Sentiment Analysis on Movie Review Data using Maximum Entropy Classifier: In search of the Best Feature Set

Yan Huo

Abstract

Due to the large amount of available data coming from the web, the interest in quickly and correctly determining the overall sentiment orientation (positive or negative) of a review has dramatically increased. Current literature about sentiment analysis mainly focuses on two approaches, which are rule based and machine learning based. This project focuses on how machine learning based MaxEnt classifier works in sentiment analysis. I investigated different feature sets for training (including using unigram features only, using bigram features only, using both unigram and bigram features, adding position information and adding parts of speed tagging) in order to obtain the best results in the classification. Information gain calculation was used to prevent from overfitting and the curse of dimensionality. Negations (e.g. not, isn't) were handled by a preprocessing technique to improve the classification performance. The results were evaluated by accuracy, precision, recall and F-measure. The best result was obtained when using 2000 most frequent adjectives, adverbs and verbs as feature set with information gain calculation and negation preprocessing. The average accuracy for three runs is 0.880, compared with 0.798 obtain from Naïve Bayes baseline method, indicating that MaxEnt outperforms Naïve Bayes at sentiment analysis tasks.

1. Introduction

1.1 Sentiment Analysis

The problem I chose is sentiment analysis, that is to determine the overall sentiment orientation of a review, e.g. whether it is positive or negative. Today, readers are always overwhelmed by the large amounts of information available online, the motivation of sentiment analysis is to provide readers succinct summaries of the reviews and to help them understand the main meanings of the reviews and feedbacks more quickly. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service.[1]

Sentiment analysis problem can be solved using NLP, because the keywords in the reviews can provide us some information about the overall sentiment. For example, the reviews contain keywords *'bad'* and *'terrible'* are more likely to be negative reviews. But sentiment analysis is not an easy task, since sentiment sometime is expressed in a subtle manner. For example, the review *"…, this is probably the high point in the movie."* contains no single word that has obviously negative meaning; however, the whole sentence implies the negative emotional orientation of the review. It makes sentiment analysis a more difficult problem than other classification problems, such as topic-based classification. Besides, the informal language of reviews is another challange involved in sentiment analysis.[2]

1.2 Literature Review

Most of existing approaches to sentiment analysis can be divided into two

categories: rule based[3] and machine learning based.

Rule based methods make use of sentiment lexicons of the text, that is, classifying the reviews by a set of pre-selected seed words with semantic orientation (positive or negative)[4].

Machine learning based methods in general are similar to classical task of text classification. The commonly used classifiers are Naïve Bayes[5], MaxEnt[6] and support vector machine[7].

2. Dataset(s)

The data set used for this project is *Polarity Dataset 2.0* from Cornell. It consists of 2000 supervised movie reviews (1000 positive ones and 1000 negative ones). Each review is labeled with respect to its overall sentiment polarity (positive or negative). This corpus is included in NLTK, so I can easily include them into my code using the name *movie_reviews*. I shuffled the dataset and randomly chose 1600 reviews as training set, 200 reviews as held out set and 200 reviews as test set.

Each review contains several sentences and the overall sentiment polarity is given for the whole review instead of for each sentence. All words in the reviews are in lower cases.

For instance, the first review in this data set (*cv000_29416.txt*) is labeled as 'negative'. It consists of 35 sentences. The first sentence of this review is "*plot : two teen couples go to a church party , drink and then drive .",* which gives us no sentiment information. The sixth sentence is "*critique : a mind-fuck movie for the teen generation that touches on a very cool idea , but presents it in a very bad package .*", which indicates that this review is negative.

3. Experimental method

3.1 Main Method

3.1.1 MaxEnt

The main method I used in this project is Maximum Entropy Classification. As discussed in the lecture, unlike Naïve Bayes, MaxEnt makes no assumption about the independency between features, so it may potentially perform better than Naïve Bayes in this project, since the conditional independence assumptions may not meet here.

3.1.2 Information Gain

In order to prevent from overfitting and the curse of dimensionality, I calculated the information gain for each word and eliminated low information features. Information gain is a measure of how common a feature is in a particular class compared to how common it is in all other classes. For example, a word that occurs primarily in positive movie reviews and rarely in negative reviews is high information.[8] And we want to keep only these high information features in our feature set.

