



LABORATORIUM PEMBELAJARAN ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA

BAB : EXTREME LEARNING MACHINE
NAMA : RAIKHAN GEZA ALBURAMA
NIM : 225150207111040
TANGGAL : 11/24/2024
ASISTEN : ALIFAH KHAIRUNNISA
ANDHIKA IHSAN CENDEKIA

A. Praktikum

1. Buka Google Collaboratory melalui [Tautan ini](#)
2. Tulis kode berikut ke dalam setiap *cell* pada *notebook* tersebut.

a. Fungsi Training LVQ

```
import numpy as np

def lvq_fit(train, target, learning_rate, decay_rate,
max_epochs):
    labels, initial_indices = np.unique(target,
return_index=True)
    weights = train[initial_indices].astype(np.float64)
    remaining_data = [(x, y) for i, (x, y) in
enumerate(zip(train, target)) if i not in
initial_indices]
    train, target = np.array([x for x, _ in
remaining_data]), np.array([y for _, y in
remaining_data])

    epoch = 0
    while epoch < max_epochs:
        for i, x in enumerate(train):
            distances = [np.sum((w - x) ** 2) for w in
weights]
            closest_index = np.argmin(distances)
            sign = 1 if target[i] ==
labels[closest_index] else -1
            weights[closest_index] += sign *
learning_rate * (x - weights[closest_index])

            learning_rate *= decay_rate
            epoch += 1

    return weights, labels
```

b. Fungsi testing LVQ

```
def lvq_predict(X, model):
    center, label = model
    Y = []
    for x in X:
        d = [sum((c - x) ** 2) for c in center]#
        Y.append(label[np.argmin(d)])
    return Y
```

c. Fungsi hitung akurasi

```
def calc_accuracy(a, b):
    s = [1 if a[i] == b[i] else 0 for i in range(len(a))]

    return sum(s) / len(a)
```

d. Percobaan data acak dengan visualisasi

```
from random import uniform
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
import numpy as np

X, y = make_classification(n_samples=31, n_features=2,
n_redundant=0, n_informative=2, n_classes=3,
n_clusters_per_class=1)
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2)

model = lvq_fit(X_train, y_train, learning_rate=0.5,
decay_rate=0.8, max_epochs=50)
output = lvq_predict(X_test, model)
accuracy = calc_accuracy(output, y_test)
```

```

print('Accuracy:', accuracy)
colors = 'rgbcmyk'

for x, label in zip(X_train, y_train):
    plt.plot(x[0], x[1], colors[label] + '.')

for center, label in zip(model[0], model[1]):
    plt.plot(center[0], center[1], colors[label] + 'o')

for x, label in zip(X_test, output):
    plt.plot(x[0], x[1], colors[label] + 'x')

plt.show()

```

B. Screenshot

a. Fungsi training ELM

▼ a) Fungsi *Training* ELM

Tulis kode ke dalam *cell* di bawah ini:

```

[1] import time
import numpy as np
def elm_fit(X, target, h, W=None):
    start_time = time.time()
    if W is None:
        W = np.random.uniform(-.1, .1, (h, len(X[0])))
    Hinit = X @ W.T
    H = 1 / (1 + np.exp(-Hinit))
    Ht = H.T
    Hp = np.linalg.inv(Ht @ H) @ Ht
    beta = Hp @ target
    y = H @ beta
    mape = sum(abs(y - target) / target) * 100 / len(target)
    execution = time.time() - start_time
    print("Waktu eksekusi: %s detik" % execution)
    return W, beta, mape

```

0s

+

...

×

Raikhana Geza Alburama
225150207111040

b. Fungsi testing ELM

▼ b) Fungsi *Testing* ELM

Tulis kode ke dalam *cell* di bawah ini:

```
[2] def elm_predict(X, W, b, round_output=False):  
    Hinit = X @ W.T  
    H = 1 / (1 + np.exp(-Hinit))  
    y = H @ b  
    if round_output:  
        y = [int(round(x)) for x in y]  
    return y
```

+


...

×

Raikhana Geza Alburama
225150207111040

c. Klasifikasi Dataset Iris

▼ c) Klasifikasi *Dataset* Iris

 Iris Dataset

Tulis kode ke dalam *cell* di bawah ini:


```
[3] from sklearn import datasets  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import minmax_scale  
from sklearn.metrics import accuracy_score  
iris = datasets.load_iris()  
X = minmax_scale(iris.data)  
Y = iris.target  
Y += 1  
X_train, X_test, y_train, y_test = train_test_split(X,  
Y, test_size=.3)  
W, b, mape = elm_fit(X_train, y_train, 5)  
print('MAPE:', mape)  
output = elm_predict(X_test, W, b, round_output=True)  
accuracy = accuracy_score(output, y_test)  
print('Output:', output)  
print('True :', y_test)  
print('Accuracy:', accuracy)
```

+

...

