Pengelompokan Dokumen Berbasis Algoritma Genetika

Cornelius David Herianto–2015730034

1 Data Skripsi

Pembimbing utama/tunggal: **Thomas Anung Basuki**
Pembimbing pendamping: -
Kode Topik : **TAB4502**
Topik ini sudah dikerjakan selama : **1** semester
Pengambilan pertama kali topik ini pada : Semester **45 - Ganjil 18/19**
Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**
Tipe Laporan : **C -** Dokumen pendukung untuk **pengambilan ke-2 di Skripsi 2**

2 Latar Belakang

Pengelompokan (*clustering*) merupakan teknik yang digunakan untuk membagi kumpulan data (*dataset*) ke dalam kelompok objek serupa. Setiap kelompok (*cluster*) terdiri dari objek-objek yang serupa satu sama lain berdasarkan suatu ukuran dan tidak serupa dengan objek-objek lain di luar kelompoknya.

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.

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.

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 berdasarkan prinsip *survival of the fittest*. Algoritma ini dinamakan demikian karena menggunakan konsep-konsep dalam genetika sebagai model pemecahan masalahnya.

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. Oleh karena itu, diperlukan fungsi *fitness* yang cocok sehingga GA dapat menghasilkan keluaran yang optimal. Pada masalah *clus-*

tering menggunakan GA, maka fungsi *fitness* yang digunakan harus bisa menggambarkan bahwa seluruh elemen sudah berada dalam *cluster* yang terbaik dan sudah sesuai.

3 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?

4 Tujuan

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.

5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencan kerja/laporan perkembangan terkahir :

1. **Melakukan studi literatur mengenai** *Information Retrieval* **(temu kembali informasi).**
   Status : Baru ditambahkan semester ini
   Hasil : Temu kembali informasi adalah bidang ilmu yang berurusan dengan representasi, penyimpanan, pengolahan, dan akses terhadap informasi. Pada penelitian ini, temu kembali informasi memiliki peran untuk mengubah dokumen yang semula berbentuk teks menjadi sesuatu yang memiliki nilai informasi agar nantinya bisa diproses lebih lanjut (dalam kasus ini dengan pengelompokan dokumen).

   Dalam penelitian ini, temu kembali informasi digunakan untuk merepresentasikan dokumen ke dalam model ruang vektor agar bisa dilakukan pengelompokan. Tahap pertama yang dilakukan adalah membentuk kosa kata (*vocabulary*) yang berisi seluruh istilah berbeda yang ada di setiap dokumen yang akan diindeks. Kemudian, untuk setiap dokumen akan dibentuk suatu indeks yang terdiri dari pasangan antara istilah di dokumen tersebut dan jumlah kemunculannya (*term-document incidence matrix*). Tabel 1 merupakan contoh dari *term-document incidence matrix* dengan baris pada tabel menunjukkan istilah (*term*) dan kolom menunjukkan nama dokumen. Sel bernilai 1 apabila dokumen mengandung term tertentu dan 0 jika tidak.

   Selanjutnya, setiap jumlah kemunculan suatu istilah dalam sebuah dokumen akan diubah menjadi suatu bobot tertentu berdasarkan teknik pemodelannya. Dalam penelitian ini, digunakan 2 teknik pembobotan yaitu bobot frekuensi dan bobot tf-idf.

   ### Bobot frekuensi

   Bobot frekuensi merupakan teknik pembobotan yang sangat sederhana karena bobotnya merupakan jumlah kemunculan istilah tersebut dalam dokumen. Bobot frekuensi dapat digambarkan dalam

rumus:

$$w_i = tf_i \tag{1}$$

dengan w_i merupakan bobot istilah ke-i dan tf_i merupakan frekuensi kemunculan istilah ke-i pada dokumen.

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Tabel 1: *Term-document incidence matrix*

Bobot TF-IDF

Pada bobot frekuensi, bobot hanya dihitung berdasarkan kemunculan istilah dalam dokumen itu sendiri. Namun dalam bobot TF-IDF (*term frequency-inverse document frequency*), bobot juga dihitung berdasarkan kemunculan istilah pada himpunan dokumen. Metode ini sangat populer digunakan oleh sistem rekomendasi berbasis teks. Rumus dari TF-IDF adalah sebagai berikut:

$$w_i = tf_i \times log\frac{N}{N_i} \tag{2}$$

dengan w_i merupakan bobot istilah ke-i, tf_i merupakan frekuensi kemunculan istilah ke-i pada dokumen, N menyatakan banyaknya anggota himpunan dokumen, dan N_i menunjukkan frekuensi dokumen dari istilah ke-i (banyaknya dokumen pada himpunan dokumen yang memuat istilah ke-i).

Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

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

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

Model ruang vektor

Model ruang vektor adalah representasi dari koleksi dokumen sebagai vektor dalam ruang vektor yang umum. Model ruang vektor ini biasanya digunakan dalam sejumlah operasi pencarian informasi mulai dari penilaian dokumen pada *query*, klasifikasi dokumen dan pengelompokan dokumen.

Dalam pengolahan model ruang vektor, dibutuhkan cara untuk menghitung kesamaan antara dua ruang vektor. Dalam pengelompokan dokumen, apabila kesamaan antara dua buah vektor semakin besar, maka peluang kedua vektor tersebut berada dalam sebuah kelompok yang sama akan semakin besar. Dalam penelitian ini, digunakan dua cara untuk menghitung kesamaan antara dua ruang

vektor yaitu menggunakan Jarak Euclidean (*Euclidean distance*) dan persamaan cosinus (*cosine similarity*).

Jarak Euclidean

Jarak Euclidean atau biasa disebut jarak garis lurus merupakan metode paling banyak digunakan untuk menghitung jarak antara dua buah vektor. Secara umum, rumus dari Jarak Euclidean adalah sebagai berikut:

$$d_{ij} = \sqrt{\sum_{v=1}^{N}(x_{vi} - x_{vj})^2} \tag{3}$$

dengan d_{ij} adalah jarak antara vektor ke-i dengan vektor ke-j, N adalah jumlah dimensi pada vektor, dan x_{vi} adalah nilai dimensi ke-v dari vektor ke-i.

