Student Name: Prajan Shrestha

University ID: 2041402

You must use a dataset similar to that from Lab 1 for this part, but this time use the first 5 (or 6 for leading zeros) digits of your student number as the seed for random numbers.This is a classification task in which your neural network hopes to perform better than the decision tree method employed previously.

1. Train a scikit-learn MLP_Classifier to classify the dataset.

6CS012 Workshop 4

Question 1:

Train a scikit-learn MLPCLassifier to classify the dataset.

In [1]:
```
# Importing the required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [2]:
```
# Generating a random n-class classification problem using make_classification()
# Student Number: ____

features, target = make_classification(
    n_samples=200, n_features=4, n_classes=3, n_clusters_per_class=1, random_stat
```

In [3]:
```
# Getting total samples and feature

features.shape
```

Out[3]:
```
(200, 4)
```

In [4]:
```
# total targets for each samples

target.shape
```

Out[4]:
```
(200,)
```

In [5]:
```
# Getting first feature from the array

features[0]
```

Out[5]:
```
array([-1.03471842,  0.15814713,  0.31835032,  0.67579684])
```

In [6]:
```
# Getting the target of first feature

target[0]
```

Out[6]:
```
1
```

In [7]:
```
# Setting feature names and displaying them

features_names = ['feature_0', 'feature_1', 'feature_2', 'feature_3']
features_names
```

Out[7]:
```
['feature_0', 'feature_1', 'feature_2', 'feature_3']
```

In [8]:
```
# adding features to the dataframe

features_df = pd.DataFrame(features, columns=features_names)
features_df
```

Out[8]:

	feature_0	feature_1	feature_2	feature_3
0	-1.034718	0.158147	0.318350	0.675797
1	-2.239339	1.794615	1.759433	-0.036236
2	0.133316	-0.330796	-0.269813	0.233275
3	-2.137823	1.866880	1.792899	-0.193126
4	-1.739705	1.886982	1.730075	-0.536684
...	...	...	...	...
195	-1.769137	-0.227191	0.177561	1.668960
196	-1.075402	-0.630005	-0.254625	1.522141
197	-0.248169	-0.737942	-0.495504	0.962768
198	-1.813317	-0.656112	-0.129960	2.147421
199	0.275011	1.543833	1.084253	-1.816194

200 rows × 4 columns

In [9]:
```
# viewing the first 5 rows of the features dataframe

features_df.head()
```

Out[9]:

	feature_0	feature_1	feature_2	feature_3
0	-1.034718	0.158147	0.318350	0.675797
1	-2.239339	1.794615	1.759433	-0.036236
2	0.133316	-0.330796	-0.269813	0.233275
3	-2.137823	1.866880	1.792899	-0.193126
4	-1.739705	1.886982	1.730075	-0.536684

In [10]:
```
# Similarly, adding targets to the dataframe

targets_df = pd.DataFrame(target, columns=['target'])
targets_df
```

Out[10]:

	target
0	1
1	0
2	1
3	0
4	0
...	...
195	1
196	1
197	1
198	1
199	0

200 rows × 1 columns

In [11]:
```
# viewing the first 5 rows of the target dataframe

targets_df
```

Out[11]:

	target
0	1
1	0
2	1
3	0
4	0
...	...
195	1
196	1
197	1
198	1
199	0

200 rows × 1 columns

In [12]:
```
# Combining the two features and target dataframes to
# align each features to its respective targets

dataset = pd.concat([features_df, targets_df], axis=1)
```

In [13]:
```
# viewing the features with their respective targets in a
# single dataframe, dataset.

