Identifying the Emotional Polarity of Song Lyrics through Natural Language Processing

Ashley M. Oudenne
Swarthmore College
Swarthmore, PA
aoudenn1@cs.swarthmore.edu

Sarah E. Chasins
Swarthmore College
Swarthmore, PA
schasi1@cs.swarthmore.edu

Abstract

The analysis of song lyrics can provide important meta-information about a song, such as its genre, emotion, and theme; however, obtaining this information can be difficult. Classifying a song based solely on its lyrics can be challenging for a variety of reasons, and as yet no single algorithm results in highly accurate classifications.

In this paper, we examine why the task of classification based solely on lyrics is so challenging by predicting the emotional polarity of a song based on the song's lyrics. We use natural language processing to classify a song as either positive or negative according to four sets of frequency-based or machine learning algorithms which we develop and test on a dataset of 420 songs. We determine which factors complicate classification, such as generic subjectivity lexicons and non-subjective lyrics, and suggest strategies for overcoming these difficulties.

1 Introduction

Lyrics provide high-level information about a song. Humans gather meta-information such as the genre, sentiment, and theme of a song simply by reading its lyrics. However, automating this task is very difficult.

In automatic music classification systems, research typically focuses on classification by audio features or collaborative filtering. However, these methods have numerous drawbacks. Audio feature processing relies on having the actual recording of a song, which may be difficult to obtain under copyright laws. Collaborative filtering can suffer from the long-tail problem: obscure songs are not likely to have adequate representation on social networking sites, which makes similarity analysis difficult.

Recently, researchers have turned to analyzing lyrics in order to extract information about a song. This information can be combined with audio features or collaborative filtering data or used alone in a music classification system.

Song lyrics are a good source of data because many lyrics are freely available in a semi-structured format on the Internet. However, they are difficult to analyze for sentiment and, as a result, many systems that rely on lyrical analysis alone for classification yield poor results.

In this paper, we will explore song lyric analysis by focusing on the simple yet non-trivial task of categorizing music by emotional polarity. We classify songs as positive or negative depending upon whether they express a mainly positive or negative overall emotion. We test different algorithms that analyze word frequencies, word presence, and cosine similarity to classify a dataset of 420 unique lyrics. We also evaluate the performance of generic versus corpus-specific subjectivity lexicons in order to determine how lyrics-based sentiment analysis differs from traditional sentiment analysis.

The rest of this paper is laid out as follows: in Section 2 we present related work in sentiment analysis and how it has been extended to song lyrics. In Sec-

tion 3, we introduce our data set and the methods we use to classify the data. In Section 4 we present our results and discuss different ways to classify lyrics by emotion in Section 5. Finally, in Section 6, we conclude by discussing future work in song lyric analysis.

2 Related Work

2.1 Sentiment Analysis

Identifying the primary sense of a document in a corpus is very useful in natural language processing. Knowing whether a document's sentiment is positive or negative overall can aid in classification tasks. To that end, Janyce Wiebe created a lexicon of subjectivity clues, such as *repressive* and *celebrate*, that are good indicators of positive or negative sentiment (Wilson et al., 2005). These words are tagged with their part of speech, polarity (positive, negative, neutral, or both), and the strength of their subjectivity (strong or weak). Many researchers make use of this lexicon for sentiment analysis, as do we.

Subjectivity lexicons like the one described above can be used to extract the overall sentiment of a document. (O'Connor et al., 2010) use sentiment analysis to classify Twitter tweets about the economy and the presidential election. They classify tweets as either positive or negative by counting the number of positive or negative subjectivity clues from Wiebe's subjectivity lexicon that occur in the tweet. They then use this classification to predict consumer confidence poll data. They note that Wiebe's subjectivity clues lead to many instances of falsely-detected sentiment, because the subjectivity clues are used differently in tweets than in the corpus that generated the clues. However, they were able to successfully predict polling data with tweet sentiment, in spite of the fact that they took no negation of sentiment words into account (e.g. "I do *not* think that the economy is good"). Based on their results, we have included sentiment negation into some of our algorithms.

Sentiment classification has also been successfully used to classify movie reviews. (Pang et al., 2002) use machine learning techniques to classify movie reviews as either positive or negative. They conclude that subjectivity lexicons must be corpus-specific. In an experiment comparing human-suggested sentiment word lists to a corpus-specific most-frequent word list, they achieved higher accuracy using corpus-specific words. This suggests that researchers should not take a "one-size-fits-all" approach to subjectivity lexicons like Wiebe's. A lexicon specific to the corpus and the task will perform better than a generic lexicon.

Additionally, they use machine learning algorithms such as Naive Bayes to perform sentiment classification. They use a bag-of-features framework where the features can be bigrams, unigrams, positional data, and part-of-speech data. Unigram features perform the best, and their performance increases as they are combined with other features. We also experiment with the Naive Bayes algorithm in our work.

2.2 Sentiment Analysis with Lyrics

Lyrical analysis for classification is a relatively new area of research. Due largely to sites such as *Lyrics.com* and *elyrics.net*, millions of lyrics are now available on the Internet to researchers in semi-structured formats that are amenable to web-scraping. Lyrics are typically analyzed as part of a music classification task, where songs are classified by genre, mood, or emotion (Cho and Lee, 2006; Hu and Downie, 2010).

(McKay et al., 2010) use lyrical features combined with audio, cultural, and symbolic features to classify music by genre. The find that lyrics alone are poor indicators of a song's genre, but that when lyrical analysis is combined with other features, their system is able to achieve high genre classification accuracy. However, some of the songs in their data set were instrumental, so there may not have been enough data for training. In our experiments, we use a larger data set that includes lyrics for every song.

