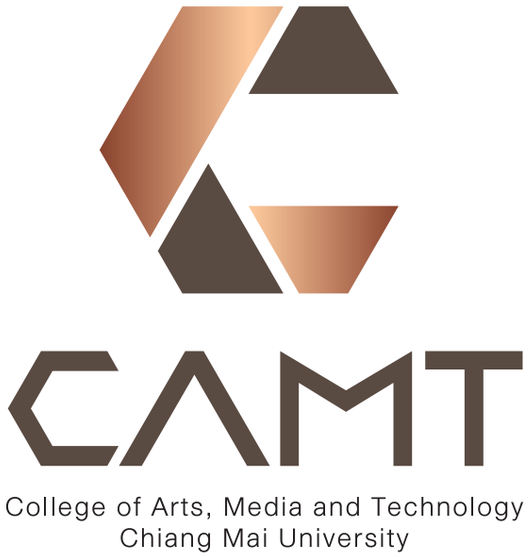
SE 481 Introduction to Information Retrieval

Module #3 — IR Models



Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

- Some histories
- Scoring documents
- Term frequency (tf)
- Vector space model

Some histories

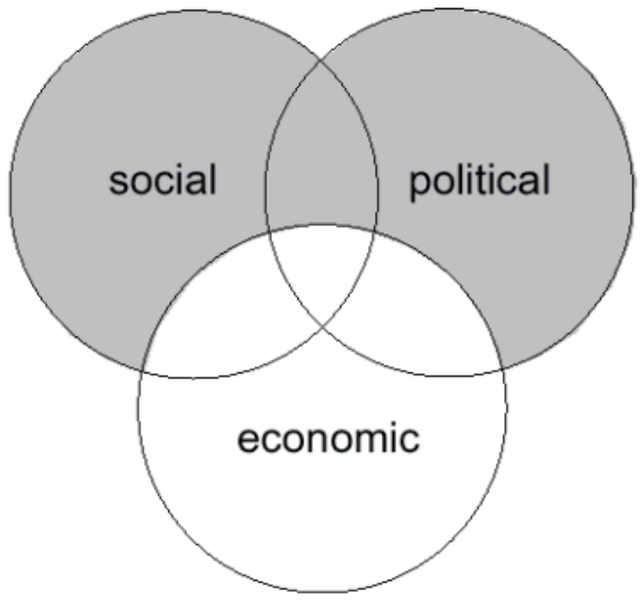
- Boolean model (±1950)
- Document similarity (±1957)
- Vector space model (±1970)
- Probabilistic retrieval (±1976)
- Language models (±1998)
- Google PageRank (±1998)

Boolean retrieval

- Boolean = yes no question (match or not match)
- It cannot measure whether **document A** is more relevant to the query than **document B**
  - Not sensible for when a query returns 1000++ matched results
- Exact matching: data retrieval (instead of information retrieval)
- Practically difficult
  - **AND** gives too few results; **OR** gives too many results

Boolean retrieval

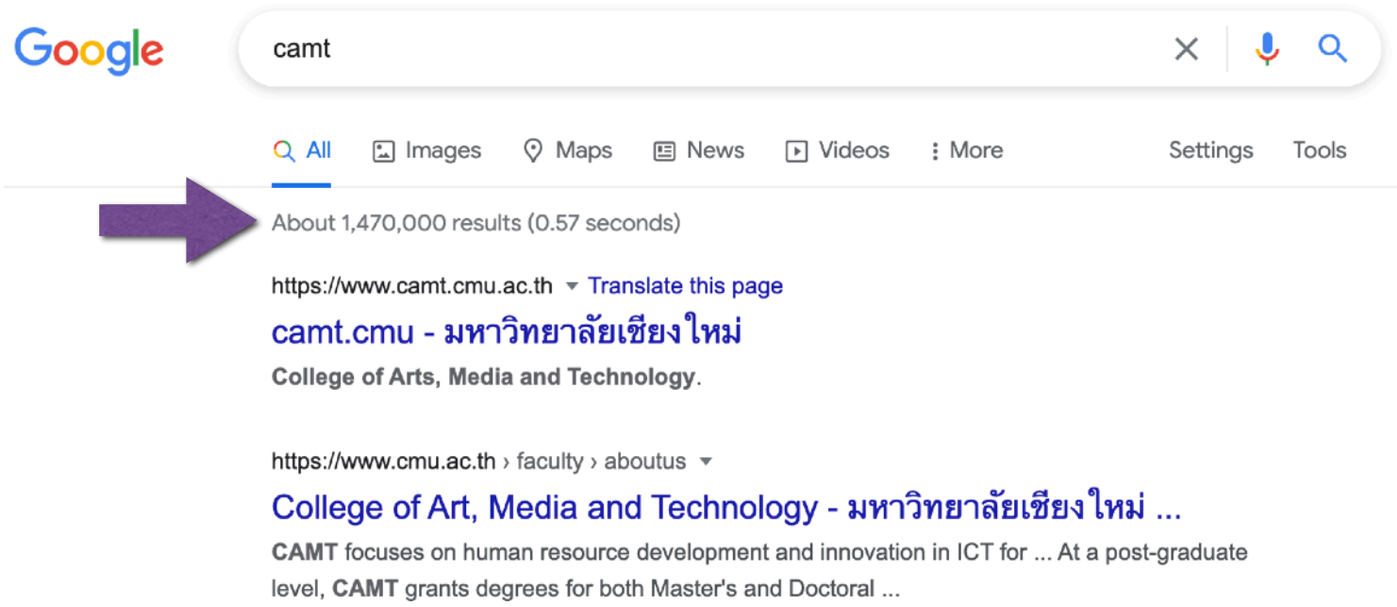
- Venn diagrams



(social OR political)
NOT economic

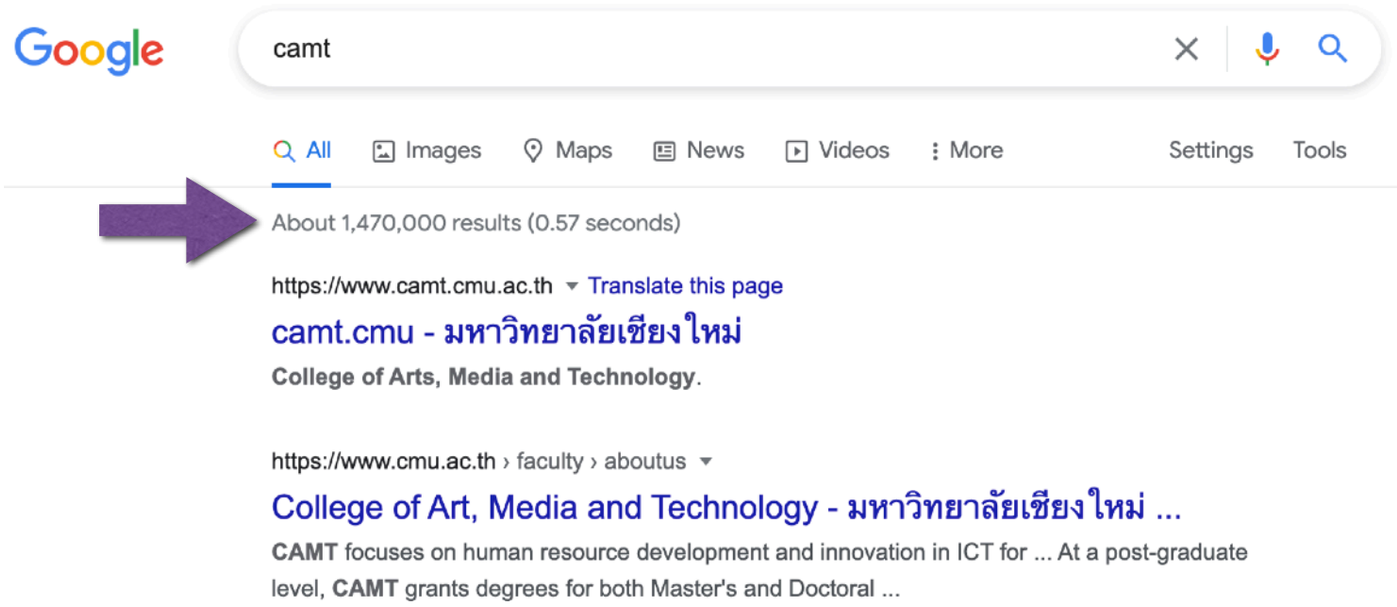
Ranked retrieval models

- Score is also returned so we can order what should be considered as best matched results.
- Large result sets are not an issue
  - We can just show the top k (≈ 10) results



What do we need?

