CS 165A Machine Problem 1 Report
Nikita Buzov
05/12/2020

Architecture.
`NaiveBayesClassifier.py` is the main piece of code that implements the Naïve Bayes Classifier. It takes as inputs two arguments: training dataset and testing dataset, both text files containing labeled collection of reviews. As the program begins to execute it runs the function `run_train_test()` that has all the functionality. It has a few major parts. First, all the variables are set, like the smoothing factor alpha, collections of stop words and logical negation words. Second is the import of training dataset. At this stage the function reads the training dataset, stores the labels and reviews in lists, builds the vocabulary set, counts the frequencies of occurrence of unique words for each class and each of the classes. Next, conditional probability table is computed based on Naïve Bayes rule from frequencies of word occurrences, this is the training stage of the classifier. Then, the program runs inference on the training data set computing the likelihoods for each review belonging to each class and computes the accuracy of the predictions. Fifth, import the testing data set storing the labels and reviews in the corresponding lists. Then finally, run inference on the testing dataset and evaluate the performance.
There is also another python program lexicons.py that imports positive and negative lexicons as sets of words from text files. These lexicons are then being used by the Naïve Bayes Classifier to add another two features whether a review contains words with positive sentiment, and with negative sentiment.

Preprocessing.
As each line from the datasets get imported into the program, it's being split into a list of words and a label. Then, if any of the words are in my set of logical negation words, the next two words will get string 'NOT_' concatenated in front. This is done in order to add more features that indicate negative sentiment in the review. Next, the list of words is converted to a set, which gets rid of the duplicate words and ordering of the words. Therefore, this model will be a Binary Naïve Bayes classifier, since we count each word only once per review, irrespective of the frequency of it inside a review. Following this step, the program begins to update the vocabulary set and word counts for every word in the review (CPT values). In order, to speed up the lookup of CPT values, all the word counts and conditional probabilities were stored in dictionaries, and not lists.

Model Building.
The training happens by computing the conditional probabilities for each word given the class. The stop words were treated as 0 word counts. Laplace correction was used to smooth out the 0 probabilities with smoothing factor alpha.

Results.
The accuracy results were 0.927 for training set and 0.882 for the testing set. The training phase took 5 seconds (including the import of the dataset, CPT values

computations and inference). The labeling of the testing dataset (including the import of testing data) took 1 second.

Challenges.
One of issues I quickly ran into is that reading and parsing through the data was very slow due to the data structure type (lists) I used for storage. So then I changed everything to sets and dictionaries were possible, which drastically decreased the runtime.
Next issue was that a simple Naïve Bayes bag-of-words model achieves top accuracy of about 85%, so a few clever tricks had to be implemented to improve the accuracy:

- Stop Words: ignore the words that a very frequent but don't carry any meaning. I discovered that only words 'the' and 'and' increase my accuracy.
- Logical Negation: if the review contains any of these words {'t','not','no','never','dont','didnt','doesnt'}, I assume that it carries negative connotation, so the following two words will be concatenated with 'NOT_' in front to create additional features
- Sentiment Analysis with lexicons: I used public lexicon dataset which has thousands of positive and negative words. I added two additional features 'positive-words' and 'negative-words', so if the review has a word that is in one of those sets of words, the feature adds 1 count.
- Laplace correction: smooth out the 0 probabilities. I discovered that alpha factor of 0.5 yields the highest accuracy and generalization of the model.
- Underflow prevention: I used a sum of logs to compute the class probabilities

Weaknesses.
Since my model only uses unigrams (i.e. single words as features), it loses all the context information. Using n-grams might improve the accuracy. However, when I tried to use bigrams as feature, that didn't improve the accuracy. Perhaps, more adjustment of the parameters is needed. I also tried Porter Stemming, but that didn't help either and only reduced the accuracy. Though when I combined bigram model with stemming, I yielded a result close to my best accuracy.