Sentiment Analysis Utilizing an Open-Source Dataset of IMDb Reviews

Chip Simmerman
Indiana University – Purdue University - Indianapolis
Indianapolis, IN, USA
csimmer@iu.edu
Advisor: Mohammad Al Hasan, Ph.D
Advisor Email: alhasan@iupui.edu

***Abstract*—This report will cover the use of various text classification models, word vectorization models, and modes of pre-processing to extract sentiment from a large dataset of movie reviews. It will begin with an introduction to briefly explain terminology and background before defining the purpose of this project. Following the problem definition, I will expand upon the approach and solution(s) implemented before evaluating the success of the project. The report will then conclude with a summary of the project, what has been achieved, and any further remarks.**

Keywords—Sentiment Analysis, Machine Learning, Opinion Mining, Natural Language Processing, Classification, Feature Extraction

I. Introduction

The main objective of this project is to use the subfield of machine learning known as natural language processing (NLP) to extract sentiments from a large, open-source dataset of IMDb movie reviews collected and compiled by A. L. Maas et. al. at Stanford University [1]. More specifically, the objective is to extract the feature of sentiment from items in the dataset as effectively as possible. Natural language processing has found numerous modern applications from virtual assistants to biometric speech recognition. One application that this project is intended to represent is sentiment analysis. Sentiment analysis is a subfield of NLP which details extraction of sentiment or opinion from natural language. This language could be written, spoken, or even gestured. Because the chosen dataset contains written text, the following implementation of sentiment analysis will focus on feature extraction and classification of text. In the following subsections, I will cover more in-depth the topics of natural language processing and sentiment analysis, as well as some specific methods used.

A. Natural Language Processing

IBM Cloud Education explains that natural language processing (NLP) is a field of Computer Science which, simply put, is concerned with allowing machines to process forms of natural language and obtain some meaning from it [3]. Computers do not typically have the ability to comprehend and interpret the meaning of human language, and therefore require some method of mechanically converting natural language to a machine-readable format. After this has been achieved, a machine can then obtain some context-defined meaning, usually through mathematical feature extraction and classification. This process, as explained in the following subsection, is the main mechanism explored in this project.

Natural language processing is now a widely-used field in everyday products, sometimes in ways that are hidden in plain sight. One example that is prevalent in today's society is the ability for a system to autocorrect or autocomplete your text input. In order for a machine to correct a typo, it must first attempt to understand what message a user is attempting to convey to ensure that the corrected word best fits what the user wants to communicate. In the case of autocomplete, such as with search engines or "intellisense", the system must again understand what the user's intention is and autocomplete with a term or phrase that best matches that intention. The method used to process natural language in the "correct" way tends to depend on the context. As explained initially, the context of this project deals with sentiment, and therefore the methodologies utilized will pertain to classifying emotion.

B. Sentiment Analysis

Sentiment analysis is the analysis of feeling, emotion, or opinion communicated via text. It is commonly referred to as "opinion mining", as the process often results in an opinion being received

after computationally digging through features and classifying data based on those features. Sentiments received through processing written language typically relate to emotion such as positive, neutral, and negative. This sort of data can be very valuable when it is relevant to assessing feedback/public perception.

Sentiment can range from binary to practically any dimensionality relevant to the context at hand, like explained in the previous subsection. A binary sentiment range would imply two directly contrasting sentiments such as happy and sad or positive and negative. Of course, many contexts can call for non-binary sentiment ranges which include more than two classifications. For example, a company looking to assess customer feedback could use a scale ranging from very negative to somewhat negative, neutral to somewhat positive, and very positive. While the number of classes indeed depends on the contextual requirements, it is often of benefit to reduce this number as much as possible to avoid complexity and even improve accuracy.

