

PART3 and PART 4

```
result_dataframe

learning_rate    neurons      result      mean      std
0        0.100         1  Average Accuracy on cross-validation: 0.911111...  0.600000  0.284149
1        0.100         3  Average Accuracy on cross-validation: 0.911111...  0.966667  0.027217
2        0.100         5  Average Accuracy on cross-validation: 0.911111...  0.966667  0.027217
3        0.010         1  Average Accuracy on cross-validation: 0.911111...  0.444444  0.111111
4        0.010         3  Average Accuracy on cross-validation: 0.911111...  0.955556  0.041574
5        0.010         5  Average Accuracy on cross-validation: 0.911111...  0.977778  0.027217
6        0.001         1  Average Accuracy on cross-validation: 0.911111...  0.444444  0.140546
7        0.001         3  Average Accuracy on cross-validation: 0.911111...  0.533333  0.282406
8        0.001         5  Average Accuracy on cross-validation: 0.911111...  0.777778  0.126686

learning_rate = [0.1, 0.01, 0.001]
num_hiddenlayer_nuerons = [1, 3, 5]
results = []

for LR in tqdm(range(len(learning_rate))):
    for N in range(len(num_hiddenlayer_nuerons)):
        clf = MLPClassifier(solver='sgd', activation = 'logistic',
                            learning_rate_init= learning_rate[LR],
                            learning_rate='constant', max_iter=1000,
                            hidden_layer_sizes=num_hiddenlayer_nuerons[N],)
        # 5-fold cross validation
        cv_results = cross_val_score(clf, X_train, y_train, cv=5)
        print(f"\n\nThe learning rate is: {learning_rate[LR]}. The number of nuerons: {num_hiddenlayer_nuerons[N]}.\n")
        print(f"Mean: {cv_results.mean()}\nstd: {cv_results.std()}\n")
        result = {
            "learning_rate" : learning_rate[LR],
            "neurons" : num_hiddenlayer_nuerons[N],
            "result" : msg,
            "mean" : cv_results.mean(),
            "std" : cv_results.std()
        }
        results.append(result)
    result_dataframe = pd.DataFrame(results)
```

(1)

(2)

```
max_index = result_dataframe["mean"].idxmax()
print("The row index of the max mean: (max_index)=")
result_dataframe.iloc[max_index]

The row index of the max mean: 5
      5
learning_rate          0.01
neurons                  5
result  Average Accuracy on cross-validation: 0.577778...
mean            0.977778
std             0.027217

dtype: object

clf = MLPClassifier(solver='sgd', activation='logistic',
                     learning_rate_init=0.01, learning_rate='constant', max_iter=1000, verbose=1,
                     hidden_layer_sizes=[10], random_state=1)
cv_results = cross_val_score(clf, X_train, y_train, cv=5)

msg = "Average Accuracy on cross-validation: %f (%f) %"
cv_results.mean(), cv_results.std())
clf.fit(X_train, y_train)

Iteration 002, loss = 0.2941019
Iteration 003, loss = 0.2867745
Iteration 054, loss = 0.286639332
Iteration 055, loss = 0.28640479
Iteration 056, loss = 0.286176888
Iteration 057, loss = 0.285935379
Iteration 058, loss = 0.28574986
Iteration 059, loss = 0.28547275
Iteration 060, loss = 0.28524125
Iteration 061, loss = 0.28501533
Iteration 062, loss = 0.28478002
Iteration 063, loss = 0.284558630
Iteration 064, loss = 0.28432217
Iteration 065, loss = 0.284099263
Iteration 066, loss = 0.283876686
Iteration 067, loss = 0.283637332
Iteration 068, loss = 0.283416054
Iteration 069, loss = 0.28318435
Iteration 070, loss = 0.282957972
Iteration 071, loss = 0.282733578
```

(3)

```
/usr/local/lib/python3.12/dist-packages/sklearn/network/_multilayer_perceptron.py:691
    warnings.warn(
        "MLPClassifier"
        "MLPClassifier(activation='logistic', hidden_layer_sizes=(5,),"
        "learning_rate_init=0.01, max_iter=1000, solver='sgd', verbose=1)"
```

```
y_prediction = clf.predict(X_test)
accuracy = accuracy_score(y_test,y_prediction)
```

```
y_prediction
```

```
array(['Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-setosa', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica'], dtype='|S15')
```

```
accuracy
```

```
0.95
```

(4)

I tested all nine combinations of hyperparameters and created a Pandas DataFrame to record the results (see Figure 1 and 2). Based on the analysis, the optimal configuration was a learning rate of **0.01** and **5 neurons** in the hidden layer (see Figure 3). With these parameters, the model achieved an accuracy of **95%** on the test dataset (see Figure 4).

PART 5

5.1

$$y = 1x_1 + 2x_2 + 0 \Rightarrow 1(0.1) + 2(0.2) + 0 = 0.1 + 0.4 = 0.5$$

$$\text{Sigmoid} = \frac{1}{1+e^{-0.5}} = 0.622$$

5.2

$$\Delta W_i = -\alpha \cdot (\text{target} - \text{out}) \times \text{out} \times (1-\text{out}) \times x_i$$

$$\begin{matrix} & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0.1 & 0 & 0.776 & 0.776 & 0.776 & 0.622 \end{matrix}$$

$$\Delta W_i = -(0.1) \cdot (0 - 0.776) \times (0.776) \times (1 - 0.776) \times (0.622) = 8.39 \times 10^{-3}$$

$$W_{01} = 2 - (8.39 \times 10^{-3}) = 1.99 \Rightarrow \Delta W_{01} = 8.39 \times 10^{-3}$$



~~↑ this is the new weight.~~