8. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

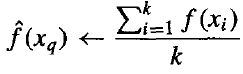
***K-Nearest Neighbor Algorithm***

Training algorithm:

* For each training example (x, f (x)), add the example to the list training examples

Classification algorithm:

* Given a query instance xq to be classified,
* Let x1 . . .xk denote the k instances from training examples that are nearest to xq
* Return

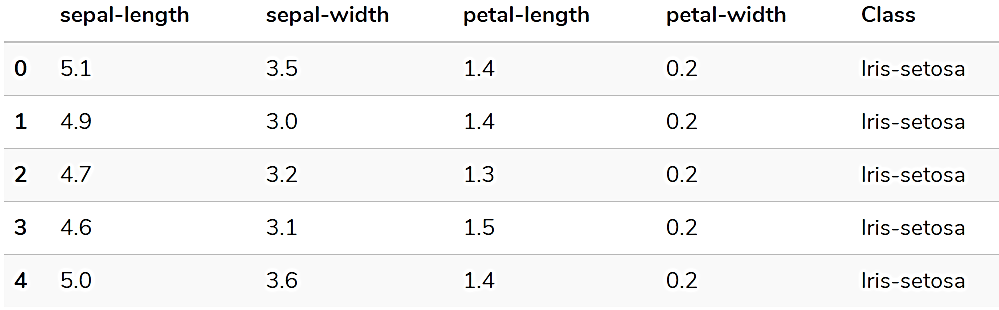


* Where, f(xi) function to calculate the mean value of the k nearest training examples.

***Data Set:***

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the Class



***Program:***

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

"""

Iris Plants Dataset, dataset contains 150 (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class

"""

iris=datasets.load\_iris()

"""

The x variable contains the first four columns of the dataset

(i.e. attributes) while y contains the labels.

"""

x = iris.data

y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')

print(y)

"""

splits the dataset into 70% train data and 30% test data. This means that out of total 150 records, the training set will contain 105 records and the test set contains 45 of those records

"""

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

#to Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train, y\_train)

#to make predictions on our test data

y\_pred=classifier.predict(x\_test)

"""

For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics.

"""

print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

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

Output:

sepal-length sepal-width petal-length petal-width

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

. . . . .

. . . . .

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica

[0 0 0 ………0 0 1 1 1 …………1 1 2 2 2 ………… 2 2]

Confusion Matrix

[[20 0 0]

[ 0 10 0]

[ 0 1 14]]

Accuracy Metrics

precision recall f1-score support

0 1.00 1.00 1.00 20

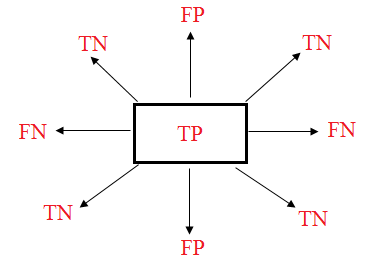
1 0.91 1.00 0.95 10

2 1.00 0.93 0.97 15

avg / total 0.98 0.98 0.98 45

**Basic knowledge**

**Confusion Matrix**

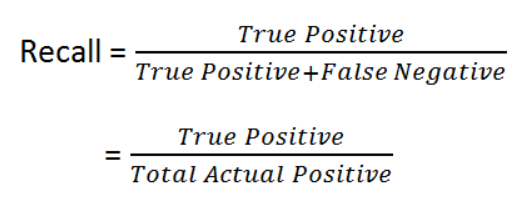
****

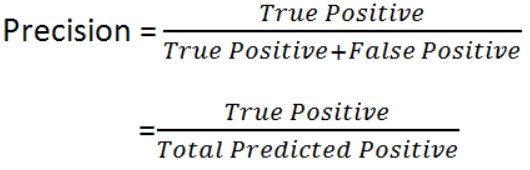
**True positives:** data points labelled as positive that are actually positive

**False positives:** data points labelled as positive that are actually negative

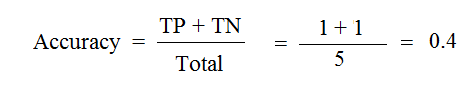
**True negatives:** data points labelled as negative that are actually negative

**False negatives:** data points labelled as negative that are actually positive

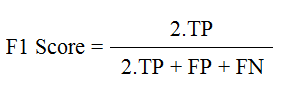




**Accuracy:** how often is the classifier correct?



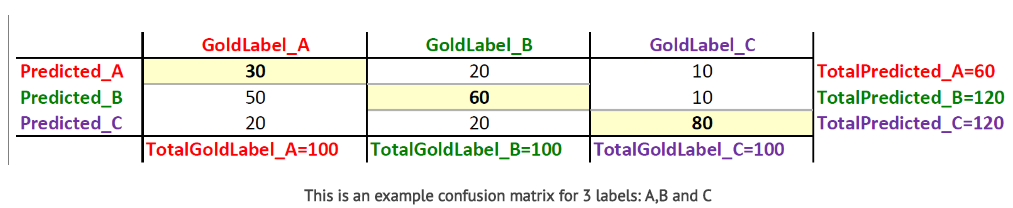
**F1-Score:**

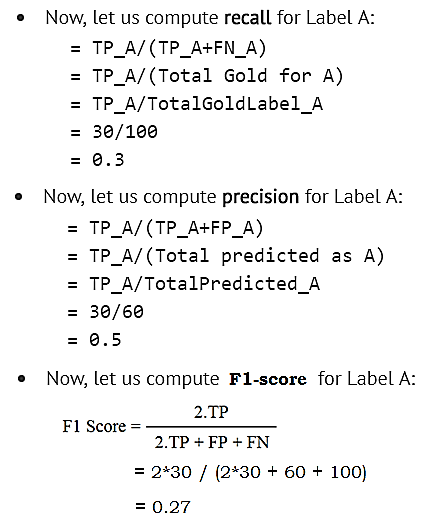
****

**Support:** Total Predicted of Class.

Support = TP + FN

**Example:**

****



* Support \_ A = TP\_A + FN\_A

= 30 + (20 + 10)

= 60