Information Retrieval

Shuai Wei
wei6@g.clemson.edu

Yunqing Zhang
yunqinz@g.clemson.edu

Zian Chen
zianc@g.clemson.edu

INTRODUCTION

Information Retrieval (IR) is the discipline that deals with retrieval of unstructured data, especially textual documents, in response to a query or topic statement, which may itself be unstructured, e.g., a sentence or even another document, or which may be structured, e.g., a boolean expression.The need for effective methods of auto- mated IR has grown in importance because of the tremendous explosion in the amount of unstructured data, both internal, corporate document collections, and the immense and growing number of document sources on the Internet.This report discussed the 'boolean retrieval' model, 'tf-idf' model,'BM25' model. Besides theses three models, we talked about the new idea to implement the information retrieval.[1]

THREE ALGORITHMS

During our project, 'boolean retrieval', 'tf-idf' model and 'BM25'model are implemented.Now we discuss these three algorithms and compare them with each other.

Boolean Approach to IR

In the boolean case, the query is formulated as a boolean combination of terms. A conventional boolean query uses the classical operators AND, OR, and NOT. The query "t1 AND t2" is satisfied by a given document D1 if and only if D1 contains both terms t1 and t2. Similarly, the query "t1 OR t2" is satisfied by D1 if and only if it contains t1 or t2 or both. The query "t1 AND NOT t2" sat- isfies D1 if and only if it contains t1 and does not contain t2. More complex boolean queries can be built up out of these operators and evaluated according to the classical rules of boolean algebra. Such a classical boolean query is either true or false.[2, 3, 4, 5] Correspondingly, a document either satisfies such a query (is "relevant") or does not satisfy it (is non-relevant"). Several kinds of refinement of this classical boolean query are possible when it is applied to IR. First, the query may be applied to a specified syntactic component of each document, e.g., the boolean condition may be applied to the title or the abstract rather than to the document as a whole. Second, it may be specified that the condition must apply to a specified position within a syntactic component, e.g., to the words at the beginning of the title rather than to any part of the title.
Disadvantage:

1. The classical boolean approach does not use term weights. Or, what comes to the same thing, it uses only two weights, zero (a term is absent) and one (a term is present).

2. No ranking is possible, a significant limitation.

TF-IDF model TO IR

The "term frequency" (tf) is the frequency of occurrence of the given term within the given document. Hence, tf is a document-specific statistic; it varies from one document to another, attempting to measure the importance of the term within a given document. By contrast, inverse document frequency (idf) is a "global" statistic; idf characterizes a given term within an entire collection of documents. It is a measure of how widely the term is distributed over the given collection, and hence of how likely the term is to occur within any given document by chance. The idf is defined as "ln (N/n)" where N is the number of documents in the collection and n is the number of documents that contain the given term. Hence, the fewer the documents con- taining the given term, the larger the idf. If every document in the collection contains the given term, the idf is zero. This expresses the commonsense intuition that a term that occurs in every document in a given collection is not likely to be useful for distinguishing relevant from non-rele- vant documents. Or what is equivalent, a term that occurs in every document in a collection is not likely to be useful for distinguishing documents about one topic from documents about another topic. To cite a commonly-used example, in a collection of documents about computer science or software, the term "computer" is likely to occur in all or most of the documents, so it won't be very good at discriminating documents relevant to a given query from documents that are non-rel- evant to the given query.
Advantege:
Computing the weight of a given term in a given document as tf*idf says that the best descriptors of a given document will be terms that occur a good deal in the given document and very little in other documents. Similarly, a term that occurs a moderate number of times in a moderate proportion of the documents in the given collection will also be a good descriptor. Hence, the terms that are the best document descriptors in a given collection will be terms that occur with moderate frequency in that collection. The lowest weights will be assigned to terms that occur very infrequently in any document (low-frequency documents), and terms that occur in most or all of the documents (high frequency documents).

TF-IDF implementation comparision

In TF-IDF stage, the implementation of our three had some different.Shuai Wei used three mapreduce job to implement tf-idf algorithm. And the three mapreduce is like below,

1. Create index files for tf-idf model.
   and the i^{th} job's output will be used for the $(i+1)^{th}$ job's input for $i = 0, 1, 2$.