Compared to sentiment analysis in movie reviews, we expect sentiment analysis to be more difficult for lyrics. Movie reviewers typically use opinion-words (e.g. "I *liked* this movie because...", "This scene was *horrible*") in their reviews, which can be extracted with relative ease with the proper subjectivity lexicon. Lyrics, however, are more challenging. There are three main difficulties with lyric-based sentiment analysis:

1. Songs can contain a series of negative lyrics but end on an uplifting, positive note, or vice versa. Love songs in particular can be misleading because the lyrics often express how happy the singer was while in love, and then at the end of the song the singer expresses his sadness over a sudden breakup.

2. Songs may not contain any of the subjectivity clues in a general subjectivity lexicon, yet express positive or negative emotions. For example, the song "It's Still Rock And Roll To Me" by Billy Joel includes the following stanza:

   What's the matter with the clothes I'm wearing?
   Can't you tell that your tie's too wide?
   Maybe I should buy some old tab collars?
   Welcome back to the age of jive.

   It's not immediately apparent which of the words in this stanza would have positive connotations; yet, taken together, the stanza expresses a positive emotion. This occurs in both positive and negative songs, and it can be difficult to separate the overall emotion of a song from the sentiments expressed by each line of its lyrics.

3. Songs can express positive emotions about negative things, and vice versa. Rap songs in particular suffer from this problem: their lyrics often express positive emotions about negative events like shootings and robbery. This adds an additional level of confusion to a classification system.

We address these problems in various ways in our different algorithms for sentiment analysis.

3 Experimental Methodology

3.1 The Dataset

We use a dataset of 420 unique song lyrics, divided equally between positive and negative emotional polarities (hereafter referred to as positive lyrics and negative lyrics).

3.1.1 Gathering the Data

In order to gather a dataset of unique song lyrics, we utilized Jamrock Entertainment's list of top 100 songs by year[1]. We mined the top-100 song lists for the years 1980-2009, and then searched for lyrics matching those songs from *Lyrics.com*[2]. We mined the lyrics and cleaned them of xml tags and other extraneous text. This created an initial data set of 1,652 lyrics.

3.1.2 Classifying the Data

To gather the ground truth emotional polarity labels for the song lyrics, we used Last.fm's developer API[3]. For each song in our initial dataset, we queried the Last.fm API and retrieved the user-specified top tags—semantic text descriptors— for that song. We then searched through the top tags to see if any of them occurred in Jan Wiebe's subjectivity lexicon. We kept a count of the number of positive and negative subjectivity clues that occurred in the top tags of the song, and then labeled the song as the emotion with the greatest number of subjectivity clues that occurred in the lyrics. We manually annotated the few songs that did not appear in the Last.fm database. This resulted in an intermediate dataset of 1,515 lyrics.

3.1.3 Equalizing the Data

It is necessary to equalize the dataset so that there is an equal number of positive and negative lyrics. This prevents classification systems from learning biases inherent in the dataset. When we attempted to equalize our dataset, we realized that the number of positive lyrics far surpasses the number of negative lyrics. This is to be expected, because we relied on top-100 songs to create our dataset, and top-100 songs are more likely to express positive emotions. Consequently, our equalized data set consists of 420 songs, 210 of which are labeled 'positive' and 210 of which are labeled 'negative'. These labels were manually checked by a human annotator.

3.2 Word Lists

We test a variety of algorithms on our dataset. The first and simplest algorithm is based on straightforward word counts. A classification system built on this approach requires very little information, nothing more than two lists of words. The system begins by segregating the training set into positive songs and negative songs. Next, it creates a list of the

[1] http://www.jamrockentertainment.com/billboard-music-top-100-songs-listed-by-year.html

[2] http://www.lyrics.com

[3] http://www.last.fm/api

words that appear in the positive set, and a list of the words that appear in the negative set. The lists are ordered by word frequencies in their respective classes. A maximum list size n is passed to the algorithm, which is selected by the researchers. All positive words that are not among the n most frequent positive words are discarded, as are all negative words that do not appear among the n most frequent negative words. From these two lists of size n, the algorithm removes any word that appears in both lists. (Note that even if a word is in first place in the positive list and nth place in the negative list, the word is removed from both lists.) Thus, no words are shared. The two lists that remain at this stage are the only resources necessary for classification.

Each song is classified, in this method, by keeping separate counts of the number of times a word from each list appears in the lyrics. The algorithm loops through the song's words. Each time it encounters a word from one of the lists, it increments the appropriate count, even if the word has already appeared elsewhere in the lyrics. The song's sentiment is predicted simply by comparing these two counts. If the positive word count is higher, the system predicts that the song is positive.

3.3 Word Dictionaries

The second variety of classification algorithms relies on retaining more information. It uses knowledge of, at the very least, every word that appears in the training set and in which class of lyrics it appears. One algorithm in this category requires no more information than that. The Present In One algorithm is provided with a dictionary of all words that appear in positive lyrics, and a second dictionary of all the words that appear in negative lyrics. To classify a song, it loops through the words, checking whether each word is in both dictionaries, neither dictionary, or in only a single dictionary. The first two cases are ignored. If, however, the word appears in only one dictionary, the count for that sentiment is incremented. That is, if the word appears in the negative dictionary but not the positive dictionary, a point is added to the negative score. If, in the end, the negative score is higher than the positive score, the song is classified as negative.

The next algorithm in this class still requires knowledge of every word in the training set, but also needs a record of the frequency in the training set. More specifically, it needs to know the percentage of times a word appears in each list. The Prevalent In One algorithm also keeps positive and negative scores for each song. A threshold t is passed into this algorithm. A word in the lyrics increments the positive score if its occurrences in the positive training data divided by its occurrences in the negative training data exceeded t. That is, if count(word|positive)/count(word|negative)>t. After all words have been examined, the song is labeled with the sentiment of the higher score.