3.1.3 Deal with Negation

Obviously, negation will switch the sentiment orientation of a review. For example, '*good*' and '*not good*' indicate opposite sentiment polarities. I tried a preprocessed technique that adding *NOT_* to every word between a negation word and the first punctuation after the negation word.[9] The negation words includes '*not*', '*isn't*', etc.

3.2 3rd Party Tools

a. nltk.MaxentClassifier

I used *MaxentClassifier* from nltk as my optimizer.

Specifically, I implemented "feature extractors" (e.g. Function *Get_uni_features(…):* etz.) on my own and used them to build the feature set and then used *train(…):*method in *nltk.MaxentClassifier* class to obtain the classifier.

b. nltk.metrics.BigramAssocMeasures

I used *nltk.metrics.BigramAssocMeasures* to calculate the information gain for each word.

Specifically, I calculated some frequencies for each word on my own (in Function *Set_word_stat(…):*) For example, the number of times each word appeared in positive or negative reviews, total number of words in positive and negative corpus. And then I used function *chi_sq(…):* in *nltk. metrics. BigramAssocMeasures class* to calculate the positive score and negative score for each word. Then I chose the words with high scores.

In order to call the function *nltk. metrics. BigramAssocMeasures.chi_sq(…):* I also installed *numpy*, using *"pip install numpy"* command.

c. nltk.pos_tag

I used *nltk.pos_tag(…):* to obtain the part of speech tagging for each word. I can also use the function written on my own in problem set 3.

3.3 Overall Complexity

The worst-case time complexity of Maxent algorithm per iteration is $O(KSN_C|D|)$, where K is the number of clusters, S is the number of the features, N_C is the number of classes and |D| is the number of data records.[10] For sentiment analysis, K = 1 and N_C = 2, so the overall time complexity is O(2S|D|), which is linear to the number features and linear to the number of data records.

Since I kept the number of features relatively low (using information gain), this approach scales well to large amounts of data.

I used movie reviews as my data; however, I did not use any feature that are specific to movie reviews, so this method should be easily applicable to other kind to data, such as product reviews.

4. Evaluation

4.1 Evaluation metrics

- Confusion matrix:

		Predicted Label	
		Positive	Negative
Real Label	Positive	A	B
	Negative	C	D

- Accuracy evaluates the effectiveness of the classifier.
  Accuracy = (A + D) / (A + B + C + D),
- Precision measures the exactness of a classifier.

Positive Precision = A / (A + C)
Negative Precision = D / (B + D)

- Recall measures the completeness of a classifier.
  Positive Recall = A / (A + B)
  Negative Recall = D / (C + D)
- F-measure combines precision and recall. In this project, it is just as useful as accuracy, providing no extra information.

4.2 Baseline

The baseline method I chose is Naïve Bayes Classification with only unigram features assuming all features are independent with each other, which is obviously not true. I used Laplace smoothing to deal with the case of unknown words. It is slightly different from the one we covered in the lecture. Prior $P(c_j)$ is involved in the formula because the number of positive reviews and negative reviews may not be the same in the training and test set. (We randomly picked 1600 reviews as training set.)

$$c_{BN} = argmax_{cj=c}\, P(c_j) \prod_{i\, position} P\,(w_i|\, c_j)$$

$$P(w_i|c_i) = \frac{count(w_i, c_i) + 1}{count(c_i) + v}$$

4.3 Results

4.3.1 Baseline

The code of this part is in baseline_and_unigram_only.py

I ran baseline algorithm three times and the results are shown in the *Table 1*. Because we shuffled the dataset and randomly picked 1600 reviews as my training set, I got slightly different results for each run.

The classification accuracies are 0.790, 0.840, 0.765. The average accuracy for baseline method is 0.798.

	accuracy	pos precision	neg precision	pos recall	neg recall	pos F_measure	neg F_measure
1st run	0.790	0.805	0.770	0.820	0.753	0.812	0.761
2nd run	0.840	0.842	0.838	0.825	0.854	0.833	0.846
3rd run	0.765	0.785	0.748	0.730	0.800	0.756	0.773

Table 1. Results from baseline method.

4.3.2 Main Method

4.3.2.1 Choose feature set

I investigated the effect of choosing different features and different number of features.

Note: Since I shuffled the dataset and randomly picked the training set. The result for each run is slightly different. However, the trends for the comparison are the same.

a. Unigram only:

The code of this part is in baseline_and_unigram_only.py

In this part, the feature set contained only unigram features. I changed the number of unigram features in training (chose the n most frequent unigrams) and calculated the accuracies for training set and held out set. The results with and without information gain calculation are shown in *Table 2* and plotted in *Figure 1*.

