Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search

Pandu Nayak and Prabhakar Raghavan

Lecture 6: Scoring, Term Weighting and the Vector Space Model

Recap of lecture 5

- Collection and vocabulary statistics: Heaps' and Zipf's laws
- Dictionary compression for Boolean indexes
  - Dictionary string, blocks, front coding
- Postings compression: Gap encoding, prefix-unique codes
  - Variable-Byte and Gamma codes

	MB
collection (text, xml markup etc)	3,600.0
collection (text)	960.0
Term-doc incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ–encoded	101.0

This lecture; IIR Sections 6.2-6.4.3

- Ranked retrieval
- Scoring documents
- Term frequency
- Collection statistics
- Weighting schemes
- Vector space scoring

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't. Only return True or False
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user dlink 650*" → 200,000 hits

  Boolean search 對於每個查詢都只會回傳「符合」或「不符合」，結果不是太多就是太少 -> 必須找一個更好的查詢法
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Ranked retrieval models

對所有結果評分
而非只回傳ture or false

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Free text queriers: A free text query is simply one or more words, terms, numbers, and optionally operators.
其實就是自然語言

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top k (≈ 10) results
  - We don't overwhelm the user

  - Premise: the ranking algorithm works

對使用者來說，搜尋出幾個結果並不是重點，使用者通常只有耐心去看前10個結果
所以需要對所有結果排序，使用Ranking algorithm

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query “match”.

評分越高的document，代表越接近使用者想要查詢的結果
分數會落在0, 1之間

Query-document matching scores

- We need a way of assigning a score to a query/ document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- Recall from Lecture 3: A commonly used measure of overlap of two sets *A* and *B*
- jaccard$(A,B) = |A \cap B| \ / \ |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1.

通常A = Query, B = Document
分數越高，代表該document出現越多query中的單字

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march* |Q| = 3, Q = {ides, of, march}
- Document 1: *caesar died in march* |D1| = 4, D1 = {caesar, died, in, march}
- Document 2: *the long march* |D2| = 3, D2 = {the, long, march}

jaccard(Q, D1) = (Q ∩ D1) / (Q ∪ D1) = Q和D1重複字數 / Q和D1總共字數 = 1/6
jaccard(Q, D2) = (Q ∩ D2) / (Q ∪ D2) = Q和D2重複字數 / Q和D2總共字數 = 1/5
（重複的字只能算一次！）

缺點：
分數高的document可能只是因為他比較短(cosine similarity可以解決)
沒考慮到Query在Document出現的頻率(tf-idf可以解決)

Issues with Jaccard for scoring

只有用出現與否的True / False

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information 罕見字比常用字更有價值，jaccard卻沒考慮到這點
- We need a more sophisticated way of normalizing for length
- Later in this lecture, we'll use $\boxed{|A \cap B|/\sqrt{|A \cup B|}}$
- . . . instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

Recall (Lecture 1): Binary term-document incidence matrix

Bi-word

Doc	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

只考慮每一組Bi–word有沒有出現在某個doc中

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\boxed{\mathbb{N}^{v}}$: a column below

改良原本的boolean matrices，把單純的true false改為每個bi–word出現在各document的次數

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.

一袋文字，只考慮有沒有出現過，沒考慮順序

- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

只考慮出現的次數，不考慮順序
所以事先建立positional index
再使用bag of words model
即可改善這個問題

Term frequency tf 某個term在某個document出現了幾次?

- The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
    出現1次query和出現10次query的documents，重要性並不剛好相差10倍
    而是會隨著出現頻率的增加，重要性的成長會漸漸趨緩 –> 取log
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

(w –> tf)

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

如果query有兩個terms
那就分別算出他們的log freq再相加

- The score is 0 if none of the query terms is present in the document.

