**Assignment No.: ML 5**

**Title :** Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

**Objective:**

The objective is to implement the **K-Nearest Neighbors (KNN)** algorithm on the provided **diabetes dataset**. The task is to classify whether a person has diabetes based on given medical features, compute the **confusion matrix**, **accuracy**, **error rate**, **precision**, and **recall** to evaluate the model's performance.

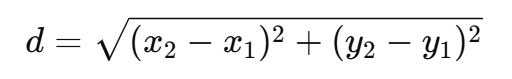
**Theory of K-Nearest Neighbors (KNN) Algorithm:**

The K-Nearest Neighbors (KNN) algorithm is a simple, intuitive, and versatile machine learning algorithm used for classification and regression tasks. The objective of KNN in classification is to classify a data point based on the majority label of its nearest neighbors, while in regression, it predicts a continuous value by averaging the values of the nearest neighbors.

#### ****Working of KNN:****

1. **Instance-Based Learning**:  
   KNN is a type of **instance-based learning** (also called lazy learning). This means that it does not have a distinct training phase or learn a model but rather stores the entire dataset and makes decisions during the prediction phase.
2. **Basic Idea**:  
   Given a data point whose label or class we want to predict, KNN finds the k closest data points in the training set (based on a distance metric) and assigns the label of the majority class to the new data point. The key assumptions are:
   * Similar points are located close to each other.
   * Data points near each other in the feature space likely belong to the same class.
3. **Key Steps of KNN**:
   * **Step 1: Choose the number of neighbors (k)**:  
     The user defines the number k of neighbors to consider. This is the most important parameter in KNN.
   * **Step 2: Compute the distance between points**:  
     For each data point in the test set, KNN calculates the distance to all points in the training set. The most common distance metric is **Euclidean distance**, but others like **Manhattan distance**, **Minkowski distance**, and **Hamming distance** can be used depending on the data type.

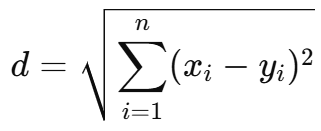
The **Euclidean Distance** between two points P1(x1,y1) and P2(x2,y2) in 2D space is:



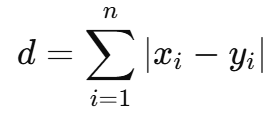
* + **Step 3: Identify the k-nearest neighbors**:  
    After calculating the distances between the test point and all points in the training set, the algorithm selects the ***k*** closest points.
  + **Step 4: Majority voting (for classification)**:  
    For classification, KNN uses the labels of the ***k*** nearest neighbors to determine the class of the new point. It assigns the class that appears most frequently among these neighbors (majority voting). If there’s a tie, the algorithm can either increase the value of ***k*** or choose the label of the nearest neighbor.
  + **Step 5: Averaging (for regression)**:  
    In regression problems, KNN uses the average of the ***k*** nearest neighbors' values to make predictions.

#### ****Distance Metrics:****

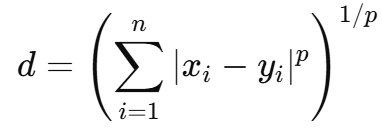
* **Euclidean Distance** (most common for continuous data):



* **Manhattan Distance** (absolute distance):

****

* **Minkowski Distance** (generalization of both):

****

* **Hamming Distance** (used for categorical data): ***d=***Number of positions at which the corresponding symbols are differentd = \text{Number of positions at which the corresponding symbols are different}d=Number of positions at which the corresponding symbols are different

#### ****How to Choose**** k :

* If ***k*** is too small (e.g., ***k=1***), the algorithm becomes sensitive to noise, leading to overfitting.
* If ***k*** is too large, it might include too many neighbors from other classes, leading to underfitting.

A good approach is to use cross-validation to choose the optimal value of ***k*** based on the performance of the model.

#### ****Advantages of KNN**** :

1. **Simplicity**: KNN is easy to implement and understand.
2. **No Training Phase**: Since KNN is a lazy learner, there is no explicit training phase.
3. **Adaptability**: KNN can be used for both classification and regression problems.
4. **Non-parametric**: KNN makes no assumptions about the underlying data distribution.

