Batch: B - 1 Roll No.: 16014022050

Experiment No. 6

Title : Implementation and Analysis of Perceptron Learning Algorithm

Course Outcome:

CO3: Differentiate between types of machine learning and implement supervised and unsupervised learning

---

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata McGraw Hill Publications**

Introduction:

The perceptron is one of the earliest and simplest supervised learning algorithms for binary classification. It attempts to find a linear decision boundary that separates data points of different classes. The algorithm adjusts its weights iteratively based on misclassified examples until convergence or until a fixed number of epochs.

In this experiment, we use the **Iris dataset** (a popular dataset with 150 flower samples from three species: Setosa, Versicolor, and Virginica). Since the perceptron works only for binary classification, we restrict our analysis to **two classes (Setosa vs. Versicolor)**. We will analyze the effect of varying the number of epochs on model accuracy and visualize the learning process with multiple graphs.

Implementation:

Part 1: Install & Import Libraries

```
[1]
✓ 14s
# Make sure libraries are available
!pip install scikit-learn matplotlib numpy

# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

Part 2: Load the Iris Dataset

```
[2]
✓ 0s
# Load Iris dataset from sklearn
iris = load_iris()

# Use only first 100 samples (Setosa & Versicolor) → binary classification
X = iris.data[:100, :2]  # only first 2 features for easy visualization
y = iris.target[:100]

# Convert labels {0,1} → {-1,1}
y = np.where(y == 0, -1, 1)

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
print("Classes in y:", np.unique(y))
```

```
Shape of X: (100, 2)
Shape of y: (100,)
Classes in y: [-1  1]
```

Part 3: Train-Test Split & Standardization

