**Term frequency and weighting**

Thus far, scoring has hinged on whether or not a query term is present in a zone within a document. We take the next logical step: a document or zone that mentions a query term more often has more to do with that query and therefore should receive a higher score. To motivate this, we recall the notion of a *free text query* introduced in Section [1.4](https://nlp.stanford.edu/IR-book/html/htmledition/the-extended-boolean-model-versus-ranked-retrieval-1.html#sec:boolean-querying) : a query in which the terms of the query are typed freeform into the search interface, without any connecting search operators (such as Boolean operators). This query style, which is extremely popular on the web, views the query as simply a set of words. A plausible scoring mechanism then is to compute a score that is the sum, over the query terms, of the match scores between each query term and the document.

Towards this end, we assign to each term in a document a *weight* for that term, that depends on the number of occurrences of the term in the document. We would like to compute a score between a query term $t$ and a document $d$, based on the weight of $t$ in $d$. The simplest approach is to assign the weight to be equal to the number of occurrences of term $t$ in document $d$. This weighting scheme is referred to as *term frequency* and is denoted $\mbox{tf}_{t,d}$, with the subscripts denoting the term and the document in order.

For a document $d$, the set of weights determined by the $\mbox{tf}$ weights above (or indeed any weighting function that maps the number of occurrences of $t$ in $d$ to a positive real value) may be viewed as a quantitative digest of that document. In this view of a document, known in the literature as the *bag of words model* , the exact ordering of the terms in a document is ignored but the number of occurrences of each term is material (in contrast to Boolean retrieval). We only retain information on the number of occurrences of each term. Thus, the document ``Mary is quicker than John'' is, in this view, identical to the document ``John is quicker than Mary''. Nevertheless, it seems intuitive that two documents with similar bag of words representations are similar in content. We will develop this intuition further in Section [6.3](https://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html#sec:docvectors) .

Before doing so we first study the question: are all words in a document equally important? Clearly not; in Section [2.2.2](https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html#sec:stopwords) (page [[*]](https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html#p:stopwords)) we looked at the idea of *stop words* - words that we decide not to index at all, and therefore do not contribute in any way to retrieval and scoring

## Inverse document frequency

Raw term frequency as above suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy on a query. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this end, we introduce a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high collection frequency, defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the $\mbox{tf}$ weight of a term by a factor that grows with its collection frequency.

Instead, it is more commonplace to use for this purpose the *document frequency* $\mbox{df}_t$, defined to be the number of documents in the collection that contain a term $t$. This is because in trying to discriminate between documents for the purpose of scoring it is better to use a document-level statistic (such as the number of documents containing a term) than to use a collection-wide statistic for the term.

|  |
| --- |
| \begin{figure}\begin{tabular}{\vert l\vert l\vert l\vert} \hline % after \\ : ... ...10422 & 8760\\ insurance & 10440 & 3997 \\ \hline \end{tabular} \end{figure} |
| **Figure 6.7:** Collection frequency (cf) and document frequency (df) behave differently, as in this example from the Reuters collection. |

The reason to prefer df to cf is illustrated in Figure [6.7](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#fig:cfdf) , where a simple example shows that collection frequency (cf) and document frequency (df) can behave rather differently. In particular, the cf values for both try and insurance are roughly equal, but their df values differ significantly. Intuitively, we want the few documents that contain insurance to get a higher boost for a query on insurance than the many documents containing try get from a query on try.

How is the document frequency df of a term used to scale its weight? Denoting as usual the total number of documents in a collection by $N$, we define the *inverse document frequency* of a term $t$ as follows:

|  |  |
| --- | --- |
| \begin{displaymath} \mbox{idf}_t = \log {N\over \mbox{df}_t}. \end{displaymath} | (21) |

Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low. Figure [6.8](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#fig:figureidf) gives an example of idf's in the Reuters collection of 806,791 documents; in this example logarithms are to the base 10. In fact, as we will see in Exercise [6.2.2](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#ex:logbase) , the precise base of the logarithm is not material to ranking. We will give on page [11.3.3](https://nlp.stanford.edu/IR-book/html/htmledition/probability-estimates-in-practice-1.html#p:justificationofidf) a justification of the particular form in Equation [21](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#eqn:idf).

