PROBABILISTIC & LANGUAGE MODELS

Chapter 7

Outline

- Probability ranking principle
- Classical probabilistic model
  - Binary Independence Model
  - BM25
  - feedback methods
- Language Models

Probability Ranking Principle

- Robertson (1977)
  - “If a reference retrieval system’s response to each request is a ***ranking*** of the documents in the collection in order of decreasing ***probability of relevance*** to the user who submitted the request,
  - 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.”
- Basis for most probabilistic approaches to IR

Let's dissect the PR

- rank documents … by probability of relevance
  - P (relevant | document) or P(R|D)
- estimated as accurately as possible
  - $P_{est}(R \mid D) \rightarrow P_{true} (R \mid D)$ in some way
- based on whatever data is available to system
  - P_{est} (R | D, query, context, user profile, …)
- best possible accuracy one can achieve with that data
  - recipe for a perfect IR system: just need P_{est} (R| …)
  - strong stuff, can this really be true?

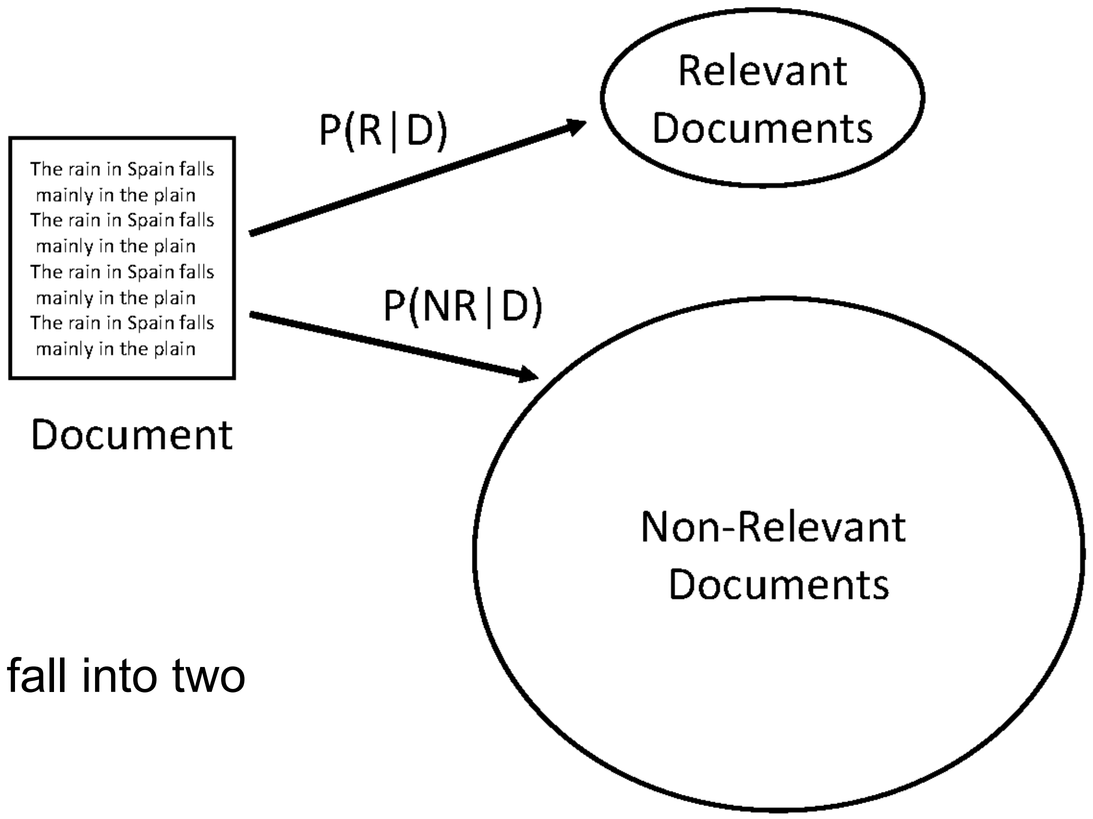
Probability of relevance

- What is: P_{true} (relevant | doc, qry, user, context) ?
  - isn't relevance just the user's opinion?
  - user decides relevant or not, what's the “probability” thing?

Take away … this is a difficult problem!

