```
Student name: Sunil Kumar Shrestha
University ID:1928580
```

6CS012 Workshop 07

Question 1:

Train a scikit-learn MLPCLassifier to classify the dataset.

In [101]:
```python
# 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 [89]:
```python
# Generating a random n-class classification problem using make_classification()
# Student Number: 1928580
features, target = make_classification(n_samples = 200,
                                       n_features = 4,
                                       n_classes = 3,
                                       n_clusters_per_class = 1,
                                       random_state = 19285)
```

In [90]:
```python
# Getting total samples and feature
features.shape
```

Out[90]: (200, 4)

In [91]:
```python
# total targets for each samples
target.shape
```

Out[91]: (200,)

In [92]:
```python


# Getting first feature from the array
features[0]
```

Out[92]: array([0.89241978, 1.51273855, -1.90157752, 1.07465425])

In [93]:
```
# Getting the target of first feature
target[0]
```
Out[93]: 1

In [94]:
```
# Setting feature names and displaying them
feature_names = ['feature_0', 'feature_1', 'feature_2', 'feature_3']
feature_names
```
Out[94]: ['feature_0', 'feature_1', 'feature_2', 'feature_3']

In [95]:
```
# adding features to the dataframe
features_df = pd.DataFrame(features, columns = feature_names)
```

In [96]:
```
# viewing the first 5 rows of the features dataframe
features_df.head()
```
Out[96]:

	feature_0	feature_1	feature_2	feature_3
0	0.892420	1.512739	-1.901578	1.074654
1	-1.250046	0.552437	-0.804774	-0.279702
2	2.158932	1.583991	-1.905415	1.647522
3	0.529968	0.989213	-1.247237	0.679877
4	3.975900	1.957094	-2.262620	2.593650

In [97]:
```
# Similarly, adding targets to the dataframe
target_df = pd.DataFrame(target, columns=['target'])
```

In [98]:
```
# viewing the first 5 rows of the target dataframe
target_df.head()
```
Out[98]:

	target
0	1
1	2
2	1
3	1
4	1

In [99]:
```
# Combining the two features and target dataframes to
# align each features to its respective targets
dataset = pd.concat([features_df, target_df], axis=1)
```

In [100]:
```
# viewing the features with their respective targets in a
# single dataframe, dataset.
dataset.head()
```

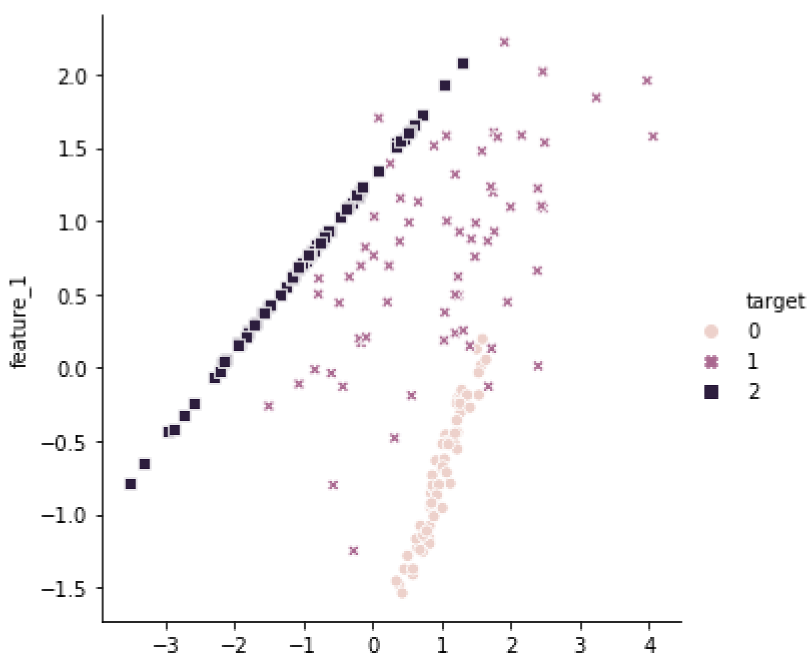
Out[100]:

	feature_0	feature_1	feature_2	feature_3	target
0	0.892420	1.512739	-1.901578	1.074654	1
1	-1.250046	0.552437	-0.804774	-0.279702	2
2	2.158932	1.583991	-1.905415	1.647522	1
3	0.529968	0.989213	-1.247237	0.679877	1
4	3.975900	1.957094	-2.262620	2.593650	1

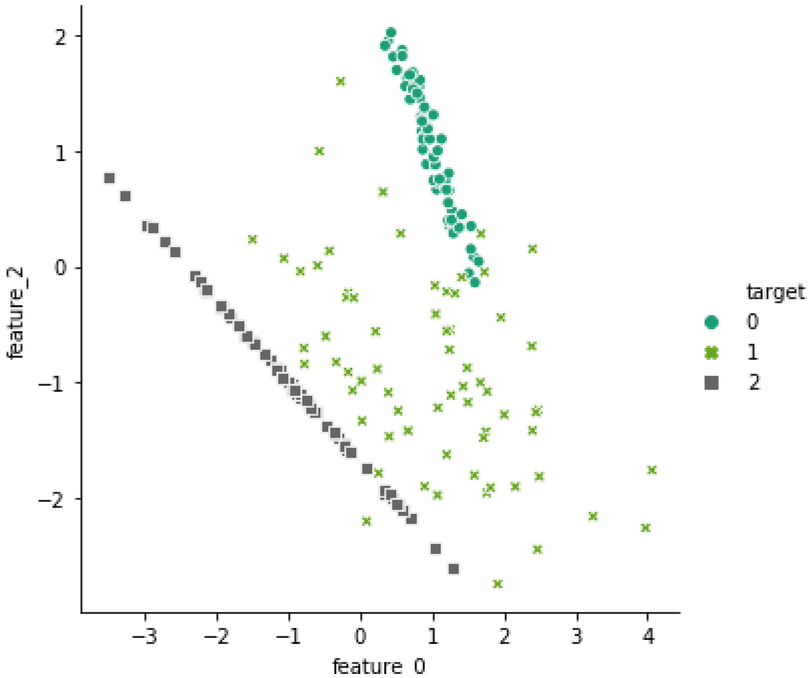
Relationship Plots

