A Naive Bayes Classifier for Sentiment Analysis using Different Smoothing Techniques

Prasangsha Ganguly

IEM Kolkata

June 2015

Outline

Introduction

- Sentiment Analysis is a computational process to identify and classify the opinions expressed in a text document
- Sentiment Analysis is useful in analysis of user reviews for a particular product
- **Problem Statement:** Using a training data set of text documents classified as positive or negative, we train a Naive Bayes Classifier model which is further used to classify test data

Bayes Classifier for Text Classification

- In supervised text classification, we have a set of classes C and training data set of N documents$(d_1, d_2, ..., d_N)$ that are manually classified into corresponding classes: $(d_1, c_1), (d_2, c_2), ..., (d_N, c_N)$
- Our objective is to train a classifier that can assign a correct class $c \in C$ to an unseen document d
- Bayes classifier is a probabilistic classifier where for a document d, out of all classes $c \in C$ the classifier returns the class $\hat{c}$ which has the maximum posterior probability
  $\hat{c} = argmax_{c \in C} P(c|d) = argmax_{c \in C} P(d|c) * P(c)$
- A document can be represented as a set of features $f_1, f_2, ..., f_n$:
  $\hat{c} = argmax_{c \in C} P(f_1, f_2, ..., f_n|c) * P(c)$

The Naive Bayes Classifier

- Naive Bayes classifiers make two simplified assumptions:
  - The position of the word doesn't matter; the features $f_1, f_2, ..., f_n$ include only word identity not the position
  - Conditional independence of the feature probabilities $P(f_i|c)$; so, $P(f_1, f_2, ..., f_n|c) = P(f_1|c) * P(f_2|c) ... * P(f_n|c)$
- If there are N_c number of documents in training data with class c and let N_{doc} be the total number of documents, then $P(c) = \frac{N_c}{N_{doc}}$
- As a feature f_i is considered as existence of a word w_i in the documents bag of words, $P(f_i|c) = P(w_i|c) = \frac{Count(w_i,c)}{\sum_{w \in Vocab} Count(w,c)}$

Naive Bayes Classifier

- If a word w_i is not present in the training documents of class c_j, then $P(w_i|c_j) = 0$
- Since Naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term will make the probability of the class to be 0
- Smoothing techniques are helpful to overcome this data sparsity issues
- The smoothing techniques generally add some pseudo counts to the words with count 0, thus by making them nonzero

Smoothing Techniques

Laplacian Smoothing

The simplest smoothing technique is Laplacian or add-one smoothing. For each word, we add an extra pseudo count to make nonzero count.

$$\hat{P}(w_i|c) = \frac{Count(w_i,c)+1}{\sum_{w \in Vocab}(Count(w,c)+1)} = \frac{Count(w_i,c)+1}{(\sum_{w \in Vocab} Count(w,c))+|Vocab|}$$

Add-k Smoothing

It is a variant of Laplacian smoothing. Here, instead of adding pseudo count 1, an extra count of k is added to each word.

$$\hat{P}(w_i|c) = \frac{Count(w_i,c)+k}{\sum_{w \in Vocab}(Count(w,c)+k)} = \frac{Count(w_i,c)+k}{(\sum_{w \in Vocab} Count(w,c))+k*|Vocab|}$$

$k \in [0, 1]$

Smoothing Techniques contd..

Jelinek-Mercer Smoothing

Linear interpolation based, where the final probability is convex combination of document probability and vocabulary probability.

$$\hat{P}(w|c) = (1-\lambda)\frac{Count(w,c)}{\sum_{w \in Vocab} Count(w,c)} + \lambda * P(w|Vocab) \qquad \lambda \in [0,1]$$

More the value of λ, more smoothing is obtained

Dirichlet Prior Smoothing

This is a linear technique, where coefficients are dynamic.

$$\hat{P}(w|c) = \frac{Count(w,c)+\mu * P(w|Vocab)}{\sum_{w \in Vocab} Count(w,c)+\mu} \qquad \mu \in [0,+\infty)$$

we have included $\mu * P(w|Vocab)$ pseudo counts to each word in the numerator and thus added $\sum \mu * P(w|Vocab) = \mu$ in the denominator.

