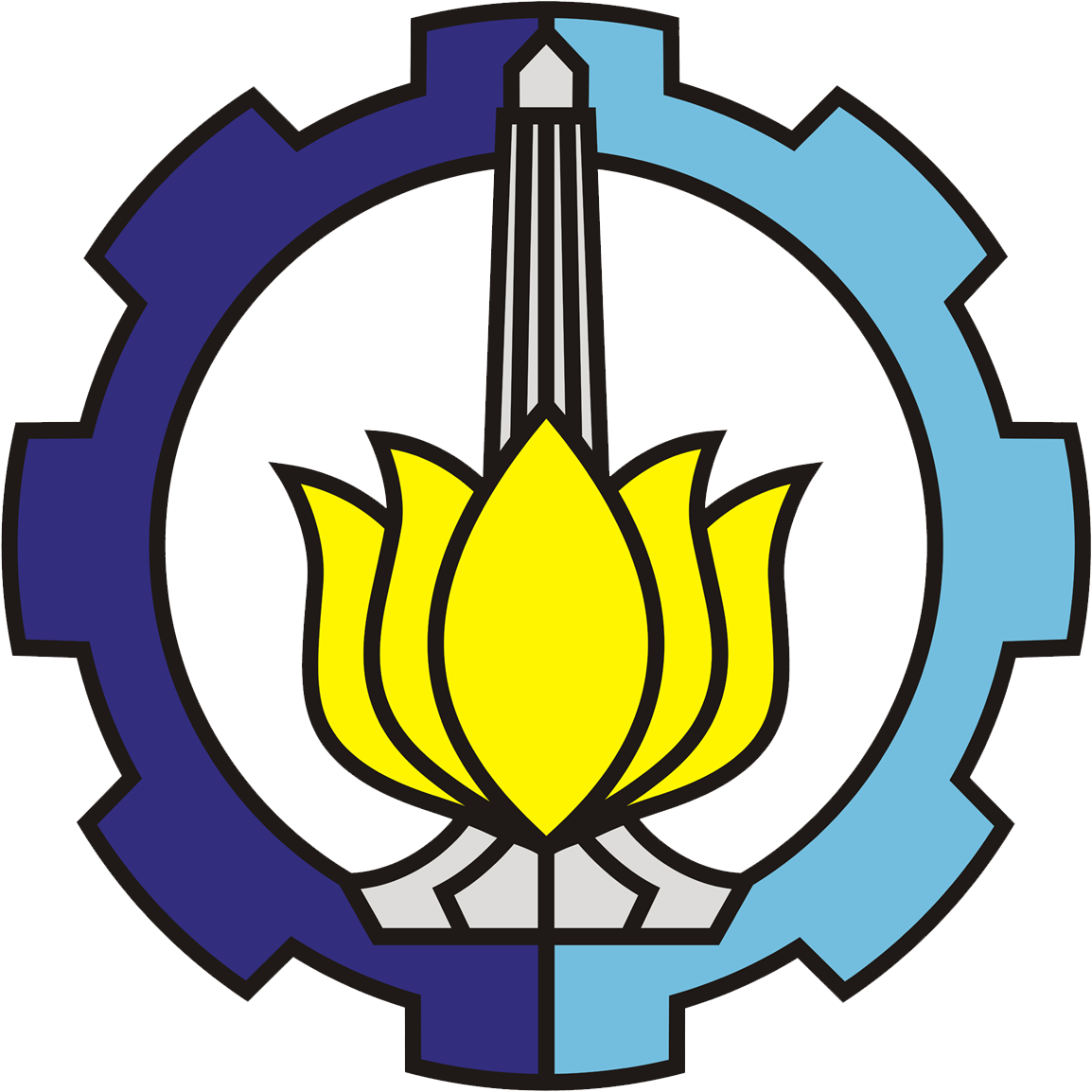
LAPORAN TUGAS 3 KECERDASAN KOMPUTASIONAL

*CLUSTERING* DATA IRIS MENGGUNAKAN METODE K-MEANS DENGAN OPTIMASI INITIAL CENTROID MENGGUNAKAN GENETIC ALGORITHM



Kelas: **D**

Anggota Kelompok :

Devira Wiena Pramintya 5113100045

Hariyanto 5113100061

Wildan Fajria Lazuardy 5113100116

Dosen: DR.Eng. Chastine Fatichah, S.Kom., M.Kom.

