CS 584: THEORY AND APPLICATIONS OF DATA MINING

HW1: TEXT REVIEW USING KNN CLASSIFIER

NAME: SRIPATH CHERUKURI
MINER2 USER-ID: MAVERICK
ACCURACY SCORE: 0.73
RANK: 365
G-NO: G01395231

Problem Statement:

To implement K-Nearest-Neighbour Classifier to predict the sentiment for 18,000 reviews for various products provided in the test file (test_file.csv)

Below are the five major steps followed for implementing the solution for above problem:

- Importing required packages
- Text cleaning
- Text pre-processing
- Dimensionality reduction
- Implementing own KNN algorithm

1. Importing Required Packages:

There are certain packages that we need to import into our program for implementation of this problem. There are four major packages as mentioned below:

- **Nltk**: This is used to handle natural language
- **Pandas**: This package helps us with handling and loading data
- **Sklearn**: This is a library that contains various methods to perform analysis of our model
- **NumPy**: This package helps us while working with large arrays

All the information on other packages used is clearly explained in the code implementation.

2. Text Cleaning:

There are a lot of unwanted symbols and punctuation in the text data. This creates a lot of ambiguity if directly passed to the model without cleaning the data. There are various steps are involved in the cleaning of the data and performing these steps improves the overall accuracy of the model. The steps that were followed in the implementation of this program are mentioned below:

- **Converting the data into lowercase**: As the machine interprets uppercase and lowercase letters differently to avoid ambiguity, generally we prefer to convert all the text data into lowercase letters.

- **Removing punctuations**: 32 main punctuations are to be removed from the text before sending it to the model. In the implementation of this code using the python re module, those punctuations were eliminated from the text data.

- **Removing the stop words**: Stop words are those that do not provide any important information in the text data, so these words and propositions are removed using the stop words module from the nltk package.

- **Lemmatization**: This is one of the most important packages used in the implementation of this program. This groups together various forms of a particular word.

3. Text Pre-Processing:

After cleaning the text data, we cannot send all the data into the model for prediction. To analyse our model. A train test split is performed on the cleaned data using a certain split ratio. In the implementation of this assignment, I used an 80% and 20% split ratio where 80% is the training data and the other 20% is the test data. This helps us to check the accuracy of our model once the predictions are made.

As the data is a raw collection of text, we need to process it further. So, the TF-IDF vectorizer is used here to convert this raw data into a matrix of features. Now the text data is in numerical form and can be passed to the model. Working with text data of this scale can produce huge vectors after using the TF-IDF vectorizer. So, need to reduce the dimensionality of the vector for faster execution times.

4. Dimensionality Reduction:

While working with data of this scale, we may need dimensionality reduction techniques to speed up the process and utilize resources in the best possible manner. The technique that is used in the program is truncated singular value decomposition.

Unlike principal component analysis truncated singular value decomposition works efficiently with sparse matrices. This is also known as latent semantic analysis. The desired output of this dimensionality reduction was set to the number five in the implementation.

5. Implementing Own KNN Algorithm:

The algorithm followed while implementing code for this problem is simple. As known the KNN takes a point and calculates the distances between that point and all its neighbours to classify the label for the given point. So, we must loop through every point in train data for each point in the test data and calculate the distance metric between the two given points in the dataset. The same is implemented here while building the KNN here using for loops. The distance metric used here is the Euclidean distance. This gives us a straight-line distance between two points.

After calculating the distances, we sort the distances in ascending order and pick the first 'k' values from the sorted list and check for the most common label data occurring which becomes the prediction for a given point. We collect all these predictions and then finally return them as an output.

6. Parameters Considered:

There are various parameters that are considered while implementing this code. They are mentioned below:

- **K-value**: The number of nearest neighbours that we need to look for is a variable here. Generally, this is considered an odd number. Here by trail and error there was a certain pattern observed. The accuracy curve goes up until a certain value of 'k' and falls after that. In the implementation of this code a 'k' value of nine was found best suitable.

- **Random state**: This parameter controls the shuffling of the data while splitting the data into train and test. If this parameter is set to zero, we get multiple train and test groups for multiple executions. So having this fixed keeps the shuffling of data consistent and controlled. A random number 42 is used in our implementation.

- **Train-test split ratio**: We considered and 80/20 split in the implementation of our code. 80% is train data and other 20% is test data. This gives the best results while training our model as we provide more train data for the model to train on which produces better accuracy.

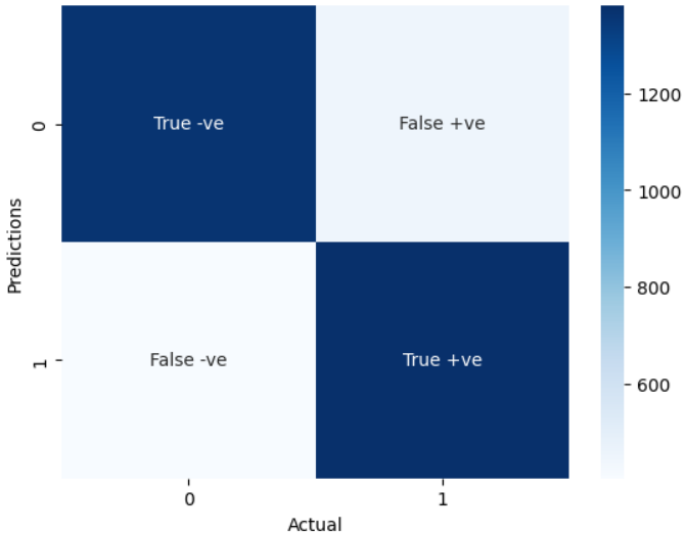
- **Dimensionality reduction**: While performing the dimensionality reduction we used number of features as five in the code so that the resulting dimensionality size could be down sampled to features of five.

Final parameters considered: k-value = 9, Random State= 42, Train-test split = 80/20, Number of features in dimensionality reduction = 5

7. Confusion Matrix:

A confusion matrix is an error matrix that is used to evaluate the performance of the given model. It gives values of true positives, true negatives, false positives, false negatives. The confusion matrix and heatmap for the KNN developed in this implementation is as follows:

	Actual Positive	**Actual Negative**
Predicted Positive	1381	404
Predicted Negative	450	1365



8. Classification Report:

The classification report provides all the metrics evaluating the performance of the model such as F1-score, accuracy, recall value. The classification report for the KNN model implemented in the code is as below:

	precision	recall	f1-score	support
-1	0.78	0.76	0.77	1815
1	0.76	0.78	0.77	1785
accuracy			0.77	3600
macro avg	0.77	0.77	0.77	3600
weighted avg	0.77	0.77	0.77	3600