

Advanced topics in Computer Science

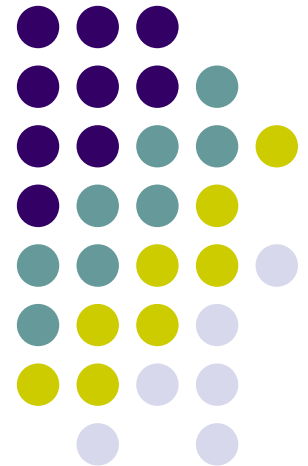
Jiaheng Lu

Department of Computer Science

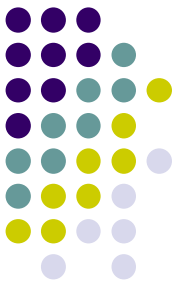
Renmin University of China

www.jiahenglu.net

An Introduction to SaaS and Cloud Computing

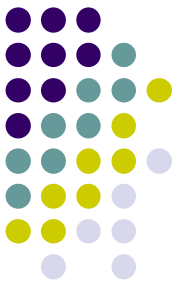


The challenge



Add new services for your users quickly and cost effectively





Expand your Infrastructure!

Buy new servers, increase your software costs, provision more datacenter capacity!!





Look to the cloud!

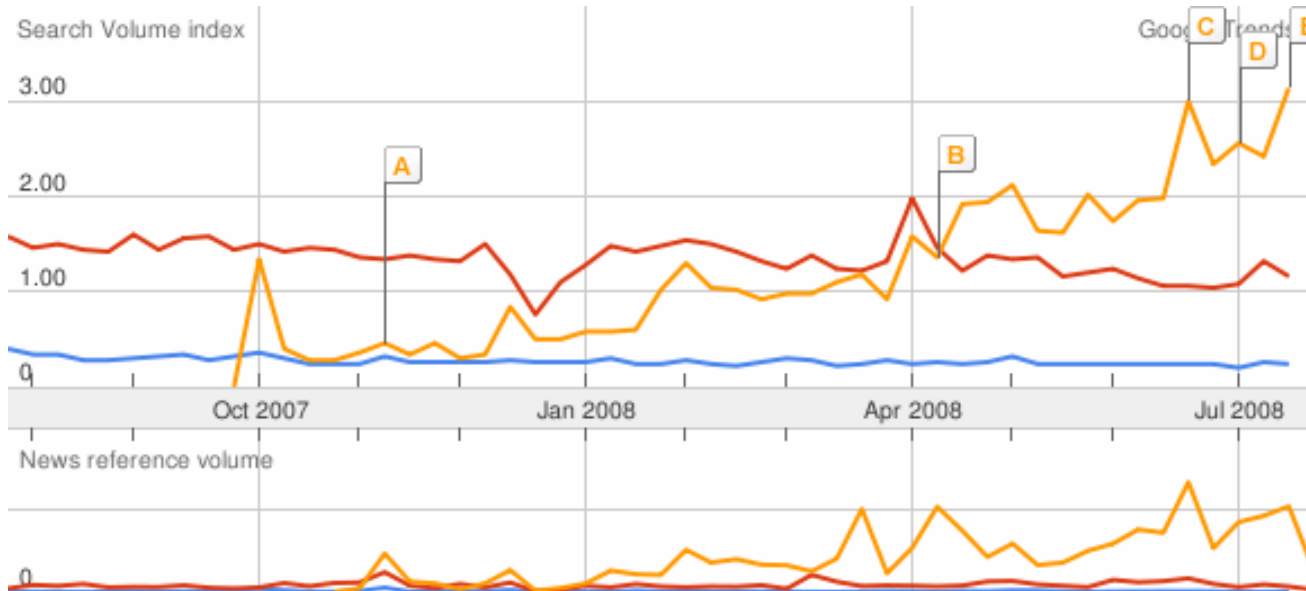
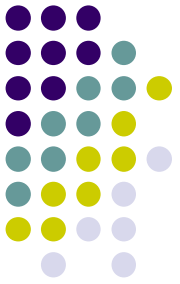
Pay for the bandwidth and server resources that you need. When your push is done then turn the whole thing off!





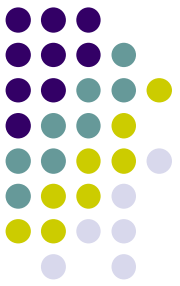
What is
Cloud Computing?

The hype



Cluster Computing
Cloud Computing
Grid Computing





SaaS
Software as a Service

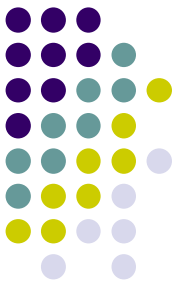
PaaS
Platform as a Service

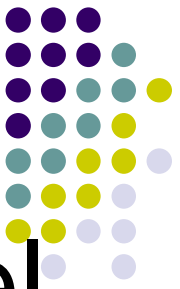
IaaS
Infrastructure as a Service



SaaS

Software as a Service

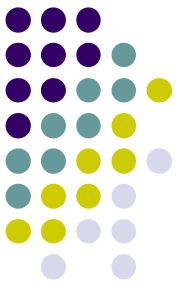




Software delivery model

- No hardware or software to manage
- Service delivered through a browser

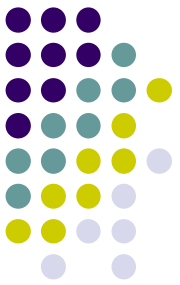




Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs





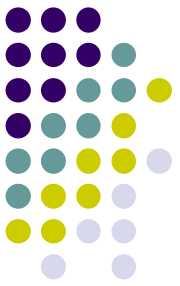
Examples

- CRM
- Financial Planning
- Human Resources
- Word processing

Commercial Services:

