EARIN
Laboratory report
EXERCISE 5: Artificial Neural Networks

Paweł Borsukiewicz & Bartłomiej Mastej

Table of contents

1. Introduction

For the purpose of the laboratory N-layer perceptron, utilizing stochastic gradient descent during learning process, was created. Further, impact of the number of perceptron layers on final metrics was assessed utilizing 3 cost functions.

2. Implementation

Firstly, dataset was analysed. Relations between metrics were shown of figures 1 and 2. It was discovered that 2 out of 4 metrics have much greater class correlation than the remaining two metrics as presented below, on figure 3. Those features were petal width and petal length. Therefore, for the purpose of the training process only these two metrics were used.

```
#Show on the matrix graph how metrics are distributed
dataset = pd.read_csv('Iris.csv')
scatter_matrix(dataset, alpha=1, figsize=(12, 12))
plt.show()
```

Figure 1. Feature matrix generation

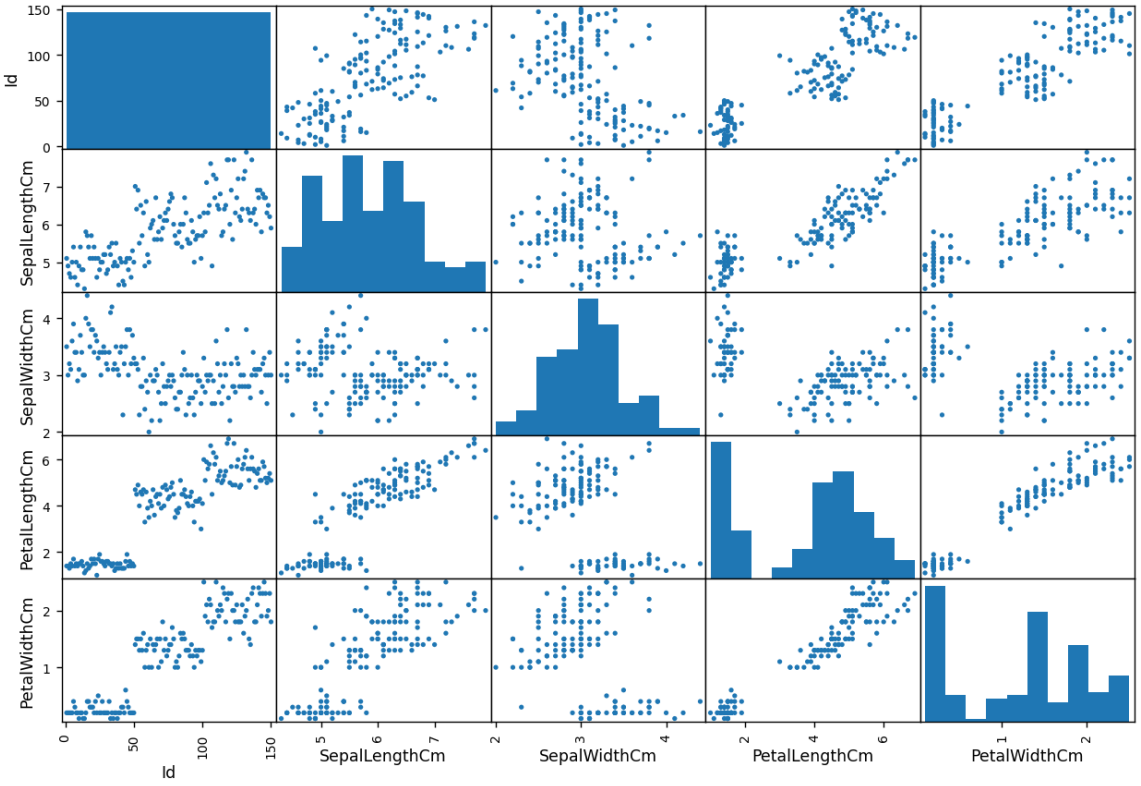

Figure 2. Feature matrix

	Min	Max	Mean	SD	Class Correlation	
sepal length:	4.3	7.9	5.84	0.83	0.7826	
sepal width:	2.0	4.4	3.05	0.43	-0.4194	
petal length:	1.0	6.9	3.76	1.76	0.9490	(high!)
petal width:	0.1	2.5	1.20	0.76	0.9565	(high!)