```
[3]
✓ 0s
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Standardize features (better convergence)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])
```

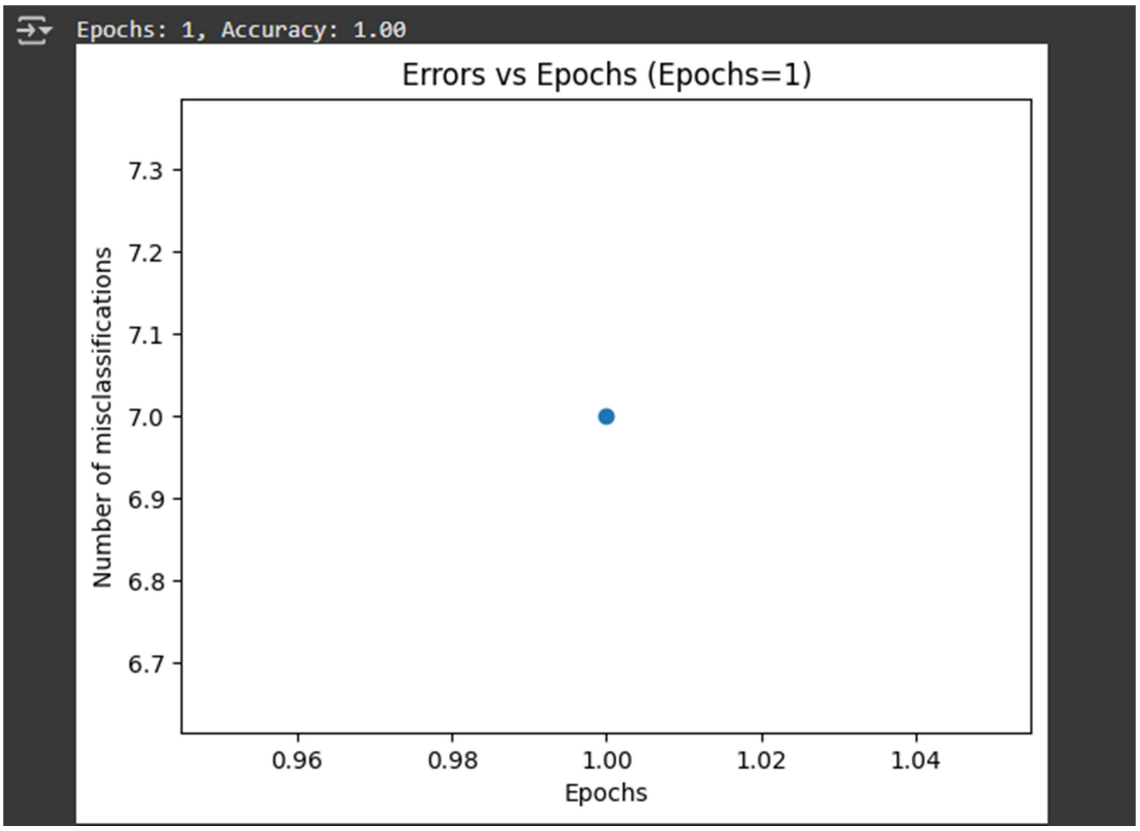
```
Training samples: 80
Testing samples: 20
```

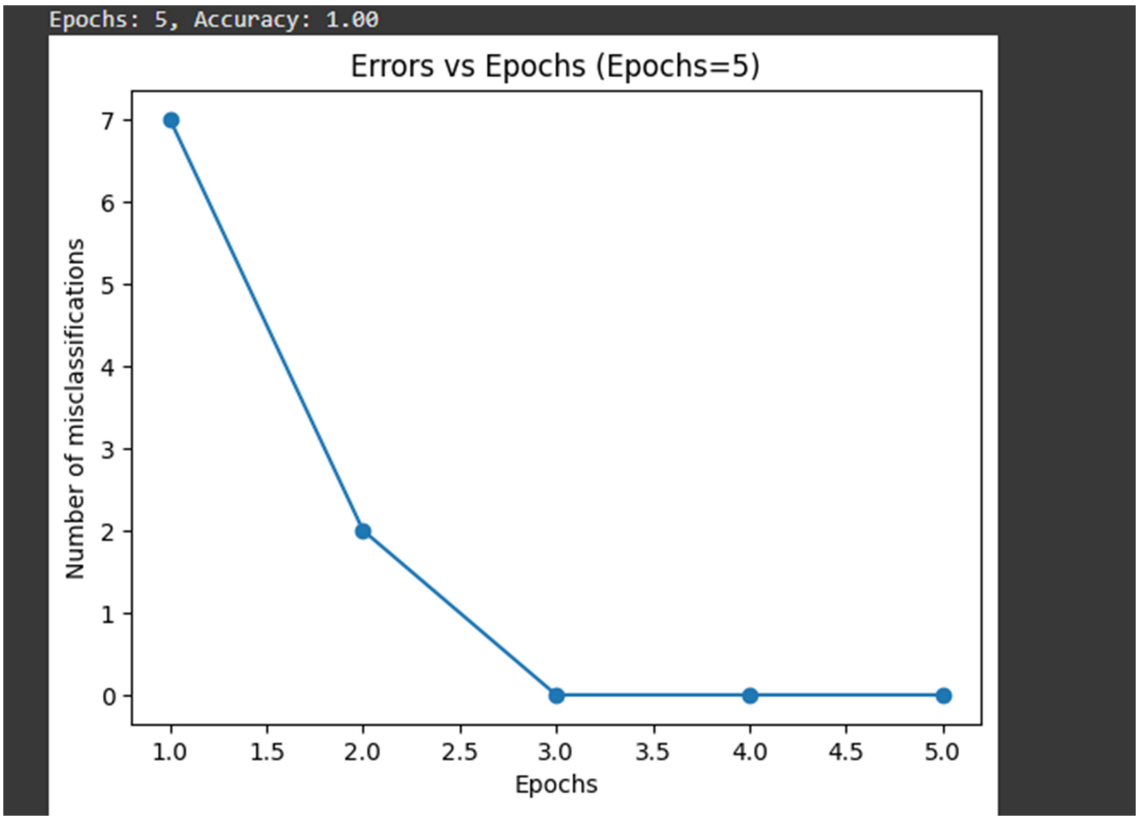
Part 4: Define Perceptron Class

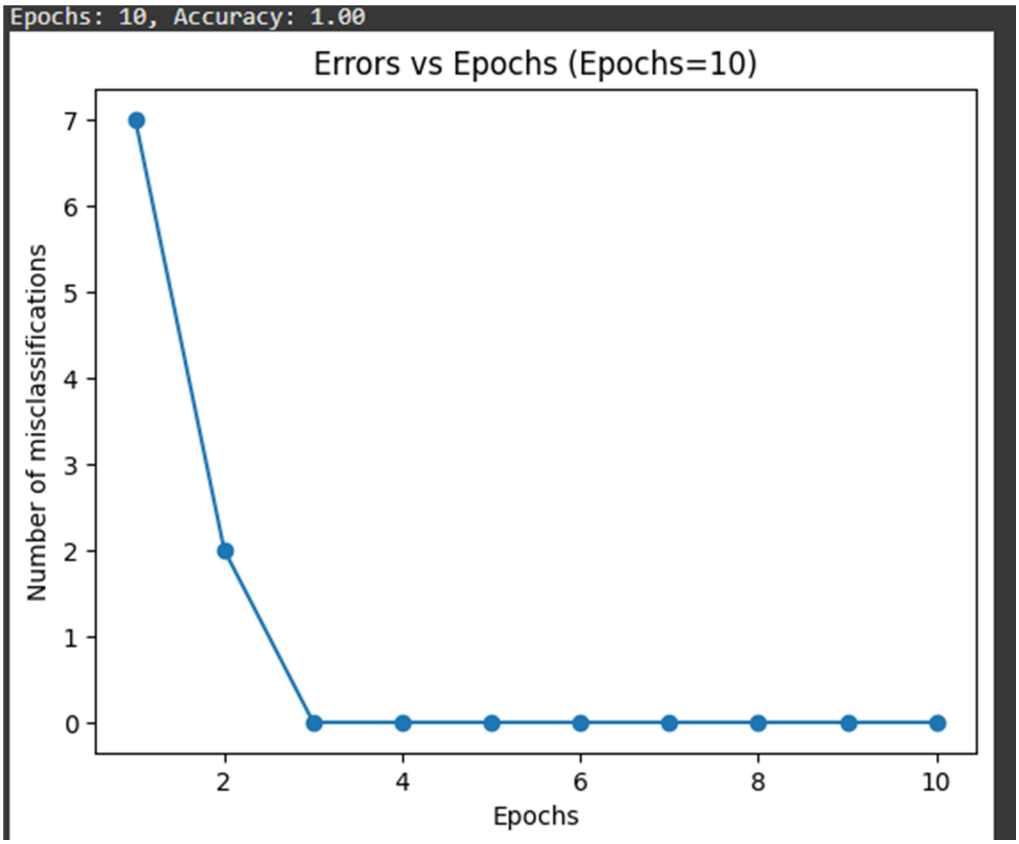
```
[4]
✓ 0s
class Perceptron:
    def __init__(self, learning_rate=0.01, n_epochs=10):
        self.lr = learning_rate
        self.n_epochs = n_epochs
        self.weights = None
        self.bias = 0
        self.errors_ = []

