CSE Natural Language Processing

Ankur Mittal
anmittal@cs.stonybrook.edu

Vasudev Bhat
vabhat@cs.stonybrook.edu

March 14, 2014

Homework 1

1. **Language Model Classifier:**

(a) Unigram Based Language Model Classifier: For words that do not occur in the training data (i.e. words that occur in positive review only once but not in negative review set or viceversa), we assign an <UNKNOWN> tag and calculate the probability of this tag. During classification we apply the probability of this <UNKNOWN> tag for all new/unseen words. We performed our tests both with and without Laplace Smoothing. The average success rate across all folds increased by 26% when we used Laplace smoothing.

Fold No.	Success % (With Smoothing)	Success % (Without Smoothing)
1	81.0	54.75
2	80.5	54.25
3	78.75	54.0
4	82.25	53.25
5	79.5	53.75
Average	**80.4**	**54.0**

(b) Bigram Based Language Model Classifier: In the bigram model, we ignore any new words that occur in the testing set. The average success rate across all folds increased by 23% when we used Laplace smoothing for the bigram based classifier. We have tabulated the results for the Bigram language model classifier below.

Fold No.	Success % (With Smoothing)	Success % (Without Smoothing)
1	69.75	50.0
2	74.75	50.0
3	74.0	50.0
4	74.75	50.0
5	73.25	50.0
Average	**73.3**	**50.0**

The unigram based classifier performs better than the bigram based classifier since the model becomes more constrained and specific when we use bigrams for training.

2. **Perceptron Classifier:**
We implemented the Perceptron Classifier based on the approach mentioned in [1] . We tested with different number of iterations and observed that there was not much change in the accuracy of prediction. The features used here are: presence of unigrams and presence of bigrams. During the learning phase, if a review is wrongly classified, we calculate the correction by substracting the obtained value of the feature vector from the value corresponding to the expected class (1 or -1). But when performing experimentation we observed that this approach was causing over-fitting. Hence we used an alternate approach where we calculate the correction at all times (not just during wrong classification) and then change the weight vector accordingly. We normalize the correction by the size of the feature vector of the document. We multiply the normalized correction with the learning rate parameter and then add or substract these values from the actual weight vector. We tried different learning rates with which we multiply the correction. We have tabulated the results below which show experiments with different parameters. Every trial is a five fold cross-validation trial and we have reported the average accuracy across all the folds. The formula for correction is shown in equation 1..

$$correction = Expected\_Class - \frac{Obtained\_Value}{sizeof(Feature\_Vector)} \tag{1}$$

Trial No.	**Iterations**	**Learning Rate**	**Unigram or Bigram**	**Avg Success %**
1	1	1.0	Bigram	79.05
2	5	1.0	Bigram	78.7
3	10	1.0	Bigram	78.85
4	1	0.7	Bigram	77.2
5	5	0.7	Bigram	78.8
6	10	0.7	Bigram	78.8
7	1	1.0	Unigram	82.85
8	5	1.0	Unigram	81.75
9	10	1.0	Unigram	81.8
10	1	0.7	Unigram	83.05
11	5	0.7	Unigram	82.6
12	10	0.7	Unigram	81.8

From the results, we can infer that the unigram based perceptron classifier again performs better than the bigram based perceptron classifier as in the case of Language Model Classifier.

3. **SVM Classifier:**
We used the libsvm[2] library for training and prediction using SVM. We trained the model using unigrams and the frequency/presence of these unigrams in the training set. We experimented with different C values as tabulated below. We were unable to set the regularization parameter in libsvm. The features we used here are presence of words and the frequency of words. We have tabulated the results below which show experiments with different parameters. Every trial is a five fold cross-validation trial and we have reported below the average accuracy across all the folds.

Trial No.	Presence/Frequency	Cost Parameter	Avg Success %
1	Presence	1	85.1
2	Presence	10	85.1
3	Presence	20	85.1
4	Presence	30	85.1
5	Presence	50	85.1
6	Frequency	1	66.85
7	Frequency	10	66.85
8	Frequency	20	66.85
9	Frequency	30	66.85
10	Frequency	50	66.85

From the results, we can observe that the presence feature performs better than the frequency feature. We did not observe any change by varying the cost parameter.

4. **Error Analysis:** We found 40 reviews which were being wrongly classified by both the classifiers Perceptron and Language Model classifier. We analyzed a few of these reviews manually and observed that the reviews actually indicate a tone that is opposite to the intention (positive or negative) of the reviewer. For example a positive review (in file cv050_11175.txt) contains words that normally indicate a negative tone and these words would be in the negative set of the training data. Hence the classifiers fail to classify such reviews properly. We have listed some of the files from both the positive and negative set below. The listed categories are the actual category of the files which have been predicted wrongly.

Positive Reviews	Negative Reviews	Positive Reviews	Negative Reviews
cv050_11175.txt	cv010_29063.txt81.0	cv082_11080.txt	cv104_19176.txt
cv214_12294.txt	cv262_13812.txt	cv420_28795.txt	cv571_29292.txt
cv603_17694.txt	cv697_12106.txt	cv842_5866.txt	cv835_20531.txt

To further analyze these errors, we extracted all the common unigrams from the positive set which were classified as negative reviews. We also mined the unigrams from the negative set in the training data and found that there were many unigrams which occured frequently in the negative set which probably influenced these reviews being classified as negative. Now, when these unigrams were found in the positive set in the prediction data, the classifier is confused and predicts the reviews as positives. Some of these words include silly, bastard, stupid and other abusive words.

References

[1] http://en.wikipedia.org/wiki/Perceptron

[2] Chang, Chih-Chung, and Chih-Jen Lin. ”LIBSVM: a library for support vector machines.” ACM Transactions on Intelligent Systems and Technology (TIST) 2, no. 3 (2011): 27.