Favorites-Based Search Result Ordering

Ben Flamm and Geoffrey Schiebinger

CS 229 Fall 2009

1 Introduction

Search engine rankings can often benefit from knowledge of users' interests. The query "jaguar", for example, has different meaning for a British car enthusiast than for a Paraguayan park ranger. Here we implemented a naïve Bayes classifier and a cosine-angle similarity metric to sort a set of documents by their relevance to a "bookbag" of user-selected documents. Moreover, we demonstrated that combining the outputs of these algorithms with a support vector machine produces a better estimate of user-relevance than either algorithm does individually.

We used the UCSF Legacy Tobacco Documents Library as our source of documents[1]. This online library contains more than 10 million documents released as a result of federal lawsuits against the tobacco industry. In an effort to overwhelm the library and its users, the tobacco companies flooded the library with millions of documents, including folder covers, blank pages, and duplicate documents. Researchers are therefore presented with the problem of sifting through a vast number of "junk" documents to find references that are suitable to their particular research topic. Since this library is used by many researchers, and is growing daily, adding relevance-based search result ordering will allow the users to more easily discover useful documents.

2 Data Acquisition and Feature Selection

Our first step in acquiring data was to generate "bookbags," or lists of user-selected documents with which we will order the search results. We extracted three sample bookbags from the references of research papers that cited documents from the Legacy library[3, 4, 5]. We also selected 1000 random documents from the library to estimate the baseline word frequencies of the library. We call the bookbags B_1, B_2, B_3 and the set of random documents R.

For each document, we retrieved the PDF version from the Legacy website, stripped the text from the PDF, and generated a list of tokens using the Porter stemming algorithm.

We then combined the token lists for all of the documents, and found the token frequency distribution. We selected the medium frequency tokens as our working set[6], eliminating tokens occurring less than 12 times and more than 2000 times total over the 1500 documents. We constructed a document frequency matrix, D, whose ij th entry is the frequency of word j in document i.

3 Algorithms

Our goal is to provide the user with documents relevant to their interests. To this end, we assign each document s_i, from a set of search results S, a relevance score $d(s_i, \hat{B})$. We compute this score with the aid of a bookbag B, a set of documents that are relevant to the user. We observe (i.e. train on) a portion of the bookbag, $\hat{B} \subset B$, and random documents, $\hat{R} \subset R$. We train a naïve Bayes classifier and vector space classifier on these sets, and generate relevance scores $d_{NB}(s_i, \hat{B})$ and $d_{VS}(s_i, \hat{B})$ for each document s_i in the held-out documents. In our testing, we sorted the documents via an optimal linear combination of these two metrics, which we computed with a support vector machine.

3.1 Naïve Bayes Classifier

We implemented a naïve Bayes classifier using the multinomial event model and Laplace smoothing. Our training set consisted of "positive" examples$\hat{B}$ drawn from the bookbag B and "negative" examples $\hat{R}$ drawn from a set of random documents R (representative of the baseline word frequency of the library). For a document $s_i \in S$ and a training collection of relevant documents $\hat{B}$, we defined the naïve Bayes relevance parameter as the ratio of: [the log probability that the document is in class "0" (not in $\hat{B}$)] to [the log probability that the document is in class "1" (in $\hat{B}$)]. I.e.

$$d_{NB}(s_i, \hat{B}) \equiv \frac{\log P(s_i = 0)}{\log P(s_i = 1)}$$

Note that the more relevant documents have smaller d_{NB} values. This technique is called the *Probability Ranking Principle (PRP)*. This ranking method is optimal in the sense that it minimizes the expected "loss at k", where the loss at k is defined as the number of documents $b_i \in B$ that are not ranked in the top k positions. A proof of this fact can be found in [2].

3.2 Vector Space Classifier

For our next classifier, we treated each document as a vector in a high-dimensional vector space and used the cosine of the angle between two document vectors as a pair-wise document similarity metric. That is, for document vectors d_1, d_2 we computed $\frac{d_1^T d_2}{\|d_1\|\|d_2\|}$, where $\|d\| = \left(\sum_i d_i^2\right)^{\frac{1}{2}}$ is the Euclidean norm of d.

For this vector space classifier, unlike for the naïve Bayes classifer, we used a term frequency-inverse document frequency (tf-idf) normalization scheme for the token frequency matrix, as suggested in [6]. The term frequency is simply the number of times a token appears in a document. The inverse document frequency is the log of 1/[the fraction of documents the term appears in]. The two weightings play opposite roles: term frequency rewards tokens that occur more often in an individual document, while inverse document frequency down-weights tokens common across many documents, which likely play little role in determining the document's relevance. To generate the tf-idf weight, we multiply the tf and idf weights for each token. The ijth entry of D is then

$$D_{ij} = n_{ij} \times \log\left(\frac{M}{m_j}\right)$$

where n_{ij} is the number of times that word j appears in document i, m_j is the number of documents that word j appears in: $m_j = \sum_i \mathbf{1}\{n_{ij} > 0\}$, and M is the total number of documents: $M = \sum_j m_j$.