Figure 3. Correlation statistics

Petal characteristics, presented on figure 5, were visualized using matplotlib library as shown below. Classes were converted to numeric values 0, 1 and 2, which were Iris-setosa, Iris-versicolor and Iris-virginica respectively.

```
#Petal width and length vs specie
scatter_plot = plt.scatter(dataset.data[:,2], dataset.data[:,3], alpha=1, c=dataset.target, edgecolors='black')
plt.colorbar(ticks=([0, 1, 2]))
plt.title('Petals')
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.show()
```

Figure 4. Graph generation with matplotlib

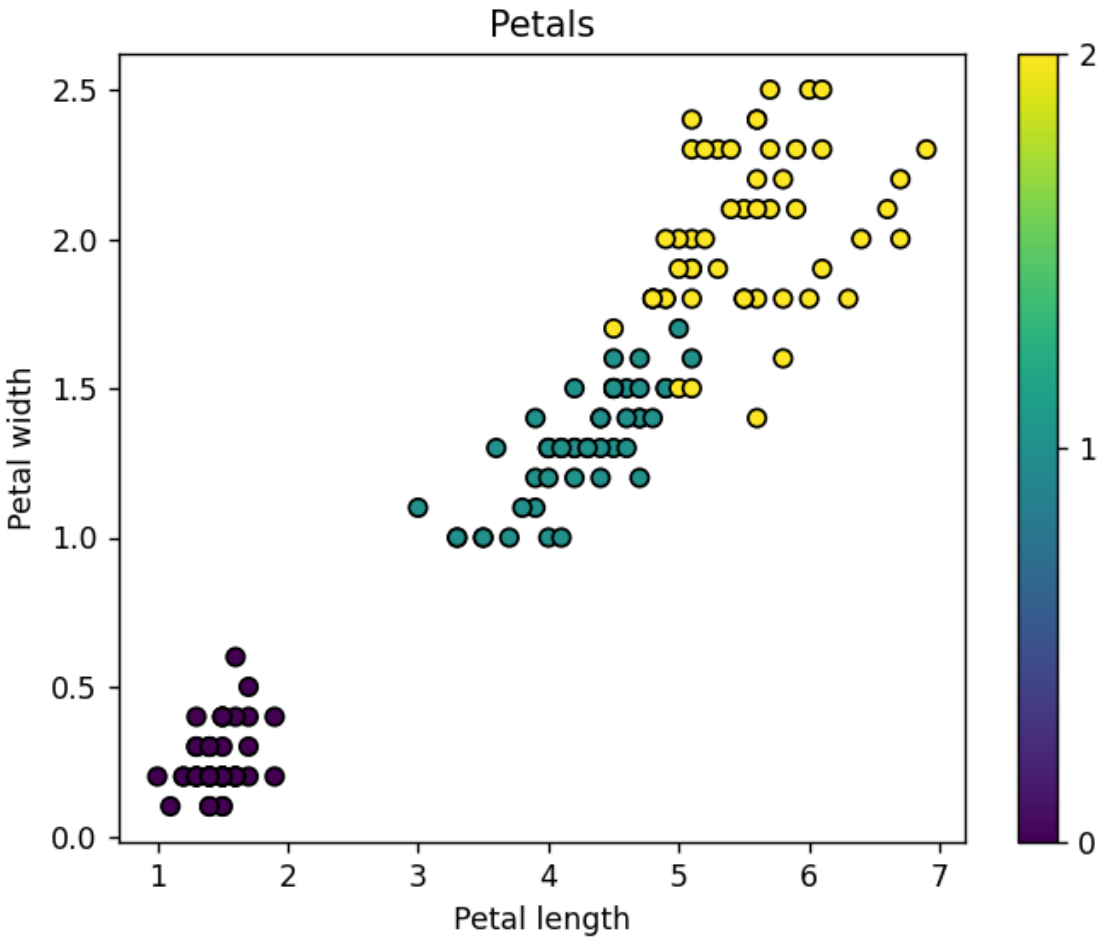

Figure 5. Petal width and length for each class

Further, scikit-learn and numpy libraries were used to prepare the neural network. Firstly, data was pre-processed and divided into training and test set as presented on figure 6.

```
#We create DataFrame by accessing our dataset
#WARNING - relative path
dataset = pd.read_csv("Iris.csv")

#We want to predict values from 'Species' column
toPredict = np.array(dataset['Species'])

# Remove the data to predict from the dataset
# We will just focus on columns 3 and 4 with petal width and length
dataset = dataset.iloc[0:150, 2:4]

#Store column labels
metrics_list = list(dataset.columns)

#Store metrics
metrics = np.array(dataset)

#Divide dataset into training and testing set, we can change random_state to change the
shuffling of the data
train_metrics, test_metrics, train_toPredict, test_toPredict = train_test_split(metrics,
toPredict, test_size = 0.75, random_state = 42)
```

Figure 6. Preparation of the dataset.

Then, number of hidden layers for a range of tests was declared and arrays for storage of error measurements were initialized. In a following step, presented on figure 7, each of the declared models was trained using stochastic gradient descent and tanh function as activation function. Number of learning iterations was increased to 10000 as convergence was sometimes not finished for larger numbers of layers.

```
for layers in hidden_layers:
    start = timeit.default_timer()
    #Solver is stochastic gradient descent
    #Also I added more iteration to assure convergence
    clf = MLPClassifier(activation = "tanh", solver="sgd", hidden_layer_sizes=layers,
