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 masukkan 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

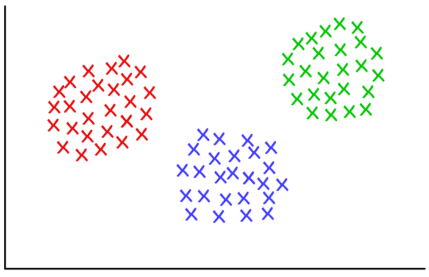
LANDASAN TEORI

Pengelompokan dokumen berkaitan erat dengan dua bidang ilmu dalam informatika. Pe-ngelompokan 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 sedemikan 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.

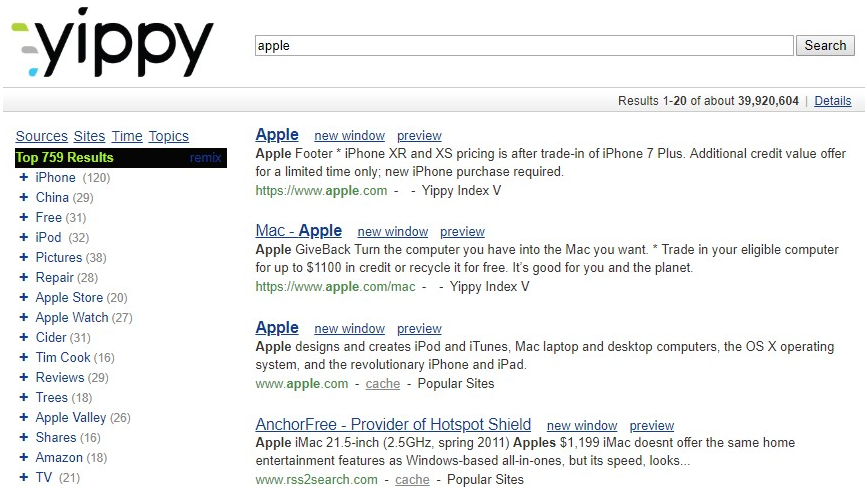
Recommender System

Recommender system adalah suatu sistem yang berfungsi untuk memprediksikan keinginan pengguna berdasarkan masukkan 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 disuka 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 disukainya.
- *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 mengelompokan 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 mengelompokan hasil pencariannya dinamakan dengan *clustering search engine*. Salah satu contoh *clustering search engine adalah* Yippy[1].



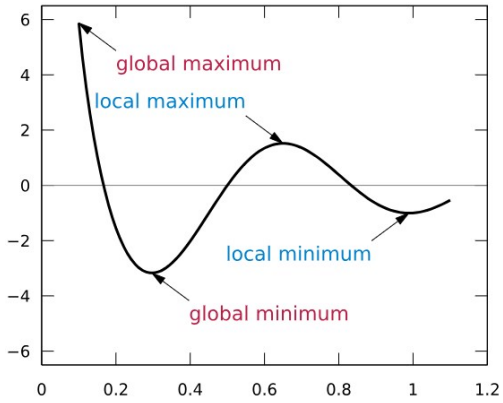
Gambar 2.2: Hasil pencarian (*clustered search result*) di Yippy

[1] 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. Persamaaan 2.1 merupakan persamaan untuk menghitung *centroid*

$$\mu_i = \frac{1}{N_i} \sum_{q=1}^{N_i} x_q \tag{2.1}$$

dengan μ_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.

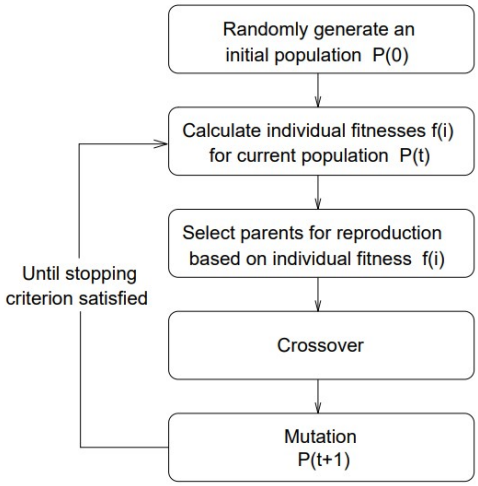
Penjelasan dari Algoritma 1 adalah sebagai berikut:

- 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 ϵ 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.

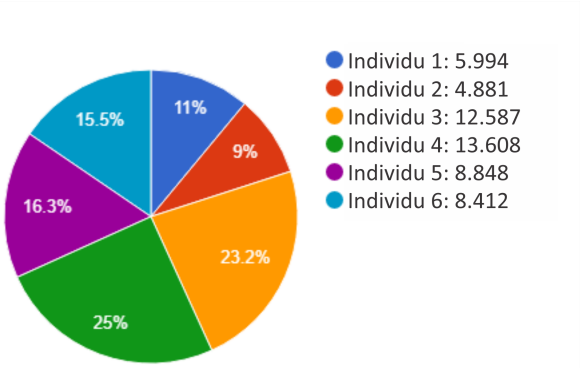
2.3.1 *Fitness*

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 memililiki 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.

$$P_i = \frac{f_i}{\sum_{j \in Pop} f_j} \tag{2.2}$$

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.

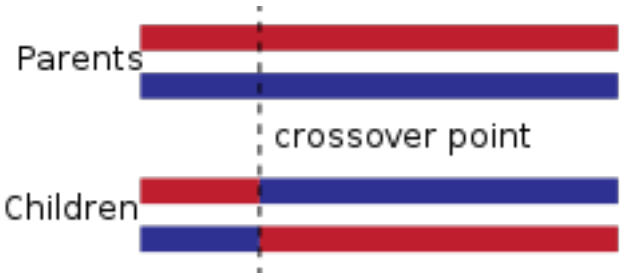
Proses seleksi dalam GA akan memilih sejumlah induk yang cukup untuk reproduksi dan membentuk generasi selanjutnya. Untuk meningkatkan performa GA, dapat juga diterapkan strategi *elitism* yaitu dengan langsung memindahkan satu atau beberapa individu dengan nilai *fitness* terbaik ke generasi selanjutnya. 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*

GA terjebak di *local optimum* [4]. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi terjadi berdasarkan probabilitas mutasi μ_m yang sudah ditentukan sebelumnya. Probabilitas mutasi menentukan seberapa sering kromosom akan dimutasi.