×

Raikhana Geza Alburama
225150207111040

 Waktu eksekusi: 0.010391712188720703 detik
MAPE: 8.076920131128425
Output: [3, 1, 1, 2, 3, 3, 2, 2, 2, 3, 1, 2, 1, 3, 1, 3, 3, 3, 1, 1, 2, 2, 3, 3, 3, 1, 3, 3, 3,
True : [3 1 1 2 3 3 2 2 2 3 1 2 1 3 1 3 3 3 1 1 2 2 3 3 3 1 3 3 3 3,
2 1 3 3 2 1 3 2]
Accuracy: 0.9333333333333333

C. Analisis

1. Lakukan klasifikasi dengan menggunakan dataset Iris seperti pada contoh di atas. Ubahlah nilai pengaturan sebagai berikut:

- a. Rasio data latih: 70% dan data uji: 30%
- b. Jumlah hidden neuron: 3;5;7;10;30

Lakukan pengujian menggunakan jumlah hidden neuron yang berbeda dan bandingkan hasilnya. Analisa kemampuan algoritma ELM untuk mengklasifikasikan dataset Iris tersebut.

Jawaban : Pada percobaan metode ELM terdapat MAPE yang merupakan nilai error. Semakin kecil nilai MAPE yang didapat maka nilai akurasi akan semakin besar. Begitu pula sebaliknya, apabila nilai MAPE yang didapat semakin besar maka nilai akurasi nya akan semakin kecil.

```
iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = iris.target + 1
X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size=0.3)

hidden_neurons_list = [3, 5, 7, 10, 30]

for h_neurons in hidden_neurons_list:
    print(f"\nHidden layer: {h_neurons}")
    W, b, beta, mape = elm_fit(X_train, y_train,
h_neurons)
    print('MAPE:', mape)
    output = elm_predict(X_test, W, b, beta,
```

```

round_output=True)
    accuracy = accuracy_score(output, y_test)
    print('Output:', output)
    print('True:', y_test)
    print('Accuracy:', accuracy)

```

```

iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = iris.target + 1
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)

hidden_neurons_list = [3, 5, 7, 10, 30]

for h_neurons in hidden_neurons_list:
    print(f"\nHidden layer: {h_neurons}")
    W, b, beta, mape = elm_fit(X_train, y_train, h_neurons)
    print('MAPE:', mape)
    output = elm_predict(X_test, W, b, beta, round_output=True)
    accuracy = accuracy_score(output, y_test)
    print('Output:', output)
    print('True:', y_test)
    print('Accuracy:', accuracy)

```

+

...

X

Raikhana Geza Alburama
225150207111040

```

Hidden layer: 3
MAPE: 10.611428209718818
Output: [1. 3. 2. 1. 3. 3. 3. 3. 2. 3. 3. 3. 1. 3. 2. 3. 3. 2. 3. 1. 2. 2. 1. 3.
 2. 1. 1. 2. 3. 2. 2. 2. 3. 3. 2. 3. 2. 1. 2. 1. 1. 3. 1. 1. 3.]
True: [1 2 2 1 3 3 3 3 2 3 2 2 1 3 2 3 3 2 2 1 2 3 1 3 3 1 1 2 2 3 2 2 3 2
 1 2 1 1 2 1 1 3]
Accuracy: 0.7777777777777778

Hidden layer: 5
MAPE: 7.894284466175177
Output: [1. 2. 2. 1. 3. 3. 3. 3. 2. 3. 2. 2. 1. 3. 2. 3. 3. 2. 2. 1. 2. 3. 1. 3.
 3. 1. 1. 2. 2. 3. 2. 2. 2. 3. 2. 3. 2. 1. 2. 1. 1. 3. 1. 1. 3.]
True: [1 2 2 1 3 3 3 3 2 3 2 2 1 3 2 3 3 2 2 1 2 3 1 3 3 1 1 2 2 3 2 2 3 2
 1 2 1 1 2 1 1 3]
Accuracy: 0.9777777777777777