The final member of this grouping was a traditional approach to classification problems, a simple Naive Bayes model. To calculate P(sentiment|lyrics), the algorithm calculates P(lyrics|sentiment)*P(sentiment)/P(lyrics). Since there was no need to compare across multiple lyrics, but only to compare the probabilities of different sentiments for the same lyrics, P(lyrics) would have no effect on the comparisons under consideration. Thus, P(lyrics|sentiment)*P(sentiment) became the essential calculation. The lyrics are represented by the series of n words that constitutes the lyrics, and the algorithm makes the naive assumption that the probability of seeing any given word depends solely on the sentiment, and not at all on the presence of other words in the lyrics. That is, it assumes that P(w_k|sentiment) is the same value as P(w_k|sentiment, w_1^{k-1}). The final probability is therefore:

$$\text{P(sentiment|lyrics)}=\prod_{i=1}^{n}\text{P}(w_i|\text{sentiment})$$

The probability of w_i conditional on sentiment is calculated using dictionaries of words and their frequencies in the positive and negative training data. If P(negative|lyrics) is greater than P(positive|lyrics), the song is classified as negative.

3.4 Cosine Similarity

We also attempt to classify song lyrics by clustering. In this algorithm, we calculate the inverse document frequency (idf) of a training set of lyrics. For each word in the training set, we define the idf of that term as $idf_i = \log \frac{|D|}{|d:t_i \in d|}$, where *D* is the training set of song lyrics, *d* is a song lyric document in the training corpus, and *i* is the index of the current

term *t* in document *d*. Idf measures the importance of a term— terms that occur in many song lyrics are down-weighted, and terms that are rare are up-weighted.

For every word in each document in the entire corpus (both training and test), we then compute the term frequency of the word. The term frequency (tf) is defined as $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, where $n_{i,j}$ is the number of times a specific term t_i appears in document d_j, and the denominator is the number of tokens in d_j. Tf measures the importance of a term within a given document.

For every document, we then compute the tf-idf score of the document by multiplying the tf score by the idf score for every term in the document: $(tf - idf)_{i,j} = tf_{i,j} * idf_i$. This up-weights terms that occur frequently in a specific document yet occur infrequently in the corpus as a whole. We use the tf-idf score to construct a vector of terms and their tf-idf score for every document in the corpus.

We then calculate the cosine similarity between each document in the test corpus and every document in the training corpus. Cosine similarity is defined as $cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$, where *A* is a document in the test corpus and *B* is a document in the training corpus. After the cosine similarity is calculated, we take the nine training songs with the highest similarity scores to the test song. We then classify the test song based on the polarity of the majority of those nine songs: if the majority of the songs are positive, the test song is labeled as positive, and vice versa. We use 5-fold cross-validation to test the entire dataset.

In another experiment, we replace the tf-idf weight with a word-sense similarity weight. We use the WordNet similarity package to calculate the path similarity of each word in every document in the corpus to the words "happy" and "sad" (Miller et al., 1993). We then take the ratio of the similarities, so that the weight of each term t_i in document d_j can be expressed as $weight_{t,i,j} = \frac{pathSimilarity(t_i, \text{“happy”})}{pathSimilarity(t_j, \text{“sad”})}$. In our experiments, we calculated the similarity by first calculating the synsets— sets of synonyms— of each term t_i and the emotion words "happy", and "sad". We then use the first sense of the emotion words and compare it to every sense in the term's synset. We choose the sense that maximizes the similarity between the term and the emotion words, and use this to calculate the ratio between the two sense scores. We place these values in a vector and calculate the cosine similarity as described above.

4 Results

4.1 Exploratory Statistics

Before attempting to classify our documents, we evaluate the claim that a subjectivity lexicon should be corpus-specific, or created based on statistics from the corpus on which it will be used (Pang et al., 2002; O'Connor et al., 2010). For these experiments, we use Jan Wiebe's subjectivity clues (Wilson et al., 2005). Of her 8,221 subjectivity clues, we extract only the clues marked "strongsubj", the words that are strong indicators of subjectivity. We also only extract words with positive or negative polarity, but ignore words with a neutral polarity or words that are marked as both positive or negative. We remove repeated instances of the same word with different parts of speech, since we do no part-of-speech tagging. This results in 4,746 subjectivity clues. We examine the prevalence of these subjectivity clues in our dataset to see if they are useful in classifying the polarity of song lyrics. Table 1 shows the results of this experiment.

It is interesting that while there are about twice as many negative subjectivity clues as there are positive clues, only 6.67% of the dataset includes a majority of these words, ignoring duplicates. In other words, the best negative lyric recall we will ever achieve using unique negative subjectivity clues is 0.0667. This suggests that at least the negative portion of the subjectivity lexicon is inappropriate for lyrics.

We evaluate the subjectivity lexicon in a basic classification task. Given a training set, we calculated the 30 positive and 30 negative subjectivity clues that occur most frequently in the training set. Examples of these clues can be found in Table 2. We then use these clues to classify a test set. For each song in the test set, we count the occurrences of the positive and negative clues. If the positive clue count is highest, the song is classified as positive, and vice versa.

We compare the performance of these generic subjectivity clues to clues generated from the training set. We calculate the most frequently occurring

Table 1: Baseline Statistics on Our Dataset. The dataset includes 420 unique song lyrics, 210 labeled 'positive' and 210 labeled 'negative'.