- “user” does not mean the human being
  - doc, qry, user, context … representations
    - parts of the real thing that are available to the system
  - typical case: P_{true} (R|D , query)
    - query: 2-3 keywords, user profile unknown, context not available
    - whether document is relevant is uncertain
      - depends on the factors which are not available to our system
    - think of P_{true} (R|D ,qry) as proportion of all unseen users/contexts/... for which the document would have been judged relevant

IR as Classification


The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain
Document
P(R|D)
P(NR|D)
Relevant Documents
Non-Relevant Documents

For a given query, documents fall into two classes

− relevant (R) and non-relevant (NR)

− compute P(R|D) and P(NR|D)
  - retrieve if P(R|D) > P(NR|D)

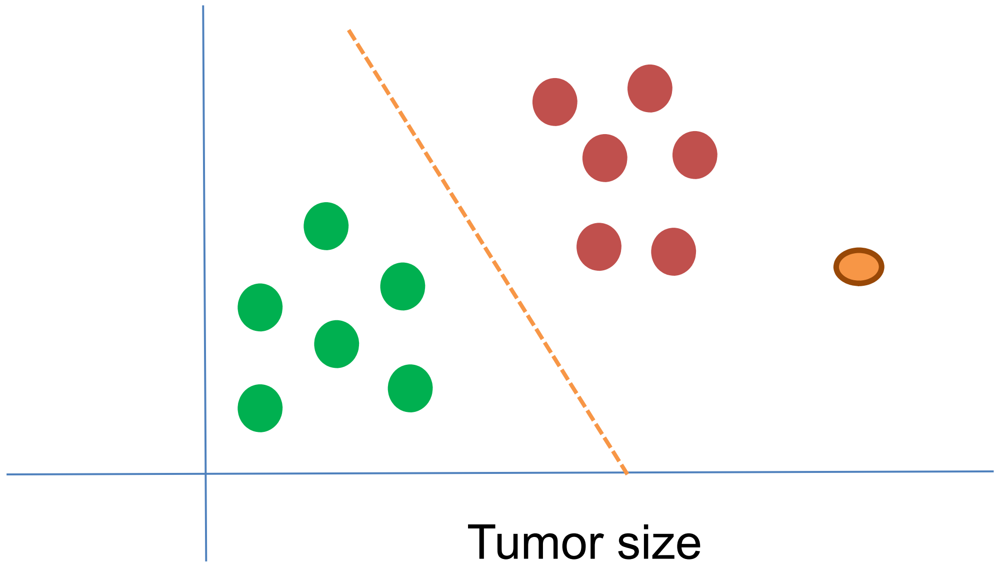
How do we compute these probabilities?

Generative vs. Discriminative Classifiers

Training classifiers involves estimating f: X → Y, or P(Y|X)

Discriminative classifiers (also called ‘informative’ by Rubinstein&Hastie):

1. Assume some functional form for P(Y|X)
2. Estimate parameters of P(Y|X) directly from training data

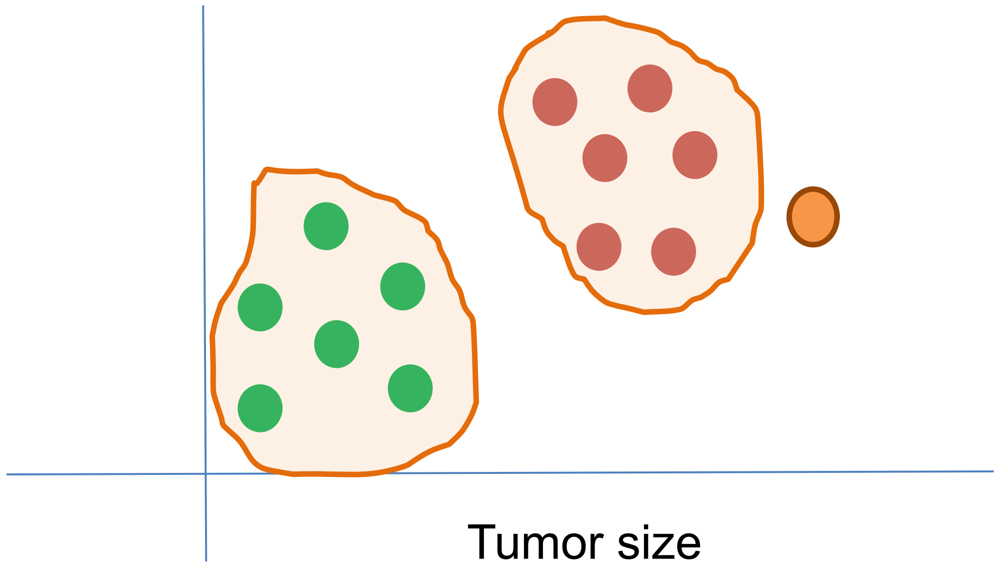


Generative vs. Discriminative Classifiers

Training classifiers involves estimating f: X → Y, or P(Y|X)

Generative classifiers

1. Assume some functional form for P(X|Y), P(X)
2. Estimate parameters of P(X|Y), P(X) directly from training data
3. Model predictions based on $P(Y|X= x_i)$



Bayes Classifier

- Bayes Decision Rule
  - A document *D* is relevant if *P*(*R*|*D*) > *P*(*NR*|*D*)

- Estimating probabilities
  - use Bayes Rule
  - classify a document as relevant if P(R|D) > P(NR|D).