Document frequency 一個term在幾個documents出現過

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

越罕見的單字，在搜尋時應該越有價值
當一個罕見字出現在某一文件時，該文件有很大的機率會是使用者要的，所以罕見字的df低
反之，常見的單字在任何文件中都看得到，就沒辦法作為判斷的標準，所以常見字的df高
所以取inverse–df(IDF)，就可以代表一個字的權重，越稀有的字權重越高

Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high*, *increase*, *line*)如果查詢的是常用字，即使出現在document，也不一定是使用者要的
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high*, *increase*, *and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this. 越常見的單字給予越高的分數–>最後再inverse取log

idf weight

Inverse document frequency

Term t 出現在「幾個」documents中，不會重複計算

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\mathrm{df}_t \le N$
- We define the idf (inverse document frequency) of t by

$$\boxed{\mathrm{idf}_t = \log_{10}\left(N/\mathrm{df}_t\right)}$$

如果df越小，代表這個字愈罕見
他能得到的idf分數就愈高

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

同樣，取log是因為出現頻率和相關程度並非線性關係
就像出現1000次並不代表分數應該比出現1次的高1000倍

Will turn out the base of the log is immaterial.

idf example, suppose N = 1 million

term	df_t	idf_t 可視為這個term的權重(價值)
calpurnia	1	log(1,000,000 / 1) = 6
animal	100	log(1,000,000 / 100) = 4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	$\boxed{0}$

零代表每個document都會出現這個字，完全無法作為判斷基準

$$\mathrm{idf}_t = \log_{10}\,(N/\mathrm{df}_t)$$

There is one idf value for each term t in a collection.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  IDF是拿來衡量Query中每一個字的權重
  所以如果Query只有一個字，那IDF就沒用
  - iPhone
  因為不同於tf，idf是指一個term出現在「幾個」document，不會重複計算
  但可以透過idf分數來決定每個term的稀有度，作為最後documents ranking依據
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

搜尋”capricious person”時
應該著重在較罕見的”capricious”而非”person”
把出現”capricious”的document排在前面

Collection vs. Document frequency

Term t 出現在collection的「次數」，會重複算！

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example: 此字在collection出現次數 此collection中多少documents出現此字

Word	Collection frequency	Document frequency
insurance	10440 次數差不多	較少文件中看到 3997 –> 每個文件可能有2–3個insurance
try	10422	較多文件中看到 8760 –> 每個文件可能有1–2個try

- Which word is a better search term (and should get a higher weight)? insurance屬於較罕見的字 因為df較低，所以idf較低

tf-idf weighting

某個term在某個document出現的次數 x 該term的權重
權重愈高，出現愈多，評分愈高

- The tf-idf weight of a term is the product of its tf weight and its idf weight. [1 + log(tf)] * [log(N/df)] ??

$$\boxed{w_{t,d} = \log(1 + \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)}$$

先log再+1
參考Example

- Best known weighting scheme in information retrieval
  - Note: the “-” in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Log(TF) x Log(IDF) = LOG(TF+IDF)
取LOG是為了平滑化
就像出現1000次不代表分數應該要是出現1次的一千倍

Score for a document given a query

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

若query中有不只一個terms，那就算出「每一個term的tf.idf」再相加

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted 越罕見的字出現越多次->分數月高
  - …

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

V is the number of words

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

把Queries和Documents視為向量去做比較

Queries as vectors

把Query和Document轉成向量
去比較他們的距離

- Key idea 1: Do the same for queries: represent them as vectors in the space
  Query和Document距離愈短
  –>他們之間的相關性愈高
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

