WEB SEARCH Indexing:
Weighting terms for Index

CT102
Information Systems

PRE-PROCESSING SUMMARY

Indexing automatically scans the web page downloaded by the crawlers for the most important words and converts these to terms following a sequence of steps involving:

- case folding/change
- punctuation removal
- stop word removal
- stemming or lemmatisation

- These terms are then weighted and stored as the *representation of the web page in an index*

CALCULATING THE WEIGHTS OF TERMS

The abstraction is: the *meaning* of a document is represented using terms (derived from words in the document) and a weight for each term where:

- A weight is a binary or real number
- If binary: then we are representing the presence/absence of the term in a document
- Generally, we want something more representative that this where a real number represents the weight such that the higher the weight the more important the term is in describing the *meaning* of the document. One approach to calculate this weight is called: tf-idf:

  Term Frequency - Inverse Document Frequency

(tf) term frequency
(idf) inverse document frequency

- tf: If a term occurs very often in a document it is an important term describing the document (term frequency), e.g. the term shakespear was the most frequent term in the sample Shakespeare text (after pre-processing).
- df: However, if the term occurs often across all documents which are being searched* then it is not very useful at distinguishing one document from another (document frequency for term is high), e.g. if term shakespear occurs in many of the documents

* In the index

(idf) inverse document frequency

If df is high we want to reduce the weight of the tf component by a small amount

If df is low we want to increase the weight of the tf component

To do this we “inverse” the df value and use logs to ensure multiplication is only be a “small amount”, i.e. the df will not have a very large effect on the tf weight

NOTES:

- Stop words will have a high tf and df
- The weighting is performed on the terms remaining **after** stop word removal and stemming
- Thus if a term is not removed as a stop word, but it occurs frequently in most documents, it will get a low weight and not be considered important in determining the meaning of a web page/document (which is why we can use a small stop word list)

Calculating term frequency (tf)

To not penalise short documents, normalise (compare like with like) by dividing the raw count of the number of times the term occurs by the number of total terms in a document:

> Term frequency =
>
> Number times a term *t* occurs in document divided by the number of terms in the document

This ensures longer documents do not get an "unfair" advantage … essentially considering a ratio

Does this make sense? Why?

ASIDE (If it's not making sense):

- For a term *t* the term frequency can be a raw count of the number of times the term occurs in a document
- However, this is not ideal as a term is likely to occur more often in longer documents, thus longer documents would have an unfair advantage over shorter documents
- Thus it is the ratio of a terms occurrence we would like, **not** the raw count

EXAMPLE …

Given the following information for the term "*shakespear*" in 3 documents of different lengths find the term frequency:

Doc ID	**Frequency**	**Number of terms in doc**	***tf* = ?**
d1	10	20	
d2	10	200	
d3	100	2000	

Calculating the inverse document frequency (idf)

For a term t and N documents in the index with t occurring in df_t documents the inverse document frequency of t is defined as:

$$idf_t = \log 10\left(1 + \frac{N}{df_t}\right)$$

The idf of a rare term should be high, whereas the idf of a frequent term should be low.

To prevent multiplication by 0 the1 is added

NOTE: LOGS

A **logarithm** is the power to which a number must be raised in order to get some other number

e.g.,

If $\log_{10}x = y$ then $10^y = x$

$\log_{10}100 = ?$

On a calculator `log` is usually $\log_{10}$

Scientific calculator online:

https://web2.0calc.com/

(Sometimes natural log might be used in the formula but we will use log10)

EXAMPLE FOR TERM *shakespear* with $N = 100$ and $df = 40$

Note: idf stays the same for the same term

doc	**tf**	**idf** (with $\log_{10}$)	**tf*idf**
d1	0.5	$\log_{10}(1+100/40) = 0.544$	0.272
d2	0.05	0.544	0.0272
d3	0.05	0.544	0.0272

What if we had the same tf but this time were told …

N = 100 and df = 3 (i.e. these are the only 3 documents that have the term)

doc	**tf**	**idf** (with $\log_{10}$)	**tf*idf**
d1	0.5	$\log_{10}(1+100/3) = 1.5357$	0.7679
d2	0.05	1.5357	0.07679
d3	0.05	1.5357	0.07679

Therefore, tf*idf for a term t a document d:

- has the highest weight when t occurs many times in d and occurs a small number of times in all documents (thus lending high discriminating power to those documents)
- has a lower weight when t occurs fewer times in d, or occurs in many documents
- has the lowest weight when t occurs in nearly all documents and does not occur frequently in d

You try it . . .