dataset
```

Out[13]:

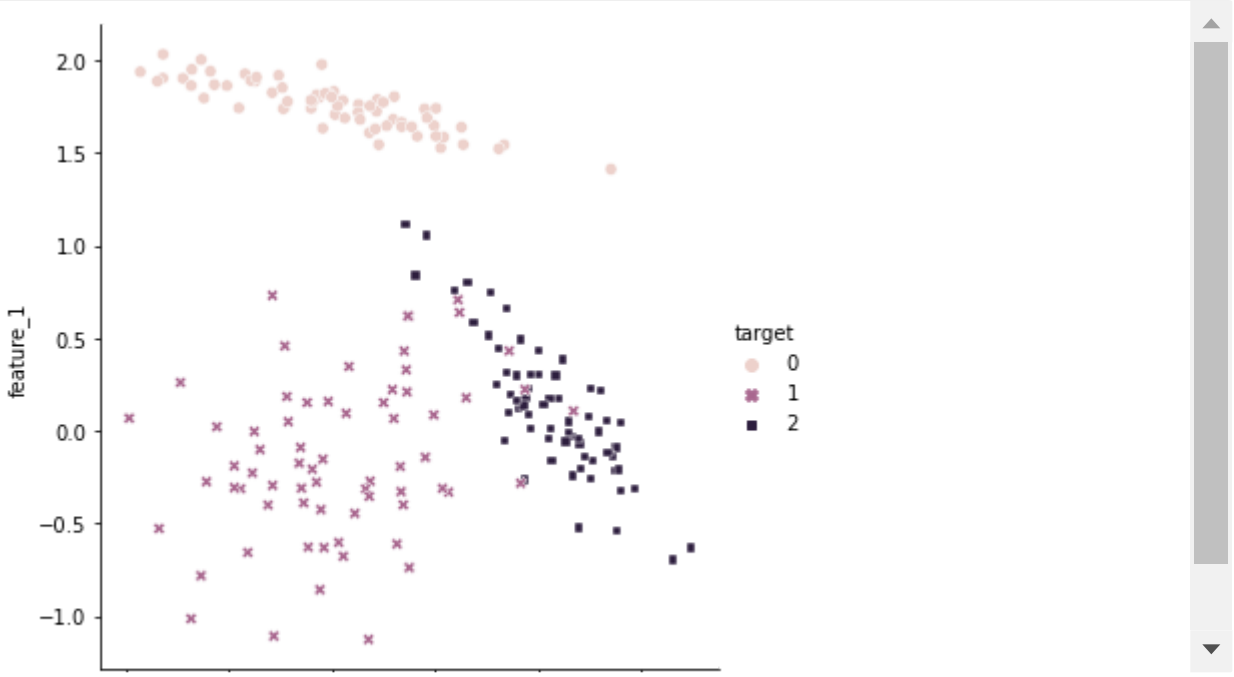
	feature_0	feature_1	feature_2	feature_3	target
0	-1.034718	0.158147	0.318350	0.675797	1
1	-2.239339	1.794615	1.759433	-0.036236	0
2	0.133316	-0.330796	-0.269813	0.233275	1
3	-2.137823	1.866880	1.792899	-0.193126	0
4	-1.739705	1.886982	1.730075	-0.536684	0
...	...	...	...	...	...
195	-1.769137	-0.227191	0.177561	1.668960	1
196	-1.075402	-0.630005	-0.254625	1.522141	1
197	-0.248169	-0.737942	-0.495504	0.962768	1
198	-1.813317	-0.656112	-0.129960	2.147421	1
199	0.275011	1.543833	1.084253	-1.816194	0

200 rows × 5 columns

Relationship Plots

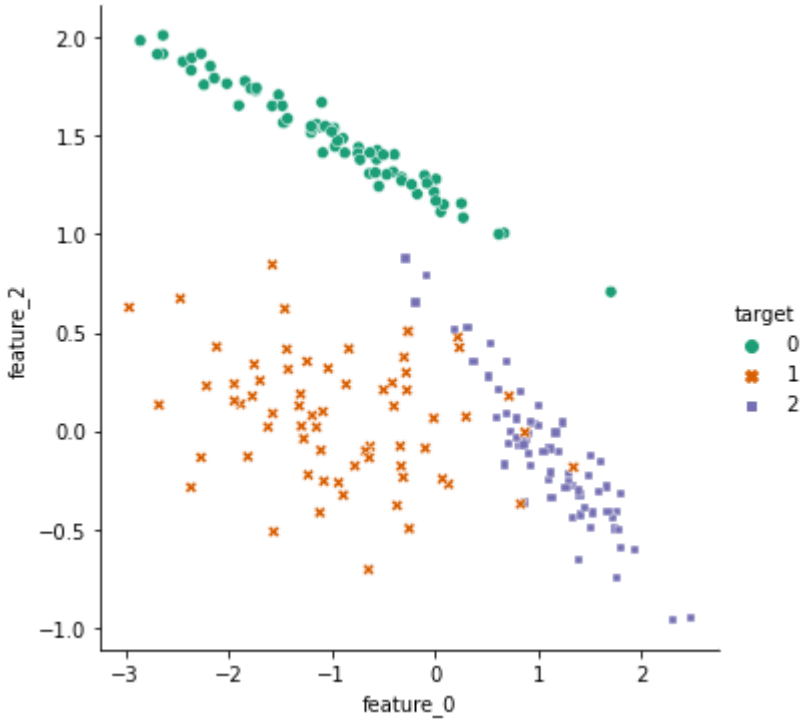
```
Here, the features were plotted and visulized their relationship between each
other. Statistical analysis is the process of understanding how different
variables are related to each other in a dataset and how they depend on other
variables. By visualizing the data properly, we can see different patterns and