- Ranking (scoring) algorithm
  - Assign a score – [0, 1] – to each document
  - This score say how well document and query match
  - In other words, it is a Query-document matching score



Query-document matching scores

- Keys
  - We need an approach to assign a score to a query/document pair
  - If the query term does not occur in the document: score should be 0
  - The more relevant the query to the documents, the higher the scores should be, e.g.,
    - higher score —> more frequent the query term in the document
    - Any other alternatives?

Query-document matching scores

- Recall a binary representation
  - E.g., Binary term-document incidence matrix

	0	1	2	3	4	5	6	…	280	281	282	283	284	285	286
0	0	1	1	1	1	0	1	…	1	1	1	0	1	1	0
1	1	0	0	0	0	1	0	…	0	1	0	1	1	0	1

Query-document matching scores

- Term-document count matrices
  - Consider the number of occurrences of a term in a document:

	0	1	2	3	4	5	6	...	280	281	282	283	284	285	286
0	0	1	1	1	1	0	1	...	1	2	1	0	5	1	0
1	1	0	0	0	0	1	0	...	0	1	0	1	1	0	1

  - This is also known as a Bag of words model

A suggested Scikit-learn apprach

```
01 def preProcess(s):
02     ps = PorterStemmer()
03     s = word_tokenize(s)
04     stopwords_set = set(stopwords.words())
05     stop_dict = {s: 1 for s in stopwords_set}
06     s = [w for w in s if w not in stop_dict]
07     s = [ps.stem(w) for w in s]
08     s = ' '.join(s)
09     return s
10
11 def sk_vectorize():
12
13     cleaned_description = m1.get_and_clean_data()
14     vectorizer = CountVectorizer(preprocessor=preProcess)
15     vectorizer.fit_transform(cleaned_description)
16     query = vectorizer.transform(['good at java and python'])
17     print(query)
18     print(vectorizer.inverse_transform(query))
18
18 sk_vectorize()
```

Bag of words

- Bag of words
  - Vector representation doesn't consider the ordering of words in a document.
  - *C++ is much faster than Python* and *Python is much faster than C++* are represented by the same vectors.
  - Naive by works well when #documents increase unlimited.

Ref: https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/

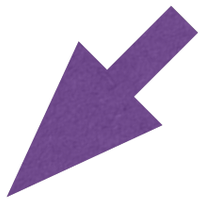
A suggested Scikit-learn apprach

```
19  query = vectorizer.transform(['good at java and python'])
20  print(query)
21  print(vectorizer.inverse_transform(query))
```

```
1   query = vectorizer.transform(['good at python and java'])
2   print(query)
3   print(vectorizer.inverse_transform(query))
```

- Bag of words >> this generates the same result.

Handing sequence



```
1   vectorizer = CountVectorizer(preprocessor=preProcess,ngram_range=(1, 2))
2   X = vectorizer.fit_transform(cleaned_description)
3   print(vectorizer.get_feature_names())
```

Term frequency (tf)

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
  - Note that frequency in IR refers to #counts
- Anyway using just a raw *tf* does not fully making sense
  - 10 times higher occurrences does not imply 10 times more relevant
  - Practical solution — Applying weighting

Term frequency (*tf*) with *log* weighting

$$w_{t,d} = 1 + log_{10}(tf_{t,d}) \text{ if } tf_{t,d} > 0$$

- Score for a document-query pair:
  - sum over terms t in both q and d
  - 0 if none of the query terms is present in the document.

Logarithm Table

Table of base 10, base 2 and base e (ln) logarithms:

x	$\log_{10} x$	$\log_2 x$	$\log_e x$
1	0.000000	0.000000	0.000000
2	0.301030	1.000000	0.693147
3	0.477121	1.584963	1.098612
4	0.602060	2.000000	1.386294
5	0.698970	2.321928	1.609438
6	0.778151	2.584963	1.791759
7	0.845098	2.807355	1.945910
8	0.903090	3.000000	2.079442
9	0.954243	3.169925	2.197225
10	1.000000	3.321928	2.302585
20	1.301030	4.321928	2.995732
30	1.477121	4.906891	3.401197
40	1.602060	5.321928	3.688879
50	1.698970	5.643856	3.912023
60	1.778151	5.906991	4.094345
70	1.845098	6.129283	4.248495
80	1.903090	6.321928	4.382027
90	1.954243	6.491853	4.499810
100	2.000000	6.643856	4.605170
200	2.301030	7.643856	5.298317
300	2.477121	8.228819	5.703782
400	2.602060	8.643856	5.991465
500	2.698970	8.965784	6.214608
600	2.778151	9.228819	6.396930
700	2.845098	9.451211	6.551080
800	2.903090	9.643856	6.684612
900	2.954243	9.813781	6.802395
1000	3.000000	9.965784	6.907755

Ref: https://www.quora.com/In-log-table-log10-0-but-in-Quora-some-say-log10-1-which-is-correct-and-what-is-the-reason

Observation: Rare terms are more informative

- E.g., stop words have no information
- Thus, we want a high weight for rare terms

Collection vs document frequency

- Collection frequency of *t* is the number of occurrences of *t* in the collection
- Document frequency of *t* is the number of documents in which *t* occurs

Word	Collection Freq	Document Freq
software	20000	5000
work	19980	15000

- The word software should get higher weight

Inversion document frequency (idf)

- df_t is the document frequency of *t*:
  - The number of documents that contain t
- $idf_t = \log_{10}(N/df_t)$
- idf example, suppose that N = 1,000,000

Term	df_t	idf_t
qwerty	1	6
algorithm	100	4
application	1,000	3
internet	10,000	2
bug	100,000	1
the	1,000,000	0

Effect of idf on ranking

- idf affects the ranking of documents for queries with at least two terms
- For the query like **iOS application**, idf weighting makes occurrences of **iOS** count for much more in the final document ranking than occurrences of **application**.

