**MLA0401-Deep Learning for Sustainability**

**Name: K. Lahari**

**Reg No:192325118**

**Experiment 1:**

**Aim:** To demonstrate confusion matrix using python

**Program:**

import numpy as np

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

actual    = np.array(

  ['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not Dog','Not Dog'])

predicted = np.array(

  ['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not Dog','Not Dog'])

conf\_matrix=confusion\_matrix(actual,predicted)

sns.heatmap(conf\_matrix,annot=True,fmt='g',xticklabels=['Dog','Not Dog'],yticklabels=['Dog','Not Dog'],cmap='RdPu')

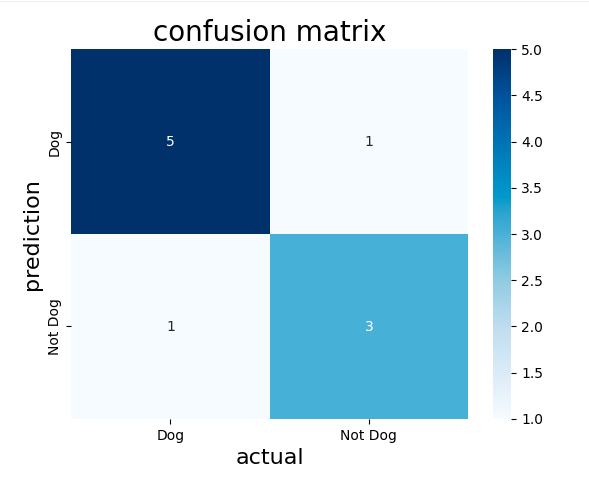
plt.ylabel("prediction",fontsize=16)

plt.xlabel("actual",fontsize=16)

plt.title("confusion matrix",fontsize=20)

plt.show()

**Output:**



**Experiment 2:**

**Aim:** To demonstrate 2 class confusion matrix using python

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score,f1\_score,precision\_score,recall\_score,classification\_report,confusion\_matrix

wine=load\_wine()

data=pd.DataFrame(data=wine.data,columns=wine.feature\_names)

data['Target']=wine.target

data=data[data['Target']!=2]

x=data.drop('Target',axis=1)

y=data['Target']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=1)

model=DecisionTreeClassifier(random\_state=1)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy:",accuracy)

class\_report = classification\_report(y\_test, y\_pred, target\_names=wine.target\_names[:2])

print("Classification Report:\n", class\_report)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1 Score: {f1:.2f}")

conf\_matrix=confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8,6))

sns.heatmap(conf\_matrix,annot=True,fmt='d',cmap='PuBuGn',xticklabels=wine.target\_names[:2],yticklabels=wine.target\_names[:2])

plt.xlabel("predicted label")

plt.ylabel("true label")

plt.title("confusion matrix")

plt.show()

**Output:**

accuracy: 0.9743589743589743

Classification Report:

precision recall f1-score support

class\_0 1.00 0.94 0.97 17

class\_1 0.96 1.00 0.98 22

accuracy 0.97 39

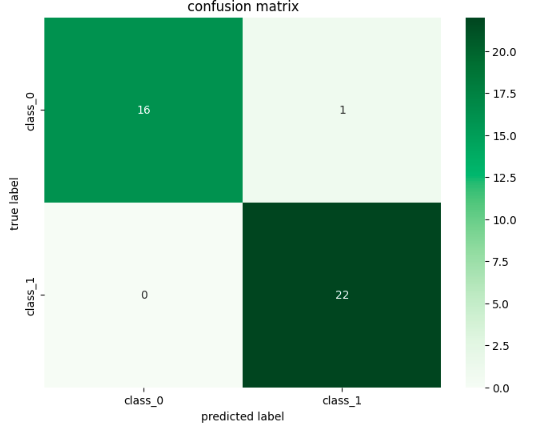
macro avg 0.98 0.97 0.97 39

weighted avg 0.98 0.97 0.97 39

Precision: 0.96

Recall: 1.00

F1 Score: 0.98



**Experiment 3:**

**Aim:** To analyse the performance of a multi class confusion matrix by using choosen database with python code

**Program:**

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

X, y= load\_digits(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,test\_size=0.25)

clf = RandomForestClassifier(random\_state=23)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

cm = confusion\_matrix(y\_test,y\_pred)

sns.heatmap(cm,

            annot=True,

            fmt='g',cmap="winter")

plt.ylabel('Prediction',fontsize=13)

plt.xlabel('Actual',fontsize=13)

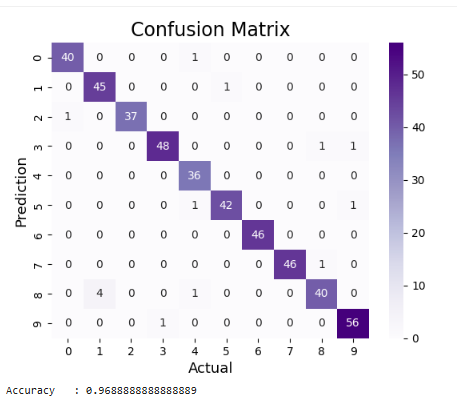
plt.title('Confusion Matrix',fontsize=17)

plt.show()

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy   :", accuracy)

**Output:**



**Experiment 4:**

**Aim:** To analyse the performance of a over fitting by using choosen database with python code

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import cross\_val\_score

def true\_fun(X):

    return np.cos(1.5 \* np.pi \* X)

np.random.seed(0)

n\_samples = 30

degrees = [1, 4, 15]

X = np.sort(np.random.rand(n\_samples))

y = true\_fun(X) + np.random.randn(n\_samples) \* 0.1

plt.figure(figsize=(14, 5))

for i in range(len(degrees)):

    ax = plt.subplot(1, len(degrees), i + 1)

    plt.setp(ax, xticks=(), yticks=())

    polynomial\_features = PolynomialFeatures(degree=degrees[i], include\_bias=False)

    linear\_regression = LinearRegression()

    pipeline = Pipeline(

        [

            ("polynomial\_features", polynomial\_features),

            ("linear\_regression", linear\_regression),

        ]

    )

    pipeline.fit(X[:, np.newaxis], y)

    scores = cross\_val\_score(

        pipeline, X[:, np.newaxis], y, scoring="neg\_mean\_squared\_error", cv=10

    )

    X\_test = np.linspace(0, 1, 100)

    plt.plot(X\_test, pipeline.predict(X\_test[:, np.newaxis]), label="Model")

    plt.plot(X\_test, true\_fun(X\_test), label="True function")

    plt.scatter(X, y, edgecolor="y", s=20,marker="^", label="Samples")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.xlim((0, 1))

    plt.ylim((-2, 2))

    plt.legend(loc="best")

    plt.title(

        "Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(

            degrees[i], -scores.mean(), scores.std()

        )

    )

plt.show()

**Output:**



**Experiment 5:**

**Aim:** To demonstrate the performance of a linear regression by using choosen database with python code

**Program:**

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.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['species'] = iris.target

print(data.head())

X = data[['sepal length (cm)']]

y = data['sepal width (cm)']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error on test set: {mse:.2f}')

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, color='r',marker='\*', label='Actual')

plt.plot(X\_test, y\_pred, color='y', linewidth=3, label='Predicted')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.title('Linear Regression: Sepal Width vs Sepal Length')

plt.legend()

plt.show()

new\_sample = pd.DataFrame([[5]], columns=['sepal length (cm)'])

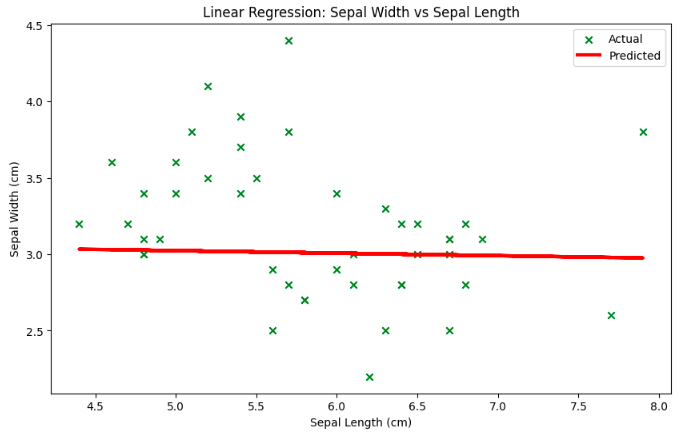
predicted\_width = model.predict(new\_sample)

print(f'The predicted sepal width for sepal length {new\_sample.values.tolist()} is {predicted\_width[0]:.2f} cm')

**Output:**

Mean Squared Error on test set: 0.23

The predicted sepal width for sepal length [[5]] is 3.02 cm



**Experiment 6:**

**Aim:** To demonstrate the performance of knn using wine dataset

**Program:**

import pandas as pd

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score,confusion\_matrix

wine=load\_wine()

data=pd.DataFrame(data=wine.data,columns=wine.feature\_names)

data['Target']=wine.target

x=data.drop('Target',axis=1)

y=data['Target']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=1)

model=KNeighborsClassifier(n\_neighbors=5)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='summer', xticklabels=wine.target\_names, yticklabels=wine.target\_names)

plt.xlabel('Predicted Label')

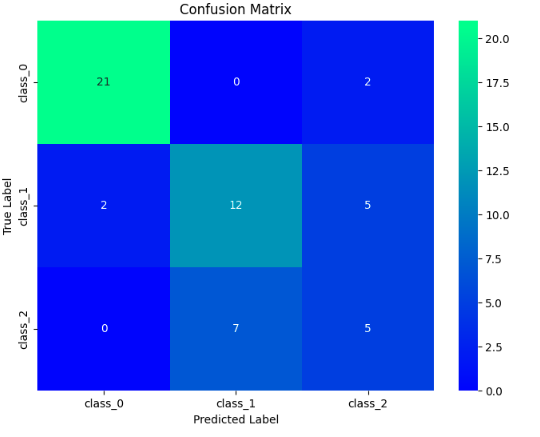
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 0.7037037037037037



**Experiment 7:**

**Aim:** To demonstrate the performance of a logistic regression by using choosen database with python code.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

def sigmoid(z):

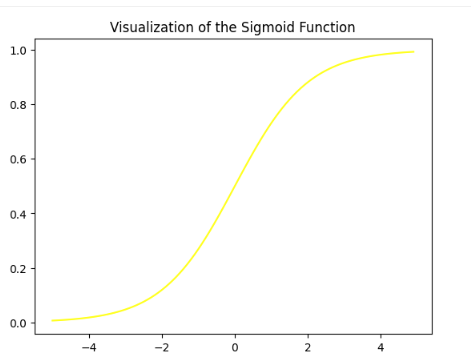
    return 1 / (1 + np.exp( - z))

plt.plot(np.arange(-5, 5, 0.1), sigmoid(np.arange(-5, 5, 0.1)),color='pink')

plt.title('Visualization of the Sigmoid Function')

plt.show()

