# ELG 5255[EG] Applied Machine Learning Summer

## Assignment -Four (Multi-Layer perceptron)

(Group 34)

Ahmed Hallaba  Omar Sorour

Kamel El-Sehly

## Questions 1

Apply Multi-Layer Perceptron (MLP) on the provided dataset by using activation functions listed below. Use the given parameters. Run MLP 10 times for each case, plot runtime vs. accuracy with average line as shown in below.

(Number of hidden layer: 1, Learning rate: 0.1, maximum iteration:1000 )

1. Relu
2. Sigmoid
3. Tanh

(Note: for simplicity we set the number of nodes per each hidden layer to be 4 so if the hidden layers input is a tuple like this (4,4,4) this means that it has three hidden layers with 10 nodes for each layer)

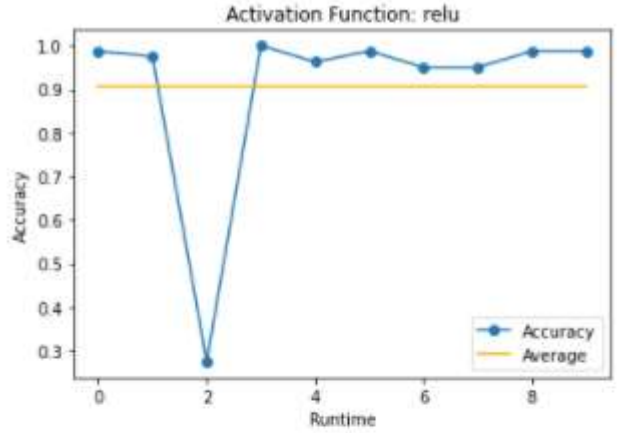
This has been implemented in the following lines of code:

**Training with different activation function**

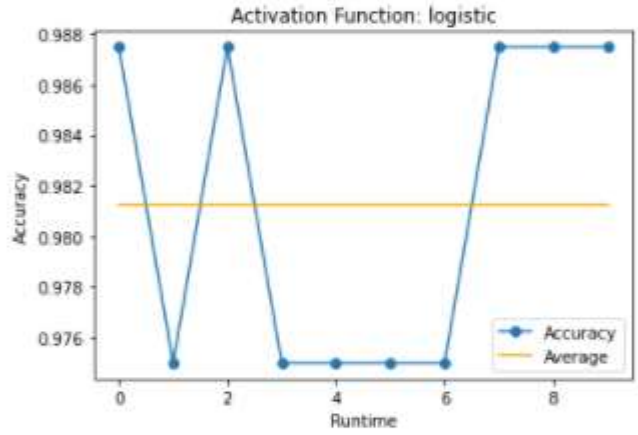
```
lr=0.1

base=0
for act in ['relu','logistic','tanh']:
    scores=[]
    for i in range(10):
        clf = MLPClassifier(random_state=i,hidden_layer_sizes=(10), max_iter=1000,
                            learning_rate_init=lr,activation=act).fit(TrX, Try)
        clf.predict_proba(TeX)
        clf.predict(TeX)
        sc=clf.score(TeX, Tey)
        scores.append(sc)
    smean=st.mean(scores)
    base=max(base,smean)
    print(smean)
    print(clf.n_layers_)
    plt.plot(scores,marker='o',label='Accuracy')
    x = []
    y=[]

    for j in range(10):
        x.append(j)
        y.append(smean)
    plt.plot(x,y,color='orange',label='Average')

    plt.xlabel('Runtime')
    plt.ylabel('Accuracy')
    plt.title('Activation Function: '+act)
    plt.legend()
    plt.show()
```

The best accuracy was for the sigmoid activation function, It achieved average accuracy of 98.125% as we can see in the following figures:

