Procedure and Results

- **Members (GR-G, 65)**
  - ➔ **Name:** Surajit Kundu, Roll No: 21MM91R09
  - ➔ **Name:** Ankur Kumar Jaiswal, Roll No: 21MM62R08

❖ Introduction

We have implemented a K-NN classifier for doing sentiment analysis on movie reviews. The input data are classified with various distance matrices. Then we compare the performance of each distance similarity with the other distance matrix.

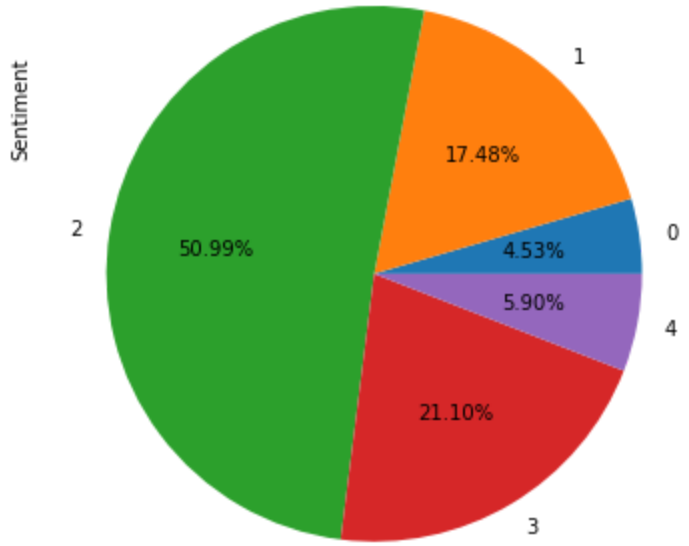
We have downloaded the dataset from Sentiment analysis on movie reviews: https://www.kaggle.com/c/sentimentanalysis-on-movie-reviews.

We have used the Python programming language. And the following libraries are used like NumPy, Pandas, math, nltk, scipy, etc.

The classification is performed based on three different distance metrics/similarities, following Euclidean distance, Manhattan distance, and Cosine similarity.

Read the dataset

To read the Sentiment Analysis dataset on movie reviews, we have used the **read_csv()** function from the panda's library, we read the TSV file into the data frame format. We have chosen "Phrases" and "Sentiment" columns for preprocessing the training data. Here, the "Sentiment" column is our target value, which has five distinct classes (negative, somewhat negative, neutral, somewhat positive, positive).



In the sentiment column, out of 5 distinct classes, we have 4.53 % of zeroes, 17.48 % of ones, 50.99 % of two's, 21.10 % of three's and 5.9 % of fours.

Data Preprocessing

As the input data is in the text form, we need to preprocess the data to eliminate the unnecessary records. Then the features input is converted to numeric values. Few mandatory preprocessing are: converting to lowercase, removing punctuation, removing numbers, removing stop words, and lemmatization/stemming. In our problem statement, it seems like the basic preprocessing steps will be sufficient.

- ☐ The phrase column has words in both uppercase and lowercase. So, we have converted the words to **lowercase** for the sake of simplicity in preprocessing.
- ☐ **Stop words** are removed from the text before training the machine learning models since stop words occur in abundance, hence

providing little to no unique information that can be used for classification or clustering.

- ☐ We have also **removed numbers** from the phrase column as it will be of no use in predicting the outcomes of the test instances.
- ☐ We also **removed the special characters** like hyphens, apostrophes, commas from the same column.
- ☐ After cleansing the training dataset, there are some rows that are empty in the phrase column. So, we have removed these empty rows from the dataset. Now, Our data is ready to be trained.

We have a total of 156060 records in the input data(training). After preprocessing and cleansing, we are left with 154885 records. It actually means that 1175 unnecessary records have been removed from the training dataset. In the sentiment column, we have 5 distinct classes(**0** - 7069, 1 - 27279, **2** - 79575, **3** - 32929, **4** - 9208) before preprocessing.

Splitting the dataset

For splitting the training data, we have to import the **train_test_split()** function from the scikit learn library. With the help of this function, we have split the training data into an **80:20** ratio keeping the random state 42. It means 80 % is used for training the model and 20 % for testing the model. After data preprocessing, we have a total of 154885 records. After the splitting, we have 123908 records for the training purpose and 30977 for testing the model.

TF-IDF Vectorizer

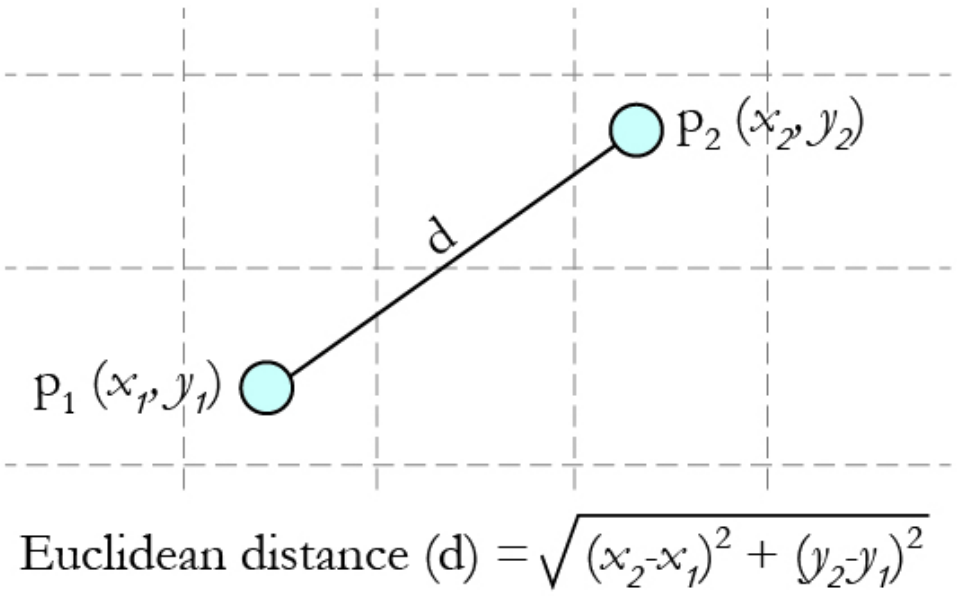
Now, we have imported the function **TfidfVectorizer()** from the sklearn library. It will convert a collection of raw documents to a matrix of TF-IDF features. The **TfidfVectorizer** will tokenize documents, learn the vocabulary

