

# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Pengelompokan (*clustering*) merupakan prosedur untuk mencari struktur alami dari suatu kumpulan data. Proses ini melibatkan pemilihan data atau objek ke dalam kelompok (*cluster*) sehingga objek-objek dalam cluster yang sama akan lebih mirip satu sama lain dibandingkan dengan objek yang berada di *cluster* lain. *Clustering* berguna untuk mereduksi data (mereduksi data dengan volume besar ke dalam kelompok-kelompok dengan karakteristik tertentu), mengembangkan skema klasifikasi (juga dikenal sebagai taksonomi), dan memberikan masukan atau dukungan terhadap hipotesis mengenai struktur suatu data.

*Clustering* merupakan salah satu teknik pembelajaran tak terarah (*unsupervised learning*). Pembagian kelompok dalam *clustering* tidak berdasarkan sesuatu yang telah diketahui sebelumnya, melainkan berdasarkan kesamaan tertentu menurut suatu ukuran tertentu [1].

Salah satu algoritma pengelompokan yang paling sering digunakan adalah *K-means* yang dilakukan dengan cara membagi data ke dalam  $K$  kelompok. Kelompok tersebut dibentuk dengan cara meminimalkan jarak antara titik pusat *cluster* (*centroid*) dengan setiap anggota *cluster* tersebut. Titik pusat *cluster* dicari dengan menggunakan rata-rata (*mean*) dari nilai setiap anggota *cluster*. Dalam hal ini, setiap anggota *cluster* dimodelkan sebagai vektor dalam  $n$  dimensi ( $n$  merupakan banyaknya atribut). *K-means* sudah terbukti efektif dalam melakukan pengelompokan dalam situasi apapun. Namun, cara tersebut tetap saja memiliki kekurangan yaitu dapat terjebak dalam *local optima* tergantung dengan pemilihan *centroid* awal [2].

Masalah *local optima* dapat ditangani menggunakan *Genetic Algorithm* (GA) yang telah terbukti efektif dalam menyelesaikan masalah pencarian dan optimasi. GA merupakan teknik pencarian heuristik tingkat tinggi yang menirukan proses evolusi yang secara alami terjadi [3] berdasarkan prinsip *survival of the fittest*. Algoritma ini dinamakan demikian karena menggunakan konsep-konsep dalam genetika sebagai model pemecahan masalahnya [4].

Dalam GA, parameter dari *search space* dikodekan dalam bentuk deretan objek yang disebut kromosom. Kumpulan kromosom tersebut lalu dikenal sebagai populasi. Pada awalnya, populasi dibangkitkan secara acak. Kemudian, akan dipilih beberapa kromosom menggunakan teknik *roulette wheel selection* berdasarkan fungsi *fitness*. Operasi dasar yang terinspirasi dari Ilmu Biologi seperti persilangan (*crossover*) dan mutasi (*mutation*) digunakan untuk membangkitkan generasi berikutnya. Proses seleksi, persilangan, dan mutasi ini berlangsung dalam jumlah generasi tertentu atau sampai kondisi akhir tercapai.

Fungsi *fitness* tidak hanya berfungsi untuk menentukan seberapa baik solusi yang dihasilkan namun juga menentukan seberapa dekat solusi tersebut dengan hasil yang optimal [4]. Oleh karena itu, diperlukan fungsi *fitness* yang cocok sehingga GA dapat menghasilkan keluaran yang optimal. Pada masalah *clustering* menggunakan GA, maka fungsi *fitness* yang digunakan harus bisa menggambarkan bahwa seluruh elemen sudah berada dalam *cluster* yang terbaik dan sudah sesuai.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, rumusan masalah dari penelitian ini adalah sebagai berikut:

1. Bagaimana algoritma genetik dapat digunakan untuk mengelompokkan dokumen?
2. Bagaimana membangun perangkat lunak yang menggunakan algoritma genetik untuk dapat mengelompokkan dokumen?

## 1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah disebutkan, tujuan dari penelitian ini adalah sebagai berikut:

1. Mempelajari algoritma genetik dan hubungannya dengan pengelompokan dokumen.
2. Membangun perangkat lunak yang mengimplementasikan algoritma genetik untuk dapat mengelompokkan dokumen.

## 1.4 Batasan Masalah

Rumusan masalah yang telah disebutkan memiliki ruang lingkup yang cukup luas. Dengan menyadari terbatasnya waktu serta kemampuan, penelitian ini akan difokuskan dengan memperlihatkan batasan masalah sebagai berikut:

1. Jenis dokumen yang dapat diproses dengan perangkat lunak yang akan dibuat hanyalah *Text Document* dengan ekstensi *TXT*.
2. Informasi dari dokumen yang akan diproses dalam pengelompokan hanya berasal dari teks yang menjadi isi dari dokumen tersebut. Gambar dan *metadata* (pemilik, tanggal modifikasi) tidak akan diperhitungkan.

## 1.5 Metodologi

Langkah-langkah yang akan dilakukan dalam penelitian ini adalah:

1. Melakukan studi literatur mengenai model ruang vektor, *Document Clustering* (pengelompokan dokumen), *Genetic Algorithm* (algoritma genetik), dan penggunaan algoritma genetik dalam pengelompokan dokumen.
2. Mencari dokumen yang akan dijadikan *training* dan *test datasets*.
3. Membuat rancangan perangkat lunak yang menggunakan algoritma genetik sebagai algoritma pengelompokan dokumen.
4. Mengimplementasikan hasil rancangan menjadi perangkat lunak dalam bahasa pemrograman Java.
5. Melatih dan menguji perangkat lunak dengan dokumen yang telah tersedia.
6. Mengevaluasi hasil pengujian lalu lakukan implementasi dan pengujian kembali sampai didapatkan hasil yang sudah sesuai dengan harapan.

## 1.6 Sistematika Pembahasan

Dokumentasi dari penelitian ini akan disajikan dalam enam bab dengan sistematika pembahasan sebagai berikut:

1. Bab 1 Pendahuluan

Bab 1 berisi latar belakang pemilihan "Pengelompokan Dokumen berbasis Algoritma Genetika" sebagai judul dari penelitian ini. Selain itu, dibahas juga rumusan masalah, tujuan penelitian, batasan masalah, serta metodologi penelitian yang menjadi acuan dari penelitian ini.

2. Bab 2 Dasar Teori

Bab 2 memuat landasan teori yang digunakan dalam penelitian ini. Konsep-konsep yang dibahas yaitu temu kembali informasi, pembelajaran mesin, algoritma genetika, dan GA dalam pengelompokan.

3. Bab 3 Analisis

4. Bab 4 Perancangan

5. Bab 5 Implementasi dan Pengujian

6. Bab 6 Kesimpulan



## BAB 2

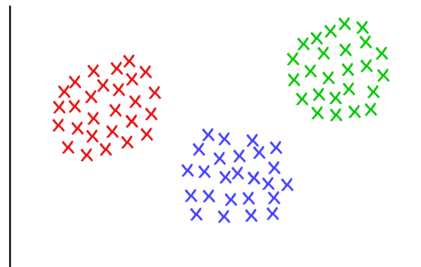
### LANDASAN TEORI

Pengelompokan dokumen berkaitan erat dengan dua bidang ilmu dalam informatika. Pengelompokan dalam informatika merupakan bagian dari bidang pembelajaran mesin. Terdapat dua jenis pengelompokan dalam pembelajaran mesin yaitu *clustering* dan *classification*. *Clustering* merupakan salah satu jenis pembelajaran tak terarah (*unsupervised learning*) karena setiap elemen dikelompokkan berdasarkan karakteristik dari elemen tersebut. Sedangkan *classification* merupakan jenis pembelajaran terarah (*supervised learning*) karena setiap elemen dikelompokkan berdasarkan label yang telah ditentukan sebelumnya. Pada penelitian ini, jenis pengelompokan yang akan digunakan adalah *clustering*.

#### 2.1 Pengelompokan

##### 2.1.1 Definisi Pengelompokan

Pengelompokan (*clustering*) merupakan sebuah metode untuk menggabungkan himpunan objek ke dalam kelompok-kelompok sedemikian rupa sehingga objek dalam kelompok (*cluster*) lebih mirip (karena suatu hal) satu sama lain daripada objek di kelompok lain [5]. Pengelompokan seringkali tertukar dengan istilah klasifikasi yang hanya bertugas untuk memisahkan objek ke dalam kelas-kelas yang telah ditentukan sebelumnya. Masukan dari proses pengelompokan adalah kumpulan objek dan banyaknya kelompok (*cluster*) yang akan dibentuk. Keluaran yang dihasilkan dari proses pengelompokan adalah kelompok objek yang telah dibentuk beserta anggotanya. Setiap objek dikelompokkan berdasarkan kesamaan tertentu. Sebagai ilustrasi dari pengelompokan (Gambar 2.1), terdapat tiga *cluster* yang ditandai dengan warna merah, biru, dan hijau. Objek-objek yang berwarna sama dianggap mirip sehingga dimasukkan ke dalam kelompok yang sama. Begitu juga dengan objek yang berbeda warna dianggap tidak mirip sehingga perlu dipisahkan.



Gambar 2.1: Contoh *cluster* hasil pengelompokan

## 2.1.2 Aplikasi Pengelompokan

Pengelompokan memegang peran penting dalam beberapa bidang seperti *recommender system* dan penambangan data. Berikut adalah penjelasan singkat aplikasi pengelompokan dalam setiap bidang yang telah disebutkan.

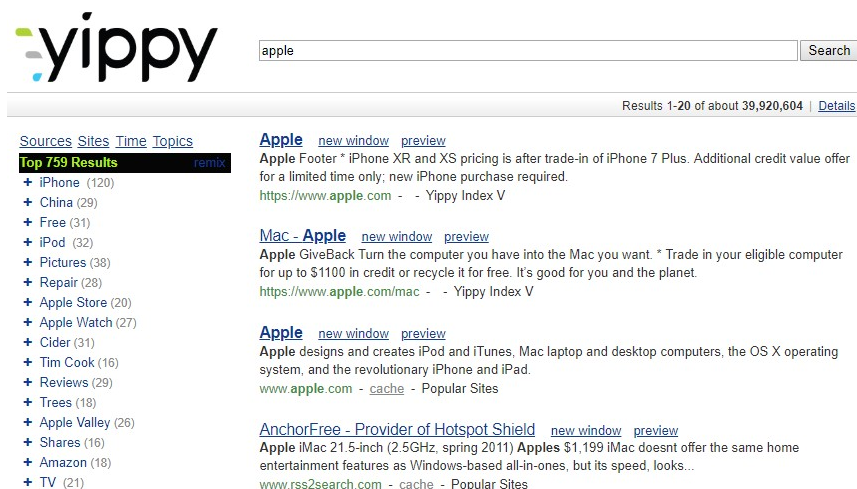
### *Recommender System*

*Recommender system* adalah suatu sistem yang berfungsi untuk memprediksikan keinginan pengguna berdasarkan masukan yang diberikan oleh pengguna [6]. Ada dua jenis sistem rekomendasi yaitu *Content-based Recommendation* dan *Collaborative Filtering*.

- *Content-based Recommendation*: mempelajari apa yang pengguna sukai lalu mencari objek lain yang mungkin juga disukai oleh pengguna tersebut berdasarkan apa yang disukainya. Pencarian ini dilakukan dengan mengusulkan objek-objek yang berada dalam kelompok (*cluster*) yang sama dengan objek yang disukai oleh pengguna. Hal ini dilakukan dengan asumsi pengguna akan menyukai barang yang mirip dengan barang yang disukai.
- *Collaborative Filtering*: memberikan usulan berdasarkan apa yang disukai pengguna yang serupa dengan seseorang dengan mengasumsikan jika pengguna serupa menyukai suatu objek, maka orang tersebut akan menyukai objek yang sama. Pengguna serupa didapatkan dengan mencari orang-orang yang berada dalam satu *cluster* dengan memperhitungkan atribut dari pengguna (usia, jenis kelamin, tempat domisili, hobi, dll).

### *Search Result Clustering*

Salah satu kegunaan pengelompokan dalam bidang penambangan data adalah untuk mengelompokkan hasil pencarian (*search result clustering*) [7]. Setiap kata kunci dalam sebuah pencarian mungkin dapat masuk ke dalam berbagai kategori. Misalkan kata kunci "apple" dapat berarti buah apel atau perusahaan teknologi *apple*. Dengan menggunakan pengelompokan hasil pencarian, maka hasil dari pencarian akan dimasukkan ke dalam kelompok-kelompok topik dan pengguna dapat memilih topik mana yang dimaksud untuk dapat mengeluarkan hasil yang lebih spesifik. Mesin pencari yang mengelompokkan hasil pencariannya dinamakan dengan *clustering search engine*. Salah satu contoh *clustering search engine* adalah Yippy<sup>1</sup>.



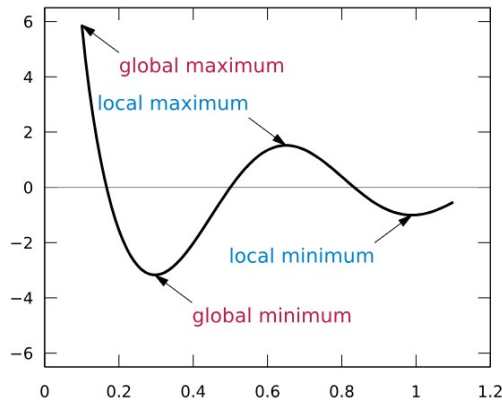
Gambar 2.2: Hasil pencarian (*clustered search result*) di Yippy

<sup>1</sup><https://yippy.com/>

Hasil pencarian di Yippy dengan kata kunci "apple" ditunjukkan dalam Gambar 2.2. Bagian sebelah kiri pada Gambar 2.2 merupakan kelompok-kelompok kategori dari kata kunci yang dimasukkan sehingga pengguna dapat memilih kategori yang sesuai dengan yang mereka maksud. "iPhone", "China", "Free", dan seterusnya merupakan kelompok yang dihasilkan apabila pengguna memasukkan kata kunci "apple".

