Testing Five Tokenization Methods on IMDb Movie Review Sentiment

Graham Harris (writer)

Judy Zhang (researcher)

Christine Ma (programmer)

Jong Heon Han (analyst)

CSCI-UA.0480 Natural Language Processing

Prof. Adam Meyers

ABSTRACT:

In this experiment, five different tokenization methods were used to determine what components are necessary for sentiment analysis. By testing split-by-space, lowercase, no punctuation, adjectives-only, and lemmatization tokenizers, the research team tested what parts of a given movie review are necessary for the sentiment to remain intact. Each tokenization method was run through a machine learning model to evaluate the accuracy of the system on a binary classification model (0 for negative, 1 for positive). The team found that the lowercase and no punctuation methods worked best, likely because these tokenizers change the review to a more machine-readable version. The adjectives-only method performed the worst; since adjectives are not the only part of speech that contains sentiment, it is proposed that more than just adjectives are necessary for sentiment analysis. The split-by-space and lemmatization methods performed relatively the same. Each tokenization method performed under 90% accuracy, likely due to a combination of the data set used and lack of context understanding by the model. Our system lies in the 80-90% range as opposed to state of the art (as of the writing of this paper) at 96.21%.

INTRODUCTION

Classification of text is an important feature of natural language processing. Given sequences of text, systems have been built that identify document types, study language patterns, create predictions, and even generate entirely new bodies of text.

However, there are many difficulties when attempting sentiment analysis of a given corpus because human language is complex and interpretable. To simplify the problem, models are typically executed on corpuses with strong sentiment, because classification is easier when a corpus undeniably evokes a certain emotion or opinion. Since movie reviews contain bodies of text with strong sentiment, they are a great starting point for analysis. Movie reviews, in this case IMDb reviews, are also usually concise and straightforward, often lacking the length and linguistic complexity of other bodies of text like fiction. Most importantly, movie reviews typically only contain binary sentiment, described as “positive” or “good” reviews and “negative” or “bad” reviews.

Problem

Given a dataset of reviews with an assigned binary classified sentiment of positive and negative (represented in the data set as probabilistic range between 0 and 1 respectively) our team aims to test different tokenization

methods on a reputable model in order to analyze how various tokens are weighted in their sentiment value. Several different tokenization methods will be compared to discern the best system. Through our analysis, we hope to determine which tokens are necessary to determine sentiment and which are not.

The tokenizers used are splitting by space (used as a control), casting to lower-case, adjectives as sole tokens, ignoring punctuation, and lemmatization. By using different tokenization methods, our team will discover which method leads to the most accurate classifications. After analyzing each tokenization method, we explore what components of a sentence, out of the tokens experimented on, are necessary to accurately describe binary sentiment.

At the end of the paper, we cite sources of error and discuss examples where the model failed to predict the sentiment of a given review. We also present some ideas for further and continued research.

DATA SET

The data set consists of 50,000 IMDb movie reviews provided by Stanford University. The set contains a balanced distribution of highly positive and highly negative movie reviews divided into a validation set, test set, and training set. Of the reviews, 25,000 reviews are split into a validation set containing 5,000 balanced reviews and a training set containing 20,000 balanced reviews, randomly distributed by a seed value. The test set contains 25,000 balanced reviews. Also included is a set of all 50,000 unlabeled balanced reviews.

In the training set, each review has an ID value, a binary sentiment score (1 for positive, 0 for negative) and the movie review text. The test set is the same format as the training set and is used to evaluate the system.

MODEL AND TRAINING DETAILS

The code for the model is from the SciKit Learn Python library and is coded in Jupyter Notebook. To start the model for a given tokenizer, first the movie reviews are put through a count vectorizer. The count vectorizer reads through all the reviews in the training set and assigns a counter value based on the number of instances of the token in the corpus.

