Automatic Classification of Previously Unseen Proper Noun Phrases into Semantic Categories Using an N-Gram Letter Model

Stephen Patel spatel37@stanford.edu
Joseph Smarr jsmarr@stanford.edu

CS 224N Final Project, Stanford University, Spring 2001

Abstract

We investigate the ability to automatically classify previously unseen proper noun phrases as drug names, company names, movie titles, and place names. The classifier relies on learning probabilistic features of each of the categories, the most important being a linearly interpolated 4-gram letter model of the proper nouns. In addition, prior probabilities, suffix probabilities, phrase length, number of words per phrase, and naï ve bayes over common word(s) in the phrase all contributed to system performance. Overall system performance was at or near that of human trials conducted, typically in the 94-98+% range for most classification tasks.

1 Introduction

1.1 Overview

The performance of many NLP systems suffers because no statistical data has been generated for previously unseen words. This problem is particularly acute in domains where there are large numbers of specific names, or where new names are constantly being generated, such as names of drugs, names of companies, names of places, and so on. Despite having never seen these words before, most people seem fairly capable of identifying them as drug names, company names, etc., which suggests that these classes of words can be modeled using word-level features, such as n-gram letter distributions, suffixes, word lengths and other features. Being able to classify new words in this way would allow natural language text-processing systems to continue unhindered when unknown phrases are encountered. It would particularly aid in the semantic processing of sentences, since assigning an unknown word to a category like “drug names” would impart a great deal of semantic information not present in “unknown word." It could also be useful for automating tasks such as finding stock price data for all companies in a document even though it isn't clear which proper nouns in a document are actually company names.

A possible model for using this classifier would be to augment an existing natural text processing system, which segments and flags all unidentifiable proper noun phrases, and then passes these to the classifier to obtain additional semantic information before performing additional tasks. However this paper focuses entirely on the design and performance of the classifier itself, starting with segmented and tagged proper noun phrases.

1.2 Related Work

There is surprisingly little existing literature on using word-level features to classify unseen words. Some related work in using such features for POS tagging of unknown words can be found in (Mikheev, 1997). There has also been relevant work in the more general field of name-finding and named-entity extraction, some of which makes use of word-level features, for example (Baluja et. al., 2000) and (Bikel et. al,, 1997). The basic research question in our project was to find out how accurately an n-gram letter model combined with other word-level features would be able to classify unseen proper nouns. An interesting characteristic of the problem addressed by this project is that it focuses exclusively on word-level features as supposed to the discourse-contextual information that might be gathered by examining the surrounding words. While many systems examine surrounding context, this project aims to demonstrate how much information is available inherently in the word construction alone without considering the surrounding context.

1.3 Linguistic Assumptions

The linguistic assumption here is that proper names in different semantic categories are constructed from different letter distributions, tend to use different suffixes, and tend to vary in length and number of words. This seems somewhat motivated especially in domains like drug names and company names where people are actively creating new names all the time, and generally want them to "sound" like existing names in that category. Furthermore, empirically suffixes like "Inc." are strong indicators of company names, and there is reason to believe this phenomenon is common across many such categories. A major strength of building a probabilistic model across these features is that it if there is information to be had, it will show up in the distributions.

2 Implementation

2.1 Features Used in Classifier

The classifier initially trains on a set of proper nouns whose categories are known. While training, the system builds up statistical data about the proper nouns in different categories and then uses this information to classify a test set of proper nouns into their respective categories. During training, the system records how many times a particular n-gram sequence occurs in all of the proper nouns of a category. Similarly, the system keeps track of counts for the suffixes, number of words in the phrase, words and lengths for the proper nouns that occur in each of the categories. Thus probabilities are estimated using MLE. After training, small counts are pruned, and entries with 0 counts are smoothed using lidstone's law (λ=0.001). See results section below for a detailed description of the size and nature of data sets used.

The system takes into account six main probabilistic features. The prior probability of a given category is considered and this probability is 'boosted' by an alpha constant. This constant is determined through a linear search of possible values between 0 and 10. The value that works best on a held-out set of data is then used for the test set. Perhaps the most important feature is a linearly interpolated 4-gram letter model. A given proper noun with z letters also has z 4-grams, trigrams, bigrams and unigrams within it (for the purpose of building n-gram counts, the first letter of each noun is artificially preceded by n-1 special ^ markers). As the characters of a word are traversed, the system looks at each of the z n-grams in the word and sums up the log of the interpolated probabilities that the n-gram occurs in the category. A suffix model is also built that is used only for multi-word proper nouns. The suffix model looks at the probability of the last word occurring in a proper noun given a category. We also consider the probability of not having a suffix in a proper noun (i.e. probability of one word proper nouns) given a category. Another probabilistic feature used is proper noun length. This is the probability that a proper noun has a certain number of characters given the category it belongs to. The system also uses a number-of-words model that looks at the probability that a proper noun contains some number of words given the category it is in. In addition, a word model is constructed by taking the naï ve bayes over all words in a proper noun. This simply looks at each of the common word(s) in a proper noun, and takes the sum of the log of the probability that the word occurs in a given category. The linguistic intuition in both the suffix model and naï ve bayes word model is that while there are previously unseen words in each new proper name phrase, there are often also common words that differ with the semantic category.

