Abstract

Introduction

algoritms used, features results

Related Work

1. 20newgroups
2. reuters
3. all mnb papers
4. knn papers
5. tfidf papers
6. svd papers

Data Pre-processing

We performed a series of simple transforms on the data in an attempt to normalize it and extract the most meaning. These transforms are applied at the document level, and later transforms attempt to rectify corpus level issues.

Latex Removal

Latex formatting generally takes the form of mathematical expressions expressed between two $ symbols. The primary issue with latex formatting is that semantically similar phrases can take very different and hard to parse appearances. To remove this kind of formatting we first begin by searching the text for all text between two matched $ symbols. Afterwards, we replace all instances of mathematical expressions by the symbole $math123$. The purpose of this expression is to show that the text contains mathematical expressions rather than which exact ones are contained.

Text Normalization

After removing all latex formatting, the text is normalized using standard techniques[cite]. First the entire document is transformed to lowercase, reducing vocabulary size significantly. Next, it is broken up into word chunks by spaces, removing all punctuation in the process. Theses two approaches can reduce the size of the vocabulary significantly. Finally, a Porter stemmer [cite] is employed to reduce words to their base forms. The Porter stemmer will catch issues like different verb tenses and plurals of words. We found that using the stemmer reduced the size of our vocabulary by approximately 25%.

Feature Design and Selection

We considered two sets of features: word count and TF-IDF.

Word Count

This feature vector is simple, consisting of the word counts n_{wd} by word and document. From this set of features we count the instances of words across an entire class. In Naive Bayes we use this to estimate $P(w|c)$ for every word w and class c. Besides normalization of words, there is no secondary selection that happens when using raw word counts.

Perf

TF-IDF

TF-IDF, calculates $n'_{wd} = \log(n_{wd} + 1) \times log(|D|/df(w))$ where $df(\cdot)$ is the *document frequency* of a word, or the number of documents containing a word. The purpose of this transformation is to compensate for the influence of common words and emphasize the influence of rare ones in word counts. While TF-IDF can compensate for rare and frequent words it still doesn't adjust for disparities in document length. We compensate for document length by dividing the document vectors with the average document length, according to the L_2 Norm.

Perf

A second transformation that was explored is a class-length prior distribution for word frequencies. This means that word frequencies for each class were scaled by the total word count within the class. The purpose of this rescaling is to consider the words in a manner purely relative to their own class, rather than in comparison to other classes.

Perf

When dealing with kNN the dimensionality of our vector space becomes an issue. Though the metric accomodates higher dimensions more easily, when faced with the dimensions of the word space, some form of feature selection is required. To this end, multiple different options were considered.

SVD

SVD or singular value decomposition is a technique that will break a matrix M into three matrices U, S, V^T such that $U \cdot S \cdot V^T = M$. The use of SVD in feature selection is that reducing the rank of the diagonal matrix S to k, it will produce the closest matrix M' of degree k to M. By interpretting the feature vectors as a document-feature matrix, or in this case as a tfidf matrix, we can find the best features.

The primary issue with SVD is that allowable python packages for this project do not have very efficient implementations. Calculating the SVD for a matrix of 96000 rows by approximately 88000 columns took approximately 1 hour and required several gigabytes of ram to store the three output matrices. Additionally, since our kNN implementation suffered performance issues adding matrix multiplication to each distance calculation would further reduce throughput.

Random

Random or heuristic approaches can yield interesting results in many situations. In this case, k features were chosen at random and used during kNN. Combined with algorithms such as gradient descent or cross-validation, this can yield interesting results. However, due to time constraints, gradient descent was not implemented and cross-validation did not yield interesting results in the alloted time.

Most Frequent

This amounts to considering the k most frequent features by term frequency. Of course, this approach has several obvious problems. First, and most obvious is the issue of stop words, words which are frequent but carry little meaning. While TF-IDF will rank these lowly, choosing the k most frequent words will ignore these scores. Second, this method of selection assumes that the most frequent words are most classifiable. It ignores situations where two seperate terms may by highly correlated with half of each class but not in the k most frequent words. These issues can be mitigated by filtering out stop words ahead of time and experimenting with values of k but will never completely disappear.

Algorithm Selection

The class of algorithms that are usable during text classification is specific, due to the large feature space. Several types of algorithms have shown good performance in the literature, including: LLSF, Multinomial Naive Bayes, Decision Trees and kNN. Due to ease of implmentation we chose to use Naive Bayes as our baseline algorithm. Naive Bayes is known for high performance in text envirnonments since it's structure allows for easy learning of many features.

Baseline: Naive Bayes using word counts

In order to establish a baseline for future algorithms, the simplest version of Naive Bayes was implemented. Naive Bayes estimates the probability $P(c|w)$ through the use of Bayes Rule.

$$\frac{P(c)P(w|c)}{P(w)}$$

While $P(c)$ can be estimated easily by counting the raw occurences of the label c within the training data, $P(w|c)$ requres more work. We estimate $P(w|c)$ through the following formula:

$$\frac{\sum_{d\in D_c} n_{wd}}{\sum_{w'} 2 \sum_{d\in D_c} n_{wd}}$$

While performing sums is not a very complex task for a computer, calculating the denominator on the entire training set can prohibitive with $O(mn)$ for m features and n documents. To solve this issue, we simply reverse the sums, taking advantage of the sparsity of features within documents. While we keep the same the order-bound, given that a single abstract will not contain more than a few hundred unique words we can expect less iteration in the sums.

Standard: (Naive Bayes with TF-IDF & Ensemble) and kNN with TF-IDF

For our more 'standard' approach, we explored the usage of TF-IDF and other transformations on MBN as well as kNN. Another approach that was considered was Text Weighted Complement Naive Bayes.

For Naive Bayes, rewriting the algorithm to operate on a two dimensional document-feature matrix rather than the actual class based word counts allows us to change the features. By doing so, we can replace the raw word counts with TF-IDF values.

kNN attempts to search for the k nearest points to the query point. The first step then, is to define some metric (distance) function. For this, we used the cosine similarity of TF-IDF vectors.

$$sim(A,B) = \frac{\sum_{w\in A\cap B} A_w \times B_w}{\|A\|\|B\|}$$

Where A, B are TF-IDF vectors. Once the metric function is defined, kNN simply becomes a matter of iterating over the training set to find the k documents with the highest scores. To label a test document, take the most frequent of the labels in the neighborhood and return that.

The primary issue with kNN is that each query will be computationally expensive. Due to implementation, the trick used in Naive Bayes to reduce the side of iterations couldn't be used requiring an inner loop of 80000. This issue is why feature selection and reduction is paramount in kNN, since it will significantly reduce computation time.

### Optimization of kNN

Advanced: Random Forests

Parameter Selection

Normalization Length

Vocabulary size

Testing and Validation

k-fold X-Val

ROC

Confusion

Discussion

kNN performance

Difficulty of TF-IDF

Future Work