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

    - *likelihood ratio*

    - *In practice, we rank documents by*

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$$

Now the question is how to compute P(D|R) and P(D|NR).

Maximum A Posteriori (MAP) Classifier

- Given the data feature vector ***x***, we would like to find the class with the largest probability:
$$\hat{C} \triangleq \arg\max_{C} p(C|\boldsymbol{x})$$

- To accomplish this, we iterate through all possible classes C_1, C_2, …., C_K and evaluate the quantity $p(C_i|\boldsymbol{x})$, and pick the class that has the largest probability.

MAP Classifier: Bayes Rule (1/4)

- How do we compute $p(C_i|\boldsymbol{x})$? Apply Bayes' Rule!

$$p(A|B)p(B) = p(B|A)p(A) \Rightarrow p(A|B) = \frac{\overset{\text{likelihood}}{p(B|A)}\,\overset{\text{prior}}{p(A)}}{\underset{\text{evidence}}{p(B)}}$$

- Applying Bayes' rule to $p(C_i|\boldsymbol{x})$, we obtain:

$$p(C_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_i)p(C_i)}{p(\boldsymbol{x})}$$

Extra Slides

MAP Classifier: Bayes Rule (2/4)

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)}$$

- $p(C_i)$ is called the prior probability of a class
- $p(x|C_i)$ is called the likelihood of the data (what is the probability of observing ***x*** if the class was C_i)
- p(x) is the probability of seeing the data so its simply a normalizing factor (applies to all C_i), so doesn't affect which C_i attains MAP.

Extra Slides

MAP Classifier: Bayes Rule (3/4)

- To compute the normalizing factor, we use:

$$p(x) = \sum_{i=1}^{K} p(x|C_i)p(C_i)$$

So that

$$\sum_{i=1}^{K} p(C_i|x) = 1$$

Hence,

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{\sum_{\ell=1}^{K} p(x|C_\ell)p(C_\ell)}$$

But really, we don’t need to calculate the denominator p(x), so we can simply write it as

$$(p(C_i|x) = p(x|C_i)p(C_i)$$

MAP Classifier: Bayes Rule (4/4)

- Usually, the likelihood $p(\boldsymbol{x}|C_i)$ is difficult to compute because it is N-dimensional (length of feature vector).
- This is because the distribution considers correlations between the features when computing the likelihood.

- We can write $p(\boldsymbol{x}|C_i)$ equivalently as:

$$p(\boldsymbol{x}|C_i) = p(x_1, x_2, \ldots, x_N|C_i)$$

which makes the dependence on individual features explicit.

Naive Bayes Classifier

- So, P(c) is easy to compute…
- What about $P(x_1, x_2, \ldots, x_n \mid c)$?
  - The probability of the class given the features.
- This is hard to compute given that there are many features.
  - $O(|X|^n * |C|)$ parameters
- Also, its hard to generate this unless we have a large training dataset.

Naive Bayes Classifier

P(x1, x2, … , xn | c)

- Bag of Words assumption : order of the words don't matter
- Conditional independence: Assume the feature probability P(xi|c) are independent given the class c.

$$P(x_1, x_2, \cdots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet \ldots \bullet P(x_n \mid c)$$

Obviously this assumption may not always be true, but this simplification allows us to solve the more easily.

Naïve Bayes Classifier: Independence Assumption

- The Naïve Bayes classifier introduces one major assumption regarding the features: independence
- That is, the Naïve Bayes classifier assumes:

$$p(\boldsymbol{x}|C_i) = p(x_1, x_2, \ldots, x_N|C_i) = \prod_{n=1}^{N} p(x_n|C_i)$$

- That is, the complicated likelihood $p(\boldsymbol{x}|C_i)$ can now be factored into a product of N 1-dimensional likelihoods, which are easy to compute.

- It is good to note that independence implies that the features are not correlated (but generally not the other way around---so independence is a stronger assumption).

Binary independence model

- Objective: Need to compute P(D|R) and P(D|NR)
- Assumptions
  - In this model, documents are represented as a *vector of binary features*
    - D = (d_1, d_2, . . . , d_t), where d_i = 1 if term i is present in the document, and 0 otherwise.
  - Also assume term independence (also known as the Naïve Bayes assumption).
- Hence, P(D|R) is computed as the product of the individual term probabilities and similarly for P(D|NR)).

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R)$$

Binary Independence Model

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i}\right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \boxed{\prod_i \frac{1-p_i}{1-s_i}}$$

Document-independent

- p_i is probability that term i occurs in relevant document,
- s_i is probability that term i occurs in non-relevant document

Binary Independence Model

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_i \frac{1-p_i}{1-s_i}$$

Second term is the same for all documents, so we can drop it for purpose of ranking.