### 2.1.3 Local Optimum

*Local optimum* adalah suatu solusi yang optimal (baik maksimal maupun minimal) diantara kandidat solusi yang berdekatan dalam masalah optimasi. Dikatakan lokal karena solusi ini hanya optimal apabila dibandingkan dengan kandidat solusi yang berdekatan, tidak optimal secara keseluruhan (*global optimum*). Contoh dari *local* dan *global* optimum ditunjukkan pada Gambar 2.3. *Local minimum* memiliki nilai yang paling kecil apabila dibandingkan dengan nilai-nilai lain yang berdekatan dengannya, begitu pula dengan *local maximum*. Sedangkan *global minimum* dan *global maximum* memiliki nilai yang paling minimum dan maksimum dalam keseluruhan himpunan kandidat solusi.



Gambar 2.3: *Local Optimum* dan *Global Optimum*

Suatu program optimasi dapat terjebak di *local optimum*. Sebagai contoh pada Gambar 2.3, apabila suatu program mencari solusi yang merupakan nilai maksimum maka program dapat terjebak pada nilai 2 yang merupakan *local maximum* karena seharusnya keluaran dari program tersebut adalah 6 yang merupakan *global maximum*. Sedangkan apabila suatu program mencari solusi berupa nilai minimum maka program dapat terjebak pada nilai -1 yang merupakan *local minimum* karena seharusnya keluaran dari program tersebut adalah -3 yang merupakan *global minimum*.

## 2.2 K-Means

K-means merupakan salah satu algoritma pengelompokan yang umum digunakan saat ini. Algoritma ini membagi objek ke dalam  $K$  cluster. Setiap cluster direpresentasikan dengan titik tengahnya (*centroid*). Titik tengah akan dihitung sebagai rata-rata dari semua titik objek dari cluster tersebut dalam setiap iterasinya. Persamaan 2.1 merupakan persamaan untuk menghitung *centroid*

$$\mu_i = \frac{1}{N_i} \sum_{q=1}^{N_i} x_q \quad (2.1)$$

dengan  $\mu_i$  merupakan *centroid* ke- $i$ ,  $N_i$  merupakan jumlah titik objek pada cluster ke- $i$ , dan  $x_q$  merupakan titik ke- $q$  pada cluster ke- $i$ .

---

**Algoritma 1** K-Means
 

---

**Input:**  $S$  (himpunan titik objek),  $K$  (Jumlah *cluster*)

**Output:** himpunan *cluster*

- 1: Pilih  $K$  titik objek sebagai himpunan awal *centroid*.
  - 2: **repeat**
  - 3:   Bentuk  $K$  *cluster* dengan menempatkan setiap titik objek ke *cluster* dengan *centroid* terdekat.
  - 4:   Hitung ulang *centroid* untuk setiap *cluster*.
  - 5: **until** *Centroid* tidak berubah.
- 

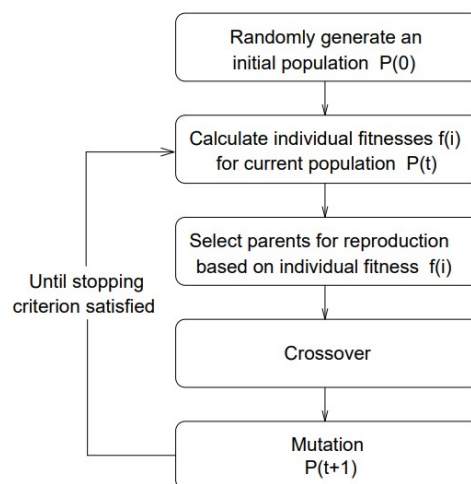
Penjelasan dari Algoritma 1 adalah sebagai berikut:

- Pada baris 4, *centroid* baru akan ditentukan dengan menggunakan Persamaan 2.1.
- Pada baris 5, pengulangan akan berhenti saat *centroid* mengalami pergeseran yang tidak terlalu signifikan (lebih kecil dari suatu nilai  $\epsilon$  yang merupakan masukan dari pengguna).

## 2.3 Algoritma Genetika

Algoritma genetika atau biasa disebut *Genetic Algorithm*(GA) adalah suatu algoritma pencarian yang terinspirasi dari proses seleksi alam yang terjadi secara alami dalam proses evolusi. Di alam, individu dalam suatu populasi berkompetisi satu sama lain untuk memperebutkan tempat tinggal, makanan, dll [4]. Bahkan setiap individu dalam spesies yang sama pun harus bersaing menarik lawan jenis untuk berkembang biak. Individu yang kurang baik akan memiliki peluang bertahan hidup lebih kecil, dan individu yang bisa beradaptasi dengan baik atau "*fit*" akan menghasilkan keturunan dengan jumlah yg relatif banyak.

GA merupakan metode penyelesaian masalah yang menggunakan genetika sebagai pemodelannya. Suatu calon solusi dalam GA dimodelkan sebagai suatu individu. Kumpulan individu-individu ini disebut dengan populasi. Setiap individu dalam populasi direpresentasikan dengan kromosom. Kromosom merupakan kumpulan parameter yang membentuk suatu solusi. Parameter-parameter yang menyusun kromosom disebut dengan gen. Setiap kromosom memiliki suatu nilai yang terkait dengan *fitness* dari solusi yang direpresentasikannya. Nilai itu biasanya disebut dengan nilai *fitness*.



Gambar 2.4: Alur algoritma genetika dasar



Secara umum, proses pada GA ditunjukkan dalam Gambar 2.4. Penjelasan dari proses-proses pada GA yang disebutkan dalam Gambar 2.4 adalah sebagai berikut:

#### 1. Inisialisasi Populasi

Inisialisasi populasi merupakan tahap paling awal dari GA. Pada proses ini, akan dibentuk populasi  $P(0)$  secara acak.

#### 2. Perhitungan *Fitness*

Menghitung nilai *fitness*  $f(i)$  dari setiap individu dalam populasi saat ini  $P(t)$ . Nilai *fitness* ini akan digunakan dalam operasi genetik selanjutnya.

#### 3. Seleksi

Proses seleksi akan terjadi berdasarkan nilai *fitness*  $f(i)$  setiap individu. Individu yang lebih "kuat" akan memiliki peluang terpilih lebih besar dalam proses seleksi.

#### 4. Persilangan

Individu yang terpilih pada proses seleksi akan disilangkan untuk menghasilkan keturunan. Proses persilangan ini terjadi antara dua induk hasil seleksi.

#### 5. Mutasi

Keturunan yang dihasilkan pada proses persilangan dapat memiliki peluang untuk mengalami mutasi. Mutasi dapat terjadi dengan suatu peluang terjadinya mutasi.

Proses 2 sampai 5 akan diulang terus-menerus sampai ditemukan suatu solusi yang optimal. Berikut merupakan penjelasan lebih lanjut mengenai istilah yang ada pada GA.

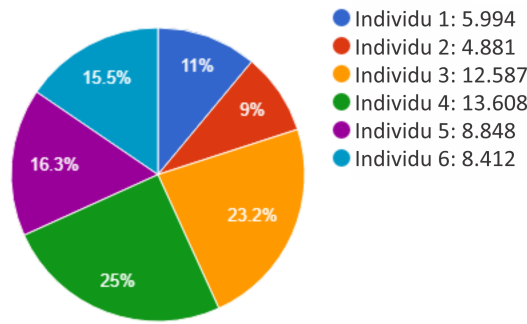
##### 2.3.1 *Fitness*

*Fitness* dari suatu individu dalam algoritma genetika adalah suatu nilai fungsi objektif untuk fenotipenya [4]. *Fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan solusi yang optimal. Suatu solusi yang optimal akan memaksimalkan fungsi *fitness*.

##### 2.3.2 Seleksi

Seleksi adalah proses pemilihan dua induk dari populasi untuk disilangkan [4]. Tujuan dari proses seleksi adalah untuk menonjolkan individu yang memiliki nilai *fitness* tinggi dalam populasi dengan harapan keturunannya akan memiliki *fitness* yang lebih tinggi. Seleksi adalah suatu metode yang memilih kromosom secara acak dari populasi berdasarkan fungsi *fitness*. Semakin tinggi fungsi *fitness* maka semakin tinggi peluang suatu individu akan terpilih.

Salah satu teknik yang populer digunakan dalam seleksi adalah *roulette-wheel selection* atau *fitness proportional selection*. *Roulette-wheel selection* memilih suatu individu dari populasi dengan probabilitas yang sebanding dengan nilai *fitness* relatifnya. Berdasarkan ilustrasi yang terdapat dalam Gambar 2.5, setiap individu memiliki sebuah bagian pada diagram sesuai dengan nilai *fitness* relatif. Semakin tinggi nilai *fitness*, maka semakin besar bagian yang dialokasikan dan semakin besar kemungkinan individu tersebut akan terpilih dalam proses seleksi.

Gambar 2.5: Ilustrasi *roulette-wheel*

Sebagai penjelasan dari ilustrasi pada Gambar 2.5 mengenai nilai *fitness* relatif adalah sebagai berikut:

- Ada enam individu dalam suatu populasi (Individu 1, Individu 2, dst).
- Setiap individu memiliki nilai *fitness* seperti yang tertera pada gambar (Individu 1 memiliki nilai *fitness* sebesar 5.994, Individu 2 memiliki nilai *fitness* sebesar 4.881, dst).
- Berdasarkan nilai *fitness* tersebut, maka tiap Individu dalam populasi memiliki peluang terpilih yang tercantum dalam persentase pada diagram lingkaran (Individu 1 memiliki peluang terpilih sebesar 11%, Individu 2 memiliki peluang terpilih sebesar 9%, dst). Peluang tersebut dapat dihitung menggunakan persamaan 2.2.

$$P_i = \frac{f_i}{\sum_{j \in Pop} f_j} \quad (2.2)$$

dengan  $P_i$  merupakan peluang terpilihnya individu  $i$ ,  $f_i$  merupakan nilai *fitness* dari individu  $i$ ,  $Pop$  merupakan populasi, dan  $f_j$  merupakan nilai *fitness* dari individu  $j$ .

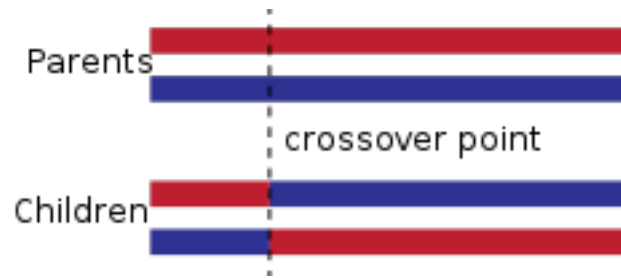
Proses seleksi dalam GA akan memilih sejumlah induk yang cukup untuk reproduksi dan membentuk generasi selanjutnya. Untuk meningkatkan performa GA, dapat juga diterapkan strategi *elitism* yaitu dengan langsung memindahkan satu atau beberapa individu dengan nilai *fitness* terbaik ke generasi selanjutnya. Sisanya akan dilakukan dengan cara yang sudah ditentukan sebelumnya seperti *roulette-wheel selection*. Hal ini dilakukan untuk mencegah individu tersebut hilang dari populasi dalam proses reproduksi.

### 2.3.3 Persilangan

Persilangan adalah operasi genetik yang digunakan untuk menggabungkan informasi genetik dari dua induk untuk menghasilkan keturunan baru [4]. Persilangan dilakukan untuk menghasilkan suatu individu baru yang diharapkan memiliki *fitness* yang lebih baik daripada orangtuanya. Salah satu teknik yang dapat digunakan dalam persilangan adalah *Single-point crossover*. Sebuah titik pada kedua induk dipilih untuk menjadi titik persilangan (*crossover point*). Gen yang berada di sebelah kanan titik persilangan bertukar antara kedua kromosom induk seperti yang ditunjukkan pada Gambar 2.6.

### 2.3.4 Mutasi

Mutasi adalah suatu operator genetik yang digunakan untuk mempertahankan keragaman genetik dari satu generasi populasi dalam algoritma genetika. Oleh karena itu, mutasi juga dapat mencegah

Gambar 2.6: *Single-point crossover*

- 1 GA terjebak di *local optimum* [4]. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi  
2 terjadi berdasarkan probabilitas mutasi  $\mu_m$  yang sudah ditentukan sebelumnya. Probabilitas mutasi  
3 menentukan seberapa sering kromosom akan dimutasi.  
4 Jika tidak terjadi mutasi, maka keturunannya akan langsung masuk ke populasi setelah per-  
5 silangan tanpa perubahan. Apabila terjadi mutasi, satu atau beberapa bagian dari kromosom  
6 akan diubah. Mutasi seharusnya tidak dilakukan terlalu sering, karena jika terlalu sering dilakukan  
7 maka GA akan menjadi sama dengan algoritma pencarian acak primitif (*primitive random search*).  
8 Mutasi akan dilakukan dengan mengubah nilai dari suatu gen yang telah dipilih menjadi nilai  
9 lainnya dengan teknik tertentu (tergantung struktur data dari gen yang akan diubah).

### 10 2.3.5 Proses Pencarian Dalam Algoritma Genetika

- 11 Proses pencarian dalam GA secara umum dijelaskan dalam Algoritma 2. Proses pencarian ini  
12 memanfaatkan operasi genetik yang telah dijelaskan sebelumnya (Subbab 2.3.1 sampai dengan  
13 Subbab 2.3.4). Selain itu, terdapat algoritma lain yang menjelaskan masing-masing operasi genetik  
14 yang digunakan dalam Algoritma 2. Operasi seleksi yang telah dibahas pada Subbab 2.3.2 dijelaskan  
15 pada Algoritma 4. Operasi persilangan yang telah dibahas pada Subbab 2.3.3 dijelaskan pada  
16 Algoritma 3. Operasi mutasi yang telah dibahas pada Subbab 2.3.4 dijelaskan pada Algoritma 5.

