Lecture 6.
Probabilistic Model

BM (Best Matching) 25

Language Models

BM25 (Best Match 25)

BM25 (Best Match 25)

- BM25 was created as the result of a series of experiments on variations of the probabilistic model
- A good term weighting is based on three principles
  - inverse document frequency
  - term frequency
  - document length normalization
- The classic probabilistic model covers only the first of these principles
- This reasoning led to a series of experiments with the Okapi system, which led to the BM25 ranking formula

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{N - n_i + 0.5}{n_i + 0.5}$$

BM1, BM11 and BM15 Formulas

- At first, the Okapi system used the Equation below as ranking formula

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{N - n_i + 0.5}{n_i + 0.5}$$

  which is the equation used in the probabilistic model, when no relevance information is provided

- It was referred to as the BM1 formula (*Best Match 1*)

BM1, BM11 and BM15 Formulas

- The first idea for improving the ranking was to introduce a **term-frequency** factor $\mathcal{F}_{i,j}$ in the BM1 formula
- This factor, after some changes, evolved to become

$$\mathcal{F}_{i,j} = S_1 \times \frac{f_{i,j}}{K_1 + f_{i,j}}$$

  where

  - $f_{i,j}$ is the frequency of term k_i within document d_j
  - K_1 is a constant setup experimentally for each collection
  - S_1 is a scaling constant, normally set to $S_1 = (K_1 + 1)$

- If $K_1 = 0$, this whole factor becomes equal to 1 and bears no effect in the ranking

BM1, BM11 and BM15 Formulas

- The next step was to modify the $\mathcal{F}_{i,j}$ factor by adding **document length normalization** to it, as follows:

$$\mathcal{F}'_{i,j} = S_1 \times \frac{f_{i,j}}{\frac{K_1 \times len(d_j)}{avg\_doclen} + f_{i,j}}$$

  where

  - $len(d_j)$ is the length of document d_j (computed, for instance, as the number of terms in the document)
  - avg_doclen is the average document length for the collection

BM1, BM11 and BM15 Formulas

- Next, a correction factor $G_{j,q}$ dependent on the document and query lengths was added

$$\mathcal{G}_{j,q} = K_2 \times len(q) \times \frac{avg\_doclen - len(d_j)}{avg\_doclen + len(d_j)}$$

where

  - $len(q)$ is the query length (number of terms in the query)
  - K_2 is a constant

BM1, BM11 and BM15 Formulas

- A third additional factor, aimed at taking into account term frequencies within queries, was defined as

$$\mathcal{F}_{i,q} = S_3 \times \frac{f_{i,q}}{K_3 + f_{i,q}}$$

  where

  - $f_{i,q}$ is the frequency of term k_i within query q
  - K_3 is a constant
  - S_3 is an scaling constant related to K_3, normally set to $S_3 = (K_3 + 1)$

BM1, BM11 and BM15 Formulas

- Introduction of these three factors led to various BM (Best Matching) formulas, as follows:

$$sim_{BM1}(d_j, q) \sim \sum_{k_i[q,d_j]} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM15}(d_j, q) \sim \mathcal{G}_{j,q} + \sum_{k_i[q,d_j]} \underset{\text{term frequency}}{\mathcal{F}_{i,j} \times \mathcal{F}_{i,q}} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM11}(d_j, q) \sim \mathcal{G}_{j,q} + \sum_{k_i[q,d_j]} \underset{\text{document length normalization}}{\mathcal{F}'_{i,j} \times \mathcal{F}_{i,q}} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

where $k_i[q, d_j]$ is a short notation for $k_i \in q \wedge k_i \in d_j$

BM1, BM11 and BM15 Formulas

- Experiments using TREC data have shown that BM11 outperforms BM15
- Further, empirical considerations can be used to simplify the previous equations, as follows:
  - Empirical evidence suggests that a best value of K_2 is 0, which eliminates the $G_{j,q}$ factor from these equations
  - Further, good estimates for the scaling constants S_1 and S_3 are $K_1 + 1$ and $K_3 + 1$, respectively
  - Empirical evidence also suggests that making K_3 very large is better. As a result, the $\mathcal{F}_{i,q}$ factor is reduced simply to $f_{i,q}$
  - For short queries, we can assume that $f_{i,q}$ is 1 for all terms

BM1, BM11 and BM15 Formulas

- These considerations lead to simpler equations as follows

$$sim_{BM1}(d_j, q) \sim \sum_{k_i[q,d_j]} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM15}(d_j, q) \sim \sum_{k_i[q,d_j]} \frac{(K_1 + 1) f_{i,j}}{(K_1 + f_{i,j})} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM11}(d_j, q) \sim \sum_{k_i[q,d_j]} \frac{(K_1 + 1) f_{i,j}}{\frac{K_1 \; len(d_j)}{avg\_doclen} + f_{i,j}} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

BM25 Ranking Formula

- BM25: combination of the BM11 and BM15
- The motivation was to combine the BM11 and BM25 term frequency factors as follows

BM15

$$\mathcal{B}_{i,j} = \frac{(K_1 + 1) f_{i,j}}{K_1 \left[(1 - b) + b \frac{len(d_j)}{avg\_doclen} \right] + f_{i,j}}$$

where b is a constant with values in the interval $[0, 1]$

- If $b = 0$, it reduces to the BM15 term frequency factor
- If $b = 1$, it reduces to the BM11 term frequency factor
- For values of b between 0 and 1, the equation provides a combination of BM11 with BM15

BM25 Ranking Formula

- The ranking equation for the BM25 model can then be written as

$$sim_{BM25}(d_j, q) \sim \sum_{k_i[q,d_j]} \mathcal{B}_{i,j} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

  where K_1 and b are empirical constants

  - $K_1 = 1$ works well with real collections
  - b should be kept closer to 1 to emphasize the document length normalization effect present in the BM11 formula
  - For instance, $b = 0.75$ is a reasonable assumption
  - Constants values can be fine tunned for particular collections through proper experimentation

