Presentasion Cascade Classify

Anggota Kelompok :
Faizal Rizqi Kholily - 1313621029
Rasyaad Maulana Khandiyas - 1313621020

Penjelasan Tahapan Kode

Import Library

```
1  using DataFrames, Serialization, Statistics
```

Baris ini mengimpor beberapa pustaka yang diperlukan, seperti DataFrames untuk manipulasi data frame, Serialization untuk mengimpor data dari file, dan Statistics untuk menghitung statistik dasar.

Fungsi calculating_mean

```
function calculating_mean(data)
    setosa = data[data[:, end].==1, :]
    versicolor = data[data[:, end].==2, :]
    virginica = data[data[:, end].==3, :]

    mean_setosa = mean(setosa[:, 1:4], dims=1)
    mean_versicolor = mean(versicolor[:, 1:4], dims=1)
    mean_virginica = mean(virginica[:, 1:4], dims=1)

    concatenated_means = [
        mean_setosa
        mean_versicolor
        mean_virginica
    ]

    return concatenated_means
end
```

Fungsi ini mengambil matriks data dan menghitung rata-rata untuk setiap fitur (kolom 1-4) dari setiap kelas (setosa, versicolor, virginica). Hasilnya disusun menjadi satu matriks dan dikembalikan.

Fungsi euclidean_distance

```julia
function euclidean_distance(v1, v2)
    return sum(abs.(v1 - v2))
end
```

Fungsi ini menghitung jarak Euclidean antara dua vektor. Dimana :

- **v1 dan v2** = dua vektor yang akan dibandingkan jarak Euclidean-nya. Dimana vektor ini dapat mewakili titik data atau vektor fitur.
- **v1 - v2** menghitung selisih antara setiap elemen vektor v1 dengan elemen yang sesuai dalam vektor v2.
- **abs** = mengaplikasikan fungsi nilai absolut terhadap setiap elemen hasil pengurangan. Ini memastikan bahwa perbedaan antar elemen dihitung sebagai bilangan non-negatif.
- **sum(...)** = menghitung jumlah dari semua elemen dalam vektor hasil perhitungan sebelumnya. Ini menciptakan nilai total perbedaan antara kedua vektor.
- Hasilnya adalah jarak Euclidean antara v1 dan v2. Fungsi ini mengukur seberapa jauh kedua vektor berada satu sama lain dalam ruang Euclidean.

Fungsi predict_class

```
function predict_class(data_utuh, concatenated_means)
    num_rows = size(data_utuh, 1)
    num_classes = size(concatenated_means, 1)

    predictions = []

    for i in 1:num_rows
        distances = []
        for j in 1:num_classes
            distance = euclidean_distance(data_utuh[i, :], concatenated_means[j, :])
            push!(distances, distance)
        end
        closest_class = argmin(distances)
        push!(predictions, closest_class)
    end

    return predictions
end
```

Fungsi ini digunakan untuk memprediksi kelas untuk setiap baris data dalam suatu dataset. Ini dilakukan dengan menghitung jarak Euclidean antara setiap baris data dan rata-rata kelas yang telah dihitung sebelumnya. Baris data kemudian diklasifikasikan ke dalam kelas dengan jarak Euclidean terdekat. Fungsi ini mengembalikan array prediksi kelas untuk setiap baris data.

Fungsi count_accuracy

```
function count_accuracy(kolom, real_class, predicted_class)
    new_data = hcat(kolom, real_class, predicted_class)
    benar = findall(new_data[:, 2] .== new_data[:, 3])
    accuracy = length(benar) / size(new_data, 1) * 100
    accuracy = floor(accuracy)
    return benar, accuracy
end
```

Fungsi ini membandingkan kolom, class asli dan class prediction dengan meletakkannya pada vector yang sama dan membandingkan kolom asli dan prediction. Setelah itu akan dihitung akurasinya per kolom.

Fungsi update_mu

Fungsi ini memisahkan masing2 kolom dari data, kemudian dihitung meannya, dan dilakukan prediksi dengan fungsi predict_class tadi. Setelah itu akan dihitung akurasi dan dilakukan store index yang prediksinya benar dalam benar1 sampai benar4. Lalu akan diurutkan akurasi tertinggi yang dimana tentu index benar juga akan ikut terurut.

```
function update_mu(data_origin, mean_origin, real_class)
    # Pisahkan data utuh menjadi 4 databaru berdasarkan kolom
    data_kolom_1 = data_origin[:, 1]
    data_kolom_2 = data_origin[:, 2]
    data_kolom_3 = data_origin[:, 3]
    data_kolom_4 = data_origin[:, 4]

    # Pisahkan data hasil concatenated_means menjadi 4 data baru berdasarkan kolom
    mean_kolom_1 = mean_origin[:, 1]
    mean_kolom_2 = mean_origin[:, 2]
    mean_kolom_3 = mean_origin[:, 3]
    mean_kolom_4 = mean_origin[:, 4]

    # Hitung prediksi kelas untuk setiap kolom
    predictions_kolom_1 = predict_class(data_kolom_1, mean_kolom_1)
    predictions_kolom_2 = predict_class(data_kolom_2, mean_kolom_2)
    predictions_kolom_3 = predict_class(data_kolom_3, mean_kolom_3)
    predictions_kolom_4 = predict_class(data_kolom_4, mean_kolom_4)

    benar1, accuracy1 = count_accuracy(data_kolom_1, real_class, predictions_kolom_1)
    benar2, accuracy2 = count_accuracy(data_kolom_2, real_class, predictions_kolom_2)
    benar3, accuracy3 = count_accuracy(data_kolom_3, real_class, predictions_kolom_3)
    benar4, accuracy4 = count_accuracy(data_kolom_4, real_class, predictions_kolom_4)

    accuracies = [accuracy1, accuracy2, accuracy3, accuracy4]
    sorted_accuracies = sort(accuracies, rev=true)

    if sorted_accuracies[1] == accuracy1
        selected_benar = benar1
    elseif sorted_accuracies[1] == accuracy2
        selected_benar = benar2
    elseif sorted_accuracies[1] == accuracy3
        selected_benar = benar3
    elseif sorted_accuracies[1] == accuracy4
        selected_benar = benar4
    end

    return selected_benar, accuracies
end
```

