversely proportional to the cosine of the angle:

$$s(\vec{d}_i, \vec{d}_j) = \cos(\theta_{i,j}) = \frac{\vec{d}_i \cdot \vec{d}_j}{\left\|\vec{d}_i\right\| \left\|\vec{d}_j\right\|}$$

where:

$$\vec{d}_i \cdot \vec{d}_j = \sum_{k=1}^{M} w_{d_i,t_k} w_{d_j,t_k} \text{ and } \left\|\vec{d}_i\right\| = \sqrt{\sum_{k=1}^{M} w_{d_i,t_k}^2}$$

Therefore, $s(\vec{d}_i, \vec{d}_j) = 1$ if $\vec{d}_i / \left\|\vec{d}_i\right\| = \vec{d}_j / \left\|\vec{d}_j\right\|$, otherwise $-1 \leq s_{i,j} < 1$.

Queries. Using the Vector Space Model, we can compare documents to documents, but we want to be able to compare queries to documents as well (to find the documents most relevant to a query).

Fortunately, a query is a set of words, and can be seen as a very short document. Therefore, we can represent a query using a document vector.

Example: Query the document index. If we have the query "best Web unit", we create the query vector $\vec{q}$ in the eight dimensional vector space:

	Social	Web	analytics	is	the	best	greatest	unit
$\vec{q}$	0	1	0	0	0	1	0	1

then compute the similarity of the query to all documents:

	$\vec{d} \cdot \vec{q}$	$\left\Vert\vec{d}\right\Vert$	$S(\vec{d}, \vec{q})$
$\vec{d}_1$	2	$\sqrt{6}$	$2/(\sqrt{6}\sqrt{3}) = 0.47$
$\vec{d}_2$	2	$\sqrt{7}$	$2/(\sqrt{7}\sqrt{3}) = 0.44$
$\vec{d}_3$	4	$\sqrt{10}$	$4/(\sqrt{10}\sqrt{3}) = 0.73$
$\vec{q}$		$\sqrt{3}$	

Problems with term frequency. In the previous example we used the the frequency $f_{d,t}$ as the word weight $w_{d,t}$. This lead to problems:

1. Repeating words: If a word appears twice as many times in a document, it should not double the document score. Web page designers used to repeat words to attempt to artificially boost their pages raking in search results.
2. Common words dominate the results: The previous example shows that when using the query "one won" all documents scored high because they contained the word "one". The word "one" contains no information because it appears in every document, and so should not effect the results.

To combat these problems we will use TF-IDF weighting. TF for problem 1, IDF for problem 2.

TF-IDF. TF-IDF weighting was designed using experimentation, and has shown to provide good results for a simple method.

- TF: term frequency
- IDF: Inverse Document Frequency

The weight for term in a document is computed as the product of the term frequency weight and inverse document frequency weight.

Term weight function (IDF). IDF is a measure of the terms importance across the document collection.

The term weight can be computed using the function:

$$\mathrm{IDF}_t = \log_e\left(\frac{N}{f_t}\right)$$

where N is the number of documents, f_t is the number of documents containing term t, and IDF_t is the weight of term t.

Term weight plot ($N = 100$). As the term document count increases, its weight decreases.