trends which indicates the relationships.
```

In [14]:
```
# Relationship between feature_0 and feature_1
sns.relplot(
    x='feature_0', y='feature_1', hue='target', style='target', data=dataset)
plt.show()
```



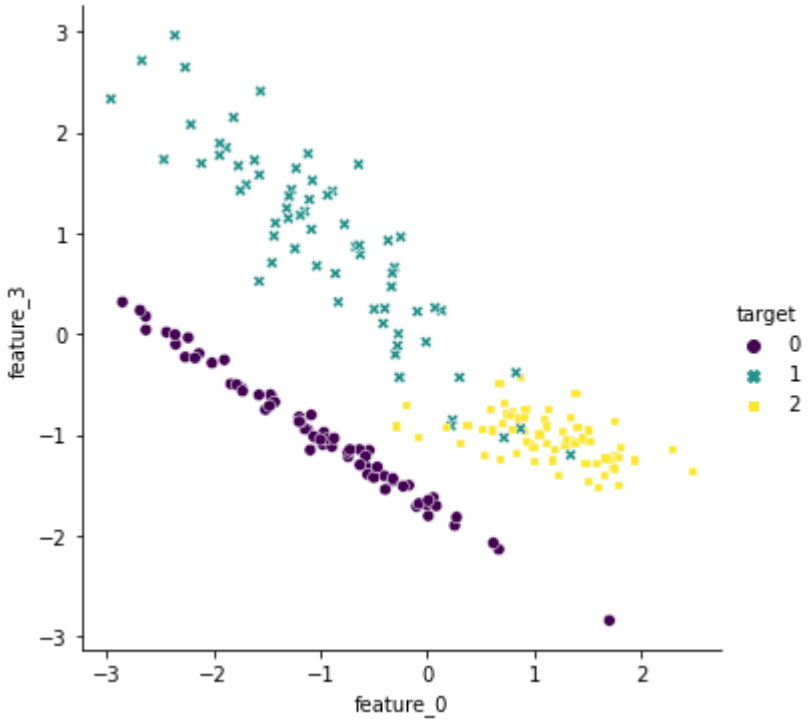
Seaborn.replot provides access to several different axes-level functions that show the relationship between two variables with semantic mappings of subsets. Here, it is marking the values of "feature_1" on different locations of "feature_0".

In [15]:
```
# Relationship between feature_0 and feature_2
sns.relplot(
    x='feature_0', y='feature_2', hue='target', style='target', palette='Dark2',
plt.show()
```



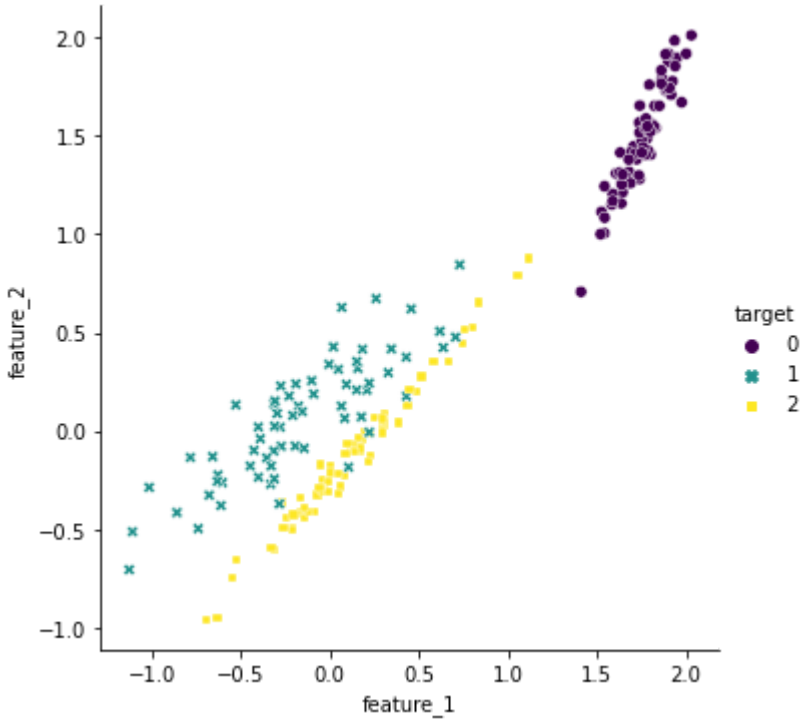
Here, it is marking the values of "feature_2" on different locations of "feature_0".

In [16]:
```
# Relationship between feature_0 and feature_3
sns.relplot(
    x='feature_0', y='feature_3', hue='target', style='target', palette='viridis'
plt.show()
```



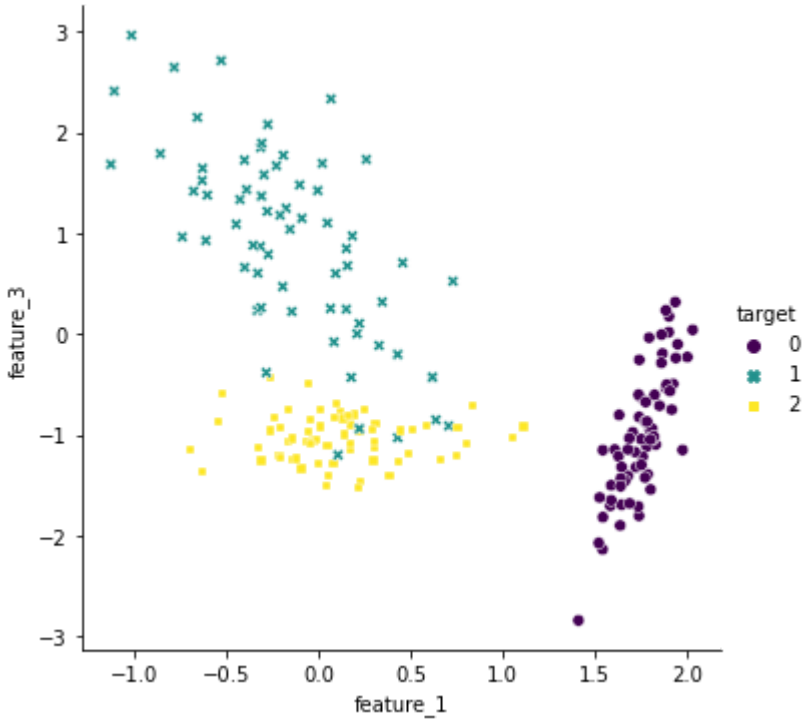
