Reid Glaze
CSCI 5922

Lab Assignment 1: Feedforward Neural Networks

1b)

For this Lab Assignment, I used the Iris dataset. This is a popular dataset that can be loaded from sklearn. It includes a total of 150 samples of iris. There are 3 species, and the dataset contains 50 samples from each species. There are 4 features in the dataset.

The hyperparameters were kept constant when training this Neural Network. The number of neurons per a layer was kept at 10. A maximum of 200 iterations was used. The batch size was kept at default, which is 200. The optimization approach was kept at default, which is "adam", referring to a stochastic gradient-based optimizer.

To train the 9 neural network models, 3 different numbers of hidden layers were used, and 3 different types of activation functions were used. I used 2, 3, or 4 hidden layers, all with 10 neurons per a layer. The activation functions used were "tanh", "relu", or "logistic". Each of the 9 unique models produced different accuracy scores and a unique confusion matrix.

The results from this problem have been included along with my code. It is important to note that each time the program is run, the results are different. This is because the test train split function is done with randomization. However, generalizations can be made from my results, especially when the difference between different models is significant.

Most importantly, it appears that the "tanh" and "relu" functions have proven to be far more accurate than the "logistic" function. The "logistic" function is a type of sigmoid function. One problem with a sigmoid function is that it causes the gradient to vanish if it is very small. Another problem is that the output is not zero centered. Since the output is always positive, going from 0 to 1, the gradients of the weights are either all positive or all negative. This causes gradient updates to go too far in either direction, making optimization difficult. This is shown in the results, as all 3 of the accuracy scores using this function are very low.

When comparing the "tanh" and "relu" functions, it appears that "tanh" yields a slightly higher accuracy score. However, it is hardly enough to make a determinization and when the program is re-run "relu" often yields better accuracy scores. Online sources say that "relu" is often a better option than "tanh". Although they are both zero centered, "relu" does not have the problem where the gradient vanishes if it is very small. However, this cannot be determined from these results because the differences in accuracies are not significant. Furthermore, they change between trials.

When observing the number of hidden layers in the results, it appears that there are mixed results. When looking at the "tanh" and "relu" functions, it appears that 3 or 4 layers will often be more accurate than just 2 layers. In the "relu" function 3 layers produces the highest accuracy score. In "tanh" they produce the same score. This fluctuates as the program is re-run. Complex datasets require more hidden layers. Since this dataset is not extremely complex, it may not be necessary to use more than 3.

2b)

Like part 1, I used the Iris dataset taken from sklearn. It includes 150 samples, 4 features, and 3 outputs. Similarly, the hyperparameters were kept the same, using 10 neurons per layer, 200 iterations, a batch size of 200, and "adam" as the optimization approach.

The goal of this problem was to determine how the amount of training data used and the length of the training duration played a role in determining. After using test train split to separate the training data from the test data, I used the function once again to minimize the training dataset even further. I did this to test the effectiveness of 20% , 40%, 60%, 80%, and 100% of the training data. For each of these, I created a graph to show how the accuracy changed over the number of epochs, starting at 10 and ending at 100. The results of these 5 tests are shown in a single plot.

The general trend is that a higher percentage of training data used results in a higher accuracy. The biggest difference in accuracy appears to be in between 40 and 60 percent. Between 60 and 100 percent, the differences are not as noticeable. Oddly, 60 percent appears to yield a slightly higher accuracy than 80 percent. My assumption is that this is simply due to randomization that occurs from test train split. It makes logical sense that more training data would result in a more accurate model. Exceptions may be due to outliers being excluded randomly.

It is also evident that a higher number of epochs results in a higher accuracy score. Although some of the lines go down at points, the general upward trend is prevalent. Additionally, it appears that the lines separate and then come back together. Although none of the lines flatten out at 100 epochs, it seems that the lines with more training data will reach this point sooner. When less training data is used, more epochs may be necessary to achieve a reliable accuracy.

