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: The input here is taken from a single text file consisting of 8544 Movie reviews. The reviews are split into phrases and each phrase is tagged with labels 0-4, in the increasing order of positive emotion.

* Tokenization

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

In this project we tokenize the text into Unigrams , bigrams and trigrams. The nltk library is used to generate bigrams and trigrams. These is generated on a sentence by sentence basis.\

* 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.

We obtain the list of stop-words from the nltk corpus. Each token that we had obtained in the previous step is checked for the presence of stop-words. A token is only added to the feature space if it is void of any stop-words.

* 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.

This is the done with the help of WordNetLemmatizer form the nltk library. Each word in a token is lemmatized before it is added to the feature space and creating a feature vector.

* Spell Checking and 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.

In our project we employ both - spell checking and spell correction. Each word in a token is first check for the correctness of its spelling. The spell checking is done using the enchant library. If it fails the spell check then we try to correct its spelling. For spell correction we employ the approach as explained by Norvig here[1].I first build a corpora of words using a text file called big.text. This is a book which uses many of the frequently used English words. To correct a word , we check for the word in this corpora and if it is absent then we assign the closest match.

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.

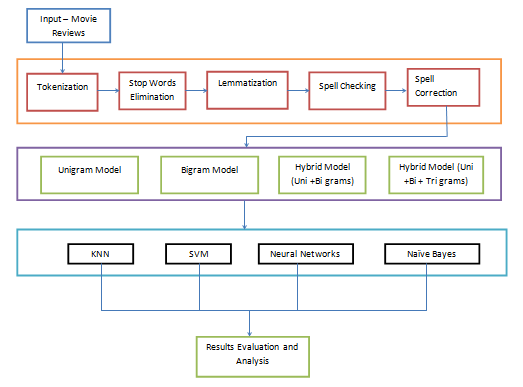
* Bigram

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. This can be considered as a linear interpolation of unigrams, bigrams and trigrams.

Process



* 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.

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

hybrid – hybrid (unigram + bigram)

hybrid2 – hybrid(unigram + bigram + trigram)

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

# Observations

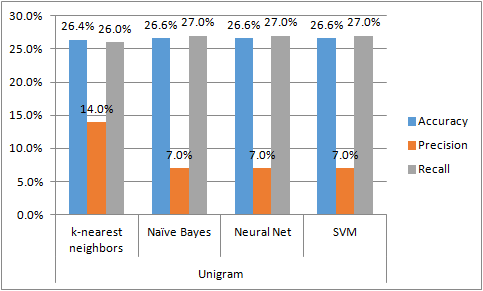
Findings about dataset, different type of reviews (using slang)

* Using slang in a review makes it difficult to categorize. Often slang words depend upon the user's region and/or background. In such cases it is difficult to built model taking such words into consideration.
* Too short or too detailed review. In case of reviews that are too short , there isn't much information to process with. The feature vectors created in such cases are extremely sparse. In case of too detailed review, there could be too much unnecessary information. The model might learn too many extra words that serve no tangible purpose.
* Using sarcasm in reviews can also lead to wrong interpretation by the classifier.
* Mixed review. Sometimes reviews can applaud some aspects of the film and criticize others. Thus the analysis of overall sentiment can be tricky.

# Results

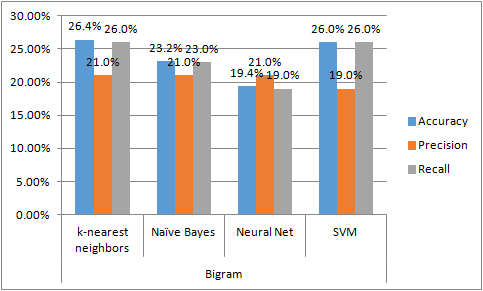
The baseline result when we consider random guessing is 0.2 or 20%. The results of the various models with respect to the different classifiers are as follows:

* Unigram



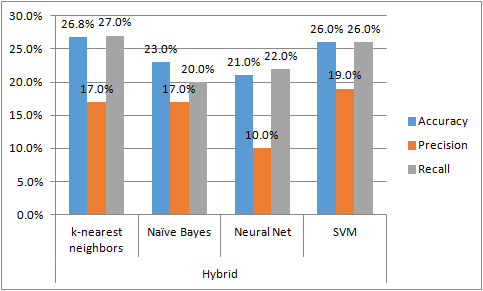
The graph below shows that although the accuracy is nearly same for all the classifiers, precision for k-nearest neighbors is better than other classifiers. This may be acknowledged be choosing optimal value for k = 5 which suits the model and reduces the overall noise.

