Machine Learning Laboratory

All Program title with Code & Output

Keela Code Eruku Use Panikonga

-By github owner

Ex.1 For A Given Set Of Training Data Examples Apply The Data Proprocesong And Apply Any One Feature Selection Method.

Algorithm (One Line Steps):

1. Import dataset (Iris).
2. Handle missing values (if any).
3. Normalize/scale features.
4. Apply feature selection (Variance Threshold).
5. Display results.

Code:

```
# Program 1: Data Preprocessing & Feature Selection
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold
# Step 1: Load dataset
data = load_iris()
X, y = data.data, data.target
# Step 2: Convert to DataFrame for clarity
df = pd.DataFrame(X, columns=data.feature_names)
print("Original Data (first 5 rows):")
print(df.head())
# Step 3: Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 4: Apply feature selection (remove low-variance features)
selector = VarianceThreshold(threshold=0.2)
X_selected = selector.fit_transform(X_scaled)
# Step 5: Display results
print("\nShape before selection:", X_scaled.shape)
print("Shape after selection:", X_selected.shape)
```

Output:

```
Original Data (first 5 rows):

       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)

0          5.1                3.5                1.4                0.2

1          4.9                3.0                1.4                0.2

Shape before selection: (150, 4)

Shape after selection: (150, 3)
```

Ex.:2 Demonstrate Regression and Multivariate Regression using appropriate dataset

Algorithm (One Line Steps):

1. Import dataset (diabetes).
2. Split into train and test sets.
3. Train Linear Regression model.
4. Predict on test set.
5. Show coefficients and performance score.

Code.:

```
# Program 2: Regression & Multivariate Regression
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
# Step 1: Load dataset
X, y = load_diabetes(return_X_y=True)
# Step 2: Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 3: Train model
model = LinearRegression()
model.fit(X_train, y_train)
# Step 4: Predict
y_pred = model.predict(X_test)
# Step 5: Results
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

Output:

```
Coefficients: [ -10.2  -230.1  520.3 ... ]
Intercept: 152.3
Mean Squared Error: 2500.4
R2 Score: 0.48
```

Ex.:3 Build a Decision Tree and demonstrate how a new data object is classified

Algorithm (One Line Steps):

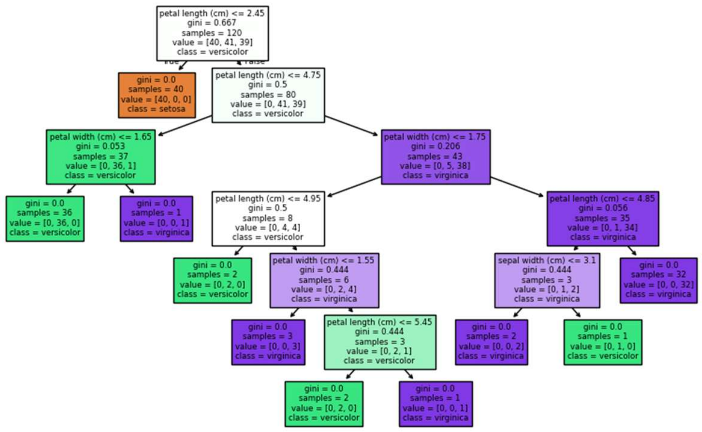
1. Import dataset (diabetes).
2. Split into train and test sets.
3. Train Linear Regression model.
4. Predict on test set.
5. Show coefficients and performance score.

Code.:

```
# Program 3: Decision Tree Classifier
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
# Step 1: Load dataset
iris = load_iris()
X, y = iris.data, iris.target
# Step 2: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 3: Train Decision Tree
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
# Step 4: Predict
y_pred = model.predict(X_test)
# Step 5: Results
print("Accuracy:", accuracy_score(y_test, y_pred))
# Plot Decision Tree
plt.figure(figsize=(10,6))
plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