Lab_Assignment_1

September 11, 2022

```
[1]: #Reid Glaze
     #CSCI 5922: Neural Nets and Deep Learning
     #Sept 11, 2022
     #----------
     from sklearn.datasets import load_iris
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.neural_network import MLPClassifier
     from sklearn import metrics
     import seaborn as sns
     import matplotlib.pyplot as plt
     #  Load dataset
     iris = load_iris()
     X = iris.data
     y = iris.target
     #-----------------------
     # Create the train/test split and preprocess
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
      ↪random_state = 1)
     # Standardize data
     stdsc = StandardScaler()
     stdsc.fit(X_train)
     X_train_std = stdsc.transform(X_train)
     X_test_std = stdsc.transform(X_test)
     #-----------------------

     # tanh with 2 hidden layers
     mlp1 = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10])
     mlp1.fit(X_train_std, y_train)
     y_predicted = mlp1.predict(X_test_std)
     print("Activaton Function: {}".format(mlp1.activation))
     print("Number of  Hidden Layers: 2")
     print("Accuracy score on the test set: {}".format(mlp1.score(X_test_std,␣
      ↪y_test)))
     mat = metrics.confusion_matrix(y_test, y_predicted)
     sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
     plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")

# tanh with 3 hidden layers
mlp2 = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp2.fit(X_train_std, y_train)
y_predicted = mlp2.predict(X_test_std)
print("Activaton Function: {}".format(mlp2.activation))
print("Number of  Hidden Layers: 3")
print("Accuracy score on the test set: {}".format(mlp2.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# tanh with 4 hidden layers
mlp3 = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10, 10])
mlp3.fit(X_train_std, y_train)
y_predicted = mlp3.predict(X_test_std)
print("Activaton Function: {}".format(mlp3.activation))
print("Number of  Hidden Layers: 4")
print("Accuracy score on the test set: {}".format(mlp3.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# relu with 2 hidden layers
mlp4 = MLPClassifier(activation = 'relu', hidden_layer_sizes=[10, 10])
mlp4.fit(X_train_std, y_train)
y_predicted = mlp4.predict(X_test_std)
print("Activaton Function: {}".format(mlp4.activation))
print("Number of  Hidden Layers: 2")
print("Accuracy score on the test set: {}".format(mlp4.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# relu with 3 hidden layers
mlp5 = MLPClassifier(activation = 'relu', hidden_layer_sizes=[10, 10, 10])
mlp5.fit(X_train_std, y_train)
y_predicted = mlp5.predict(X_test_std)
print("Activaton Function: {}".format(mlp5.activation))
print("Number of  Hidden Layers: 3")
print("Accuracy score on the test set: {}".format(mlp5.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# relu with 4 hidden layers
mlp6 = MLPClassifier(activation = 'relu', hidden_layer_sizes=[10, 10, 10, 10])
mlp6.fit(X_train_std, y_train)
y_predicted = mlp6.predict(X_test_std)
print("Activaton Function: {}".format(mlp6.activation))
print("Number of  Hidden Layers: 4")
print("Accuracy score on the test set: {}".format(mlp6.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# logistic with 2 hidden layers
mlp7 = MLPClassifier(activation = 'logistic', hidden_layer_sizes=[10, 10])
mlp7.fit(X_train_std, y_train)
y_predicted = mlp7.predict(X_test_std)
print("Activaton Function: {}".format(mlp7.activation))
print("Number of  Hidden Layers: 2")
print("Accuracy score on the test set: {}".format(mlp7.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
```

