**Introduction**

Due to the proliferation of content throughout the Internet today, and the rise of social media allowing anyone to post whatever they want, there has been a strong movement to develop methods that discern what information is relevant to users. The importance of this idea is clear: without filtering out unnecessary content, it can be difficult to find trends or meanings among these troves of data. Thus, big data manipulation is becoming more important every day, as content flows in from across the world from 7 billion plus potential sources.

One of the primary applications of big data manipulation is sentiment analysis. As much of the data from different users revolves around personal experiences, through Facebook posts or Instagram pictures, there is a growing collection of personal opinions, which is confused for factual information. The goal of sentiment analysis is to separate these types of data and categorize the different opinions that are being shared according to the different emotions they represent or express. Not only does this help separate all the different available types of data, but it also allows us to apply meaning to these datasets and understand what’s driving these trends through careful analysis.

Numerous different systems have been developed to help solve the problem of big data, and the number of different applications for sentiment analysis is increasing. The most notable is the Google search engine, which filters out noisy content from certain keyword searches to select the most relevant data. Additional systems include the application Shazam!, which searches for the current song being played through its database of songs, or even speech recognizers like Microsoft’s Cortana, which detect changes in pitch to discern a person’s happiness or sadness.

**Problem Definition**

Sentiment analysis reflects the core tenet of computational linguistics, which is to analyze linguistic data and produce relevant conclusions based on the content of the data. Specifically, it implores the use of machine learning algorithms to determine the writer’s opinion towards the content they’re writing about.

Sentiment analysis in the context of this project revolves around determining whether different reviews are positive, negative or neutral for a variety of different topics. These reviews, which are collected within the Cornell movie review dataset, comprise a variety of different opinions. The analyzing system, therefore, must be able to both parse and categorize these reviews by determining which type of sentiment category each review belongs to.

The input to this sentiment analysis system could be a dataset of different sentiments, of all kinds of different types. Depending on design choice, the system will employ a specific machine learning algorithm that best categorizes the data, and therefore the output will be the data separated into these different categories.

**Previous Work**

Sentiment Analysis has been attempted by many different research groups across several different applications. The most common application has been to parse a dataset that contains a wide variety of different sentiments, such as movie reviews or Twitter results. Because these media are open to the public, there is potential for different classifications, and thus models can be well-tested for their performance across such a large variety of data.

In addition, many companies use this as part of the product review process, as in the star rating system. Similar to the Yelp review dataset that is used for the extra credit portion, ratings like these help store owners to determine their inefficiencies, and building sentiment analyzers can help them to parse all of these reviews to determine the overall opinion toward their product.

Previous research done to explore some of the challenges posed by the task of sentiment analysis include studies about the relationship between keywords and opinions. These studies often were used as a baseline approach to get information about the opinions presented in a certain body of text, but were not primarily useful in unearthing the true sentiment of the text, mainly due to the complexities of modern language. In terms of applications like Twitter data mining, other difficulties that have been studied include the effect of slang on sentiment and how difficult tasks like sentiment analysis become when there isn’t standard language being parsed.

**Computational Approach/ ML Model**

To build the baseline system, the primary feature used was a unigram approach to determine what words were present in each review and compare the test set of words to the collection amassed by the classifier through the training set. This method allows the classifier to individually associate each word with which type of review it was found in, either positive or negative, and then use that classification to identify the test set.

Baseline System Performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Accuracy | Precision | Recall | F-Measure |
| Unigrams | 80.75% | 0.808 | 0.808 | 0.807 |

The first machine learning algorithm used was the Naïve Bayes classification because of its easy application to the bag of words approach. Naïve Bayes represents an apriori approach, using prior information to generate a probabilistic assumption of the likelihood that the input corresponds to a case previously seen. In this case, this model uses the bag of words that have been classified in the training set as the prior probability, assuming independence between each case, and then generates the likelihood assumption to classify the test set as either positive or negative.

Another useful ML model would be the Decision Tree classifier, which uses a series of binary decisions to classify inputs. If a certain distinction has been seen before by the classifier, then the decision tree will recognize that and classify accordingly. One of the main problems with decision trees, however, is that they tend to overfit the data from the training set, leading to inaccurate representations of unseen data and a lower performance overall compared to other algorithms when encountering unknown phenomena.

