Document Classification for Newspaper Articles

Dennis Ramdass & Shreyes Seshasai
6.863 Final Project
Spring 2009

May 18, 2009

1 Introduction

In many real-world scenarios, the ability to automatically classify documents into a fixed set of categories is highly desirable. Common scenarios include classifying a large amount of unclassified archival documents such as newspaper articles, legal records and academic papers. For example, newspaper articles can be classified as 'features', 'sports' or 'news'. Other scenarios involve classifying of documents as they are created. Examples include classifying movie review articles into 'positive' or 'negative' reviews or classifying only blog entries using a fixed set of labels.

Natural language processing offers powerful techniques for automatically classifying documents. These techniques are predicated on the hypothesis that documents in different categories distinguish themselves by features of the natural language contained in each document. Salient features for document classification may include word structure, word frequency, and natural language structure in each document.

Our project looks specifically at the task of automatically classifying newspaper articles from the MIT newspaper *The Tech*. *The Tech* has archives of a large number of articles which require classification into specific sections (News, Opinion, Sports, etc). Our project is aimed at investigating and implementing techniques which can be used to perform automatic article classification for this purpose.

At our disposal is a large archive of already classified documents so we are able to make use of supervised classification techniques. We randomly split this archive of classified documents into training and testing groups for our classification systems (hereafter referred to simply as classifiers). This project experiments with different natural language feature sets as well as different statistical techniques using these feature sets and compares the performance in each case. Specifically, our project involves experimenting with feature sets for Naive Bayes Classification, Maximum Entropy Classification, and examining sentence structure differences in different categories using probabilistic grammar parsers.

The paper proceeds as follows: Section 2 discusses related work in the areas of document classification and give an overivew of each classification technique. Section 3 details our approach and implementation. Section 4 shows the results of testing our classifiers. In Section 5, we discuss possible future extensions and suggestions for improvement. Finally, in Section 6, we discuss retrospective thoughts on our approach and high-level conclusions about our results.

2 Related Work and Overview of Classification Techniques

There have been variety of supervised learning techniques that have demonstrated reasonable performance for document classification. Some of these techniques includes k-nearest neighbor [1], support vector machines [2], boosting [3] and rule learning algorithms [4, 5].

For this project, we focus on related work in the areas of Naive Bayes classification [6, 7, 8], Maximum Entropy classification [9] and probabilistic grammar classification [10].

2.1 Naive Bayes Classification

This subsection cites material from [7] extensively to explain the basics of Naive Bayes Classification.

Bayesian classifiers are probabilistic approaches that make strong assumptions about how the data is generated, and posit a probabilistic model that embodies these assumptions. Bayesian classifiers usually use

supervised learning on training examples to estimate the parameters of the generative model. Classification on new examples is performed with Bayes' rule by selecting the category that is most likely to have generated the example.

The naive Bayes classifier is the simplest of these classifiers, in that it assumes that all features of the examples are independent of each other given the context of the category. This is the so-called "'naive Bayes assumption." While this assumption is clearly false in most real-world tasks, naive Bayes often performs classification very well. Despite this, practical applications of naive Bayes classifier have had high degrees of accuracy in many cases.

The naive Bayes classifier is much simpler and much more efficient than other supervised learning. This is largely because of the independence assumption which allows the parameters for each features to be learned separately. In the case of document classification, number of features is document classification is usually proportional to the vocabulary size of the training document set. This number can be quite large in many cases so the efficiency of the naive Bayes classifier is a major advantage over other classification techniques.

2.2 Maximum Entropy Classification

This subsection cites material from [9] extensively to explain the basics of Maximum Entropy Classification.

Maximum entropy has been widely used for a variety of natural language tasks, including language modeling, text segmentation, part-of-speech tagging, and prepositional phrase attachment [9]. Maximum entropy has been shown to be a viable and competitive algorithm in these domains.

Maximum entropy is a general technique for estimating probability distributions from data. The core principle in maximum entropy is that nothing should be assumed about the probability distribution other than observations during supervised learning i.e. the distribution should be as uniform as possible. A uniform distribution has the property of maximal entropy.

