Multi-Way, Multi-Stage Categorization on Project Gutenberg

CS224N final project, David Borowitz and Fred Wulff

June 8, 2007

1 Corpus/Background

Project Gutenberg is a collection of about 20,000 freely distributable e-texts, most of which are in the public domain. These selection of texts is very diverse, spans a number of languages, and includes scientific journals, horror novels, and the Declaration of Independence. However, there is no good system of categorizing these texts in the manner one might expect in a library or bookstore. The are two impromptu systems of classification: a series of wiki-based bookshelves contributed by volunteers and a catalog that includes Library of Congress classes for some of the texts, which is presumably albeit not explicitly also contributed by volunteers. Neither of these systems has anywhere near full coverage: the bookshelf system includes on the order of 500 books, and the catalog contains on the order of 3,000-4,000 books. Additionally, both of these systems are poorly normalized and poorly defined: the bookshelves include tags such as "Racism", which includes Darwin's *Origin of the Species* since it is "the basis for many racist theories". Therefore, it would be useful if we could engage a supervised learning algorithm to give a first approximation attempt at categorizing the entire set. This approach would of course be useful for tagging the remaining 80% of Project Gutenberg, and can provide useful techniques for tagging of other electronic documents, especialy long books.

For the purposes of this project we used 103 tags over 4,785 of the Gutenberg texts, which were divided randomly into 4,307 training texts and 478 test texts. We applied a Pearson stemmer to decrease the sparsity of our counts as well as to reduce the total vocabulary size. Our feature vectors were simple term-frequency inverse-document-frequency (TF-IDF) vectors calculated from stemmed word counts. The term frequency is defined as the normalized count of a given word within a document: $tf(w,d) = \frac{count_d(w)}{\sum_{w' \in d} count(w')}$. Given a training set $|D|$, the inverse document frequency is calculated as:

$$idf(w) = \log \frac{|D|}{|\{d \in D : count_d(w) > 0\}|}$$

Then the TF-IDF score for word w in document d is simply $tfidf(w,d) = tf(w,d) \cdot idf(w)$. The TF-IDF score was chosen under the assumption that words that only occur in a small number of documents are more likely to be associated with a common category among those documents, and as such should be given more weight.

We wanted to follow the line of the work done by Ghamrawi in "Collective Multi-Label Classification" in exploiting the the interdependence of labels, but due to the size of the data set, memory and performance is at a premium, so we attempted to try to model the relationship in a manner similar to the CML model, but with two distinct classifiers rather than an additional dependence to maximize in the same step, with the hope that this approach would result in a more efficient and customizable algorithm.

2 k-Nearest Neighbor

The first classifier we built was a simple k-nearest-neighbor classifier (kNN). The principle of the kNN classifier is, given n-dimensional real-valued vector space representations of a test text, find the k nearest neighbors using a chosen similarity metric, and use the categories of the nearest neighbors to guess the category of the test text. The outline of kNN is described in Manning and Schütze, chapter 16. The similarity score we used is the cosine similarity metric, which is defined as the cosine of the angle between the two vectors, and is calculated as follows:

$$sim_{cos}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}||\ ||\vec{y}||}$$

The cosine similarity metric has the advantage that it is relatively resistant to features that have similar values across all texts, since all the vectors will lie close to the same hyperplane across these dimensions and hence the angle between them will not be affected.

After computing the cosine similarity of the test vector with each training vector, we created a score for each category c proportional to the similarity between test and training vectors, and then normalized:

$$score(\vec{x}, c) = \frac{\sum_{\vec{y} \in training} sim_{cos}(\vec{x}, \vec{y})}{\sum_{c'} score(\vec{x}, c')}$$

