Practical-10

Objective

Implement weighted K- Nearest neighbors model.

Dataset -

- For this program an Iris flower dataset is used.
- This data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems.
- The dataset contains a set of 150 records under 5 attributes - Petal Length, Petal Width, Sepal Length, Sepal width and Class(Species).
- The dataset contains 5 columns and the target field is *'Species'.*
- Income column is divided into three classes: *'Iris Setosa'* , *'Iris virginica'* and *'Iris versicolor '.*

Species	No of values
Iris Setosa	50
Iris virginica	50
Iris versicolor	50

Model structure -

- Weighted K-nearest neighbors (KNN) algorithm is a KNN algorithm in which the weights of each of the nearest neighbours is made proportional to the distance from x, the closer the neighbour the greater its impact.
- A weight is assigned to each neighbour with the help of some mathematical concepts.
- The one used here for this program is given below -

$$Wi = \begin{cases} (d_k - d_i)/(d_k - d_1) & d_k \neq d_1 \\ 1 & d_k = d_1 \end{cases}$$

Where -
d1, d2 , d3…….dk are distances from x so that d1 is smallest and dk is greatest distance.

Implementation & Output:

```
# Mounting google colab and project path
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter
```

```
# read and store data
data = pd.read_csv(r"/content/drive/MyDrive/Dataset/Iris.csv")
data
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
#convert string column of dataset to int column
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
for i in range(len(classes)):
  data.loc[(data.Species == classes[i]), 'Species'] = i
data.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0

```
[ ] #split the data into X and Y
    X = data.drop('Species', axis=1)
    Y = data['Species']
    Y = Y.to_numpy()
```

```
[ ] X = X.to_numpy(dtype='float')
```

```
[ ] # split data into training and testing
    X_train, X_test, Y_tarin, Y_test = train_test_split(X, Y, test_size = 0.3)
```

```
# function to return class index based on maximum weights
def getClassIdx(a, b, c):
  if a>b :
    if a>c:
      return 0
    else:
      return 2
  else:
    if b > c:
      return 1
    else :
      return 2
```

```
[ ] # KNN model class
    class KNN:
      def __init__(self, k =3):
        self.k = k

      def fit(self, x, y):
        self.X_train = x
        self.Y_train = y

      def predict(self, X):
        predicted_values = [self._predict(x) for x in X]
        return np.array(predicted_values)

      def _predict(self, x):
        # distances
        distnaces = []
        for i in range(len(X_train)):
          dist = np.sqrt(sum(np.square(X_train[i] - x)))
          distnaces.append([i, dist])
        # get k nearest data
        k_indices = sorted(distnaces, key= lambda a : a[1])[: self.k]
        # get weights for K nearest data
        d1 = k_indices[0][1]
        dk = k_indices[self.k - 1][1]
        weights = []
```

```
[ ]     for i in range(len(k_indices)):
          wt = (dk - k_indices[i][1] )/ (dk - d1)
          weights.append([k_indices[i][0], wt])
        k_nearest_labels = [self.Y_train[i[0]] for i in weights]
        c1 , c2, c3 = 0 , 0 , 0
        #calculate majority vote and common class label
        for i in range(len(k_nearest_labels)):
          if k_nearest_labels[i] == 0:
            c1+=weights[i][1]
          if k_nearest_labels[i] == 1:
            c2+=weights[i][1]
          else:
            c3+=weights[i][1]
        return getClassIdx(c1, c2, c3)
```

```
[ ] # fit the model
    classifier = KNN(k= 5)
    classifier.fit(X_train, Y_tarin)
    predictions = classifier.predict(X_test)
    accuracy = np.sum(predictions == Y_test) / len(Y_test)
    accuracy = round((accuracy * 100), 2)
    print('Classifier Accuracy : ',accuracy , '%')
```

```
Classifier Accuracy :  77.78 %
```

```
[ ] # new data defined by user
    x = [[0.22, 0.56,0.3,0.25,0.8]]
    y = classifier.predict(x)
    print('Predicted Class : ', classes[y[0]])
```

```
Predicted Class :  Iris-virginica
```