2.2 Combining Features

For each feature, each category is assigned a score for the likelihood of seeing the unknown proper noun in question. The scores for all of the features are multiplied by a constant weight and then summed together to get a composite score for a category. The system will predict that the proper noun belongs to the category with the highest score. By merely multiplying the class-conditional probabilities for each feature together, an (unrealistic) assumption is being made that the features are class-conditionally independent. This is the same assumption that a naï ve-bayes model makes, and while there is a significant amount of correlation between features we have chosen, each feature does capture some new information about the words, which usually helps boost the system's performance. This is discussed further in our examination of test results.

Formally then, we are using the following probabilistic model (where w is our unknown word/phrase and c is the semantic category):

$$\text{Predicted-Category}(w) = \text{argmax}_c\ P(c|w)$$
$$\text{argmax}_c\ P(c|w) = \text{argmax}_c\ P(w|c)P(c)^{\alpha}$$
$$P(w|c) = P_{ngram}(w|c)\ P_{suffix}(w|c)\ P_{\#\text{-words}}(w|c)\ P_{length}(w|c)\ P_{naive\text{-}bayes}(w|c)$$
$$P_{ngram}(w|c) = \Pi_{i=1..length(w)}\ P(l_i|l_{i-3},l_{i-2},l_{i-1})\ [l_i \text{ is the } i^{th} \text{ letter in } w]$$
$$P_{suffix}(w|c) = P(\text{last word in } w \text{ is } s|c) \text{ or } P(\text{no-suffix}|c)$$
$$P_{\#\text{-words}}(w|c) = P(w \text{ has exactly } n \text{ words}|c)$$
$$P_{length}(w|c) = P(w \text{ has exactly } n \text{ letters (including spaces)}|c)$$
$$P_{naive\text{-}bayes}(w|c) = \Pi_{i=1..\#\text{-words}(w)}\ P(word_i \text{ in } w|c)$$

The system is implemented in java primarily because the built-in data structures and text-token-iterators provided by java helped to reduce implementation time. The perl script eval.pl provided for pp2 was used to evaluate the accuracy of our classifier. A human test program was also created to evaluate human performance in classifying these same proper nouns (see results section below).

3 Experimental Results

3.1 Data Sets

We gathered four data sets of proper noun phrases from different semantic categories that we felt were representative of the domains in which we felt this classifier would be most useful. These data sources are rather "unclean" in that they contain mixes of known and unknown words, inconsistent capitalization, punctuation, and number information, and other "noisy features." However, we consider this a benefit as it leads to a more robust classifier that should be better suited for real-world data.

Table 1. Data Sets Used

Name	Description	Count	URL
Drug	Micromedex 2000 USP Drug Index	6871	http://my.webmd.com/drugs/asset/uspdi_patient/a_to_z
NYSE	Listed Companies on the New York Stock Exchange	3849	http://www.nyse.com/listed/listed.html
Movie	Internet Movie Database (IMDB) listing of 2000 Movies & Videos	8739	http://us.imdb.com/Sections/Years/2000/
Place	Collection of Country, State/Province, and City Names from around the world	4753	http://dir.yahoo.com/Regional/Countries/

Table 2. Typical Examples from Each Data Set

Drug	NYSE	Movie	Place
Ibuprin	Capital Trust, Inc.	Aeon – Countdown im All	Culemborg
Oragrafin Sodium	FOREST LABS INC	Cielo cade, Il	Gairloch
Presun Sunscreen for Kids	Sensient Technologies Corporation	What Lies Beneath	Rwanda
Q-Tuss	The R.O.C. Taiwan Fund	Wolverine	Valenciennes

3.2 Overall Performance

Final performance testing consisted of a set of six "1-1 trials" in which two categories had to be classified between, four "1-all trials" in which each category had to be separated from the conjunction of the other three, and one "4-way trial" in which the classifier had to make a 4-way choice among all categories. Each training set consisted of 90% of the relevant combined data sets, 10% of which was held-out for parameter turning and then trained on, and the remaining 10% of the combines data sets was tested on. The following results are for raw classification accuracy using all available features:

Table 3. Classifier Performance with all Features

1-1 Trials	
Trial	**Score**
Drug-NYSE	98.881%
Drug-Movie	94.875%
Drug-Place	94.406%
NYSE-Movie	98.887%
NYSE-Place	98.721%
Movie-Place	88.955%
1-1 Average	*95.787%*

1-all Trials	
Trial	**Score**
Drug-Others	95.722%
NYSE-Others	99.252%
Movie-Others	91.987%
Place-Others	90.325%
1-all Average	*94.321%*

4-way Trial	88.971%

These data suggest a number of interesting conclusions. First, classification accuracy is extremely high, suggesting that using word-level features to classify unseen words is a viable task. Note that this evaluation is on 100% off-training-set examples, i.e. it has never seen any of the test examples during training. This means that the extracted information about letter distributions, suffixes, length, and so are truly responsible for the performance.

