Retrieval Models

June 8, 2015

1 Documents and query representation

1.1 Term incidence matrix

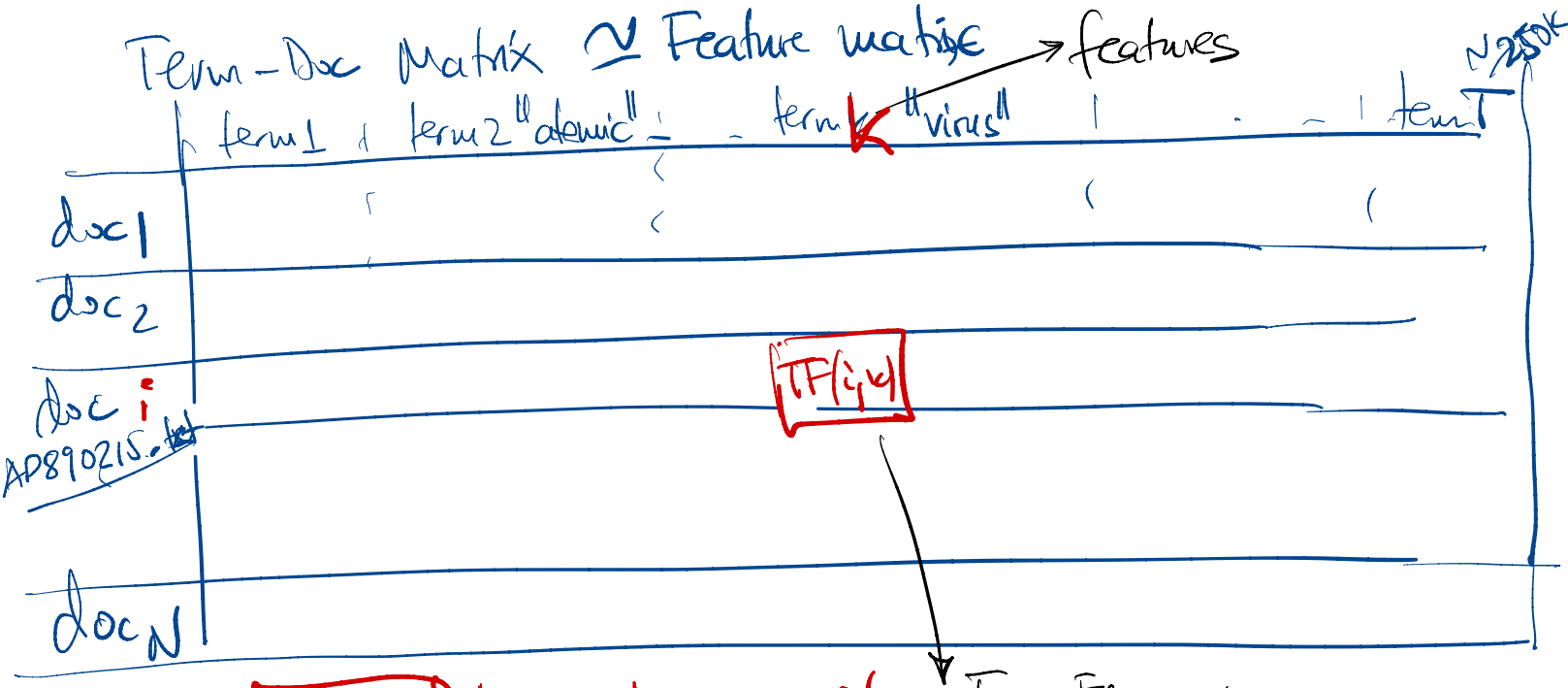
A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to terms in the collection and columns correspond to documents (or vice versa). There are various schemes for determining the value that each entry in the matrix should take [1]. If you look at the matrix on the bottom, you can see some of the plays of William Shakespeare as columns, and terms found in those plays as rows. For instance, the word "Antony" appears in Antony and Cleopatra, Julius Caesar, and Macbeth, but not in any of the others:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

► **Figure 1.1** A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise.

Figure 1: Term Incidence Matrix for the William Shakespeare plays

[1] Usually, boolean flags which is used in Boolean Matching Model but other attributes could be used as the values as well like term frequency - i.e. number of times a particular term occurs in the document (plays in the above example); or inverse term frequency (discussed later)



datapoints
$N \simeq 85K$ (HW1)
reality $N \gg$ JB static for dynamic

Sparse Most values are 0!
dont explicitly write "0"
dense (not allowed)

Term Frequency of term k in doc i
$\simeq$ count?
$= \frac{count}{DL}$?
$= \frac{count + \alpha}{DL + B}$? Smoothing.

$= \frac{Curve(count) + \alpha}{DL + \beta}$ ← Sublinear

1st occ ++
2nd/3rd occ ++
(
15th occ --

1.2 Retrieval Models

The basic form of retrieval model can be represented as below. It is basically comprised of the below sub parts

- Document Store : Index, documents, records, images etc. Basically the data which could be potentially retrieved.
- Query : These are the inputs which could be free-text, boolean keywords, formal query language statements etc.
- Matching Rule : This is basically the scoring model to check the similarity between documents and the query.
- Retrieval Results : Usual output which contains the top matching results.