Jika tidak terjadi mutasi, maka keturunannya akan langsung masuk ke populasi setelah persilangan tanpa pengubahan. Apabila terjadi mutasi, satu atau beberapa bagian dari kromosom akan diubah. Mutasi seharusnya tidak dilakukan terlalu sering, karena jika terlalu sering dilakukan maka GA akan menjadi sama dengan algoritma pencarian acak primitif (*primitive random search*). Mutasi akan dilakukan dengan mengubah nilai dari suatu gen yang telah dipilih menjadi nilai lainnya dengan teknik tertentu (tergantung struktur data dari gen yang akan diubah).

2.3.5 Proses pencarian dalam algoritma genetika

Proses pencarian dalam GA secara umum dijelaskan dalam Algoritma 2. Proses pencarian ini memanfaatkan operasi genetik yang telah dijelaskan sebelumnya (Subbab 2.3.1 sampai dengan Subbab 2.3.4). Selain itu, terdapat algoritma lain yang menjelaskan masing-masing operasi genetik yang digunakan dalam Algoritma 2. Operasi seleksi yang telah dibahas pada Subbab 2.3.2 dijelaskan pada Algoritma 4. Operasi persilangan yang telah dibahas pada Subbab 2.3.3 dijelaskan pada 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* ← *array* yang menyimpan solusi tiap generasi
2: **repeat**
3: tambahkan individu dengan nilai *fitness* tertinggi dari *populasi* ke *solusi*
4: *populasi_baru* ← himpunan kosong
5: **for** i=1 **to** Size(*populasi*) **do**
6: x ← Seleksi-acak(*populasi*)
7: y ← Seleksi-acak(*populasi*)
8: *anak* ← Persilangan(x, y)
9: *rand* ← Random(0,1)
10: **if** *rand* ≤ *prob_mutasi* **then**
11: *anak* ← Mutasi(*anak*)
12: **end if**
13: tambahkan *anak* ke *populasi_baru*
14: **end for**
15: *populasi* ← *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 ← Length(x)
2: c ← 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*.

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

Penjelasan untuk fungsi Seleksi-acak pada Algoritma 4 adalah sebagai berikut:

- Pada baris 3, fungsi *Fitness* akan mengembalikan nilai *fitness* dari individu yang menjadi parameternya.
- Pada baris 5, variabel *terpilih* akan berisi suatu bilangan 0 sampai *sum*. Hal ini dilakukan dengan cara mengalikan sebuah bilangan riil dengan *sum*. Bilangan riil tersebut merupakan 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*

Penjelasan untuk fungsi Mutasi pada Algoritma 5 adalah sebagai berikut:

- Pada baris 1, fungsi *Length* akan mengembalikan panjang dari *string* parameterya.
- Pada baris 2, fungsi *Random* akan mengembalikan bilangan bulat acak antara 1 sampai N.

2.3.6 GA dalam Pengelompokan

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

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

2.4 Model Ruang Vektor

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

$$d_i = (w_{1,i}, w_{2,i}, ..., w_{n,i}) \tag{2.3}$$

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

$$s_{ij} = \frac{i \cdot j}{\| i \| \times \| j \|} \tag{2.4}$$

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

2.5 Pembobotan *Term* (*Term Weighting*)

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

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

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

Bobot TF-IDF merupakan pengembangan dari bobot frekuensi dengan memperhitungkan kemunculan suatu *term* secara global.

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

$$w_i = tf_i \tag{2.5}$$

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

- Doc 1: new home sales top forecasts
- Doc 2: home sales rise in july
- Doc 3: increase in home sales in july
- Doc 4: july new home sales rise

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

Term	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*

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

2.5.2 Bobot TF-IDF

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

Term frequency atau biasa dilambangkan sebagai $\text{tf}_{t,d}$ untuk *term* t pada dokumen d. Sama seperti yang telah dijelaskan pada Subbab 2.5.1, TF merupakan banyaknya kemunculan *term* pada

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

$$\mathrm{tf}_{t,d} = \frac{\mathrm{f}_{t,d}}{\sum_{t' \in d} \mathrm{f}_{t',d}} \tag{2.6}$$

dengan $\mathrm{tf}_{t,d}$ merupakan TF *term* t pada dokumen d, $\mathrm{f}_{t,d}$ merupakan banyaknya kemunculan *term* t dalam dokumen d, $t' \in d$ merupakan seluruh *term* t' yang merupakan anggota dari dokumen d, dan $\mathrm{f}_{t',d}$ merupakan banyaknya kemunculan *term* t' pada dokumen d.

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

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t} \tag{2.7}$$

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

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

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{2.8}$$

Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

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

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

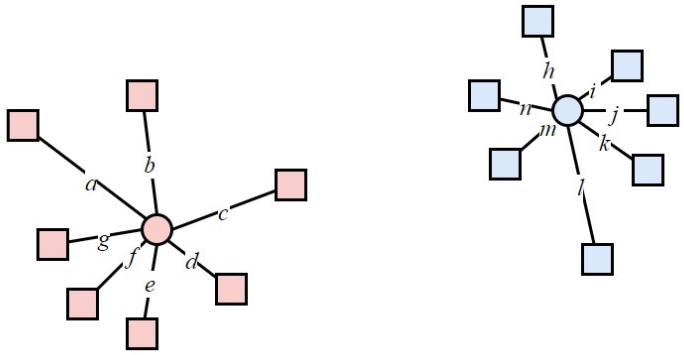
2.6 Metrik *Intracluster* untuk Mengukur Kinerja Metode *Clustering*