---

**Algoritma 2** Algoritma Genetika [9]

---

**function** Algoritma-Genetika(*populasi*) **returns** solusi berupa individu

**input:** *populasi*, himpunan individu

```

1: solusi  $\leftarrow$  array yang menyimpan solusi tiap generasi
2: repeat
3:   tambahkan individu dengan nilai fitness tertinggi dari populasi ke solusi
4:   populasi_baru  $\leftarrow$  himpunan kosong
5:   for i=1 to Size(populasi) do
6:     x  $\leftarrow$  Seleksi-acak(populasi)
7:     y  $\leftarrow$  Seleksi-acak(populasi)
8:     anak  $\leftarrow$  Persilangan(x, y)
9:     rand  $\leftarrow$  Random(0,1)
10:    if rand  $\leq$  probab_mutasi then
11:      anak  $\leftarrow$  Mutasi(anak)
12:    end if
13:    tambahkan anak ke populasi_baru
14:  end for
15:  populasi  $\leftarrow$  populasi_baru
16: until N solusi terakhir pada solusi tidak memiliki perubahan yang signifikan
17: return individu terbaik dalam populasi, berdasarkan nilai fitness

```

---

Penjelasan untuk fungsi Algoritma-Genetika pada Algoritma 2 adalah sebagai berikut:

- Pada baris 1, variabel *solusi* berfungsi untuk mencatat sejarah dari solusi yang pernah dihasilkan pada setiap generasi.
- Pada baris 3, solusi pada generasi saat ini akan dicatat ke variabel *solusi*.
- Pada baris 9, variabel *rand* akan berisi bilangan riil antara 0 sampai dengan 1. Bilangan riil ini akan dibangkitkan secara acak dengan distribusi *uniform*.
- Pada baris 16, proses pengulangan akan berhenti saat nilai *fitness* pada *N* solusi terakhir sudah mengalami konvergensi.
- Pada baris 17, fungsi mengembalikan individu terbaik dalam populasi terakhir GA.

---

**Algoritma 3** Persilangan Algoritma Genetika

---

**function** Persilangan(*x*,*y*) **returns** anak berupa individu

**inputs:** *x* dan *y*, individu induk

```

1: n  $\leftarrow$  Length(x)
2: c  $\leftarrow$  Random(1,N)
3: return Append(Substring(x,1,c), Substring(y,c+1,n))

```

---

Penjelasan untuk fungsi Persilangan pada Algoritma 3 adalah sebagai berikut:

- Pada baris 2, variabel *c* akan berisi bilangan bulat antara 1 sampai *N*. Bilangan bulat ini akan dibangkitkan secara acak dengan distribusi *uniform*.

- 1 • Pada baris 3, *Append* merupakan fungsi untuk menggabungkan dua *string* dan *Substring*
- 2 merupakan fungsi untuk memotong *string* mulai dari batas tertentu sampai dengan batas
- 3 tertentu juga.

---

**Algoritma 4** Seleksi Algoritma Genetika

---

**function** Seleksi-acak(*populasi*) **returns** sebuah individu hasil seleksi

**input:** *populasi*, populasi saat ini

```

1: sum  $\leftarrow$  0
2: for all individu  $\in$  populasi do
3:   sum  $\leftarrow$  sum + Fitness(individu)
4: end for
5: terpilih  $\leftarrow$  Random(0,1)  $\times$  sum
6: for all individu  $\in$  populasi do
7:   terpilih  $\leftarrow$  terpilih - Fitness(individu)
8:   if terpilih  $\leq$  0 then
9:     return individu
10:  end if
11: end for
12: return individu dengan urutan terakhir di populasi

```

---

- 4 Penjelasan untuk fungsi Seleksi-acak pada Algoritma 4 adalah sebagai berikut:
- 5 • Pada baris 3, fungsi *Fitness* akan mengembalikan nilai *fitness* dari individu yang menjadi
  - 6 parameternya.
  - 7 • Pada baris 5, variabel *terpilih* akan berisi suatu bilangan 0 sampai *sum*. Hal ini dilakukan
  - 8 dengan cara mengalikan sebuah bilangan riil dengan *sum*. Bilangan riil tersebut merupakan
  - 9 bilangan antara 0 sampai 1 yang dibangkitkan secara acak dengan distribusi *uniform*.

---

**Algoritma 5** Mutasi Algoritma Genetika

---

**function** Mutasi(*individu*) **returns** individu hasil mutasi

**input:** *individu*, individu yang akan dilakukan mutasi

```

1: n  $\leftarrow$  Length(x)
2: c  $\leftarrow$  Random(1,n)
3: ubah nilai gen ke-c pada individu {nilai bervariasi tergantung metode}
4: return individu

```

---

- 10 Penjelasan untuk fungsi Mutasi pada Algoritma 5 adalah sebagai berikut:
- 11 • Pada baris 1, fungsi *Length* akan mengembalikan panjang dari *string* parameternya.
  - 12 • Pada baris 2, fungsi *Random* akan mengembalikan bilangan bulat acak antara 1 sampai *N*.

### 2.3.6 GA dalam Pengelompokan

- 14 Algoritma yang umum diterapkan untuk pengelompokan adalah *K-means*. Namun, algoritma
- 15 *K-means* masih memiliki kekurangan. Salah satu kekurangannya adalah masih dapat terjebak

1 pada *local optimum*. *Local optimum* dapat diatasi oleh GA yang sudah terbukti efektif dalam  
 2 masalah pencarian dan optimasi [8]. Oleh karena itu, diharapkan pengelompokan berbasis GA  
 3 dapat menghasilkan solusi yang lebih baik dibandingkan dengan algoritma *K-means*.

## 4 2.4 Model Ruang Vektor

5 Model ruang vektor adalah representasi dari koleksi dokumen sebagai vektor dalam ruang vektor  
 6 yang umum [10]. Model ruang vektor ini biasanya digunakan dalam sejumlah operasi pencarian  
 7 informasi mulai dari penilaian dokumen pada *query*, klasifikasi dokumen, dan pengelompokan  
 8 dokumen. Pada penerapannya, akan dilakukan pengukuran kemiripan suatu dokumen terhadap  
 9 *query* untuk dapat menentukan peringkat relevansi dokumen terhadap *query* (*relevance ranking*).  
 10 Dokumen dan *query* akan direpresentasikan sebagai model ruang vektor seperti pada Persamaan  
 11 2.3.

$$d_i = (w_{1,i}, w_{2,i}, \dots, w_{n,i}) \quad (2.3)$$

12 dengan  $d_i$  merupakan dokumen ke- $i$ ,  $w_{n,i}$  merupakan bobot dari *term*  $n$  untuk dokumen  $i$ .  
 13 Setiap dimensi pada vektor tersebut menggambarkan *term* berbeda dalam dokumen. Kemiripan  
 14 antara dokumen dan *query* akan ditentukan dengan mengukur perbedaan sudut antara vektor  
 15 dokumen dan vektor *query*. Semakin kecil sudut antara dokumen dan *query*, maka dokumen dan  
 16 *query* dianggap semakin mirip. Namun pada praktiknya, dilakukan perhitungan jarak cosinus  
 17 untuk menggantikan pengukuran sudut antara dua vektor karena jarak cosinus berbanding terbalik  
 18 dengan besar sudut antara dua vektor sehingga tidak perlu dilakukan perhitungan lebih lanjut  
 19 untuk mendapatkan besar sudutnya. Jarak cosinus dapat dihitung menggunakan Persamaan 2.4.

$$s_{ij} = \frac{i \cdot j}{\|i\| \times \|j\|} \quad (2.4)$$

20 dengan  $s_{ij}$  adalah kesamaan antara vektor ke- $i$  dengan vektor ke- $j$ ,  $i$  adalah vektor ke- $i$ , dan  $j$   
 21 adalah vektor ke- $j$ . Persamaan ini menjelaskan bahwa semakin kecil sudut antara dua vektor, maka  
 22 tingkat kemiripannya semakin besar.

## 23 2.5 Pembobotan *Term* (*Term Weighting*)

24 Suatu dokumen teks terdiri dari deretan karakter. Sebelum suatu dokumen teks dapat diolah  
 25 informasinya, maka dokumen tersebut perlu melalui suatu proses yang disebut dengan tokenisasi.  
 26 Menurut [10], tokenisasi merupakan proses pemotongan suatu dokumen menjadi potongan-potongan  
 27 (*token*) tertentu. Pada proses yang sama, karakter tertentu akan turut dibuang (tanda baca, spasi,  
 28 dll). Token adalah urutan karakter dalam dokumen tertentu yang dikelompokkan bersama sebagai  
 29 unit semantik. Selain token, terdapat juga istilah yang bernama *type* dan *term*. *Type* adalah kelas  
 30 dari semua token yang berisi urutan karakter yang sama. *Term* adalah *type* (mungkin dinormalisasi)  
 31 yang terdapat pada suatu sistem.

32 Pembobotan *term* merupakan suatu proses menentukan nilai dari suatu *term* dalam sebuah  
 33 dokumen. Pembobotan *term* bertugas untuk memetakan *term* kepada suatu nilai numerik yang  
 34 merepresentasikan seberapa penting *term* tersebut dalam suatu dokumen. Tidak semua *term* dalam  
 35 dokumen itu penting sehingga dengan memberikan nilai kepada masing-masing *term* dapat dengan  
 36 lebih tepat merepresentasikan isi dokumen. Secara umum, apabila suatu *term* semakin penting  
 37 dalam suatu dokumen (semakin menggambarkan isi dokumen), maka nilai bobotnya akan semakin  
 38 besar. Sebaliknya jika suatu *term* semakin tidak penting (kata-kata yang umum digunakan seperti  
 39 kata sambung, kata ganti, dan lain-lain), maka nilai bobotnya akan semakin kecil.

40 Ada beberapa cara untuk menghitung bobot suatu *term*. Dua metode yang umum digunakan  
 41 diantaranya adalah bobot frekuensi (*Frequency weighting*) dan bobot TF-IDF (*TF-IDF weighting*).

- 1 Bobot TF-IDF merupakan pengembangan dari bobot frekuensi dengan memperhitungkan kemun-
- 2 culan suatu *term* secara global.

### 3 2.5.1 Bobot frekuensi

Bobot frekuensi merupakan teknik pembobotan yang sangat sederhana karena bobotnya merupakan jumlah kemunculan *term* tersebut dalam dokumen. *Term* yang sering muncul pada suatu dokumen akan dianggap berkaitan dengan dokumen tersebut. Misalkan suatu dokumen banyak memuat *term* "properti", maka dokumen tersebut dianggap merupakan suatu dokumen yang membahas masalah "properti". Bobot frekuensi dapat digambarkan dengan Persamaan 2.5

$$w_i = tf_i \quad (2.5)$$

- 4 dengan  $w_i$  merupakan bobot *term* ke- $i$  dan  $tf_i$  merupakan frekuensi kemunculan *term* ke- $i$  pada
- 5 dokumen. Sebagai ilustrasi, akan digunakan empat buah dokumen (masing-masing terdiri dari satu
- 6 kalimat) yang berasal dari contoh pada [10] sebagai berikut:

- 7 • Doc 1: new home sales top forecasts
- 8 • Doc 2: home sales rise in july
- 9 • Doc 3: increase in home sales in july
- 10 • Doc 4: july new home sales rise

- 11 Berdasarkan keempat contoh dokumen, maka dibentuk tabel ketetanggaan antara *term* dengan
- 12 dokumen pada Tabel 2.1.

<i>Term</i>	d1	d2	d3	d4
new	1	0	0	1
home	1	1	1	1
sales	1	1	1	1
top	1	0	0	0
forecast	1	0	0	0
rise	0	1	0	1
in	0	1	2	0
july	0	1	1	1
increase	0	0	1	0

Tabel 2.1: *Term-document incidence matrix*

- 13 *Term* "in" dalam tabel pada d1 dan d4 bernilai 0 karena *term* "in" tidak muncul pada d1 dan
- 14 d4. Sedangkan pada d2, *term* "in" muncul satu kali sehingga bernilai 1 pada tabel. Begitu juga
- 15 pada d3, *term* "in" muncul dua kali sehingga bernilai 2 pada tabel.

### 16 2.5.2 Bobot TF-IDF

- 17 Selain menggunakan bobot frekuensi, ada teknik pembobotan lain yang disebut dengan TF-IDF
- 18 (*Term Frequency-Inverse Document Frequency*). Teknik pembobotan ini merupakan pengembangan
- 19 dari pembobotan frekuensi. TF-IDF merupakan gabungan dari *term frequency* (tf) dengan *inverse*
- 20 *document frequency* (idf) untuk menghasilkan suatu bobot komposit untuk setiap *term* dalam setiap
- 21 dokumen [10].

- 22 *Term frequency* atau biasa dilambangkan sebagai  $tf_{t,d}$  untuk *term*  $t$  pada dokumen  $d$ . Sama
- 23 seperti yang telah dijelaskan pada Subbab 2.5.1, TF merupakan banyaknya kemunculan *term* pada

1 suatu dokumen. Namun pada TF-IDF, umumnya digunakan bobot frekuensi yang telah dinormalisasi  
 2 dengan jumlah kemunculan semua term pada suatu dokumen. TF yang telah dinormalisasi dapat  
 3 dihitung menggunakan persamaan 2.6.

$$tf_{t,d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \quad (2.6)$$

4 dengan  $tf_{t,d}$  merupakan TF *term*  $t$  pada dokumen  $d$ ,  $f_{t,d}$  merupakan banyaknya kemunculan  
 5 *term*  $t$  dalam dokumen  $d$ ,  $t' \in d$  merupakan seluruh *term*  $t'$  yang merupakan anggota dari dokumen  
 6  $d$ , dan  $f_{t',d}$  merupakan banyaknya kemunculan *term*  $t'$  pada dokumen  $d$ .