Number of Strong Subjective Clues	
Pos. 1671	Neg. 3075
Lyrics with 1 or More Subjectivity Clues with Same Polarity as Song's Polarity	
Pos. 0.9952	Neg. 0.9476
Lyrics with Majority of Non-Unique Subjectivity Clues with Same Polarity as Song's Polarity	
Pos. 0.9619	Neg. 0.0762
Lyrics with Majority of Unique Subjectivity Clues with Same Polarity as Song's Polarity	
Pos. 0.9429	Neg. 0.0667

Table 2: A Comparison of Generic and Corpus-Specific Subjectivity Clues

	Pos.	Neg
Generic	love, know, baby, want	long, cry, crazy, sorry
Specific	oh, hey, wanna, will	but, was, back, duuh

words in positive lyrics and negative lyrics. For the positive words, we discard any word that occurs less than 1.5 more times for positive words than for negative words, and we do the same for negative words. We set this parameter low so words that occur in both positive and negative songs, such as "promises", are counted more heavily than song-specific words (e.g. "fergalicious"), which may occur frequently in only one document. 1.5 was chosen after experimenting with values between 1 and 20, and 1.5 was selected because it discarded overly frequent words without penalizing common sentiment indicator words. We then select the 30 most frequent positive and negative clues and classify each song in the training set as described above. Examples of these words are listed in Table 2. The results of these experiments are shown in Table 3. Both experiments were conducted using 5-fold cross-validation.

Table 3: Comparison of Lyric Classification Using Generic Subjectivity Clues and Corpus-Specific Subjectivity Clues

	Pos. Prec.	Neg. Prec.	Pos. Rec.	Neg. Rec.	Acc.
Generic	0.50	1.0	1.0	0.01	0.50
Specific	0.54	0.54	0.50	0.59	0.54

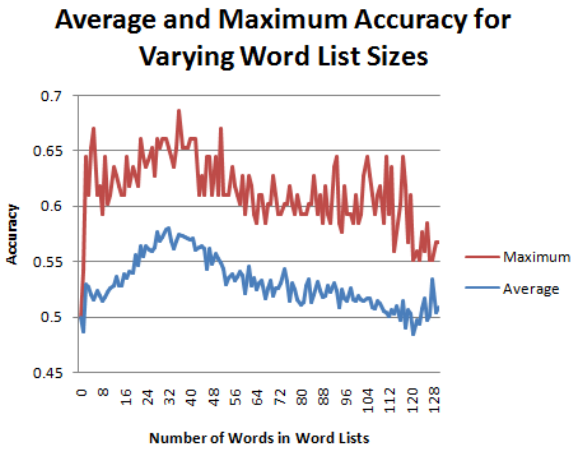

Figure 1: Average and maximum accuracy from 100 trials of the Word List algorithm.

While generic subjectivity clues had perfect negative precision, they had incredibly low negative recall (0.01). This confirms the results of our first experiment, in which we determined that generic subjectivity clues cannot attain a negative lyric recall of higher than 6.67%. However, positive precision was 0.5 and positive recall was 1.0, suggesting that the generic positive subjectivity clues are useful in classifying documents.

Corpus-specific subjectivity clues performed slightly better. Although negative precision and positive recall dropped by half, negative recall rose to 0.59, and the accuracy increased to 0.54. This demonstrates that even a corpus-based subjectivity lexicon created with a simplistic algorithm will generally outperform generic lexicons.

4.2 Word Lists

We run two different versions of the Word List algorithm. In the first version, each space-separated word is treated as its own entry in the lists. Punctuation marks (e.g. periods, commas, and quotation marks) before or after any alphabetic characters are removed and entered as separate entries. However, non-alphabetic characters that appear between alphabetic characters (such as internal apostrophes or hyphens) remain within the words. Thus:

She didn't say "yes."

would be entered as:

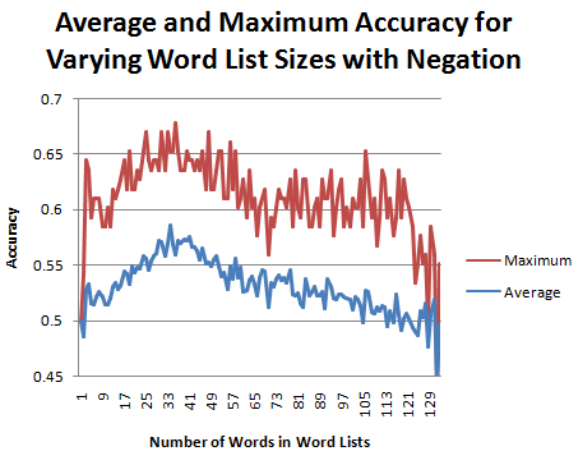

Figure 2: Average and maximum accuracy from 100 trials of the Word List algorithm with simple negation.

She / didn't / say / " / yes / . / "

The second version of the Word List algorithm treats punctuation in exactly the same way. However, it is designed to control for negation by a very simple mechanism. In the Negation Word List, any word that follows "no" or "not" is appended, and the two words are included in the list as a single entry. Thus:

I am not happy.

would be entered as:

I / am / not happy / .

It is also important to note, however, that the sentence:

I am not very happy.

would be entered as:

I / am / not very / happy / .

These two versions of the Word List algorithm are trained on 150 positive songs and 150 negative songs. They are then tested on 60 positive songs and 60 negative songs. This training and testing process is repeated 100 times. Each of the 100 times, the 210 songs from each class are randomly shuffled, so that the training and testing sets vary from trial to trial.

Casual inspection of early data revealed that the highest accuracies were achieved with maximum list sizes below 500 words. We thus run the algorithm for maximum list size thresholds from 0 to 490, at increments of 10 words, repeating the algorithm 100 times for each threshold value. Keeping track of the accuracy of each trial's classifier yields average and maximum accuracies for word lists of varying sizes. Recall from Section 3.2 that with a maximum list size of n, we keep the n most frequent words from the positive list, and the n most frequent words from the negative list. We then eliminate words that appear in both. Because words common in one are often common in the other, the final word lists are generally much smaller than the maximum list size. In the 200 trials (100 without negation, and 100 with negation) that used a maximum list size of 490, there is never a final list with more than 136 words. In the no-negation condition, there are multiple data points for each final list size up to 131 words. In the negation condition, there are multiple data points for each final list size up to 133 words.

