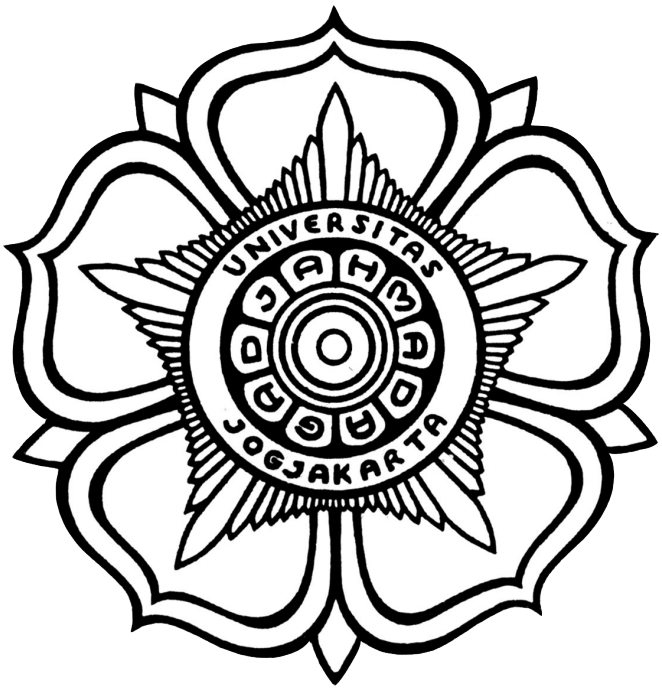
Laporan Pembelajaran Mesin

K-Fold Cross-Validation untuk Klasifikasi 2 Jenis Kelas



Disusun Oleh:

Nendra Haryo Wijayandanu

16/394097/PA/17188

Universitas Gadjah Mada

K-*FOLD CROSS-VALIDATION* UNTUK KLASIFIKASI KELAS IRIS DENGAN DATA BUNGA IRIS

Link GitHub: https://github.com/nendraharyo/Tugas3SLP-K-Fold-Implementation

Bahasa Pemrograman: Python (PyCharm IDE)

Library yang digunakan: Pandas, Matplotlib.pyplot, Numpy

Dokumentasi Program:

Inisiasi Library:

```
import numpy as npy
import pandas as pd
import matplotlib.pyplot as plt
```

Mengimpor Dataset:

```
dataset = pd.read_csv("iris.csv")
dataset = dataset.sample(frac=1)
```

Dataset yang digunakan adalah dataset bunga Iris yag diambil dari website UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/iris

Inisiasi Fungsi:

```
# Penjumlahan Bobot
def weightsum(char, weight, bias):
    w_sum = npy.dot(char, weight) + bias
    return w_sum

# Fungsi Aktivasi (Sigmoid)
def activation(w_sum):
    sigmoid = 1 / (1 + npy.exp(-w_sum))
    return sigmoid

# Membulatkan fungsi aktivasi
def predict(sigmoid):
    if sigmoid > 0.5:
        return 1
    else:
        return 0

# Menentukan error tiap prediksi
def Error(sigmoid, type):
    return 0.5 * ((type - sigmoid) * (type - sigmoid))

# Rumus penentuan bobot
def dTtheta(char, sigmoid, type):
    dtheta = list()
    for x in range(4):
        dtheta.append(2 * char[x] * (type - sigmoid) * (1 - sigmoid) * sigmoid)
```

```
    dtheta.append(2 * (type - sigmoid) * (1 - sigmoid) * sigmoid)
    return dtheta

# Memperbarui Bobot
def W_Update(dtheta, weight):
    for x in range(4):
        weight[x] = weight[x] + (0.1 * dtheta[x])
    return weight

# Memperbarui Bias
def B_Update(dtheta, bias):
    return bias + (0.1 * dtheta[4])
```