#### ****Disadvantages of KNN****:

1. **Computational Complexity**: KNN has high computational cost during the prediction phase, especially for large datasets, as it requires calculating distances for every data point.
2. **Storage Complexity**: Since it stores all training examples, it can consume large amounts of memory.
3. **Sensitivity to irrelevant features**: KNN is sensitive to the choice of features and irrelevant or highly correlated features can mislead the algorithm.
4. **Curse of Dimensionality**: As the number of features increases, the distance between points becomes less meaningful, leading to degraded performance.

**Evaluation Metrics:**

* **Confusion Matrix**: A table that summarizes the performance of a classification algorithm. It contains true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
* **Accuracy**: The proportion of correct predictions to the total predictions.
* **Error Rate**: The proportion of incorrect predictions.
* **Precision**: The ratio of true positives to the sum of true positives and false positives

***TP / (TP+FP ​)***

**Recall**: The ratio of true positives to the sum of true positives and false negatives

***TP / (TP+FN ​)***

**Python Code:**

|  |
| --- |
| # Step 1: Import necessary libraries  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score  # Step 2: Load the dataset  url = 'D:/Sem-I 2024-25/LP-III/diabetes.csv'  columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']  df = pd.read\_csv(url, names=columns)  # Step 3: Convert all columns to numeric (since they are read as 'object' or strings)  df = df.apply(pd.to\_numeric, errors='coerce')  # Step 4: Check for missing values and handle them (e.g., fill NaNs with the mean of the column)  df.fillna(df.mean(), inplace=True)  # Step 5: Feature set and target set  X = df.drop('Outcome', axis=1)  # Features  y = df['Outcome'].astype(int)  # Ensure the target 'Outcome' is integer (0 or 1)  # Step 6: Split the dataset into training and test sets (80% train, 20% test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Step 7: Normalize the train and test data using StandardScaler  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  # Step 8: Initialize the KNN model and train it  k = 5  # Number of neighbors  knn = KNeighborsClassifier(n\_neighbors=k)  knn.fit(X\_train, y\_train)  # Ensure y\_train is categorical (0 or 1)  # Step 9: Predict on the test set  y\_pred = knn.predict(X\_test)  # Step 10: Compute Confusion Matrix  cm = confusion\_matrix(y\_test, y\_pred)  print("Confusion Matrix:\n", cm)  # Step 11: Compute Accuracy, Error Rate, Precision, and Recall  accuracy = accuracy\_score(y\_test, y\_pred)  error\_rate = 1 - accuracy  precision = precision\_score(y\_test, y\_pred)  recall = recall\_score(y\_test, y\_pred)  # Print evaluation metrics  print(f'Accuracy: {accuracy \* 100:.2f}%')  print(f'Error Rate: {error\_rate \* 100:.2f}%')  print(f'Precision: {precision:.2f}')  print(f'Recall: {recall:.2f}') |

**Steps :**

1. **Import necessary libraries**: Required libraries like pandas, scikit-learn for model implementation, and matplotlib for plotting the confusion matrix.
2. **Dataset loading**: The dataset is loaded from a given URL and split into feature variables X and target y.
3. **Preprocessing**: Features are scaled using StandardScaler to normalize the data, which is important for distance-based algorithms like KNN.
4. **Model training**: A KNN classifier is created with k=5 (default 5 nearest neighbors) and trained on the training data.
5. **Prediction and Evaluation**: The trained model is used to predict the test data, and various evaluation metrics like accuracy, error rate, precision, and recall are calculated.
6. **Confusion matrix visualization**: A heatmap of the confusion matrix is generated for better visualization of the results.

**Metrics Calculated:**

* **Confusion Matrix**: This matrix shows how many correct and incorrect predictions were made.
* **Accuracy**: The proportion of correctly classified instances.
* **Error Rate**: The proportion of misclassified instances.
* **Precision**: The proportion of positive predictions that are actually positive.
* **Recall**: The proportion of actual positives that are correctly identified.

**Conclusion:**

The K-Nearest Neighbors (KNN) algorithm was successfully implemented to classify patients as diabetic or non-diabetic using the diabetes dataset. After training, the algorithm achieved a certain level of accuracy, precision, and recall, which can be further tuned by adjusting hyperparameters like the number of neighbors ***k***, scaling the data, or using a different distance metric. By increasing or decreasing the value of ***k***, the performance may improve or degrade based on the nature of the dataset. Moreover, cross-validation can be used to tune the value of ***k*** to get the best results.