   - MapReduce1
     - Input: Zip file

  - Output:
    * key : term
    * value: <docId1,tf1; docId2,tf2,...>
- MapReduce2
  - Input:
    * key : term
    * value: <docId1,tf1; docId2,tf2,...>
  - multiple outputs:
    * Output1:
      · key : term
      · value: <docId1; docId2;...>
    * Output2:
      · key : docId
      · value: <term1,tf1,df1; term2,tf2,df2;...>
- MapReduce3
  - Input:
    * key : docId
    * value: <term1,tf1,df1; term2,tf2,df2;...>
  - Output:
    * key : docId,mold,dl
    * value: <term1,tf1,df1; term2,tf2,df2;...>

For Zian ,he used just one Mapreduce to calculate tf-idf value. The reason why he could put all these in one job is that he calculated all the mold values for each single word in evey document, and put them into the index file. So in this way, he could get all the mold values he needed when he calculate idf part.And the main calculation of his tf-idf is in conbiner step. Mapper was just responsible for index tokenization. And when he gets all the sub-results in combiner, he just did a ranking within each combiner(several maps). And when map-reduce goes into reducer, he merged all the sub-results which was ranked,and did the whole ranking within all of them.Then he got his final result and showed up the top 10.
For Yunqing, her medthod is more similiar to Zian, also use one Mapreduce. Parse ZipFile, calculate the target value for tf-idf for every word of each document.The Index file store the word, docId, target value.Then goes through the normal Mapper, combiner, reducer process.

Okapi BM25 to IR

BM25 represent "Best Match 25", developed in the context of the Okapi system. BM25 is a famiy of widespread used scoring functions based on probabilistic term weighting models. One of the most common formula is as follows.

$$RSV^{BM25} = \sum_{i \in q} log \frac{N}{df_i} \frac{(k_1+1)tf_i}{k_1((1-b)) + b(\frac{dl}{avdl})) + tf_i} \tag{1}$$

where dl is document length and $avdl$ is the average document length.
A lot of parameters are contained in the BM25 formula which need to be learned from relevance assessment.

(a) k_1 controls term frequency scaling.

- If $k_1 = 0$, it is a binary model since $RSV = sum_{i \in q} log \frac{N}{df_i}$.
- If k_1 is large,it is raw term frequency.

Because when $k_1 \to \infty$,

$$\begin{aligned} RSV^{BM25} &= \lim_{k_1 \to \infty} RSV^{BM25} \\ &= \lim_{k_1 \to \infty} \sum_{i \in q} log \frac{N}{df_i} \frac{(k_1+1)tf_i}{k_1((1-b)) + b(\frac{dl}{avdl})) + tf_i} \\ &= \sum_{i \in q} log \frac{N}{df_i} \frac{tf_i}{(1-b)) + b(\frac{dl}{avdl})}. \end{aligned} \tag{2}$$

(b) $b_1 \in [0,1]$ controls document length normalization.

- When $b = 0$, there is no length normalization.
- When $b = 1$, it is fully scaled by document length.

Typically, k_1 is set around 1.5–2 and b about 0.75.

Moreover, from this formula we infer that BM25 algorithm can control the effect of each term frequency since when $tf_i \to \infty$,
we have

$$\begin{aligned} RSV^{BM25} &= \lim_{tf_i \to \infty} RSV^{BM25} \\ &= \lim_{k_1 \to \infty} \sum_{i \in q} log \frac{N}{df_i} \frac{(k_1+1)tf_i}{k_1((1-b)) + b(\frac{dl}{avdl})) + tf_i} \\ &= \sum_{i \in q} log \frac{N}{df_i} (k_1+1). \end{aligned} \tag{3}$$

Let $RSV_i = \frac{(k_1+1)tf_i}{k_1((1-b))+b(\frac{dl}{avdl}))+tf_i}$.
Then the $RSV_i - tf_i$ relation graph is given in the following.

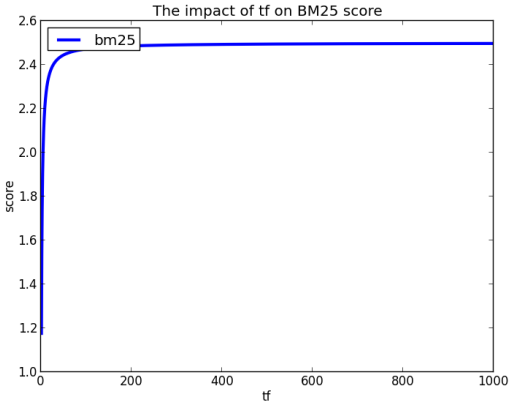

Figure 1. The RSV-tf relationship.

implementation

(a) Create index files for tf-idf and BM25 model.
In the process of creating index, we use chain map reduce with 4 jobs. That is, the i^{th} job's output will be used for the $(i+1)^{th}$ job's input for $i = 0, 1, 2, 3$.

