

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"\nThe learning rate is: {learning_rate[LR]}. The number of neurons: {num_hiddenlayer_nuerons[N]}.*")
        print(f"Mean: {cv_results.mean()} and std: {cv_results.std()}")
        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(f'The row index of the max mean: {max_index}')
result_dataframe.iloc[max_index]
```

The row index of the max mean: 5

learning_rate	0.01
neurons	5
result	Average Accuracy on cross-validation: 0.977778...
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=(5,))
# 5-fold cross validation
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 952, loss = 0.2911919
Iteration 953, loss = 0.29687445
Iteration 954, loss = 0.29663932
Iteration 955, loss = 0.29640479
Iteration 956, loss = 0.29617088
Iteration 957, loss = 0.29593757
Iteration 958, loss = 0.29570486
Iteration 959, loss = 0.29547225
Iteration 960, loss = 0.29524125
Iteration 961, loss = 0.29501033
Iteration 962, loss = 0.29478802
Iteration 963, loss = 0.29455090
Iteration 964, loss = 0.29432117
Iteration 965, loss = 0.29409263
Iteration 966, loss = 0.29386468
Iteration 967, loss = 0.29363732
Iteration 968, loss = 0.29341054
Iteration 969, loss = 0.29318435
Iteration 970, loss = 0.29295873
Iteration 971, loss = 0.29273320
```

(3)

```
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-virginica', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica'], dtype='<U15')
```

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

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

$0.1 \quad 0 \quad 0.776 \quad 0.776 \quad 0.776 \quad 0.622$

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

$$W_{0,1} = 2 - (8.39 \times 10^{-3}) = 1.99 \Rightarrow \Delta W_{0,1} = 8.39 \times 10^{-3}$$

→ This is the new weight.