Fungsi Validasi:

```
# Fungsi validasi
def Validate(range1, range2, weight, bias):
    char = dataset.loc[range1, "x1":"x4"]
    type = dataset.loc[range1, "type"]
    # mereset nilai sigmoid dan jumlah akurasi
    sigmoid = 0
    acc_sum = 0
    PlotError = list()
    Acc = list()
    for x in range(range1, range2):
        sigmoid = activation(weightsum(char, weight, bias))
        if predict(sigmoid) == type:
            acc_sum += 1
        type = dataset.loc[x + 1, "type"]
        char = dataset.loc[x + 1, "x1":"x4"]
    type = dataset.loc[range2, "type"]
    PlotError.append(Error(sigmoid, type))
    Acc.append(acc_sum / 20)
    Pair = [PlotError, Acc]
    return Pair
```

Fungsi K-Fold

```
# Fungsi K-Fold dengan K sebanyak 5 kali
def KFold(range1, range2, range3, range4, range5, range6):
    pos = 0
    char = dataset.loc[pos, "x1":"x4"]
    type = dataset.loc[pos, "type"]
    # Inisiasi bobot dan bias
    weight = [0.5, 0.5, 0.5, 0.5]
    bias = 0.5
    PlotError_T = list()
    Acc_T = list()
    PlotError_V = list()
    Acc_V = list()


    # Iterasi epoch sebanyak 300 kali
    for epoch in range(300):
        acc_sum = 0
        sigmoid = 0
        for x in range(range1, range3):
            sigmoid = activation(weightsum(char, weight, bias))

            if predict(sigmoid) == type:
```

```
                acc_sum += 1

            dtheta = dTtheta(char, sigmoid, type)
            weight = W_Update(dtheta, weight)
            bias = B_Update(dtheta, bias)
            type = dataset.loc[x + 1, "type"]
            char = dataset.loc[x + 1, "x1":"x4"]
        for x in range(range4, range6):
            sigmoid = activation(weightsum(char, weight, bias))

            if predict(sigmoid) == type:
                acc_sum += 1

            dtheta = dTtheta(char, sigmoid, type)
            weight = W_Update(dtheta, weight)
            bias = B_Update(dtheta, bias)
            type = dataset.loc[x + 1, "type"]
            char = dataset.loc[x + 1, "x1":"x4"]
        type = dataset.loc[range6, "type"]
        PlotError_T.append(Error(sigmoid, type))
        Acc_T.append(acc_sum / 80)
        validate = Validate(range2, range5, weight, bias)
        PlotError_V.append(validate[0])
        Acc_V.append(validate[1])
    Pair = [PlotError_T, Acc_T, PlotError_V, Acc_V]
    return Pair


# Range 1,3,4, dan 6 :data training, range 2 dan 5:validasi
K1 = KFold(0, 39, 80, 99, 40, 79)
K2 = KFold(0, 59, 60, 79, 80, 99)
K3 = KFold(0, 39, 40, 59, 60, 99)
K4 = KFold(0, 19, 20, 39, 40, 99)
K5 = KFold(20, 59, 0, 19, 60, 99)
```

