BTCOL606: Machine Learning Lab **(Date: 05/05/2023)**

Experiment NO : 08

Instructor: Mr.Katkar Atish R. *Name:* Suyog Rawas, Roll No: CS3257

AIM

Implement and demonstrate the Perceptron Algorithm:

INTRODUCTION TO PERCEPTRON ALGORITHM:

The Perceptron algorithm is a fundamental supervised learning algorithm used for binary classification tasks. It is one of the early artificial neural network models, introduced by Frank Rosenblatt in 1957. The Perceptron algorithm is based on the concept of a single artificial neuron called a perceptron, which is inspired by the functioning of biological neurons.

The goal of the Perceptron algorithm is to find a decision boundary (hyperplane) that separates the input feature space into two regions, corresponding to the different classes. It learns a linear classifier by adjusting the weights and bias term of the perceptron based on labeled training examples.

At its core, the Perceptron algorithm takes a set of input features and computes a weighted sum of these features, which is then passed through an activation function. The activation function determines the output of the perceptron, typically producing a binary output based on a predefined threshold.

During the training process, the Perceptron algorithm iteratively adjusts the weights and bias term of the perceptron based on the prediction errors. If a misclassification occurs, the weights are updated to reduce the error. The update rule involves multiplying the input features by a learning rate and the difference between the predicted and true class labels.

The Perceptron algorithm has some important properties. It can only learn linear decision boundaries and is designed for datasets that are linearly separable. If the training data is linearly separable, the algorithm guarantees convergence. However, if the data is not linearly separable, the algorithm may not converge, and additional techniques or more advanced models may be required.

Despite its limitations, the Perceptron algorithm has historical significance in the field of artificial neural networks. It laid the foundation for the development of more complex models, such as multi-layer perceptrons (MLPs), which can learn non-linear decision boundaries. The concepts and principles of the Perceptron algorithm are still relevant today and have influenced the advancement of neural networks and machine learning algorithms.

In summary, the Perceptron algorithm is a simple yet powerful linear binary classifier. It forms the basis for understanding neural networks and serves as an important stepping stone towards more advanced algorithms for solving classification problems.

The steps involved in the K-means clustering algorithm :

1. Initialization: Initialize the weight vector $w$and bias term $b$to small random values or zeros. The weight vector represents the importance of each feature, and the bias term represents the threshold for activation.

2. Training: Iterate over the training data until convergence or a maximum number of iterations is reached:

- For each training example, compute the weighted sum of the input features and the bias term: $z = \sum_{i=1}^{n} w_i \cdot x_i + b$where x_i is the value of the $i-$th feature
- Apply the activation function (usually a step function) to the weighted sum to obtain the predicted output $y_{\text{pred}} = \text{step}(z)$
- Update the weights and bias term based on the prediction error:

3.Prediction: : Once the weights and bias have been learned during training, the Perceptron can make

predictions for new, unseen data by applying the following steps:
- Compute the weighted sum of the input features and the bias term:$z = \sum_{i=1}^{n} w_i \cdot x_i + b$.
- Apply the activation function to obtain the predicted output: $y_{\text{pred}} = \text{step}(z)$

3. Training: Assess the performance of the Perceptron using evaluation metrics such as accuracy, precision, recall, or F1 score on a separate validation or test dataset.

4.Repeat: If the performance is unsatisfactory, consider adjusting the learning rate, changing the activation function, or exploring other variations of the Perceptron algorithm.

The Perceptron algorithm aims to find a hyperplane that separates the two classes in the feature space. It updates the weights and bias iteratively based on the prediction errors, attempting to minimize the classification errors until convergence or a stopping criterion is met.

DATASET DESCRIPTION :

The Perceptron algorithm is a supervised learning algorithm used for binary classification tasks.When working with the Perceptron algorithm, it is important to have a well-defined dataset that contains labeled examples for training and evaluation. Here is a description of the dataset requirements for the Perceptron algorithm:

IMPLEMENTATION CODE FOR PERCEPTRON ALGORITHM:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
np.random.seed(123)
%matplotlib inline
X, y = make_blobs(n_samples=1000, centers=2)
fig = plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
plt.title("Dataset")
plt.xlabel("First feature")
plt.ylabel("Second feature")
plt.show()
y_true = y[:, np.newaxis]
X_train, X_test, y_train, y_test = train_test_split(X, y_true)
print(f'Shape X_train: {X_train.shape}')
print(f'Shape y_train: {y_train.shape})')
print(f'Shape X_test: {X_test.shape}')
print(f'Shape y_test: {y_test.shape}')
class Perceptron():
    def __init__(self):
        pass
    def train(self, X, y, learning_rate=0.05, n_iters=100):
        n_samples, n_features = X.shape
        # Step 0: Initialize the parameters
        self.weights = np.zeros((n_features,1))
        self.bias = 0

        for i in range(n_iters):
            # Step 1: Compute the activation
            a = np.dot(X, self.weights) + self.bias
            # Step 2: Compute the output
            y_predict = self.step_function(a)
            # Step 3: Compute weight updates
            delta_w = learning_rate * np.dot(X.T, (y - y_predict))
            delta_b = learning_rate * np.sum(y - y_predict)
            # Step 4: Update the parameters
            self.weights += delta_w
            self.bias += delta_b
        return self.weights, self.bias
```