```
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.
```

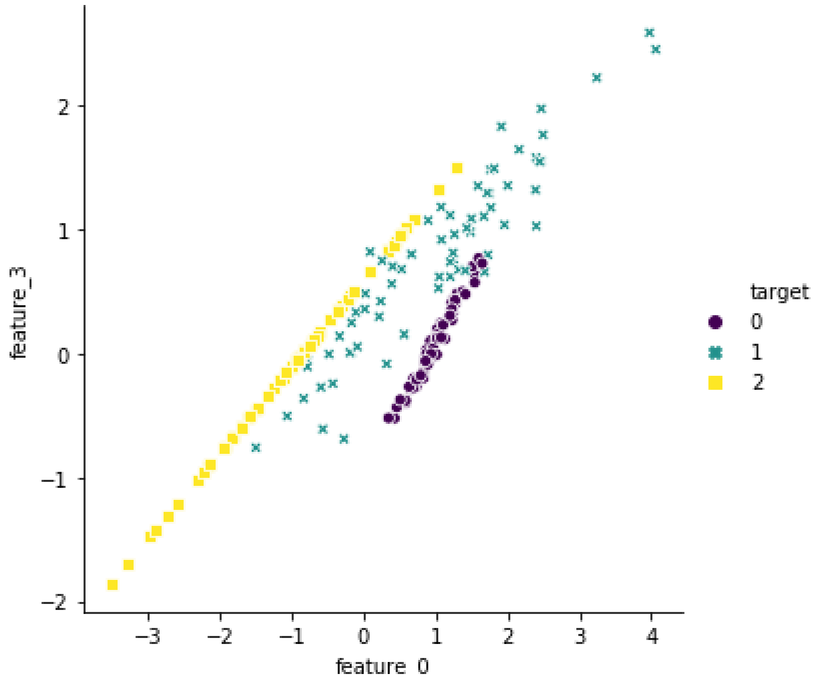
In [59]:
```
# Relationship between feature_0 and feature_1
sns.relplot(
    x='feature_0', y='feature_1', hue='target', style='target', data=dataset)
plt.show()
```



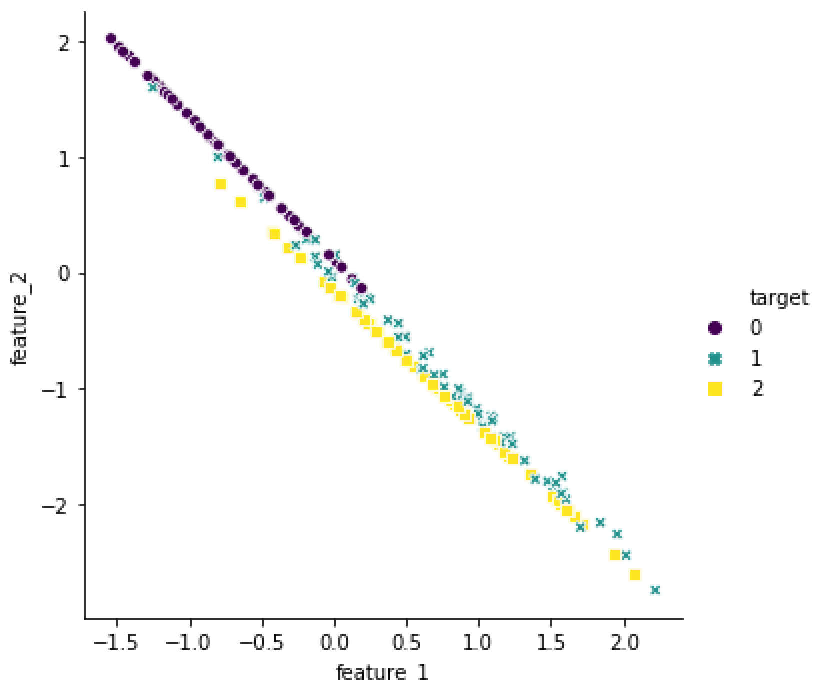
In [60]:
```
# Relationship between feature_0 and feature_2
sns.relplot(
    x='feature_0', y='feature_2', hue='target', style='target', palette='Dark2',
plt.show()
```



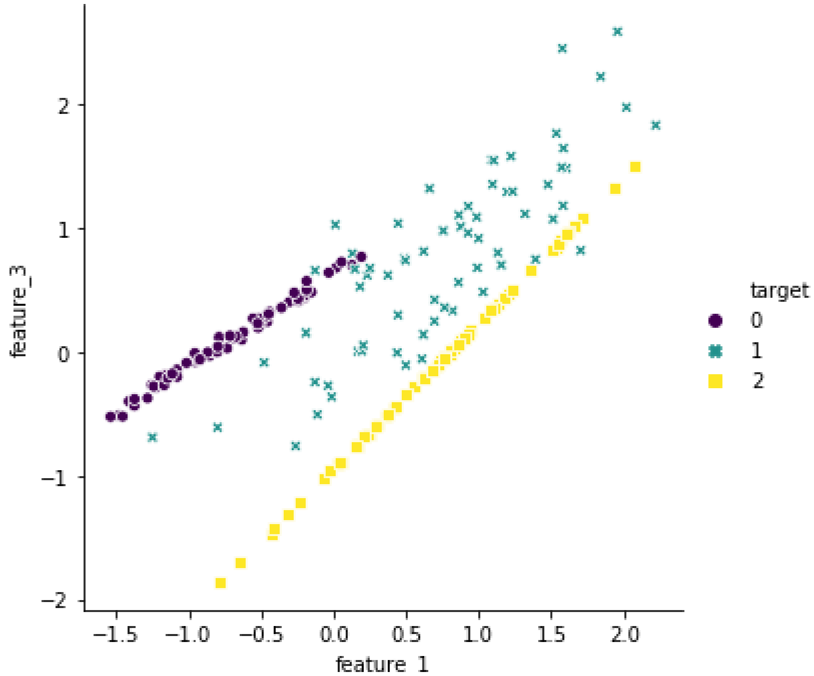
In [61]:
```
# Relationship between feature_0 and feature_3
sns.relplot(
    x='feature_0', y='feature_3', hue='target', style='target', palette='viridis
plt.show()
```



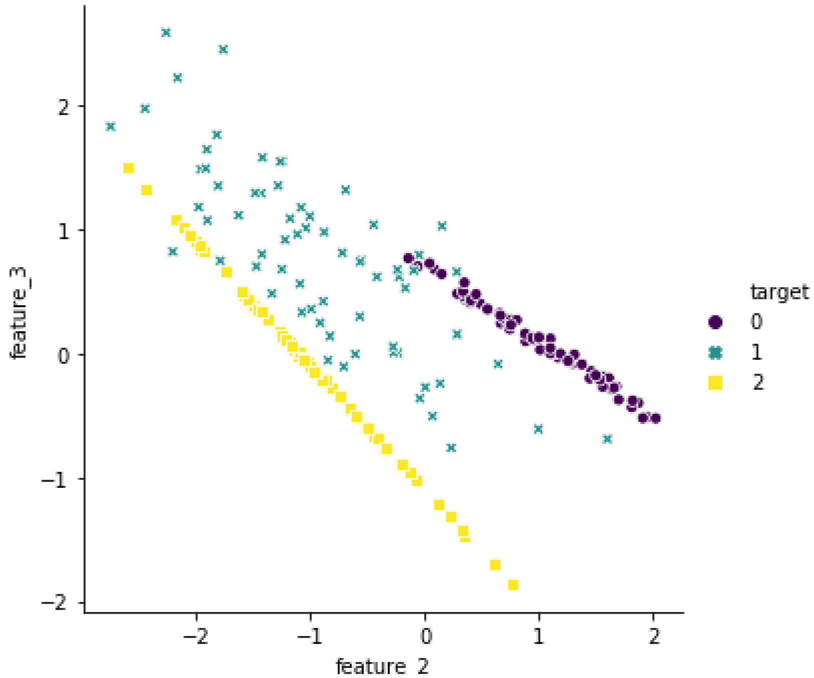
In [62]:
```
# Relationship between feature_1 and feature_2
sns.relplot(
    x='feature_1', y='feature_2', hue='target', style='target', palette='viridis
plt.show()
```



In [63]:
```
# Relationship between feature_1 and feature_3
sns.relplot(
    x='feature_1', y='feature_3', hue='target', style='target', palette='viridis
plt.show()
```



In [64]:
```
# Relationship between feature_2 and feature_3
sns.relplot(
    x='feature_2', y='feature_3', hue='target', style='target', palette='viridis
plt.show()
```



In [65]:
```
# Splitting the dataset into training and testing set
training_features, test_features, training_target, test_target = train_test_spli
    features, target, random_state=191)
```

In [66]:
```
# Showing the splition
print(training_features.shape, test_features.shape)
```

