Information Retrieval

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Christopher Manning, Prabhakar Raghavan, Hinrich Schütze, Jim Martin, Donald Patterson, Min-Yen Kan, Dell Zhang, Yisong Yue, Zhuyun Dai, Jamie Callan

Information Retrieval

Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

These days we frequently think first of web search, but there are many other cases:

- E-mail search
- Searching your laptop
- Corporate knowledge bases
- Legal information retrieval

Basic Assumptions of IR

Collection: A set of documents

- Assume it is a static collection for the moment

Goal: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task

Outline

Classic IR systems

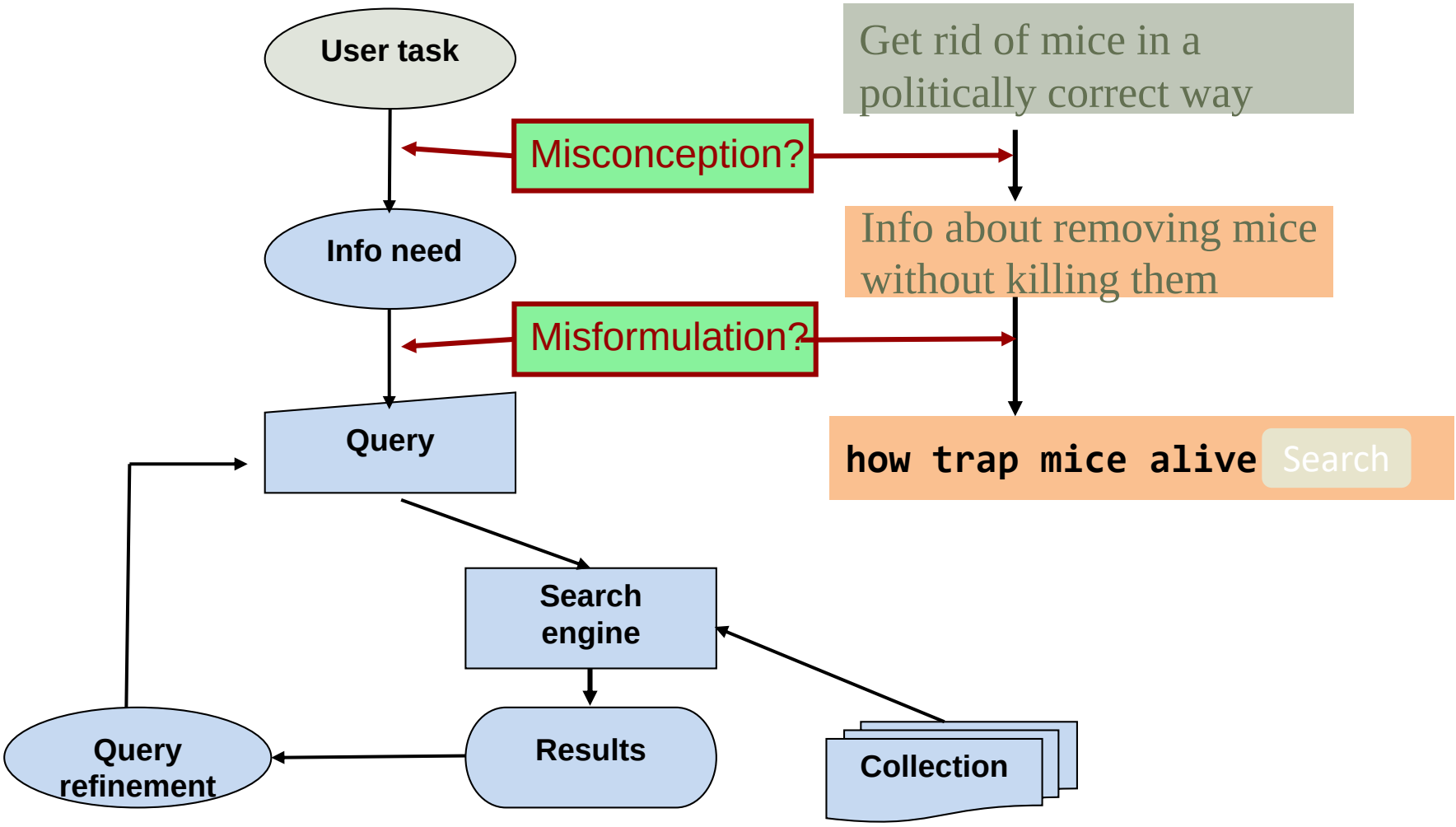
Relevance feedback and query expansion

Evaluation metrics

Learning to rank

NN Ranking models

The Classic Search Model



Start with a Large Document Collection

Consider N = 1 million documents

Say there are M = 500K *distinct* terms among these.

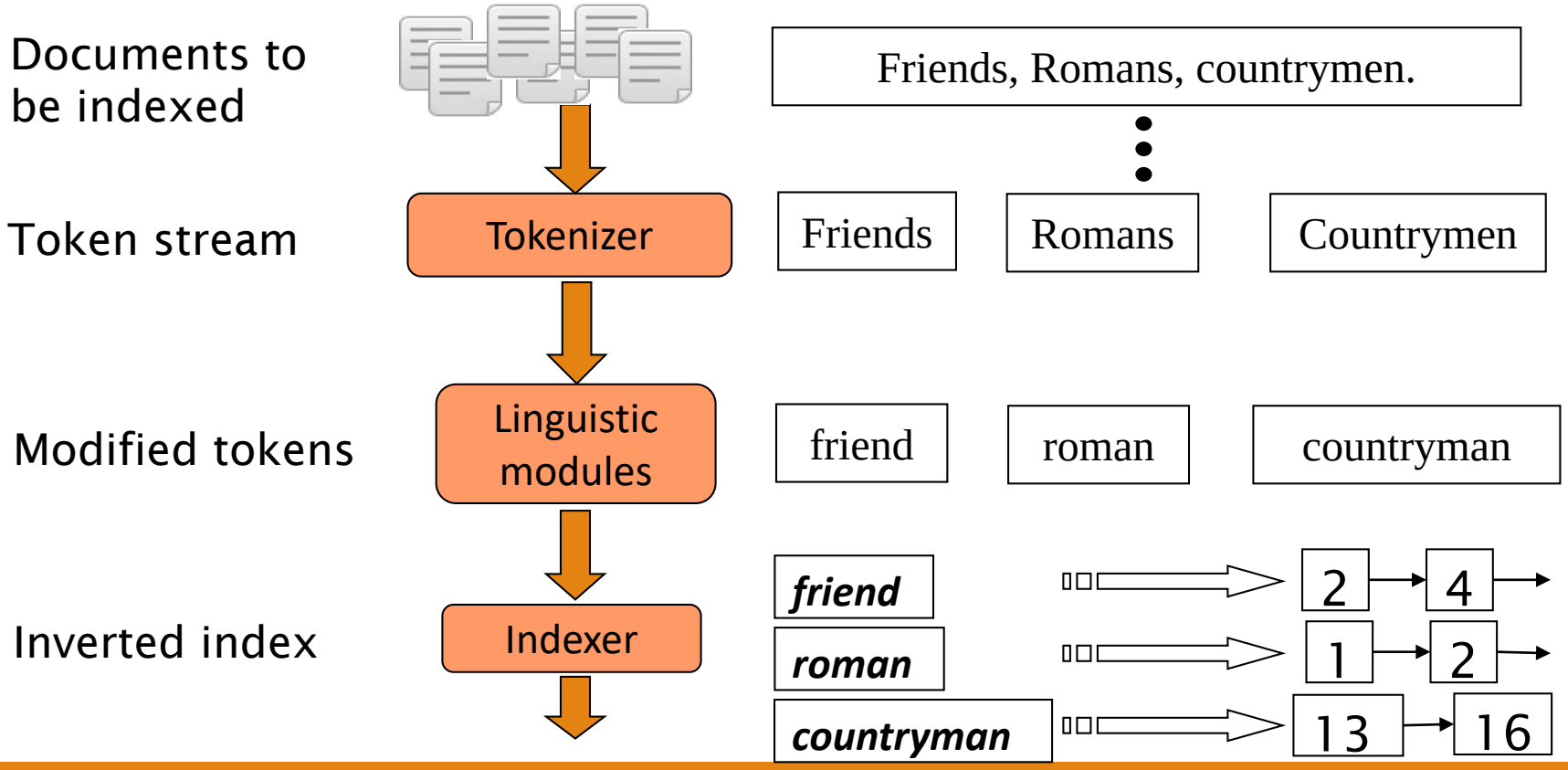
Can't build a N by M matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's: *extremely sparse*
- What's a better representation?: *only record the 1 positions*

