CS3245

Information Retrieval

11

Lecture 11: Probabilistic IR

Live Q&A
https://pollev.com/jin

Last Time

- Relevance Feedback
  - Documents
- Query Expansion
  - Terms
- XML Retrieval
  - Lexicalized Subtrees
  - Context Resemblance
- XML Evaluation
  - Content and Structure
  - Partial Relevance

Today

Chapter 11

1. Probabilistic Approach to Retrieval

Chapter 12

1. Language Models for IR

Probabilistic Approach to Retrieval

- There are a lot of **uncertainties** in IR…
  - An IR system has an **uncertain** understanding of a user information need (represented as a query) and a collection of documents.
  - It must make an **uncertain** guess of whether a document satisfies the query and perform ranking.
- Probability theory provides a principled foundation for such **reasoning under uncertainty**
  - Probabilistic models exploit this foundation to estimate how likely it is that a document is relevant to a query

Probabilistic IR Models at a Glance

1. Classical probabilistic retrieval model
   - How likely the document **is relevant to** a given query?
   - Widely used and robust

2. Language model approach to IR
   - How likely the document **generates** a given query?
   - More recent and competitive

Probabilistic methods are one of the oldest but also one of the currently hottest topics in IR

Basic Probability Theory

- For events *A* and *B*
  - **Joint probability** *P(A, B)* of both events occurring.
  - **Conditional probability** *P(A|B)* of event *A* occurring given that event *B* has occurred.
- Chain rule gives fundamental relationship between joint and conditional probabilities:

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

- Odds of an event positively correlated to its probability

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

THE CLASSIC APPROACHES

Probabilistic Ranking

- Assume **binary** notion of **relevance**: $R_{d,q}$ is a binary random variable, such that
  - $R_{d,q} = 1$ if document *d* is relevant to *q*
  - $R_{d,q} = 0$ *otherwise*
- Probabilistic ranking orders documents decreasingly by their estimated probability of relevance to the query: $P\ (R = 1 \mid d, q)$
  - Example:
    - $P\ (R_{d1,q} = 1 \mid d_1, q) = 0.7$, $P\ (R_{d2,q} = 1 \mid d_2, q) = 0.5$
    - $d_1 > d_2$

Probability Ranking Principle (PRP)

- PRP in brief
  - If the retrieved documents (w.r.t. a query) are ranked decreasingly on their probability of relevance, then the effectiveness of the system will be the best that is obtainable

- PRP in full
  - If [the IR] system's response to each [query] is a ranking of the documents [...] in order of decreasing probability of relevance to the [query], **where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose**, the overall effectiveness of the system to its user will be the best **that is obtainable on the basis of those data**

Binary Independence Model (BIM)

- Traditionally used with the PRP

Assumptions:

- Binary (equivalent to Boolean): documents and queries represented as binary term incidence vectors
  - E.g., document *d* represented by vector $\vec{x} = (x_1, \ldots, x_m)$, where x_t = 1 if term *t* occurs in *d* and x_t = *0* otherwise
- Independence: no association between terms (not true, but works in practice – naïve assumption)

Binary Independence Model

$P(R|d,q)$ is modeled using term incidence vectors as $P(R|\vec{x},\vec{q})$

$$P(R=1|d,q) = P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

$$P(R=1|\vec{x},\vec{q}) = \frac{P(R=1,\vec{x},\vec{q})}{P(\vec{x},\vec{q})}$$
$$= \frac{P(\vec{x}|R=1,\vec{q})P(R=1,\vec{q})}{P(\vec{x},\vec{q})}$$
$$= \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})P(\vec{q})}{P(\vec{x},\vec{q})}$$
$$= \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

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

Binary Independence Model

Same equation as on previous slide

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})}$$

- $P(\vec{x}|R=1,\vec{q})$: The probability that if a relevant document is retrieved for a query q, that document's representation is $\vec{x}$
- $P(R=1|\vec{q})$: The prior probability of retrieving a relevant document for a query *q*
- $P(\vec{x}|\vec{q})$: The probabity that given a query q, there exists a document whose representation is $\vec{x}$

Deriving a Ranking Function for Query Terms

- To ignore the common denominator and drop some terms, we rank the documents by their **odds** of relevance instead.

$$O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1 - P(A)} \qquad P(R=1|\vec{x},\vec{q}) + P(R=0|\vec{x},\vec{q}) = 1$$

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

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

Cancel out each other

Constant for a given query and can be dropped.

