Sentiment analysis using linear classifiers and semantic orientation

Aaron Knodell

1 Task and Problem

Sentiment analysis is a popular problem in Natural Language Processing; a search for "Sentiment analysis" on Google Scholar returns over 1 million results. After completing two programming projects in this course that were related to sentiment analysis, I attempted to combine the best features of both approaches to produce a single classifier that was more accurate than either individually.

2 Description of Approach

2.1 Prior Work

Previously I had implemented two linear classifiers, Naive Bayes and Perceptron, that were limited to a unigram "bag of words" method. I also implemented the method described in Peter Turney's paper *Thumbs up or thumbs down?* utilizing pointwise mutual information for adjective phrases in proximity to the words "poor" and "excellent" to calculate the semantic orientation of a given review.

2.2 Initial Testing

The first step I took was to evaluate ways to make the individual classifiers more accurate. I started by running through the linear classifiers using the untagged IMDB review test set from the previous assignments. Next I ran through the linear classifiers using a set of the same IMDB reviews that had been tagged using the tagChunk POS tagger. Since the POS tagger includes an additional tag for the grammatical role of each word, I also ran the linear classifiers with this last tag removed. For the Naive Bayes classifier I ran through each data set with all possble combinations for options flags (binary, stop words, both, and neither). For the Perceptron classifier I ran though each dataset doing 1, 5, and 10 iterations. Since the Perceptron results varied each time it was run, I ran it 10 times for each number of iterations and took an average of the accuracies.

Once I had these baseline results, I implemented an N-gram functionality for each of the linear classifiers. I ran through each of the data sets again, with all permutations of flags/iterations testing for bigrams and trigrams and stored the results in a series of `.out` files.

The next step I took was to investigate ways to improve the accuracy of the semantic orientation classifier. One of the biggest issues with the original implementation was that it left a large number of reviews unclassified because they didn't contain the words "excellent" or "poor". Originally, I had been classifying such reviews as "positive" by default, expecting the difference to even out over time. However, this obscured the true accuracy of the classifier, so I modified the output to show both the accuracy of the classifier for the reviews it was able to classify and the percentage of reviews it was unable to classify.

I also had a few personal issues with the choice of the words "excellent" and "poor" for determining semantic orientation. In the case of "excellent", it didn't seem like a common enough word to provide an accurate count for positive review. "Poor" was even more problematic to me. For one thing, "poor" seems like a much weaker sentitment than "excellent", so I didn't feel like it was appropriate to give occurences of "poor" the same weight as

”excellent” when evaluating overall sentiment. Additionally, ”poor” has a second meaning related to wealth that seemed likely to me to cause confusion when a review is simply describing a character’s attributes rather than the film’s. For these reasons, I ran through the review dataset using a several different combinations of words to calculate positive (”good”, ”great”, ”excellent”) and negative (”poor”, ”bad”, ”terrible”) orientation and saved the results in a series of `.out` files.

2.3 Combining Classifiers

Once I had gathered data about which features produced the most accurate results for each of the classifiers, I wrote an implementation of a classifier that contained both the Naive Bayes classifier and the semantic orientation classifier. I decided not to include the Perceptron in my combined classifier because it was not as accurate or as fast as the Naive Bayes classifier. I ran the combined classifier a few times to get information about what the average semantic orientation and the magnitude of the positive versus negative probabilities for the Naive Bayes subclassifier were for each review in cases when both subclassifiers were correct, when only the Naive Bayes subclassifier was correct, when only the semantic orientation subclassifier was correct, and when both were incorrect.

Using this information, I attempted to devise a way to determine which subclassifier was correct when there was a conflict between their guesses. To test my classifier on a different dataset than what was trained on, I downloaded the polarity dataset v1.1 from https://www.cs.cornell.edu/people/pabo/movie-review-data/, which contained about 1400 reviews, sorted almost evenly into positive and negative (There are 694 positive reviews and 692 negative reviews).

3 System Specific Implementations

3.1 Naive Bayes Subclassifier

The Naive Bayes subclassifier uses the set of reviews with only the first part of speech tag attached by the tagChunk tagger. I found that my implementation of the Naive Bayes classifier was most accurate when using bigrams, rather than unigrams or trigrams. To remove the last tag and form the bigrams, I used the following for loop when reading each file:

```
for i in range(len(tagged) - 1):
  word1 = tagged[i].split('_')[0] +
    '_' + tagged[i].split('_')[1]
  word2 = tagged[i+1].split('_')[0] +
    '_' + tagged[i+1].split('_')[1]
  bigrams.append(words[i] + ' '
    + words[i+1])
```

I did not implement either the binary or stop words functionality, since using one or both of these options was consistently less accurate than not using them.

3.2 Semantic Orientation Subclassifier

For the semantic orientation subclassifier, I opted to use the keywords ”great” and ”bad” since they produced both a high degree of accuracy and a low percentage of unclassified reviews.

