Baliganahalli Narayana Murthy, Sunil
Apr 28, 2017

Natural Language Processing

Programming Assignment 4 Report

Dataset

The movie reviews dataset from NLTK consists of 2000 documents tagged with positive and negative sentiments. The focus of this assignment is to build a model to classify movie reviews.

Preprocessing

- *Baseline or Bag of words (BoW):* The first and foremost thing I did was to obtain a baseline accuracy without any feature selection i.e throw the entire document (bag of unigram words) at the model. With the Naive bayes classifier I was able to achieve accuracy of low 70's.

- *Stopwords:* The next thing i tried was to remove the frequently used words in english. I thought words such as *"the", "that", "be"* etc doesn't give any information about the review. But the accuracy of the model went down due to the fact that stopwords also contain negative words like *"won't", "aren't", wh-words etc* which are relatively good indicators for the review.

- *Stemming & lemmatization:* In english there are so many word forms like inflectional (singular and plurals) and derivational. Stemming and lemmatization also reduces the sparseness of the words by converting all words to their simplified form (common base form). But stemming and lemmatization by itself didn't improve the accuracy.

- *Remove digits and punctuations:* I also removed all the non-alphanumeric characters which doesn't add any information about the reviews. The accuracy went down (in comparison to the baseline) due to the fact that punctuations like *"?", "!"* etc also help in determining the type of review. For e.g. *"What was the director thinking?"* here question mark is critical to determine tone of sentence (sarcasm).

Feature Engineering

- *Bigrams:* Baseline model used only the unigram words which doesn't capture the semantic context of the sentence/ movie review. So I included bigrams to capture contextual information such as *"not good", "didn't like", "bad movie"* etc. This model is still a bag of words but the bag contains bigrams instead of unigrams.

- *Trigrams:* Trigrams create a even better model because they capture better contextual information. For e.g. Trigrams capture *"not good movie"* where as bigrams capture *"not*

good", and *"good movie".* The phrase is clearly a negative review but it's not evident in bigram model because of the bigram occurrence *"good movie"*.

- *N-Grams:* In general we can use N-gram model and find the optimal value of N. As appealing it might be to run for larger N to create better model it turns out that as N increases it also increase the computational complexity. For the purposes of easier computation and running the program with a reasonable time I will stick with bigram / trigram model.

- *POS tagging:* The idea of including parts of speech for individual words feature to improve the model was adopted from this paper [2]. However in my model it didn't improve the accuracy because the paper adopts more complex way of scoring the POS tags. Where in my model I just joined words with tag delimited by slash (" */ ")*.

- *Information Gain:* As previously discussed bigram model does capture the semantic contextual information. The model can be further improved by ignoring all the bigram words that doesn't contribute towards the classification task. This can be done by calculating the information gain for the entire corpus of training data and keep only top N words with max information gain. Where N is to be determined by trial and error.

Features	**Accuracy**	**Precision**	**Recall**	**F1 Measure**
Baseline (F0)	0.714	0.791	0.713	0.69
No stopwords (F1)	0.706	0.789	0.705	0.682
Stemming + lemmatization (F2)	0.71	0.792	0.709	0.686
No numerics, punctuations (F3)	0.711	0.795	0.711	0.688
F3+F4	0.7	0.791	0.7	0.672
Bigrams (F5)	0.782	0.824	0.78	0.771
Bigram + stem + lemmatization	0.759	0.813	0.759	0.748
Trigrams (F6)	0.81	0.826	0.808	0.806
Bigram + POS tagging	0.772	0.819	0.772	0.762
Information Gain	0.929	0.933	0.93	0.929

Table-1: Features, accuracy tabulation. All numbers are normalized.

Other Classification techniques:

I also experimented with other models using sklearn package. The sklearn expects the data to be formatted in a certain way as in it cannot directly take in words unlike NLTK models. So i have to represent the bag of words into vector representation.

Document representation:

- *CountVectorizer:* This sklearn transformer represents the corpus as a 2D matrix by counting the occurrences of all words in the vocabulary.

- *TfidfVectorizer:* This transformer converts all corpus into a 2D matrix of TF-IDF (term frequency - inverse document frequency) features. TF-IDF assigns higher weights to less frequently seen words and lower weights to most commonly used words such as stopwords.

Models:

Below are some of the different classifier I experimented with. I did not tune the classifier parameters to improve the model. The number shown below are without optimization and without any kind of document preprocessing.

Classifier	Accuracy	Precision	Recall	F1 Measure
SGDClassifier	0.746	0.794	0.747	0.731
SVM (linear kernel)	0.829	0.828	0.829	0.828
LogisticRegression	0.839	0.839	0.84	0.838
KNeighborsClassifier	0.617	0.618	0.618	0.616
RandomForestClassifier	0.686	0.697	0.687	0.682
DecisionTreeClassifier	0.656	0.66	0.656	0.652

Table-2: Accuracy, precision etc using CountVectorizer representation.

