ابتدا داده هارا لود میکنیم:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier, MLPRegressor
# لود دیتا

iris_sklearn = load_iris()
print(iris_sklearn)
# تبدیل به فرمت دیتا
iris = pd.DataFrame(data=iris_sklearn.data,
columns=iris_sklearn.feature_names)
iris
```

سپس فیچر و تارگتمون را تعریف میکنیم

```
# Load Iris dataset from scikit-learn
iris_sklearn = load_iris()
X = iris_sklearn.data
y = iris_sklearn.target

# Splitting the 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=83)
```

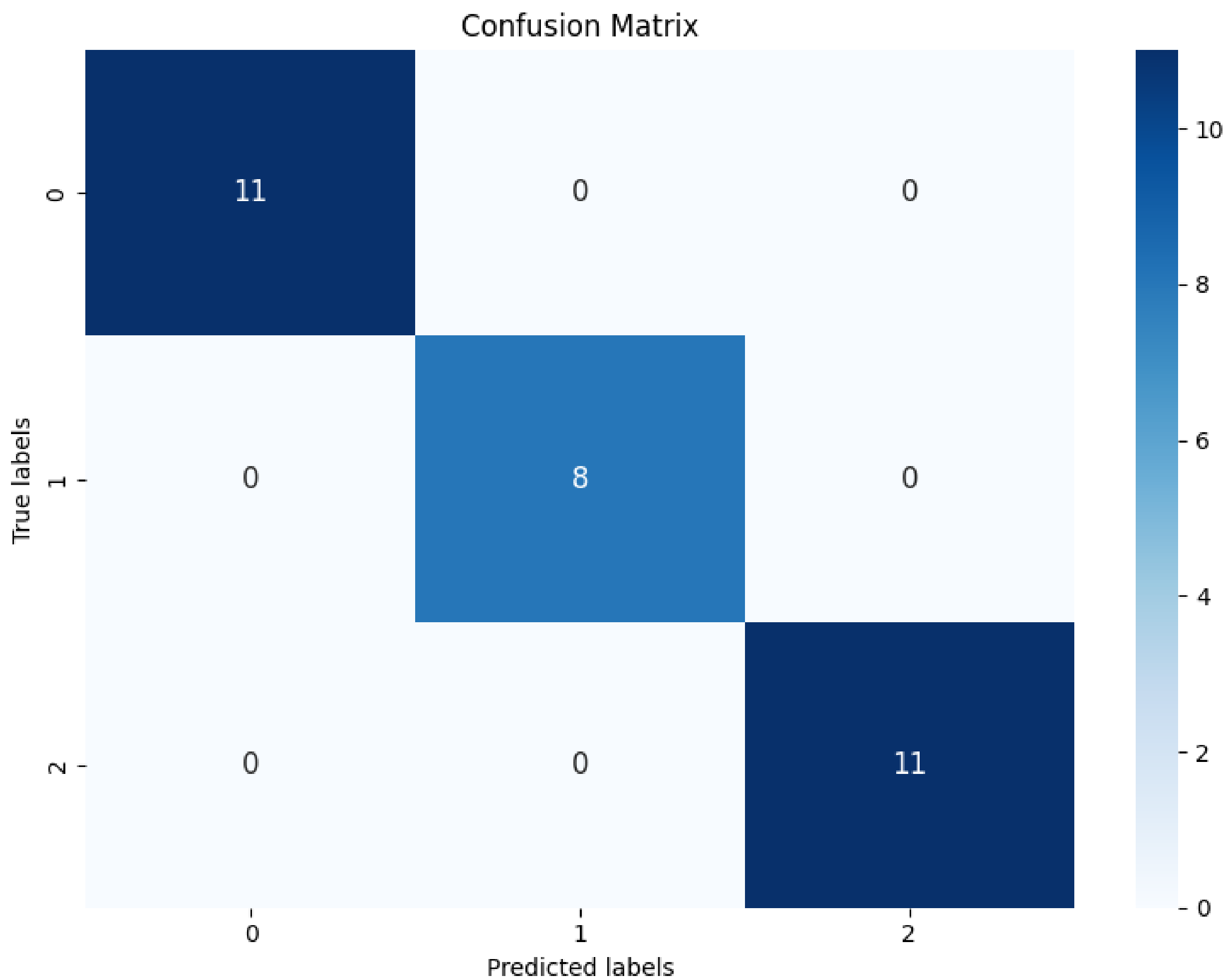
مدل خود را با روش mlp می سازیم. مدل ساخته یک شبکه عصبی با یک لایه و 100 نرون است با تابع فعالساز relu وبهینه ساز adamوضریب یادگیری 0.001 است.