**Output:**



**Experiment 8:**

**Aim:** To demonstrate the performance of KNN algorithm by using iris dataset

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

iris=load\_iris()

data=pd.DataFrame(data=iris.data,columns=iris.feature\_names)

data['Species']=iris.target

x=data.drop('Species',axis=1)

y=data['Species']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=1)

model=KNeighborsClassifier(n\_neighbors=5)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='pink', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel('Predicted Label')

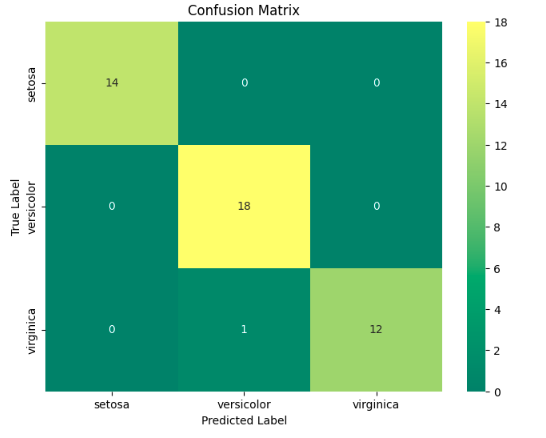
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 0.9777777777777777



**Experiment 9:**

**Aim: :** To demonstrate the performance of Naïve Bayes algorithm by using iris dataset

**Program:**

#naive bayes iris

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score,confusion\_matrix

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['species'] = iris.target

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='vanimo', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel('Predicted Label')

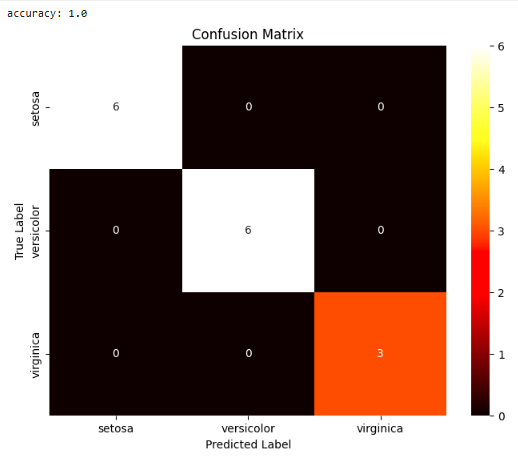
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 1.0



**Experiment 10:**

**Aim:** To demonstrate the performance of Logistic Regression using iris dataset

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score,confusion\_matrix

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['species'] = iris.target

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

model =LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='berlin', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel('Predicted Label')

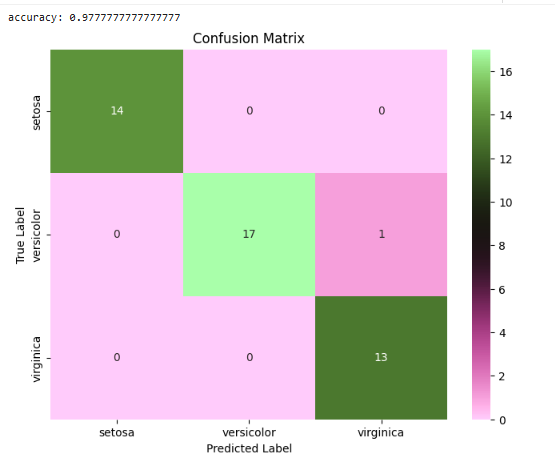
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 0.977777777777777



**Experiment 11:**

**Aim:** To demonstrate the performance of Decision tree Classifier algorithm using iris dataset.

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score,confusion\_matrix

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['species'] = iris.target

X = data.drop('species', axis=1)

y = data['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

model =DecisionTreeClassifier()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='PRGn', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel('Predicted Label')

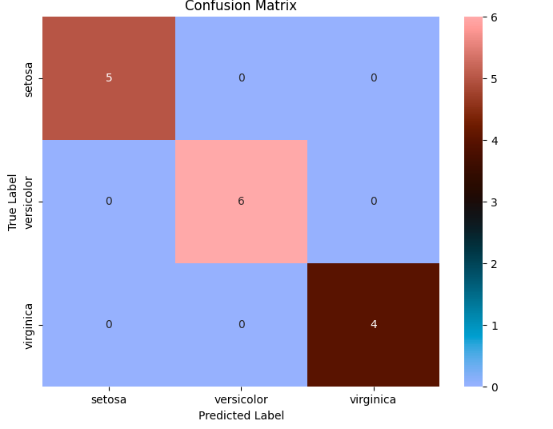
plt.ylabel('True Label')

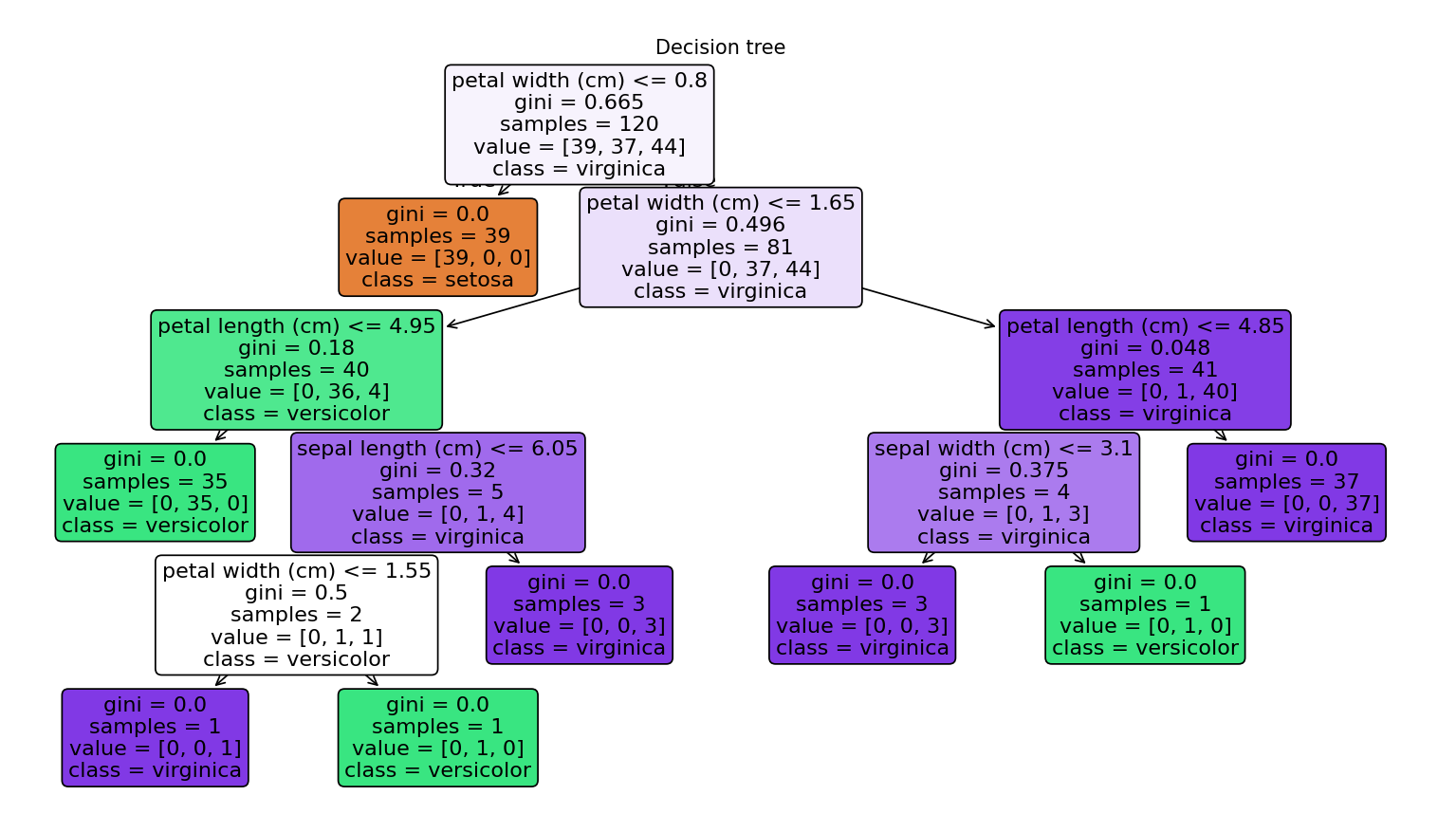
plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 1.0



****

**Experiment 12:**

**Aim:** To demonstrate the performance of Random Forest classifier algorithm by using iris dataset

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score,confusion\_matrix

iris=load\_iris()

data=pd.DataFrame(data=iris.data,columns=iris.feature\_names)

data['Species']=iris.target

x=data.drop('Species',axis=1)

y=data['Species']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=1)

model=RandomForestClassifier(n\_estimators=100,random\_state=1)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='managua', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

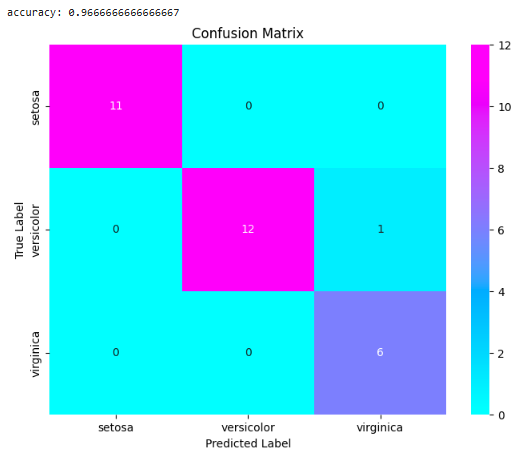
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**



**Experiment 13:**

**Aim:** To demonstrate the performance of SVM algorithm by using iris dataset

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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,confusion\_matrix

iris=load\_iris()

data=pd.DataFrame(data=iris.data,columns=iris.feature\_names)

data['Species']=iris.target

x=data.drop('Species',axis=1)

y=data['Species']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=1)

model=SVC(kernel='poly',random\_state=1)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy:",accuracy)

conf\_matrix = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='turbo', xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.xlabel('Predicted Label')