- MapReduce1
  - Input: Zip file
  - Output:
    - key : term
    - value: <docId1,tf1; docId2,tf2,...>
- MapReduce2
  - Input:
    - key : term
    - value: <docId1,tf1; docId2,tf2,...>
  - multiple outputs:
    - Output1:
      - key : term
      - value: <docId1; docId2;...>
    - Output2:
      - key : docId
      - value: <term1,tf1,df1; term2,tf2,df2;...>
- MapReduce3
  - Input:
    - key : docId
    - value: <term1,tf1,df1; term2,tf2,df2;...>
  - Output:
    - key : docId,mold,dl
    - value: <term1,tf1,df1; term2,tf2,df2;...>
- MapReduce4
  - Input:
    - key : docId,mold,dl
    - value:<term1,tf1,df1; term2,tf2,df2;...>
  - Output:
    - key : docId,mold,dl,avdl
    - value: <term1,tf1,df1; term2,tf2,df2;...>

In the 2^{nd} job, we produce two index files, output2 is the input for the next job and output1 is the useful index which will be used for listing a batch of relevant documents for a given term.
The reason why we can do this is that in reduce function, we save all the values with the same key in the index file into a ArrayList variable "cache", then we use two iterations in reduce funtion to produce two kinds of keys and values which will be wrote into two index files.
In java, the way we use multiple output in Reducer is that

- State a mutiple outputs variable "mos" using command "MultipleOutputs<Text, Text> mos";
- Define the variable "mos" in setup function by using
  "mos = new MultipleOutputs<Text,Text>(context)";
- Write to multiple outputs file
  - mos.write("outputFileName1", key1, value1);
  - mos.write("outputFileName2", key2, value2);
- In cleanup function, using "mos.close()" to close the variable "mos";

(b) Compute Score

- MapReduce1
  If only a document has at least one of searched terms, we regard it as a relevant document. Using this idea, then we find all relevant documents by the second index file.
- MapReduce2
  Using the boolean, BM25 or tf-idf algorithm, we can compute all the scores of relevant documents.
- MapReduce3
  Use TreeMap in java to get top 10 results which are put in TreeMap variable in each map.
  - Map
    The key of the TreeMap variable is score and the value is the combination of docId and score seperated by comma. In the cleanup function of this map, we transfer the top 10 results to the reducer by "context.write".
  - Reducer
    Collect all the top 10 results from each map and put them in a new TreeMap variable with size 10. As a result, we can get the top 10 results of the total documents since the TreeMap data structure can help us select the top 10 automatically no matter how large the data collection is.
- MapReduce4
  After we get the top 10 results from last job, we retrieve the content of the top 10 results by this MapReduce process.

(c) Search results showing

- Use ssh2 contained in php to connect with a master in palmetto.
- A java program which calls shell commands is used to "ssh" to one of the node distributed by palmetto system, and excute search task.

SEARCH RESULT

Here shows the search demo
Step 1: input the query

Step 2:Display the search result

Step 3:Click one result to read all of the content.

In order to compare and contrast BM25 and 'tf-idf',search "Nelson Mandela and senior African National Congress associates were jailed for life for resisting apartheid", which is got from the document 5905.Here is the search result:

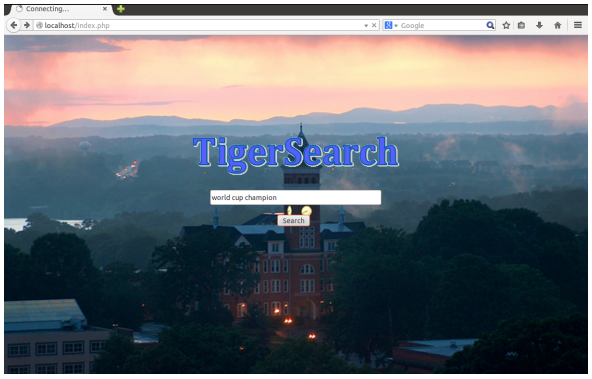

Figure 2. Search Engine

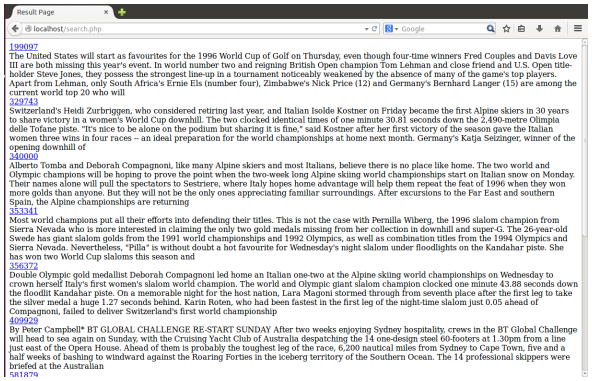

Figure 3. Result Page

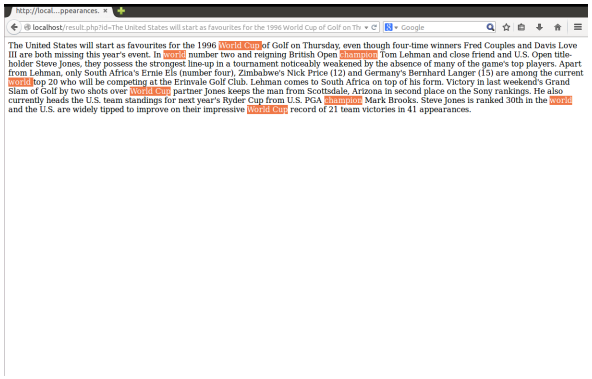

Figure 4. One Content

TF-IDF		BM25	
369017	29.49876649943	5914	0.30306876105
251647	29.63529706331	5905	0.30338657591
110350	29.65262268305	708457	0.30629735444
159664	30.34158454166	437258	0.3072169332
790894	30.5805480715	99392	0.319098426
346348	31.371512389	555486	0.31949260912
666168	31.60889186539	555492	0.330326531
662192	33.12723930106	594346	0.34332221423
34194	33.48011978853	24201	0.35003248515
5905	38.638907894458384	601467	0.3718526900

Using TF-IDF model the document 5905 gets the highest score.Because the query is from this document.However the highest score is not the 5905 tf using BM25. It is because that BM25 model consider the document length. Because of the if saturation, the document 601467 is much longer than 5905, and it is quickly into the tf saturation. After saturation, the longer the document the higher possibility the keywords appears in the document. Then it gets the highest score.

According to the result data and score,BM25 is more precise than 'tf-idf'.However none of their precision rate reaches one hundred percent,because they both not consider the relationship between words

ALGORITHMS COMPARISON

Boolean retrieval is much more different from the 'tf-idf' model and 'BM25'.Its disadvantages discussed above has implied the difference with other twos. Now focus on the difference between 'tf-idf' and 'BM25'.

BM25 VS TF-IDF

Here is a table to demonstrate the difference between BM25 and td-idf

	BM25	TF-IDF
Originates from	Vector space	probabilistic relevance
Performance for short docs	good	better
performance for long docs	better	good
two parameters affect if saturation	don't have	have

Here is the similarity between BM25 and td-idf:

1.Use inverse document frequency to distinguish between common (low value) words and uncommon (high value) words.

2.Both recognize (see Term frequency) that the more often a word appears in a document, the more likely is it that the document is relevant for that word.

PROS AND CONS OF BOOLEAN ALGORITHM

Prons:

* Easy to implement and it is computationally

* enables users to express structural and conceptual constraints to describe important linguistic features

Cons:

* Only documents that satisfy a query exactly are retrieved.

* Difficult to control the number of retrieved documents. *

Traditional Boolean approach does not provide a relevance ranking of the retrieved documents

New Idea

The Divergence from Randomness (DFR) paradigm is a generalisation of one of the very first models of Information Retrieval, Harter's 2-Poisson indexing-model. The 2-Poisson model is based on the hypothesis that the level of treatment of the informative words is witnessed by an elite set of documents, in which these words occur to a relatively greater extent than in the rest of the documents.

On the other hand, there are words, which do not possess elite documents, and thus their frequency follows a random distribution, which is the single Poisson model. Harter's model was first explored as a retrieval-model by Robertson, Van Rijsbergen and Porter. Successively it was combined with standard probabilistic model by Robertson and Walker and gave birth to the family of the BMs IR models (among them there is the well-known BM25 which is at the basis the Okapi system).

DFR models are obtained by instantiating the three components of the framework: selecting a basic randomness model, applying the first normalisation and normalising the term frequencies.

Below is the overall approach DFR used for IR.