Deriving a Ranking Function for Query Terms

- It is at this point that we make use of the (Naïve Bayes) **conditional independence assumption** that there are no associations between terms:

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} = \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}$$

M is the number of dimensions.

- E.g., If x = {1, 0, 0, 1, 1}, the number of dimensions is 5.
  - We multiply the individual probabilities of 5 (independent) terms.

Deriving a Ranking Function for Query Terms

- Since each x_t is either present (1) or absent (0), we can separate the terms to give:

$$\prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} = \prod_{t:x_t=1} \frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})}$$

- E.g., x = {1, 0, 0, 1, 1} → $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 1$
  - x_1, x_4 and x_5 will be in the first product
  - x_2 and x_3 will be in the second product.

Deriving a Ranking Function for Query Terms

- Let $p_t = P(x_t = 1|R = 1, \vec{q})$ be the probability of a term **appearing** in a **relevant** document
- Let $u_t = P(x_t = 1|R = 0, \vec{q})$ be the probability of a term **appearing** in a **non-relevant** document

	document	relevant ($R = 1$)	nonrelevant ($R = 0$)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

$$\prod_{t:x_t=1} \frac{P(x_t = 1|R = 1, \vec{q})}{P(x_t = 1|R = 0, \vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t = 0|R = 1, \vec{q})}{P(x_t = 0|R = 0, \vec{q})} = \prod_{t:x_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0} \frac{1 - p_t}{1 - u_t}$$

Deriving a Ranking Function for Query Terms

- Additional simplifying assumption: terms not occurring in the query do not matter.

- Now we need only to consider terms in the products that appear in the query:

$$\prod_{t:x_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0} \frac{1-p_t}{1-u_t} = \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}$$

Over query terms found in the document

Over query terms NOT found in the document

- E.g., if x = {1, 0, 0, 1, 1} and q = {1, 0, 1, 0, 0}
  - Only x_1 and x_3 are considered.
  - x_1 is in the first product and x_3 is in the second.

Deriving a Ranking Function for Query Terms

- We can include the query terms found in the document into the right product and divide through by them in the left product.

$$\prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \prod_{t:x_t=q_t=1} \frac{1-u_t}{1-p_t} \prod_{t:x_t=q_t=1} \frac{1-p_t}{1-u_t} \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}$$

- The formula is then:

$$\prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t:q_t=1} \frac{1-p_t}{1-u_t}$$

Constant for a given query and can be dropped.

Deriving a Ranking Function for Query Terms

- We take the log of the product and call it **Retrieval Status Value** (RSV)

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

- RSV is basically a sum of c_t for each term where

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

- Therefore, we compute and sum c_t to get the score for each document and rank accordingly.

Probability Estimates in **Theory**

- For each term *t* in a query, estimate c_t as follows:
  - s is the number of relevant documents containing t
  - S is the total number of relevant documents
  - df_t is the document frequency of t
  - N is the collection size

	documents	relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

$$p_t = P(x_t = 1 | R = 1, \vec{q}) = s/S$$

$$u_t = P(x_t = 1 | R = 0, \vec{q}) = (\mathrm{df}_t - s)/(N - S)$$

$$c_t = \log \frac{p_t(1 - u_t)}{u_t(1 - p_t)} = \log \frac{s/(S - s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S - s))}$$

Probability Estimates in Practice

- An alternative view:

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} + \log \frac{1-u_t}{u_t}$$

- Assuming that relevant documents are a very small percentage of the collection:

$$u_t = (df_t - s)/(N - S) = df_t/N$$

$$\log\left[\frac{1-u_t}{u_t}\right] = \log\left[\frac{N-df_t}{df_t}\right] \cong \log[\frac{N}{df_t}]$$ ← This is basically IDF!