Labeled training data is used to derive a set of constraints for the model that characterize the category-specific expectations for the distribution. Constraints are represented as expected values of "'features", any real-valued function of an example. A variety of hill-climbing algorithms are available to find the maximum entropy distribution that is consistent with the given constraints.

For document classification, maximum entropy estimates the conditional distribution of the category label given a document. Features are defined for each document. The labeled training data is used to estimate the expected value of these features on a category-by-category basis. The hill-climbing algorithm finds a text classifier of an exponential form that is consistent with the constraints from the labeled data.

2.3 Probabilistic Grammar Classification

Probabilistic statistical parsers use information from a tagged training set to create a model for the structure of the grammar in a language. They look at empirical data of what part of speech follows what part of speech in the training data, and can thus make predictions on what it expects to see as it parses sentences whose words are already tagged with their part of speech [11].

One important observation to make about this method compared to the previous ones is that it depends more on the structure of the language, and not on the language itself. For example, the actual content of the words in the sentence matter less than their part of speech in some areas, so this method is looking for ways that the grammar of different sections are structured differently.

For example, consider a news article compared to a sports article. Both articles may be written with the same style (sentence structure) because they are both reporting on the facts of an event or people. If we look at the content (as we do in the classifiers above), we may be able to distinguish them. Sports articles have words like baseball, team, game, whereas news articles have terms like class, explosion, and department.

However, if we compare news and opinion articles, their content may be the same, but the sentence structure may be different. Opinion columns try to be persuasive, so their language is more drawn out and flowery. This is information that can be captured by theses probabilistic statistical parsers, that the above classifiers may miss.

The classifier we use here is Bikel's parser, with settings set to emulate the Collins parser.

3 Approach

3.1 Problem Specifics

Our project focused on classifying articles for the MIT newspaper, "'The Tech."' The following is the list of document category labels in our project:

- `Arts`
- `Features`
- `News`
- `Opinion`
- `Sports`
- `World`

Supervised learning took place on an archive of 3000 articles with 500 articles from each of the 6 categories. These articles were the most recently published 500 articles in each category. The implications of choosing articles from a specific time period are discussed in Section 4.

This set of documents was split into training sets and testing sets randomly every time the classifier was trained and tested.

3.2 Naive Bayes Classification

3.2.1 Probabilistic Model

The probability model for a classifier is conditional model given by $p(C|F_1, ..., F_n)$ over a dependent class variable C representing one of the possible classification labels, conditioned on several feature variables F_1 through F_n. Using Bayes' theorem this can be rewritten as:

$$p(C|F_1, ..., F_n) = \frac{p(C)p(F_1|C)p(F_2|C, F_1)...p(F_n|C, F_1, F_2, ..., F_{n-1})}{p(F_1, ..., F_n)}$$

Naive Bayes classifiers assume that all features of the training documents are independent of each other given the context of the category called the "'naive Bayes assumption" or "'independent feature assumption."' Using this assumption, the conditional probabilities for each feature F_i in the numerator of the above equation become conditional only on C. This means the above equation can be rewritten as:

$$p(C|F_1, ..., F_n) = \frac{1}{Z}p(C)\prod_{i=1}^{n} p(F_i|C)$$

where Z is the normalization factor to ensure the distribution is a valid probability distribution.

3.2.2 Implementation

We used the Natural Language Toolkit (nltk) package to implement our classifier in Python. NLTK contained a Naive Bayes classifier object provided by `nltk.classify.naivebayes.NaiveBayesClassifier`. Naive Bayes classifiers are parameterized by two probability distributions:

- $P(label)$ gives the probability that an input document will receive each classification label, given no information about the input's features.
- P($fname = fval|label$) gives the probability that a given feature (fname) will receive a given value (fval), given that the label (label).

The Naive Bayes classifier in nltk contained a `train` method which took as input a list of `(feature, label)` pairs and calculated the parameters for the aforementioned probability distributions. The nltk framework provides useful methods for outputting testing accuracy and the most informative features which we use to output results in Section 4.

3.2.3 Feature Selection

In all our featuresets we disregarded words in documents which belonged to "English stopword" list that corresponded to a list of generally non-informative common words in the English language e.g. 'to', 'from' etc.

