Machine Learning (ML)

Name - Preksha A. Patel
sapid - 60004210126
Branch - Computer Engineering
Div - C2, Batch - 1
T.Y.B-Tech

Experiment no. 6

Aim - To implement K-Nearest Neighbour.

Theory :-

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on supervised learning technique. It assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories. The algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K-NN algorithm. It can be used for Regression as well as for classification but mostly it is used for the classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.

The K-NN working can be explained based on the below algorithm:

1. Select the number K of the neighbours
2. Calculate the Euclidean distance of K number of neighbours
3. Take the K nearest neighbours as per the calculated Euclidean distance.
4. Among these k neighbours, count the number of the data points in each category.

5. Assign the new data points to that category for which the number of the neighbour is maximum.

Selecting the value of K in the K-NN algorithm :

1. There is no way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
2. A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
3. Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm :

1. It is simple to implement
2. It is robust to the noisy training data
3. It can be more effective if the training data is large.

Disadvantages of KNN Algorithm :

1. Always needs to determine the value of K which may be complex some time.
2. The computation cost is high because of calculating the distance between the data points for all the training samples.

Conclusion :

KNN is a simple yet robust algorithm for classification tasks, relying on the similarity between new and existing data points. Its advantages include ease of implementation and resilience to noisy data, particularly effective with large datasets. Challenges lie in determining the optimal value for K and managing the computational cost associated with distance calculations. Nonetheless, K-NN remains a valuable tool for classification in diverse domains.

Solved Problems :-

Q1] New input Height = 5.5, Age = 38, weight = ?

$x = 5.5, y = 38$

$d = \sqrt{(x - x_1)^2 + (y - y_1)^2}$

1) $x_1 = 5, y_1 = 45$

$d = \sqrt{(5.5 - 5)^2 + (38 - 45)^2} = 1.017$

2) $x_1 = 5.11, y_1 = 26$

$d = \sqrt{(5.5 - 5.11)^2 + (38 - 26)^2} = 12$

3) $x_1 = 5.6, y_2 = 30$

$d = \sqrt{(5.5 - 5.6)^2 + (38 - 30)^2} = 8$

4) $x_1 = 5.9, y_2 = 34$

$d = \sqrt{(5.5 - 5.9)^2 + (38 - 34)^2} = 4.019$

5) $x_1 = 4.8, y_2 = 40$

$d = 2.11$

6) $x_1 = 5.8, y_2 = 36$

$d = 2.022$

7) $x_1 = 5.3, y_2 = 19$

$d = 19$

8) $x_1 = 5.8, y_2 = 28$

$d = 10$

9) $x_1 = 5.5, y_2 = 23$

$d = 15$

10) $x_1 = 5.6, y_2 = 32$

$d = 6$

Considering $k = 3$, the nearest distances are 2.022, 2.11, 4.01 and their weights are 60, 72, 59 respectively.

$\therefore$ weight of new input $= \frac{60 + 72 + 59}{3} = 63.67$

Q2] New input: Height = 170, weight = 57, class = ?

$x = 170, y = 57$

$d = \sqrt{(x - x_1)^2 + (y - y_1)^2}$

1) $x_1 = 167, y_1 = 51$

$d = \sqrt{(170-167)^2 + (57-51)^2} = 6.708$

2) $x_2 = 182, y_2 = 62$

$d = \sqrt{(170-182)^2 + (57-62)^2} = 13$

3) $x_1 = 176, y_2 = 69$

$d = 13.416$

4) $x_1 = 173, y_2 = 64$

$d = 7.61$

5) $x_1 = 172, y_2 = 65$

$d = 8.246$

6) $x_1 = 174, y_2 = 56$

$d = 4.123$

7) $x_1 = 169, y_2 = 58$

$d = 1.414$

8) $x_1 = 173, y_2 = 57$

$d = 3$

9) $x_1 = 170, y_2 = 55$

$d = \sqrt{(170-170)^2 + (55-57)^2} = 2$

Considering $K = 3$, the nearest three distances have class Normal each.

∴ Class of new input = Normal

Also for $K = 1, 2, 4, 5$ the class of new input is Normal

∴ There are 5 Normal & 2 underweight.

