Sentiment Analysis of Movie Reviews

CS339 Topics in Computer Science Final Project

Michael Yang

Abstract. Semantic analysis allows the categorization of opinions in textual data according to subjectivity and polarity. By using the Naïve Bayes approach along with methods to deal with negation and neutral words, an accuracy of 85% was achieved for the sentiment analysis of movie reviews. The results are further analyzed and possible improvements are discussed.

1 Introduction

Today, there is a huge quantity of data on the internet, and a lot of attention is going into processing such data in order to turn them into something more useful. Since most of this data constitutes textual data, it is especially essential to study the nature and properties of text. Such textual data can be roughly divided into two main domains: facts and opinions. The task of sentiment analysis is to categorize the opinions expressed in a text according to subjectivity and polarity. This can be achieved in a number of different ways, as it will be seen from the previous works section. By categorizing the opinions on the internet, we can basically find out how the internet feels towards a topic at a given time. This information can have various applications, and some will be discussed in the next section.

My project focused on the polarity of movie reviews, categorizing each review as either positive or negative. I was interested in answering the following three questions:

1. What is the accuracy of the program with a simple Naïve Bayes approach?
2. How does dealing with negation and blacklisting certain neutral words affect the accuracy?
3. How does the size of the training corpus affect the accuracy?

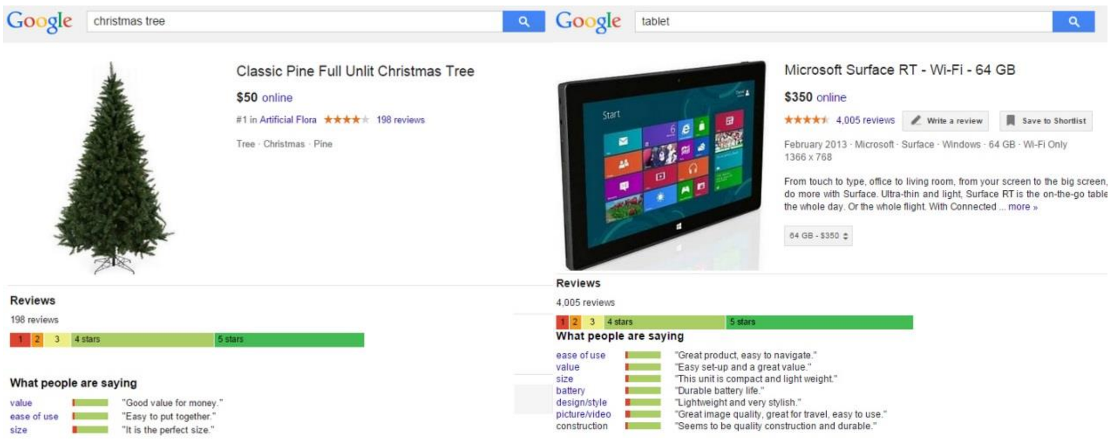
By answering these three questions, the accuracy of the program with the given training and test set is maximized, the results are analyzed, and possible improvements are discussed.

2 Applications

2.1 Google Product Search

Sentiment analysis can be applied to products. Google product search[1] automatically extracts facts and sentiments for these facts from consumer reviews. Google finds certain attributes or aspects that are important features of the product, and assigns a sentiment values for each of these attributes. The process starts by Google reading through all the reviews and detecting important attributes. From *Figure 1*, it can be seen that the Christmas tree and the tablet have different attributes that are fitting to each product. Christmas tree has attributes such as size and value, while the tablet has attributes such as battery and design.

[1] http://www.google.com/shopping


christmas tree

Classic Pine Full Unlit Christmas Tree

$50 online

#1 in Artificial Flora ★★★★☆ 198 reviews

Tree · Christmas · Pine

Reviews

198 reviews

1 2 3 4 stars 5 stars

What people are saying

value	"Good value for money."
ease of use	"Easy to put together."
size	"It is the perfect size."

tablet

Microsoft Surface RT - Wi-Fi - 64 GB

$350 online

★★★★☆ 4,005 reviews Write a review Save to Shortlist

February 2013 · Microsoft · Surface · Windows · 64 GB · Wi-Fi Only 1366 x 768

From touch to type, office to living room, from your screen to the big screen, do more with Surface. Ultra-thin and light, Surface RT is the on-the-go table the whole day. Or the whole flight. With Connected ... more »

64 GB - $350

Reviews

4,005 reviews

1 2 3 4 stars 5 stars

What people are saying

ease of use	"Great product, easy to navigate."
value	"Easy set-up and a great value."
size	"This unit is compact and light weight."
battery	"Durable battery life."
design/style	"Lightweight and very stylish."
picture/video	"Great image quality, great for travel, easy to use."
construction	"Seems to be quality construction and durable."

Figure 1. Google Product Reviews

Next, Google determines the sentiment for each review, and aggregate these sentiments to give a summary of "What people are saying". The bar indicates the proportion of positive to negative reviews for every attribute.