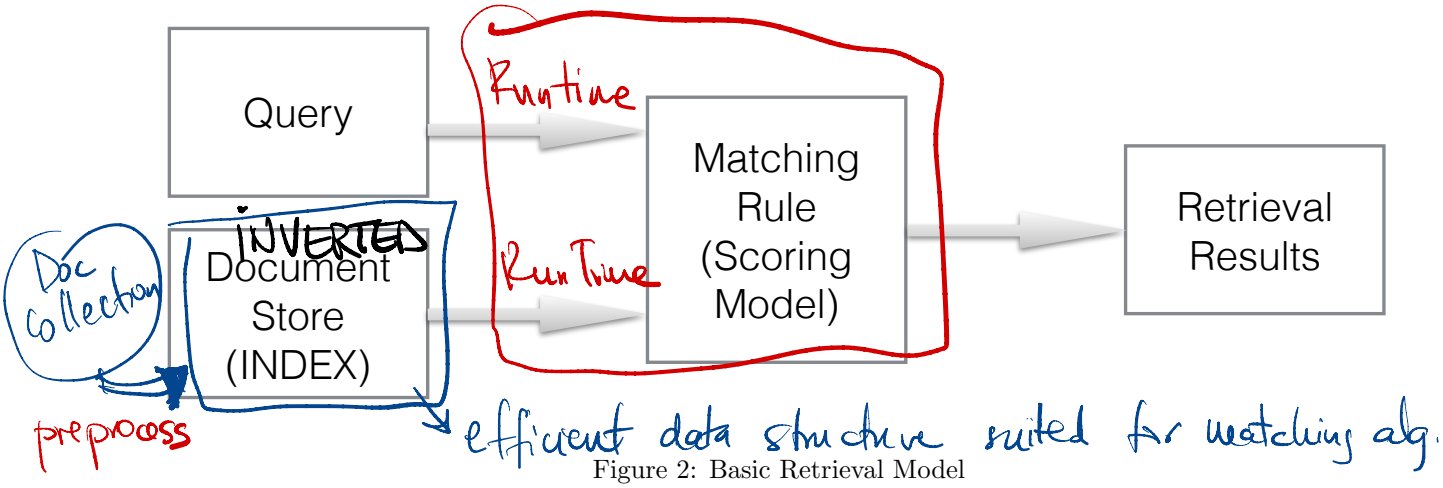
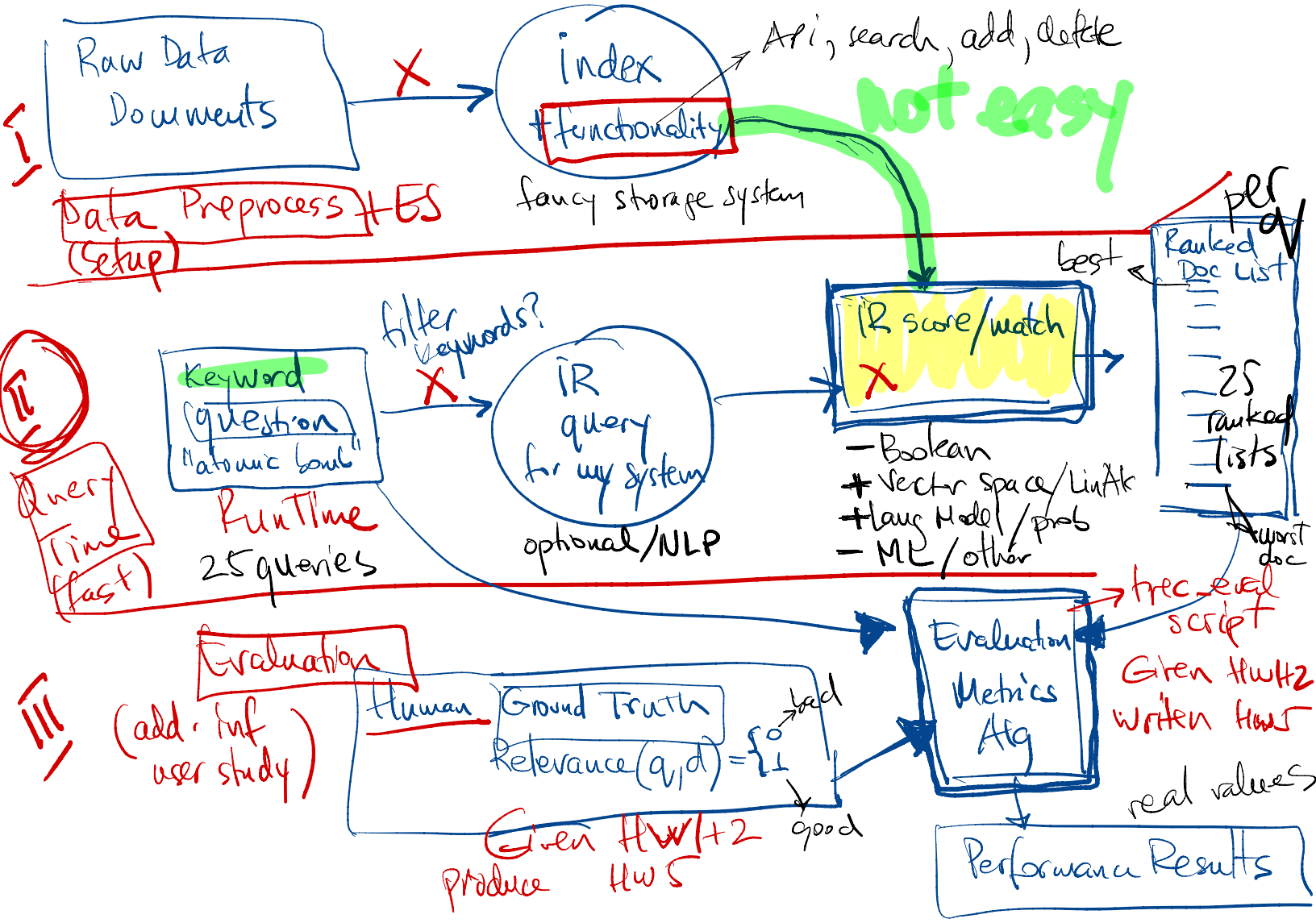