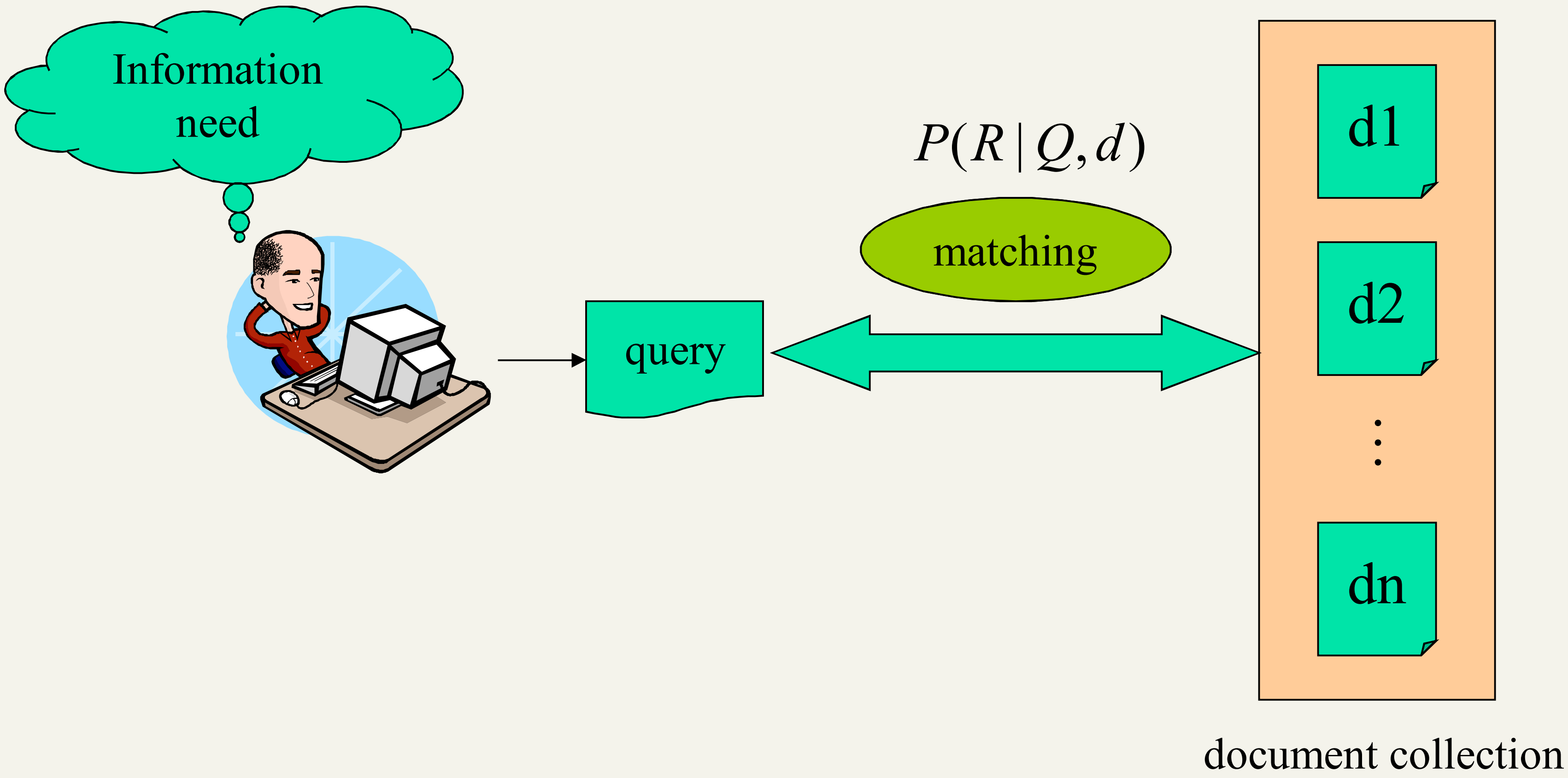
BM25 Ranking Formula

- Unlike the probabilistic model, the BM25 formula can be computed without relevance information
- There is consensus that BM25 outperforms the classic vector model for general collections
- Thus, it has been used as a baseline for evaluating new ranking functions, in substitution to the classic vector model