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = log_{10}(1 + tf_{t,d}) \times log_{10}(\frac{N}{df_t})$$

- Considered the best known weighting scheme in IR.
- Scores increase with
  - the number of occurrences within a document;
  - the rarity of the term in the collection.

tf-idf weighting

```
01  N = 5
02  cleaned_description = m1.get_and_clean_data()
03  cleaned_description = cleaned_description.iloc[:N]
04  vectorizer = CountVectorizer(preprocessor=preProcess)
05  X = vectorizer.fit_transform(cleaned_description)
06  print(X.toarray())
07
08  X.data = np.log10(X.data+1)
09  X = X.multiply(np.log10(N / X.sum(0))[0])
10
11  print(X.toarray())
12
13  print(pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names()))
```

$$w_{t,d} = log_{10}(1 + tf_{t,d}) \times log_{10}(\frac{N}{df_t})$$

Score for a document in a given query

$$Score(q,d) = \sum_{t \in q \cap d} (tf \cdot idf)_{t,d}$$

- Binary —> count —> weight matrix

N=5

	abil	accommod	accommodationaerotek	...	year	yield	zaur
0	0.000000	0.486826	0.367365	...	0.334668	0.367365	0.000000
1	0.276746	0.000000	0.000000	...	0.129467	0.000000	0.276746
2	0.000000	0.000000	0.000000	...	0.205200	0.000000	0.000000
3	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
4	0.276746	0.000000	0.000000	...	0.129467	0.000000	0.276746

Documents as vectors

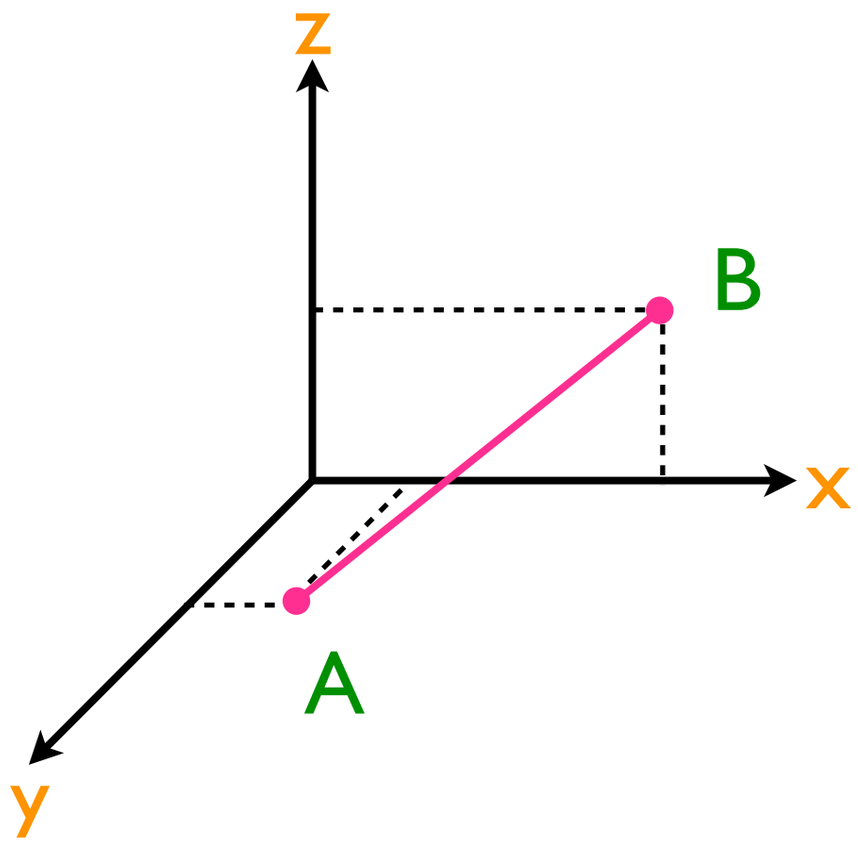
- The term *vector* has been referred to several time before this slide
  - |V|-dimensional vector space
  - Terms are axes of the space
- These term *vectors* are very high dimensional
  - Maybe higher than billion dimensions for a real-world search engine.
  - But their dimension is very sparse (most are zeroes)

Queries as vectors

- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
  - proximity = similarity of vectors = inverse of distance

Formalizing vector space proximity

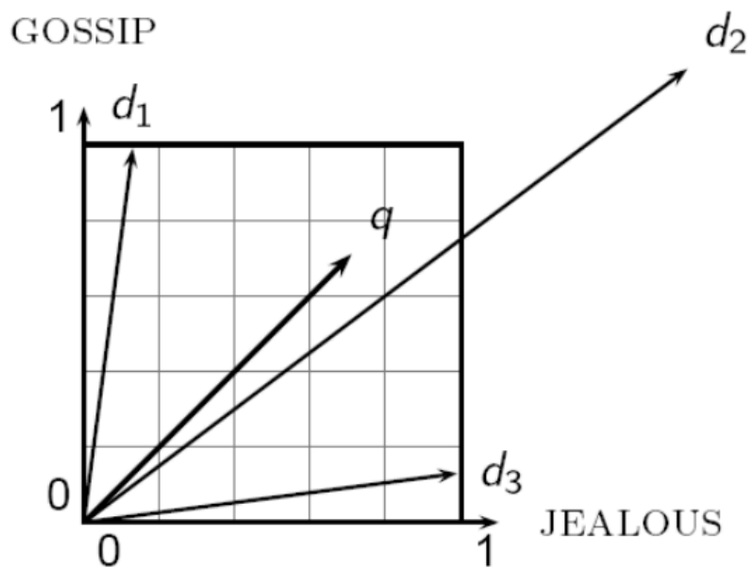
- Discuss about distance function
- Most common distance function is the **euclidean distance**



$$D(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

Why not the euclidean

- Euclidean distance is large for vectors of different lengths
  - In other words, not good when dimension is high.
  - E.g., the distance between q and d_2 is large
    - even though the distribution of the terms in q and the distribution of the terms in d_2 are very similar.



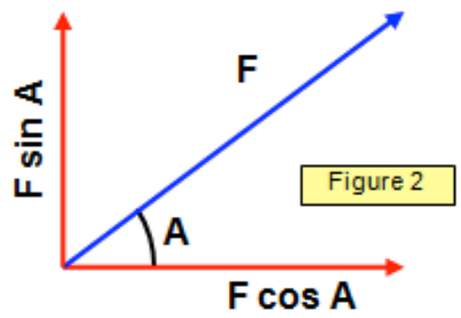
Then, what shall be used?

- Angle is more meaningful than distance.
  - E.g., take a document d and append it to itself. Call this document d′.
    - d and d′ have the same content
      - But the Euclidean distance between d and d′ is quite large

- The angle between the two documents is 0
  - = maximal similarity

Vectors and angle

- Cosines
  - Rank documents in **decreasing** order of the angle between query and document
  - Rank documents in **increasing** order of cosine$_{(q,d)}$

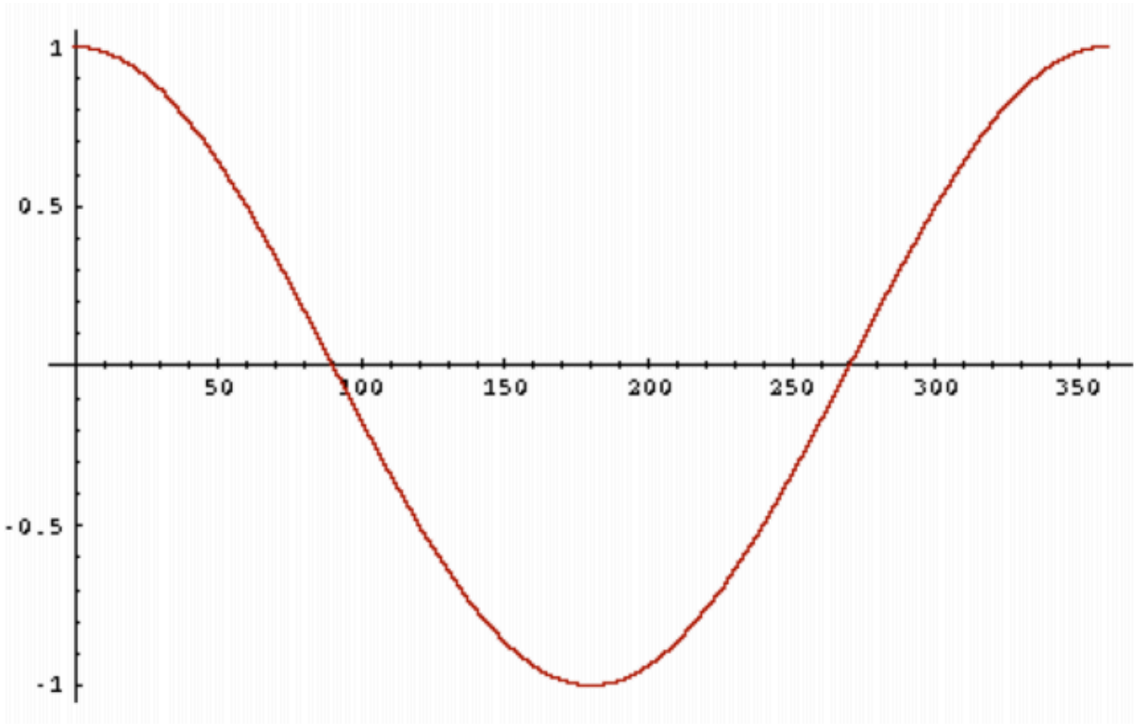


- Dot product

Ref: https://www.schoolphysics.co.uk/age16-19/Mechanics/Statics/text/Components_of_vectors/index.html