There were 3 featuresets we decided to experiment with:

- **Multi-Variate Bernoulli Featureset**:

  This model specifies that a document featureset is represented by a vector of binary attributes indicating whether each word from the vocabulary occurs in the document or not [7]. An example of a feature is `('contains students'=TRUE)` which corresponds to the word 'students' from the training vocabulary being present in the input document.

  Instead of using the entire vocabulary of words in the training set, a reduced vocabulary can be used for computational efficiency. The vocabulary in our case was reduced by taking the 2000 most frequent words in the training set.

- **Multinomial Featureset**:

  This model specified that a document featureset is represented by the frequency counts of every unique word in the document. An example feature is `'students'=5` which corresponds to the frequency count of the word 'students' in the input document. This corresponds to a 'unigram language model' [7].

- **Normalized Multinomial Featureset**:

  This model specified that a document featureset is represented by the normalized frequency of every unique word in the document i.e. the count frequency of a word in a document divided by the document length. An example feature is `'students'=0.07`. This uses the 'unigram language model' but attempts to correct for the biasing of frequency word counts for longer documents [9]. Since no constraint was placed on document lengths in different categories, this attempts to control for the variable length of documents. It is left to experimentation to see whether this control mechanism is effective or even necessary or desirable.

3.3 Maximum Entropy Classification

3.3.1 Probabilistic Model

The probability model for a classifier is conditional model given by $p(C|F_1, ..., F_n)$ over a dependent class variable C representing one of the possible classification labels, conditioned on several feature variables F_1 through F_n. Using a maximum entropy approach, we want to estimate this probability distribution by learning conditional distributions from labeled training documents [9].

Specifically, in maximum entropy, we place the constraint on the model distribution that every feature must have the same expected value for this feature as seen in the training data i.e. the expected value of features must be the same in the estimated distribution as the empirical distribution.

The desired model distribution can be parameterized by an exponential distribution of the following form:

$$p(C|F_1, ..., F_n) = exp(\sum_{i=1}^{n} \lambda_i F_i)$$

where Z is the normalization factor to ensure the distribution is a valid probability distribution and λ_i are the parameters (weights) to be estimated from training.

These parameters are estimated from training documents using a hill-climbing algorithm such as Improved Iterative Scaling [9].

3.3.2 Implementation

Similar to our Naive Bayes classification implementation, we used the Natural Language Toolkit (nltk) package to implement our classifier in Python. NLTK contained a Maximum Entropy classifier object provided by `nltk.classify.maxent.MaxentClassifier`.

The Maximum Entropy classifier in nltk contained a `train` method which took as input a list of `(feature, label)` pairs which can be used to calculate the parameters for the estimated exponential distribution. The particular hillclimbing algorithm was specified to be `'IIS'` corresponding to Improved Iterative Scaling.

3.3.3 Feature Selection

We experimented with the same 3 featuresets as the Naive Bayes case but with a few modifications forced by the computational complexity of the maximum entropy classifier. We also disregarded the "'english stopwords"' like we did for the Naive Bayes case.

- **Multi-Variate Bernoulli Featureset** We attempted to use the same featureset as in the Naive Bayes case but because of the computational complexity, we needed to reduce the vocabulary size to a size much smaller than 2000 words. We did this in 2 ways. We reduced the vocabulary size to the 500 highest frequency words. This was still computationally very inefficient. A smarter, more efficient way to reduce the vocabulary size was by taking the words from the top 100 most features from the results of the Naive Bayes classifier. This was much more computationally efficient.
- **Multinomial Featureset** We attempted to use the same featureset as in the Naive Bayes case but because of the computational complexity, we needed to reduce the size of the training document set to 300 documents.
- **Normalized Multinomial Featureset** We attempted to use the same featureset as in the Naive Bayes case but because of the computational complexity, we needed to reduce the size of the training document set to 300 documents.

3.4 Probabilistic Grammar Classification

The general approach we took with probablistic grammars was to train different models using training data from each of the six different labels. We could then attempt to parse new articles with each of the six models, and the one that resulted in the highest likelihood would be the classification we would assign to the article.

Specifically, classification was done in three steps: preparing the training data, training the statistical grammar, and parsing the test set of articles.