- Salesforce.com
- emailcloud

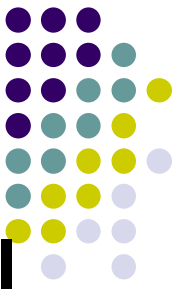




PaaS

Platform as a Service



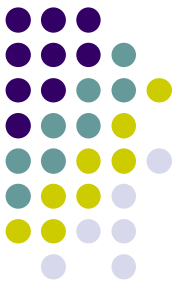


Platform delivery model

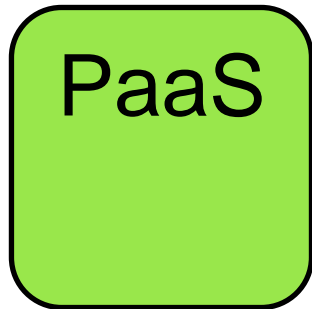


- Platforms are built upon Infrastructure, which is expensive
- Estimating demand is not a science!
- Platform management is not fun!



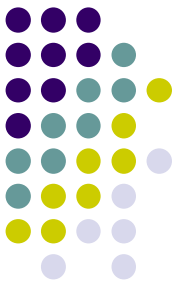


Popular services

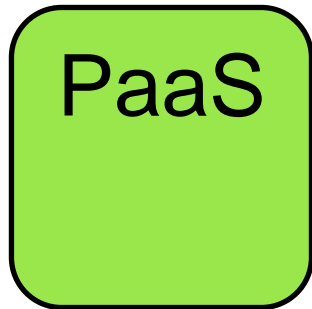


- Storage
- Database
- Scalability



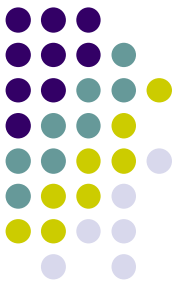


Advantages

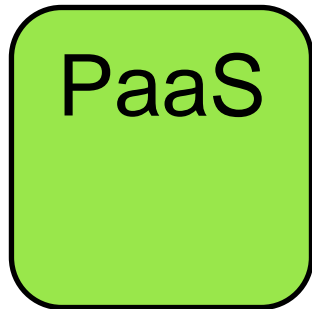


- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs



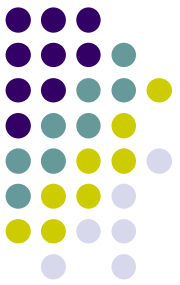


Examples



- Google App Engine
- Mosso
- AWS: S3

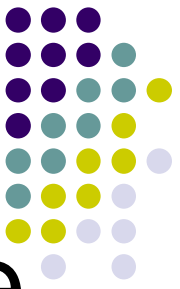




laaS

Infrastructure as a Service

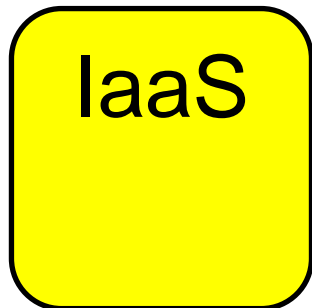


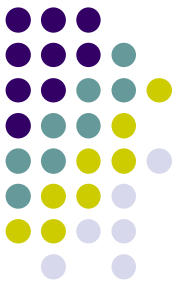


Computer infrastructure delivery model

Access to infrastructure stack:

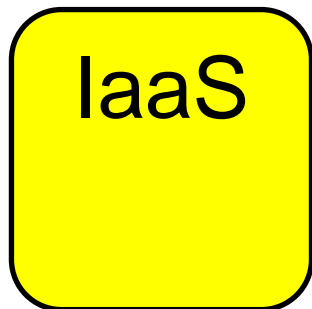
- Full OS access
- Firewalls
- Routers
- Load balancing

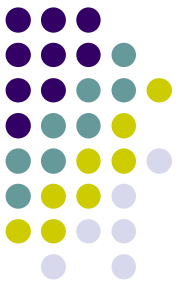




Advantages

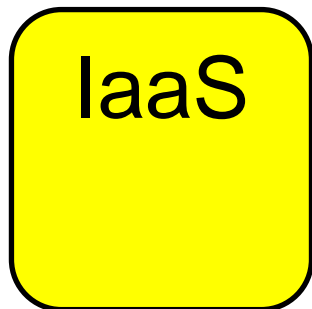
- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