\begin{figure}
% latex2html id marker 7963
\par
\begin{tabular}{\vert\vert l\ver...
...quencies in the Reuters collection of 806,791 documents.}
\par
\par
\end{figure}

## Tf-idf weighting

We now combine the definitions of term frequency and inverse document frequency, to produce a composite weight for each term in each document. The *tf-idf* weighting scheme assigns to term $t$ a weight in document $d$ given by

|  |  |
| --- | --- |
| \begin{displaymath} \mbox{tf-idf}_{t,d} = \mbox{tf}_{t,d} \times \mbox{idf}_t. \end{displaymath} | (22) |

In other words, $\mbox{tf-idf}_{t,d}$ assigns to term $t$ a weight in document $d$ that is

1. highest when $t$ occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

At this point, we may view each document as a *vector* with one component corresponding to each term in the dictionary, together with a weight for each component that is given by ([22](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#eqn:tfidf)). For dictionary terms that do not occur in a document, this weight is zero. This vector form will prove to be crucial to scoring and ranking; we will develop these ideas in Section [6.3](https://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html#sec:docvectors) . As a first step, we introduce the *overlap score measure*: the score of a document $d$ is the sum, over all query terms, of the number of times each of the query terms occurs in $d$. We can refine this idea so that we add up not the number of occurrences of each query term $t$ in $d$, but instead the tf-idf weight of each term in $d$.

|  |  |
| --- | --- |
| \begin{displaymath} \mbox{Score}(q,d)=\sum_{t\in q} \mbox{tf-idf}_{t,d}. \end{displaymath} | (23) |

In Section [6.3](https://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html#sec:docvectors) we will develop a more rigorous form of Equation [23](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#eqn:docscore).

**Exercises.**

* Why is the idf of a term always finite?
* What is the idf of a term that occurs in every document? Compare this with the use of stop word lists.
* Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Figure [6.9](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#fig:tfgraph) .

|  |
| --- |
| \begin{figure}\begin{tabular}{\vert\vert l\vert r\vert r\vert r\vert\vert} \hlin... ...rance & 0 & 33 & 29 \\ best & 14 & 0 & 17 \\ \hline \end{tabular} \end{figure} |
| **Figure 6.9:** Table of tf values for Exercise [6.2.2](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#ex:tfidf). |

* Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure [6.8](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#fig:figureidf) .
* Can the tf-idf weight of a term in a document exceed 1?
* How does the base of the logarithm in ([21](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#eqn:idf)) affect the score calculation in ([23](https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html#eqn:docscore))? How does the base of the logarithm affect the relative scores of two documents on a given query?
* If the logarithm in ([21](https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html#eqn:idf)) is computed base 2, suggest a simple approximation to the idf of a term.

http://wdf-idf.com/

# WDF\*IDF

**WDF\*IDF – The new key for top search engine ranking**

In 2014 keyword density is out. This, or a similar presumption could be ascertained when the new star in search engine optimisation cropped up. It is called WDF\*IDF and at least forces the keyword density into the background. Keywords for their part remain up to date, after all the new type of analysis discloses which keywords are of interest in the context.

**What is the WDF\*IDF analysis?**

The analysis is actually not as innovative as the SEO communities worldwide treat the topic. The origins of the analysis go back to the seventies, even though they naturally were not drawn for search engines or other rankings. However the formula and the calculation of terms on which the analysis is based dates back to the seventies. This information only mentioned incidentally. It is primarily all about understanding and clarifying the abbreviation, what is hidden behind the six letters:

* **Within Document Frequency – WDF**

In words, this section describes nothing but the term weighting within a document. In turn the term in this case stands for the keyword which appears in a certain frequency in the text. This principle could represent the customary keyword density, however, WDF at the same time operates with a correction value which then allows for a uniform reference to result. In addition WDF is limited by the

* **Inverse Document Frequency – IDF**

This area of the analysis does not only concentrate on one single document but refers the weighting of a certain term in the entire site. If, for instance in a blog only one single contribution on the site deals with photo albums the word “Garden chair” is important for the analysis – after all it is only mentioned in one single text on the site. However, if a term goes through the whole text, it is not interesting for the analysis as most probably not only one keyword is concerned.