Vectors and angle

- From angles to cosines

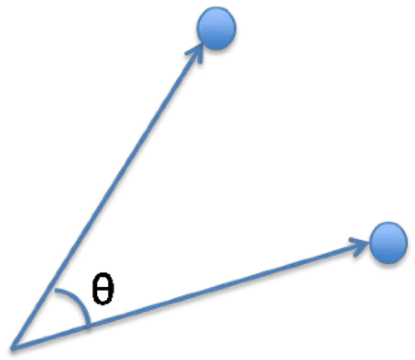


Ref: https://halalhassan.wordpress.com/2013/08/31/cosine-similarity-in-mapreduce-hadoop/

Vectors and angle

- From angles to cosines

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



- Length normalization
  - A vector can be length-normalized by dividing each of its components by its length

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Ref: https://halalhassan.wordpress.com/2013/08/31/cosine-similarity-in-mapreduce-hadoop/

Vectors and angle

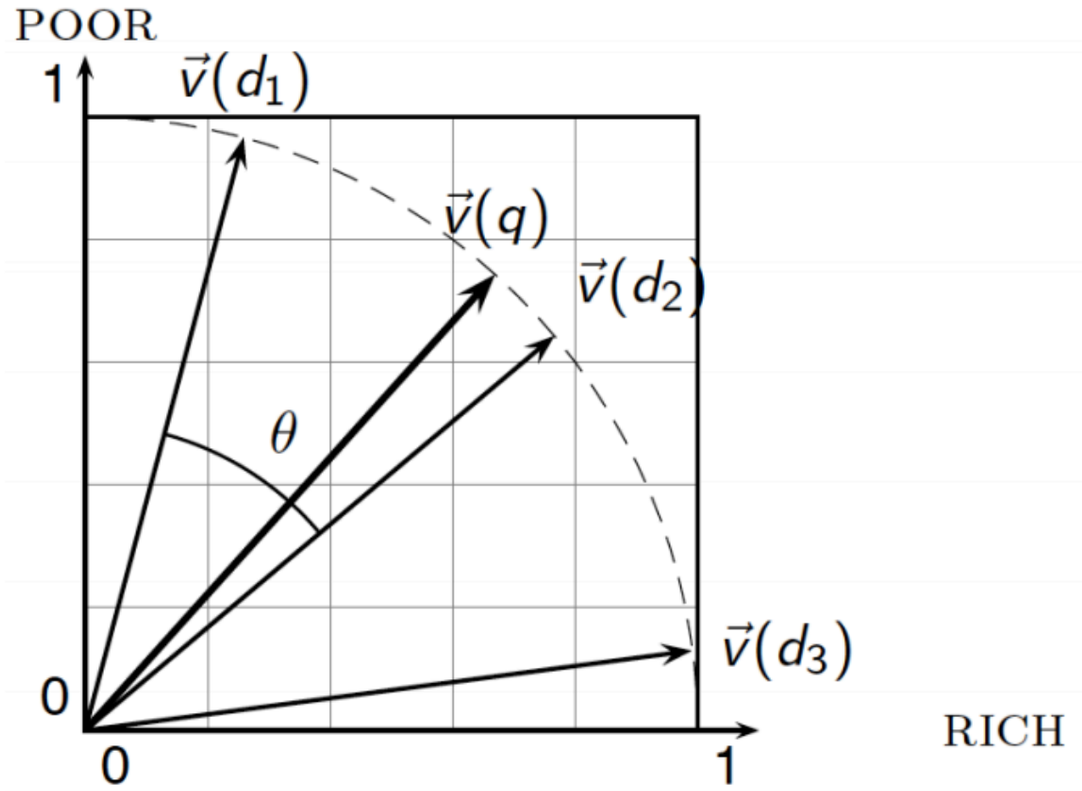
- cosine$_{(q,d)}$

$$cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$cos(\vec{q},\vec{d}) = \vec{q}\cdot\vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

Cosine similarity illustrated



More examples

- How similar are these three books
  - DH: DevOps Handbook
  - CD: Continuous Delivery
  - DC: Distributed Computing

Term	DH	CD	DC
business	100	90	5
computer	200	200	200
git	200	300	10
parallel	50	0	200

tf (counts)

Let’s simplify by ignoring idf in this example

$1 + \log_{10}(tf)$

Term	DH	CD	DC
business	2.004	1.959	0.778
computer	2.303	2.303	2.303
git	2.303	2.478	1.041
parallel	1.707	0.000	2.303

dot(DH,CD) ≈ 14.94

dot(CD,DC) ≈ 9.410

dot(DH,DC) ≈ 13.19

Length normalization

Term	DH	CD	DC
business	0.478	0.501	0.221
computer	0.549	0.589	0.656
git	0.549	0.633	0.297
parallel	0.407	0.000	0.656

cos(DH,CD) ≈ 0.912

cos(CD,DC) ≈ 0.686

cos(DH,DC) ≈ 0.898

Scikit-learn builtin tf-idf

```
1   N = 5
2   cleaned_description = m1.get_and_clean_data()
3   cleaned_description = cleaned_description.iloc[:N]
4   vectorizer = TfidfVectorizer(preprocessor=preProcess)
5   X = vectorizer.fit_transform(cleaned_description)
6   print(pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names()))
```

Tf Idf

	abil	accommod	accommodationaerotek	...	year	yield	zaur
0	0.00000	0.093182	0.046591	...	0.131243	0.046591	0.00000
1	0.07596	0.000000	0.000000	...	0.053042	0.000000	0.07596
2	0.00000	0.000000	0.000000	...	0.092824	0.000000	0.00000
3	0.00000	0.000000	0.000000	...	0.000000	0.000000	0.00000
4	0.07596	0.000000	0.000000	...	0.053042	0.000000	0.07596

Summary – vector space ranking

- Represent the **query** as a weighted tf-idf vector
- Represent each **document** as a weighted tf-idf vector
- Compute the **cosine similarity score** for the query vector and each document vector
- **Rank** documents with respect to the query by score
- Return the **top K** (e.g., K = 10) to the user

Further methods

- Probability ranking
  - E.g., probability of retrieving a relevant document from the set of documents indexed by **social**.
  - Does not need term weighting; however, within document statistics (i.e., tf) do not play a role.
- Language model
  - Linear combination of document model and many other models, i.e., Document priors, Relevance models, Translation models, and Aspect models.
- Google PageRank
  - Authoritative models based

Probability ranking

- Probability ranking
  - E.g., probability of retrieving a relevant document from the set of documents indexed by **social**.
  - Question: In what order do we present documents to the user?
    - Have to determine what is the “best” document, the “second best”, …
  - Idea: Rank by probability of relevance of the document with regards to **information need**

