CS 584 - Theory and Application of Data Mining Spring 2019
HW 1 – True and Deceptive Hotel Reviews

Name : Akshaya Damodaran
G No. G01129364
Registered Name on Miner: trytry77
Rank : 39
Accuracy Score : 74%

Feb 14, 2019

Problem

Use of KNN Classifier to classify an opinion dataset consisting of truthful and deceptive hotel reviews into their appropriate classes i.e. 'Truthful' indicated by a '0' and 'Deceptive' indicated by a '1'.

Initial Steps

Firstly all text files are fetched from the training dataset path, their labels are extracted (Truthful or Deceptive), dataframes of the labels and reviews are created and finally both the dataframes are merged.

Pre-processing and Cleaning

Before text and feature extraction, the training and test data is cleaned in order to obtain better features. The pre-processing done on the training and test data for this problem are as follows –

- Transforming all upper-case characters in the reviews to lower-case.
- All punctuation is removed since it does not provide any information while processing text data.
- Stop words (or commonly occurring words) were removed from the text data. For this purpose, stopwords from the predefined ntlk library were used.
- Previously, just commonly occurring words in a general sense were removed. Next, the 10 most commonly occurring words particular to our text data were also removed sice the presence of these words will not be of any use in classification of the reviews.
- Likewise, 10 least commonly occurring words were also removed because they are so rare, the association between them and other words is dominated by noise.
- Then, Lemmatization is used on words to capture the canonical part i.e. the roots of the words.

Feature Extraction using TF-IDF

For this problem, feature extraction is done by using the Term frequency Inverse Document frequency (TF-IDF). The problem with simply counting how many times each word appears in every review is that this method does not consider the relative importance of words in the texts. A word that appears in almost every text would not likely bring useful information for analysis. On the contrary, rare words may have a lot more meaning.
The TF-IDF metric solves this problem:

- TF computes the classic number of times the word appears in the text
- IDF computes the relative importance of this word which depends on how many texts the word can be found.

So, the training and test data are vectorized using the TfIdfVecorizer.

Program Flow Outline

- The training dataset (1441 text files) is first loaded using os.walk and put into a dataframe.
- A dataframe of 'labels' is created for each of the text files in each sub-directory of the training dataset(Deceptive or Truthful).
- Dataframes of the labels and training reviews are merged and pre-processing is then done on the text reviews in the dataframe.
- Similarly, the test data set is also put into a dataframe and pre-processing is done on the reviews in the test dataset.
- Now, the training data is split into 2 parts using train_test_split. These 2 parts are the training data part and validation part stored as X_train and X_test. The labels for them are divided into y_train and y_test respectively.
- Next, we use k-fold validation by running our knn algorithm (function **kNearestNeighbor**) with the train and test vectors obtained from the splitting in order to find the accuracy of classification by our knn algorithm.
- We can then see the accuracy for different values of k i.e.(more or lesser number of neighbors). Once k is tuned to get the best possible accuracy, we proceed to run the knn algorithm (function **kNearestNeighbor**) again but this time, with the parameters being the train vectors and the vectorized form of the actual test data(160 text files).
- *kNearestNeighbor function* **:** The **kNearestNeighbor** function calculates Cosine distances for each review in the test data with respect to every review in training data. These distances along with their labels ('deceptive' or 'truthful') for each test data are appended into a list (**k_truth_values**) and sorted in ascending order of distance.
- Then, the most common label among the first 'k' labels from the **k_truth_values** is appended into the predictions list for each of the 160 reviews and this list is returned by the **kNearestNeighbor** function**.**
- Finally, for every label in the predictions list, a '1' is written into the output file if the label was 'deceptive' else, a '0' is written if the label was 'truthful'.