The no negation results appear in Figure 1. The results with negation appear in Figure 2. In the no-negation condition, the highest average accuracy is 58.0%, with a final list size of 32. The highest maximum accuracy is 68.6%, achieved with a final list size of 36. In the negation condition, the highest average accuracy is 58.53%, with 33 words in the final list. The highest maximum accuracy is 67.7%, with a final list size of 35.

4.3 Word Dictionaries

The three algorithms that require knowledge of every word in the training set are run under four different conditions. Unlike the Word List algorithm, these algorithms do not all automatically disregard items common to both positive and negative songs. It is possible that altering classifications based on such words, since they appear so frequently and affect any given song significantly, could skew results. We thus introduce an ignore list condition, in which the algorithm maintains a list of very common words, and does not allow these words to alter a song's classification.

A third condition again takes negation into account, operationalizing it in the same manner as in the Word List algorithm. The fourth and final condition utilizes both negation and the ignore list.

Each of the three algorithms is run 100 times in each of these four conditions. Average and maximum accuracies are collected over the course of the

Table 4: Average Accuracy for Basic, Ignore List, and Negation Conditions

	Basic	Ignore List	Negation	Ignore and Negation
Present In One	0.487542372881	0.487966101695	0.492372881356	0.492711864407
Prevalent In One	0.566016949153	0.566440677966	0.55813559322	0.562372881356
Naive Bayes	0.557881355932	0.562627118644	0.553728813559	0.557372881356

Table 5: Maximum Accuracy for Basic, Ignore List, and Negation Conditions

	Basic	Ignore List	Negation	Ignore and Negation
Present In One	0.576271186441	0.576271186441	0.576271186441	0.576271186441
Prevalent In One	0.669491525424	0.661016949153	0.661016949153	0.652542372881
Naive Bayes	0.635593220339	0.64406779661	0.64406779661	0.652542372881

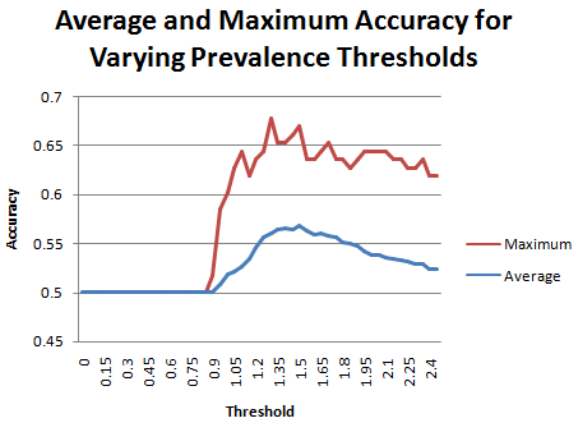

Figure 3: Average and maximum accuracy from 100 trials of the Prevalent In One algorithm.

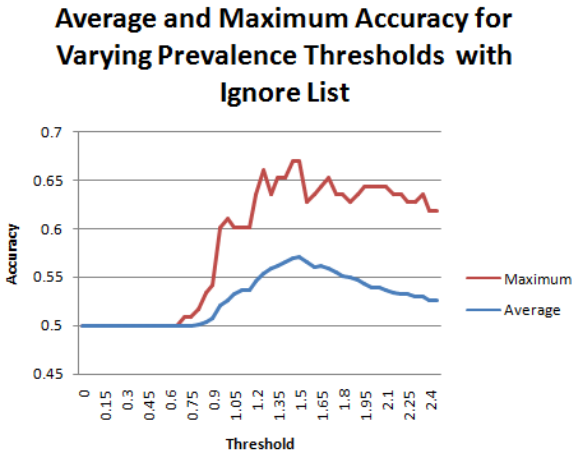

Figure 4: Average and maximum accuracy from 100 trials of the Prevalent In One algorithm with the ignore list.

100 runs. The average accuracies for each condition appear in Table 4, and the maximum accuracies appear in Table 5.

Of the algorithms that require knowledge of every word in the training set, only the Prevalent In One algorithm requires a parameter selected by the researcher. One important task, before the above experiments were run, was to select the best possible value for this threshold. It was evident from early data that negation made no significant improvements in the performance of the Prevalent In One algorithm. Whether the ignore list had an impact was less clear. We decided to run the basic Prevalent In One algorithm and the ignore list Prevalent In One versions 100 times each, for each of fifty thresholds. The thresholds range from 0 to 2.45 by increments of .05. Average and maximum accuracies are collected for each threshold value. The accuracies for the basic version can be seen in Figure 3, and the accuracies for the ignore list version are found in Figure 4.

In the basic version of the algorithm, the highest average accuracy is 65.5%, achieved with a threshold of 1.4. The highest maximum accuracy in this condition is 67.0%, achieved with a threshold of 1.5. In the ignore list condition, the highest average accuracy is 57.1%, when using a threshold of 1.5. The highest maximum is 67.0%, with threshold values of 1.45 and 1.5. We elected to use the high average accuracy thresholds for further testing of this algorithm, in which it is compared to the other two. Therefore, a threshold of 1.4 is used for the two conditions that lacked the ignore list, and a threshold of 1.5 is used for the two conditions that utilized the

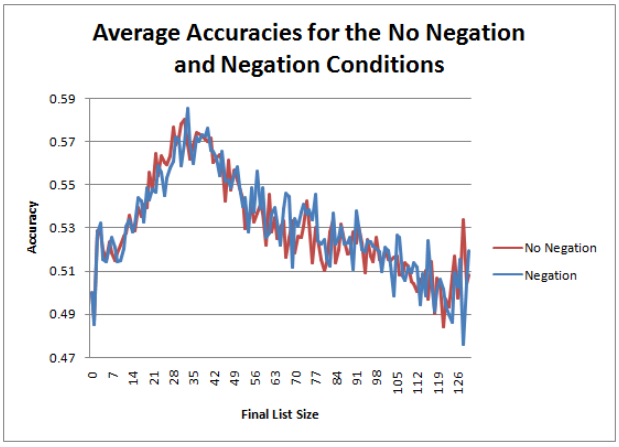

Figure 5: Average accuracy, by list size, for Word List with and without negation over 100 trials.

ignore list.