Probability ranking — principle

- The IR system responses to each user's request with a ranking of the documents in the collection in order of **decreasing probability of relevance to the user who submitted the request.**

Recall a few probability basics

- For events A and B:

$$p(A, B) = p(A \cap B) = p(A|B)p(B) = p(B|A)p(A)$$

- Bayes's rule:

prior

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} = \frac{p(B|A)p(A)}{\sum_{X=A,\overline{A}} p(B|X)p(X)}$$

posterior

- Odds:

$$O(A) = \frac{p(A)}{p(\overline{A})} = \frac{p(A)}{1 - p(\overline{A})}$$

Probability ranking — principle

- Let **x** represent a document in the collection
- Let **R** the level of **relevance** of a document with regard to the given query, **where R=1 means relevant and R=0 mean not relevant**

- What to find $p(R=1|x)$ —

$$p(R=1|x) = \frac{p(x|R=1)p(R=1)}{p(x)}$$

$$p(R=0|x) = \frac{p(x|R=0)p(R=0)}{p(x)}$$

$$p(R=0|x) + p(R=1|x) = 1$$

- p(R=1) and p(R=0) — prior probability of retrieving a relevant or non-relevant document at random
- p(x|R=1) and p(x|R=0) — probability that if a relevant (not relevant) document is retrieved, it is x.

Probabilistic retrieval strategy

- Estimate how each term contributes to relevance
- Combine to find document relevance probability
- Order documents by decreasing probability
- Minimize the loss, i.e., error

Traditional — Binary independence model

- **Binary = Boolean**: documents are represented as binary incidence vectors of terms … the simple representation
- $\vec{x} = (x_1, ..., x_n)$
- $x_i = 1$ **if and only if** term *i* is present in document *x*
- Queries: binary term incidence vectors

Traditional — Binary independence model

- Given query **q**,
  - Fore each document **d**, compute p(R|q,**x**) where **x** is binary term incidence vector representing **d**
  - Applying the combined odds and Bayes' rule:

$$O(R|q,\vec{x}) = \frac{p(R=1|q,\vec{x})}{p(R=0|q,\vec{x})} = \frac{\frac{p(R=1|q)p(\vec{x}|R=1,q)}{p(\vec{x}|q)}}{\frac{p(R=0|q)p(\vec{x}|R=0,q)}{p(\vec{x}|q)}}$$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = \frac{p(R=1|q,\vec{x})}{p(R=0|q,\vec{x})} = \frac{p(\vec{x}|R=1,q)}{p(\vec{x}|R=0,q)}$$

Constant for a given query

Needs estimation

- Assume that all terms are **independent**

$$\frac{p(\vec{x}|R=1,q)}{p(\vec{x}|R=0,q)} = \prod_{i=1}^{n} \frac{p(x_i|R=1,q)}{p(x_i|R=0,q)}$$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{i=1}^{n} \frac{p(x_i|R=1,q)}{p(x_i|R=0,q)}$$

Traditional — Binary independence model

- It can be further simplified since x_i can be only 0 or 1.

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{x_i=1} \frac{p(x_i=1|R=1,q)}{p(x_i=1|R=0,q)} \cdot \prod_{x_i=0} \frac{p(x_i=0|R=1,q)}{p(x_i=0|R=0,q)}$$

- Let $p_i = p(x_i=1|R=1,q); r_i = p(x_i=1|R=0,q);$

- Assume, for all terms not occurring in the query ($q_i = 0$) $p_i = r_i$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

	Document	Relevant (R=1)	Not relevant (R=0)
Term present	$x_i = 1$	p_i	r_i
Term absent	$x_i = 0$	$(1 - p_i)$	$(1 - r_i)$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

All matching terms

Non-matching query terms

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \left(\frac{1-r_i}{1-p_i} \cdot \frac{1-p_i}{1-r_i}\right) \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=q_i=1}} \frac{p_i(1-r_i)}{r_i(1-p_i)} \cdot \prod_{\substack{q_i=1}} \frac{1-p_i}{1-r_i}$$

All matching terms

All query terms

Traditional — Binary independence model

Constant for each query

$$O(R|q,\vec{x}) = O(R|q) \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \prod_{q_i=1} \frac{1-p_i}{1-r_i}$$

Only quantity to be estimated for rankings

- Residual status value

$$RSV = log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i=q_i=1} log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Traditional — Binary independence model

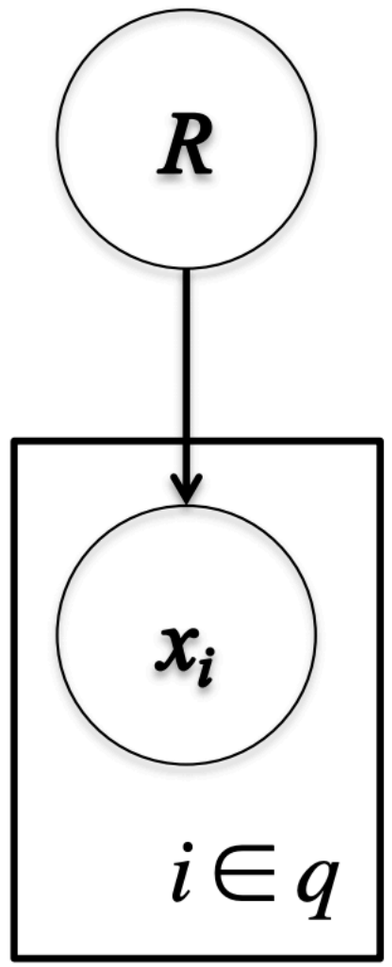
- Stripping all the constants, all we need to do is to compute the Retrieval Status Value (RSV)

$$RSV = log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i=q_i=1} log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$RSV = \sum_{x_i=q_i=1} c_i; c_i = log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

- The c_i are log odds ratios
- c_i functions as the term weights in this model

Graphical model for BIM



Binary variables

$$x_i = (tf_i \neq 0)$$

Binary independence model

- Estimating RSV coefficients in theory

$$c_i = log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

- For each term i look at the below table of document counts

Documents	Relevant	Non-relevant	Total
$x_i = 1$	s	n - s	n
$x_i = 0$	S - s	N - n - S + s	N - n
Total	S	N - S	N

- Estimates: $p_i \approx \frac{s}{S}$ $r_i \approx \frac{n-s}{N-S}$

$$c_i \approx K(N,n,S,s) = log \frac{s/(S-s)}{(n-s)/(N-n-S+s)}$$

Estimation

- If non-relevant documents are approximated by the whole collection, then r_i (the probability of occurrence in **non-relevant documents** for query) is n/N and

$$log\frac{1-r_i}{r_i} = log\frac{N-n-S+s}{n-s} \approx log\frac{N-n}{n} \approx log\frac{N}{n} = IDF$$

Estimation — on the other hand

- p_i (the probability of occurrence in **relevant documents** for query) cannot be approximated easily
- Various approaches to estimate p_i
  - Applying relevance weighting
    - constant, e.g., $p_i = 0.5$
    - proportional to the probability of occurrence in collection

Okapi BM25

- BM stands for Best Match (25 seems to come from a brute force)
- Today's conventional standard if saying about a probabilistic model

- **Main idea:** be sensitive to term frequency and document length while not adding too many parameters

Generative model for documents

- Words are drawn independently from the vocabulary using a multinomial distribution

Ref: https://dataaspirant.com/bag-of-words-bow/

Okapi BM25

- Distribution of term frequencies (tf) follows a binomial distribution – approximated by a Poisson
  - Binomial distribution $P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$
  - Poisson distribution $P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$
- Recommended ref — http://makemeanalyst.com/normal-distribution-binomial-distribution-poisson-distribution/

