
AdaBoost for Query-by-Example in Text

Jonathan Ultis Computer Science & Engr. Dept. (0014)
Univ. California, San Diego
La Jolla, CA 92091-0014
jultis@cs.ucsd.edu

Abstract

This paper describes an implementation of query-by-example, or relevance feedback, for text. The implementation uses Google's search engine to perform a keyword query as requested by the user. If the user requires more information, the user may score documents in the result set as relevant or irrelevant. An implementation of the AdaBoost algorithm is then used choose words that separate the relevant documents from a random document set. Examples of negative document sets are also tested. An example query and refinements of the query is presented. The results seem promising. The system seems to propose new keywords that are sensible to the requested context. Many of the keywords prove useful in constructing new queries. However, refinement using exactly the new terms predicted by the system does not seem to return noticeably better or worse results. This may be the result of an inexact fit between the design of AdaBoost and the capabilities of Google as a back end engine.

1 Motivation for Query-by-Example (QBE) in Text

Today's search engines do a very good job of indexing and retrieving documents by keyword. In my experience with Google over the course of this project, I found it difficult to compose queries that were not answered well by Google's engine [2].

However, keyword queries are still limited by the ability of a user to compose an effective query. In some cases, it is easier for users to provide samples of what they want than it is to formulate a query the will retrieve documents of interest. In addition, user provided positive samples can contain more information about the query than a small set of keywords.

The use of a positive sample set for querying can be considered a query-by-example. That is, the user provides examples of what they really want and the engine tries to find similar documents. There are several different reasons for failure in keyword queries, at least two of which can be improved using additional information from a positive sample set.

First, keyword queries can fail when the desired results have hidden requirements that are not specified in the keywords. An example I will use later in the paper is the query *best guitar players*. My goal in presenting this query to a search engine is to find information on a large number of different guitar players, descriptions of what they played, and an analysis of why they are considered great. These complex conditions can be difficult to

specify through keywords. This results in lower precision when results that match the keywords, but not the hidden conditions, are returned.

The great utility of Google's search engine has been attributed to its ability to correct for two hidden conditions that do not relate to keywords. That is, searchers tend to want documents that other people have found useful or documents that link to results that others have found useful.

The positive example set used for QBE may contain information about the user's hidden requirements on a query. Discovering arbitrary hidden requirements could significantly improve precision.

Second, keyword queries can perform poorly when the vocabulary in the query is overly limiting. In the query above, a guitar can also be called an "axe". A guitar player can be called a "guitarist". Any number of different words could be used in place of "best". As a result, this keyword query effectively selects a subset of the documents that talk about good guitarists. This is the standard information retrieval issue of synonymy which can reduce recall.

A set of positive examples could be used to discover a wider range of vocabulary that deals with the desired topic. Words that separate the positive example set from a random set of documents in the corpus are good search terms for the positive sample set and may be good general search terms for the topic the user wants to locate.

2 Background of QBE

Image retrieval has been a productive source of QBE research. Many images have little or no textual description. It can be very expensive to create textual descriptions for large sets of images, so many people have investigated alternatives to keyword based search for images.

One of the most impressive approaches to image retrieval was introduced by Tieu and Viola [9]. They were able to produce around 14,000 features, most of which reacted well with only a small subsets of images. They then used AdaBoost to select the features that best separated a small positive sample set of images from a slightly larger set of negative examples. The positive examples usually consisted of about 5 sample images, and the negative examples consisted of 100 randomly chosen images.

In their introduction, the authors mentioned the similarity of this structure to natural language. Natural language documents also contain a large feature set, words. Empirical studies of natural language word distribution suggests that words can be used as highly selective features [4] [5]. This suggests that Tieu and Viola's technique for QBE may work for text queries as well.

When dealing with textual information, QBE is usually called relevance feedback. The usual approach is to allow a keyword query, then use the result set, possibly the user's opinion of the result set, to refine the initial query. QBE is also strongly related to text filtering and text classification. Schapire, et. al., proposed AdaBoost for text filtering with favorable results [8].

This problem is a subset of text classification where there are few positive examples, a large feature set, and training must be very fast. These conditions suggest that a support vector machine [6] classifier may also prove effective.

3 AdaBoost Algorithm

The AdaBoost algorithm, introduced by Schapire and Freund, is a method for combining classifiers created by a *weak learner* to create a stronger classifier. The algorithm can be proven to converge to zero error on any training set given a few assumptions about the learner [1].

AdaBoost requires a set of positive and negative examples that can be weighted for importance. In the normal AdaBoost algorithm, weights are all initially assigned the same value. AdaBoost then calls a weak learner algorithm which returns a classifier for the input examples. The classifier is added to the set of all classifiers that have been generated so far and the examples are re-weighted to emphasize the examples that the classifier misclassified. The boosted classifier is simply a combination of all the weak classifiers. Each weak classifier's vote is weighted by the inverse of its error, so better classifiers have more influence.

A weak learner is a learning algorithm that is guaranteed to generate a classifier that is correct on more than half of the examples. It must also generate a wide variety of classifiers so that the class predictions are not artificially correlated.