As the number of features increases, the accuracy of the training set increases;

however the accuracy of the held out set does not always increase. This is due to the overfitting, as we can tell from the most informative features. The top five informative features from the training results with 100, 2000, 10000 total unigram features without information gain are shown in *Table 3*. When we used small amount of features to train (e.g. 100 unigram features), the features were all most common words, such as *'a', 'of', etc*. The information obtained from these words was limited, which lead to low accuracies for both training set and held out set. When we used reasonable amount of features to train (e.g. 2000 unigram features), the feature set contained some word with sentiment orientation, such as *'outstanding', 'wasted', etc.* These words were helpful for us to determine the overall sentiment of the review, so both training accuracy and held out accuracy increased. When we used too many features to train (e.g. 10000 unigram features), the most informative features were more special words, like *'stupidity' and 'sucks'*. These words were less likely to appear in the held out set. Therefore, adding too many unigram features into our feature set not only slows down the training process but also causes the held out accuracy to decline.

The optimal numbers of features without and with information gain are 2000 and 500, respectively. This makes sense, since information gain helps us to select high information features, so we can obtain high accuracy with fewer features. But information gain does not always improve the performance; the accuracy did not increase when we applied information gain.

num of uni features		100	200	500	1000	2000	3000	5000	10000
Without info gain	Train Accuracy	0.672	0.746	0.801	0.860	0.887	0.903	0.910	0.926
	Held out Accuracy	0.670	0.735	0.785	0.845	**0.855**	0.840	0.820	0.820
With info gain	Held out Accuracy	0.813	0.851	0.898	0.914	0.936	0.941	0.958	0.973
	Test Accuracy	0.795	0.830	**0.840**	0.815	0.815	0.815	0.810	0.800

Table 2. Accuracies of using unigram features only (The optimal values are in bold)

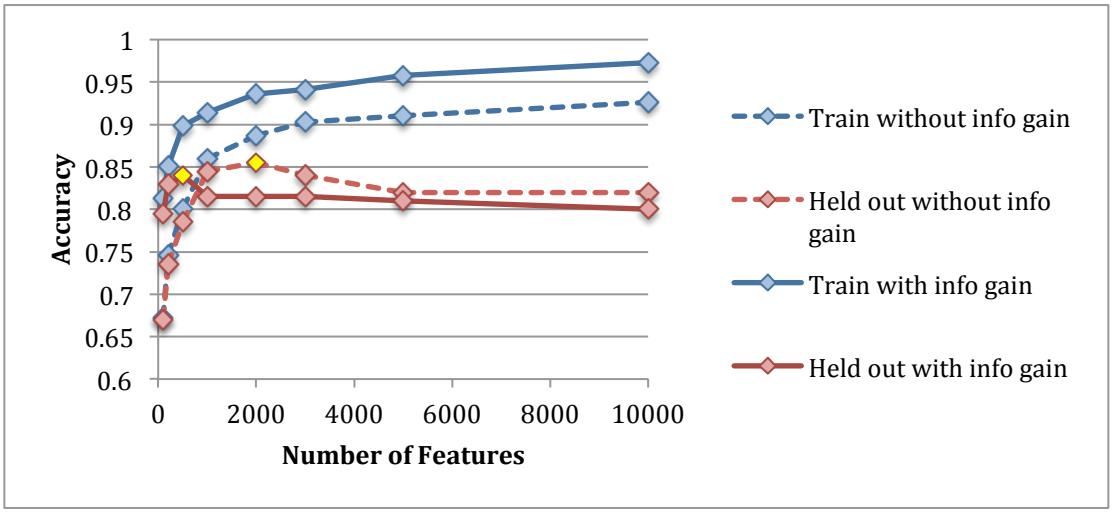

Figure 1. Plot of accuracies of using unigram features only (The optimal values are marked in yellow)

Total number of Features	Top Five Informative Features				
100	a	of	the	it	is
2000	outstanding	wasted	pointless	wonderfully	poorly
10000	stupidity	sucks	insulting	warns	effortlessly

Table 3. Top five informative features when using unigram features only (without information gain)

b. Unigram + Bigrams:

The code of this part is in uni_and_bigram.py

I studied the effect of context by involving bigram features when using information gain. I fixed the number of features of unigram to be 500 (which is the optimal value obtained from the previous part) and added bigram features to the feature set. The accuracies with different number of bigram features are shown in *Table 4* and plotted in *Figure 2*.