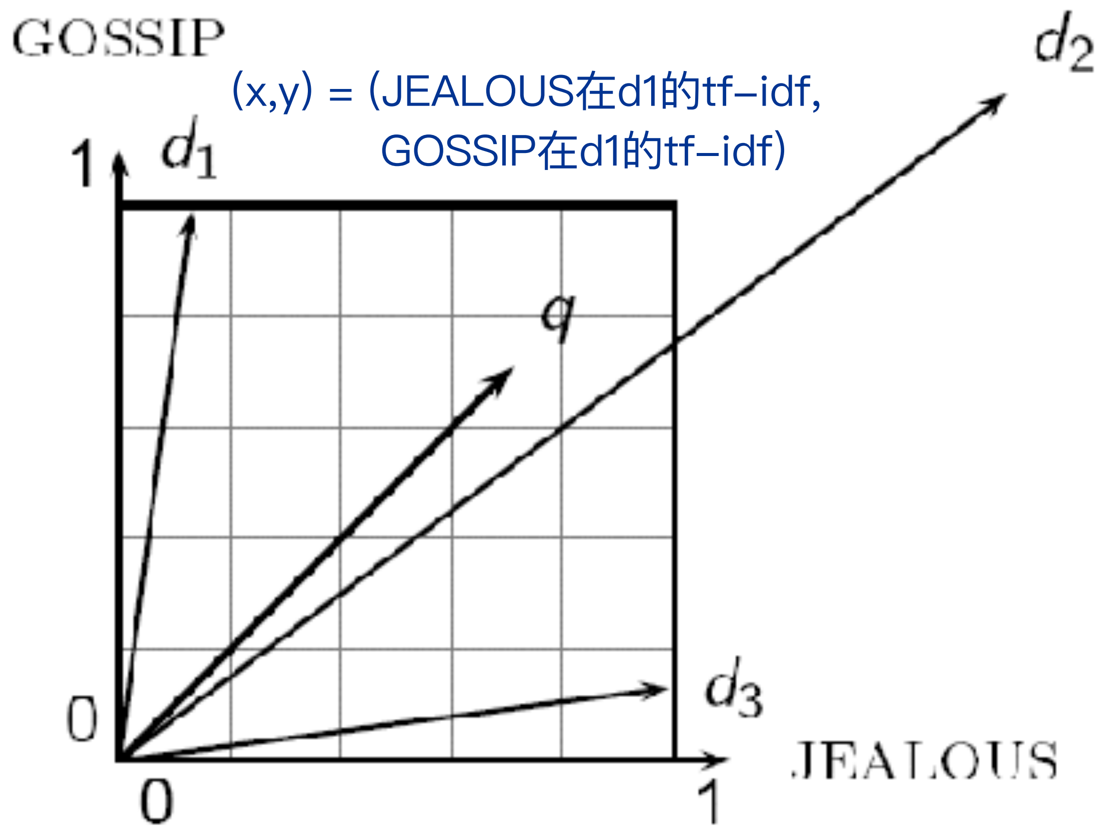
Formalizing vector space proximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance? $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $\text{distance}(p_1,p_2) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

可能因為documents比較長，導致距離變長，進而誤判queries和documents的相關性

Why distance is a bad idea

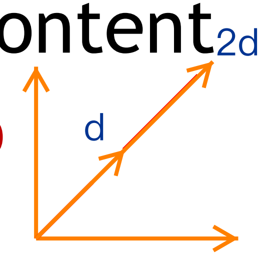
The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Q = 找出包含GOSSIP和JEALOUS的Document
從向量空間來看，很明顯是d_2最符合需求
但就因為他最長，導致distance(q, d_2)最大–>誤判為最不相關

Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
- “Semantically” d and d′ have the same content
- The Euclidean distance between the two documents can be quite large



- The angle between the two documents is 0, corresponding to maximal similarity.

  d和2d內文的單字完全一致，僅長度不同，但算距離卻會得到很大差異，誤判為兩個文章不相關

  –>改用角度，query和document夾角越大，代表他們越不相關

- Key idea: Rank documents according to angle with query.