Tieu and Viola modified their implementation of AdaBoost to spread half of the weight among the positive samples and half the weight among the negative samples to compensate for the disparity in sample size. I felt that my results were better without this correction.

3.1 Word Existence as a Weak Classifier

The existence of a word in a document can be used as a weak binary classifier, usually. Consider an arbitrary set of p positive example documents and n negative example documents. Pick an arbitrary word and use its existence as a basis for classification. Let p_c and n_c be the count of positive examples that contain the word and negative examples that do not, respectively (correct classifications). Let p_i and n_i be the count of incorrect classifications based on the word.

The accuracy of this classifier is given by $\frac{p_c+n_c}{p+n}$. If the accuracy is greater than 0.5, then the existence of the word is a weak classifier for positive samples. If the accuracy is less than 0.5, then existence of the word is a weak classifier of the negative samples. Of course, it is possible for the accuracy to be exactly 0.5, in which case the word can not be used as a weak classifier.

The weak learning algorithm in this system simply examines all words used in the positive and negative examples to see which word provides the classification accuracy that is the farthest away from 0.5. If the accuracy is very low, then the lack of the word is used as the classifier. Otherwise, the existence of the word is used. This search guarantees that, if a word exists that can be used as a weak classifier, then a weak classifier will be returned. This seems sufficient for any practical example.

Ties in classification error were broken by calculating the total TF-IDF score for each tying word across all positive examples. The word with the highest TF-IDF score was used. TF-IDF stands for term frequency, inverse document frequency. The term frequency is the number of times that a word is used in the document and the inverse document frequency is $\log(\frac{d}{d_f})$ where d is the total number of documents in the positive and negative sets and d_f is the number of documents in which the word occurred. This is a fairly standard measure of the importance of a particular term in a particular document. This tie breaking works in favor of terms that are important to documents in the positive sample set.

4 A Practical Application with Google

In order to test the effectiveness of this QBE technique, I created a wrapper for Google's search site. When using the system, the user is presented with a keyword search screen and allowed to enter a keyword query. The first 20 results returned by Google are then displayed. Beneath each result, 5 rating options are presented. Users can click on any result link to view it in a separate window. At any time, the user can rate a particular result on the five point scale with 1 being very good and 5 being very bad. In this implementation, only the distinctions between positive, neutral, and bad are important. The additional options are purely for the appearance of flexibility.

Once the user has evaluated a few documents, including at least one positive example, he may choose to refine the query. When the refine button is pressed, the positively rated results are downloaded, the html tags are stripped out, and the plain text of the pages is extracted. The negative example set, which may or may not include the negatively rated results, is then constructed and the two example sets are passed to the AdaBoost algorithm. The AdaBoost algorithm selects the single words that best separate the two samples and returns them. Those words are then submitted to Google as a new search and the rating process can begin again.

5 Constructing a Negative Example Set

When the user hits the refine button there are two different sources of negative examples.

First, a random sampling of documents on the internet could provide a good source for negative examples since positive matches are unlikely in a random set. Second, the user may have explicitly separated a set of close negatives from the result set.

In the samples below I will report the refinement results for a particular hard query using three different negative sample sets. First, I will use a random sampling from the first 10,000 documents retrieved by a spider started on www.yahoo.com. Second, I will use both the random documents and the negative samples specified by the user. Third, I will use only the negative samples specified by the user.

6 Samples of Query Refinement

I do not have a good metric for testing the results of this refinement mechanism, so I will simply present examples of the refinement mechanism in action. Interested readers can try the system for themselves and form an opinion.

I chose to use the query *best "guitar players"* as an example because I was unable to find a result set that had more than three definite good results in the top 20 returns within 30 minutes of trying different possible queries. This particular query returns two definite good results and either one or two marginal results, depending on an unknown internal state in the query engine.

The marginal result that appears intermittently is a different version of one of the pages returned as a good result. The other marginal result is a discussion of guitar players who are also good singers, so it is slightly off topic but interesting. The other 16 to 17 results were useless message boards with no real content, commercial sites that sell guitar related material, or lists of popular guitar players with no additional justification for their popularity. In the result set I report, only one of the marginal results was returned, so there were 17 negative examples.

I tried the QBE algorithm on the positive results from this query with several different

Random	Random3	R & N	Negative
guitar	guitar	texas	australia
musical	play	concert	unique
acoustic	blues	unique	texas
bit	recording	thousands	buy
recording	robert	unfamiliar	effects
jazz	johnson	flawless	north
guitarist	playing	brilliant	park
play	perfectly	speed	concert
blues	ray	equally	thousands
guitarists	licks	impressive	internet

Table 1: Refinements for each negative example set

negative sample sets, each of which produced a slightly different result. In addition, I added the marginal result and repeated one of the refinements to test the sensitivity of the algorithm to the positive sample set.

One very interesting similarity between all of the above terms is that they are all positively correlated with the positive sample set. This is probably a result of the distribution of positive and negative samples. It is easier to find words that are used in most of the positive samples and few of the negative samples than it is to find words that are used in most of the large number of negative samples and few of the positive samples.