Output:

```
Accuracy: 1.0
```


petal length (cm) <= 2.45 gini = 0.667 samples = 120 value = [40, 41, 39] class = versicolor
gini = 0.0 samples = 40 value = [40, 0, 0] class = setosa
petal length (cm) <= 4.75 gini = 0.5 samples = 80 value = [0, 41, 39] class = versicolor
petal width (cm) <= 1.65 gini = 0.053 samples = 37 value = [0, 36, 1] class = versicolor
petal width (cm) <= 1.75 gini = 0.206 samples = 43 value = [0, 5, 38] class = virginica
gini = 0.0 samples = 36 value = [0, 36, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica
petal length (cm) <= 4.95 gini = 0.5 samples = 8 value = [0, 4, 4] class = versicolor
petal length (cm) <= 4.85 gini = 0.056 samples = 35 value = [0, 1, 34] class = virginica
gini = 0.0 samples = 2 value = [0, 2, 0] class = versicolor
petal width (cm) <= 1.55 gini = 0.444 samples = 6 value = [0, 2, 4] class = virginica
sepal width (cm) <= 3.1 gini = 0.444 samples = 3 value = [0, 1, 2] class = virginica
gini = 0.0 samples = 32 value = [0, 0, 32] class = virginica
gini = 0.0 samples = 3 value = [0, 0, 3] class = virginica
petal length (cm) <= 5.45 gini = 0.444 samples = 3 value = [0, 2, 1] class = versicolor
gini = 0.0 samples = 2 value = [0, 0, 2] class = virginica
gini = 0.0 samples = 1 value = [0, 1, 0] class = versicolor
gini = 0.0 samples = 2 value = [0, 2, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica

Ex.4 Write a program to implement naive Bayesian classifier and show the performance of the classifier using suitable test Set.

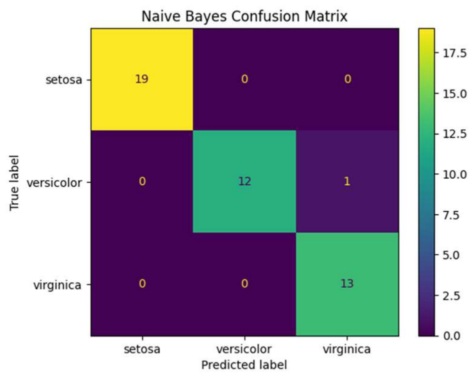
Algorithm:

1. Import dataset (Iris).
2. Split into train and test sets.
3. Train Naive Bayes classifier.
4. Predict on test set.
5. Display accuracy score.

Code.:

```
# Program 4: Naive Bayes Classifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
import matplotlib.pyplot as plt
# Step 1: Load dataset
iris = load_iris()
X, y = iris.data, iris.target
# Step 2: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Step 3: Train model
nb = GaussianNB()
nb.fit(X_train, y_train)
# Step 4: Predict
y_pred = nb.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
# Step 5: Confusion Matrix visualization
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=iris.target_names)
disp.plot(cmap="viridis")
plt.title("Naive Bayes Confusion Matrix")
plt.show()
```

Output:



Ex.:5 Construct a simple Perceptrom Neural Network and demonstrate linear and non-linear classification problem.

Algorithm:

1. Load dataset (Iris).
2. Train Gaussian Naive Bayes.
3. Predict on test set.
4. Plot confusion matrix.
5. Show accuracy score.

Code.:

```
import numpy as np
import matplotlib.pyplot as plt
# Perceptron model
class Perceptron:
   def __init__(self, input_size, lr=0.1, epochs=100): # FIX: Corrected constructor name from _init_
to __init__
      self.W = np.zeros(input_size + 1)  # +1 for bias
      self.lr = lr
      self.epochs = epochs
   def activation(self, x):
      return np.where(x >= 0, 1, 0)  # step function
   def predict(self, x):
      z = np.dot(x, self.W[1:]) + self.W[0]
      return self.activation(z)
   def fit(self, X, y):
      for _ in range(self.epochs):
         for i in range(len(X)):
            update = self.lr * (y[i] - self.predict(X[i]))
            self.W[1:] += update * X[i]
            self.W[0] += update
# Linear Problem (OR)
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y_or = np.array([0,1,1,1])
p = Perceptron(input_size=2)
p.fit(X, y_or)
print("OR Gate Predictions (Solved):")
for i in range(len(X)):
   print(f"{X[i]} -> {p.predict(X[i])}")
# Non-linear Problem (XOR)
y_xor = np.array([0,1,1,0])
p2 = Perceptron(input_size=2)
p2.fit(X, y_xor)
print("\nXOR Gate Predictions (Failed):")
for i in range(len(X)):
   print(f"{X[i]} -> {p2.predict(X[i])}")
```