3.4.1 Preparing the Data

The articles were originally formatted as individual html files within a page template for The Tech's website. We first stripped all of the html from the pages, including removing any template information such as the surrounding header, sidebar, and footer from the page. The resulting file represented only the article's content in plaintext. We also did our best to clean up the encoding of the file,

The next step was to split up each article into individual sentences, as that is the format required of most statistical parsers. To do this, we used the `PunktSentenceTokenizer` that comes with the NLTK package [12].

This sentence tokenizer is based upon the work of Kiss, Tibor, and Strunk [13]. It uses an unsupervised algorithm to find elements of the text that may or may not be sentence boundaries, and builds a model that can be later used to find sentence boundaries on new text. For example, it performs well in detecting abbreviations of words that include a period, but should not end a sentence (such as Dr. or Mr.).

The sentence tokenizer also was set to make a second pass, to check for sentence boundaries that may have been missed due to the placement of particular punctuation. For example, quotations are common in newspaper articles when an author quotes from a source. The style for this is to put the ending period before the right-quotes, which would look like: setence one." Sentence two. While the first pass would split the sentence between the . and ", the second pass would correct it to between the " and S.

Once the sentences were split so that one appeared on each line, each line was edited so that it could be tagged appropriately. This editing was the standard tokenization for the Penn Treebank [15]. The main changes were splitting contractions, e.g. `won't -> wo n't`, normalizing quotation marks, and spacing out punctuation properly.

Old:

```
MIT recently paid $14,000 to a copyright licensing company that found evidence that several
```

```
student groups held publicized, unlicensed showings of copyrighted movies.
```
New:
```
MIT recently paid $ 14 , 000 to a copyright licensing company that found evidence that several
student groups held publicized , unlicensed showings of copyrighted movies .
```

The next step in preparing the data was to tag each token in each sentence with a part of speech. This would enable us to eventually parse the sentence into a tree structure, and eventually use them to create rules for our grammar.

We use the MXPOST part of speech tagger developed by Adwait Rathnaparki [14]. This tagger uses a maximum entropy approach to tag tokens with the standard Penn treenbank tags [15], and has been shown to achieve high accuracy in other settings. In this particular case, it was trained using the standard training set that comes with the mxpost distribution — sections 0-18 of the Wall Street Journal corpus in the Penn Treebank.

The difficulty that can arise here are lexical differences between the Wall Street Journal corpus and The Tech's corpus. It is likely that there is language in Tech articles that are not represented in the training data, so MXPOST will not be able to recognize this. The solution to this would be to enumerate all of these words and add them to the vocabulary of the trained data, which is possible with MXPOST. However, we decided against because we felt that these minor omissions wouldn't have that great of an effect on the resulting structure of the trees made from these training sentences.

Here is that same example sentence, now with part of speech tags:

```
MIT_NNP recently_RB paid_VBD $_$ 14_CD ,_, 000_CD to_TO a_DT copyright_NN licensing_NN company_NN
that_WDT found_VBD evidence_NN that_IN several_JJ student_NN groups_NNS held_VBD publicized_VBN
,_, unlicensed_JJ showings_NNS of_IN copyrighted_VBN movies_NNS ._.
```

The statistical grammar parser that we us here requires the format of these sentences in a slightly different format, so we run the adwait2bikel.pl script provided in the dbparser distribution to convert our sentences to the final format that will be used for the parser.

Sticking with our example sentence, we now have:

```
((MIT (NNP)) (recently (RB)) (paid (VBD)) (()) (14 (CD)) (, (,)) (000 (CD)) (to (TO)) (a (DT))
(copyright (NN)) (licensing (NN)) (company (NN)) (that (WDT)) (found (VBD)) (evidence (NN))
(that (IN)) (several (JJ)) (student (NN)) (groups (NNS)) (held (VBD)) (publicized (VBN)) (,
(,)) (unlicensed (JJ)) (showings (NNS)) (of (IN)) (copyrighted (VBN)) (movies (NNS)) (.  (.))
)
```

3.4.2 Training the Bikel Parser

We chose the Bikel parser to preform the statistical grammar modeling, which includes an implementation of the Collins parser. The goal here is to train six different models of the Bikel parser: one with arts training data, one with news data, etc. We can use the training data that we prepared in the previous section to train the model, and then evaluate them on a test set of articles that we've set aside.