Here, it is marking the values of "feature_3" on different locations of "feature_0".

In [17]:
```
# Relationship between feature_1 and feature_2
sns.relplot(
    x='feature_1', y='feature_2', hue='target', style='target', palette='viridis'
plt.show()
```



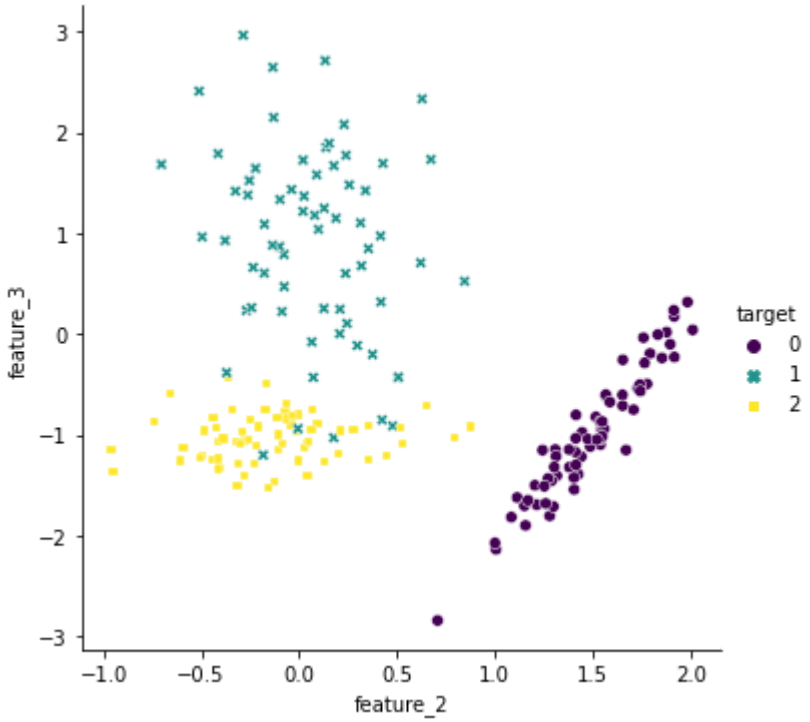
Here, it is marking the values of "feature_2" on different locations of "feature_1".

In [18]:
```
# Relationship between feature_1 and feature_3
sns.relplot(
    x='feature_1', y='feature_3', hue='target', style='target', palette='viridis'
plt.show()
```



Here, it is marking the values of "feature_3" on different locations of "feature_1".

In [19]:
```
# Relationship between feature_2 and feature_3
sns.relplot(
    x='feature_2', y='feature_3', hue='target', style='target', palette='viridis'
plt.show()
```



Here, it is marking the values of "feature_3" on different locations of "feature_2".

In [20]:
```
# Splitting the dataset into training and testing set