The count vectorizer returns a 2D array that is fed into a TF-IDF algorithm. The purpose of performing TF-IDF on the vectorized tokens is to compute the importance of the word in a given movie review. TF-IDF compares the number of instances of a word relative to how important the word is in each movie review. It defines a relationship between each word in the document, which is used to determine the weight of the tokens in the review.

The model used is logistic regression. The input to this model is the tokenized movie review and the output is a binary classification of either 0 indicating a negative review or 1 indicating a positive review. It is important to note that a threshold of 0.6-1 was used to determine a positive review because a threshold of 0.5-1 would include some more neutral reviews. Neutral reviews are more likely to lean negative than positive, which is why the threshold for positive reviews was moved higher.

For training and testing purposes, the weights are then transformed into an array that is used to train the model. The movie review training set was divided into 80% training data and 20% development using the same seed each

time to provide consistency between each model epoch.

For each epoch, the accuracy and loss function of the model were plotted to determine the number of total iterations needed to train the model. To determine the epochs, we looked at where the lowest point the training and development/validation set are to determine the necessary number of epochs. This epoch was recorded for use on the testing set.

After each tokenization method was trained, it was then ready to be tested by running the test set through the model. All these steps were performed for each tokenization method.

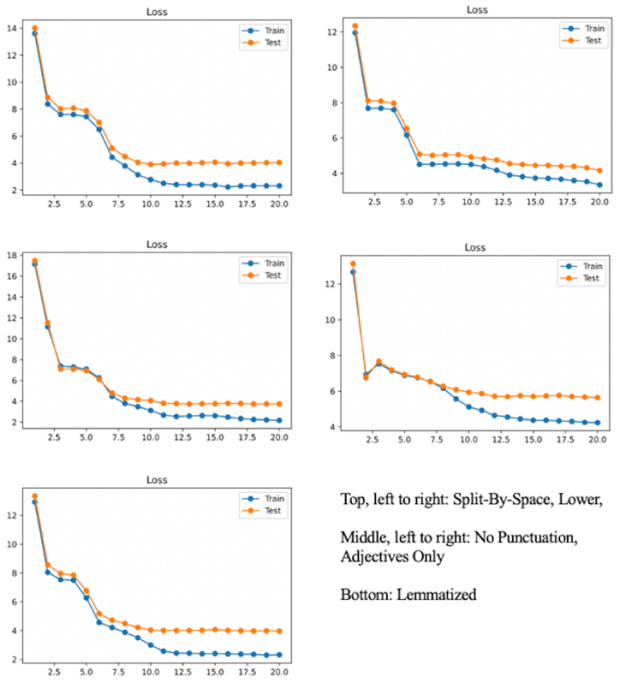

Figure 1: Loss functions determined for each tokenization method.

EXPERIMENT

Tokenizers

Five different tokenization methods were performed: split-by-space (used as a control), casting all tokens to lower-case, stripping punctuation, only adjectives, and lemmatization. Both the lemmatization and adjective tagger were performed with the help of the NLTK Python library. Our team discovered accuracies for each system by comparing the output of each tokenizer to the answer key.

The split-by-space method is the most trivial tokenization method. Each of the reviews were read as strings, which were then tokenized by the spaces in the review. These tokens were then fed into the model.

The lower case tokenization follows the same logic as the split-by-space method. Reviews were split the same way, but each token was then cast to a lower case string. This method was performed to test the effect of removing capitalization on sentiment.

For the no punctuation method, the review was tokenized, then all punctuation tags were ignored during the rest of the experiment.

Similarly to the removal of punctuation method, adjective tokenization focused on part of speech tagging and only included tokens tagged as adjectives. By experimenting on just one part of speech, our team tested whether just one part of speech could determine the sentiment of a movie review.

The last tokenization method was lemmatization. Lemmatization is the process of deriving a root for each token. Each review was split by space and then each word was lemmatized to its root when applicable. For example, the word “better” was tokenized to “good”, and the word “sang” has the word “sing” as its root. Similar to the adjective experiment, we are testing what the most important parts of a sentence are to determine sentiment.