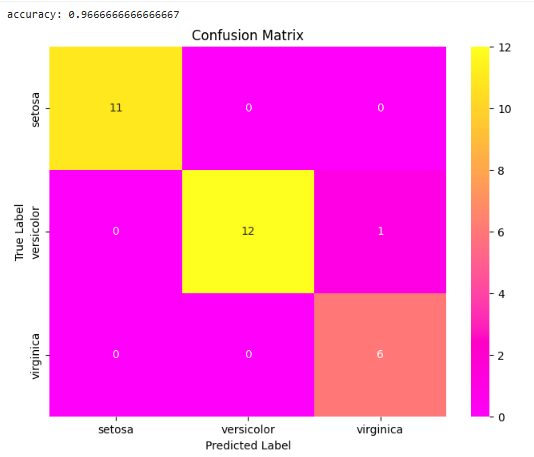
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

**Output:**

accuracy: 0.9666666666666667



**Experiment 14:**

**Aim:** To demonstrate the gradient descent

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

def mean\_squared\_error(y\_true, y\_predicted):

    cost = np.sum((y\_true - y\_predicted)\*\*2) / len(y\_true)

    return cost

def gradient\_descent(x, y, iterations=1000, learning\_rate=0.01, stopping\_threshold=1e-6):

    current\_weight = 0.0

    current\_bias = 0.0

    n = float(len(x))

    costs = []

    previous\_cost = None

    for i in range(iterations):

        y\_predicted = current\_weight \* x + current\_bias

        current\_cost = mean\_squared\_error(y, y\_predicted)

        if previous\_cost and abs(previous\_cost - current\_cost) <= stopping\_threshold:

            break

        previous\_cost = current\_cost

        costs.append(current\_cost)

        weight\_derivative = -(2/n) \* sum(x \* (y - y\_predicted))

        bias\_derivative = -(2/n) \* sum(y - y\_predicted)

        current\_weight = current\_weight - learning\_rate \* weight\_derivative

        current\_bias = current\_bias - learning\_rate \* bias\_derivative

        if i % 100 == 0:

            print(f"Iteration {i+1}: Cost {current\_cost}, Weight {current\_weight}, Bias {current\_bias}")

    plt.figure(figsize=(8,6))

    plt.plot(range(len(costs)), costs, 'r.')

    plt.title("Cost vs Iterations")

    plt.xlabel("Iterations")

    plt.ylabel("Cost")

    plt.show()

    return current\_weight, current\_bias

def main():

    X = np.array([32.5, 53.4, 61.5, 47.4, 59.8, 55.1, 52.2, 39.2, 48.1, 52.5, 45.4, 54.3, 44.1, 58.1, 56.7, 48.9, 44.6, 60.2, 45.6, 38.8])

    Y = np.array([31.7, 68.7, 62.5, 71.5, 87.2, 78.2, 79.6, 59.1, 75.3, 71.3, 55.1, 82.4, 62.0, 75.3, 81.4, 60.7, 82.8, 97.3, 48.8, 56.8])

    scaler = StandardScaler()

    X\_normalized = scaler.fit\_transform(X.reshape(-1, 1)).flatten()

    estimated\_weight, estimated\_bias = gradient\_descent(X\_normalized, Y, iterations=2000, learning\_rate=0.01)

    print(f"Estimated Weight: {estimated\_weight}, Estimated Bias: {estimated\_bias}")

    Y\_pred = estimated\_weight \* X\_normalized + estimated\_bias

    plt.figure(figsize=(8,6))

    plt.scatter(X, Y, color='black', label='Data Points',marker='\*')

    plt.plot(X, Y\_pred, color='red', linestyle='--', label='Fitted Line')

    plt.xlabel("X")

    plt.ylabel("Y")

    plt.title("Linear Regression using Gradient Descent")

    plt.legend()

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

Iteration 1: Cost 5031.3015, Weight 0.21744528996901658, Bias 1.3877

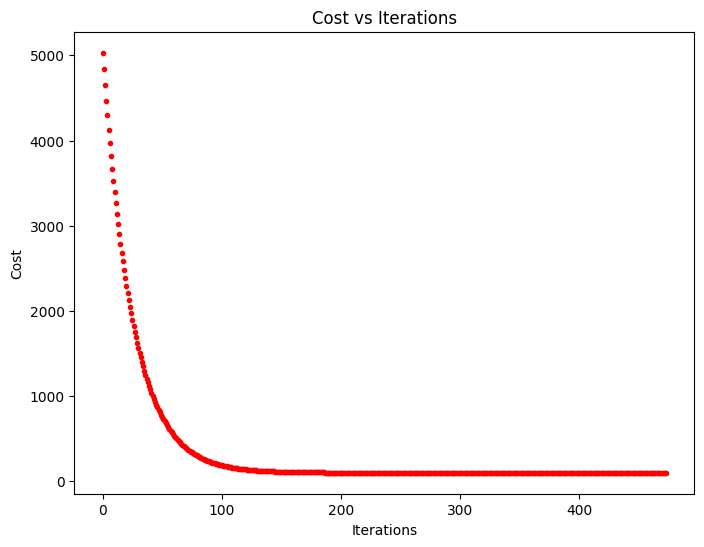
Iteration 101: Cost 185.56941123866557, Weight 9.459227106883091, Bias 60.3672282719577

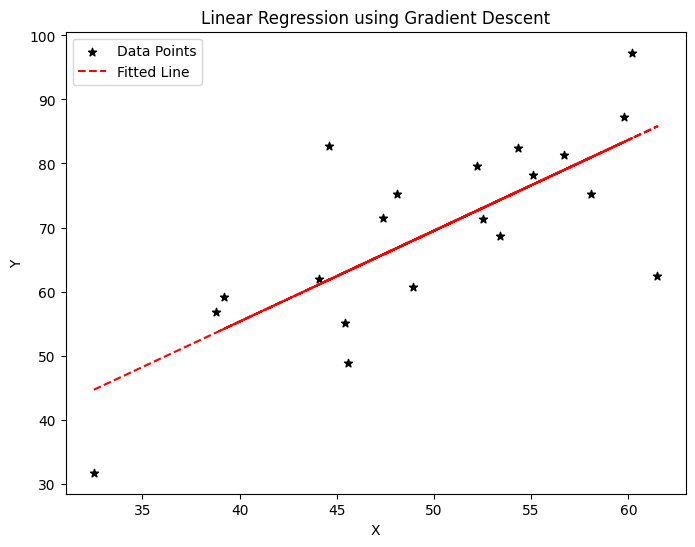
Iteration 201: Cost 100.34293399589959, Weight 10.684868107118445, Bias 68.18906711826675

Iteration 301: Cost 98.84397526476002, Weight 10.847412072256065, Bias 69.22639591234459

Iteration 401: Cost 98.81761165863254, Weight 10.868968580725983, Bias 69.36396599633204

Estimated Weight: 10.87151030998278, Estimated Bias: 69.38018689350649





**Experiment 15:**

**Aim:** To demonstrate the segmentation of image using python

**Program:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import drive

# Mount Google Drive

drive.mount('/content/drive')

# For Colab: !wget <https://example.com/cataract_image.jpg> -O image.jpg (replace with actual URL)

img = cv2.imread('/content/drive/MyDrive/dog.jpeg') # Replace with your path

# Check if the image was loaded successfully

if img is None:

print("Error: Could not load image. Please check the file path.")

else:

rgb\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

pixels = np.float32(rgb\_img.reshape((-1, 3)))

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 100, 0.2)

K = 3

\_, labels, centers = cv2.kmeans(pixels, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)

centers = np.uint8(centers)

segmented\_img = centers[labels.flatten()].reshape(rgb\_img.shape)

plt.figure(figsize=(10, 5))

plt.subplot(121)

plt.imshow(rgb\_img)

plt.title('Original Image')

plt.axis('off')

plt.subplot(122)

plt.imshow(segmented\_img)

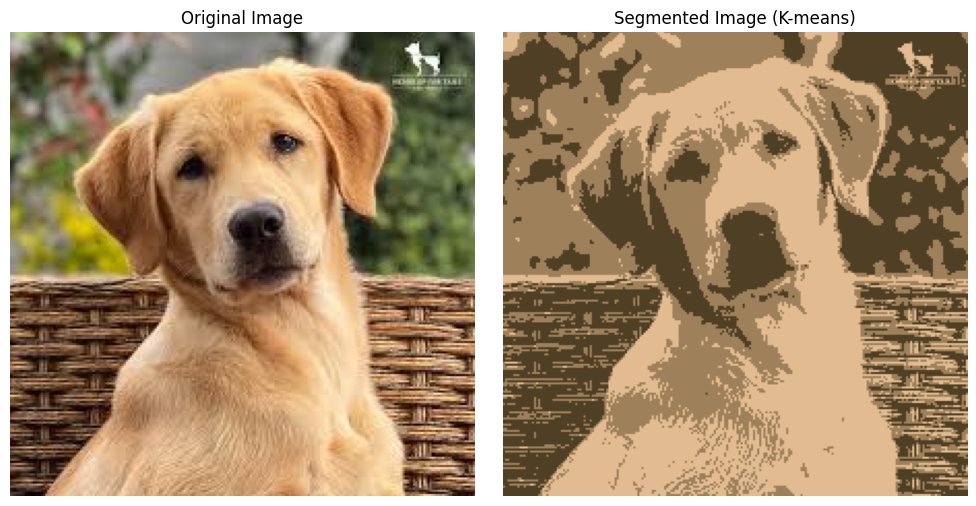
plt.title('Segmented Image (K-means)')

plt.axis('off')

plt.tight\_layout()

plt.show()

**Output:**



**Experiment 16:**

**Aim:** To demonstrate the segmentation of image using python

**Program:**

import numpy as np

import cv2

from matplotlib import pyplot as plt

from google.colab import drive

# Mount Google Drive if not already mounted

try:

drive.mount('/content/drive')

except:

pass # Drive is already mounted

# For Colab: !wget <https://example.com/cataract_image.jpg> -O image.jpg

img = cv2.imread('/content/drive/MyDrive/dog.jpeg')

# Check if the image was loaded successfully

if img is None:

print("Error: Could not load image. Please check the file path.")

else:

b, g, r = cv2.split(img)

rgb\_img = cv2.merge([r, g, b])

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

kernel = np.ones((2, 2), np.uint8)

closing = cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, kernel, iterations=2)

sure\_bg = cv2.dilate(closing, kernel, iterations=3)

plt.figure(figsize=(12, 8))

plt.subplot(231); plt.imshow(rgb\_img); plt.title("Original Image"); plt.axis('off')

plt.subplot(232); plt.imshow(gray, 'gray'); plt.title("Grayscale Image"); plt.axis('off')

plt.subplot(233); plt.imshow(thresh, 'gray'); plt.title("Otsu's Threshold"); plt.axis('off')

plt.subplot(234); plt.imshow(closing, 'gray'); plt.title("MorphologyEx: Closing (2x2)"); plt.axis('off')