As expected, the 1-1 trials appear to be easier on average than the 1-all trials, because the categories are more homogeneous, and thus tighter models can be learned (in the 1-all tests, the "all" category is a conjunction of three real categories, so it should be expected that features like n-grams and length get "diluted"). The 4-way test is obviously more stringent, since there is more potential for error (chance would be 25%), yet performance remains high. In general, it appears as though company names were most easily identified, followed by drug names, then movie titles, and place names were the most difficult. This is a reasonable result, since company names have a number of distinctive features (like the extremely common "Inc." and "Corp" suffixes), whereas place names have relatively few identifying features, and have an impure letter distribution, since they were gathered from all around the world, and thus are in many different languages. Movie titles also had a large number of foreign entries, which probably explains their difficulty.

We regard the 1-all trials as the most relevant to "real-world" applications, since they are analogous to the scenario mentioned above in which a large number of proper noun phrases have been extracted from text, and the goal is to pull out all the names of a given semantic category. We were surprised at the performance of the 1-all trials, despite the fact that the all category is not homogeneous, but we attribute it to the fact that there is still one distinct category per trial, and that the compound categories still capture the identifying information in each of their component categories.

3.3 Human Performance

In order to benchmark the performance of our classifier, we tried to ascertain how good people are at this task. To measure this, we created an interactive version of the classifier in which examples from the 1-1 trials were presented one at a time and human subjects labeled the categories (we chose the 1-1 trials because they seemed the easiest, and least confusing). The test consisted of 6 trials (one for each 1-1 trial), each with 100 examples. The computer of course did over 1000 examples per trial, but humans have limited patience, and even doing 100 examples usually took around 2-4 minutes to complete, compared with roughly 10 seconds by the computer on a data set more than 10 times as large. We ran three subjects through each of the 6 trials, and recorded individual and average performance.

