CS224N Final Project: Movie Title Recognition in E-Mails

Albert Lai
alai24

June 3, 2009

Abstract

For this project, a system was designed to first identify whether or not an email mentioned a movie, and if it does, to extract the title, time and date of the movie in question. The classifier used to determine whether an email is a "movie" or "non-movie" is an extension of the Naive Bayes classifier. The classifier was fairly successful in terms of precision, although it tended to yield a lot of false positives. A named entity recognition system using MEMM was utilized to tag movie titles, locations, addresses, dates, and times. The NER system saw much less success than the classifier, although some labels like 'time' did fairly well. The dataset used to train both the classifier and the NER turned out to be fairly small so with more training data the system could see some improvement.

Introduction

The ability to extract information from emails has much utility; after all, the motivating concept behind email is to transmit information from one to another. There already exists a feature in Gmail where events can be detected automatically, and the user can possibly to add the event to one's calendar with the "when," "what," and "where" information already filled in. A similar feature that would be very useful would be a system that uses Named Entity Recognition to detect when a movie is mentioned in an email. This knowledge can be handy in displaying possible movie times and related reviews, removing the need for a user to go out and search for that information on their own. The system can also be seen as a sort of targeted advertisement for ticket vendors. With the addition of being able to detect dates and times in the email that mentioned a movie, the proper movie times and locations can be presented.

Previous Work

Google already has an event extractor in place in Gmail that identifies location time and events. Julie Black and Nisheeth Ranjan worked in event extraction using ANNIE, a nemed entity recognizer, for tagging and the RAPIER (Robust Automated Production of IE Rules) algorithm for information extraction.[1] Also, Shashank Senapaty worked on a similar system using simple Naive Bayes to determine whether an email was an event or not, then using MEMM to gain structured event information with some limited success.[2] With the a more narrow scope, the use of a more complicated form of Naive Bayes, and clever feature engineering, this system hopefully performs better.

Dataset

The dataset used for this project was derived from my personal email account. For the classifier, 1000 emails were selected randomly using a Python script to be included in the dataset. All emails were preprocessed using a python script. The preprocessor turns elements like html links into single tokens to reduce the number of distinct words being counted by the classifier as well as stripping html markup. Additionally, a simple stemmer was used to get word roots. The stemmer removes common suffixes like "ing" and "ly" in addition to the letter "s" and "es" The letter "s" is not removed when these emails were hand classified as "movie" or "non," for training purposes. As it turns out, there were only around 6 of the 1000 samples that contained a movie title. Clearly this dearth of emails labelled "movie" would not be enough for training an NER or even the classifier.

Consequently an additional 50 emails were labelled and used for training the NER as well as used in the classification training. These emails were retrieved by going through emails starting from the most recent and back, keeping the ones that mentioned movie titles. These emails were clearly not randomly chosen, but given the limitations of my own email, they were suf-

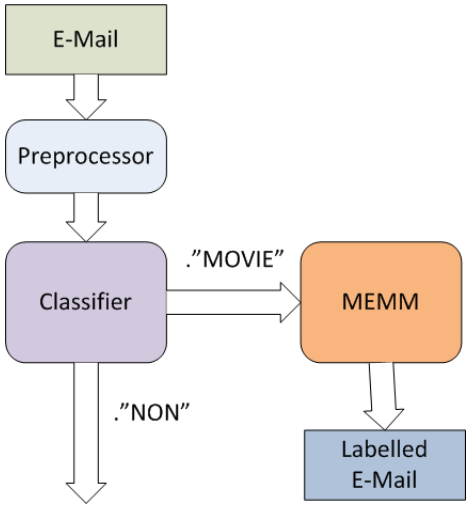

Figure 1: Diagram of Movie title extraction system

ficient for training purposes and nearly exhausted all emails in my inbox that mentioned movies. Details on how the data was used in training are found in future sections.