2.2 Twitter Sentiment Analysis

Sentiment analysis can also be applied outside of products. Currently, Twitter Tweets provide a great amount of sentiment data from a wide range of people. A lot of attention is going into analyzing these Tweets, and people were able to essentially predict the stock market using this information. Bollen et al in their research[2] extracted certain sentiment attributes from Twitter such as calmness, sureness, happiness, and alertness. To these attributes, they assigned sentiment values from a massive amount of tweets every day, and found out that there is a high correlation between the level of calmness in the Tweets and the Dow Jones average 3 days later (*Figure 3*).

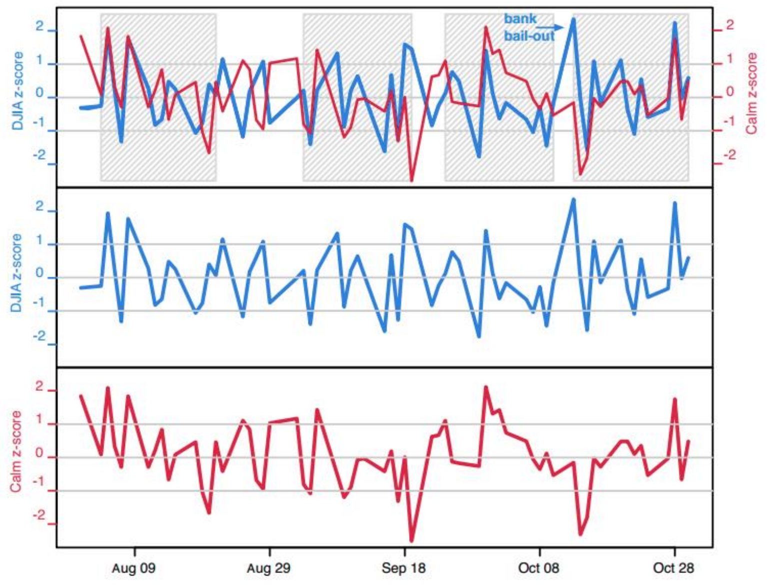


Figure 2. Twitter Sentiment Analysis: Calmness and Dow Jones

[2] Available at http://hughchristensen.co.uk/papers/socialNetworking/1010.3003v1.pdf

3 Previous Works

The topic of movie review sentiment analysis have also received some attention. Rather than being a big topic in and of itself, it is usually used to test the accuracy of the predictions made by a program since movie reviews tend to contain strong opinions and polarities. Most notable examples of work in this field are by Stanford and Cornell.

3.1 Stanford NLP

The Stanford NLP group[3] approaches the problem of sentiment analysis a bit differently than others. While most sentiment prediction systems work by just looking at single words in isolation and figuring the sentiment associated with these individual words, Stanford implements what is known as the deep learning model. This model builds up a representation of whole sentences based on the sentence structure, and directly takes into consideration all the grammatical structures. Such approach most effectively tackles the issue of contrastive conjunctions and negation, which will be discussed later on. The model of the training dataset is the Stanford Sentiment Treebank, which essentially breaks every sentences of the reviews while still retaining the grammatical structure of the sentence and the parts-of-speech of the words. They also allow users to help the model learn by labeling sentences which the program would intake as additional datasets. *Figure 3* is an example of their analysis done on a simple movie review, and it very well captures the effectiveness of the Treebank structure.

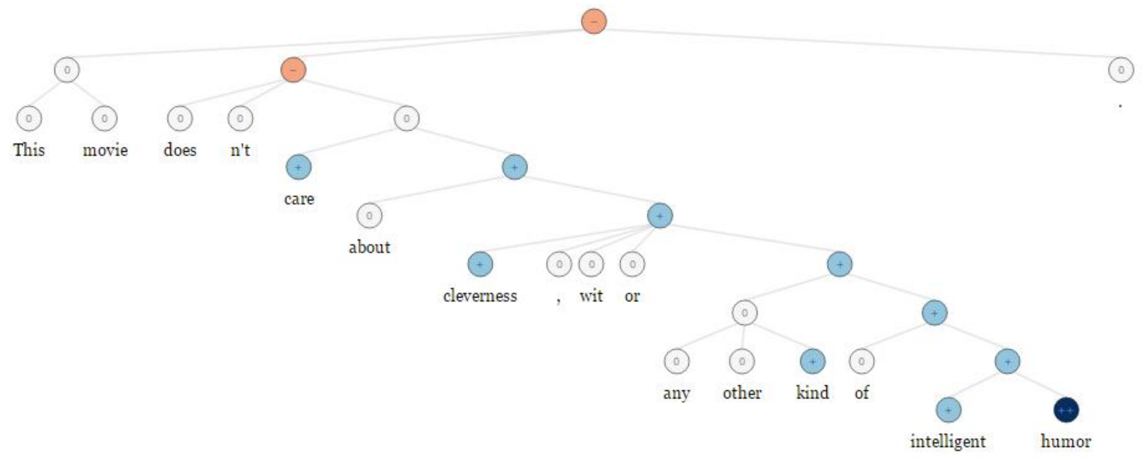


Figure 3. Stanford NLP Movie Sentiment Analysis

3.2 Cornell

Bo Pang and Lillian Lee[4], researchers at Cornell, took a different approach to the problem. Rather than focusing on polarity alone, they also focused on subjectivity, as they first of all extracted sentences from movie reviews via subjectivity detection. This allowed them to detect sentiments much more efficiently with larger amounts of datasets. Then they used their default polarity classifier, which is based on individual words rather than phrases unlike Stanford's

[3] http://nlp.stanford.edu/software/corenlp.shtml
[4] Summary of paper at http://www.cs.cornell.edu/home/llee/papers/cutsent.home.html

Treebank, in order to determine whether the test review is positive or negative. They achieved close to 90 percent accuracy on various types of movie reviews, and showed how the subjectivity detection itself can affect the outcome by a large degree.