train_features, test_features, train_target, test_target = train_test_split(
    features, target, random_state=20414, test_size=0.1)
```

The "train_test_split(features, target, random_state=20414, test_size = 0.1)" function returns a tuple and this function is used to train and test data. Here, it is used to train and test data for "features" and "target" datasets. The random_state is assigned to 20414 and test_size is 0.1 which means out of 100%, 10% is kept for testing.

It is unpacked in four variables i.e. training_features, test_features, training_target and test_target.

In [21]:
```
# Showing the splition
print(train_features.shape, test_features.shape)
```

```
(180, 4) (20, 4)
```

It is showing the shape of train_features and test_features splition in the form of touple. Here training set is 180 by 4 whereas the testing set of data is 20 by 4.

In [22]:
```
# Using the MLPClassifier to train the model
# The DecisionTreeClassifier function was used earlier in Week 1
# to classify and this is also capable of performing
# multi-class classification on a given datasets.
# Our dataset contains 3 target labels,
# so it is also a multi class classification problem.
```

In [23]:
```
mlp = MLPClassifier(random_state=20414, max_iter=200)
mlp
```

Out[23]:
```
MLPClassifier(random_state=20414)
```

MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. MLPClassifier relies on an underlying Neural Network to perform the task of classification.

In [24]:
```
# Fitting the data into the MLP_Classifier model
# Now, we fit the decision tree classifier model.
# Fitting is same as training and after the model,
# is trained the model can used to make predictions.