Training can be done with the Bikel parser by giving it a series of parse trees from which it can learn its statistical grammar. From these golden training trees, it is able to extract empirical information about possible sequences of part of speech tags, and develop an internal probibliistic model of what part of speech follows what.

In order to get these training parse trees, we need to convert the sentences that we prepared in the previous section to actual parser trees. These "gold" parse trees will represent the correct structure of the sentences that we wish to model. Ideally one would do this manually, as was done with the Wall Street Journal corpus, but unfortunately time constraints made this option unfeasible.

Instead, we used the model created through the WSJ corpus, and parsed all of our training sentences into trees. We then make the assumption (albeit probably a poor one) that these trees produced by parsing the sentences against the WSJ corpus are the true, correct parse trees. The potential problem here is that the trees may actually just represent the qualities of the WSJ corpus, and not the Tech articles themselves.

The hope is that this effect will be drowned out by the fact that all six models are starting from that same baseline.

Here is an example of what our example sentence parsed as, using the WSJ corpus as training data. This tree was then used as part of the training data for the model for our news parser.

```
(S (NP (NNP MIT))
   (ADVP (RB recently))
   (VP (VBD paid)
       (NP (NP (QP ($ $)
                   (CD 14)))
           (, ,)
           (NP (NP (CD 000))
               (PP (TO to)
                   (NP (NP (DT a)
                           (NN copyright)
                           (NN licensing)
                           (NN company))
                       (SBAR (WHNP (WDT that))
                             (S (VP (VBD found)
                                    (NP (NP (NN evidence))
                                        (SBAR (IN that)
                                              (S (NP (JJ several)
                                                     (NN student)
                                                     (NNS groups))
                                                 (VP (VBD held)
                                                     (NP (NP (JJ publicized)
                                                             (, ,)
                                                             (JJ unlicensed)
                                                             (NNS showings))
                                                         (PP (IN of)
                                                             (NP (VBN copyrighted)
                                                                 (NNS movies))))))))))))))
   (. .))
```

We then run Bike's trainer on this parse tree data, using the standard settings for Collins parser that came with the distribution. The trainer outputs two files: one that contains a list of observations that it made from the input data, and one a file containing serialized Java objects that contains the empirical information from the trained data. This is the file that we can pass to the parser along with our new data that we wish to parse, as we will see in the next section.

3.4.3 Parsing the Test Set

At this point, we have 6 different files, each representing a model for each of our six classifications of articles. Now we'd like to test and see how well each one can successfully parse sentences from each of six categories.

Each model was trained with roughly 480 articles from that respective section. We kept 20 articles aside from each section (so 120 total) to test on each of the models. These 120 articles were tokenized and tagged in the same way that the trianing articles were.

For each of the 120 articles, we then attempted to parse the sentences into parse trees, using each of the models. For each sentence, the parser returned the log likelihood that the parse tree would have formed given the underlying probabilistic model of the grammar it was using. We then totaled the log likelihood of all the sentences in an article to obtain the log likelihood that that article would be formed by that grammar. Thus, for each article, we have six measures of log likelihood, corresponding to the six models created for the six possible classifications. The one with the highest log likelihood is the one we choose as our classification.

Because all of the test data is consistent among all six models, we don't have to worry about the some problems that may arise when using log likelihood as a metric for matching the training set. For example, because the same sentence is used on all six models, the length of the sentence does not matter. We expect

all six likelihoods to be on the same scale, so they are comparable, which is important if we are to suggest that the article should be labeled as the one with highest value.

