Text Retrieval and Search Engines

Probabilistic Model

The probabilistic text retrieval model uses classic probabilistic arguments to determine whether or not a document is likely to be relevant.

$$f(d,q) = p(R=1|d,q) \ \ where \ R \in \{0,1\}$$

Query Likelihood

The query likelihood model calculates the probability that a user searching for the relevant document *d* will enter the query *q*. This would allow the ranking function to be approximated as follows:

$$f(d,q) = p(R=1|d,q) \approx \boldsymbol{p(q|d,R=1)}$$

The success of this model is based on the assumption that the user formulates a query based on an imaginary relevant document.

The problem in the query likelihood model becomes that of calculating the probability $p(q|d)$. This problem is related to that of calculating the probability of text in general.

Language Model (LM)

A language model is a statistical method for generating the probability of word sequences in a text. Given a language model a user can generate likely text sequences based on high probability words so the language model is also sometimes referred to as a **Generative Model**.

Since word sequences are context dependent the language model can be used to examine **topics** and **word associations** within a text.

A language model that considers all possible sequences of text would produce too many useless sequences so a number of assumptions need to be made to restrict the possibilities. One such assumption is the **n-gram** model which assumes that the probability of a sequence depends on the probability of the n words preceding it. In the case where $n = 1$ the language model is known as the **Unigram Model**.

The Unigram Model

In the unigram language model the probability of a text sequence is calculated by looking at the probability of each **individual word** assuming they are generated independently. The probability of a sequence is the product of the probabilities of each word in the sequence:

$$\boldsymbol{p(w_1 w_2 \dots w_n) = p(w_1)p(w_2) \dots p(w_n)} \ \ where \ p(w_1) + \cdots + p(w_N) = 1 \ (N \ is \ the \ vocabulary \ size)$$

Estimation of Unigram LM

Given a document *d* it is useful to attempt to calculate its language model. One method is to use the Maximum Likelihood (ML) estimator which bases the probability of a word on its frequency within the document:

$$p(w|\theta) = p(w|d) = \frac{count(w,d)}{|d|} \quad where \ |d| is \ the \ total \ number \ of \ words \ in \ the \ document$$

Topic Representation with LMs

The **Background LM** $p(w|B)$ is the language model of a generic text in a given language and indicates the probability of a word regardless of any other context. The **Collection LM** $p(w|C)$ is the language model for a document collection and the **Document LM** is th

e language model for an individual document. These different LM's can be used to analyze associations within texts. For example the language model of a topic represent by the word w_T can be more accurately represented by normalizing it with the background LM:

$$normalized\ topic\ LM = \frac{p(w|w_{T)}}{p(w|B)}$$

Maximum Likelihood Method for Query Probability

The maximum likelihood (ML) method can be used to generate the probability of a query given a relevant document by looking at the frequency of query terms in the document:

$$probabilty\ (query = \ w_1 \dots w_n) = \prod_{i=1}^{n} \frac{count(w_i, d)}{|d|}$$

One obvious problem with this approach is in the case where a query term is not in the document, the count value will be zero and the whole calculation will have a value of zero, an improved model is required.

Unigram Query Likelihood

The unigram query likelihood model generates a ranking function for a query using the LM of the query and the LM of the document. Using the document LM the probability of a query $q = \ w_1 \dots w_n$ is:

$$p(q|d) = p(w_1|d) \times \dots \times p(w_i|d)$$

In order to avoid multiplying by lots of small numbers $p(q|d)$ is replaced with $\log p(q|d)$ and the ranking function becomes:

$$f(q,d) = \log p(q|d) = \sum_{i=1}^{n} \log p(w_i|d) = \sum_{w \in V} count(w,q) \log p(w|d)$$

This means **that the problem of generating the ranking function becomes one of generating the LM for the document $p(w|d)$.**

Estimation of the Document LM

The simplest way to determine the document LM using the maximum likelihood (ML) method gives:

$$P_{ML}(w|d) = \frac{count(w,d)}{|d|}$$

This method has a problem in that query words not contained in the document generate a value of zero. In order to avoid this a **smoothed LM** is used so that $p(w|d) > 0$ even if $count(w|d) = 0$.

