Machine Learning Programs with Sample Output

Program 1: Random Forest

Aim:

To build a classification model using Random Forest algorithm.

Algorithm:

1. Load dataset.

2. Split data.

3. Train RandomForestClassifier.

4. Predict and display output.

Explanation:

Random Forest uses multiple decision trees and combines their results for better accuracy.

Code:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
model = RandomForestClassifier(n_estimators=10).fit(X, y)
print("Prediction:", model.predict([X[0]]))
```

Sample Output:

```
Random Forest Prediction: [0] (for first iris sample)
```

Program 2: Simple Linear Regression

Aim:

To implement simple linear regression using scikit-learn.

Algorithm:

1. Prepare input and output data.

2. Train LinearRegression model.

3. Predict and print result.

Explanation:

This model learns the best fit line y = wx + b to minimize error.

Code:

```
from sklearn.linear_model import LinearRegression
X = [[1], [2], [3]]
y = [2, 4, 6]
model = LinearRegression().fit(X, y)
```

```
print("Prediction:", model.predict([[4]])[0])
```

Sample Output:

```
Prediction: 8.0 (since model learns y = 2x)
```

Program 3: Multiple Linear Regression

Aim:

To perform regression using multiple input variables.

Algorithm:

1. Prepare multi-dimensional input X and target y.
2. Train LinearRegression.
3. Predict outcome.

Explanation:

It fits a linear equation to predict output using multiple features.

Code:

```
X = [[1, 2], [2, 3], [3, 4]]
y = [6, 9, 12]
model = LinearRegression().fit(X, y)
print("Prediction:", model.predict([[4, 5]])[0])
```

Sample Output:

```
Prediction: 15.0 (model learns weights for both features)
```

Program 4: Logistic Regression

Aim:

To classify binary data using logistic regression.

Algorithm:

1. Prepare binary labeled data.
2. Train LogisticRegression.
3. Predict and print result.

Explanation:

Logistic Regression outputs probabilities and maps them to 0 or 1 for classification.

Code:

```
from sklearn.linear_model import LogisticRegression
X = [[1], [2], [3], [4]]
y = [0, 0, 1, 1]
model = LogisticRegression().fit(X, y)
```

```
print("Prediction:", model.predict([[2.5]])[0])
```

Sample Output:

```
Prediction: 1 or 0 depending on sigmoid output at 2.5
```

Program 5: Support Vector Machine

Aim:

To classify data using SVM with default kernel.

Algorithm:

1. Create labeled data.
2. Train SVC.
3. Predict using model.

Explanation:

SVM finds the hyperplane that best separates the classes.

Code:

```
from sklearn.svm import SVC
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
model = SVC().fit(X, y)
print("Prediction:", model.predict([[1.5]])[0])
```

Sample Output:

```
Prediction: 0 (since 1.5 is closer to class 0)
```

Program 6: Maximum Margin Classifier

Aim:

To find the max margin hyperplane using linear SVM.

Algorithm:

1. Use SVC with linear kernel.
2. Print coefficients and intercept.

Explanation:

A linear SVM tries to maximize margin between classes.

Code:

```
model = SVC(kernel='linear').fit(X, y)
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

Sample Output:

```
Coefficients and intercept represent max margin hyperplane.
```

Program 7: Gradient Descent

Aim:

To manually implement gradient descent for a linear model.

Algorithm:

1. Initialize weight.

2. Calculate gradient.

3. Update weight iteratively.

Explanation:

Gradient descent optimizes weights to minimize error.

Code:

```
x = [1, 2, 3]
y = [2, 4, 6]
w, lr = 0, 0.01
for _ in range(100):
 grad = sum((w*x[i] - y[i])*x[i] for i in range(3)) / 3
 w -= lr * grad
print("Weight:", round(w, 2))
```

Sample Output:

```
Final weight converges to ~2.0 showing correct slope fit.
```

Program 8: Perceptron Algorithm

Aim:

To classify logic data using perceptron.

Algorithm:

1. Prepare input and target.

2. Train Perceptron.

3. Predict on test input.

Explanation:

Perceptron learns weights for linear classification.

Code:

```
from sklearn.linear_model import Perceptron
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0, 0, 0, 1]
model = Perceptron().fit(X, y)
print("Prediction:", model.predict([[1, 1]])[0])
```

Sample Output:

```
Prediction: 1 (AND logic - perceptron learns [1,1] -> 1)
```

Program 9: Locally Weighted Regression

Aim:

To implement a basic LWR with exponential weights.

Algorithm:

1. Define query point.

2. Calculate weights.

3. Compute prediction.

Explanation:

LWR assigns higher weights to nearby points for local fitting.

Code:

```
import numpy as np
X = np.array([1,2,3])
y = np.array([2,4,6])
query = 2.5
tau = 1.0
weights = np.exp(-((X - query)**2)/(2*tau**2))
theta = sum(weights*X*y)/sum(weights*X*X)
print("Prediction:", round(theta*query,2))
```

Sample Output:

```
Prediction: 5.0 (weighted average near query = 2.5)
```