With Big Data becoming a common issue to deal with for any modern company, sentiment analysis can provide automatic, accurate information that would take ages for humans to process. This information can be mission-critical, especially in industries where public perception is closely tied with revenue. One of many examples given by MonkeyLearn is the necessity to assess a company's public image while a public relation situation is escalating in real-time [3]. One may then ask why this task could not be handled with a simple poll, with answer choices correlated some positivity scale such as 1-10 with 10 being most positive. MonkeyLearn makes the great point that this idea could give an estimate of overall sentiment, but it does not offer anything beyond that [3]. If a company wants to use this information to improve its product(s) or service(s), it may require much more detailed input. This can be obtained through methods of sentiment analysis, such as finding words most frequently associated with positive reviews. It is with the offerings of sentiment analysis that I will attempt to address the problem of this project.

C. *Pre-processing*

Various pre-processing methods were used to reduce the raw data used in this project to both remove unwanted elements and to reduce the elements to root forms. One method, used across all implementations in the project, was to use regex to remove punctuation, mark-up text, and reduce the data to lowercase text. The other 4 methods were a bit more intensive and focused on changing the data's representation rather than removal. The first three were Porter stemming, Snowball (Porter 2) stemming, and Lancaster stemming. These use special algorithms to essentially chop off the latter half of each text item in the dataset. Porter stemming is typically the slowest and least aggressive form of stemming. Snowball stemming is widely regarded as an improvement on Porter stemming. Lancaster stemming uses the most aggressive algorithm for stemming and is rarely used, but it is still considered. The last method of pre-processing is called lemmatization. Lemmatization, rather than chopping off half of words, actually uses lookup tables to reduce words to their base forms. For example, the word "saw" would be reduced to "see". Lemmatization typically leads to the most accurate implementations, since the words' meanings are preserved; whereas stemming can actually result in items that aren't actually words.

D. *Feature Extraction*

DeepAI explains that feature extraction refers to reducing the dimensionality of a raw dataset to make it easier to process, resulting in a product that should still describe the original dataset [5]. Essentially, feature extraction can be thought of as a conversion from a raw format of text to a processing-friendly format such as a set of vectors. This formatting can often reduce the time it takes for processing and training to occur as it compresses the amount of data a system needs to deal with. This is especially helpful not only because it can convert language to a machine-friendly format but also because it makes dealing overly-large datasets much easier. This project made use of two different methods of feature extraction, both essentially converting data into vectors; a process aptly named vectorization or word embedding.

1) *Bag of Words (BOW)*

The Bag of Words vectorization method is arguably the simplest method of representing text in a numeric format. As an article written on Analytics Vidhya explains, the Bag of Words method is one method of word embedding which seeks to use a vocabulary of unique words from each item of text in the dataset [2]. Then, every text item in the dataset is then represented as a vector of 1's and 0's of length equal to the number of unique words in the vocabulary. If a word occurs in a text item, a 1 appears in that word's index; otherwise, a 0 is used.

For large datasets like the one used in this project, this method results in a large number of very sparse vectors, meaning that most vectors consist mainly of 0's rather than 1's. This should be taken into consideration when a classification model is chosen, as it should be able to deal with sparse data. One of the biggest downsides to this method is that it does not retain any information regarding grammar or sentence structure. Each vector is essentially a "bag" of "words" simply indicating via a 1 or 0 if a word occurred in the vector's sentence. While this will inevitably lead to an impact on accuracy, it is still a viable option for most scenarios and is relatively simple to implement.

2) Term Frequency – Inverse Document Frequency (TF-IDF)

Term Frequency – Inverse Document Frequency (TF-IDF) is another form of word embedding which utilizes word vectorization to extract features from data. Term Frequency comes from how often a particular word occurs in one "document" in a dataset. A document could refer to a file in a collection of files, a sentence in a paragraph, etc. This frequency is calculated by counting the number of times the word occurs in the document and then dividing it by how many words are in the document. Therefore, for a sentence with 10 words and the word "at" appears twice, the term frequency for "at" would be 2/10 = 1/5. Term frequencies would then be calculated for each word in the vocabulary which is collected just like it is for Bag of Words. If a word doesn't occur in the document, it's term frequency is 0.