3.3 Conflict Resolution

The biggest difficulty was how to determine when to use the semantic orientation result instead of the Naive Bayes result. There were 697 cases in the training set where only Naive Bayes had the correct classification and only 149 where only semantic orientation was correct. Finding a balance between true positives where only the semantic orientation guess was correct and the false positives where only naive bayes was correct was a challenge. To determine the overall guess based on the results of the two subclassifiers, I used the following conditional block:

```
if pmiGuess == nbGuess:
  guess = pmiGuess
elif pmiGuess == 'unknown':
  guess = nbGuess
elif nbGuess == 'unknown':
  guess = pmiGuess
else:
  if abs(averageSo) > 1.7543 and
      abs((posProb-negProb)/
      (posProb+negProb)) < 0.0009:
    guess = pmiGuess
  else:
    guess = nbGuess
```

If the two subclassifiers agreed, then it didn’t matter which guess I used. If one of the subclassifiers

was unable to determine a result (this did not happen with Naive Bayes in this training set, but it is a possiblity), then the classifier used the result of the other subclassifier. Otherwise, if the subclassifiers disagreed, I check whether the semantic orientation returned by the semantic orientation classifier was greater than 1.7543 and whether the difference between the logs of the positive and negative probabilites divided by the sum of the logs of the positive and negative probabilites was less than 0.0009. If both conditions were true,I used the semantic orientation classification. In every other case I used the Naive Bayes result for the overall result.

4 Results and Evaluations

4.1 Naive Bayes Testing

N-Gram	Opt	Untagged	POS Tag	Tagged
1	None	0.816500	0.815000	**0.817000**
1	-b	0.683000	0.682500	**0.689500**
1	-f	0.812000	**0.813500**	0.811000
1	-b -f	0.681500	0.679000	**0.692500**
2	None	0.842500	**0.843000**	**0.843000**
2	-b	0.775000	0.775000	**0.780500**
2	-f	0.815500	**0.816500**	0.808000
2	-b -f	0.768000	**0.784000**	0.774000
3	None	**0.775500**	0.774500	0.761500
3	-b	**0.788500**	0.785500	0.779500
3	-f	**0.776500**	0.713500	0.697000
3	-b -f	**0.789500**	0.740000	0.742000

Table 1: Naive Bayes Classifier Results. Bolded value is the highest classification accuracy for each N-gram/Option combination.

The Naive Bayes testing (**Table 1**) yielded several interesting results. The biggest surprise to me was that adding Part of Speech tags to the words did not necessarily improve accuracy. It seemed like the higher the N-gram value was, the less accurate more detailed tagging of the words made the classifier. It was also interesting to note that the trigram was the only N-gram value that performed better with the binary and stop word options than without them, yielding its highest accuracy when using both. I believe that these results are related; the more specific a feature is, the more it is fitted to the training set. More detailed tags and higher n-gram values can result in overfitting to the the training set, while removing stop words and using presence rather than counts (binary) make the classifier more general.

Ultimately, the most accurate combination of options for the Naive Bayes classifier was for tagged bigrams without the binary or filtering options. These options resulted in 0.843000 accuracy for both single and full versions of the POS tagging. Based on the fact that other bigram variations performed better with just the single POS tag, I decided to use that option for the combined classifier.

4.2 Perceptron Testing

N-Gram	Iter	Untagged	POS Tag	Tagged
1	1	**0.611050**	0.590650	0.580650
1	5	**0.682400**	0.678200	0.659950
1	10	0.729000	**0.733200**	0.701800
2	1	**0.684700**	0.677500	0.672600
2	5	**0.757200**	0.752100	0.738550
2	10	**0.768350**	0.765450	0.750000
3	1	0.684650	**0.694650**	0.685450
3	5	**0.744850**	0.739750	0.738400
3	10	**0.742100**	0.733700	0.734700

Table 2: Perceptron Classifier Results. Bolded value is the highest classification accuracy for each N-gram/iteration count combination.

The Perceptron results(**Table 2**) were similar to the Naive Bayes results, with the most accurate result belonging to the the bigram implementation. One this that was interesting about this set was that trigrams were actually more accurate than unigrams across the board, and actually was the most accurate of all the results when only one iteration was run. Once again, the untagged reviews were generally more accurate than either of the variations of tagged words I ran though. Based on how much the Perceptron lagged behind the Naive Bayes classifier in accuracy and the added strain of increasing the number of iterations it would need to run to make up the difference, I decided not to include a Perceptron implementation as part of my final classifier.

4.3 Semantic Orientation Testing

When I looked at the reults from my tests of the semantic orientation classifier (**Table 3**), some of my suspicions regarding the use of the words ”excellent” and ”poor” were confirmed. When these words were used to calculate the semantic orientation of a review, they left nearly 36% of the reviews unclassified. The word ”excellent” appeared to be

Pos Word	Neg Word	Accuracy	Unclassified
”good”	”poor”	0.507322	0.044000
”good”	”bad”	0.545175	**0.026000**
”good”	”terrible	0.508891	0.044000
”great”	”poor”	0.525768	0.088000
”great”	”bad”	**0.592765**	0.032500
”great”	”terrible	0.519135	0.098500
”excellent”	”poor”	0.567863	0.359000
”excellent”	”bad”	0.556158	0.078500
”excellent”	”terrible	0.573191	0.392000

Table 3: Semantic Orientation Results. Bolded values are the highest accuracy and lowest percentage of unclassified reviews.

the culrit, since all the permutations using ”good” or ”great” left fewer than 10% of the reviews unclassified, while two of the three that used ”excellent” categorized less than 75% of the reviews. ”Excellent” did produce accurate results, making up the second, third, and fourth most accurate permutations, but left so many unclassified I decided not to use it in the combined classifier.