7 Terdapat suatu masalah pada pengukuran menggunakan TF yaitu semua *term* dianggap sama  
 8 penting. Pada kenyataannya, tidak semua *term* itu sama penting karena *term* yang sangat umum  
 9 akan memiliki frekuensi yang sangat besar pada banyak dokumen. Seperti yang sudah dijelaskan  
 10 pada Subbab 2.5.1, suatu *term* yang sering muncul akan dianggap berkaitan dengan suatu dokumen.  
 11 Namun hal itu ternyata belum tentu benar karena banyak *term* yang sering muncul padahal *term*  
 12 tersebut sebenarnya tidak memiliki nilai informasi seperti *term* yang merupakan kata penghubung.  
 13 Oleh karena itu, diperlukan suatu mekanisme untuk mengurangi efek dari suatu *term* yang sering  
 14 muncul di banyak dokumen. Salah satu cara yang digunakan untuk menangani hal tersebut  
 15 adalah dengan menggunakan *inverse document frequency* (IDF). IDF ini dapat dihitung dengan  
 16 menggunakan persamaan 2.7.

$$idf_t = \log \frac{N}{df_t} \quad (2.7)$$

17 dengan  $idf_t$  merupakan idf dari *term*  $t$ ,  $N$  merupakan banyaknya anggota himpunan dokumen,  
 18 dan  $df_t$  merupakan *document frequency* dari *term*  $t$ . *Document frequency* adalah banyaknya dokumen  
 19 pada himpunan dokumen yang memuat *term*  $t$ .

20 Bobot TF-IDF menggabungkan teknik pembobotan TF dengan IDF. Nilai dari TF digabungkan  
 21 dengan nilai dari IDF dengan cara mengalikan keduanya. Metode TF-IDF ini sangat populer  
 22 digunakan oleh sistem rekomendasi berbasis teks [11]. Pembobotan menggunakan TF-IDF dapat  
 23 dihitung menggunakan Persamaan 2.8.

$$tf-idf_{t,d} = tf_{t,d} \times idf_t \quad (2.8)$$

24 Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

- 25 • Semakin sering suatu *term* muncul di suatu dokumen, maka semakin representatif *term*  
 26 tersebut terhadap isi dokumen.
- 27 • Semakin banyak dokumen yang memuat suatu *term*, maka nilai informasi *term* tersebut  
 28 semakin kecil.

29 Metode penetapan bobot TF-IDF dianggap sebagai metode yang berkinerja baik karena mem-  
 30 pertimbangkan frekuensi kemunculan *term* baik secara lokal (TF) maupun global (IDF).

## 31 2.6 Metrik *Intracuster* untuk Mengukur Kinerja Metode *Clus-* 32 *tering*

33 Untuk mengukur performa dari suatu algoritma, perlu dilakukan pengujian. Pengujian ini dapat  
 34 menentukan apakah algoritma yang digunakan sudah cukup baik dalam menyelesaikan masalah  
 35 yang dibuat. Dalam pengelompokan, ada beberapa cara untuk mengukur apakah objek-objek sudah  
 36 berhasil dikelompokkan secara baik atau tidak. Cara yang pertama adalah dengan mengukur jarak  
 37 antara tiap objek ke titik pusat *cluster* (*centroid*) atau biasa disebut jarak *intracuster*. Lalu cara  
 38 yang kedua adalah mengukur jarak antar kelompok yang dapat diukur dengan cara menghitung  
 39 jarak setiap *centroid* ke *centroid* lainnya.

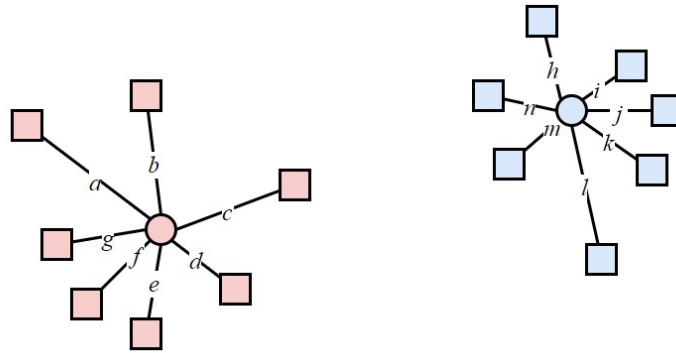


1 Metrik yang digunakan [2] dalam mengukur kinerja suatu metode pengelompokan adalah  
 2 perhitungan jarak *intracuster*. Jarak *intracuster* dapat diukur dengan cara menjumlahkan jarak  
 3 setiap objek ke masing-masing titik pusatnya. Cara untuk menghitung jarak *intracuster* ditunjukkan  
 4 dalam Persamaan 2.9.

$$M = \sum_{i=1}^K M_i, \quad (2.9)$$

$$M_i = \sum_{x_j \in C_i} \|x_j - z_i\|$$

5 dengan  $M$  merupakan jumlah jarak seluruh objek ke *centroid* masing-masing,  $K$  merupakan  
 6 banyaknya *cluster*,  $M_i$  merupakan jumlah jarak seluruh objek anggota *cluster* ke- $i$  ke titik pusatnya,  
 7  $x_j$  merupakan objek ke- $j$ ,  $C_i$  merupakan *centroid* ke- $i$ , dan  $z_i$  merupakan *centroid* dari *cluster*  
 8 ke- $i$ . Untuk memperjelas perhitungan, Gambar 2.7 akan digunakan sebagai ilustrasi dari jarak  
 9 *intracuster*.



Gambar 2.7: Ilustrasi untuk jarak *intracuster*

10 Pada Gambar 2.7, terdapat 14 objek yang telah dibagi ke dalam dua *cluster* berwarna merah  
 11 dan biru. Persegi pada Gambar 2.7 mengilustrasikan objek yang akan dikelompokkan. Lingkaran  
 12 mengilustrasikan titik pusat *cluster* (*centroid*). Garis yang menghubungkan objek dan *centroid*  
 13 menggambarkan jarak antara objek dengan *centroid* dan ditandai dengan label antara huruf  $a$   
 14 sampai dengan  $n$ . Persegi berwarna merah merupakan anggota dari *cluster* 1, sedangkan persegi  
 15 berwarna biru merupakan anggota dari *cluster* 2. Berdasarkan Persamaan 2.9, jarak *intracuster*  
 16 dari *cluster* 1 ( $M_1$ ) didapat dengan menjumlahkan  $a, b, c, d, e, f$ , dan  $g$ . Sedangkan, jarak *intracuster*  
 17 dari *cluster* 2 ( $M_2$ ) didapat dengan menjumlahkan  $h, i, j, k, l, m$ , dan  $n$ . Jarak *intracuster* total  
 18 didapatkan dengan menjumlahkan jarak *intracuster* milik *cluster* 1 dan *cluster* 2 ( $a + b + \dots + n$ ).

19 Semakin kecil nilai  $M$ , maka pengelompokan dianggap semakin baik. Ini berarti setiap objek  
 20 dalam suatu *cluster* mirip satu sama lain. Metrik ini dapat mengukur seberapa baik objek sudah  
 21 dikelompokkan dengan mempertimbangkan kedekatan setiap objek ke titik pusatnya.



## BAB 3

### ANALISIS

Bab ini membahas hasil analisis berdasarkan dasar teori yang sudah dijelaskan sebelumnya. Pada bab ini akan dijelaskan hasil analisis *dataset* yang akan digunakan dalam pengujian, representasi dokumen dalam perangkat lunak, dan pemodelan ruang vektor pada dokumen. Selain itu pada bab ini juga akan dibahas mengenai representasi kromosom, fungsi *fitness*, dan beberapa operasi genetik yang akan digunakan dalam membuat pengelompokan dokumen berbasis algoritma genetika.

#### 3.1 Analisis *Dataset*

Pada bagian ini akan dibahas mengenai *dataset* yang akan digunakan dalam proses pengujian. *Dataset* yang akan digunakan berisi artikel berita *BBC News* dan disediakan untuk menjadi tolok ukur dalam penelitian pembelajaran mesin. Karakteristik dari *dataset* ini antara lain:

- Terdiri dari 2225 dokumen yang berasal dari *website BBC News* dari tahun 2004-2005.
- Dokumen ditulis dalam Bahasa Inggris.
- Terbagi menjadi lima topik yaitu *business*, *entertainment*, *politics*, *sport*, dan *tech*.
- Pada topik *business* terdapat 510 dokumen, *entertainment* terdapat 386 dokumen, *politics* terdapat 417 dokumen, *sport* terdapat 511 dokumen, dan *tech* terdapat 401 dokumen.
- Dokumen merupakan *plain text* yang ditulis dalam file dengan ekstensi *TXT*.
- Rata-rata dalam satu dokumen terdapat 384 kata.

#### 3.2 Representasi Dokumen

Dokumen tidak bisa langsung digunakan begitu saja dalam proses pengelompokan. Tidak seperti manusia yang dapat melakukan proses secara manual, komputer tidak dapat menemukan nilai informasi dari data mentah berupa dokumen. Dokumen yang ada perlu direpresentasikan menjadi bentuk yang bisa diambil informasinya baru kemudian dapat diolah dalam proses lebih lanjut. Model ruang vektor (Subbab 2.4) digunakan dalam penelitian ini untuk merepresentasikan dokumen sehingga informasinya dapat diproses dengan lebih mudah.

#### 3.3 Model Ruang Vektor

Pada subbab 2.4 telah dijelaskan bahwa dokumen akan dibentuk ke dalam sebuah vektor yang memiliki banyak dimensi berdasarkan banyaknya *term* berbeda dalam dokumen. Seperti yang telah dibahas pada subbab 2.5, ada dua cara untuk menentukan bobot dari suatu *term* dalam dokumen yaitu bobot frekuensi dan bobot tf-idf. Sebagai contoh akan digunakan tiga dokumen berikut:

1. Penjualan properti di Indonesia meningkat di Bulan Februari.

2. Penjualan asuransi kendaraan di Indonesia meningkat.

3. Bulan Februari merupakan bulan puncak penjualan kendaraan.

Berdasarkan tiga dokumen tersebut akan ditentukan bobot dari masing-masing *term* dalam tiap dokumen untuk setiap jenis pembobotan.

### 3.3.1 Bobot Frekuensi

Pada subbab 2.5.1 telah dijelaskan bahwa bobot frekuensi dari suatu *term* dapat ditentukan dengan cara menghitung banyaknya kemunculan *term* tersebut dalam dokumen. Hasil perhitungan bobot frekuensi berdasarkan contoh pada Subbab 3.3 ditunjukkan dalam Tabel 3.1.

<i>Term</i>	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	1	1	1
properti	1	0	0
di	2	1	0
indonesia	1	1	0
meningkat	1	1	0
bulan	1	0	2
februari	1	0	1
asuransi	0	1	0
kendaraan	0	1	1
merupakan	0	0	1
puncak	0	0	1

Tabel 3.1: Hasil perhitungan bobot frekuensi

Semakin besar bobot, maka *term* dianggap semakin mewakili isi dokumen. Sebagai contoh pada Tabel 3.1, *term* "di" muncul dua kali pada dokumen 1 sehingga berbobot 2, muncul satu kali pada dokumen 2 sehingga berbobot 1, dan tidak muncul sama sekali pada dokumen 3 sehingga memiliki bobot 0 sehingga *term* "di" dianggap mewakili dokumen 1 karena muncul dua kali pada dokumen tersebut.

### 3.3.2 Bobot TF-IDF

Berbeda dengan bobot frekuensi yang hanya menghitung frekuensi kemunculan *term* pada dokumen, perhitungan bobot TF-IDF ini memerlukan perhitungan seperti yang telah dijelaskan dalam Persamaan 2.8 pada Subbab 2.5.2. Sama dengan perhitungan bobot frekuensi, contoh yang berasal dari Subbab 3.3 akan digunakan sebagai ilustrasi perhitungan TF-IDF. Sesuai dengan apa yang telah dijelaskan pada Subbab 2.5.2, perhitungan dari TF-IDF akan dibagi menjadi dua tahap yaitu perhitungan TF dan perhitungan IDF.

*Term* "properti" akan digunakan dalam contoh perhitungan TF. Berdasarkan Persamaan 2.6 pada Subbab 2.5.2, maka perhitungan TF dari *term* "properti" ditunjukkan pada Persamaan 3.1.

$$\begin{aligned}
 \text{tf}_{t,d} &= \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \\
 \text{tf}_{\text{"properti"},1} &= \frac{1}{8} = 0.125
 \end{aligned}
 \tag{3.1}$$

Penjelasan dari Persamaan 3.1 adalah sebagai berikut.  $f_{\text{"properti"},1}$  bernilai 1 karena *term* "properti" hanya muncul 1 kali pada dokumen 1. Dokumen 1 terdiri dari 8 *term* sehingga  $\text{tf}_{\text{"properti"},1}$

- 1 bernilai 0.125. Berdasarkan Persamaan 3.1, maka hasil perhitungan TF dari ketiga dokumen  
 2 tersebut ditunjukkan pada Tabel 3.2.

<i>Term</i>	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	0.1250	0.1667	0.1429
properti	0.1250	0	0
di	0.2500	0.1667	0
indonesia	0.1250	0.1667	0
meningkat	0.1250	0.1667	0
bulan	0.1250	0	0.2857
februari	0.1250	0	0.1429
asuransi	0	0.1667	0
kendaraan	0	0.1667	0.1429
merupakan	0	0	0.1429
puncak	0	0	0.1429

Tabel 3.2: Hasil perhitungan TF

- 3 Untuk perhitungan IDF akan digunakan Persamaan 2.7 pada Subbab 2.5.2. Perhitungan IDF  
 4 untuk *term* "properti" ditunjukkan oleh Persamaan 3.2.

$$\text{idf}_t = \log \frac{N}{\text{df}_t}$$

$$\text{idf}_{\text{"properti"}} = \log \frac{3}{1} = 0.4771 \quad (3.2)$$

- 5 Penjelasan dari Persamaan 3.2 adalah sebagai berikut.  $N$  bernilai 3 karena banyaknya dokumen  
 6 dalam seluruh koleksi dokumen adalah 3.  $\text{df}_t$  bernilai 1 karena hanya ada 1 dokumen yang memuat  
 7 *term* "properti" yaitu dokumen 1. Hasil dari perhitungan IDF untuk setiap *term* ditunjukkan oleh  
 8 Tabel 3.3.