4.4 Cosine Similarity

The results of five-fold cross-validation on the dataset using cosine similarity can be see in Table 6. Cosine Similarity using tf-idf weighting performs about as well as the algorithms in Table 4, with an accuracy of 0.57. Positive recall was lower than the other values in the table, which is somewhat surprising considering that, according to the experiment described in Section 4.1, there are more positive subjectivity clues in our corpus than negative ones, which should make it easier to identify positive lyrics. However, since we did not use the Wiebe subjectivity lexicon in this experiment, it is possible that the algorithm is identifying negative terms that are more meaningful for classification than the ones in the subjectivity lexicon. The WordNet algorithm performs worse than the tf-idf algorithm, with an accuracy of 0.52 and precision and recall across both classifications of about 0.5.

Table 6: Cosine Similarity Results

	Pos. Prec.	Neg. Prec.	Pos. Rec.	Neg. Rec.	Acc.
TF-IDF	0.59	0.56	0.49	0.66	0.57
WordNet	0.52	0.51	0.52	0.53	0.52

Table 9: Positive and Negative Word Lists for Varying Maximum List Sizes

Max	Final	Pos. Words	Neg. Words
30	3	baby, we, just	but, no, when
40	5	oh, this, can, one, got	no, when, what, up, out
70	3	go, come, I'll	back, not, they
100	12	gonna, will, man, from, away, hey, too, life, every, day, more, ”	!, why, duuh, won't, her, he, could, ya, little, about, that's, boy

5 Discussion

5.1 Exploratory Statistics

Based on the results in Table 2, it initially seems as if generic subjectivity clues should generate more accurate classification results. Words like “sorry” and “baby” are more useful to humans than words like “oh” and “duuh” when they are faced with classification tasks. However, the list of corpus-specific words performs better. (Pang et al., 2002) saw similar results when they attempted to classify movie reviews based on human generated subjectivity clues and corpus-based subjectivity clues. Regardless of how unhelpful they seem to humans, corpus-based clues provide useful data to a classification algorithm.

The improvement gained by using corpus-specific words, however, is minimal. We used a very simplistic algorithm to generate the corpus-specific lexicon, which probably resulted in the relatively low accuracy. However, we expect that better subjectivity-lexicon generation algorithms will result in better classification accuracy, and it is encouraging that an algorithm as basic as ours results in improvement over using generic subjectivity clues.

5.2 Word Lists

The Word List algorithm was moderately successful. With word list sizes that reliably yield 57-58% accuracy, and which can occasionally reach even 68% accuracy, this algorithm compares favorably with the other methods explored in this paper. However, from the gap between maximum and average performance, hovering at approximately 10%, it is clear that performance is extremely variable.

The variability of the accuracy suggests several facts about the Word List algorithm. First, since only composition of the training and data set varies from trial to trial of a single threshold, it seems clear that the contents of the training set has a vast influence

Table 7: Accuracies from Sample Run 1

	Basic	Ignore List	Negation	Ignore and Negation
Present In One	0.5	0.5	0.508474576271	0.508474576271
Prevalent In One	0.64406779661	0.635593220339	0.64406779661	0.635593220339
Naive Bayes	0.627118644068	0.661016949153	0.627118644068	0.661016949153

Table 8: Accuracies from Sample Run 2

	Basic	Ignore List	Negation	Ignore and Negation
Present In One	0.525423728814	0.525423728814	0.516949152542	0.516949152542
Prevalent In One	0.669491525424	0.64406779661	0.627118644068	0.669491525424
Naive Bayes	0.610169491525	0.610169491525	0.593220338983	0.601694915254

on the accuracy of the resulting classifier. This is not surprising, but it does suggest that performance could be tremendously improved by access to additional training data.

The comparison of average negation and no-negation Word List accuracies in Figure 5 demonstrates that there appears to be no significant effect of including negation in the algorithm. The negation version performs better in some trials and worse in others. And, while negation has the slightly higher average accuracy peak at 58.5%, the no-negation condition has the higher maximum accuracy with a peak of 68.6%. In both cases, the difference between the peaks is trivial. Since negation also does not harm performance, it seems possible that the compound entries formed by “no” and “not” and the words that follow them are rarely frequent enough to be included in the final list of frequent words. While this is true for trials with low maximum list sizes, this is not the case with maximum list sizes of greater than 200. In several runs with multiple list sizes, the entries “no one”, “no more” and “no matter” appear in multiple final lists, beginning at a maximum list size of 230 to 380. In other runs, “not goin” appears in the lists beginning with maximum list sizes of approximately 340.

Considering negation’s apparent lack of effect on performance, it is interesting to note that “no” and “not” do appear in many of the final lists in the no-negation condition. With maximum list sizes of 70 to 80, “not” becomes a common entry, and “no” consistently appears with maximum list sizes between 30 and 50. Their short-lived presence in the lists presumably explains the fact that their absence in the negation condition matters very little.

Examination of the lists used to distinguish between positive and negative songs does provide several insights into the results. Table 9 gives some sense of the number of frequent words that appear on both the positive and negative lists. In the run depicted in the table, 67 of the words in the 70 most frequent positive and negative words appear in both lists. Only three words from each list are specific to a particular sentiment.