Menampilkan diagram:

```
# Diagram Error Func.
plot_error_training = npy.add(npy.add(npy.add(npy.add(K1[0], K2[0]), K3[0]), K4[0]),
K5[0]) / 5
plot_error_validation = npy.add(npy.add(npy.add(npy.add(K1[2], K2[2]), K3[2]), K4[2]),
K5[2]) / 5
plt.plot(plot_error_training, label="Data Training", color='red')
plt.plot(plot_error_validation, label="Data Validasi", color='green')
plt.title('Diagram Error')
plt.ylabel('Persentase Error')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Diagram Akurasi
plot_acc_training = npy.add(npy.add(npy.add(npy.add(K1[1], K2[1]), K3[1]), K4[1]),
K5[1]) / 5
plot_acc_validation = npy.add(npy.add(npy.add(npy.add(K1[3], K2[3]), K3[3]), K4[3]),
K5[3]) / 5
plt.plot(plot_acc_training, label="Data Training", color='red')
plt.plot(plot_acc_validation, label="Data Validasi", color='green')
plt.title('Diagram Akurasi')
plt.ylabel('Persentase Akurasi')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

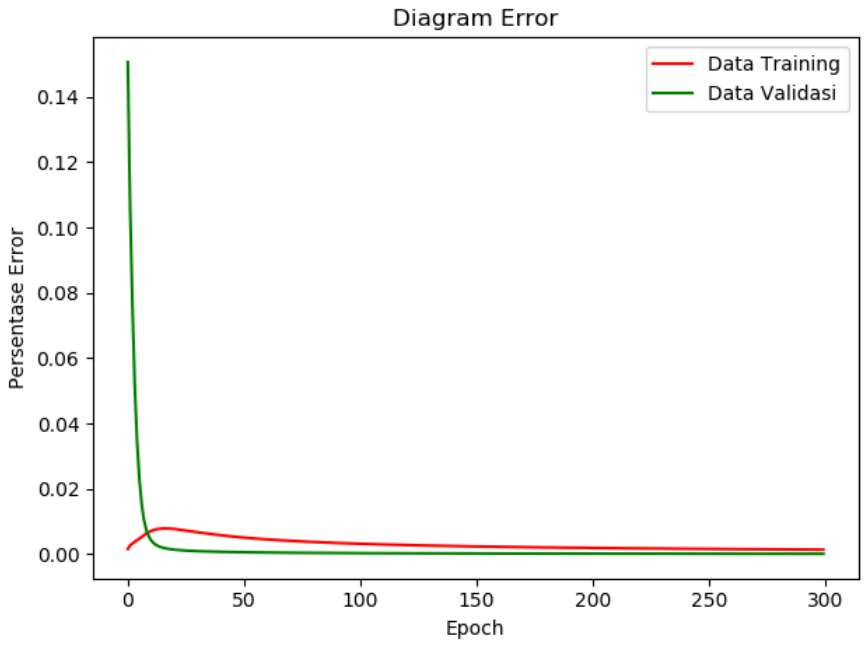
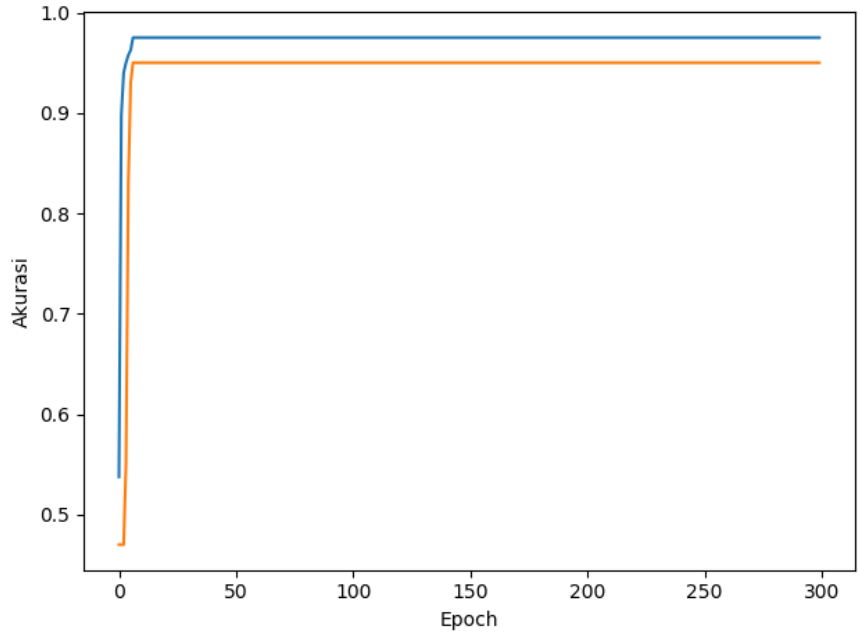


Diagram Akurasi



Kesimpulan:

1. Karena data yang masih sedikit, hanya diperlukan >25 epoch untuk mencapai tingkat akurasi yang tinggi.
2. Tingkat akurasi setelah 25 menjadi sangat stabil (Tidak berubah)

3. Nilai error pada setiap K berbeda-beda