```
model = MLPClassifier(hidden_layer_sizes=(100), activation='relu',
solver='adam',
                      alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001,
                      power_t=0.5, max_iter=200, shuffle=True,
random_state=83, tol=0.0001, verbose=False,
                      warm_start=False, momentum=0.9,
nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1,
```

```
                      beta_1=0.9, beta_2=0.999, epsilon=1e-08,
n_iter_no_change=10, max_fun=15000)
```

سپس ماتریس کانفیشون را رسم میکنیم

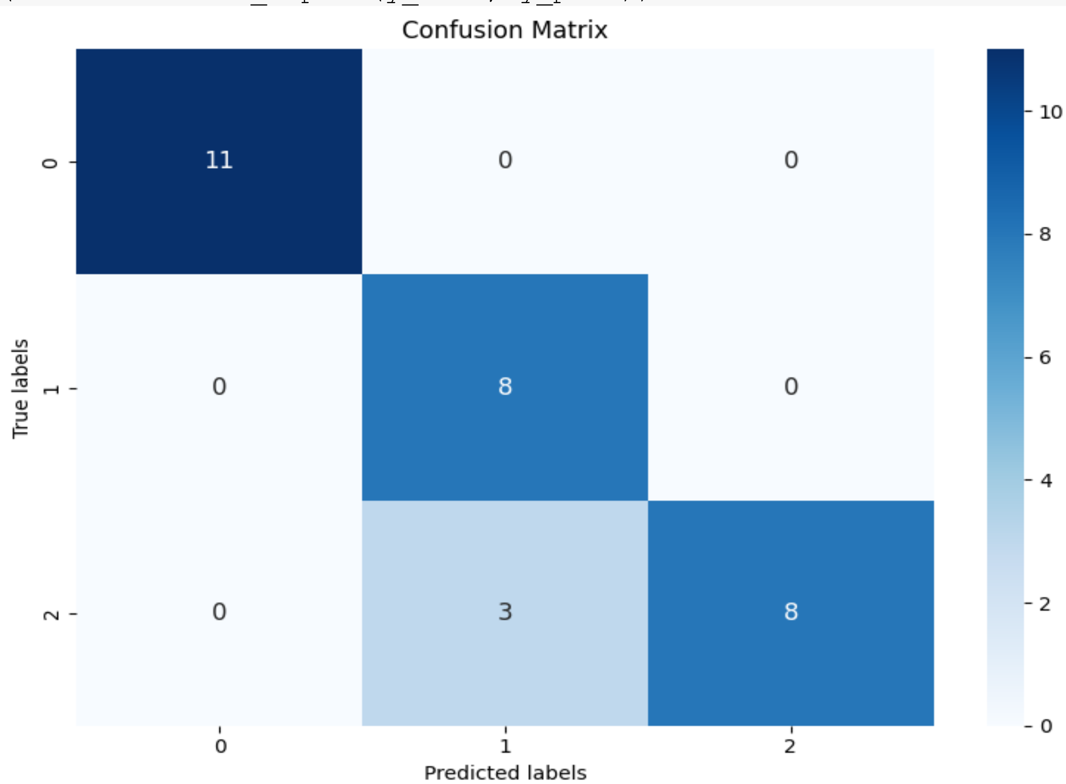
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



سپس داده هارا با گرادیان نزولی آموزش میدهیم و باز ماتریس کانفیوشن را رسم میکنیم

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split


# Assuming you've trained your model already
model = LogisticRegression(max_iter=200, random_state=23)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)
```

```
# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

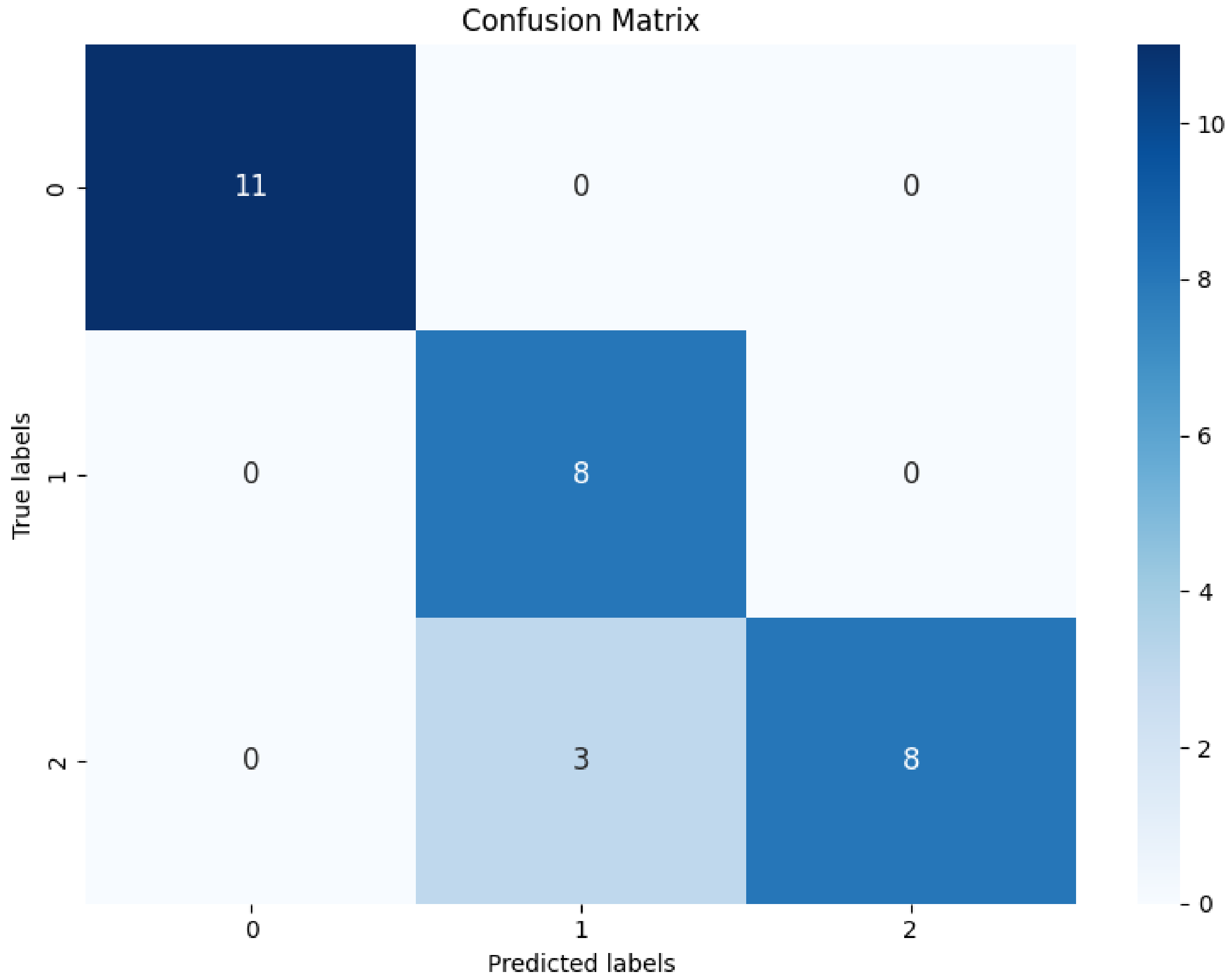


مشاهده میشود در مدل قبل داده ها بهتر آموزش دیده بودند

برای حالت rbf نیز به همین صورت عمل میکنیم

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split


# Assuming you've trained your model already
model = SVC(kernel='rbf', random_state=23)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