4 Results

4.1 Naive Bayes Classification

The Naive Bayes Classifier trained very quickly and never gave "'Out of Memory"' errors during the training process which was quite pleasing. The results are shown in Table 1 below:

Table 1: Results from Naive Bayes Classification

FeatureSet	**Accuracy**	**5 Most Informative Features**
Multi-Variate Bernoulli	0.7667	1. contains(column) = True ; [opinion : world] = 173.3 : 1.0 2. contains(federal) = True ; [world : arts] = 67.9 : 1.0 3. contains(columnist) = True ; [features : world] = 65.5 : 1.0 4. contains(minister) = True; [world : news] = 63.4 : 1.0 5. contains(characters) = True; [arts : world] = 61.3 : 1.0
Multinomial	0.6800	1. column = 1 ; [opinion : world] = [134.8 : 1.0] 2. briefs = 2 ; [world : opinion) = [87.2 : 1.0] 3. review = 2 ; [arts : world] = [71.2 : 1.0] 4. columnist = 1 ; [feature : world] = 62.8 : 1.0 5. mit = 1 ; [feature : world] = 47.1 : 1.0
Normalized Multinomial	0.3600	1. column = None ; [world : opinion] = [20.9 : 1.0] 2. tech = 0.0400 ; [arts : news] = [3.7 : 1.0] 3. times = None ; [news : world] = [3.1 : 1.0] 4. girl = 0.0013 ; [feature : news] = [3.1 : 1.0] 5. deans = 0.0009 ; [opinion : news] = [3.1 : 1.0]

As we can clearly see, the Naive Bayes classifier gets the best accuracy in the case where we used a Multi-Variate Bernoulli featureset. The most informative features make intuitive sense considering the categories they were able to distinguish between clearly. For example, the word 'federal' is expected to be much more common in newspaper "'World"' articles since we expect references to federal issues far more often than in the "'Arts"' articles.

The Multinomial featureset is less accurate than the Multi-Variate featureset but not by a large margin. The fact that the most informative features did not contain features with high frequency word counts suggest that the number of occurences of the word in a document is not as important as its mere presence in the document.

The Normalized Multinomial featureset failed woefully suggesting either difficulty in classifying using continuous real-valued feature values by the classifier or more likely, this normalization for length overcorrects

for a variable for which it may not be important/desirable to set a control.

4.2 Maximum Entropy Classification

The Maximum Entropy classifier took very lengthy periods of time to train. This is due to the large amount of features for which the IIS algorithm has to estimate parameters (weights). Given a large training data set, the training process became very slow and computationally intensive causing "'Out of Memory"' errors or the algorithm failing to converge on a good set of parameters resulting in poor accuracy as shown in Table 2 below.

Table 2: Results from Maximum Entropy Classification

FeatureSet	**Accuracy**	**5 Most Informative Features**
Multi-Variate Bernoulli (Small Vocabulary)	0.1767	1. contains(works)==False and label is 'sports' 2. contains(received)==False and label is 'sports' 3. contains(major)==False and label is 'sports' 4. contains(million)==False and label is 'sports' 5. contains(lot)==False and label is 'sports'
Multi-Variate Bernoulli (Naive-Bayes Vocabulary)	0.7233	1. contains(column)==True and label is 'world' 2. contains('04)==True and label is 'world' 3. contains(mit's)==True and label is 'world' 4. contains(federal)==True and label is 'arts' 5. contains(debate)==True and label is 'sports'
Multinomial	0.4483	1. consulting==1 and label is 'feature' 2. professors==1 and label is 'feature' 3. number==1 and label is 'feature' 4. school==3 and label is 'feature' 5. subject==1 and label is 'feature
Normalized Multinomial	0.2414	1. essay==0.0003 and label is 'feature' 2. concept==0.0003 and label is 'feature' 3. dollar==0.0003 and label is 'feature' 4. focus==0.0006 and label is 'feature' 5. illustrated==0.0003 and label is 'feature'

The Multi-Variate Bernoulli featureset with the vocabulary size of 500 performs very poorly. This vocabulary size still seems too big to get accurate and fast parameter estimation with the Maximum Entropy

classifier. As we can also see, the "'most informative features"' found were not very informative at all.

However, we also ran the Multi-Variate Bernoulli featureset with the vocabulary specified by the 100 most informative features given by Naive Bayes classification. This produced very promising results with a 73% accuracy value which is close to the 77% accuracy from just the Naive Bayes classification. Perhaps a similar but slightly more clever way of combining the results of Naive Bayes classification with those of Maximum Entropy classification can actually lead to increased accuracy. Interestingly, one of the most informative features was the word `'04` corresponding to the year 2004. This shows that some of the most informative features can actually be time-dependent.