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

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

$$\begin{aligned} M &= \sum_{i=1}^{K} M_i, \\ M_i &= \sum_{x_j \in C_i} \| x_j - z_i \| \end{aligned} \tag{2.9}$$

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



Gambar 2.7: *Ilustrasi untuk jarak intracluster*

Persegi pada Gambar 2.7 mengilustrasikan objek yang akan dikelompokkan. Lingkaran mengilustrasikan titik pusat *cluster* (*centroid*). Garis yang menghubungkan objek dan *centroid* menggambarkan jarak antara objek dengan *centroid* dan ditandai dengan label antara huruf a sampai dengan n. Objek berwarna merah merupakan anggota dari *cluster* 1, sedangkan objek berwarna biru merupakan anggota dari *cluster* 2. Berdasarkan Persamaan 2.9, jarak *intracluster* dari *cluster* 1 (M_1) didapat dengan menjumlahkan a, b, c, d, e, f, dan g. Sedangkan, jarak *intracluster* dari *cluster* 2 (M_2) didapat dengan menjumlahkan h, i, j, k, l, m, dan n. Jarak *intracluster* total didapatkan dengan menjumlahkan jarak *intracluster* miliki *cluster* 1 dan *cluster* 2 ($a + b + ... + n$).

Semakin kecil nilai M, maka pengelompokan dianggap semakin baik karena jarak setiap objek ke titik pusatnya dekat. Metrik ini dapat mengukur seberapa baik objek sudah dikelompokan 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.

Term	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.

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

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

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

Term	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

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

$$\begin{aligned} \text{idf}_t &= \log \frac{N}{\text{df}_t} \\ \text{idf}_{\text{"}properti\text{"}} &= \log \frac{3}{1} = 0.4771 \end{aligned} \tag{3.2}$$

Penjelasan dari Persamaan 3.2 adalah sebagai berikut. N bernilai 3 karena banyaknya dokumen dalam seluruh koleksi dokumen adalah 3. df_t bernilai 1 karena hanya ada 1 dokumen yang memuat *term* "properti" yaitu dokumen 1. Hasil dari perhitungan IDF untuk setiap *term* ditunjukkan oleh Tabel 3.3.

Term	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

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

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

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

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

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

3.4 Representasi Kromosom

Dalam penelitian ini, kromosom tersusun dari gen-gen yang merupakan *centroid* dari K buah *cluster* yang akan dibentuk. *Centroid* disimpan dalam bentuk vektor yang terdiri dari N buah bilangan riil. N merupakan dimensi dari vektor tersebut yang merupakan banyaknya *term* berbeda yang terdapat pada seluruh koleksi dokumen. Secara umum representasi *centroid* ke dalam kromosom ditunjukkan pada Gambar 3.1.

$$\begin{array}{cccc} C_1 & C_2 & & C_k \\ w_{1,1}, w_{1,2}, ..., w_{1,n}, & w_{2,1}, w_{2,2}, ..., w_{2,n} & ... & w_{k,1}, w_{k,2}, ..., w_{k,n} \end{array}$$

Gambar 3.1: Formula representasi kromosom

Berdasarkan Gambar 3.1, kromosom akan memiliki $N \times K$ buah gen. N kata pertama merepresentasikan N dimensi dari *centroid* pertama C_1 $(w_{1,1}, w_{1,2}, ..., w_{1,n})$, N kata selanjutnya merepresentasikan N dimensi dari *centroid* kedua C_2 $(w_{2,1}, w_{2,2}, ..., w_{2,n})$, dan seterusnya sampai *centroid* C_k $(w_{k,1}, w_{k,2}, ..., w_{k,n})$. Sebagai contoh apabila diketahui ada tiga buah *centroid* dalam bidang dua dimensi **C1** (3.05, 1.43), **C2** (15.85, 14.23), dan **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 $x_i, i = 1, 2, ..., n$ ke dalam sebuah *cluster* C_j dengan *centroid* z_j sehingga memenuhi Persamaan 3.4.

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

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.

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

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.

$$\begin{aligned} \mathrm{f} &= \sum_{i=1}^{K} \mathrm{f}_i, \\ \mathrm{f}_i &= \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\| x_j \| \times \| z_i \|} \end{aligned} \tag{3.6}$$

Penjelasan dari Persamaan 3.6 adalah sebagai berikut. Nilai *fitness* f didapatkan dengan menjumlahkan nilai *fitness* f_i setiap *centroid* $i = 1, 2, ..., K$. 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

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*.

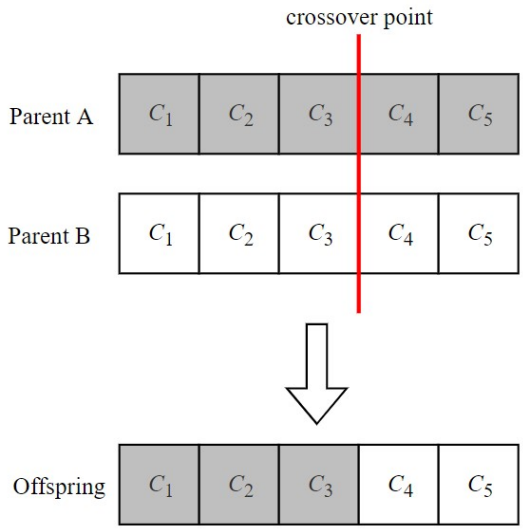
3.6.3 Persilangan

Dua kromosom yang terpilih dalam proeses 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).