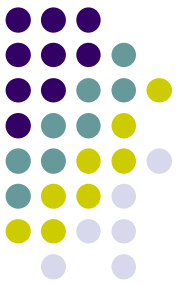




Examples

- Flexiscale
- AWS: EC2



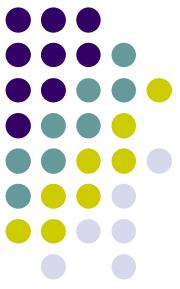


SaaS
Software as a Service

PaaS
Platform as a Service

IaaS
Infrastructure as a Service





SaaS

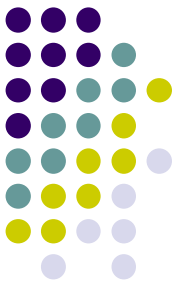
Common Factors

PaaS

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

IaaS





SaaS

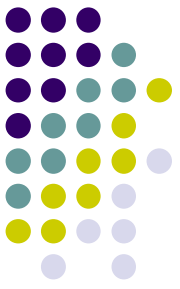
Advantages

PaaS

IaaS

- Lower cost of ownership
- Reduce infrastructure management responsibility
- Allow for unexpected resource loads
- Faster application rollout





SaaS

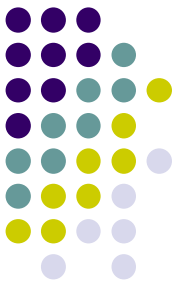
PaaS

IaaS

Cloud Economics

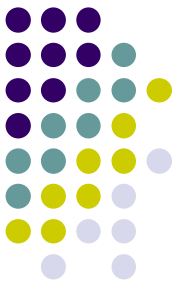
- Multi-tenanted
- Virtualisation lowers costs by increasing utilisation
- Economies of scale afforded by technology
- Automated update policy





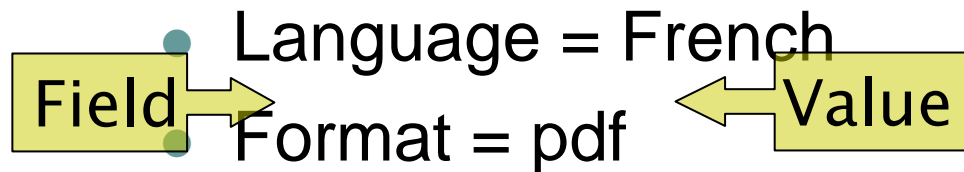
This lecture

- Parametric and field searches
 - Zones in documents
- Scoring documents: zone weighting
 - Index support for scoring
- Term weighting
- Vector space retrieval



Parametric search

- Most documents have, in addition to text, some “meta-data” in fields e.g.,



- Subject = Physics etc.
- Date = Feb 2000
- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
 - language, date range, etc.

Parametric search example



CarFinder.com 

Over one million fictional vehicles to choose from!

Choose your search criteria from the drop down menus:

Number of results to display:

Make Model Category Year
City Color Price



Reset Filters

Reset Sorts

Notice that the output is a (large) table. Various parameters in the table (column headings) may be clicked on to effect a sort.

Make	Model	Year	City	Mileage	Price	Category	Description	Color
BMW	5-Series	1995	San Francisco	16100	11100	Luxury	Never driven in winter conditions. Body work makes it look like new. Keyless entry and security features. This is a bargain.	Silver
BMW	5-Series	1995	San Francisco	16600	11100	Luxury	Great first car for your teen-aged kid. Solid, dependable, affordable with 0% down and owner financing.	Blue
BMW	5-Series	1995	San Francisco	16800	11200	Luxury	Upgraded sound system really rocks. Customized interior features wood grain dash and beige leather seats. Power locks, windows, steering. Price firm.	White
BMW	5-Series	1995	San Francisco	16100	11300	Luxury	Safe choice for a young family: ABS, driver and passenger air bags. Roomy interior with power everything. Low mileage driving kids back and forth to soccer.	Maroon
BMW	5-Series	1995	San Francisco	16300	11400	Luxury	This baby's got it all: power steering, cruise, power locks, power windows, remote entry, leather interior, security alarm, AM/FM/CD/Cassette. Priced to sell!	Brown

Parametric search example



CarFinder.com 

Over one million fictional vehicles to choose from!

We can add text search.

Choose your search criteria from the drop down menus:

Number of results to display:

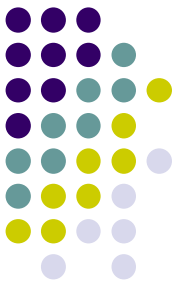
Make Model Category Year
City Color Price Description

Reset Filters

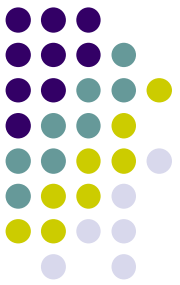
Reset Sorts

Make	Model	Year	City	Mileage	Price	Category	Description	Color
BMW	5-Series	1997	San Francisco	14300	13100	Luxury	5-speed, heavy-duty suspension, extra wide tires. Well-maintained by mechanic-owner. Cloth seats and upgraded stereo system.	White
BMW	5-Series	1997	San Francisco	14600	13100	Luxury	Is that price for real? You bet it is. Fully loaded with all factory options. Former floor model.	Beige
BMW	5-Series	1997	San Francisco	14900	13100	Luxury	Fun to drive. Manual 5-speed transmission, turbo charger. Garaged all winter and pampered the rest of the year. This is a steal!	Orange
BMW	5-Series	1997	San Francisco	14800	13200	Luxury	Fully loaded, automatic transmission. Power everything. Anti-lock brakes and full safety features. Must test drive. Price firm.	Green
BMW	5-Series	1997	San Francisco	14300	13200	Luxury	Formerly an executive's vehicle. Interior has been professionally maintained, engine factory serviced every 3000 miles. Great gas mileage. Price negotiable.	Maroon
BMW	5-Series	1997	San Francisco	15000	13200	Luxury	Sun roof, air, CD player, driver side air bag. 10% deposit required. Owner financing available. Best offer by end of weekend buys it.	Red