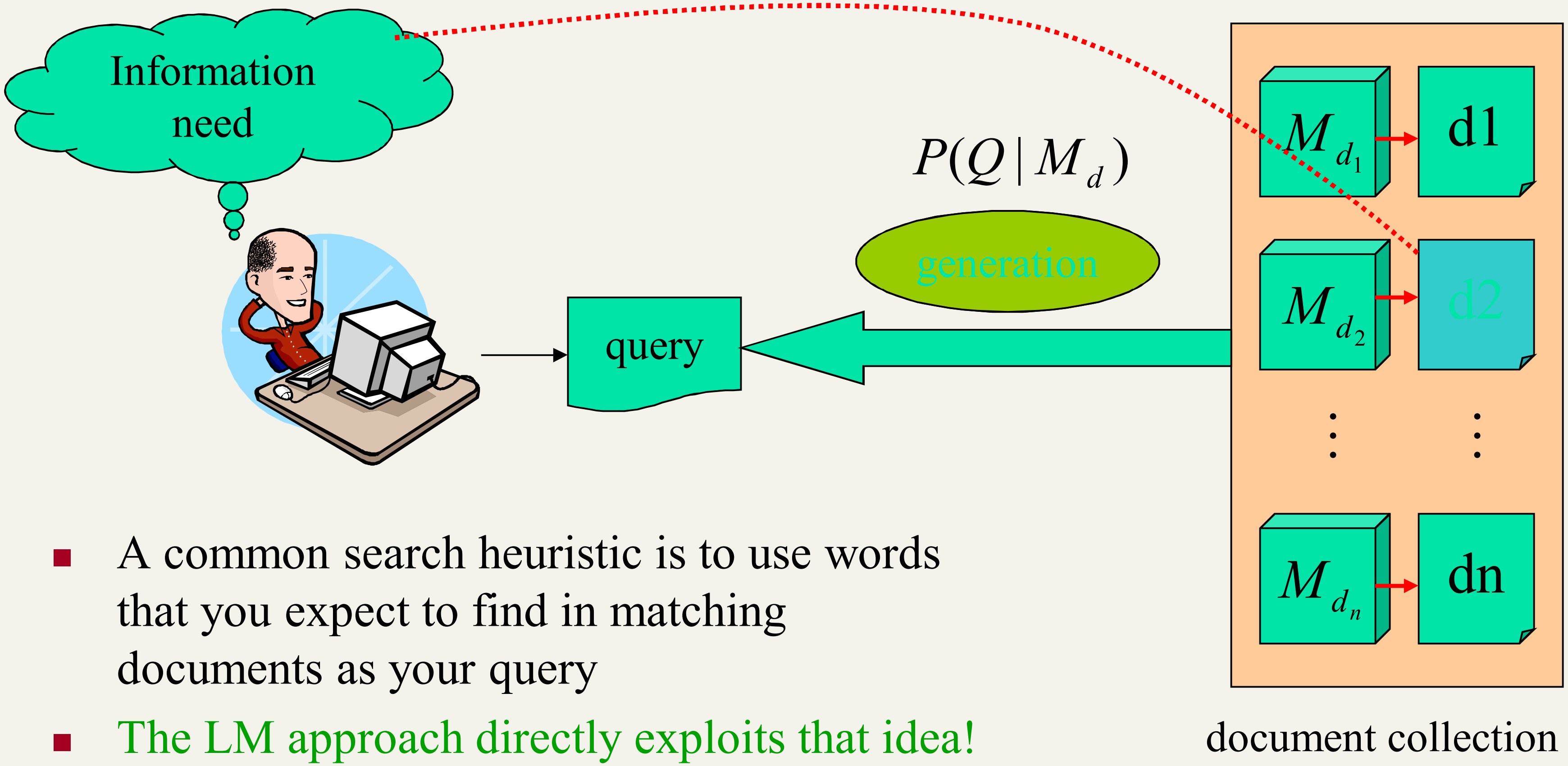
Language Models

Basic Concept

Standard Probabilistic IR



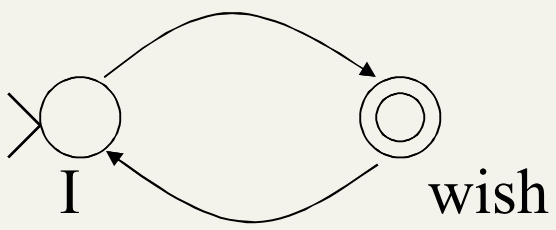
IR based on Language Model (LM)



- A common search heuristic is to use words that you expect to find in matching documents as your query
- The LM approach directly exploits that idea!

Formal Language (Model)

- Traditional generative model: generates strings
  - Finite state machines or regular grammars, etc.
- Example:



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish
…

*wish I wish

Stochastic Language Models

- Models *probability* of generating strings in the language (commonly all strings over alphabet ∑)

Model M

0.2	the
0.1	a
0.01	man
0.01	woman
0.03	said
0.02	likes
…	

the	man	likes	the	woman
0.2	0.01	0.02	0.2	0.01

multiply

$P(s \mid M) = 0.00000008$

Stochastic Language Models

- Model *probability* of generating any string

Model M1

0.2	the
0.01	class
0.0001	sayst
0.0001	pleaseth
0.0001	yon
0.0005	maiden
0.01	woman

Model M2

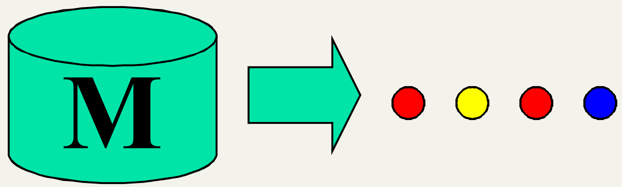
0.2	the
0.0001	class
0.03	sayst
0.02	pleaseth
0.1	yon
0.01	maiden
0.0001	woman

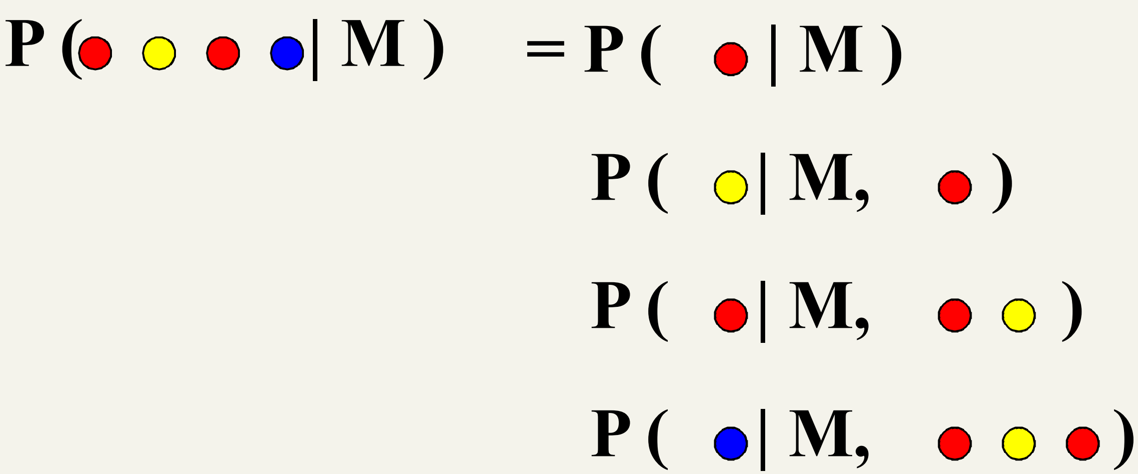
the	class	pleaseth	yon	maiden
0.2	0.01	0.0001	0.0001	0.0005
0.2	0.0001	0.02	0.1	0.01

$$P(s|M2) > P(s|M1)$$

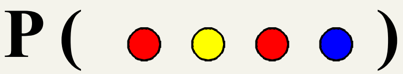
Stochastic Language Models

- A statistical model for generating text
  - Probability distribution over strings in a given language





Unigram and higher-order models



- Unigram Language Models

Easy.
Effective!

- Bigram (generally, *n*-gram) Language Models

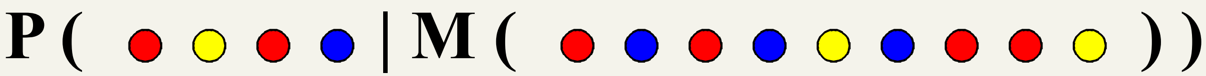
- Other Language Models
  - Grammar-based models (PCFGs), etc.