    def fit(self, X, y):
        self.weights = np.zeros(X.shape[1])
        for _ in range(self.n_epochs):
            errors = 0
            for xi, target in zip(X, y):
                update = self.lr * (target - self.predict(xi))
                self.weights += update * xi
                self.bias += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        return np.dot(X, self.weights) + self.bias

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

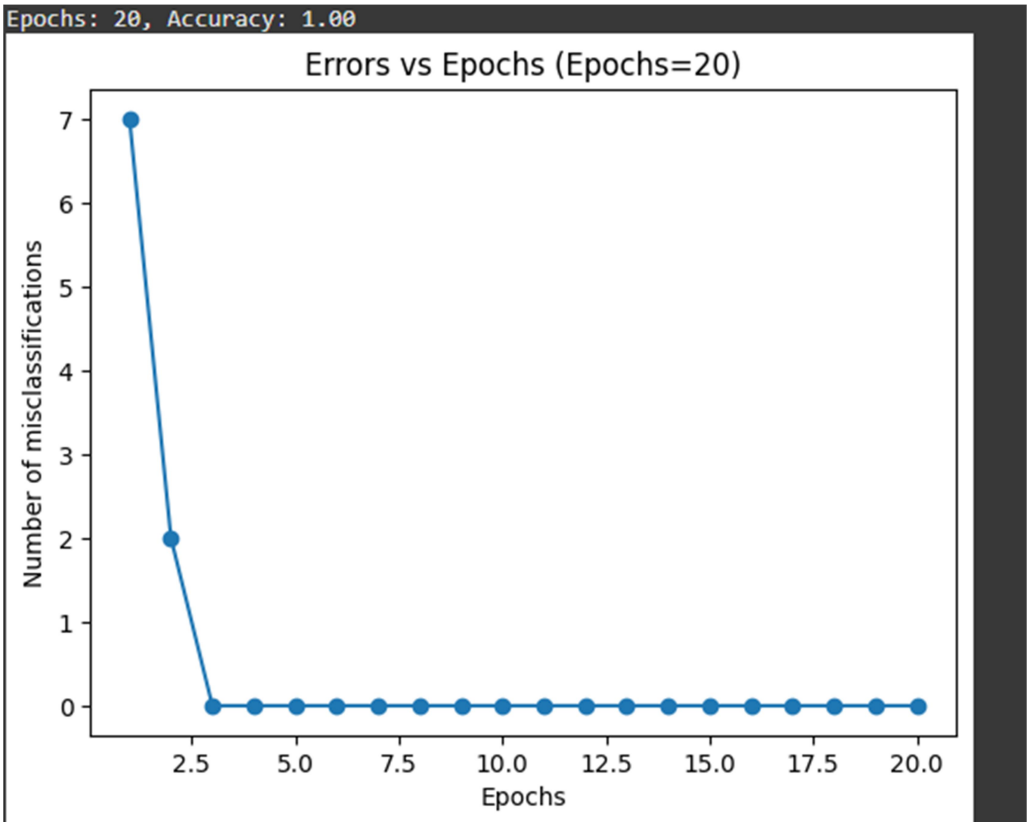
Part 5: Train with Different Epochs & Plot