    random_state = 1, max_iter = 10000).fit(train_metrics, train_toPredict)
    stop = timeit.default_timer()
    times.append("{0:0.3f}".format(stop - start))
    accuracies.append(clf.score(test_metrics, test_toPredict))
    proba = clf.predict_proba(test_metrics)
    mse_temp = 0
    mae_temp = 0
    ce_temp = 0
```

Figure 7. Training.

Further, 3 cost functions were calculated – mean squared error, mean absolute error and cross entropy – as presented on figure 8.

```
#COST Functions
for flower in test_toPredict:
    if flower != "Iris-setosa":
        mse_temp = mse_temp + proba[iter2][0] * proba[iter2][0]
        mae_temp = mae_temp + proba[iter2][0]
    else:
        ce_temp = ce_temp - np.log(proba[iter2][0])
    if flower != "Iris-versicolor":
        mse_temp = mse_temp + proba[iter2][1] * proba[iter2][1]
        mae_temp = mae_temp + proba[iter2][1]
    else:
        ce_temp = ce_temp - np.log(proba[iter2][1])
    if flower != "Iris-virginica":
        mse_temp = mse_temp + proba[iter2][2] * proba[iter2][2]
        mae_temp = mae_temp + proba[iter2][2]
    else:
        ce_temp = ce_temp - np.log(proba[iter2][2])
    iter2 = iter2 + 1
iter = iter + 1
mse.append(mse_temp/iter2)
mae.append(mae_temp/iter2)
cross_entropy.append(ce_temp/iter2)
iter2 = 0
```

Figure 8. Training.

Finally, results were saved to file and confusion matrix for the latest model was plotted.

```
#Save results to file
filename = "TestResults/MLP.csv"
with open(filename,"w+") as my_csv:
    csvWriter = csv.writer(my_csv,delimiter=';')
    csvWriter.writerow(["Hidden layer sizes","Train time","Accuracy", "MSE", "MAE",
    "Cross-entropy"])
    for _ in times:
        csvWriter.writerow([hidden_layers[iter], times[iter], accuracies[iter], mse[iter]
        , mae[iter], cross_entropy[iter]])
        iter = iter + 1

fig = plot_confusion_matrix(clf, test_metrics, test_toPredict, display_labels=
["Iris-Setosa","Iris-Versicolor","Iris-Virginica"])
fig.figure_.suptitle("Confusion Matrix")
plt.show()
```

Figure 9. Saving results to file and confusion matrix generation

3. Results and comparison of cost functions

For the purpose of the experiment 30 neural networks were trained. Results in detail may be found below.