Secondly, Inverse Document Frequency is calculated. This is essentially a calculation that relates to how important a term is. It is calculated by taking the log of the number of documents in the collection divided by the number of documents containing the term. This is again done for every word in the vocabulary. Often, very common words like "is" which are usually not very important will have their score dropped further to 0, reflecting their lack of importance.

Finally, the TF-IDF score is computed by multiplying the TF score by the IDF score. This is done for every word in each document. The resulting TF-IDF score is typically higher for less-frequent words and for words which are rare but more frequent in a single document.

This method retains more information about the words after they've been embedded in a vector than the Bag of Words method, but it is still not perfect. It also does not consider grammar, sarcasm, or differences between similar words. Nonetheless, it typically still performs better in machine learning than Bag of Words.

E. Classification

Text classification is relatively self-explanatory, but it is still necessary to cover it and some of the methods considered in this project. Classification is the process of taking data and selecting a label to identify it. This is often associated directly with the sentiment that is desired, and the dimensionality will depend on it.

For example, if the desired sentiment is a range of emotions including happy, angry, and sad, the number of classes that a classifier will have to consider is 3. In order for a classifier to make a correct choice, it requires context to work from. This is often accomplished by training a classification model which uses machine learning techniques to identify similarities between the data it was trained with, and the data being tested. It is critical that a machine learning model is tested with data that is the best representation of what it will be classifying, because the classifier can very well depend heavily on the data it already knows. Some classifiers have "hyperparameters" which can be tuned to avoid issues such as overfit, but this depends on the method itself.

1) Logistic Regression

Logistic Regression is one of the more common classifiers used in machine learning applications. It's similar to the Linear Regression model, which uses a linear equation of feature coefficients to draw a best fit line between input features and the desired output while simultaneously reducing the distance from each point to the line. Linear regression does not perform well as a classifier, and thus Logistic Regression is used instead. Logistic Regression uses a similar linear equation as linear regression but makes it part of an inverse exponential decay function which creates a sigmoid curve (S-curve) that varies probability of belonging to a class. Chirag Goyal explains that there are three types of Logistic Regression to know about: binary Logistic Regression, multinomial Logistic Regression, and ordinal Logistic Regression [6].

Binary Logistic Regression attempts to classify data into two categories, such as positive/negative, high/low, or pass/fail. Multinomial Logistic Regression allows for more than 2 categories without order being a necessity. For example, a multi-step scale from very negative to very positive would make good use of this type of Logistic Regression. Ordinal Logistic Regression is similar to multinomial Logistic Regression, but the classes have some sort of order associated with them, such as a numerical scale. This project will make use of binary Logistic Regression, where the classes are positive and negative.

The binary Logistic Regression model uses a regression formula that varies between 0 and 1, where 1 is 100% probability that our input belongs to class 1, and 0 is 100% probability that our input belongs to class 0. Logistic Regression has a hyperparameter that allows regularization to be adjusted. This hyperparameter is known as lambda, where the greater lambda is, the stronger the regularization is. As Max Miller explains, regularization essentially dilutes the power of coefficients in the regression model so that the accuracy doesn't vary too strongly when the input is changed [8]. Doing so reduces the chance of overfitting your classifier to your training set, which in turn allows your classifier to better deal with hidden data and test data.

Logistic Regression has the advantage of being easy to train, easy to understand, and typically works well with large datasets like used in this project. Unfortunately, it is also susceptible to overfitting and can require features to be catered for it beforehand.

2) Multinomial Naïve Bayes (MNB)

Multinomial Naïve Bayes makes the assumption that the features are independent of each other. As Christopher Ottesen explains in his article, a dependency between features can lead to a less-than-ideal accuracy when using MNB. This may not be ideal for our dataset since the features are technically related through the reviews each word appears in, but limitations considered when collecting the dataset (explained in problem definition) should still make this a viable classifier to consider.