- Scoring function is

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

Binary Independence Model

- Scoring function is $$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

- Query provides information about relevant documents
- If we assume p_i constant for terms in query, s_i approximated by entire collection, get *idf*-like weight

$$\log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}$$

N number of documents in collection

N_i is the number of documents that contain term i

Similar to idf weight, where there is no tf component since documents have binary features

Contingency Table

What if we had some kind of relevance feedback from users?

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - r_i$
Total	R	$N - R$	N

n_i:	# of docs that contain term i
N:	total # of docs in collection
r_i:	# of relevant docs containing term i
R:	# relevant docs

Contingency Table

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - r_i$
Total	R	$N - R$	N

$$p_i = \frac{r_i}{R}$$

The number of relevant documents that contain a term divided by the number of relevant documents)

$$s_i = \frac{(n_i - r_i)}{(N - R)}$$

The number of non-relevant documents that contain a term divided by the total number of non-relevant documents)

Binary Independence Model (Cont.)

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

Gives scoring function:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

BM25 Ranking Algorithm

- BM25 is a scoring function based on the binary independence model
  - Stands for Best Match & 25 is a numbering scheme used by its director Robertson.
  - It includes document and query term weights.
  - It has performed very well in TREC experiments, although it does not exactly a formal model.

BM25

- N : total number of documents in the collection
- R : number of relevant documents
- ni : number of documents with term i
- ri : number of relevant documents with term i
- f_i : frequency of term i in the document
- qf_i : frequency of term i in the query
- k_1, k_2, K : parameters whose values are set empirically

$$\underbrace{N, R, n_i, r_i}_{\text{same as before}}$$

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$$

BM25 Example

- Query with two terms, “president” & “ lincoln”, (*qf = 1)*
- No relevance information (*r and R are* zero)
- N = 500,000 documents
- *“president”* occurs in 40,000 documents (n_1 = 40, 000)
- *“lincoln”* occurs in 300 documents (n_2 = 300)
- “president” occurs 15 times in doc (f_1 = 15)
- *“lincoln”* occurs 25 times (f_2 = 25)
- document length is 90% of the average length (*dl/avdl* = .9)
- k_1 = 1.2, b = 0.75, and k_2 = 100
- $K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11$

BM25 Example

$$
\begin{aligned}
BM25(Q,D) \quad &= \\
&\log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\
&\times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\
&+ \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\
&\times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1}
\end{aligned}
$$

Log$_e$() is used

$$
\begin{aligned}
&= \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\
&\quad + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\
&= 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\
&= 5.00 + 15.66 = 20.66
\end{aligned}
$$

BM25 Example

- Effect of term frequencies

Frequency of “president”	Frequency of “lincoln”	BM25 score
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

Language Model

- Language Models (LMs) assign a probability distribution over sequences of words.

- Unigram language model
  - probability distribution over the words in a language
  - generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them

- N-gram language model
  - some applications use bigram and trigram language models where probabilities depend on previous words

- LMs are popular in a variety of applications, such as:
  - Speech recognition, machine translation, and handwriting recognition

Language Modeling Approach

- Probability distribution over strings of text
  - How likely is a given string (observation) in a given “language”
  - For example, consider probability for the following four strings

$p_1 > p_2 > p_3 > p_4$

P1 = P(“a quick brown dog”)

P2 = P(“dog quick a brown”)

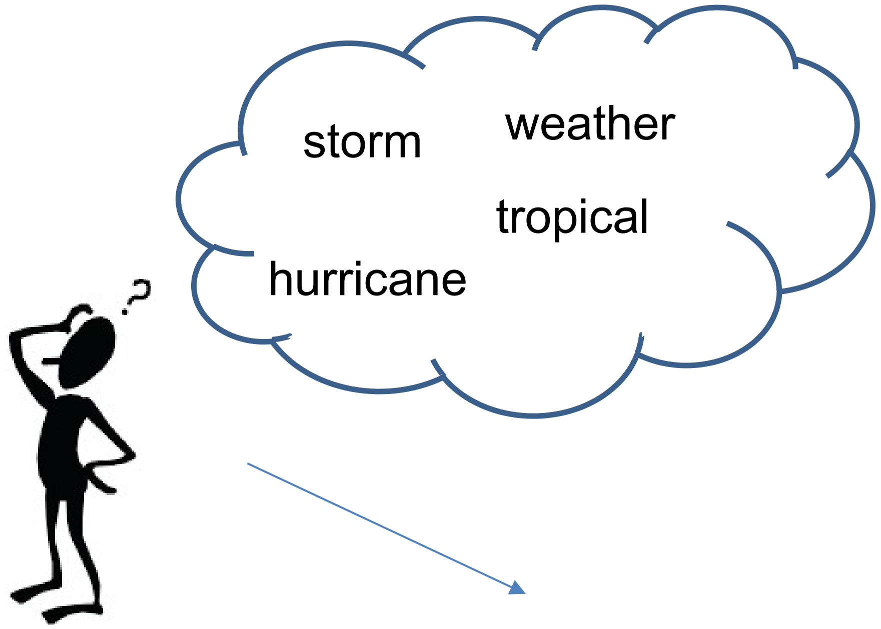
P3 = P(“un chien quick brown”)