4 Methodology

4.1 Naïve Bayes

The classifier was written in Java. The supervised classification approach of Naïve Bayes was taken, which is the most basic approach. Naïve Bayes assumes that all features in the feature vector are independent, which is a rather strong assumption. While there is also the binarized, or the Boolean, multinomial approach, which clips all the word counts in each document at one, such approach is not taken and the frequencies are recorded as it occurred in the training corpus. To be more specific, given a review *r*, the probability *p(c|r)* needs to be computed for every class *c*, which in our case is positive and negative. This means, by Bayes Theorem, that we need to choose the one that maximizes:

$$\underset{c}{\operatorname{argmax}}\, p(c|r) = \underset{c}{\operatorname{argmax}} \frac{p(c)p(r|c)}{p(r)}$$

Since *p(r)* is the same in all classes, we can cross it out, and what's left is the numerator. Then we represent the reviews in a model known as bag of words, which is discussed shortly after. The value *p(r|c)* is then determined based on the multinomial distribution, using the frequency of occurrences of each word in the review.

4.2 Feature Selection

4.2.1 Bag-of-Words Model

The bag of words representation of the text is a rather simple way of looking at a body text. It first makes a list of all the words that are present in the text and then counts up all the number of occurrences of each word. This type of representation loses all information about the order of the text, which is in contrast with the Treebank approach taken by the Stanford NLP group. While there are obvious downfalls to this approach, the bag of words model allows us to quickly apply the Bayes algorithm in order to compute the probabilities of each word for a given polarity. The effectiveness of such approach in simplifying the computational process is seen below in the experimental set-up section.

4.2.2 Tokenization

In order to achieve the bag of words model, the reviews first need to be tokenized. The *Tokenizer* class written for our class homework was used, and as will be discussed later, the class was modified according to a set of rules to handle negation.

4.2.3 Laplace Smoothing

During the calculation for the probabilities using the Naiive Bayes approach, there can't be any zero probabilities since they would result in the end product equaling zero. This calls for a way to deal with the unknown words the classifier will encounter during the testing process. For this exact reason, Laplace smoothing is implemented, which is the process of just adding one to all the count in the dictionary. This may lead to some problems when the training corpus is extremely small as even an occurrence of one can lead to a probability that can impact the outcome.

4.3 Challenges

Most of the challenges with sentiment analysis of movie reviews are those that are common to any type of natural language processes. The most obvious one is with typos. While most of the reviews written by users are informal and contains a lot of spelling mistakes, this problem was essentially looked over for this project. Building a way to catch and correct spelling errors or grammatical mistakes is a whole different field in and of itself, and that would require a lot of time and coding to achieve. There are also difficulties with sarcasm and ambiguity. Due to the nature of the methodology being used, keywords are the most essential in determining the polarity of the text. However, sentences such as "Honestly the best 3 hour nap I've ever had!", which is obviously a negative review, would be seen as positive with the presence of words such as "best" and "!". The problem of ambiguity was also not dealt with, as it is an issue that is hard to solve with the bag of word model being used. A few challenges, however, could be dealt with, such as negation and neutral words.

4.3.1 Negation

Negation plays a significant role in polarity analysis. The presence of words such as "not" and "never" essentially flips the meaning of the words following them. For example, when a reviewer says the movie is "not good", it is the same as saying "bad". The flipping of the meaning is especially important for the adjective. However, the adjective might not come directly after the negation, as it is common to say "That movie was not too great". In such example, simply flipping the meaning of the word after *not* would not suffice. Thus the approach I took was to add an exclamation point to the beginning of every word that follows the keywords "n't", "not", and "never", up to a punctuation mark. This approach would tokenize "I didn't like the movie." into {"I", "did", "n't", "!like", "!the", "!movie", "."}. While this may potentially double the size of the dictionary and deem inefficient for large training sets, the small size of both the training corpus and the size of the dictionary allowed me to take a rather brute force method.

4.3.2 Neutral Words

Koppel and Schler in their research[5] showed the importance of a third class of polarity, the neutral class. Often times, people ignore the neutrals and only assign positives or negatives to words. However, as it is true for most languages, the English dictionary contains words that are mostly neutral in polarity, and the words that have either positive or negative sentiments only constitute a small portion. Their work primarily focused on SVM and they used geometric properties in order to improve the accuracy of their three binary classifiers, and they achieved results much better than when only two categories (positive, negative) were identified. In my project, due to Naïve Bayes approach, the simplest approach was to implement a blacklist to

[5] Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.9735&rep=rep1&type=pdf

the dictionary that would get rid of common neutral words such as “a”, “the”, etc. The specifics of the blacklisted word will be discussed in the evaluations, but this approach evidently has its own downfalls and weaknesses to it.