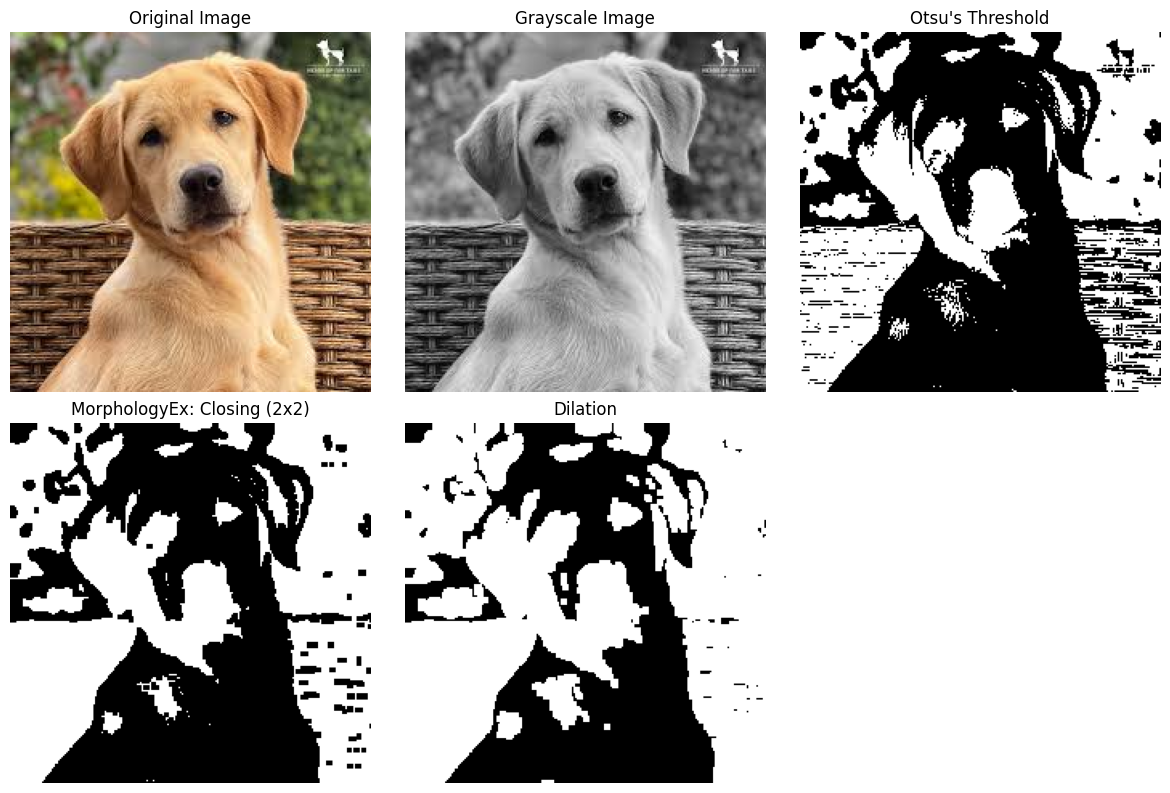
plt.subplot(235); plt.imshow(sure\_bg, 'gray'); plt.title("Dilation"); plt.axis('off')

plt.tight\_layout()

plt.show()

plt.imsave('dilation.png', sure\_bg)

**Output:**



**Experiment 17:**

**Aim:** To demonstrate linear separability using python code

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

X, y = make\_classification(n\_samples=100, n\_features=2, n\_informative=2, n\_redundant=0, n\_clusters\_per\_class=1, random\_state=42)

model = LogisticRegression()

model.fit(X, y)

y\_pred = model.predict(X)

accuracy = accuracy\_score(y, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

plt.figure(figsize=(10, 6))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='berlin', edgecolor='m', s=50)

coef = model.coef\_[0]

intercept = model.intercept\_

x\_vals = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)

y\_vals = - (coef[0] \* x\_vals + intercept) / coef[1]

plt.plot(x\_vals, y\_vals, 'k--')

plt.xlabel('Feature 1')

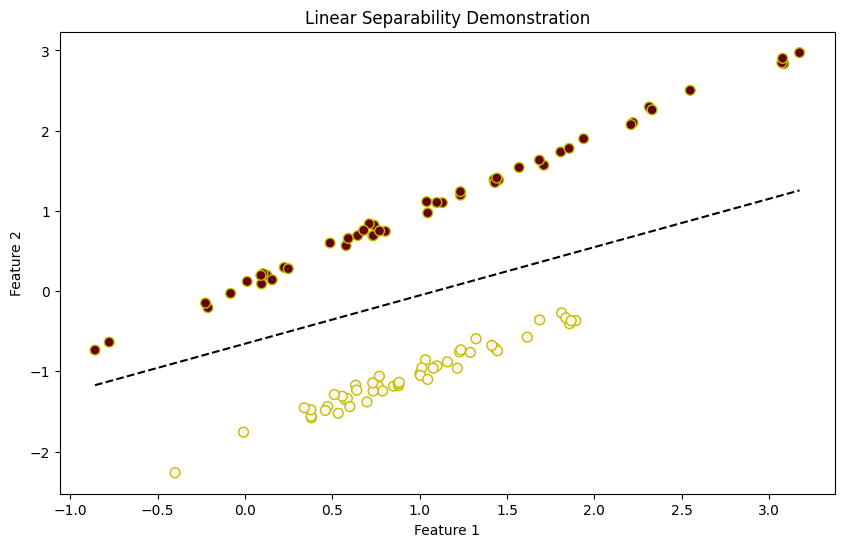
plt.ylabel('Feature 2')

plt.title('Linear Separability Demonstration')

plt.show()

i

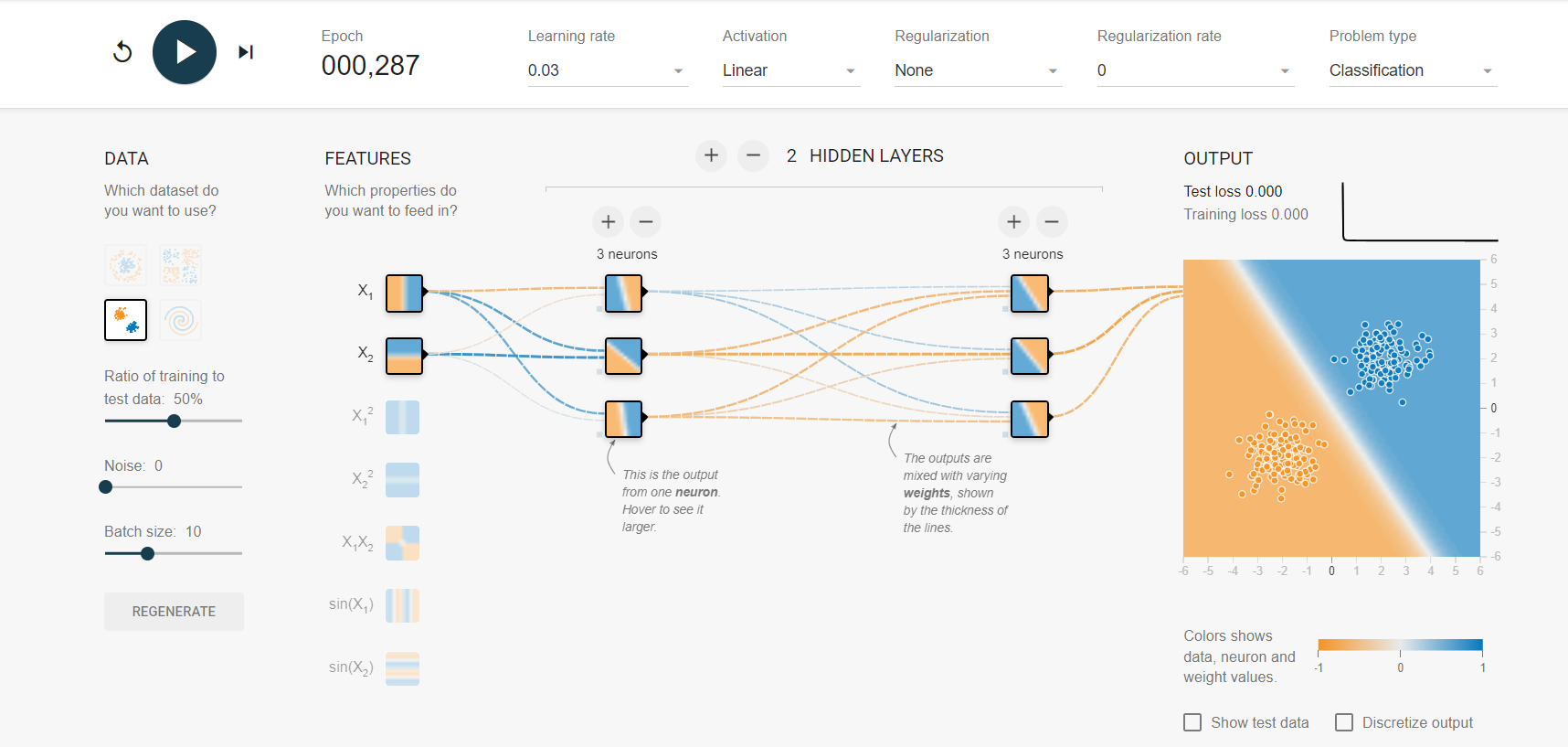
**Output:**



**Experiment 18:**

**Aim:** Neural network analysis for Two class, Learning rate: 0.03, Activation: Linear, Hidden Layers: 02, and Hidden neurons: 03.