As we can see from the results, adding bigrams features to the feature set does not seriously impact the results. That is probably due to that bigrams features and unigrams features carry the similar kinds of information. For example, *bi 'extremely well'* does not provide much extra information than *uni 'well'*. Therefore, using both unigram and bigram features just behave the same as using unigram features only.

num of bi features (+500 uni feature)	0	12	25	50	125	250	500
Train Accuracy	0.897	0.894	0.898	0.892	0.893	0.891	0.907
Held out Accuracy	0.850	0.825	0.840	0.845	0.850	0.820	0.850

Table 4. Accuracies of using both unigram features and bigram features, fix the number of unigram features to be 500 and varying the numbers of bigram features (with info gain)

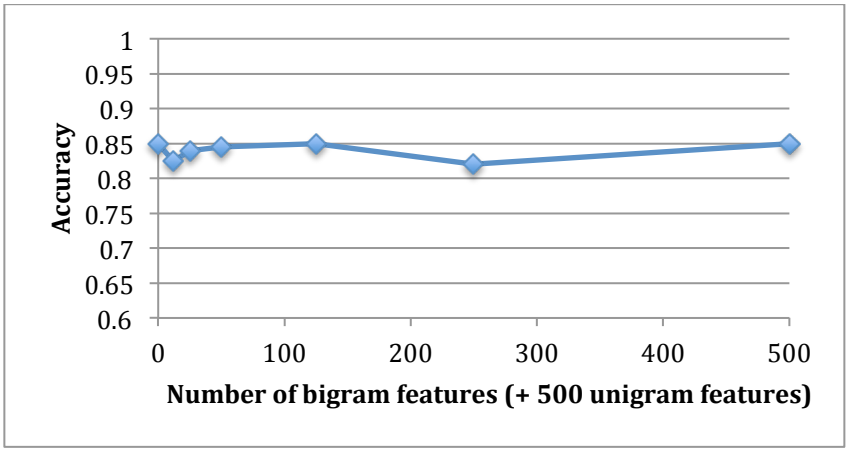

Figure 2. Plot of accuracies of using both unigram features and bigram features

Total number of Features	Top Five Informative Features				
500 uni + 500 bi	uni (insulting)	bi (extremely,well)	bi (a,place)	uni (uninvolving)	bi (and,boring)

Table 5. Top five informative features of using 500 unigram features and 500 bigram features

c. Bigrams only:

The code of this part is in bigram_only.py

Besides, relying just on bigram features does not obviously improve the performance, as shown in *Table 6* and *Figure 3*, indicating that in contrast to our intuition, bigram features do not provide much extra information for sentiment analysis.

num of bi features	100	200	500	1000	2000	3000	5000
Without info gain	0.650	0.640	0.695	0.715	0.760	0.775	0.810
With info gain	0.675	0.690	0.745	0.790	0.835	0.835	0.855