Evaluation of Performance

4.4 Experimental Set-up

4.4.1 Training Corpus

The training corpus[6], provided by the NLP group at Cornell, contains 1000 positive and 1000 negative reviews all written before 2002. There are 312 different authors in total, and one author wrote no more than 20 reviews. A very wide range of movies was covered in the corpus, and this led to some problems with proper nouns and verbs. First of all, due to the small number of reviews, many of the verbs and nouns that are known to be neutral came out to have a strong polarity in either direction. Moreover, the biggest problem was with proper nouns. A portion of the test reviews was from the recent movie *Hunger Games: Mocking Jay*. However, the classifier determined words such as *Jennifer*, *Lawrence*, and *mocking* as negative, which created some issue of the code classifying most of the reviews as negative. While the word *mocking* is rightly determined to have a negative polarity, it should not be taken as such in those test reviews where the title contains the word itself. This could create a bias towards the negative polarity, and this is apparent in the results. In order to deal with this issue, the previously mentioned method of blacklisting was used. In addition to the common conjunctions and pronouns, certain proper nouns and movie titles were also blacklisted, although some weaknesses were introduced. The most obvious is the blacklisting of words that have rightly determined to have a certain polarity. The previously shown example of the word *mocking* fits this category. Also, there is the issue of having to define the blacklist for every movie added for the test reviews. The program is no longer unsupervised, and the blacklist must be manually determined and added, which can be very tedious.

4.4.2 Test Reviews

The test set was taken from IMDB[7]. The five movies from which the reviews were taken were: *The Hunger Games: Mockingjay*, *Penguins of Madagascar*, *Big Hero 6*, *Interstellar*, and *Gone Girl*. The set included 50 positive and 50 negative reviews. The positive reviews were taken at random from the top reviews filtered by the “Loved it” option, and the negative reviews were taken at random from the top reviews filtered by the “Hated it” option. The dictionary was then defined by all the words present in the test reviews. Thus, the test reviews in question were all tokenized by the program, and a list was created with every unique word in the reviews. As it was mentioned earlier, this approach would be highly inefficient when the test set is large.

4.5 Effect of Negation and Blacklisting

The results for the sentiment analysis of movie reviews is shown in *Figure 4* and the entire set of results is reported in Appendix A. Without dealing with negation and neutral words, or in

[6] Available at http://www.cscornell.edu/people/pabo/movie-review-data/ (polarity dataset v2.0)
[7] http://www.imdb.com/

other words without implementing the negation tokenization and blacklist, the overall accuracy came out to be 80%. It is important to note the large discrepancy between the accuracy for the positive reviews and the negative reviews. As it can be seen from the far left set of bar graphs, the accuracy for positive reviews sits around at 60% while the accuracy for negative reviews is above 95%. This may possibly due to not taking negation into consideration or the neutral words having strong polarities as it has been previously discussed.