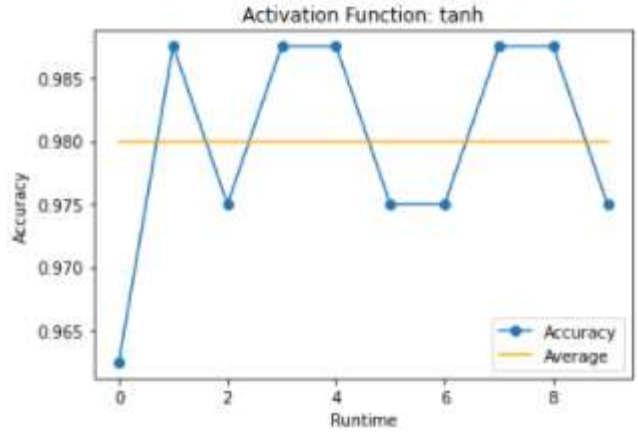
```
Mean accuracy is : 0.90625
```




```
Mean accuracy is : 0.98125
```




```
Mean accuracy is : 0.98
```



## Question 2

Choose the activation function that provides highest average accuracy value in Q1. Try different number of hidden layers as given below. Run MLP 10 times for each case, plot runtime vs. accuracy with average line. Plot the best average accuracy value from Q1 as baseline performance.

Number of hidden layers: 1, 2, 3 and 4

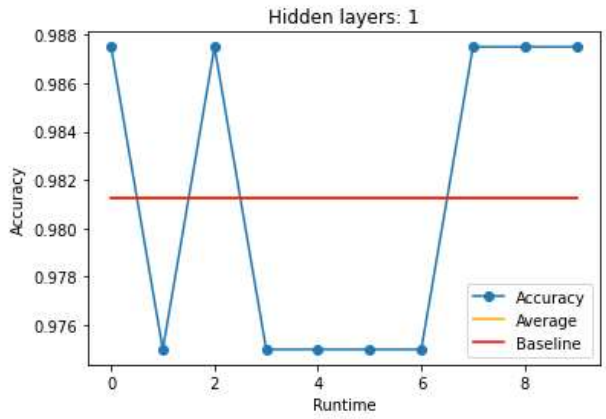
We further tried to tune the hidden layers of the best of the models trained in the first question which was the model with sigmoid activation function. In the following lines of code, we trained the model 10 times with different number of the hidden layers and plotted the average accuracy versus the Runtime. We ran a for loop over a list of tuples each tuple represents number of hidden layers and number of nodes within each layer.

**Traning the best model with different number of hidden layers**

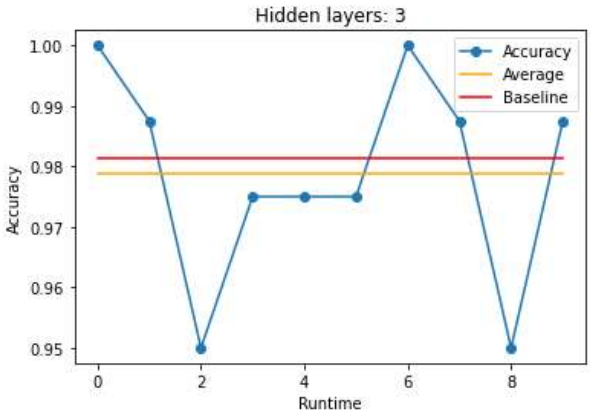
```
act='logistic'
lr=0.1
hl = [(4),(4,4),(4,4,4),(4,4,4,4)]
for hli in hl:
    scores=[]
    for i in range(10):
        clf = MLPClassifier(random_state=i,hidden_layer_sizes=hli, max_iter=1000,
                            learning_rate_init=lr,activation=act).fit(TrX, Try)
        clf.predict_proba(TeX)
        clf.predict(TeX)
        sc=clf.score(TeX, Tey)
        scores.append(sc)
    smean=st.mean(scores)
    print("Mean accuracy is:",smean)
#     print(clf.n_layers_)
    plt.plot(scores,marker='o',label='Accuracy')
    x=[]
    y=[]
    z=[]

    for j in range(10):
        x.append(j)
        y.append(smean)
        z.append(base)

    plt.plot(x,y,color='orange',label='Average')
    plt.plot(x,z,color='red',label='Baseline')


    plt.xlabel('Runtime')
    plt.ylabel('Accuracy')
    plt.title('Hidden layers: '+str(clf.n_layers_ -2))
    plt.legend()
    plt.show()
```