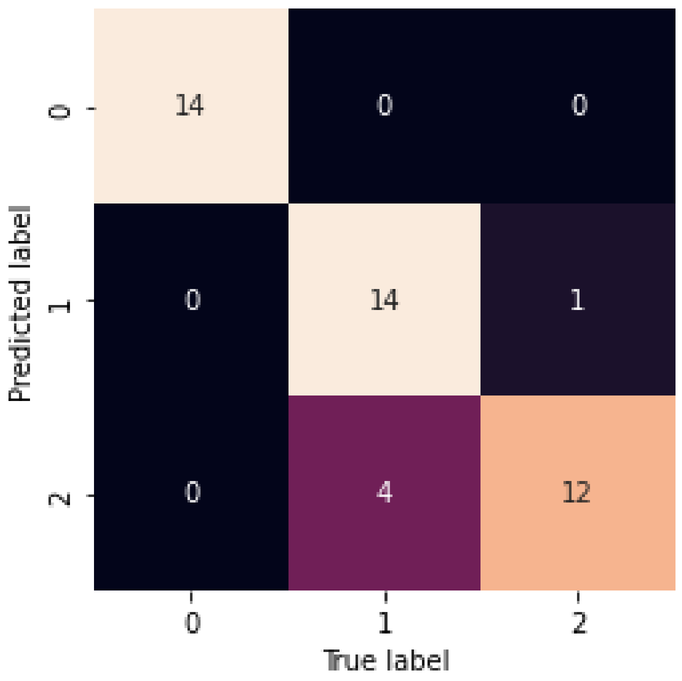
```
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# logistic with 3 hidden layers
mlp8 = MLPClassifier(activation = 'logistic', hidden_layer_sizes=[10, 10, 10])
mlp8.fit(X_train_std, y_train)
y_predicted = mlp8.predict(X_test_std)
print("Activaton Function: {}".format(mlp8.activation))
print("Number of  Hidden Layers: 3")
print("Accuracy score on the test set: {}".format(mlp8.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
print("--------------------------------------------")


# logistic with 4 hidden layers
mlp9 = MLPClassifier(activation = 'logistic', hidden_layer_sizes=[10, 10, 10,␣
 ↪10])
mlp9.fit(X_train_std, y_train)
y_predicted = mlp9.predict(X_test_std)
print("Activaton Function: {}".format(mlp9.activation))
print("Number of  Hidden Layers: 4")
print("Accuracy score on the test set: {}".format(mlp9.score(X_test_std,␣
 ↪y_test)))
mat = metrics.confusion_matrix(y_test, y_predicted)
sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar = False)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
```

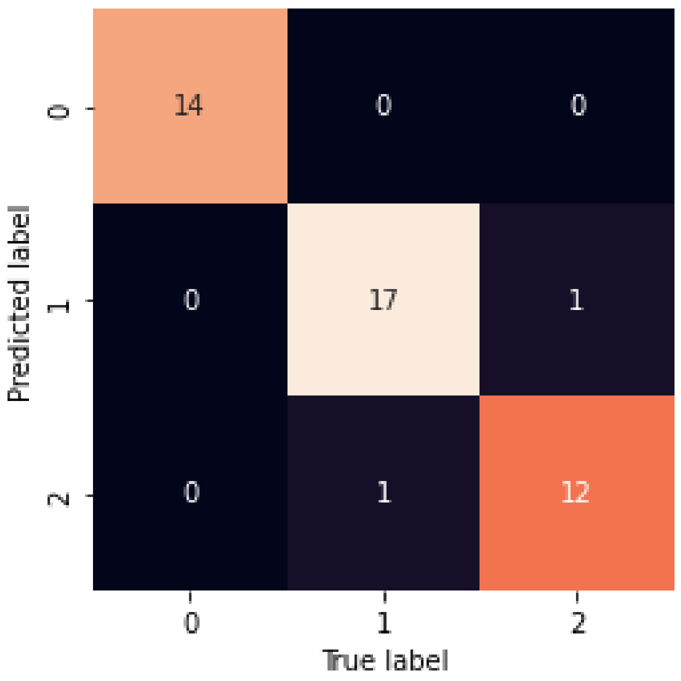
```
Activaton Function: tanh
Number of  Hidden Layers: 2
Accuracy score on the test set: 0.8888888888888888
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



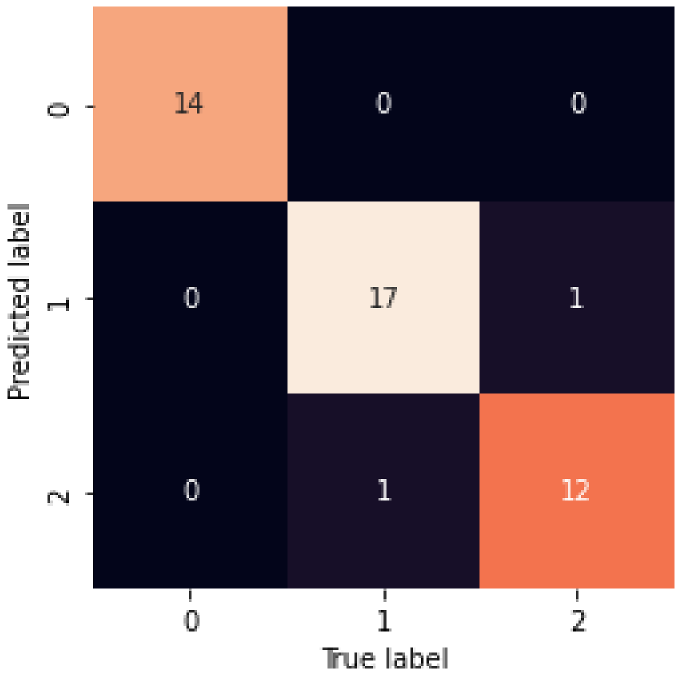
```
--------------------------------------------
Activaton Function: tanh
Number of  Hidden Layers: 3
Accuracy score on the test set: 0.9555555555555556
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