Hidden layer sizes	Train time[s]	Accuracy	MSE	MAE	Cross-entropy
10	0.646	0.885	0.076	0.205	0.278
(10, 10)	0.556	0.920	0.056	0.154	0.215
(10, 10, 10)	0.870	0.929	0.053	0.134	0.196
(10, 10, 10, 10)	0.659	0.929	0.047	0.113	0.176
(10, 10, 10, 10, 10)	0.804	0.938	0.047	0.111	0.177
(10, 10, 10, 10, 10, 10)	1.088	0.938	0.047	0.108	0.175
(10, 10, 10, 10, 10, 10, 10)	1.435	0.938	0.047	0.109	0.178
(10, 10, 10, 10, 10, 10, 10, 10)	1.671	0.938	0.046	0.106	0.175
(10, 10, 10, 10, 10, 10, 10, 10, 10)	1.033	0.938	0.046	0.104	0.180
(10, 10, 10, 10, 10, 10, 10, 10, 10, 10)	1.338	0.938	0.047	0.108	0.178
25	0.537	0.903	0.067	0.187	0.248
(25, 25)	0.487	0.929	0.051	0.130	0.186
(25, 25, 25)	0.569	0.929	0.052	0.116	0.184
(25, 25, 25, 25)	0.624	0.929	0.051	0.111	0.185
(25, 25, 25, 25, 25)	0.822	0.929	0.049	0.102	0.175
(25, 25, 25, 25, 25, 25)	0.914	0.929	0.050	0.098	0.176
(25, 25, 25, 25, 25, 25, 25)	0.815	0.929	0.047	0.088	0.171
(25, 25, 25, 25, 25, 25, 25, 25)	0.977	0.938	0.047	0.091	0.170
(25, 25, 25, 25, 25, 25, 25, 25, 25)	1.162	0.938	0.046	0.093	0.168
(25, 25, 25, 25, 25, 25, 25, 25, 25, 25)	1.085	0.938	0.046	0.089	0.168
100	0.728	0.938	0.058	0.167	0.218
(100, 100)	1.798	0.929	0.051	0.123	0.180
(100, 100, 100)	2.289	0.929	0.048	0.102	0.165
(100, 100, 100, 100)	2.927	0.929	0.047	0.093	0.162
(100, 100, 100, 100, 100)	3.283	0.929	0.047	0.086	0.162
(100, 100, 100, 100, 100, 100)	3.692	0.938	0.046	0.084	0.158
(100, 100, 100, 100, 100, 100, 100)	3.842	0.938	0.046	0.082	0.161
(100, 100, 100, 100, 100, 100, 100, 100)	3.757	0.938	0.045	0.080	0.161
(100, 100, 100, 100, 100, 100, 100, 100, 100)	4.179	0.938	0.046	0.079	0.161
(100, 100, 100, 100, 100, 100, 100, 100, 100, 100)	5.068	0.938	0.046	0.080	0.164