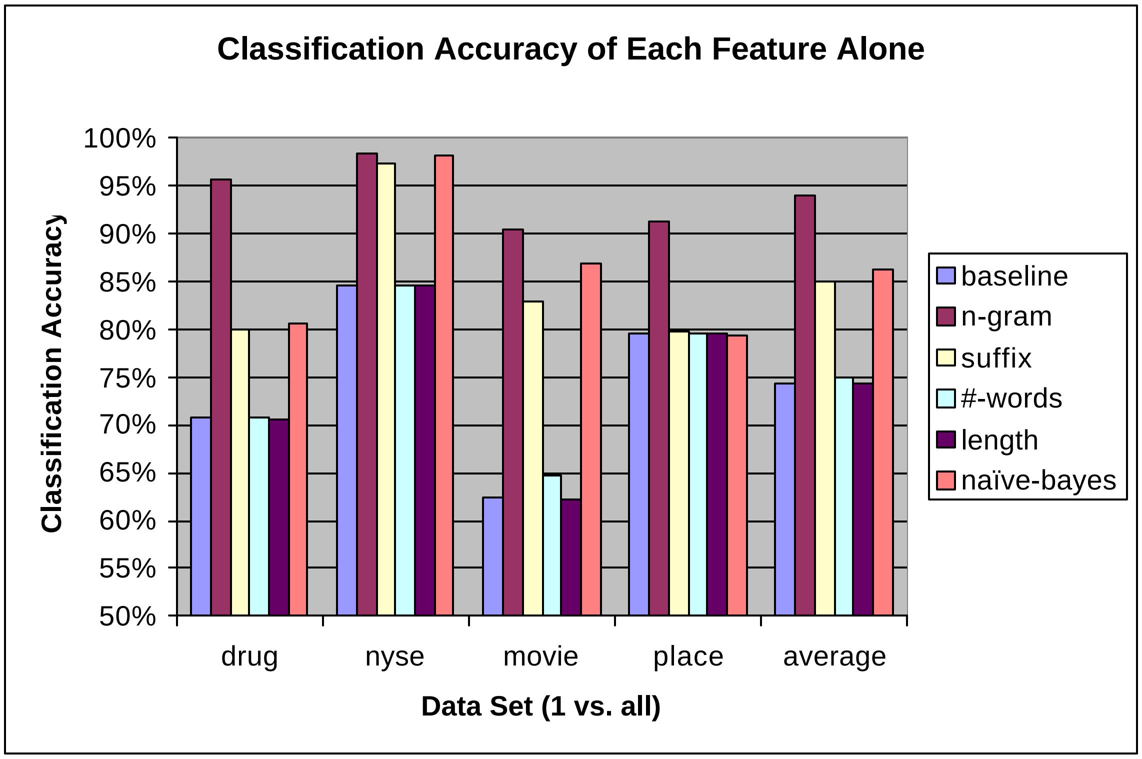
Table 4. Human and Computer Performance

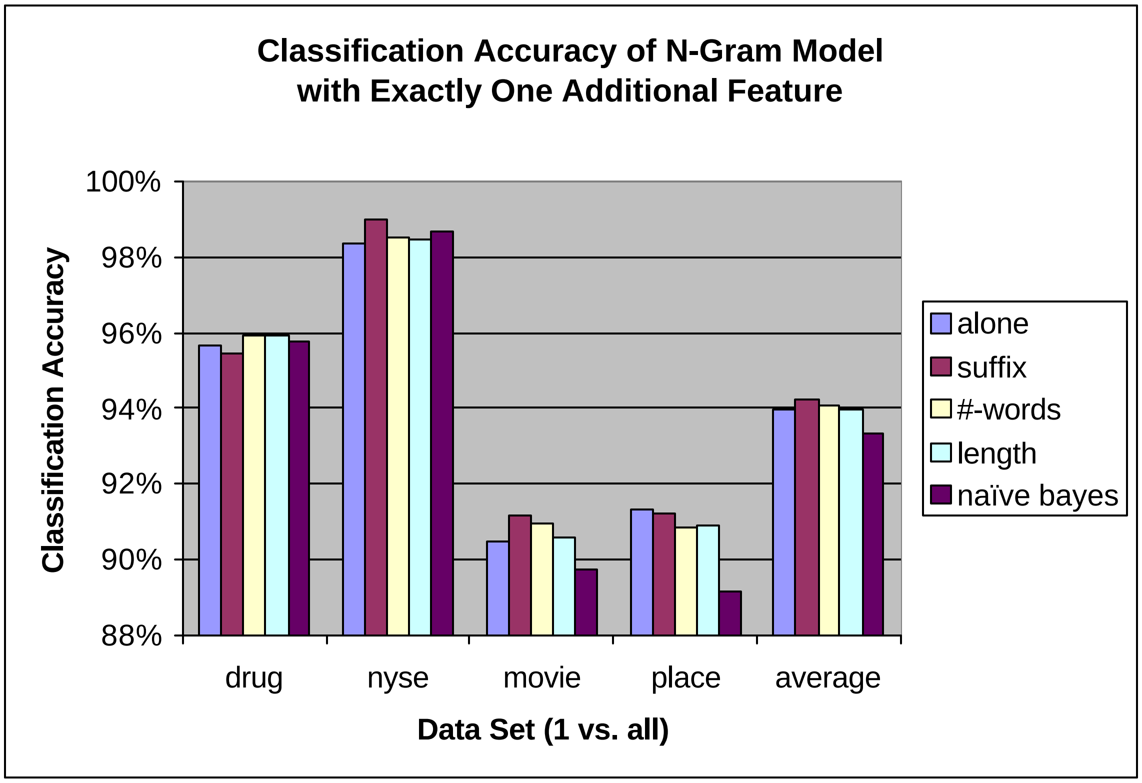
1-1 Trial	**Average Human Performance**	**Computer Performance**	**Difference (C – H)**
Drug-NYSE	97.667%	98.881%	+1.214%
Drug-Movie	93.000%	94.875%	+1.875%
Drug-Place	87.000%	94.406%	+7.406%
NYSE-Movie	97.000%	98.887%	+1.887%
NYSE-Place	93.333%	98.721%	–0.612%
Movie-Place	93.667%	88.955%	–3.712%
Average	*94.444%*	*95.787%*	*+1.343%*

