MACHINE LEARNING

Assignment - 2

k-NN Classifier

GROUP (GID: 53)
19CS10050 Rupinder Goyal
19CS30043 Shrinivas Khiste

K-NN Algorithm

K-NN (or k-Nearest Neighbour) is a **non-parametric supervised learning algorithm** that can be used for classification and regression. It is based on the idea that similar samples will belong to the same class. It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. It is one of the simplest Machine Learning Algorithms that classifies new data based on its similarity with available train data.

Working of the K-NN Algorithm

1. While training we just save the training data that is the feature vectors of the input and the corresponding labels.
2. When we get a new sample, we compare it with the stored training data and extract the k most similar samples based on some similarity function (discussed later).
3. The prediction which has the highest frequency among the k samples is the prediction of the algorithm.

Similarity Function

Here we are going to look at 3 similarity functions

1. **Cosine Similarity:** It is the cosine of the angle between two n-dimensional vectors in an n-dimensional space. It is described mathematically as the division between the dot product of vectors and the product of the euclidean norms or magnitude of each vector. It is a value bound between 0 and 1. The closer it is to 1, the more similar the vectors are. It is generally used in text analytics.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

2. **Euclidean Distance:** It is the square root of the sum of the squared difference between the corresponding attributes in the feature vectors. It is the measure of the true straight line distance between two points in Euclidean space.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

3. **Manhattan Distance:** It is the sum of the absolute difference between the corresponding attributes in the feature vectors. It is sometimes also called taxicab distance or city block distance.

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

TF - IDF Vectorizer

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency.

Term Frequency(TF): The number of times a word appears in a document divided by the total number of words in the document.

Inverse Data Frequency (IDF): The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

TF-IDF is simply the TF multiplied by IDF

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Stop Words

Stop words are a set of commonly used words in a language. Examples of stop words in English are “a”, “the”, “is”, “are” and etc.

This is in-built in the TF - IDF Vectorizer which is used in the assignment to remove the english stop words.

Stepwise Explanation

Importing Libraries and loading data

We have used the following libraries in the project

1. Pandas: For handling the tabular data
2. Numpy: For handling data vectors
3. Matplotlib: for the graphs
4. Sklearn train test split: for splitting the data into train and test set
5. Sklearn TfidfVectorizer: To convert text into numbers

Distance (Similarity) Functions

1. **EuclideanDistance(point1 , point2):**
    - Parameters:
        - Point1: numpy array of data point 1
        - Point2: numpy array of data point 2
    - This function is used to find the Euclidean distance between both points.
    - Numpy functions are used (just to speed up).
    - Returns:
        - Distance between the points

2. **manhattanDistance(point1, point2):**
    - Parameters:
        - Point1: numpy array of data point 1

    - Point2: numpy array of data point 2
- This function is used to find the manhattan distance between both the points.
- Numpy functions are used (just to speed up).
- Returns:
    - Distance between the points

3. cosineSimilarity(point1, point2):

- Parameters:
    - Point1: numpy array of data point 1
    - Point2: numpy array of data point 2
- This function is used to find the cosine similarity between both points.
- Numpy functions are used (just to speed up).
- The value is close to 1 then the points are said to be similar and if the value is close to -1, then the points are said to be dissimilar.
- 1 - cosine value is returned to make it consistent with other distance functions (as the value increases, the similarity between points decreases)
- Returns:
    - Distance between the points

KNN Classifier Class

1. Constructor(Kvalue = 0):

- Parameters:
    - Kvalue - an integer whose default value is 0
- This function creates the KNN Classifier object

2. fit(x_train, y_train):

- Parameters:
    - x_train: training data, numpy array
    - Y_train: outputs of training data, numpy array

- This function just stores the data passed in the object

3. predict(x_test, distanceFunction)

- Parameters:
    - X_test: the test data, numpy array
    - distanceFunction: function to be used to calculate distances
- For each test data point, this function calculates the distance of it (using the distance function passed as the parameter) from all training data points and then prints the output values using the nearest k data points.
- If the number of spam and non-spams in the neighbourhood are equal, then the test data point is classified as spam.
- Returns:
    - Y_test: the predicted output, a list

4. findOptimalK (x_test, distanceFunction)

- Parameters:
    - X_test: the test data, numpy array
    - distanceFunction: function to be used to calculate distances
- This function returns the prediction for all k's ranging from 1 to the size of training data. This can then help to predict the best k to be used for further predictions
- For each test data point, the function calculates the distance of it from all the training data points (distance calculated from the distance function passed as the parameter) and then predicts the output value for each possible k
- Returns:
    - Y_test: a list of lists, where the ith list contains the predictions using i nearest neighbours

Accuracy Functions

1. accuracy(list1, list2):

- Parameters:
    - list1 - a numpy array
    - list2 - a numpy array
- The function raises an exception if the size of the lists is not the same or the lists are empty.
- Then this function calculates the number of rows in which both have the same values and then divide this number by the size of the list to calculate the accuracy
- Returns:
    - Accuracy

Data Loading and Text Preprocessing

- The data is first loaded using pandas and all the unnecessary columns are dropped.
- Train_test_split is then used to split the data into training and test data
- TfidfVectorizer object is then created and trained using the training text data
- English stop words are removed using TfIdfVectorizer
- The training and testing text data are then transformed into numbers using the vectorizer.