Now, given two sets of documents $\hat{B}$ and S, for each $s_i$in S we define the vector space relevance parameter $d_{VS}(s_i, \hat{B})$ as the mean pair-wise document similarity over the elements of $\hat{B}$:

$$d_{VS}(s_i, \hat{B}) \equiv \frac{1}{\left|\hat{B}\right|} \sum_{b_i \in \hat{B}} \frac{b_i^T s_i}{\|b_i\| \, \|s_i\|}$$

3.3 SVM Combination of Naïve Bayes and Vector Space Scores

After computing the naïve Bayes (probabilistic) and vector space (non-probabilistic) scores, we seek the optimal linear combination of the two. For each document b_i in $\hat{B}$ we calculated the naïve Bayes and vector space parameters $d_{NB}(b_i, \hat{B}/b_i)$ and $d_{VS}(b_i, \hat{B}/b_i)$. We did the same for each document r_i in $\hat{R}$. Then we used sequential minimal optimization (SMO) to find the optimal margin classifier separating the documents in $\hat{B}$ from the documents in $\hat{R}$ in this two dimensional space (see figure 1, below).

One advantage of this "SVM combo" algorithm is that it can be used to combine any number of ranking metrics. Here we demonstrated its use with two common ranking algorithms, but one could easily imagine adding other ranking metrics such as the mean KL divergence from the documents in $\hat{B}$, or the minimum vector space distance to $\hat{B}$ instead of the mean, etc.

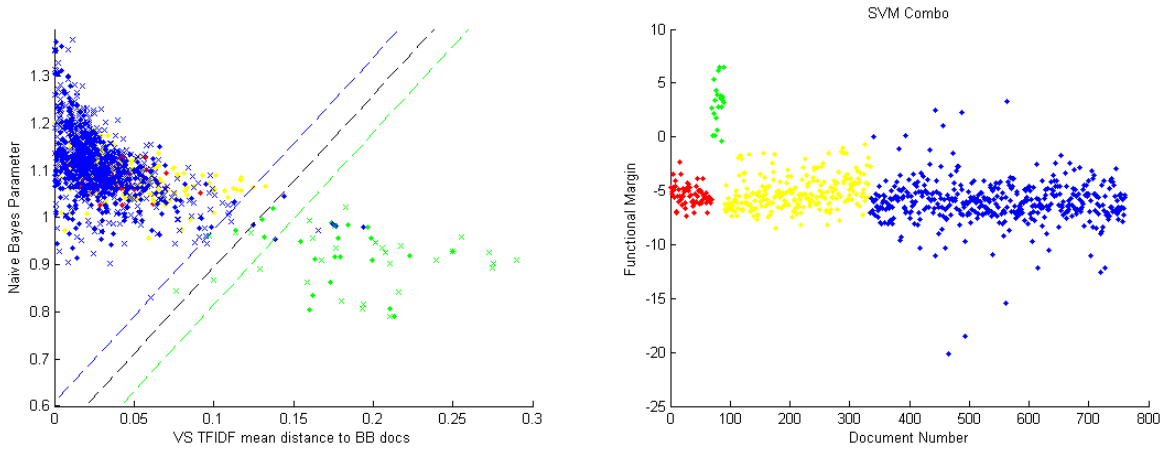

Figure 1. Above are the results of training on 30 documents from B_2 (the green set) and 500 documents from R (blue). We then tested on 50 documents from B_1 (red), 20 from B_2 (green), 240 from B_3 (yellow) and 400 from R (blue). On the left is a 2D scatterplot of the documents in NB-VS score space. Each "x" represents a document in the training set and each dot represents a document in the test set. The value on the x-axis is d_{VS} and the y-axis is d_{NB}. If we were to draw an imaginary horizontal line at y=.7 and ascend vertically, picking points as we meet them, this would amount to ranking the documents by the naïve Bayes parameter. Similarly, if we were to draw an imaginary vertical line at x=.3 and sweep left, this would rank the documents by their vector space score. The dotted diagonal line shows the optimal linear combination of the two parameters, calculated with a SVM as discussed in section 3. The figure on the right shows the functional margin of each point. Note the clear separation of the green documents from the rest, which shows that we separate one set of bookbag documents from both random documents and documents from other bookbags.