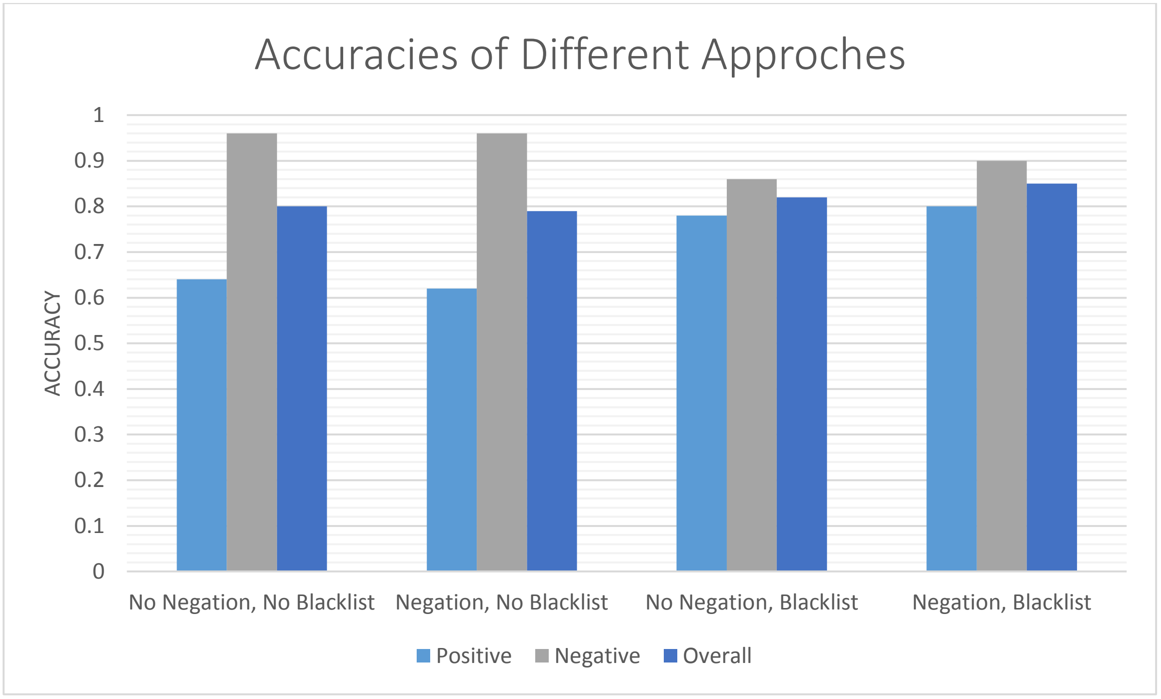


Figure 4. Accuracies of Different Approaches

Once the negation rules were implemented in the tokenization stage, the results became slightly worse, as the accuracy for the positive reviews dropped slightly. However, when the blacklist was set in place (no negation rules) in order to restrict certain keywords from the dictionary, the accuracy saw a big improvement. The third set of bar graphs show that while the overall accuracy only increased by one percent, the discrepancy between the accuracy of positive and negative reviews was reduced by a large amount. The blacklist was defined by looking at the individual probabilities of words. Most of the problems was coming from the positive reviews for the movie *The Hunger Games: Mockingjay*, and many of the proper nouns associated with the movie were blacklisted. When the negation rules was applied on top of the blacklisted model, the accuracy saw a slight increase of 3 %, with the program now getting more of the negative reviews correct. These results show how dealing with negation helps increasing the accuracy of determining negative reviews, while dealing with neutral words helps with the accuracy of determining positive reviews.

4.6 Effect of Training Corpus Size

It was predicted before that a small training corpus would lead to more highly polarized neutral words. It can be seen from *Figure 5* that the size of the training corpus does have an effect on the overall accuracy of the code. However, it is slightly different from the prediction that the accuracy actually experiences a drop at when 800 reviews for each polarity (1600 total reviews) were used as the training corpus.

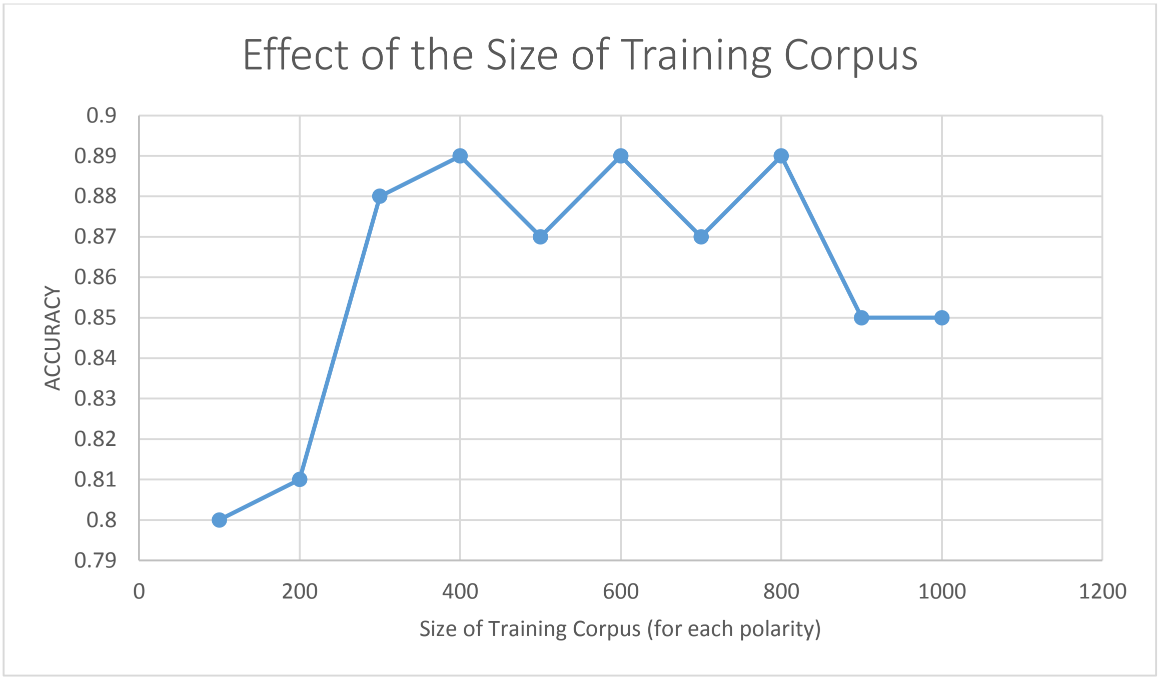


Figure 5. Effect of the Size of the Training Corpus

Upon investigation, the problem turned out to lie in the positive reviews that was added last. When the training corpus for the positive polarity was set for the last 200 reviews that was added to the training corpus in the previous investigation, the accuracy for the positive came out to be 70%. By taking a look at specific reviews, I took only the reviews for the *Hunger Games*, and tested those for the two different training corpus. The training corpus made up of the first 800 reviews for each polarity resulted in a 73% accuracy for positive reviews while the training corpus made up of the last 200 reviews for each polarity resulted in a 53% accuracy, a percentage that is slightly better than just guessing at random. This large difference seems to be where most of the issue is coming from, and it is deduced that the 200 reviews contain keywords that are yet to be blacklisted that affect the neutral words' polarities.