and inverse document frequency weightings, and allow you to encode new documents. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. It basically calculates how many times does a number appear in a document. It extracts a total of 15028 features from the input dataset.

Three different Similarity/Distance Measures :

➢ Euclidean Distance

For finding the euclidean distance between the data points, we have defined the function for finding the Euclidean distance**.** This function will return the measured euclidean distance between the data points. Euclidean Distance is a measure of the true straight line distance between two points in Euclidean space. For each dimension, we subtract one point's value from the other's to get the length of that “side” of the triangle in that dimension, square it, and add it to our running total. The square root of that running total is our Euclidean distance, as shown in the below figure.



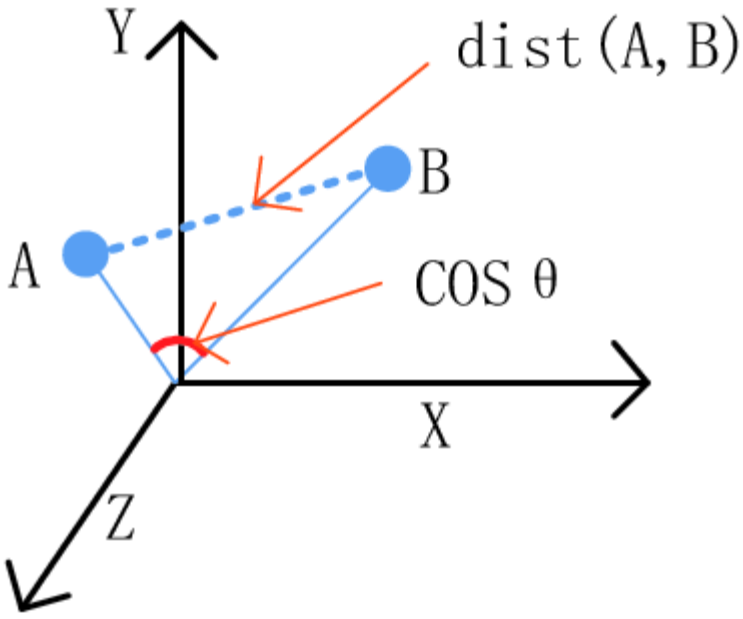
➢ Manhattan Distance

For finding the Manhattan distance between the data points, we have defined a function for finding the Manhattan distance**.** This function will return the measured Manhattan distance between train data to each test data point. Manhattan distance is calculated as the sum of the absolute differences between the two vectors, as shown in the below figure.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

➢ Cosine Similarity

For finding the cosine similarity between the data points, we have defined a function for finding the cosine similarities**.** This function will return the cosine similarity between the training data point to each test data point. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space, as shown in the below figure.



KNN Classifier

The KNN classifier is built based on the 80% training and 20% test data. We will give training data, the value of *K*, and test instances as the input to this classifier. Then, the classifier will first measure the distances(like euclidean, manhattan, etc) between the data points. When we have the distances, it will sort it out on the basis of distances(small to large). Then, based on the value of *K*, it will choose the top *K* neighbors. At last, it will calculate the most frequent class of the neighbors.

From the training data, out of 156060, after preprocessing and cleansing, we get a total of 154885 instances. The total number of correctly classified instances is 16416 out of 30977 for the test instances. When we have randomly taken 5 *K* values and used euclidean distance measure, we have achieved an accuracy of 52.99 %.

Analyze the performance with varying K

We have changed the value of K in the range of 2 to 45 in the interval of 3 to analyze the accuracy of the KNN classifier through different similarity/distance measures.

When we selected cosine similarity as the distance measure and varied the value of *K* in the range of 2 to 45, we got the maximum accuracy of *54.930 % at K = 11*.

When we selected euclidean distance as the distance measure and varied the value of K in the range of 2 to 45, we got the maximum accuracy of *53.557 % at K = 8.*

When we selected the Manhattan distance as the distance measure and varied the value of K in the range of 2 to 45, we got the maximum accuracy *53.209 % at K = 8.*

We can conclude that out of three different distance measures, cosine similarity is having best accuracy and should be preferred over euclidean and manhattan.

Plotting of Graphs

We have imported the matplotlib.pyplot library for plotting the graphs. We plotted the graphs of different distance measures and analyzed the results. Cosine similarity is having maximum accuracy among all. And a trend of generalization can be seen that with increasing K, accuracy also increases initially and decreases after reaching its maximum and converges to minimum.