Figure 2: Basic Retrieval Model

1.3 Bag of words representation + inverted lists

By representing documents just as "bag of words" the order of words makes no difference, and there is no concept of meaningful phrases. This representation loses the relative ordering of the terms in each document, for example *Mary is quicker than John* and *John is quicker than Mary* are identical in such a bag of words representation. Vector Space Model use this concept where only the occurrence of word in a document is considered. In VSM the similarity between document and a query mainly depends on the following factors (none of which signify any correlation between the terms, hence called bag of words/terms):

1. TF : the number of occurrences of a term in a document. This reflects an intuition that a term which appears more often is more central to the document.
2. DF : the term?s document frequency is the number of documents in the collection which contain term t.
3. DLength : the length of the document. It needs to be normalized because with a naive matching score, such as the dot product, longer documents have an unfair advantage due to term repetition (see the details of document normalization below).
4. AVG(DLength) : Average of all the document lengths in the collection.
5. V : Vocabulary of all the distinct words in the collection (words left after stemming and stopping, discussed below)
6. N : Total number of documents in the collection.



keywords

good (useful for IR matching): names, locations, companies → "covid" A++

"stomach" "leg" → "basketball" "lasagna" "atomic" → A+

dates format? → A-

"sports" "news" "entertainment" "movie" → B+

when, where, why → B

"good" "sad" "fancy" → B-/C

bad

Stopwords "the" "a" "of" "about", ... → F

7. IDF : this is inverse document frequency which is obtained by multiplying the TF score by the logarithm of N divided by the DF

$$IDF = TF * \log \frac{N}{DF} \tag{1}$$

2 Retrieval Scoring Function, Matching

Retrieval Scoring Function helps to match the best documents for a given query and they are based on the notion of similarity. One example example is semantic matching which measures the similarity of meaning between two texts, e.g. a query and a document.

match(q, doc) = { 1 if doc contains all query terms
0 if not

2.1 Boolean Retrieval

not good for production
good for exploration of data

In this model, queries are Boolean expressions. Words match only themselves, and complex information needs are expressed by building complex queries (i.e. by logical combination of the boolean results). A keyword matches itself, and only itself. That means we know nothing about synonyms or other nuances of a language. In order to express a complex information need, the user will have to build a complex query that combines a lot of keywords using Boolean operators like AND and OR. Boolean Retrieval Model uses a term incidence matrix as the data structure to keep track of which keywords apply to which documents.

Here is an example of the Boolean queries:

Information need: *Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company.*
Query: *"trade secret" /s disclos! /s prevent /s employe!!*

Information need: *Requirements for disabled people to be able to access a workplace.*
Query: *disab! /p access! /s work-site work-place (employment /3 place)!*

Information need: *Cases about a host's responsibility for drunk guests.*
Query: *host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest*

These systems were mainly used by expert searchers who were trained in the custom query language used by the search software. There are a few interesting query language features on display. Let's look at the first query. First, you can surround a phrase with quotation marks. Also, if you look at the second and the last keywords, you can see the prefix of a word followed by an exclamation point. This is used to refer to any word that starts with that prefix. It's a simple approach to what's called stemming.

Finally, these queries contain a lot of proximity operators. The slash-s means that the terms on either side should be found in the same sentence, and slash-p means they should be found in the same paragraph. There is a slash-3 in the second query which means that the terms should be found within three words of each other. So these languages can get fairly complicated. There are quite a few other operators people have built into query languages.[2]

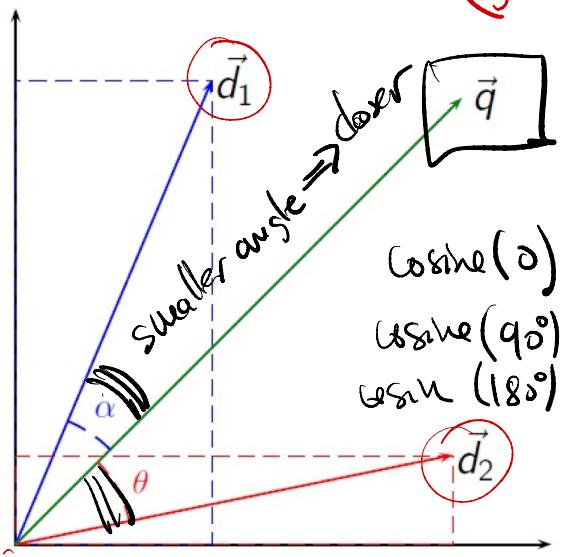
- **Advantages**
  - Boolean Retrieval is a simple model and easy to implement. Its users feel they have control over the system, and can build up complex queries iteratively to refine the results.
  - Boolean Retrieval has been commercially successful, with products surviving for decades.