- But the above approximation cannot easily be extended to the statistics of relevant documents (p_t).

Probability Estimates in Practice

- Statistics of relevant documents (p_t) can be estimated in various ways:
  1. Use the frequency of term occurrence in known relevant documents (if any).
  2. Set as a constant, e.g., assume that p_t is constant over all terms x_t in the query and that $p_t = 0.5$ → RSV is basically IDF in this case.

Okapi BM25: A Nonbinary Model

The simplest score for document *d* is just *idf* weighting of the query terms present in the document:

$$RSV_d = \sum_{t \in q} \log \frac{N}{\mathrm{df}_t}$$

Improve this formula by factoring in the term frequency and document length:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d / L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

- tf_{td}: term frequency in the document *d*
- L_d (L_{ave}): length of document *d* (average document length in the whole collection)
- k_1: tuning parameter controlling the document term frequency scaling
- b: tuning parameter controlling the scaling by document length

Okapi BM25: A Nonbinary Model

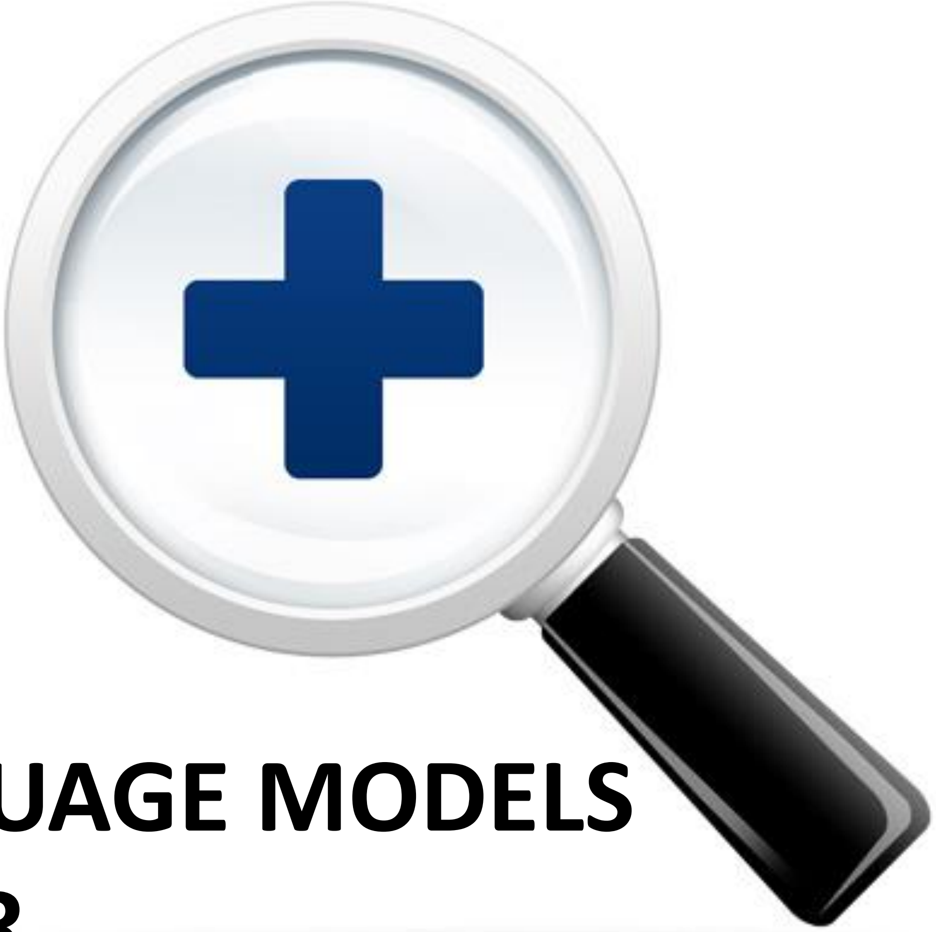
- If the query is long, we might also use similar weighting for query terms