The entire principle is expressed in a formula which will, with the exception of mathematicians, probably confuse many people:

[ormel](http://wdf-idf.com/wp-content/uploads/2013/08/formel.jpg)

Luckily no one is forced even at this point to compare the term weighting in a document manually with those on a complete site.

If a text has an own DNA, the WDF\*IDF analysis will reveal it. The latter of course only functions in a figurative sense, however the practical technique offers the possibility to reveal how well a certain word reflects the contents of a text. As already indicated in the example above, a word such as “and” will hardly point towards a relative content. The conjunction is found in every text and has absolutely no value to offer – other than to connect clauses and sentences. The word “garden chair” on the other hand already has a value and provides the opportunity to clearly localise search results. Those feeling like calculating the value manually can have a try at the following formula:

[ormel-wdf](http://wdf-idf.com/wp-content/uploads/2013/08/formel-wdf.png)

To really be able to implement the formula, the precise number of documents circulating around the World Wide Web with the term should be known. Of course it is almost impossible to filter it, thus principally only a reference value is used to work with.

This part of the formula can be seriously compared with keyword density, therefor does not represent an innovative achievement in the world of search engine optimisation. Therefore the actual WDF\*IDF formula consists of two sections which unite to become a whole. In this case the word acts on the principle that a term is always important the more its presence in a document deviates from the numerous occurrences in other documents on the same website. This mathematical formula is as follows:

[ormel-idf](http://wdf-idf.com/wp-content/uploads/2013/08/formel-idf.png)

If the results of both formulas are multiplied by each other, the frequency of interest for the actual term weighting appears.

**What benefit do I get from the analysis?**

This is a question which anyone who tackles the subject matter for the first time asks himself. The mathematics may interest mathematical wizards, however, it is not of interest for most of the site operators on their way to the desired ranking in the search engines. Put simply, the analysis reveals at a glance which terms are found in connection with the desired search term. Thus, based on the example “Garden chair of wood” not only

* Garden chair
* Wood

could be of importance, but at the same time the terms

* furniture
* outdoor
* garden
* patio
* tables
* outdoor furniture
* chairs
* table
* dining
* benches

The text created on the topic should therefore not only contain the main keyword “Garden chair of wood” but also the additional terms. Thereby know-how gained from the analysis could, of course, act as guideline for the developing text. It is certain that a good text on the topic will anyway contain the additional words – at least if the topic is viewed from its different aspects and in detail.

**For whom does WDF\*IDF  analysis make sense?**

It is a general problem that the onpage analysis principally reveals numerous words and marks them with a clear keyword curve in the diagram. The tools provided usually operate with a maximum and an average word frequency, so that the text resulting should orientate itself as closely as possible to the average value. Yet what happens if the resulting text should have a mere 250 words, the analysis however already refers to a total of 100 words? In the latter case the text should reveal a clear focus and ignore a number of words displayed in the diagram. Should the text on the garden chairs of wood, for example, particularly emphasis that a manufacturer particularly backing sustainable timer production is concerned, it is recommended that the focus be put on the FSC seal and to associate this with tropical timber.

The WDF\*IDF analysis is, however, of particular benefit for long documents for which the entire range of words in connection with the search term can be used.