This classifier uses Bayesian probability to calculate the probability of an item belonging to a class given a feature. For a class A and a feature B, the probability P(A|B) is calculated by: P(A)*P(B|A)*P(B). P(A|B) is the probability of belonging to class A given a feature B. P(A) is the probability of belonging to class A in general. P(B) is the probability of feature B occurring. P(B|A) is the probability of feature B occurring given a class A. This formula is known to many in statistics and is widely used in calculating probabilities, but it is considered naïve because of the assumption the method makes.

3) Linear Support Vector Classification (Linear SVC)

Support Vector Classification is related to Support Vector Machines, which can work on both linear and non-linear problems. It essentially works to create hyperplane (essentially a line) that sorts your data into classes. Linear SVC is not to be confused with SVC, as SVC allows you to choose from different kernels for non-linearity whereas Linear SVC restricts you to a linear kernel. Linear SVC actually tends to run faster with larger datasets due to the kernel having been optimized, as Harry Davis explains [7].

Because Linear SVC is a SVM with a linear kernel, it essentially works the same as any SVM. A support vector machine plots data with the features given in a plane corresponding to the dimensionality of the number of classes. Using this for a linear implementation like in this project, the plane is 2D and the hyperplane is essentially a line. The hyperplane is placed and varied so that it separates the datapoints into the correct classes and maximizes the distance from the hyperplane to the nearest datapoint of each class. This method can become a bit more complicated when dealing with non-linear data, but it is unnecessary to cover those details for this project.

4) Stochastic Gradient Descent (SGD)

Gradient Descent is a technique used to optimize machine learning through the use of a gradient. A gradient can be thought of as the slope of a line, which measures the rate of change of some variable as another variable changes. Stochastic Gradient Descent tends to pick a portion of the dataset rather than using the entire dataset, which can improve processing time when the dataset is very large. Because of the randomness in the descent to convergence, SGD can be quite "noisy", meaning it can take more iterations than

typical gradient descent, as explained in an article on GeeksforGeeks [9]. Even then, SGD is still more computationally efficient than Batch Gradient Descent. There aren't many qualities that suggest this classifier will be a champion, and it's often possible that it will perform worse than Logistic Regression.

II. Problem Definition

The problem that this project is intended to address is one of automating sentiment analysis of a dataset that is too large to process manually. The largest subproblem that follows is being able to assess sentiment as accurately as possible. The automation of sentiment analysis will be performed using machine learning models such as Logistic Regression and support vector classifiers, which heavily depend on the data they are trained with and must be tuned to deal with obstacles such as sarcasm, double negation, etc. that would otherwise be easy to understand as a human.

In order to demonstrate a solution to this problem, I have used the dataset of IMDb movie reviews obtained by A. L. Maas et. al. [1]. The dataset consists of 50,000 total reviews and it is split evenly into a training set and a testing set. Both sets of 25,000 reviews are again evenly divided into 12,500 positive reviews and 12,500 negative reviews. This data was collected with no more than 30 reviews per movie, and no neutral reviews were allowed in the dataset. This restriction was placed to reduce the correlation between too many reviews, as this could lead to biased training and hinder a classifier's ability to identify sentiment correctly. Additionally, the lack of neutral reviews makes training easier as the dimensionality is reduced to simply positive and negative. This dataset fulfilled the requirement of being sufficiently large to require automation, but reasonably-sized so as to keep processing times as low as possible.

Acquiring the dataset for use in this project was simple, and the true difficulty came from a few different areas. First, the data needed to be formatted in a way that could be easily handled via a high-level programming language. Second, the data required significant pre-processing before being handled, as items such as special characters, punctuation, and mark-up text relay no meaning to be extracted for sentiment analysis. Similarly, the data also requires further removal of certain language words that also carry little or no meaning. Next, I needed to test different feature extraction methods. Both of the methods used to obtain features from the review texts work differently and have different tradeoffs, which can affect accuracy, processing time, etc. Subsequently, I also needed to test a number of classification methods which also have different methods of determining class based on features. Each of these steps, which will be explained further in the following section, required a large amount of adjustment and matching to reach an optimal solution with the highest accuracy.