**Jurusan Teknik Informatika**

**Fakultas Teknologi Informasi**

**Institut Teknologi Sepuluh Nopember Surabaya**

**2016**

1. **DATASET**
2. Dataset : Iris Data Set
3. Format data : .csv (*comma separated value*)
4. Informasi atribut :

* *Sepal Length in cm*: Real kolom 1
* *Sepal Width in cm*: Real kolom 2
* *Petal Length in cm*: Real kolom 3
* *Petal Width in cm*: Real kolom 4
* *Class*: Nominal (Label) kolom 5 setosa, virginica, versicolor

1. Record yang ada : 150 baris
2. Metode : K-Means (untuk *Clustering*)
3. Bahasa : Python 3.4
4. **DASAR TEORI dan IMPLEMENTASI**

K-Means merupakan *unsupervised learning* sehingga tidak memerlukan data pelatihan, maka data yang ada langsung diklasifikasikan berdasarkan kemiripan kelas (*cluster*).

**Normalisasi 🡪 Menggunakan *Standard Score (Z-Score)***

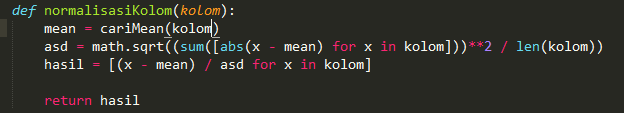
Data dinormalisasi agar *range* fitur menjadi sama.

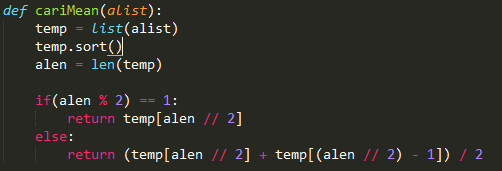
*Standard score* =

x = value

= mean dari cluster

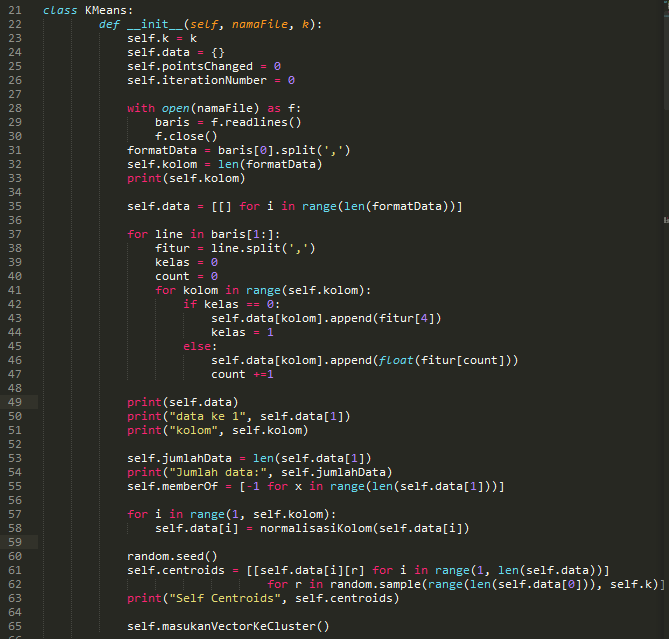
= standard deviasi =





**Algoritma K-Means :**

1. Pilih k data secara random untuk menjadi *centroids*.
2. REPEAT.
3. Tentukan setiap data ke *centroid* terdekat.
4. Update *centroid* dengan menghitung mean setiap cluster.
5. STOP ketika *centroid* tidak berubah.