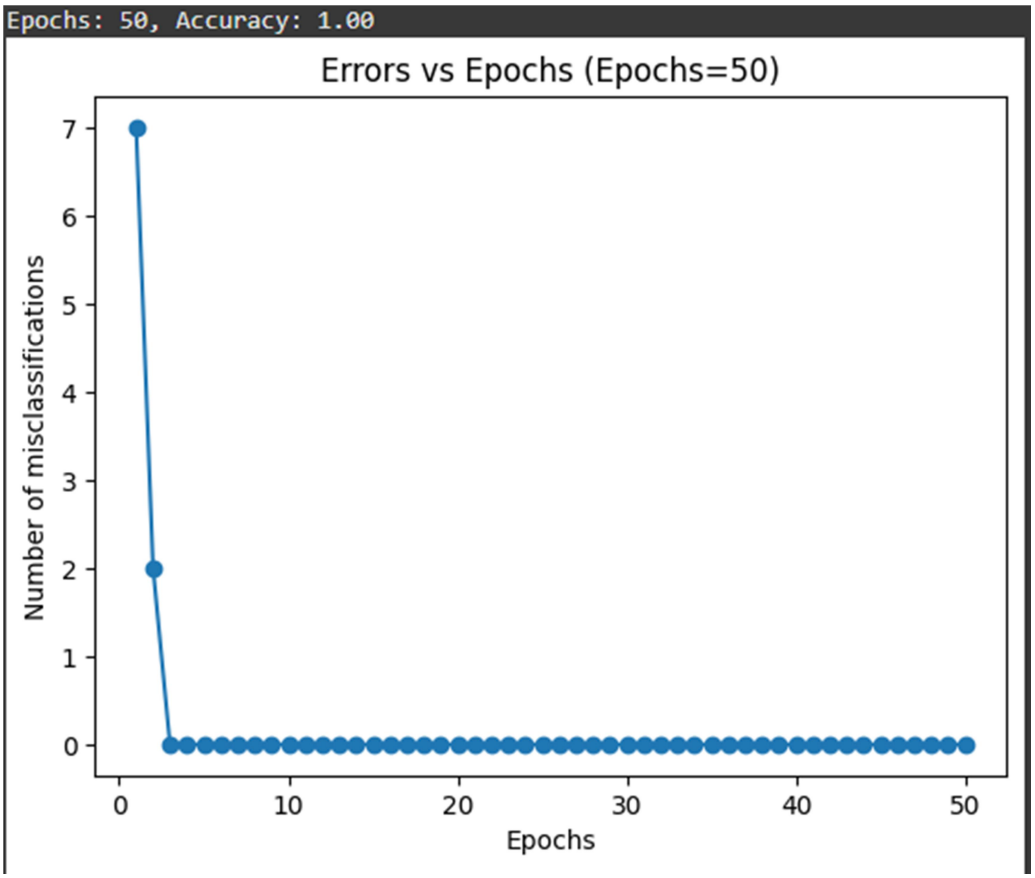
```
[5]
✓ 2s
epoch_list = [1, 5, 10, 20, 50]
accuracies = []

for ep in epoch_list:
    ppn = Perceptron(learning_rate=0.01, n_epochs=ep)
    ppn.fit(X_train, y_train)
    y_pred = ppn.predict(X_test)
    acc = np.mean(y_pred == y_test)
    accuracies.append(acc)
    print(f"Epochs: {ep}, Accuracy: {acc:.2f}")

    # Misclassification plot
    plt.plot(range(1, ep+1), ppn.errors_, marker='o')
    plt.title(f"Errors vs Epochs (Epochs={ep})")
    plt.xlabel("Epochs")
    plt.ylabel("Number of misclassifications")
    plt.show()
```