Smoothed Document LM

A smoothed document LM assigns a non-zero probability to all words, even those not in the document or the query. One way to do this is by using a reference LM such as the document collection LM. The document LM is then equal to the discounted maximum likelihood estimate for words that are in the document and is equal to the collection LM for words that are not in the document.

$$p(w|d) = \begin{cases} P_{Seen}(w|d)\ if\ w\ is\ in\ d \\ \alpha_d p(w|C)\ if\ w\ is\ not\ in\ d \end{cases}$$

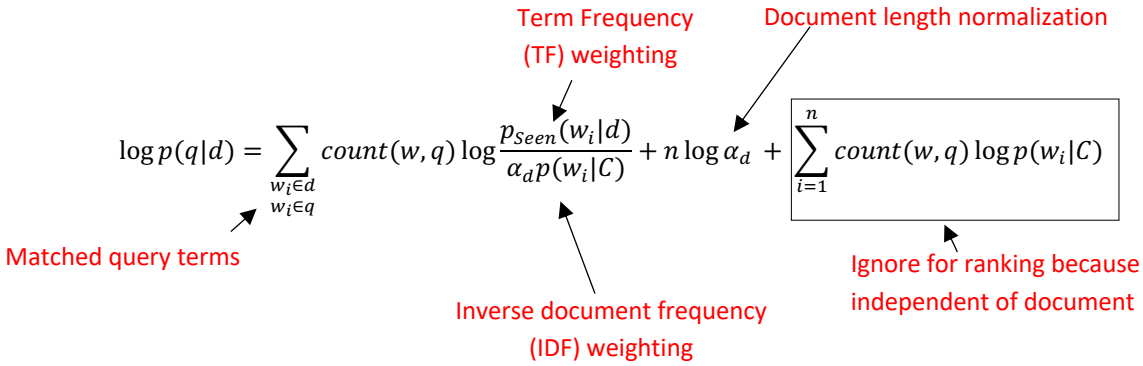
Where α_d is a user-defined parameter that constrains the probability of unseen words.

So the ranking function becomes:

$$\log p(q|d) = \sum_{w \in V} count(w,q) \log p(w|d)$$

$$= \sum_{w \in V, c(w,d) > 0} count(w,q) \log p_{Seen}(w|d) + \sum_{w \in V, c(w,d) = 0} count(w,q) \log \alpha_d p(w|C)$$

Which can be rewritten as:



So the ranking function can be expressed as:

$$f(q,d) = \sum_{\substack{w_i \in d \\ w_i \in q}} count(w,q) \log \frac{p_{Seen}(w_i|d)}{\alpha_d p(w_i|C)} + n \log \alpha_d$$

Which leaves the characteristics of the smoothed document LM $p_{Seen}(w_i|d)$ and α_d to be determined.

Linear Interpolation, Jelinek-Mercer (JM) Smoothing

This is a fixed coefficient linear interpolation smoothing method with only one smoothing parameter λ.

$$p(w|d) = (1-\lambda)\frac{c(w,d)}{|d|} + \lambda p(w|C) \; where \; \lambda \in [0,1]$$

Ranking Function for JM Smoothing

$$f_{JM}(q,d) = \sum_{\substack{w \in d \\ w \in q}} c(w,q) \log \left[1 + \frac{1-\lambda}{\lambda} \frac{c(w,d)}{|d| p(w|C)}\right] \; where \; \lambda \in [0,1]$$

Dirichlet Prior (Bayesian) Smoothing

This is an adaptive interpolation smoothing method which works by adding pseudo counts. It has only one smoothing parameter μ.

$$p(w|d) = \frac{c(w,d) + \mu p(w|C)}{|d| + \mu} = \frac{|d|}{|d| + \mu}\frac{c(w,d)}{|d|} + \frac{\mu}{|d| + \mu} p(w|C) \; where \; \mu \in [0, +\infty]$$

Ranking Function for Dirichlet Prior Smoothing

$$f_{DIR}(q,d) = \left(\sum_{\substack{w \in d \\ w \in q}} c(w,q) log \left[1 + \frac{c(w,d)}{\mu p(w|C)}\right]\right) + n log \frac{\mu}{\mu + |d|} \; where \; \mu \in [0, +\infty]$$

Query Feedback

Query feedback is used to refine a query to improve the results returned from the retrieval system. The approach improves a query by learning from examples.