The best accuracy was for the sigmoid activation function when trained with 2 hidden layers, It achieved average accuracy of 98.75% as we can see in the following figures:

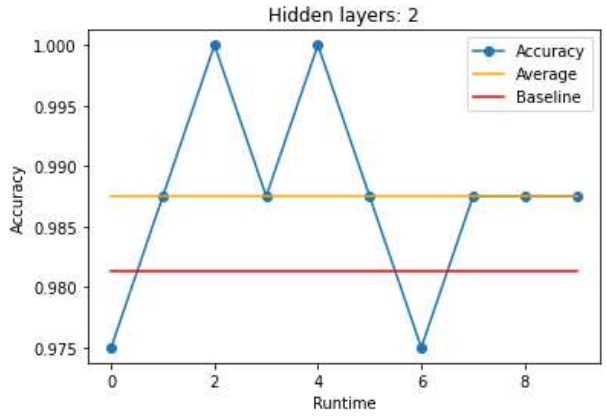
```
Mean accuracy is: 0.98125
```



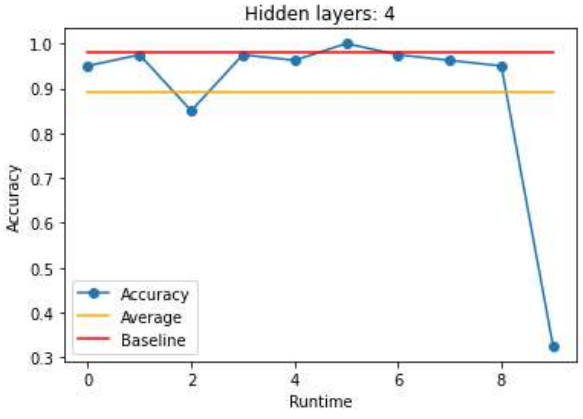

```
Mean accuracy is: 0.97875
```




```
Mean accuracy is: 0.9875
```




```
Mean accuracy is: 0.8925
```



## Question 3

Use the number of hidden layers that achieves highest average accuracy in Q2. In this step, try 3 different learning rate values from the given interval. Run MLP 10 times for each case, plot runtime vs. accuracy with average line and updated baseline performance.

1. Learning rate: 0.08 - 0.3

2. Train your model with final parameters. Display the training curve and confusion matrix with accuracy value.

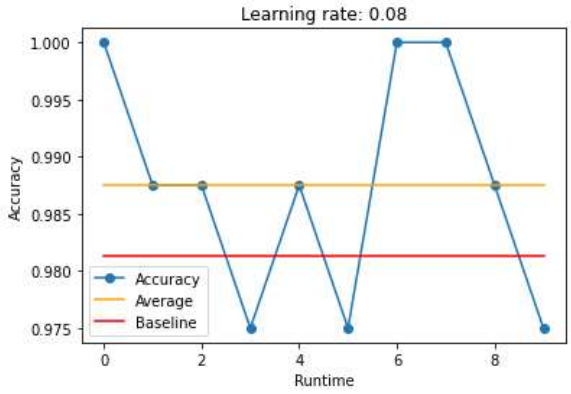
1- We used different values for the learning rates as the following:

**Training the model generated in the previous step with different values for learning rate**

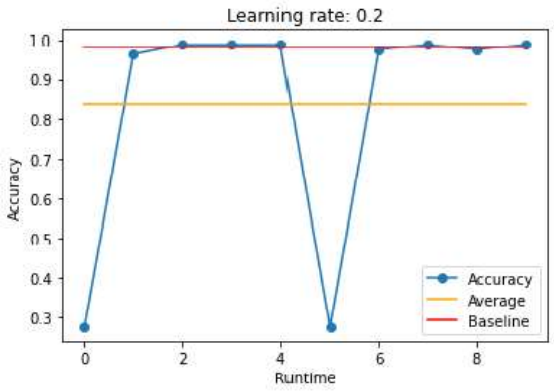
```
act='logistic'
hl=(4,4)

for lr in [0.08,0.15,0.2,0.25,0.3]:
    scores=[]
    for i in range(10):
        clf = MLPClassifier(random_state=i,hidden_layer_sizes=hl, max_iter=1000,
                            learning_rate_init=lr,activation=act).fit(TrX, Try)
        clf.predict_proba(TeX)
        clf.predict(TeX)
        sc=clf.score(TeX, Tey)
        scores.append(sc)
    smean=st.mean(scores)
    print(smean)
    plt.plot(scores,marker='o',label='Accuracy')
    x = []
    y=[]

    for j in range(10):
        x.append(j)
        y.append(smean)
    plt.plot(x,y,color='orange',label='Average')
    plt.plot(x,z,color='red',label='Baseline')
    plt.xlabel('Runtime')
    plt.ylabel('Accuracy')
    plt.title('Learning rate: '+str(lr))
    plt.legend()
    plt.show()
```

The best average accuracy was 98.75% for the learning rate 0.08 as the following figures:

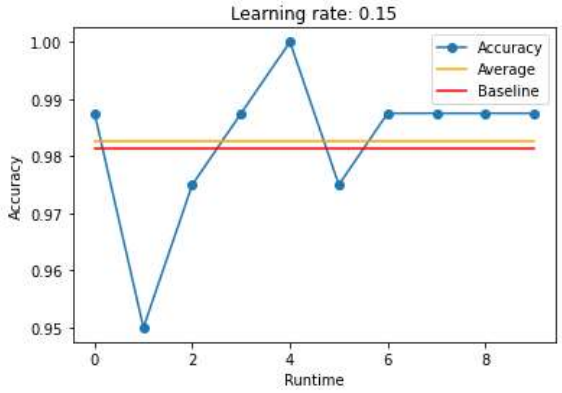
Average accuracy 0.9875



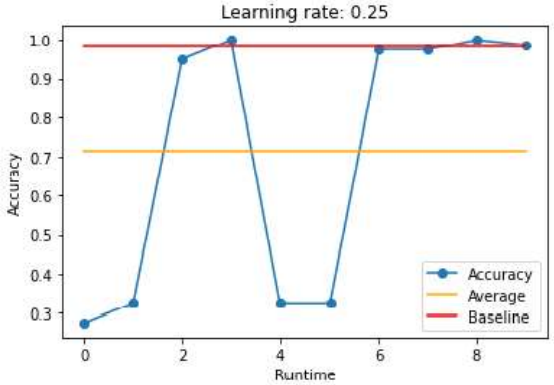

Average accuracy 0.8400000000000001



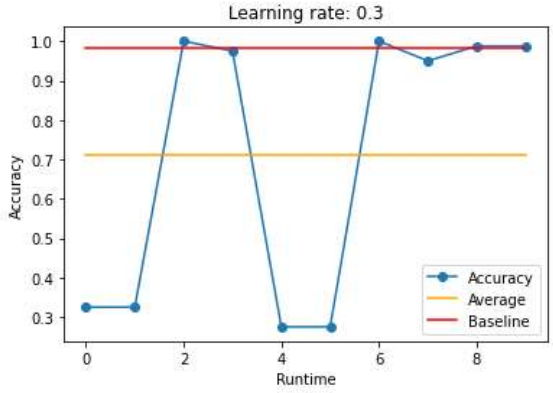

Average accuracy 0.9825




Average accuracy 0.71375




Average accuracy 0.71



2- The final parameters for the model were:
Learning rate = 0.08
Activation function ➔ Sigmoid function
2 hidden layers

We trained the model with them as the following:

## Final best Model

```
clf = MLPClassifier(random_state=0, max_iter=1000,activation='logistic',
                    hidden_layer_sizes=(4,4),learning_rate_init=0.08).fit(TrX, Try)
clf.predict_proba(TeX)
clf.score(TeX, Tey)
print(clf.score(TeX, Tey))
```

```
1.0
```

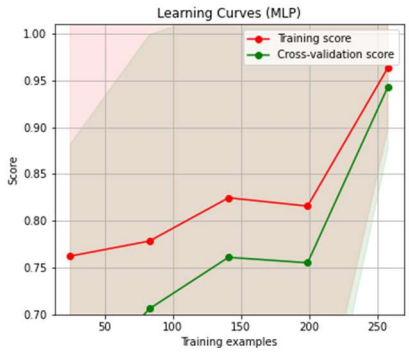
After this we used the scikit-learn function plot_learning_curve to plot the learning curve for the final model.

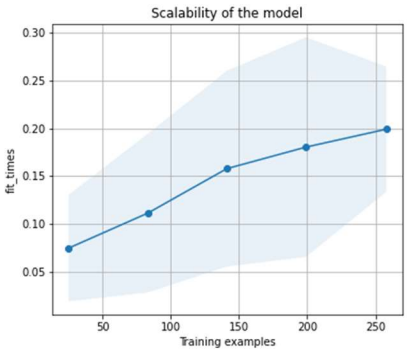
## Plotting the learning Curve for the final model

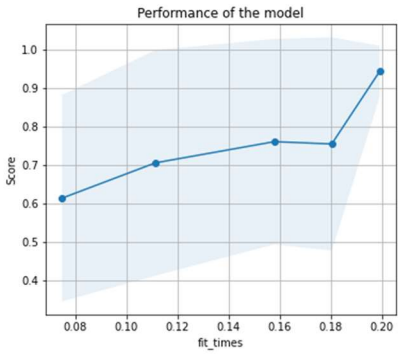
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
```

```
    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


# fig, axes = plt.subplots(1, 1, figsize=(10, 15))

# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)



title = r"Learning Curves (MLP)"
plot_learning_curve(clf, title, TrX, Try, ylim=(0.7, 1.01),
                    cv=cv, n_jobs=4)

plt.show()
```

These were the resulted learning curves:



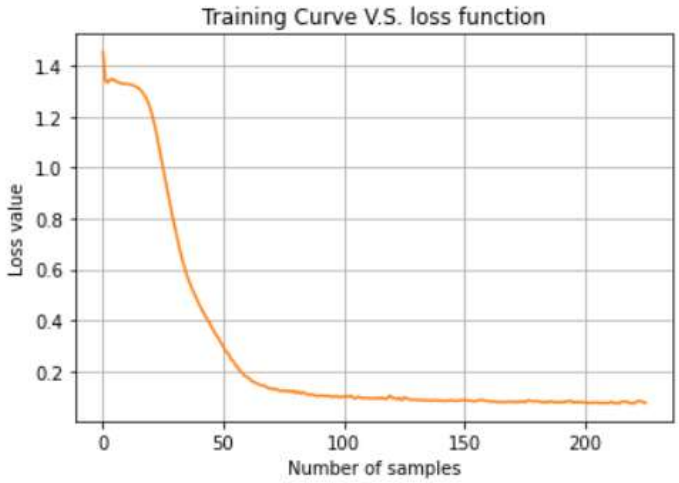



We also plotted the learning curve V.S the loss function as the following:

## Training curve V.S. Loss function

In [29]:
```
axes = plt.plot(1, 1)
plt.plot(clf.loss_curve_);
plt.grid()
plt.xlabel("Number of samples")
plt.ylabel("Loss value")
plt.title("Training Curve V.S. loss function");
```




Here was the confusion matrix combined with the accuracy:

```
from sklearn.metrics import confusion_matrix,plot_confusion_matrix
y_test_list = Tey.values.tolist()
print(clf.score(TeX, Tey))
plot_confusion_matrix(clf,TeX,Tey)
```

```
1.0

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x203fb6c93a0>
```