Persamaan Cosinus

Persamaan cosinus merupakan normalisasi hasil kali titik dengan panjang masing-masing vektor. Persamaan cosinus memiliki persamaan sebagai berikut:

$$s_{ij} = \frac{i \cdot j}{\| i \| \times \| j \|} \tag{4}$$

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. **Melakukan studi literatur mengenai** *Document Clustering* **(pengelompokan dokumen).**
Status : Ada sejak rencana kerja skripsi
Hasil : Tugas pengelompokan (*clustering*) merupakan salah satu bagian dari pembelajaran mesin. Pengelompokan terdiri dari dua jenis berdasarkan metode pembelajarannya. *Classification* merupakan jenis pembelajaran terarah karena sudah diberikan label sejak awal, lalu dilakukan pengelompokan berdasarkan label untuk setiap kelompoknya. Sedangkan *clustering* merupakan jenis pembelajaran tak terarah karena tidak diberikan label sejak awal sehingga pengelompokan dilakukan berdasarkan suatu kesamaan tertentu.

K-Means

K-means merupakan algoritma pengelompokan yang paling populer digunakan saat ini. algoritma ini membagi data ke dalam K *cluster*. Setiap *cluster* direpresentasikan dengan titik tengahnya (*centroid*). Setiap iterasi, titik tengah akan dihitung sebagai rata-rata dari semua titik data dari *cluster* tersebut. Rumus untuk menghitung *centroid* adalah sebagai berikut:

$$\mu_i = \frac{1}{N_i}\sum_{q=1}^{N_i} x_q \tag{5}$$

dengan μ_i merupakan *centroid* ke-i, N_i merupakan jumlah titik data pada *cluster* ke-i, dan x_q merupakan titik ke-q pada *cluster* ke-i.

Algoritma K-means diawali dengan penentuan *centroid* secara acak. Pada setiap iterasi, setiap data ditetapkan pada *cluster* yang memiliki *centroid* dengan jarak terdekat dari titik data tersebut, kemudian posisi *centroid* dari setiap *cluster* akan dihitung ulang dengan Persamaan 5. Iterasi akan terus diulang sampai posisi dari semua *cluster* tidak berubah.

Algorithm 1 K-Means

Input: S (himpunan titik data), K (Jumlah *cluster*)
Output: himpunan *cluster*

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

3. **Melakukan studi literatur mengenai** *Genetic Algorithm* **(algoritma genetik).**
Status : Ada sejak rencana kerja skripsi
Hasil : Algoritma genetika (GA) adalah suatu algoritma pencarian yang terinspirasi dari proses seleksi alam yang terjadi secara alami dalam proses evolusi. Ada beberapa istilah yang digunakan dalam algoritma genetika di antaranya kromosom, seleksi, persilangan, mutasi, dan fungsi *fitness*.

Kromosom

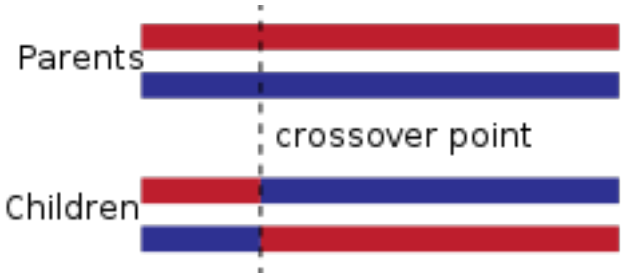
Dalam GA, kromosom adalah himpunan parameter yang mendefinisikan suatu solusi yang diusulkan. Kromosom biasanya direpresentasikan sebagai string yang berisi kumpulan nilai (gen), meskipun berbagai struktur data lainnya juga digunakan.

Seleksi

Seleksi dalam algoritma genetika bertugas untuk memilih kromosom dari populasi untuk proses persilangan. Salah satu teknik yang populer digunakan dalam seleksi adalah *roulette-wheel selection*. Teknik ini memungkinkan terjadinya pemilihan berdasarkan fungsi *fitness* sehingga kromosom yang memiliki fungsi *fitness* lebih besar memiliki peluang lebih besar untuk terpilih dan menjadi induk dari generasi selanjutnya.

Persilangan

Persilangan adalah operasi genetik yang digunakan untuk menggabungkan informasi genetik dari dua induk untuk menghasilkan keturunan baru. Teknik persilangan yang digunakan dalam penelitian ini adalah *Single-point crossover*. Dalam teknik ini, sebuah titik pada kedua induk dipilih untuk menjadi titik persilangan (*crossover point*). Bit yang berada di sebelah kanan titik persilangan bertukar antara kedua kromosom induk seperti yang ditunjukkan pada Gambar 1.



Gambar 1: *Single-point crossover*

Mutasi

Mutasi adalah suatu operator genetik yang digunakan untuk mempertahankan keragaman genetik dari satu generasi populasi dalam algoritma genetika. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi terjadi berdasarkan probabilitas mutasi yang sudah ditentukan sebelumnya. Probabilitas ini seharusnya bernilai kecil, karena jika terlalu besar maka akan menjadi sama dengan algoritma pencarian acak primitif (*primitive random search*). Pada penelitian ini, kromosom memiliki gen yang bernilai non-biner. Oleh karena itu, mutasi dilakukan dengan memilih sebuah bilangan acak antara batas bawah dan batas atas nilai suatu gen.

Fungsi *Fitness*

Fungsi *fitness* adalah fungsi objektif yang digunakan untuk mengukur seberapa baik suatu kromosom sebagai calon solusi. Sebuah fungsi *fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan solusi yang optimal. Dalam algoritma genetika, fungsi *fitness* juga digunakan dalam proses seleksi (*roulette-wheel selection*) untuk menentukan seberapa baik suatu kromosom untuk menjadi induk dari generasi berikutnya.

Algorithm 2 Algoritma Genetika

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

inputs: *populasi*, himpunan individu
Fung-Fitness, fungsi yang mengukur *fitness* dari tiap individu

```
repeat
  populasi_baru ← himpunan kosong
  for i=1 to Size(populasi) do
    x ← Seleksi-acak(populasi, Fung-Fitness)
    y ← Seleksi-acak(populasi, Fung-Fitness)
    anak ← Persilangan(x, y)
    if probabilitas mutasi then
      Mutasi(anak)
    end if
    tambahkan anak ke populasi_baru
  end for
  populasi ← populasi_baru
until kriteria berakhir dipenuhi atau sampai jumlah generasi tertentu
return individu terbaik dalam populasi, berdasarkan Fung-Fitness
```

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

inputs: x, y, individu induk

```
n ← Length(x)
c ← angka acak antara 1 sampai n
return Append(Substring(x,1,c), Substring(y,c+1,n))
```

Proses pencarian dalam algoritma genetika

Seperti yang disebutkan dalam Algoritma 2, algoritma genetika dimulai dengan menginisialisasi suatu populasi (biasanya dibangkitkan secara acak jika tidak diketahui heuristik masalahnya). Lalu, akan dibangkitkan populasi baru untuk generasi berikutnya dengan cara mengambil dua kromosom secara acak dengan teknik *roulette-wheel selection*. Setelah itu akan dilakukan persilangan dengan

teknik *single-point crossover*. Teknik ini dilakukan dengan cara menentukan sebuah titik potong c yang diambil secara acak antara angka 1 sampai panjang kromosom n. Keturunan dari kedua induk tersebut akan memiliki kromosom induk pertama dari gen ke-1 sampai gen ke-c dan dari induk kedua mulai dari gen ke-(c+1) sampai gen ke-n. Setelah itu, untuk apabila terjadi mutasi, maka salah satu gen dari anak akan diubah nilainya. Iterasi ini akan dilakukan terus-menerus hingga kriteria berhenti tercapai atau sudah mencapai batas jumlah generasi tertentu.