Figure 10. Detailed results of the experiment

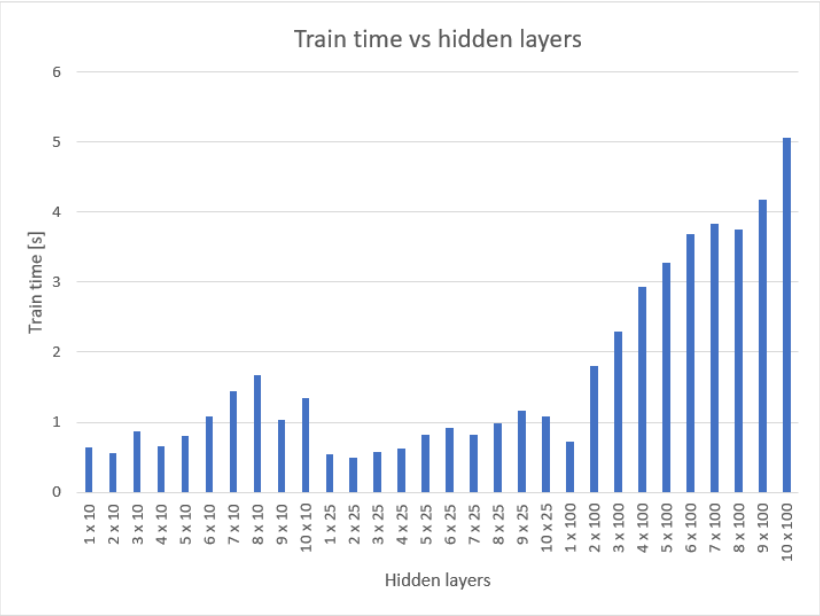

Figure 11. Training times

When training times were considered, it could be observed that usually the more the layers the longer training time, however, there were some exceptions probably due to faster convergence for particular perceptron or system resources allocation instability. It was also observed that perceptron with 25 neurons in each layer used to be trained faster that its counterparts with 10 or 100 neurons in each layer. Hence, training time growth cannot be directly associated with number of neurons in each layer.

Further, accuracy was assessed. Not surprisingly, with the growth of number of layers accuracy tended to increase. The exception can be observed for 2-5 layer perceptrons with 100 neurons , which had lower accuracy than single layer perceptron. Additionally, it seems that larger number of neuron in a layer results in faster accuracy growth for small number of layers.