Zones

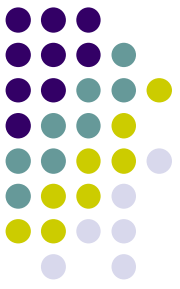


- A zone is an identified region within a doc
 - E.g., Title, Abstract, Bibliography
 - Generally culled from marked-up input or document metadata (e.g., powerpoint)
- Contents of a zone are free text
 - Not a “finite” vocabulary
- Indexes for each zone - allow queries like
 - ***sorting*** in Title AND ***smith*** in Bibliography AND ***recur**** in Body

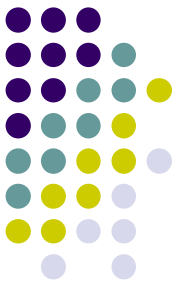


Scoring and Ranking

Scoring

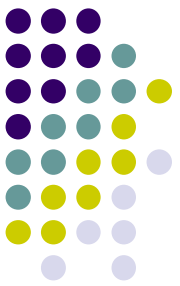


- Thus far, our queries have all been Boolean
 - Docs either match or not
- OK for expert users with precise understanding of their needs and the corpus
- Not good for (the majority of) users with poor Boolean formulation of their needs
- Most users don't want to wade through 1000's of results – cf. use of web search engines



Scoring

- *We wish to return in order the documents most likely to be useful to the searcher*
- How can we rank order the docs in the corpus with respect to a query?
- **Assign a score – say in $[0,1]$ for each doc d on each query q**
- Begin with a perfect world – no spammers
 - Nobody stuffing keywords into a doc to make it match queries



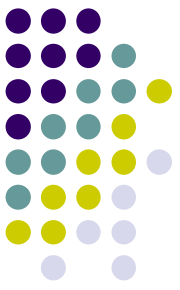
Linear zone combinations

- First generation of scoring methods: use a linear combination of Booleans:
 - E.g.,

$$\text{Score} = 0.6 * \langle \textit{sorting} \text{ in } \underline{\text{Title}} \rangle + 0.3 * \langle \textit{sorting} \text{ in } \underline{\text{Abstract}} \rangle + 0.05 * \langle \textit{sorting} \text{ in } \underline{\text{Body}} \rangle + 0.05 * \langle \textit{sorting} \text{ in } \underline{\text{Boldface}} \rangle$$

- Each expression such as $\langle \textit{sorting} \text{ in } \underline{\text{Title}} \rangle$ takes on a value in $\{0,1\}$.
- Then the overall score is in $[0,1]$.

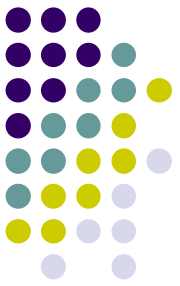
For this example the scores can only take on a finite set of values – what are they?



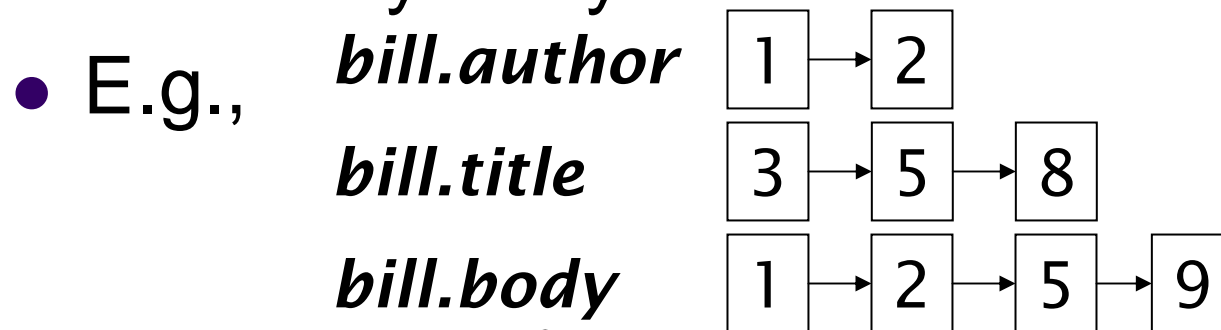
Linear zone combinations

- In fact, the expressions between $\langle \rangle$ on the last slide could be *any* Boolean query
- Who generates the Score expression (with weights such as 0.6 etc.)?
 - In uncommon cases – the user, in the UI
 - Most commonly, a query parser that takes the user's Boolean query and runs it on the indexes for each zone

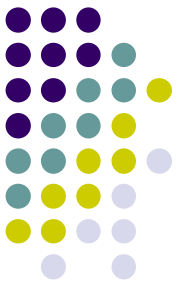
Index support for zone combinations



- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone



Of course, compress zone names like author/title/body.