Graph Plotting

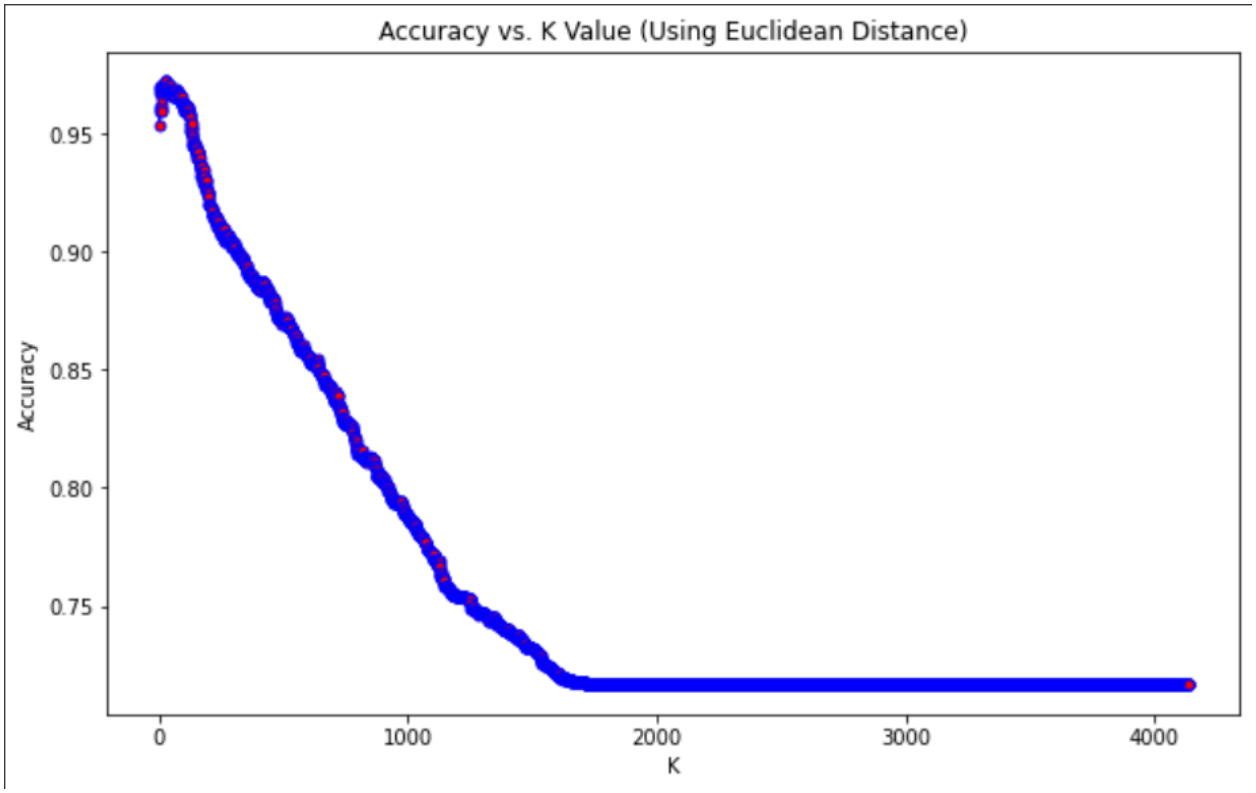
- For each of the distance functions, graphs of accuracy vs k-value are plotted.
- findOptimalK function of the KNN Classifier object is used to return the predicted output for all possible values of k's.
- Then accuracy is calculated for all possible k's and graphs are plotted.