```
(150, 4) (50, 4)
```

In [67]:
```
# 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.
MLP_Classifier = MLPClassifier(hidden_layer_sizes = (300,),
                               activation='relu',
                               verbose=1, solver='adam',
                               batch_size=32,
                               learning_rate = 'constant',
                               learning_rate_init = 0.001,
                               max_iter= 1000)
```

In [68]:
```
MLP_Classifier
```

Out[68]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size=32, beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(300,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=1, warm_start=False)
```

In [69]:
```
# 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_Classifier.fit(training_features,
                           training_target)
```

```
Iteration 108, loss = 0.15867262
Iteration 109, loss = 0.15712235
Iteration 110, loss = 0.15808554
Iteration 111, loss = 0.15902723
Iteration 112, loss = 0.15734604
Iteration 113, loss = 0.15683602
Iteration 114, loss = 0.15532984
Iteration 115, loss = 0.15488387
Iteration 116, loss = 0.15430115
Iteration 117, loss = 0.15732499
Iteration 118, loss = 0.15413534
Iteration 119, loss = 0.15394742
Iteration 120, loss = 0.15216542
Iteration 121, loss = 0.15437325
Iteration 122, loss = 0.15289806
Iteration 123, loss = 0.15274950
Iteration 124, loss = 0.15085182
Iteration 125, loss = 0.15362572
Iteration 126, loss = 0.15021272
Iteration 127, loss = 0.14979109
```