The effect of the last 200 reviews on the positive review accuracy can be seen more clearly from *Figure 6*. The light blue bar shows the accuracy for the positive reviews. From the first 3 sets of bar graphs, the effect of the size of the training corpus is clear; as the corpus size increases, the positive accuracy, as well as the overall accuracy, increases. However, when the corpus size goes from 800 to 900, there is a large dip in the accuracy. This suggests that certain movie reviews in the training corpus can negatively affect the outcome by a large degree, and a simple increase in size doesn't mean better accuracy. The context and accuracy of the testing set also needs to be evaluated, and this is where the weakness of the blacklisting method becomes apparent.

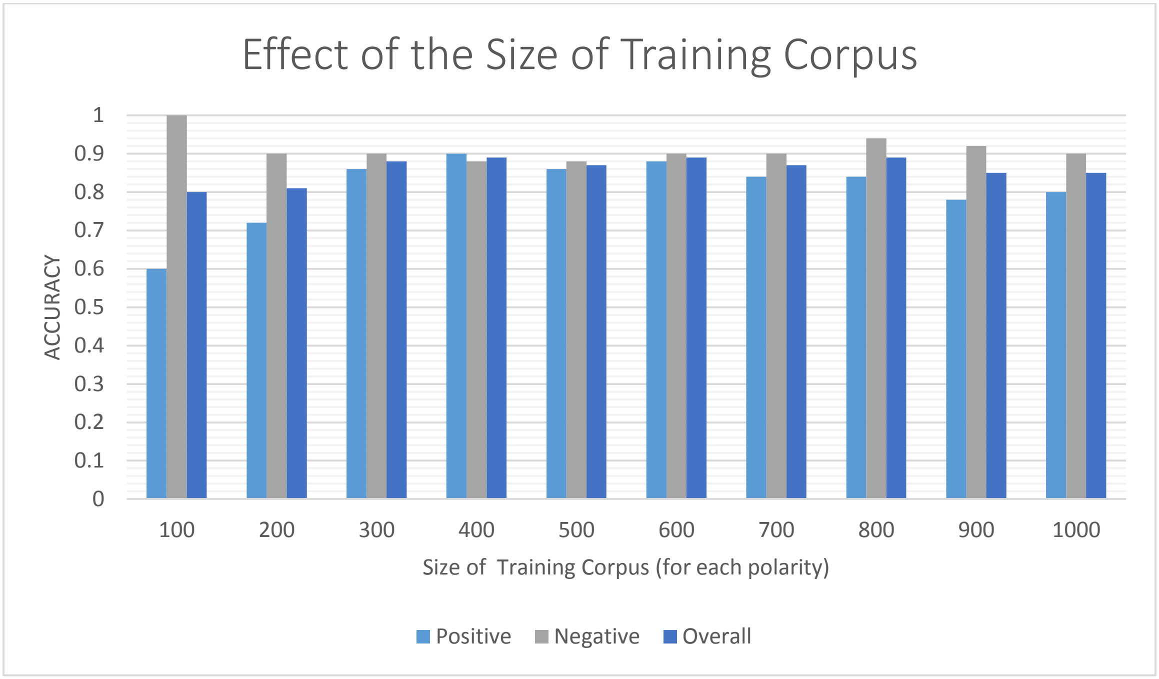


Figure 6. Effect of the Size of the Training Corpus on Positive Accuracy

It is still important to note that as the size of the training corpus increases, the effect of the last 200 reviews would diminish, and the accuracies would go back up. Thus it is safe to say that there is still a positive correlation between training corpus size and accuracy, despite the results showing signs of the opposite.

5 Conclusion

In this project, I have analyzed the sentiment analysis of movie reviews. The Naïve Bayes classifier was used using the bag of words model, and the dictionary for the classifier was taken directly from the test set. The effect of blacklisting and negation rules along with the relationship between size of training corpus and accuracy were evaluated. The final accuracy came out to be around 85%, with a peak of 89% using the first 600 "good reviews", but it is predicted that the accuracy would go back towards this peak as the size of the training corpus is increased. The two approaches taken in order to deal with negation and neutral words turned out to be effective, despite its apparent weaknesses.

Future improvements and works may include:

1. Larger and more focused training set, possibly focused on different genres of movies. This would allow the training corpus to be more fitting for every case, and blacklists, if used, can be more appropriately made.

2. POS tagging and taking the adjectives, adverbs, and some nouns while largely ignoring most verbs, pronouns, and proper nouns. The most polarized words are usually the descriptive words, and verbs, pronouns, conjunctions, and other parts of speeches tend to have a neutral polarity. By only taking into account what is known to be largely polarized words, the probabilities can be more accurate in portraying the polarity of the review as a whole.
3. Feature extraction. A more advanced way of extracting feature, going beyond simply taking adjectives and adverbs. This may include the subjectivity determination method discussed in the previous work section. By doing so, the features would be more centered around opinions rather than facts, and there would be a higher accuracy in the assignments of polarities and less noise created by unimportant facts.
4. Incorporation of SentiWordNet. SentiWordNet provides all the words in the dictionary and its different forms with a predefined polarity. By giving some of the words in the dictionary a predefined polarity, the chance of those words being misrepresented as the other polarity becomes less. In my study, it was shown that the word “amazingly” had a higher calculated probably to be negative than positive. If SentiWordNet was used and the word was predefined as positive, then the couple of occurrences of the word the negative training corpus would not affect the polarity of the adverb as much.