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d/L_{ave})) + \mathrm{tf}_{td}} \cdot \frac{(k_3 + 1)\mathrm{tf}_{tq}}{k_3 + \mathrm{tf}_{tq}}$$

- tf_{tq}: term frequency in the query q
- k_3: tuning parameter controlling the query term frequency scaling
- No length normalization of queries
  (because retrieval is being done with respect to a single fixed query)
- The above tuning parameters should be set by optimization on a development test collection. Experiments have shown reasonable values for k_1 and k_3 as values between 1.2 and 2 and $b = 0.75$

An Appraisal of Probabilistic Models

- The difference between Vector Space and Probabilistic IR is not that great
  - In either case you build an information retrieval scheme in the exact same way.
  - Difference: for probabilistic IR, in the end, your score queries not by cosine similarity and *tf.idf* in a vector space, but by a slightly different formula motivated by probability theory



LANGUAGE MODELS FOR IR

Language Models for IR

- **Book A** by Shakespeare
- **Book B** by J.K. Rowling

- *Which book is more likely to be relevant to the following queries?*

  1. A nice normal day
  2. Wherefore art thou

Language Models for IR

- Give a query *q*, rank documents based on *P(d|q),* which is the probability of *d* being relevant given q.

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

- *P(q|d)* is the probability of *q* being relevant given d (= being generated by the language model of d).
- *P(d)* is the prior of d being relevant – often treated as the same for all *d*
  - But we can give a prior to "high-quality" documents, e.g., those with high static quality score g(d) (cf. Section 7.14).
- *P(q)* is the same for all documents, so ignore

How to compute *P*(*q*|*d*)?

- Let's take a sentence from each of these artists and build two language models:

$M_{d\text{-}lg}$

… I want your love and I want your revenge // …

I	2 (.22)	love	1 (.11)
want	2 (.22)	and	1 (.11)
your	2 (.22)	revenge	1 (.11)

$M_{d\text{-}es}$

… Girl you know I want your love

Girl	1 (.14)	want	1 (.14)
you	1 (.14)	your	1 (.14)
know	1 (.14)	love	1 (.14)
I	1 (.14)		

How to compute *P*(*q*|*d*)?

- q: want to want love

Prob (LadyGaga) = P (want) * P(to) * P (want) * P (love)

I	2 (.22)	love	1 (.11)
want	2 (.22)	and	1 (.11)
your	2 (.22)	revenge	1 (.11)

$M_{d\text{-}lg}$

Prob (LadyGaga) = P (q | $M_{d\text{-}lg}$)
= P (want to want love | $M_{d\text{-}lg}$)
= P (want | $M_{d\text{-}lg}$) * P (to | $M_{d\text{-}lg}$) * P (want | $M_{d\text{-}lg}$) * P (love | $M_{d\text{-}lg}$)

Girl	1 (.14)	want	1 (.14)
you	1 (.14)	your	1 (.14)
know	1 (.14)	love	1 (.14)
I	1 (.14)		

$M_{d\text{-}es}$

$$P(q|M_d) = P(\langle t_1, \ldots, t_{|q|}\rangle | M_d) = \prod_{1 \le k \le |q|} P(t_k|M_d)$$

(|q|: length q; t_k: the token occurring at position *k* in *q*)

How to compute *P*(*q*|*d*)?