III. Approach & Solution

The approach I used for my implementation was relatively straightforward. I began by researching various methods of pre-processing text, extracting features from text, and classifying data based on features. I felt it would be best to begin with the most limited I could put together and then improve upon it wherever possible. All implementations and tests were ran in Python using various libraries such as Natural Language Toolkit (NLTK) and scikit-learn. I began with light pre-processing, only removing punctuation and mark-up text before converting the text to lowercase using regex. Then, I used a Bag of Words method to vectorize word counts for each review. Finally, I trained a Logistic Regression model before fitting it to the test set. After doing so, I was able to predict the values and assess the accuracy of the implementation.

Given my lack of knowledge before beginning this project, I relied heavily on online research to guide my implementation. I initially found that Logistic Regression and the Bag of Words models are two of the easiest to implement and they tend to work well on sparse datasets like the one I worked with. While I will touch on evaluation and accuracy in a later section, I was pleased that my baseline implementation actually performed well.

I chose to use this implementation as a baseline to improve upon by using different combinations of pre-processing methods, classifiers, and feature extraction methods in order to maximize the accuracy of analyzing sentiment in my dataset. In total, I have used 4 different pre-processing methods, 4 different classifiers, and 2 different feature extractors. This leads to a possibility of 32 different combinations, of which I was able to implement 18. Additionally, I also implemented two cases using Bag of Words where the vectors held actual word counts instead of simple 1's and 0's.

In order to test each implementation, I would vectorize the reviews in both the training and test sets using either Bag of Words or TF-IDF. Then, I would pass these vectorized training set to one of the 4 classifiers in order to perform a 75%/25%

split to train with. To clarify, I further split the training set into 75% training and 25% test to train the classifier before predicting the values of the true test set. This was done for each separate iteration, and then I recorded the resulting accuracy score to keep track of the optimal and least-optimal solutions.

IV. Experimental Evaluation

Evaluating the success and accuracy of each combination of pre-processing, vectorization, and classification was carried out using sklearn.metrics accuracy scoring. This was done by comparing a target list of 12,500 1's and 12,500 0's to the output of each implementation. Because we know that the first 12,500 reviews should be positive, the first 12,500 places in the target list have a 1. The accuracy scorer compares the predicted list to the target list and outputs how many reviews (as a percentage) the classifier classified correctly.

Overall, I tested 20 different combinations, using a library called nltk.corpus to remove stop words from my dataset during pre-processing. The accuracy I achieved with my baseline implementation with Logistic Regression and Bag of Words was 87.924% correctness. On following implementations, I was seeing very frustratingly-low accuracy improvements despite adding in new methods of pre-processing, vectorizing, and classifying which were supposed to be more accurate. Using the nltk library, the highest accuracy improvement over the baseline implementation was 0.856% at 88.78%.

After examining the set of stop words nltk uses, I realized there could be a large number of words it was removing which could actually convey meaning in movie reviews; especially with the movie reviews explicitly including positive and negative reviews. Therefore, I attempted 5 more combinations using some of the previously most-accurate implementations, but I only removed a small portion of hand-picked stop words. This list is only 9 elements long and includes words like "is", "at", "an", etc. that include no negation or description. After changing the list of stop words to this shortened version, I started seeing accuracy improvements of just over 2%. The best implementation that I found used Linear SVC, Bag of Words, and the Snowball stemming method with an accuracy of 90.06%. This was an accuracy improvement of about 2.136%, and while this may still seem small, that's equivalent to about 534 more reviews classified correctly than the baseline.

The top three implementations all achieved an increase by over 2%, and all three involved Linear SVC. The lowest accuracy I achieved was actually 86.652% with Multinomial Naïve Bayes, Snowball stemming, and TF-IDF. This was a decrease in accuracy from the baseline by -2.272%. In fact, all of my tests involving Multinomial Naïve Bayes resulted in loss of accuracy. I explained in the MNB section of the introduction that it was possible this classifier wouldn't be the best because the features weren't necessarily dependent, and it appears to be true.

