CSE 6240

HW 3

Ke Wang

903058889

1. Rocchio's algorithm

a. We need to analyze the formula to check what does each of these symbols mean o decide their value.

$$\vec{q}_{opt} = \alpha\vec{q}_0 + \frac{\beta}{|C_r|}\sum_{\vec{d}_j \in C_r}\vec{d}_j - \frac{\gamma}{|C_{nr}|}\sum_{\vec{d}_j \in C_{nr}}\vec{d}_j$$

The user wants to "find pages like this", so the original query is not import, we can set alpha to 0. The negative feedback is not import, so it can be set to 0. The positive feedback (the only page that user specifies) is important, so it can be set to 1 (1-0-0=1). All in all, α=0;β=1;γ=0.

b. From the above formula, if we set α=1;β=0;γ=0, then we can get $q_{opt} = q_0$. Generally, if we set α=1, and when the sum of the latter two terms happen to be 0, we can get the same query.

No. For example, if we choose α=1;β=0; and γ to a very large number, say 10000000, then q_0 might be closer.

2. Relevance feedback

a. 1) Relevance feedback could be used to increase recall, but in reality, people mainly concerns about precision.

2) The application of relevance feedback takes more time to give results to the users, which often cannot be tolerated.

3) The "feedback" is subject to users, so there is no ground truth for what relevance feedback is.

b. Positive feedback is more likely to be useful because of user reaction and the way we define "relevant". Users often tend to choose positive feedback rather than negative feedback (users choose several images to be relevant to the query). If relevance is inferred from user clicks. Then in fact negative feedback is hardly applicable.

The work from *Xuanhui Wang et al*.[1] shows that *using non-relevant documents which are close to the original queries is more effective than using all non-relevant documents in the whole collection*. So, using less non-relevant documents is better than using more because often the negative feedback shows the result is far from expectation. We can sometimes refine "less" by saying that using only one non-relevant document. This is because of how user defines 'non-relevance'. Often, the item that a user doesn't choose as 'relevant' is not 'non-relevant' because the user may simply does not notice (the results on the second page) or he skips the item or he has already found what he wants. So, using only one non-relevant document may be helpful. (However, there is no ground truth.)

3. Boosting

1) Boosting is a general method of converting rough rules of thumb into highly accurate prediction rules. AdaBoost is one basic algorithm of this general method. Specifically, the outline of Boosting is give below:

1. given training set (x1,y1),…,(xm,ym)
2. for t = 1…T
3. construct distribution Dt
4. find weak classifier ht with small error
5. output final classifier

Boosting assumes that weak learning algorithm is given, so it does not need to be taken into consideration. But there are still two things in the outline we need to figure out:

1. How to construct Dt
2. How to output final classifier

AdaBoost is a simple algorithm, which gives out the details of these two.

2) AdaBoost is a simple algorithm for Boosting general method. Gradient Boosting is a combination of Gradient Descent and Boosting. In AdaBoost, we try to fix the wrong prediction by modifying the distribution. If the previous prediction is correct, then decreases its weight, else increase its weight. Thus, we can give high weight value to those data we need to take into consideration in the next step. Gradient Boosting tries to improve the prediction by gradients. In addition, AdaBoost is proposed mainly for classification problem (because it is just an algorithm of Boosting). Gradient

[1] Wang, X., Fang, H., & Zhai, C. (2008, July). A study of methods for negative relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 219-226). ACM.

Boost can be used in regression, classification and ranking.

4. I use the following distribution construction method and final classifier as used in AdaBoost:

Constructing Dt:

D1(T) = 1/m

Given Dt and ht: $D_{t+1}(i) = D_t(i)e\text{^}(\alpha_t I(prediction\ for\ i\ wrong))$

$\alpha_m = \log_{10}(1 - err)/err$

Final classifier:

H_final(x) = sign($\sum_t \alpha_t\ h_t(x)$)

I will try three iterations below:

Input training set: (0,1) (1,1) (2,1) (3,-1) (4,-1) (5,-1) (6,1) (7,1) (8,1) (9,-1)

1) each key is assigned weight 1/10

choose v to be 2.5

$$h_1(x) = \begin{cases} 1\ (x < 2.5) \\ -1\ (x > 2.5) \end{cases}$$

error = 3/10

$$\alpha_1 = \ 0.368$$

2) now attach each training pair a modified weight:

(0,1,0.07) (1,1,0.07) (2,1,0.07) (3,-1,0.07) (4,-1,0.07) (5, -1, 0.07) (6,1,0.14) (7,1,0.14) (8,1,0.14) (9,-1,0.07)

choose v to be 5.5

$$h_2(x) = \begin{cases} 1\ (x > 5.5) \\ -1\ (x < 5.5) \end{cases}$$

error = 4/13

$$\alpha_1 = \ 0.352$$

3) now attack each training pair a modified weight:

(0,1,0.1) (1,1,0.1) (2,1,0.1) (3,-1,0.05) (4,-1,0.05) (5, -1, 0.05) (6,1,0.1) (7,1,0.1) (8,1,0.1) (9,-1,0.1)

choose v to be 8.5

$$h_2(x) = \begin{cases} -1\ (x > 8.5) \\ 1\ (x < 8.5) \end{cases}$$

error = 1/5

$$\alpha_1 = \ 0.6$$

4) now combine the results of two iterations

$$H_{result}(x) = \begin{cases} 1(x < 2.5\ or\ x > 5.5 and x < 8.5) \\ -1(x > 2.5 and x < 5.5\ or\ x > 8.5) \end{cases}$$

5) discussion:

The above calculation only keeps two decimal spaces, so there losses precision.

Although only three iterations are used, and there is a loss of precision, the result is good because the training set is simple. If the training set is more complicated, we need to keep more decimals and run more iterations.