Binomial

- n — repeated experiments
- x — success trial form n trials
- p — probability of success

$$P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$$

- Example — calculate the probability that a student randomly answers 3 exam questions without reading the questions. All of which is a 4-multiple choice.
  - All correct
  - 2 correct
  - 1 correct
  - All fail

Binomial

- All correct — p(TTT) = 0.25^3 = 0.02
- Two correct — p(TTF) + p(TFT) + p(FTT) = (0.25 * 0.25 * 0.75) * 3 = 0.14
- One correct — p(FFT) + p(FTF) + p(TFF) = (0.75 * 0.75 * 0.25) * 3 = 0.42
- All fail — p(FFF) = 0.75^3 = 0.42

$$P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$$

Poisson model

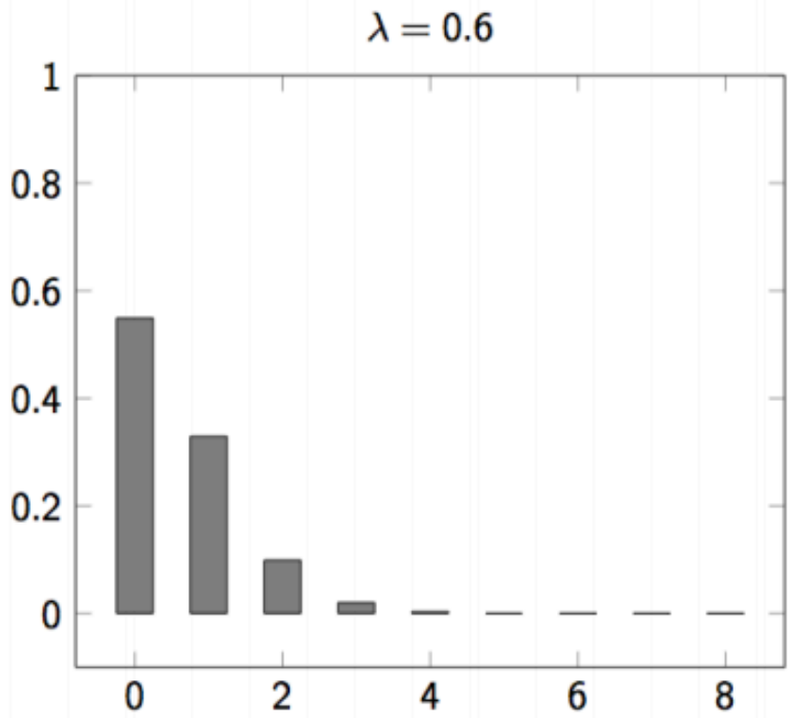
- Poisson can used for estimated a binomial distribution
- The Poisson distribution models the probability of x, the number of events occurring in a fixed interval of time/space, with known average rate $\boldsymbol{\lambda}$ ($\boldsymbol{\lambda}$ = cf/T), independent of the last event

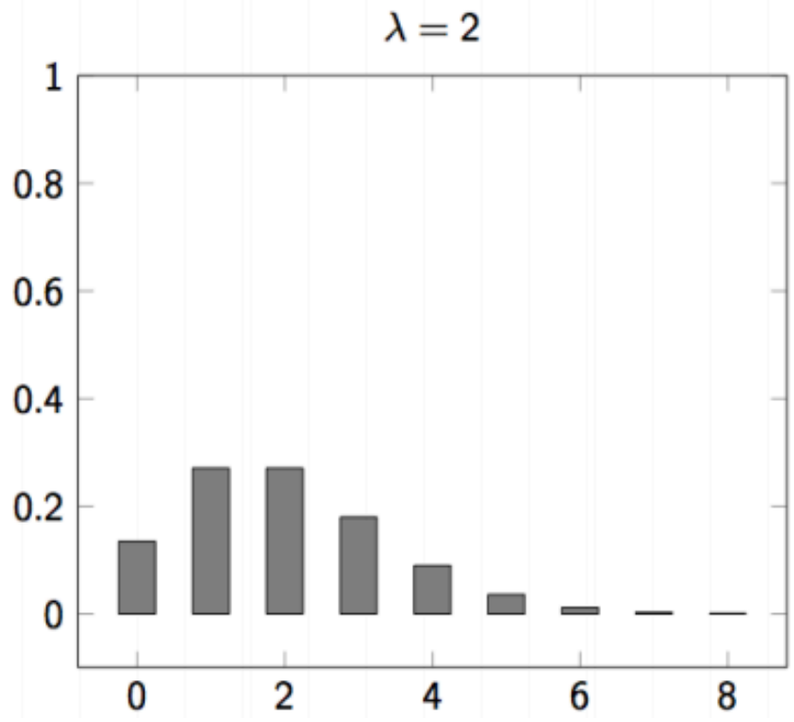
$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

- Interesting applications, e.g., estimate the #customers by counting #customer in 1 minute

Poisson model

- Assume that term frequencies in a document (tf_i) follow a Poisson distribution
  - **Fixed interval** implies fixed document length





Poisson model — Application example

		Documents containing *k* occurrences of word (λ = 53/650)												
Freq	**Word**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
53	expected	599	49	2										
52	*based*	600	48	2										
53	*conditions*	604	39	7										
55	*cathexis*	619	22	3	2	1	2	0	1					
51	*comic*	642	3	0	1	0	0	0	0	0	0	1	1	2

Eliteness

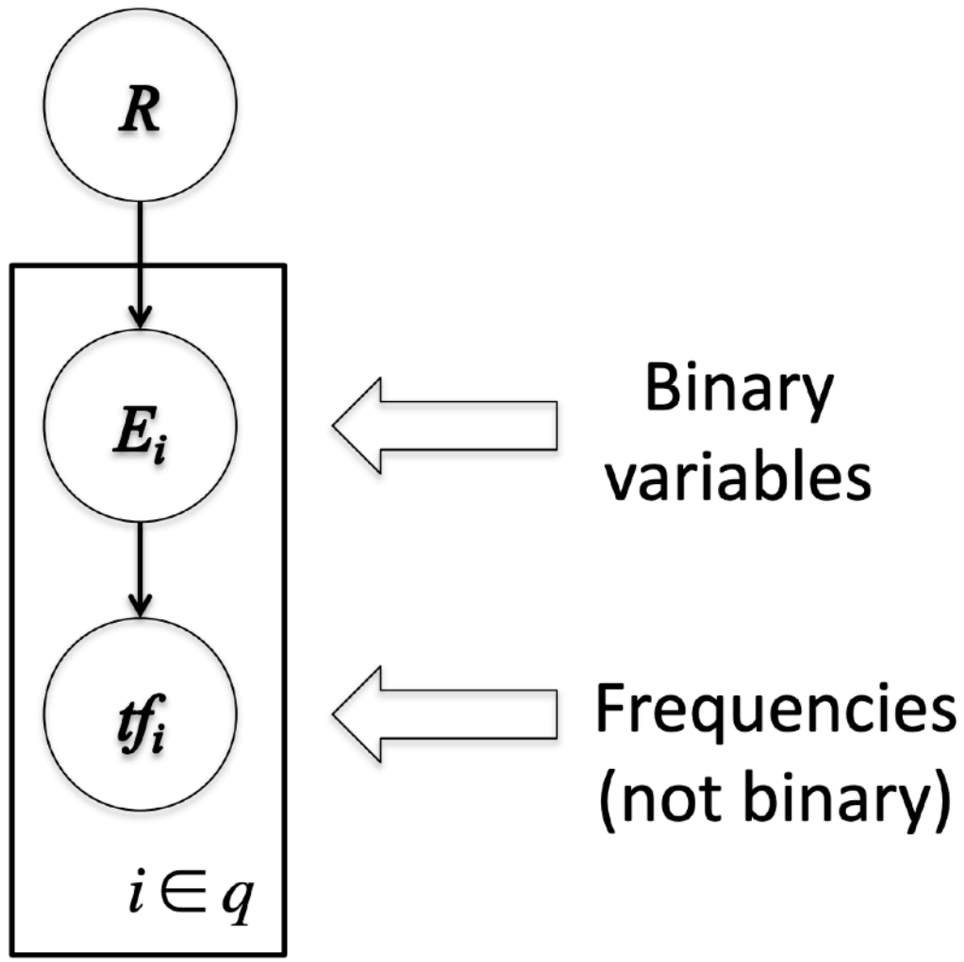
- Term frequencies ~ eliteness
- eliteness?
  - Hidden variable for each document-term pair, denoted as E_i for term *i*
  - Represents **aboutness**, i.e., a term is elite in a document if, in some sense, the document is about the concept denoted by the term