[2]The textbook by Croft has a section on a query language for a search engine called Galago if you?re interested in seeing what a modern query language looks like.

– It has mainly been used by trained searchers in vertical search and enterprise search engines, e.g. in the leading legal and journalistic search engines from the 1970s on. [3] [4]

- **Disadvantages**
  - Since it's based on set membership, a document either matches the query or doesn't. There is no way for some documents to match more than others.
  - It also has little flexibility to allow us to try to match the information need. We often want to refine the submitted query before we run it, but the complex operators in Boolean Retrieval queries make that difficult.
  - It is helpful to use simpler queries in order to support more complex processing behind the scenes. The computer should generally do more work than the user.

3 Vector Space Models



- q is the query vector
- d_1 is the vector for document 1
- d_2 is the vector for document 2
- α is the angle between the query vector and document 1 vector
- θ is the angle between the query vector and document 2 vector

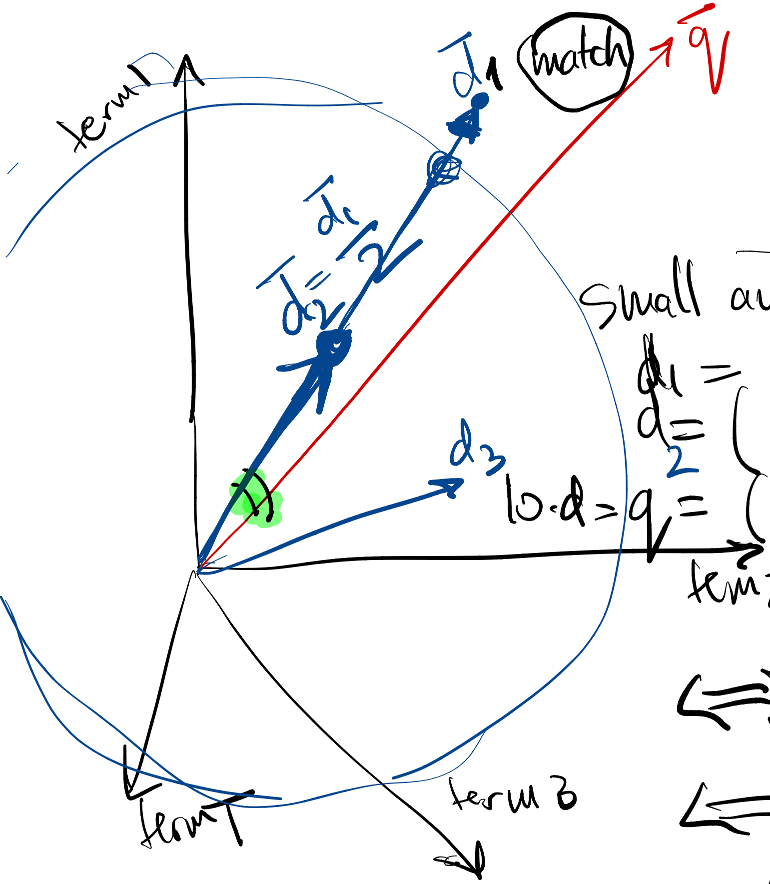
The vector space model uses a nice mathematical generalization of query/document matching which, unlike the set-based Boolean Retrieval model, and has a lot of flexibility for tuning retrieval performance.

Recall the term incidence matrix above, we can also represent keyword queries as vectors. From this point of view, queries are just "short documents." We can use term frequencies as the values in the matrix since it reflects an intuition that a term which appears more often is more central to the document.

VSM was introduced to move beyond the simple binary set membership world, which is perfectly captured by ones and zeros, and into a more nuanced world where documents can be just a little relevant or highly relevant. When we match a given term from a query, we'd like the documents scores for that query term to be higher if those documents are more likely to be good matches. If the query has multiple terms, we'll combine the scores from all its terms so that documents which match all terms will end up with higher scores than documents that just match one.

[3] This happened somewhat to the frustration of IR researchers, whose more advanced matching models took years to become widely adopted.

[4] Boolean Retrieval was the dominant IR system in commercial software for decades, from the 60s and 70s until it was finally replaced by more sophisticated semantic matching systems in the 90s as the Internet gained in popularity. This was a little bit of an ?I told you so? moment for the IR research community, whose improved models had been around for quite some time by then.



∢angle(q, d) small ⟺

⟹ doc is relevant (usefull) for the q

Small angle ⟺ proportional coordinates

	china	algebra	virus	disks	underground
d_1 =	2	0	4	0	10
d_2 = (	1	0	2	0	5)
$10 \cdot d = q$ = (	10	0	20	0	50)

$\measuredangle = 0$

⟺ same keywords with same proportion

⟺ doc relevant for query

(independent of word ordering in d, q)

Alternative for cosine? L_1 norm (dif) $= \sum_t |d^t - q^t|$

L_2 norm $= \sum_t |d^t - q^t|^2$

many-options math / linear algebra.
Manhattan, Edit Dist, Naive Dis,

Which one?
- go by the task, results / debug
- not primarily by math

Cosine: why normalize by length? good? yes usually

$\text{cosine}(d_1, q) = \text{cosine}(d_2, q)$ d_1, d_2 same line

long document with same keywords/prop, just repeated more times ⟹ not better usually if prop (d_1, d_2) satisfied

• alternative: normalize TF = 5 ⟶ normalized TF $\frac{5}{DL}$ → ?
vector component
doc length = # terms in doc

Matching Score - In order to sort documents by their relevance to a particular query, we will generate a query matching score's for each document and sort by that.

$$s =< Q, D >= Q.D \tag{2}$$

The matching score should be a function of two vectors which outputs a number. The number should be larger when the vectors express more similar topics.