C_1	C_2	C_3	C_4	C_5
$w_{1,1}, w_{1,2}, ..., w_{1,n},$	$w_{2,1}, w_{2,2}, ..., w_{2,n},$	$w_{3,1}, w_{3,2}, ..., w_{3,n},$	$w_{4,1}, w_{4,2}, ..., w_{4,n},$	$w_{5,1}, w_{5,2}, ..., w_{5,n}$

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 μ_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	**1.43**	15.85	14.23	5.12	9.45

(b) Pemilihan gen yang akan dimutasi

C1		**C2**		**C3**	
3.05	**4.18**	15.85	14.23	5.12	9.45

(c) Kromosom hasil mutasi

Gambar 3.5: Ilustrasi mutasi kromosom

BAB 4

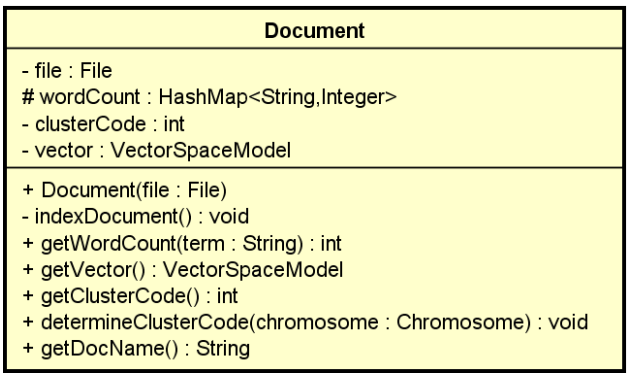
PERANCANGAN

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 4.11 sebagai gambaran dari perangkat lunak yang akan dibuat.

4.1.1 *Document*



Gambar 4.1: Kelas *Document*

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

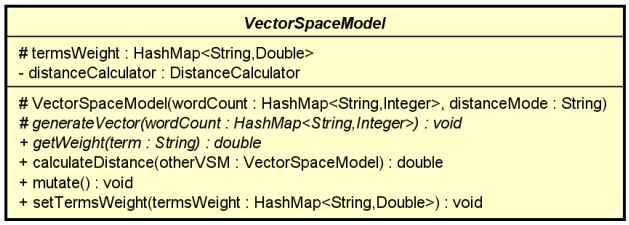
- *wordCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini menyimpan pasangan kata yang dimiliki oleh dokumen tersebut dan frekuensinya.
- *file*: atribut ini bertipe *File* milik *package* java.io yang berfungsi untuk merepresentasikan *file* dari dokumen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

Method yang terdapat dalam kelas ini adalah:

- *Document*: merupakan *constructor* dengan sebuah parameter bertipe *File* yaitu *file* dari dokumen yang akan dikelompokkan.

- *indexDocument*: *method* tanpa kembalian (*void*) yang berfungsi untuk mengindeks dokumen untuk mengisi atribut *wordCount*.
- *getWordCount*: berfungsi untuk mengembalikan banyaknya istilah *term* muncul dalam dokumen.
- *getVector*: merupakan *getter* dari variabel *vector*.
- *determineClusterCode*: berfungsi untuk menentukan *cluster* dari dokumen.
- *getDocName*: mengembalikan nama *file* dari dokumen.

4.1.2 *VectorSpaceModel*



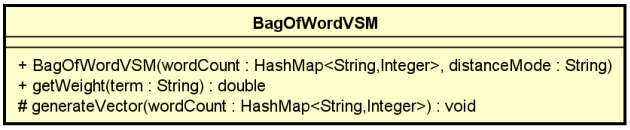
Gambar 4.2: Kelas *VectorSpaceModel*

Kelas ini merupakan kelas abstrak yang merepresentasikan model ruang vektor. kelas ini memiliki fungsi-fungsi yang umum dimiliki oleh sebuah model ruang vektor. Atribut yang dimiliki kelas ini adalah:

- *termsWeight*: bertipe *Hashmap* dengan *key* berupa *String* dan *value* berupa *Double*. atribut ini menyimpan pasangan istilah dan bebatnya sesuai dengan metode pembobotan.
- *distanceCalculator*: bertipe *DistanceCalculator* dan merupakan objek yang akan digunakan untuk menghitung jarak antar vektor.

Method yang terdapat dalam kelas ini adalah:

- *VectorSpaceModel*: merupakan constructor dengan dua parameter yaitu *wordCount* bertipe *Hashmap<String,Integer>* yang merupakan pasangan kata dan banyak kemunculannya dalam dokumen serta *distanceMode* bertipe *String* yang akan menentukan tipe perhitungan jarak antar vektor.
- *generateVector*: merupakan *method* tanpa parameter untuk mengubah banyak kemunculan kata menjadi berat.
- *getWeight*: berfungsi untuk mengembalikan berat dari istilah *term*.
- *calculateDistance*: berfungsi untuk menghitung jarak antara vektor ini dengan *otherVSM* menggunakan metode yang dipilih pada parameter di *constructor*.
- *mutate*: merupakan *method* untuk melakukan mutasi pada sebuah dimensi dalam vektor.
- *setTermsWeight*: merupakan setter dari atribut *termsWeight*.

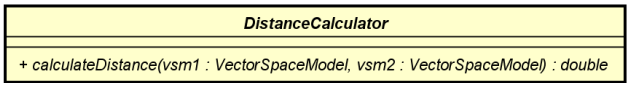


Gambar 4.3: Kelas *BagOfWordVSM*

4.1.3 *BagOfWordVSM*

Kelas ini merupakan kelas yang mengimplementasikan kelas *VectorSpaceModel*. Kelas ini menggunakan metode pembobotan *BagOfWord* di mana bobot tiap istilah merupakan banyaknya kemunculan istilah itu sendiri. Atribut yang dimiliki kelas ini seluruhnya merupakan atribut yang diturunkan dari kelas *VectorSpaceModel* dan hanya melakukan *override* pada *method* yang masih bersifat abstrak.