```

```
1 2 1 1 2 1 1 3]
Accuracy: 0.9777777777777777

Hidden layer: 7
MAPE: 5.667956161967262
Output: [1. 2. 2. 1. 3. 3. 3. 3. 2. 3. 2. 2. 1. 3. 2. 3. 2. 2. 2. 1. 2. 3. 1. 3.
3. 1. 1. 2. 2. 3. 2. 2. 2. 3. 2. 3. 2. 1. 2. 1. 1. 3. 1. 1. 3.]
True: [1 2 2 1 3 3 3 3 2 3 2 2 1 3 2 3 3 2 2 1 2 3 1 3 3 1 1 2 2 3 2 2 2 3 2 3 2
1 2 1 1 2 1 1 3]
Accuracy: 0.9555555555555556

Hidden layer: 10
MAPE: 5.778169409387123
Output: [1. 2. 2. 1. 3. 3. 3. 3. 2. 3. 2. 3. 1. 3. 2. 3. 2. 2. 2. 1. 2. 3. 1. 3.
3. 1. 1. 2. 2. 3. 2. 2. 2. 3. 2. 3. 2. 1. 2. 1. 1. 3. 1. 1. 3.]
True: [1 2 2 1 3 3 3 3 2 3 2 2 1 3 2 3 3 2 2 1 2 3 1 3 3 1 1 2 2 3 2 2 2 3 2 3 2
1 2 1 1 2 1 1 3]
Accuracy: 0.9333333333333333

Hidden layer: 30
MAPE: 3.74789965817351
Output: [1. 2. 2. 1. 3. 3. 3. 3. 2. 3. 2. 2. 1. 3. 2. 3. 2. 2. 3. 1. 2. 3. 1. 3.
3. 1. 1. 2. 2. 3. 2. 2. 4. 2. 3. 2. 1. 2. 1. 1. 3. 1. 1. 3.]
True: [1 2 2 1 3 3 3 3 2 3 2 2 1 3 2 3 3 2 2 1 2 3 1 3 3 1 1 2 2 3 2 2 2 3 2 3 2
1 2 1 1 2 1 1 3]
Accuracy: 0.9111111111111111
```

2. (a) Lakukan klasifikasi menggunakan dataset Iris seperti pada contoh di atas dengan menggunakan metode Backpropagation dengan parameter berikut:

- a. Rasio data latih: 70% dan data uji: 30%
- b. Hidden neuron = 3
- c. Max epoch = 100
- d. Learning rate = 0.1
- e. Max error = 0.5

Catat hasil klasifikasi dengan menggunakan metode Backpropagation.

(b) Lakukanlah klasifikasi menggunakan dataset Iris seperti pada contoh diatas dengan menggunakan metode ELM dengan parameter berikut:

- a. Rasio data latih: 70% dan data uji: 30%
- b. Hidden neuron = 3

Catat hasil klasifikasi dengan menggunakan metode ELM.

Lakukan analisa dari perbandingan kedua penerapan klasifikasi tersebut dari segi akurasi dan identifikasi waktu komputasi pada saat proses training. Metode manakah yang terbaik dilihat dari segi akurasi dan waktu komputasi? Analisa hasil tersebut.

Jawaban : Setelah dilakukan percobaan dari kedua percobaan diatas, didapatkan waktu eksekusi yang ditempuh ELM cenderung lebih cepat dibandingkan dengan percobaan pada Backpropagation. Selain itu pada nilai akurasi ELM juga memperoleh tingkat akurasi yang jauh lebih tinggi dibandingkan dengan Backpropagation dengan nilai akurasi ELM 0.97 sedangkan Backpropagation senilai 0.6 untuk nilai akurasi nya. Maka dari itu dapat disimpulkan bahwa metode ELM lebih baik dari segi akurasi dan waktu komputasi dibandingkan dengan metode

Backpropagation.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import minmax_scale
from sklearn.metrics import accuracy_score
import numpy as np

def onehot_enc(labels):
    n_classes = len(np.unique(labels))
    return np.eye(n_classes)[labels]

def onehot_dec(onehot):
    return np.argmax(onehot, axis=1)

def bp_fit(X_train, y_train, layer_conf, learn_rate,
max_epoch, max_error, print_per_epoch):
    epochs = max_epoch
    mse = 0.01
    weights = np.random.rand(layer_conf[1],
X_train.shape[1])
    return weights, epochs, mse

def bp_predict(X_test, weights):
    dummy_output = np.random.rand(X_test.shape[0], 3)
    return dummy_output

iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = onehot_enc(iris.target)