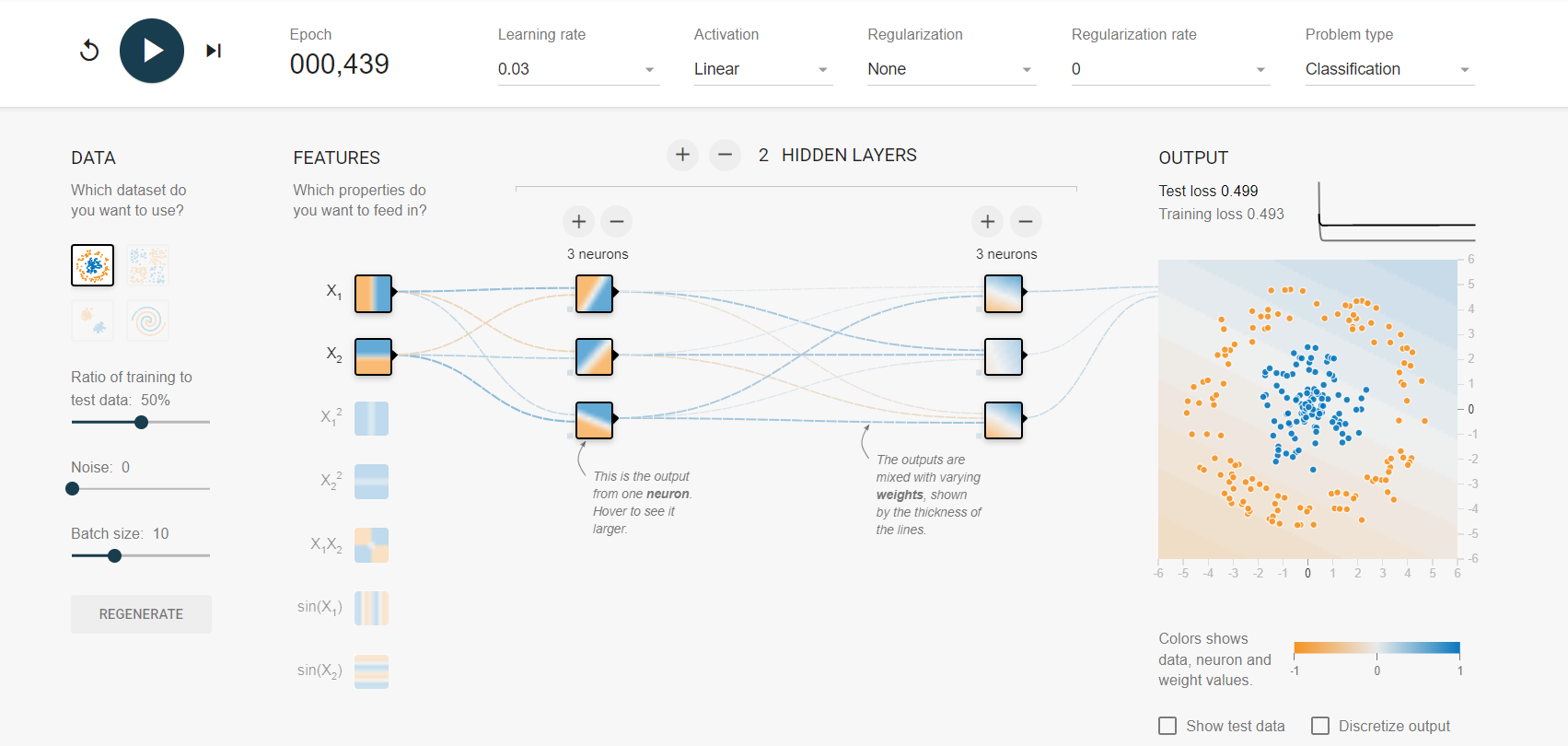
**Output:**

****

**Experiment 19:**

**Aim:** Neural network analysis for circular data class, Learning rate: 0.03, Activation: Linear, Hidden Layers: 02, and Hidden neurons: 03.

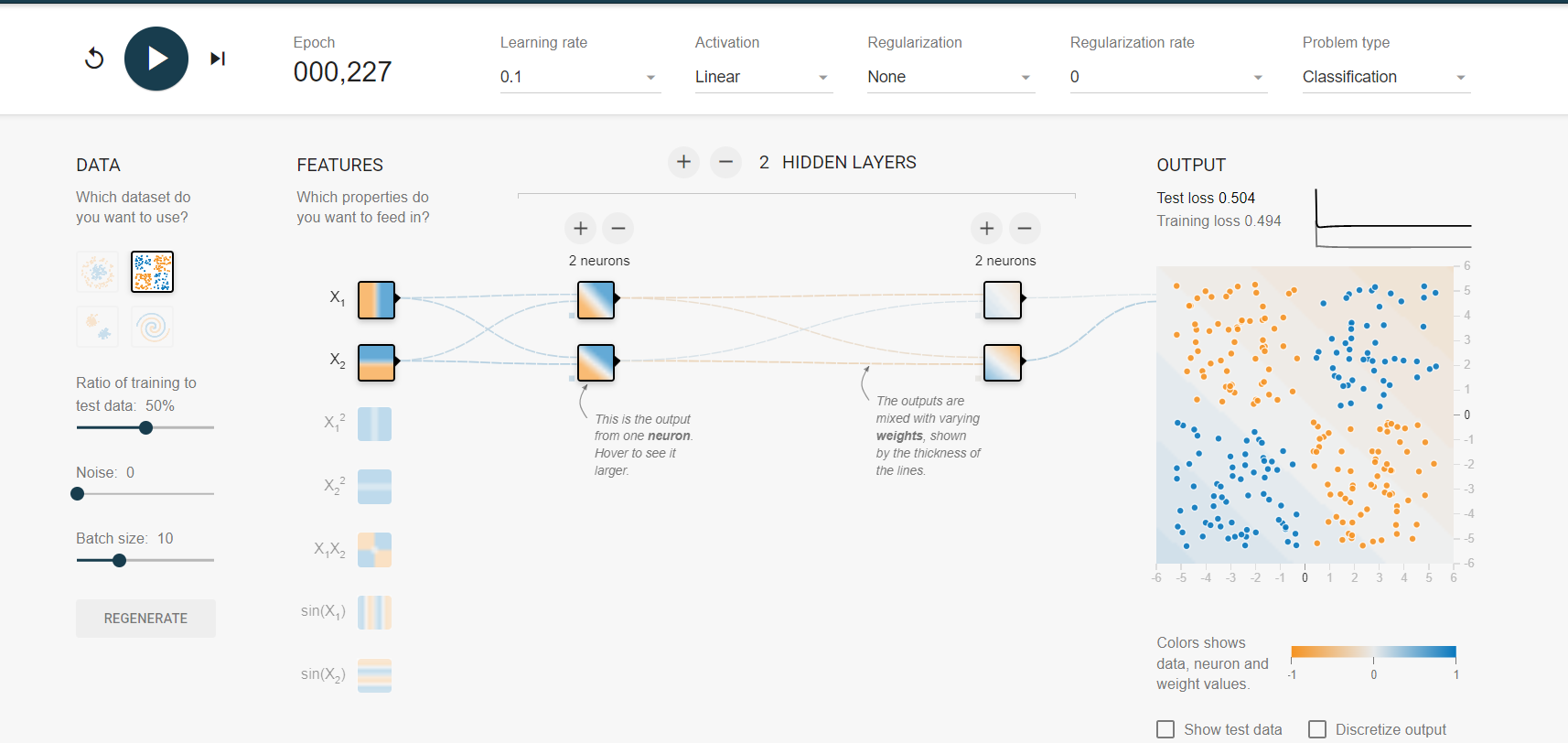
**Output:**

****

**Experiment 20:**

**Aim:** Neural network analysis for Multi class, Learning rate: 0.01, Activation: Linear, Hidden Layers: 02, and Hidden neurons: 02

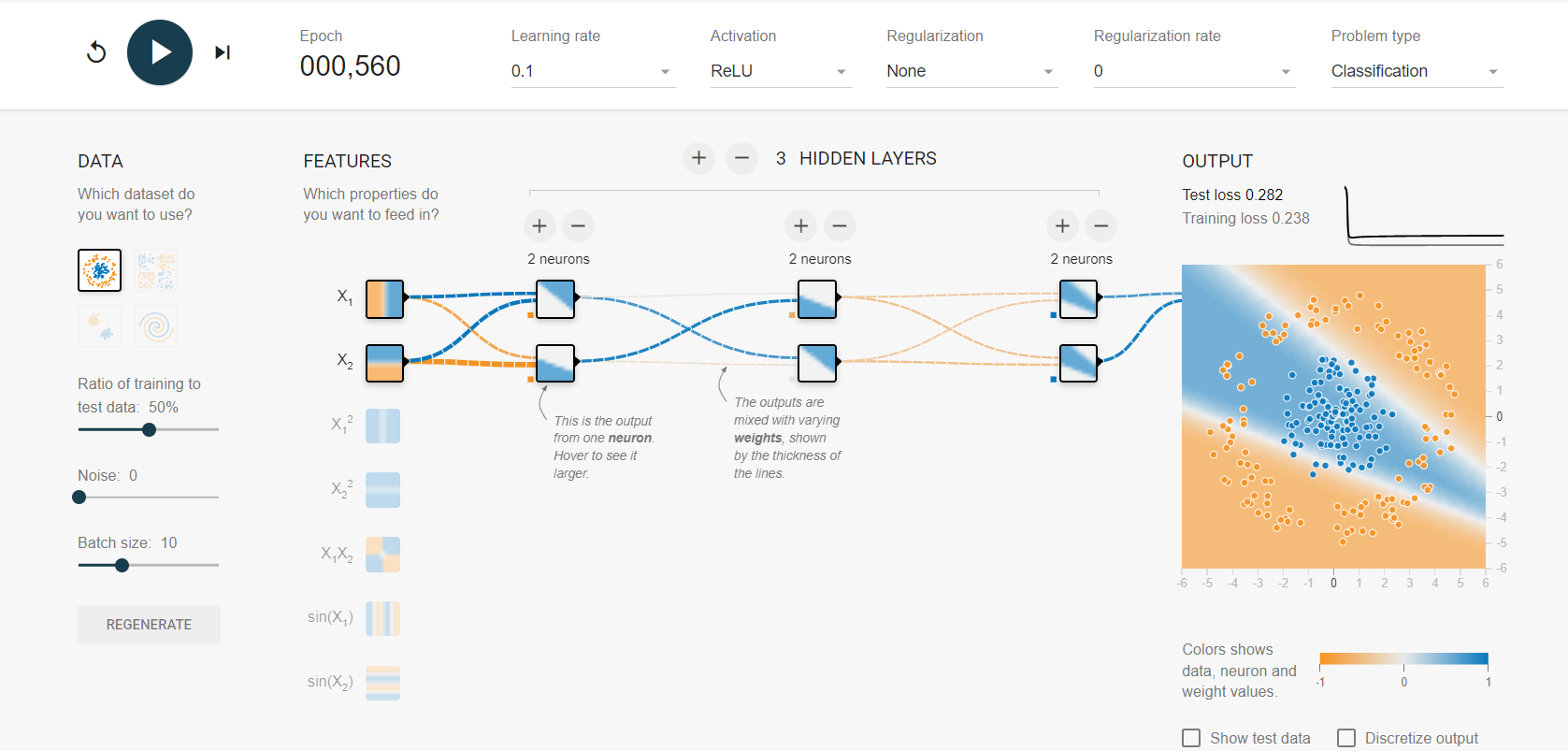
**Output:**

****

**Experiment 21:**

**Aim:** Neural network analysis forCircular data, Learning rate: 0.1, Activation: ReLU, Hidden Layers: 03, and Hidden neurons: 02.