NAME:-Preksha Ashok Patel

Sapid:-60004210126

Branch:-Computer Engineering

T.YB-Tech Div:-C2-1

ML

EXPERIMENT NO.6

DATASET 1]

CODE:-

```
import numpy as np

def euclidean_distance(p1, p2):
  return np.sqrt(np.sum((p1 - p2) ** 2))

def knn_predict(data, target, new_point, k):

  distances = np.array([euclidean_distance(new_point, point) for point in data])

  sorted_data_with_distances = np.vstack((data.T, distances)).T
  sorted_data_with_distances = sorted_data_with_distances[sorted_data_with_distances[:, -1].argsort()]

  k_nearest_neighbors = sorted_data_with_distances[:k, :-1]
  k_nearest_targets = target[sorted_data_with_distances[:k, -1].astype(int)]

  predicted_value = np.mean(k_nearest_targets)

  return predicted_value

data = np.array([[5, 45, 77], [5.11, 26, 47], [5.6, 30, 55], [5.9, 34, 59],
                 [4.8, 40, 72], [5.8, 36, 60], [5.3, 19, 40],
                 [5.8, 28, 60], [5.5, 23, 45], [5.6, 32, 58]])

features = data[:, :-1]
target = data[:, -1]

new_point = np.array([5.5, 38])

for k in range(1, 6, 1):

    predicted_weight = knn_predict(features, target, new_point, k)
    print(f"K = {k}, Predicted Weight: {predicted_weight}")
```

OUTPUT:-

```
K = 1, Predicted Weight: 55.0
K = 2, Predicted Weight: 55.0
K = 3, Predicted Weight: 60.666666666666664
K = 4, Predicted Weight: 55.5
K = 5, Predicted Weight: 56.4
```

DATASET 2]

CODE:-

```python
import numpy as np

def euclidean_distance(p1, p2):
  return np.sqrt(np.sum((p1 - p2) ** 2))

def knn_predict(data, target, new_point, k):

  distances = np.array([euclidean_distance(new_point, point) for point in data])
  # Sort data points based on distances
  sorted_data_with_distances = np.vstack((data.T, distances)).T
  sorted_data_with_distances = sorted_data_with_distances[sorted_data_with_distances[:, -1].argsort()]

  # Get the k nearest neighbors
  k_nearest_neighbors = sorted_data_with_distances[:k, :-1]
  k_nearest_targets = target[sorted_data_with_distances[:k, -1].astype(int)]

  # Prediction for regression
  predicted_value = np.mean(k_nearest_targets)

  return predicted_value
data = np.array([
    [167, 51, 0],
    [182, 62, 1],
    [176, 69, 1],
    [173, 64, 1],
    [172, 65, 1],
    [174, 56, 0],
    [169, 58, 1],
    [173, 57, 1],
    [170, 55, 1]])

features = data[:, :-1]
target = data[:, -1]
new_point = np.array([170,57])
for k in range(1, 6, 1):

    predicted_weight = knn_predict(features, target, new_point, k)
    if predicted_weight==1:
      predicted_class="Normal"
    else:
      predicted_class="UnderWeight"
    print(f"K = {k}, Predicted Class: {predicted_class}")
```

OUTPUT:-

```
K = 1, Predicted Class: Normal
K = 2, Predicted Class: Normal
K = 3, Predicted Class: Normal
K = 4, Predicted Class: Normal
K = 5, Predicted Class: Normal
```

DATASET 3]

CODE and OUTPUT:-

```python
import numpy as np
from sklearn.datasets import load_iris

def euclidean_distance(p1, p2):
       return np.sqrt(np.sum((p1 - p2) ** 2))

def knn_predict(data, target, new_point, k):

    distances = np.array([euclidean_distance(new_point, point) for point in data])

    sorted_indices = distances.argsort()
    k_nearest_neighbors = data[sorted_indices[:k]]
    k_nearest_targets = target[sorted_indices[:k]]

       most_frequent_class = np.argmax(np.bincount(k_nearest_targets))
    return most_frequent_class

# Load Iris dataset
iris = load_iris()
data = iris.data
target = iris.target

# New data point for prediction
new_point = np.array([4.8, 3.4, 1.6, 0.2])  # Example Iris data point

for k in range(1, 6, 2):
    predicted_class = knn_predict(data, target, new_point, k)
    predicted_class_name = iris.target_names[predicted_class]
    print(f"K = {k}, Predicted Class: {predicted_class_name}")
```

```
K = 1, Predicted Class: setosa
K = 3, Predicted Class: setosa
K = 5, Predicted Class: setosa
```