4. **Mempelajari penggunaan algoritma genetik dalam pengelompokan dokumen.**
   Status : Ada sejak rencana kerja skripsi
   Hasil : Meskipun pada umumnya algoritma *K-means* digunakan dalam pengelompokan, tetapi ternyata algoritma *K-means* masih memiliki kekurangan yaitu masih dapat terjebak pada *local optimum*. Oleh karena itu, pada penelitian ini digunakan algoritma genetika sebagai solusi dari permasalahan *local optimum*. Algoritma genetika memiliki kemampuan untuk menghindari terjadinya *local optimum* karena algoritma genetika merupakan algoritma pencarian yang bersifat stokastik karena memperbolehkan terjadinya variasi acak dalam proses pencarian solusinya. Oleh karena itu, diharapkan pengelompokan berbasis GA dapat menghasilkan solusi yang lebih baik dibandingkan dengan pengelompokan pada umumnya yang menggunakan algoritma *K-means*.

5. **Mencari dokumen yang akan dijadikan *training* dan *test datasets*. Status :** Ada sejak rencana kerja skripsi
   Hasil : Belum dikerjakan.

6. **Melakukan analisis masalah dan mencari solusinya.**
   Status : Baru ditambahkan semester ini
   Hasil : Untuk menyelesaikan masalah pengelompokan dokumen menggunakan algoritma genetika, maka perlu dibangun sebuah model yang dapat diterapkan ke dalam algoritma tersebut.

Representasi Kromosom

Setiap *string* kromosom merupakan deretan bilangan riil yang merepresentasikan K titik pusat *cluster* (*centroid*). Dalam ruang N dimensi, panjang dari kromosom akan menjadi $N \times K$ gen. N kata pertama merepresentasikan N dimensi dari *centroid* pertama, N kata selanjutnya merepresentasikan N dimensi dari *centroid* kedua, dan seterusnya.

Fungsi *Fitness*

Perhitungan *fitness* dalam penelitian ini terdiri dari dua tahap. Pada tahap pertama, terjadi pembentukan *cluster* berdasarkan titik pusat yang terkandung dalam kromosom. Hal ini dilakukan dengan menetapkan setiap titik $x_i, i = 1, 2, ..., n$ ke dalam sebuah *cluster* C_j dengan *centroid* z_j sehingga

$$\|x_i - z_j\| < \|x_i - z_p\|, p = 1, 2, ..., K, \text{ dan } p \neq j. \tag{6}$$

Setelah proses pengelompokan selesai, titik pusat yang terkandung dalam kromosom diganti dengan rata-rata titik dari tiap *cluster*. Dengan kata lain, untuk *cluster* C_i, *centroid* baru z_i^* dapat dihitung dengan rumus:

$$z_i^* = \frac{1}{n_i} \sum_{x_j \in C_i} x_j, i = 1, 2, ..., K. \tag{7}$$

dengan z_i^* merupakan titik pusat *cluster* ke-i, n_i merupakan jumlah anggota *cluster* ke-i, dan x_j merupakan titik ke-j yang merupakan anggota dari *cluster* ke-i. z_i^* ini akan menggantikan z_i sebelumnya di kromosom. Ada dua metode perhitungan fungsi fitness yang diimplementasikan dalam penelitian ini yaitu *euclidean distance* dan *cosine similarity*.

Euclidean Distance

Pada perhitungan dengan *euclidean distance*, akan dihitung sebuah *clustering metric* M dengan rumus:

$$\begin{aligned} M &= \sum_{i=1}^{K} M_i, \\ M_i &= \sum_{x_j \in C_i} \parallel x_j - z_i \parallel \end{aligned} \tag{8}$$

Lalu, fungsi *fitness* akan didefinisikan sebagai $f = 1/M$, sehingga maksimalisasi terhadap nilai f akan meminimalkan nilai M.

Cosine Similarity

Perhitungan *fitness* menggunakan *cosine similarity* dapat dilakukan dengan rumus:

$$\begin{aligned} f &= \sum_{i=1}^{K} f_i, \\ f_i &= \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\parallel x_j \parallel \times \parallel z_i \parallel} \end{aligned} \tag{9}$$

semakin besar nilai dari fungsi *fitness* f, maka kromosom tersebut semakin mendekati solusi yang optimal.

Operasi Genetik

Ada beberapa operasi genetik yang akan dibahas, di antaranya: inisialisasi populasi, seleksi, persilangan, dan mutasi.

Inisialisasi Populasi

K *centroid* yang terkandung dalam kromosom pada mulanya dipilih secara acak sebanyak K titik dari keseluruhan himpunan data. Lalu proses ini diulang sebanyak P kali di mana P merupakan ukuran populasi yang diinginkan.

Seleksi

Proses seleksi ini terjadi berdasarkan konsep *survival of the fittest* yang diadaptasi dari sistem genetika alami. Dalam penelitian ini, calon induk dari generasi selanjutnya dipilih dengan menggunakan teknik *roulette-wheel selection* yang merupakan salah satu teknik yang digunakan karena mengimplementasikan *proportional selection strategy*.

Persilangan

Persilangan dalam penelitian ini terjadi terhadap dua induk dengan satu titik potong. Misalkan kromosom memiliki panjang l, sebuah angka acak akan diambil sebagai titik potong dalam batas $[1,l-1]$. Bagian kromosom sebelah kanan titik potong akan ditukar antara kedua induk sehingga menghasilkan dua individu keturunan.