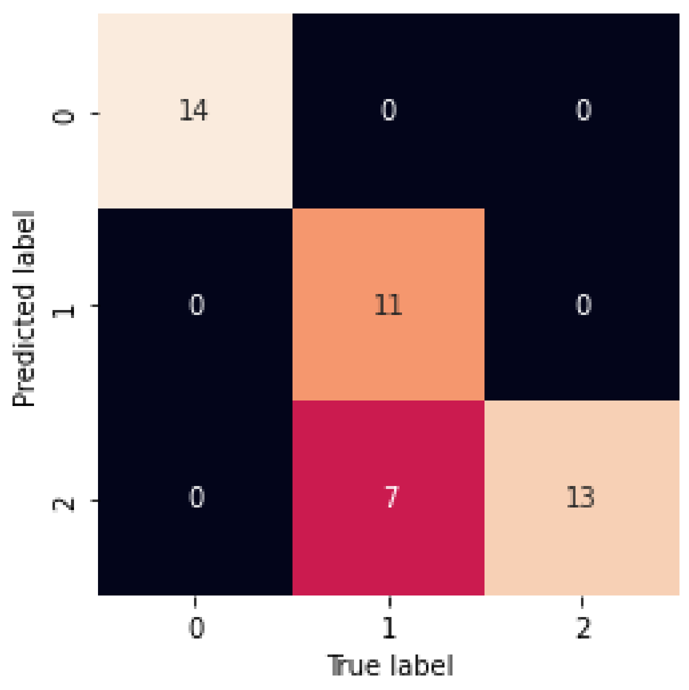
```
--------------------------------------------
Activaton Function: tanh
Number of  Hidden Layers: 4
Accuracy score on the test set: 0.9555555555555556
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



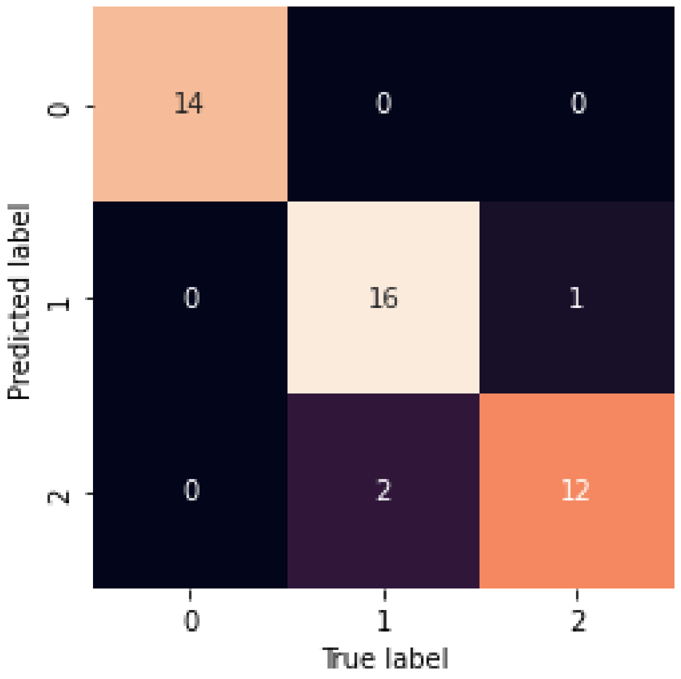
```
---------------------------------------------
Activaton Function: relu
Number of  Hidden Layers: 2
Accuracy score on the test set: 0.8444444444444444
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