Class K-Means

**Fungsi 1**

namaFile: nama *file* yang akan di *load* datanya (ada di direktori yang sama)

k: jumlah *cluster* yang diminta

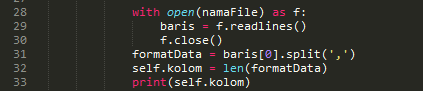
**\_\_init\_\_(namaFile, k)** 🡪 fungsi ini di gunakan untuk *load dataset* dan menerima jumlah **k** (*cluster*) yang diminta serta melakukan normalisasi data.

Line 21 🡪 Mendefinisikan kelas KMeans yang nantinya akan menjadi **object** yang menghandle DataSet.

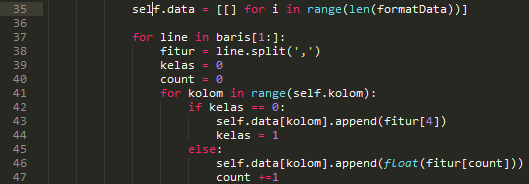
Line 22 🡪 Constructor kelas KMeans

Line 23-27 🡪 atribut kelas:

* K = jumlah *cluster* yang diingikan
* Data = dataset yang akan di *cluster* dimasukkan dalam atribut data yang bertipe *dictionary* **per kolom**
* pointsChanged = untuk menghitung perubahan data pada setiap *cluster*
* iterationNumber = untuk menghitung jumlah iterasi sampai *centroid* tidak berubah



Line 28-33 🡪 meload dataset Iris dan setiap linenya di simpan di variabel **baris**



Line 35 🡪 inisiasi variabel **data** yang bertipe data *dictionary* dengan ukuran sebesar formatData, jadi bentuknya untuk data iris:

Data = [[NamaKelas], [Sepal Length], [Sepal Width], [Petal Length], [Petal Width]] 🡪 Menyimpan data per kolom

Line 37-47 🡪 memasukkan data yang di baca dari dataset iris ke dalam variabel **data.** Bentuknya: (di split berdasar koma, karena .csv)

Data = [[‘setosa’, ‘setosa’, ‘virginica’, …], [5.1, 4.9, 4.7,…], [3.5, 3.0,3.2,…], [1.4, 1.4, 1.3, …], [0.2, 0.2, 0.2, …]]



Line 53 🡪 menyimpan jumlah data (150 *record*)

Line 54 🡪 inisiasi member untuk semua data -1 (belum masuk ke *cluster* manapun)



Line 60-61 🡪 normalisasi setiap kolom (algoritma sudah dijelaskan diatas)

Salah satu *capture* hasil analisis data (agar *range*nya sama) 🡪 menggunakan Z-Score (**Algoritma sudah di jelaskan di atas**)



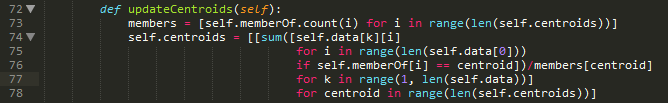


Line 66-67 🡪 mengambil *centroid* sejumlah **k** secara random dari data

**Fungsi 2**

**updateCentroids()** 🡪 Fungsi untuk *update centroids* pada setiap *cluster*.

Fungsi ini akan di lakukan sampai tidak ada lagi *centroid* yang berubah.



Line 73 🡪 mencari jumah anggota pada masing-masing *centroid*

Line 74-78 🡪 mencari *centroid* baru pada setiap *cluster* dengan cara menjumlah atribut pada seluruh anggota *cluster* dan di bagi sebanyak jumlah anggota di *cluster* tersebut.