**Finally some know-how**

Search terms, links or keyword densities have not completely vanished into thin air with the WDF\*IDF analysis. Those wanting to deal with the new tools correctly and wanting to created their documents in line with the new know-how, should consider the analysis results as direction signs and guidelines. If matching terms and synonyms in connection with the topic also occur, the document will be found way up in the ranking of search engines. At least until the term weighting is changed again by Google & Co. and a new marvel comes into being.

http://ethen8181.github.io/machine-learning/clustering\_old/tf\_idf/tf\_idf.html

# TF-IDF, Term Frequency-Inverse Document Frequency

## [TF-IDF, Term Frequency-Inverse Document Frequency](http://ethen8181.github.io/machine-learning/clustering_old/tf_idf/tf_idf.html#content)

* [TF-IDF](http://ethen8181.github.io/machine-learning/clustering_old/tf_idf/tf_idf.html#tf-idf)
* [R Session Information](http://ethen8181.github.io/machine-learning/clustering_old/tf_idf/tf_idf.html#r-session-information)
* [Reference](http://ethen8181.github.io/machine-learning/clustering_old/tf_idf/tf_idf.html#reference)

*This documentation assumes you are familiar with hierarchical clustering. To follow along, all the code (tf-idf.R) and the news data (news.csv) can be found*[*here*](https://github.com/ethen8181/machine-learning/blob/master/clustering_old/tf_idf)*.*

# TF-IDF

tf-idf, short for term frequency–inverse document frequency, is a numeric measure that is use to score the importance of a word in a document based on how often did it appear in that document and a given collection of documents. The intuition for this measure is : If a word appears frequently in a document, then it should be important and we should give that word a high score. But if a word appears in too many other documents, it’s probably not a unique identifier, therefore we should assign a lower score to that word. The math formula for this measure :

tfidf(t,d,D)=tf(t,d)×idf(t,D)tfidf(t,d,D)=tf(t,d)×idf(t,D)

Where t denotes the terms; d denotes each document; D denotes the collection of documents.

In the following documentation, we’ll break down this formula using four small documents to illustrate the idea.

*# environment*

**library**(tm)

**library**(proxy)

**library**(dplyr)

doc <- **c**( "The sky is blue.", "The sun is bright today.",

"The sun in the sky is bright.", "We can see the shining sun, the bright sun." )

## TF Term Frequency

The first part of the formula tf(t,d)tf(t,d) is simply to calculate the number of times each word appeared in each document. Of course, as with common text mining methods: stop words like “a”, “the”, punctuation marks will be removed beforehand and words will all be converted to lower cases.

*# create term frequency matrix using functions from tm library*

doc\_corpus <- **Corpus**( **VectorSource**(doc) )

control\_list <- **list**(removePunctuation = TRUE, stopwords = TRUE, tolower = TRUE)

tdm <- **TermDocumentMatrix**(doc\_corpus, control = control\_list)

*# print*

( tf <- **as.matrix**(tdm) )

## Docs

## Terms 1 2 3 4

## blue 1 0 0 0

## bright 0 1 1 1

## can 0 0 0 1

## see 0 0 0 1

## shining 0 0 0 1

## sky 1 0 1 0

## sun 0 1 1 2

## today 0 1 0 0

And that’s it for the term frequency part, easy peasy!!

## IDF Inverse Document Frequency

Let’s first write down the complete math formula for IDF.

idf(t,D)=log| D |1+| {d∈D:t∈d} |idf(t,D)=log| D |1+| {d∈D:t∈d} |

* The numerator : D is infering to our document space. It can also be seen as D = d1,d2,…,dnd1,d2,…,dnwhere n is the number of documents in your collection. Thus for our example | D || D |, the size of our document space is 4, since we’re only using 4 documents.
* The denominator : | {d∈D:t∈d} || {d∈D:t∈d} | implies the total number of times in which term t appeared in all of your document d ( the d∈Dd∈D restricts the document to be in your current document space ). Note that this implies it doesn’t matter if a term appeard 1 time or 100 times in a document, it will still be counted as 1, since it simply did appear in the document. As for the plus 1, it is there to avoid zero division.

Using our term frequency matrix, the idf weight for can be calculated like below.

*# idf*

( idf <- **log**( **ncol**(tf) / ( 1 + **rowSums**(tf != 0) ) ) )

## blue bright can see shining sky sun

## 0.6931472 0.0000000 0.6931472 0.6931472 0.6931472 0.2876821 0.0000000

## today

## 0.6931472

Now that we have our matrix with the term frequency and the idf weight, we’re ready to calculate the full tf-idf weight. To do this matrix multiplication, we will also have to transform the idf vector into a diagonal matrix. Both calculations are shown below.