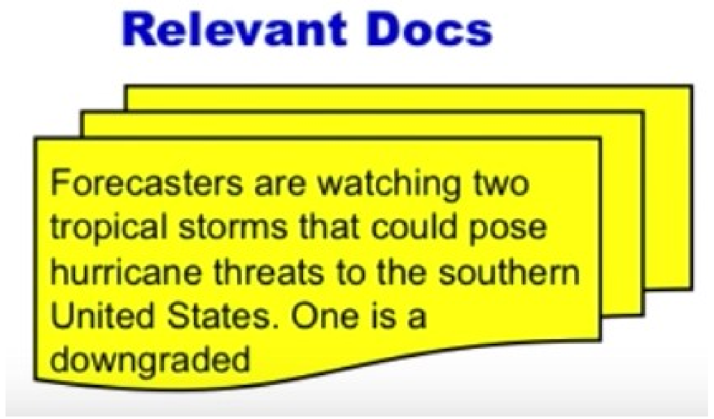
P4 = P(“un chien brun rapide”)

  - In IR, most likely P_1 = P2

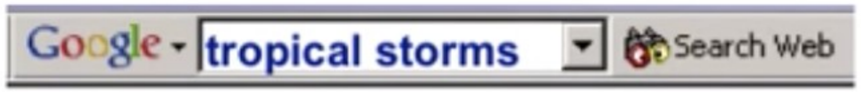
How can we use LMs in IR?

- Use LMs to model the process of query generation:
  - User thinks of some relevant document
  - Picks some keywords to use as the query




Relevant Docs

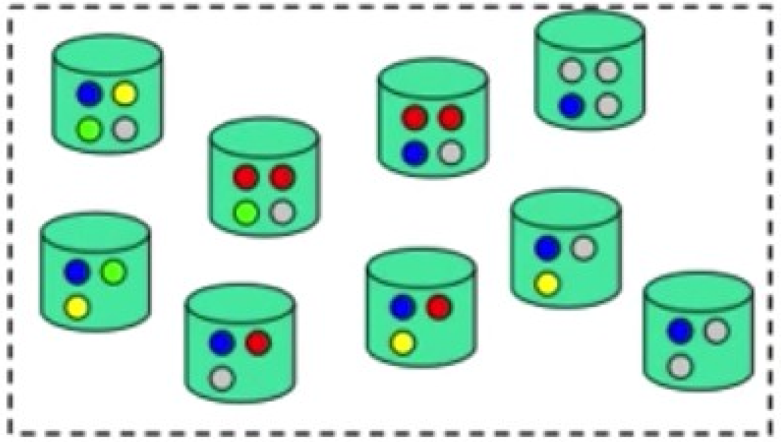
Forecasters are watching two tropical storms that could pose hurricane threats to the southern United States. One is a downgraded

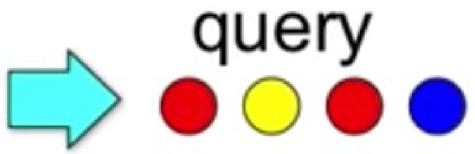


Retrieval with Language Model

- Each document D in the collection defines a “language”
  - All possible sentences the author of D could have written
  - P(S|MD) is the probability that author would write string s
    - Intuition: write a billion variants of D, count how many times we get “s”
    - Language model of what the author of D was trying to say

