HW1: Amazon Review Classification

Prakhar Dogra

Email: pdogra@gmu.edu

G Number: G01009586

Team Name: pdogra

Introduction

Given Amazon Review Classification problem has been solved using K Nearest Neighbor Algorithm. The solution includes tokenizing the given text data set, removing stop words and using TFIDF Vectorizer to create the feature vectors. Euclidean distance has been used as the similarity measure for the above-mentioned feature vectors. Using the calculated similarity measure, K Nearest Neighbor Algorithm was applied to classify the reviews as of either positive sentiment (+1) or negative sentiment (-1).

Rank and Accuracy

As of 02/13/2017 7:00 p.m., public accuracy score is 0.75 and rank is 5.

Team Name

pdogra

Approach

Following Python files were created in the order of approach taken:

create_bag_of_words.py (CREATING BAG OF WORDS):

At first the training data set ("trainhw1.txt") was read line by line and a bag of words was created. This bag of word has neglected words of length 2 or smaller and frequency (number of times the document occurred in the whole data set) 3 or less.

test.py (TESTING ALGORITHM FOR DIFFERENT K):

This Python file was created to test the accuracy of the KNN for different parameters like different values of K or whether to lemmatize/stem the words or not. Training data set was tokenized and saved into a corpus. TFIDF vectorizer was used to create feature vectors of that corpus. A random set of examples were chosen as test sets. For most iterations, 30% of the data set was chosen as the test set and rest of the data set (70%) as the training data set. At the end, the Nearest Neighbor Algorithm was applied for different values of K. Euclidean distance was used as the similarity measure. For each training data point and test data point, distances were recorded in a list along with the sentiment. This list was sorted in increasing order of said distance. And majority sentiment of first K feature vectors (basically reviews) was considered as the sentiment of the respective test data point.

The program was tested for both, Unigram model as well as Bigram model.

main.py (IMPLEMENTING ALGORITHM FOR OPTIMAL K):

This Python file was created to write the results of the given test data ("testdatahw1.txt") into a .txt file for the optimal value of K selected during testing. Most of the code matches that of test.py with minor changes. Firstly, the corpus and bag of words created contained both training and test sets. Secondly, the program was run for a single value of K (which gave best accuracy from several iterations of test.py).

Methodology

BAG OF WORDS:

Creating bag of words and writing it to a file was necessary to reduce the initial preprocessing time. Moreover, only those words were taken into the corpus that were present in the bag of words in order to reduce the size of feature vectors created using TFIDF Vectorizer.

STEMMING:

Before a review was added to the corpus for vectorization, it was passed through a stemming method so that all the words in that review were reduced to their shorter form. For example, 'caresses' is reduced to 'caress', 'ponies' is reduced to 'poni', 'cats' is reduced to 'cat', etc. Stemming usually refers to chopping off the ends of words in order to generate a common word from different words that actually mean the same thing most of the time, like the removal of derivational affixes. Maximum accuracy achieved when using stemming was 75%.

LEMMATIZATION:

Lemmatization usually refers to the use of a vocabulary and morphological analysis of words, basically to remove inflectional endings only and to return the base or dictionary form of a word, also known as the lemma. For example, words like 'is', 'am', 'are' are reduced to 'be'. During testing and implementation, lemmatization was also used but it gave slightly lower accuracy for same values of K. Maximum accuracy achieved when using lemmatization was 74%.

TF-IDF:

Sentiment analysis in this problem is reduced to knowing which words occur most frequently in positive reviews and which ones occur most frequently in negative reviews. TF-IDF involves two steps – one is the computation of a normalized Term Frequency that is indicative of how often a word appears in a document. TF is calculated as follows:

TF(t) = (number of times term t appears in a document) / (total number of terms in the document)

Inverse document frequency measures how important a term is. To avoid one pitfall of IDF calculations which is measuring the influence of stop words like "is", "this", "a", "the", "and", etc., stop words have been filtered out from the corpus in the document pre-processing stage. IDF is calculated as follows:

IDF (t) = $\log_{10}$(total number of documents) / (number of documents with term t in them)

TF-IDF vectorization of train data would give 18506 vectors containing TF-IDF weights for each unique word in the training corpus. The immediately apparent drawback of this method is that the dimensions of the vectors increase with the size of the vocabulary. With 18506 reviews in the train file and an equal number in the test file, the total size of the vocabulary from both data sets was 12178. When TF-IDF Vectorizer is applied on given data set we get a list of feature vectors where each feature vector is of length 7297 (reduced due to stemming of words).

