Movie Sentiment Analysis with Machine Learning

Jun Xin Cao

1001478171

27/11/2019

STAD92

## Abstract

The purpose of this research project is to develop a program that allows the users to find the sentiment of a certain movie. The program will gather reviews from IMDB from the “most helpful” section of the reviews to maximize user sentiment. The idea is the more upvotes a review has, the stronger it represents the viewer base. For this project, the programming language python and appropriate libraries will be used. A naïve Bayes classifier will be built using term frequency-inverse document frequency (TFIDF).

## Introduction

Term frequency-inverse document frequency (TFIDF) is a numerical statistic that shows the importance a word is in a corpus. The TFIDF value is adjusted relative to the frequency of a word in a document and is taken account by the number of documents in the corpus containing the word. This allows the program to determine which word are consider “common” and thus have little or no value to determining the sentiment. In movie reviews, the words “movie” and “film” appear often in both positive and negative reviews so they will hold almost no value.







The program utilizes a naive Bayes model to predict the sentiment of the movie review.

A screenshot of a cell phone

Description automatically generated

Together with TFIDF it would be: P(word1|class)+P(word2|class) + ... + P(wordn|class) = (total\_words\_in\_class + total\_unique\_words\_in\_class) / (total\_words\_in\_class + total\_unique\_words\_in\_all\_classes)

## Cleaning the data

Whenever raw data is gathered there is bound to be a lot of noise. In order to use the data, it must be standardized. Texts can be standardized by placing everything in lower case, stemming remove special characters, numbers, alphanumeric characters and stop words.

The first step would be to place everything in lower case with the “lower” method from Python. Special characters except spaces and dashes will be removed because they provide additional information such as word separation and joining. Numbers and alphanumeric characters will be removed as they hold no value.

Next, stop words will be removed because they are too common to provide additional information. Words such as “the”, “and”, “with”, etc. can be thought of as place holders in sentences to make it flow and easy to read. However, to the program, they are unnecessary and slows down the classifier.

Lastly, words need to be stemmed. Stemming refers to the process transforming inflected forms of a word into their stem form. For instance, words such as “carries” and “carrying” both have the stem form of “carry”. This is important as words with similar meaning can placed into a single group (stem form). This helps the program learn and determine the sentiment of the text with higher precision. The NLTK stemming function will be utilized here.

## The classifier

The classifier needs to be trained using movie reviews that someone has sorted as positive or negative so the program can learn from it. For this endeavor, the NLTK movie corpus will be used. The collection contains 1000 positive and 1000 negative IMDB movie reviews. However, the reviews look like the following:

A screen shot of a person

Description automatically generated

Clearly, none of the noise in the text have been removed so the corpus cleaner function from cleaner.py will be used to clean it.

The next first step would be to split the reviews into a training set and testing set to verify the classifier’s accuracy. The CountVectorizer function from Scikit-learn will be used to gather the 5000 most common words in positive and negative reviews and map them to an index in a dictionary. The review text will be “tokenized”. Tokenization refers to the process of classifying sections of a string of input characters. In this case, the text will be converted to a list of words. The NLTK library will be used for this process.

The next phase would be to transform and fit the data using fit\_transform function. And the text would need to be converted to TF-IDF values using TfidfTransformer function. All that is left to do is train the classifier using the Multinomial model using the MultinomialNB function. Python’s built in pickle library will be used to save the trained classifier so that it can be used later to predict new reviews. Lastly as a sanity check, a confusion matrix will be created using the test set of reviews to see how accurate the classifier is. It turns out, it is about 80.5% accurate – this accuracy will be explored in the limitation section of this report.

All that is left to do now is create a function that opens the saved classifier and take in reviews so that it can make a prediction. This function will be called “Categorize” within classifier.py file.

## Getting the movie reviews

Since the program utilizes IMDB reviews for training, it makes sense to gather user reviews from the same source. The site is tricky to navigate so the library selenium will be used to automate this process. Reviews with the spoiler tag do not load until it is expanded by the user. Selenium will click on all the spoiler review for this part. Additionally, an ad blocker will be added to the web driver (browser) to disable ads that block the review button from being clicked. Images will be prevented from loading to accommodate slow internet connection. The sample size will be up to 50 movie reviews. As a sanity check, user ratings will also be collected to verify the program’s predictions.

An algorithm called Levenshtein distance will be used to match the user inputs to the closest movie title. The algorithm calculates the number of edits required to transform the user’s inputs (string 1) into the search result (string 2) and returns a number. The lower the number, the closer the match. An edit consists can be inserting, deleting, replacing, or rearranging a character. All these functionalities will be taken care of by the “IMDB” class found in cleaner.py.

A screenshot of a cell phone

Description automatically generated





Also, the top 20 most used words for the movie will also be gathered to get a glimpse of what the movie is about and the general sentiment of the movie. This achieved simply by combining all reviews (after the data is cleaned) and used python’s “most\_common” function.

## Making the user interface

I opted to use a basic console application using the command line. Since all we are interested are numerical values that the program spits out it would be the perfect fit. I used a while loop to gather user input for the movie name and the year it was released. The while loop also allows the user to search for another movie if they choose to afterwards. The year is particularly important as some movies have the same name and it also differentiates the remakes of the same movie. This is all stored in the file “user.py” and it should be run to start the program as seen in the demo video.

A screenshot of a computer screen

Description automatically generated

## Limitation of the classifier

Firstly, it is important to note that the classifier is not trained to detect sarcasm. As such any sarcastic comment may be treated as positive reviews. Sarcasm is difficult to detect in the form of text even for humans because there is no tone that can be heard in text. Sarcasm is defined as “a sharp and often satirical or ironic utterance designed to cut or give pain” (*Merriam-Webster's collegiate dictionary*, 1999). This would require knowing references and context in order to detect it. Some are obvious such as italic words, capital or bolded letters. However, the program was not designed to capture any of that.

Secondly, the program does not pick up negatives such as “do not like” or “I really want to like this movie”. This is one of the major reasons that contributed to the 80% accuracy of the program. A great example of this was when reviews of “Star Wars: The Last Jedi” was tested.

Finally, the classifier is binary in nature, meaning there is no neutral review detection. In testing, neutral reviews seems to steer towards negative so a movie sample’s sentiment may seem more negative than it is. For example, a review that contained mostly neutral tone such as “it’s okay”, “somewhat enjoyable” was predicted as negative by the classifier.

## Programs, libraries and add on

Programs needed:

* Python 3.6 or higher
* Firefox web browser

Use the pip or pip3 (on MacOs) command on terminal to download the libraries**: Numpy**, **Scipy**, **Scikit-Learn**, **Statistics**, **Selenium**, **NLTK**

Additional add on for NLTK can be installed by running these lines of code on the Python shell:

* Import NLTK
* NLTK.download(“stopwords”)
* NLKT.download(“punkt”)

## References

Sarcasm. (n.d.) In Merriam-Webster’s collegiate dictionary. Retrieved from <https://www.merriam-webster.com/dictionary/sarcasm>

Wikipedia contributors. (2019, September 11). Levenshtein distance. In *Wikipedia, The Free Encyclopedia*. Retrieved 20:04, October 31, 2019, from <https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=915176394>

Wikipedia contributors. (2019, September 24). Tf–idf. In *Wikipedia, The Free Encyclopedia*. Retrieved 22:48, October 31, 2019, from <https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=917588674>

Wikipedia contributors. (2019, October 22). Naive Bayes classifier. In *Wikipedia, The Free Encyclopedia*. Retrieved 22:54, October 31, 2019, from <https://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=922565874>