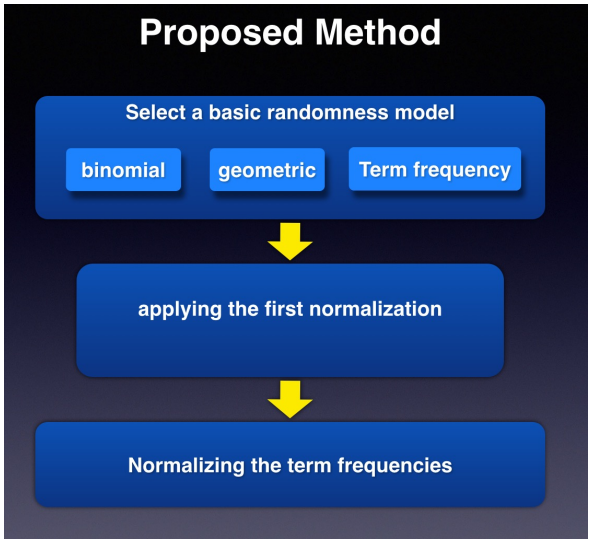

Figure 5. The DFR approach

Basic Randomness Models

The DFR models are based on this simple idea: "The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in the document d". In other words the term-weight is inversely related to the probability of term-frequency within the document d obtained by a model M of randomness:

$$\text{weight}(t|d) \propto -\log \text{Prob}_M\,(t \in d|\text{Collection}) \quad (4)$$

where the subscript M stands for the type of model of randomness employed to compute the probability. In order to choose the appropriate model M of randomness, we can use different urn models. IR is thus seen as a probabilistic process, which uses random drawings from urn models, or equivalently random placement of coloured balls into urns. Instead of urns we have documents, and instead of different colours we have different terms, where each term occurs with some multiplicity in the urns as anyone of a number of related words or phrases which are called tokens of that term. There are many ways to choose M, each of these provides a basic DFR model. The basic models are derived in the following table.

Basic DFR Model	
D	Divergence approximation of the binomial
P	Approximation of the binomial
BE	Bose-Einstein distribution
G	Geometric approximation of the Bose-Einstein
I(n)	Inverse Document Frequency model
I(F)	Inverse Term Frequency model
I(ne)	Inverse Expected Document Frequency model

If the model M is the binomial distribution, then the basic model is P and computes the value1:

$$-\log \text{Prob}_P(t \in d|\text{Collection}) = -\log(\ TF\ tf\ )\, p^{tf} q^{TF-tf} \quad (5)$$

where:
TF is the term-frequency of the term t in the Collection
tf is the term-frequency of the term t in the document d
N is the number of documents in the Collection
p is 1/N and q=1-p

First Normalisation

When a rare term does not occur in a document then it has almost zero probability of being informative for the document. On the contrary, if a rare term has many occurrences in a document then it has a very high probability (almost the certainty) to be informative for the topic described by the document. Similarly to Ponte and Croft's language model, we include a risk component in the DFR models. If the term-frequency in the document is high then the risk for the term of not being informative is minimal. In such a case Formula (4) gives a high value, but a minimal risk has also the negative effect of providing a small information gain. Therefore, instead of using the full weight provided by the Formula (4) , we tune or smooth the weight of Formula (4) by considering only the portion of it which is the amount of information gained with the term:

$$\text{gain}(t|d) = P_{\text{risk}} \cdot (-\log \text{Prob}_M\,(t \in d|\text{Collection})) \quad (6)$$

The more the term occurs in the elite set, the less term-frequency is due to randomness, and thus the smaller the probability Prisk is, that is:

$$P_{\text{risk}} = 1 - \text{Prob}(t \in d|d \in \text{Elite set}) \quad (7)$$

Term Frequency Normalisation

Before using Formula (6) the document-length dl is normalised to a standard length sl. Consequently, the term-

frequencies tf are also recomputed with respect to the standard document-length, that is:

$$tfn = tf \cdot \log\left(1 + \frac{sl}{dl}\right) \tag{8}$$

REFERENCES

1. Information Retrieval. `http://web.stanford.edu/class/cs276/`.
2. Croft, W.B., T. H. L. D. The uses of phrases and structured queries in information retrieval, Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. *pp 10*, 4 (1992), 453–469.
3. Cutting, D.R., K. D. P. J. T. J. S. Where do web sites come from?: capturing and interacting with design history. In *A Cluster-based Approach to Browsing Large Document Collections, Proceedings of the 15th Annual Interna- tional ACM SIGIR Conference on Research and Development in Information Retrieval*, pp (2002), 1–8.
4. Damashek, M. In *Gauging similarity with n-grams: Language-independent categorization of text, Science, Volume 267,* (1995), 843–848.
5. Zellweger, P. T., Bouvin, N. O., Jehøj, H., and Mackinlay, J. D. Evaluation of feedback retrieval using modified freezing. In *Proc. Hypertext 2001*, ACM Press (2001), 9–18.