Inverted Index

For each term *t*, we must store a list of all documents that contain *t*

Inverted index construction:



Initial Stages of Text Processing

Tokenization: cut character sequence into word tokens

- Deal with ***"John's"***, ***"state-of-the-art"***

Normalization: map text and query term to same form

- You want ***U.S.A.*** and ***USA*** to match
- Reduce all letters to lower case

Stemming: we may wish different forms of a root to match

- ***authorize***, ***authorization***

Stop words: we may omit very common words (or not)

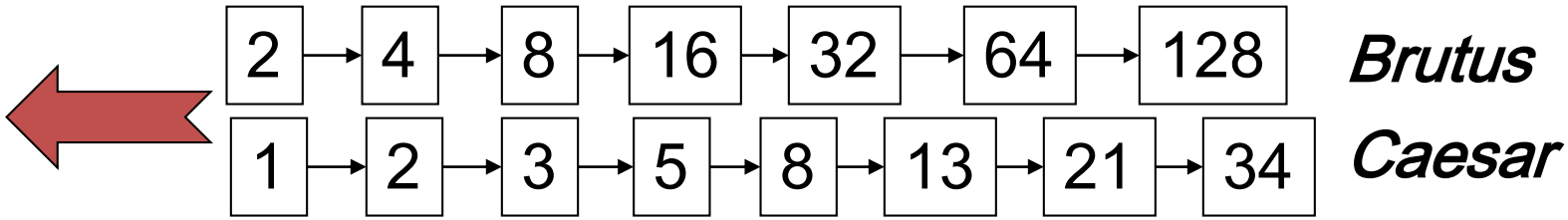
- ***the, a, to, of***

Query Processing

Starting with AND

Consider processing the query: ***Brutus*** *AND* ***Caesar***

- Locate ***Brutus*** in the Dictionary
  - Retrieve its documents
- Locate ***Caesar*** in the Dictionary
  - Retrieve its documents
- "Merge" the two doc lists (intersect the document sets):



Boolean Queries: Exact Match

The Boolean retrieval model is being able to ask a query that is a Boolean expression:

- Boolean Queries are queries using AND, OR and NOT to join query terms
  - Views each document as a set of words
  - Is precise: document matches condition or not.
- Perhaps the simplest model to build an IR system on

Primary commercial retrieval tool for 3 decades

Many search systems you still use are Boolean:

- Email, library catalog, Mac OS X Spotlight

Phrase Queries

We want to be able to answer queries, e.g., ***"stanford university"*** – as a phrase

Thus the sentence *"I went to university at Stanford"* is not a match

- The concept of phrase queries has proven easily understood by users
- Many more queries are *implicit phrase* queries

For this, it no longer suffices to store only *<term : docs>* entries

First attempt: index bigrams

- Does it work for the example above?

Support Phrase Queries: Position Indexes

In the document store, for each ***term*** the position(s) in which tokens of it appear:

<***term***, number of docs containing ***term***;

doc1: position1, position2 … ;

doc2: position1, position2 … ;

etc.>

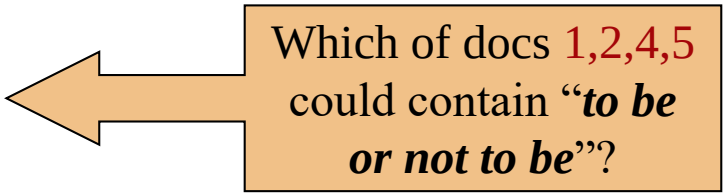
<***be***: 993427;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, **430, 434**;

5: 363, 367, …>



For phrase queries, we use a merge algorithm recursively at the document level

- We now need to deal with more than just equality

More on Token Processing: Thesauri and Soundex

Do we handle synonyms and homonyms?

- E.g., by hand-constructed equivalence classes
  - ***car*** = ***automobile*** ***color*** = ***colour***
- We can rewrite to form equivalence-class terms
  - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
- Or we can expand a query
  - When the query contains ***automobile***, look under ***car*** as well

What about spelling mistakes?

- One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

More on Token Processing: Stemming and Lemmatization

Lemmatization

- Reduce inflectional/variant forms to base form, e.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's Algorithm

A commonly used algorithm for stemming English

- Results suggest it's at least as good as other stemming options

Conventions + 5 phases of reductions

- phases applied sequentially
- each phase consists of a set of commands
- sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses* $\rightarrow$ *ss*
- *ies* $\rightarrow$ *i*
- *ational* $\rightarrow$ *ate*
- *tional* $\rightarrow$ *tion*

Does Stemming Help?

- English: very mixed results. Helps recall for some queries but harms precision on others
  - E.g., operative (dentistry) ⇒ oper
- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

Problem With Boolean Search: Feast Or Famine

Thus far, our queries have all been Boolean.

- Documents either match or don't.

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.

Boolean queries often result in either too few (=0) or too many (1000s) results

- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits

Ranked Retrieval Models

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

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".

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

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.

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

TF-IDF: Term Frequency

The term frequency $\text{tf}_{t,d}$ 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.
- But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.