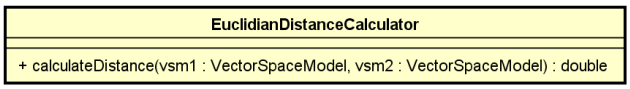
4.1.4 *DistanceCalculator*



Gambar 4.4: Kelas *DistanceCalculator*

Kelas ini merupakan kelas abstrak yang berfungsi untuk menghitung jarak antara dua buah objek bertipe *VectorSpaceModel*. Kelas ini tidak memiliki atribut dan hanya memiliki sebuah *method* abstrak yaitu *calculateDistance* yang memiliki dua buah parameter *vsm1* dan *vsm2*. Hasil yang dikembalikan oleh *method* ini adalah jarak dari kedua vektor tersebut sesuai dengan metode perhitungan jaraknya.

4.1.5 *EuclideanDistanceCalculator*



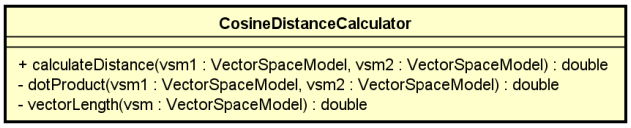
Gambar 4.5: Kelas *EuclideanDistanceCalculator*

Kelas ini mengimplementasikan kelas abstrak *DistanceCalculator*. Kelas ini hanya melakukan *override* pada *method calculateDistance* dengan melakukan perhitungan menggunakan jarak euclidean untuk menghitung jarak antara dua buah vektor (Subbab **??**).

4.1.6 *CosineDistanceCalculator*

Kelas ini juga mengimplementasikan kelas abstrak *DistanceCalculator*. Kelas ini memiliki dua *method* tambahan selain melakukan *override* pada *method calculateDistance*. *Method* yang ada pada kelas ini adalah:

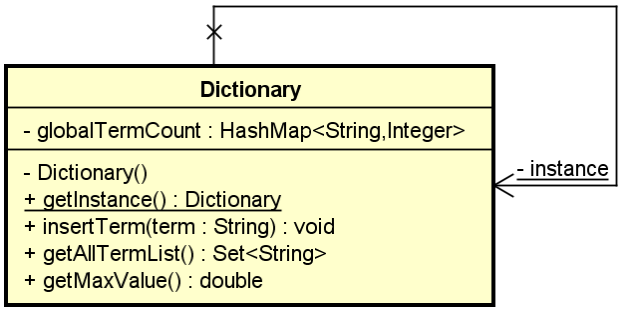
- *calculateDistance*: merupakan *method* yang diturunkan dari kelas *DistanceCalculator*. *Method* ini mengembalikan jarak dari *vsm1* dan *vsm2* yang dihitung menggunakan persamaan cosinus (Subbab **??**).
- *dotProduct*: berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vsm1* dan *vsm 2*.



Gambar 4.6: Kelas *CosineDistanceCalculator*

- *vectorLength*: berfungsi untuk menghitung panjang dari vektor *vsm*.

4.1.7 *Dictionary*



Gambar 4.7: Kelas *Dictionary*

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

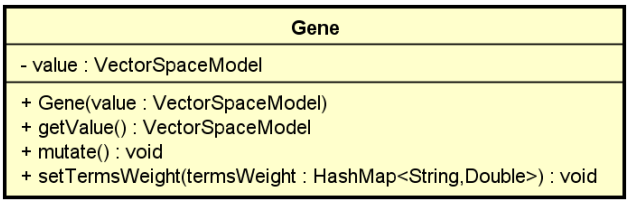
- *globalTermCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini berfungsi untuk menyimpan seluruh istilah yang muncul dan banyak kemunculannya dalam keseluruhan koleksi dokumen.
- *instance*: merupakan objek bertipe *Dictionary* sebagai instansiasi satu-satunya dari kelas *Dictionary* karena kelas ini bersifat *singleton*.

Method yang ada pada kelas ini adalah:

- *Dictionary*: merupakan *constructor private* untuk menjamin tidak akan ada lebih dari satu *instance* selama perangkat lunak berjalan.
- *getInstance*: merupakan *method static* yang berfungsi sebagai *getter* dari atribut *instance*.
- *insertTerm*: berfungsi untuk memasukkan istilah *term* ke dalam variabel *globalTermCount*.
- *getAllTermList*: bertugas mengembalikan daftar seluruh istilah yang pernah muncul di seluruh koleksi dokumen.
- *getValue*: bertugas mengembalikan banyaknya kata *term* muncul dalam seluruh koleksi dokumen.

4.1.8 *Gene*

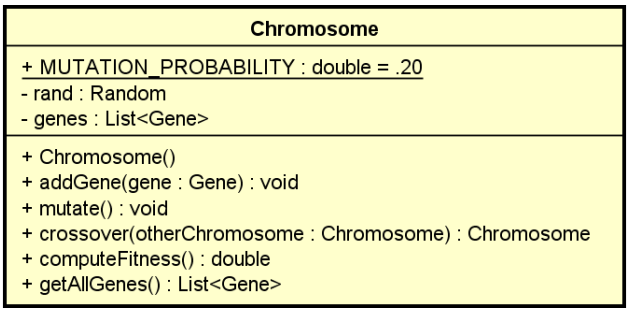
Kelas ini merepresentasikan gen dalam algoritma genetika. Kelas ini hanya memiliki sebuah atribut *value* bertipe *VectorSpaceModel*. Atribut ini menyimpan model ruang vektor yang menjadi titik pusat *cluster* (*centroid*). *Method* yang ada pada kelas ini adalah:



Gambar 4.8: Kelas *Gene*

- *Gene*: merupakan *constructor* dari kelas *Gene* yang membutuhkan sebuah parameter bertipe *VectorSpaceModel* untuk mengisi variabel *value*.
- *getValue*: merupakan *getter* dari atribut *value*.
- *mutate*: berfungsi untuk melakukan mutasi pada gen. *Method* ini sebenarnya hanya bertugas memanggil fungsi *mutate()* dari atribut *value*.
- *setTermsWeight*: berfungsi untuk mengubah nilai atribut *termsWeight* milik atribut *value*.

