Assignment 2 - Retrieval System

Lu Chen (luchen@student.ethz.ch)
Zhichao Han (zhhan@student.ethz.ch) Group **9**
Yeyao Zhang (yezhang@student.ethz.ch) 2 December 2015

1 Overview

We develop 3 term-based models(Naive tf, log tf, and BM25), unigram language model, and Pointwise Online AdaGrad approach to select top 100 documents of each query. We use MIN((TP+FN),100) as denominator when calculating AP. We also perform some methods in preprocessing and running stage to obtain better MAP, as well less running time of the whole program. 2 rounds of scanning are needed in our system. The results(evaluated by MAP) of our approaches are shown in Table1

2 Preprocessing

Content Extraction We note that the content layout of documents in different packages varies a lot. For example, document whose name starting with "AP"has title field, document whose name starting with "WSJ"has headline field. Our target is to extract as much content of each document as possible to generate more precise rankings. Our approach is to extract content with respect to different types of document, then concatenate content extracted from different fields to the text field. These fields and text field as a whole, constitute the content of a document.

Stemming & Stop-words filtering We use PorterStemmer in TinyIR as the stemming module. Words whose occurrences add up to 30% of the number of words in all documents are eliminated as Stop-words.

3 Term-based model

3.1 TF: Term Frequency

Use three different definitions for tf:
TF: Number of times term t appears in document d
Ld: Total number of terms in the document
L: Average document length(Ld) in the text collection

1. Nave frequency
   tf(t,d) = TF / Ld
2. Logarithmically-scaled frequency
   tf(t,d) = 1 + log10(TF)
3. Okapi BM25 frequency
   tf(t,d) = (k1 + 1) * TF / (k1 * (1 - b + b * Ld / L) + TF)
   in which k1 and b are free parameters

Tabel 1: MAP of Different Approaches

Model	Parameters	MAP(without stem)	MAP(stemmed)
Naive tf		0.044	
Log-tf		0.143	
BM25	k=1.2 b=0.75	0.164	0.191
Unigram Language Model	$\lambda = 0.3$	0.158	0.164
Pointwise Online AdaGrad		0.188	

3.2 IDF: Inverse Document Frequency

N: Total number of documents
DF: Number of documents with term t in it
idf(t) = log10(N / DF)

3.3 Ranking Score

$$score(d,q) = \sum_{t \in q} tf(t,d) \times idf(t) \tag{1}$$

3.4 Implementation

First Scan: compute and save df(t), idf(t) for every query and N, L for the collection

Second Scan: compute score for every doc-query pair and save the top 100 docs for every query

4 Language-based model

4.1 Overview

Use unigram language model and linear interpolation smoothing
CF: Number of times term t appears in the collection
Lc: Total number of terms in the collection
TF: Number of times term t appears in document d
Ld: Total number of terms in the document

$$P(t|c) = CF/Lc \tag{2}$$

$$P(t|d) = TF/Ld \tag{3}$$

4.2 Ranking Score

$$score(d,q) = \prod_{t \in q} \lambda \times P(t|d) + (1-\lambda) \times P(t|c) \tag{4}$$

4.3 Implementation

First Scan: compute and save cf(t), p(t—c) for every query and Lc for the collection

Second Scan: compute score for every doc-query pair and save the top 100 docs for every query

5 Learning to Rank model

5.1 Overview

We use point-wise learning approach and formalize the ranking problem as SVM for Ordinal Classification [1]. Then we resort to AdaGrad Algorithm [2] (an online SVM algorithm PEGASOS with geometry adaptation) to train the model.

5.2 Implementation Details

Feature Construction We construct a 4-dimension feature vector for each query-document pair a, in which x1 = logtf-score(d,q), x2 = BM25-score(d,q), x3 = language-0.3-score(d,q), x4 = language-0.5-score(d,q).

Dealing with imbalanced data For each query in training set, if we assign -1 to irrelevant doc, 1 to relevant doc. Then the number of irrelevant docs are overwhelming over the number of relevant docs. So the training set is very imbalanced, meaning that they are much more negative instances than the positive instances, which is harmful for us to perform online SVM(The result will be almost a random split over the space)
We introduce the weight of instances and modify the hinge loss function, specifically modifies the slopes for different instances.
The instance weight for positive instances is $\frac{|pos|}{|neg|+|pos|}$; for negative instances is $\frac{|pos|}{|neg|+|pos|}$.

6 Marvelous Improvement on Efficiency!

At first, the running time of our program is about 7 hours. However, we conduct following methods to decrease it to 30 minutes, thus improve the efficiency of our system.

Hashcode Instead of use string to represent a single word, we use it's hashcode to represent the word. This would greatly increase the speed of searching process.

Vector Instead of using normal List as the collection, we use vector. The reason is that List is more like a tree-structured shape, whereas vector is consecutive, which is better for us to perform the parallelism.

Parallelism In some operations, such as tokenizing and stemming process, and calculation on each query, we use .par method in Scala to convert a List to a ParVector, which could utilize computing resources of our laptops.

7 How to Run our Code?

1. Export the project into eclipse
2. To run the required two scan term-based model & language model, see scala file TraditionalRetrieval.scala.
3. To run the learning-to-rank model, see scala file LearningToRank.scala.

8 References

[1] Shashua A, Levin A. Ranking with large margin principle: Two approaches[C]//Advances in neural information processing systems. 2002: 937-944. [2] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization."The Journal of Machine Learning Research 12 (2011): 2121-2159.