Membaca File

```
#deserilize data_9m
data_origin = deserialize("data_9m.mat")

#convert data to Float64
data_origin = convert(Array{Float64,2}, data_origin)
```

```
mean_origin = calculating_mean(data_origin)

display(mean_origin)
```

Deserialize dan simpan dalam data origin untuk membaca data, lalu kita hitung rata-rata masing2 kolom yang dipisah pada class dan distore pada mean_origin

Menghitung akurasi data_origin

```
mean_origin = calculating_mean(data_origin)

display(mean_origin)
```

```
println("Akurasi")
display(accuracies_iter1)
```

```
3×4 Matrix{Float64}:
 3.22788  1.77705   0.540895  -2.16581
 4.49334  1.16859   3.79687   -0.517301
 5.75513  0.920039  3.96252   -0.3533
Akurasi
4-element Vector{Float64}:
 39.0
 36.0
 44.0
 40.0
```

Setelah menghitung meannya, kita mendapatkan akurasi dengan function count_accuracy

Iterasi 1

```julia
############## Iterasi 1 ##################
selected_benar1, accuracies_iter1 = update_mu(data_origin, mean_origin, data_origin[:, end])
data_iter1 = data_origin[selected_benar1, :]
mean_iter1 = calculating_mean(data_iter1)


println("Akurasi")
display(accuracies_iter1)

println()
println("Update miu Iterasi 1")
display(mean_iter1)
```

```
Update miu Iterasi 1
3×4 Matrix{Float64}:
 3.72337  1.47748   -2.45175  -2.62298
 5.38314  1.01426    3.05864   0.268719
 5.7925   0.640776   8.52761  -0.23875
```

Update miu pada iterasi ke 1

Iterasi 2

```
############## Iterasi 2 ##################
selected_benar2, accuracies_iter2 = update_mu(data_iter1, mean_iter1, data_iter1[:, end])
data_iter2 = data_iter1[selected_benar2, :]
mean_iter2 = calculating_mean(data_iter2)

println()
println("Update miu Iterasi 2")
display(mean_iter2)
```

```
Update miu Iterasi 2
3×4 Matrix{Float64}:
 3.80893  1.00816   -3.59791  -3.13809
 5.38314  1.01426    3.05864   0.268719
 5.4892   0.309955  10.0297   -0.14037
```

Update miu pada iterasi ke 2

Iterasi 3

```
############## Iterasi 3 ##################
selected_benar3, accuracies_iter3 = update_mu(data_iter2, mean_iter2, data_iter2[:, end])
data_iter3 = data_iter2[selected_benar3, :]
mean_iter3 = calculating_mean(data_iter3)

println()
println("Update miu Iterasi 3")
display(mean_iter3)
```

```
Update miu Iterasi 3
3×4 Matrix{Float64}:
 3.8805   0.862511  -3.93312  -3.23292
 5.38314  1.01426    3.05864   0.268719
 5.25519  0.193951  10.6997   -0.0708135
```

Update miu pada iterasi ke-3

Iterasi 4

```julia
############## Iterasi 4 ##################
selected_benar4, accuracies_iter4 = update_mu(data_iter3, mean_iter3, data_iter3[:, end])
data_iter4 = data_iter3[selected_benar4, :]
mean_iter4 = calculating_mean(data_iter4)

println()
println("Update miu Iterasi 4")
display(mean_iter4)
```

```
Update miu Iterasi 4
3×4 Matrix{Float64}:
 3.88963  0.807316  -4.03548  -3.25603
 5.38314  1.01426    3.05864   0.268719
 5.10763  0.128084  10.9987    0.0521168
```

Update miu pada iterasi ke 4

Evaluasi Temuan pada Metode Cascade

Pada metode cascade meskipun waktu *running code* tidak secepat metode lain seperti *Euclidean Distances* biasa tapi kita bisa memperoleh centroid yang lebih optimal dikarenakan iterasi yang berulang kali.