6 Appendix A – Result Tables

Table 1.

Set-up	Positive	Negative	Overall
No Negation, No Blacklist	0.64	0.96	0.8
Negation, No Blacklist	0.62	0.96	0.79
No Negation, Blacklist	0.78	0.86	0.82
Negation, Blacklist	0.8	0.9	0.85

Table 2.

# of Test Reviews Each	Positive	Negative	Overall
100	0.6	1	0.8
200	0.72	0.9	0.81
300	0.86	0.9	0.88
400	0.9	0.88	0.89
500	0.86	0.88	0.87
600	0.88	0.9	0.89
700	0.84	0.9	0.87
800	0.84	0.94	0.89
900	0.78	0.92	0.85
1000	0.8	0.9	0.85

7 Appendix B – Code

MRSentiment Class

```
import java.util.*;
import java.io.*;
import java.math.BigDecimal;

public class MRSentiment {
  static private Map<String,Double> wordProbs = new HashMap<String,Double>();
  static private Set<String> dictionary = new HashSet<String>();
  static public Map<String,Double> getWordProbs(){return wordProbs;}
  static public Set<String> getDictionary(){return dictionary;}

  //Makes Dictionary. Negation.
  static private void makeDictionaryNegation(String myDirectoryPathPos, String myDirectoryPathNeg)throws IOException{
    Set<String> blacklist = new HashSet<String>(Arrays.asList(new String[]
{"games","!games","katniss","!Jennifer","Jennifer","Lawrence","mocking","!mocking","watching","!watching","got","whole","!whole","having",".","to
","!to","n't","can","!can","then","there","this","!this","!then","be","!be","else","!else","so","!so","i","!i","movie","!movie","they","!they","is","!is","wa
s","!was","were","!were"}));
    List<List<String>> posReviews=readReviewDataNegation(myDirectoryPathPos);
    for(List<String> review : posReviews){
      for(String word : review){
        dictionary.add(word.toLowerCase());
      }
    }
    List<List<String>> negReviews=readReviewDataNegation(myDirectoryPathNeg);
    for(List<String> review : negReviews){
      for(String word : review){
        dictionary.add(word.toLowerCase());
      }
    }
    for(String word : blacklist){
      dictionary.remove(word);
    }
  }

  //Turns all the reviews inside a directory into a list of lists of words. Negation.
  static private List<List<String>> readReviewDataNegation(String myDirectoryPath)throws IOException{
    List<List<String>> reviews = new LinkedList<List<String>>();
    File dir = new File(myDirectoryPath);
    File[] directoryListing = dir.listFiles();
    if (directoryListing != null) {
      for (File child : directoryListing) {
        String stringReview = fileToString(child);
        NegationTokenizer tokenizer = new NegationTokenizer();
        reviews.add(tokenizer.tokenize(stringReview));
      }
    }
    return reviews;
  }

  //Reads files. Used by readReviewData & readReviewDataNegation
  static private String fileToString(File file) throws IOException{
    StringBuilder fileContents = new StringBuilder((int)file.length());
    Scanner scanner = new Scanner(file);
    String lineSeparator = System.getProperty("line.separator");
    try {
      while(scanner.hasNextLine()) {
        fileContents.append(scanner.nextLine() + lineSeparator);
      }
      return fileContents.toString();
    } finally {
      scanner.close();
    }
  }

  //Naiive Bayes Classifier
  static private void train(List<List<String>> reviews, int sentiment){
    Map<String,Integer> wordCounts = new HashMap<String,Integer>();
    for(String word: dictionary){wordCounts.put(word,1);}
    int tokenCount=0;
    for(List<String> review : reviews){
      for(String word : review){
        tokenCount++;
        Integer wcount = wordCounts.get(word.toLowerCase());
        if(wcount!=null)
          wordCounts.put(word.toLowerCase(),wcount+1);
      }
    }
    // Compute observation probabilities
    for(String word : dictionary){
      double prob = (double)wordCounts.get(word)/(tokenCount+dictionary.size());
      if(sentiment==1){
        wordProbs.put(word+"^+",prob);
      }
      else if(sentiment==-1){
        wordProbs.put(word+"^-",prob);
      }
    }
  }

  //Tests reviews. Sentiment is 1 for postive, -1 for negative.
  static private double test(List<List<String>> reviews, int sentiment){
    int reviewCount = 0;
    int correctReviewCount = 0;
    BigDecimal pos= new BigDecimal("1.0");
    BigDecimal neg= new BigDecimal("1.0");
    for(List<String> review : reviews){
      reviewCount++;
      for(String word : review){
        if(dictionary.contains(word.toLowerCase())){
          BigDecimal probpos= new BigDecimal(wordProbs.get(word.toLowerCase()+"^+"));
          pos=pos.multiply(probpos);
          BigDecimal probneg= new BigDecimal(wordProbs.get(word.toLowerCase()+"^-"));
          neg=neg.multiply(probneg);
        }
      }
      if(pos.compareTo(neg)>0 && sentiment==1){correctReviewCount++;}
      if(pos.compareTo(neg)<0 && sentiment==-1){correctReviewCount++;}
      pos= new BigDecimal("1.0");
      neg= new BigDecimal("1.0");
    }
    System.out.println(correctReviewCount+" " +reviewCount);
    double correctProb=(double)correctReviewCount/(double)reviewCount;
    return correctProb;
  }

  public static void main(String[] args) throws IOException {
    if(args.length != 4){
      System.err.println("Usage: java <Positive_Train_Directory> <Negative_Train_Directory> <Positive_Test_Directory> <Negative_Test_Directory");
      System.exit(1);
    }
    makeDictionaryNegation(args[2],args[3]);
    List<List<String>> trainingReviewsPos = readReviewDataNegation(args[0]);
    List<List<String>> trainingReviewsNeg = readReviewDataNegation(args[1]);
    train(trainingReviewsPos,1);
    train(trainingReviewsNeg,-1);
    List<List<String>> testReviewsPos = readReviewDataNegation(args[2]);
    List<List<String>> testReviewsNeg = readReviewDataNegation(args[3]);
    System.out.println("Positive Reviews:");
    double posProb=test(testReviewsPos,1);
    System.out.println("Negative Reviews:");
    double negProb=test(testReviewsNeg,-1);
    System.out.println("Positive Review Accuracy: " + posProb);
    System.out.println("Negative Review Accuracy: " + negProb);
    System.out.println("Overall Accuracy: " + (double)Math.round(((posProb+negProb)/2.0)* 1000)/1000);
  }
}
```