4.1.9 *Chromosome*



Gambar 4.9: Kelas *Chromosome*

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

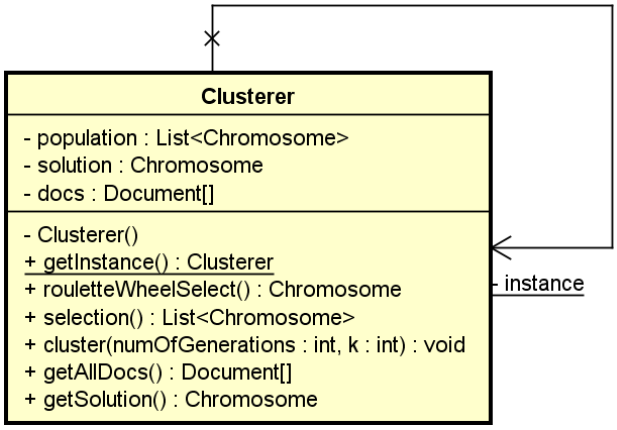
- *genes*: bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam kromosom.
- *MUTATION_ PROBABILITY*: merupakan atribut yang bersifat *static* dan *final* yang berisi probabilitas terjadinya mutasi dalam proses pembangkitan keturunan.

Method yang terdapat dalam kelas ini adalah:

- *Chromosome*: merupakan *constructor* tanpa parameter untuk membentuk objek dari kelas *Chromosome*.
- *addGene*: bertugas untuk menambahkan satu gen ke dalam kromosom (ke dalam atribut *genes*).
- *mutate*: berfungsi untuk melakukan mutasi pada kromosom dengan cara melakukan mutasi pada sebuah gen secara acak (Subbab 2.3.4).
- *crossover*: bertugas untuk melakukan persilangan dengan kromosom lain untuk mengahsilkan keturunan (Subbab 2.3.3).

- *computeFitness*: mengembalikan nilai *fitness* dari kromosom (Subbab **??**).
- *getAllGenes*: merupakan *getter* dari atribut *genes*.

4.1.10 *Clusterer*



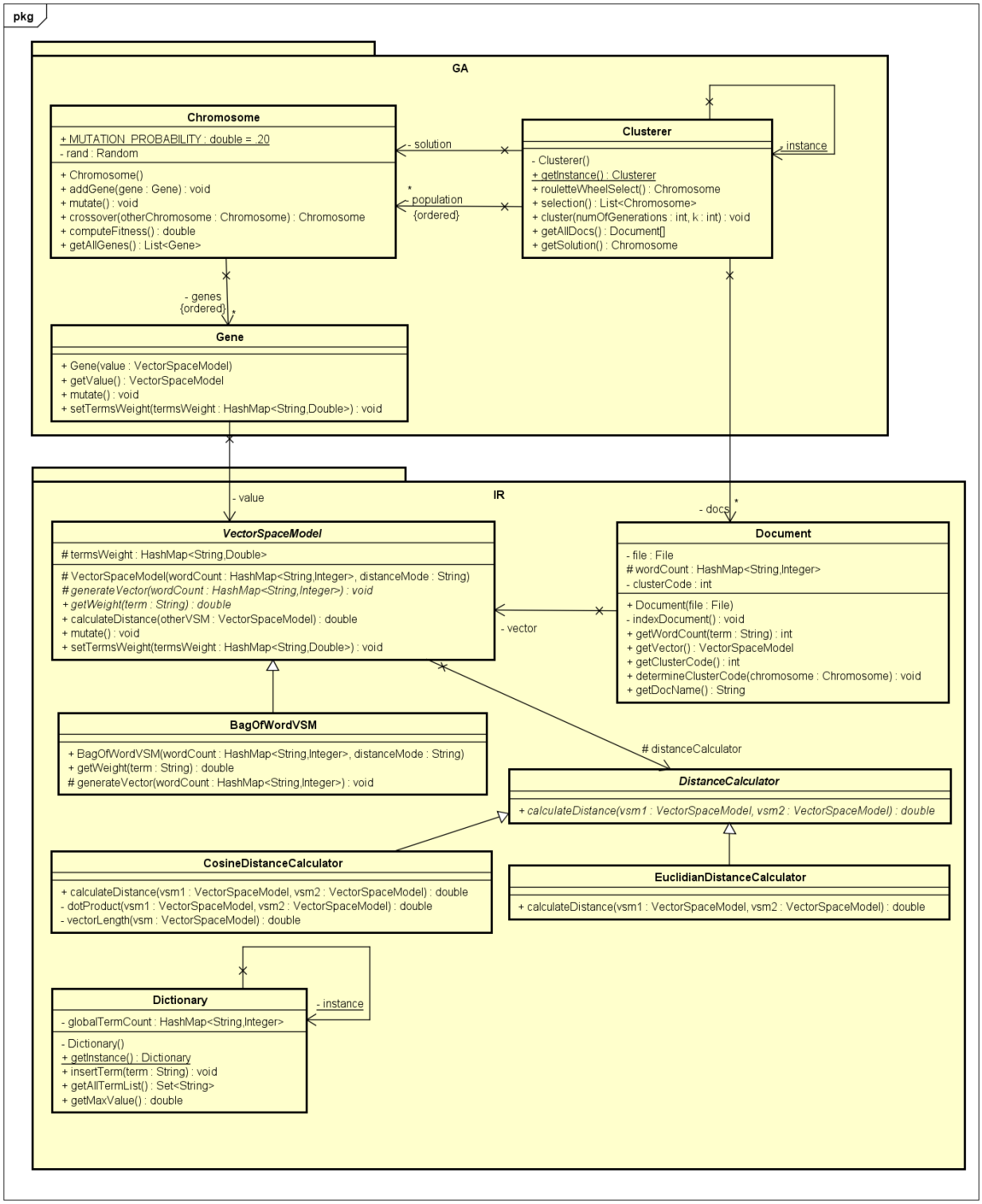
Gambar 4.10: Kelas *Clusterer*

Kelas ini merupakan kelas utama yang akan mengatur jalannya proses pengelompokan. Kelas ini merupakan kelas singleton. Atribut yang terdapat dalam kelas ini adalah:

- *docs*: bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi dokumen.
- *instance*: variabel *static* ini berfungsi untuk menyimpan *instance* dari kelas *Clusterer*.
- *population*: bertipe *List of Chromosome* yang merepresentasikan populasi pada generasi saat ini.
- *solution*: bertipe *Chromosome* yang mencatat kromosom dengan nilai *fitness* terbaik.