Once we have our two vectors, we use an inner product [5] as a similarity function. The inner product function we choose defines what's called an "inner product space", such that vectors within that space use the inner product for their notion of distance. Intuitively, the closer two vectors are to each other, based on their inner product, the more semantic content they should have in common.

3.1 DLength Normalization

Storing raw TF (usually called as "TF") has some problems, because there is no normalization done for the occurrence of the words. For example, With a naive matching score, such as the dot product, longer documents have an unfair advantage due to term repetition.

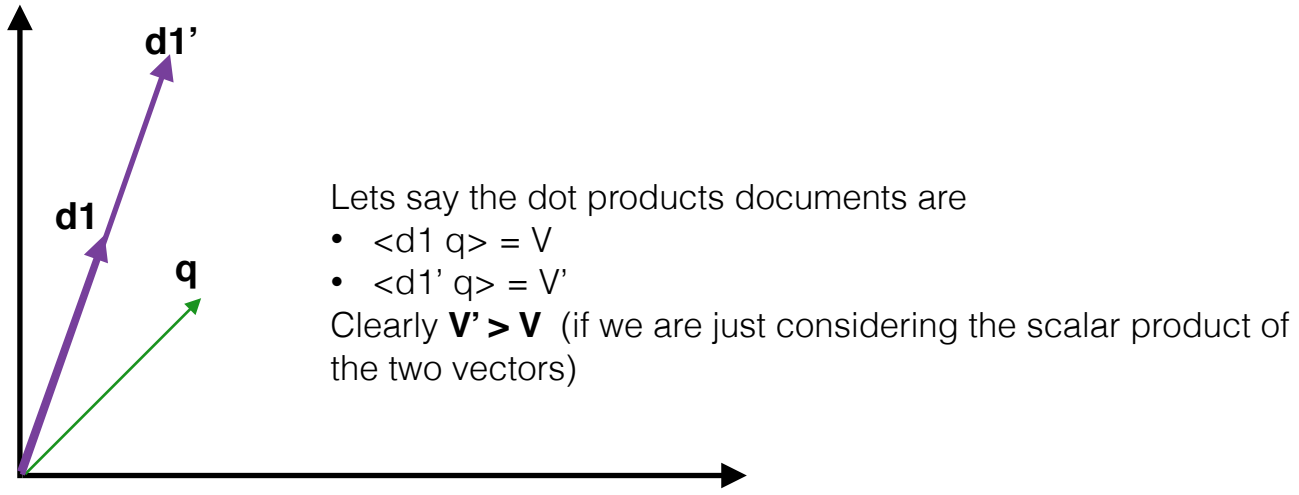


Figure 3: Drawback of considering raw TF without normalization

Very short documents may be less relevant (which might not be always true for example how about a collection of FAQ answers?), but very long documents may also be less relevant. To avoid this there a few ways to normalize term vectors:

- Instead of TF, consider the fraction of a document the term occupies.
- Take into account the number of distinct terms in the document to account for repetitive documents.

3.2 Okapi TF and other TF scaling values instead of TF

This is a vector space model using a slightly modified version of TF to score documents. The Okapi TF score for term w in document d is as follows.

$$okapi\_tf(w,d) = \frac{tf_{w,d}}{tf_{w,d} + 0.5 + 1.5 \cdot (len(d)/avg(len(d)))}$$

Where:

[5] "an inner product" because there are many functions which we could potentially use for this purpose. The simplest one is the dot product, but there are many more. It will be discussed later.

$tf_{w,d}$ is the term frequency of term w in document d $len(d)$ is the length of document d $avg(len(d))$ is the average document length for the entire corpus The matching score for document d and query q is as follows.

$$tf(d,q) = \sum_{w \in q} okapi\_tf(w,d)$$

This helps in normalizing the documents by taking into account the average document length across corpus and the giving less weight to longer documents.

3.3 Similarity of vectors

Similarity between two vectors can be expressed by various means:

- distances similarities:

$$dist(q,d) = \sqrt{\sum_t (q_t - d_t)^2}$$

$$sim(q,d) = \frac{1}{1 + dist(q,d)}$$

  Euclidean distance is the standard distance function you're probably used to. It considers the two vectors as points in a Euclidean space, and measures how far those points are from each other. It's the square root of the sum of the squared distances along each axis. This is exactly like Pythagorus' theorem, scaled up to any number of dimensions.

- dot products:

$$sim(q,d) = q \cdot d$$

  This just uses the scalar product of two vector. One reason this doesn't work well is that it unfairly favors repetitive documents. If you made a new play by copying Julius Caesar twice, that new play would match everything the original play matched, but twice as much.This similarity function is fast and works fairly well, so it does get used sometimes. However, there's a more principled way to compare these vectors.