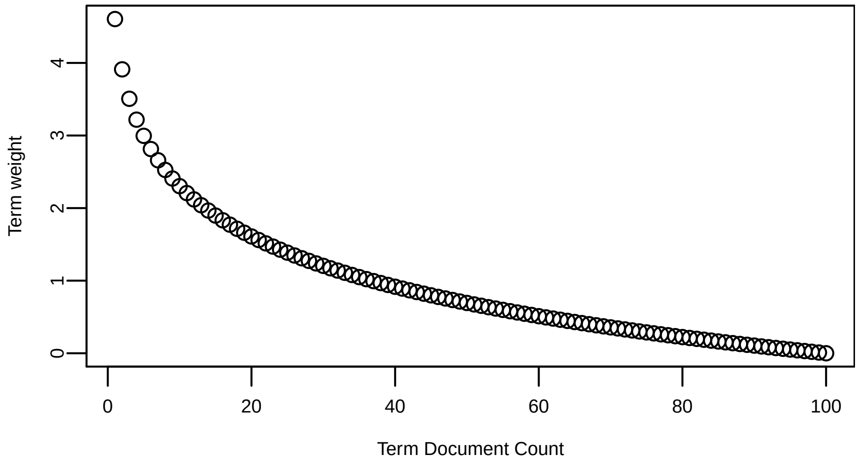


FIGURE 1. Inverse document frequency (IDF)

Within document term weight (TF). The within document term weight (TF) is a measure of how important the term is within the document. This weight is dependent on the frequency of the term in the document.

The within document term weight can be computed using:

$$\mathrm{TF}_{d,t} = \log_e \left(f_{d,t} + 1\right)$$

where $f_{d,t}$ is the frequency of term t in document d and $\mathrm{TF}_{d,t}$ is the weight of term t in document d.

TF weighted frequency plot. As the frequency increases, the increase in weight reduces.

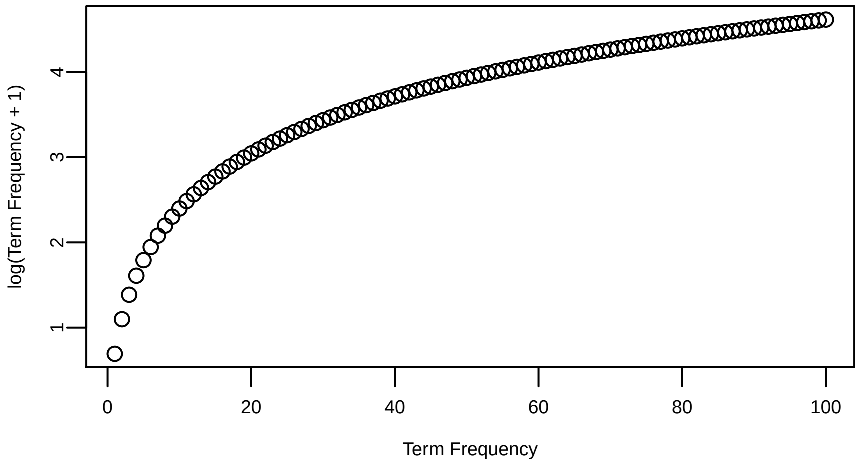


FIGURE 2. Weighted term frequency (TF)

TF-IDF: Putting it together. The weighted term frequency is given as:

$$\begin{aligned} w_{d,t} &= \mathrm{TF}_t \times \mathrm{IDF}_{d,t} \\ &= \log_e\left(f_{d,t}+1\right)\log_e\left(\frac{N}{f_t}\right) \end{aligned}$$

where $f_{d,t}$ and $w_{d,t}$ are the frequency and weight of term t in document d, N is the number of documents, and f_t is the number of documents containing term t.

Problem. Compute the TF-IDF weights for documents 2 and 3. Use them to compute the cosine similarity documents scores for documents 2 and 3 and query "one won".

- One one was a race horse
- Two two was one too
- One one won one race
- Two two won one too

Document Matrix. Computing the similarity is the inner product of normalised vectors. Therefore, we can compute the similarity to all documents using a matrix multiply.

Our document matrix is:

$$D = \begin{bmatrix} \vec{d_1}/\|d_1\| \\ \vec{d_2}/\|d_2\| \\ \vdots \\ \vec{d_N}/\|d_N\| \end{bmatrix} = \begin{bmatrix} w^\star_{d_1,t_1} & w^\star_{d_1,t_2} & \cdots & w^\star_{d_1,t_M} \\ w^\star_{d_2,t_1} & w^\star_{d_2,t_2} & \cdots & w^\star_{d_2,t_M} \\ \vdots & \vdots & \ddots & \vdots \\ w^\star_{d_N,t_1} & w^\star_{d_N,t_2} & \cdots & w^\star_{d_N,t_M} \end{bmatrix}$$

where:

$$w^\star_{d_i,t_j} = w_{d_i,t_j}/\|d_i\|$$

and $w^\star_{d_i,t_j}$ and w_{d_i,t_j} are the normalised weight and weight of word t_j in document d_i.

