Smart Document Ranking

Jingwei Lu (0537619)

Abstract: In information retrieval area, one interesting problem is that how to provide additional documents to answer user's query. Suppose user is looking for documents contain keyword "Amazon", first we will return all documents contain keyword Amazon. To provide additional value for customer, we may also return documents similar to documents contain keyword Amazon, such as a book talks about rain forest. The important question is how to rank the documents. The problem can be considered as you have an undirected graph. Each node is a document. You can calculate the similarity between any two documents and the similarity will be the weight value on the edge between two document nodes. Each node will have a rank value for certain query. When we answer the query, the query first will directly hit some nodes which contain the keyword in the graph. Now the problem becomes how to use the weight value to calculate the new rank value for all the documents connected to the hits and return top N value. The rank problem can be modeled as MARKOV RANDOM FIELDS and it can be solved using linear programming. There are major problems when we try to implement it to handle hundreds millions of documents, such as how to efficiently build the graph, how to efficiently do the linear programming.

Introduction

Information retrieval becomes a hot area in recent years. Google, msn, yahoo, etc. are mainly deal with web page search. There is another big search area which is enterprise search. In enterprise search area, search engine needs to deal with different type of textural data. It could be document, record inside the database, or all kinds of texture attributes of a file. In web page search area, one of the best ranking components is page rank, which considers the hyperlink relationship between pages. In enterprise search, there is no hyperlink can be used. So right now each document is considered independent. This often leads to poor ranking.

Microsoft SQL Server Full-text Search is investing to extend ranking algorithm for enterprise search to take consideration of similarity between documents. Even though there is no hyperlink can be used directly. There is other metadata information and the document itself can be used to measure similarity between documents. We proposed to use BM-25[1] to model similarity between documents first. Then we use an undirected graph model in a Markov Random Field framework to model the association of rank of a document with the features it contains and the rank of the nearby documents. Markov Random Field with Laplacian distribution was successfully used in [2] for modeling spatial dependencies of different regions in the image. It can be used to model the dependencies between documents. However, our biggest problem is how the method can be used to deal with millions of documents efficiently. Our group has done some preliminary test of the ranking idea as an intern

project and it works reasonably well. The performance of this algorithm is not fast enough to handle document in millions range. Our goal is to handle hundred millions documents before it can be productized. In section 2, I will talk about document similarity briefly. In second 3, I will talk about how to use Markov Random Field to rank the documents for a query. In second 4, I will discuss using linear programming to solve the ranking. In second 5, I will discuss some idea of how to scale the method to millions of documents. Finally it is conclusion.

Document similarity

Various methods are used for document similarity, ranging from simple "bag-of-words" to those syntactically modeling languages. To our surprise, the simple bag-of-words model works reasonably well in many scenarios. The best part of the "bag-of-words" model is that it is not computationally intensive. The idea behind "bag-of-words" is to treat each document as a bag of independent words. We don't consider the relationship between words but only consider the frequency of words in the document. We follow the 2-Poisson model [3], where within document word frequencies are modeled as a mixture of two Poisson distributions. It is hypothesized that occurrences of the terms in the document has a random or stochastic element. It reflects a distinction between those documents which are about the concept represent by the term and those which are not. With some approximation, a widely used algorithm BM-25[1] is as following:

$$w_j = \frac{(k_1+1)d_j}{k_1(1-b)+b\times\frac{dl}{avdl}+d_j}\log\left(\frac{N-df_j+0.5}{df_j+0.5}\right)$$

w_j is weight for a term, k_1, b are constant, d_j is term frequency in a document, dl is document length, $avdl$ is average document length of the corpse. N is total number of document. df_j document frequency(how many times the term appear in all documents.). We do not normalize the frequencies by the document length terms dl and $avdl$ because document length is a factor which plays important role in document similarity. Two documents are more similar if their length is close. So above formula can further simplifier to:

$$w_j = \frac{d_j}{1+k_2 d_j}\log\left(\frac{N-df_j+0.5}{df_j+0.5}\right)$$

For each document, all non-noise words(word that carries meaning) can get a weight value based on above formula. The feature vector of the document is the all non-noise word with a weight value. The distance between to document x, y is given as:

$$\beta_{xy} = x\cdot y/\| x \|\| y \|$$

x, y is the feature vector of two document. Thus we have a similarity measure between two documents. In future work, both the feature selection (for example, including metadata into features) and the way to measure similarity can be improved independent of the rest of algorithm to improve the

quality of overall ranking. I will not drill into more detail of document similarity because this is not the main focus of this report. At this point let's assume we have a way to calculate similarity between two documents.

Using Markov Random Field

Intuitively, we think the rank of a document given a query depends on the features (terms) of that document as well as other document which is similar to this document. For example, if two documents are very similar, their ranks should not be very different. In current full-text search system, two documents are considered independent, so it could not refine rank value based on document similarity. For example, if user query for "Amazon" in all books in a library, current system will only return books that contains exactly keyword Amazon. For the proposed system, it will evaluate all books which is similar to the book that contains keyword "Amazon", it may also return book talks about rain forest because those books are similar to the book talk about Amazon rain forest.