Mutasi

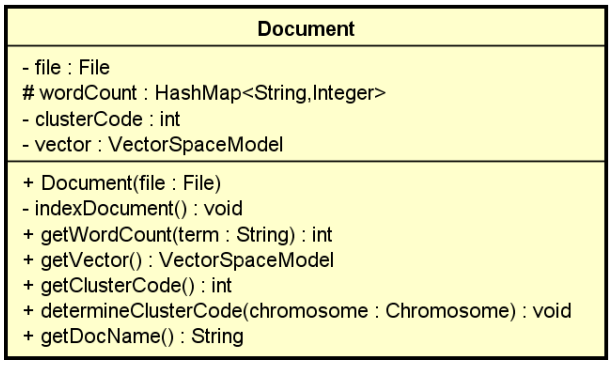
Setiap kromosom mengalami mutasi dengan probabilitas mutasi tetap μ_c. Setelah itu, ditentukan gen mana yang akan mengalami mutasi dengan mengambilnya secara acak. Perubahan nilai gen tersebut juga merupakan angka acak yang diambil mulai dari angka 0 sampai dengan jumlah kemunculan kata keseluruhan dalam *vocabulary*.

7. **Membuat rancangan perangkat lunak yang menggunakan algoritma genetik sebagai algoritma pengelompokan dokumen.**
   Status : Ada sejak rencana kerja skripsi
   Hasil : Ada dua rancangan yang sudah dibuat yaitu rancangan kelas dan rancangan antarmuka pengguna.

Rancangan Kelas

Berdasarkan hasil analisis dari masalah yang dihadapi, dibentuklah diagram kelas pada gambar 12 sebagai gambaran dari perangkat lunak yang akan dibuat.

Document



Gambar 2: Kelas *Document*

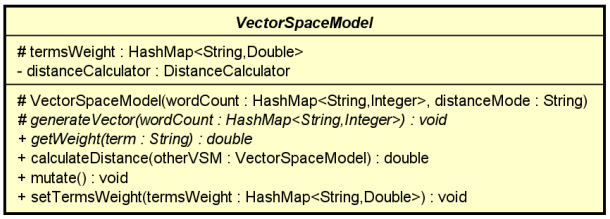
Kelas ini merupakan representasi dari dokumen yang akan diproses dalam pengelompokan. Kelas ini berfungsi untuk menyimpan informasi yang dibutuhkan dari sebuah dokumen selama proses pengelompokan. Atribut yang dimiliki oleh kelas *Document* adalah:

- *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.

VectorSpaceModel



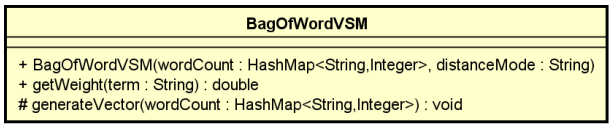
Gambar 3: Kelas *VectorSpaceModel*

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*.

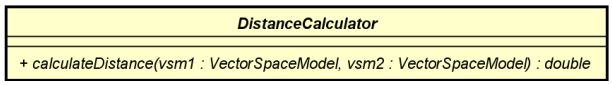


Gambar 4: Kelas *BagOfWordVSM*

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.

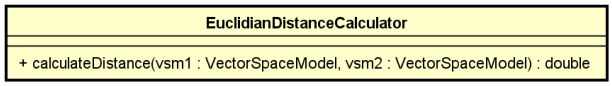
DistanceCalculator



Gambar 5: 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.

EuclideanDistanceCalculator



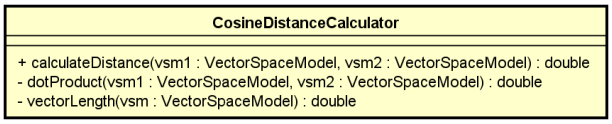
Gambar 6: Kelas *EuclideanDistanceCalculator*

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 1).

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:

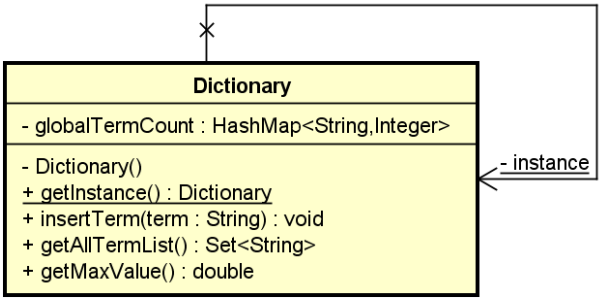
- *calculateDistance*: merupakan *method* yang diturunkan dari kelas *DistanceCalculator*. *Method* ini mengembalikan jarak dari *vsm1* dan *vsm2* yang dihitung menggunakan persamaan cosinus (Subbab 1).



Gambar 7: Kelas *CosineDistanceCalculator*

- *dotProduct*: berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vsm1* dan *vsm 2*.
- *vectorLength*: berfungsi untuk menghitung panjang dari vektor *vsm*.

Dictionary



Gambar 8: 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:

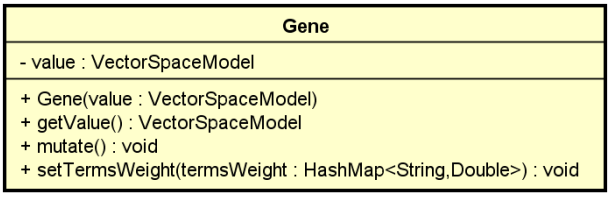
- *globalTermCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini berfungsi untuk menyimpan seluruh istilah yang muncul dan banyak kemunculannya dalam keseluruhan koleksi dokumen.
- *instance*: merupakan objek bertipe *Dictionary* sebagai instansiasi satu-satunya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

Gene

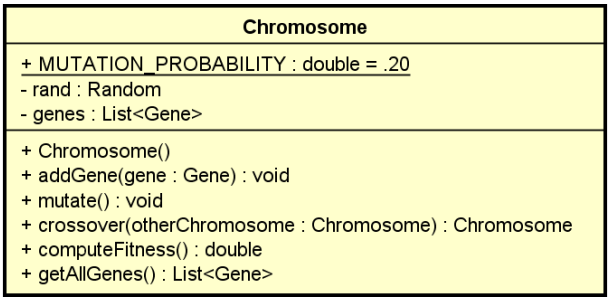
Kelas ini merepresentasikan gen dalam algoritma genetika. Kelas ini hanya memiliki sebuah atribut *value* bertipe *VectorSpaceModel*. Atribut ini menyimpan model ruang vektor yang menjadi titik pusat *cluster* (*centroid*). *Method* yang ada pada kelas ini adalah:



Gambar 9: 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*.