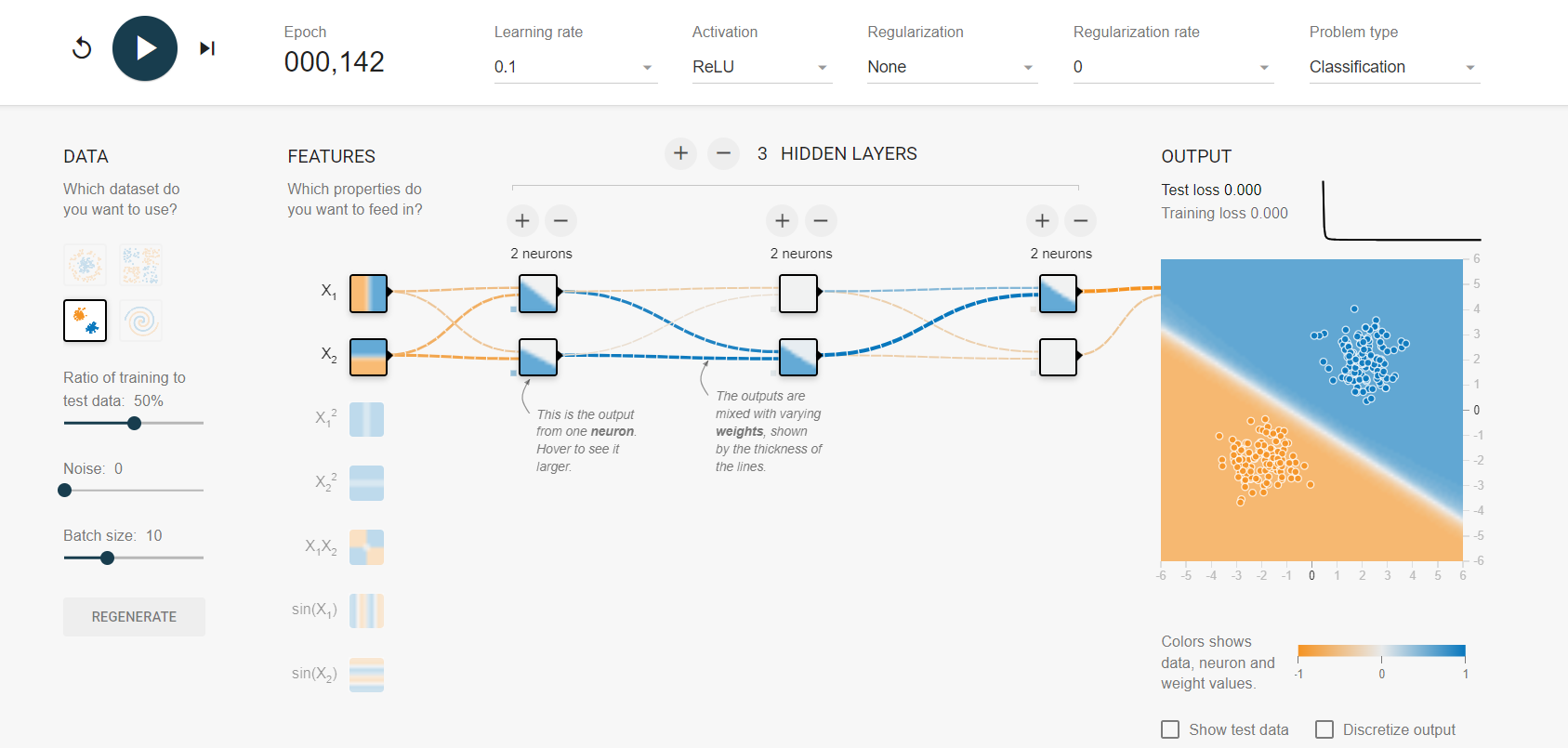
**Output:**

****

**Experiment 22:**

**Aim:** Neural network analysis fortwo class data, Learning rate: 0.1, Activation: ReLU, Hidden Layers: 03, and Hidden neurons: 02.

**Output:**

****

**Experiment 23:**

**Aim:** Neural network analysis forSpiral data, Learning rate: 0.1, Activation: Sigmoid, Hidden Layers: 03, and Hidden neurons: 03.

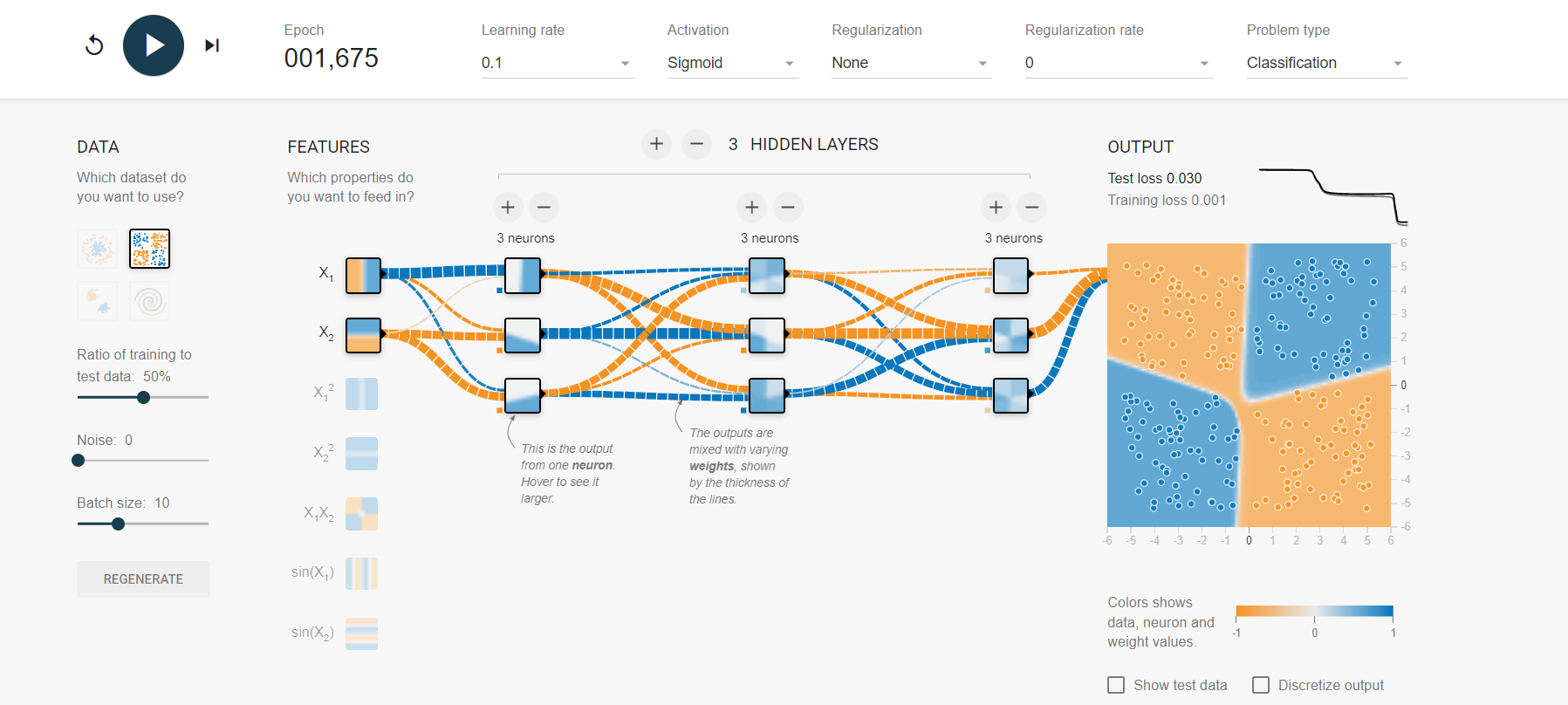
**Output:**

****

**Experiment 24:**

**Aim:** Neural network analysis formulti class data, Learning rate: 0.1, Activation: Sigmoid, Hidden Layers: 03, and Hidden neurons: 03.

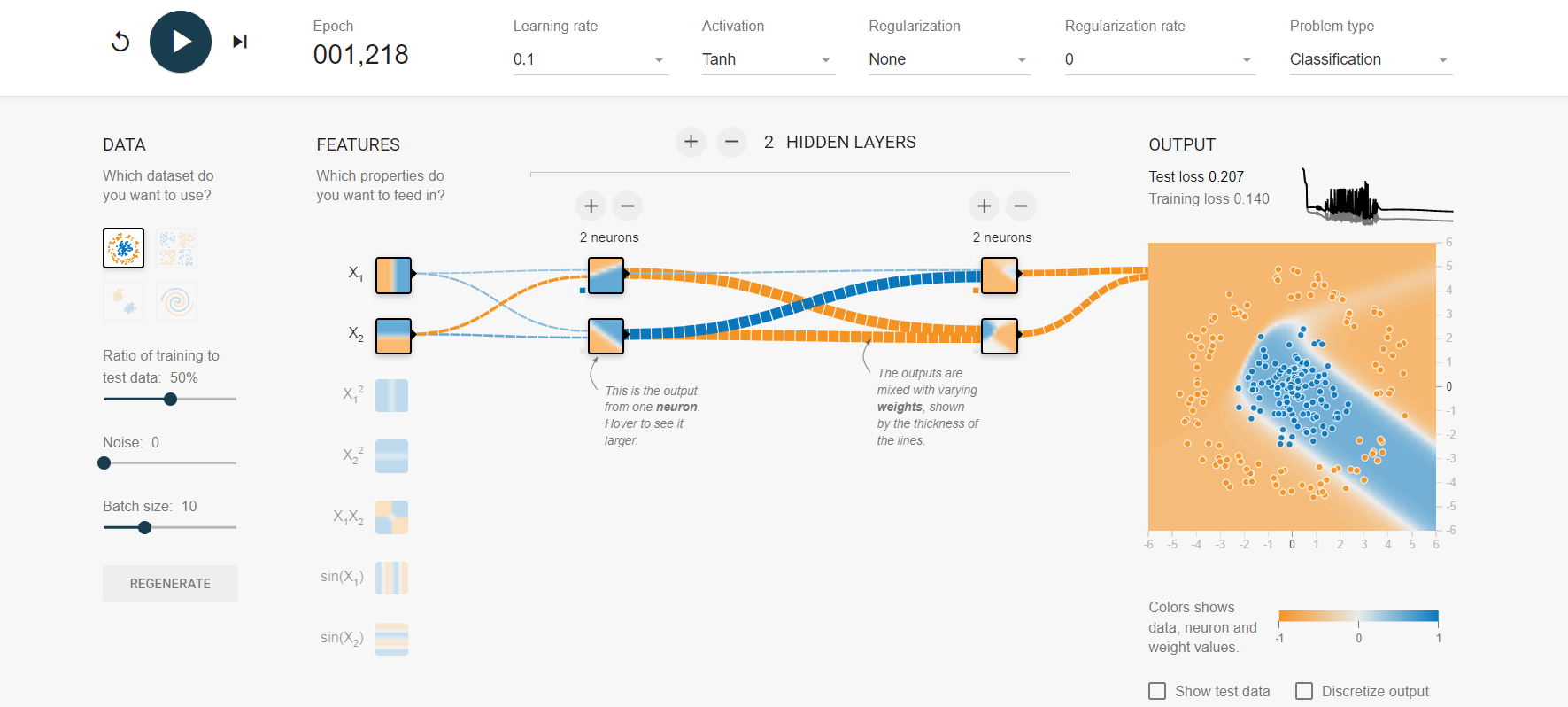
**Output:**

****

**Experiment 25:**

**Aim:** Neural network analysis for Circular data, Learning rate: 0.1, Activation: Tanh, Hidden Layers: 02, and Hidden neurons: 02.