4 Results

We tested our naïve Bayes, vector space, and SVM combination classifiers on three bookbags of varying sizes, B_1, B_2, B_3. The first set had 69 documents, the second 51, and the third had roughly 1100. Our methods generated favorable results, particularly for the SVM combination classifier, which regularly returned more than half of the held-out bookbag documents at the top of the ordered results. Below we discuss the results for each method, examine learning curves for the various methods, and note several issues we encountered.

Precision Recall Curve

From the ordered results returned by each algorithm, we generated precision-recall curves.

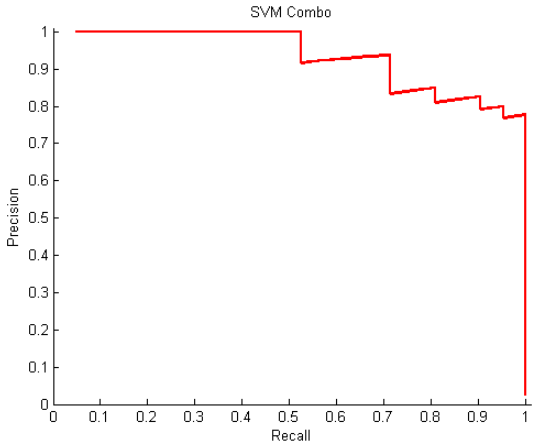

Figure 2. The precision recall plot for the test described in figure 1. Our precision is 1 when our recall is .5, meaning that the first 10 returned documents are all true bookbag documents. Similarly, since our precision is .75 when our recall is 1, all of our 20 bookbag documents are in the top 27 search results.

Learning curves

We generated learning curves by training each method on a varying number of documents, testing on a fixed number of documents, and computing the area under the precision-recall curves.