Using Language Models in IR

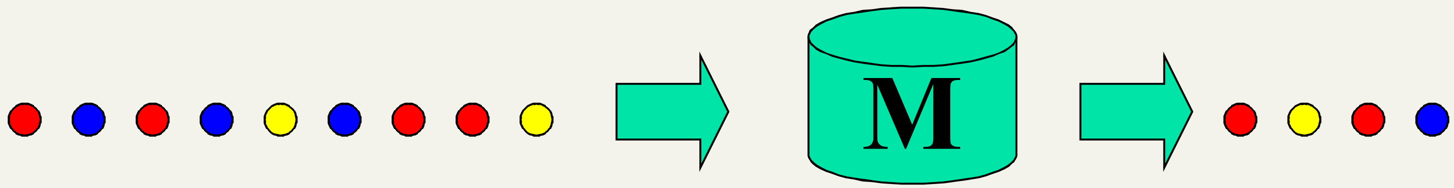
- Treat each document as the basis for a model (e.g., unigram sufficient statistics)
- Rank document d based on P(d | q)
- P(d | q) = P(q | d) x P(d) / P(q)
  - P(q) is the same for all documents, so ignore
  - P(d) [the prior] is often treated as the same for all d
    - But we could use criteria like authority, length, genre
  - P(q | d) is the probability of q given d's model
- Very general formal approach

The fundamental problem of LMs

- Usually we don't know the model **M**
  - But have a sample of text representative of that model



- Estimate a language model from a sample
- Then compute the observation probability



Language Models

- Language models are used in many natural language processing applications
  - Ex: part-of-speech tagging, speech recognition, machine translation, and information retrieval
- To illustrate, the regularities in spoken language can be modeled by probability distributions
- These distributions can be used to predict the likelihood that the next token in the sequence is a given word
- These probability distributions are called **language models**

Language Models

- A **language model** for IR is composed of the following components
  - A set of document language models, one per document d_j of the collection
  - A probability distribution function that allows estimating the likelihood that a document language model M_j *generates* each of the query terms
  - A ranking function that combines these generating probabilities for the query terms into a rank of document d_j with regard to the query

Statistical Foundation

- Let S be a sequence of r consecutive terms that occur in a document of the collection:

$$S = k_1, k_2, \ldots, k_r$$

- An n-gram language model uses a Markov process to assign a probability of occurrence to S:

$$P_n(S) = \prod_{i=1}^{r} P(k_i | k_{i-1}, k_{i-2}, \ldots, k_{i-(n-1)})$$

  where n is the order of the Markov process

- The occurrence of a term depends on observing the $n-1$ terms that precede it in the text

Statistical Foundation

- **Unigram language model** ($n = 1$): the estimatives are based on the occurrence of individual words estimates
- **Bigram language model** ($n = 2$): the estimatives are based on the co-occurrence of pairs of words
- Higher order models such as **Trigram language models** **(**$n = 3$**)** are usually adopted for speech recognition
- **Term independence assumption**: in the case of IR, the impact of word order is less clear
  - As a result, Unigram models have been used extensively in IR

Multinomial Process

- Ranking in a language model is provided by estimating $P(q|M_j)$
- Several researches have proposed the adoption of a multinomial process to generate the query
- According to this process, if we assume that the query terms are independent among themselves (unigram model), we can write:

$$P(q|M_j) = \prod_{k_i \in q} P(k_i|M_j)$$

Multinomial Process

- By taking logs on both sides

$$- \sum_{k_i \in q \wedge d_j} \log P_{\notin}(k_i \,|\, M_j) + \sum_{k_i \in q \wedge d_j} \log P_{\notin}(k_i \,|\, M_j)$$

$$\begin{aligned}
\log P(q|M_j) &= \sum_{k_i \in q} \log P(k_i|M_j) \\
&= \sum_{k_i \in q \wedge d_j} \log P_{\in}(k_i|M_j) + \sum_{k_i \in q \wedge \neg d_j} \log P_{\notin}(k_i|M_j) \\
&= \sum_{k_i \in q \wedge d_j} \log \left(\frac{P_{\in}(k_i|M_j)}{P_{\notin}(k_i|M_j)} \right) + \sum_{k_i \in q} \log P_{\notin}(k_i|M_j)
\end{aligned}$$

where $P_{\in}$ and $P_{\notin}$ are two distinct probability distributions:

- The first is a distribution for the query terms in the document
- The second is a distribution for the query terms not in the document

Multinomial Process

- For the second distribution, statistics are derived from all the document collection
- Thus, we can write

$$P_{\not\in}(k_i|M_j) = \alpha_j P(k_i|C)$$

where α_j is a parameter associated with document d_j and $P(k_i|C)$ is a collection C language model

Multinomial Process

- $P(k_i|C)$ can be estimated in different ways
- For instance, Hiemstra suggests an idf-like estimative:

$$P(k_i|C) = \frac{n_i}{\sum_i n_i}$$

where n_i is the number of docs in which k_i occurs

- Miller, Leek, and Schwartz suggest

$$P(k_i|C) = \frac{F_i}{\sum_i F_i}$$

where $F_i = \sum_j f_{i,j}$

Multinomial Process

- Thus, we obtain

$$\log P(q|M_j) = \sum_{k_i \in q \wedge d_j} \log \left(\frac{P_\in(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j + \sum_{k_i \in q} \log P(k_i|C)$$

$$\sim \sum_{k_i \in q \wedge d_j} \log \left(\frac{P_\in(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j$$

where n_q stands for the query length and the last sum was dropped because it is constant for all documents

$$\sum_{k_i \in q \wedge d_j} \log \left(\frac{P_\in(k_i|M_j)}{P_\notin(k_i|M_j)} \right) + \sum_{k_i \in q} \log P_\notin(k_i|M_j)$$

$$P_\notin(k_i|M_j) = \alpha_j P(k_i|C)$$

Multinomial Process

- The ranking function is now composed of two separate parts
- The **first part** assigns weights to each query term that appears in the document, according to the expression

$$\log \left(\frac{P_{\in}(k_i|M_j)}{\alpha_j P(k_i|C)} \right)$$

- This term weight plays a role analogous to the tf plus idf weight components in the vector model
- Further, the parameter α_j can be used for document length normalization

Multinomial Process

- The **second part** assigns a fraction of probability mass to the query terms that are not in the document—a process called **smoothing** $n_q \log \alpha_j$
- The combination of a multinomial process with smoothing leads to a ranking formula that naturally includes tf, idf, and document length normalization
- That is, smoothing plays a key role in modern language modeling, as we now discuss