Zone combinations index

- The above scheme is still wasteful: each term is potentially replicated for each zone
- In a slightly better scheme, we encode the zone in the postings:

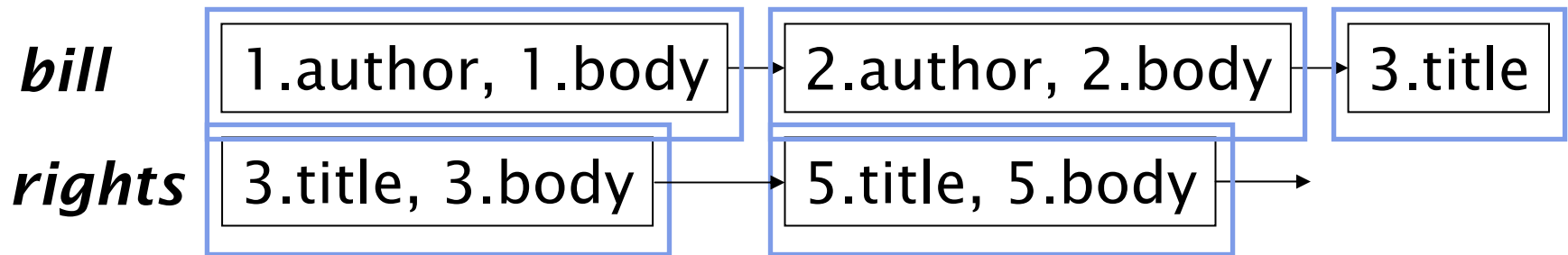
bill 1.author, 1.body → 2.author, 2.body → 3.title

As before, the zone names get compressed.

- At query time, accumulate contributions to the total score of a document from the various postings, e.g.,

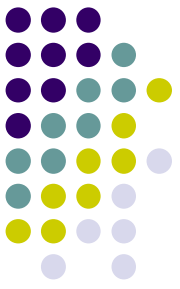
Score accumulation

1	0.7
2	0.7
3	0.4
5	0.4

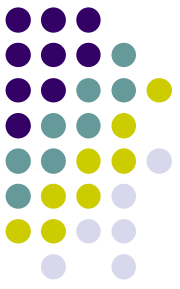


- As we walk the postings for the query **bill OR rights**, we accumulate scores for each doc in a linear merge as before.
- Note: we get both **bill** and **rights** in the Title field of doc 3, but score it no higher.
- Should we give more weight to more hits?

Where do these weights come from?



- Machine learned scoring
- **Given**
 - *A test corpus*
 - *A suite of test queries*
 - *A set of relevance judgments*
- Learn a set of weights such that relevance judgments matched



Simple example

- Each doc has two zones, Title and Body
- For a chosen $w \in [0, 1]$, score for doc d on

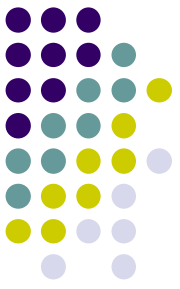
$$\text{score}(d, q) = w \cdot s_T(d, q) + (1 - w) s_B(d, q)$$

where:

$s_T(d, q) \in \{0, 1\}$ is a Boolean denoting whether q matches the Title and

$s_B(d, q) \in \{0, 1\}$ is a Boolean denoting whether q matches the Body

Where do these weights come from?



- Machine learned scoring
- **Given**
 - *A test corpus*
 - *A suite of test queries*
 - *A set of relevance judgments*
- Learn a set of weights such that relevance judgments matched

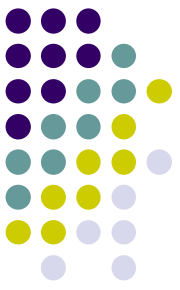
Learning w from training examples



Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

We are given *training examples*, each of which is a triple: DocID d , Query q and Judgment *Relevant/Non*.

From these, we will learn the best value of w .



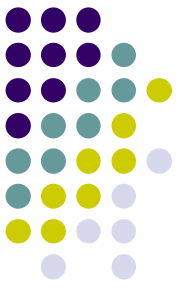
How?

- For each example Φ_t we can compute the score bas $score(d_t, q_t) = w \cdot s_T(d_t, q_t) + (1 - w)s_B(d_t, q_t)$.
- We quantify Relevant as 1 and Non-relevant as 0
 - Would like the choice of w to be such that the computed scores are as close to these 1/0 judgments as possible
 - Denote by $r(d_t, q_t)$ the judgment for Φ_t
- $\text{T} \sum_{\Phi_t} (r(d_t, q_t) - score(d_t, q_t))^2$



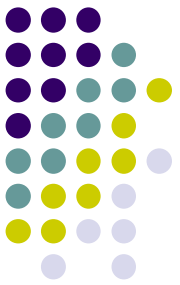
Full text queries

- Most users more likely to type ***bill rights*** or ***bill of rights***
 - How do we interpret these full text queries?
 - No Boolean connectives
 - Of several query terms, some may be missing in a doc
 - Only some query terms may occur in the title, etc.