Similarity using the Document Matrix. Given a document matrix D, we can compute the similarity of all documents to query $\vec{q}$ using:

$$\vec{s} = D\vec{q}^T$$

where $\vec{s}$ is the vector of similarity scores, and $\vec{q}^T$ is the transpose of $\vec{q}$.

Language Model.

Language models in Information Retrieval. A language model is a statistical model of how a sequence of words is produced.

A simple language model takes the form of a categorical distribution with M categories over the set of all M words.

	word 1	word 2	...	word M
$\vec{d}$	p_1	p_2	...	p_M

where p_t is the probability of the next word being word t and $\sum_t p_t = 1$.

Note that just like the vector space model, this simple language model ignores the order of the terms (but more complex language models can be designed to use the term ordering).

Probability of a Document. Using a language model, we can compute the probability that a given document can be produced. We assume that a document is a set of independent terms, therefore the probability of all terms is the product of each term.

$$P(d|L) = \prod_t p_t^{f_{d,t}}$$

where L is the language model providing term probabilities p_t, $f_{d,t}$ is the frequency of term t in document d.

For long documents, the document probability will be very small, so we usually compute the log probability.:

$$\log P(d|L) = \log \prod_t p_t^{f_{d,t}} = \sum_t f_{d,t} \log p_t$$

Querying documents. To find documents best matching a query, we must be able to compare the query to each document. When using a language model, we assume that each document has been generated using its own language model, and we compute the probability that the query could be generated from each of the language models

If it is likely that a query could be generated using a document model that has generated document i, then the query and document i are likely to contain the same topics.

Given query q and the document model for document d (L_d), the probability of query q is:

$$s(d,q) = \log P(q|L_d) = \log \prod_t p_t^{f_{q,t}} = \sum_t f_{q,t} \log p_t$$

The documents are then ordered by their $s(d,q)$ score.

Computing Language Models. To compute the query score, we must be able to compute the language model for a given document (document model). Our model is a categorical distribution, so we must fit the values p_t to the document (the document is our sample from the model).

We must ask, for document d, what are the most likely values of p_t that would lead to generating document d. The answer is:

$$p_t = P(t|d) = \frac{f_{d,t}}{\sum_t f_{d,t}}$$

So $P(t|d)$ is the proportion of term t in document d.

Problem: Document Models.

Problem. Construct document models documents for documents 2 and 3 of the following four documents.

- One one was a race horse
- Two two was one too
- One one won one race
- Two two won one too

Then compute the similarity of the documents 2 and 3 to the query: “one won”

Zero Probability. We saw in the previous problem that if a document model has zero probability for any query term, then the probability of the query is zero (log probability of $-\infty$).

We computed the zero probability for p_t based on the sample document not having any occurrence of term t ($f_{d,t} = 0$), but should the model have zero probabilities? It is more likely that the model has very small probabilities, but not zero probability. Zero probability implies that the term will never exist in a document generated from the model, but it is more likely that the term is improbable, not impossible.

Sunrise Problem. What is the probability that the sun will rise tomorrow? If we base the answer on our prior knowledge (sample), the probability is 1. `https://en.wikipedia.org/wiki/Sunrise_problem`

Model Smoothing. To remove the problem of zero term probabilities, we smooth out the probabilities based on what we believe the probabilities should be (background probability).

A good estimate of background probability comes from combining all documents into a document collection C.

The background probability of term t is:

$$P(t|C) = \frac{\sum_d f_{d,t}}{\sum_t \sum_d f_{d,t}}$$

The smoothed probability for term t is:

$$\lambda P(t|d) + (1 - \lambda) P(t|C)$$

where $\lambda \in [0, 1]$

Choosing λ. The smoothed document model has a smoothing parameter λ, but what do we set it to?