Epochs: 1, Accuracy: 1.00
Errors vs Epochs (Epochs=1)
Number of misclassifications
7.3
7.2
7.1
7.0
6.9
6.8
6.7
0.96
0.98
1.00
1.02
1.04
Epochs

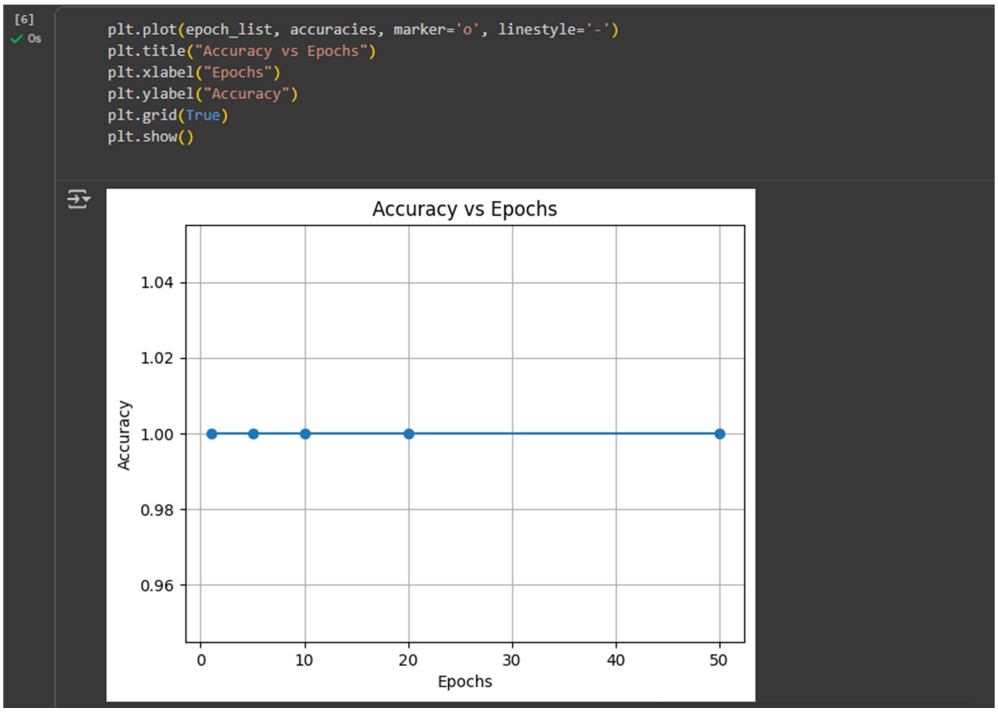

Epochs: 5, Accuracy: 1.00
Errors vs Epochs (Epochs=5)
Number of misclassifications
7
6
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs


Epochs: 10, Accuracy: 1.00
Errors vs Epochs (Epochs=10)
Number of misclassifications
Epochs
0
1
2
3
4
5
6
7
2
4
6
8
10


Epochs: 20, Accuracy: 1.00
Errors vs Epochs (Epochs=20)
Number of misclassifications
Epochs
0
1
2
3
4
5
6
7
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