Instance-based algorithms, such as the k-Nearest Neighbor approach, employ similarity measurements between stored instances and new data to classify the new data. Based on the specificity of this similarity measure, the results can be very accurate for cases that exactly match or are very close to the stored instances. The downside is that this algorithm requires the entire training set to be stored and compared, so there is no active learning accomplished by the approach. However, if the training set is curated well enough, with erroneous outliers removed, then the KNN approach remains valid for the problem of sentiment analysis.

For my approach, I decided to use the Naïve Bayes approach, because it represents the most memory efficient and most appropriate for the size of the dataset given. Based on the performance described in the ‘Results’ section, this classifier also performed the best among the three discussed with the same feature set. The features used in this system were all used to improve upon the baseline system, and therefore all use different implementations of the String to Word Vector filter provided in the Weka Toolkit.

**Results**

To compare the performance of different machine learning models along with the different features used, the primary metrics used were accuracy, precision, f-measure and recall.

The experiments described below are based off a few different comparisons, each one used to discern which approaches will produce the best performing system. All three of the ML models described earlier are used in these experiments, to determine the performance of each using the same sets of features. When testing the different features, two different sets of experiments were completed per ML model. First, each feature was added independently to the model, and then the models performance was measured using 10-fold cross validation. Second, each feature was added incrementally to determine which had the greatest effect on performance of the entire model.

The specific features used throughout these experiments are employed to build off the baseline system and theoretically improve the baseline’s performance. Because the baseline used a unigram approach, the first two features revolve around the same idea. Feature 1 describes a combined unigram/bigram approach, with the filtering allowing for 1 word minimum and 2 words maximum, reducing the overall set to only include those types. This feature was used to allow for detection of potential negations in the dataset, for phrases such as “not great” or “wasn’t bad”. Without the bigram model, these would be considered false positives and false negatives, respectively, by the system, so implementing this additional feature is thought to reduce these incorrect assumptions.

Feature 2 restricts this distinction even further, allowing for only bigrams in its filtering, reducing the number of available attributes of the dataset. Theoretically, this feature should improve upon the previous one in that using bigrams is more reliable as a testing set than using unigrams because bigrams convey more meaning and are more likely to appear in the same context for testing data. Therefore, when processing future positive/negative reviews, comparing the bigrams from the new test set should correlate more easily with the bigrams from the training set.

Feature 3 used the Lovins stemmer, which is a stemmer used to combine data points for words that have different stems but contain the same root. Implementing this filter removes a large volume of entropy surrounding the different variations of a single word, because the uncertainty around each new version creates more data that needs to be parsed and classified. Instead, by using a stemmer, a single word such as “great” now contains all the data for its derivations (“greatly”, “greatest”, etc.). Because the underlying meaning of all the words is the same, this feature is useful in improving efficiency of the classifier.

The 4th feature is a Stop Words list, which is a list of words that return unnecessary information in search/filtering queries, similar to the classification task posed here. The Rainbow list is a common one that is provided in the Weka filtering list, providing a list of nearly 550 words that should be ignored because of their relative uselessness.

As evidenced by the tables below, the Naïve Bayes ML approach proved to be the most accurate across all feature sets, including individual testing and incremental testing. The Naïve Bayes classifier performance was consistently >15% better in terms of accuracy, as well as boasting greater precision, recall and F-measure than any of the other 2 models.

The Decision Tree and K-Nearest Neighbors models performed much worse, mainly because these models aren’t appropriate for this type of dataset. Because decision trees use prior input to develop a binary decision making model, using a dataset with so many different possibilities for leaves leads to large, inefficient trees that will not accurately map to any unseen information found in a test set. This reasoning explains why the compile and run time for the decision tree classifier took nearly three times as long as the other models, as it was forced to build complex trees to represent the size of the dataset. The K-Nearest Neighbor approach proves inefficient and inaccurate because it is more effective for categorical classification versus sentiment classification. Because there are only two umbrella classes, positive and negative, it’s difficult for the model to discern when one is more similar to one over another. Compared to a different problem of, say, differentiating between types of fish, the binary problem of sentiment analysis is not as conducive to using the k-NN approach.

The interesting aspect of these results is which features are helpful to improving the performance of the system, which are irrelevant and which features work well together. The first feature resulted in the second-highest accuracy when implemented on its own, with a correct classification of 80.1%. The fourth feature was the most effective in improving the performance of the system, posting an accuracy of 81.25%, marginally improving upon the mark set by the first feature.