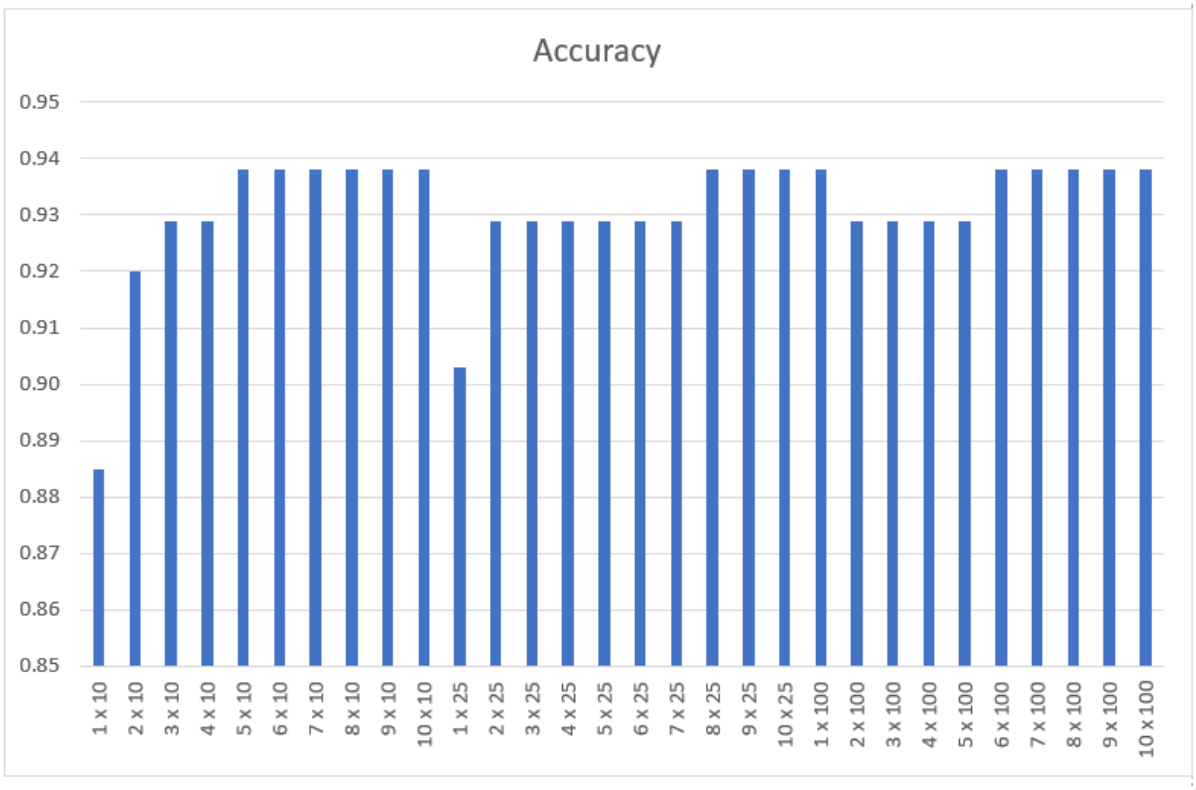

Figure 12. Perceptrons' accuracies

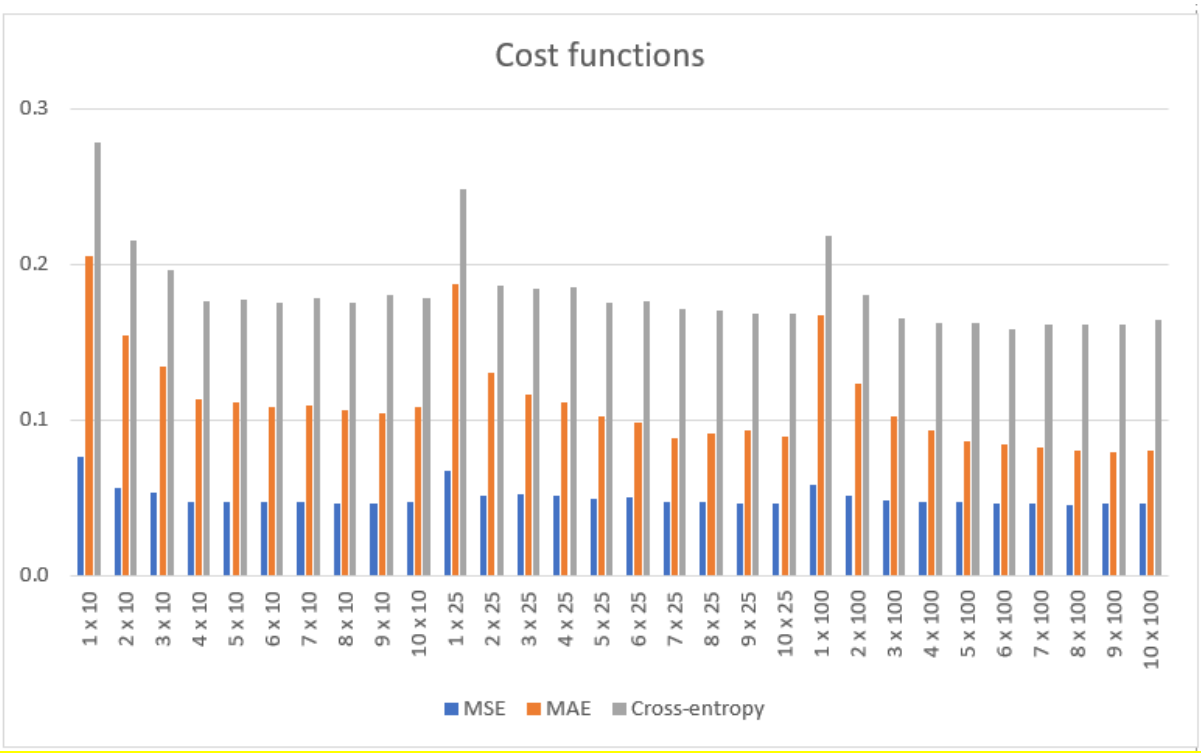

Figure 13. Cost functions

Subsequently, cost functions were considered – presented on figure 13. It could be observed that errors for each function were decreasing with the number of layers used. Minor fluctuations could be observed, however, they did not seem to have significant impact on general trend. Within the tested range of networks it also seemed that the higher the number of neurons in each layer the lower the error. Therefore, there does not seem to be any significant reason to favour any of used cost function above others as all of them provide similar information.