Because NER is applied to emails labelled as "movie," and those labelled "movie" are the only emails with movie titles in them anyway, only the emails with the "movie" label were used in NER training. The emails were annotated with the following tags: title, date, time, locale.

Overall System

There are two main components to the overarching system: the classifier and the NER. Other components like the preprocessor are merely there to help the classifier by reducing the number of unique tokens. The diagram in Figure 1illustrates the system.

Classifier

Naive Bayes

The first classifier implemented was the Naive Bayes classifier. The classifier estimates the probability that an email is has a certain label given a set of features. Naive Bayes assumes all features are independent, resulting in a vastly simplified equation as follows:

$$P(C|F_1, F_2, ..., F_n) = P(C)P(F_1, |C)P(F_2|C)...P(F_n|C)$$

The features used in this classifier is the presence of a word in the email, and the estimates for their conditional probabilities is calculated from their Laplace-smoothed relative frequencies. Thus

$$P(F_i|c) = \frac{n_{i,c} + 1}{N_c + T}$$

where $n_{i,c}$is the number of times word i appears in emails of class c and $N_c$is the total number of words in emails with class c , or $\sum_i n_{i,c}$. T is the total number of unique words in all emails. The model made no assumptions regarding the likelihood of an email being labelled one way vs another, so $P(C) = 0.5$ for both values of c. The label was given to whichever value of c had the greatest probability given the features.

Table 1: Average Metrics for Naive Bayes Classifier

Accuracy	0.96	0.01
Precision	0.72	0.14
Recall	0.79	0.25

Naive Bayes Analysis

Using Naive Bayes worked out surprisingly well in properly classifying the emails. As mentioned earlier, a small percentage of the emails were truly labeled "movie" so a higher precision was considered to be more important than a higher recall. This is because in the practical usage, it would be more beneficial to accidentally classify a email as having a movie so the NER can be used than to miss one completely.

The classifier was run and cross validated with 1/5 of the data randomly chosen to be the test set. Then, the classifier was run 10 times, each with a new test set. The results show that Naive Bayes had a precision of 72% and a recall of 79% on emails labeled "movie." However, the standard deviation on these scores were fairly high - 14% for precision and 25% for recall. This indicated that the performance of the classifier varied with the different test sets and has a high variance. Consequently, decreasing the number of features may help mitigate the large variance.

Another interesting point is a result of the fact that there are many more "nonmovie" emails than "movie." Since we are using every seen word as a feature, a bulk of those words will be ones seen in "nonmovie" emails since there are there are almost 1000 emails labeled "non" and only around 50 labeled "movie." Naturally there will be more unique words in the set of 1000 compared to the set of 50. Thus when we apply Laplace smoothing, for the features $P(F_i|C = "movie")$, the counts for word i will be zero. Consequently, shorter emails or emails with fewer words will usually have a higher score than an email with many words, since the more words there are the more parameters $P(F_i = True|C = "movie")$ which are close

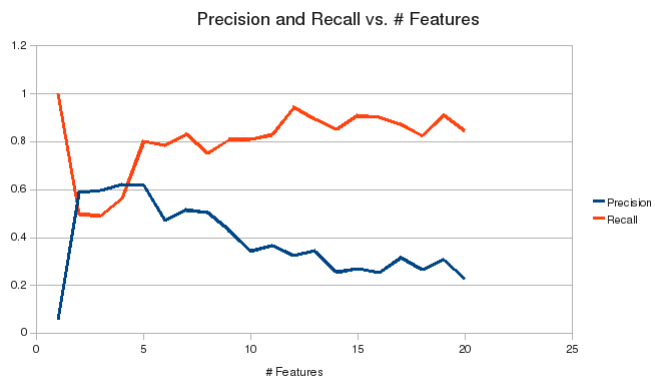

Figure 2: Plot of precision and recall versus the number of features used in Naive Bayes using PMI

to zero (due to their counts of zero), bringing down $P(C|F_1...F_n)$. Again, using fewer features may help eliminate this issue.