**Contoh Hasil Update Centroid:**



Berarti pada setiap *cluster* terdapat anggota sebanyak 🡪 [49,49,52]

Update Centroid menghasilkan *centroid* baru 🡪 *centroid* untuk *cluster 1*



**P.S.**: Data pada *range* yang sama setelah di normalisasi

**Fungsi 3**

**masukanVectorKeCluster()** 🡪 Fungsi ini menentukan setiap data (vector) masuk ke *cluster* yang mana



Line 101 🡪 untuk menghitung perubahan data masuk ke *cluster* yang mana

Line 102 🡪 mencari data (vector) masuk ke *cluster* yang mana 🡪 dengan fungsi **masukanVectorKeClusterBerdasarCentroid()**

Salah satu contoh *screenshot* setiap data di masukkan ke *cluster* yang mana

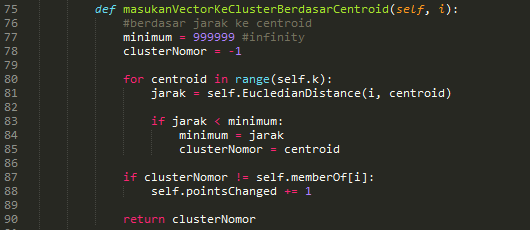


**Fungsi 4**

**i:** data (vector) ke-i yang akan di cari jaraknya ke *centroid*

**masukanVectorKeClusterBerdasarCentroid(i)** 🡪 Untuk membandingkan setiap data ke *centroid* 🡪 untuk *assign* data ke *cluster* di fungsi **masukanVectorKeCluster()**

**Return Value:** nomorCluster



Line 77 🡪 inisiasi jarak minimum ke bilangan tak hingga (*infinity*)

Line 78 🡪 inisiasi nomorCluster **-1** (belum masuk ke *cluster* manapun)

Line 80-85 🡪 mencari jarak setiap data **dengan semua *centroid***, dibandingkan ke *centroid* mana yang paling dekat. Kemudia di *assign* masuk ke *cluster* mana sesuai nomorCentroid (clusterNomor)

Rumus Pencarian Jarak 🡪 untuk menghitung jarak data ke centroid terdekat

**Fungsi 5**

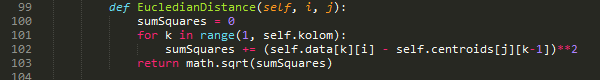
i: nomor data (data ke-i) 🡪 i sejumlah *record*

j: nomor centroid (centroid ke-j) 🡪 j sejumlah **k**

**EucledianDistance(i,j)** 🡪 Untuk mencari jarak dari data ke-i ke *centroid* dari *cluster* ke j

Jika: x = (a,b) dan y = (c,d)

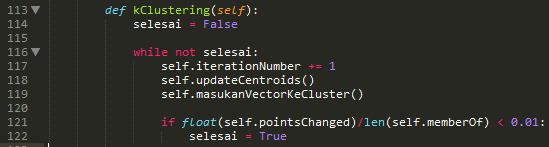
* *Eucledian Distance*:



**Fungsi 6**

**kClustering()** 🡪 fungsi yang melakukan *clustering* (dengan memanggil fungsi-fungsi yang diatas)

Fungsi ini akan melakukan *update centroid* pada setiap *cluster* sampai tidak ada *centroid* berubah dan juga memasukkan setiap data ke dalam sejumlah *cluster* yang diminta.



Line 117-118 🡪 mengupdate *centroid* untuk setiap *cluster* dan memasukkan setiap data ke sejumlah *cluster* yang diminta.

Line 121 🡪 *terminate condition*, yaitu untuk cek kondisi ketika *centroid* tidak lagi berubah dengan cara cek ketika data yang di *assign* ke *centroid* tidak berubah lagi *membership*nya (atau berubahnya kecil sekali sekitar 1%)

**Hasil Clustering Data**

Fungsi **pengelompokanData()** 🡪 Untuk menampilkan anggota *cluster* dan mencari akurasi dari *clustering* oleh K-Means.