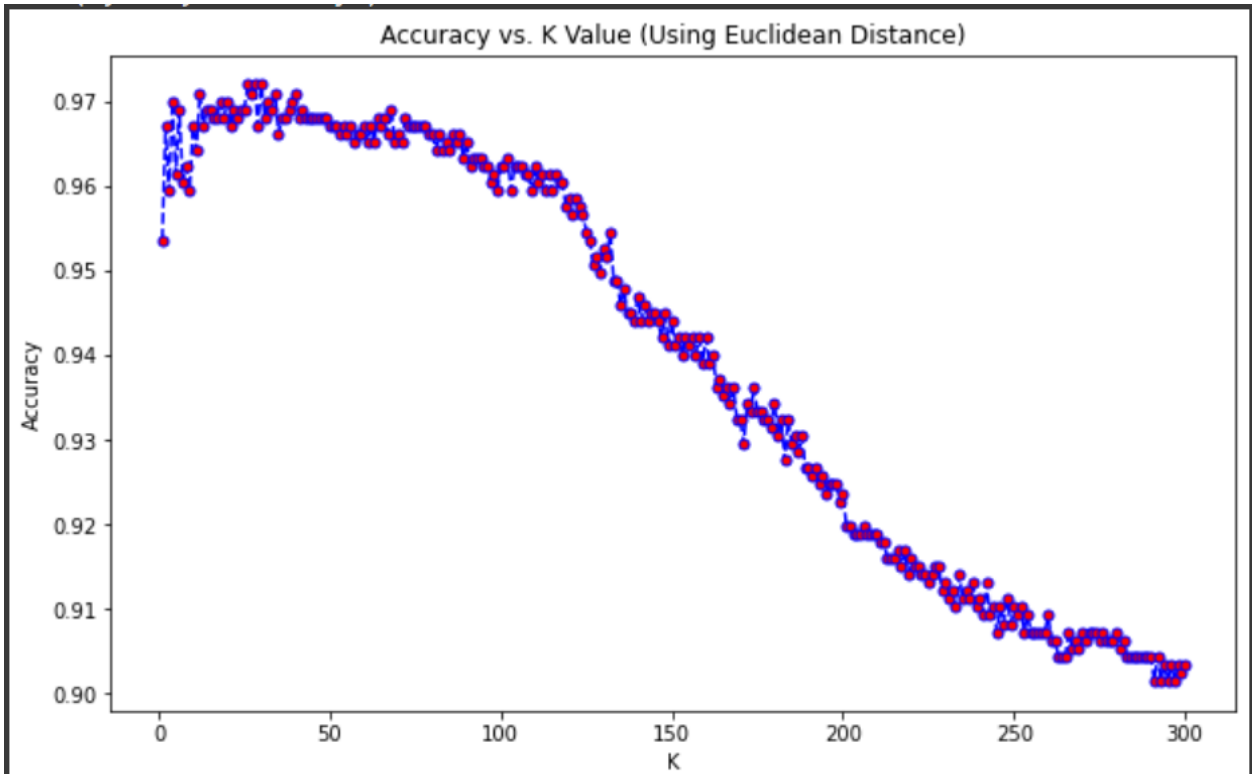
Results and Analysis

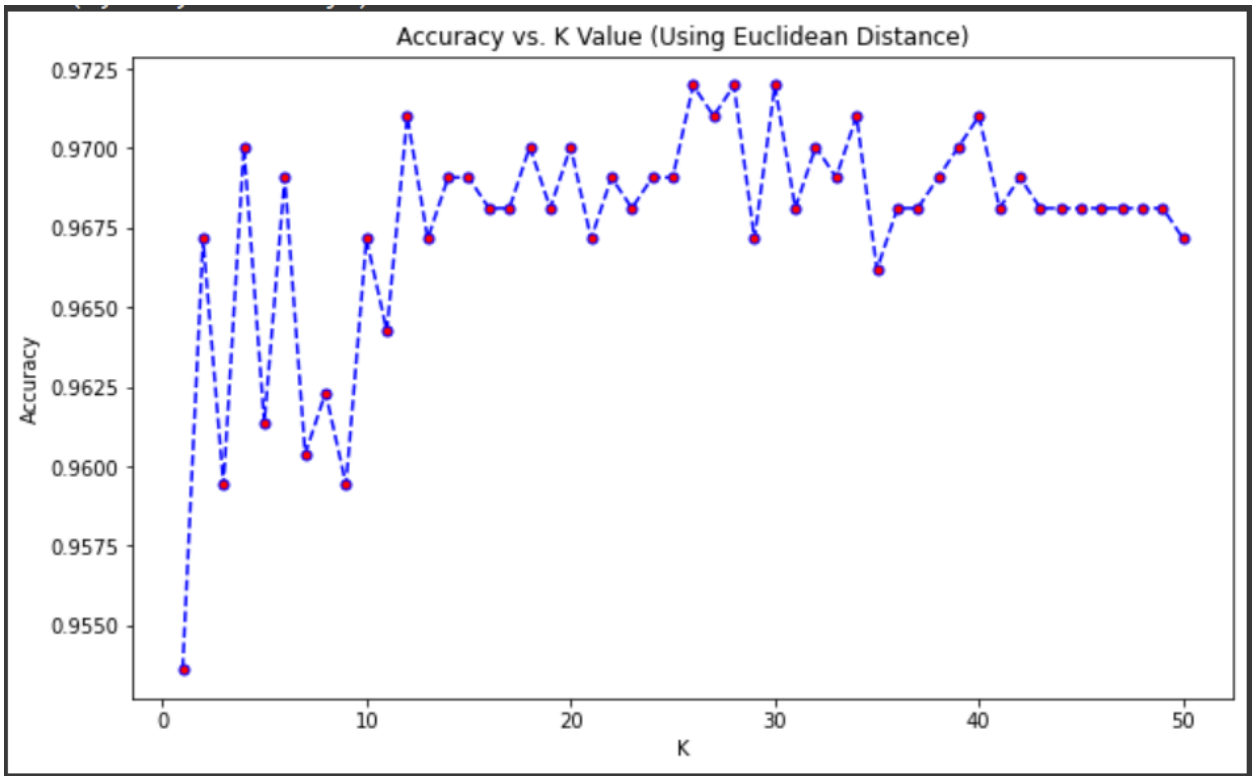
1) Euclidean Distance

Optimal K: 26

Maximum Accuracy (at k = 26) = 97.198067%



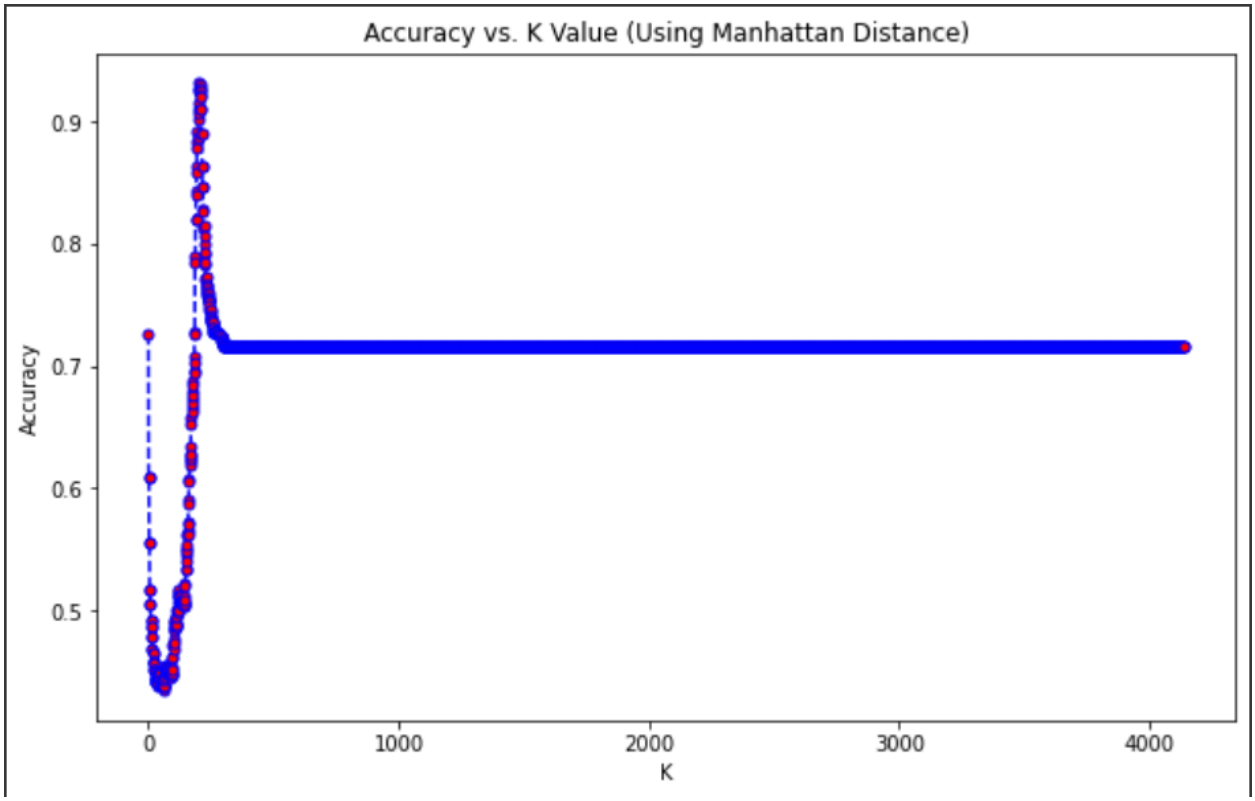

Accuracy vs. K Value (Using Euclidean Distance)