Play	TF	Similarity
Henry VI, part 2	1	2.34
Hamlet	1	2.34
Antony and Cleopatra	4	9.38
Coriolanus	109	255.65
Julius Caesar	379	888.91
Julius Caesar x 2	758	1777.83
Julius Caesar x 3	1137	2666.74

Figure 4: Dot Product Similarity Scores

- cosine similarity:
  Cosine Similarity solves the problems of both Euclidean-based similarity and the dot product.
  - Instead of using distance between the vectors, we should use the angle between them.

– Instead of using the dot product, we should use a length-normalized dot product. That is, convert to unit vectors and take their dot product.

$$sim(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

$$= \frac{u \cdot v}{\sqrt{\sum_i u_i^2} \cdot \sqrt{\sum_i v_i^2}}$$

$$= \frac{u}{\sqrt{\sum_i u_i^2}} \cdot \frac{v}{\sqrt{\sum_i v_i^2}}$$

Cosine Similarity solves the problems with our prior two similarity functions. It?s based on the intuition that the magnitude of a term vector isn?t very important. What matters is which terms show up in the vector, and what their relative sizes are. In other words, what's the term vector's angle. This function uses the angle between two vectors as the distance between them, and totally ignores their relative lengths.

Play	TF	Similarity
Henry VI, part 2	1	0.002
Antony and Cleopatra	4	0.004
Coriolanus	109	0.122
Julius Caesar	379	0.550
Julius Caesar x 2	758	0.550

Figure 5: Cosine Similarity Scores

If we look at the same plays with Cosine Similarity, we get exactly the results we want. Henry VI is the worst match, and Julius Caesar the best. Double Julius Caesar doesn?t change its matching score at all, because it just changes the vector?s magnitude and we're ignoring that. However, this function isn't perfect either.

Approximating Cosine Similarity: The normalization term for cosine similarity can't be calculated in advance, if it depends on $df_t or cf_t$.
For faster querying, we sometimes approximate it using the number of terms in the document. This preserves some information about relative document length, which can sometimes be helpful.

$$sim(q, d) \approx \frac{q}{\sqrt{len(q)}} \cdot \frac{d}{\sqrt{len(d)}}$$

3.4 TF-IDF

We now combine the definitions of term frequency and inverse document frequency, to produce a composite weight for each term in each document. The main idea is that we want term scores to be proportional to TF, but we want to discount scores for too-common terms. Two common ways to discount:

- A term's cumulative frequency cf_t is total number of occurrences of term t in the collection.
- The term's document frequency df_t is the number of documents in collection which contain term t.

Term	Doc	$tf_{t,d}$	df_t	cf_t	$tf_{t,d}$ / df_t	$tf_{t,d}$ / cf_t	tf-idf$_{t,d}$
and	King Lear	737	37	25,932	19.92	0.028	0
love	Romeo and Juliet	150	37	2,019	4.05	0.074	0
rome	Hamlet	2	16	332	0.125	0.006	1.68
rome	Julius Caesar	42	16	332	2.625	0.127	35.21
romeo	Romeo and Juliet	312	1	312	312	1	1126.61

Figure 6: Various term score functions including tf-idf

The most common way to discount is to multiply by $\log(\frac{D}{df_t})$., where D is the number of documents in the collection. This is called IDF, for inverse document frequency, and leads to TF-IDF scores.

$$tf\text{-}idf_{t,d} := tf_{t,d} \cdot \log\left(D/df_t\right)$$

The term "and" shows up in every document. Its TF in King Lear is 737. If you divide that by the DF you get roughly 20, and by CF you get almost zero. Its TF-IDF score is zero, because it appears in every document.

The term "rome" shows up much more in Julius Caesar than in Hamlet, though it's much less common than ?and.? If we just discount by df or cf, it still has a lower score than ?and.? However, its TF-IDF score is higher for Hamlet, and much higher for Julius Caesar. That's exactly what we want.

As an extreme example, the term Romeo, which only shows up in a single play, has a very high TF-IDF score. That's perfect, because this term is a perfect feature in this corpus for finding plays about Romeo.

4 Term Concurrence

Concurrence[6] can be exploited to deduce important features out of the corpus by measuring the co-occurences. Though, here we're focusing on term co-occurrence, but these measures can be used for many other statistical tasks. For instance, you might be interested in which users have reviewed the same product, or which web pages link to the same URL.

Here are simplified versions of four different association measures. Mutual Information and Expected Mutual Information come from information theory, and Chi-squared tests and Dice?s Coefficient are from statistics.

It's worth mentioning that these formulas are not the full versions of these measures. However, they are rank-equivalent to the full formulas, so if you?re just using them to sort terms it?s faster and simpler to use these forms.

1. Dice's coefficient, aka the Sorensen index, is used to compare two random samples. In this case, we compare the population of documents containing terms a and b to the populations containing a and containing b.

[6]From Wikipedia definition: In Western jurisprudence, concurrence (also contemporaneity or simultaneity) is the apparent need to prove the simultaneous occurrence of both actus reus ("guilty action") and mens rea ("guilty mind"), to constitute a crime. In theory, if the actus reus does not hold concurrence in point of time with the mens rea then no crime has been committed.