*# diagonal matrix*

( idf <- **diag**(idf) )

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]

## [1,] 0.6931472 0 0.0000000 0.0000000 0.0000000 0.0000000 0 0.0000000

## [2,] 0.0000000 0 0.0000000 0.0000000 0.0000000 0.0000000 0 0.0000000

## [3,] 0.0000000 0 0.6931472 0.0000000 0.0000000 0.0000000 0 0.0000000

## [4,] 0.0000000 0 0.0000000 0.6931472 0.0000000 0.0000000 0 0.0000000

## [5,] 0.0000000 0 0.0000000 0.0000000 0.6931472 0.0000000 0 0.0000000

## [6,] 0.0000000 0 0.0000000 0.0000000 0.0000000 0.2876821 0 0.0000000

## [7,] 0.0000000 0 0.0000000 0.0000000 0.0000000 0.0000000 0 0.0000000

## [8,] 0.0000000 0 0.0000000 0.0000000 0.0000000 0.0000000 0 0.6931472

tf\_idf <- **crossprod**(tf, idf)

**colnames**(tf\_idf) <- **rownames**(tf)

tf\_idf

##

## Docs blue bright can see shining sky sun

## 1 0.6931472 0 0.0000000 0.0000000 0.0000000 0.2876821 0

## 2 0.0000000 0 0.0000000 0.0000000 0.0000000 0.0000000 0

## 3 0.0000000 0 0.0000000 0.0000000 0.0000000 0.2876821 0

## 4 0.0000000 0 0.6931472 0.6931472 0.6931472 0.0000000 0

##

## Docs today

## 1 0.0000000

## 2 0.6931472

## 3 0.0000000

## 4 0.0000000

Don’t start cheering yet, there’s still one more step to do for this tf-idf matrix. Recall that in the tf (term frequency) section, we’re representing each term as the number of times they appeared in the document. The main issue for this representation is that it will create a bias towards long documents, as a given term has more chance to appear in longer document, making them look more important than actually they are.

Thus the approach the resolve this issue is the good old L2 normalization. Math formula :

v̂ =v→‖v→‖v^=v→‖v→‖

For each vector v→v→, you divide it by its norm (length, magnitude). Calculation as below

*# Note that normalization is computed "row-wise"*

tf\_idf / **sqrt**( **rowSums**( tf\_idf^2 ) )

##

## Docs blue bright can see shining sky sun today

## 1 0.9236103 0 0.0000000 0.0000000 0.0000000 0.3833329 0 0

## 2 0.0000000 0 0.0000000 0.0000000 0.0000000 0.0000000 0 1

## 3 0.0000000 0 0.0000000 0.0000000 0.0000000 1.0000000 0 0

## 4 0.0000000 0 0.5773503 0.5773503 0.5773503 0.0000000 0 0

And that’s it, our final tf-idf matrix, when comparing it with our original document text.

doc

## [1] "The sky is blue."

## [2] "The sun is bright today."

## [3] "The sun in the sky is bright."

## [4] "We can see the shining sun, the bright sun."

One thing you can see is that the word “bright”, which appeared only in 3 out of the 4 documents is a given really low score across all the documents. This matches what we’ve said about the intuition of tf-idf in the beginning. A word should be representative of a document if it shows up a lot, but if that word occurs too often across all the documents, then it is most likely a meaningless indicator.

## Text Clustering

Now that we have this tf-idf matrix, one thing we can do with it is to perform text clustering !!

To performing document clustering using the tf-idf weight matrix, we’ll use the cosine similarity to measure how close are two given documents. Math formula :

cos(θ)=v⋅w‖v‖‖w‖=∑ni=1viwi∑ni=1v2i‾‾‾‾‾‾‾√∑ni=1w2i‾‾‾‾‾‾‾‾√cos(θ)=v⋅w‖v‖‖w‖=∑i=1nviwi∑i=1nvi2∑i=1nwi2

Where v and w are the two vectors that you wish to calculate the distance; vivi and wiwi are components of vector v and w respectively; and n is the number of components you have. A toy example of the calculation is shown below with two simple vector.