Table 6. Accuracies of using bigram features only

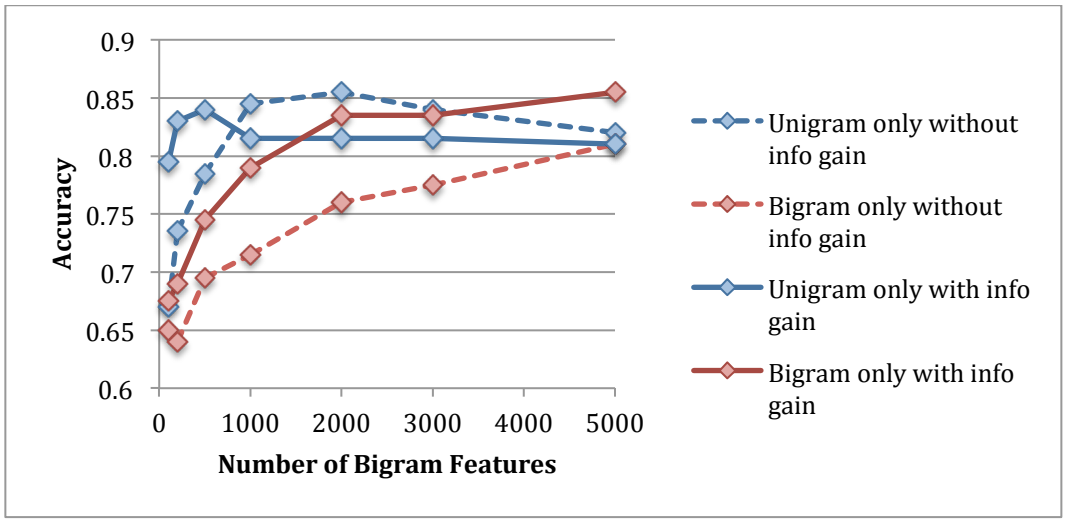

Figure 3. Plot of accuracies of using bigram features only, compared with that of using unigram features only.

d. Position:

The code of this part is in position.py

The movie reviews, as well as other kinds of reviews, always start with an overall sentiment statement, followed by some discussions, and end up with a conclusion of the author's views. So the position of the words may provide extra information for sentiment analysis. To investigate that, I tagged every word with its position ('*start*'- in the first quarter of the review, '*end*'- in the last quarter of the review, '*middle*'- in the middle of the review). The results are shown in *Table 7* and plotted *Figure 4*.

Adding position information to the feature set causes accuracy to decline by as much as 0.055. Besides, the feature set is 3 times larger than the original one to obtain the optimal accuracy. (Without position information, the optimal accuracy was obtained when using 500 features; while with position information, the optimal accuracy was obtained when using 1500 features.) This make sense, because when adding position information, each word becomes three features, (word, start), (word, middle) and (word, end).

Also, as shown in the *Table 8*, the words in the beginning and at the end of the reviews do not contain more information than those in the middle of the review. Thus, adding position information to the feature set does not improve the performance.

num of features	100	200	500	1000	1500	2000	3000
Without position info	0.795	0.830	**0.840**	0.815	-	0.815	0.815
With position info	0.740	0.760	0.785	0.775	**0.785**	0.780	0.760

Table 7. Accuracies of using unigram features plus position information (with info gain)

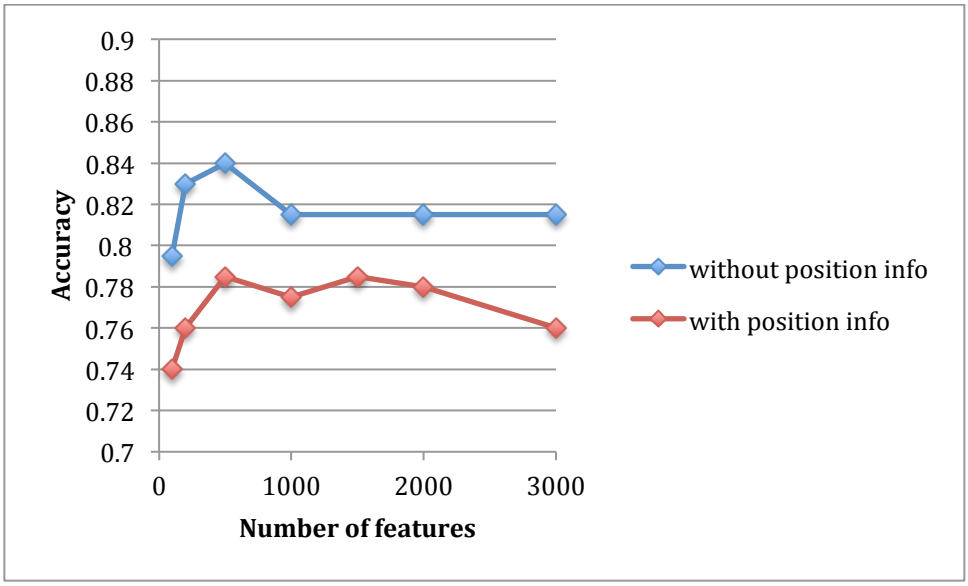

Figure 4. Plot of accuracies of using unigram features plus position information (with info gain)

Total number of Features	Top Five Informative Features				
(info gain)	waste end	random middle	compelling start	wonderful end	awful end

Table 8. Top five informative features of using 500 unigram features plus position information

e. Parts of Speech Tagging:

The code of this part is in POS.py

The parts of speech tags of the words are important in sentiment analysis task. For example, the word *'love'(NN)* in *"The film is about **love**, truth and honesty."* provides no sentiment information; while *'love'(VB)* in *"I **love** this film"* indicates that the review is positive.