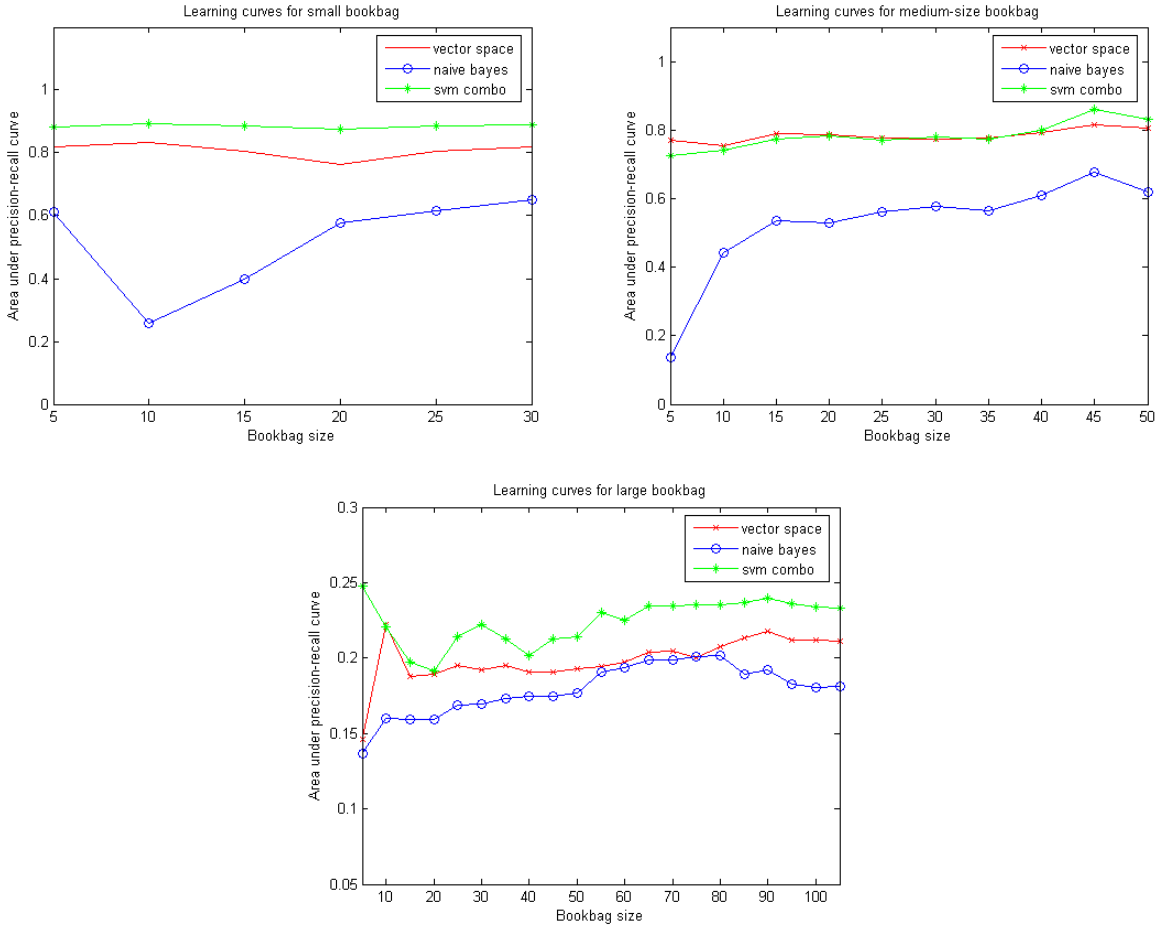

Figure 3. Learning curves for different classifiers. The x-axis is the number of positive bookbag training examples, or $\left|\hat{B}\right|$. The y-axis is the area under the precision-recall curve.

We notice several things in these learning curves. First, the SVM combination is better than either the naïve Bayes or vector space classifiers individually, ignoring slight noise issues for the medium-size bookbag. Second, the learning curve values increase only slightly as the positive training example size increases. This is a good sign, indicating that even if users specify few documents, the classifier returns relevant results. Third, the results for B_3 are much worse than the results for B_1 or B_2. This is probably because B_3 is a much larger collection of documents and has a greater variety of topics. Our algorithm does best with fewer topics in a bookbag (see below). It also could be that the random examples that received high relevance scores (see figure 1, for example) were actually relevant documents.

5 Conclusions and future work

We see that our SVM combination classifier does a good job of returning held-out results for the compact, single-topic bookbags B_1 and B_2. Our algorithm has the advantage of being independent of the size of the

document library (~10 million documents). However, the algorithm does not handle multiple bookbag topics very well, so we naturally cast our collective eye towards more sophisticated methods.

There are a number of algorithms that address these issues that we considered but did not implement due to time constraints. One simple solution, for example, would be to try running K-means clustering (using the cosine angle distance metric) on the document library to uncover about 300-500 clusters. One could then assign high relevance scores to the search results that clustered with the bookbag documents.

Another approach would be to try Latent Semantic Indexing (LSI), i.e. to do a singular value decomposition on the (very large) document frequency matrix, D, representing all the documents in the library. To our knowledge, LSI has not been done on such a large dataset before. However, it might be possible to compute the eigenvalues and eigenvectors of the matrix $D^T D \in \mathbb{R}^{|V| \times |V|}$ (where $|V|$ is the size of the vocabulary). The eigenvectors of $D^T D$ are the right singular vectors of D, and multiplication by D gives the left singular vectors (scaled by the singular value). We can then form a low-rank approximation of D. This way of conducting LSI is independent of the size of the document library.

Yet another approach would be to run Latent Dirichlet Allocation (LDA) on the entire library (or a large representative subset) to represent each document as a mixture of latent topics, and then rank the search results according to how many topics they shared with the bookbag documents.

It would also be interesting to tinker with ways to test and improve the feature vector. This might involve more aggressively testing the effect of changing the lower and upper cutoffs on token frequency. We could also use n-tuples of letters or words as tokens. The problem with this is that the documents in the Legacy library have been OCRed, which injects a lot of noise into the data, were we to look at these features. We also did not include metadata such as author, publication date, or document title. This would likely be a straightforward addition to our SVM combination classifier.

Favorites-based search result ordering has applications beyond the Legacy Tobacco Documents Library. We believe this algorithm can be used to rank results for web search based on user preferences, in a manner similar to Amazon.com's related product display.

References

[1] Legacy Tobacco Document Library. http://legacy.library.ucsf.edu/

[2] Ripley, B. D. 1996. Pattern Recognition and Neural Networks. Cambridge University Press.

[3] Cortese DK, Lewis MJ, Ling PM. Tobacco Industry Lifestyle Magazines Targeted to Young Adults. Journal of Adolescent Health 2009 Sep. 45(3):268-280. http://dx.doi.org/10.1016/j.jadohealth.2009.02.008.

[4] Campbell RB, Balbach ED. Building Alliances in Unlikely Places: Progressive Allies and the Tobacco Institute's Coalition Strategy on Cigarette Excise Taxes. Am J Public Health 2009 Jul;99(7):1188-1196. http://dx.doi.org/10.2105/AJPH.2008.143131

[5] Proctor RN, The Golden Holocaust, unpublished manuscript

[6] Manning CD, Raghavan, P, Schütze H, Introduction to Information Retrieval, Cambridge University Press. 2008. http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf. Ch. 5,6