```
42
43      def step_function(self, x):
44          return np.array([1 if elem >= 0 else 0 for elem in x])[:,
45  np.newaxis]
46      def predict(self, X):
47          a = np.dot(X, self.weights) + self.bias
48          return self.step_function(a)
49
50  p = Perceptron()
51  w_trained, b_trained = p.train(X_train, y_train,learning_rate=0.05,
52  n_iters=500)
53  y_p_train = p.predict(X_train)
54  y_p_test = p.predict(X_test)
55  print(f"training accuracy: {100 - np.mean(np.abs(y_p_train - y_train)) *
56  100}%")
57  print(f"test accuracy: {100 - np.mean(np.abs(y_p_test - y_test)) * 100}%")
58  def plot_hyperplane(X, y, weights, bias):
59      """
60      Plots the dataset and the estimated decision hyperplane
61      """
62      slope = - weights[0]/weights[1]
63      intercept = - bias/weights[1]
64      x_hyperplane = np.linspace(-10,10,10)
65      y_hyperplane = slope * x_hyperplane + intercept
66      fig = plt.figure(figsize=(8,6))
67      plt.scatter(X[:,0], X[:,1], c=y)
68      plt.plot(x_hyperplane, y_hyperplane, '-')
69      plt.title("Dataset and fitted decision hyperplane")
70      plt.xlabel("First feature")
71      plt.ylabel("Second feature")
72      plt.show()
73  plot_hyperplane(X, y, w_trained, b_trained)
```

OUTPUT

1 Output Of Perceptron Algorithm:

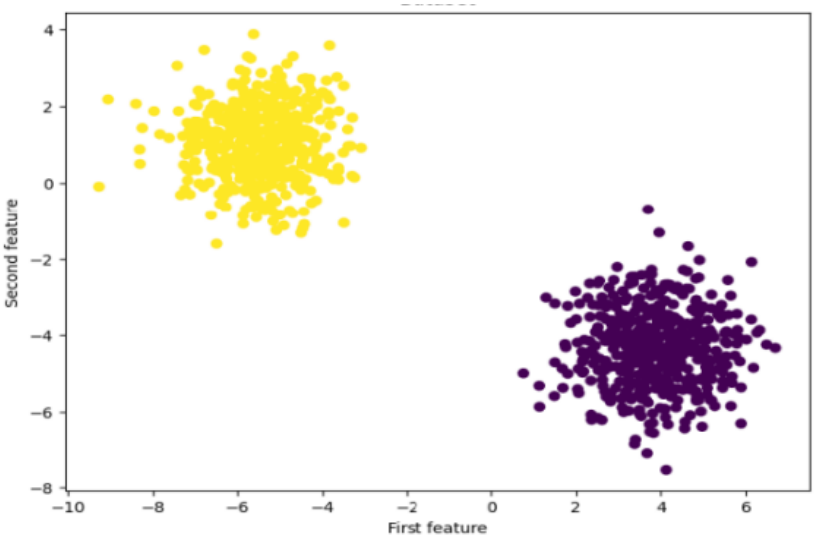

Figure 1: Diffrent and fittered decision hyperplane

2 Diffrent and fittered decision hyperplane

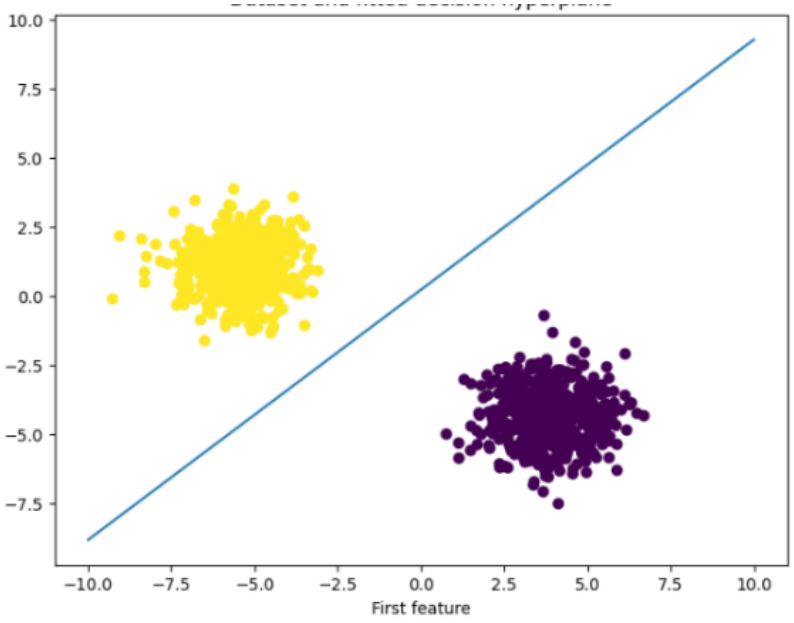

Figure 2: Diffrent and fittered decision hyperplane

CONCLUSION

In conclusion, the Perceptron algorithm is a simple yet powerful linear binary classification algorithm. It is based on the concept of a single artificial neuron called a perceptron, which learns a linear decision boundary to separate data points into two classes. Here are the key points to summarize the Perceptron algorithm