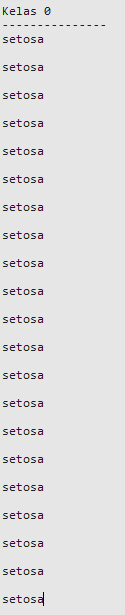
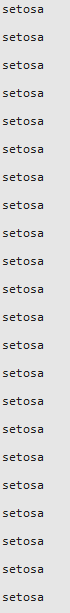
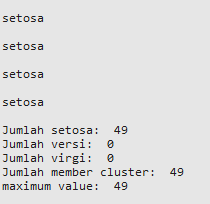
Asumsi: Sudah tahu nilai k = 3 sehingga setiap *cluster* dianggap merepresentasikan setiap *class*

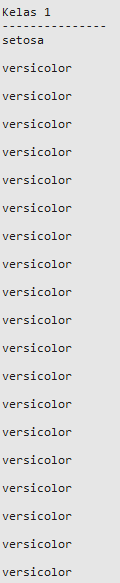
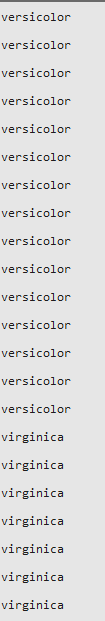
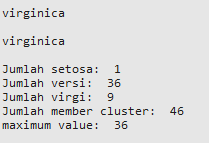


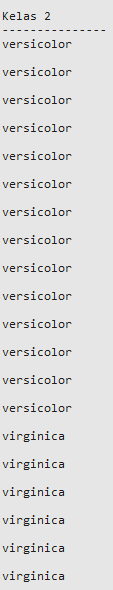
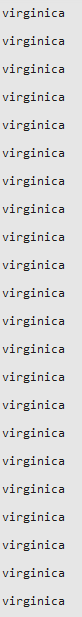
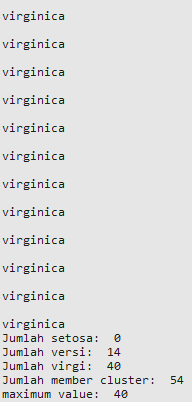
Line 132-139 🡪 dari setiap data di bandingkan *membership*nya dengan nomor *centroid* kemudian di print nama (kelas) yang sudah ada pada data

* Akurasi di cari dengan cara mencari nilai *maximum* dari setiap *cluster* kemudian di bagi dengan sejumlah data pada setiap *cluster*.
* Setelah di temukan akurasi pada setiap *cluster* maka di tambahkan akurasi setiap *cluster* kemudian di bagi 3 (karena asumsi k = 3)





**OPTIMASI METODE K-MEANS MENGGUNAKAN GENETIC ALGORITHM**

Genetic Algorithm merupakan sebuah algoritma *search* yang masuk ke dalam *evolutionary algorithm*. Untuk problem *clustering* ini, *Genetic Algorithm* membantu K-Means menuju **global maxima** dengan mencari *centroid* yang optimal.

Berdasarkan Teori Evolusi Darwin, maka bisa di gambarkan bahwa yang di sebut Populasi adalah kumpulan solusi (*candidate solution*) berupa kromosom (individu) yaitu Genotipe yang kemudian akan melakukan proses evolusi: *selection, cross-over, mutation* dan kemudian menghasilkan kromosom hasil reproduksi antara 2 *parent* yang menentukan *fenotipe* (menjadi apakah individu tersebur: manusia, komputer program, bunga).

Mungkin secara *abstract* algoritma dapat di gambarkan:

While(fitness not satisfied){

Selection();

Cross\_over();

Mutation();

}

Catatan: Pada problem ini tidak di lakukan proses mutasi, di karenakan jika di lakukan mutasi maka *centroid* yang di dapat akan berbeda/tidak ada di dalam *dataset*.

**Implementasi pada Problem *Clustering* Data Iris**

**Fungsi 1**

make\_populasi() 🡪 Fungsi yang akan menginisialisasi populasi

Fungsi ini yang akan membuat populasi dari kumpulan solusi (*candidate solution*).

