TFIDF document summarization

MSAN692
Terence Parr

1 Goal

The goal of this lesson is to learn a core technique used in text analysis called *TFIDF* or *term frequency, inverse document frequency*. This document is background material for your TFIDF project. We will use what is called a *bag-of-words* representation where the order of words in a document doesn't matter—we care only about the words and how often they are present. A word's TFIDF value is often used as a feature for document clustering or classification. We will use it simply as a summarization tool for documents. The more a term helps to distinguish its enclosing document from other documents, the higher its TFIDF score.

2 Discussion

One way to summarize a text document is to list, say, the top 25 words that seem most important. That could also be used to compare documents to see if they're talking about the same thing. For example, I had to solve a problem 15 years ago to reduce noise in the forums of a Java developer's website. Users were posting stupid posts about movies and were also putting database questions in the forum on GUIs. The goal was to detect non-Java posts and also to detect misplaced posts. What does it mean to "talk about Java"? How do I know when someone is talking about databases versus GUIs? My solution was to identify the words important to Java as a whole ("Java-speak"), database, and GUI posts. Any posts that did not have words important to Java, were tossed out as irrelevant after giving them a mild smack on the snout. Similarly, posts without words relevant to databases were compared to vocabularies associated with other topics to see if another forum would be more appropriate.

Collecting all of the words commonly used in the database forum did not work well because the model thought that "the" was important to the database topic, and the GUI topic, etc... I needed to discount words used frequently in all topics.

To make this discussion simpler, let's think about individual documents versus all documents in a corpus. We boost words used frequently in a document but attenuate that word if it is used in a lot of documents. For more on this topic, see Introduction to Information Retrieval.

A word's *term frequency* is just the term (word) count within a document divided by the number of words in that document (some people use "frequency" to mean "count" but that is technically inaccurate):

$$tf(t,d) = \frac{count(t)}{|d|} \qquad \text{(Term frequency of term } t \in d\text{, document } d\text{)}$$

The term frequency is higher for commonly used words within a document. So, "the" will be very common and thus have high term frequency in most documents.

A word's *document frequency* is the count of documents containing that word divided by the total number of documents:

$$df(t, N) = \frac{|\{d_i : t \in d_i,\ i = 1..N\}|}{N} \qquad \text{(Document frequency of } t \text{ in } N \text{ documents)}$$

The document frequency is higher for words used in many documents. For example, "the" will have very high document frequency because it is present in just about every document. On the other hand, a word specific to a few articles such as "simulation" will have much lower document frequency.

The goal is to find words that help to distinguish one document from the others. To do that we want to amplify the words used commonly in that document but attenuate that word frequency if the word is used across documents frequently. Quantifying that information per word is called the TFIDF score. A simple division of term frequency within a document by document frequency is a simple but effective starting point:

$$tfidf(t, d, N) = \frac{tf(t, d)}{df(t, N)} \qquad \text{(First approximation to TFIDF)}$$

Heavy use of t in d bumps the score up, but heavy use of t across other documents pulls the score down.

This formula is meaningful but gives a poor score because the document frequency tends to overwhelm the term frequency in the numerator, so we take the log of the denominator first. Here's the formula slightly rewritten as it is normally shown:

$$tfidf(t, d, N) = tf(t, d) \times log(\frac{1}{df(t, N)}) \qquad \text{(TFIDF with attenuated document frequency)}$$

When t is in every document, *idf* is $log(1/df(t, N)) = log(1) = 0$, which gives a TFIDF score of 0. Such a word is a terrible word for distinguishing between documents as it exists in all of them. When t is in very few documents, such as $1/10^5$, *idf* is $log(10^5)$, which is about 11.5.

To summarize a document, we can order its terms by *tfidf* in reverse order and look at the top 20 words, for example. (To get the lexicon of a topic like databases, we can collect a known set of database posts into a single document and compute the tfidf in association with the aggregated documents of the other topics.) For example, here is a set of terms and the associated *tfidf* computed from a sample Reuters article:

Term	*tfidf*
rating	0.123
fox	0.119
nbc	0.119
homes	0.114
cbs	0.079
audiences	0.079
neilsen	0.079
evening	0.067
abc	0.067
watching	0.063

It's clear that it's talking about Nielsen ratings for news programs, without even looking at the original article.

To compute TFIDF using code, we would need an overall index that maps term t to document frequency $df(t, N)$ for all t in all N documents and an index that maps document d to another index that maps each $t \in d$ to $tf(t, d)$. From that, we could easily compute all of the TFIDF scores.

