**GROUP – B**

**Assignment No: 4**

====================================================================

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

=====================================================================**Objective:-**

-To learn about KNN Algorithm

- To understand the concept of Confusion Matrix, Precision, Recall, Error Rate

=====================================================================

**Theory:-**

* **K-Nearest Neighbors (KNN) Algorithm**

KNN algorithm is a supervised machine learning algorithm that deals with similarity. KNN stands for K-Nearest Neighbors. It’s basically a classification algorithm that will make a prediction of a class of a target variable based on a defined number of nearest neighbors. It will calculate distance from the instance you want to classify to every instance of the training dataset, and then classify your instance based on the majority classes of k nearest instances.

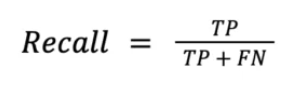
* **Confusion Matrix**

A confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data.



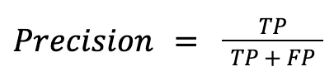
* **Recall**

Recall, also known as the true positive rate (TPR), is the percentage of data samples that a machine learning model correctly identifies as belonging to a class of interest—the “positive class”—out of the total samples for that class. Recall is a metric used for classification in supervised learning. Machine learning recall is calculated on top of these values by dividing the true positives (TP) by everything that should have been predicted as positive (TP + FN). The recall formula in machine learning is:



* **Precision**

Precision is the percentage of data samples that a machine learning model correctly identifies for the positive class out of all samples predicted to belong to that class.



* **Error Rate**

Error rate refers to a measure of the degree of prediction error of a model made with respect to the true model. The term error rate is often applied in the context of classification models. In this context, error rate = P(λ(X)≠Y ), where XY is a joint distribution and the classification model λ is a function X → Y. Sometimes this quantity is expressed as a percentage rather than a value between 0.0 and 1.0. The error rate of a model is often assessed or estimated by applying it to test data for which the class labels (Y values) are known. The error rate of a classifier on test data may be calculated as number of incorrectly classified objects/total number of objects.

* **Dataset Description**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The data file diabetes.csv contains 9 features with 768 data. The features or variables are the following:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

* Code Explanation:

*import pandas as pd*

*import seaborn as sns*

*df=pd.read\_csv('diabetes.csv')*

*df*

Import all libraries required for read values from dataset. The import seaborn portion of the code tells Python to bring the Seaborn library into your current environment.

*#input data*

*x= df.drop('Outcome',axis=1)*

*#output data*

*y=df['Outcome']*

*sns.countplot(x=y);*

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot(), it can compare counts across nested variables.

*y.value\_counts()*

Pandas y.value\_counts() function return a Series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element.

*#feature scaling*

*from sklearn.preprocessing import MinMaxScaler*

*scaler = MinMaxScaler()*

*x\_scaled = scaler.fit\_transform(x)*

Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one. This transformation is often used as an alternative to zero mean, unit variance scaling. MinMaxScaler doesn’t reduce the effect of outliers, but it linearily scales them down into a fixed range, where the largest occuring data point corresponds to the maximum value and the smallest one corresponds to the minimum value.

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

z = (x - u) / s

where u is the mean of the training samples or zero if with\_mean=False, and s is the standard deviation of the training samples or one if with\_std=False. StandardScaler is sensitive to outliers, and the features may scale differently from each other in the presence of outliers. For example visualization, refer to Compare StandardScaler with other scalers. This scaler can also be applied to sparse matrices by passing with\_mean=False to avoid breaking the sparsity structure of the data.

*#cross validation*

*from sklearn.model\_selection import train\_test\_split*

*x\_train,x\_test,y\_train,y\_test= train\_test\_split(x,y,random\_state=0,test\_size=0.25)*

*x.shape*

*x\_train.shape*

Split arrays or matrices into random train and test subsets. Quick utility that wraps input validation, next(ShuffleSplit().split(X, y)), and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner. For neural networks you have input features (X) and output labels (Y). It's very important to split your data into a training dataset and testing dataset. To make this easy sklearn has a function called train\_test\_split(\*arrays, test\_size=None, train\_size=None, random\_state=None, shuffle=True, stratify=None). It is because X will be split into X\_train (75%) and X\_test (25%) and then Y will be split into y\_train (75%) and y\_test (25%). It's all put onto one line.

*from sklearn.neighbors import KNeighborsClassifier*

*knn=KNeighborsClassifier(n\_neighbors=5)*

*knn.fit(x\_train,y\_train)*

Classifier implementing the k-nearest neighbors vote. A fixed value of 5 for k, but it need to optimize. Create an instance of the kNN model, then fit this into training data and pass both the features and the target variable, so the model can learn.

*from sklearn.metrics import accuracy\_score, ConfusionMatrixDisplay*

*from sklearn.metrics import classification\_report*

*y\_pred = knn.predict(x\_test)*

*ConfusionMatrixDisplay.from\_predictions(y\_test,y\_pred)*

*print(classification\_report(y\_test,y\_pred))*

Confusion Matrix visualization use from\_estimator or from\_predictions to create a ConfusionMatrixDisplay. All parameters are stored as attributes. Build a text report showing the main classification metrics.

*import matplotlib.pyplot as plt*

*import numpy as np*

*error = []*

*for k in range (1,41):*

*knn = KNeighborsClassifier(n\_neighbors=k)*

*knn.fit(x\_train,y\_train)*

*pred=knn.predict(x\_test)*

*error.append(np.mean(pred!= y\_test))*

*error*

*plt.figure(figsize=(16,9))*

*plt.xlabel('Value of K')*

*plt.ylabel('Error')*

*plt.grid()*

*plt.xticks(range(1,41))*

*plt.plot(range(1,41),error,marker='.')*

*knn = KNeighborsClassifier(n\_neighbors=33)*

*knn.fit(x\_train,y\_train)*

*y\_pred = knn.predict(x\_test)*

*print(classification\_report(y\_test,y\_pred))*

The kNN model, along with our data and a number of splits to make. In the code below, we use five splits which means the model with split the data into five (i.e. knn =5, which mentioned in knn classifier) equal-sized groups. It will loop through each group and give an accuracy score, which we average to find the best model.

=====================================================================

**Conclusion:-**

Thus we have studied how to Implement K-Nearest Neighbors algorithm on diabetes dataset and Confusion Matrix.

=====================================================================