Name: Nikola Baci
Date: Apr 27th, 2021
Topic: Movie Review Sentiment Classifier - Naïve Bayes Classifier

Report

Part I:
s = "I always like foreign films"

$$P(s|+) = 0.4 \times 0.09 \times\ 0.07 \times 0.29 \times 0.04 \times 0.08 = 2.33856 \times 10^{-6}$$
$$P(s|-) = 0.6 \times 0.16 \times 0.06 \times 0.06 \times 0.15 \times 0.11 = 5.70240 \times 10^{-6}$$
$$P(s|-) > P(s|+)$$

The sentence s is classified as negative by NB classifier.

Part II:

Overview

This homework builds a Naïve Bayes classifier with add one smoothing, used to classify movie reviews as positive or negative. It is composed of two python scripts: preprocessing.py and NB.py.

preprocessing.py is the python script that takes the following command line arguments:

1. the path to the training directory for the first class
2. the path to the training directory for the second class
3. the path to the vocabulary file
4. the path to the test directory for the first class
5. the path to the test directory for the second class

Here we are assuming that both the training and the testing data is already labeled and is separated in the appropriate file directories, which then are passed to the preprocessing.py script. The script does the following:

1. It reads the vocabulary (third argument) into a hash set
2. Preprocess' the test files
    a. First reads the test review file, lowercases it, and separates the punctuations
    b. Creates a file named *test_features_file.txt* of the following format:
        i. prints the true label alongside with the file name (used for observation)
        ii. prints the review file "as-is" after step 2.a
        iii. does this for all the test files in both classes
3. It builds two dictionaries for each class that will be used to calculate the probabilities for each feature.
    a. Reads each review, lowercases it and the separates punctuation
    b. Then puts in the dictionary and increments the value count if the given word is in the vocabulary from step 1.

c. For each word in the dictionary, calculates its probability with add one smoothing using the following formula

$$p(w|C_i) = \frac{count(w_{C_i}) + 1}{|C_i| + |V|}$$

where $p(w|C_i)$ is the probability of the word w of the given class C_i,
and $count(w_{C_i})$ is the count of the word w in the set of the given class C_i,
and $|C_i| + |V|$ is the total count of the words in the given class C_i plus the total count of the words in the given vocabulary V.

d. Outputs each log-probability in the *train_features_file.txt.* The format of this output file is the following:

i. First prints the prior probability of the first class
ii. Second prints the prior probability of the first class
iii. Third prins the column names
iv. The rest is a list of all the words taken from the dictionaries in step 2.b and the calculated log-probabilities for each of the two classes
v. Does this for both training files in both classes

This concludes the job of preprocessing.py.

NB.py is the script that makes the predictions for each movie review in the *test_features_file.txt* using the calculations saved in the *test_featuers_file.txt.*
NB.py is the python script that takes the following command line arguments:

1. the path to the *train_features_file.txt*
2. the path to the *test_features_file.txt*
3. the path to the file where the prediction will be written

The script does the following:

1. Creates two dictionaries, one for each class, to save the word and the corresponding log-probability for that class
2. For each review in the *test_featuers_file.txt*:
    a. Saves the label
    b. Uses the dictionaries form step 1 to calculate the final log-probabilities for each movie review using the following formula:

$$P(T|C_i) = log_2 P(C_i) + \sum_{w \in T} log_2 P(w|C_i)$$

where $P(T|C_i)$ is the log-probability that test T is in class C_i,
and $log_2 P(C_i)$ is the prior log-probability of class C_i,
and $\sum_{w \in T} log_2 P(w|C_i)$ is the sum of log-probabilities of all the words in the given movie review with respect to class C_i

    c. Predicts the class that has the highest probability and keeps track of the correct predictions
    d. Prints each prediction in the output file provided by the user in the following format:
        i. Prints the columns name: the true label, the prediction, the log-probability for each class
        ii. The rest are the values for each movie review in the above format

iii. The last line prints the accuracy of the model

This concludes the work of NB.py script.

Point c) in homework file:

The NB.py calculated that $P(s|comdey) =$ -13.73695 and $P(s|action) = -12.50974$
The classifier chose action as the label of the sentence {fast, couple, shoot, fly}

Point d) in homework file:

The vanilla-form Naïve Bayes classifier performed with an accuracy of **81.524%**. Afterwards, I experimented with the following features:

1. Created my own vocabulary during the runtime
2. Decided to remove the common stop words
3. Changed the way I separated punctuation

All these new features did not improve my accuracy score, the classifier performed a bit more poorly than the original version dropping down to 81%.
The feature that resulted with an increase of accuracy to **83.092%** is the Binary Naïve Bayes feature, where I count only the unique words of a review. However, I had to apply the binary feature to both the test and the train reviews in order for the accuracy to go up. This last version is the one found in the submitted zip file .

Observation
In the prediction file, the last two lines show how many reviews the classifier has labeled incorrectly for each class. Here are the numbers for easy reference:

accuracy = 83.092%
missed positive = 2814
missed negative = 1413

As we can see there are almost exactly double the number of misclassified positive review than negative reviews. So, I started to investigate why this was happening by reading the misclassified reviews and checking the log-probabilities in the *train_features_file.txt.* From my observation I concluded that negative words like “worst” and “bad” have much higher log-probabilities in the negative class than in the positive class. On the other hand, positive words like “funny” and “happy” do not lead by much in the positive classes. The table below shows some of these examples:

Word	+ prob	-prob
happy	-11.8360852	-12.4980388
happy-ending	-19.7931873	-19.7834411
funny	-10.2491888	-10.1332869

worst	-12.9730083	-9.79475642
bad	-10.2696253	-8.69630981

When words are independent of each other as in the case of out Naïve Bayes Classifier, their meaning is taking in isolation of the other words. So, in positive sentiment review, when the user writes sentences like “never seen a movie so funny”, the word never will weigh more than the word funny making the probability drift towards the negative class. However, we can clearly see that the context of the word never in this case is to emphasize the magnitude of the word funny, thus making the sentence a positive sentiment. It appears that many positive reviews contain words whose meaning can be generally related to negative sentiment, hance bumping up the number of misclassified positive reviews.

The examples I have illustrated above are focused on the negative class, however the same can apply on the other direction. A handful of the negative reviews labeled incorrectly include words that independently have a positive connotation which outweigh the negative words, thus confusing the classifier.