SENTIMENT BASED CLASSIFICATION OF DOCUMENTS

1. Introduction:

The task of text categorization or classification based on the sentiment of the text is generally tougher than classification based on the topic of the text [1]. So, I examined three machine learning techniques namely language model classifier, perceptron and support vector machines, to classify the documents based on the sentiment.

2. Dataset description

The evaluations were done for the movie review domain. The dataset was obtained from [2]. It consisted of 1000 positive movie reviews and 1000 negative movie reviews.

3. Evaluation

3.1 Experimental Setup

The dataset consisted of 2000 movie reviews with equal number of positive as well as negative reviews. For the evaluation for each of the three techniques I used 5 fold cross validation technique. Each of the fold consisted of 20% of the dataset with even distribution of positive and negative reviews. The training of the classifier was done on 4 folds while testing was done on remaining 1 fold of the dataset and this process was repeated five times while rotating the test dataset in each iteration.

The entire project has been coded in Python. I have used the **Python NLTK (Natural Language Toolkit)** extensively [3]. In most cases I have also included our own implementation of the NLTK routines used. These have been commented.

I have also used **SciKit's SVM** classifier which essentially provides wrappers in Python for **libSVM** for the third experiment [4].

3.2 Language Model Classifier

For the evaluations of the language model classifier I used the following feature vectors:

1. Unigram Frequency
2. Bigram Frequency

The high level summary is as follows:

1. NLTK **Plain Text Corpora** were created for test/train data.
2. There were separate corpora for Positive and Negative training sets
3. Frequency distributions with **Laplace smoothing** for both positive and negative training sets was calculated. This was done for both unigram and bigram data.
4. Using these frequency distributions **Perplexity** was calculated for unigram as well as bigram model for each document in the test set.
5. For Perplexity calculation, I used the logarithm of the probabilities to **prevent underflow**
6. The outcome for both the positive and negative test data were stored in an **excel file** for evaluation.
7. This process was **repeated for five times** with the test data being rotated in each iteration.

Code is present in the file named "*Lang_Model.py".*

3.3 Perceptron Classifier

For the Perceptron classifier I used unigram frequency as the feature vector.

1. For the training data, feature vectors were generated for each file
2. The feature vectors for each file were the unigram frequencies. These were stored in a Python Dictionary which is essentially a HashMap.
3. The weight vector too was a Python Dictionary. For each unique word as key, the values were initialized to zero
4. **Training**: A temporary variable held the dot product of the weight and feature vector for each training instance. The weight vector was increased by the feature vector if the Perceptron incorrectly predicted the document as negative and decreased it by the feature vector if it was incorrectly predicted as Positive.
5. I evaluated this approach and observed that the weight vector converged at 37 iterations.
6. After training, the classifier was used to predict the sentiment of the test data. For this, the feature vector for test data was generated which was then used along with the weight vector to predict the outcome.

Code is present in the file named "*Perceptron_class.py".*

3.4 Support Vector Machine (SVM)

1. For the feature vector I used:

   a. Count **of unique words**.
   b. Count of **1000 most frequent words** in the training set.

2. For training, I created a NumPy array of dimensions: **[NumSamples x NumFeatures].**

3. I also created another one dimensional NumPy array with the actual labels for the Samples. I labelled the positive reviews as 1 and the negative documents as 0.

4. This was passed to the libSVM library to fit the SVM.

5. For testing, I created a NumPy array of dimensions [**NumSamples x NumFeatures]** again.

6. I evaluated SVM using both L1 and L2 regularization.

Code is present in the file named "*SVM_class.py".*

4. Results:

4.1 Accuracy

All figures represent accuracy, i.e. the proportion of test documents predicted correctly

LMClassifier	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Unigram	0.805	0.8175	0.8075	0.825	0.79	0.809
Bigram	0.7525	0.765	0.7675	0.81	0.785	0.776

Perceptron	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Accuracy	0.815	0.845	0.83	0.86	0.8475	0.8395

SVM	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
L2	0.805	0.8325	0.8275	0.8575	0.8475	0.834
L1	0.795	0.8025	0.7975	0.785	0.835	0.803

5. References

[1] *Thumbs up? Sentiment Classification using Machine Learning Techniques* by Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, EMNLP, 2002

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[3] http://www.nltk.org/

[4] http://scikit-learn.org/stable/index.html