Pointwise Mutual Information

A problem with using a simple implementation of Naive Bayes is that there are a wide variety of features to choose from. The easiest case is to use all seen words as a feature; however the results are not that great and the feature set will grow to unmanageable sizes as the training set increases. Thus it would be prudent to come up with a way to select which features were the best.

One way to accomplish this is by seeing which words have the most pointwise mutual information (PMI) with the "movie" label. The PMI is calculated by taking the log of the joint distribution divided by the product of the two individual probabilities.[5]

$$PMI(f,c) = \log \frac{P(f,c)}{P(f)P(c)}$$

Since we are using this information as simply a ranking system, the log is not needed. Moreover, since we care mainly about which words are indicative of something being of a "movie" class, $P(c)$ is the same across all calculated values, the scoring expression can be simplified as:

$$S(f,c) = \frac{P(f,c)}{P(f)}$$

Each word seen in the training set is scored, and the top k words with the best score are kept as features.

PMI Analysis

By running the test on a sweep of the number of features, some trends can be gleaned. With just one feature, all the emails are labeled "movie" which is not the behavior we desire. Recall takes a sharp dip with a small number of features, but eventually approaches the high 80's once you reach 20 features. Precision suffers as a consequence, but as stated earlier it is not a significant detriment if there are more emails classified as "movie" than is necessary.

However the good recall values show that the majority of the emails that mention movies can labeled appropriately with as few features as 10. Due to the small sample size, it would be good to explore the results further on other people's email and see if there it isn't just over-fitting my personal email. Since these values are averages from cross validation, it appears as though it is not over-fitting; however more training data would help the classifier.

tf-idf weighting

Another metric used to determine the importance of a word to a class of documents is term frequency - inverse document frequency (tf-idf) weighting. This measure has two main components, the first being how often a word appears in a document and the second how many documents in the set have that word.[6]

$$tf_{i,j} - idf_i = \frac{n_{i,j}}{N_j} \log \frac{D}{d_i}$$

where $n_{i,j}$is the number of instances of word i in document j and N_j is the total number of words in document j. D is the total number of documents and d_i is the number of documents containing the word i. The second part of the expression gives a measure of how "unique" the term is because the smaller the number of documents containing that word is, the larger the quantity $\log \frac{D}{d_i}$. This indicated the word is in general uncommon, and thus a large presence in a document is worth noting.

Since tf-idf is mainly usually in the context of determining the document's relevance to a particular term, the function was adapted to act as a count. Thus instead of simply using relative frequencies to estimate the parameters, the tf-idf was used instead, acting as a weighted count. Thus, the denominator $N_j$is removed since we don't need the individual counts normalized to the size of the document.

tf-idf Analysis

Interestingly, when using the new parameters derived from tf-idf weights, the performance actually suffers when using PMI to determine which features ought to be used. This was most likely the result of a discrepancy between what PMI believed were the important