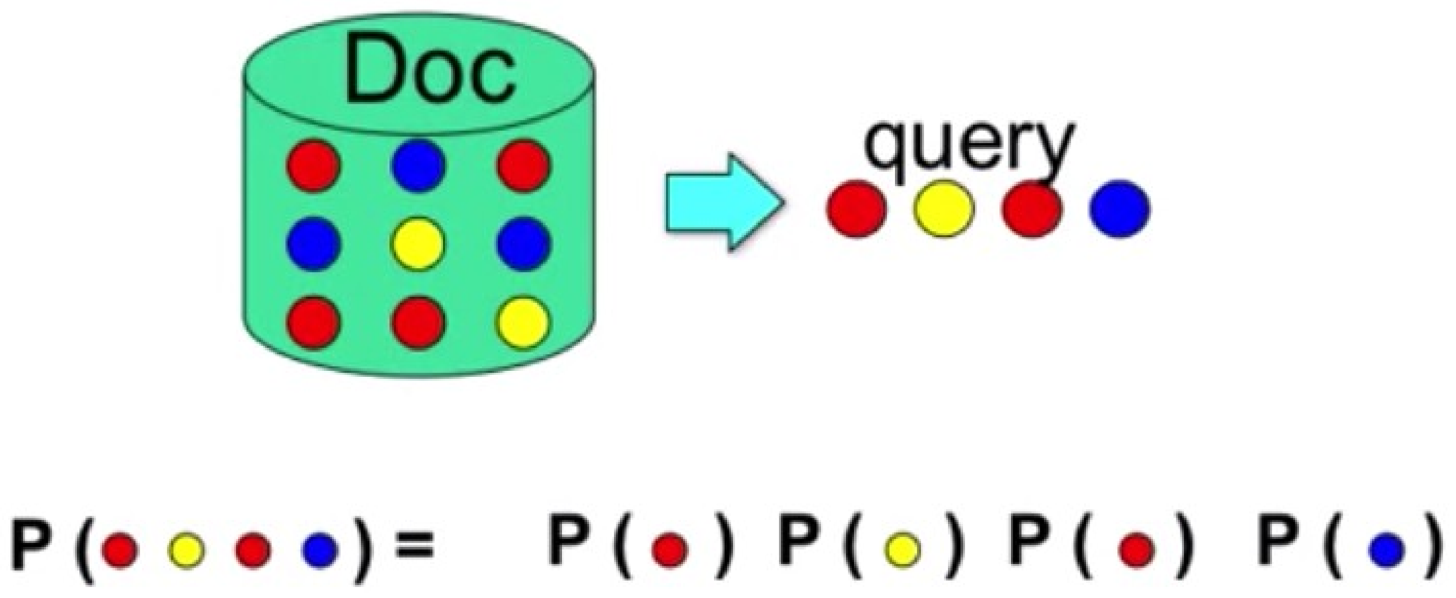
  - Retrieval: rank documents by P(q|MD)
    - Probability that the author would write “q” while creating D





Unigram Language Models

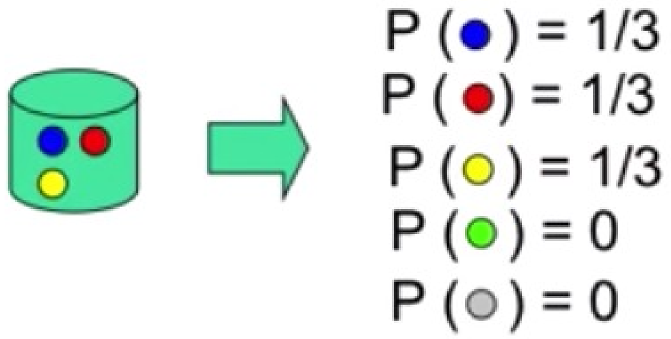
- Words are ”sampled” independently of each other
  - Metaphor: randomly pulling out words from a jar (with replacement)
  - Joint probability decomposes into a product of marginals
  - Estimation of probabilities: simple counting



Estimation of Language Models

- Usually we don’t know the model M
  - But we may have a sample of text representative of that model
  - Estimate a language model from that sample

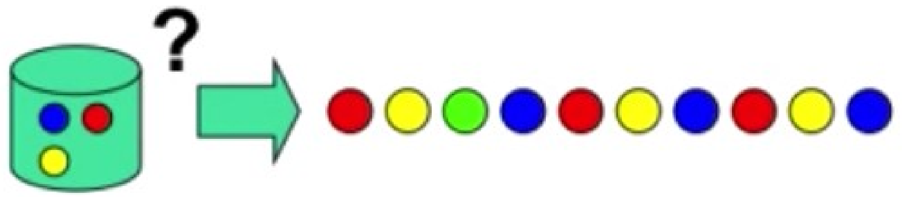
- Maximum likelihood estimator:
  - Count relative frequency of each word



The zero-frequency problem

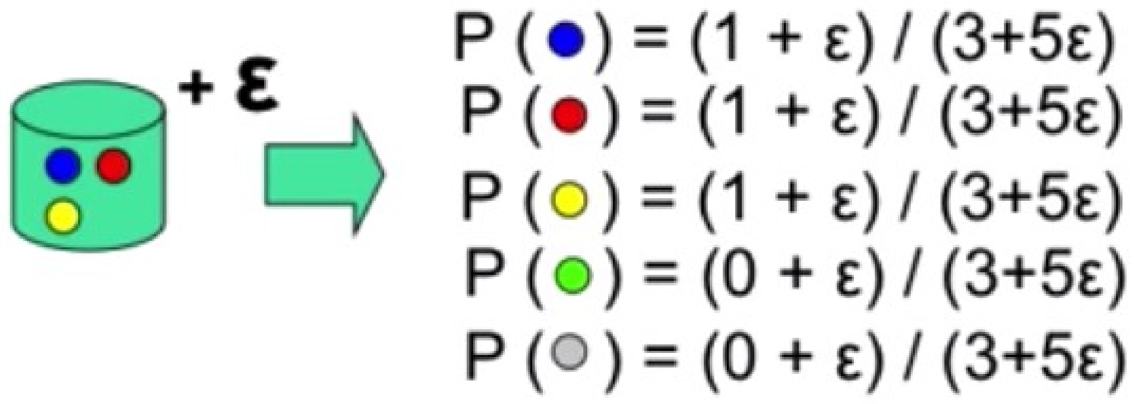
- Suppose some event not in our example
  - Language is sparse, so not every event will be present in our document.
  - Model will assign zero probability to that event (and any set of events involving the unseen event)