Classifier	Accuracy	Precision	Recall	F1 Measure
SGDClassifier	0.849	0.855	0.848	0.848
SVM (linear kernel)	0.847	0.85	0.848	0.847
LogisticRegression	0.809	0.814	0.809	0.807
KNeighborsClassifier	0.608	0.674	0.609	0.567
RandomForestClassifier	0.643	0.661	0.644	0.633
DecisionTreeClassifier	0.631	0.632	0.631	0.629

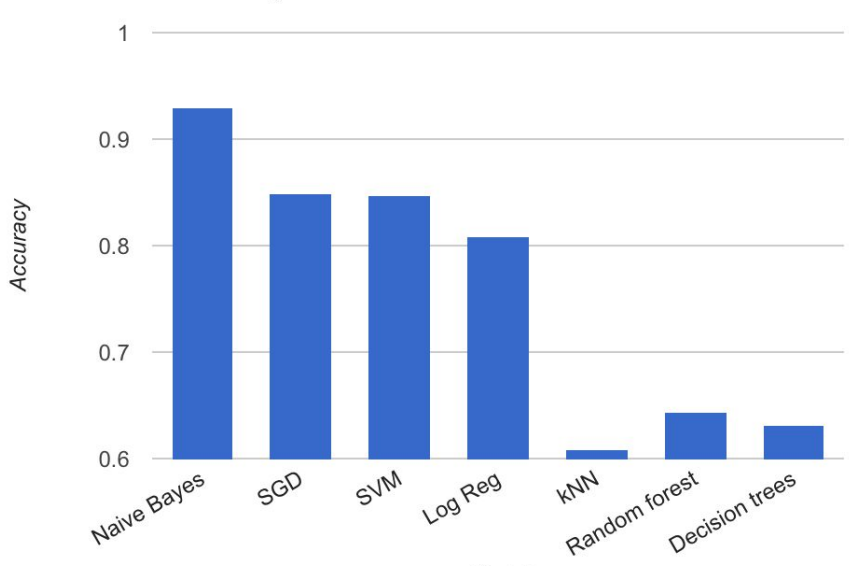
Table-3: Accuracy, precision etc using TF-IDF Vectorizer representation.

Naive comparison of models:

Below table shows how the different models performed with the classification task. Naive bayes outperformed because I did a better job selecting the features. Where as other models like SGD classifier, SVM (with linear kernel) etc did significantly good given the fact these models weren't feature engineered or fine tuned.

Model	Accuracy
Naive Bayes	0.929
SGDClassifier	0.849
SVM (linear kernel)	0.847
LogisticRegression	0.809
KNeighborsClassifier	0.608
RandomForestClassifier	0.643
DecisionTreeClassifier	0.631

Table-4: Accuracy across different models



Most Informative information from Naive bayes model

```
Most Informative Features
                chilling = True              +  :  -      =     15.1 : 1.0
               ludicrous = True              -  :  +      =     14.9 : 1.0
                  seagal = True              -  :  +      =     12.9 : 1.0
                    slip = True              +  :  -      =     11.7 : 1.0
                seamless = True              +  :  -      =     10.4 : 1.0
              astounding = True              +  :  -      =     10.4 : 1.0
               strongest = True              +  :  -      =     10.4 : 1.0
             fascination = True              +  :  -      =     10.4 : 1.0
             outstanding = True              +  :  -      =     10.3 : 1.0
                    3000 = True              -  :  +      =     10.3 : 1.0
              accessible = True              +  :  -      =      9.7 : 1.0
               animators = True              +  :  -      =      9.7 : 1.0
                 rousing = True              +  :  -      =      9.7 : 1.0
                  hatred = True              +  :  -      =      9.7 : 1.0
                   sucks = True              -  :  +      =      9.3 : 1.0
           concentration = True              +  :  -      =      9.1 : 1.0
                 conveys = True              +  :  -      =      9.1 : 1.0
                bothered = True              -  :  +      =      8.9 : 1.0
               illogical = True              -  :  +      =      8.9 : 1.0
                  hudson = True              -  :  +      =      8.9 : 1.0
```

Final Model: Naive Bayes
Final features: I kept top 10000 bigrams out of ~40,000 total bigrams. All other bigrams / words with low information gain were eliminated. Keeping 10K words improved the accuracy of the model at the same it also improved the runtime. One of the critical aspects of classical ML (not neural nets) is that it's *quality over quantity*. For in depth explanation refer to Feature engineering.

References:

1. Text classification using collocation.
2. Improving sentiment analysis with Parts of speech weighting