Log-frequency:

$$= \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

TF-IDF: Inverse Document Frequency (IDF)

Frequent terms are less informative than rare terms

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*
- $df_t \leq N$

We define the idf (inverse document frequency) of *t* by

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

$$idf_t = \log_{10}(N/df_t)$$

Tf-idf Weighting

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

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

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

Many variants of tfidf:

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)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

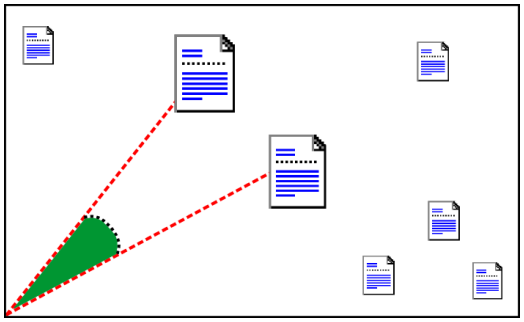
Vector Space Ranking: Cosine (query, document)

Represent documents as vectors

- with tf-idf weighting of words
- Queries as short documents

Similarity Measures

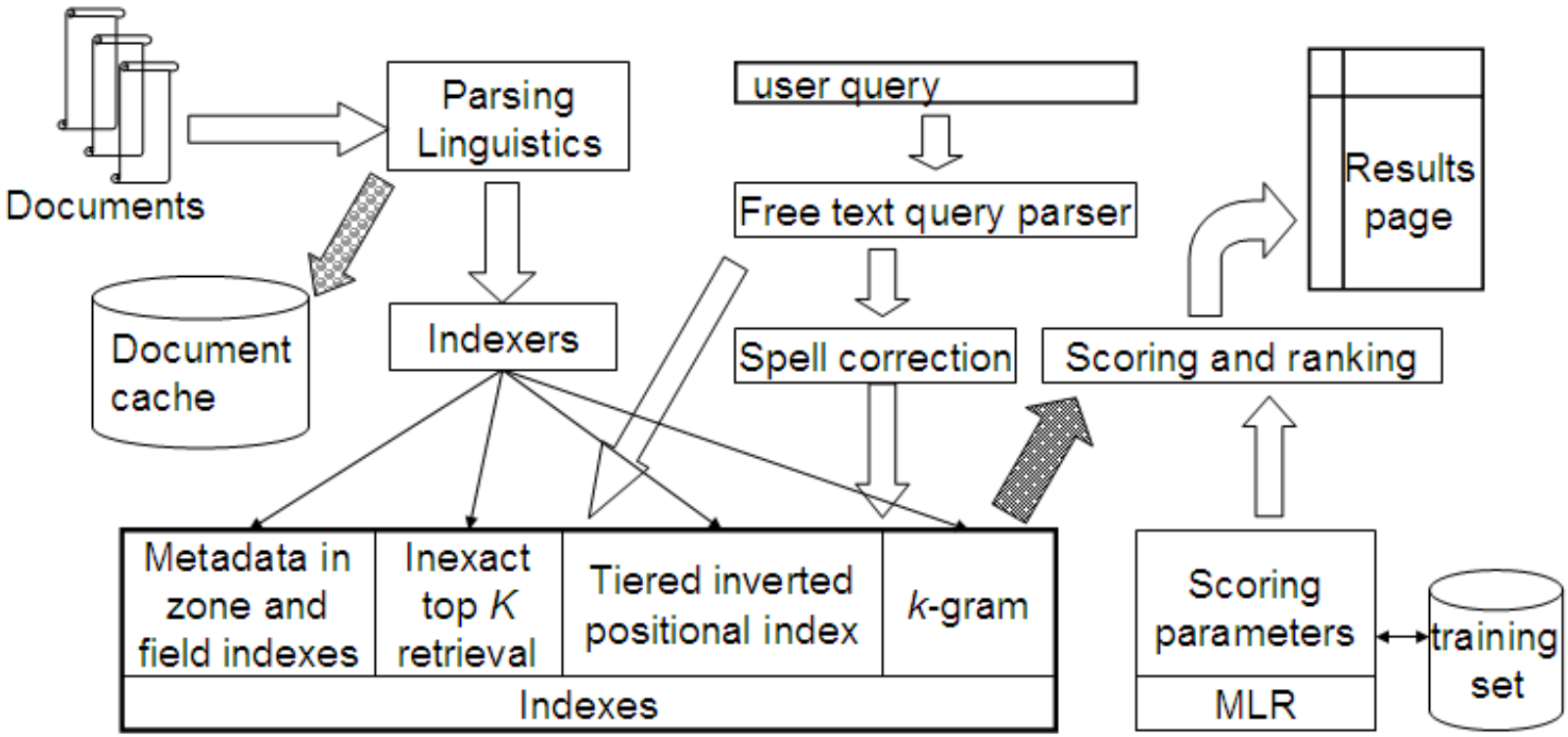
- Cosine similarity = normalized dot product



Cosine Similarity Example

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

Putting It All Together



Outline

Classic IR systems

Relevance feedback and query expansion

Evaluation metrics

Learning to rank

NN Ranking models

Relevance Feedback

Relevance feedback: user feedback on relevance of docs in initial set of results

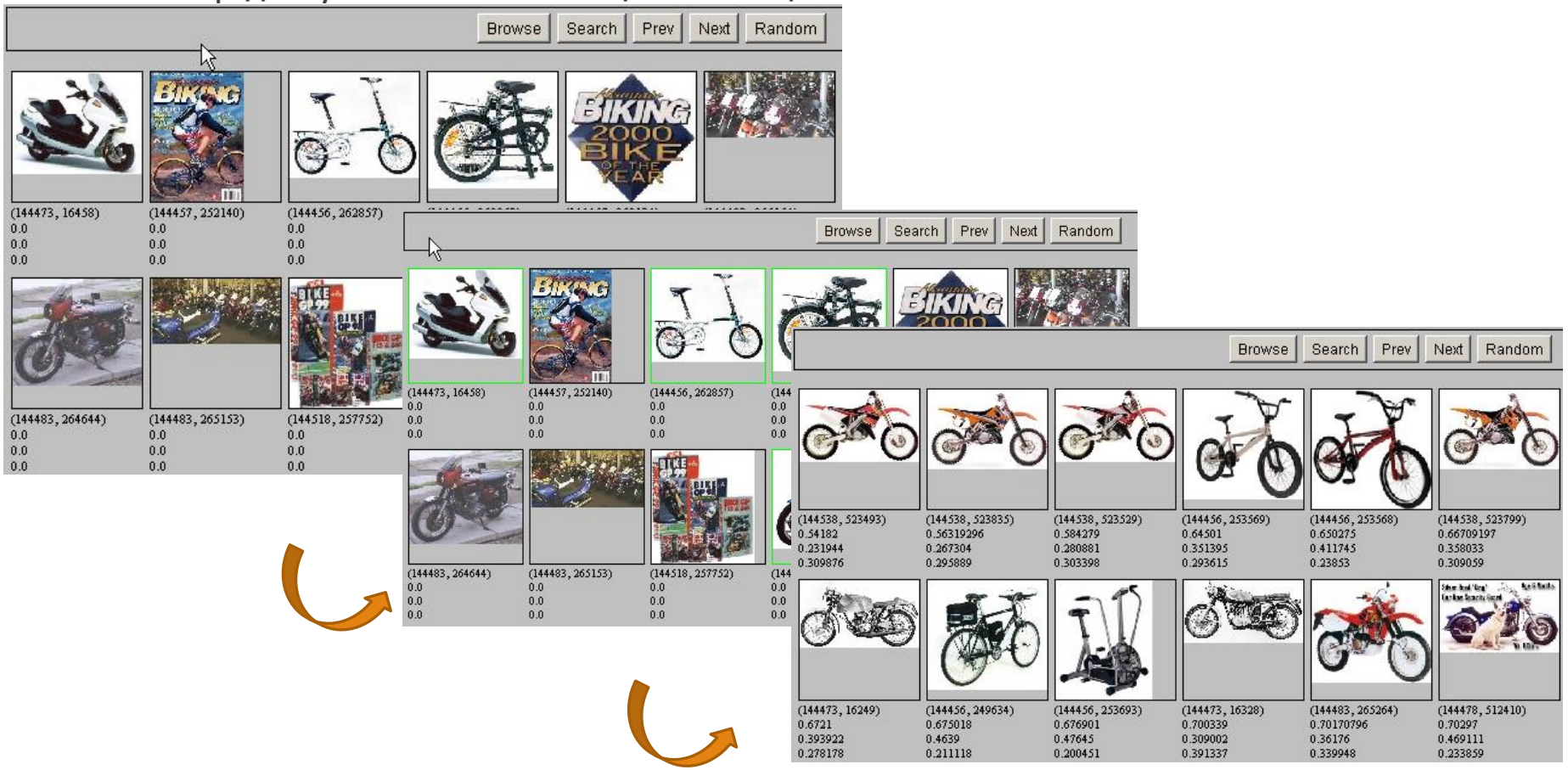
- User issues a (short, simple) query
- The user marks some results as relevant or non-relevant.
- The system computes a better representation of the information need based on feedback.
- Relevance feedback can go through one or more iterations.