Representasi kromosom pada Populasi:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C1 | C1 | C1 | C1 | C2 | C2 | C2 | C2 | C1 | C2 | C3 | C4 |

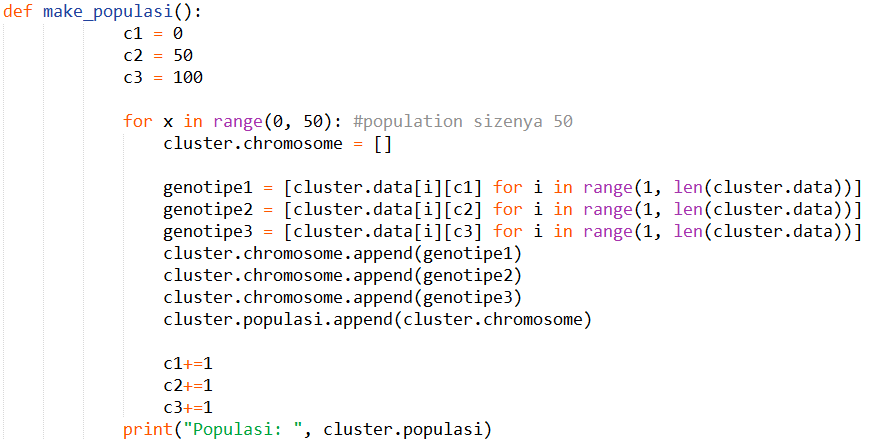
C1 = atribut *centroid 1* (Iris-Setosa)

C2 = atribut *centroid 2* (Iris-Versicolor)

C3 = atribut *centroid 3* (Iris-Virginica)

Kromosom-kromosom ini yang akan di masukkan ke dalam populasi.

JumlahKromosom = JumlahData/3 = 50

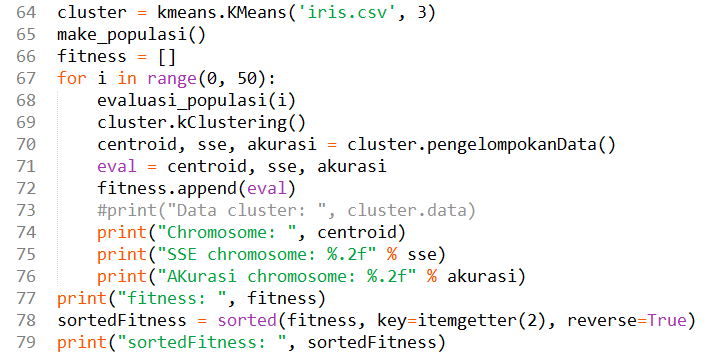
* Maka Populasi awal terdiri dari 50 individu

**Fungsi 2**

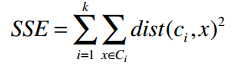
i: kromosom ke-i

Evaluasi\_populasi(i) 🡪 Fungsi yang akan mengeset centroid berdasar kromosom di populasi



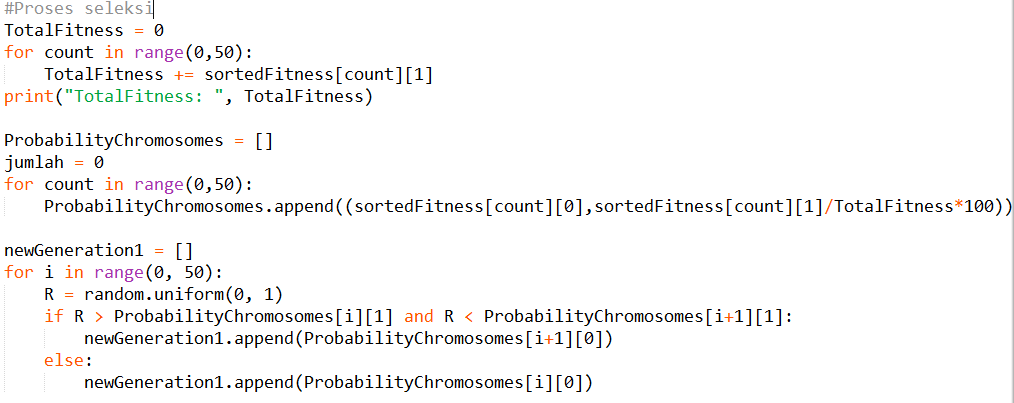


Kode di atas adalah ketika membaca data iris dan menerapkan seluruh *candidate solution* pada algoritma K-Means. Hasil semua *candidate solution* di simpan pada *list* fitness kemudian di *sorting* berdasar nilai sse (*sum of square*) dan di masukkan ke dalam variable sortedFitness. 🡪 Mencari *fitness*, untuk problem ini *fitness function* di ambil nilai *sum of square error*.



