Fernando Mendoza
9637984

MP1 Report
Used python 3

Parsing and Stop Words

The first thing that my project does is parse all reviews and remove all stop words. This is done in function **extractUniqueWordsFromReviews(reviewsArray).** This function takes in an array of strings/reviews. For every review, I split the string into a list of tokens with " " (space) being the delimiter. Then for each word in the review I pass in word to **extractWordFromString(string)**. The function takes in a string and eliminates any characters that are not in an alphabet that I created. The alphabet consists of a-z and 0-9. **extractWordFromString(string)** then returns the string that is left. Back in **extractUniqueWordsFromReviews(reviewsArray)** I look to see if the string that I get is a stop word or an empty string. If so I disregard it, otherwise I append it to a new review list. **extractUniqueWordsFromReviews(reviewsArray)** then returns an array of reviews with each review being an array of words.

Extracting Information from Training Data

extractUniqueWordsFromReviews(reviewsArray) not only returns an array of reviews with each review being an array of words. The function also returns 4 dictionaries and the total number of words in all reviews. **dictIndex** contains all unique words as keys and an index assigned to the word as value. This helps me later on so that I can create a matrix and know what column each word belongs to. **countInAllReviews** contains all unique words as the keys and the number of times that word occurs in all the reviews as value. **wordCountInReview** contains the (review index) + (word) as the key and the number of times that word appears in that review as value. **countOfReviewsWithWord** contains all unique words as keys and the number of reviews that contain the word. I run this function on the training data and test data.

Multivariate Naïve Bayes Classification

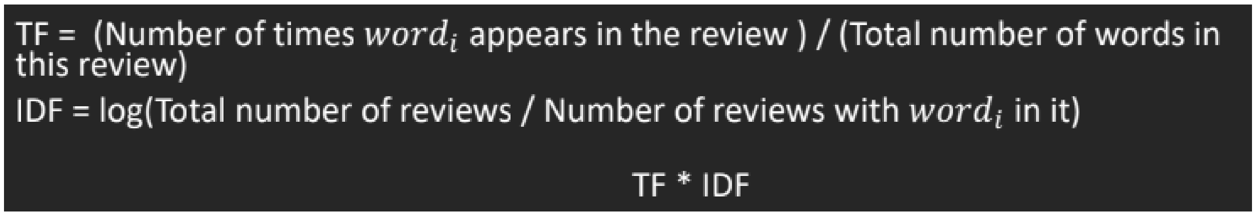
After running **extractUniqueWordsFromReviews(reviewsArray)** on the training data I can now use the information being returned to classify the test data. To do this, I call **MultivariateBayesClassification(reviews, countInAllPos, countInAllNeg,numberOfPositveWords, numberOfNegativeWords, vocabLength)**
In this case, reviews are the reviews I am classifying, **countInAllPos**, is the equivalent of **countInAllReviews** for the positive training data, and **countInAllNeg** is the equivalent of **countInAllReviews** for the negative training data. In the function I iterate through all reviews and for each word I use the formula below to find the P(word | Positive) and P(word | Negative).

P(Word | Class) = (n_k + a) / (n_c + a*n_v)
n_k = number of times word_i appears in the class
n_c = total number of words in the class
n_v = the total number of words in all classes
a = 0.001

Once I get the probability of the word, I take log of the problability and accumulate it to a variable until I am done with that review. Then I compare the P(review|Negative) and P(review|Positive) and classify that review as positive or negative depending on what ever value is bigger. I call the function on the negative test data and positive test data. The function then returns the percentage of reviews that were classified as positive and the percentage of reviews that were classified as negative.

Extracting BOW and TFIDF Features of Training Data

To extract the BOW features and TFIDF features of the positive and negative training data, I call **ExtractBOWTFIDFFeatures(reviews, wordIndex, wordsInReview, reviewsWithWord).** In the function, I create a matrix for BOW and for TFIDF. I iterate through all the reviews and all the words in the reviews. I then use **wordIndex** to find the column of the word we are looking at. Then I use **wordsInReview** to populate the matrix with the frequency of each word in each review. Then I use **wordsInReview** and **reviewsWithWord** to calculate the TFIDF using the formula below and populate the matrix for TFIDF.


TF = (Number of times $word_i$ appears in the review) / (Total number of words in this review)
IDF = log(Total number of reviews / Number of reviews with $word_i$ in it)

TF * IDF

Once I have iterated through all the reviews, I look at both of the matrices and for each column I find the STD and Mean. Then the function returns two arrays one with the BOW features and one with TFIDF features. Both arrays have dictionaries as the items and the dictionaries have "mean" and "std" as keys.

Classifying Test Data Through BOW Features and TFIDF Features

To classify the test data I call **ClassifyReviewsWithBOWFeatures(reviews, negativeFeatures, positiveFeatures, negativeIndex, positiveIndex)** and **ClassifyReviewsWithTFIDFFeatures(reviews, negativeFeatures, positiveFeatures, negativeIndex,positiveIndex, negReviewsWithWord, posReviewsWithWord).** Both the functions as very similar. They both iterate through all the reviews and use the formula below to find P(word | Positive) and P(word | Negative).

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In the equation x is the count of the word in that review for the BOW classification. For the TFIDF classification x is the TFIDF of the word. I then accumulate the probability for each positive and negative prediction and compare the two at the end of each review. Both functions return the percentage of positive review classifications and the percentage of negative review classifications.

Multivariate BOW Classifications

Positive Test Data
Positive Percentage: 58.5102680125304
Negative Percentage: 41.4897319874695

Negative Test Data
Positive Percentage: 24.551856594110117
Negative Percentage: 75.4481434058899

Gaussian BOW

Positive Test Data
Positive Percentage: 71.806474068917
Negative Percentage: 28.08910546467108

Negative Test Data
Positive Percentage: 47.21510883482715
Negative Percentage: 52.6248399487836

Gaussian TFIDF

Positive Test Data
Positive Percentage: 50.92238078663418
Negative Percentage: 48.10302819352593

Negative Test Data
Positive Percentage: 21.92701664532650
Negative Percentage: 76.31241997439182