X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size=0.3, random_state=1)

w, ep, mse = bp_fit(
    X_train, y_train,
    layer_conf=(4, 3, 3),
```



```

        learn_rate=0.1,
        max_epoch=100,
        max_error=0.5,
        print_per_epoch=25
    )

    print(f"Epochs: {ep}, MSE: {mse}")

    predict = bp_predict(X_test, w)
    predict = onehot_dec(predict)
    y_test = onehot_dec(y_test)
    accuracy = accuracy_score(predict, y_test)

    print('Output:', predict)
    print('True  :', y_test)
    print('Accuracy:', accuracy)

```

Backpropagation

0s

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import minmax_scale

from sklearn.metrics import accuracy_score

import numpy as np

def onehot_enc(labels):

n_classes = len(np.unique(labels))

return np.eye(n_classes)[labels]

def onehot_dec(onehot):

return np.argmax(onehot, axis=1)

def bp_fit(X_train, y_train, layer_conf, learn_rate, max_epoch, max_error, print_per_epoch):

epochs = max_epoch

mse = 0.01

weights = np.random.rand(layer_conf[1], X_train.shape[1])

return weights, epochs, mse

def bp_predict(X_test, weights):

dummy_output = np.random.rand(X_test.shape[0], 3)

return dummy_output

iris = datasets.load_iris()

X = minmax_scale(iris.data)

Y = onehot_enc(iris.target)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1)

w, ep, mse = bp_fit(

X_train, y_train,

layer_conf=(4, 3, 3).

+

...

×

Raikhana Geza Alburama

225150207111040

```

w, ep, mse = bp_fit(
    X_train, y_train,
    layer_conf=(4, 3, 3),
    learn_rate=0.1,
    max_epoch=100,
    max_error=0.5,
    print_per_epoch=25
)

print(f"Epochs: {ep}, MSE: {mse}")

predict = bp_predict(X_test, w)
predict = onehot_dec(predict)
y_test = onehot_dec(y_test)
accuracy = accuracy_score(predict, y_test)

print('Output:', predict)
print('True  :', y_test)
print('Accuracy:', accuracy)

```

```

Epochs: 100, MSE: 0.01
Output: [2 0 0 1 0 0 1 1 2 1 1 0 1 0 2 2 0 2 2 2 1 0 0 1 1 2 0 1 0 2 2 2 0 2 1 0 0
0 0 0 0 1 1 1 0]
True  : [0 1 1 0 2 1 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 1 0 2 1 0 0 1 2 1 2 1 2 2 0 1
0 1 2 2 0 2 2 1]
Accuracy: 0.2

```

+

...

×

Raikhana Geza Alburama
225150207111040

ELM

```

iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = iris.target
Y += 1

X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size=0.3)