This ensures that the scores for each vector are weighted by the similarity. That is, we want the tags of closer neighbors to be weighted more heavily than those of more remote neighbors. We then hand-picked a cutoff score value α, and assigned document $\vec{x}$ to category c if $score(\vec{x}, c) > \alpha$. There are two factors to consider when choosing α. First, note that the normalization of $score(\vec{x}, c)$ ensures that we can choose α to be a constant; otherwise, α would have to be inferred for each choice of $\vec{x}$ depending on how close its nearest neighbor is. Second, if there are many similarly highly scored categories for $\vec{x}$ and α is sufficiently high, then we will likely not assign $\vec{x}$ to any of these categories. For example, we chose $\alpha = 0.2$. If $\vec{x}$ has 8 equally-likely categories, then their scores will all be 0.125, and we will choose none of them. At first glance, this seems like odd behavior for a classifier, but it reflects the property of the Gutenberg corpus that books rarely (though not never) have more than one tag. In our example, we consider it better behavior to assign no category (decreasing our correct tags by 1) than assign all 8 (increasing our incorrect tags by 7).

There are still some concerns about the kNN algorithm that prevented our implementation from having optimal results. Our main concerns largely stem from the computational difficulties of working with a very large corpus, which we discuss below. One problem is that in large-dimensional vector spaces, it becomes clear that we should weight all features (dimensions) equally. That is, for any given pair of categories, only a very small subset of the features are very useful in distinguishing between them. There have been several approaches to feature selection (Han et al. 1999) based on weighting the features differently, based on pointwise mutual information and other metrics. However, as Han et al. point out, these are relatively computationally intensive, and greatly add to the (otherwise non-existant) training time of kNN. Another large concern with kNN is that the classification algorithm must compute a distance score for every document in the training set, so classification time of a single document is $O(mn)$, where m is the number of training documents and n the number of dimensions. Some approaches that we could consider further involve limiting the neighborhood in which we search for neighbors, but we did not have the resources to implement those.

3 Bayesian Classifier

Taking Hand's advice in "Classifier Technology and the Illusion of Progress" heart combined with our own concern about the tractability of the problem, our first attempt at a classifier was a simple Multinomial Naïve Bayesian Classifier as described in Chapter 13 of *Introduction to Information Retrieval*. The basic idea is that given a document d with feature vector f, we pick the class C that maximizes $P(C) \prod_i P(f_i|C)$, which, thanks to Bayes's Law, is the same as finding the maximum likelihood probability class if one assumes that features are independent. While this assumption is not necessarily valid, it generally works acceptably well. Training takes $O(mnp)$ on average where m is the number of training documents, n is the number of dimensions of the average feature vector and p is the number of classes. In practice, the constants are fairly low. Classification is $O(np)$, both of which are relatively low numbers, and is extremely fast in practice. To prevent the problems incurred 0 count numbers, we use add-one or Laplace smoothing, which starts with an a priori distribution that has each word in the vocabulary appear once in the class.

The main problem that we encountered is simply that many of the tags only have on the order of 1 - 4 books in them. Therefore, in many cases the majority of the features encountered in a test text were unknown within the positive class for that tag, and were entirely dependent on the smoothing behavior. Since add-one smoothing, the de-facto standard for Bayesian models, is generally dependent on assuming that each word in the vocabulary occurs once within the class, it serves as a particularly poor estimate for the many tags that have relatively few actual token occurrences. It's also not outside the realm of possibility that our classifier may have an error in its implementation, although we tested it with a generated data set, and it performed well on that. The next logical step would be to experiment with different smoothing models or pruning the feature set so that the vocabulary contains only particularly interesting words by some metric. However, by this point we decided to focus on our k-NN model that was working well, and shelve the Bayesian model until we come up with a better training set.

4 Optimizations (or: How We Learned To Stop Worrying [About Object Overhead] and Love the Primitive)

For the purposes of this project we used 103 tags over 5,000 of the Gutenberg texts. We created a simple regular expression based program to remove the front matter and back matter added by Project Gutenberg from the texts. We then applied a Porter stemmer obtained from the canonical website[1] and computed and stored the TF-IDF measure for each token. Additionally, we assign the tokens unique identifiers in order to save memory when reading them in. We also create a map between e-texts and tags from the various tag sources for use in training.