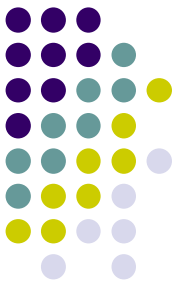


Scoring: density-based

- Thus far: position and overlap of terms in a doc – title, author etc.
- Obvious next: idea if a document talks about a topic *more*, then it is a better match
- This applies even when we only have a single query term.
- Document relevant if it has many occurrences of the term(s)
- This leads to the idea of term weighting.



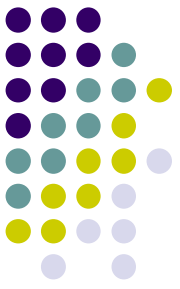
Term frequency and weighting



Term frequency vectors

- Consider the number of occurrences of a term t in a document d , denoted $tf_{t,d}$
 - Document is a vector: a column below
 - Bag of words model

	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



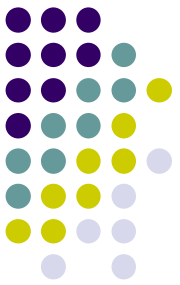
Scores from term frequencies

- Given a free-text query q , define

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

Simply add up the term frequencies of all query terms in the document

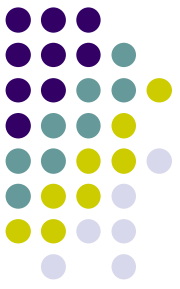
This assigns a score to each document; now rank-order documents by this score.



Bag of words view of a doc

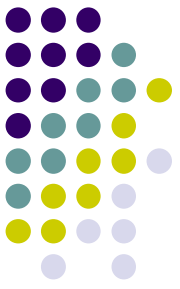
- Thus the doc
 - *John is quicker than Mary.*
- is indistinguishable from the doc
 - *Mary is quicker than John.*

Which of the indexes discussed so far distinguish these two docs?



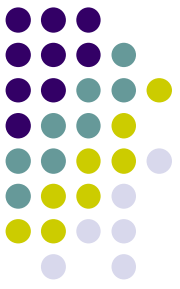
Adding frequencies

- Consider query *ides of march*
 - *Julius Caesar* has 5 occurrences of *ides*
 - No other play has *ides*
 - *march* occurs in over a dozen
 - All the plays contain *of*
- By this scoring measure, the top-scoring play is likely to be the one with the most *of* s



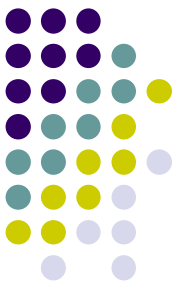
Digression: terminology

- WARNING: In a lot of IR literature, “frequency” is used to mean “count”
 - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
 - Not divided by document length (which would actually make it a frequency)
- We will conform to this misnomer
 - In saying term frequency we mean the number of occurrences of a term in a document.



Term frequency $tf_{t,d}$

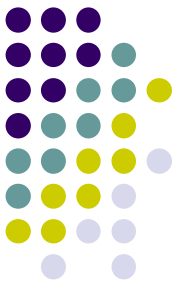
- Long docs are favored because they're more likely to contain query terms
- Can fix this to some extent by normalizing for document length
- But is raw $tf_{t,d}$ the right measure?



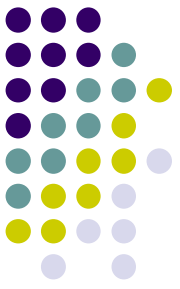
Weighting term frequency: *tf*

- What is the relative importance of
 - 0 vs. 1 occurrence of a term in a doc
 - 1 vs. 2 occurrences
 - 2 vs. 3 occurrences ...
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few
 - Can just use raw *tf*
 - Another option commonly used in practice:
 $wf_{t,d} = 0$ if $tf_{t,d} = 0$, $1 + \log tf_{t,d}$ otherwise

Weighting should depend on the term overall



- Which of these tells you more about a doc?
 - 10 occurrences of *hernia*?
 - 10 occurrences of *the*?
- Would like to attenuate the weights of *common terms*
 - But what is “common”?
- Suggestion: look at collection frequency (*cf*)
 - The total number of occurrences of the term in the entire collection of documents

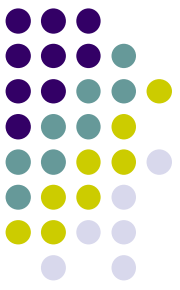


Document frequency

- But document frequency (df) may be better:
- df = number of docs in the corpus containing the term

Word	cf	df
<i>try</i>	10422	8760
<i>insurance</i>	10440	3997

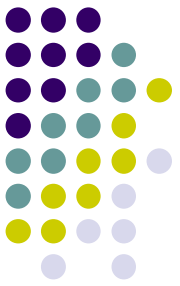
- Document/collection frequency weighting is only possible in known (static) collection.
- So how do we make use of df ?