model = mlp.fit(train_features, train_target)
model
```

```
C:\ai\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: C
onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and t
he optimization hasn't converged yet.
  warnings.warn(
```

Out[24]:
```
MLPClassifier(random_state=20414)
```

In [25]:
```
# Predicting for the test features to test the performance,
# of our MLP_Classifier model.

predictions = model.predict(test_features)
predictions
```

Out[25]:
```
array([1, 1, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 0, 0])
```

In [26]:
```
# Creating confusion matrix from predictions

matrix = confusion_matrix(predictions, test_target)
matrix
```

Out[26]:
```
array([[9, 0, 0],
       [0, 8, 0],
       [0, 0, 3]], dtype=int64)
```

In [27]:
```
# Displaying the confusion matrix
print(matrix)
```

```
[[9 0 0]
 [0 8 0]
 [0 0 3]]
```

In [28]:
```
# showing the classification report for the predictions

print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         3

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

classification_report builds a text report showing the main classification metrics

Accuracy

Accuracy is the most commonly used matrix to evaluate the model which is actually not a clear indicator of the performance.

Accuracy = (9+8+3)/20 = 1

Error

Error is used to know what fraction of predictions were incorrect.

Error = (1-Accuracy) = (1-1) = 0

Precision / Positive Predicted Value

Precision is the percentage of positive instances out of the total predicted positive instances which means precision or positive predicted value means how much model is right when it says it is right.

Precision of Class 0 = 9/(9+0+0) = 1 Precision of Class 1 = 8/(8+0+0) = 1 Precision of Class 2 = 3/(3+0+0) = 1

Recall / True Positive Rate / Sensitivity

Recall defines how many of the true positives were recalled (found), i.e. how many of the correct hits were also found.

Recall of Class 0 = 9/(9+0+0) = 1 Recall of Class 1 = 8/(8+0+0) = 1 Recall of Class 2 = 3/(3+0+0) = 1

F1-Score

F1- Score is the harmonic mean of the precision and recall which means the higher the value of the f1-score better will be the model.

F1-Score of Class 0 = 2PR/P+R = 2x1x1/1+1 = 1 F1-Score of Class 1 = 2PR/P+R = 2x1x1/1+1 = 1 F1-Score of Class 1 = 2PR/P+R = 2x1x1/1+1 = 1

Macro Average

Macro Average for precision = (Precision of Class 0 + Precision of Class 1 + Precision of Class 2) / 3 = (1+1+1) / 3 = 1

Macro Average for recall = (Recall of Class 0 + Recall of Class 1 + Recall of Class 2) / 3 = (1+1+1) / 3 = 1

Macro Average for f1-score = (F1-Score of Class 0 + F1-Score of Class 1 + F1-Score of Class 2) / 3 = (1+1+1) / 3 = 1

Weighted Average

Weighted Average for precision = (1x9+1x8+1x3) / 9+8+3 = 20 / 20 = 1

Weighted Average for recall = (1x9+1x8+1x3) / 9+8+3 = 20 / 20 = 1

Weighted Average for f1-score = (1x9+1x8+1x3) / 9+8+3 = 20 / 20 = 1

Question 2:

Write a paragraph to explain how the confusion matrix and other metrics regard the MPL or decision tree to be most applicable.

```
A confusion matrix is a table that is used to describe the performance of a
classification model or classifier on a set of test data for which the true
values are known. The confusion matrix is a functional function that can be
used in both machine learning and deep learning models to analyze problem
classification and assign it to the appropriate solution stage. Confusion
matrix is extremely useful to measure Recall, Precision, Specificity, Accuracy
and most importantly AUC-ROC Curve. In comparison to the classification of
similar dataset i.e. samples=200, n_features=4, n_classes=3,
n_clusters_per_class=1, random_state=20414 by using two different models i.e.
Multi Layer Perceptron and Decision Tree. I have found outputs from both models
which are as follows:
```

```
For MLP Classifier:
    I have found accuracy of 100% with different percentage of precision,
recall and f1 score value.

For Decision Tree Classifier:
     I have found accuracy of 80% with different percentage of precision,
recall and f1 score value.
```