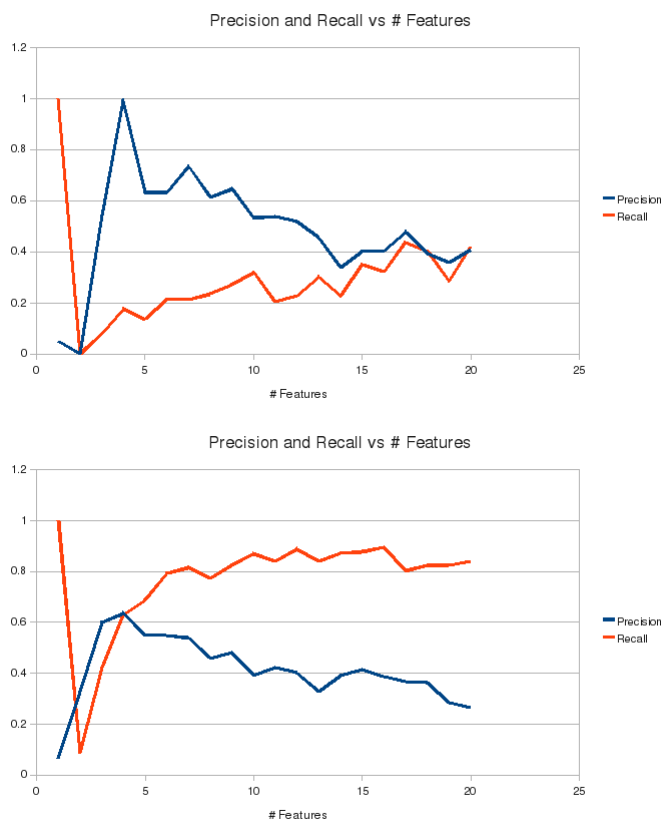

Figure 3: Top: Naive Bayes using PMI to determine features and tf-idf weights as parameters. Bottom: Naive Bayes using tf-idf weights to determine features as well as setting parameters.

words and what the tf-idf weights favored. To investigate, the Naive Bayes was also run using the tf-idf weights and the k features with the largest weights. The result was that the two systems used very different feature sets. For example, the top five features with PMI were: quarter, watchmen, square, seafood, and musical. On the other hand, tf-idf had the following features: movie, rain, NUMBER (a token), and, the. Surprisingly, common words like "and" and "the" were chosen, but they occurred with enough frequency to have a high weight despite the $\log \frac{D}{d_i}$ term.

Using the tf-idf weights to choose features and act as counts resulted in similar precision and recall values as conventional Naive Bayes. Naive Bayes is known for performing remarkably well despite its simplicity and appears that this case is no different.

Named Entity Recognition

Maxent Implementation

The implementation of the maxent model is largely unchanged from Programming Assignment 3. The distribution of the probability of a label, c, given a datum, d, and feature weights λ. is as follows according to the lecture notes [4]:

$$P(c|d,\lambda) = \frac{\exp \sum_i \lambda_i f_i(c,d)}{\exp \sum_{c'} \exp \sum_j \lambda_j f_j(c',d)}$$

The exponent is taken which means that it will always be positive, a requirement for a probability distribution. Also, the log probability can be expressed as:

$$\log P(c|d,\lambda) = \sum_i \lambda_i f_i(c,d) - \sum_{c'} \exp \sum_j \lambda_j f_j(c',d)$$

The function $f_i(c,d)$ is the feature count of feature f_i for label c and datum d and could be seen as a 'vote' from that feature. Thus if it is 0, it does not effect the sum. The denominator makes sure the probability is a distribution since it is a sum across all labels c', so the sum of probabilities across c will equal 1.

Using these log probability distributions, the classifier learns the weights λusing the derivative of the objective function F for each λ_i. In addition, a penalty was imposed to keep the weights small. The resulting objective function is

$$F = -\sum_{(c,d)} \log P(c|d,\lambda) + \sum_i \frac{\lambda_i^2}{2\sigma^2}$$

and the derivative with respect to a weight λ_i is

$$\frac{\delta F}{\delta \lambda_i} = \sum_{(c,d)} \sum_{c'} P(c'|d,\lambda) f_i(c',d) - \sum_{(c,d)} f_i(c,d) + \sum_i \frac{\lambda_i}{\sigma^2}$$

To be efficient in computing the sums and eliminating redundant for loops, the derivative was rewritten as

$$\frac{\delta F}{\delta \lambda_i} = \sum_{(c,d)} \left(\sum_{c'} P(c'|d,\lambda) f_i(c',d) - f_i(c,d) \right) + \sum_i \frac{\lambda_i}{\sigma^2}$$

Additionally, the objective and derivatives were calculated in the same loops.