Reuters RCV1 800K docs

- Logarithms are base 10

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

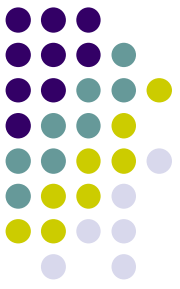


tf x idf term weights

- tf x idf measure combines:
 - term frequency (*tf*)
 - or *wf*, some measure of term density in a doc
 - inverse document frequency (*idf*)
 - measure of informativeness of a term: its rarity across the whole corpus
 - could just be raw count of number of documents the term occurs in ($idf_t = 1/df_t$)
 - but by far the most commonly used version is:

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

- See Papineni, NAACL 2, 2002 for theoretical justification



Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term i in each document d

$$w_{t,d} = tf_{t,d} \times \log(N / df_t)$$

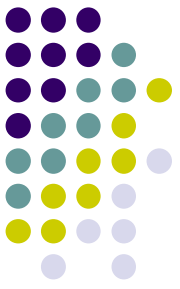
What is the wt of a term that occurs in all of the docs?

$tf_{t,d}$ = frequency of term t in document d

N = total number of documents

df_t = the number of documents that contain term t

- Increases with the number of occurrences *within* a doc

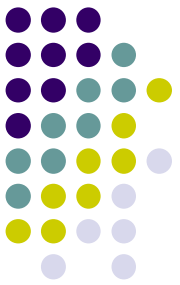


Real-valued term vectors

- Still Bag of words model
- Each is a vector in \mathbb{R}^M
 - Here log-scaled *tf.idf*

Note can be >1!

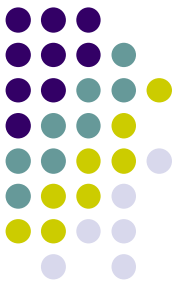
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0



Documents as vectors

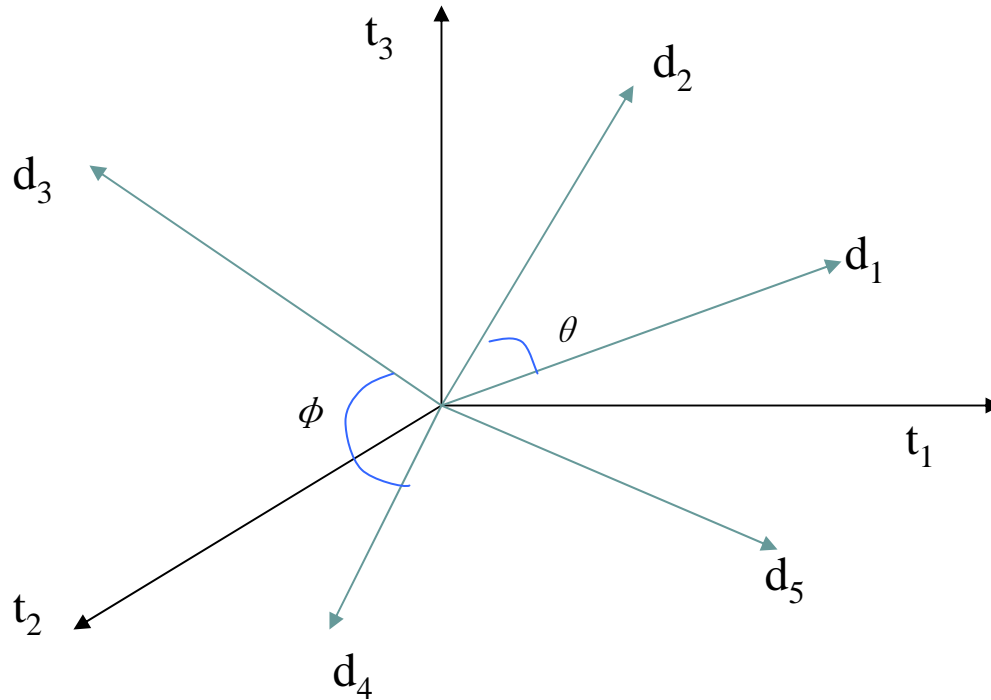
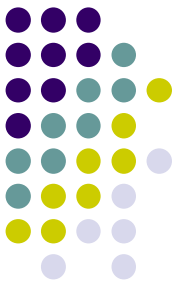
- Each doc j can now be viewed as a vector of $wf \times idf$ values, one component for each term
- So we have a vector space
 - terms are axes
 - docs live in this space
 - even with stemming, may have 20,000+ dimensions

Why turn docs into vectors?



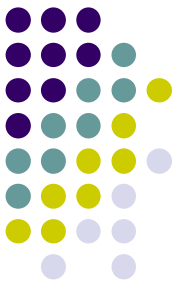
- First application: Query-by-example
 - Given a doc D , find others “like” it.
- Now that D is a vector, find vectors (docs) “near” it.

Intuition



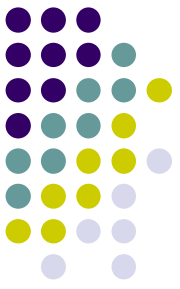
Postulate: Documents that are “close together” in the vector space talk about the same things.