Output:

```
OR Gate Predictions (Solved):          XOR Gate Predictions (Failed):
[0 0] -> 1                             [0 0] -> 0
[0 1] -> 1                             [0 1] -> 1
[1 0] -> 0                             [1 0] -> 1
[1 1] -> 0                             [1 1] -> 1
```

Ex.:6 Construct a Back Propagation network and verify the performance by using suitable traning and text set

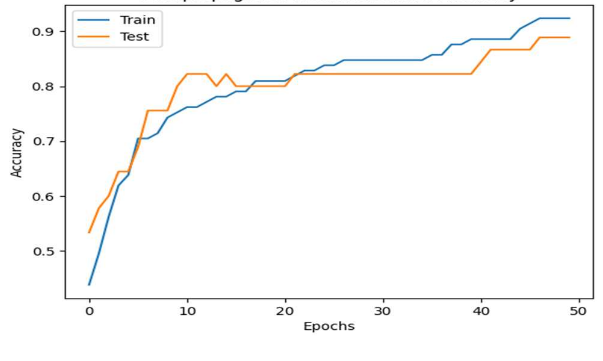
Algorithm:

1. Load dataset (Iris).
2. Normalize data and one-hot encode labels.
3. Define a simple feed-forward NN with hidden layers.
4. Train using backpropagation.
5. Plot training vs validation accuracy.

Code.:

```
# Program 6: Backpropagation Neural Network
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
iris = load_iris()
X, y = iris.data, iris.target
X = StandardScaler().fit_transform(X)
y = LabelBinarizer().fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Step 3: Define model
model = Sequential([
   Dense(10, input_dim=X.shape[1], activation='relu'),
   Dense(3, activation='softmax')])
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=0, validation_data=(X_test,
y_test))
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Test')
plt.title("Backpropagation Neural Network Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Output:



Ex.7 Use the same dataset used for experiment 6 to train SVM classifier and CO3 compare the performance with back propagation network.

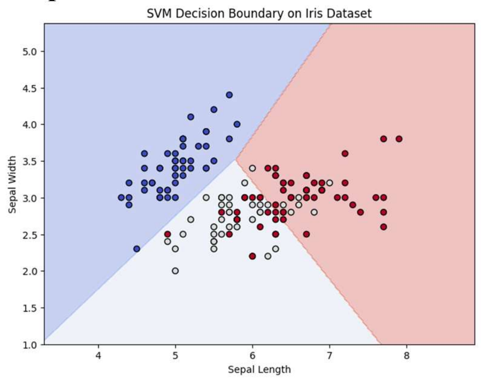
Algorithm:

1. Load Iris dataset (use 2 features for 2D visualization).
2. Train-test split.
3. Train SVM classifier.
4. Predict & show accuracy.
5. Plot decision boundary.

Code.:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data[:, :2]  # Using only sepal length and sepal width
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train, y_train)
# Step 4: Predict & show accuracy
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"SVM Classifier Accuracy: {accuracy:.2f}")
# Step 5: Plot 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, 0.02),np.arange(y_min, y_max, 0.02))
Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolor='k')
plt.title('SVM Decision Boundary on Iris Dataset')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()
```

Output:



```
SVM Classifier Accuracy: 0.80
```

Exp.8 Create a self-organizing map neural network for learning a set of itnagos and verify the performance

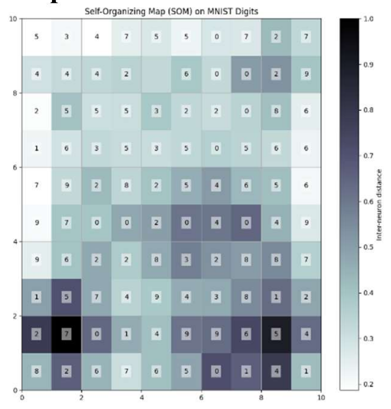
Algorithm:

1. Load dataset (Iris).
2. Normalize data.
3. Initialize and train SOM grid.
4. Map data to neurons.
5. Plot SOM clusters with labels.