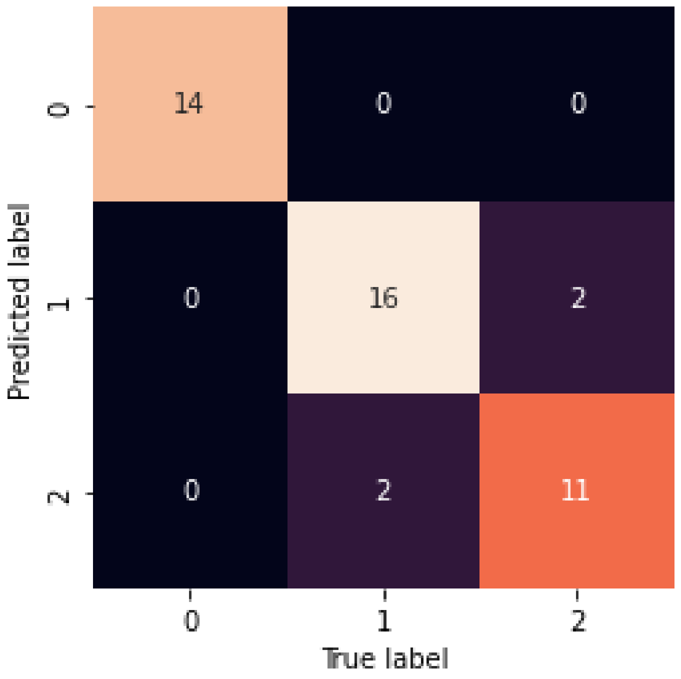
```
---------------------------------------------
Activaton Function: relu
Number of  Hidden Layers: 3
Accuracy score on the test set: 0.9333333333333333
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



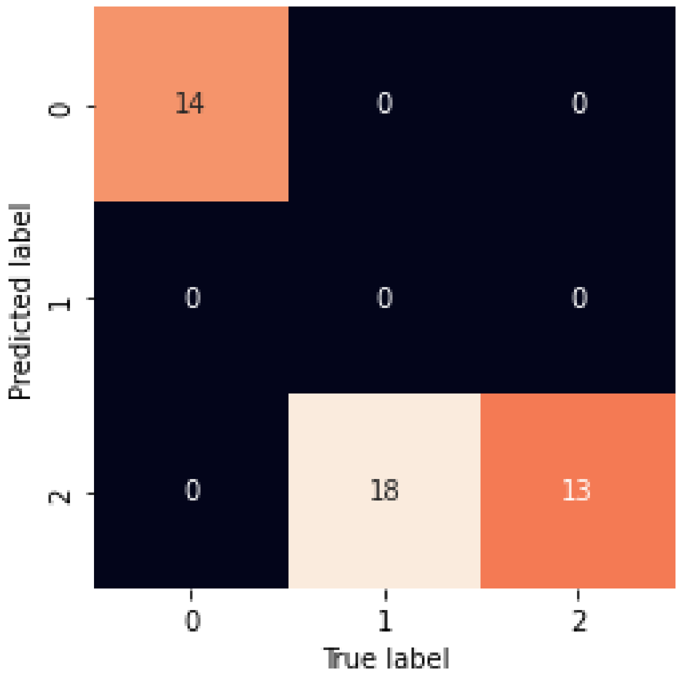
```
---------------------------------------------
Activaton Function: relu
Number of  Hidden Layers: 4
Accuracy score on the test set: 0.9111111111111111
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



