Web IR Homework1 Report

R02944047 Ya Zhu

2015/4/12

1 Vector Space Machine

1.1 Vectorize

Use TF*IDF value as vector element. Each dimension represents the one term. I have defined two kinds of terms, one is all the unigrams from *vocal.all*, the other is bigrams collected specifically. The reason why I use bigram is that bigram can reveal more topic information than unigram. However, since we are not given the segmentation of all the documents, the amount of bigrams appear in *inverted-file* is too large, and many of the bigrams are useless.

So I tried to collect some representative bigrams by their document frequencies. Moreover, according to my observation, those bigrams appear many times in a few documents may be more typical than those appear once in many documents, so I also considered their term frequencies. Finally, I picked 27727 bigrams as vector terms. As a result, there are totally 29907+27727 dimensions in my vector space. Note that I always set the columns of some stop words as zeros.

1.2 Query vector

As for query vectors, I considered all the text in the query file, expects the stop words and some question words such as "find","relevant" and etc. and the description sentence of irrelevant documents. And set the term frequency of those terms appear in title as 5 times as its frequency and the terms in other tags as 2 times as its frequency, and bigram terms 2 times as unigram.

1.3 Normalization

According to our course slides, normalizing TF can improve the result. So I implement all the normalisation methods mention in the slides. And in the experiment, I tried different methods, and different parameters k and b for Okapi/BM25. The comparison and selection of methods and parameters will be discussed in the third sections. My best result is achieved by Okapi/BM25. normalisation with $k = 2.5, b = 0.5$.

1.4 Similarity

I use *cosine* as the way to compute the similarity of two vectors.

1.5 Impletement

The hardest problem for this part is store and compute the vectors. I use sparse matrix to store the vectors and do computation, which can save a lot of time and memory.

2 Rocchio Relevance Feedback

I define top 20 documents in the result as relevant document, and ignore the non-relevant part because it is less important. According to the experiment result, I set $\beta = 0.8$

3 Experiment Results and Analysis

3.1 Normalize vs. non-Normalize

method	method1	method2	method3
MAP	0.686955929022	0.679794981409	0.747063307688

Table 1: Normalize vs. non-Normalize

Table1. shows the different performance of normalized and non-normalized vector space. Method1 is TF*IDF without normalization, while method2 normalizes TF with max term frequency of each document, and method 3 normalized TF with Okapi/BM25 with k=1.5, b=0.75. From the result, we can see that Okapi/BM25 normalisation has improved a lot for non-normalised vectors, and normalization by max term frequency seems no help.

parameters	k=1.4 b=0.7	k=1.5 b=0.7	k=1.5 b=0.5	k=1.9 b=0.5	k=2.3 b=0.5	k=2.5 b=0.5
MAP	0.7462516	0.7470633	0.6867955	0.7520048	0.7433727	0.7644327

Table 2: Okapi/BM25 parameters

Table2. illustrates some of the parameters that I have tried. From the table we can find that different combination of k and b will have quite different results, but in general, larger k and b can have better result. This means the length of document have great impact on similarity computation, we should eliminate this bias. According to wikipedia, the best k usually in $[1.2, 2]$ and best $b = 0.75$, but in my experiment, I have got a magical pair $(2.5, 0.5)$.

3.2 Feedback vs. non-Feedback

feedback	non-feedback	β=0.5	β=0.7	β=0.8	β=0.9
MAP	0.764432779082	0.764416874404	0.764477792647	0.764509592942	0.764426476871

Table 3: Feedback vs. non-Feedback

From Table3 we can infer that the feedback method improves only a little for the result. This might because the weight of "key term" of query is large enough and the adjusted term weight has little impact on the whole, or because the extra terms and weights from the relevant document bring noise.