Accuracy
K
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0
50
100
150
200
250
300

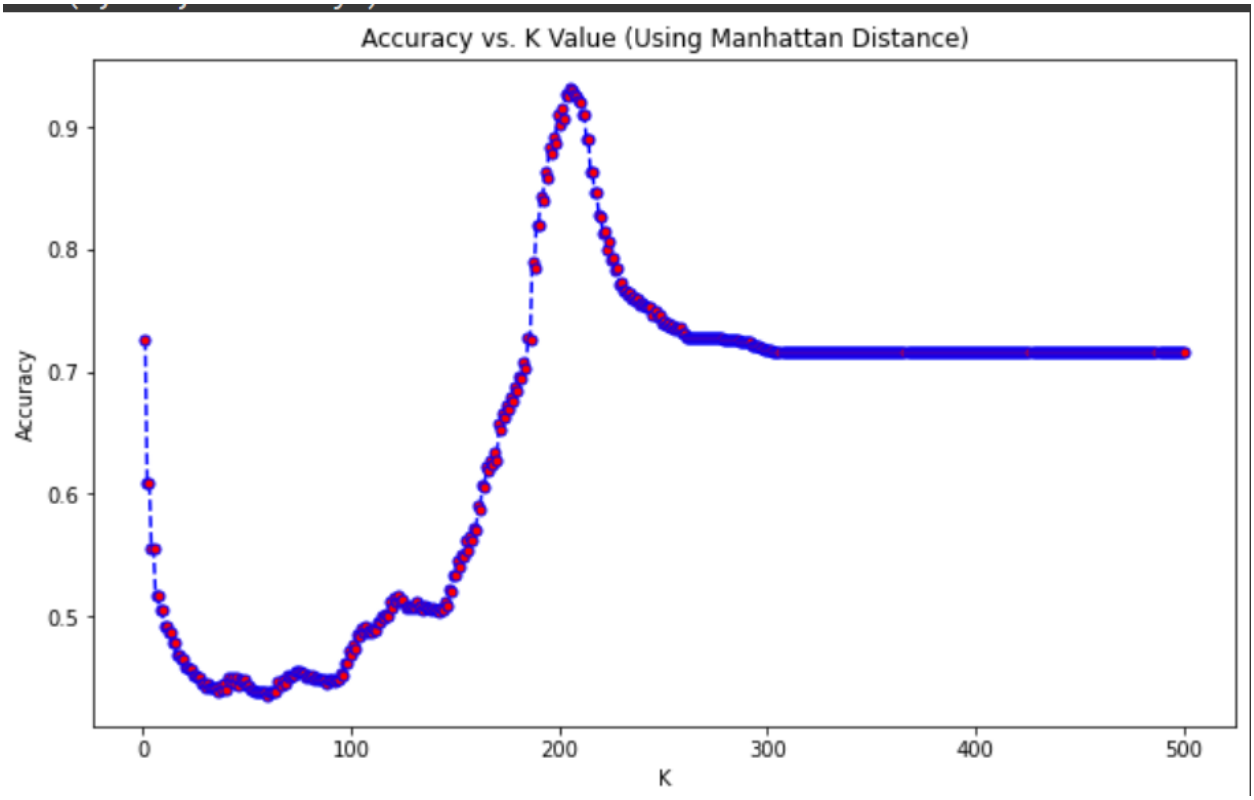

Accuracy vs. K Value (Using Euclidean Distance)
Accuracy
K
0.9725
0.9700
0.9675
0.9650
0.9625
0.9600
0.9575
0.9550
0
10
20
30
40
50