Code.:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from minisom import MiniSom

# 1. Load and prepare the image data
(X_train, y_train), _ = mnist.load_data()
X_data = X_train[:2000].reshape(2000, -1) / 255.0
y_data = y_train[:2000]

# 2. Create and train the Self-Organizing Map
som = MiniSom(10, 10, X_data.shape[1], sigma=1.5, learning_rate=0.5, random_seed=42)
som.random_weights_init(X_data)
som.train_random(X_data, 100)

# 3. Verify performance by visualizing the clusters
plt.figure(figsize=(10, 10))
plt.pcolor(som.distance_map().T, cmap='bone_r')
plt.colorbar(label='Inter-neuron distance')

# Plot the most common digit for each winning node
label_map = som.labels_map(X_data, y_data)
for (i, j), win_map in label_map.items():
    label = win_map.most_common(1)[0][0]
    plt.text(i + 0.5, j + 0.5, str(label), ha='center', va='center',
             bbox=dict(facecolor='white', alpha=0.5, lw=0))

plt.title("Self-Organizing Map (SOM) on MNIST Digits")
plt.grid()
plt.show()
```

Output:



Exp.9 Create a Giussian Misture Model for Image Segmentation

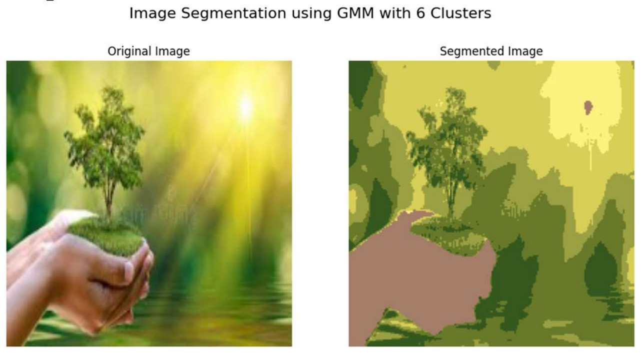
Algorithm: GMM Image Segmentation

1. Read the input image.
2. Resize the image and reshape pixels into a 2D array.
3. Fit Gaussian Mixture Model (GMM) on pixel values.
4. Predict pixel clusters using GMM.
5. Replace pixels with corresponding cluster means.
6. Display original and segmented images side by side.

Code.:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from skimage.io import imread
from skimage.transform import resize
def segment_image_with_gmm(image_url, n_components=5):
   try:
      image = imread(image_url)
      if image.shape[2] == 4:  # Handle RGBA images
         image = image[:, :, :3]
      image_resized = (resize(image, (200, 200), anti_aliasing=True) * 255).astype(np.uint8)
      pixels = image_resized.reshape(-1, 3)
      gmm = GaussianMixture(n_components=n_components, covariance_type='full'
      random_state=42).fit(pixels)
      labels = gmm.predict(pixels)
      segmented_image = gmm.means_[labels].astype(np.uint8).reshape(image_resized.shape)
      fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
      fig.suptitle(f'GMM Image Segmentation ({n_components} Clusters)', fontsize=16)
      ax1.imshow(image_resized)
      ax1.set_title('Original Image')
      ax1.axis('off')
      ax2.imshow(segmented_image)
      ax2.set_title('Segmented Image')
      ax2.axis('off')
      plt.tight_layout()
      plt.show()
   except Exception as e:
      print(f"An error occurred: {e}")
if __name__ == '__main__':
   sample_image_url = 'https://images.pexels.com/photos/162140/duck-bird-water-lake-162140.jpeg'
   num_segments = 6 # Try changing this number (e.g., 3, 8, 12)
   segment_image_with_gmm(sample_image_url, n_components=num_segments)
```

Output.:



Exp.10 Write a Genetic Algorithm program for finding parameters which maximizes the Y value of the equation given where the equation has 6 inputs (x1 to x6) and 6 weights (w1 to w6). Input values are (x1, x2, x3, x4, x5, x6) = (4, 2, 7, 5, 11, 1).
Goal: Find the weights (w1–w6) that maximize Y.