Question 3:

Experiment with 3 hyper-parameters included in the lecture and write a short summary of what you have learnt.

```
As we already know that MLP is best for this classification task, we can
experiment changing some hyper-parameters to see if there will be some
improvement in the performance of the model.
```

Experiment 1:

```
Changing the following paramaters:
Hidden Layer: 500
batch_size: auto
activation function: relu
loss function: adam
```

In [29]:
```
mlp = MLPClassifier(
    random_state=20414, hidden_layer_sizes=100, batch_size='auto', activation='re
mlp
```

Out[29]:
```
MLPClassifier(hidden_layer_sizes=100, random_state=20414)
```

This function is defined in the sklearn.neural network library, which uses the latest Neural Network technology in Artificial Intelligence. Different values are used to parameterize this function in order to construct an effective model. All of these values combined will aid in the classification of any data with greater accuracy.

The model will work with 500 dense layers inside, which appears to be similar to the architecture of our nervous system to process any information, with the hidden layer size parameter set to 500. Although it is true that increasing the dense layer will aid in the development of a well-accurate model, it is also true that if a large number of layers are provided to train a simple model, it will often overfit. As a result, neural networks are best suited to datasets with a high volume of data when compared to machine learning models.

In a neural network, the activation function is one of the most important parameters to define. The activation functions are designed to convert into a nonlinear function that can be used to track the real-world situation. i.e. the real world only has non-linear inputs that must be handled.

The activation function relu is used in this case. relu is a piecewise linear function that, if the input is positive, outputs the input directly; otherwise, it outputs 0.

Adam is used as the solver in this case, which will aid in the optimization process, which is crucial in neural networks.

Batch size is set to auto in this case, indicating the number of samples to process before updating the internal model parameters. Consider a batch to be a loop that iterates over one or more samples and makes predictions.

In [30]:
```
model_2 = mlp.fit(train_features, train_target)
model_2
```

```
C:\ai\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: C
onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and t
he optimization hasn't converged yet.
  warnings.warn(
```

Out[30]:
```
MLPClassifier(hidden_layer_sizes=100, random_state=20414)
```

In [31]:
```
predictions_2 = model.predict(test_features)
predictions_2
```

Out[31]:
```
array([1, 1, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 0, 0])
```

In [32]:
```
matrix_2 = confusion_matrix(predictions, test_target)
matrix_2
```

Out[32]:
```
array([[9, 0, 0],
       [0, 8, 0],
       [0, 0, 3]], dtype=int64)
```

In [33]:
```
print(matrix_2)
```

```
[[9 0 0]
 [0 8 0]
 [0 0 3]]
```

In [34]:
```
print(classification_report(test_target, predictions_2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         3

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Experiment 2:

```
Changing the following paramaters:
Hidden Layer: 350
batch_size: auto
learning rate: adaptive
activation function: relu
loss function: sgd
```

In [35]:
```
mlp = MLPClassifier(
    random_state=20414, hidden_layer_sizes=350, batch_size='auto', learning_rate=
mlp
```

Out[35]:
```
MLPClassifier(hidden_layer_sizes=350, learning_rate='adaptive',
              random_state=20414, solver='sgd')
```

This function is defined in the sklearn.neural network library, which uses the latest Neural Network technology in Artificial Intelligence. Different values are used to parameterize this function in order to construct an effective model. All of these values combined will aid in the classification of any data with greater accuracy.

The model will work with 350 dense layers inside, which appears to be similar to the architecture of our nervous system to process any information, with the hidden layer size parameter set to 350. Although it is true that increasing the dense layer will aid in the development of a well-accurate model, it is also true that if a large number of layers are provided to train a simple model, it will often overfit. As a result, neural networks are best suited to datasets with a high volume of data when compared to machine learning models.

In a neural network, the activation function is one of the most important parameters to define. The activation functions are designed to convert into a nonlinear function that can be used to track the real-world situation. i.e. the real world only has non-linear inputs that must be handled.

The activation function relu is used in this case. relu is a piecewise linear function that, if the input is positive, outputs the input directly; otherwise, it outputs 0.

sgd is used as the solver here which will help in optimization and is important in neural network.

Batch size is set to auto in this case, indicating the number of samples to process before updating the internal model parameters. Consider a batch to be a loop that iterates over one or more samples and makes predictions.

In [36]:
```
model_3 = mlp.fit(train_features, train_target)
model_3
```