Smoothing

- In our discussion, we estimated $P_{\notin}(k_i|M_j)$ using $P(k_i|C)$ to avoid assigning zero probability to query terms not in document d_j
- This process, called **smoothing**, allows fine tuning the ranking to improve the results.
- One popular smoothing technique is to move some mass probability from the terms in the document to the terms not in the document, as follows:

$$P(k_i|M_j) = \begin{cases} P^s_{\in}(k_i|M_j) & \text{if } k_i \in d_j \\ \alpha_j P(k_i|C) & \text{otherwise} \end{cases}$$

where $P^s_{\in}(k_i|M_j)$ is the **smoothed distribution** for terms in document d_j

Smoothing

- Since $\sum_i P(k_i|M_j) = 1$, we can write

$$\sum_{k_i \in d_j} P^s_{\in}(k_i|M_j) + \sum_{k_i \notin d_j} \alpha_j P(k_i|C) = 1$$

- That is,

$$\alpha_j = \frac{1 - \sum_{k_i \in d_j} P^s_{\in}(k_i|M_j)}{1 - \sum_{k_i \in d_j} P(k_i|C)}$$

$$\sum_i P(k_i \mid C) = 1$$

$$\sum_{k_i \in d_j} P(k_i \mid C) + \sum_{k_i \notin d_j} P(k_i \mid C) = 1$$

$$\sum_{k_i \notin d_j} P(k_i \mid C) = 1 - \sum_{k_i \in d_j} P(k_i \mid C)$$

$$P(k_i|M_j) = \begin{cases} P^s_{\in}(k_i|M_j) & \text{if } k_i \in d_j \\ \alpha_j P(k_i|C) & \text{otherwise} \end{cases}$$

Smoothing

- Under the above assumptions, the smoothing parameter α_j is also a function of $P^s_\in(k_i|M_j)$
- As a result, distinct smoothing methods can be obtained through distinct specifications of $P^s_\in(k_i|M_j)$
- Examples of smoothing methods:
  - Jelinek-Mercer Method
  - Bayesian Smoothing using Dirichlet Priors

$$\alpha_j = \frac{1 - \sum_{k_i \in d_j} P^s_\in(k_i|M_j)}{1 - \sum_{k_i \in d_j} P(k_i|C)}$$

Jelinek-Mercer Method

- The idea is to do a linear interpolation between the document frequency and the collection frequency distributions: term

$$P_{\in}^{s}(k_i|M_j,\lambda) = (1-\lambda)\frac{f_{i,j}}{\sum_i f_{i,j}} + \lambda\frac{F_i}{\sum_i F_i}$$

  where $0 \leq \lambda \leq 1$

- It can be shown that

$$\alpha_j = \lambda$$

- Thus, the larger the values of λ, the larger is the effect of smoothing

Dirichlet smoothing

- In this method, the language model is a multinomial distribution in which the conjugate prior probabilities are given by the Dirichlet distribution
- This leads to

$$P_{\in}^{s}(k_i|M_j,\lambda) = \frac{f_{i,j} + \lambda \frac{F_i}{\sum_i F_i}}{\sum_i f_{i,j} + \lambda}$$

- As before, closer is λ to 0, higher is the influence of the term document frequency. As λ moves towards 1, the influence of the term collection frequency increases

Dirichlet smoothing

- Contrary to the Jelinek-Mercer method, this influence is always partially mixed with the document frequency
- It can be shown that

$$\alpha_j = \frac{\lambda}{\sum_i f_{i,j} + \lambda}$$

- As before, the larger the values of λ, the larger is the effect of smoothing

Smoothing Computation

- In both smoothing methods above, computation can be carried out efficiently
- All frequency counts can be obtained directly from the index
- The values of α_j can be precomputed for each document
- Thus, the complexity is analogous to the computation of a vector space ranking using tf-idf weights

Applying Smoothing to Ranking

- The IR ranking in a multinomial language model is computed as follows:
  - compute $P^s_\in(k_i|M_j)$ using a smoothing method
  - compute $P(k_i|C)$ using $\frac{n_i}{\sum_i n_i}$ or $\frac{F_i}{\sum_i F_i}$
  - compute α_j from the Equation $\alpha_j = \frac{1-\sum_{k_i \in d_j} P^s_\in(k_i|M_j)}{1-\sum_{k_i \in d_j} P(k_i|C)}$
  - compute the ranking using the formula

$$\log P(q|M_j) = \sum_{k_i \in q \wedge d_j} \log \left(\frac{P^s_\in(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j$$

Bernoulli Process

- The first application of languages models to IR was due to Ponte & Croft. They proposed a Bernoulli process for generating the query, as we now discuss
- Given a document d_j, let M_j be a reference to a language model for that document
- If we assume independence of index terms, we can compute $P(q|M_j)$ using a multivariate Bernoulli process:

$$P(q|M_j) = \prod_{k_i \in q} P(k_i|M_j) \times \prod_{k_i \notin q} [1 - P(k_i|M_j)]$$

  where $P(k_i|M_j)$ are term probabilities
- This is analogous to the expression for ranking computation in the classic probabilistic model

Bernoulli process

- A simple estimate of the term probabilities is

$$P(k_i|M_j) = \frac{f_{i,j}}{\sum_\ell f_{\ell,j}}$$

  which computes the probability that term k_i will be produced by a random draw (taken from d_j)

- However, the probability will become zero if k_i does not occur in the document
- Thus, we assume that a non-occurring term is related to d_j with the probability $P(k_i|C)$ of observing k_i in the whole collection C

Bernoulli process

- $P(k_i|C)$ can be estimated in different ways
- For instance, Hiemstra suggests an idf-like estimative:

$$P(k_i|C) = \frac{n_i}{\sum_\ell n_\ell}$$

  where n_i is the number of docs in which k_i occurs

- Miller, Leek, and Schwartz suggest

$$P(k_i|C) = \frac{F_i}{\sum_\ell F_\ell} \quad \text{where} \quad F_i = \sum_j f_{i,j}$$

- This last equation for $P(k_i|C)$ is adopted here

Bernoulli process

- As a result, we redefine $P(k_i|M_j)$ as follows:

$$P(k_i|M_j) = \begin{cases} \frac{f_{i,j}}{\sum_i f_{i,j}} & \text{if } f_{i,j} > 0 \\ \\ \frac{F_i}{\sum_i F_i} & \text{if } f_{i,j} = 0 \end{cases}$$