Idea: it may be difficult to formulate a good query when you don't know the collection well, so iterate

Relevance Feedback: Example

Image search with “bike” as the query:

- http://nayana.ece.ucsb.edu/imsearch/imsearch.html



Rocchio Algorithm

The Rocchio algorithm uses the vector space model to pick a relevance feedback query

Rocchio seeks the query q_{opt} that maximizes

$$\vec{q}_{opt} = \arg\max_{\vec{q}} [\cos(\vec{q}, \vec{\mu}(C_r)) - \cos(\vec{q}, \vec{\mu}(C_{nr}))]$$

Tries to separate docs marked relevant and non-relevant

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|} \sum_{\vec{d}_j \notin C_r} \vec{d}_j$$

Rocchio 1971 Algorithm (SMART)

Used in practice:

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j$$

D_r = set of known relevant doc vectors

D_{nr} = set of known irrelevant doc vectors
- Different from C_r and C_{nr}

q_m = modified query vector

q_0 = original query vector; α,β,γ: weights (hand-chosen or set empirically)

New query moves toward relevant documents and away from irrelevant documents

Subtleties to note
- Tradeoff α vs. β/γ : If we have a lot of judged documents, we want a higher β/γ.
- Some weights in query vector can go negative
  - Negative term weights are ignored (set to 0)

Relevance Feedback In Vector Spaces

We can modify the query based on relevance feedback and apply standard vector space model.

Use only the docs that were marked.

Relevance feedback can improve recall and precision

Relevance feedback is most useful for increasing *recall* in situations where recall is important

- Users can be expected to review results and to take time to iterate

Positive vs Negative Feedback

- Positive feedback is more valuable than negative feedback (so, set $\gamma < \beta$; e.g. $\gamma = 0.25$, $\beta = 0.75$).
- Many systems only allow positive feedback ($\gamma=0$).

Pseudo Relevance Feedback

Pseudo-relevance feedback automates the “manual” part of true relevance feedback

Pseudo-relevance algorithm

- Retrieve a ranked list of hits for the user’s query
- Assume that the top k documents are relevant.
- Do relevance feedback (e.g., Rocchio)

Works very well on average

But can go horribly wrong for some queries

Several iterations can cause query drift

Why?

Indirect Relevance Feedback

On the web, DirectHit introduced a form of indirect relevance feedback.

DirectHit ranked documents higher that users look at more often.

- Clicked on links are assumed likely to be relevant
  - Assuming the displayed summaries are good, etc.

Globally: Not necessarily user or query specific

- This is the general area of ***clickstream mining***

Today – handled as part of machine-learned ranking

Query Expansion

In relevance feedback, users give additional input (relevant/non-relevant) on documents, which is used to reweight terms in the documents

In query expansion, users give additional input (good/bad search term) on words or phrases

How do we augment the user query?

- Manual thesaurus
  - E.g. MedLine: physician, syn: doc, doctor, MD, medico
  - Can be query rather than just synonyms
- Global Analysis: (static; of all documents in collection)
  - Automatically derived thesaurus
    - (co-occurrence statistics)
  - Refinements based on query log mining
    - Common on the web
- Local Analysis: (dynamic)
  - Analysis of documents in result set

Thesaurus-based Query Expansion

For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus

- feline → feline cat

May weight added terms less than original query terms.

Generally increases recall

Widely used in many science/engineering fields

May significantly decrease precision, particularly with ambiguous terms

- "interest rate" → "interest rate fascinate evaluate"

There is a high cost of manually producing a thesaurus

- And for updating it for scientific changes

Automatic Thesaurus Generation

Attempt to generate a thesaurus automatically by analyzing the collection of documents

Fundamental notion: similarity between two words

- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
  - You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate.

word	ten nearest neighbors
absolutely	absurd whatsoever totally exactly nothing
bottomed	dip copper drops topped slide trimmed slig
captivating	shimmer stunningly superbly plucky witty
doghouse	dog porch crawling beside downstairs gazec
Makeup	repellent lotion glossy sunscreen Skin gel p
mediating	reconciliation negotiate cease conciliation p
keeping	hoping bring wiping could some would othe
lithographs	drawings Picasso Dali sculptures Gauguin
pathogens	toxins bacteria organisms bacterial parasite
senses	grasp psyche truly clumsy naive innate awl

Outline

Classic IR systems

Relevance feedback and query expansion

Evaluation metrics

Learning to rank

NN Ranking models

Evaluating an IR system

Note: the **information need** is translated into a **query**

Relevance is assessed relative to the **information need** *not* the **query**

E.g., Information need: *I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.*

Query: ***wine red white heart attack effective***

Evaluate whether the doc addresses the information need, not whether it has these words

Standard Relevance Benchmarks

TREC - National Institute of Standards and Technology (NIST) has run a large IR evaluation for many years

Reuters and other benchmark doc collections used

“Retrieval tasks” specified

- sometimes as queries

Human experts mark, for each query and for each doc, Relevant or Nonrelevant

- or at least for subset of docs that some system returned for that query

Unranked Retrieval Evaluation: Precision And Recall

Precision: fraction of retrieved docs that are relevant

= P(relevant|retrieved)

Recall: fraction of relevant docs that are retrieved

= P(retrieved|relevant)

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

Precision P = tp/(tp + fp)