Data

The first step of the experiment was to clean the dataset of miscellaneous characters and

number values. Then the reviews were tokenized by each specification. The tokens were then outputted to a CSV file that contained one row for each tokenized review across five columns of each method of tokenization.

Then, the model was trained and the loss per iteration was analyzed. By determining the number of epochs necessary for each tokenization, our team chose the maximum number of iterations necessary to avoid overfitting the model. Subsequently, split-by-space required 16 epochs, lower case and no punctuation both required 20 epochs, the adjective tokenizer required 15, and the lemmatization method required 19.

Then, the output was compared to the answer key. The total number of incorrect predictions by the model were calculated, and the graph of the accuracies are below.

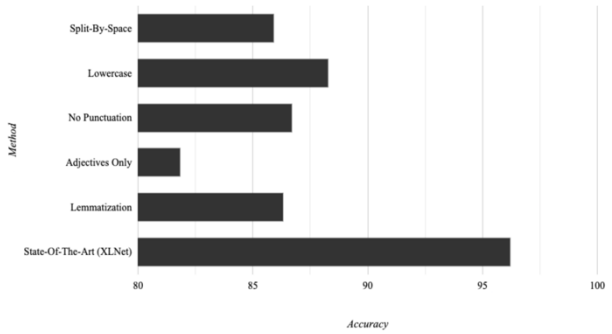

Figure 2: The accuracies for each trial compared to state-of-the-art (BERT) accuracy in the XLNet experiment (6).

The accuracies discovered were as follows: split-by-space at 85.92%, lowercase was 88.29%, no punctuation was 86.71%, adjectives only was 81.84%, and lemmatization was 86.32%.

After the data was collected and analyzed, the reviews that were inaccurate by a margin of 90% were compiled for analysis. The reviews were compared to the tokenizations derived from them to discover reasons why the system may have failed in certain cases.

CONCLUSION

The accuracy of each trial is under 90%, which is below the range of acceptable machine error. Therefore, some components of the system were inaccurate and it is crucial to determine what those errors are.

Comparing the accuracies to each other may be a better way to analyze the data and determine which systems worked better than others. Since split-by-space is the most trivial of all the systems, it is a good base to compare the other four tokenizers against. By comparing the failures of split-by-space against the failures of other systems, it may become more clear where the sources of error were.

Lowercase vs. Split-By-Space

The split-by-space tokenization achieved a 85.92% accuracy while the lowercase method achieved a 88.29% accuracy, a net difference of +2.37%, which is the best our team recorded. Lowercase seems to have done much better than the other systems because the meaning of capitalization is trivial, but by categorizing more identical words together in the vectorization step, the system was able to learn with more data. Both systems seemed to struggle with two types of reviews: long reviews and reviews with opposite connotation to their meaning.

Since both split-by-space and lowercase tokenizers leave all tokens in a sentence intact, there may have been an overwhelming number of tokens to be analyzed. For example, a review with a lot of satire may be labeled as positive instead of negative because the system is unable to tell that long strings of positive words may be understood as negative, and same with negative

connotative words that may have been used in positive context. Cultural understanding is lacking from our models and is something that all machine learning systems struggle to record and quantify. It is the lack of this understanding of context that seems to be a common theme between all the tokenizers.

No Punctuation vs. Split-By-Space

The no punctuation method also did better than the split-by-space method, but only by a margin of +0.79%. This is within the range of error that is acceptable machine error. The team concluded that removing punctuation does not have a very large impact on the ability of the model to understand sentiment. It is worth noting, however, that removing punctuation reduced the "white noise" in the system, so that the model would give more weight to verbal characters. The no punctuation method also seemed to struggle with the same issues as lowercase and split-by-space.

Adjectives vs. Split-By-Space

The adjectives-only method performed the worst of all the tokenizers at an abysmal 81.84%, which is a net of -4.08% off of the split-by-space method. Regardless of the error that has already been described, the adjective method performed particularly poorly. This may be because just one part of speech is not enough for the system to accurately be able to determine sentiment. For example, there is a very large difference between "good" and "not good," but the system was not able to determine context because "not" is not an adjective. Since so much context is missing, it is not surprising that the adjective trial would not be able to determine nuance in the movie reviews.