Elite terms

In software engineering, a **design pattern** is a general repeatable **solution** to a commonly occurring **problem** in **software design**. A **design pattern** isn't a finished **design** that can be transformed directly into **code**. It is a **description** or **template** for how to **solve** a **problem** that can be used in many different situations.

Ref: https://sourcemaking.com/design_patterns

Graphical model with eliteness



Using RSV

$$RSV^{elite} = \sum_{i \in q, tf_i > 0} c_i^{elite}(tf_i)$$

- where $c_i^{elite}(tf_i) = log \frac{p(TF_i = tf_i|R = 1)p(TF_i = 0|R = 0)}{p(TF_i = 0|R = 1)p(TF_i = tf_i|R = 0)}$

- Then, using eliteness we will have:

$$p(TF_i = tf_i|R) = p(TF_i = tf_i|E_i = elite)p(E_i = elite|R)$$
$$+p(TF_i = tf_i|E_i = \overline{elite})(1 - p(E_i = elite|R)$$

2-Poisson model

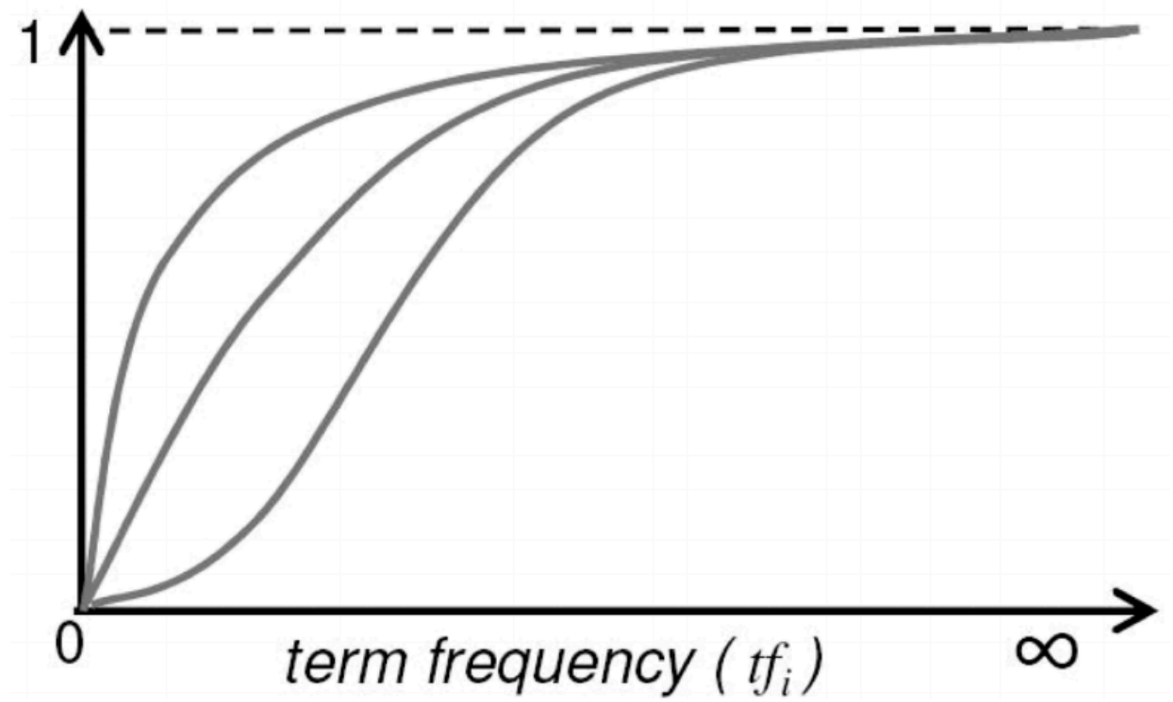
- The distribution of the elite terms and non-elite terms are not the same.
- The distribution of the elite terms and non-elite terms are not the same.

$$p(TF_i = k_i|R) = \pi \frac{\lambda^k}{k!} e^{-\lambda} + (1-\pi) \frac{\mu^k}{k!} e^{-\mu}$$

- π is the probability that the document is elite for term
- We don't know π, λ, μ

Observing the behavior

$$c_i^{elite}(tf_i)$$

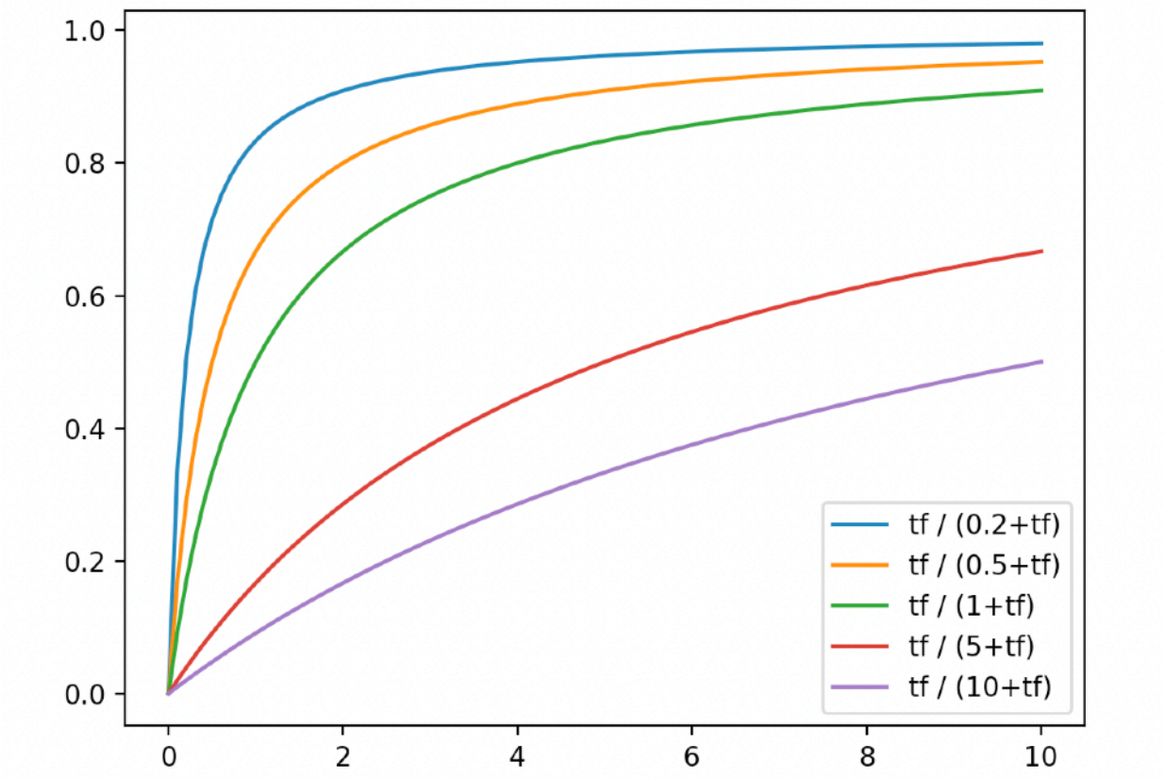


2-Poisson model

- Since it is hard to precisely simulate, it may be fine to just approximate it using equation with the same qualitative properties:

$$\frac{tf}{k_1 + tf}$$

Saturation function



- For high value of k_1, increment in tf_i continue to contribute to significantly to the score
- Contribution tail of quickly for low values of k_1