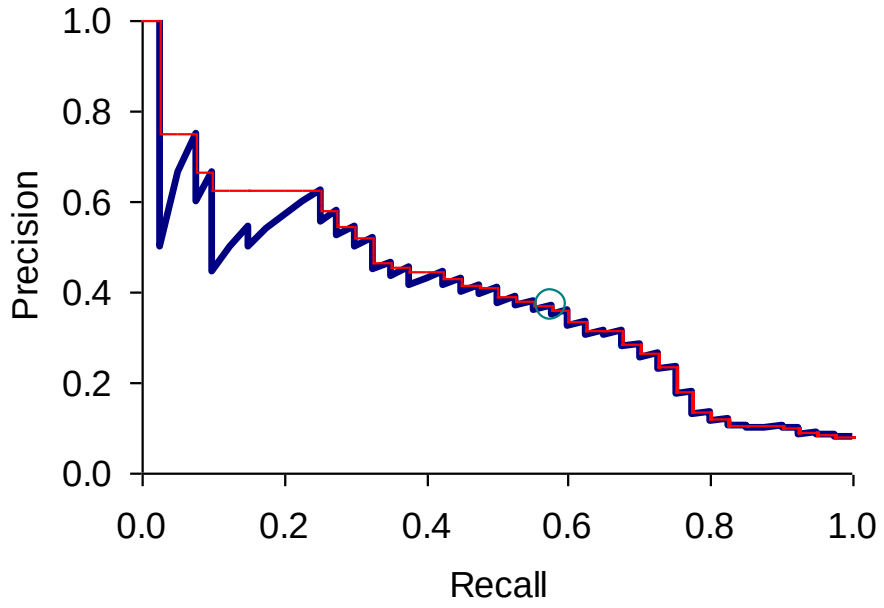
Recall R = tp/(tp + fn)

Evaluating Ranked Results

Evaluation of ranked results:

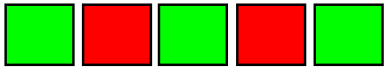
- The system can return any number of results
- By taking various numbers of the top returned documents (levels of recall), the evaluator can produce a *precision-recall curve*

A precision-recall curve



Precision@K and R-Precision

Precision@K

- Set a rank threshold K
- Compute % relevant in top K
- Ignores documents ranked lower than K
- Ex: 
  - Prec@3 of 2/3
  - Prec@4 of 2/4
  - Prec@5 of 3/5

R-precision

- If we have a known (though perhaps incomplete) set of relevant documents of size *Rel,* then calculate precision of the top *Rel* docs returned
- Perfect system could score 1.0.

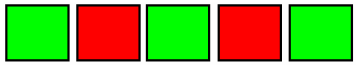
Mean Average Precision

Consider rank position of each **relevance doc**

- K_1, K_2, … K_R

Compute Precision@K for each K_1, K_2, … K_R

Average precision = average of P@K

Ex:  has AvgPrec of $\frac{1}{3}\cdot\left(\frac{1}{1}+\frac{2}{3}+\frac{3}{5}\right)\approx 0.76$

MAP is Average Precision across multiple queries/rankings

Mean Reciprocal Rank

Consider rank position, K, of first relevance doc

Reciprocal Rank score = $\frac{1}{K}$

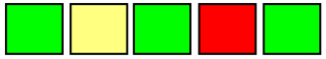
MRR is the mean RR across multiple queries

NDCG

Normalized Discounted Cumulative Gain

Multiple Levels of Relevance

DCG:

- contribution of i-th rank position: $\frac{2^{y_i}-1}{\log(i+1)}$
- Ex:  has DCG score of

$$\frac{1}{\log(2)}+\frac{3}{\log(3)}+\frac{1}{\log(4)}+\frac{0}{\log(5)}+\frac{1}{\log(6)}\approx 5.45$$

NDCG is normalized DCG

- best possible ranking as score NDCG = 1

Outline

Classic IR systems

Relevance feedback and query expansion

Evaluation metrics

Learning to rank

NN Ranking models

Discussed So Far

Basic Approach to IR

- Given query q and set of docs d_1, … d_n
  - Find documents relevant to q
  - Typically expressed as a ranking on d_1,… d_n
- Similarity measure sim(a,b)⟶R
  - Sort by sim(q,d_i)
  - Optimal if relevance of documents are independent. [Robertson, 1977]

Methods

- Cosine
- TF-IDF [Salton & Buckley, 1988]
- Okapi BM25 [Robertson et al., 1995]
- Language Models
  - [Ponte & Croft, 1998]
  - [Zhai & Lafferty, 2001]

Learning to Rank

IR uses fixed models to define similarity scores

Many opportunities to learn models

- Appropriate training data
- Appropriate learning formulation

Supervised learning problem with training data:

- Document/query pairs
  - Embedded in high dimensional feature space
- Labeled by relevance of doc to query
  - Traditionally 0/1
  - Recently ordinal classes of relevance (0,1,2,3,…)

Feature Space

Use to learn a similarity/compatibility function

Based off existing IR methods

- Can use raw values
- Or transformations of raw values

Based off raw words

- Capture co-occurrence of words

$$x_{q,d} = \phi(q,d) = \begin{bmatrix} \sum_i [TF(q_i,d) > 0.1] \\ \sum_i [TF(q_i,d) > 0.05] \\ [rank_{IR}(q,d) \text{ in top } 5] \\ [rank_{IR}(q,d) \text{ in top } 10] \\ sim_{IR}(q,d) \\ \sum_i [w_j == q_i \wedge w_j \in d] \\ \sum_i [w_k == q_i \wedge w_k \in d] \end{bmatrix}$$

Training Instances

Learning Problem

Given training instances:

- $(x_{q,d}, y_{q,d})$ for q = {1..N}, d = {1 .. N_q}

Learn a ranking function

- $f(x_{q,1}, \ldots x_{q,Nq}) \rightarrow$ Ranking

Typically decomposed into per doc scores

- $f(x) \rightarrow R$ (doc/query compatibility)
- Sort by scores for all instances of a given q