```
C:\ai\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: C
onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and t
he optimization hasn't converged yet.
  warnings.warn(
```

Out[36]:
```
MLPClassifier(hidden_layer_sizes=350, learning_rate='adaptive',
              random_state=20414, solver='sgd')
```

In [37]:
```
predictions_3 = model.predict(test_features)
predictions_3
```

Out[37]:
```
array([1, 1, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 0, 0])
```

In [38]:
```
matrix_3 = confusion_matrix(predictions, test_target)
matrix_3
```

Out[38]:
```
array([[9, 0, 0],
       [0, 8, 0],
       [0, 0, 3]], dtype=int64)
```

In [39]:
```
print(matrix_3)
```

```
[[9 0 0]
 [0 8 0]
 [0 0 3]]
```

In [40]:
```
print(classification_report(test_target, predictions_3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         3

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Experiment 3:

```
Changing the following paramaters:
Hidden Layer: 400
batch_size: auto
learning rate: invscaling
activation function: relu
solver: sgd
loss function:
max_iter: 500
```

In [41]:
```
mlp = MLPClassifier(
    random_state=20414, hidden_layer_sizes=400, batch_size='auto', learning_rate=
    solver='sgd', max_iter=500)
mlp
```

Out[41]:
```
MLPClassifier(hidden_layer_sizes=400, learning_rate='invscaling', max_iter=500,
              random_state=20414, solver='sgd')
```

This function is defined in the sklearn.neural network library, which uses the latest Neural Network technology in Artificial Intelligence. Different values are used to parameterize this function in order

to construct an effective model. All of these values combined will aid in the classification of any data with greater accuracy.

The model will work with 400 hidden layers inside, which appears to be similar to the architecture of our nervous system to process any information, with the hidden layer size parameter set to 400. Although it is true that increasing the dense layer will aid in the development of a well-accurate model, it is also true that if a large number of layers are provided to train a simple model, it will often overfit. As a result, neural networks are best suited to datasets with a high volume of data when compared to machine learning models.

In a neural network, the activation function is one of the most important parameters to define. The activation functions are designed to convert into a nonlinear function that can be used to track the real-world situation. i.e. the real world only has non-linear inputs that must be handled.

The activation function relu is used in this case. relu is a piecewise linear function that, if the input is positive, outputs the input directly; otherwise, it outputs 0.

sgd is used as the solver here which will help in optimization and is important in neural network.

Here, after changing the layers to 400, learning rate to adaptive, and solver as sgd optimiser and batch size to auto, we can find increase an accuracy from 86% to 1%. It is due to limited amount of hidden layer used for this smaller dataset and adaptive learning rate that suits to a model.

In [42]:
```
model_4 = mlp.fit(train_features, train_target)
model_4
```

Out[42]:
```
MLPClassifier(hidden_layer_sizes=400, learning_rate='invscaling', max_iter=500,
              random_state=20414, solver='sgd')
```

In [43]:
```
predictions_4 = model.predict(test_features)
predictions_4
```

Out[43]:
```
array([1, 1, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 0, 0])
```

In [44]:
```
matrix_4 = confusion_matrix(predictions, test_target)
matrix_4
```

Out[44]:
```
array([[9, 0, 0],
       [0, 8, 0],
       [0, 0, 3]], dtype=int64)
```

In [45]:
```
print(matrix_4)
```

```
[[9 0 0]
 [0 8 0]
 [0 0 3]]
```

In [46]:
```
print(classification_report(test_target, predictions_4))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         3

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Here, in all the classification reports of experiment 1, 2 and 3, all the values are same.

End Of Assignment!!