I	2 (.22)	love	1 (.11)
want	2 (.22)	and	1 (.11)
your	2 (.22)	revenge	1 (.11)

$M_{d\text{-}lg}$

- How to estimate $P(t|M_d)$?
  - e.g., P (want | $M_{d\text{-}as}$)
- Start with maximum likelihood estimates:

$$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

(|*d*|: length of *d*; $tf_{t,d}$: # occurrences of *t* in *d*)

- But a single *t* with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
  - E.g., P (love | $M_{d\text{-}as}$) = 0 and hence P (q | $M_{d\text{-}as}$) = 0. That's bad.
- We need to smooth the estimates to avoid zeros.

Add 1 Smoothing

- Idea: add 1 count to all entries in the LM, including those that are not seen

I	2 (.22)	revenge	1 (.11)
want	2 (.22)	**Girl**	**0 (0)**
your	2 (.22)	**you**	**0 (0)**
love	1 (.11)	**know**	**0 (0)**
and	1 (.11)		

I	3 (.17)	revenge	2 (.11)
want	3 (.17)	Girl	1 (.06)
your	3 (.17)	you	1 (.06)
love	2 (.11)	know	1 (.06)
and	2 (.11)		

Add 1 count to all entries and recompute the probabilities

Girl	1 (.14)	your	1 (.14)
you	1 (.14)	love	1 (.14)
know	1 (.14)	**and**	**0 (0)**
I	1 (.14)	**revenge**	**0 (0)**
want	1 (.14)		

Girl	2 (.13)	your	2 (.13)
you	2 (.13)	love	2 (.13)
know	2 (.13)	and	1 (.06)
I	2 (.13)	revenge	1 (.06)
want	2 (.13)		

Smoothing via the collection model

- A non-occurring term is **possible** (even though it didn't occur), … **but no more likely than the chance in the collection**

$$\hat{P}(t|M_c) = \frac{cf_t}{T}$$

M_c: the collection model; cf_t: the number of occurrences of t in the collection; $T = \sum_t cf_t$: the total number token in the collection.

  - E.g., Collection = I want your love and I want your revenge … Girl you know I want your love
  - P (love | M_c) = 2 / 16

- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

Mixture model

- $P(t|d) = \lambda P(t|M_d) + (1 - \lambda)P(t|M_c)$

- Mixes the probability from the document with the general collection frequency of the word.
  - High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words.
  - Low value of λ: more disjunctive, suitable for long queries

- Correctly setting λ is very important for good performance

Mixture model: Summary

- To sum up…

$$P(q|d) = P(q|M_d) = P(\langle t_1, \ldots, t_{|q|}\rangle | M_d)$$
$$= \prod_{1 \le k \le |q|} (\lambda P(t_k|M_d) + (1-\lambda) P(t_k|M_c))$$

- This is Language modelling + Smoothing via the collection model.

Blanks on slides, you may want to fill in

Exercise

Collection: d_1 and d_2

- d_1: Jackson was one of the most talented entertainers of all time
- d_2: Michael Jackson anointed himself King of Pop

Query *q*: Michael Jackson

Use mixture model with $\lambda = 1/2$

- $P(q|d_1)$
- $P(q|d_2)$
- Ranking:

Vector space ($tf.idf$) vs. LM

	precision			significant?
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.4	0.2093	0.2572	+22.9	*
0.6	0.1024	0.1405	+37.1	*
0.8	0.0160	0.0432	+169.6	*
1.0	0.0028	0.0050	+76.9	
11-point average	0.1868	0.2233	+19.6	*

- The language modeling approach always does better in these experiments . . . but note that where the approach shows significant gains is at higher levels of recall.

Summary

- Probabilistically grounded approach to IR
  - Probability Ranking Principle
  - Models: BIM, OKAPI BM25
- Language Models for IR

Resources:

- Chapters 11 and 12 of IIR
- Ponte and Croft's 1998 SIGIR paper (one of the first on LMs in IR)
- Lemur toolkit (good support for LMs in IR, http://www.lemurproject.org/)