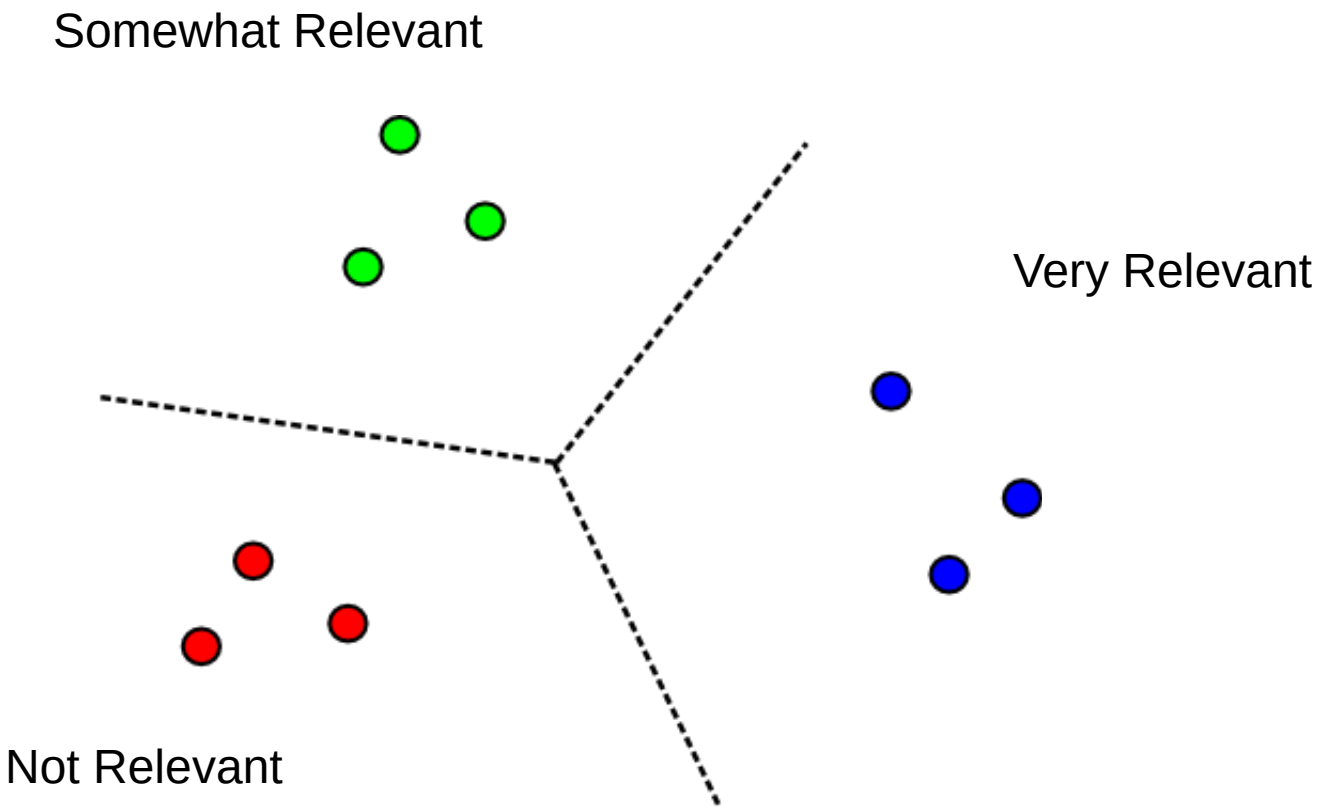
How to Train?

Classification & Regression

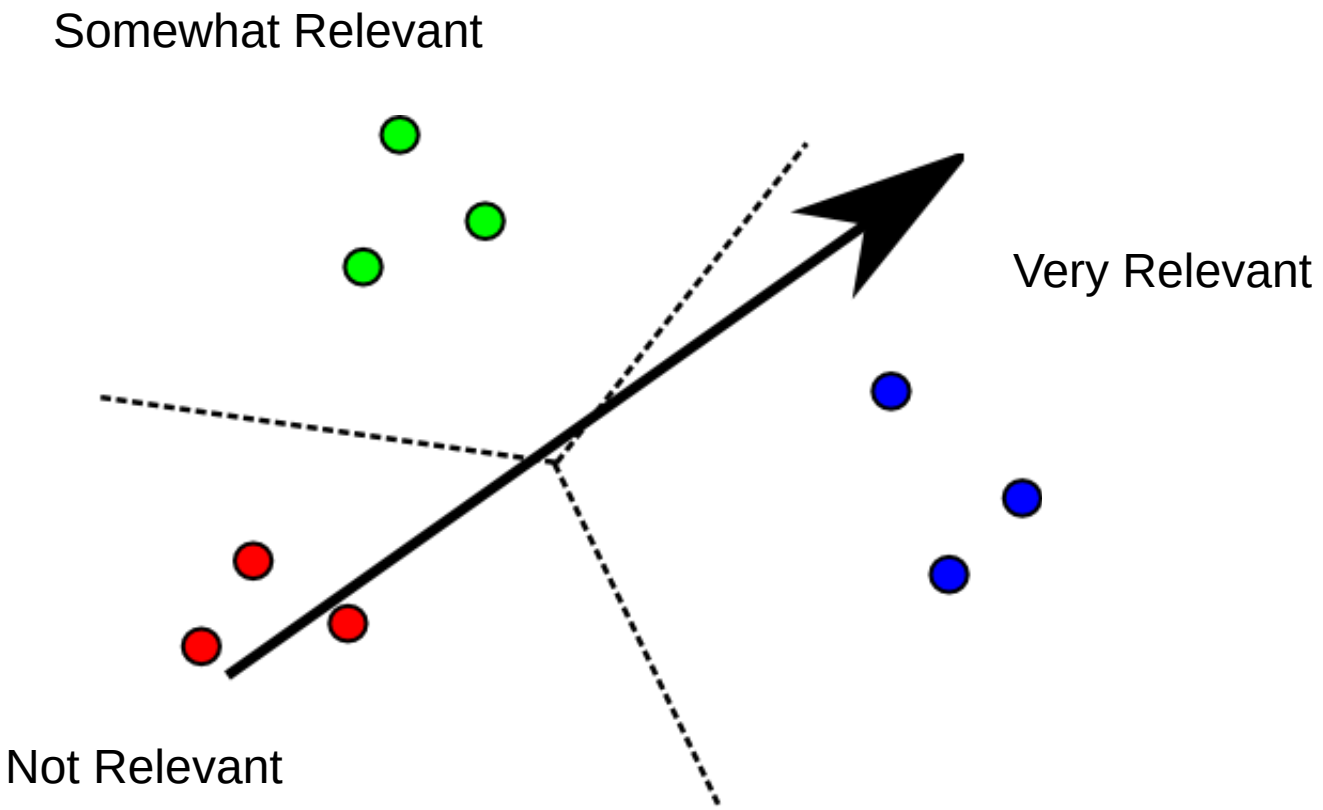
- Learn f(x) $\rightarrow$ R in conventional ways
- Sort by f(x) for all docs for a query

2 Major Problems

- Labels have ordering
  - Additional structure compared to multiclass problems
- Severe class imbalance
  - Most documents are not relevant



Conventional multiclass learning does not incorporate ordinal structure of class labels



Conventional multiclass learning does not incorporate ordinal structure of class labels

Ordinal Regression

Assume class labels are ordered

- True since class labels indicate level of relevance

Learn hypothesis function $f(x) \longrightarrow R$

- Such that the ordering of f(x) agrees with label ordering
- Ex: given instances (x, 1), (y, 1), (z, 2)
  - $f(x) < f(z)$
  - $f(y) < f(z)$
  - Don't care about f(x) vs f(y)

Ordinal regression vs classification & regression

- Compare with classification
  - Similar to multiclass prediction
  - But classes have ordinal structure
- Compare with regression
  - Doesn't necessarily care about value of f(x)
  - Only care that ordering is preserved

Ordinal Regression Approaches

Approach 1: Learn multiple thresholds

- Maintain T thresholds (b_1, … b_T)
- $b_1 < b_2 < \ldots < b_T$
- Learn model parameters + (b_1, …, b_T)

Approach 2: Learn multiple classifiers

- Use T different training sets
  - Classifier 1 predicts 0 vs 1,2,…T
  - Classifier 2 predicts 0,1 vs 2,3,…T
  - …
  - Classifier T predicts 0,1,…,T-1 vs T
- Final prediction is combination
  - E.g., sum of predictions

Ordinal Regression Approaches

Approach 3: Optimize pairwise preferences

- Consider instances (x_1,y_1) and (x_2,y_2)
- Label order has $y_1 > y_2$
- Create new training instance
  - $(x', +1)$ where $x' = (x_1 - x_2)$
- Repeat for all instance pairs with label order preference
- Result: new training set!
  - Often represented implicitly
  - Has only positive examples
  - Mispredicting means that a lower ordered instance received higher score than higher order instance.