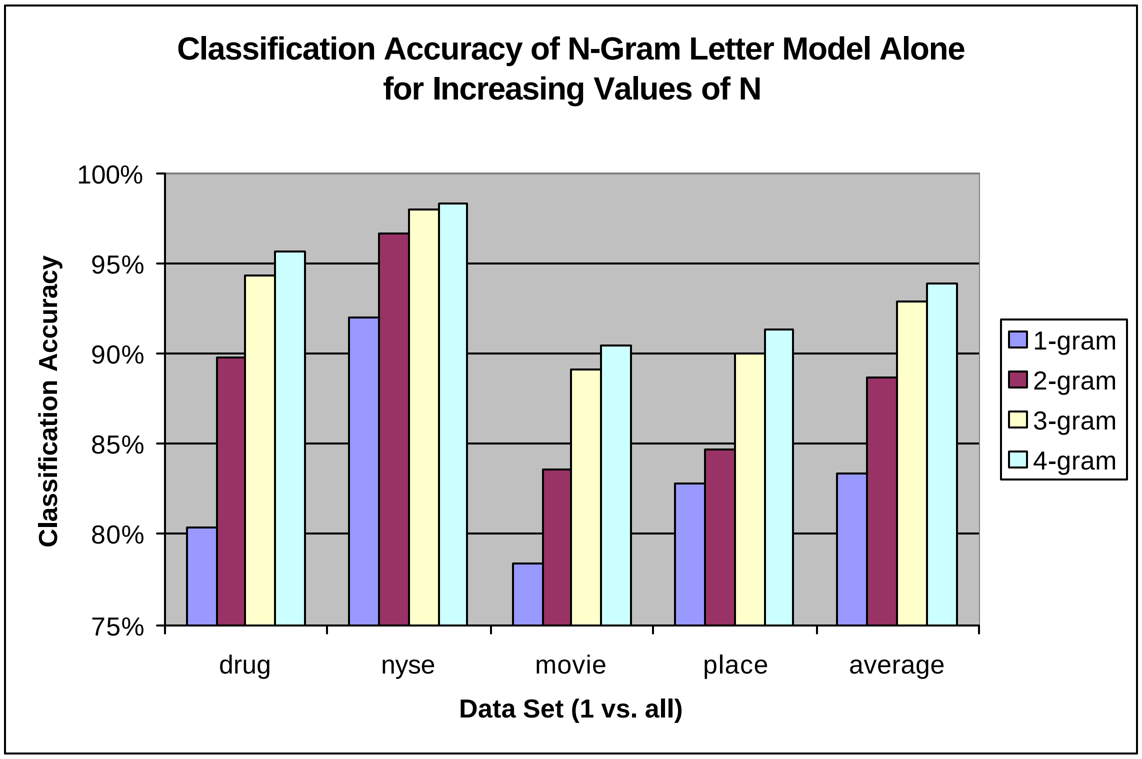
While one should always be somewhat skeptical of an N=3 survey, there seems to be a clear trend in human performance, and in fact the results across subjects were very consistent. It is humbling to realize that in average performance, the computer is winning against Stanford students by 1.3%, which at this level of performance means that it's making roughly 20% fewer errors (though the set of human and computer errors are not necessarily the same). The difficulty of the different trials also seems to show through in both human and computer trials. The two notably unequal scores are drug-place, where the computer dominated (subjectively that was a hard trial because there were lots of one-word place and drug names that looked weird and generally confusing), and movie-place, where people won out decisively (subjectively that was a trial where "higher-level semantic information" like what types of names tend to be for movies vs. places seemed to help a lot). We feel it is fair to claim that our classifier performs at or near the human level of competence.