**Proses *Selection***

Metode Seleksi yang di gunakan adalah *Roulette Wheel*, yaitu mencari probabilitas dari setiap *chromosome* kemudian *generate* sebuah nilai Random dan di bandingkan dengan *probability chromosome*. Jika nilai Random lebih besar dari pada *probability chromosome* maka di masukkan ke generasi baru.



**Fungsi 3**

ProbabilityChromosomes: populasi yang mau di reproduksi membentuk generasi baru.

Parent di pilih dengan membandingkan *generate random number* dengan *crossOverRate*. Kalau tidak dipilih maka akan 2450 anak.

cross\_over2(ProbabilityChromosomes) 🡪 Fungsi untuk *cross-over* antar 2 kromosom membentuk generasi baru. Metode yang di gunakan adalah *two-point cross-over*.

Mother

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

Father

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

Offspring1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

Offspring2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

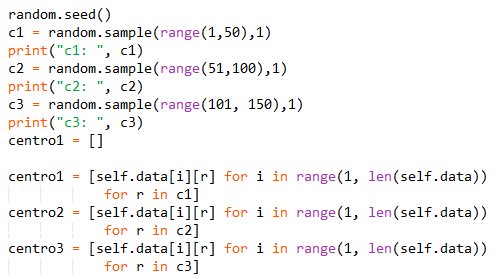


**Proses pembentukan generasi-generasi lain**

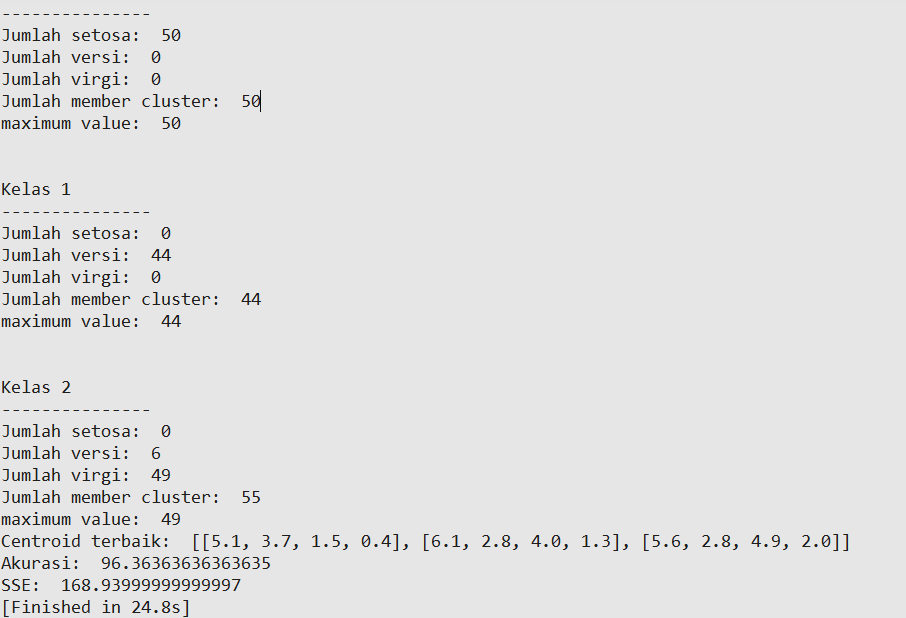