Rank-Based Measures

Pairwise Preferences not quite right

- Assigns equal penalty for errors no matter where in the ranking

People (mostly) care about top of ranking

- IR community use rank-based measures which capture this property.

Measures

- Binary relevance
  - Precision@K (P@K)
  - Mean Average Precision (MAP)
  - Mean Reciprocal Rank (MRR)
- Multiple levels of relevance
  - Normalized Discounted Cumulative Gain (NDCG)

Optimizing Rank-Based Measures

Let's directly optimize these measures

- As opposed to some proxy (pairwise prefs)
- But the objective function no longer decomposes
  - Pairwise prefs decomposed into each pair
- Objective function flat or discontinuous

Relaxed Upper Bound

- Structural SVMs for hinge loss relaxation
  - e.g., SVM-map [Yue et al., 2007] to optimize MAP directly
- Boosting for exponential loss relaxation
  - e.g., AdaRank [Xu et al., 2007]

Smooth Approximations for Gradient Descent

- LambdaRank [Burges et al., 2006]
- SoftRank GP [Snelson & Guiver, 2007]

Outline

Classic IR systems

Relevance feedback and query expansion

Evaluation metrics

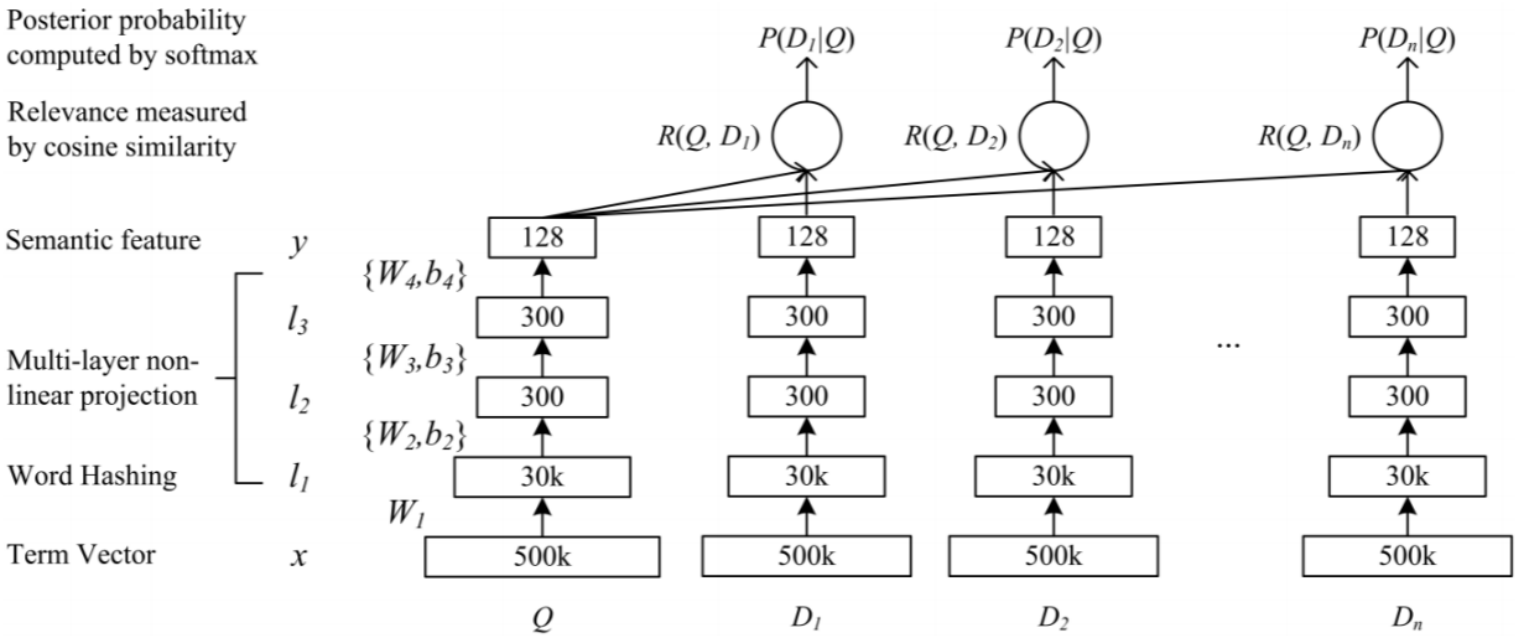
Learning to rank

NN Ranking models

Deep Structured Semantic Model (DSSM)

Used bag-of-words term vector + word hashing as the word representation

Loss function: $L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q)$



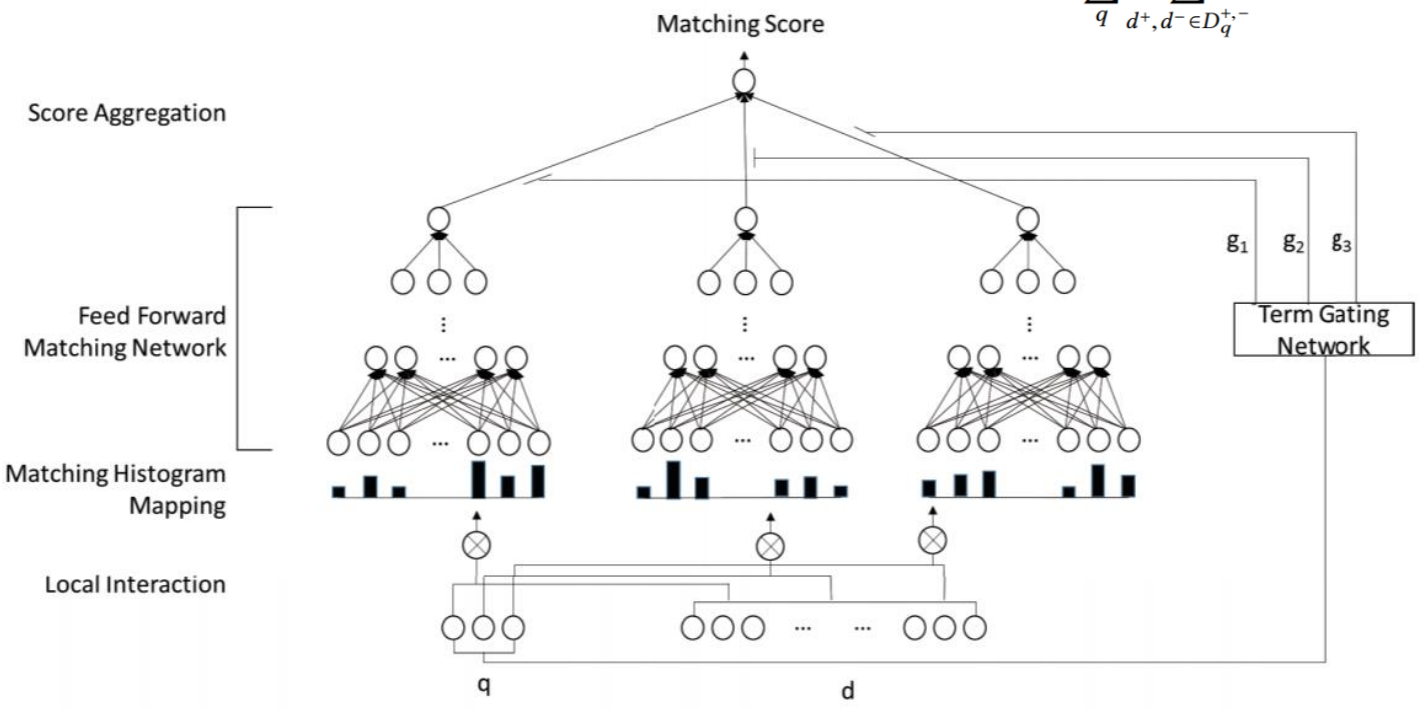
Deep Relevance Matching Model (DRMM)