- In this expression, $P(k_i|M_j)$ estimation is based only on the document d_j when $f_{i,j} > 0$
- This is clearly undesirable because it leads to instability in the model

Bernoulli process

- This drawback can be accomplished through an average computation as follows

$$P(k_i) = \frac{\sum_{j|k_i \in d_j} P(k_i|M_j)}{n_i}$$

- That is, $P(k_i)$ is an estimate based on the language models of all documents that contain term k_i
- However, it is the same for all documents that contain term k_i
- That is, using $P(k_i)$ to predict the generation of term k_i by the M_j involves a risk

Bernoulli process

- To fix this, let us define the average frequency $\overline{f}_{i,j}$ of term k_i in document d_j as

$$\overline{f}_{i,j} = P(k_i) \times \sum_i f_{i,j}$$

Bernoulli process

- The risk $R_{i,j}$ associated with using $\overline{f}_{i,j}$ can be quantified by a geometric distribution:

$$R_{i,j} = \left(\frac{1}{1 + \overline{f}_{i,j}} \right) \times \left(\frac{\overline{f}_{i,j}}{1 + \overline{f}_{i,j}} \right)^{f_{i,j}}$$

- For terms that occur very frequently in the collection, $\overline{f}_{i,j} \gg 0$ and $R_{i,j} \sim 0$
- For terms that are rare both in the document and in the collection, $f_{i,j} \sim 1$, $\overline{f}_{i,j} \sim 1$, and $R_{i,j} \sim 0.25$

Bernoulli process

- Let us refer the probability of observing term k_i according to the language model M_j as $P_R(k_i|M_j)$
- We then use the risk factor $R_{i,j}$ to compute $P_R(k_i|M_j)$, as follows

$$P_R(k_i|M_j) = \begin{cases} P(k_i|M_j)^{(1-R_{i,j})} \times P(k_i)^{R_{i,j}} & \text{if } f_{i,j} > 0 \\ \\ \frac{F_i}{\sum_i F_i} & \text{otherwise} \end{cases}$$

- In this formulation, if $R_{i,j} \sim 0$ then $P_R(k_i|M_j)$ is basically a function of $P(k_i|M_j)$
- Otherwise, it is a mix of $P(k_i)$ and $P(k_i|M_j)$

Bernoulli process

- Substituting into original $P(q|M_j)$ Equation, we obtain

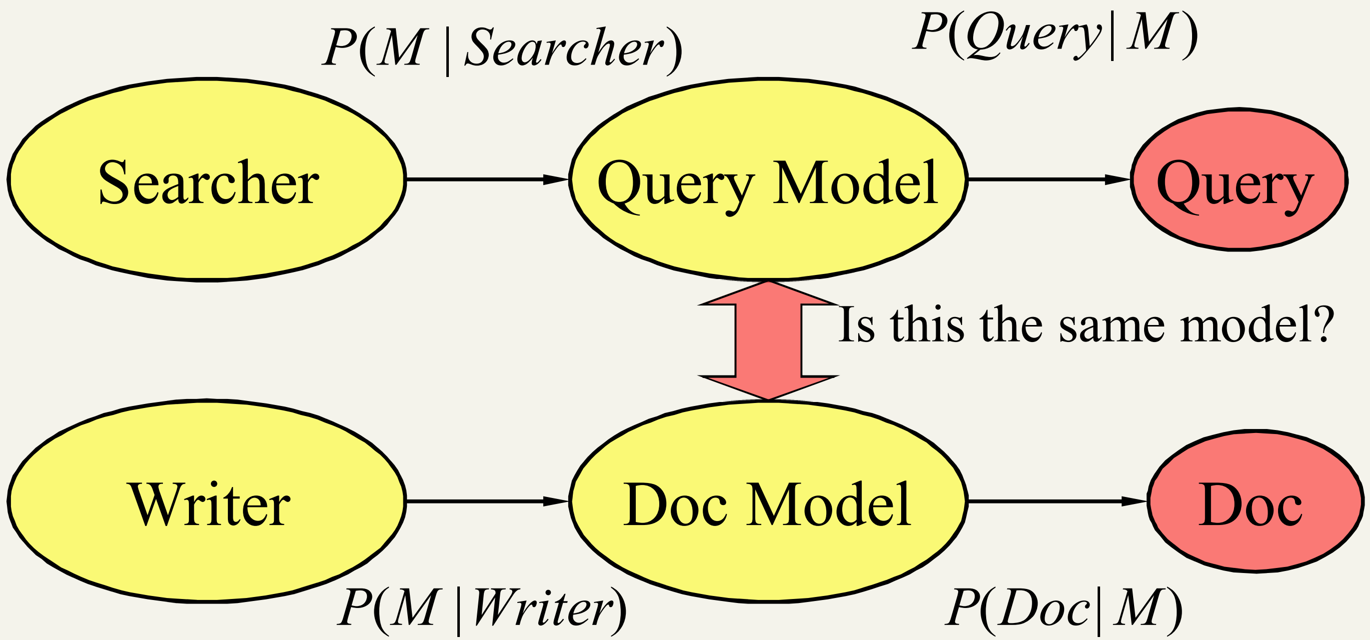
$$P(q|M_j) = \prod_{k_i \in q} P_R(k_i|M_j) \times \prod_{k_i \notin q} [1 - P_R(k_i|M_j)]$$

  which computes the probability of generating the query from the language (document) model