Smoothing Techniques contd..

Absolute Discounting Smoothing

Discounting based smoothing techniques reduce some count from the words having count more than 0, and add some pseudo counts to the words having 0 count. Thus, keeping the probability mass constant. If $|c|_u$ is the number of unique words in the class c, then

$$\hat{P}(w|c) = \frac{max(Count(w,c) - \delta, 0) + \delta * |c|_u * P(w|Vocab)}{\sum_{w \in Vocab} Count(w,c)} \qquad \delta \in [0,1]$$

Two Stage Smoothing

A convex combination of Dirichlet Prior probability and the probability of the word in the vocabulary.

$$\hat{P}(w|c) = (1-\lambda) * \frac{Count(w,c) + \mu * P(w|Vocab)}{\sum_{w \in Vocab} Count(w,c) + \mu} + \lambda * P(w|c)$$
$$\mu \in [0, +\infty) \quad \lambda \in [0,1]$$

Implementation

- The Sentiment labeled data set of UCI [3] is used as the data set.
- The sentences are converted into lower case strings and stored along with the corresponding sentiment.
- The words are extracted using tokenization and three non-disjoint bag of words: vocaball, vocabpos and vocabneg are created containing all words, positive words only and negative words only respectively.
- The total data set is divided into two parts; 80% data is used for training and remaining 20% for testing.
- A matrix Mymat is created having 7 rows and n columns where n is the number of unique words in the vocabulary

Implementation contd..

- For the rows having unique words and counts (first 4 rows), the construction of each row takes $O(n^2)$ for other rows it has $O(n)$ complexity.
- For each smoothing technique only the last three rows are to be modified and for each row it has $O(n)$ complexity.

The Mymat data structure

$word_1$	$word_2$	...	$word_n$
$Count(word_1, Vocaball)$	...	...	$Count(word_n, Vocaball)$
$Count(word_1, Vocabpos)$	...	...	$Count(word_n, Vocabpos)$
$Count(word_1, Vocabneg)$	...	...	$Count(word_n, Vocabneg)$
$P(word_1 \mid Vocaball)$	...	...	$P(word_n \mid Vocaball)$
$P(word_1 \mid Vocabpos)$	...	...	$P(word_n \mid Vocabpos)$
$P(word_1 \mid Vocabneg)$	...	...	$P(word_n \mid Vocabneg)$

Implementation contd..

- To evaluate the performance of different smoothing techniques with different parameter values, two methods are written: Test() and Testdoc()
- The Test() method asks to input a single statement and find the sentiment of the sentence
- The Testdoc() method takes a file having multiple sentences already manually classified, it then find the predicted sentiment using the algorithm, construct a confusion matrix; and finally evaluate the accuracy of the algorithm using

$$Accuracy = \frac{True\_Positive + True\_Negative}{Total\_number\_of\_sentences}$$

Results

- We compared the performance of different smoothing techniques for a test document. According to [2], the parameter $\lambda = 0.5$ for Jelinek-Mercer (JM), $\mu = 0.95$ for Dirichlet prior (DP), $\delta = 0.6$ for Absolute discounting (AD) and $\lambda = 0.6, \mu = 100$ for Two stage (TS) smoothing.