This could be improved upon by better tokenizers in the future. If a tokenizer was to store data about a word's predecessor, such as whether it was a negating word, there would be more context to the system's understanding of the part of speech.

Lemmatization vs. Split-By-Space

Lemmatization also performed only marginally better than the split-by-space method, at 86.32%, a +0.40% change from split-by-space (similarly to the no punctuation method). While this difference could also be explained by machine error, it could also be because lemmatization reduces the tokens into a more machine-readable form. By lemmatizing (stemming) each of the tokens, the machine was able to break down words into base forms, which may have been easier to determine the sentiment of.

Sources of Error

Some possible sources of error are now proposed. As with many language systems, some of this analysis may be subjective due to different interpretations of the meanings of words.

One source of error is that only one experiment was done with each tokenization method. The split-by-space, lower case, and no punctuation tokenizers are all more objective, because there is only one way to do each of those tokenizations. However, the lemmatization and adjective only methods are much more subjective than the other three methods. For both lemmatization and adjective tagging, our team used the NLTK Python library. The lemmatization method used the NLTK tokenizer and the adjective method used the NLTK adjective part of speech tagger. Since the team restricted ourselves to just one method for each, there may also have been errors. Therefore, further experiments may benefit from diversifying the number of library tokenizers and taggers used, or from testing just one tokenization method with many different tokenizers.

It is also worth noting that the reviews are not all consistent in structure. Some lack proper punctuation and grammar, which may also have resulted in more machine error than normal. Some include slang and foreign language words, which have not been sufficiently analyzed and tagged in language processing libraries.

The number of epochs performed may have also been a source of error. Each model was trained on a training set, but if there were any errors in either the model or the tokenization, then the number of epochs discovered may not have been accurate.

Likely the largest source of error was that tokens were used without regard to order of the words. None of the tokenizers measured how one word effected the meaning of the following tokens. TFIDF looks at the document as a whole, not as token relations. For example, in the case of negation, the computer did not differentiate between tokens like “good” and “not good.” This is a critical error, because negation is fundamental to human understanding of sentiment.

All of these sources of error combined led to the inaccuracy of the system. The team intentionally chose a simple logistic regression model and a very basic tokenization; the idea was to look for the simplest methods and what results those methods would achieve. Even though the data set and program are relatively small, our results provide interesting pointers as to why the BERT models work better and what obstacles exist when processing natural language. The most fundamental machine learning models already achieve results like this, which is impressive when considering the future trajectory of the field.

REFERENCES

1. Bonfil, Ben, and Ieva Staliunaite. "Breaking Sentiment Analysis of Movie Reviews." *Proceedings of the First Workshop on Building Linguistically Generalizable NLP Systems*, 2017, pp. 61-64, https://www.aclweb.org/anthology/W17-5410.pdf.

2. Johnson, Rie, and Tong Zhang. "Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings.", 2016, https://arxiv.org/pdf/1602.02373.pdf.

3. Raford, Alec, Rafal Josefowicz, and Ilya Sutskever. "Learning to Generate Reviews and Discovering Sentiment.", 2017, https://arxiv.org/pdf/1704.01444.pdf.

4. Sun, Chi, et al. "How to Fine-Tune BERT for Text Classification?", https://arxiv.org/pdf/1905.05583.pdf.

5. Yang, Yi, and Jacob Eisenstein. "Overcoming Language Variation in Sentiment Analysis with Social Attention." *Transactions of the Association for Computational Linguistics*, vol. 5, 2017, pp. 295–307, https://www.aclweb.org/anthology/Q17-1021.pdf.

6. Yang, Zhilin, et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." *NeurIPS 2019*, no. 33rd Conference on Neural Information Processing Systems, 2019, https://arxiv.org/pdf/1906.08237.pdf.