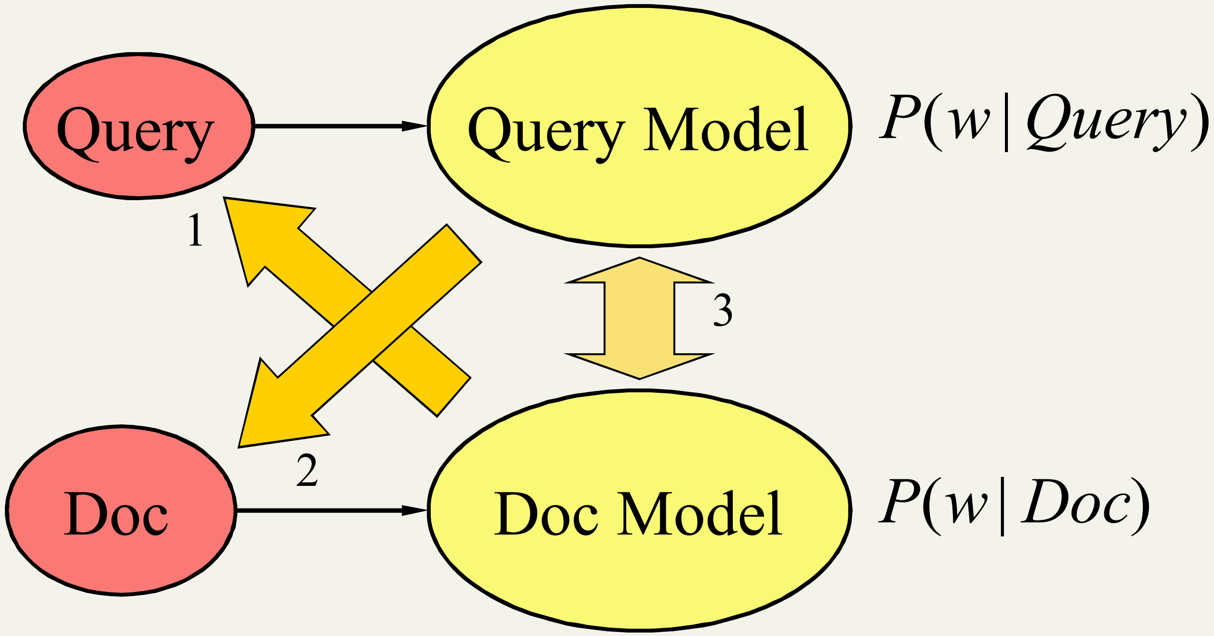
- This is the basic formula for ranking computation in a language model based on a Bernoulli process for generating the query

Query Model vs. Document Model

Alternative Models of Text Generation



Retrieval Using Language Models



Retrieval: Query likelihood (1), Document likelihood (2), Model comparison (3)

Query Likelihood

- Language Modeling Approaches
  - Attempt to **model query generation process**
  - Documents are ranked by **the probability that a query would be observed as a random sample from the respective document model**
- $P(Q|D_m)$
- Major issue is estimating document model
  - i.e. smoothing techniques instead of tf.idf weights
- Problems dealing with relevance feedback, query expansion, structured queries

Document Likelihood

- Rank by likelihood ratio P(D|R)/P(D|NR)
  - treat as a *generation* problem
  - P(w|R) is estimated by P(w|M_Q)
  - M_Q is the query or relevance model
  - P(w|NR) is estimated by collection probabilities P(w)
- Issue is estimation of query model
  - Treat query as generated by mixture of topic and background
  - Estimate relevance model from related documents (query expansion)
  - Relevance feedback is easily incorporated

Model Comparison

- Estimate query and document models and compare
- Suitable measure is KL divergence $D(M_Q \| M_d)$

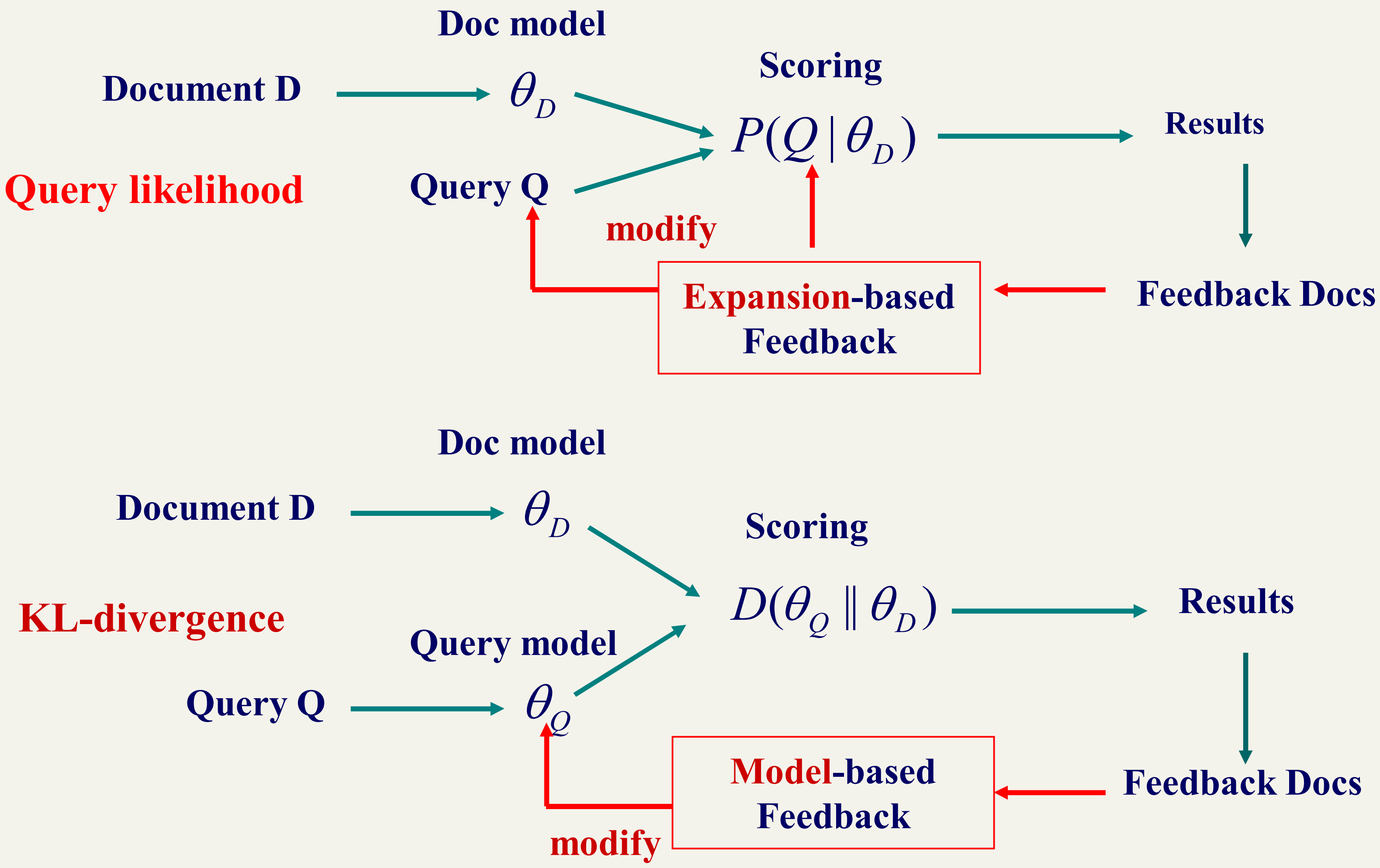
$$R(d;Q) = KL(M_Q \| M_d) = \sum_{t \in V} P(t \mid M_Q) \log \frac{P(t \mid M_Q)}{P(t \mid M_d)}$$