Features 2-3 actually reduced the performance of the system, posting lower performance marks across all four metrics. The decline caused by Feature 2 can be explained by its tight restrictiveness – since it only allows for bigrams to be processed, some of the meaning provided by specific unigrams is lost, and this can lead to inaccuracies in the classification. For instance, if the sentence given is “The movie was not great, but it was wonderfully surprising in its comedic timing”, then the whole second half of the sentence, which contains a positive sentiment, is canceled out by the presence of the bigram “not great”. Instances like these prevent the bigram model from effectively serving as a necessary feature of the classifier, instead making way for the dual bigram/unigram Feature 1 to serve as the more effective of the two.

Feature 3, the Lovins stemmer, rates as the third most effective feature in the set. Again, seeing as this feature is more

For the incremental approach, with each feature added into another layer of the filtering provided by the Weka toolkit, the primary feature added had the most prominent effect on the performance of the system. The results below confirm this. Because each feature set is so restrictive by nature, it stands to reason that adding another filter layer on top an already restrictive set will not make a large difference, if any, in the filtering and classification performance of the system. Therefore, the incremental testing reported that whichever feature was added first, that feature was the one that dictated the performance of the system as a whole, regardless of whether more features were added. This occurred for all 3 different machine learning models.

Naïve Bayes-

With each feature added independently

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 80.1% | 0.806 | 0.801 | 0.800 |
| Feature 2 | 76.6% | 0.767 | 0.766 | 0.760 |
| Feature 3 | 79.95% | 0.801 | 0.798 | 0.797 |
| Feature 4 | 81.25% | 0.815 | 0.813 | 0.812 |

Incremental Approach (multi-filter)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 80.1% | .806 | .801 | .800 |
| Feature 2 | 80.1% | .806 | .801 | .800 |
| Feature 3 | 80.1% | .806 | .801 | .800 |
| Feature 4 | 80.1% | .806 | .801 | .800 |

Decision Tree-

With each feature added independently

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 65.45% | 0.655 | 0.655 | 0.654 |
| Feature 2 | 62.3% | 0.623 | 0.623 | 0.623 |
| Feature 3 | 65.25% | 0.653 | 0.653 | 0.652 |
| Feature 4 | 66.85% | 0.669 | 0.669 | 0.668 |

Incremental Approach (multi-filter)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 65.45% | 0.655 | 0.655 | 0.654 |
| Feature 2 | 65.45% | 0.655 | 0.655 | 0.654 |
| Feature 3 | 65.45% | 0.655 | 0.655 | 0.654 |
| Feature 4 | 65.45% | 0.655 | 0.655 | 0.654 |

K-Nearest Neighbor (1 neighbor criteria, Euclidean distance) –

With each feature added independently

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 58.00% | 0.586 | 0.580 | 0.573 |
| Feature 2 | 51.95% | 0.543 | 0.520 | 0.444 |
| Feature 3 | 56.45% | 0.574 | 0.565 | 0.550 |
| Feature 4 | 51.00% | 0.529 | 0.510 | 0.414 |

Incremental Approach (multi-filter)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F-Measure |
| Feature 1 | 58.00% | 0.586 | 0.580 | 0.573 |
| Feature 2 | 58.00% | 0.586 | 0.580 | 0.573 |
| Feature 3 | 58.00% | 0.586 | 0.580 | 0.573 |
| Feature 4 | 58.00% | 0.586 | 0.580 | 0.573 |

**Discussion and Conclusions**

Throughout this project, I’ve learned a great deal about the problem of sentiment analysis and its solution through the form of different machine learning models. In trying to create and improve a baseline system, this project allowed me to discover the entire process behind machine learning, and how to incorporate different feature sets and experiment with them to produce the best possible system.

Regarding sentiment analysis, I learned a much greater deal about the complexities presented by such a seemingly simple problem. Beyond the task of classification, there is a lot of preprocessing work that should go in to filter the input text to make the data worthwhile to your model.

To try and mitigate some of these difficulties, I needed to determine which machine learning model would perform the best against this task of sentiment classification, selecting between the Naïve Bayes approach, the Decision Tree model, and the k-Nearest Neighbor model. All 3 have their own advantages and disadvantages, but the key in choosing which model to use depends on the properties of the dataset, and this realization was critical in determining which to use.

