PROGRAM 2 **LINEAR AND POLYNOMIAL REGRESSION**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Load dataset
data = pd.read_csv('./Position_Salaries.csv')
X, y = data.iloc[:, 1:2].values, data.iloc[:, 2].values
print(X,y)
# Linear Regression
lin_reg = LinearRegression().fit(X, y)

# Polynomial Regression
poly_features = PolynomialFeatures(degree=3)
X_poly = poly_features.fit_transform(X)
print(X_poly)
poly_reg = LinearRegression().fit(X_poly, y)

# Plot Linear Regression
plt.scatter(X, y, color="blue")
plt.plot(X, lin_reg.predict(X), color="red")
plt.title("Bluff Detection Model (Linear Regression)")
plt.xlabel("Position Levels")
plt.ylabel("Salary")
plt.show()

# Plot Polynomial Regression
plt.scatter(X, y, color="blue")
plt.plot(X, poly_reg.predict(poly_features.transform(X)), color="red")
plt.title("Bluff Detection Model (Polynomial Regression)")
plt.xlabel("Position Levels")
plt.ylabel("Salary")
plt.show()

# Predictions
print("Linear Prediction for Level 11:", lin_reg.predict([[11]]))
print("Polynomial Prediction for Level 11:", poly_reg.predict(poly_features.transform([[11]])))
```

PROGRAM 3 LOGISTIC REGRESSION

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

# Load dataset
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv('./pima-indians-diabetes.csv', names=col_names)

# Split dataset into features and target variable
X = pima.drop('label', axis=1)
y = pima['label']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)

# Instantiate and fit the model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluation
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
accuracy = metrics.accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("Predictions vs Actual:\n", pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}))
print(f"Accuracy: {accuracy * 100:.2f}%")
```

PROGRAM 4 --- DECISION TREE CLASSIFIER

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn import metrics
from IPython.display import Image
import pydotplus
from six import StringIO

# Load dataset
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("./pima-indians-diabetes.csv", names=col_names)

# Prepare features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp', 'pedigree']
X = pima[feature_cols]
y = pima['label']

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Train Decision Tree Classifier
clf = DecisionTreeClassifier(criterion="entropy", random_state=1)
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)
print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred) * 100:.2f}%")

# Export and visualize the decision tree
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data, filled=True, feature_names=feature_cols, class_names=['0', '1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

PROGRAM 5 RANDOM FOREST CLASSIFIER

```
from sklearn import datasets
import pandas as pd
# Loading the iris plants dataset (classification)

iris = datasets.load_iris()

# creating dataframe of IRIS dataset
data = pd.DataFrame({'sepallength': iris.data[:, 0], 'sepalwidth': iris.data[:, 1],'petallength': iris.data[:, 2],
'petalwidth': iris.data[:, 3],'species': iris.target})
print(data)
print(data.head())
# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters

# Spliting arrays or matrices into random train and test subsets
from sklearn.model_selection import train_test_split
x,y = datasets.load_iris( return_X_y = True)
print(x,y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30)

from sklearn.ensemble import RandomForestClassifier
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)

y_pred=clf.fit(x_train, y_train).predict(x_test)

from sklearn import metrics
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, y_pred)*100)
print("Target Names are", iris.target_names)
# predicting which type of flower it is.
guess=clf.predict([[4, 4, 1, 1]])
print("Predicted value for the given instance is",iris.target_names[guess])
```

PROGRAM 6 SVM MACHINES

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# Load dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Use only the first two features for simplicity
y = iris.target

# Train SVM model
svc = svm.SVC(kernel='linear', C=1)
svc.fit(X, y)

# Create mesh grid for plotting decision boundary
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.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

# Predict class for each point in the mesh
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary and data points
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.4)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVC with Linear Kernel')
plt.show()
```

PROGRAM 7 KNN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load and prepare data
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Class'] = data.target_names[data.target]

X = df.iloc[:, :-1].values
y = df['Class'].values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train k-NN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Make predictions
y_pred_train = knn.predict(X_train)
y_pred_test = knn.predict(X_test)

# Evaluate model
print(f"Training accuracy: {accuracy_score(y_train, y_pred_train) * 100:.2f}%")
print(f"Testing accuracy: {accuracy_score(y_test, y_pred_test) * 100:.2f}%")
print(f"Training Confusion Matrix:\n{confusion_matrix(y_train, y_pred_train)}")
print(f"Testing Confusion Matrix:\n{confusion_matrix(y_test, y_pred_test)}")
```

PROGRAM 8 BACKPROPOGATION

```
import numpy as np

# Data
X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)
y = np.array([[92], [86], [89]], dtype=float) / 100
X /= np.amax(X, axis=0)

# Sigmoid functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Parameters
epochs = 7000
lr = 0.1
input_neurons, hidden_neurons, output_neurons = 2, 3, 1

# Weights and biases
w_hidden = np.random.uniform(size=(input_neurons, hidden_neurons))
b_hidden = np.random.uniform(size=(1, hidden_neurons))
w_out = np.random.uniform(size=(hidden_neurons, output_neurons))
b_out = np.random.uniform(size=(1, output_neurons))

# Training
for _ in range(epochs):
    # Forward pass
    h_input = sigmoid(np.dot(X, w_hidden) + b_hidden)
    output = sigmoid(np.dot(h_input, w_out) + b_out)

    # Backward pass
    error = y - output
    d_output = error * sigmoid_derivative(output)
    d_hidden = d_output.dot(w_out.T) * sigmoid_derivative(h_input)

    # Update weights and biases
    w_out += h_input.T.dot(d_output) * lr
    b_out += np.sum(d_output, axis=0, keepdims=True) * lr
    w_hidden += X.T.dot(d_hidden) * lr
    b_hidden += np.sum(d_hidden, axis=0, keepdims=True) * lr

print("Input:\n", X)
print("Actual Output:\n", y)
print("Predicted Output:\n", output)
```

PROGRAM 10 K-MEANS

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Load and prepare the data
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris.target, columns=['Targets'])

# K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
kmeans_labels = kmeans.labels_

# Standardize the data for GMM
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# GMM Clustering
gmm = GaussianMixture(n_components=3, random_state=0)
gmm.fit(X_scaled)
gmm_labels = gmm.predict(X_scaled)

# Plot results
plt.figure(figsize=(14, 14))
colormap = np.array(['red', 'lime', 'black'])

# Original Classifications
plt.subplot(2, 3, 1)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[y['Targets']])
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# K-Means Clustering
plt.subplot(2, 3, 2)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[kmeans_labels])
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# GMM Clustering
plt.subplot(2, 3, 3)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[gmm_labels])
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.tight_layout()
plt.show()
```

PROGRAM 1 PRE-PROCESSING LINEAR DISCRIMINANT

```
from numpy import nan
from pandas import read_csv
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# load the dataset
dataset = read_csv('./pima-indians-diabetes.csv',header=None)
print(dataset)
# replace '0' values with 'nan'
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
print(dataset)
# drop rows with missing values
dataset.dropna(inplace=True)
print(dataset)
# split dataset into inputs and outputs
values = dataset.values
x = values[:,0:8]
y = values[:,8]
# define the model
model = LinearDiscriminantAnalysis()

# define the model evaluation procedure
v1 = KFold(n_splits=3, shuffle=True, random_state=1)

# evaluate the model
result = cross_val_score(model, x, y, cv=v1)

# report the mean performance
final=result*100
print(final)
print('Accuracy: %.3f' %final.mean())
```