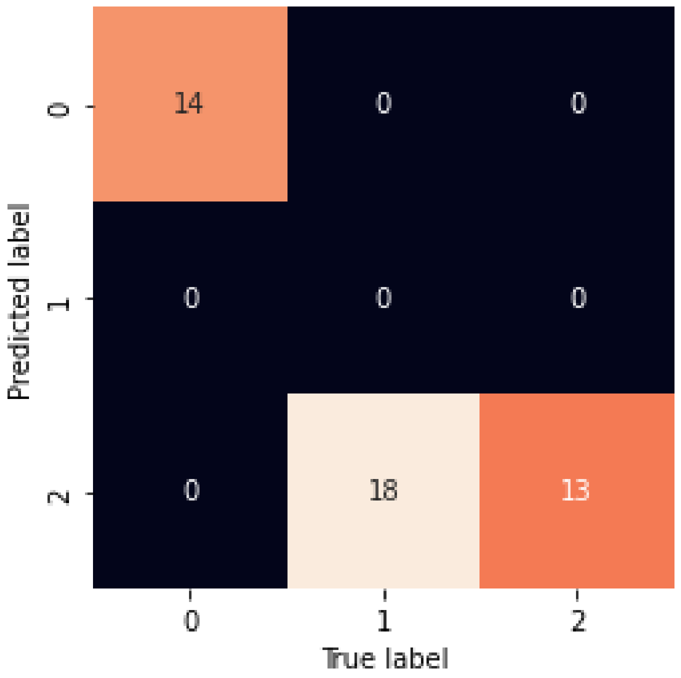
```
---------------------------------------------
Activaton Function: logistic
Number of  Hidden Layers: 2
Accuracy score on the test set: 0.6
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

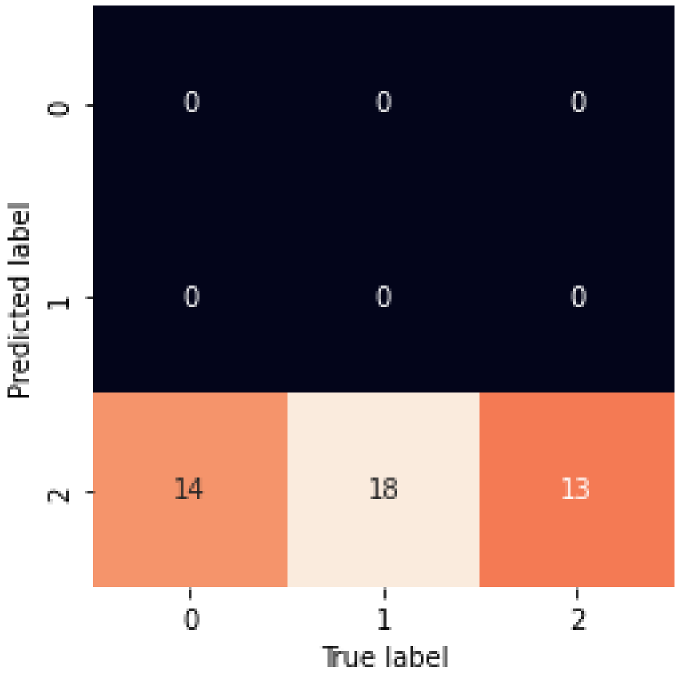


```
---------------------------------------------
Activaton Function: logistic
Number of  Hidden Layers: 3
Accuracy score on the test set: 0.6

/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



```
--------------------------------------------
Activaton Function: logistic
Number of  Hidden Layers: 4
Accuracy score on the test set: 0.28888888888888886
```