It also becomes clear that words are eliminated from the final lists very quickly, indicating that the frequency rankings of those words are very similar in the negative and positive lyrics lists. Notice that when the maximum list size goes from 30 to 40, bringing into consideration only 10 more words from each list, the new lists are composed entirely of new words, save for “no” and “when” in the negative list. This demonstrates that “baby”, “we”, and “just” - all among the top 30 positive words - are also among the top 40 negative words. Thus, even words that can be found to appear more frequently in one set will appear only barely more frequently.

The presence of extremely song-specific words, such as ”duuh” and ”ya,” which are unlikely to appear with the same spelling in multiple songs, indicates the need for a larger training set. The final word lists should reflect the entirety of the sentiment class, and not be overly influenced by the unusual words of any individual song. The fact that this word appears already in the 100 most frequent words suggests that many of the words that enter consideration after the 100-word level may also be similarly impractical for classifying other songs. This is in fact

the case, with words such as ”mmm”, ”t”, ”1”, and ”da” appearing in later lists with greater maximum list sizes.

Interestingly, however, these larger lists do begin to reflect prototypical notions of the types of words that a positive and negative list might contain. For instance, with a maximum list size of 290, one positive list includes such words as: “kiss”, “sweet”, “friends”, “together”, “music”, and “sugar.” The negative list for the same run includes: “miss”, “gone”, “leave”, “left”, “mean”, “nobody”, “fool”, “tears”, “hurt”, “goodbye”, “broken”, “cold”, “gun”, and “pain.” These lists, however, with final sizes of 68, are significantly above the optimal level of final list size. It seems likely that, while these words would be excellent for identifying the sentiment of a song that contains many of them, such uncommon, specific words may not appear in enough songs to be useful for categorizing all of the test lyrics.

5.3 Word Dictionaries

It is clear again that there are large differences in the average and maximum accuracy values for the Word Dictionary algorithms, as evident in Tables 4 and 5. This brings to the fore the central role of training data in determining the accuracy of a classifier and suggests the possible improvements that might be accomplished with additional data.

Examining the average performance in Table 4 reveals that Present In One is an extremely ineffective algorithm. Prevalent In One and Naive Bayes, on the other hand, yield good results, with Prevalent In One slightly outperforming Naive Bayes in most cases. The accuracy differences between the two are unlikely to be significant.

The best average accuracy is found with Prevalent In One, in the ignore list condition. Negation appears to slightly harm Prevalent In One performance, with or without the ignore list. Naive Bayes also shows very slight losses in accuracy with the introduction of negation, while Present In One appears to benefit by it. The Present In One performance is on average boosted by approximately 1 percentage point through the use of negation, although its maximum accuracy appears to be entirely independent of condition.

The use of an ignore list provides a very slight advantage in all three algorithms, and a slightly larger advantage in the Naive Bayes approach. Again, the differences in performance are so slight as to be almost certainly insignificant.

In Tables 7 and 8 we see the algorithms' performance in runs on the same training and test sets. The performance of the three algorithms appears to covary, with high performance of one predicting high performance of the others. The pattern of best results is not, however, always consistent with the average accuracies. In Run 1, Naive Bayes performs better than Prevalent In One in two conditions. In Run 2, Prevalent In One outperforms Naive Bayes in all conditions. In some runs, Naive Bayes outperforms Prevalent In One under all conditions.

Ultimately, it appears that these three algorithms are best used with an ignore list, and depend on receiving a particular distribution of data to create a high-performing classifier. Of the three, Prevalent In One is the most reliably accurate, trailed immediately by Naive Bayes. The Present In One algorithm is unproductive, performing on average worse than a single-sentiment classifier, and worse than random chance.

5.4 Cosine Similarity

The cosine similarity algorithms suffer from the third lyrics analysis problem described in Section 2.2— they often miscategorize songs that express positive emotions about negative events or words. For example, in ”Whatta Man” by Salt-N-Pepa, the following stanza is problematic:

> My man gives real loving, that's why I call him Killer
> He's not a wham-bam-thank-you-ma'am, he's a thriller
> He takes his time and does everything right
> Knocks me out with one shot for the rest of the night

In this stanza, the words “Killer”, “thriller”, and “shot” have negative connotations, and one would expect them to appear most frequently in negative documents. However, the singer is using these words in a positive manner, which the cosine similarity algorithm cannot model. The overall emotion expressed by this stanza–and this song in general– is positive. The WordNet algorithm has particular difficulty with this problem because it uses the highest similarity score in its ratio calculation. The path distance score of “thriller” and “sad” will be much higher than the score of “thriller” and “happy”, which will result in misclassification.

Cosine similarity algorithms also suffer from the second problem with lyrics analysis described in Section 2.2— the words in a song may have no obvious polarity, yet the song expresses a polar emotion. In Will Smith's "Miami", the following stanza contains mainly neutral words:

Hottest club in the city and its right on the beach.
Temperature, get to ya', it's about to reach
Five hundred degrees in the Caribbean seas
With the hot mommies screaming "Ayy papi"

There is no word in this stanza that specifically expresses a polar emotion. "Hot" and "hottest" can be either positive or negative, and the rest of the words are relatively neutral. The only exception is the phrase "Ayy papi", which is not present in any other document in the corpus, so tf-idf weighting is useless in identifying it as a positive emotion indicator. Additionally, most of these words are not at all related to either "happy" or "sad", so the WordNet algorithm has difficulty classifying songs like this one.

