Kyle L Jackson
CS 584
HW1

HW1: Amazon Review Classification

Team Name: econguy19

Rank & Accuracy (as of 9/19/16): 11, 0.76

Approach

My approach is broken up into two parts: preprocessing and classification using the k-nearest neighbors algorithm.

Preprocessing

For the preprocessing step, existing libraries were used for most of the work. To start, the train and test files were loaded using Pandas. I then dropped the training samples with missing text to reduce noise. Pandas' read_table function was used for reading in the data since it had a parameter for reading empty lines.

Once the train and test data were read in, I began processing the text. The preprocessing() function I defined iterates through each line in a file and does the following: change all instances of a digit, from one through five, next to the word "star" or "stars" to a word and combine the two words into a one word string (e.g. "1 star" changed to "onestar"); tokenize sentences; drop all digits, punctuation, whitespace, and symbols; make all characters lower case; remove stop words; stem the words; join the tokenized text.

The next step was to turn the text in the train and test sets, which at this point are represented by a vector with each line being a string of text, into a term frequency-inverse document frequency (tf-idf) matrix. This was implemented with scikit_learn's TfidfVectorizer. The training text's vocabulary is learned and an inverse document frequency weight is assigned to each word in the document, followed by a conversion to a sparse term-document matrix. The test text is similarly transformed using the vocabulary learned from the training text. In addition, these data are normalized.

Classification

First, the similarity score is calculated between one tf-idf vector in the test data against the entire training tf-idf matrix using the cosine similarity. The entire classification method is encapsulated in a for-loop that iterates through each line in the test file. Since the tf-idf matrices were row-normalized during the transformation into a tf-idf matrix, the cosine similarity is calculated by the dot product of a test sample's tf-idf vector and all the train samples. This is implemented with scikit-learn's linear_kernel, which returns an array of similarity scores. The array is then sorted using argsort() and indexing is used to get the k highest values' indices (k is specified as a parameter for the function) from the original train.data file, which are the most similar training vectors to the one test vector. Next, a list is created of the training labels based on the indices just found. Then a new list is created that contains the similarity scores that have the same signs as the labels. For example, a similarity score of *0.6* that corresponds with the label *-1* will now become *-0.6*.

The list of similarities with varying positive and negative signs is then summed, with higher similarity values having more weight than less similar values. If the sum is negative, that specific test sample is classified as a negative review and a *-1* is appended to a list containing all the predicted labels for the test set. If the sum is positive, a *+1* is appended to the same list. Once the for-loop has finished iterating through all the test samples, the list is returned as a Pandas DataFrame and written to a text file for submission.

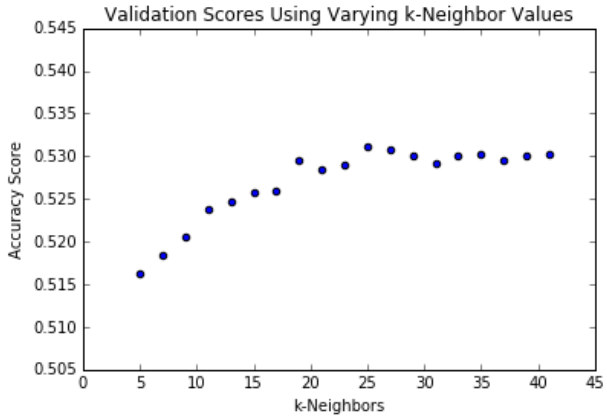
Methodology of Choosing the Approach and Associated Parameters

There were a number of issues that arose when reading in the data. In the training data, there were nine samples with labels, but missing text. Further investigation showed that there were five samples with "-1" labels and four samples with "+1" labels. Since the labels appeared arbitrary, I dropped these samples. In the test dataset, there were two issues: the missing lines were not being read and the special characters in line 6146 were leading to errors in reading the data. I identified these issues when the dimensions of the Pandas DataFrame in the test set were less than the number of lines in the test .data file.

The way I went about preprocessing the data wasn't particularly efficient, though I created a preprocessing function to call on the test and train raw data in order to improve code readability. Though I used to write code in a scripting format, containing the preprocessing steps that would have to be repeated on multiple inputs within a defined function provided multiple benefits that I had not realized before. In the future, I would have added spelling correction to my preprocessing step and would have used the number of words in each line as an additional feature.

On the topic of using a tf-idf representation: I decided to go with this method because it increases the value given to each unique word proportionally to the number of times it appears in the document, but reduces the value of the word as the frequency of the word in the corpus, or the collection of documents, increases. I chose this method over the word frequency representation, which is noisy and has been shown to provide poorer results than a tf-idf representation.

My implementation of the k-nearest neighbors algorithm was relatively simple. At first I used the sums of the labels to find a majority vote, but then decided to sum the similarity values since higher similarity values should be given more weight. The improvement from taking the sum of the labels compared to taking the sum of the similarity values was about a 0.03 gain in accuracy.



While my implementation of the k-nearest neighbors algorithm had a few parameters which were mostly file inputs and whether the user wants weights or not, the key driver was the selection of k, or the number of nearest neighbors used for classifying the test samples. To find the optimal number of neighbors, I used cross validation (contained within validation_workbook.py). I used a randomized 80/20% split on the training data to find the optimal k. I found that the accuracy score's maximum plateaued somewhere around 20 neighbors (see graph above) with a peak at 25 neighbors. My final submission contains the k parameter being set to 25.

References

A quick note about my references for this assignment: the documentation for nltk and scikit-learn were excellent and were my most used references. I also found ogrisel's (a contributor to scikit-learn) explanation of document similarities between tf-idf matrices to be beneficial.