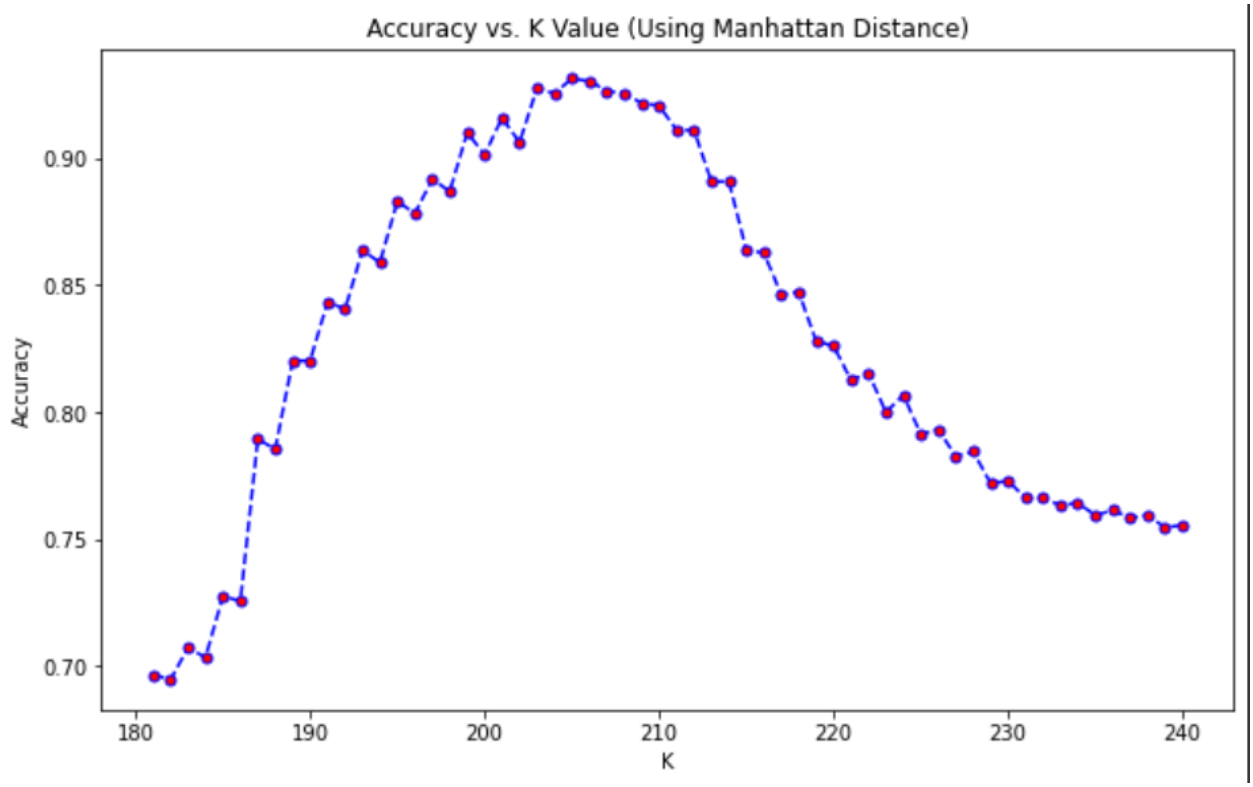
2. Manhattan Distance

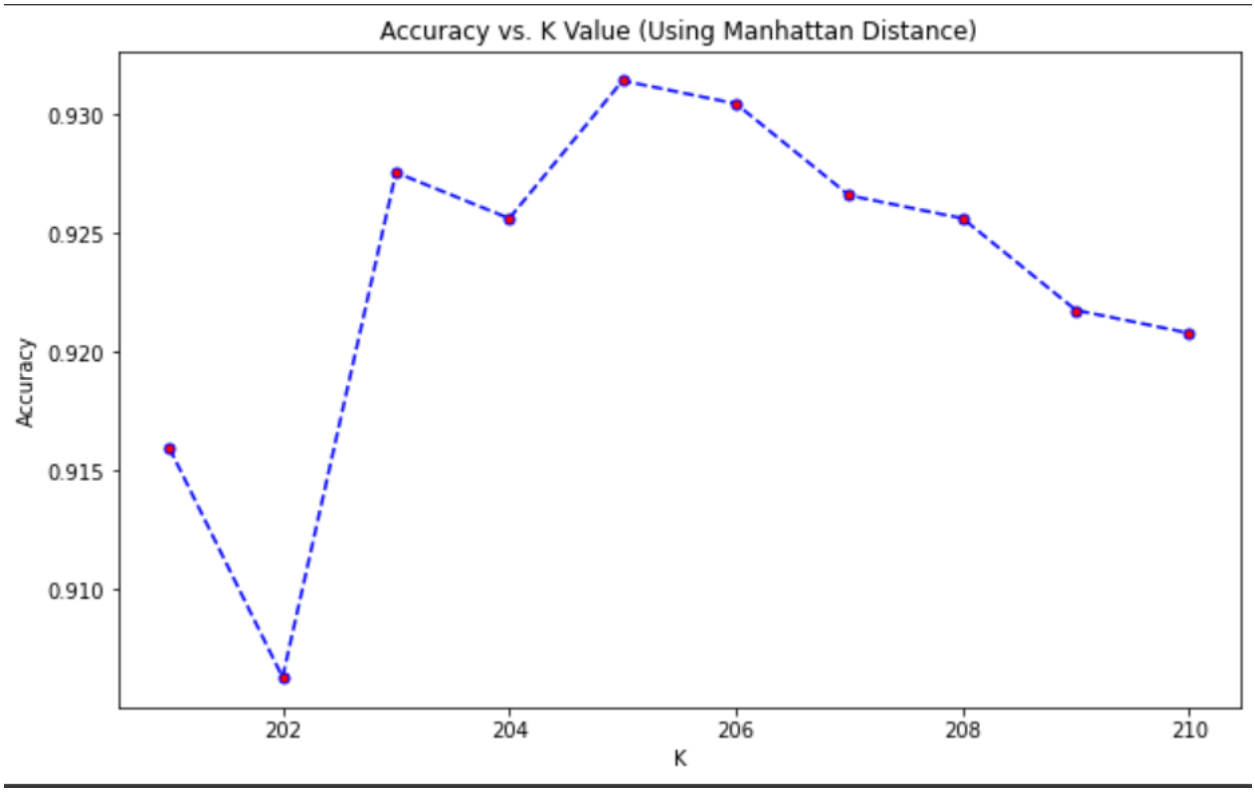
Optimal K: 205

Maximum Accuracy (at k = 205) = 93.14%




Accuracy vs. K Value (Using Manhattan Distance)
Accuracy
K
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400
500