A raw copy of Project Gutenberg's entire media collection (excepting some audio) is around 14 GB. After removing duplicates in various formats, and non textual data, the size is 6.7 GB. As tokenizing, stemming, and computing TF-IDF values for the values takes a significant amount of time for that much data, we did those operations in a pre-processing step. Selecting only e-texts for which we had some tag information cut the size of the corpus by about $\frac{1}{3}$, which combined with the size effects of the transformations yielded a final size of a little over 1 GB. Therefore, when we loaded it, our first implementation, which used the CS224N Counter and CounterMap classes, ran into problems with running out of memory. We solved this by using the GNU Trove library, as well as our own modifications of Counter, CounterMap, and a couple of the Trove files to support the requisite operations more sequentially. Not only does switching to primitives save time (over 30% reduction in file reading, and over 50% in the Bayesian classifier's classification function), but it also saves the 8 bytes of Object overhead on each Integer, as well as removing the memory and space concerns present with possibly non-canonicalized Objects. While in retrospect this may seem obvious, much of our time was spent learning the lessons of how to effectively deal with large data sets in Java the hard way, and it was here where we gained a great deal of insight.

5 Multi-round Selection

As part of our investigation into ways to optimize training and classification on large data sets, we came up with a simple method for reducing the dimensionality of the data, which also has the advantage of taking into consideration relationships between categories. We call our classification process multi-round selection, and it can be thought of as a simple example of general multistage classification described in the literature (e.g. Kaynak and Alpaydin 1997). However, our motivation (simplicity, reduction of dimensionality and harnessing of category dependence) and implementation differ considerably from other systems. The principle is that, with slight modification, the output of a classifier can be thought of as a real-valued vector whose dimension is equal to the number of categories in the training data. (The modification is that we omit the very last stage of classification, both in Bayes and kNN, where a real-valued vector of scores or probabilities is converted into a set of binary classifications.)

The first step in our multi-round selection procedure is to run a classifier on the training data, reducing each of the very large TF-IDF-space vectors into a very small category-space vectors. Another approach, which would probably avoid over-fitting, would be to run the first-round classifier over held-out validation data rather than the original training data. Assuming a decently-working first-round classifier, these vectors will naturally cluster around groups of categories with high correlation. The second round is to run a second classifier (which may or may not be the same as the first) over the reduced data. In terms of texts and categories, this step chooses a category for a text based on how other categories with similar first-round classifications are actually classified. Ideally, this has the advantage of reducing system-wide errors introduced by the first classifier. For example, if the first classifier consistently mistags "Children's Literature" as "American History," the second-round classifier will "notice" that texts with a high score for the latter category are in fact frequently the former.

As explained in the following section, the actual results for the second-round classification are approximately equal to, but a bit lower than, the results for only a single round. Although this result may seem underwhelming, the implication is that we can vastly reduce the dimensionality (and hence amount) of our training data without sacrificing much of the underlying structure. As a result, it opens up a whole new avenue of approaches to the classification

[1] http://www.tartarus.org/m̃artin/PorterStemmer/

problem for very large corpora. For example, we could focus more on efficiency in the first round, using some simplifying assumptions, and use a much more computationally-intensive approach in the second round. During this process we could write out the class-space vectors to disk, vastly reducing the computational overhead required for training.

6 Results

Our optimization yielded strong, tangible results. In the end we were able to run both classifiers in 1.5GB of memory, in a reasonable amount of time (¡2 hours for the kNN classifier and ¡15 minutes on Bayes on a reasonably fast machine). Furthermore, our architecture is such that were space at a higher premium, it would be fairly trivial to further optimize it. We broke the initial processing steps out into series of distinct programs, each using its own Java VM, following a map-reduce model that can be easily serialized or parallelized. It's trivial, via an extension of the Counter class that we wrote, to store the result of the first round of classification to disk, so that we could do that part of the process off-line and then respond to additional second-round categories or classification requests without repeating the initial training. Furthermore, if need be.

For our numerical classification resuts, we computed the F_1 score, which weights precision p and recall r equally: $F_1 = \frac{2pr}{p+r}$. We define precision to be the number of correct tags out of the total number of tags assigned by the classifier, and recall to be the number of correct tags out of the total number of golden tags in the test data. These F_1 scores are not directly comparable to multi-way classifiers that only assign a single category to a document, since those are frequently rated on simple classification accuracy.