Markov Random Field is a powerful tools used in computer vision to model relationship between segments. To apply the MRF to document ranking problem, we can model each document rank value as probability value of each node in Markov Random Field. The similarity value between two documents becomes dependency between two nodes. Our ranking idea maps nicely to MRF. A node has high probability value will more likely to influence the connected nodes to have a high probability value. A document which is near documents having high rank (high probability value) should be considered important. Therefore, we need a probability model that gives ranks of documents taking consideration of relationship between the documents. We model the rank r of a document for a given query q as:

$$P(r|q) = \left(\frac{1}{Z}\right) \exp(-|r - r_0| - \mu \sum_{i,j \in G} \beta_{ij} |r_i - r_j|)$$

Z is normalizing constant. r is new ranking value consider the document similarity. r_0 is ranking value consider document is independent. β_{ij} is similarity between two documents. μ is an adjustable coefficient to control how much similarity should influent the ranking. Above formula pose the probability distribution of ranks as a Markov network, given the original ranks r_0 determined by document feature set only. The probability of a set of rank assignment decreases if two similar document are assigned different ranks because of term $\mu \sum_{i,j \in G} \beta_{ij} |r_i - r_j|$. We also want to constrain the new rank value to be close to the old value. The first term actually controls this concept.

For simplicity, we use linear function ($|r - r_0|$) in above formula to model the error. To better capture the statistic model of document, 2-poisson model can be used for the first term. For second term, we also used a linear function. If we assume the noise is Gaussian, we can use standard least squares penalty function ($\mu \sum_{i,j \in G} \beta_{ij} |r_i - r_j|^2$). Choose the linear function for both terms gives us an easy solution to find maximum likelihood of the probability. We can use linear programming to solve the problem. If 2-poisson model and least squares penalty function are used. We will need other tool

such as Second Order Cone Program to find a solution. Our prototype shows that the linear model works reasonable well.

Using linear programming to solve the problem

To estimate the maximum likelihood of $P(r|q)$ for rank value of all nodes, it is equally to solve following linear program problem:

$$Min \qquad 1^T\xi_1 + \mu\xi_2$$

$$Constrains: \qquad |r_{(n)} - r_{0_{(n)}}| \leq \xi_{1_{(n)}}, \ \mathrm{n} = 1, 2, \dots, \mathrm{N}$$

$$\sum_{i,j \in G} \beta_{ij} |r_i - r_j| \leq \xi_2$$

Where N is the total number of documents, G is the undirected weighted graph of the documents. μ is a free parameter to tune how important is the similarity in the final rank. We also don't want the rank of original M documents that found by the keyword search to decrease after the calculation. So we will add additional constrains for that:

$$r_{(m)} - r_{0_{(m)}} \geq 0, \ \mathrm{m} = \mathrm{k}_1, \mathrm{k}_2, \dots, \mathrm{k}_\mathrm{m}$$

$\mathrm{k}_1, \mathrm{k}_2, \dots, \mathrm{k}_\mathrm{m}$ are original m documents hit by keyword search.

To do smart document ranking, we will first use keyword search to find all documents that contains the keyword and assign initial rank value for those documents. Then we solve the linear program to refine the rank value for all documents that connected to the first pass result. This minimization is to make sure the new rank value to have as small error as possible and similar document has similar rank value.

Scale to large dataset

In real enterprise search, we are dealing with millions of documents. The first problem is that to calculate and store full similarity matrix between every two document becomes impossible. Second, to solve the linear programming efficiently, we need to trim down the related documents to reduce the number of variables.

The complexity of generate the similarity matrix between every two documents is $O(n^2)$ and the space complexity is also $O(n^2)$. When n is in hundred millions of range, the algorithm becomes unusable due to the time and space complexity. Our idea is to solve the problem by using off-the-shelf clustering algorithm (such as K-Means Clustering) to cluster the documents on the fly. The full-text indexing engine processes document one by one. When a new document comes into the system, it compares its feature vector with each cluster's feature vector. The document then is assigned to one of the best match cluster. Then it will calculate similarity between each document in the cluster. The cluster will automatically split according to the clustering algorithm. The important thing is that we

need to maintain a limited number of clusters and limit the size of each cluster to be not so large. Suppose we have m cluster and each one has on average k documents. The complexity to generate the similarity matrix is $O(n * m + k^2)$. Since when m and k is $\sqrt{n}$ we get optimal solution, the complexity becomes $O(n^{1.5})$. The space complexity is also $O(n^{1.5})$ since we only need to calculate similarity within the cluster. After using clustering, the complexity of the algorithm is much more acceptable than original algorithm. However, we would like to get a close to $O(n)$ algorithm so that we can use it in real product.

The second problem is that there are too many variables in the linear programming problem. Each variable correspond to a rank value of a document. There are two source of document that will come into final result set: the direct hit documents of keyword search and the connected documents from the direct hit documents by the query. We need to trim both the direct hit documents as well as connected documents. The direct hit has an initial rank value. We can trim the documents which has initial rank value less than a threshold. We will also trim connected document which has similarity less than another threshold. The reason that trimming the document works for us is that user usually only wants top n(n is small and usually is less than 100) result sorted by rank. As long as trim keep certain percentage of ideal result in the final result set (this is called precision of the query) it will work for us. For given precision requirement and top n value, we can dynamically decide the threshold for the trimming.

Conclusion

We have tried an idea of using bag of word model to calculate document similarity. Then use Markov Random Field to refine full-text query ranking based on the document similarity matrix. Our prototype [4] shows that the idea actually works well for small dataset. When we need to handle hundred millions of documents, the performance becomes a problem. We suggested using clustering and trimming document to improve the performance. However, it is still not acceptable to be used in real production. Our future direction is to find a new linear complexity algorithm to do similar smart document query ranking.

References:

[1] Stephen Robertson, Hugo Zaragoza and Michael Taylor, Simple BM25 Extension to Multiple Weighted Fields, CIKM 2004, Washington, DC, USA

[2] Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng, Learning Depth from Single Images, to appear in NIPS 18, 2006, to be held in Canada.

[3] S E Robertson and S Walker, Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In W B Croft and C J van Rijsbergen, editors, SIGIR 1994.

[4] Ashutosh Saxena, Jingwei Lu, Nimish Khanolkar, SmartQuery: Software Design Documentation, Microsoft technical report, Sep 2005.