Word2vec for matching query terms to document terms

Histogram Pooling to `count' matches of different qualities

Pairwise learning-to-rank loss function:

$$l = \sum_{q} \sum_{d^+, d^- \in D_q^{+,-}} \max(0, 1 - f(q, d^+) + f(q, d^-))$$

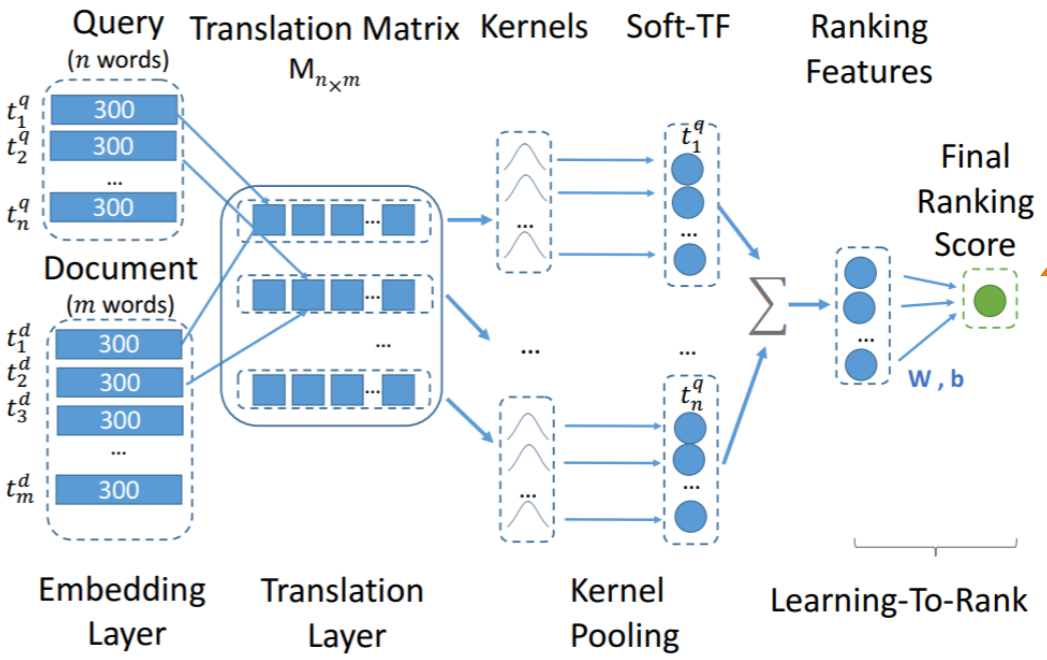


Kernel based Neural Ranking Model (K-NRM)

DRMM uses word2vec, which may not be the "right" embeddings:

- Query: "Tokyo hotels" Documents: "Toyo motels" "Hotels in London"

Learn a word-similarity metric tailored for matching query and document in ranking



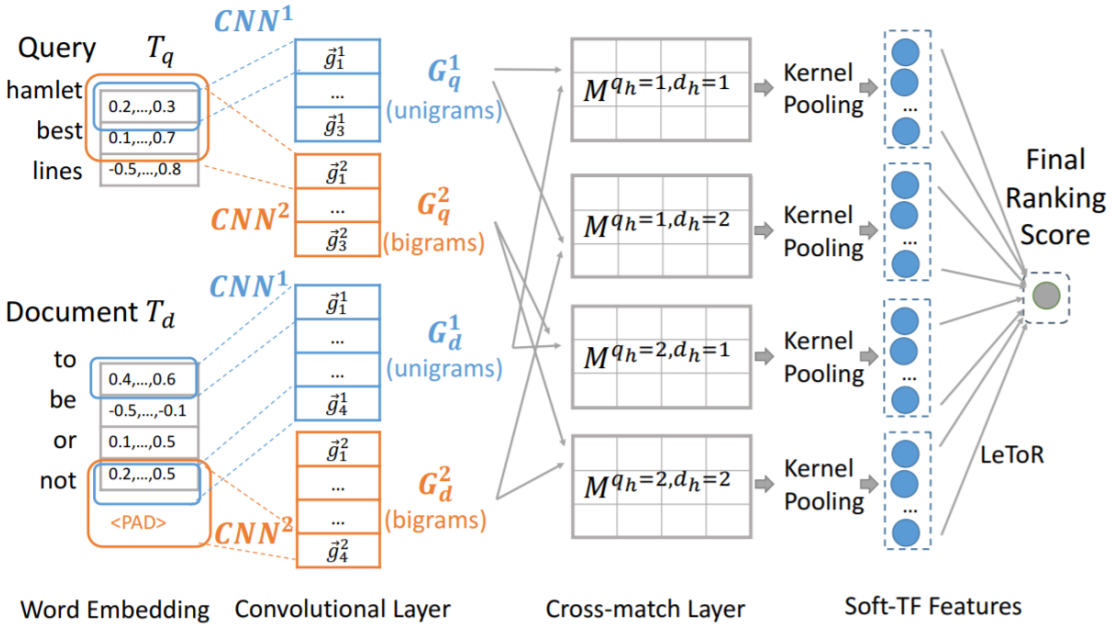
The learning-to-rank layer combines the soft-TF ranking features Φ(M) into a ranking score:

$$f(q,d) = \tanh\left(w_r^T \Phi(\mathcal{M}) + b_r\right)$$

Convolutional Kernel-based Neural Ranking Model (Conv-KNRM)

Queries and documents often match at n-gram level

- Query: “Atypical squamous cells” Document: “to prevent cervical cancel…”
- Traditional IR: exact matching n-grams (vocabulary mismatch)



- Convolution: Compose n-gram embeddings
- Cross-matching: soft match n-grams of different lengths (e.g., query tri-gram to doc bi-gram)
- Kernel pooling: extract multi-level soft-match features (e.g., exact match, strong match, weak match, …)