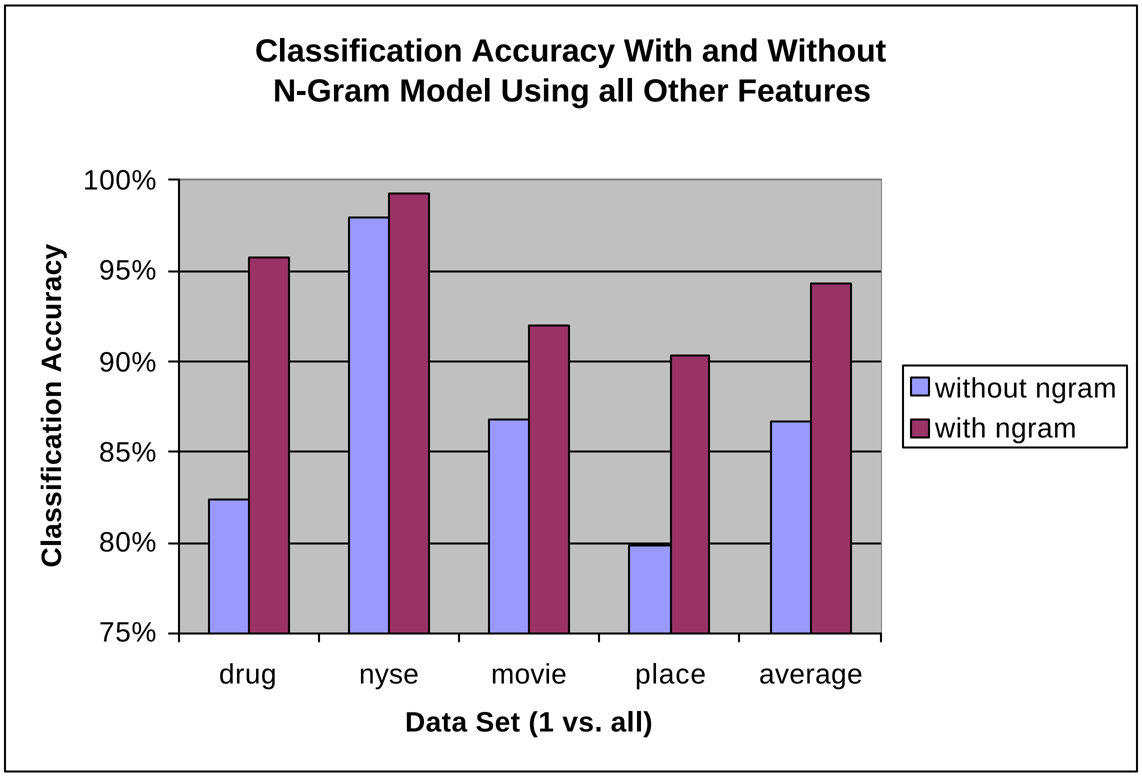
3.4 Performance Breakdown

Our next set of experiments were designed to discover how the different features we used in the classifier contributed to overall performance. To test this, we first ran the 1-all trials using each of the features in isolation. Since the n-gram letter model was clearly the most powerful single feature, we next reran the trials with n-gram and exactly one other feature, to determine which gave the biggest boost. We also tested the performance of the n-gram model in isolation using different values of n (1 through 4) to see how much use was being made of longer-distance letter dependencies. Finally, we tested performance using all features *except* the n-gram model, and for comparison, we also ran a baseline model that only used prior information, though this statistic is overly optimistic, since the "all" category was always the largest and thus the "recall" of the "1" category was always 0.




Classification Accuracy of N-Gram Model
with Exactly One Additional Feature
Classification Accuracy
100%
98%
96%
94%
92%
90%
88%
drug
nyse
movie
place
average
Data Set (1 vs. all)
alone
suffix
#-words
length
naïve bayes


Classification Accuracy of N-Gram Letter Model Alone
for Increasing Values of N
Classification Accuracy
100%
95%
90%
85%
80%
75%
drug
nyse
movie
place
average
Data Set (1 vs. all)
1-gram
2-gram
3-gram
4-gram



For each feature in isolation, the n-gram letter model was by far the most powerful feature, followed by the naï ve-bayes word model, which reflects the relatively distinct sets of common words found in each category (since no information about the unknown words themselves is factored into the naï ve-bayes score). This suggests that additional discourse context would only improve the classification accuracy.

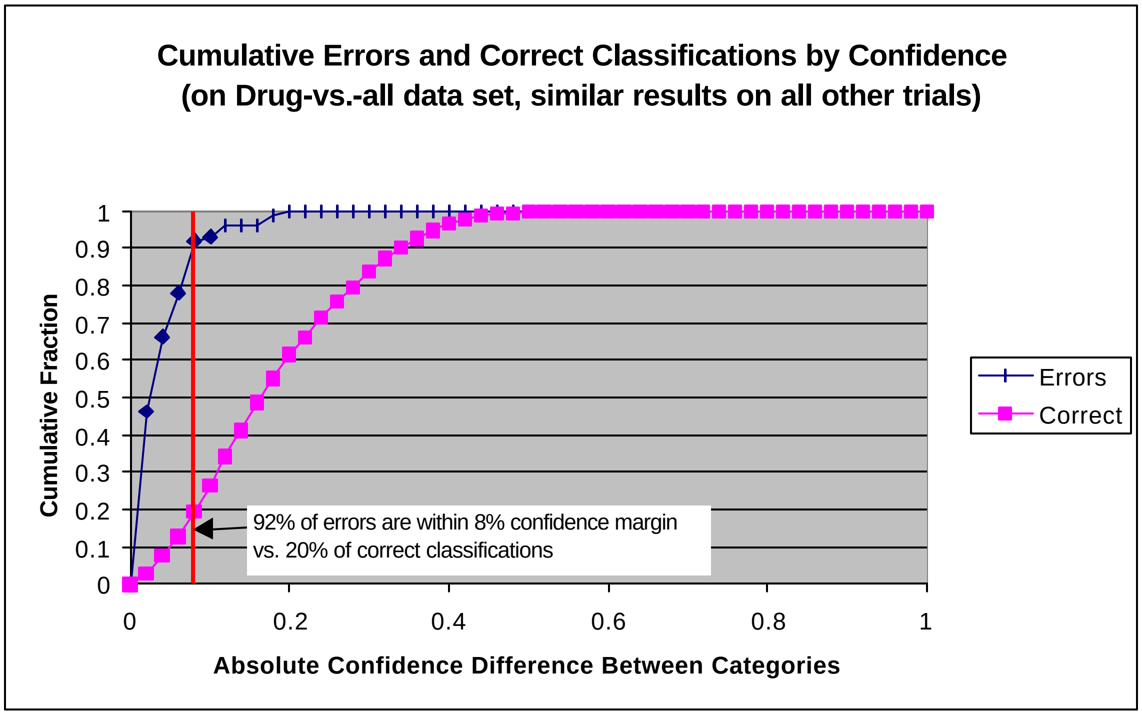
The most powerful additional feature on top of the n-gram model was the suffix model, followed by the #-words and length models. Interestingly, the later two are less captured by the n-gram model, but since suffixes tend to be such giveaways, capturing this information explicitly proved very useful. Notice that using the naï ve-bayes model with the n-gram model actually hurt performance vs. using the n-gram model alone.