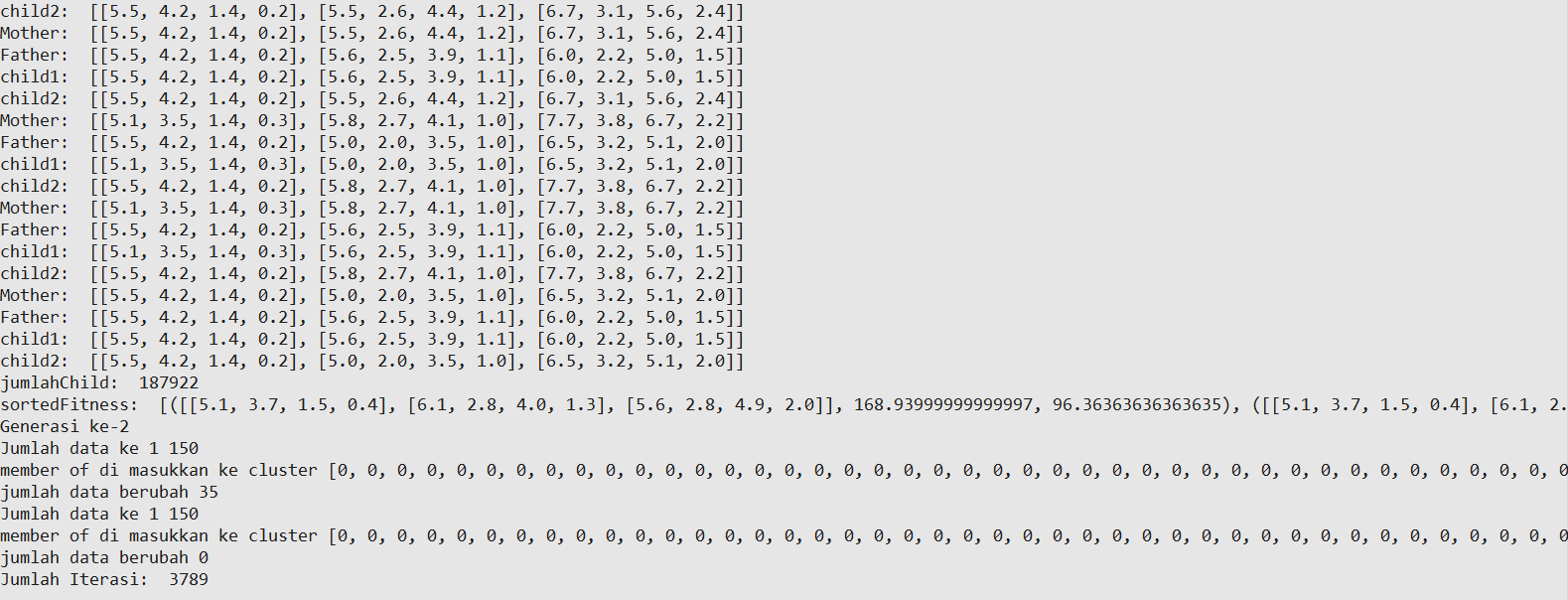
Untuk problem ini, di hasilkan sampai generasi ke-2. Hasil populasi generasi sebelumnya di *cross-over* untuk menghasilkan generasi berikutnya (line 116-117, cross-over line 124). Setelah di lakukan sampai 2 generasi, maka solusi terbaik di ambil dari hasil kromosom populasi generasi 2 dengan nilai akurasi terbaik (tidak memandang sse)



Pada metode K-Means with GA ini, di lakukan modifikasi terhadap K-Means sebelumnya. Jika sebelumnya *centroid* diambil benar-benar *random* (tanpa memandang *class*, karena asumsi kita tidak mengetahui *class* dari tiap *field*). Untuk metode K-Means yang di optimasi GA ini, *centroid* diambil berdasar *class.*



**Output K-Means with GA**



**Tabel Perbandingan Akurasi**

|  |  |  |
| --- | --- | --- |
| Method’s Name | Accuracy | SSE (Sum of Square Error) |
| K-Means Random Centroid | 83.50 | - |
| K-Means Centroid per Class | 84.03 | 2.5 |
| K-Means Random Centroid no Normalization | 91.14 | - |
| K-Means Centroid per Class no Normalization | 89.32 | 128.48 |
| K-Means with GA | 96.3636 | 168.94 |