Chromosome



Gambar 10: Kelas *Chromosome*

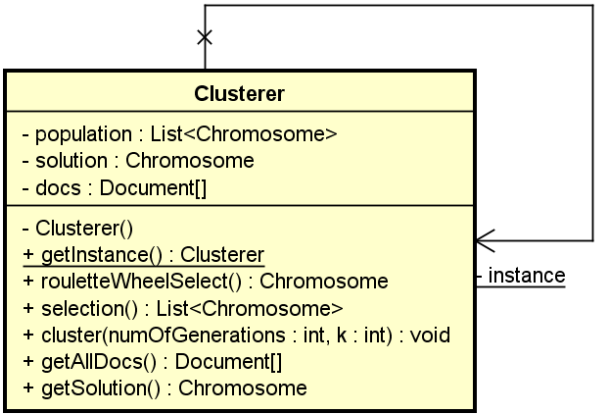
Kelas ini merepresentasikan kromosom dalam algoritma genetika (Subbab 3). Atribut yang terdapat dalam kelas ini adalah:

- *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 3).
- *crossover*: bertugas untuk melakukan persilangan dengan kromosom lain untuk mengahsilkan keturunan (Subbab 3).
- *computeFitness*: mengembalikan nilai *fitness* dari kromosom (Subbab 3).
- *getAllGenes*: merupakan *getter* dari atribut *genes*.

Clusterer



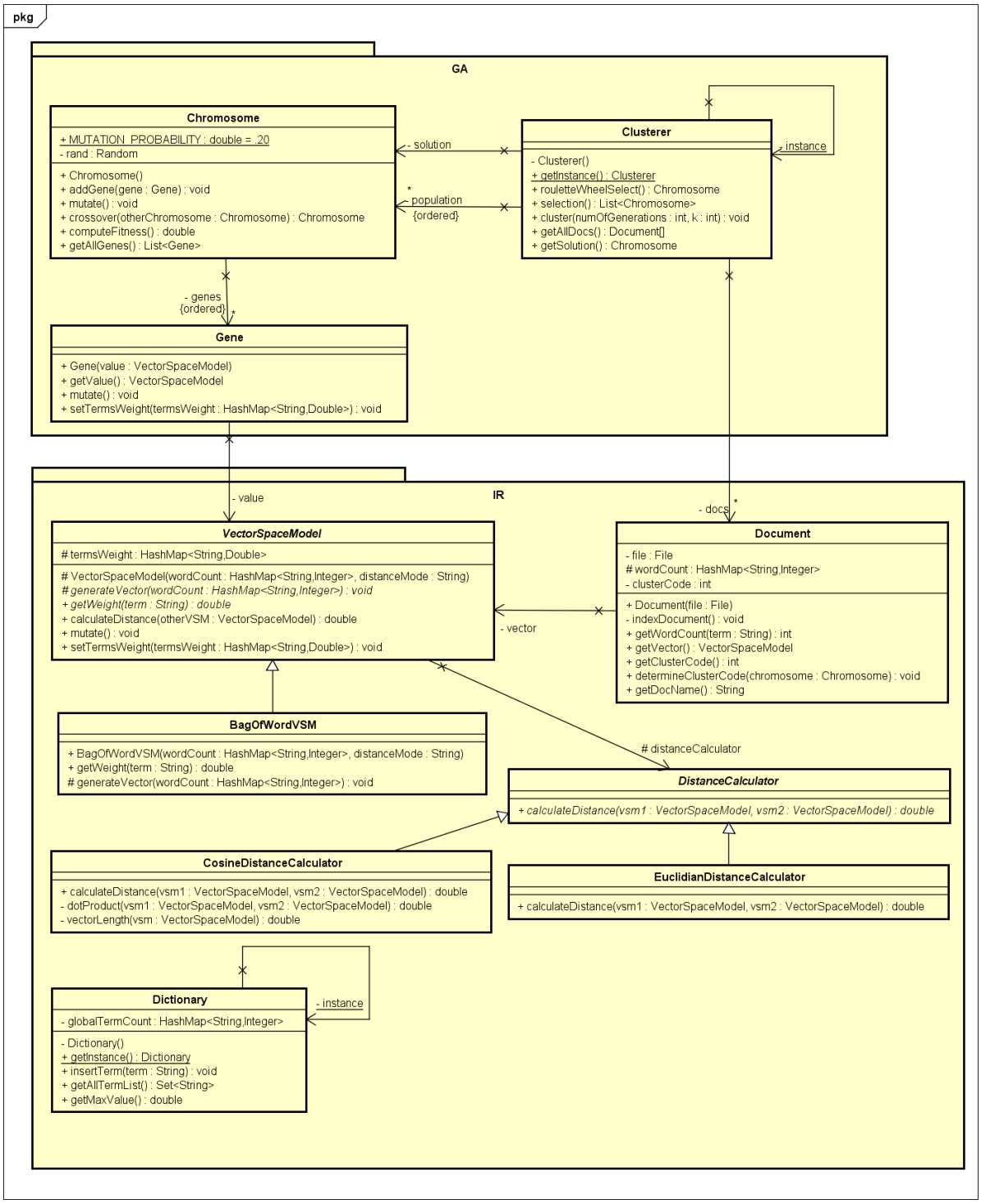
Gambar 11: Kelas *Clusterer*

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*.



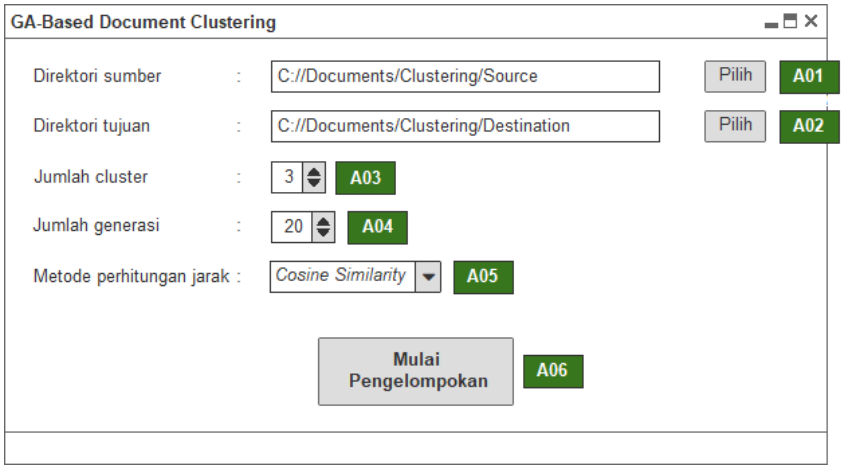
Gambar 12: Diagram kelas pada tahap perancangan

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.