Since adjectives usually carry a great deal of information regarding a review's sentiment, I investigated the performance using adjectives only. Adverbs and verbs may also contain sentiment orientation information. Thus, I also did the training process on adjectives and adverbs only, as well as on adjectives, adverbs and verbs only. The results are shown in *Table 9* and plotted *Figure 5a* (without info gain) and *Figure 5b* (with info gain)

Using feature set containing adjectives, adverbs and verbs with information gain lead to the accuracy increased to 0.880, which is the better than using other feature sets. That is probably due to the reason that using adjectives alone or using adjectives and adverbs causes some useful information lost; while using words with all kinds of tags involves too many useless information. Thus, including only adjectives, adverbs and verbs into the feature set is the best choice for this project.

Num of features		100	200	500	1000	2000	3000
adj only	without info gain	0.760	0.745	0.780	0.805	0.805	0.805
	with info gain	0.795	0.830	0.815	0.815	0.830	0.835
adj + adv	without info gain	0.735	0.745	0.780	0.805	0.800	0.795
	with info gain	0.750	0.805	0.830	0.855	0.840	0.845
adj + adv + v	without info gain	0.715	0.730	0.785	0.830	0.840	0.850
	with info gain	0.800	0.810	0.855	0.875	**0.880**	0.870
all	without info gain	0.670	0.735	0.785	0.845	**0.855**	0.840
	with info gain	0.795	0.830	0.840	0.815	0.815	0.815

Table 9. Accuracies of held out set using adjectives, adverbs and verbs as features.

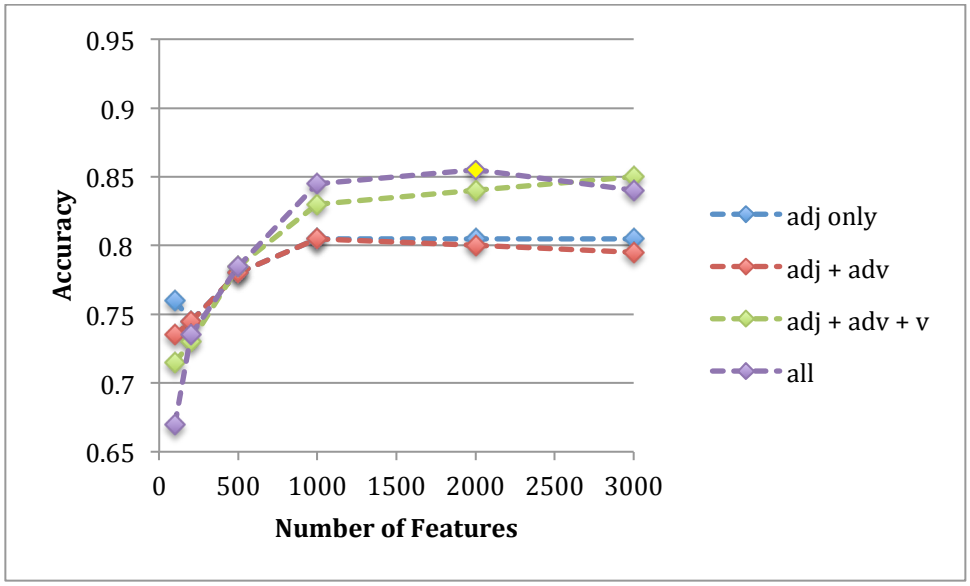

Figure 5.a. Plot of accuracies of held out set using adjectives, adverbs and verbs as features without information gain.