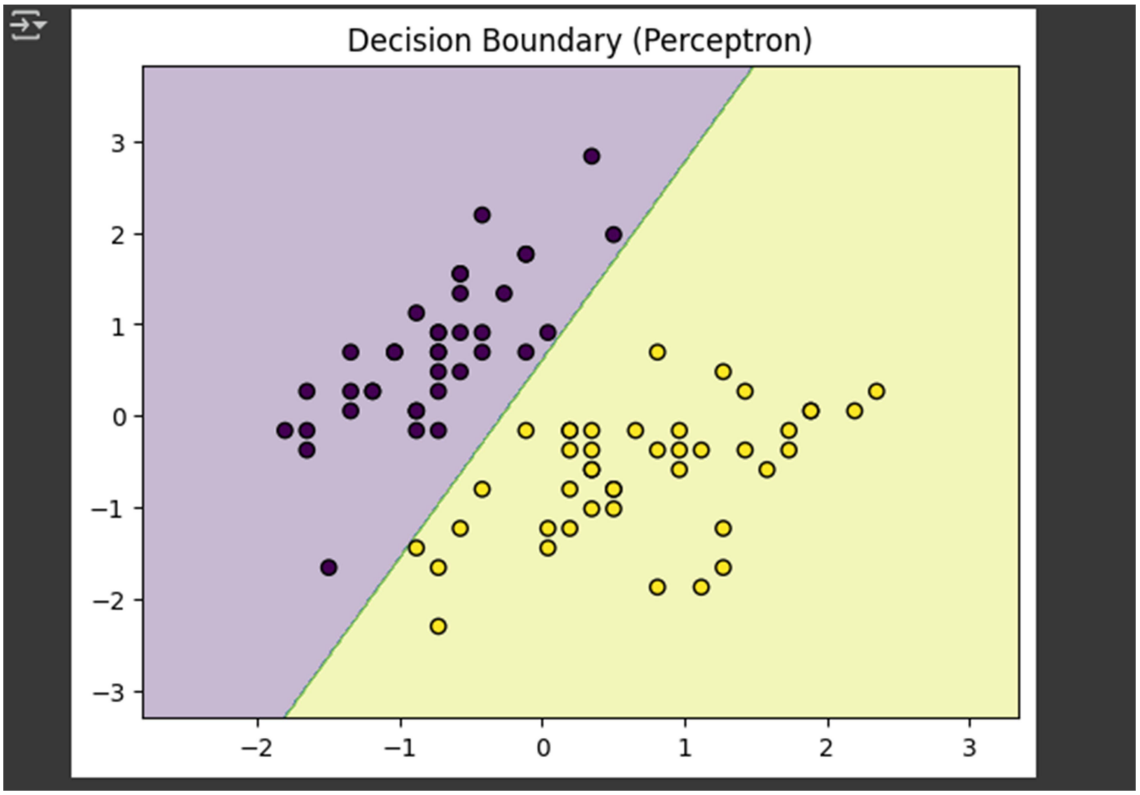


Part 6: Accuracy vs Epochs Graph



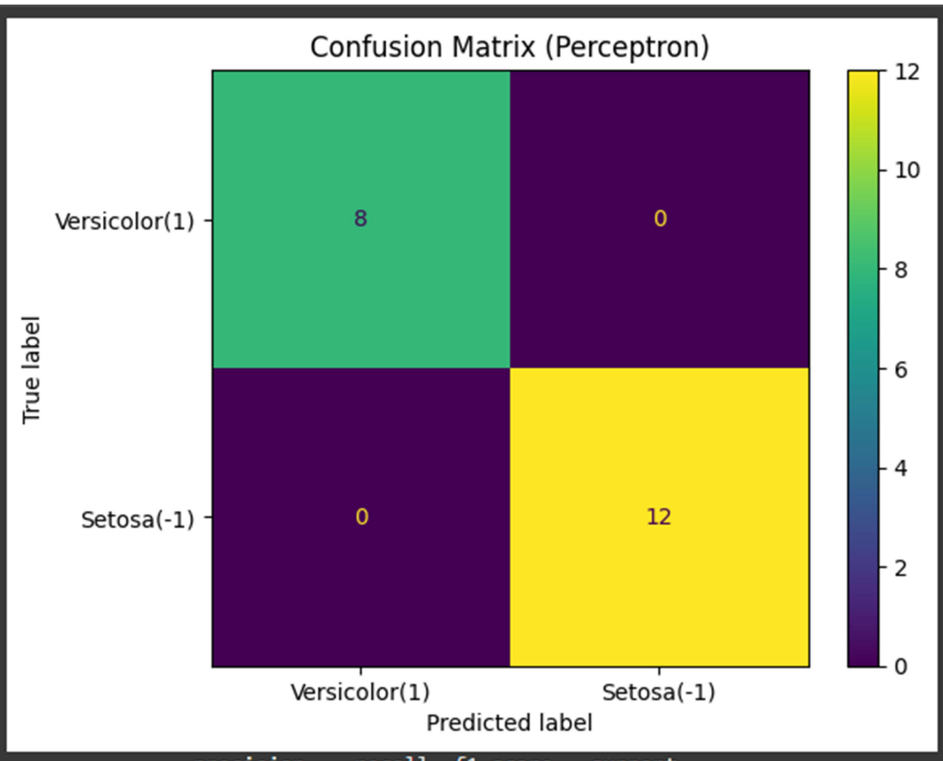
Part 7: Decision Boundary Visualization

```
[7]
✓ 0s
    # Train a final perceptron with 20 epochs
    ppn = Perceptron(learning_rate=0.01, n_epochs=20)
    ppn.fit(X_train, y_train)

    # Create grid
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    Z = ppn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolor='k')
    plt.title("Decision Boundary (Perceptron)")
    plt.show()
```