Features and Error Analysis

A wide variety of features were tested to order to build an NER system. 1/5 of the sentences that were annotated in the emails labeled "movie" were used as the test set. Given that there weer not that many emails labeled "movie," the resulting dataset was about 4000 words in the training set and 1000 in the test. There were three main types of features used: orthographic, prefix and suffixes to detect common roots, and using the previous word itself.

The base features which included the word itself and the label of the previous word performed terribly, but it is to be expected. The precision and recall for "time" and "locale" were zero. Having the same time or date come up in the different emails would be pretty uncommon, so a larger training set would be needed for these features to be of use.

Orthographic

The most important orthographic feature appears to be the presence of a colon in the word. This feature improved the precision and recall for time labels to 93% precision and 54% recall. This feature is a bit of a common sense feature since using colons in times like "11:00am" is quite common and an obvious choice in differentiating it from regular words.

Prefix and Suffix

Using the suffix "day" was a reasonable choice for a feature as every day of the week ends with the word "day." While there was improvement seen for the "date" label, the boost was not as great as expected.

Previous word

Intuitively, the word directly preceding a word should provide some clues as to what the label is. For entities like dates, times, and locations, the word preceding should be a preposition, for example "at 10," "on Sunday," or "at AMC 20." The reason why a bigram was not used was due to the sparsity of the dataset. Given that unigrams were not enough to provide a good sense of whether something was a "date" or "time," using bigrams would end up with even more sparsity.

Table 2: NER Performance for Various Labels

	Precision	Recall	FB1
date	18.18%	10.53%	13.33
locale	66.67%	13.33%	22.22
time	90.00%	37.50%	52.94
title	100.00%	7.14%	13.33

The result from adding these features yielded better performance in the time and locale labels, but the date was unfortunately unchanged. By adding some joint features like the previous word being "on" and the current word having the suffix "day," encouraging results were seen. Surprisingly, the feature that takes into account whether the previous word is "see" or "watch" does nothing to improve the precision nor recall of the"title" tag. There were many cases where the movie title was simply the subject header and thus without context.

Label Analysis

Despite all these improvements, a category that conspicuously did not perform better was "title." Movie titles are tricky to classify simply due to their variety. Whereas times and dates can take on certain styles and values, movie titles could range from the numerical "300" to ones that could be regular sentences like "He's Just Not That Into You." Additionally in the realm of emails, one cannot assume proper capitalization is used for titles either.

With such an elusive label, a possible solution would be to utilize a database of movies, or simply a list of movies currently playing to train on. This could be useful in identifying movies with unique names like "Star Trek."

Conclusion and Future Work

While the classifier managed to reach the goal of achieving high recall, the NER falls short. For certain labels like 'time' and 'locale,' the NER performs fairly well, but for movie titles themselves, there is limited success. Movie titles do not tend to share any distinct features to differentiate them from other words.

Future steps would be to train the classifier and NER on a larger and more diverse corpus. The classifier works well on the dataset used, however more data would not hurt and the NER may see drastic improvement in labeling dates, locations and times. In order to improve movie title detection, integrating the system

with a database of movie titles is an option. Whether training against a short list of recent movies is the simplest solution that may result in improvement.

References

[1] Black, J. A., Ranjan, N, Automated Event Extraction from Email, Stanford University

[2] Senapaty, S., Detection and Extraction of Events from Email, 2008, Stanford University

[3] Brown et al, The Mathematics of Statistical Machine Translation: Parameter Estimation, IBM T.J. Watson Research Center

[4] Manning C, Polynomial Time Parsings of PCFG's Lecture Slides, Stanford CA, 2009

[5] Jurafsky, D, Martin, J, Speech and Language Processing: An Introduction to Natural Language Processing, Computaional Linguistics, and Speech Recognition, 2nd edition. Prentice-Hall 2008

[6] Manning C, Raghavan P, Schutze, H, Introduction to Information Retrieval, Cambridge University Press 2008