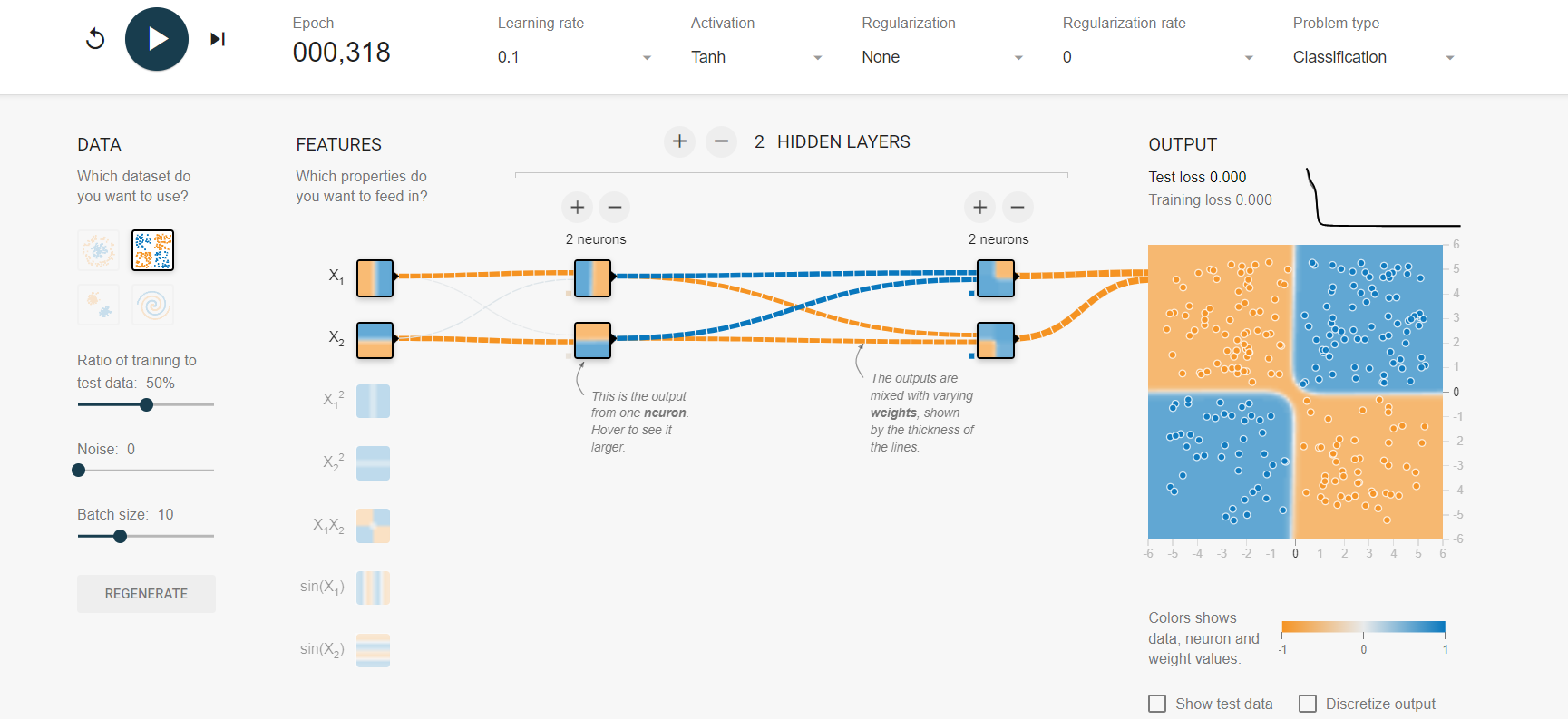
**Output:**

****

**Experiment 26:**

**Aim:** Neural network analysis for multi class data, Learning rate: 0.1, Activation: Tanh, Hidden Layers: 02, and Hidden neurons: 02.

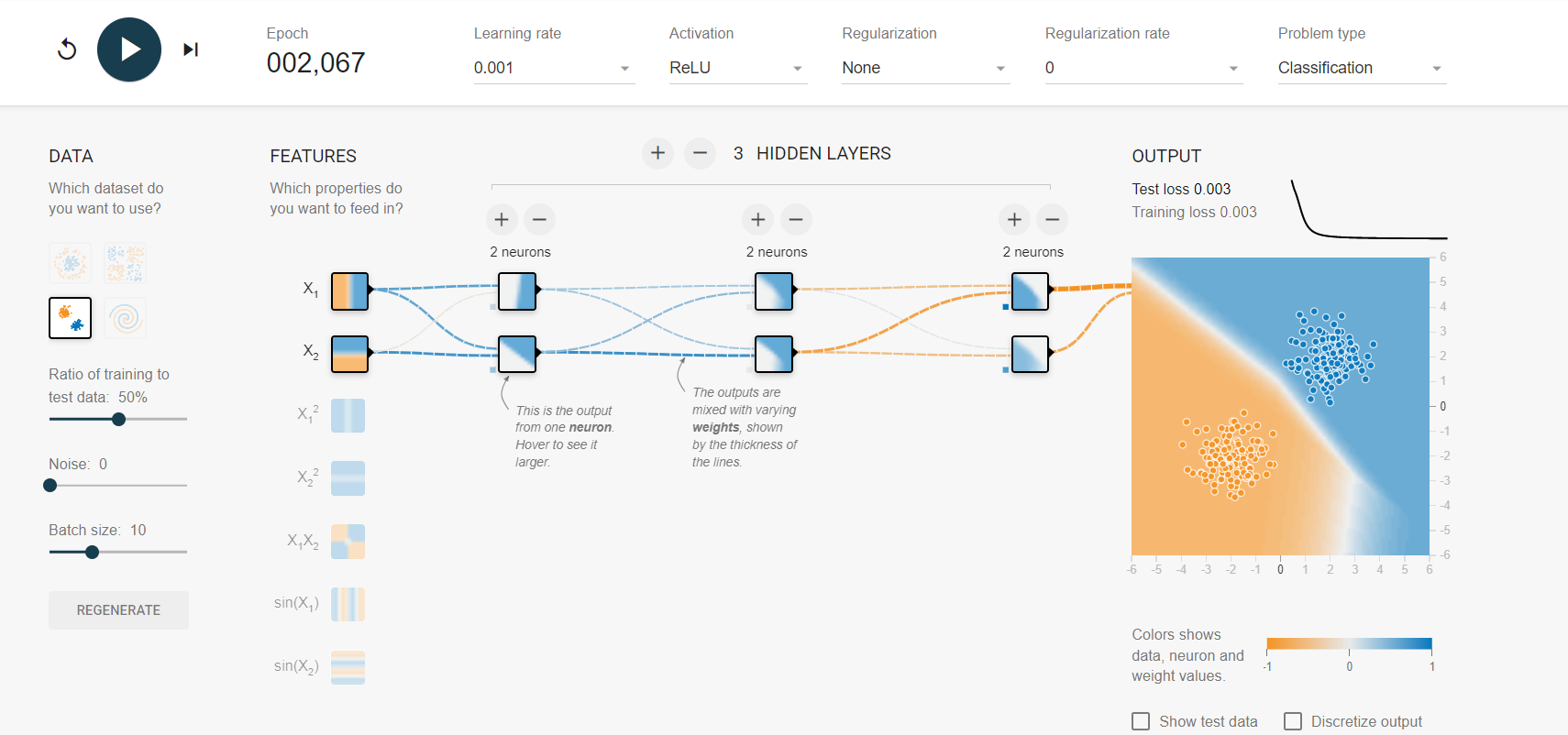
**Output:**

****

**Experiment 27:**

**Aim:** Neural network analysis for Two-class data, Learning rate: 0.001, Activation: ReLU, Hidden Layers: 03, and Hidden neurons: 02.