Finally, when the confusion matrix for the highest accuracy networks was generated it could be observed that Iris-setosa was always correctly identified. Perceptron tended to make minor mistakes with classification of boundary individuals of Iris-versicolor and Iris-virginica.

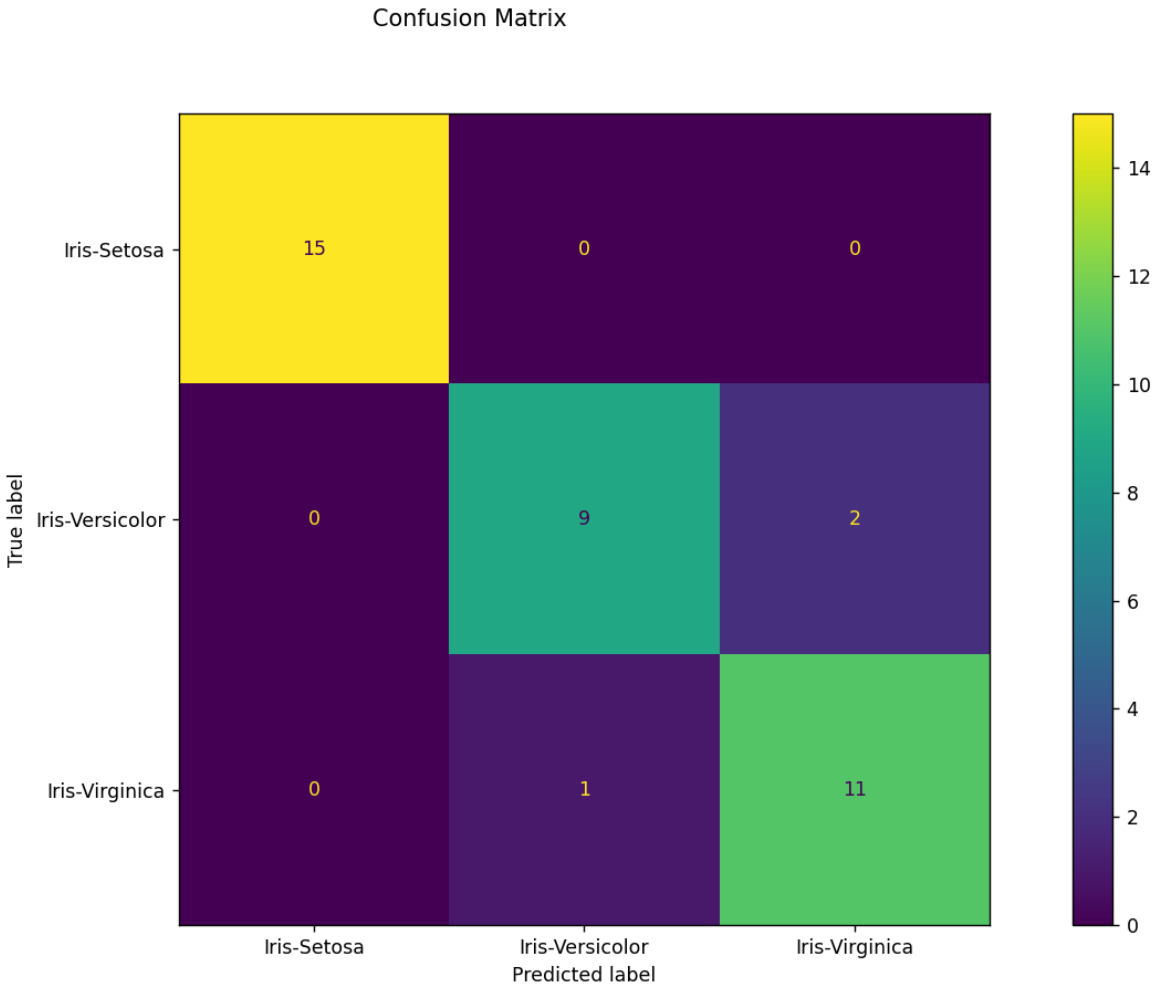

Figure 14.Confusion matrix for the best result.