```
# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



که نتیجه مشابهی با گرادیان نزولی دارد

آموزش بدون استفاده از توابع از پیش ساخته شده:

1:گرادیان نزولی

```
import numpy as np

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        # Gradient Descent
        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            # Gradient calculation
            dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls
```

```
model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
model.fit(x_train, y_train)

predictions = model.predict(x_test)

from sklearn.metrics import accuracy_score, classification_report

print("Accuracy:", accuracy_score(y_test, predictions))
print("Classification Report:\n", classification_report(y_test,
predictions))

cf_matrix = confusion_matrix(y_test, predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

plt.gca().set_ylim(len(np.unique(y_test)), 0)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

همانطور که مشاهده میشود accuerecy تابع بسیار پایین و برابر 0.26 است

RBF

```
import numpy as np
from scipy.spatial.distance import cdist

class RBFNetwork:
    def __init__(self, num_centers, learning_rate=0.1, num_epochs=100):
        self.num_centers = num_centers
        self.learning_rate = learning_rate
```

```
        self.num_epochs = num_epochs
        self.centers = None
        self.weights = None
        self.beta = None

    def radial_basis_function(self, x, center, beta):
        return np.exp(-beta * np.linalg.norm(x - center) ** 2)

    def fit(self, X, y):
        # Randomly initialize centers
        self.centers = X[np.random.choice(X.shape[0], self.num_centers,
replace=False)]

        # Calculate spread parameter beta
        distances = cdist(X, self.centers)
        self.beta = 1 / (2 * np.mean(np.var(distances, axis=1)))

        # Calculate RBF activations
        phi = np.array([self.radial_basis_function(x, center, self.beta)
for x in X for center in self.centers])
        phi = phi.reshape(X.shape[0], self.num_centers)

        # Add bias term to the input data
        phi = np.insert(phi, 0, 1, axis=1)  # Bias term

        # Initialize weights
        self.weights = np.random.randn(phi.shape[1])

        # Training using gradient descent
        for _ in range(self.num_epochs):
            for i in range(X.shape[0]):
                output = np.dot(phi[i], self.weights)
                error = y[i] - output
                self.weights += self.learning_rate * error * phi[i]

    def predict(self, X):
        # Calculate RBF activations for test data
        phi = np.array([self.radial_basis_function(x, center, self.beta)
for x in X for center in self.centers])
        phi = phi.reshape(X.shape[0], self.num_centers)

        # Add bias term to the input data
        phi = np.insert(phi, 0, 1, axis=1)  # Bias term

        # Make predictions
```

```
        predictions = np.dot(phi, self.weights)
        return predictions

rbf = RBFNetwork(num_centers=10, learning_rate=0.01, num_epochs=1000)
rbf.fit(x_train, y_train)
predictions = rbf.predict(x_test)

mse = np.mean((predictions - y_test) ** 2)
print("Mean Squared Error:", mse)
```