<i>Term</i>	IDF
penjualan	0
properti	0.4771
di	0.1761
indonesia	0.1761
meningkat	0.1761
bulan	0.1761
februari	0.1761
asuransi	0.4771
kendaraan	0.1761
merupakan	0.4771
puncak	0.4771

Tabel 3.3: Hasil perhitungan IDF

- 9 *Term* penjualan pada Tabel 3.3 bernilai 0. *Term* "penjualan" muncul di ketiga dokumen sehingga  
 10 pada saat perhitungan IDF menghasilkan nilai nol ( $\log \frac{N}{N_i} = \log \frac{3}{3} = 0$ ). Dapat disimpulkan bahwa  
 11 *term* "penjualan" tidak mewakili dokumen manapun karena muncul di semua dokumen.

- 12 Hasil dari perhitungan TF dan IDF digabung dengan cara dikalikan. Perhitungan TF-IDF

1 untuk *term* "properti" pada dokumen 1 ditunjukkan dalam Persamaan 3.3.

$$\begin{aligned} \text{tf-idf}_{t,d} &= \text{tf}_{t,d} \times \text{idf}_t \\ \text{tf-idf}_{\text{"properti"},1} &= 0.125 \times 0.4771 = 0.0075 \end{aligned} \quad (3.3)$$

2 Nilai yang berasal dari Persamaan 3.1 dan Persamaan 3.2 dikalikan sehingga didapatkan hasil  
3 sesuai dengan Persamaan 3.3. Hasil perhitungan TF-IDF untuk seluruh *term* dalam ketiga dokumen  
4 ditunjukkan dalam Tabel 3.4.

Term	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	0	0	0
properti	0.0075	0	0
di	0.0055	0.0049	0
indonesia	0.0028	0.0049	0
meningkat	0.0028	0.0049	0
bulan	0.0028	0	0.0072
februari	0.0028	0	0.0036
asuransi	0	0.0133	0
kendaraan	0	0.0049	0.0036
merupakan	0	0	0.0097
puncak	0	0	0.0097

Tabel 3.4: Hasil perhitungan bobot TF-IDF

### 5 3.4 Representasi Kromosom

6 Dalam penelitian ini, kromosom tersusun dari gen-gen yang merupakan *centroid* dari  $K$  buah *cluster*  
7 yang akan dibentuk. *Centroid* disimpan dalam bentuk vektor yang terdiri dari  $N$  buah bilangan  
8 riil.  $N$  merupakan dimensi dari vektor tersebut yang merupakan banyaknya *term* berbeda yang  
9 terdapat pada seluruh koleksi dokumen. Oleh karena itu, setiap kromosom akan memiliki  $N \times K$   
10 buah gen. Secara umum representasi *centroid* ke dalam kromosom ditunjukkan pada Gambar 3.1.

$$\begin{array}{ccc} C_1 & C_2 & C_k \\ w_{1,1}, w_{1,2}, \dots, w_{1,n}, & w_{2,1}, w_{2,2}, \dots, w_{2,n} & \dots \quad w_{k,1}, w_{k,2}, \dots, w_{k,n} \end{array}$$

Gambar 3.1: Formula representasi kromosom

11 Berdasarkan Gambar 3.1,  $N$  gen pertama merepresentasikan  $N$  dimensi dari *centroid* pertama  
12  $C_1$  ( $w_{1,1}, w_{1,2}, \dots, w_{1,n}$ ),  $N$  gen selanjutnya merepresentasikan  $N$  dimensi dari *centroid* kedua  $C_2$   
13 ( $w_{2,1}, w_{2,2}, \dots, w_{2,n}$ ), dan seterusnya sampai *centroid*  $C_k$  ( $w_{k,1}, w_{k,2}, \dots, w_{k,n}$ ). Sebagai contoh apabila  
14 diketahui ada tiga buah *centroid* dalam bidang dua dimensi **C1** (3.05, 1.43), **C2** (15.85, 14.23), dan  
15 **C3** (5.12, 9.45). Hasil representasi ketiga *centroid* pada kromosom ditunjukkan oleh Gambar 3.2.

C1		C2		C3	
3.05	1.43	15.85	14.23	5.12	9.45

Gambar 3.2: Contoh representasi *centroid* ke dalam kromosom

### 3.5 Fungsi *Fitness*

Seperti yang telah dijelaskan pada Subbab 3.4, kromosom akan tersusun atas *centroid* yang berbentuk vektor. Oleh karena itu, perhitungan *fitness* akan mengalami beberapa penyesuaian. Perhitungan *fitness* dalam penelitian ini terdiri dari tiga tahap. Pada tahap pertama, terjadi pembentukan *cluster* berdasarkan titik pusat yang terkandung dalam kromosom. Hal ini dilakukan dengan menetapkan setiap dokumen  $x_i, i = 1, 2, \dots, n$  ke dalam sebuah *cluster*  $C_j$  dengan *centroid*  $z_j$  sehingga memenuhi Persamaan 3.4.

$$\|x_i - z_j\| < \|x_i - z_p\|, p = 1, 2, \dots, K, \text{ dan } p \neq j. \quad (3.4)$$

Persamaan 3.4 menjelaskan bahwa akan dicari suatu *cluster*  $C_j$  yang paling dekat dengan dokumen  $x_i$ . Hal ini dilakukan dengan cara membandingkan jarak antara dokumen  $x_i$  dengan seluruh *centroid* dari  $K$  buah *cluster*.

Setelah proses pengelompokan selesai, maka akan dilanjutkan dengan tahap kedua yaitu mengganti *centroid* terkandung dalam kromosom dengan *centroid* baru. *Centroid* baru ini ditentukan dengan cara menghitung rata-rata vektor dari tiap *cluster*. Untuk *cluster*  $C_i$ , *centroid* baru  $z_i^*$  dapat dihitung menggunakan Persamaan 3.5.

$$z_i^* = \frac{1}{n_i} \sum_{x_j \in C_i} x_j, i = 1, 2, \dots, K. \quad (3.5)$$

dengan  $z_i^*$  merupakan *centroid* dari *cluster*  $i$ ,  $n_i$  merupakan jumlah anggota *cluster*  $i$ , dan  $x_j$  merupakan dokumen  $j$  yang merupakan anggota dari *cluster*  $i$ .  $z_i^*$  ini akan menggantikan  $z_i$  sebelumnya di kromosom.

Tahap terakhir dari perhitungan *fitness* adalah menghitung nilai *fitness* itu sendiri. Pada penelitian ini, nilai *fitness* akan dihitung menggunakan *cosine similarity* seperti yang sudah dijelaskan pada Subbab 2.4. Sesuai dengan Persamaan 2.9 pada Subbab 2.6, perhitungan *fitness* sebagai metrik yang menggambarkan seberapa baiknya suatu calon solusi ditunjukkan dalam Persamaan 3.6 dengan beberapa penyesuaian.

$$f = \sum_{i=1}^K f_i, \quad (3.6)$$

$$f_i = \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\|x_j\| \times \|z_i\|}$$

Penjelasan dari Persamaan 3.6 adalah sebagai berikut. Nilai *fitness*  $f$  didapatkan dengan menjumlahkan nilai *fitness*  $f_i$  setiap *centroid*  $i = 1, 2, \dots, K$ . Nilai *fitness*  $f_i$  tiap *centroid* didapatkan dengan menjumlahkan jarak setiap dokumen ke *centroid* masing-masing (Subbab 2.6). Perhitungan jarak antara dokumen dengan *centroid* akan dihitung dengan menggunakan *cosine similarity* sehingga persamaan untuk menghitung *cosine similarity* (Persamaan 2.4 pada Subbab 2.4) dimasukkan ke dalam Persamaan 3.6. Semakin besar nilai *fitness*  $f$ , maka kromosom tersebut semakin mendekati solusi yang optimal.

### 3.6 Operasi Genetik Dalam Pengelompokan Dokumen

Seperti yang telah di bahas pada Subbab 2.3, ada beberapa operator genetik yang digunakan dalam algoritma genetika. Beberapa operasi genetik yang akan digunakan dalam penelitian ini di antaranya adalah inisialisasi populasi, seleksi, persilangan, dan mutasi. Sama seperti kromosom, operator genetik ini juga perlu dimodifikasi sedemikian rupa sehingga dapat digunakan untuk proses pengelompokan dokumen. Berikut merupakan pembahasan lebih detil mengenai setiap operator genetik yang akan digunakan.

### 3.6.1 Inisialisasi Populasi

Proses pengelompokan dengan menggunakan algoritma genetika akan dimulai dengan inisialisasi populasi awal. Seperti yang sudah dijelaskan pada Subbab 3.4, kromosom dari masing-masing individu akan terdiri dari  $K$  buah *centroid* dalam bentuk vektor. Populasi awal akan dibangkitkan dengan cara memilih  $K$  dokumen secara acak yang akan dijadikan *centroid* mula-mula. Kemudian proses tersebut akan dilakukan sebanyak  $P$  kali dengan  $P$  adalah banyaknya individu dalam populasi.

### 3.6.2 Seleksi

Proses seleksi akan dilakukan setiap iterasi (setiap generasi) untuk memilih calon induk dari generasi selanjutnya. Pada penelitian ini akan digunakan teknik seleksi *roulette-wheel selection* seperti yang telah dijelaskan pada Subbab 2.3.2. *Roulette-wheel selection* digunakan secara langsung pada penelitian ini dan tidak dimodifikasi. Individu dengan nilai *fitness* lebih tinggi akan memiliki probabilitas lebih tinggi untuk terpilih menjadi induk dari generasi selanjutnya dan akan masuk ke dalam proses persilangan. Namun karena masih ada peluang individu dengan nilai *fitness* tertinggi tidak terpilih, maka dalam penelitian ini juga akan digunakan teknik elitisme [12]. Dengan digunakannya elitisme, maka individu dengan nilai *fitness* terbaik akan disalin dan langsung menjadi anggota dari generasi berikutnya. Hal ini dilakukan untuk menjamin individu terbaik tidak hilang akibat tidak terpilih oleh *roulette-wheel selection*.

### 3.6.3 Persilangan

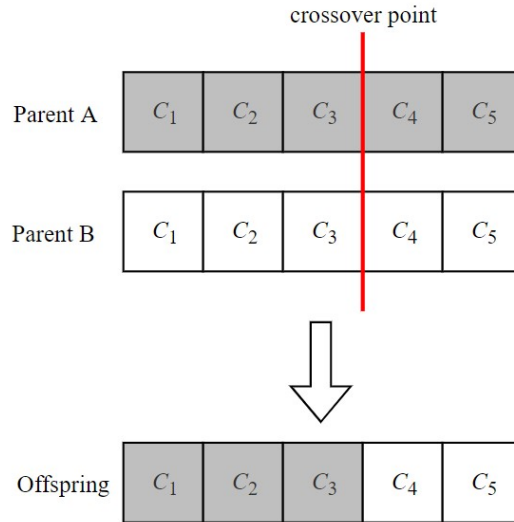
Dua kromosom yang terpilih dalam proses seleksi akan disilangkan untuk menghasilkan keturunan yang akan dimasukkan ke dalam generasi selanjutnya. Dalam penelitian ini akan digunakan teknik persilangan dengan satu titik potong (*single-point crossover*). Teknik *single-point crossover* yang digunakan dalam penelitian ini sedikit berbeda karena perlu disesuaikan dengan kebutuhan pengelompokan dokumen. Titik potong tidak ditentukan pada tingkat gen, namun ditentukan pada tingkat *centroid*. Apabila diketahui terdapat kromosom seperti pada Gambar 3.3, maka titik potong yang dipilih bukan berupa gen ( $w_{k,n}$ ) melainkan *centroid* ( $C_k$ ).

$$\begin{array}{ccccc} C_1 & C_2 & C_3 & C_4 & C_5 \\ w_{1,1}, w_{1,2}, \dots, w_{1,n}, & w_{2,1}, w_{2,2}, \dots, w_{2,n}, & w_{3,1}, w_{3,2}, \dots, w_{3,n}, & w_{4,1}, w_{4,2}, \dots, w_{4,n}, & w_{5,1}, w_{5,2}, \dots, w_{5,n} \end{array}$$

Gambar 3.3: Ilustrasi kromosom untuk persilangan

Sebagai contoh, diketahui dua buah kromosom yang akan mengalami persilangan adalah kromosom  $A$  dengan kromosom  $B$  seperti pada Gambar 3.4. Setelah itu akan ditentukan suatu titik potong (*crossover point*) secara acak dan ditandai dengan garis tegak berwarna merah pada Gambar 3.4. Proses selanjutnya adalah pembentukan individu keturunan. Proses ini dilakukan dengan cara menggabungkan seluruh gen induk  $A$  yang berada di sebelah kiri garis merah dengan seluruh gen induk  $B$  yang berada di sebelah kanan garis merah. Keturunan yang dihasilkan dari proses ini hanya satu individu seperti yang dapat dilihat pada Gambar 3.4.





Gambar 3.4: Ilustrasi persilangan

### 3.6.4 Mutasi

Setiap individu keturunan yang berasal dari proses persilangan akan memiliki peluang untuk mengalami mutasi. Seperti yang telah dijelaskan pada Subbab 2.3.4, mutasi dapat terjadi berdasarkan peluang mutasi  $\mu_m$ . Apabila mutasi terjadi, maka akan ditentukan gen mana yang akan mengalami mutasi dengan mengambilnya secara acak. Nilai gen yang baru akan ditentukan dari pembangkitan suatu angka acak yang berada antara batas minimum kemunculan istilah (0) dan total kemunculan istilah tersebut dari keseluruhan dokumen. Sebagai contoh dengan menggunakan Gambar 3.5a, akan ditentukan satu dari enam gen yang akan mengalami mutasi. Misalkan dalam contoh ini gen kedua yang mengalami mutasi (Gambar 3.5b). Lalu akan dilakukan pembangkitan angka acak antara 0 sampai dengan total kemunculan istilah dari keseluruhan dokumen (dalam contoh ini bernilai 9). Kromosom hasil mutasi ditunjukkan dalam Gambar 3.5c.