[2]:
```
#Create the train/test split and preprocess (use 20 percent of train data after␣
↪70/30 split)
X_train_1, X_test, y_train_1, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state = 1)
X_train, X_test_1, y_train, y_test_1 = train_test_split(X_train_1, y_train_1,␣
↪test_size=0.8, random_state = 1)
# Standardize data
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_std = stdsc.transform(X_train)
X_test_std = stdsc.transform(X_test)

#-------------------------
# tanh with 3 hidden layers, each a size of 10 (this will be kept constant)
#------------------------
#classifier
mlp = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp.fit(X_train_std, y_train)
y_predicted = mlp.predict(X_test_std)
mat = metrics.confusion_matrix(y_test, y_predicted)
%matplotlib inline
start_num_epochs = 10
finish_num_epochs = 100
inc_amt = 10
pred_scores = []
num_epochs = []
for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
  my_classifier = MLPClassifier(random_state = 20, max_iter = epoch_count)
  my_classifier.fit(X_train_std, y_train)
  score = my_classifier.score(X_test_std, y_test)
  pred_scores.append(score)
  num_epochs.append(epoch_count)
num_epochs_20 = num_epochs
pred_scores_20 = pred_scores
accuracy_20 = format(mlp.score(X_test_std, y_test))

#--------------------------------

#Create the train/test split and preprocess (use 40 percent of train data)
X_train_1, X_test, y_train_1, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state = 1)
X_train, X_test_1, y_train, y_test_1 = train_test_split(X_train_1, y_train_1,␣
↪test_size=0.6, random_state = 1)
# Standardize data
```

```
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_std = stdsc.transform(X_train)
X_test_std = stdsc.transform(X_test)

#classifier
mlp = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp.fit(X_train_std, y_train)
y_predicted = mlp.predict(X_test_std)
mat = metrics.confusion_matrix(y_test, y_predicted)
%matplotlib inline
start_num_epochs = 10
finish_num_epochs = 100
inc_amt = 10
pred_scores = []
num_epochs = []
for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
  my_classifier = MLPClassifier(random_state = 20, max_iter = epoch_count)
  my_classifier.fit(X_train_std, y_train)
  score = my_classifier.score(X_test_std, y_test)
  pred_scores.append(score)
  num_epochs.append(epoch_count)

num_epochs_40 = num_epochs
pred_scores_40 = pred_scores
accuracy_40 = format(mlp.score(X_test_std, y_test))

#--------------------

#Create the train/test split and preprocess (use 60 percent of train data)
X_train_1, X_test, y_train_1, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 1)
X_train, X_test_1, y_train, y_test_1 = train_test_split(X_train_1, y_train_1,␣
 ↪test_size=0.4, random_state = 1)
# Standardize data
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_std = stdsc.transform(X_train)
X_test_std = stdsc.transform(X_test)

#classifier
mlp = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp.fit(X_train_std, y_train)
y_predicted = mlp.predict(X_test_std)
mat = metrics.confusion_matrix(y_test, y_predicted)
%matplotlib inline
start_num_epochs = 10
```

```
finish_num_epochs = 100
inc_amt = 10
pred_scores = []
num_epochs = []
for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
  my_classifier = MLPClassifier(random_state = 20, max_iter = epoch_count)
  my_classifier.fit(X_train_std, y_train)
  score = my_classifier.score(X_test_std, y_test)
  pred_scores.append(score)
  num_epochs.append(epoch_count)

num_epochs_60 = num_epochs
pred_scores_60 = pred_scores
accuracy_60 = format(mlp.score(X_test_std, y_test))

#------------------------

#Create the train/test split and preprocess (use 80 percent of train data)
X_train_1, X_test, y_train_1, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 1)
X_train, X_test_1, y_train, y_test_1 = train_test_split(X_train_1, y_train_1,␣
 ↪test_size=0.2, random_state = 1)
# Standardize data
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_std = stdsc.transform(X_train)
X_test_std = stdsc.transform(X_test)

#classifier
mlp = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp.fit(X_train_std, y_train)
y_predicted = mlp.predict(X_test_std)
mat = metrics.confusion_matrix(y_test, y_predicted)
%matplotlib inline
start_num_epochs = 10
finish_num_epochs = 100
inc_amt = 10
pred_scores = []
num_epochs = []
for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
  my_classifier = MLPClassifier(random_state = 20, max_iter = epoch_count)
  my_classifier.fit(X_train_std, y_train)
  score = my_classifier.score(X_test_std, y_test)
  pred_scores.append(score)
  num_epochs.append(epoch_count)

num_epochs_80 = num_epochs
```

```
pred_scores_80 = pred_scores
accuracy_80 = format(mlp.score(X_test_std, y_test))
#------------------------------
#Create the train/test split and preprocess (use 100 percent of train)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state = 1)
# Standardize data
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_std = stdsc.transform(X_train)
X_test_std = stdsc.transform(X_test)

#classifier
mlp = MLPClassifier(activation = 'tanh', hidden_layer_sizes=[10, 10, 10])
mlp.fit(X_train_std, y_train)
y_predicted = mlp.predict(X_test_std)
mat = metrics.confusion_matrix(y_test, y_predicted)
%matplotlib inline
start_num_epochs = 10
finish_num_epochs = 100
inc_amt = 10
pred_scores = []
num_epochs = []
for epoch_count in range(start_num_epochs, finish_num_epochs, inc_amt):
  my_classifier = MLPClassifier(random_state = 20, max_iter = epoch_count)
  my_classifier.fit(X_train_std, y_train)
  score = my_classifier.score(X_test_std, y_test)
  pred_scores.append(score)
  num_epochs.append(epoch_count)

num_epochs_100 = num_epochs
pred_scores_100 = pred_scores
accuracy_100 = format(mlp.score(X_test_std, y_test))
```

```
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and
the optimization hasn't converged yet.
```

```
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (30) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (60) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (70) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (80) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (90) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
```