Measure	Formula
Mutual Information (MIM)	$\frac{n_{ab}}{n_a \cdot n_b}$
Expected Mutual Inf. (EMIM)	$n_{ab} \cdot \log\left(N \cdot \frac{n_{ab}}{n_a \cdot n_b}\right)$
Chi-square (X^2)	$\frac{\left(n_{ab} - \frac{1}{N} \cdot n_a \cdot n_b\right)^2}{n_a \cdot n_b}$
Dice's coefficient (Dice)	$\frac{n_{ab}}{n_a + n_b}$

Figure 7: Measures of Co-Occurrence

Dice's coefficient imagines that we have random samples of two events: the event that term a occurs in a document, and the event that term b occurs. It compares how often these events occur together to the total number of times either event occurs.

Let's take a look at the simplified formula $n_{ab}/n_a + n_b$: n_{ab} is the number of documents in the index which contain both term a and term b. n_a is the number of documents that contain term a, and n_b is the number of documents containing term b. If a and b always occur together, then n_{ab} is going to equal n_a and n_b, and this formula will equal 1/2. The more they occur without each other, the bigger $n_a + n_b$ will become relative to n_{ab}, so the smaller the number will get.

2. Pointwise mutual information is a measure of correlation from information theory.

   It measures how correlated two random events are to each other. That is, if you know that term a appears in a document, how much information does that give you about whether term b will appear?

$$\begin{aligned} pmi(a,b) &:= \log\left(\frac{p(a,b)}{p(a)p(b)}\right) \\ &= \log\left(\frac{\frac{n_{ab}}{N}}{\frac{n_a}{N}\frac{n_b}{N}}\right) \\ &= \log N + \log\frac{n_{ab}}{n_a n_b} \\ &\overset{\text{rank}}{=} \frac{n_{ab}}{n_a n_b} \end{aligned}$$

   If we use base 2 for the logarithm, the amount of information is measured in bits. It's going to have the largest magnitude when knowing whether term a appears in a document lets you predict with perfect accuracy whether term b appears, and be closest to zero when they're totally independent.

   The rank-equivalent formula on the bottom of the slide is similar to Dice's coefficient, except that the denominator grows faster as the terms appear without each other more often. This produces a different ordering that punishes terms much more harshly for not co-occurring as often. You could argue that it punishes them too harshly.

3. Expected mutual information corrects a bias of pointwise mutual information toward low frequency terms.

$$
\begin{aligned}
emim(a,b) \propto & P(a,b) \cdot \log \frac{P(a,b)}{P(a)P(b)} \\
= & \frac{n_{ab}}{N} \log \left(N \cdot \frac{n_{ab}}{n_a \cdot n_b} \right) \\
\overset{\text{rank}}{=} & n_{ab} \cdot \log \left(N \cdot \frac{n_{ab}}{n_a \cdot n_b} \right)
\end{aligned}
$$

It's still measuring the level of dependence between the two random events, but this function is smoother and gives larger values for the lower-frequency terms. You can still see our approximation of mutual information inside the log function. We're essentially just scaling this up by multiplying by n_{ab} outside the logarithm, so that we tend to pay more attention to co-occurrences even when one of the two terms is relatively infrequent compared to the other.

4. Pearson's Chi-squared test is a test of statistical significance which compares the number of term co-occurrences to the number we'd expect if the terms were independent. (This is also not the full form of this measure.)

$$
\begin{aligned}
chi2(a,b) = & \frac{\left(n_{ab} - N \cdot \frac{n_a}{N} \cdot \frac{n_b}{N} \right)^2}{N \cdot \frac{n_a}{N} \cdot \frac{n_b}{N}} \\
\overset{\text{rank}}{=} & \frac{\left(n_{ab} - \frac{1}{N} \cdot n_a \cdot n_b \right)^2}{n_a \cdot n_b}
\end{aligned}
$$

This is a statistical significance test that's used to measure whether the co-occurrences happen by chance, or whether they happen because the words are really related to each other. We?ll talk more about significance testing in the module on evaluation. For now, it's enough to point out that the denominator is the same as for pointwise mutual information, but the numerator is looking at the squared difference between the number of co-occurrences and something related to how often each of the terms occurs on its own. If the two terms are very common, we expect the co-occurrences are more likely to happen by chance. If they?re very rare, but always happen with each other, they are more likely to be related. That's the key insight behind this measure.

Now let's take a look at how these four measures perform on real data to get a sense of what they do differently.

Association Measure Example

Suppose the user has run the query "tropical fish", and we want to expand the query by adding terms related to each of the query terms. These tables show the 15 highest-ranking terms with each of the four association measures, for each of the two query terms.

The distributions for point wise mutual information, labeled MIM here, and for chi^2, are fairly similar. The distributions for expected mutual information and dice?s coefficient resemble each other more than the other two. Well, MIM and chi^2 are focusing on the lowest-frequency terms that tend to appear with the query terms. The words in this list tend to be very rare, and to almost always appear with the query term when they do appear. That isn't necessarily what we want: these terms might be too infrequent to point out relevant documents. They are also rare enough that the terms for the query term "tropical" are unlikely to show up in documents about the query term "fish", and vice versa.

The lists for EMIM and Dice's coefficient are better: they don't focus as much on extremely rare terms. They still have the problem, though, that they aren't related to the overall query. They are focused on one particular query term at the expense of the other.

