Single Layer Perceptron

AIM:

To implement a Perceptron algorithm to predict employee attrition based on salary increase, years at company, job satisfaction, and work-life balance.

ALGORITHM:

Step 1: Create a dataset with employee attributes and attrition labels.
Step 2: Normalize the feature values using standard scaling.
Step 3: Split the dataset into training and testing sets.
Step 4: Initialize the weights and bias to zero.
Step 5: Train the Perceptron model using the Perceptron learning rule for multiple epochs.
Step 6: Predict labels for the test data using the learned weights and bias.
Step 7: Evaluate the model using accuracy, precision, recall, and F1-score.
Step 8: Plot the decision boundary using the first two features.
Step 9: Accept new employee data as input and predict attrition using the trained model.

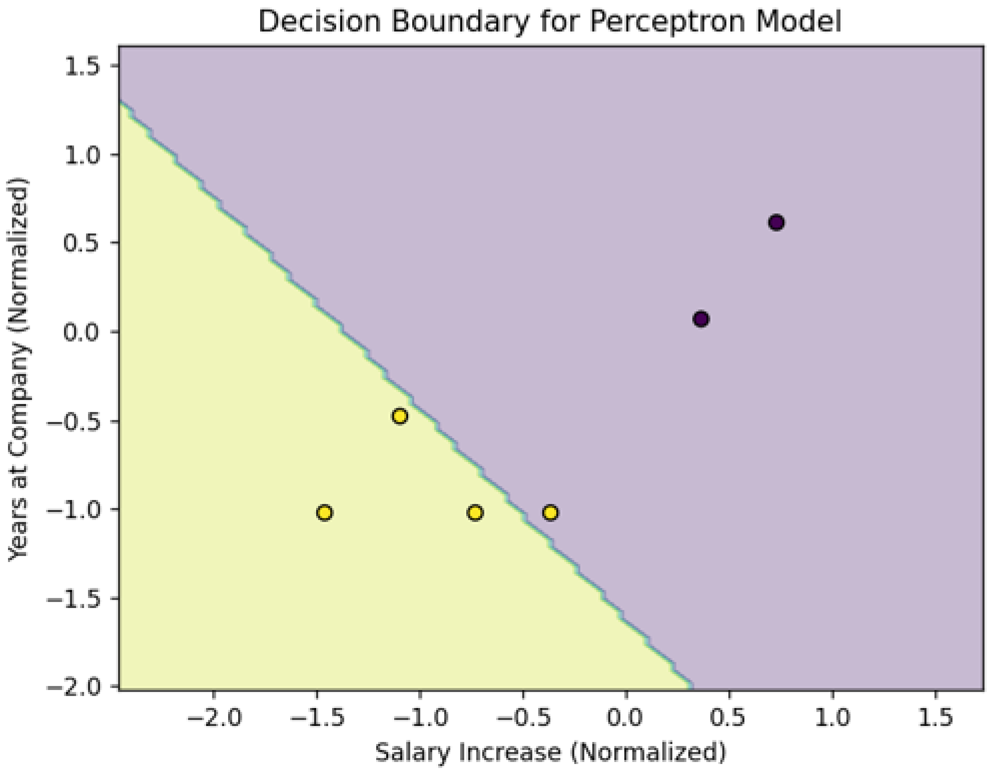
SOURCE CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Step 1: Create a Sample Dataset (Salary Increase, Years at Company, Job Satisfaction, Work-
Life Balance, Attrition)
data = pd.DataFrame({
    'Salary Increase': [5, 10, 2, 7, 3, 9, 4, 8],
    'Years at Company': [1, 5, 1, 3, 2, 6, 1, 4],
    'Job Satisfaction': [2, 4, 1, 3, 2, 5, 3, 4],
    'Work-Life Balance': [2, 4, 1, 3, 2, 5, 2, 4],
    'Attrition': [1, 0, 1, 0, 1, 0, 1, 0]})

X = data.iloc[:, :-1].values  # Features (Salary Increase, Years at Company, Job Satisfaction,
Work-Life Balance)
```

```
y = data.iloc[:, -1].values   # Labels (Attrition: 1 = Leave, 0 = Stay)

# Step 2: Normalize the Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Split into Training and Testing Data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 4: Initialize Parameters
learning_rate = 0.1
epochs = 10
n_samples, n_features = X_train.shape
weights = np.zeros(n_features)
bias = 0

def activation(x):
    return 1 if x >= 0 else 0

# Step 5: Train the Perceptron Model
for _ in range(epochs):
    for i in range(n_samples):
        linear_output = np.dot(X_train[i], weights) + bias
        y_pred = activation(linear_output)

        # Perceptron Learning Rule
        update = learning_rate * (y_train[i] - y_pred)
        weights += update * X_train[i]
        bias += update

# Step 6: Test the Model
def predict(X):
    linear_output = np.dot(X, weights) + bias
    return np.array([activation(x) for x in linear_output])

y_pred = predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
# Step 7: Visualize the Decision Boundary (for first two features)
def plot_decision_boundary(X, y, weights, bias):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    Z = predict(np.c_[xx.ravel(), yy.ravel(), np.zeros_like(xx.ravel()),
np.zeros_like(xx.ravel())])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.xlabel("Salary Increase (Normalized)")
    plt.ylabel("Years at Company (Normalized)")
    plt.title("Decision Boundary for Perceptron Model")
    plt.show()

plot_decision_boundary(X_train, y_train, weights, bias)

# Step 8: Take User Input for Prediction
print("Enter details for a new employee:")
salary_increase = float(input("Salary Increase (%): "))
years_at_company = float(input("Years at Company: "))
job_satisfaction = float(input("Job Satisfaction (1-5): "))
work_life_balance = float(input("Work-Life Balance (1-5): "))

new_employee = np.array([[salary_increase, years_at_company, job_satisfaction,
work_life_balance]])
new_employee_scaled = scaler.transform(new_employee)
prediction = predict(new_employee_scaled)

if prediction[0] == 1:
    print("Prediction: Employee is likely to leave.")
else:
    print("Prediction: Employee is likely to stay.")
```

OUTPUT:



RESULT:

RESULT:

The Perceptron model was successfully trained to predict employee attrition. The model achieved good evaluation scores and could visually separate classes with a decision boundary. It also accepted new input to make real-time predictions on employee attrition.