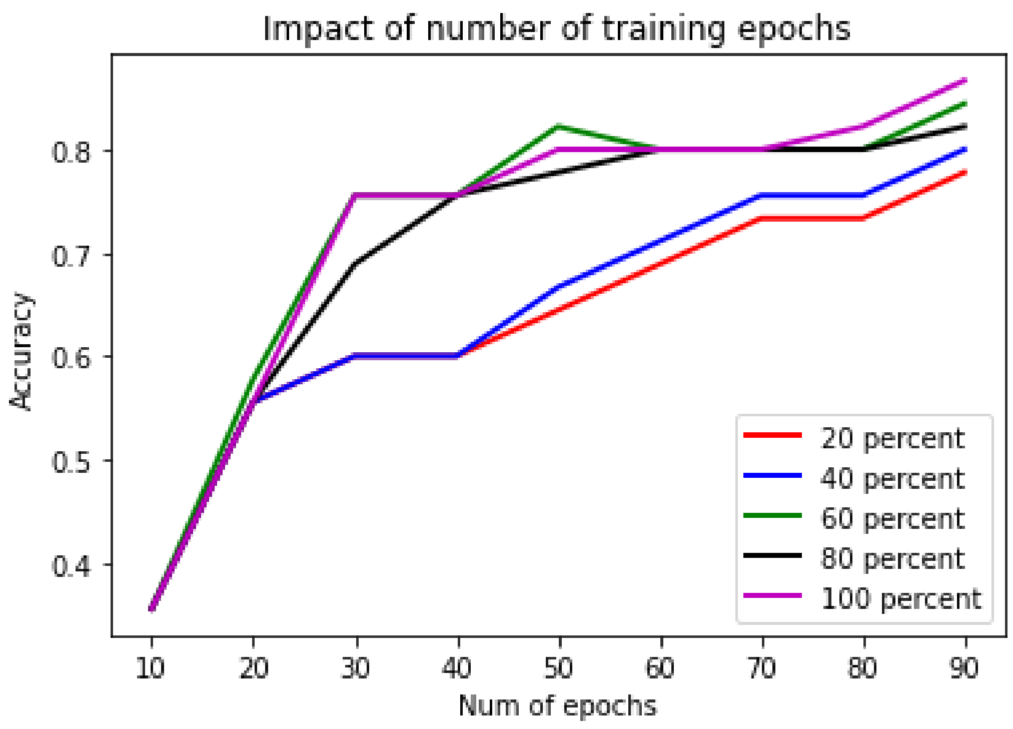
```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (30) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (60) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (70) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (80) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (90) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (30) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (60) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (70) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (80) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (90) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
```

```
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (30) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (60) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (70) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (80) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (90) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
```

```
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (30) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (40) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (60) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (70) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (80) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/anaconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (90) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[3]:
```
print("Accuracy when using 20 percent of train:", accuracy_20)
print("Accuracy when using 40 percent of train:", accuracy_40)
print("Accuracy when using 60 percent of train:", accuracy_60)
print("Accuracy when using 80 percent of train:", accuracy_80)
print("Accuracy when using 100 percent of train:", accuracy_100)
line1 = plt.plot(num_epochs_20, pred_scores_20, 'r', linewidth = 2, label="20␣
 ↪percent")
line2 = plt.plot(num_epochs_40, pred_scores_40, 'b', linewidth = 2, label="40␣
 ↪percent")
line3 = plt.plot(num_epochs_60, pred_scores_60, 'g', linewidth = 2, label="60␣
 ↪percent")
line4 = plt.plot(num_epochs_80, pred_scores_80, 'k', linewidth = 2, label="80␣
 ↪percent")
line5 = plt.plot(num_epochs_100, pred_scores_100, 'm', linewidth = 2,␣
 ↪label="100 percent")
leg = plt.legend(loc='lower right')
plt.xlabel("Num of epochs")
plt.ylabel("Accuracy")
plt.title("Impact of number of training epochs")
plt.show()
```

```
Accuracy when using 20 percent of train: 0.8
Accuracy when using 40 percent of train: 0.9111111111111111
Accuracy when using 60 percent of train: 0.8222222222222222
Accuracy when using 80 percent of train: 0.8666666666666667
Accuracy when using 100 percent of train: 0.8222222222222222
```



```
[ ]:
```