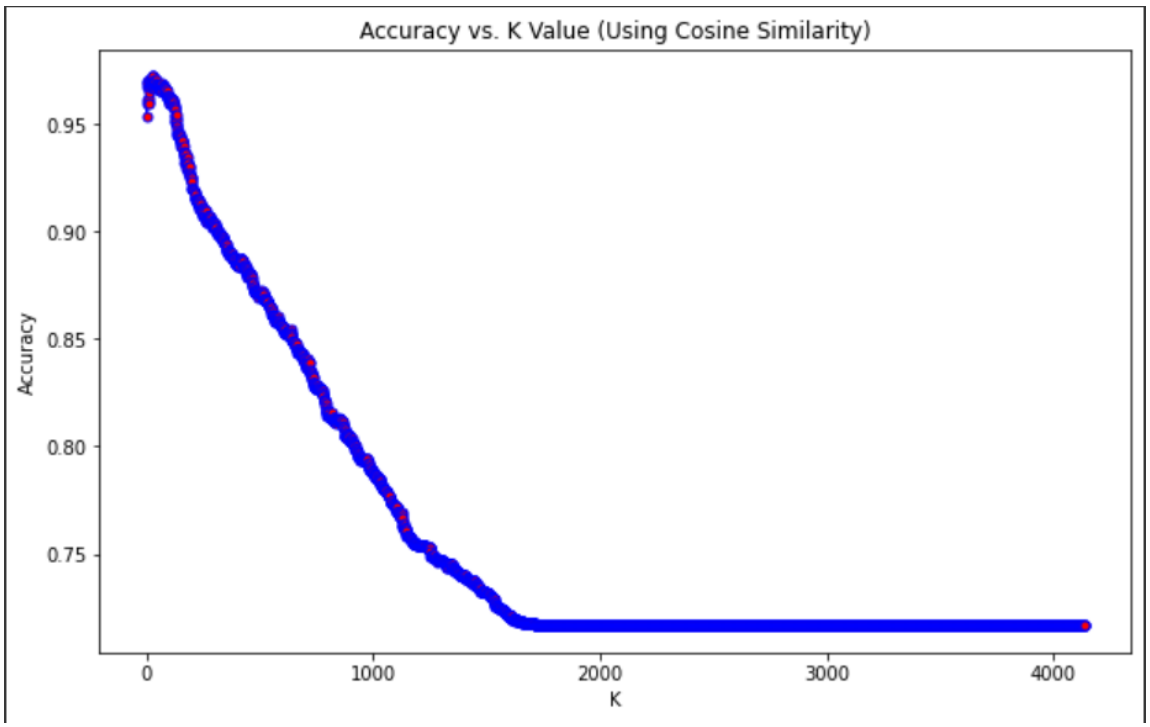

Accuracy vs. K Value (Using Manhattan Distance)
Accuracy
K
0.90
0.85
0.80
0.75
0.70
180
190
200
210
220
230
240