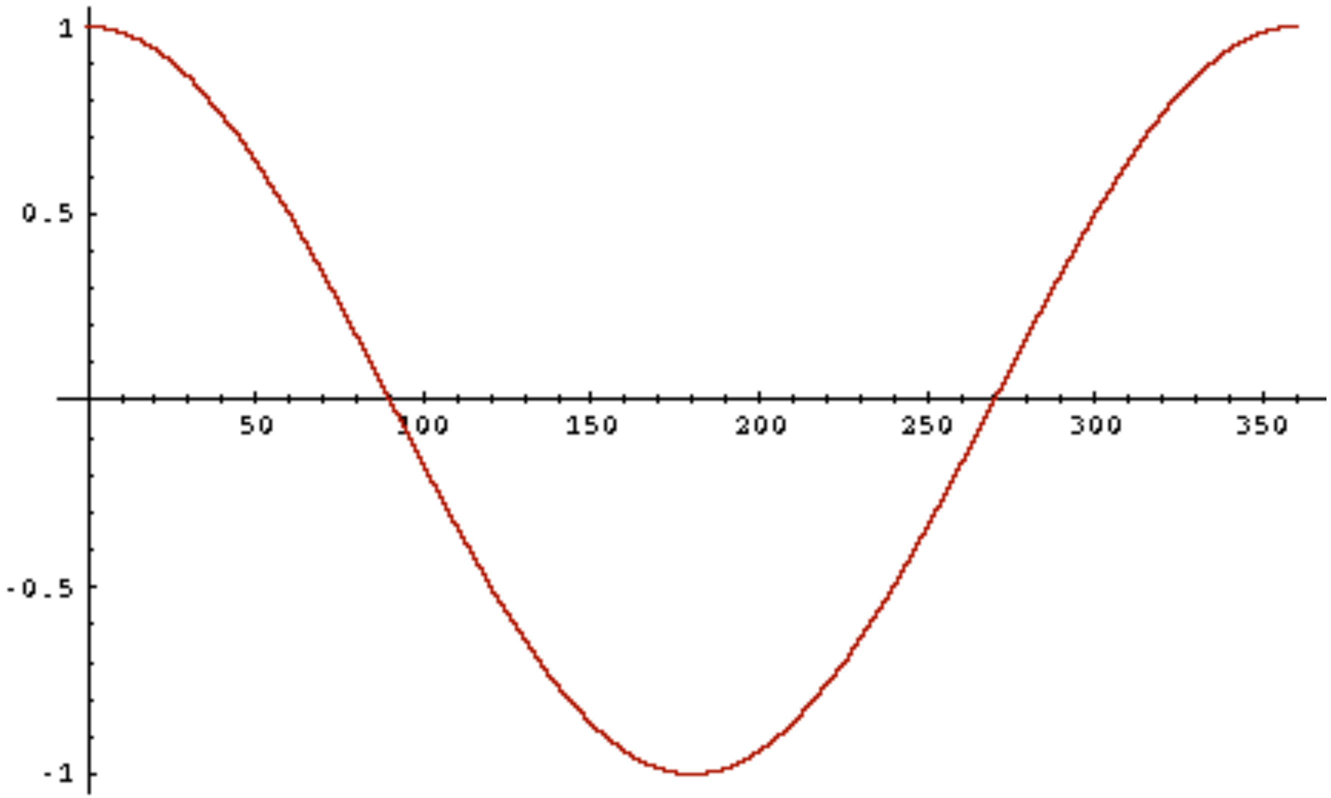
  用夾角來取代距離

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0°, 180°]

角度愈大，cosine愈小，愈不相關

From angles to cosines



- But how – *and why* – should we be computing cosines?

Length normalization

Cosine可以直接解決長度不同造成的誤判
因為Cosine的分母會對長度做標準化

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm: $\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$

將向量標準化(x ,y座標分別除以長度)後再算Ranking，即可排除長度造成的錯估

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

cosine(query,document)

a•b = |a| * |b| * cosθ
–> cosθ = (a•b) / (|a| * |b|)

Dot product

Unit vectors

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

算出query中每個term的tf–idf
以及這些term在document中的tf–idf

q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

這邊用tf–idf
所以還不是word2vec的cosine similarity

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine for length-normalized vectors