When using the n-gram model in isolation, moving up from a unigram model through a 4-gram model resulted in a clear monotonic increase in performance, suggesting both that there are useful long-distance letter dependencies and that data sparseness was not a serious problem (this is not too surprising, since there were only about 70 distinct characters, as opposed to tens of thousands of distinct words in an n-gram word model). The slope of the performance-vs-n curve is clearly leveling off, suggesting that moving to a 5-gram or higher model would likely provide marginal improvement, and could clearly introduce sparseness problems that might hurt performance.

Performance using all features except the n-gram letter model was in general less than using the n-gram model alone, suggesting that the other features have useful information, but miss a lot of the value the n-gram model has. This is further supported by the observation that when using all other features, adding in the n-gram model produces a significant boost in performance. The n-gram model clearly captures some of the information in the other features (except word length), which helps explain why it performs well on its own, and why adding the additional features results in modest improvements, despite the fact that the other features without the n-gram model do fairly well. In other words, performance with the features together is less than the sum of performance of the features apart, because they contain overlapping information.

3.5 Confidence Thresholds

Our final experiment considered how "confident" our classifier was in the decisions it made. We define confidence as the relative difference in scores between the two categories, so a confidence of 1 is complete certainty, a confidence of 0 is random guessing, and a confidence of 0.5 means the most likely category had 75% of the total score over all categories. As an illustrative example, we took each test sample in the drug-others 1-all trial and plotted the confidence of each of the classifier's answers for its correct guesses and incorrect guesses.

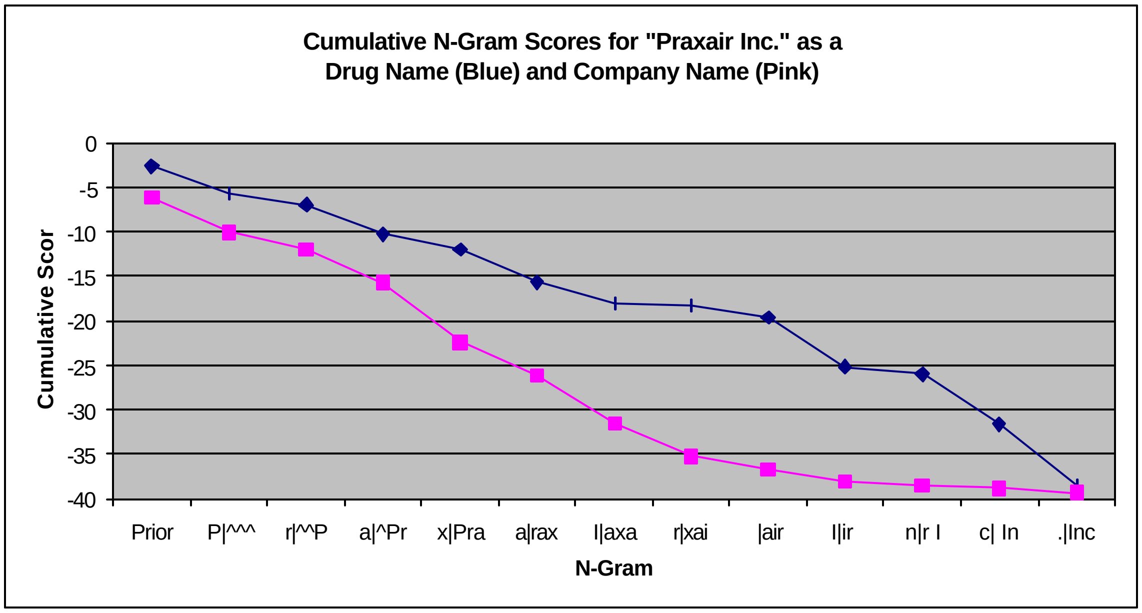


This plot reveals that our classifier is "aware of its own limitations"—all of its errors are below a confidence of 0.2, meaning it has less than a 60-40 belief in them. Every answer that it gives with greater than 20% confidence is indeed correct. However, 60% of the correct answers are also within the 20% confidence threshold, meaning that if we desired 100% precision, we would have to settle for 40% recall. The good news is that if we lower our confidence threshold to 8% (representing a 54-46 chance), we still avoid 92% of the errors, and yet we get 80% of the correct answers that we would normally get.

In other words, if this system were being used to alleviate the work of "human classifiers", it could flawlessly take care of roughly half of the work, leaving the rest for a human, but tagged with a highly accurate "best guess", and if absolute precision weren't critical, it could take care of 80% of the work with 99+% accuracy, leaving only 20% for the human, of which the classifier's best guess is still over 80% accurate. This suggests that not only is the classifier extremely accurate overall, but it could be used in conjunction with a human supervisor, and it would be able to flag most of its errors ahead of time while still preserving the vast majority of its correct decisions.