The vector space model



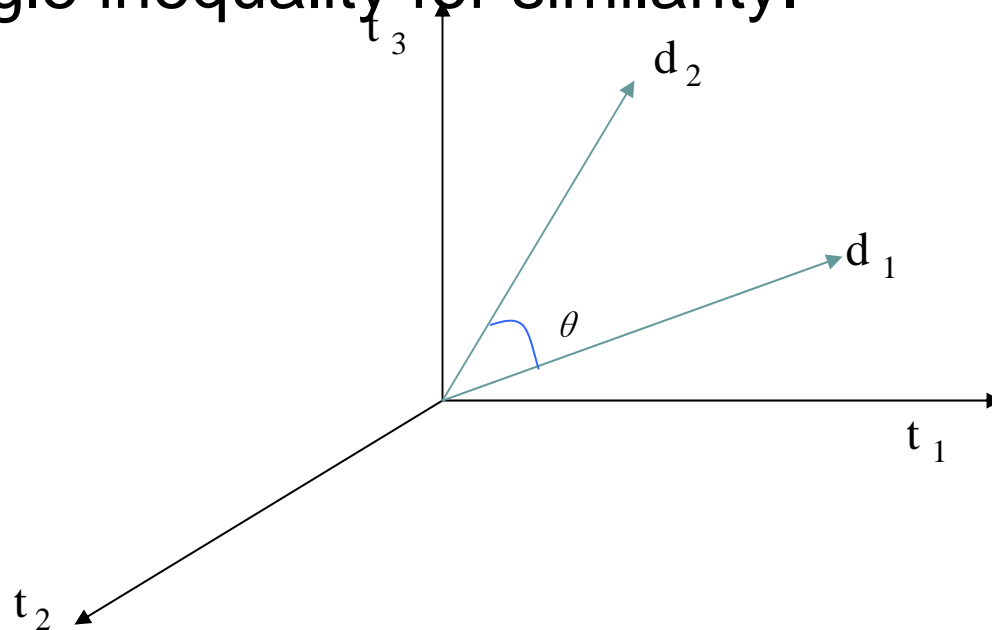
Freetext query as vector:

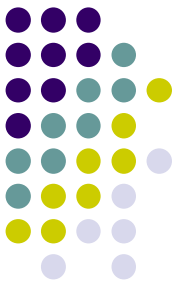
- We regard freetext query as short document
- We return the documents ranked by the closeness of their vectors to the query vector.



Cosine similarity

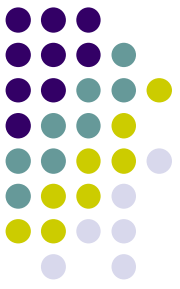
- Distance between vectors d_1 and d_2 captured by the cosine of the angle x between them.
- Note – this is *similarity*, not distance
 - No triangle inequality for similarity.





Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm
$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$$
- This maps vectors onto the unit sphere:
$$|d_j| = \sqrt{\sum_{i=1}^M w_{i,j}^2} = 1$$
- Then,
- Longer documents don't get more weight

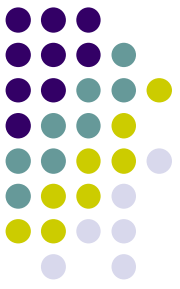


Cosine similarity

$$\text{sim}(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{|\vec{d}_j| |\vec{d}_k|} = \frac{\sum_{i=1}^M w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^M w_{i,j}^2} \sqrt{\sum_{i=1}^M w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

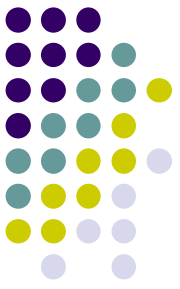
Normalization



Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$



Example

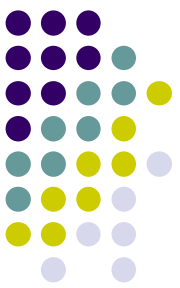
- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*

	SaS	PaP	WH
<i>affection</i>	115	58	20
<i>jealous</i>	10	7	11
<i>gossip</i>	2	0	6

	SaS	PaP	WH
<i>affection</i>	0.996	0.993	0.847
<i>jealous</i>	0.087	0.120	0.466
<i>gossip</i>	0.017	0.000	0.254

- $\cos(\text{SAS}, \text{PAP}) = .996 \times .993 + .087 \times .120 + .017 \times 0.0 = 0.999$
- $\cos(\text{SAS}, \text{WH}) = .996 \times .847 + .087 \times .466 + .017 \times .254 = 0.929$

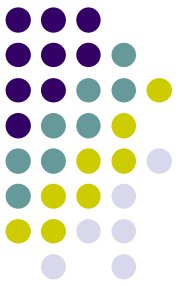
Basic cosine score computation



COSINESCORE(q)

```
1  float  $Scores[N] = 0$ 
2  Initialize  $Length[N]$ 
3  for each query term  $t$ 
4  do
5      calculate  $w_{t,q}$  and fetch inverted list for  $t$ 
6      for each pair( $d, tf_{t,d}$ ) in inverted list
7      do
8          add  $wf_{t,d} \times w_{t,q}$  to  $Scores[d]$ 
9          Read the array  $Length[d]$ 
10 for each  $d$ 
11 do Divide  $Scores[d]$  by  $Length[d]$ 
12 return Top  $K$  components of  $Scores[]$ 
```

Notes on score computation



- Why not store normalized tf-idf value at each postings entry?
- idf the same for all postings entries in a single postings list (term)
- What about tf/normalization?
 - Space blowup because of floats
 - tf's stored as integers
 - Lengths stored once per doc