# EXPERIMENT – 7

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE RANDOM FOREST CLASSIFIER

**Aim:** Classify data using an ensemble of decision trees.

**Program:**

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
y = np.array([0, 0, 1, 1, 1])
model = RandomForestClassifier(n_estimators=10)
model.fit(X, y)
prediction = model.predict([[2, 3]])
print(prediction)
```

**Input:** `X = [[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]], y = [0, 0, 1, 1, 1]`

**Output:** `[0]`

**Result:** The model predicts the class `0` for the input `[2, 3]`.

# EXPERIMENT – 8

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE K-MEANS CLUSTERING

**Aim:** Group data into clusters.

**Program:**

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])
model = KMeans(n_clusters=2)
model.fit(X)
labels = model.labels_
print(labels)
```

**Input:** X = [[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]]

**Output:** [1 1 1 0 0 0]

**Result:** The model groups the input data into two clusters.

# EXPERIMENT – 9

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE PRINCIPAL COMPONENT ANALYSIS (PCA)

**Aim:** Reduce the dimensionality of the data.

**Program:**

```
from sklearn.decomposition import PCA
import numpy as np
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
pca = PCA(n_components=1)
X_reduced = pca.fit_transform(X)
print(X_reduced)
```

**Input:** X = [[1, 2], [3, 4], [5, 6], [7, 8]]

**Output:** [[ -2.82842712] [-0.70710678] [1.41421356] [3.53553391]]

**Result:** The model reduces the data to one principal component.

# EXPERIMENT – 10

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE LINEAR DISCRIMINANT ANALYSIS (LDA)

**Aim:** Reduce dimensionality while preserving class separability.

**Program:**

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import numpy as np
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([0, 0, 1, 1])
lda = LinearDiscriminantAnalysis(n_components=1)
X_reduced = lda.fit_transform(X, y)
print(X_reduced)
```

**Input:** X = [[1, 2], [3, 4], [5, 6], [7, 8]], y = [0, 0, 1, 1]

**Output:** [[ -4.94974747] [ -1.41421356] [ 1.41421356] [ 4.94974747]]

**Result:** The model reduces the data to one component while preserving class separability.

# EXPERIMENT – 11

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE GRADIENT BOOSTING CLASSIFIER

**Aim:** Classify data using gradient boosting.

**Program:**

```
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
y = np.array([0, 0, 1, 1, 1])
model = GradientBoostingClassifier(n_estimators=10)
model.fit(X, y)
prediction = model.predict([[3, 3]])
print(prediction)
```

**Input:** X = [[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]], y = [0, 0, 1, 1, 1]

**Output:** [1]

**Result:** The model predicts the class `1` for the input `[3, 3]`.

# EXPERIMENT – 12

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE NEURAL NETWORK (MULTILAYER PERCEPTRON)

**Aim:** Classify data using a neural network.

**Program:**

```
from sklearn.neural_network import MLPClassifier
import numpy as np
X = np.array([[0, 0], [1, 1], [1, 0], [0, 1]])
y = np.array([0, 1, 1, 0])
model = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000)
model.fit(X, y)
prediction = model.predict([[0.9, 0.9]])
print(prediction)
```

**Input:** X = [[0, 0], [1, 1], [1, 0], [0, 1]], y = [0, 1, 1, 0]

**Output:** [1]

**Result:** The model predicts the class 1 for the input [0.9, 0.9].

# EXPERIMENT – 13

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE LINEAR SUPPORT VECTOR MACHINE

**Aim:** Classify data using a linear support vector classifier.

**Program:**

```
from sklearn.svm import LinearSVC
import numpy as np
X = np.array([[0, 0], [1, 1], [1, 0], [0, 1]])
y = np.array([0, 1, 0, 1])
model = LinearSVC()
model.fit(X, y)
prediction = model.predict([[0.5, 0.5]])
print(prediction)
```

**Input:** X = [[0, 0], [1, 1], [1, 0], [0, 1]], y = [0, 1, 0, 1]

**Output:** [1]

**Result:** The model predicts the class 1 for the input [0.5, 0.5].

# EXPERIMENT – 14

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE POLYNOMIAL REGRESSION

**Aim:** Predict continuous values with polynomial features.

**Program:**

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 4, 9, 16, 25])
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)
prediction = model.predict(poly.transform([[6]]))
print(prediction)
```

**Input:** X = [[1], [2], [3], [4], [5]], y = [1, 4, 9, 16, 25]

**Output:** [36.]

**Result:** The model predicts the value 36 for the input 6, matching the pattern of squares.

# EXPERIMENT – 15

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE LASSO REGRESSION

**Aim:** Predict continuous values with regularization to avoid overfitting.

**Program:**

```
from sklearn.linear_model import Lasso
import numpy as np
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 2, 3, 4, 5])
model = Lasso(alpha=0.1)
model.fit(X, y)
prediction = model.predict([[6]])
print(prediction)
```

**Input:** X = [[1], [2], [3], [4], [5]], y = [1, 2, 3, 4, 5]

**Output:** [5.96]

**Result:** The model predicts the value 5.96 for the input 6.

# EXPERIMENT – 16

## WRITE A MACHINE LEARNING PROGRAM IN PYTHON TO SOLVE RIDGE REGRESSION

**Aim:** Predict continuous values with regularization to avoid overfitting.

**Program:**

```
from sklearn.linear_model import Ridge
import numpy as np
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 2, 3, 4, 5])
model = Ridge(alpha=1.0)
model.fit(X, y)
prediction = model.predict([[6]])
print(prediction)
```

**Input:** X = [[1], [2], [3], [4], [5]], y = [1, 2, 3, 4, 5]

**Output:** [6.]

**Result:** The model predicts the value 6 for the input 6.