Part 8: Confusion Matrix & Classification Report

```
[8]
✓ 1s
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay

ppn = Perceptron(learning_rate=0.01, n_epochs=20)
ppn.fit(X_train, y_train)
y_pred = ppn.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=[1,-1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Versicolor(1)','Setosa(-1)'])
disp.plot()
plt.title('Confusion Matrix (Perceptron)')
plt.show()

# Text report
print(classification_report(y_test, y_pred, target_names=['Versicolor(1)','Setosa(-1)']))
```




```
                precision    recall  f1-score   support

Versicolor(1)       1.00      1.00      1.00        12
   Setosa(-1)       1.00      1.00      1.00         8

     accuracy                           1.00        20
    macro avg       1.00      1.00      1.00        20
 weighted avg       1.00      1.00      1.00        20
```

Part 9: Printing accuracy

```
[13]
✓ 0s
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

epoch_list = [1, 5, 10, 20, 50]

for ep in epoch_list:
    ppn = Perceptron(learning_rate=0.01, n_epochs=ep)
    ppn.fit(X_train, y_train)

    # Predictions
    y_train_pred = ppn.predict(X_train)
    y_test_pred = ppn.predict(X_test)

    # Train metrics
    train_acc = accuracy_score(y_train, y_train_pred)
    train_prec = precision_score(y_train, y_train_pred, pos_label=1)
    train_rec = recall_score(y_train, y_train_pred, pos_label=1)
    train_f1 = f1_score(y_train, y_train_pred, pos_label=1)

    # Test metrics
    test_acc = accuracy_score(y_test, y_test_pred)
    test_prec = precision_score(y_test, y_test_pred, pos_label=1)
    test_rec = recall_score(y_test, y_test_pred, pos_label=1)
    test_f1 = f1_score(y_test, y_test_pred, pos_label=1)

    # Print nicely
    print(f"\nEpochs: {ep}")
    print(f" Train -> Accuracy: {train_acc:.2f}, Precision: {train_prec:.2f}, Recall: {train_rec:.2f}, F1: {train_f1:.2f}")
    print(f" Test  -> Accuracy: {test_acc:.2f}, Precision: {test_prec:.2f}, Recall: {test_rec:.2f}, F1: {test_f1:.2f}")
```

```
Epochs: 1
 Train -> Accuracy: 0.99, Precision: 0.98, Recall: 1.00, F1: 0.99
 Test  -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00

Epochs: 5
 Train -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00
 Test  -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00

Epochs: 10
 Train -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00
 Test  -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00

Epochs: 20
 Train -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00
 Test  -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00

Epochs: 50
 Train -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00
 Test  -> Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1: 1.00
```

Post Lab Descriptive Questions:

1. **What is the perceptron learning rule, and how does it update weights?**
   The perceptron learning rule updates weights based on classification errors. If a sample is misclassified, the weight vector is adjusted in the direction of the input.
   Update rule:

$$w_{new} = w_{old} + \eta(y - \hat{y})x$$

$$b_{new} = b_{old} + \eta(y - \hat{y})$$

   where:

   - η = learning rate
   - y = true label
   - $\hat{y}$ = predicted label

2. **Explain the role of the learning rate in the perceptron algorithm. What happens if it is too high or too low?**
   - **Too high** → weights change drastically, algorithm may oscillate and fail to converge.
   - **Too low** → learning is very slow, requiring many epochs to reach a good solution.
   - Proper learning rate ensures faster convergence with stable results.

3. **In real-world applications (e.g., spam detection, medical diagnosis), where could a perceptron be applied successfully?**
   Perceptrons can be applied in:
   - **Spam detection** → classify emails as spam/ham.
   - **Medical diagnosis** → classify whether a tumor is malignant/benign.
   - **Sentiment analysis** → classify text as positive/negative.
   - **Fraud detection** → classify transactions as fraudulent or normal.