NegationTokenizer Class

```
import java.util.regex.*;
import java.util.*;

class NegationTokenizer {

  public List<String> tokenize(String rawInput){
    // Write a regular expression for each token type
    String ordinal = "(\\d+(?:st|nd|rd|th))";// ordinal number e.g 1st
    String decimal = "([\\+\\-]?\\d+\\.\\d+|[\\+\\-]?\\d+(?:,\\d+)+)"; // decimal numbers and negative numbers and numbers with comma (ignore
1,000.0 form here)
    String number = "(\\d+)"; // int numbers
    String dollar = "(\\$\\d+(?:\\.\\d\\d)?)";

    String title =  "(?:(Dr\\.|Mr\\.|Mrs\\.|Sr\\.|Ms\\.|Jr\\.|Mx\\.|Fr\\.|Prof\\.|St\\.|U\\.S\\.)\\s)";// titles list and Abbreviations (with period at the
end) list. It's a finite list but not comprehensive in this case. e.g. U.S. Dr.
    String contractionsOrPossessives = "([a-zA-Z]+)(n't|'s|'ve|'d|'ll|'m|'re)"; // Contractions and Possessives e.g. mom's>> [mom, 's]    I've >> [I, 've]
    String word = "([a-zA-Z']+(?:-[a-zA-Z']+)*)"; // words

    String specialPunct= "(--)"; // special case when -- should be tokenized to one -- instead of two -
    String punctuation = "(\\p{Punct})"; // e.g. !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

    // Combine them into a single string
    String r = ordinal + "|" + dollar + "|" + decimal + "|" + number + "|" + title + "|" + contractionsOrPossessives + "|" + word + "|" + specialPunct + "|" +
punctuation;
    // Compile it to a Pattern
    Pattern p = Pattern.compile(r);

    // Create a list for the tokens
    List<String> tokenList = new ArrayList<String>();

    // Use a Matcher to loop through the input
    Matcher m = p.matcher(rawInput);

    int groupCount =  m.groupCount();

    while(m.find()){
      for(int k = 1; k <= groupCount; k++){
        if(m.group(k)!=null){ // modified sample code from m.group(1)!=null
          //System.out.println("Group " + k +": " + m.group(k)); // test purpose
          tokenList.add(m.group(k));
        }
      }
    }
    String [] tempWords = new String[tokenList.size()];
    tempWords=tokenList.toArray(tempWords);

    int negate=0;
    for(int i = 0; i<tempWords.length;i++){
      if(tempWords[i].equals(".")|tempWords[i].equals(",")
|tempWords[i].equals("!")|tempWords[i].equals("?")|tempWords[i].equals("(")|tempWords[i].equals(")")){
        negate=0;
      }
      if(negate==1){
        tempWords[i]="!"+tempWords[i];
      }
      if(tempWords[i].equals("not") | tempWords[i].equals("n't") |tempWords[i].equals("never")){
        negate=1;
      }
    }
    List<String> finalTokenList = Arrays.asList(tempWords);
    return finalTokenList;
  }
}
```

References

Bollen, J., Mao, H., & Zeng, X.-J. (n.d.). *Twitter mood predicts the stock market*.

Ding, X., Liu, B., & Yu, P. S. (n.d.). *A Holistic Lexicon-Based Approach to Opinion Mining*.

Koppel, M., & Schler, J. (n.d.). *The Importance of Neutral Examples for Learning Sentiment*.

Pang, B., & Lee, L. (2004). *A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts*.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (n.d.). *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*.