Kevin Chan
CS165A W17

MP1 Report

Architecture

The code base is divided into two Java classes, NaiveBayesClassifier.java and helperFunctions.java. The "main" function is in NaiveBayesClassifier.java and within it, the classifier is trained using the training data, and tested twice, first with the testing data, and again with the training data. NaiveBayesClassifier.java simply extends helperFunctions.java, as helperFunctions.java contains a mini-library of functions that NaiveBayesClassifier.java calls. The overall architecture used to implement the functionality of a naïve Bayes classifier uses 3 HashMap objects, each used to contain the frequency of words used in:

1) Positive reviews
2) Negative reviews
3) (All) Both positive and negative reviews

When training the classifier, ArrayList<String> objects are used to contain the words in a review, which are then filtered into the respective HashMap frequency table depending on the sentiment of the review.

When using the classifier on testing data, each line of the document represents a single movie review, and the sentiment of each review is chosen as a combined product of the probability of each class (based on training data) and the independent conditional probability of each word given a class (prior).

The accuracy of the program is determined by creating a list of all the actual sentiments of each review in the tested data set, and comparing that list to a list of theoretical sentiments that the program creates. The accuracy is then calculated with :

accuracy = matching sentiments / # total reviews in tested set

Preprocessing

In an attempt to maximize the accuracy of the classifier and normalized, each word of a movie review document is cleaned. The algorithm:

- takes away any special characters such as apostrophes and such used in typos
- makes all words lower cases to make indifference of words that differ in capitalization
- remove trailing s's
- remove trailing ed's
- remove trailing ing's
- skip any stop words such as "the" , "a" , "such", "because"….etc
- any common words (included in stopwords) that both positive and negative reviews may share, such as "movie", "film", etc..

Model Building

The Naïve Bayes Classifier is trained by creating 3 frequency tables for unique, one corresponding to positive reviews of the training set, one to negative reviews of the training set, and one for all reviews of the training set.

Results

The results on the provided datasets are:

- 2 seconds (training)
- 5 seconds (labeling)
- 0.9242 (training)
- 0.848 (testing)

The 10 most important features for:

- Positive class
    1) Picture
    2) Well
    3) Story
    4) Viewer
    5) Hour
    6) Dark
    7) Catch
    8) Nearly
    9) Silly
    10) Family
- Negative class
    1) Sort
    2) Given
    3) Guy
    4) Not
    5) Watch
    6) Out
    7) Rival
    8) Flat
    9) Maybe
    10) Show

Challenges

- Cleanliness
    - Due to the amount of traversals needed for this program, keeping the codebase readable was a problematic
        - To solve this, I created helperFunctions.java and defined the hefty functions there and called them in NaiveBayesClassifier.java
- Improving Accuracy
    - After removing stop words and common based words, it is hard to spot patterns in the words that will create more distinction between the positive and negative reviews
        - To target this, I created a function that gave me the most commonly used words among each class, and told the algorithm to skip those words if the words did not seem to provide contextual differentiation
        -

Weaknesses

The algorithm does not do its best job to convert words to its root word. There are unaccounted cases of this that are too unique for any single sub-function to detect and transform.