Creating BM25

- The first version: just simply using the saturation function

$$c_i^{BM25v1}(tf_i) = c_i \frac{tf_i}{k_1 + tf_i}$$

- The second version: simplification to IDF

$$c_i^{BM25v2}(tf_i) = log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1 + tf_i}$$

  - (k_1+1) does not change raking, but to make the term score = 1 when tf_i = 1.
  - One significant merit is that the term score are bounded.

Document length normalization

- Typically, longer documents are likely to have larger tf_i score
- Document length:

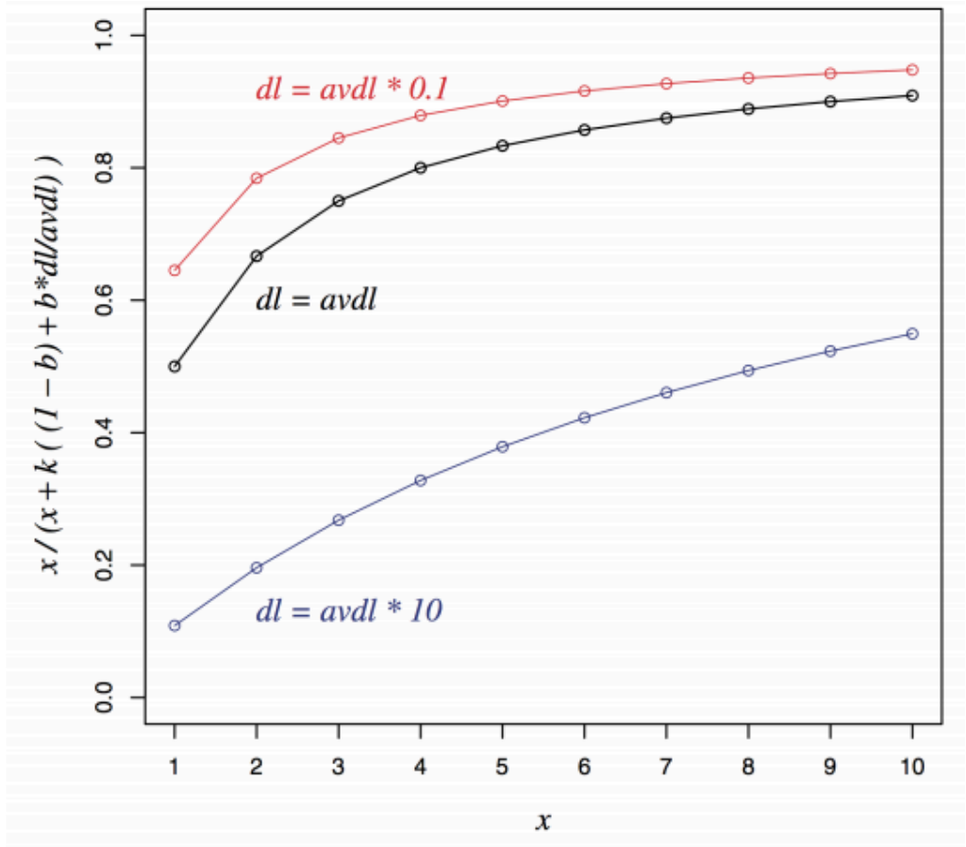
$$dl = \sum_{i \in V} tf_i$$

- *avdl*: Average document length over collection
- Length normalization component:

$$B = \left((1-b) + b \frac{dl}{avdl} \right), 0 \leq b \leq 1$$

  - b = 1: full document length normalization
  - B = 0: no document length normalization

Document length normalization



Okapi BM25

- Normalize *tf* using document length

$$tf_i' = \frac{tf_i}{B}$$

$$c_i^{BM25}(tf_i) = log\frac{N}{df_i} \times \frac{(k_1+1)tf_i'}{k_1 + tf_i'}$$

$$= log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1\left((1-b) + b\frac{dl}{avdl}\right) + tf_i}$$

- BM25 ranking function

$$RSV^{BM25} = \sum_{i \in q} c_i^{BM25}(tf_i) \quad = \sum_{i \in q} log\frac{N}{df_i} \cdot \frac{(k_1+1)tf_i}{k_1\left((1-b) + b\frac{dl}{avdl}\right) + tf_i}$$

Okapi BM25

$$RSV^{BM25} = \sum_{i \in q} log \frac{N}{df_i} \cdot \frac{(k_1 + 1) tf_i}{k_1 \left((1 - b) + b \frac{dl}{avdl} \right) + tf_i}$$

- k_1 controls term frequency scaling
  - k_1 = 0 is binary model; k_1 large is raw term frequency
- b controls document length normalization
  - b = 0 is no length normalization; b=1 is relative frequency
- Typically, k_1 is set around 1.2 - 2 and b is around 0.75 (from brute force?)

BM25 and Scikit-learn

- https://gist.github.com/koreyou/f3a8a0470d32aa56b32f198f49a9f2b8

```
01  class BM25(object):
02      def __init__(self, b=0.75, k1=1.6):
03          self.vectorizer = TfidfVectorizer(preprocessor=preProcess, norm=None, smooth_idf=False)
04          self.b = b
05          self.k1 = k1
06
07      def fit(self, X):
08          """ Fit IDF to documents X """
09          self.vectorizer.fit(X)
10          y = super(TfidfVectorizer, self.vectorizer).transform(X)
11          self.avdl = y.sum(1).mean()
12
13      def transform(self, q, X):
14          """ Calculate BM25 between query q and documents X """
15          b, k1, avdl = self.b, self.k1, self.avdl
16
17          # apply CountVectorizer
18          X = super(TfidfVectorizer, self.vectorizer).transform(X)
19          len_X = X.sum(1).A1
20          q, = super(TfidfVectorizer, self.vectorizer).transform([q])
21          assert sparse.isspmatrix_csr(q)
22
23          # convert to csc for better column slicing
24          X = X.tocsc()[:, q.indices]
25          denom = X + (k1 * (1 - b + b * len_X / avdl))[:, None]
26          # idf(t) = log [ n / df(t) ] + 1 in sklearn, so it need to be coneverted
27          # to idf(t) = log [ n / df(t) ] with minus 1
28          idf = self.vectorizer._tfidf.idf_[None, q.indices] - 1.
29          numer = X.multiply(np.broadcast_to(idf, X.shape)) * (k1 + 1)
30          return (numer / denom).sum(1).A1
```

BM25 and Scikit-learn

```
1   N = 5
2   cleaned_description = m1.get_and_clean_data()
3   cleaned_description = cleaned_description.iloc[:N]
4   bm25 = BM25()
5   bm25.fit(cleaned_description)
6   print(bm25.transform('aws github',cleaned_description))
```

```
[0.         0.         1.85457611 2.10752076 0.        ]
```

- Both contain the words aws and github, but the forth document seems to have a higher level of eliteness.

Summary

- Some histories
- Scoring documents
- Term frequency (tf)
- Vector space model

Time for questions