The results obtained for the kNN classifier are in the following table. We first determined that a k of 20 was optimal, and then compared several values of α, settling on $\alpha = 0.2$:

α	Precision	Recall	F_1 Score
0.10	0.467	0.764	0.580
0.15	0.543	0.728	0.622
0.20	0.680	0.674	0.677
0.25	0.738	0.466	0.571

Note that decreasing the threshold increases recall at the expense of precision, since the classifier will assign far more tags per text at a lower α. Likewise, increasing the threshold above $\alpha = 0.20$ increases precision at the expense of recall, since more texts will not be assigned any tags. Now, some of the best kNN classifiers have accuracy scores upwards of 90% on some corpora. While some of the reason for our low score can be attributed to our somewhat simplistic model, recall that the Project Gutenberg tag data is far from optimal.

The Naïve Bayesian model did not produce usable results on our corpus: in fact, it failed spectacularly, with an F_1 score of 0.041, when trained on our entire set. However, examining the results on tags with more data points, it performed better, and with a generated data set with a tag covering 40 out of 200 documents (with some generated sample noise), it yielded an F_1 score of 0.89, giving hope that with a better training set, it could yield more usable results. It's also possible that it would be useful for second round processing, but due to time constraints, we were not able to investigate that possibility.

For multi-round selection, we ended up choosing kNN for both the first and second rounds. This is primarily because, as noted, the Bayes results are not very good. In principle though, we could substitute in any classifier in either round (as long as it is capable of producing a real-valued vector of outputs). Indeed, since the second-round training step involves first classifying the entire training or validation set with the first-round classifier, in an ideal world we would like to use a much faster classifier for the third round. However, due to time constraints, we did not have the opportunity to test a fast classifier.

α	Precision	Recall	F_1 Score
0.20	0.630	0.711	0.668

Note that we used the same α value that we determined to be optimal in the first-round classification. As the results show, there is relatively little F_1 penalty (only 0.009) for the vast dimensionality reduction achieved in the second stage.

7 Conclusion

The important experience gained from this research is not in the results obtained in terms of classification, but primarily in learning methods for dealing with very large documents. Much of the literature on text classification points out the usefulness of different techniques on, for example, Web page or journal article classification, whereas the documents we use are orders of magnitude larger. (Note that of course there are far more Web pages than Project Gutenberg e-texts; we are simply considering computational cost for a fixed number of training documents.)

In particular, as described above, we spent a lot of time optimizing our data structures to work with large data sets in limited memory. Moreover, perhaps our most important result is showing the viability of a multi-round approach in reducing the dimensionality of very large training vectors. Given more time and the experience we have now, we believe that future work on improving the second-round classifier could improve considerably. However, there is still the problem with the Project Gutenberg training data, in taht user-submitted tags are not nearly as reliable as expert tags (as for example in the Reuters corpus that is a standard for text classification tasks). While the problems with dimensionality can be mitigated and worked around, unfortunately, the data available will probably still put some cap on the ultimate outcome of e-text classification.

8 References

Ghamrawi, Nadia, and Andrew McCallum. "Collective Multi-Label Classification." *Proceedings of CIKM '05* October 31, 2005. http://www.cs.umass.edu/ mccallum/papers/condmultilabel-cikm05.pdf. Accessed 30 May 2007.

Han, E.S., G. Karypis, and V. Kumar. "Text categorization using weight adjusted k-nearest neighbor classification." Computer Science Technical Report TR99-019, Department of Computer Science, University of Minnesota, Minneapolis, Minnesota, 1999.

Hand, David J. "Classifier Technology and the Illusion of Progress." *Statistical Science* 21.1 (2006), 1-14.

Kaynak, Cenk and Ethem Alpaydan. "Multistage Classification by Cascaded Classifiers." Proceedings of the 12th IEEE International Symposium on Intelligent Control, July 16-18, 1997.

Manning, Christopher, and Heinrich Schütze. *Introduction to Information Retrieval.* http://www-csli.stanford.edu/s̃chuetze/i
retrieval-book.html 2 Jun 2007. Accessed 4 Jun 2007.