MIM	*EMIM*	χ^2	*Dice*
trmm	forest	trmm	forest
itto	tree	itto	exotic
ortuno	rain	ortuno	timber
kuroshio	island	kuroshio	rain
ivirgarzama	like	ivirgarzama	banana
biofunction	fish	biofunction	deforestation
kapiolani	most	kapiolani	plantation
bstilla	water	bstilla	coconut
almagreb	fruit	almagreb	jungle
jackfruit	area	jackfruit	tree
adeo	world	adeo	rainforest
xishuangbanna	america	xishuangbanna	palm
frangipani	some	frangipani	hardwood
yuca	live	yuca	greenhouse
anthurium	plant	anthurium	logging

Most associated terms for "tropical" in a collection of TREC news stories.

MIM	*EMIM*	χ^2	*Dice*
zoologico	water	arlsq	species
zapanta	species	happyman	wildlife
wrint	wildlife	outerlimit	fishery
wpfmc	fishery	sportk	water
weighout	sea	lingcod	fisherman
waterdog	fisherman	longfin	boat
longfin	boat	bontadelli	sea
veracruzana	area	sportfisher	habitat
ungutt	habitat	billfish	vessel
ulocentra	vessel	needlefish	marine
needlefish	marine	damaliscu	endanger
tunaboat	land	bontebok	conservation
tsolwana	river	taucher	river
olivacea	food	orangemouth	catch
motoroller	endanger	sheepshead	island

Most associated terms for "fish" in the same collection.

Improving the Results: A few tricks can help us focus on terms more related to the overall query. They mainly involve filtering out some of the co-occurrences in order to focus on the ones we think really matter.

First, we want to filter terms out of the list that occur in the same documents by accident. We can accomplish this by only counting co-occurrences which happen within some fixed distance of the query term. So when we?re looking for terms related to "fish", we?ll only consider co-occurrences within 10 words of each occurrence of "fish" in each document from our collection. The intuition here is that the document might be talking about a lot of things, but the words closest to the query term are most likely to be closely related to its topic. The table on the right shows an improved list for "fish" that only counts co-occurrences which occur within a window of 5 terms from the word "fish".

As a second improvement, we can just count co-occurrences when they occur simultaneously for all query terms. If we use Dice's coefficient on both terms, "tropical" and "fish", we get a much better list of terms for query expansion (explained later in 8).

- Instead of counting co-occurrences in the entire document, count those that occur within a smaller window.
- Look for new terms associated with multiple query terms instead of just one.
- Using Dice with "tropical fish" gives the following list: goldfish, reptile, aquarium, coral, frog, exotic, stripe, regent, pet, wet.

MIM	*EMIM*	χ^2	*Dice*
zapanta	wildlife	gefilte	wildlife
plar	vessel	mbmo	vessel
mbmo	boat	zapanta	boat
gefilte	fishery	plar	fishery
hapc	species	hapc	species
odfw	tuna	odfw	catch
southpoint	trout	southpoint	water
anadromous	fisherman	anadromous	sea
taiffe	salmon	taiffe	meat
mollie	catch	mollie	interior
frampton	nmf	frampton	fisherman
idfg	trawl	idfg	game
billingsgate	halibut	billingsgate	salmon
sealord	meat	sealord	tuna
longline	shellfish	longline	caught

Most associated terms for "fish" with co-occurrences measured in a window of 5 terms.

5 BM25

Binary Independence Models

In Bayesian classification, documents are ranked by their **likelihood ratios** $\frac{P(D|R=1)}{P(D|R=0)}$ calculated from some

probabilistic model.

Binary Independence Model = $\frac{\prod_{i=1}^{n} P(w_i|R=1)}{\prod_{i=1}^{n} P(w_i|R=0)}$

The model predicts the features that a relevant or non-relevant document is likely to have. Our first model is a unigram language model, which independently estimates the probability of each term appearing in a relevant or non-relevant document. Any model like this, based on independent binary features $f_i \in F$, is called a binary independence model.

BM25 is a language model based on a binary independence model. Its matching score is as follows.

$$bm25(d,q) = \sum_{w \in q} \left[\log\left(\frac{D+0.5}{df_w+0.5}\right) \cdot \frac{tf_{w,d} + k_1 \cdot tf_{w,d}}{tf_{w,d} + k_1\left((1-b) + b \cdot \frac{len(d)}{avg(len(d))}\right)} \cdot \frac{tf_{w,q} + k_2 \cdot tf_{w,q}}{tf_{w,q} + k_2} \right]$$

Where:
$tf_{w,q}$ is the term frequency of term w in query q k_1, k_2, and b are constants.

Okapi BM25 is one of the strongest "simple" scoring functions, and has proven a useful baseline for experiments and feature for ranking. It combines:

- The IDF-like ranking score
- the document term frequency $tf_{i,d}$, normalized by the ratio of the document?s length dl to the average length avg(dl), and
- the query term frequency $tf_{i,q}$.

Is it Better?
Let's unpack this formula to understand it better- The numerator is a ratio of counts of relevant documents the term does and does not appear in. It's a likelihood ratio giving the amount of "evidence of relevance" the term provides.

The denominator is the same ratio, for non- relevant documents. It gives the amount of "evidence of non-relevance" for the term.
NOTE: If the term is in many documents, but most of them are relevant, it doesn't discount the term as IDF would.

6 Language Models

*probabilities as matching evidence