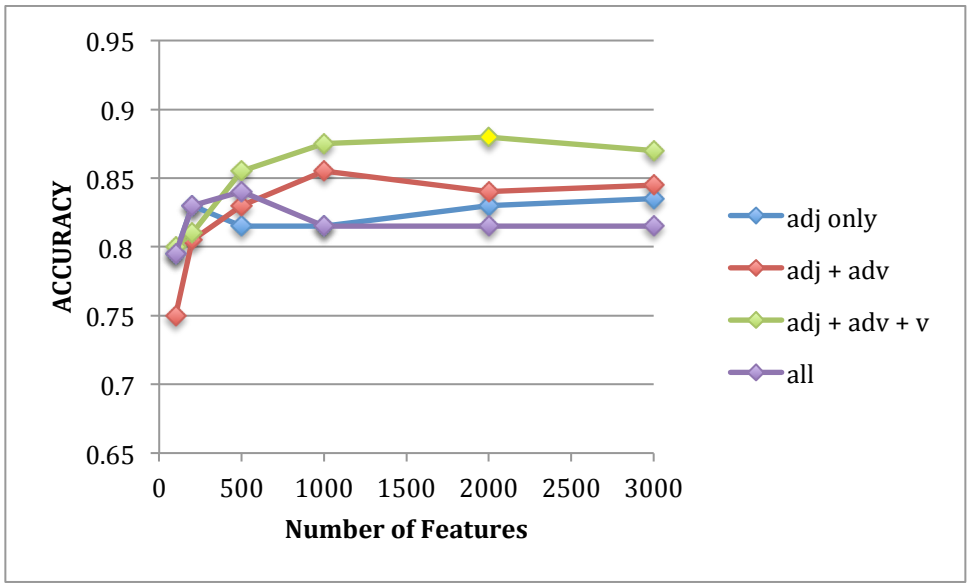

Figure 5.b. Plot of accuracies of held out set using adjectives, adverbs and verbs as features with information gain.

4.3.2.2 Deal with Negation

The code of this part is in negation.py

As discussed in previous chapter, I added *NOT_* to every word between a negation word and the first punctuation after the negation word.

For example, the 29th sentence in the first review is *"oh , and by the way , this is **not** a horror or teen slasher flick ."* After the negation preprocess, the sentence becomes "oh , and by the way , this is not NOT_a NOT_horror NOT_or NOT_teen NOT_slasher NOT_flick ." The results of adding negation preprocessing are shown in *Table 10* and plotted *Figure 6.*

By comparing the results of using and not using negation preprocessing, we can conclude that adding negation-preprocessing technique improves the classification performance.

num of features	100	200	500	1000	2000	3000	5000
Without negation preprocessing	0.795	0.830	0.840	0.815	0.815	0.815	0.810
With negation preprocessing	0.805	0.845	0.850	**0.870**	0.865	0.860	0.865

Table 10. Accuracies of using unigram features only with negation preprocessing

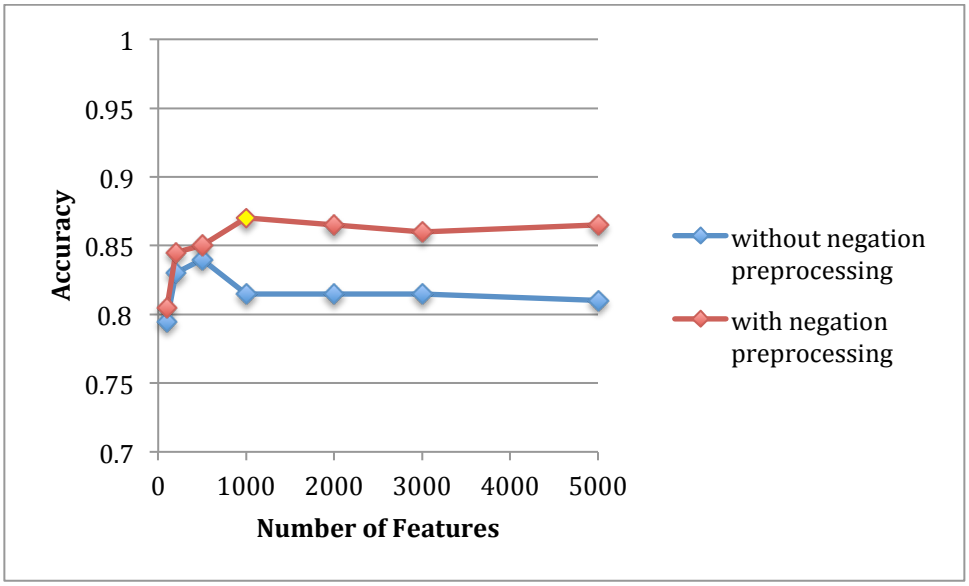

Figure 6. Plot of accuracies of using unigram features only with negation preprocessing

4.3.2.3 The Highest Accuracy I Can Obtain

The code of this part is in main.py

From the investigation in previous chapter, the best result was obtained when using 2000 most frequent adjectives, adverbs and verbs as feature set. Also, calculating information gain shortens the optimization time and doing negation preprocessing improves the performance. Thus, the best feature set for my project is 2000 most frequent adjectives, adverbs and verbs with information gain calculation and negation preprocessing.

