

```
# Importing Iris dataset from sklearn
from sklearn.datasets import load_iris
```

```
# Load Iris dataset
iris = load_iris()
X = iris.data # Features
y = iris.target # Target labels
```

```
print("Features shape:", X.shape)
print("Target shape:", y.shape)
```

```
→ Features shape: (150, 4)
   Target shape: (150,)
```

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Load Iris dataset
iris = load_iris()
X = iris.data # Features (sepal length, sepal width, petal length, petal width)
y = iris.target # Labels (0=Setosa, 1=Versicolor, 2=Virginica)
```

```
# For Binary Classification (Setosa vs Versicolor)
X_binary = X[y != 2] # Only Setosa and Versicolor
y_binary = y[y != 2]
y_binary = np.where(y_binary == 0, 0, 1) # 0 for Setosa, 1 for Versicolor
```

```
# For Multi-Class Classification (Setosa, Versicolor, Virginica)
X_multi = X
y_multi = y
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_binary, y_binary, test_size=0.3, random_state=42) # Binary case
X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(X_multi, y_multi, test_size=0.3, random_state=42) # Multi-cl
```

```
# Binary Step activation function
def binary_step(x):
    return np.where(x >= 0, 1, 0)
```

```
# Bipolar Step activation function (-1 or 1)
def bipolar_step(x):
    return np.where(x >= 0, 1, -1)
```

```
# Perceptron learning rule for Binary Classification (Binary Step)
def perceptron_binary(X_train, y_train, epochs=1000, learning_rate=0.1):
    weights = np.zeros(X_train.shape[1]) # Initialize weights to zeros
    bias = 0 # Initialize bias to 0

    for epoch in range(epochs):
        for i in range(X_train.shape[0]):
            linear_output = np.dot(X_train[i], weights) + bias
            prediction = binary_step(linear_output) # Apply binary step activation
            error = y_train[i] - prediction
            weights += learning_rate * error * X_train[i] # Update weights
            bias += learning_rate * error # Update bias

    return weights, bias
```

```
# Perceptron learning rule for Binary Classification (Bipolar Step)
def perceptron_bipolar(X_train, y_train, epochs=1000, learning_rate=0.1):
    weights = np.zeros(X_train.shape[1]) # Initialize weights to zeros
    bias = 0 # Initialize bias to 0

    # Convert y_train to bipolar labels (-1, 1)
    y_train_bipolar = np.where(y_train == 0, -1, 1)

    for epoch in range(epochs):
        for i in range(X_train.shape[0]):
            linear_output = np.dot(X_train[i], weights) + bias
            prediction = bipolar_step(linear_output) # Apply bipolar step activation
            error = y_train_bipolar[i] - prediction
            weights += learning_rate * error * X_train[i] # Update weights
            bias += learning_rate * error # Update bias

    return weights, bias
```

```

# Perceptron learning rule for Multi-Class Classification (One-vs-Rest)
def perceptron_one_vs_rest(X_train, y_train, epochs=1000, learning_rate=0.1, num_classes=3):
    weights = np.zeros((num_classes, X_train.shape[1])) # Weights for each class
    biases = np.zeros(num_classes) # Biases for each class

    for epoch in range(epochs):
        for i in range(X_train.shape[0]):
            x_i = X_train[i]
            y_i = y_train[i]

            for c in range(num_classes):
                output = np.dot(x_i, weights[c]) + biases[c]
                prediction = 1 if output >= 0 else 0 # Binary decision
                target = 1 if y_i == c else 0 # Target: 1 for current class, 0 otherwise
                error = target - prediction
                weights[c] += learning_rate * error * x_i # Update weights for class c
                biases[c] += learning_rate * error # Update bias for class c

    return weights, biases

# Train the perceptron for binary classification using binary step
weights_binary, bias_binary = perceptron_binary(X_train, y_train)

# Train the perceptron for binary classification using bipolar step
weights_bipolar, bias_bipolar = perceptron_bipolar(X_train, y_train)

# Train the perceptron for multi-class classification (One-vs-Rest)
weights_multi, biases_multi = perceptron_one_vs_rest(X_train_multi, y_train_multi, num_classes=3)

# Evaluate the binary perceptron model (binary step)
def evaluate_binary(X_test, y_test, weights, bias, activation_fn):
    if activation_fn == "binary":
        predictions = binary_step(np.dot(X_test, weights) + bias)
    elif activation_fn == "bipolar":
        y_test_bipolar = np.where(y_test == 0, -1, 1)
        predictions = bipolar_step(np.dot(X_test, weights) + bias)
    accuracy = accuracy_score(y_test, predictions)
    return accuracy

accuracy_binary = evaluate_binary(X_test, y_test, weights_binary, bias_binary, "binary")
accuracy_bipolar = evaluate_binary(X_test, y_test, weights_bipolar, bias_bipolar, "bipolar")

# Evaluate the multi-class perceptron model (One-vs-Rest)
def evaluate_multi(X_test, y_test, weights, biases):
    predictions = []
    for i in range(X_test.shape[0]):
        outputs = np.dot(X_test[i], weights.T) + biases # Outputs for each class
        prediction = np.argmax(outputs) # Class with the highest output
        predictions.append(prediction)
    accuracy = accuracy_score(y_test, predictions)
    return accuracy


accuracy_multi = evaluate_multi(X_test_multi, y_test_multi, weights_multi, biases_multi)

# Display results
print(f"Binary Perceptron Accuracy (Binary Step Activation): {accuracy_binary * 100:.2f}%")
print("Final Weights (Binary Step):", weights_binary)
print("Final Bias (Binary Step):", bias_binary)

print(f"Binary Perceptron Accuracy (Bipolar Step Activation): {accuracy_bipolar * 100:.2f}%")
print("Final Weights (Bipolar Step):", weights_bipolar)
print("Final Bias (Bipolar Step):", bias_bipolar)

print(f"Multi-Class Perceptron Accuracy (One-vs-Rest): {accuracy_multi * 100:.2f}%")
print("Final Weights (One-vs-Rest):")
print(weights_multi)
print("Final Biases (One-vs-Rest):")
print(biases_multi)

```

 Binary Perceptron Accuracy (Binary Step Activation): 100.00%
 Final Weights (Binary Step): [-0.34 -0.73 1.06 0.51]
 Final Bias (Binary Step): -0.1
 Binary Perceptron Accuracy (Bipolar Step Activation): 43.33%
 Final Weights (Bipolar Step): [-0.68 -1.46 2.12 1.02]
 Final Bias (Bipolar Step): -0.2
 Multi-Class Perceptron Accuracy (One-vs-Rest): 75.56%
 Final Weights (One-vs-Rest):
 [[0.23 0.81 -1.09 -0.46]
 [-0.42 -7.25 2.95 -10.48]
 [-5.23 -16.85 13.06 17.96]]
 Final Biases (One-vs-Rest):
 [0.1 20.3 -10.3]