Method yang terdapat dalam kelas ini adalah:

- *Clusterer*: merupakan *constructor private* yang berfungsi untuk menjamin tidak akan ada *instance* dibuat diluar dari kelas ini.
- *getInstance*: merupakan *getter* dari variabel *instance*.
- *rouletteWheelSelect*: bertugas untuk memilih dua kromosom dan melakukan persilangan untuk menghasilkan sebuah keturunan selanjutnya.
- *selection*: bertugas untuk melakukan *roulette wheel selection* sebanyak populasi untuk menghasilkan populasi dari generasi selanjutnya.
- *cluster*: merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan dua parameter yaitu jumlah generasi dan nilai k.
- *getAllDocs*: merupakan *getter* dari atribut *docs*.
- *getSolution*: merupakan *getter* dari atribut *solution*.



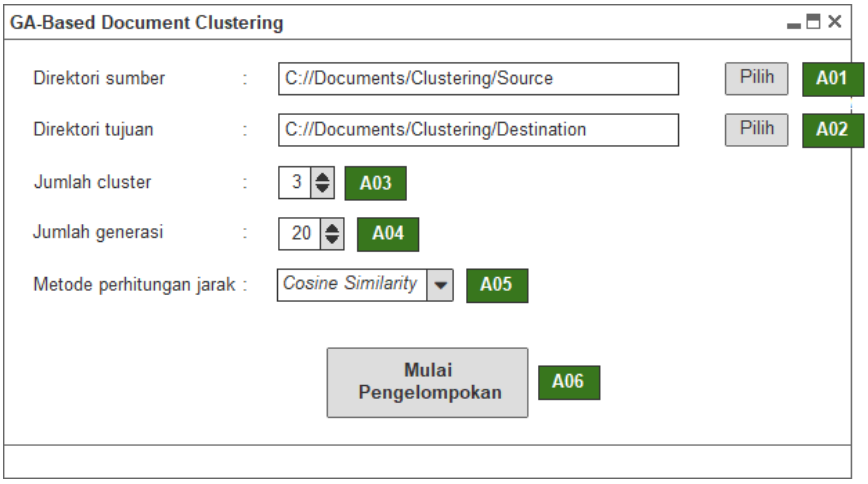
Gambar 4.11: Diagram kelas pada tahap perancangan

4.2 Perancangan Antarmuka Pengguna

Antarmuka yang dirancang untuk perangkat lunak ini hanya terdiri dari satu jendela utama dan dua jendela *pop-up*. Pada penelitian ini, perancangan antarmuka dibuat menggunakan perangkat lunak *balsamiq*[1]. 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 Jendela Utama

Gambar 4.12 merupakan jendela yang pertama kali ditampilkan saat perangkat lunak dijalankan. Jendela tersebut berisi berbagai macam hal yang dibutuhkan penguna dalam melakukan pengelompokan dokumen.



Gambar 4.12: Rancangan antarmuka jendela utama

Penjelasan setiap *field* dalam jendela utama adalah sebagai berikut:

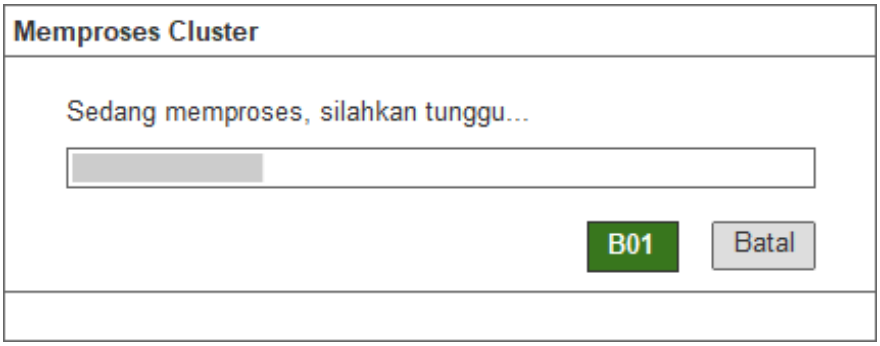
Kode	Nama	Jenis	*Defaut value*	Wajib	Aturan validasi
A01	Direktori sumber	*file chooser*	-	ya	Harus berupa direktori yang berisi dokumen (tidak boleh kosong)
A02	Direktori tujuan	*file chooser*	-	ya	Harus berupa direktori yang kosong
A03	Jumlah cluster	*spinner*	2	ya	Nilai minimum 2
A04	Jumlah generasi	*spinner*	1	ya	Nilai minimum 1
A05	Metode perhitungan jarak	*dropdown*	*Cosine Similarity*	ya	-

Tabel 4.1: Rincian *field* pada jendela utama

[1]https://balsamiq.com/

Jendela ini hanya memiliki sebuah tombol dengan kode **A06** yang berfungsi untuk memulai proses pengelompokan dan membuka jendela *loading*. Selain tombol, pada jendela ini juga terdapat lima *field* seperti yang tertera pada Tabel ??.