*# example*

a <- **c**(3, 4)

b <- **c**(5, 6)

*# cosine value and corresponding degree*

l <- **list**( numerator = **sum**(a \* b), denominator = **sqrt**( **sum**(a ^ 2) ) \* **sqrt**( **sum**(b ^ 2) ) )

**list**( cosine = l$numerator / l$denominator,

degree = **acos**(l$numerator / l$denominator) \* 180 / pi )

## $cosine

## [1] 0.9986877

##

## $degree

## [1] 2.935673

After calculating cos(θ)cos(θ), you can also obtain the actual θθ (degree) using the acos function in R. Note that the function returns the radian, you have to multiply it by 180 and divide by pi to obtain the actual degrees.

As for why we’re using this distance measure, remember what we’ve said in the normalization part, since documents are usually not of equal length, simply computing the difference between two vectors by using euclidean distance has the disadvantage that documents of similar content but different length are not regarded as similar in the vector space.

For this section, we’ll move on to a slightly larger dataset, since there’s really no point of performing text clustering when you only have 4 documents….

*# a slightly larger dataset*

**setwd**("/Users/ethen/machine-learning/clustering\_old/tf\_idf")

news <- **read.csv**("news.csv", stringsAsFactors = FALSE)

**list**( **head**(news), **dim**(news) )

## [[1]]

## title

## 1 First day of President Xi state visit

## 2 How do the Chinese view UK politicians?

## 3 In pictures: China's President Xi Jinping on day one of the state visit

## 4 China and 'the Osborne Doctrine'

## 5 Must China's leader wear a bow tie to the Queen's banquet?

## 6 Xi Jinping UK visit: History of China-British relations

## links

## 1 http://www.bbc.com/news/live/uk-34574590

## 2 http://www.bbc.com/news/uk-england-34586385

## 3 http://www.bbc.com/news/uk-34586679

## 4 http://www.bbc.com/news/world-asia-china-34539507

## 5 http://www.bbc.com/news/magazine-34576551

## 6 http://www.bbc.com/news/world-asia-china-34571946

##

## [[2]]

## [1] 70 2

These are some news articles collected from the BBC website, data consists of 70 rows and 2 columns, where the columns are simply the title of the news and its corresponding links (urls). We’ll be only be representing each news (document) with its title. Link to the data is provided at the end.

The following code :

1. Calculate the tf-idf score for this document collection.
2. Define our cosine distance.
3. Set this pre-defined cosine distance into R proxy library’s database ( backbone for the distfunction ) to calculate the pairwise distance matrix.
4. Performs hierarchical clustering and visualize the clustering result with a dendogram. Note that we WON’T be needing to normalize the tf-idf matrix before calculating the cosine distance, cosine distance will do that for us.

*#*

*# 1. [TFIDF] :*

*# @vector = pass in a vector of documents*

TFIDF <- function(vector) {

*# tf*

news\_corpus <- **Corpus**( **VectorSource**(vector) )

control\_list <- **list**(removePunctuation = TRUE, stopwords = TRUE, tolower = TRUE)

tf <- **TermDocumentMatrix**(news\_corpus, control = control\_list) %>% **as.matrix**()

*# idf*

idf <- **log**( **ncol**(tf) / ( 1 + **rowSums**(tf != 0) ) ) %>% **diag**()

**return**( **crossprod**(tf, idf) )

}

*# tf-idf matrix using news' title*

news\_tf\_idf <- **TFIDF**(news$title)

*# 2. [Cosine] :*

*# distance between two vectors*

Cosine <- function(x, y) {

similarity <- **sum**(x \* y) / ( **sqrt**( **sum**(y ^ 2) ) \* **sqrt**( **sum**(x ^ 2) ) )

*# given the cosine value, use acos to convert back to degrees*

*# acos returns the radian, multiply it by 180 and divide by pi to obtain degrees*