Jendela Utama

Gambar 13 merupakan jendela yang pertama kali ditampilkan saat perangkat lunak dijalankan. Jendela tersebut berisi berbagai macam hal yang dibutuhkan penguna dalam melakukan pengelompokan dokumen.



Gambar 13: Rancangan antarmuka jendela utama

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

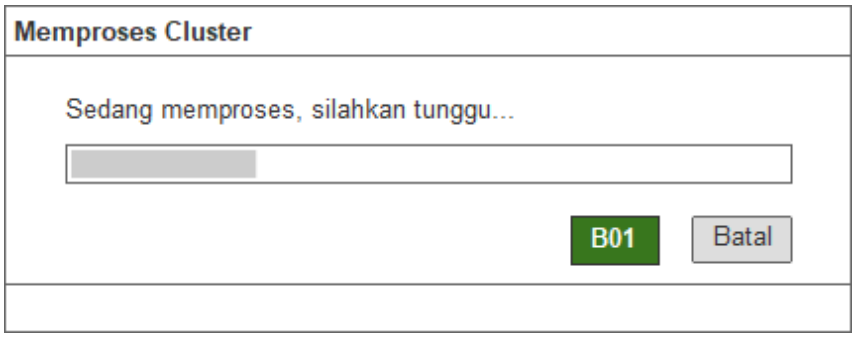
Kode	Nama	Jenis	*Defaut value*	Wajib	Aturan validasi
A01	Direktori sumber	*file chooser*	-	ya	Harus berupa direktori yang berisi dokumen (tidak boleh kosong)
A02	Direktori tujuan	*file chooser*	-	ya	Harus berupa direktori yang kosong
A03	Jumlah cluster	*spinner*	2	ya	Nilai minimum 2
A04	Jumlah generasi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

Tabel 2: Rincian *field* pada jendela utama

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 7.

[1] https://balsamiq.com/

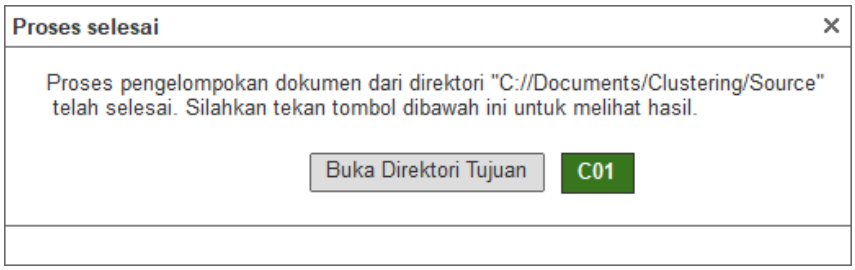
Jendela *Loading*



Gambar 14: Rancangan antarmuka jendela *loading*

Gambar 14 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 13).

Jendela Proses Berhasil



Gambar 15: Rancangan antarmuka jendela proses berhasil

Gambar 15 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.

8. **Mengimplementasikan hasil rancangan menjadi perangkat lunak.**
   Status : Ada sejak rencana kerja skripsi
   Hasil : Hasil implementasi dilampirkan.

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Melakukan studi literatur mengenai *Information Retrieval* (temu kembali informasi).
2. Melakukan studi literatur mengenai *Document Clustering* (pengelompokan dokumen).

3. Melakukan studi literatur mengenai *Genetic Algorithm* (algoritma genetik).
4. Mempelajari penggunaan algoritma genetik dalam pengelompokan dokumen.
5. Melakukan analisis masalah dan mencari solusinya.
6. Membuat rancangan perangkat lunak yang menggunakan algoritma genetik sebagai algoritma pengelompokan dokumen.
7. Mengimplementasikan hasil rancangan menjadi perangkat lunak.

7 Kendala yang Dihadapi

Kendala - kendala yang dihadapi selama mengerjakan skripsi :

- Skripsi 1 diambil bersamaan dengan Skripsi 2 sehingga waktu yang tersedia sangat terbatas sedangkan banyak hal yang perlu dikerjakan.
- Ada beberapa tugas besar dari mata kuliah lain yang menyita cukup banyak waktu sehingga mengurangi waktu untuk mengerjakan skripsi.

Bandung, 15/11/2018

Cornelius David Herianto

Menyetujui,

Nama: Thomas Anung Basuki
Pembimbing Tunggal

Lampiran Kode Program

Listing 1: Chromosome.java

```java
package GA;


import IR.Dictionary;
import IR.Document;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author CorneliusDavid
 */
public class Chromosome {
    public static final double MUTATION_PROBABILITY=.20;
    private List<Gene> genes;
    private Random rand;

    public Chromosome() {
        this.genes = new LinkedList<>();
        this.rand = new Random();
//        this.fitnessValue=-1;
    }

    public void addGene(Gene gene){
        this.genes.add(gene);
    }

    public void mutate(){
        if(rand.nextDouble()<MUTATION_PROBABILITY){
            int mutatedGeneIdx=rand.nextInt(genes.size());
            genes.get(mutatedGeneIdx).mutate();
        }
    }

    public Chromosome crossover(Chromosome otherChromosome){
        int breakPoint=rand.nextInt(genes.size());
        Chromosome result=new Chromosome();
        for (int i = 0; i < breakPoint; i++) {
            result.addGene(this.genes.get(i));
        }

        for (int i = breakPoint; i < this.genes.size(); i++) {
            result.addGene(otherChromosome.genes.get(i));
        }

        return result;
    }

    public double computeFitness(){
//        if(fitnessValue!=-1)return fitnessValue;
        int fitnessValue=0;
        Document docs[]=Clusterer.getInstance().getAllDocs();
        HashMap<String, Double> sumOfCentroid[]=new HashMap[genes.size()];
        for (int i = 0; i < genes.size(); i++) {
            sumOfCentroid[i]=new HashMap<>();
        }
        int pointCount[]=new int[genes.size()];
        for (int i = 0; i < docs.length; i++) {
            docs[i].determineClusterCode(this);
            int clusterCode=docs[i].getClusterCode();
            double tmp=docs[i].getVector().calculateDistance(this.genes.get(clusterCode).getValue());
            fitnessValue+=tmp;
            System.out.println(tmp);
            for(String term:Dictionary.getInstance().getAllTermList()){
                double nextValue=sumOfCentroid[clusterCode].containsKey(term)?sumOfCentroid[clusterCode].get(term):0;
                nextValue+=tmp;
                sumOfCentroid[clusterCode].put(term, nextValue);
            }
            pointCount[clusterCode]++;
        }

        for (int i = 0; i < genes.size(); i++) {
            for(String term:Dictionary.getInstance().getAllTermList()){
                double nextValue=sumOfCentroid[i].containsKey(term)?sumOfCentroid[i].get(term):0;
                nextValue=pointCount[i]==0.0?0.0:nextValue/pointCount[i];
                sumOfCentroid[i].put(term, nextValue);
            }
            System.out.println(sumOfCentroid[i]);
            genes.get(i).setTermsWeight(sumOfCentroid[i]);
        }
```

```
90          return fitnessValue;
91      }
92
93      public List<Gene> getAllGenes() {
94          return genes;
95      }
96
97
98  }
```