C1	C2	C3
3.05	1.43	15.85
14.23	5.12	9.45

(a) Kromosom sebelum mutasi

C1	C2	C3
3.05	<b>1.43</b>	15.85
14.23	5.12	9.45

(b) Pemilihan gen yang akan dimutasi

C1	C2	C3
3.05	<b>4.18</b>	15.85
14.23	5.12	9.45

(c) Kromosom hasil mutasi

Gambar 3.5: Ilustrasi mutasi kromosom



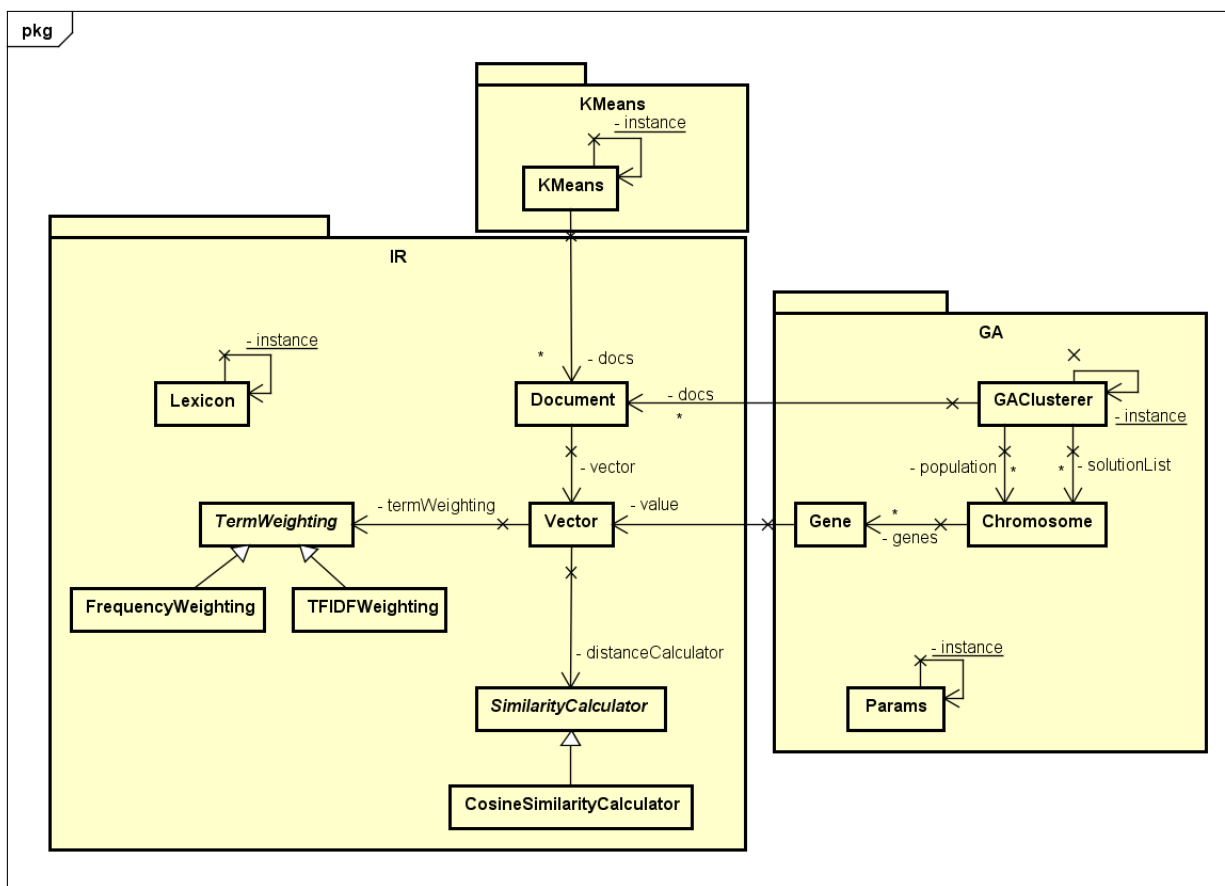
## BAB 4

### PERANCANGAN

Pada bab ini dijelaskan mengenai beberapa perancangan yang dilakukan dalam penelitian ini yaitu rancangan kelas dan rancangan antarmuka pengguna

#### 4.1 Rancangan Kelas

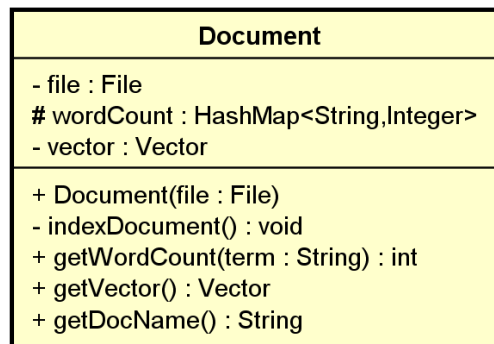
Berdasarkan hasil analisis dari masalah yang dihadapi, dibentuklah diagram kelas pada Gambar 4.1 sebagai gambaran dari perangkat lunak yang akan dibuat.



Gambar 4.1: Diagram kelas

Gambar 4.1 merupakan diagram kelas secara umum yang tidak memuat atribut dan *method* dari setiap kelas. Diagram kelas ini sengaja dibuat agar hubungan antar kelas dapat dengan lebih mudah dilihat. Penjelasan dari setiap kelas dalam Gambar 4.1 adalah sebagai berikut.

### 1 4.1.1 *Document*



Gambar 4.2: Kelas *Document*

2 Kelas ini merupakan representasi dari dokumen yang akan diproses dalam pengelompokan. Kelas  
 3 ini berfungsi untuk menyimpan informasi yang dibutuhkan dari sebuah dokumen selama proses  
 4 pengelompokan. Atribut yang dimiliki oleh kelas *Document* adalah:

- 5 • *file*: atribut ini bertipe *File* milik *package* java.io yang berfungsi untuk merepresentasikan *file*  
 6 dari dokumen yang akan diproses.
- 7 • *wordCount*: atribut ini bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*.  
 8 Atribut ini menyimpan pasangan kata yang dimiliki oleh dokumen tersebut dan frekuensinya.
- 9 • *vector*: atribut bertipe *Vector* ini merepresentasikan model ruang vektor pada sebuah dokumen.

10 *Method* yang terdapat dalam kelas ini adalah:

- 11 • *Document*: *method* ini merupakan *constructor* dengan sebuah parameter bertipe *File* yaitu  
 12 *file* dari dokumen yang akan dikelompokkan.
- 13 • *indexDocument*: *method* tanpa kembalian (*void*) yang berfungsi untuk mengindeks dokumen  
 14 untuk mengisi atribut *wordCount*.
- 15 • *getWordCount*: *method* yang berfungsi untuk mengembalikan banyaknya *term* muncul dalam  
 16 dokumen.
- 17 • *getVector*: *method* ini merupakan *getter* dari atribut *vector*.
- 18 • *getDocName*: *method* ini mengembalikan nama *file* dari dokumen.

### 4.1.2 *Vector*

Vector	
-	termsWeight : HashMap<String,Double>
-	similarityCalculator : SimilarityCalculator
-	termWeighting : TermWeighting
+	Vector(wordCount : HashMap<String,Integer>)
-	generateVector(wordCount : HashMap<String,Integer>) : void
+	getWeight(term : String) : double
+	calculateWeight(term : String, wordCount : HashMap<String,Integer>) : double
+	calculateSimilarity(otherVSM : Vector) : double
+	setTermsWeight(termsWeight : HashMap<String,Double>) : void
+	setWeight(term : String, value : double) : void
+	getLength() : double
+	getKeySet() : Set<String>
+	getTermsWeight() : HashMap<String,Double>
+	getDimension() : int

Gambar 4.3: Kelas *Vector*

Kelas ini merupakan kelas yang merepresentasikan sebuah vektor. Kelas ini memiliki fungsi dan atribut yang berfungsi untuk menunjang seluruh aktivitas yang melibatkan suatu vektor. Atribut yang dimiliki kelas ini adalah:

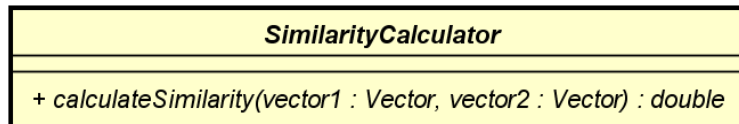
- *termsWeight*: atribut ini bertipe *Hashmap* dengan *key* berupa *String* dan *value* berupa *Double*. Atribut ini menyimpan pasangan *term* dan bobotnya sesuai dengan metode pembobotan.
- *similarityCalculator*: atribut ini bertipe *SimilarityCalculator* dan merupakan objek yang akan digunakan untuk menghitung kemiripan antar vektor.
- *termWeighting*: atribut ini bertipe *TermWeighting* dan merupakan objek yang akan digunakan untuk menghitung bobot dari setiap dimensi dalam suatu vektor.

*Method* yang terdapat dalam kelas ini adalah:

- *Vector*: *method* ini merupakan constructor dengan sebuah parameter yaitu *wordCount* bertipe *HashMap<String,Integer>* yang merupakan pasangan kata dan banyak kemunculannya dalam dokumen.
- *generateVector*: *method* ini merupakan *method* dengan sebuah parameter *HashMap<String,Integer>* untuk mengisi atribut *termsWeight* dengan bobot tiap *term*.
- *getWeight*: *method* ini berfungsi untuk mengembalikan bobot dari *term*.
- *calculateWeight*: *method* ini membutuhkan dua buah parameter yaitu sebuah *term* bertipe *String* dan sebuah *HashMap<String,Integer>*. *Method* ini berfungsi untuk menghitung bobot dari *term* berdasarkan frekuensi yang terdapat pada *HashMap*.
- *calculateSimilarity*: *method* ini berfungsi untuk menghitung kemiripan (*similarity*) antara vektor ini dengan *otherVector* menggunakan metode yang dipilih oleh pengguna.
- *setTermsWeight*: *method* ini merupakan *setter* dari atribut *termsWeight*.
- *setWeight*: *method* ini berfungsi untuk mengubah bobot *term* menjadi *value*.
- *getLength*: *method* ini berfungsi untuk mendapatkan panjang dari vektor.

- *getKeyset*: *method* ini berfungsi untuk mendapatkan himpunan *term* yang ada pada vektor ini.
- *getTermsWeight*: *method* ini merupakan *getter* dari atribut *termsWeight*.
- *getDimension*: *method* ini berfungsi untuk mendapatkan besarnya dimensi dari vektor ini.

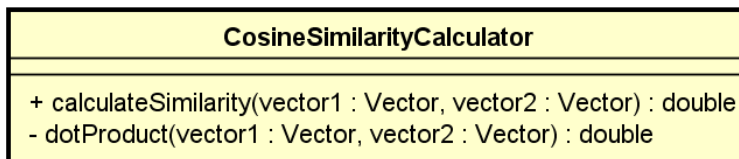
#### 4.1.3 *SimilarityCalculator*



Gambar 4.4: Kelas *SimilarityCalculator*

Kelas ini merupakan kelas abstrak yang berfungsi untuk menghitung kemiripan antara dua buah objek bertipe *Vector*. Kelas ini tidak memiliki atribut dan hanya memiliki sebuah *method* abstrak yaitu *calculateSimilarity* yang memiliki dua buah parameter *vector1* dan *vector2*. Hasil yang dikembalikan oleh *method* ini adalah kemiripan dari kedua vektor tersebut sesuai dengan metode perhitungan jaraknya.

#### 4.1.4 *CosineSimilarityCalculator*

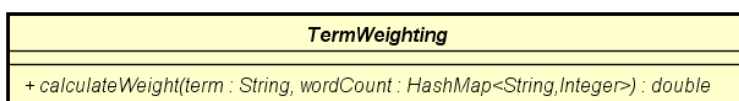


Gambar 4.5: Kelas *CosineSimilarityCalculator*

Kelas ini mengimplementasikan kelas abstrak *SimilarityCalculator*. Kelas ini memiliki satu *method* tambahan selain melakukan *override* pada *method calculateSimilarity*. *Method* yang ada pada kelas ini adalah:

- *calculateDistance*: *method* ini merupakan *method* yang diturunkan dari kelas *SimilarityCalculator*. *Method* ini mengembalikan *similarity* dari *vector1* dan *vector2* yang dihitung menggunakan persamaan cosinus (Persamaan 2.4).
- *dotProduct*: *method* ini berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vector1* dan *vector2*.

#### 4.1.5 *Term Weighting*



Gambar 4.6: Kelas *TermWeighting*

- 1 Kelas ini merupakan kelas abstrak yang merepresentasikan metode perhitungan bobot dalam suatu  
 2 vektor. Kelas ini hanya memiliki satu buah *method* yaitu *calculateWeight*. *Method* ini berfungsi  
 3 untuk menghitung bobot dari *term* berdasarkan metode pembobotan yang dipilih oleh pengguna.

#### 4 4.1.6 *FrequencyWeighting*

FrequencyWeighting
+ calculateWeight(term : String, wordCount : HashMap<String,Integer>) : double

Gambar 4.7: Kelas *FrequencyWeighting*

- 5 Kelas ini mengimplementasikan kelas abstrak *TermWeighting*. Kelas ini hanya memiliki satu *method*  
 6 yang diturunkan langsung dari kelas *TermWeighting* yaitu *method calculateWeight*. *Method* ini  
 7 berfungsi untuk menghitung bobot dari *term* menggunakan bobot frekuensi seperti yang telah  
 8 dijelaskan pada Subbab 2.5.1.