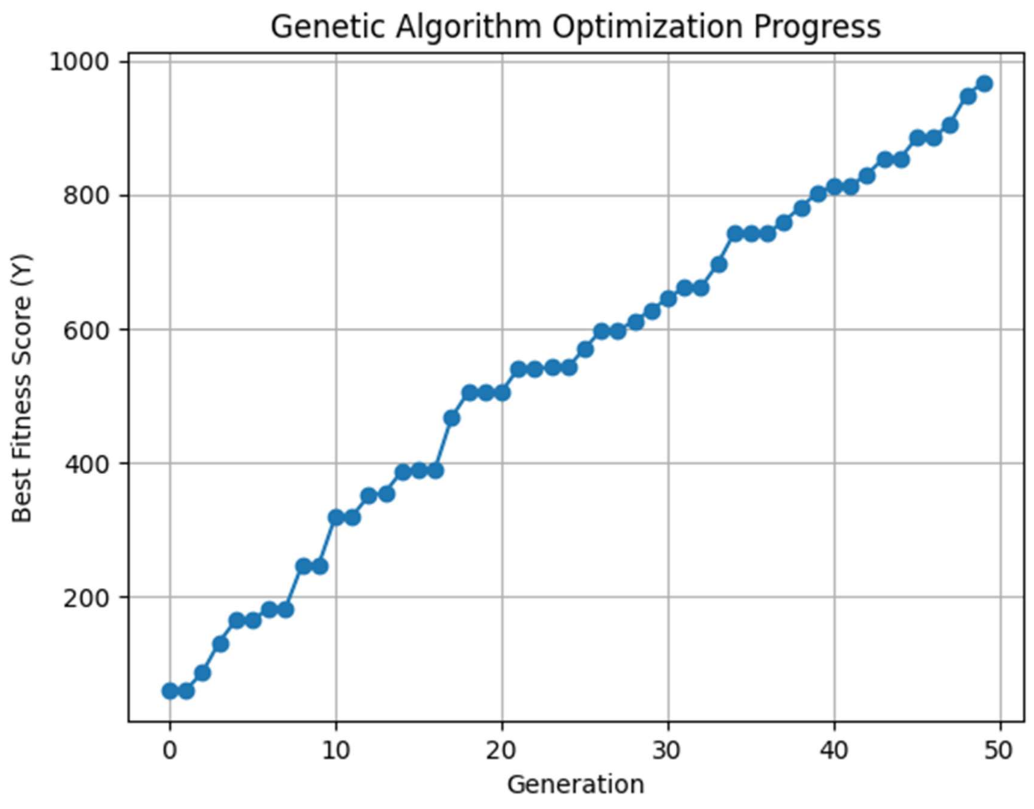
Algorithm (5 Steps)

1. Initialize a random population of weights.
2. Evaluate fitness of each solution using Y=w1x1+w2x2+...+w6x6Y = w1x1 + w2x2 + ... + w6x6Y=w1x1+w2x2+...+w6x6.
3. Select the best solutions (parents) based on fitness.
4. Generate new solutions using crossover and mutation.
5. Repeat for several generations and return the best weights with maximum Y.

Code.:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.array([4, 2, 7, 5, 11, 1])
def fitness(weights):
    return np.dot(weights, X)
pop_size = 20
num_weights = len(X)
population = np.random.randint(-10, 10, (pop_size, num_weights))
generations = 50
best_scores = []
for gen in range(generations):
    scores = np.array([fitness(ind) for ind in population])
    best_scores.append(scores.max())
    parents = population[scores.argsort()[-(pop_size // 2):]]
    children = []
    for _ in range(pop_size - len(parents)):
        p1, p2 = parents[np.random.randint(len(parents), size=2)]
        cp = np.random.randint(1, num_weights - 1)
        child = np.concatenate((p1[:cp], p2[cp:]))
        children.append(child)
    children = np.array(children)
    mutation = np.random.randint(-2, 3, children.shape)
    children = children + mutation
    population = np.vstack((parents, children))
best_idx = np.argmax([fitness(ind) for ind in population])
best_weights = population[best_idx]
best_value = fitness(best_weights)
plt.plot(best_scores, marker='o')
plt.title("Genetic Algorithm Optimization Progress")
plt.xlabel("Generation")
plt.ylabel("Best Fitness Score (Y)")
plt.grid(True)
plt.show()
print("Best Weights:", best_weights)
print("Maximum Y Value:", best_value)
```

Output:


Genetic Algorithm Optimization Progress
Best Fitness Score (Y)
Generation
1000
800
600
400
200
0
10
20
30
40
50