In [70]:
```
# Predicting for the test features to test the performance,
# of our MLP_Classifier model.
predictions = model.predict(test_features)
```

In [71]:
```
# Creating confusion matrix from predictions
matrix = confusion_matrix(test_target, predictions)
```

Confusion matrix is the performance measurement for machine learning classification problem. Here output can be two or more than two classes. It is also used to evaluate the accuracy of a classification. In our program, it is a multiclass classification with 3 class labels.

In [72]:
```
# Displaying the confusion matrix
print(matrix)
```

```
[[22  0  0]
 [ 1 14  0]
 [ 0  0 13]]
```

In [73]:
```
# showing the classification report for the predictions
print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        22
           1       1.00      0.93      0.97        15
           2       1.00      1.00      1.00        13

   micro avg       0.98      0.98      0.98        50
   macro avg       0.99      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

Question 2:

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

```
Confusion matrix is extremely useful to measure Recall, Precision, Specificity,
Accuracy and most importantly AUC-ROC Curve. In confusion matrix, the number of
correct and incorrect predictions are summarized with count values and broken
down by each class. It gives information on not only the errors that is being
made by the classifier (decision trees in our case) but more importantly the
types of errors that are being made.

 Here The total number of samples that were tested are 50. Among them, samples
belonging to the class 0 are 22  , class 1 are 15 and class 2 are 13. By
analyzing the above confusion matrix, we can see that, for the first label
(class) i.e. 0, from total of 22 prediction, 22 were correctly classified as 0.
For the second target label, i.e. 1, from total of 15 predictions, 14 were
correctly classified and 1 were misclassified as class 1. And finally, for the
third target label, i.e. 2, from total of 17 predictions, 17 were correctly
classified as class 2.

The confusion matrix for MLP is better than that of using decision tree. It has
only misclassified one data among 50 samples. Similarly the precision, f1-
score, is also better as compared to that using decision tree. So using MPL is
more applicable then using Decision tree.
```

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 [74]:
```
MLP_Classifier = MLPClassifier(hidden_layer_sizes = (500,),
                               activation='relu',
                               verbose=1, solver='adam',
                               batch_size='auto',
                               learning_rate='constant',
                               learning_rate_init=0.001,
                               max_iter= 1000)
```

In [75]:
```
model = MLP_Classifier.fit(training_features,
                           training_target)
```

```
Iteration 23, loss = 0.47052403
Iteration 24, loss = 0.46110288
Iteration 25, loss = 0.45236445
Iteration 26, loss = 0.44424021
Iteration 27, loss = 0.43666690
Iteration 28, loss = 0.42960447
Iteration 29, loss = 0.42299022
Iteration 30, loss = 0.41678087
Iteration 31, loss = 0.41093645
Iteration 32, loss = 0.40540104
Iteration 33, loss = 0.40015861
Iteration 34, loss = 0.39517354
Iteration 35, loss = 0.39042509
Iteration 36, loss = 0.38588162
Iteration 37, loss = 0.38152232
Iteration 38, loss = 0.37733791
Iteration 39, loss = 0.37332179
Iteration 40, loss = 0.36946704
Iteration 41, loss = 0.36575836
```

In [76]:
```
predictions = model.predict(test_features)
```

In [77]:
```
matrix = confusion_matrix(test_target, predictions)
```

In [78]:
```
print(matrix)
```

