Project 1 - Sentiment Analysis - NLTK

Abdelghafour Mourchid - CSC 5354

Source Code: Available HERE or with the zip file

Table of Contents

Table of Contents 1

Project Overview 2

The Corpus - Sentube 2

The Process - Sentiment Analysis 2

Step 1 - Data Retrieval 3

Step 2 - Tokenization 3

Step 3 - Data Cleanup 4

Step 4 - Stemming 5

Step 5 - Classification Training 5

Step 5.1 - Naive Bayes Classifier 5

Step 5.2 - Logistic Regression Classifier 6

Sentiment Lexicons - Sentiwordnet 6

Results and Reflection 7

Project Overview

The objective of this project is to analyze sentiments, using classification techniques as a tool, and Python and NLTK as an implementation framework. The project is a practical approach to the content of the course since sentiment analysis is a classification task that heavily relies on Natural Language Processing in order to be performed successfully.

The Corpus - Sentube

The corpus that we were asked to work with is the Sentube Corpus. It contains metadata and transcripts of some YouTube videos. The corpus contains a total of 217 JSON files that have Youtube videos descriptions of Cars (78 Documents) and Tablets (139 documents). What is noticeable is that, compared to many other online corpora that are available for use (For example, the Twitter Corpus, or the Cornell Corpus). This one seems to be very small. It is also unclassified, so similar training corpora are not available as training sets, and are not flagged to be positive, negative, or neutral.

The Process - Sentiment Analysis

In order to achieve the goal of the project, I needed to perform a sequence of operations on the dataset that we have, in order to remove any irrelevant data and optimize its format for a better use. Before performing the evaluation, the data needs to be parsed, cleaned up, tokenized, and stemmed. Then the classifier needs to be trained on some featured corpus, using **Naive Bayes**

Classifier and/or **Linear Regression**. Last, after dealing with the training sets and some sample test sets, the algorithms would have to execute the real test of evaluating the processed Sentube datasets.

Step 1 - Data Retrieval

The Sebtube corpus is in JSON (JavaScript Object Notation) format, therefore it is easy to parse them and make them as inputs of a python program. For this reason, I used the default **json** and **os** libraries in Python 3.x to open the files in the /sentube directory. The program iterates over all the files which extension is corresponding to a JSON file, and parses it to retrieve the title of the video, as well as the comments of each video, represented by each separate JSON file. The result of this operation is having a list of videos, called **dataset**, each being in turn a list of comments of the videos. A separate list contains the titles of the videos, called **videoTitles**.

Step 2 - Tokenization

The process of Natural Language Processing in general, or classification in particular, relies heavily in being able to tokenize the data in a meaningful way. NLTK contains a huge set of libraries that tackle the issue of tokenization in many languages and in different formats. For example, the TweetTokenizer class takes care of tokenizing web-based sentences, alongside special characters or slang words, in order not to be obliged to handle them separately. It also has the ability to normalize the length of a word, such as converting “gooooood” to “good”. This pushes the tokenization in favor of the user, and simplifies the task of processing data even further. One useful think I thought about, is that alongside the process of tokenization, it is a good thing to convert everything to lowercase characters, since we would be comparing them to words from the english dictionary. To make everything similar and comparable, I used the built in function **lower()** across the whole dataset.

The output of the tokenization process is the same as the **dataset** list before, but instead of having strings of comments, we have a list of words for each comment. For example, “This is a good comment.” becomes [‘this’, ‘is’, ‘a’, ‘good’,’comment’, ‘.’]. This output is stored in the list named **tokenizedData.**

Step 3 - Data Cleanup

In order to simplify the task of classification, it is a good thing to clean up the dataset and remove any data that is not relevant. My assumption is that for every comment, non-english words and special characters need to be discarded, and typos need to be fixed. For that job to be done, I used the PyEnchant library in Python 3.x in order to have access to all the words of the english dictionary. A small comparison between PyEnchant and the Words library in NLTK made me choose PyEnchant as it successfully handles contracted forms in English (don’t, I’ve, etc). It also have a function called **suggest,** that helps fixing typos and replacing words with their correct English substitutes. The output of this step is stored in the **polishedData** list of the program. **NOTE:** because this step requires iterating through all the words of the English dictionary, it slows the execution down and takes several minutes to be done successfully.