#### 9 4.1.7 *TFIDFWeighting*

TFIDFWeighting
+ calculateWeight(term : String, wordCount : HashMap<String,Integer>) : double - calculateTF(term : String, wordCount : HashMap<String,Integer>) : double - calculateIDF(term : String) : double

Gambar 4.8: Kelas *TFIDFWeighting*

- 10 Kelas ini mengimplementasikan kelas abstrak *TermWeighting*. Kelas ini memiliki dua *method*  
 11 tambahan selain melakukan *override* pada *method calculateWeight*. *Method* yang ada pada kelas ini  
 12 adalah:
- 13 • *calculateWeight*: *method* ini merupakan *method* yang diturunkan dari kelas *TermWeighting*.  
 14 *Method* ini mengembalikan bobot dari *term* yang dihitung menggunakan teknik TF-IDF.
  - 15 • *calculateTF*: *method* ini berfungsi untuk menghitung *TF* dari suatu *term*.
  - 16 • *calculateIDF*: *method* ini berfungsi untuk menghitung *IDF* dari suatu *term*.

#### 1 4.1.8 *Lexicon*

Lexicon	
-	globalTermCount : HashMap<String,Integer>
-	documentFrequency : HashMap<String,Integer>
-	instance : <u>Lexicon</u>
-	numberOfDocument : int
-	Lexicon()
+	<u>getInstance() : Lexicon</u>
+	insertTerm(term : String) : void
+	getAllTermList() : Set<String>
+	updateDF(term : String) : void
+	getNumberOfDocument() : int
+	setNumberOfDocument(numberOfDocument : int) : void
+	getDocumentFrequency(term : String) : int

Gambar 4.9: Kelas *Lexicon*

2 Kelas ini merepresentasikan sebuah kamus yang menangani seluruh kebutuhan dalam proses  
3 pengelompokan yang membutuhkan akses global untuk keseluruhan koleksi dokumen. Atribut yang  
4 ada dalam kelas ini adalah:

- 5 • *globalTermCount*: atribut ini bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe  
6 *Integer*. Atribut ini berfungsi untuk menyimpan seluruh *term* yang muncul dan banyak  
7 kemunculannya dalam keseluruhan koleksi dokumen.
- 8 • *documentFrequency*: atribut ini bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe  
9 *Integer*. Atribut ini berfungsi untuk menyimpan seluruh frekuensi dokumen dari tiap term.
- 10 • *instance*: atribut ini merupakan objek bertipe *Lexicon* sebagai instansiasi satu-satunya dari  
11 kelas *Lexicon* karena kelas ini bersifat *singleton*.
- 12 • *numberOfDocument*: atribut ini menyimpan banyaknya dokumen yang terdaftar di *Lexicon*.

13 *Method* yang ada pada kelas ini adalah:

- 14 • *Lexicon*: *method* ini merupakan *constructor private* untuk menjamin tidak akan ada lebih  
15 dari satu *instance* selama perangkat lunak berjalan.
- 16 • *getInstance*: *method* ini merupakan *method static* yang berfungsi sebagai *getter* dari atribut  
17 *instance*.
- 18 • *insertTerm*: *method* ini berfungsi untuk memasukkan *term* ke dalam atribut *globalTermCount*.
- 19 • *getAllTermList*: *method* ini bertugas mengembalikan daftar seluruh *term* yang pernah muncul  
20 di seluruh koleksi dokumen.
- 21 • *updateDF*: *method* ini berfungsi untuk menambah nilai TF dari *term* "*term*".
- 22 • *getNumberOfDocument*: *method* ini merupakan *getter* dari atribut *numberOfDocument*.
- 23 • *setNumberOfDocument*: *method* ini merupakan *setter* dari atribut *numberOfDocument*.
- 24 • *getDocumentFrequency*: *method* ini berfungsi untuk mendapatkan nilai DF dari *term* "*term*".



### 1 4.1.9 *Gene*

Gene
- value : Vector
+ Gene(value : Vector)
+ getValue() : Vector
+ mutate() : void

Gambar 4.10: Kelas *Gene*

2 Kelas ini merepresentasikan gen dalam algoritma genetika. Kelas ini hanya memiliki sebuah atribut  
 3 *value* bertipe *Vector*. Atribut ini menyimpan vektor yang menjadi titik pusat *cluster* (*centroid*).  
 4 *Method* yang ada pada kelas ini adalah:

- 5 • *Gene: method* ini merupakan *constructor* dari kelas *Gene* yang membutuhkan sebuah para-  
 6 meter bertipe *Vector* untuk mengisi atribut *value*.
- 7 • *getValue: method* ini merupakan *getter* dari atribut *value*.
- 8 • *mutate: method* ini berfungsi untuk melakukan mutasi pada gen. *Method* ini sebenarnya  
 9 hanya bertugas memanggil fungsi *mutate()* dari atribut *value*.

### 10 4.1.10 *Chromosome*

Chromosome
- genes : List<Gene> - rand : Random - fitnessValue : double - clusteringResult : HashMap<Document,Integer>
+ Chromosome() + addGene(gene : Gene) : void + mutate() : void + crossover(otherChromosome : Chromosome) : Chromosome - determineCluster(docs : List<Document>) : void + computeFitness() : double + getFitness() : double + getAllGenes() : List<Gene> + getClusteringResult() : HashMap<Document,Integer> + compareTo(o : Chromosome) : int

Gambar 4.11: Kelas *Chromosome*

11 Kelas ini merepresentasikan kromosom dalam algoritma genetika (Subbab ??). Atribut yang  
 12 terdapat dalam kelas ini adalah:

- 13 • *genes*: atribut bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam  
 14 kromosom.
- 15 • *rand*: atribut ini merupakan objek *Random* milik *Java* dan berfungsi untuk membangkitkan  
 16 bilangan acak yang dibutuhkan dalam setiap proses dalam kromosom.

1 • *fitnessValue*: atribut ini menyimpan nilai *fitness* dari kromosom.

2 • *clusteringResult*: atribut ini akan menyimpan hasil dari pengelompokan. Atribut ini bertipe  
3 *HashMap* yang menyimpan pasangan dokumen dan *cluster* dari dokumen tersebut.

4 *Method* yang terdapat dalam kelas ini adalah:

5 • *Chromosome*: *method* ini merupakan *constructor* tanpa parameter untuk membentuk objek  
6 dari kelas *Chromosome*.

7 • *addGene*: *method* ini bertugas untuk menambahkan satu gen ke dalam kromosom (ke dalam  
8 atribut *genes*).

9 • *mutate*: *method* ini berfungsi untuk melakukan mutasi pada kromosom dengan cara melakukan  
10 mutasi pada sebuah gen secara acak (Subbab 2.3.4).

11 • *crossover*: *method* ini bertugas untuk melakukan persilangan dengan kromosom *otherChro-*  
12 *mosome* untuk menghasilkan keturunan (Subbab 2.3.3).

13 • *determineCluster*: *method* ini berfungsi untuk menentukan keanggotaan dari setiap dokumen.

14 • *computeFitness*: *method* ini mengembalikan nilai *fitness* dari kromosom (Subbab ??).

15 • *getFitness*: *method* ini merupakan *getter* dari atribut *fitnessValue*.

16 • *getAllGenes*: *method* ini merupakan *getter* dari atribut *genes*.

17 • *getClusteringResult*: *method* ini merupakan *getter* dari atribut *clusteringResult*.

18 • *compareTo*: *method* ini merupakan turunan dari *interface Comparable* milik *Java* yang  
19 dibutuhkan untuk membandingkan kromosom ini dengan kromosom *o*. *Method* ini nantinya  
20 akan digunakan dalam proses *sorting*.

1 4.1.11 *GAClusterer*

GAClusterer	
-	population : List<Chromosome>
-	docs : List<Document>
-	instance : GAClusterer
-	solutionList : List<Chromosome>
-	isRunning : boolean
-	progress : ReadOnlyDoubleWrapper = new ReadOnlyDoubleWrapper()
-	GAClusterer()
+	isIsRunning() : boolean
+	setIsRunning(isRunning : boolean) : void
+	getProgress() : double
+	progressProperty() : ReadOnlyDoubleProperty
+	getInstance() : GAClusterer
+	rouletteWheelSelect() : Chromosome
-	elitism(numOfInstance : int) : List<Chromosome>
+	selection(elitismCount : int) : List<Chromosome>
+	initialize() : void
+	getAllDocs() : List<Document>
+	cluster() : void
+	getSolution() : Chromosome
+	reset() : void

Gambar 4.12: Kelas *GAClusterer*

2 Kelas ini merupakan kelas utama yang akan mengatur jalannya proses pengelompokan menggunakan  
3 algoritma genetika. Kelas ini merupakan kelas *singleton*. Atribut yang terdapat dalam kelas ini  
4 adalah:

- 5 • *population*: atribut ini bertipe *List of Chromosome* yang merepresentasikan populasi pada  
6 generasi saat ini.
- 7 • *docs*: atribut ini bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi  
8 dokumen.
- 9 • *instance*: atribut *static* ini berfungsi untuk menyimpan *instance* dari kelas *GAClusterer*.
- 10 • *solutionList*: atribut ini bertipe *List of Chromosome* yang mencatat kromosom dengan nilai  
11 *fitness* terbaik untuk setiap generasinya.
- 12 • *isRunning*: atribut ini berfungsi untuk menyimpan status dari operasi pengelompokan meng-  
13 gunakan GA. Apabila bernilai *true* maka program sedang berjalan dan *false* apabila tidak  
14 sedang berjalan.
- 15 • *progress*: atribut ini bertipe *ReadOnlyDoubleWrapper* dan berfungsi untuk menyimpan per-  
16 kembangan dari pengerjaan tugas pengelompokan ini ke antarmuka pengguna.

17 *Method* yang terdapat dalam kelas ini adalah:

- 18 • *GAClusterer*: *method* ini merupakan *constructor private* yang berfungsi untuk menjamin tidak  
19 akan ada *instance* dibuat diluar dari kelas ini.
- 20 • *setIsRunning*: *method* ini merupakan *setter* dari atribut *isRunning*.

- *getProgress*: *method* ini mengembalikan perkembangan pekerjaan program (skala 0 sampai dengan 1).
- *progressProperty*: *method* ini merupakan *getter* dari atribut *progress*.
- *getInstance*: *method* ini merupakan *getter* dari atribut *instance*.
- *rouletteWheelSelect*: *method* ini bertugas untuk memilih sebuah kromosom menggunakan teknik *roulette-wheel selection* dari populasi.
- *elitism*: *method* ini bertugas untuk memilih *n* kromosom elit yang akan langsung masuk ke generasi berikutnya.
- *selection*: *method* ini bertugas untuk melakukan *roulette-wheel selection* sebanyak populasi untuk menghasilkan populasi dari generasi selanjutnya.
- *initialize*: *method* ini berfungsi untuk melakukan *indexing* dokumen dan membentuk populasi awal.
- *getAllDocs*: *method* ini merupakan *getter* dari atribut *docs*.
- *cluster*: *method* ini merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan menggunakan algoritma genetika.
- *getSolution*: *method* ini berfungsi untuk mengembalikan solusi dari proses pengelompokan menggunakan algoritma genetika.
- *reset*: *method* ini berfungsi untuk mengatur ulang seluruh atribut untuk proses pengelompokan berikutnya.

#### 4.1.12 *Params*

Params
<ul style="list-style-type: none"> <li>- <u>instance</u> : <u>Params</u></li> <li>- <u>filepath</u> : String</li> <li>- K : int</li> <li>- P : int</li> <li>- <u>weightMethod</u> : int</li> <li>- <u>mu_m</u> : double</li> <li>- <u>maxIt</u> : int</li> <li>- <u>elitismCount</u> : int</li> <li>- <u>convergeGen</u> : int</li> <li>- <u>convergeEpsilon</u> : double</li> </ul>
<ul style="list-style-type: none"> <li>- Params()</li> <li>+ <u>getInstance()</u> : Params</li> <li>+ <u>insertParam(filepath : String, K : int, P : int, weightMethod : int, mu_m : double, maxIt : int, elitismCount : int, convergeGen : int, convergeEpsilon : double) : void</u></li> <li>+ <u>getK()</u> : int</li> <li>+ <u>getP()</u> : int</li> <li>+ <u>getWeightMethod()</u> : int</li> <li>+ <u>getMu_m()</u> : double</li> <li>+ <u>getMaxIt()</u> : int</li> <li>+ <u>getElitismCount()</u> : int</li> <li>+ <u>getConvergeGen()</u> : int</li> <li>+ <u>getConvergeEpsilon()</u> : double</li> <li>+ <u>getFilePath()</u> : String</li> </ul>

Gambar 4.13: Kelas *Params*

Kelas ini berfungsi untuk menyimpan seluruh parameter yang diberikan oleh pengguna agar dapat digunakan oleh setiap kelas yang membutuhkannya. Kelas ini bersifat *singleton* sehingga hanya akan ada satu buah *instance* selama perangkat lunak berjalan. Atribut yang ada dalam kelas ini adalah:

- *instance*: atribut ini merupakan objek bertipe *Params* sebagai instansiasi satu-satunya dari kelas *Params* karena kelas ini bersifat *singleton*.
  - *filepath*: atribut ini berfungsi untuk menyimpan alamat dari direktori dokumen yang akan dikelompokkan.
  - *K*: atribut ini berfungsi untuk menyimpan banyaknya *cluster* yang akan dibentuk dalam proses pengelompokan.
  - *P*: atribut ini berfungsi untuk menyimpan banyaknya populasi yang akan dibentuk dalam proses pengelompokan menggunakan algoritma genetika.
  - *weightingMethod*: atribut ini berfungsi untuk menyimpan metode pembobotan yang akan digunakan dalam proses pengelompokan. Apabila atribut ini bernilai 0 maka metode yang digunakan adalah TF-IDF sedangkan apabila bernilai 1 maka metode yang akan digunakan adalah bobot frekuensi.
  - *mu\_m*: atribut ini berfungsi untuk menyimpan probabilitas mutasi dalam bentuk bilangan riil bernilai antara 0 sampai dengan 1.
  - *maxIt*: atribut ini berfungsi untuk menyimpan banyaknya iterasi maksimal yang dapat dilakukan.
  - *elitismCount*: atribut ini berfungsi untuk menyimpan banyaknya individu yang akan dijadikan elit pada tahap seleksi.
  - *convergeGen*: atribut ini berfungsi untuk menyimpan banyaknya generasi konvergen sebelum proses pengelompokan diberhentikan.
  - *convergeEpsilon*: atribut ini berfungsi untuk menyimpan nilai yang akan digunakan untuk membandingkan *fitness* tiap solusi dari setiap generasi untuk menentukan apakah sudah tercapai konvergen atau belum.
- Method yang ada pada kelas ini adalah:
- *Params*: *method* ini merupakan *constructor private* untuk menjamin tidak akan ada lebih dari satu *instance* selama perangkat lunak berjalan.
  - *getInstance*: *method* ini merupakan *method static* yang berfungsi sebagai *getter* dari atribut *instance*.
  - *insertParam*: *method* ini bertugas untuk memasukkan atau mengubah nilai dari setiap atribut dalam kelas ini.
  - *getK*: *method* ini merupakan *getter* dari atribut *K*.
  - *getP*: *method* ini merupakan *getter* dari atribut *P*.
  - *getWeightingMethod*: *method* ini merupakan *getter* dari atribut *weightingMethod*.
  - *getMu\_m*: *method* ini merupakan *getter* dari atribut *mu\_m*.
  - *getMaxIt*: *method* ini merupakan *getter* dari atribut *maxIt*.
  - *getElitismCount*: *method* ini merupakan *getter* dari atribut *elitismCount*.
  - *getConvergeGen*: *method* ini merupakan *getter* dari atribut *convergeGen*.
  - *getConvergeEpsilon*: *method* ini merupakan *getter* dari atribut *convergeEpsilon*.
  - *getFilepath*: *method* ini merupakan *getter* dari atribut *filepath*.

### 4.1.13 *KMeans*

KMeans	
- docs : List<Document>	
- solution : HashMap<Document,Integer>	
- instance : KMeans	
- solutionIntracuster : double	
- progress : ReadOnlyDoubleWrapper = new ReadOnlyDoubleWrapper()	
- KMeans()	
+ getProgress() : double	
+ progressProperty() : ReadOnlyDoubleProperty	
+ getSolution() : HashMap<Document,Integer>	
+ getInstance() : KMeans	
+ cluster() : void	
+ getSolutionIntracuster() : double	
- computeIntracuster(cluster : HashMap<Document,Integer>, centroids : Vector[]) : double	
- determineCluster(docs : List<Document>, centroids : Vector[]) : HashMap<Document,Integer>	

Gambar 4.14: Kelas *KMeans*

Kelas ini merupakan kelas utama yang akan mengatur jalannya proses pengelompokan menggunakan algoritma K-Means. Kelas ini merupakan kelas *singleton*. Atribut yang terdapat dalam kelas ini adalah:

- *docs*: atribut ini bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi dokumen.
- *solution*: atribut ini berfungsi untuk menyimpan hasil pengelompokan dalam bentuk himpunan pasangan dokumen dan keanggotaan *cluster* dari dokumen tersebut.
- *instance*: atribut *static* ini berfungsi untuk menyimpan *instance* dari kelas *KMeans*.
- *solutionIntracuster*: atribut ini berfungsi untuk menyimpan nilai *intracuster* dari solusi yang telah didapatkan dari proses pengelompokan.
- *isRunning*: atribut ini berfungsi untuk menyimpan status dari operasi pengelompokan menggunakan GA. Apabila bernilai *true* maka program sedang berjalan dan *false* apabila tidak sedang berjalan.
- *progress*: atribut ini bertipe *ReadOnlyDoubleWrapper* dan berfungsi untuk menyimpan perkembangan dari pengerjaan tugas pengelompokan ini ke antarmuka pengguna.

Method yang terdapat dalam kelas ini adalah:

- *KMeans*: *method* ini merupakan *constructor private* yang berfungsi untuk menjamin tidak akan ada *instance* dibuat diluar dari kelas ini.
- *setIsRunning*: *method* ini merupakan *setter* dari atribut *isRunning*.
- *getProgress*: *method* ini mengembalikan perkembangan pekerjaan program (skala 0 sampai dengan 1).
- *progressProperty*: *method* ini merupakan *getter* dari atribut *progress*.
- *getSolution*: *method* ini merupakan *getter* dari atribut *solution*.
- *getInstance*: *method* ini merupakan *getter* dari atribut *instance*.

- *cluster*: *method* ini merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan menggunakan algoritma K-Means.
- *getSolutionIntracuster*: *method* ini merupakan *getter* dari atribut *solutionIntracuster*.
- *computeIntracuster*: *method* ini bertugas untuk menghitung *intracuster* apabila diketahui *centroid* setiap *cluster* adalah *centroids* dan keanggotaan setiap dokumen adalah *cluster*.
- *determineCluster*: *method* ini berfungsi untuk menentukan keanggotaan dari setiap dokumen.

## 4.2 Perancangan Antarmuka Pengguna

Antarmuka yang dirancang untuk perangkat lunak ini hanya terdiri dari 2 halaman. Rancangan antarmuka ini dibuat sedemikian rupa sehingga memudahkan penggunaannya dalam melakukan pengujian terhadap perangkat lunak. Pada penelitian ini, perancangan antarmuka dibuat menggunakan perangkat lunak *balsamiq*<sup>1</sup>. Setiap objek dan *field* akan diberi label unik agar dapat disesuaikan dengan tabel keterangan. Berikut akan dibahas rancangan antarmuka pengguna dari perangkat ini.

### 4.2.1 Halaman Algoritma Genetika

Gambar 4.15 menunjukkan halaman yang dapat digunakan oleh pengguna untuk mengelompokkan dokumen menggunakan algoritma genetika. Pada halaman ini terdapat beberapa *field* yang dapat digunakan pengguna untuk mengatur nilai dari masing-masing parameter.

Gambar 4.15: Rancangan antarmuka halaman algoritma genetika

Penjelasan setiap *field* dalam halaman ini dijelaskan dalam Tabel 4.2.

<sup>1</sup><https://balsamiq.com/>

Kode	Nama	Jenis	Default value	Wajib	Aturan validasi
A01	Direktori Dokumen	<i>file chooser</i>	-	ya	-
A02	Direktori Hasil	<i>file chooser</i>	-	ya	-
A03	Banyaknya Cluster	<i>spinner</i>	5	ya	Nilai minimum 1
A04	Banyaknya Populasi	<i>spinner</i>	1	ya	Nilai minimum 1 dan harus lebih besar dari Individu Elitisme (A08)
A05	Metode Pembobotan	<i>dropdown</i>	TF-IDF	ya	-
A06	Probabilitas Mutasi	<i>spinner</i>	0.05	ya	Nilai minimum 0.01, maksimum 1.00
A07	Maksimum Iterasi	<i>spinner</i>	100	ya	Nilai minimum 1 dan harus lebih besar dari Banyaknya Generasi Konvergen (A09)
A08	Individu Elitisme	<i>spinner</i>	1	ya	Nilai minimum 1 dan harus lebih kecil dari Banyaknya Populasi (A04)
A09	Banyaknya Generasi Konvergen	<i>spinner</i>	3	ya	Nilai minimum 2 dan harus lebih kecil dari Maksimum Iterasi (A07)
A10	Batas Konvergen	<i>spinner</i>	0.00001	ya	Hanya bisa bernilai $10^{-3}$ , $10^{-4}$ , $10^{-5}$ , $10^{-6}$ , dan $10^{-7}$

Tabel 4.1: Rincian *field* pada halaman algoritma genetika

Objek dengan kode A11 merupakan sebuah *progress bar* yang akan menampilkan perkembangan dari jalannya program untuk setiap iterasi. Apabila proses dalam suatu iterasi sudah selesai, maka akan mengubah nilai dari label A12 dan membuat *progress bar* kembali kosong. Label dengan kode A12 berfungsi untuk menampilkan status dari program yang sedang berjalan. Label ini tidak akan memiliki nilai apabila program belum dijalankan dan akan menampilkan status "Inisialisasi..." apabila program sedang melakukan pengindeksan dokumen dan inisialisasi populasi. Setelah iterasi dimulai maka label A12 akan menampilkan generasi saat ini. Sebagai contoh apabila label A12 menampilkan status "Generasi 1" artinya saat ini program sedang melakukan proses-proses pada generasi 1. Begitu juga untuk "Generasi 2" dan seterusnya. Tombol dengan kode A13 berfungsi



- 1 untuk memulai proses pengelompokan menggunakan algoritma genetika berdasarkan parameter
- 2 yang telah dimasukkan.

### 3 4.2.2 Halaman K-Means

The screenshot shows a software interface titled "GA-Based Document Clustering". It has two tabs: "GA" and "K-Means", with "K-Means" currently selected. The interface includes the following elements:

- Direktori Dokumen :** A text input field containing "D://Clustering/BBC" and a "Pilih" button (B01).
- Direktori Hasil :** A text input field containing "D://Clustering/result" and a "Pilih" button (B02).
- Parameter :** A section containing three controls:
  - Banyaknya Cluster (K) :** A spinner control set to "5" (B03).
  - Metode Pembobotan :** A dropdown menu showing "TF-IDF" (B04).
  - Maksimum iterasi :** A spinner control set to "100" (B05).
- Progress Bar :** A horizontal bar at the bottom left showing "Generasi 3/100" (B07). A green progress indicator is visible (B06).
- Mulai :** A large button at the bottom right (B08).

Gambar 4.16: Rancangan antarmuka halaman algoritma genetika

- 4 Gambar 4.16 merupakan tampilan yang akan digunakan oleh pengguna untuk melakukan pengelom-
- 5 pohan dokumen menggunakan algoritma K-Means. Berbeda dengan halaman algoritma genetika,
- 6 pada halaman ini hanya terdapat tiga buah parameter yang dapat diubah-ubah oleh pengguna
- 7 dalam melakukan pengelompokan.

Kode	Nama	Jenis	Default value	Wajib	Aturan validasi
B01	Direktori Dokumen	<i>file chooser</i>	-	ya	-
B02	Direktori Hasil	<i>file chooser</i>	-	ya	-
B03	Banyaknya Cluster	<i>spinner</i>	5	ya	Nilai minimum 1
B04	Metode Pembobotan	<i>dropdown</i>	TF-IDF	ya	-
B05	Maksimum Iterasi	<i>spinner</i>	100	ya	Nilai minimum 1

Tabel 4.2: Rincian *field* pada halaman algoritma genetika

- 8 Objek dengan kode B06 merupakan sebuah *progress bar* yang akan menampilkan perkembangan
- 9 dari jalannya program yang menyatakan banyaknya iterasi yang sudah selesai dijalankan diban-
- 10 dingkan dengan maksimum iterasi. Label B07 akan berisi keterangan dari proses yang sedang
- 11 berlangsung. Salah satu contoh isi dari label B07 adalah "Iterasi 2/100" yang berarti saat ini sedang
- 12 dilakukan pemrosesan pada iterasi kedua dari 100 iterasi. Tombol dengan kode B08 berfungsi untuk
- 13 memulai proses pengelompokan menggunakan algoritma K-Means berdasarkan parameter yang
- 14 telah dimasukkan.



## DAFTAR REFERENSI

- [1] Raposo, C., Antunes, C. H., dan Barreto, J. P. (2014) Automatic clustering using a genetic algorithm with new solution encoding and operators. *International Conference on Computational Science and Its Applications*, pp. 92–103. Springer.
- [2] Maulik, U. dan Bandyopadhyay, S. (2000) Genetic algorithm-based clustering technique. *Pattern recognition*, **33**, 1455–1465.
- [3] Holland, J. H. (1992) Genetic algorithms. *Scientific american*, **267**, 66–73.
- [4] Sivanandam, S. dan Deepa, S. (2007) *Introduction to Genetic Algorithms*. Springer Science & Business Media.
- [5] Gan, G., Ma, C., dan Wu, J. (2007) *Data clustering: theory, algorithms, and applications*. Siam.
- [6] Zhai, C. dan Massung, S. (2016) *Text data management and analysis: a practical introduction to information retrieval and text mining*. Morgan & Claypool.
- [7] Mecca, G., Raunich, S., dan Pappalardo, A. (2007) A new algorithm for clustering search results. *Data & Knowledge Engineering*, **62**, 504–522.
- [8] Srinivas, M. dan Patnaik, L. M. (1994) Genetic algorithms: A survey. *computer*, **27**, 17–26.
- [9] Russell, S. J. dan Norvig, P. (2016) *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
- [10] Schütze, H., Manning, C. D., dan Raghavan, P. (2008) *Introduction to information retrieval*. Cambridge University Press.
- [11] Aizawa, A. (2003) An information-theoretic perspective of tf-idf measures. *Information Processing & Management*, **39**, 45–65.
- [12] Ahn, C. W. dan Ramakrishna, R. S. (2003) Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, **7**, 367–385.



# LAMPIRAN A

## KODE PROGRAM

Listing A.1: MyCode.c

```

1
2 // This does not make algorithmic sense,
3 // but it shows off significant programming characters.
4
5 #include<stdio.h>
6
7 void myFunction( int input, float* output ) {
8     switch ( array[i] ) {
9         case 1: // This is silly code
10             if ( a >= 0 || b <= 3 && c != x )
11                 *output += 0.005 + 20050;
12             char = 'g';
13             b = 2^n + ~right_size - leftSize * MAX_SIZE;
14             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
15             strcpy(a,"hello_$@?");
16         }
17         count = ~mask | 0x00FF00AA;
18     }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```



## LAMPIRAN B

### HASIL EKSPERIMEN

Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4