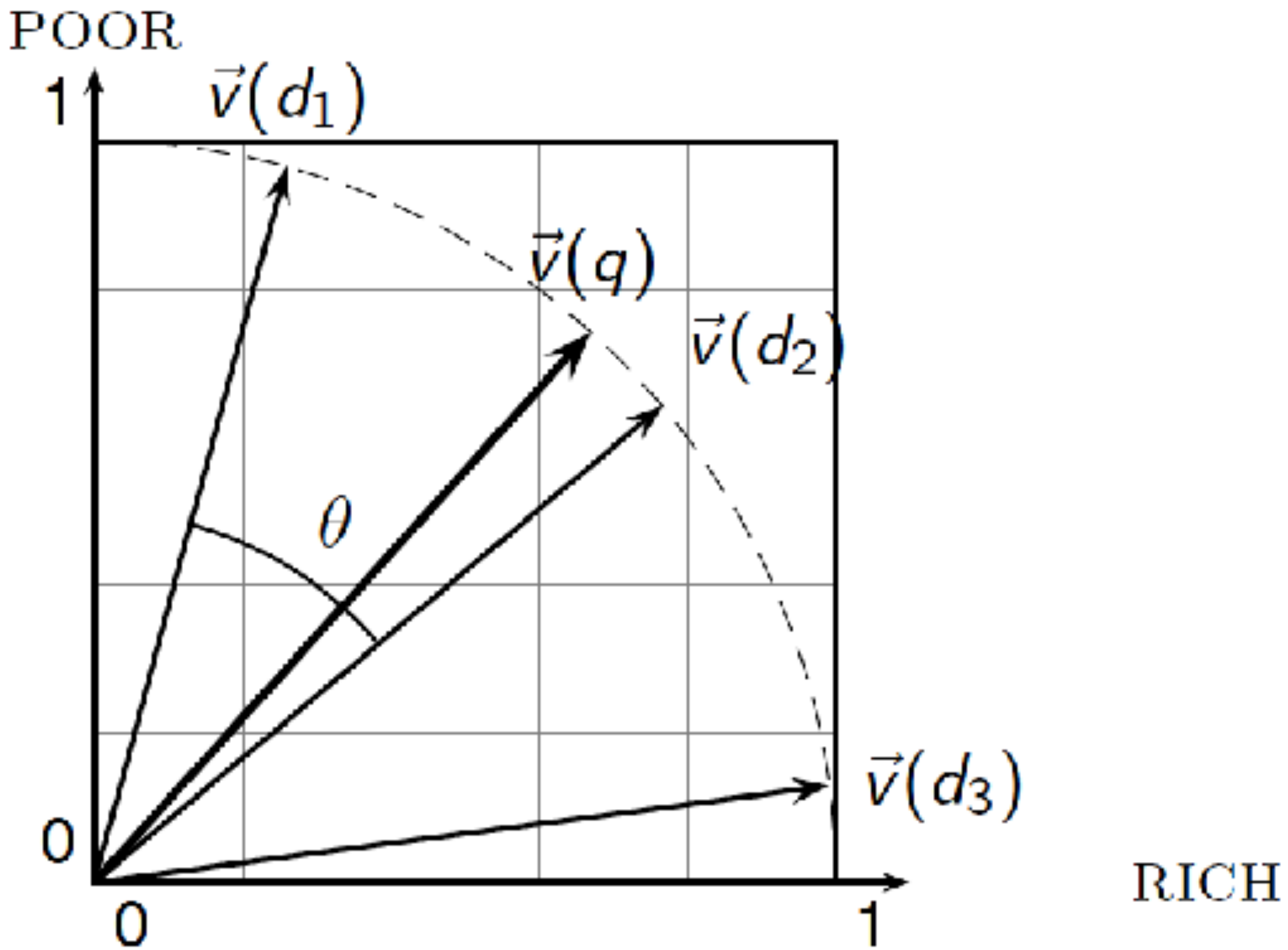
- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

已經把長度標準化後的公式，只要取內積即可得到cosine，不必再除以長度

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

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

這4個terms分別在這3個documents出現的次數

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

(P.17) 出現次數取log再+1

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

平方和： 15.0536

After length normalization

長度標準化

3.06/(√15.0536)

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

把term視為維度，把document視為空間向量
每一個document的長度都要標準化 = 1

$\boxed{0.789^2 + 0.515^2 + 0.335^2 + 0^2 = 1}$

標準化後的document，只要互相內積即可得到cosine
Cosine愈大代表他們的相關性愈高

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

Computing cosine scores

CosineScore(q)

float Scores$[N] = 0$

float Length$[N]$

for each query term t

do calculate $\boxed{w_{t,q}}$ and $\boxed{\text{fetch postings list}}$ for t

  for each pair$(d, \mathrm{tf}_{t,d})$ in postings list

  do $Scores[d] += w_{t,d} \times w_{t,q}$

Read the array *Length*

for each d

do $Scores[d] = Scores[d]/Length[d]$

return Top K components of $Scores[]$

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

最常一起用的組合，稱為SMART Notation, itc
Log(TF) with add–1 smoothing + Log(IDF)

Columns headed 'n' are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq*, using the acronyms from the previous table
- A very standard weighting scheme is: lnc.ltc
- Document: logarithmic tf (l as first character), no idf and cosine normalization

A bad idea?

- Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization …

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

IDF是拿來評價Query中每一個字的權重
所以Document不會有IDF

出現次數 | 取Log+1 | 幾個文件有 | 取inverse, Log+1 | tf–idf | cos標準化 | 出現次數 | 取Log+1 | tf=tf–wt (no DF) | cos標準化 | 評分

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Exercise: what is *N*, the number of docs?

N = 100,000
看term “best”
Log(N/50,000)+1 = 1.3
Log(N/50,000) = 0.3
N/50,000 = 2

Doc length = $\sqrt{1^2+0^2+1^2+1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

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

Resources for today's lecture

- IIR 6.2 – 6.4.3

- http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html
  - Term weighting and cosine similarity tutorial for SEO folk!