Step 4 - Stemming

Stemming is an essential part of retrieving stems from their original words. In doing this tasks, I had to either use the PorterStemmer, or use SnowBallStemmer. I had to choose the Snowball stemmer because it is very efficient in handling exceptions (like sky, skies, etc). Moreover, Martin Porter was one of the developers of the Snowball Stemmer, so it is considered as a newer version of the Porter Stemmer. It also suits the classifiers used in nltk in a very good way. Because the dataset was already tokenized and cleaned up, the stemming process was a much easier task than expected. To perform this operation, I used the snowball stemmer class available from **nltk.stem.snowball** as a tool. The output of this step is a stemmed list of tokenized comments.

Step 5 - Classification Training

Step 5.1 - Naive Bayes Classifier

As a training set for the program, I used the **MovieReviews** corpus available in NLTK. This corpus consists of several reviews of movies, from IMDB, RottenTomatoes, and other movie reviewing sites. This corpus is the closest I could find to the dataset we have. The MovieReviews training set is labeled as positive, negative, or neutral, depending on the words used. In order to keep a uniform training set, I chose a distributed dataset over the available labels (800 positive, 800 negative, 200 neutral). Then I split positive, negative, and neutral instances to keep the balance between the training sets and the test sets. NLTK Provides a NaiveBayesClassifier that can be trained using this data. By applying the features to the training sets, and feeding the set to the NaiveBayesClassifier, we can be done with training the classifier for more accurate results. Then we test the classifier using a custom test set (a sample of 200 reviews). The results were an accuracy of 0.8, which is representative of the level of training that the classifier got. I tried to double the size of the training set (2000 comments sample), and the value was up to 0.85 of accuracy. Therefore I can say that in an ideal scenario, the accuracy level would significantly increase if the classifier is trained with huge datasets.

Step 5.2 - Logistic Regression Classifier

Because logistic regression relies on a binary dependent variable, we have to consider consider a sentiment to be either a negative sentiment (0) or a positive one (1). The goal of the logistic regression in this case is to find the most accurate linkage that would perfectly describe the relationships between the stems used in a review, and whether it is a positive or a negative review. In NLTK, it is called the Maximum Entropy Classifier (MaxEntClassifier). This is mainly used in the case of the lack of knowledge about the distributions, or when it is not recommended to make any assumptions. The MaxEnt requires significantly more time to train than the Naive Bayes Classifier. While the Naive Bayes Classifier took less than 2 minutes, the MaxEnt classifier took double the time for less than half of the data.

Sentiment Lexicons - Sentiwordnet

In order to simplify the process a little bit, it was a good thing I got to use a sentiment lexicon, such as SentiWordNet. This lexicon is a database of English words, grouped into many sets of synonyms. It also records the relationships between them into one synset. SentiWordnet is a Sentiment lexicon that assigns to each synset 3 values: positivity, negativity, and objectivity. For example, the word "happy" has a pos_score = 0.875, neg_score = 0, obj_score = 0.125. The usage of the lexicon depends on the added value it would deliver to the program. In my case, I used it as a tool for evaluation of the two previous learning methods. It was very useful to provide a kind of expected value from the dataset in question (Sentube), however, it failed very miserably in case of sarcasm, slang, or complex context meanings. A very simple way was to assign a polarity value to each word of the comment, and compute a polarity average for each comment. The results were a bit far from what the two other methods have provided, but the synset evaluation helped a lot in the training datasets that we have in hand.

Results and Reflection

Since I was used to work using python, and I was very interested in Artificial Intelligence in general, and Natural Language Processing in particular, the beginning of the project seemed very interesting to me. The data handling parts were a bit tedious, but they were not very hard. The relative easiness of using Python and installing NLTK corpora made the job much easier for me, since I only had to worry about the format of the data (training sets and test sets), and rely on the NLTK implementation to do the rest. I think that from a context viewpoint of the course, I was very satisfied to know about the Naive Bayes Algorithm (Either in Artificial Intelligence or in NLP), and about Logistic Regression, and it was more interesting for me to see them in action in this project.