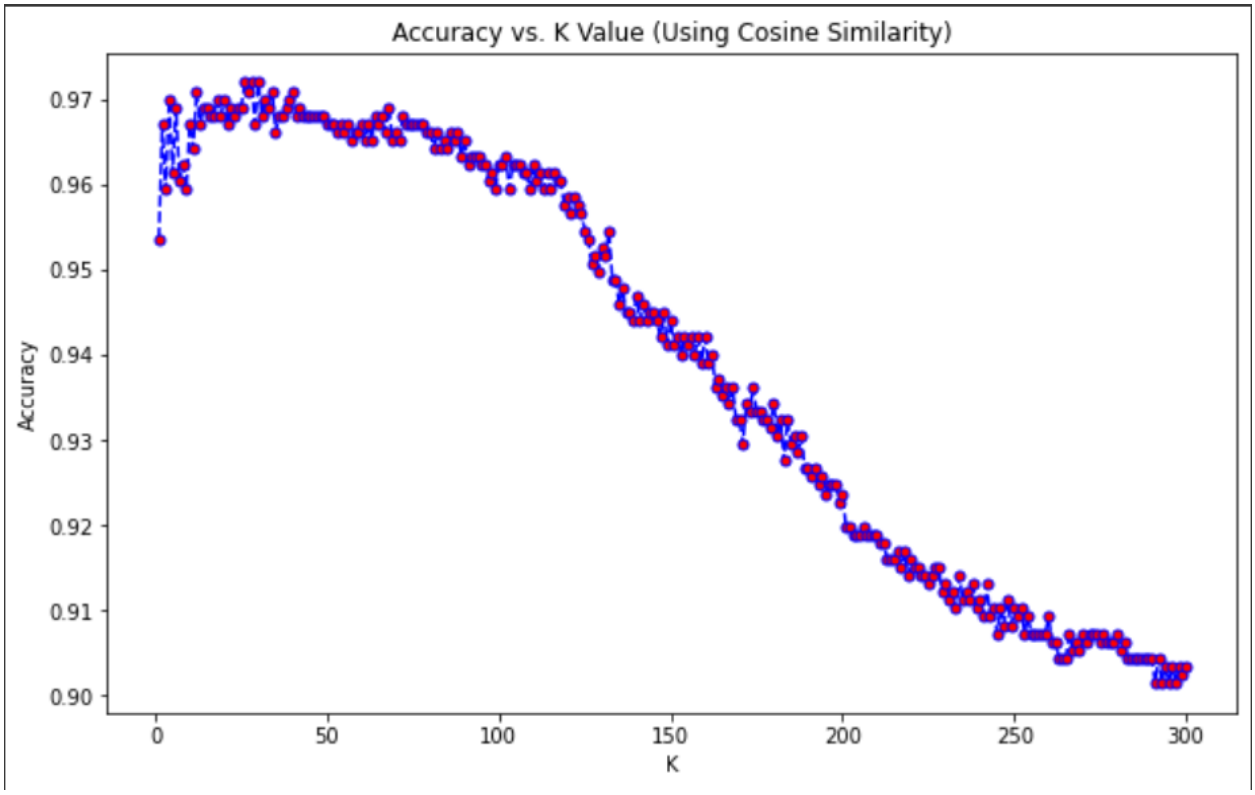

Accuracy vs. K Value (Using Manhattan Distance)
Accuracy
0.930
0.925
0.920
0.915
0.910
202
204
206
208
210
K