- It is incorrect to infer zero probabilities
  - Especially when dealing with incomplete samples



Simple Discounting Methods

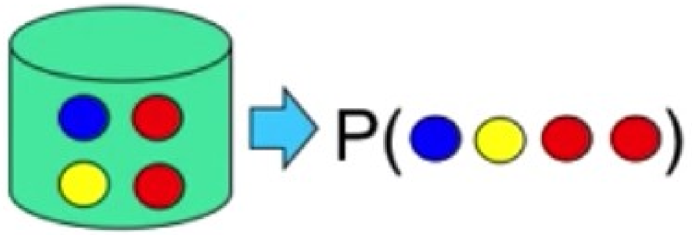
- Laplace correction:
  - Add 1 to every count, normalize
  - Problematic for large vocabularies
- Lindstone correction:
  - Add a small constant epsilon to every count
- Absolute Discounting
  - Subtract a constant epsilon from non-zero values, and re-distribute the probability mass



How to set Epsilon?

- How to set the discounting parameter epsilon

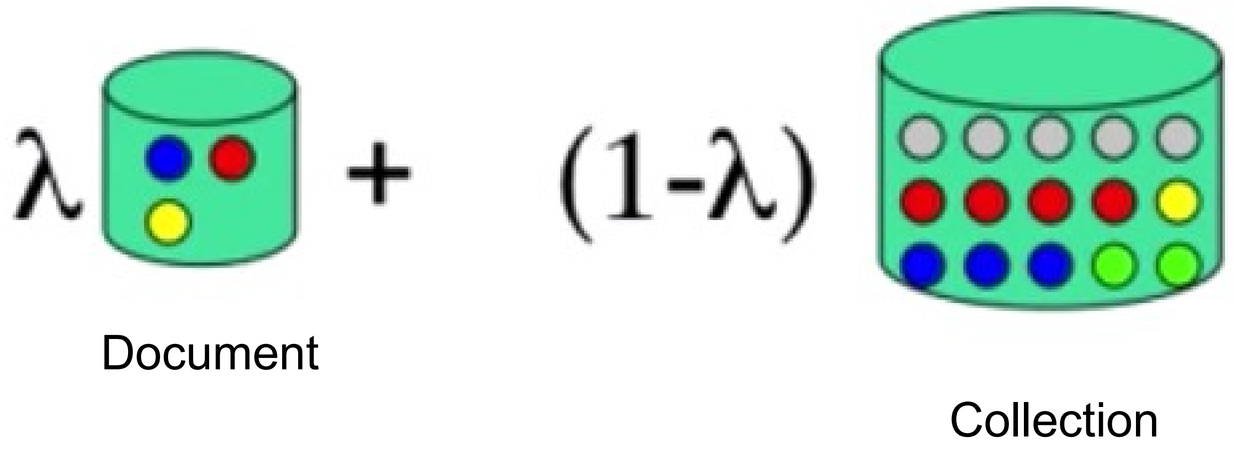
- Leave-one-out discounting
  - Remove some word w, compute P(D | D_{-w})
  - Repeat for every word in the document
  - Iteratively adjust epsilon to maximize P(D|D-w)
    - Intuitively: increase if word w occurs once, decrease if more than once



Step 1 - as is epsilon is zero
Step 2 – drop blue from document, then epsilon need to be updated to a value > 0
Step 3 –drop red from document, since we have two reds, then epsilon will need to be updated again
…