Listing 2: Clusterer.java

```
1   package GA;
2
3   import IR.Document;
4   import java.io.File;
5   import java.util.Arrays;
6   import java.util.Collections;
7   import java.util.LinkedList;
8   import java.util.List;
9   import java.util.Random;
10
11  /*
12   * To change this license header, choose License Headers in Project Properties.
13   * To change this template file, choose Tools | Templates
14   * and open the template in the editor.
15   */
16  /**
17   *
18   * @author CorneliusDavid
19   */
20  public class Clusterer {
21
22      private List<Chromosome> population;
23      private Chromosome solution;
24      private Document docs[];
25      private static Clusterer instance;
26
27      private Clusterer() {
28          population = new LinkedList<>();
29      }
30
31      public static Clusterer getInstance() {
32          if (instance == null) {
33              instance = new Clusterer();
34          }
35          return instance;
36      }
37
38      public Chromosome rouletteWheelSelect() {
39          double totalWeight = 0;
40          for (int i = 0; i < population.size(); i++) {
41              totalWeight += population.get(i).computeFitness();
42          }
43
44          double selectedValue = (new Random()).nextDouble() * totalWeight;
45
46          for (int i = 0; i < population.size(); i++) {
47              selectedValue -= population.get(i).computeFitness();
48              if (selectedValue <= 0) {
49                  return population.get(i);
50              }
51          }
52          return population.get(population.size() - 1);
53      }
54
55      public List<Chromosome> selection() {
56          List<Chromosome> nextGen = new LinkedList<>();
57          for (int i = 0; i < population.size(); i++) {
58              Chromosome temp = this.rouletteWheelSelect();
59              nextGen.add(temp);
60          }
61
62          return nextGen;
63      }
64
65      public void cluster(int numOfGenerations, int k) {
66          int popSize = 10;
67
68          //document indexing
69          File folder = new File("testdoc");
70          File[] files = folder.listFiles();
71          docs = new Document[files.length];
72          for (int i = 0; i < docs.length; i++) {
73              docs[i] = new Document(files[i]);
74          }
75
76          List<Document> list = Arrays.asList(docs);
77
78          //generate initial population
79          for (int i = 0; i < popSize; i++) {
80              Chromosome temp = new Chromosome();
81              Collections.shuffle(list);
82              for (int j = 0; j < k; j++) {
```

```
                temp.addGene(new Gene(list.get(j).getVector()));
            }
            population.add(temp);
        }


        for (int i = 0; i < numOfGenerations; i++) {
            //selection
            population = selection();
            for (int j = 0; j < popSize; j++) {
                Chromosome temp=population.get(j);
                if (solution == null || solution.computeFitness()< temp.computeFitness()) {
                    solution = temp;
                }
            }
//            System.out.println(solution.computeFitness());
            //mutation
            for (Chromosome c : population) {
                c.mutate();
            }
        }
    }

    public Document[] getAllDocs() {
        return docs;
    }

    public Chromosome getSolution() {
        return solution;
    }
}
```

Listing 3: Gene.java

```
package GA;

import IR.VectorSpaceModel;
import java.util.HashMap;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author CorneliusDavid
 */
public class Gene {
    private VectorSpaceModel value;

    public Gene(VectorSpaceModel value) {
        this.value = value;
    }

    public VectorSpaceModel getValue() {
        return value;
    }

    public void mutate(){
        value.mutate();
    }

    @Override
    public String toString() {
        return value.toString();
    }

    public void setTermsWeight(HashMap<String, Double> termsWeight) {
        this.value.setTermsWeight(termsWeight);
    }
}
```

Listing 4: BagOfWordVSM.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package IR;

import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Set;
/**
 *
 * @author CorneliusDavid
 */
public class BagOfWordVSM extends VectorSpaceModel {

```

```
    public BagOfWordVSM(HashMap<String, Integer> wordCount, String distanceMode) {
        super(wordCount, distanceMode);
    }

    @Override
    public double getWeight(String term) {
        if(!termsWeight.containsKey(term)) return 0.0;
        return termsWeight.get(term);
    }


    @Override
    protected void generateVector(HashMap<String, Integer> wordCount) {
        Set<Entry<String,Integer>> entrySet=wordCount.entrySet();
        for(Entry<String,Integer> entry:entrySet){
            this.termsWeight.put(entry.getKey(), entry.getValue().doubleValue());
        }
    }

}
```

Listing 5: CosineDistanceCalculator.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package IR;

import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author CorneliusDavid
 */
public class CosineDistanceCalculator extends DistanceCalculator{

    @Override
    public double calculateDistance(VectorSpaceModel vsm1, VectorSpaceModel vsm2) {
//        System.out.println("l1 "+vectorLength(vsm1));
//        System.out.println("l2 "+vectorLength(vsm2));
//        System.out.println("dot "+dotProduct(vsm1, vsm2));
        if(vectorLength(vsm1)==0 || vectorLength(vsm2)==0){
            System.out.println("asdasad");
            return 0;
        }
        return dotProduct(vsm1, vsm2)/(vectorLength(vsm1)*vectorLength(vsm2));
    }

    private double dotProduct(VectorSpaceModel vsm1, VectorSpaceModel vsm2){
        Set<String> terms=Dictionary.getInstance().getAllTermList();
        Iterator<String> it=terms.iterator();
        double result=0;
        while(it.hasNext()){
            String term=it.next();
            double weight1=vsm1.getWeight(term);
            double weight2=vsm2.getWeight(term);
            result+=(weight1*weight2);
        }
        return result;
    }

    private double vectorLength(VectorSpaceModel vsm){
        Set<String> terms=Dictionary.getInstance().getAllTermList();
        Iterator<String> it=terms.iterator();
        double result=0;
        while(it.hasNext()){
            String term=it.next();
            double weight=vsm.getWeight(term);
            result+=(weight*weight);
        }
        return Math.sqrt(result);
    }
}
```