* Bigram



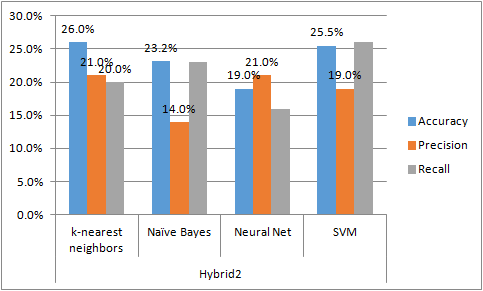
Bigram results in more sparse vectors. Hence it also reduces the accuracy

* Hybrid (unigram + bigram)



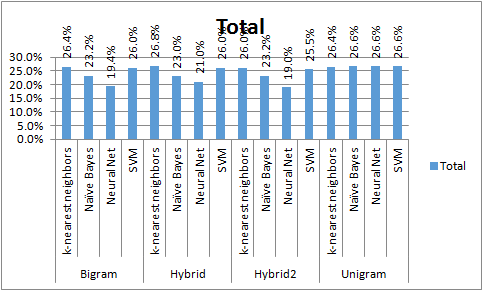
As we can see below, this gives the highest accuracy for kNN. The reason behind this is presence of more information in the vector space. With the use of unigrams+bigrams, we are able to provide better information to the classifier which can predict the label correctly.

* Hybrid2 – hybrid(unigram + bigram + trigram)



Although this has more information, it also has a sparser feature space due to trigrams. So accuracy reduces

The overcall accuracies are as follows:



Hybrid model with k-NN classifier gives the best result.

# Future Work and Conclusion

Following our short term goals, our next steps in the process would be to include part-of-speech tag information in the feature vector. The POS-tagging model would be different compared to the sparse matrix which we used in this project. It will have (word, tag) tuples as features which would hold the context information of the review.

We would also like to apply our model on large dataset (collection of nearly 1 million reviews) and draw better analysis for relationship among pre-processing techniques, classifier and execution time. For example, a large dataset may perform well for SVM classifiers, but can be extremely slow for k-nearest neighbors.

We understand the accuracy obtained by our current approach is low, hence, we plan to include more preprocessing steps in our process such as -

* handling of synonyms using WordNet (where feature vector is created after the word is replaced with root synonym),
* giving a percent score for class labels instead a discrete label where a sentence can be labelled as 'y % positive' or 'y % negative'. Such score is calculated by summing up scores of similar words, decrementing scores for negative words etc.Note that, this method will be used to test a new algorithm for sentiment analyzer and evaluate it with ground truth from train set.
* Using Smoothing could also give better results.

Another possible route could be to learn regression based model. Generally for movies, review at the time of release tend to be more extreme either positively or negatively due to the hype surrounding it. However, as time passes, the words expressing strong emotion are generally less frequent . Also reviews written after a certain passage of time could also possibly be more level-headed. So if we could train a regression-based model taking time (or other relevant parameter depending on the domain) into consideration, then we might be able to get a better sentiment analyzer.

To conclude, we can say that the project is a preliminary attempt to analyze the performance of different classifiers and pre-processing techniques and our results show that for a small dataset, SVM worked best with hybrid model with 26.6% accuracy. We can build upon this project to extend for a more complex sentiment analyzer.

# References

1. How to Write a Spelling Corrector. February 2007. Retrieved from <http://norvig.com/spell-correct.html>
2. Bing Liu. In Morgan & Claypool Publishers, May 2012. Sentiment Analysis and Opinion Mining.
3. Stemming, Lemmatization and POS-tagging with Python and NLTK. (2015, January 26). Retrieved from <https://marcobonzanini.com/2015/01/26/stemming-lemmatisation-andpos-tagging-with-python-and-nltk/>
4. Dr.S.Vijayarani et al , International Journal of Computer Science &amp; Communication  
   Networks,Vol 5(1),7-16
5. <http://pythonhosted.org/pyenchant/tutorial.html>
6. http://www.nltk.org/howto/wordnet.html