Aside. We need to prevent division by 0 errors when a term does not exist in a corpus (e.g., $df(t, N) = 0$ in search applications where we pass unknown term(s) t). If $df(t, N) = 0$, then $\frac{1}{df(t,N)}$ will be a problem. To fix this, we can simply add 1 to the numerator of the document frequency definition:

$$df(t, N) = \frac{|\{d_i : t \in d_i,\ i = 1..N\}| + 1}{N}$$

This is similar to *additive smoothing* that you will see when estimating term probabilities in document classifiers. The technique is like pretending there is an imaginary document with every unknown word (and, indeed, every possible word). To keep document frequencies in [0..1], we can to bump the document count, N, in the denominator as well.

$$df(t, N) = \frac{|\{d_i : t \in d_i,\ i = 1..N\}| + 1}{N + 1} \qquad (df \text{ with smoothing})$$

For example, if we have a vector of words in a search query $[the, apple, cat, foo]$ aggregated from 100 documents, we might get a set of document frequencies like this:

$$[\frac{100}{100}, \frac{4}{100}, \frac{9}{100}, \frac{0}{100}]$$

With smoothing, we would get:

$$[\frac{101}{101}, \frac{5}{101}, \frac{10}{101}, \frac{1}{101}]$$

This is like converting zeros to some really small number (assuming N is large) and adding that same small number to the other document frequencies.

3 Example

Consider a sample set of documents:

d_1 = "**in the new york times in**" (repeated **in**)
d_2 = "**the new york post**"
d_3 = "**the los angeles times**"

For these documents, we would get a combined vocabulary of [**angeles, in, los, new, post, the, times, york**]. A common way to represent the documents then is as a vector from this vocabulary whose elements are the *term counts* (in this case, just 0, 1, and 2):

	angeles	in	los	new	post	the	times	york
d_1	0	2	0	1	0	1	1	1
d_2	0	0	0	1	1	1	0	1
d_3	1	0	1	0	0	1	1	0

The term frequencies would be each of those numbers divided by 6 or 4 because the documents are 6 or 4 words long.

The document counts and frequencies are:

$t \in d$	$count(t)$	$df(t, N = 3)$	$log_{10}(\frac{1}{df(t,N=3)})$
angeles	1	1/3	0.4771
in	1	1/3	0.4771
los	1	1/3	0.4771
new	2	2/3	0.1761
post	1	1/3	0.4771
the	3	3/3	0.0
times	2	2/3	0.1761
york	2	2/3	0.1761

The *tfidf* for the words in the documents are:

	angeles	in	los	new	post	the	times	york
d_1	0	2/6*.4771	0	1/6*.1761	0	1/6*0	1/3*.1761	1/6*.1761
d_2	0	0	0	1/4*.1761	1/4*.4771	1/4*0	0	1/4*.1761
d_3	1/4*.4771	0	1/4*.4771	0	0	1/4*0	1/4*.1761	0

Multiplied out:

	angeles	in	los	new	post	the	times	york
d_1	0	**0.1590**	0	0.0293	0	0	0.0293	0.0293
d_2	0	0	0	0.0440	**0.1193**	0	0	0.0440
d_3	**0.1193**	0	0.1193	0	0	0	0.0440	0

From this table, we can conclude that most important terms in each document are those I have bolded (the largest TFIDF scores). Notice that **the** got totally wiped out because of the 0 *idf* score. It is useless as a distinguishing word.

4 Exercise

To get a feel for computing these scores, use Excel or some other spreadsheet to build a model you can play with. Or, go through TFIDF example using a Pandas data frame. The **key** to learning this stuff though is to do some examples by hand!

For the Excel approach, I created a number of "tables" as you can see from this figure:

	doc				*Term Counts*				*Term Frequencies*				*TFIDF scores*		
terms	**count(t)**	**df(t,N)**	**log10(1/df)**		**d1**	**d2**	**d3**		**d1**	**d2**	**d3**		**d1**	**d2**	**d3**
angeles	1	0.3333	0.4771		0	0	1		0	0	0.25		0	0	**0.1193**
in	1	0.3333	0.4771		2	0	0		0.333	0	0		**0.159**	0	0
los	1	0.3333	0.4771		0	0	1		0	0	0.25		0	0	0.1193
new	2	0.6667	0.1761		1	1	0		0.167	0.25	0		0.0293	0.044	0
post	1	0.3333	0.4771		0	1	0		0	0.25	0		0	**0.1193**	0
the	3	1.0000	0.0000		1	1	1		0.167	0.25	0.25		0	0	0
times	2	0.6667	0.1761		1	0	1		0.167	0	0.25		0.0293	0	0.044
york	2	0.6667	0.1761		1	1	0		0.167	0.25	0		0.0293	0.044	0
				total	6	4	4								

The count values in the document frequency (the first table) are entered manually by looking at the sample sentences. The next two columns are computed from the first column.

Then make a document-word table that represents the word vectors for each document, manually entering the word counts. Manually enter the length as well.

Create another document-word table that multiplies the term frequency by the inverse (log) document frequency from the first table. The term frequencies are the count in the word vectors divided by their length.

Once you have this in place, you can tweak the documents by changing word counts to see what happens to the various scores.

The Excel solution is available.