It was surprising to find that the implementation that won out used Bag of Words and stemming rather than TF-IDF and lemmatization, but it is possible that this has to do with the context of movie reviews being more sensitive to tense and inflection. Additionally, it's obvious to assume that most reviews were written by different people with different levels of vocabulary, so it is plausible for an implementation using TF-IDF to not score as well. If the reviews were all coming from one person, the vocabulary would be more consistent and term frequency would also likely be more consistent. It seems that the Bag of Words model and Snowball stemming work best for this application because of the dataset's nature.

V. Conclusion

In conclusion, I was able to introduce myself to a field of Computer Science that I was originally unfamiliar with. It was an interesting experience exploring the field of Natural Language Processing and Sentiment Analysis, as we can take these technologies for granted in our everyday lives. This was also introductory to machine learning in general and was an indirect application of statistics like I've not experienced prior. I am content with the results of this project, having achieved Sentiment Analysis with an accuracy of 90%, which was higher than I originally set out to achieve.

I was able to study multiple forms of text processing, feature vectorization, and methods of classification and then implement each of them in different combinations to maximize the accuracy of my analysis. In the end, I realized that there is no single perfect solution for every scenario; each component of Natural Language Processing can depend on another such that a change in one area propagates through the rest of the implementation.

For the future, I believe it would be interesting to do a deeper analysis of sentiment using a dataset from another environment to compare with this one. Additionally, it would be interesting to be able to collect such large-scale

data personally, although this would extend the project into the field of data mining and would likely require great effort from more than one person.

REFERENCES

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," Jun-2011. [Online]. Available: https://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf. [Accessed: 28-Jan-2022].

[2] "Bow model and TF-IDF for creating feature from text," Analytics Vidhya, 23-Dec-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/. [Accessed: 02-Feb-2022].

[3] By: IBM Cloud Education, "What is natural language processing?," IBM, 02-Jul-2020. [Online]. Available: https://www.ibm.com/cloud/learn/natural-language-processing. [Accessed: 31-Jan-2022].

[4] C. Ottesen, "Comparison between naïve Bayes and logistic regression," DataEspresso, 24-Oct-2017. [Online]. Available: https://dataespresso.com/en/2017/10/24/comparison-between-naive-bayes-and-logistic-regression/#:~:text=Na%C3%AFve%20Bayes%20has%20a%20naive,belonging%20to%20a%20certain%20class. [Accessed: 22-Apr-2022].

[5] DeepAI. (2019, May 17). Feature extraction. DeepAI. Retrieved April 22, 2022, from https://deepai.org/machine-learning-glossary-and-terms/feature-extraction

[6] Goyal, C. (2021, May 31). Logistic regression questions: Questions on logistic regression. Analytics Vidhya. Retrieved April 22, 2022, from https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-logistic-regression/

[7] H. Davis, "What is LinearSVC?," QuickAdviser, 21-Aug-2020. [Online]. Available: https://quick-adviser.com/what-is-linearsvc/. [Accessed: 22-Apr-2022].

[8] Miller, M. (2019, November 4). The basics: Logistic regression and regularization. Medium. Retrieved April 22, 2022, from https://towardsdatascience.com/the-basics-logistic-regression-and-regularization-828b0d2d206c

[9] "ML: Stochastic gradient descent (SGD)," GeeksforGeeks, 13-Sep-2021. [Online]. Available: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/. [Accessed: 22-Apr-2022].

[10] "Sentiment Analysis Guide," MonkeyLearn. [Online]. Available: https://monkeylearn.com/sentiment-analysis/. [Accessed: 31-Jan-2022].

[11] Sharma, A. (2020, July 8). Applications of natural language processing (NLP). Analytics Vidhya. Retrieved April 21, 2022, from https://www.analyticsvidhya.com/blog/2020/07/top-10-applications-of-natural-language-processing-nlp/

[12] "Support Vector Machines (SVM) algorithm explained," MonkeyLearn Blog, 22-Jun-2017. [Online]. Available: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/. [Accessed: 22-Apr-2022].