Relevance Feedback

Relevance feedback takes place when users make explicit relevance judgements on the initial results. These judgements are reliable but are often hard to obtain because they require extra effort on the part of the user.

Pseudo Feedback

Pseudo feedback, also known as **Blind** or **Automatic** feedback generates feedback without requiring input from users. The top-k initial results are assumed to be relevant and are used to modify the query. Pseudo feedback is not as reliable as relevance feedback but it is easier to automate since no user activity is required.

Implicit Feedback

Implicit feedback assumes that any user-clicked documents are relevant and skipped documents are non-relevant. Relevant documents are used to refine the query. This is more reliable than pseudo feedback but not as reliable as relevance feedback.

Query Modification

Query modification processes use feedback from retrieved results to modify the initial query either by adding new (weighted) terms (query expansion) or by adjusting the weights of the old terms.

Rocchio Feedback

Rocchio feedback is a query modification technique that utilizes the vector space model. It calculates the centroids of relevant documents and non-relevant documents in vector space and adjusts the location of the query vector according to the formula:

$$\overrightarrow{q_m} = \alpha\vec{q} + \frac{\beta}{|D_{r|}}\sum_{\forall\overrightarrow{d_j}\in D_r}\overrightarrow{d_j} - \frac{\gamma}{|D_n|}\sum_{\forall\overrightarrow{d_j}\in D_n}\overrightarrow{d_j}$$

Where $\overrightarrow{q_m}$ is the new query, $\vec{q}$ is the original query, D_r are the relevant documents and D_n are the non-relevant documents.

In practice the non-relevant documents contribute very little to any improvement in the query and they are often disregarded to increase efficiency. The vector is often truncated so that only a small number of the words with the highest weights contribute to the centroid vector. Additionally one must take care to avoid over-fitting the query and diverging too much from the original query.

Rocchio feedback can be used for relevance feedback and pseudo feedback (β should be set to a higher value for relevance feedback).

Kullback-Leibler (KL) Divergence Retrieval Model

Since query likelihood method cannot naturally support relevance feedback another approach is required. The Kullback-Leibler (KL) divergence retrieval model is one solution. Feedback is achieved through query model estimation and updating.

The KL-divergence (cross entropy) is measured using:

$$f(q,d) = \sum_{w\in d, p(w|\hat{\theta_Q})>0} p(w|\hat{\theta}_Q)\log\frac{p_{Seen}(w_i|d)}{\alpha_d p(w_i|C)} + n\log\alpha_d$$

Where $p(w|\hat{\theta}_Q)$is the query LM.

Feedback is generated by first determining a document LM and a query LM and calculating the KL-divergence, usually expressed as $D(\theta_Q||\theta_D)$, to generate results. Feedback from the results is used to generate a feedback LM θ_F which creates a modified query $\theta_Q{}'$ in the following way:

$$\theta_Q' = (1-\alpha)\theta_Q + \alpha\theta_F \;\; where\; \alpha \in [0,1]$$

The problem then becomes one of generating the feedback LM θ_F. One way of doing this is with a **Generative Mixture Model** which builds up the LM using a mixture of the background LM for the general language and the relevant topic LM so

$$\log p(F|\theta) = \sum_i\sum_w c(w,d_i)\log[(1-\lambda)p(w|\theta) + \lambda p(w|C)]$$

Where λ is a noise parameter that determines the contribution of common words from the background LM.