3.6 Design Revisions and Alternatives Explored

Our original model was a pure n-gram letter model, and we were surprised by its performance, but it also seemed to make a fair number of "dumb" mistakes, like calling names that ended in "Inc." drugs when the company name sounded "drug-like". For example, below is the cumulative ngram-by-ngram log-likelihood score for the name "Praxair, Inc." in the Drug-NYSE 1-1 trial. Notice that through the completion of "Praxair" (which looks like a drug name), drug is clearly winning (the score closer to 0 represents the higher likelihood), and then when it encounters "Inc.", drug takes a nosedive, but it is not quite enough to decide that this name is indeed a company.



The problem is that since "Praxair" is longer than "Inc.", the divergence early on cannot be overcome, despite the fact that it clearly "knows" that "Inc." goes with companies. This motivated our use of an explicit suffix feature, which proved to be very useful. Looking at the remaining errors, we noticed that there were clear differences in the canonical length and number of words in the names from each category, which we also realized the n-gram model couldn't really capture, since it's a local feature. Thus we added those features, which also led to improvements, as we have shown above. Finally, we realized that the usefulness of the suffix feature was really a specific example of the more generally useful information in common words surrounding the new words including prefixes like in "North Haverford" or middle words like in "Alconefrin Nasal Drops 12". Thus we constructed a naï ve-bayes word model, which rounded out our feature set. We elected to keep the suffix feature, since it had extra location information which we felt there was no point in ignoring.

One optimization that we originally considered but elected not to use in the final version is having separate log-multiplier weights for each feature. The intuition is that just as we "boosted" the prior by giving it an exponent (which turns into a multiplier in log-space), we could learn exponents for each feature during cross-validation that would represent how useful the different features were on the particular data set. We decided to use maximum-likelihood on the held-out data as a criterion function for gradient ascent to learn the weights, but abandoned this idea in the end, based on (1) its lack of probabilistic justification (it's essentially inserting multiple copies of the same feature and then asserting their independence!), and (2) because experimentally it added considerable training time without producing a meaningful improvement in performance.

Another idea we tried to implement but ended up not using was EM to learn interpolation parameters for our n-gram. While we still learn the lambdas on-line on held-out data, we do this in a way that parameterizes all the lambdas by a single "relative probability share" variable on which we conduct an inexpensive line search, but ideally we could treat the lambdas as hidden states in an HMM and learn them with EM. We got this basically working, but we kept finding the weights all going to the single n-gram model with the highest overall expectation (which turned out to be the 4-gram), despite the fact that empirically this weighting performed more poorly on the data than a hand-coded mixture weighting, or our optimal parameterized lambdas. We left the code in a function that we never call, so that hopefully if someone noticed an obvious flaw with our implementation, we could resurrect this feature, and perhaps extend it to learn multiple lambdas for different history counts, which might improve the quality of the n-gram model overall.

4 Conclusions

Using an n-gram letter model with other word-level features to classify previously unseen proper noun phrases into semantic categories achieved what is apparently human-level performance (with an average classification accuracy on 1-1 trials of 95.8% vs. 94.4% for average human accuracy), and did not make any errors with more than a 60-40 confidence. We were able to effectively learn a 4-gram letter model for each category, which out-performed tri/bi/uni-gram models. The n-gram letter model accounted for the majority of the performance, though adding suffix, #-words, length, and naï ve-bayes over common words as features all resulted in improvements. This classifier could be used in conjunction with a general-purpose information extraction or natural language understanding system, and could impart categorical semantics upon otherwise unrecognizable proper noun phrases, which are currently troublesome for many systems. It could also be used to alleviate the work of manual classification, and would be able to automatically handle the majority of cases correctly, while flagging the few uncertain cases, including most of its potential errors, for a human to referee.

Despite the impressive performance of this system, there is always room for improvements and future research. For example, using EM to learn finer-grained interpolations within the n-gram model could potentially boost performance. We also think it would be interesting to try using the generated category models for the general task of named-entity extraction, since it seems plausible that being able to recognize proper names would help segment them in free text, and might provide information missing in traditional extraction systems, which mainly consider discourse-context over word-level features.

References

Mikheev, A. (1997). Automatic Rule Induction for Unknown Word Guessing. *Computational Linguistics* vol 23(3), ACL 1997. pp. 405-423

Bikel, D.M., & Miller, S. & Schwartz, R. & Weischedel, R. (1997). Nymble: a High-Performance Learning Name-finder. *Proc. ANLP-97,* pages 194-201.

Baluja, S., Mittal, V., & Sukthankar, R. (1999). Applying machine learning for high performance named-entity extraction. *Proceedings of the Conference of the Pacific Association for Computational Linguistics* (pp 365--378).