For the Multinomial and Normalized Multinomial featuresets, the Maximum Entropy classifier performed very poorly and training was very computationally slow with the "'most informative features"' in both being pretty uninformative. This is due to the large amount of features for which parameters have to be estimated.

Overall, the Maximum Entropy classifier provided so much computational difficulty given our featuresets that it made good performance of the classifier almost impossible. In practical cases, it seems that the Maximum Entropy requires very carefully selected featuresets else performance can be very poor. This much is intimated by the authors of [9].

4.3 Probabilistic Grammar Classification

As described above, we created six different grammatical models, and trained out test set of articles against each of them. There were 120 articles in the test set, 20 from each section.

Precision and recall were determined by looking at the number of documents that were correctly labeled for a particular section, compared to the expected value of 20. For example, the precision of the news model is the number of articles that correctly were labeled news divided by all of the articles that were labeled news (0.222 = 4 / 18). Recall was calculated by dividing the number of correctly labeled news articles by all actual news articles in our test set (0.2 = 4 / 20).

Note that these precision and recall numbers do not reveal the quality of parser model itself, but rather its ability to disambiguate articles from its label with other labels. They differ from the precision and recall values that would be returned by the evalb framework that is popularly used to evaluate accuracy of models. The evalb results compare the generated parse tree again a gold parse tree that we expect for the sentence, which as described earlier we don't have for Tech articles.

Thus, what is important is not that a particular section was able to correctly parse a sentence into a nice, grammatical tree, but that it was able to parse it in a way that was consistent with its training data, more so than with the training data in the other models. In a rough sense, the likelihood numbers that the parser was generating was generating how similar the test sentence was to the training sentences, which is interesting as it's a slightly different goal than what these probabilistic grammars are usually used for.

Table 3: Results of testing 120 articles against each of the 6 models, each of which were created with training data soley from that section.

Articles from this Section	**Precision**	**Recall**
Arts	0.15	0.15
Features	0.176	0.15
News	0.222	0.20
Opinion	0.133	0.10
Sports	0.25	0.35
World	0.25	0.25
Total	0.196	0.20

The table shows that this method did not obtain results that are as good as the standard classifiers, such as maximum entropy and naive Bayesian. (Note that by pure guessing, we expect a 1/6, or 0.167 chance, which is actually better than some of our precisions.) This could be due for several reasons. The most probable reason is a lack of training data and test data to properly evaluate this method. The training model could have been done on a much larger set, which might have been able to capture the grammar of the section better. Also, a larger and more careful test set could have been chosen to properly account for the breadth of styles from each of the particular sections.

Table 4: How each of the 120 articles were labeled, after choosing the model which resulted in the highest likelihood value for its parse.

Actual Section	Labeled as: Arts	Features	News	Opinion	Sports	World
Arts	**3**	3	3	2	4	5
Features	5	**3**	4	3	4	1
News	2	4	**4**	3	5	2
Opinion	4	2	5	**2**	3	4
Sports	3	2	1	4	**7**	3
World	3	3	3	1	5	**5**
Total	20	17	18	15	28	20

5 Future Work

While other techniques for document classification could be explored, we believe there is plenty scope for future work with the 3 classification techniques we tried.

More complicated featuresets could be examined for Naive Bayes and Maximum Entropy classification. The featuresets we explored provided a good base for classification but in certain cases, specific features could be added to augment accuracy. For example, the featuresets that work best with newspaper articles in particular can be explored more fully. We saw in some of our "'most informative features"', words specific to different time periods e.g. `'04` and `obama` actually played a big part in helping distinguish between categories. For the classifier to be used for articles of different time periods, corresponding words in that time period can be specified as features e.g. `'92` and `bush`. In this way, time-dependent 'wildcards' can be used as possible features for newspaper articles. Further, some more complicated encoding of popular time-dependent trends into classification features can possibly produce a more accurate classifier for newspaper articles of different time periods.