In table 1, each column holds the first 10 words picked by AdaBoost for a particular negative sample set. Random represents the 100 random negative examples. Random3 is the same but includes the marginal positive document in the positive sample. R & N is the random sample set combined with the negatively ranked documents. Negative is the negatively ranked document set alone.

Of the results reported in table 1, only the first refinement actually produced any new interesting documents. The positive result in each of the other samples was one of the documents returned in the original result set. The algorithm does appear to be very sensitive to changes in the positive sample set, which is to be expected since the number of positive samples is so small.

The interesting part of the result set above is the words suggested as refinements for the original query. The words that were selected for the random negative sample seem to provide an expansion of the original query vocabulary, though that changes with the addition of the marginal good result. This result seems to suggest looking for guitarist instead of "guitar player", and possibly looking for several different genre's of guitar music.

The combination of random and negatively ranked examples seemed to produce a vocabulary that helps separate the close positives from the close negatives. That is, it suggests looking for web pages that talk about Texan guitar players who's impressive, speedy, flawless, brilliant, and unique playing drew thousands to a concert, rather than just looking for sites that talk about their brother, the best guitarist in town.

Using the negatively ranked results alone did not seem beneficial.

In looking at the refinements, my intuition immediately suggested several additional queries that might be useful. I tried several queries such as *best guitarists*, *best blues guitarists*, and so forth. Several of these queries produced additional interesting results. Intuition quickly led me to try *flawless brilliant speed unique guitarists*. This query retrieved 7 very good results in the top 20. In this result the first 5 were all good, and only 1 of the

7 was a duplicate of a good result returned by the original query. This was by far the best result of any search I was able to compose.

7 Weaknesses in the Experimental Setup

This experimental setup has two obvious shortcomings. First, AdaBoost produces a list of weak classifiers that should be given weighted votes depending on the error of the classifier. Google does not allow weights on search terms, so that information is lost. Second, it is difficult to develop any meaningful metric for this technique since the search process requires a human aided refinement step and the corpus is not well understood.

The advantage of this setup is that naive users can use the system with little difficulty since it is based on a fairly familiar interface.

8 Conclusions

It is difficult to conclude anything firm about this refinement mechanism because of the lack of a metric. However, using all of the suggested query terms did not seem to significantly decrease performance compared to my original query. In addition, the refinement process did seem to suggest reasonable new keywords. When combined with human intuition, those keywords seem to aid in constructing new queries.

To try the system yourself and form your own opinion, go to <http://24.25.217.158/research/example/index.php>.

9 Future Work

I chose to use 100 random documents as a negative sample because of Tieu and Viola's example. However, I do not believe that 100 random documents provide a sufficient standard vocabulary to allow the weak learner algorithm to make really good choices. I think the results would improve if several thousand negative examples were used instead, but that will require some preprocessing. It may be even better to use a pre-built summary of the distribution for several hundred thousand documents in the weak learn algorithm, but it is hard to calculate classification accuracy from a summary.

It is difficult to measure the performance of this algorithm since the selection of useful documents requires human intervention. It would be useful to create a sample corpus of human rated documents for several different positive sample document sets. Until such a corpus is created, the benefit for any particular change to the algorithm performance can only be established by testing with large groups of naive users. It may be possible to use some standard text classification corpora as a starting point.

It would be interesting to apply this approach to refinement in latent semantic space for the random negative sample set rather than in word vector space [3].

I would like to try using both unigrams and bigrams as features and see how that result compares to unigrams alone.

10 Acknowledgments

Charles Elkan was largely responsible for the system design. He suggested using Google as the search mechanism, implementing the refinement system through the web interface, and generally kept me from biting off too much on this project.

I'd also like to thank Google for the use of their system, although I do not have their permission and am not affiliated with them in any way.

The *Parallel URL Fetcher* project proved very useful for fetching web pages in parallel [7].

References

- [1] Yoav Freund and Robert E. Schapire. *Experiments with a new boosting algorithm*. In Proc. 13th International Conference on Machine Learning, 1996.
- [2] Google. <http://www.google.com>, 2000.
- [3] Thomas Hofmann. *Probabilistic latent semantic indexing*. In Research and Development in Information Retrieval, 1999.
- [4] T. Lau and E. Horvitz. *Patterns of search: Analyzing and modeling web query refinement*. In In Proceedings of the Seventh International Conference on User Modeling. ACM Press, 1998.
- [5] Christopher D. Manning and Hinrich Shütze. Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts, 1999.
- [6] Edgar Osuna, Robert Freund, and Federico Girosi. *Support vector machines: Training and applications*. Technical Report AIM-1602, 1997.
- [7] PUF. <http://puf.sourceforge.net>, 2000.
- [8] Robert E. Schapire, Yoram Singer, and Amit Singhal. *Boosting and rocchio applied to text filtering*. In Proceedings of (SIGIR)-92, 21st (ACM) International Conference on Research and Development, pages 215–223, Melbourne, AU, 1998. ACM Press.
- [9] Kinh Tieu and Paul Viola. *Boosting image retrieval*.