- Better results than query-likelihood or document-likelihood approaches

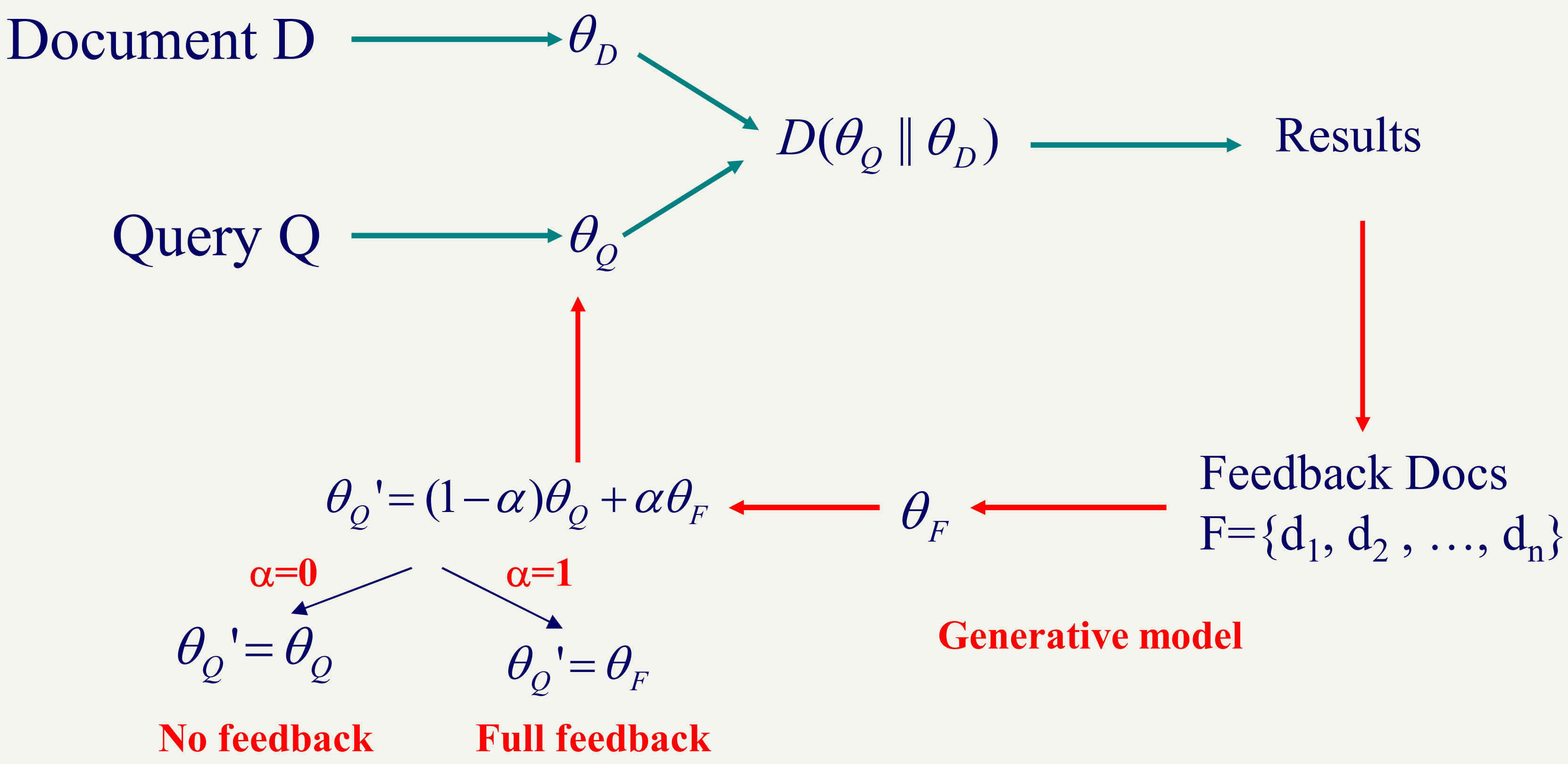
How can one do relevance feedback if using language modeling approach?

- Introduce a query model & treat feedback as query model updating
  - Retrieval function:
    - Query-likelihood => KL-Divergence
  - Feedback:
    - Expansion-based => Model-based

Expansion-based vs. Model-based



Feedback as Model Interpolation



Translation model (Berger and Lafferty)

- Basic LMs do not address issues of synonymy.
  - Or any deviation in expression of information need from language of documents
- A translation model lets you generate query words not in document via "translation" to synonyms etc.
  - Or to do cross-language IR, or multimedia IR

$$P(\vec{q} \mid M) = \prod_i \sum_{v \in Lexicon} P(v \mid M) T(q_i \mid v)$$

Basic LM Translation

  - Need to learn a translation model (using a dictionary or via statistical machine translation)

Language models: pro & con

- Novel way of looking at the problem of text retrieval based on probabilistic language modeling
  - Conceptually simple and explanatory
  - Formal mathematical model
  - Natural use of collection statistics, not heuristics (almost…)
- LMs provide effective retrieval and can be improved to the extent that the following conditions can be met
  - Our language models are accurate representations of the data.
  - Users have some sense of term distribution.*
    - *Or we get more sophisticated with translation model

Comparison With Vector Space

- There's some relation to traditional tf.idf models:
  - (unscaled) term frequency is directly in model
  - the probabilities do length normalization of term frequencies
  - the effect of doing a mixture with overall collection frequencies is a little like idf: terms rare in the general collection but common in some documents will have a greater influence on the ranking

Comparison With Vector Space

- Similar in some ways
  - Term weights based on frequency
  - Terms often used as if they were independent
  - Inverse document/collection frequency used
  - Some form of length normalization useful

- Different in others
  - Based on probability rather than similarity
    - Intuitions are probabilistic rather than geometric
  - Details of use of document length and term, document, and collection frequency differ