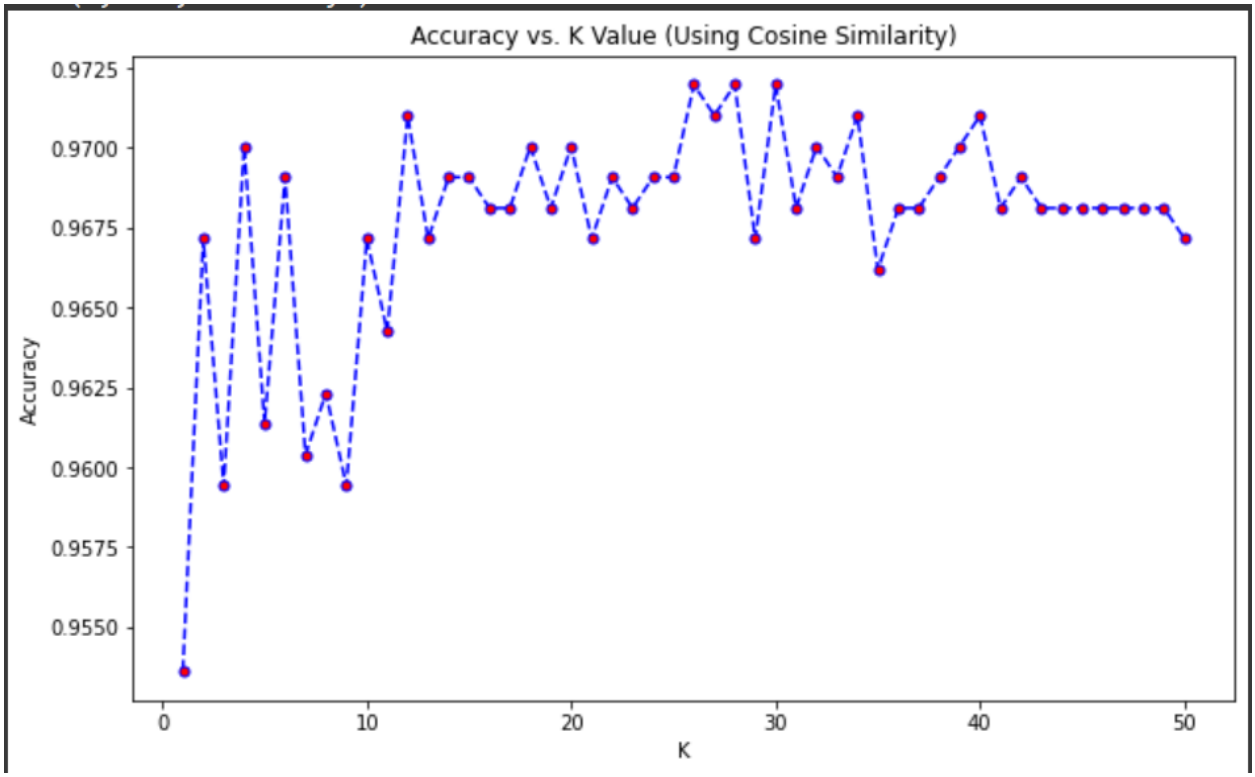
3. Cosine Similarity

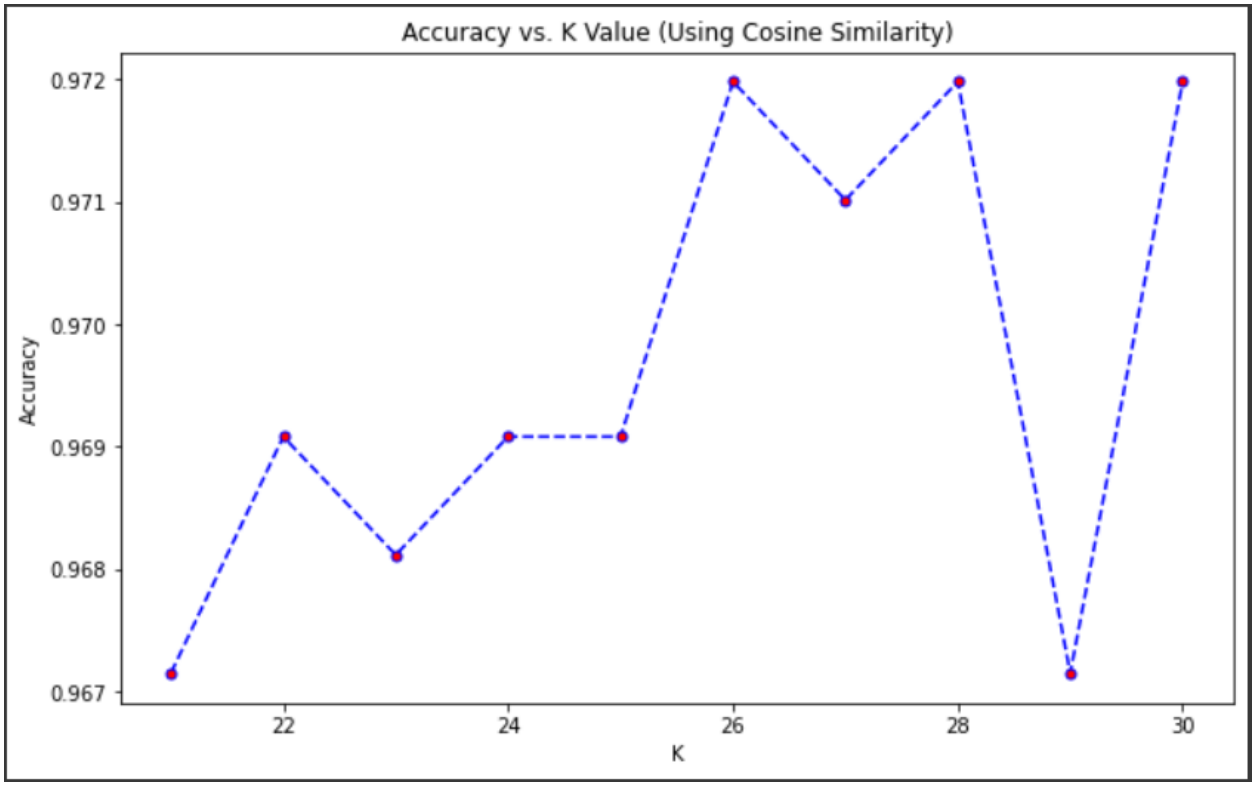
Optimal K: 26

Maximum Accuracy (at k = 26) = 97.198067%

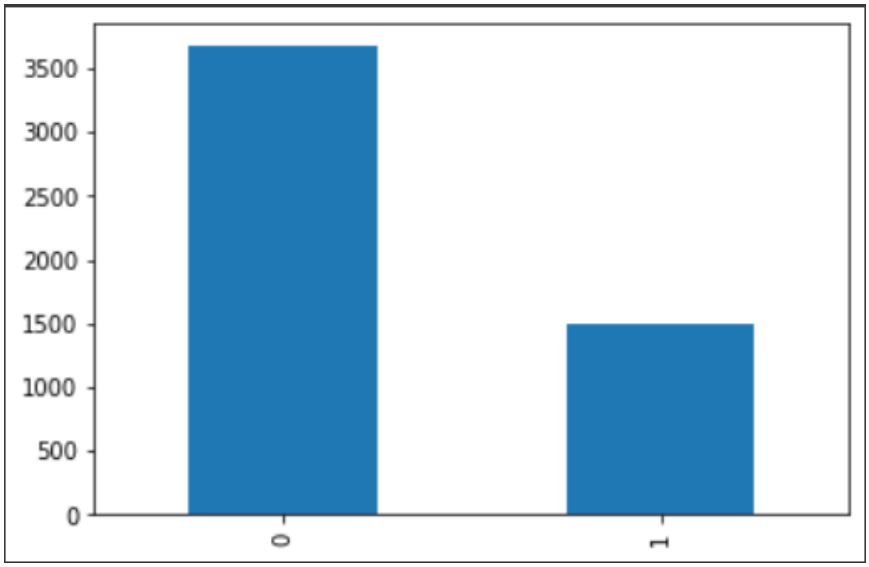



Accuracy vs. K Value (Using Cosine Similarity)
Accuracy
K
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0
50
100
150
200
250
300


Accuracy vs. K Value (Using Cosine Similarity)
Accuracy
K
0.9725
0.9700
0.9675
0.9650
0.9625
0.9600
0.9575
0.9550
0
10
20
30
40
50



Analysis of Results



- We can observe from the bar graph above that the number of ham labels is 2.5 times the number of spam labels. This imbalance in the data can affect the results.

- Accuracy with cosine and Euclidean distance is very good (about 97%) but it is a bit less with manhattan distance (about 93%).
- Thus we can say that Manhattan distance is not a good metric for text-based data whereas Cosine and Euclidean distance should be preferred for text-based data.

Trends of the Graphs

- The accuracy plot stabilises after some k. This is because a majority of the labels are not spam, they begin to dominate and thus no spam mail is detected. Thus we can see that in all the distance metrics, the accuracy stabilises to a similar value (almost the per cent of non-spam emails (70%))
- The best accuracy is achieved somewhere in the middle as this is where we get the best idea of the distribution of the data near the test data point
- Cosine distance and Euclidean distance metrics have similar graphs. The accuracy increases in the beginning (till it gets a better idea of the distribution of data around it) till it reaches the peak and then it starts decreasing and then finally stabilises to about 70%.
- Manhattan distance graph gives the highest accuracy at a much higher k than others. Its accuracy decreases earlier probably because it does not get a good idea of the data distribution around it with this metric. After some time the accuracy begins to increase and then reaches a peak and then it decreases again to become stable at around 70% finally.