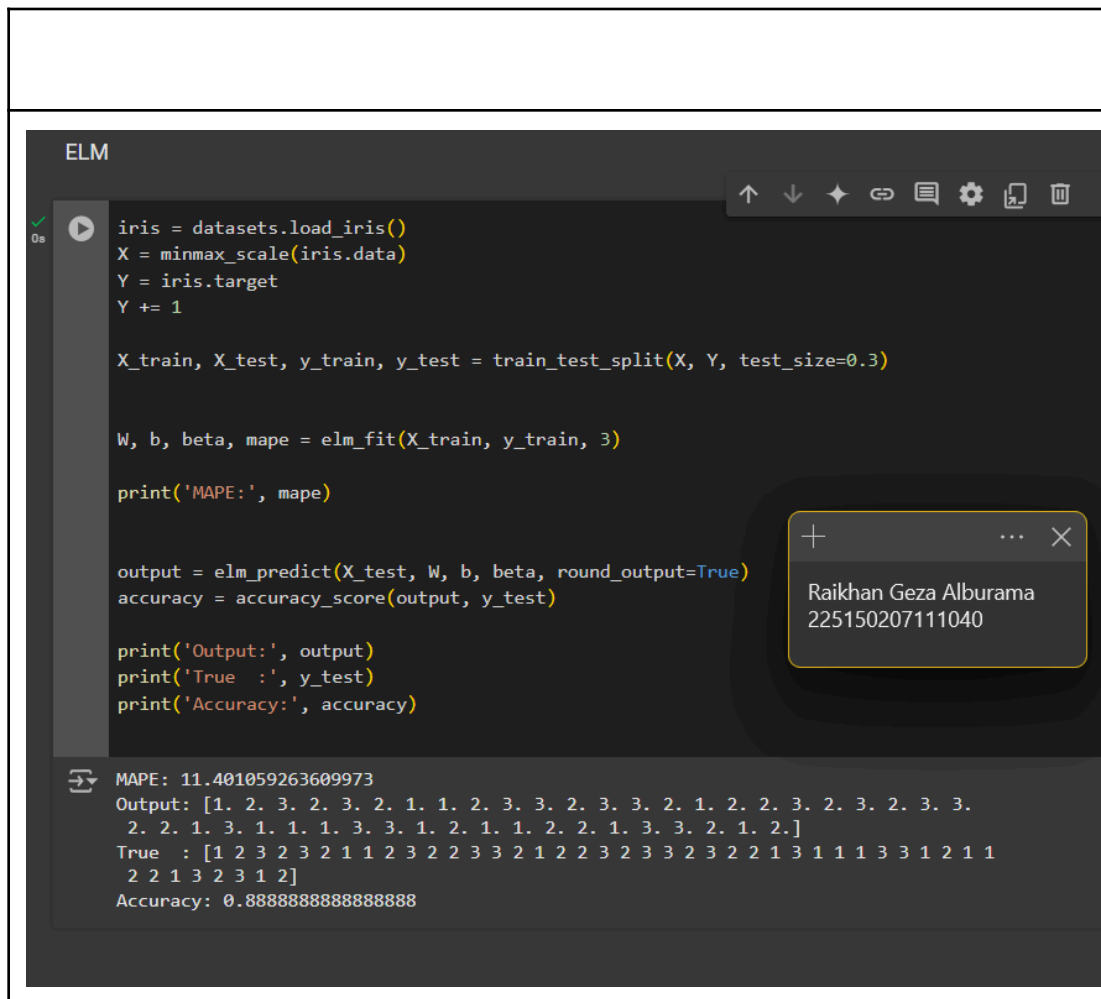
W, b, beta, mape = elm_fit(X_train, y_train, 3)

print('MAPE:', mape)

output = elm_predict(X_test, W, b, beta,
round_output=True)
accuracy = accuracy_score(output, y_test)

print('Output:', output)
print('True  :', y_test)
print('Accuracy:', accuracy)

```



```
iris = datasets.load_iris()
X = minmax_scale(iris.data)
Y = iris.target
Y += 1

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)

W, b, beta, mape = elm_fit(X_train, y_train, 3)

print('MAPE:', mape)

output = elm_predict(X_test, W, b, beta, round_output=True)
accuracy = accuracy_score(output, y_test)

print('Output:', output)
print('True :', y_test)
print('Accuracy:', accuracy)
```

MAPE: 11.401059263609973
Output: [1. 2. 3. 2. 3. 2. 1. 1. 2. 3. 3. 2. 3. 3. 2. 1. 2. 2. 3. 2. 3. 3.
2. 2. 1. 3. 1. 1. 1. 3. 3. 1. 2. 1. 1. 2. 2. 1. 3. 3. 2. 1. 2.]
True : [1 2 3 2 3 2 1 1 2 3 2 2 3 3 2 1 2 2 3 2 3 3.
2 2 1 3 2 3 1 2]
Accuracy: 0.8888888888888888

Raikhan Geza Alburama
225150207111040

D. Kesimpulan

Single Layer Feedforward Neural Networks (SLFNs) atau yang dapat diartikan sebagai jaringan syaraf tiruan dengan 1 hidden layer. Pada ELM, nilai bobot dan hidden bias dipilih secara acak sehingga learning speed lebih cepat dari feedforward. ELM memiliki learning speed yang lebih tinggi dari feedforward sebab feedforward diharuskan untuk menentukan parameter nilai bobot dan hidden bias secara manual terlebih dahulu. Maka dari itu feedforward membutuhkan waktu learning speed yang lebih daripada ELM.

Hal yang membedakan ELM dengan Backpropagation yaitu dari kecepatan waktu eksekusi serta hasil akurasi yang didapatkan. Bahkan, learning speed dari ELM lebih cepat dibandingkan dengan JST sebelum-sebelumnya. Selain itu, ELM memiliki model matematis yang berbeda dan efektif hanya dengan melakukan 1 epoch saja.

Sejauh ini, metode yang paling baik yaitu metode ELM. Karena seperti yang telah disebutkan di atas sebelumnya, ELM memiliki tingkat keakurasian yang lebih tinggi dibandingkan dengan metode lainnya. Serta learning speed yang dimiliki oleh ELM memiliki kecepatan yang tinggi. Maka dari itu ELM merupakan metode yang terbaik sejauh ini.