In the case of Maximum Entropy classification, we saw that intuitive featuresets may not scale very well resulting in computational inefficiency and eventually bad classifier performance. Further work can be done is looking to optimize training on these featuresets or intelligent ways of reducing the dimensionality of these intuitive featuresets to be computationally efficient and hopefully accurate. There is much future work that could be done with the probabilistic grammar parser to perform the classification better. Since classification is not the primary treason that these parsers are used, much of this area has not been explored, either because much better classification techniques already exist, or there is too much variability in using this method across various corpora.

Future work could be done on bettering the model that is created by training the Bikel parser with data from each section. In this particular study, we used the default parameters with the Bikel parser to achieve an environment equivalent to the Collins parser. Modifying these parameters may hvae been able to achieve a higher accuracy for our classification.

Along the same idea, we did not measure the accuracy of each individual model, as we did not have create golden parse trees with which to compare the outputted trees. If further care and study were taken into creating each individual model, either by tagging the training data more precisely or creating a set of gold trees to strive to match, then perhaps the classification would have been better.

Future work could also trying different probabilistic parsers, such as the Stanford lexical parser or the pure Collins parser.

6 Conclusions

In conclusion, we found that with the right feature selection and a large enough training size, we can create a classifier to classify documents with an accuracy of 77% into their respective sections. The best classifier used multi-variate Bernoulli features with the naive bayesian classifier. These results are interesting both in the practical and theoretical sense.

Practically, this can be applied to *The Tech*'s archives to place articles in their specific section, which will improve the value of their website. Although the distinctions may be subtle, there is a difference between

news content and opinion content in terms of how people read the article, and this might alleviate some concerns that readers may have if they read opinion content and take it to be fact.

Theoretically, can we firmly draw the conclusion that articles from these different sections are truly written differently, and thus extract some information that may help writers writing for a particular section? While we can conclude the existence of features that distinguish these articles, it's unclear that these features were a result of the writer, and not the subject content. The first half of this study, which looked at Naive Bayes and Max Entropy classifiers, used the content, which is driven by the world, not by the writer. The second half, looking at the grammatical structure, is driven by the writer, not the world. Given that the former outperformed the latter, this suggests that its really the subject matter that allows us to distinguish content among sections.

References

[1] Yiming Yang. An evaluation of statistical approaches to text categorization. Journal of Information Retrieval, 1999.

[2] Thorsten Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In Machine Learning: ECML- 98, Tenth European Conference on Machine Learning, 1998.

[3] Robert E. Schapire and Yoram Singer. BoosTexter: A boosting-based system for text categorization. Machine Learning, 1999.

[4] William W.Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. In SIGIR '96: Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1996.

[5] Sean Slattery and Mark Craven. Combining statistical and relational methods for learning in hypertext domains. In Proceedings of the 8th International Conference on Inductive Logic Programming (ILP-98), 1998.

[6] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In Machine Learning: ECML-98, Tenth European Conference on Machine Learning, 1998.

[7] Andrew McCallum and Kamal Nigam. A comparison of event models for naive Bayes text classification. In AAAI-98 Workshop on Learning for Text Categorization, 1998. Tech. rep. WS-98-05, AAAI Press.

[8] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. Machine Learning, 1999.

[9] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. IJCAI-99 Workshop on Machine Learning for Information Filtering, pages 61-67, 1999.

[10] Bikel, D. M. 2000. A statistical model for parsing and word-sense disambiguation. In Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (Hong Kong, October 07- 08, 2000).

[11] Jurafsky, D. and Martin, J.H., Speech and Language Processing, 2nd edition, Prentice-Hall: 2008.

[12] The Punkt sentence tokenizer, included in NLTK.
http://docs.huihoo.com/nltk/0.9.5/api/nltk.tokenize.punkt.PunktSentenceTokenizer-class.html

[13] Kiss, Tibor and Strunk, Jan (2006): Unsupervised Multilingual Sentence Boundary Detection. Computational Linguistics 32: 485-525.

[14] Adwait Ratnaparkhi. A Maximum Entropy Part-Of-Speech Tagger. In Proceedings of the Empirical Methods in Natural Language Processing Conference, May 17-18, 1996.

[15] The Penn Treebank Project. http://www.cis.upenn.edu/ treebank/home.html