CSCI-B659 Advanced Natural Language Processing

Qualitative comparison of classifiers and preprocessing techniques for Sentiment Analyzer

Fall 2016 Term Project

Pratik Sanghvi (psanghvi)

Sameedha Bairagi (sbairagi)

Shashanth Devadiga (ssdevadi)

Srikanth Holavanahal (sriksrin)

# Introduction

With the emergence of social media since last decade, people are more connected across the world. They can voice their opinions on almost anything despite staying miles apart from each other. For example, thousands of users can review a recent sci-fi movie worldwide. This leads to information overload on the web with unstructured data. Hence, there comes a need to process this data and gain meaningful information about the subject (Movie in our example). Characterizing or extracting the correct sentiment of the text using machine learning and NLP techniques is called Sentiment Analysis.

We started our research with a class assignment to discuss feasibility and timing constraints for a generic sentiment analysis. Although, it was a small task, our results analysis gave us the initial boost to research further in this area. In this project, we aim to evaluate performance of four classifiers(k-nearest neighbors, support vector machines, neural networks and naïve bayes classifier) and study the effect of different pre-processing techniques such as stop words, lemmatization, spell correction by categorizing movie reviews according to their sentiment.

# Project Scope

In this project, we are trying to make a qualitative comparison among different classifiers for Kaggle Movie Reviews Dataset (add link here). The dataset contains movie reviews collected from Rotten Tomatoes and each review is categorized into any one of the five classes - negative, somewhat negative, neutral, somewhat positive, positive.

For performance evaluation, we are using train set (8544 reviews) to perform a 5-fold cross validation. Additionally, we attempt to predict the label for any random movie review (phrase taken from test set or written by human). In terms of pre-processing, we limit ourselves to stop words removal, lemmatization and spell checking.

# Approach

Software  
We used python 3.5 with scikit-learn (an open source machine learning library) to perform the classification tasks. Feature extraction and pre-processing tasks are done with the help of nltk libraries (Natural language toolkit for python).

Pre-processing techniques

Each sentence is parsed through four pre-processing techniques in sequence. Following example shows the output after each technique is applied –

Input: [Add sentence here]

* Tokenization

This is a process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens.

[Add sentence here]

* Stop words Removal

Stop words are the most frequent words which appear in the text and do not add any importance to the text. They are present to connect the words to form a sentence, but their absence does not remove logical sequence or context of the text. Example of stop words are ‘to’, ’from’, ’all’, ‘are’, ‘as’ etc. Eliminating stop words enhances the content quality of text.

[Add sentence here]

* Lemmatization

Lemmatization in linguistics is the process of grouping together the inflected forms of a word so they can be analyzed as a single item, identified by the word's lemma, or dictionary form. Essentially, this technique is used to identify the root/stem of a word. For example, the words ‘fly’, ’flies’, ‘flying’, ‘flown’, all have a common root – ‘fly’. The effect of lemmatization increases recall while searching for queries.

[Add sentence here]

* Spell Correction

When we are dealing with data which is created by multiple users, we are quite certain to get erroneous results due to spell errors caused by users who input the data. We would like to work on spell check to avoid mis-categorization of words during lemmatization. This will yield us better accuracy.

[Add sentence here]

Feature extraction

To generate feature vector space, we use three types of model as discussed below, each having it’s own advantages and disadvantages. Each of these types is discussed below in brief –

* Unigram

All the tokens in corpus are considered as a bag of words, independent of each other. Hence, such model does not hold the context information. For larger datasets, such model is prone to mis-categorization.

* Bigram

The tokens are stored in pairs where adjacent words are connected to each other and form one entity to be added in the feature vector. Since, this model holds more information compared to unigram, it works well when dataset is large.

* Hybrid

This is a combination of unigram, bigram and/or trigram model where only high frequency bigrams/trigrams are selected to be added to feature vector. Since it encodes three different type of entities, it has more information but as the order increases, the vector becomes sparse.

Process

Add a diagram here (may be a flowchart)

* The kaggle dataset has 8544 movie reviews in the train set. We process each movie review to construct an **equivalent** feature vector space.
* The feature vector space is a row vector with binary values – 0 or 1. The number of columns in this vector is determined by the total number of words in corpus.
* For each sentence in the train input, pre-processing tasks are applied which include removal of stop words, lemmatization, and spell check.
* A <big text> is maintained to hold the correct spelling of an English word. This <bible> acts as ground truth for spell checker.
* A vector space is generated for each sentence where 1 represents the presence of word in sentence and 0 otherwise. For example, for a corpus with 10 words, feature vector for sentence ‘dog is asleep’ would look like

home under the dog play asleep bark good is at happy

[ 0 0 0 1 0 1 0 0 1 0 0 ]

* Next step involves fitting the generated vector space into a model by choosing a specific classifier.
* Once model is generated, we perform cross validation to compute performance measures such as precision, recall and accuracy.

# Usage

The code can be run on any machine which has Python 3.5x installed on it. Additionally, scikit-learn and nltk libraries should be imported [Add link here].

Command line syntax to run the code –

python Sentiment.py <classifier> <model> [input text]

* classifier represents the option to specify the type of classifier. Supported values are -

knn – k-nearest neighbors

nnet – neural network

svm – support vector machines

nb – Naïve Bayes

* model represent the model for feature extraction

uni – unigram

bi – bigram

hyd – hybrid

* input text is the optional parameter which represents any review which needs to be categorized

# Observations

Findings about dataset, different type of reviews (using slang)

# Results

Add details about comparison between different classifiers

Can have some text to explain why changing a certain tuning parameter gave better results

Add graphical comparison charts over here

# Future Work and Conclusion

# References