**return**( **acos**(similarity) \* 180 / pi )

}

*# 3. calculate pair-wise distance matrix*

pr\_DB$**set\_entry**( FUN = Cosine, names = **c**("Cosine") )

d1 <- **dist**(news\_tf\_idf, method = "Cosine")

pr\_DB$**delete\_entry**("Cosine")

*# 4. heirachical clustering*

cluster1 <- **hclust**(d1, method = "ward.D")

**plot**(cluster1)

**rect.hclust**(cluster1, 17)

After plotting the dendogram, I have decided that it should partitioned into 17 cluster ( Simply change it if you disagree ). A quick digression about the code chunck above. There’s already a built in cosine distance in the R’s proxy library. So you don’t have to define one yourself now that you’ve understand the implementation. Simply change the dist calculation to dist( news\_tf\_idf, method = "cosine" ).

We’ll examine three potential cluster that the algorithm provided and print out the original news’ title to determine whether the result matches our intuition.

*# split into 17 clusters*

groups1 <- **cutree**(cluster1, 17)

*# you can look at the distribution size of each cluster*

*# table(groups1)*

news$title[groups1 == 2 ]

## [1] " How do the Chinese view UK politicians? "

## [2] " Taiwan President Ma Ying-jeou quits as Kuomintang chief "

## [3] " Taiwan premier Jiang Yi-huah quits after poll loss "

## [4] " Taiwan: Chinese tourists flock to see elections "

## [5] " Taiwan backs Ma, but unease over China remains "

## [6] " Taiwan sweats on US arms sales decision "

## [7] " Taiwan in live-fire missile tests "

news$title[groups1 == 7 ]

## [1] " Yuwen Wu: What does China want from UK? "

## [2] " How China guards the Xi creation myth "

## [3] " China invests £5.2bn in UK projects "

## [4] " What does China own in the UK? "

## [5] " Obama: China cyber attacks 'unacceptable' "

news$title[groups1 == 17]

## [1] " Why is there tension between China and the Uighurs? "

## [2] " Who are the Uighurs? "

## [3] " How the Uighurs keep their culture alive in Pakistan "

Overall, news’ in the first cluster is mostly referring to something about Taiwan. The second cluster seems to be talking about China and UK, and the third cluster’s news are all related Uighurs. Not bad, huh? Given the fact that we’re only using news’ title instead of the entire news’ article to represent the news. Since clustering is an unsupervised algorithm (meaning there’re probably no such thing as a one hundred percent correct answer), I’ll leave it to you to decide whether the clustering results are actually acceptable.

One last thing before we wrap up this discussion, if you are to perform text clustering on you’re own, try not to use K-means. You can read why in this [StackOverflow](http://stackoverflow.com/questions/12497252/how-can-i-cluster-document-using-k-means-flann-with-python) (slide to the bottom).

# R Session Information

**sessionInfo**()

## R version 3.2.4 (2016-03-10)

## Platform: x86\_64-apple-darwin13.4.0 (64-bit)

## Running under: OS X 10.10.5 (Yosemite)

##

## locale:

## [1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8

##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base

##

## other attached packages:

## [1] dplyr\_0.5.0 proxy\_0.4-15 tm\_0.6-2 NLP\_0.1-9

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_0.12.5 rstudioapi\_0.6 knitr\_1.14 magrittr\_1.5

## [5] xtable\_1.8-2 R6\_2.1.2 stringr\_1.0.0 highr\_0.6

## [9] tools\_3.2.4 parallel\_3.2.4 DBI\_0.4-1 miniUI\_0.1.1

## [13] htmltools\_0.3.5 yaml\_2.1.13 assertthat\_0.1 digest\_0.6.9

## [17] tibble\_1.2 bookdown\_0.1 shiny\_0.13.2 questionr\_0.5

## [21] formatR\_1.4 evaluate\_0.9 mime\_0.4 slam\_0.1-32

## [25] rmarkdown\_1.1 stringi\_1.0-1 rmdformats\_0.3 httpuv\_1.3.3