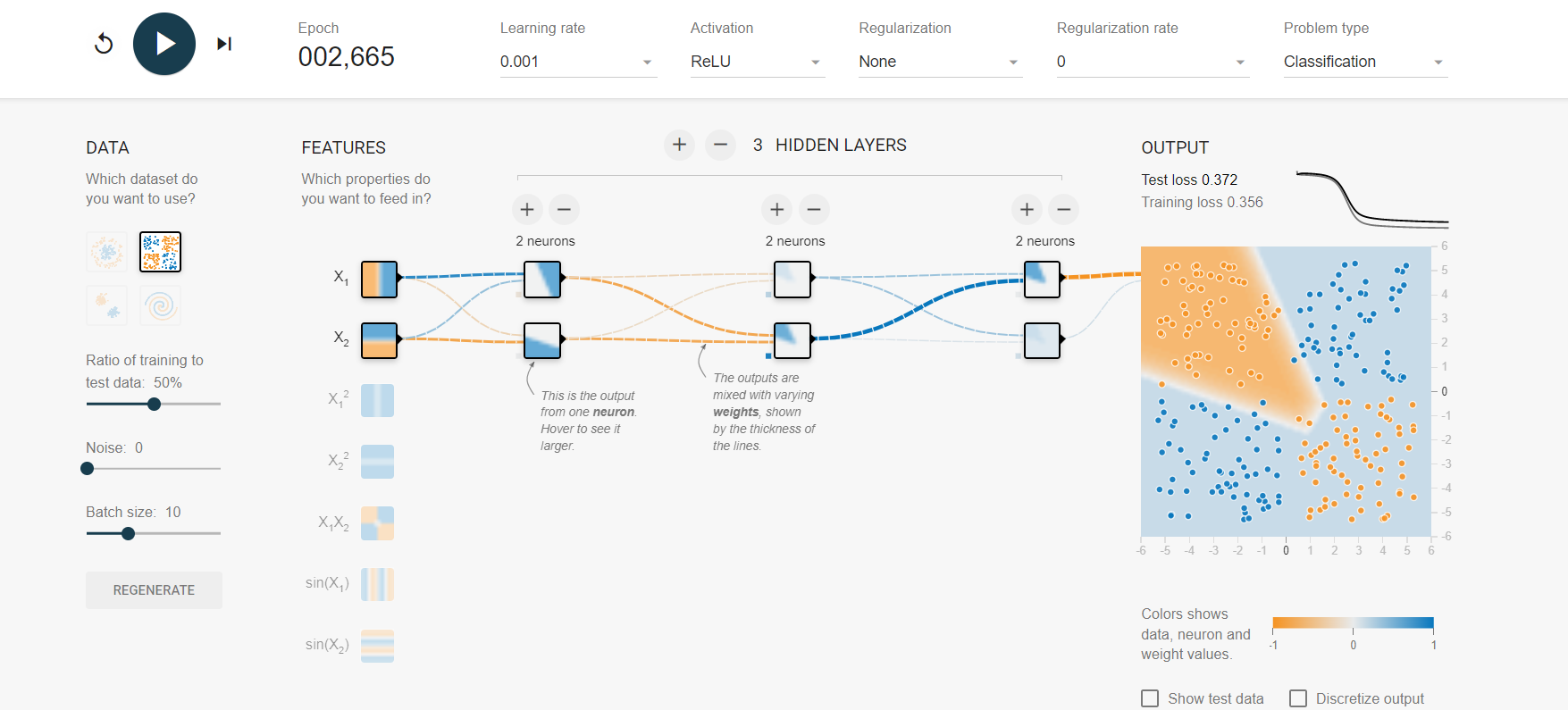
**Output:**

****

**Experiment 28:**

**Aim:** Neural network analysis for Multi class data, Learning rate: 0.001, Activation: ReLU, Hidden Layers: 03, and Hidden neurons: 02.

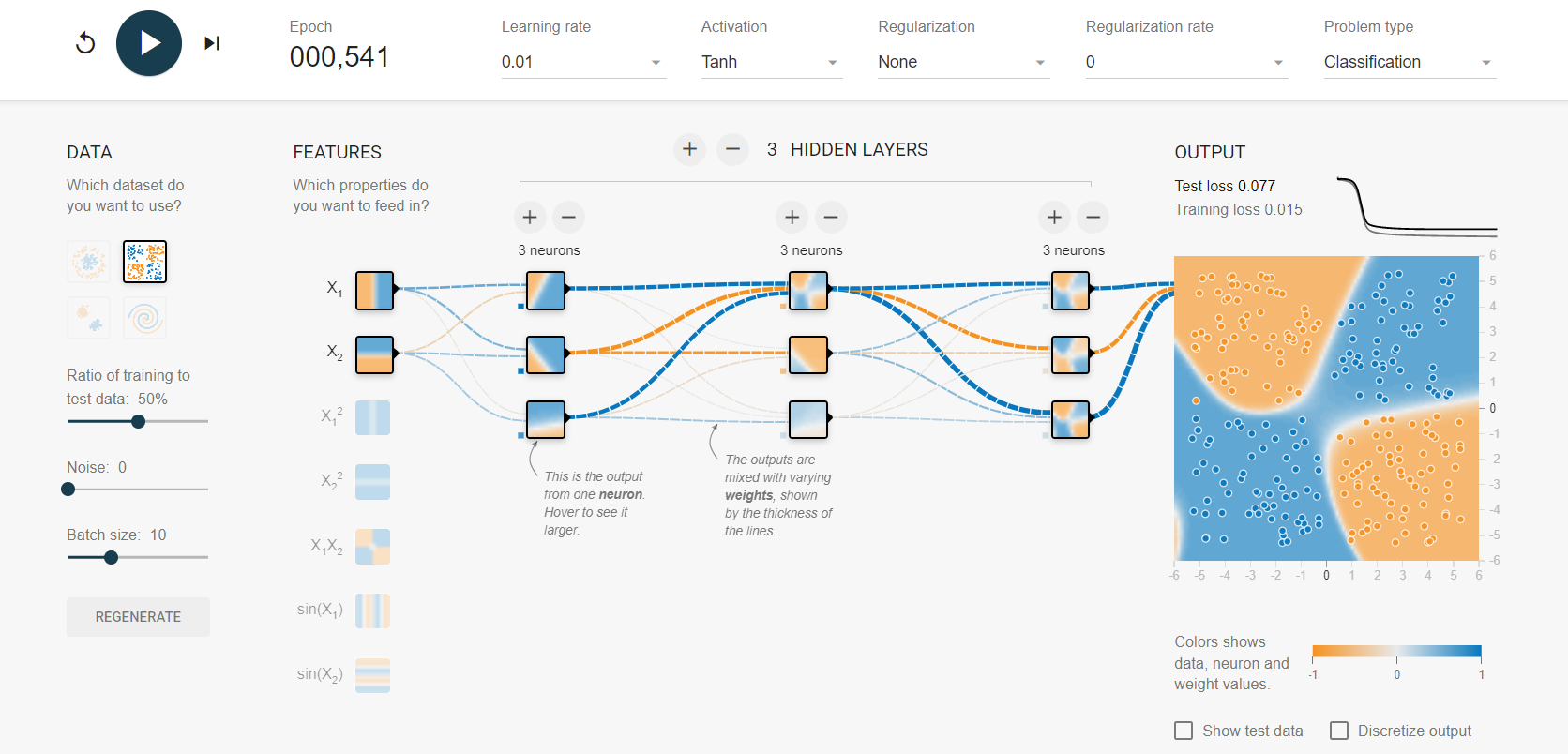
**Output:**

****

**Experiment 29:**

**Aim:** Neural network analysis for Multi-class data, Learning rate: 0.1, Activation: TanH, Hidden Layers: 03, and Hidden neurons: 03.

**Output:**

****

**Experiment 30:**

**Aim:** Neural network analysis for Two class data, Learning rate: 0.1, Activation: TanH, Hidden Layers: 03, and Hidden neurons: 03.

**Output:**

****

**Experiment 31:**

**Aim:** Neural network analysis for Multi-class data, Learning rate: 0.03, Activation: ReLu, Hidden Layers: 03, and Hidden neurons: 03.

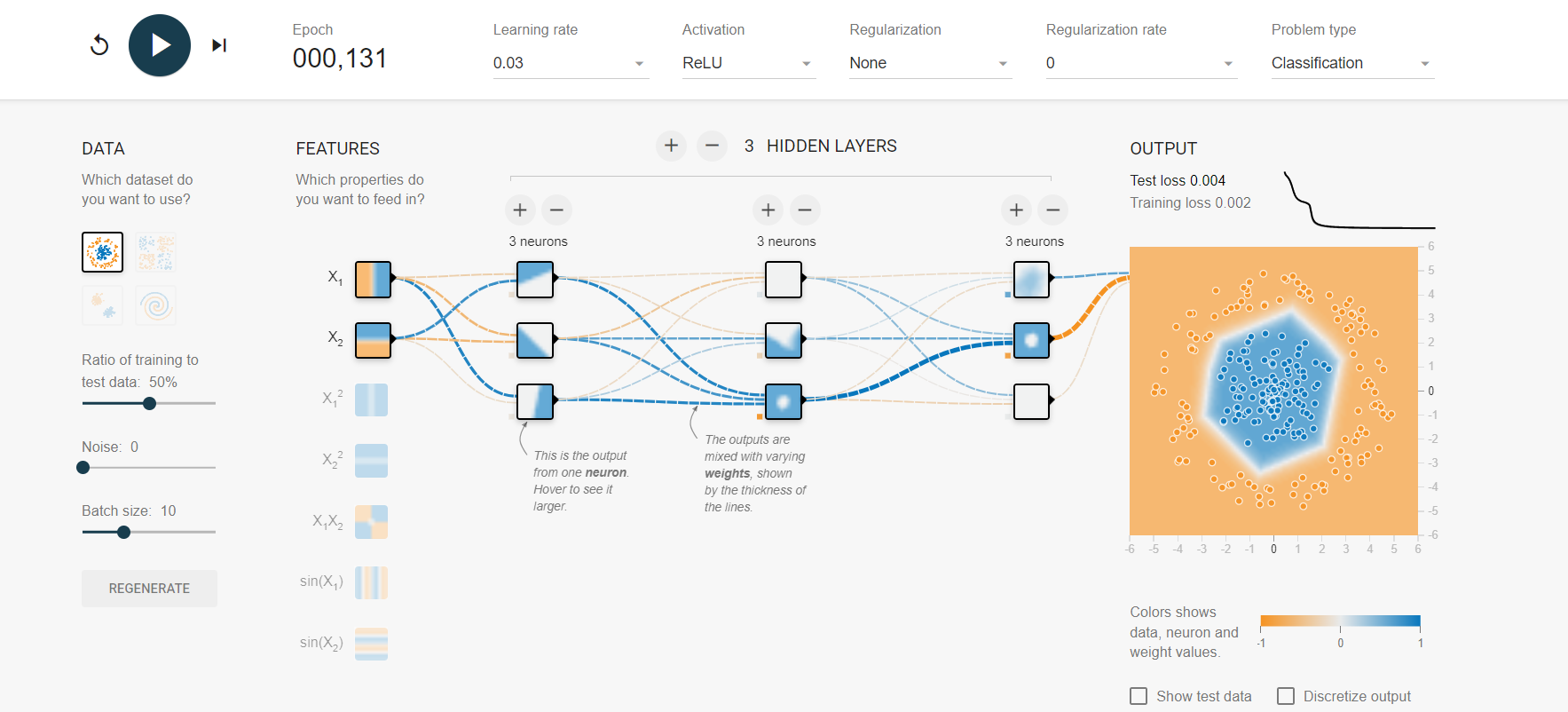
**Output:**

****

**Experiment 32:**

**Aim:** Neural network analysis for Two circular data, Learning rate: 0.1, Activation: TanH, Hidden Layers: 03, and Hidden neurons: 03.

**Output:**

****