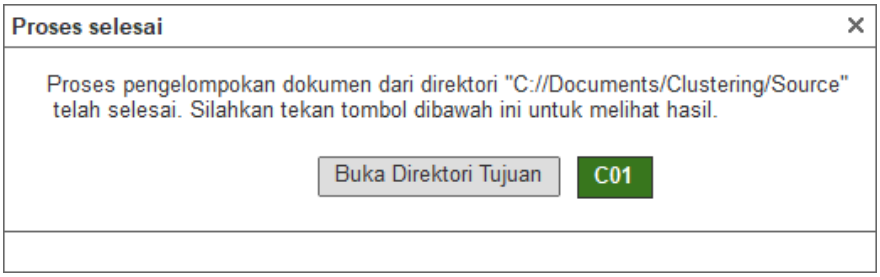
4.2.2 Jendela *Loading*



Gambar 4.13: Rancangan antarmuka jendela *loading*

Gambar 4.13 merupakan jendela yang akan muncul setelah pengguna menekan tombol "Mulai Pengelompokan". Jendela ini berisi informasi perkembangan proses pengelompokan. Informasi ini disajikan dalam bentuk *progress bar*. Hanya ada sebuah tombol pada jendela ini yaitu tombol dengan kode **B01**. Sesuai dengan labelnya, tombol ini berfungsi untuk membatalkan proses pengelompokan. Apabila tombol batal ditekan, maka pengguna akan dikembalikan ke jendela utama (Gambar 4.12).

4.2.3 Jendela Proses Berhasil



Gambar 4.14: Rancangan antarmuka jendela proses berhasil

Gambar 4.14 akan ditampilkan setelah proses pengelompokan berhasil, yaitu saat *progress bar* di jendela *loading* sudah terisi penuh. Jendela ini hanya memiliki satu buah tombol yaitu tombol dengan kode **C01** yang berfungsi untuk membuka *Windows Explorer* pada direktori hasil yang sudah dipilih pengguna pada jendela utama untuk menampilkan hasil dari proses pengelompokan.

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

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

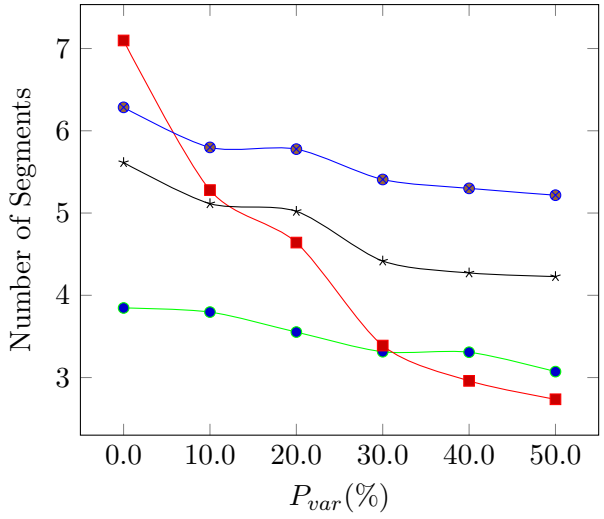
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

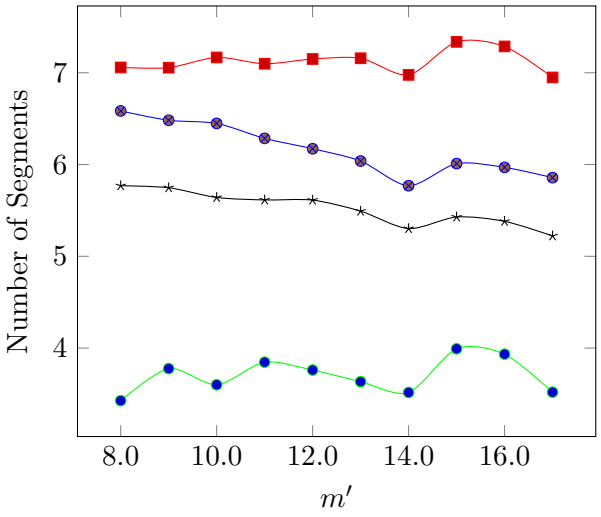
LAMPIRAN B

HASIL EKSPERIMEN

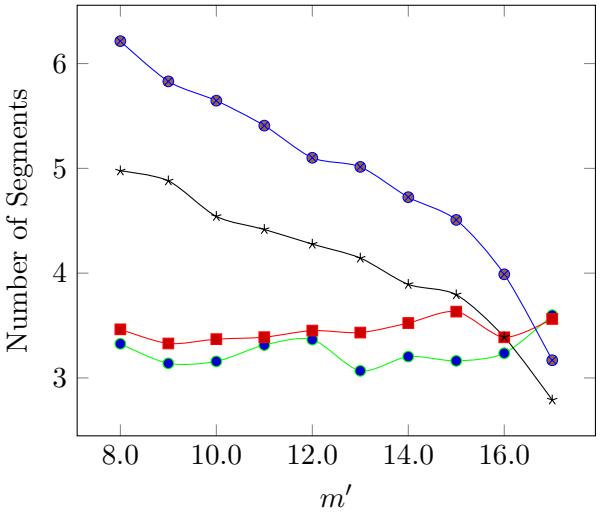
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



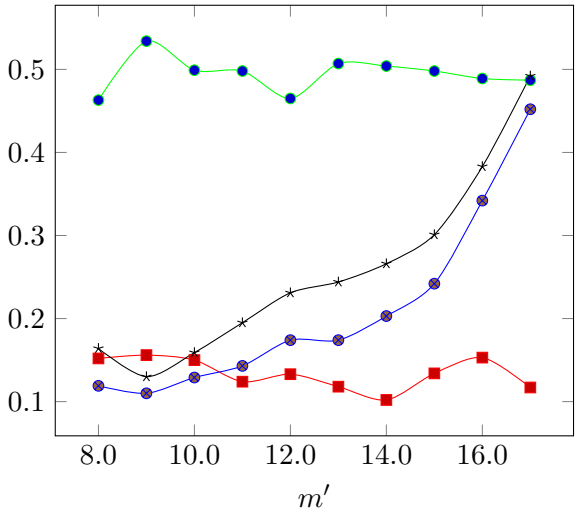
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4