Compute the tf-idf weights for the words:

- sql, databases, programming, computer

with corresponding terms:

sql, database, program, comput

for each of 3 documents using the following information:

* Use precision to 3 decimal places

Frequency of terms in docs

	d1 (length 90)	d2 (length 100)	d3 (length 50)
sql	12	4	7
database	3	13	0
program	0	13	2
comput	6	0	3

Frequency of terms across 250 documents

Term	*Frequency in Collection (df)*
sql	81
database	67
program	192
comput	240

Fill in the tf-idf weights N = 250

	sql	database	program	comput
d1				
d2				
d3				

	d1 (length 90)	d2 (length 100)	d3 (length 50)
sql	12	4	7
database	3	13	0
program	0	13	2
comput	6	0	3

Term	*(df)*
sql	81
database	67
program	192
comput	240

Fill in the tf-idf weights …
(250 documents)

	sql	database	program	comput
d1	12/90 *log10(1+250/81)	3/90 * log10(1 + 250/67)	0	6/90 * log10(1 + 250/240)
d2	4/100 *log10(1+250/81)	13/100 * log10(1 + 250/67)	13/100 * log10(1+250/192)	0
d3	7/50 *log10(1+250/81)	0	2/50 * log10(1+250/192)	3/50 * log10(1+250/240)

	d1 (l=90)	d2 (l=100)	d3 (l=50)
sql	12	4	7
database	3	13	0
program	0	13	2
comput	6	0	3

Term	*df*
sql	81
database	67
program	192
comput	240

tf	*d1*	*d2*	*d3*
sql	0.13333	0.04	0.14
database	0.03333	0.13	0
program	0	0.13	0.04
comput	0.06667	0	0.06

term	**df**	**idf**
sql	81	0.611342975
database	67	0.67498446
program	197	0.355841297
comput	240	0.309984838
N = 250		

ANSWERS (to 4 decimal places):

tf*idf	d1	d2	d3
sql	0.0815	0.0245	0.0856
database	0.0225	0.0877	0
program	0	0.0463	0.0142
comput	0.0207	0	0.0186

AFTER THIS STAGE ....
we have:

For each web page ID (url) many:

`<term, weight>`

For each term in the collection (term) many:

`<doc, weight>`

So with this information df can be calculated (and stored) for any term

These are generally stored in a complex structure to aid *fast searching and matching (with 0s not stored generally)*

	sql	database	program	comput	...	...	...
d1	0.0815	0.0225	0	0.0207			
d2	0.0245	0.0877	0.0463	0			
d3	0.0856	0	0.0142	0.0186			
...							
....							
....							
df							

CLASS WORK ….
https://web2.0calc.com/

s1: Python is a very powerful programming language.

s2: Python is often compared to the programming languages Perl, Ruby, Scheme and Java.

s3: Python, Perl, Ruby, Scheme, Java - what's the difference and is Python the best?

Step1. Find the representation (sentences) after the pre-processing steps of case folding, punctuation removal, stop word removal (using stopwords2.txt) and stemming (using Porter's) have been formed

Step 2. Calculate tf*idf of all the terms remaining in the sentences after pre-processing. You can assume that the full document collection is the 3 sentences (to calculate df))

ANSWERS
STEP 1:

S1: python power program languag

S2: python compar program languag perl rubi scheme java

S3: python perl rubi scheme java differenc python best

ANSWERS: STEP 2 … CALCULATING idf with N =3

S1: python power program languag

S2: python compar program languag perl rubi scheme java

S3: python perl rubi scheme java differenc python best

term	df	idf	idf
python	3	log10(1+3/3)	0.301029996
power	1	log10(1+3/1)	0.602059991
program	2		0.397940009
languag	2		0.397940009
compar	1		0.602059991
perl	2		0.397940009
rubi	2		0.397940009
scheme	2		0.397940009
java	2		0.397940009
difference	1		0.602059991
best	1		0.602059991

ANSWERS: STEP 2 …

S1: python power program languag

S2: python compar program languag perl rubi scheme java

S3: python perl rubi scheme java differenc python best

Calculating tf*idf with N =3 for terms python and program (to 4 decimal places)

term = python	tf	tf*idf	tf*idf
s1	¼	¼ *0.301029996	0.0753
s2	1/8	1/8 * 0.301029996	0.0376
s3	2/8	2/8 * 0.301029996	0.0753

term = program	tf	tf*idf	tf*idf
s1	¼	¼ * 0.397940009	0.0995
s2	1/8	1/8 * 0.397940009	0.0497
s3	0		0

	s1	s2	s3
python	0.0753	0.0376	0.0753
power			
program	0.0995	0.0497	0
languag			
compar			
perl			
rubi			

SUMMARY

- A web search index is built based on term weights which are calculated after pre-processing steps have been performed
- A commonly used weighting scheme is tf-idf (and variations)
  - For tf we must know the raw count of the term (frequency) and the total number of words in the document
  - For idf we must know the number of documents in the collection and the count of how many of these contain the term