I ran the main method on test set for three times, the results were shown in *Table 11.* The average accuracy for five runs is 0.880, compared with 0.798 obtain from Naïve Bayes baseline method, indicating that MaxEnt outperforms Naïve Bayes at sentiment

analysis tasks.

From the values of precision and recall, no positive or negative preference was observed.

	accuracy	pos precision	neg precision	pos recall	neg recall	pos F_measure	neg F_measure
1st run	0.875	0.905	0.848	0.843	0.908	0.873	0.877
2nd run	0.890	0.882	0.898	0.900	0.880	0.891	0.889
3rd run	0.875	0.865	0.885	0.874	0.876	0.869	0.880

Table 11. Results from main method.

4.3.2.4 MaxEnt vs. Naïve Bayes

As we predicted, my main method beats my baseline method. There are lots of cases that the main method does the right thing but the baseline gets it wrong:

a. when parts of speech tagging information involved

For example, in the training set, there is a sentence *"I **love** this film"* (positive) and in the test set there is a sentence *"The film is about **love**, truth and honesty."* Since baseline method uses only lexical information, it will treat the two *'love'* as the same and probably predicted that the test sentence is also a positive one. On the contrary, in the main method, we only use adjectives, adverbs and verbs, so the love (noun) is not used for the prediction. Thus, it will result more accurate prediction.

b. when the independence assumptions not met

MaxEnt makes no assumption about the independency between features, while Naïve Bayes assumes words to be independent with each other. So when the independence assumptions not met, the main method potentially performs better than baseline. For example, *"high quality"* has a positive meaning, but neither '*high'* or *'quality'* contains positive information. In this case, the baseline method cannot make a correct prediction.

c. when negation preprocessing need

For example, baseline method cannot figure out that *"not good"* and *"good"* have opposite sentiment orientation, but the main method can.

4.3.2.5 Limitations of the main method

There are some cases our method cannot predict correctly, because sometimes the sentiment orientation expressed in a subtle and implicit way. For instance, our method cannot predict *"…, this is probably the high point in the movie."* as a negative review.

5. Conclusions

The basic task of my project is to classify the polarity of a give movie review – whether the expressed opinion is positive or negative. Existing approaches to sentiment analysis can be grouped into two categories: rule-base techniques and machine learning based techniques. My project used the later one. More specifically, I used MaxEnt classifier and investigated different feature sets. I also calculated the information gain and handled the case of negations. The accuracy obtained by using "best" feature set is 0.880. Compared with the accuracy of 0.798 obtain from baseline, it is a large improvement.

In the future work, I may try some other feature sets, such as involving syntactic parser. I'll also try other techniques, for example support vector machines. The other

direction is to compare the results of using knowledge based techniques, machine learning based techniques and some hybrid approaches.

---

[1] https://en.wikipedia.org/wiki/Sentiment_analysis
[2] Mavljutov R. R. and Ostapuk N. A. Using basic syntactic relations for sentiment analysis. International Conference on Computational Linguistics, 2013
[3] Sreenivasa P. Sista and S. H. Srinivasan. Polarized Lexicon for Review Classification. Proceedings of the International Conference on Machine Learning; Models, Technologies & Applications, 2004.
[4] Peter D. Turney and Michael L. Littman: *Measuring Praise and Criticism: Inference of Semantic Orientation from Association.* Technical Report EGB-1094, National Research Council Canada.
[5] Liviu P. Dinu and Iulia Iuga. *The Naive Bayes Classifier in Opinion Mining: In Search of the Best Feature Set.* CICLing 2012.
[6] Bo Chen, Hui He, Jun Guo. Constructing Maximum Entropy Language Models for Movie Review Subjectivity Analysis. Journal of Computer Science and Technology 23(2): 231-- 239, 2008.
[7] Tony Mullen and Nigel Collier. *Sentiment Analysis using Support Vector Machines with Diverse Information Sources.* Proceedings of EMNLP, 2004.
[8] http://streamhacker.com/2010/06/16/text-classification-sentiment-analysis-eliminate-low-information-features/
[9] Basant Agarwal, Namita Mittal, et.al. *Prominent Feature Extraction for Sentiment Analysis.* Page 36
[10] Dmitry Pavlov, Alexandrin Popescul, et.al. *Mixtures of Conditional Maximum Entropy Models.* Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.