Listing 6: Dictionary.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this termlate file, choose Tools | Templates
 * and open the termlate in the editor.
 */
package IR;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map.Entry;
import java.util.Set;
```

```java

/**
 *
 * @author CorneliusDavid
 */
public class Dictionary {

    private HashMap<String, Integer> globalTermCount;
    private HashSet<String> stopwords;
    private static Dictionary instance;

    private Dictionary() {
        this.globalTermCount = new HashMap<>();
        this.stopwords=new HashSet<>();
        this.loadStopwords();
    }

    private void loadStopwords(){
        try {
            BufferedReader br=new BufferedReader(new FileReader("stopword-list.txt"));
            String tmp=br.readLine();
            while(tmp!=null && tmp.length()>0){
                this.stopwords.add(tmp);
                tmp=br.readLine();
            }
        } catch (Exception ex) {}
    }

    public static Dictionary getInstance() {
        if (instance == null) {
            instance = new Dictionary();
        }
        return instance;
    }

    /**
     * memasukkan term ke dalam
     *
     * @param term String yang akan dimasukkan
     */
    public void insertTerm(String term) {
        if (!globalTermCount.containsKey(term)) {
            globalTermCount.put(term, 0);
        }
        globalTermCount.put(term, globalTermCount.get(term) + 1);
    }

    public Set<String> getAllTermList(){
        return this.globalTermCount.keySet();
    }

    public double getMaxValue(){
        double max=0;
        String str="";
        for(Entry<String,Integer> x:globalTermCount.entrySet()){
            if(max<x.getValue()){
                max=x.getValue();
                str=x.getKey();
            }
        }
        return max;
    }

    public boolean isStopWord(String term){
        return stopwords.contains(term);
    }
}
```

Listing 7: DistanceCalculator.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package IR;

/**
 *
 * @author CorneliusDavid
 */
public abstract class DistanceCalculator {
    public abstract double calculateDistance(VectorSpaceModel vsm1,VectorSpaceModel vsm2);
}
```

Listing 8: Document.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
```

```java
package IR;

import GA.Chromosome;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Scanner;

/**
 *
 * @author CorneliusDavid
 */
public class Document {
    private File file;
    protected HashMap<String,Integer> wordCount;
    private VectorSpaceModel vector;
    private int clusterCode;

    public Document(File file){
        this.file=file;
        wordCount=new HashMap<>();

        try {
            this.indexDocument();
        } catch (FileNotFoundException ex) {}


        this.vector=new BagOfWordVSM(wordCount,"COS");//TODO: ganti param
    }

    private void indexDocument() throws FileNotFoundException {


        Scanner sc=new Scanner(new InputStreamReader(new FileInputStream(file)));
//        StemmerIndo stemmer=new StemmerIndo();
        while(sc.hasNext())
        {
            String temp=sc.next();
            temp=temp.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
            if(temp.length()==0)continue;
//              if(ClusteringConfig.getInstance().USE_STEMMER){
//                  temp=stemmer.getRootWord(temp);
//              }
            if(ClusteringConfig.getInstance().STOPWORD_REMOVAL){
                if(Dictionary.getInstance().isStopWord(temp)){
                    continue;
                }
            }
            Dictionary.getInstance().insertTerm(temp);
            if (!wordCount.containsKey(temp)) {
                wordCount.put(temp, 0);
            }
            wordCount.put(temp, wordCount.get(temp) + 1);
        }
    }

    public int getWordCount(String term) {
        if(!wordCount.containsKey(term)){
            return 0;
        }
        return wordCount.get(term).intValue();
    }

    @Override
    public String toString() {
        return vector.toString();
    }

    public VectorSpaceModel getVector() {
        return vector;
    }

    public int getClusterCode() {
        return clusterCode;
    }

    public void determineClusterCode(Chromosome chromosome) {
        this.clusterCode=0;
        double distance=Double.MAX_VALUE;
        for (int i = 0; i < chromosome.getAllGenes().size(); i++) {
            double tempDist=chromosome.getAllGenes().get(i).getValue().calculateDistance(vector);
            if(tempDist<distance){
                distance=tempDist;
                clusterCode=i;
            }
        }
    }

    public String getDocName(){
        return file.getName();
    }
}
```

Listing 9: EuclidianDistanceCalculator.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package IR;

import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author CorneliusDavid
 */
public class EuclidianDistanceCalculator extends DistanceCalculator{

    @Override
    public double calculateDistance(VectorSpaceModel vsm1, VectorSpaceModel vsm2) {
        Set<String> terms=Dictionary.getInstance().getAllTermList();
        Iterator<String> it=terms.iterator();
        double result=0;
        while(it.hasNext()){
            String term=it.next();
            double weight1=vsm1.getWeight(term);
            double weight2=vsm2.getWeight(term);
            result+=((weight1-weight2)*(weight1-weight2));
        }
        return Math.sqrt(result);
    }

}
```

Listing 10: VectorSpaceModel.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package IR;

import java.util.HashMap;
import java.util.Random;

/**
 *
 * @author CorneliusDavid
 */
public abstract class VectorSpaceModel {

    protected HashMap<String, Double> termsWeight;
    protected DistanceCalculator distanceCalculator;

    protected VectorSpaceModel(HashMap<String, Integer> wordCount, String distanceMode) {
        termsWeight = new HashMap<>();

        switch (distanceMode) {
            case "EUC":
                distanceCalculator = new EuclidianDistanceCalculator();
                break;
            case "COS":
                distanceCalculator = new CosineDistanceCalculator();
                break;
        }

        generateVector(wordCount);
    }

    protected abstract void generateVector(HashMap<String, Integer> wordCount);

    public abstract double getWeight(String term);

    public double calculateDistance(VectorSpaceModel otherVSM) {
        return distanceCalculator.calculateDistance(this, otherVSM);
    }

    @Override
    public String toString() {
        return termsWeight.toString();
    }

    public void mutate(){
        Random rand=new Random();
        String key=(String) termsWeight.keySet().toArray()[rand.nextInt(termsWeight.size())];
        termsWeight.put(key, rand.nextDouble()*termsWeight.get(key)*2);//mutasi dari 0 - nilai sendiri dikali 2
    }

    public void setTermsWeight(HashMap<String, Double> termsWeight) {
        this.termsWeight = termsWeight;
    }


}
```