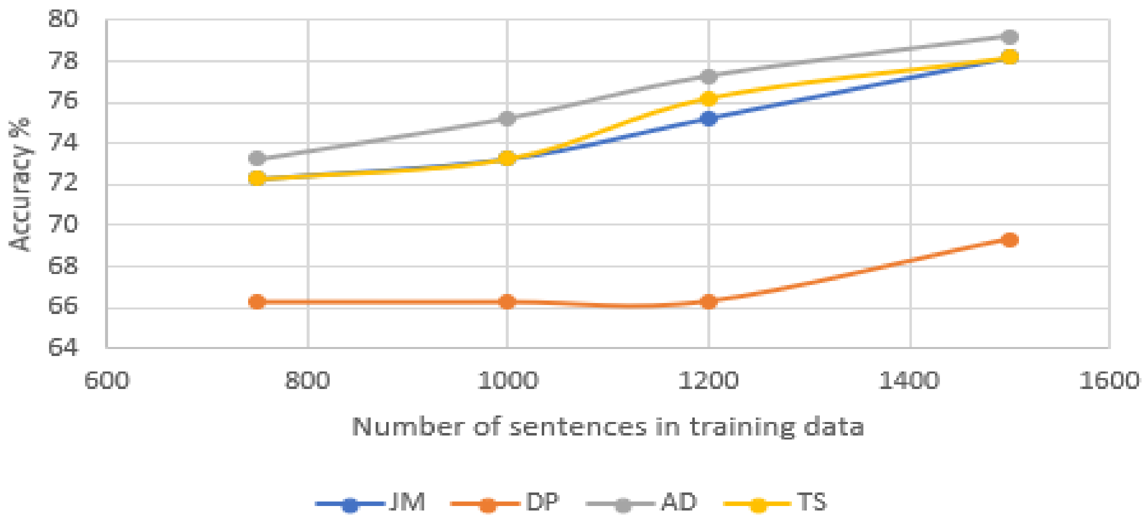
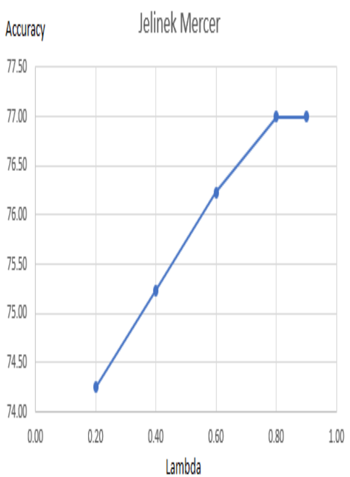


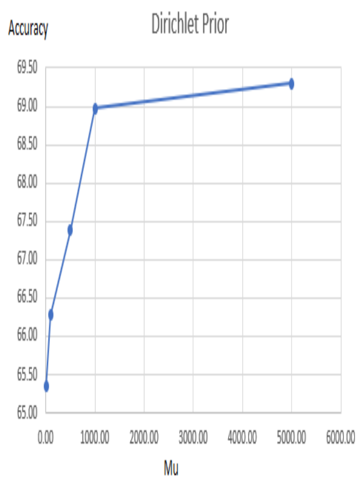
Figure 1: Smoothing algorithms: Training document size vs Accuracy

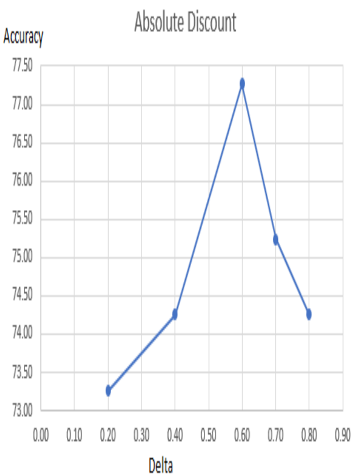
Results

- For each smoothing technique, we compared the performance depending on the parameter of the particular technique.

Different smoothing: Parameter vs Accuracy







Improvements to be done

- Accuracy:
  - As Naive Bayes doesn't consider position of a word in a document, but considers only existence of the word, the test documents like "Not good" or "Not bad at all" can be classified wrong.
  - Models where the occurrence and relative position of the words matter (like the state chains in Markov Model), can perform better.
  - Using a list of stop words, the performance of the classification can be improved
- Complexity: Using a Dictionary data structure (Hashing), the complexity of training algorithm can be reduced substantially.

References

- Daniel Jurafsky and James H. Martin *Speech and Language Processing*. Pearson Education, 2000.
- Quan Yuan, Gao Cong and Nadia Thalman *Enhancing Naive Bayes with Various Smoothing Methods for Short Text Classification*. WWW, 2012.
- Dheeru, Dua and Karra Taniskidou, Efi *UCI Machine Learning Repository* `http://archive.ics.uci.edu/ml`