- $\lambda = 1 \rightarrow P(t|d)$
- $\lambda = 0 \rightarrow p(t|C)$

If we examine the document d that is used to compute $P(t|d)$, if d has many words, it is a large sample from the document model, so the probabilities $P(t|d)$ should be good estimates of the document model term proportion. If d contains only a few words, then it is a small sample from the document model, and so the estimate $P(t|d)$ may not be good estimates.

Therefore, if a document has many words, λ should be closer to 1.

Dirichlet Smoothing. Dirichlet smoothing uses:

$$\lambda = \frac{n_d}{n_d + \alpha}$$

where $n_d = \sum_t f_{d,t}$ (the number of words in document d), and α is a positive real number.

- As n_d increases, λ approaches 1,
- as n_d decreases, λ approaches 0.

The Dirichlet parameter α is related to the sample size (number of words in the document). If $\alpha = n_d$, $\lambda = 0.5$.

Example: Language Model with Dirichlet Smoothing.

Example. Using your document models of the below documents, compute the similarity of the document 2 to the query: "one won", using $\alpha = 10$.

- One one was a race horse
- Two two was one too
- One one won one race
- Two two won one too

3. Preprocessing Text

Preparing for index construction. The English language contains a lot of redundancy that allows us to express ideas, but provides confusion to computer processes that do not understand the relationships between the words.

In this section we will examine a few methods to strip down the text to increase the effectiveness of analysis.

Stop words. Stop words provide no or only little information to our analysis and so can be safely removed. Removing stop words can be beneficial to an analysis, but it always depends on the text being examined.

Problem. Which words can we remove from these documents:

1. Social Web analytics is the best!
2. Social Web analytics is the greatest unit.
3. The best Web unit is Social Web analytics.

Stop word list. Here is an example stop word list:

a, able, about, across, after, all, almost, also, am, among, an, and, any, are, as, at, be, because, been, but, by, can, cannot, could, dear, did, do, does, either, else, ever, every, for, from, get, got, had, has, have, he, her, hers, him, his, how, however, i, if, in, into, is, it, its, just, least, let, like, likely, may, me, might, most, must, my, neither, no, nor, not, of, off, often, on, only, or, other, our, own, rather, said, say, says, she, should, since, so, some, than, that, the, their, them, then, there, these, they, this, tis, to, too, twas, us, wants, was, we, were, what, when, where, which, while, who, whom, why, will, with, would, yet, you, your

Lists can be compiled for a specific task.

When not to remove stop words. By removing stop words, we remove all occurrences of "To be or not to be."

Letter case and punctuation. When examining words in documents, we want to identify how often they appear and in which documents they appear.

To keep all occurrences of a word consistent, we must remove any difference in letter case. This is done by adjusting all text to be lower case.

Punctuation is important for text sequences (sentences), but not needed when examining individual terms. All punctuation can be removed.

Our document set has been reduced to:

1. social web analytics best
2. social web analytics greatest unit
3. best web unit social web analytics

Stemming. There are many words that have the same stem, but are adjusted due to their use in a sentence. By removing the variation, we obtain a better understanding the occurrence of the word.

For example the words "fishing", "fished", "fish", and "fisher" have the stem "fish". The words "argue", "argued", "argues" and "arguing" have the stem "argue".

The most commonly used stemming algorithm is Porter's stemmer.

Our document set has been reduced to:

1. social web analyt best
2. social web analyt great unit
3. best web unit social web analyt

Note that the stems do not have to be words.

Summary.

- Text documents can be represented as a bag of words (treating all words independently).
- To compare documents we can use the Vector Space Model or Language Model.
- The Vector Space Model uses TF-IDF weighting and Cosine Similarity.
- Language Models use a Dirichlet Smoothed Categorical distribution.
- Before building the models we can preprocess (remove stop words, lowercase, remove punctuation, stem)

Next Week. Visualisation