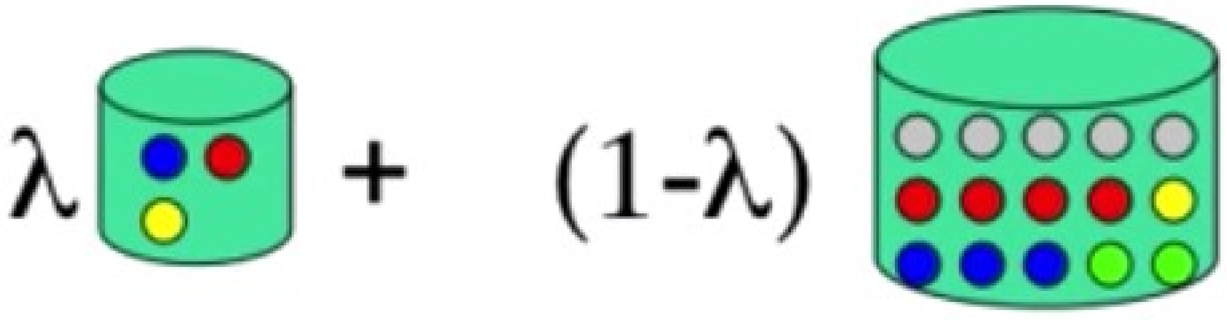
Interpolation Methods

- Problem with all discounting methods:
  - Discounting treats unseen words equally (add or subtract epsilon)
  - Some words are more frequent than others
- Idea – instead of a fixed epsilon for all terms, lets use probability of terms in the collection
  - Plays a similar role as IDF measure



Jelinek-Mercer Smoothing

- Correctly setting lamda is very important
- Start simple
  - Set lambda to be a constant (independent of the document and query)
- Tune
  - Tune to optimize retrieval performance



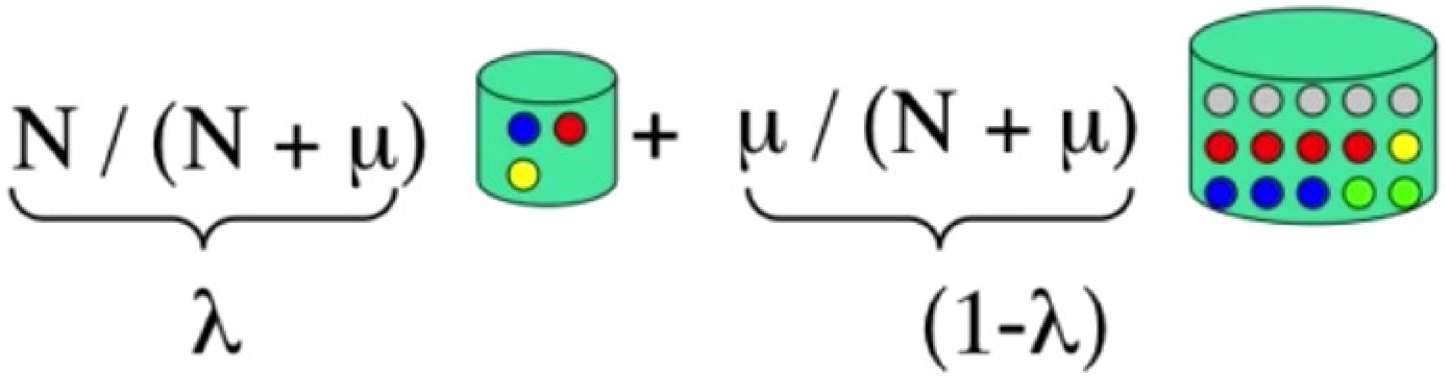
Document

Collection

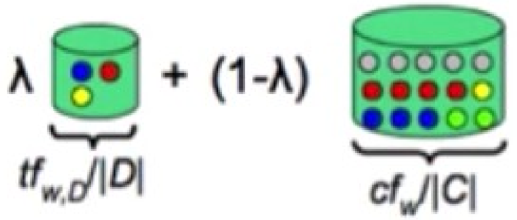
Dirichlet Smoothing

- Problem with Jelinek-Mercer
  - Issue - Using the same smoothing for documents of different length
  - Hence, longer documents will provide better estimates

- Make smoothing depend of sample size
- Formal derivation from Bayesian (Dirichlet) prior on LMs
- Best out-of-the-box choice for short queries
  - Tune u to optimize effectiveness


$\underbrace{N / (N + \mu)}_{\lambda}$
+
$\underbrace{\mu / (N + \mu)}_{(1-\lambda)}$

Role of smoothing as IDF



$$P(Q \mid D) = \prod_{w \in Q} \left(\lambda \frac{tf_{w,D}}{|D|} + (1 - \lambda) \frac{cf_w}{|C|} \right)$$

Probability of drawing a query Q from model D

Rank documents using this probability

Major issues in applying LMs

- What kind of language model should we use?
  - Unigram or higher-order models?
  - Multinomial or multiple-Bernoulli?

- How can we estimate model parameters?
  - Maximum likelihood and zero frequency problem
  - Discounting methods: Laplace, Lindstone, …
  - Interpolation methods: Jelinek-Mercer, Dirichlet prior
  - Leave-one-out method

- Ranking methods
  - Query likelihood, document likelihood, model comparison

Why unigram models?

- Unigram = word independence
  - Best for information retrieval

- What about higher-order models?
  - N-gram models – critical in other fields like speech recognition
  - But don’t really seem to work out well in IR
  - Must produce grammatical sequences of words that match documents and queries, and the probabilities are low