However, cosine similarity provides good results when the song lyrics exhibit the first problem of misleading initial lyrics from section 2.2. "Always" by Bon Jovi demonstrates this problem. The beginning of the song includes lines like "It's been raining since you left me" and "But without you I give up", which might lead a human reader to expect the song to express a negative emotion. However, it ends on a positive note:

Well, there ain't no luck in these loaded dice
But baby, if you give me just one more try
We can pack up our old dreams, and our old lives,
We'll find a place, where the sun still shines

The overall tone of the song is positive, in spite of the negative initial lyrics. The tf portion of the tf-idf weighting that we use in this algorithm down-weights the repeated negative words and up-weights the less frequent positive words, so that both can be fairly compared to other documents. As a result, tf-idf weighting is more useful for a classification task than WordNet similarity scoring according to our experiments.

6 Conclusion and Future Work

Lyric sentiment analysis is not an easy task. Lyrics are more difficult to analyze than traditional sentiment analysis corpora, such as movie reviews, because they often express an emotion without using words that are sentiment-laden. Classifiers that rely on the presence of subjectivity clues perform poorly in this situation because of the paucity of subjectivity clues in most lyrics. Additionally, lyrics sometimes begin by expressing one emotion and then conclude with the opposite emotion. This complicates frequency-based analysis techniques because the first emotion is usually more prevalent, and causes the system to classify based on the incorrect emotion. A final problem with lyrics is that they express opposite emotions than the connotation of the lyrics would lead an algorithm to expect. Many songs in recent years use words like "shoot" and "killer" in a positive way, which makes modeling these usages difficult for a system that trains on data from different time periods.

As a result, our accuracies are lower than are typically seen in other sentiment analysis tasks. Movie review tasks in particular are able to achieve high accuracy. However, our results are consistent with (McKay et al., 2010), leading us to conclude that while lyrics may improve a music classification system that relies on additional data, current analysis methods limit their effectiveness when they are used alone for classification

In our experiments, we determined that, in addition to the problems listed above, the lexicon of typical songs is different than that of a generic lexicon. For this reason, using a generic subjectivity lexicon to classify songs results in worse accuracy than using a corpus-specific one. The lexicon that we generated for our experiments was created with the most basic statistical analysis, and we expect that if it were improved, the accuracy of our classifications would increase. This is something that we would like to explore at a later date.

Algorithms that consider the prevalence of terms seem to do better than presence-based algorithms. We see improvements in our classification when we consider the frequency of terms in positive or negative documents, rather than just their presence for two main reasons. The first is that prevalence-based algorithms do not ignore words that occur in both positive and negative documents, since this excludes most of the words in each song class. The second reason is these algorithms have a threshold parame-

ter that could be tuned to the corpus. For our dataset, we achieve the best accuracy with a threshold of about 1.5; for other corpora, a different value may be more appropriate.

There is still much work to be done in lyrical analysis. Our current implementation of the Naive Bayes algorithm takes as features the words in the lyrics. However, the words themselves may not be the best features to model the emotional polarity of songs. Perhaps the number of syllables, the average length of each line in the lyrics, or the number of repeated words could also provide useful information in a classification task. It would be interesting to see if training the Naive Bayes classifier on these features in isolation would result in more accurate classifications, or if combining these new features with our current implementation would be better.

In fact, many of our algorithms were limited to analyzing only words. Many other language features could be relevant to this classification task. For instance, the song length, the repetition of word series, the use of proper nouns, the number of slang words, and any number of other characteristics could all prove to be predictive metrics, which we did not explore. Incorporating these features and many others could vastly improve our algorithms' performance. The reduction of a set of lyrics into only a series of words is a naive simplification.

Additionally, algorithms such as Word List, Naive Bayes, and Prevalent In One are tested using the full sequence of words in the lyrics as they appear in the song. It would be interesting to use similar approaches with simplified song representations that remove repeated words, to provide only a list of the words that appear in the song. Classifying songs based entirely on the kind of words that appear in them would be a challenging and revealing task. Conversely, utilizing word order, or knowledge of word placement within a song, could conceivably simplify the classification task. Consider, for example, the possible effects of identifying which part of a song constitutes the chorus, and whether treating the chorus differently from the remaining lyrics could improve accuracy. Perhaps words in the chorus should be down-weighted because they are more frequent than the other words, or perhaps they should be up-weighted because they are important enough to be a part of the repeated chorus. emotion of the song

In song classification, as in so many areas of sentiment classification, negation proves a difficult phenomenon to model. Our approach to negation notably worsened performance in at least one subclass of algorithms. Accurately measuring emotion in lyrics may require additional advances in techniques for accounting for negation. Whether negation should be modeled differently in song lyrics than in other texts remains to be seen, and could prove an intriguing topic for future research.

We are confident that lyrical analysis for music classification will one day achieve the accuracies of other classification tasks. Until that time, however, more research needs to be conducted on extracting and analyzing the features that are salient to classification.

7 Acknowledgements

We would like to thank Dr. Richard Wicentowski for his guidance on this project.

References

Young Hwan Cho and Kong Joo Lee. 2006. Automatic affect recognition using natural language processing techniques and manually built affect lexicon. *IEICE - Transactions on Information and Systems*, E89-D(12), December.

Xiao Hu and J. Stephen Downie. 2010. Improving mood classification in music digital libraries by combining lyrics and audio. In *JCDL '10: Proceedings of the 10th Annual Joint Conference on Digital Libraries*.

Cory McKay, John Ashley Burgoyne, Jason Hockman, Jordan B. L. Smith, Gabriel Vigliensoni, and Ichiro Fujinaga. 2010. Evaluating the genre classification performance of lyrical features relative to audio, symbolic, and cultural features. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, pages 213–218.

George A. Miller, Claudia Leacock, Randee Tengi, and Ross T. Bunker. 1993. A semantic concordance. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 303–308.

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From tweets to polls: Linking text sentiment to public opinion time series. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, May.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, July.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *HLT '05 Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*.