The other key aspect of building an efficient and accurate model is to select the right set of features. Through the Weka interface, the option to filter the strings provided in the input text allows for different ways to parse the dataset and produce relevant attributes for the classifier to use as training/test sets. Among the list of available features, it is important to choose the ones that relate most directly to the task at hand, and experimenting between these features was the best way for me to learn which would actually prove useful for sentiment analysis. I decided to use: the N Gram tokenizer, to implement both unigram and bigram models; the Lovins stemmer, to reduce the complexity introduced by similar-rooted words with different stems; and the Rainbow stop words list, which prevents the system from parsing any irrelevant words and using them to classify. These different features all help to reduce the complexity of the parsed input text, and allow the classifier to make much better decisions. As evidenced by the results table above, certain features performed more accurately than others, but each feature still contributed in some way to the performance of the overall system. Going through the process of feature selection/experimentation was helpful for me to understand how each different feature could impact the classifier, and understand how different features are more suited to certain datasets. Through this process, I decided on these features to help and improve the performance of the system.

Testing these features involved two different approaches, as described in the Results section. Determining the best ways to test these features was also important for me to understand how the process of machine learning and ML testing occurs. The two approaches I decided on allowed me to test each feature individually, along with measuring the overall change in performance of the system by adding each feature incrementally. This approach allowed me to determine which features were most impactful to the system: Feature 4, the Rainbow stop words list improved the performance of the baseline system the most, proving that it was the most beneficial feature. In order to improve the performance of the sentiment analyzer, more features should be tested and analyzed to see the impact they can have on the system. Other features that could be added include a lexicon-based system that bases its classification on the sentiment implied by the word itself. This type of system would rely more directly on the meaning of each word rather than the relationships between words, which would lead to a differently focused system.

**Preprocessing Code**

Experiment 1: All 3 ML Models with Feature 1 (bigram and unigram model)

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.NGramTokenizer -max 2 -min 1 -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

Experiment 2: All 3 ML Models with Feature 2 (Only bigram Stemmer)

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.NGramTokenizer -max 2 -min 2 -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

Experiment 3: All 3 ML Models with Feature 3 (Lovins Stemmer)

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.LovinsStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

Experiment 4: All 3 ML Models with Feature 4 (Rainbow Stop Word List)

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Rainbow -M 1 -tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

Incremental Experiments

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.NGramTokenizer -max 2 -min 1 -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.NGramTokenizer -max 2 -min 2 -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.LovinsStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Rainbow -M 1 -tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r\\n\\t.,;:\\\'\\\"()?!\""

**Extra Credit**

**Option 1:**

**Previous Work**

There have been several studies completed previously to prove the success of sentiment analyzers in categorizing different types of reviews.

In a study done by Bo Pang and Lillian Lee at Cornell University, the comparison between performance of standard machine learning techniques and human-produced baselines is discussed, along with each approach’s respective performance in sentiment classification. The study discovered that although standard ML techniques greatly outperform human-produced baselines in traditional topic categorization, they are inferior for sentiment-based classification. The three different ML methods employed were Naïve Bayes, maximum entropy classification, and support vector machines. The paper discusses several of the difficulties faced by sentiment classification that aren’t encountered by topic categorization, including the fact that topic classification can generally be accomplished by keyword searches, while different combinations of the same words can have totally different meanings when analyzing for sentiment. Because of this, context plays a very important role and requires many more processing considerations when trying to develop an accurate sentiment analyzer.

A team of researchers at the National Research Council in Rome, Italy conducted a series of experiments to compare different approaches, tools and applications for sentiment analysis. The paper begins by outlining the different types of approaches to sentiment analysis and the corresponding features and techniques that are used in those approaches. For example, for the machine learning approach, the paper lists the different methods, including: Bayesian networks, Naïve Bayes Classification, Maximum Entropy, Neural Networks, and Support Vector Machine. It then continues to list the different techniques that can be used, including term presence and frequency, POS information, or opinion words. This study also explores the effect that different literary devices such as sarcasm and irony can have on the meaning of a sentence.

In a dissertation entitled “Discourse-Level Relations for Opinion Analysis”, Swapna Somasundaran discusses more complex forms of sentiment analysis by trying to tackle discourse analysis. At its heart, discourse analysis is used to find relationships between spans of text, which can be used extensively in sentiment analysis because relationships between words are more indicative of true opinion than simply just combining different keyword results. In this approach, two different levels of discourse level relations are defined: those used to improve polarity classification of sentences, and those used to improve recognition of the overall position of the span of text. In combination, these two approaches define a more accurate representation of the true opinion presented in the sentence.