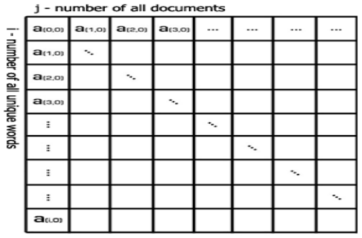

Figure 1: Sample Weight matrix (list of feature vectors)

During testing, Unigram and Bigram models were considered. Bigram models took too much time but gave slightly more accuracy. But during the program run of main.py, we got Memory error even with a vocabulary of 3000 when trying to use the Bigram model.

K NEAREST NEIGHBOR ALGORITHM:

The variant of the Nearest Neighbor algorithm implemented is called k-Nearest Neighbor (KNN). The KNN assigns a class label to an unseen data instance based on the class labels of the nearest 'K' seen instances. This is called majority voting. The similarity metric used is Euclidean distance. The computational complexity is important while setting a while for 'K'. A general rule of thumb seems to be that we can start with K = √n for n samples, which would be K = 136 for our 18506 samples. But I started with K = 1. As the value of K increased the accuracy increased up to K ~130-145 and then it declined upon further increasing the value of K.

KNN algorithm was also tested with Cosine Similarity measure as the similarity measure but achieved accuracy as high as 51%. And as the values were increased from K = 1, the accuracy decreased. This accuracy was approximately 20% lesser than when Euclidean distance was used as the similarity measure. Hence this similarity wasn't used in the latest implementation.

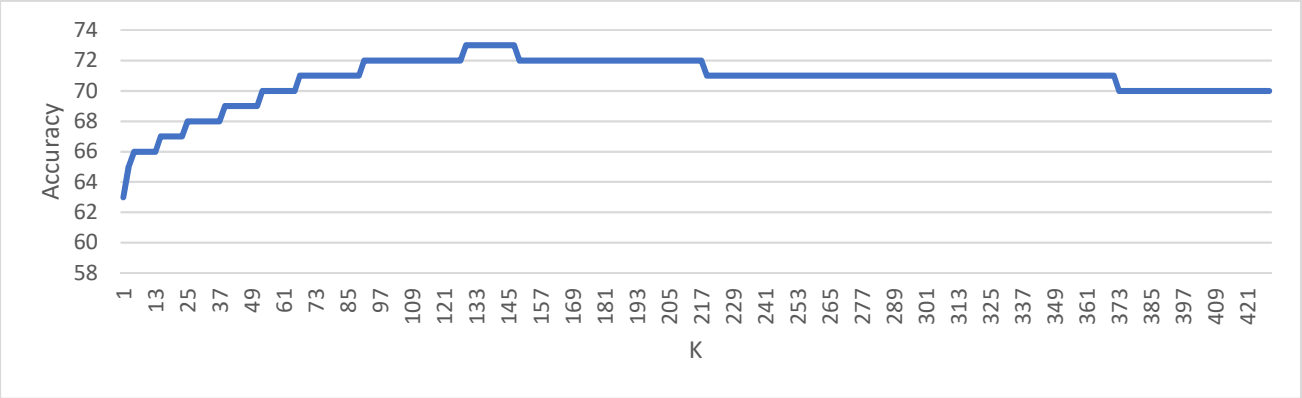

Figure: Graph of Accuracy vs K during testing

References

"Scoring, Term Weighting And The Vector Space Model". Nlp.stanford.edu. N.p., 2017. Web. 13 Feb. 2017. < http://nlp.stanford.edu/IR-book/html/htmledition/scoring-term-weighting-and-the-vector-space-model-1.html>

Duran, Fletcher. "Dealing with Ties, Weights and Voting in KNN." StackOverflow. N.p., Dec. 2010fle. Web. 13 Feb. 2017. < http://stats.stackexchange.com/questions/45580/dealing-with-ties-weights-and-voting-in-knn?rq=1>

Trstenjak, Bruno, Sasa Mikac, and Dzenana Donko. "KNN With TF-IDF Based Framework For Text Categorization". N.p., 2017. Print

Python?, What. "What Is The Best Stemming Method In Python?". Stackoverflow.com. N.p., 2017. Web. 13 Feb. 2017. < http://stackoverflow.com/questions/24647400/what-is-the-best-stemming-method-in-python>

"Stemming And Lemmatization". Nlp.stanford.edu. N.p., 2017. Web. 13 Feb. 2017. < http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>