The word ”poor” did not have as much of an issue, but it did appear to cause a problem with the accuracy, as it was never the most accurate negative word when paired with any of the positive words (”bad” outperformed it twice, and ”terrible” outperformed it once).

The combination of ”great” and ”bad” seemed to produce the best results when the semantic orientation classifier was run in isolation, producing the highest accuracy and the second lowest rate of unclassified reviews. Based on these results, I decided to use ”great” and ”bad” as my positive and negative keywords when running the combined classifier.

4.4 Combined Classifier Testing

Result	Mean NB Diff%	Mean Avg SO
Both Correct	0.003104	2.094035
NB Only Correct	0.002962	1.899439
SO Only Correct	0.001153	2.175149
Both Incorrect	0.001393	2.10818

Table 4: Average absolute values of the difference between the positive and negative probabilities divided by the sum of the positive and negative probabilites ((pos-neg)/(pos+neg)) and the average of the semantic orientation of each review, broken down by the result returned by the classifiers. All values rounded to six decimal places.

Result	Med NB Diff%	Med Avg SO
Both Correct	0.002854	1.72901
NB Only Correct	0.002582	1.407773
SO Only Correct	0.000905	1.754255
Both Incorrect	0.001224	1.927902

Table 5: Median absolute values of the difference between the positive and negative probabilities divided by the sum of the positive and negative probabilites ((pos-neg)/(pos+neg)) and the average of the semantic orientation of each review, broken down by the result returned by the classifiers. All values rounded to six decimal places.

Once I had decided on which classifiers and features to use in my combined classifier, I ran the program once to see which subclassifiers correctly identified which reviews. Out of 2000 reviews, 987 were correctly classified by both subclassifiers, 697 were correctly classified by the Naive Bayes subclassifier and incorrectly classified by the semantic orientation classifier, 149 were correctly classified by the semantic orientation classifier and incorrectly classified by the Naive Bayes classifier, and 167 were incorrectly classified by both. I figured the main challenge I had was finding a way to identify when the semantic orientation classifier was correct and the Naive Bayes was incorrect without creating more false positives than true positive (cases where Naive Bayes was correct and semantic orientation was incorrect, but it looked like the opposite). To perform more in-depth analysis, I had the program print out the absolute values of the average semantic orientation and the difference in the positive and negative probabilities divided by the sum of the probabilites of each review for each possible outcome and took the arithmetic means and medians of the results (**Table 4** and **Table 5** respectively). I cases where one of the classifiers did not make a guess, I removed the probability difference percentage and average semantic orientation of that review when calculating the mean and median.

Looking at these results, I noticed that both the mean and median values of the probability difference percentage for the Naive Bayes subclassifier was significantly lower in cases where it was incorrect, indicating the classifier was less sure in those cases. I also noticed that although there was not a signficiant difference in the mean for the average SO of a review between when the semantic orientation

guess correctly, it was noticably lower when only the Naive Bayes subclassifier was correct, and was actually higher when both were incorrect. Based on these numbers, I tested the combined classifier using both the mean and the median to as thresholds for when to use which subclassifier to resolve conflicts. I found that the median was slightly more accurate, and used these results to arrive at the conditional block mentioned in section 3.3.

I also tried to implement a way of telling when both classifier were incorrect, but it resulted in more false negatives than true positives, bringing down the overall accuracy of the classifier.

In the final result of the classifier, using the Naive Bayes subclassifier on untagged bigrams and the semantic orientation classifier using "great" and "bad" to calculate semantic orientation, the I ran the classifier on the second set of 1400 reviews using 10 fold validation. I achieved an accuracy of 0.808802, compared to 0.821798 achieved when running the Naive Bayes classifier by itself. This may suggest I overfit the classifier to the training data when trying to determine when to use the semantic orientation subclassifer's guess.

Out of curiosity, I also ran the classifier over the test set using "excellent" and "poor" for semantic orientation. Doing this resulted in an accuracy of 0.810245. I also tried running it with "great" and "bad", but requiring at least two proximity hits for a phrase to be included in the dictionary, and this resulted in an accuracy of 0.819625. These improvements suggest that maybe there was too much noise in the dataset when using more common words.

5 Conclusions

As Turney mentioned in his paper, movie reviews present a unique challenge in sentiment analysis. Since the review can praise some aspects of a movie and criticize others, it can be difficult to find the balance between the positive and negative comments and arrive at an overall evaluation.

Overfitting seemed to be a problem, with the subclassifier performing better on the training data than the test data. Perhaps incorporating an additional method of sentiment analysis would allow the combined classifier to filter out some of the noise and catch more of the reviews that were incorrectly classified by both current subclassifiers.

References

Turney, P. D. (2002, July). Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. *In Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 417-424). Association for Computational Linguistics.