**Comprehensive Discussion and Conclusions- more discussion of improvements and potential changes**

Most of the improvements that can be made to this system and overall solution revolve around the type of machine learning model used. Because this dataset contained a lot of different attributes with many different possibilities for combinations of words, decision trees proved to be the worst model in terms of performance. Since they use prior combinations to build a binary search approach, this model was not able to correctly classify unseen data from the test set because its generated decision tree did not contain enough information to represent the different multitudes of language combination. K-Nearest Neighbor also was not appropriate for this dataset because with only two categories to consider, there is simply a binary decision in choosing the result. Nearest neighbor algorithms are generally more effective when classifying between several different categories, because there is more of a range from which to differentiate and classify data differently. If the task related to more than just positive v. negative classification, such as classifying different emotions, then Nearest neighbor search may prove to be more effective. Naïve Bayes, in its approach to individually create a probabilistic assumption for each new piece of data, is more likely to perform better given such a large and diverse dataset. To improve the analysis, more models should be considered and tested to determine if there is a model that suits the data even better than the 3 considered, such as Support Vector Machines, or even adding the SMO (sequential minimization optimization) algorithm, which supports the SVM model and makes it more efficient. Additionally, more categories can be used to classify the data, beyond the simple binary denotation of negative v. positive.

In terms of the linguistic complexities that still need to solved, the problem of ambiguous literary devices needs to be addressed. Processing language inherently poses new sets of problems, such as literary device analysis (sarcasm) or poor grammar/spelling. Spelling errors make words unrecognizable and therefore indecipherable by a computerized system, while language nuances like tone and mood are not easily recognized by non-human systems, and this prevents the message from being fully understood. These problems can easily change the meaning of the text one is processing, and therefore the analysis of these texts becomes exponentially more difficult. To combat these problems, systems need to be developed that accurately interpret text, considering these different language complexities to develop a reliable and accurate system. Because it is very difficult to design a system that appropriately understands human sarcasm or other aspects of language, there must be newer features that can recognize these different linguistic nuances. Additionally, more modern classifiers have to deal with interpreting slang, and relating these new words to common dictionary-based words to discern a legitimate meaning. These problems are some of the key issues still facing the natural language processing research community, and only with the development of more intensive features that mimic human language nuances will we be able to fully employ a sentiment analyzer to the desired degree.

**Option 2:**

Although I was not able to get the data conversion from the given files to the required .arff format for Weka to use, I still thought it would be a good exercise to still theorize potential implementations for the different datasets.

Because the Yelp dataset is implemented with different sets of categories, rather than a simple binary classification of positive vs. negative, I believe it is more difficult to implement. Categorical classification is by nature more difficult to implement because there are more possibilities for classification, and thus it’s not a task of either/or. Classifying each review from a star rating requires more precise determinations of the meaning of each word and how each word carries different weights to the spectrum of positive v. negative classification (ie. Amazing vs. okay). Implementing a system to account for of these different possibilities would require a level of analysis beyond the basic morpheme/keyword approach that can be taken for simpler classification tasks. To accomplish this, some level of syntactic or discourse analysis can be used, implementing POS tagging along with sentence-level discourse to unearth the meaning of sentences in relation to each other.

In terms of which is more accurate, I believe the original task of classifying the movie reviews is more accurate, because there are less levels of language that need to be deciphered. When the task becomes more complicated, as parsing the Yelp dataset would be, it becomes more difficult to achieve the same level of performance. Additionally, different features would be required, including a more intensive N-Gram approach, to give a better representation of the relationships between words. Because the size of the feature set is going to increase to provide more curated data, the runtime for this algorithm is also going to increase, which might not affect the accuracy of the system, but will definitely affect the overall desired performance, because a slow algorithm is rarely an efficient one.

A good baseline approach would still be a unigram model, because it will provide the requisite filtering to produce a decent result, even runs a bit slower than other features. The model most likely to perform on this dataset is still the Naïve Bayes model; the others discussed for the initial task are still less likely to be efficient given the properties of the dataset. Another ML model to consider for this would be a regression model or a supervised neural network, because the categorical portion of this task becomes easier to implement with the clustering properties of these different models.