```
[[22  0  0]
 [ 0 15  0]
 [ 0  0 13]]
```

In [79]:
```
print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00        13

   micro avg       1.00      1.00      1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

```
When training the model by increasing the layers from 300 to 500, it will allow
to decrease the training error but it also reduces the amount of
generalization. When we add layers we increase the dimensional complexity of
the data . Every time we add a layer, it change the shape of the discriminator.
Using adam optimiser helps to update network weights iterative based in
training data. Adam also makes use of the average of the second moments of the
gradients (the uncentered variance).So the model classify accurately without
any error.
```

Experiment 2:

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

In [80]:
```
MLP_Classifier = MLPClassifier(hidden_layer_sizes = (350,),
                               activation='relu',
                               verbose=1, solver='sgd',
                               batch_size='auto',
                               learning_rate='adaptive',
                               max_iter= 1000)
```

In [81]:
```
model = MLP_Classifier.fit(training_features,
                           training_target)
```

```
Iteration 256, loss = 0.44181900
Iteration 257, loss = 0.44128499
Iteration 258, loss = 0.44075464
Iteration 259, loss = 0.44022790
Iteration 260, loss = 0.43970473
Iteration 261, loss = 0.43918507
Iteration 262, loss = 0.43866887
Iteration 263, loss = 0.43815640
Iteration 264, loss = 0.43764740
Iteration 265, loss = 0.43714179
Iteration 266, loss = 0.43663954
```

In [82]:
```
predictions = model.predict(test_features)
matrix = confusion_matrix(test_target, predictions)
print(matrix)
```

```
[[22  0  0]
 [ 0 12  3]
 [ 0  0 13]]
```

In [83]:
```
print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       1.00      0.80      0.89        15
           2       0.81      1.00      0.90        13

   micro avg       0.94      0.94      0.94        50
   macro avg       0.94      0.93      0.93        50
weighted avg       0.95      0.94      0.94        50
```

Here, after changing the layers to 350, the hidden layer is decrease, training error is more and less dimensional complexity of the data. The solver used is sgd where a few samples are selected randomly instead of the whole data set for each iteration. so this model misclassified 3 data among 50 data of the sample.

Experiment 3:

Changing the following paramaters:
Hidden Layer: 400
batch_size: auto

```
learning rate: invscaling
activation function: relu
loss function:
max_iter: 500
```

In [84]:
```python
MLP_Classifier = MLPClassifier(hidden_layer_sizes = (400,),
                               activation='relu',
                               verbose=1, solver='adam',
                               batch_size='auto',
                               learning_rate = 'invscaling',
                               max_iter= 500)
```

In [85]:
```python
model = MLP_Classifier.fit(training_features,
                           training_target)
```

```
Iteration 176, loss = 0.19100822
Iteration 177, loss = 0.19040963
Iteration 178, loss = 0.18981455
Iteration 179, loss = 0.18923094
Iteration 180, loss = 0.18865317
Iteration 181, loss = 0.18808063
Iteration 182, loss = 0.18751711
Iteration 183, loss = 0.18695996
Iteration 184, loss = 0.18640554
Iteration 185, loss = 0.18585504
Iteration 186, loss = 0.18532194
Iteration 187, loss = 0.18479291
Iteration 188, loss = 0.18426563
Iteration 189, loss = 0.18374763
Iteration 190, loss = 0.18324318
Iteration 191, loss = 0.18274575
Iteration 192, loss = 0.18225339
Iteration 193, loss = 0.18176772
Iteration 194, loss = 0.18128679
```

In [86]:
```python
predictions = model.predict(test_features)
matrix = confusion_matrix(test_target, predictions)
print(matrix)
```

```
[[22  0  0]
 [ 0 15  0]
 [ 0  0 13]]
```

In [87]:
```python
print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        22
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00        13

   micro avg       1.00      1.00      1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

When increasing the layers to 400. The model classified correctly. The model uses more hidden layer which are sufficient for training the data. Using adam optimiser helps to update network weights iterative based in training data. Adam also makes use of the average of the second moments of the gradients (the uncentered variance). so the model can classify properly with out any misclassification.

Finally, from the above 3 experiment it is found that the model can classify properly if the model contain more number of hidden layer so that the data can be trained properly. The best solver for the neural network design is adam. Relu can be used as actiivation function for better result.

End Of Assignment!!