IMPLEMENTASI METODE *GENERALIZED VECTOR SPACE MODEL* (GVSM) MENGGUNAKAN ALGORITMA LESK PADA SISTEM TEMU KEMBALI

Abdurrahman Auliya Fatahillah[1], Ednawati Rainarli, S.Si., M.Si.[2]

[1,2] Program Studi Teknik Informatika
Fakultas Teknik dan Ilmu Komputer – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
E-mail : thanxs.rahman@gmail.com[1], irene_edna@yahoo.com[2]

ABSTRAK

Sistem Temu Kembali (*Information Retrieval System*) dirancang untuk menemukan data (*file*) atau informasi yang diperlukan. Sistem Temu Kembali (*Information Retrieval System*) bertujuan untuk menjembatani kebutuhan informasi dengan sumber informasi yang tersedia secara relevan. *Generalized Vector Space Model* (GVSM) merupakan salah satu model Sistem Temu Kembali (*Information Retrieval System*).

Terkadang apa yang dicari tidak relevan dengan apa yang diinginkan, terdapat beberapa penyebab yang mengakibatkan apa yang dicari tidak relevan dengan apa yang diinginkan, salah satunya adalah kata kunci (*keyword*) yang ambigu. Solusi yang dapat digunakan untuk mengoptimalkan kata kunci (*keyword*) adalah dengan mengimplementasikan algoritma lesk. Algoritma lesk merupakan algoritma yang digunakan untuk menemukan makna kata yang berkaitan dengan kata kunci utama, dengan kata lain terdapat kata yang tidak dimasukan kedalam *query* tapi masuk kedalam kata pencarian.

Berdasarkan hasil penelitian yang telah dilakukan dapat dilihat bahwa penambahan algoritma lesk untuk sistem temu kembali (*Information Retrieval System*) bisa diimplementasikan dan bekerja dengan cukup baik. Algoritma lesk dapat bekerja sesuai yang diharapkan sebagai masukan sistem temu kembali (*Information Retrieval System*) dalam menghilangkan ambiguitas. Nilai yang dihasilkan oleh algoritma lesk sangat berpengaruh terhadap bobot similiaritas. Tidak hanya bobot similiaritas yang bertambah, tetapi nilai *recall* dan *precission* pun ikut meningkat. Berdasarkan penelitian terhadap 30 dokumen, nilai *recall* dan *precission* tanpa menggunakan algoritma lesk sebesar 94.736% dan 78.26% sedangkan menggunakan algoritma lesk sebesar 100% dan 82.608%.

Kata kunci : *Information Retrieval System*, *Generalized Vector Space Model*, Algoritma Lesk, *Keyword*, *Query*.

1. PENDAHULUAN

Sistem Temu Kembali (*Information Retrieval System*) dirancang untuk menemukan data (*file*) atau informasi yang diperlukan. Sistem Temu Kembali (*Information Retrieval System*) bertujuan untuk menjembatani kebutuhan informasi dengan sumber informasi yang tersedia secara relevan. *Generalized Vector Space Model* (GVSM) merupakan salah satu model Sistem Temu Kembali (*Information Retrieval System*).

Dalam hal ini, sistem temu kembali informasi berkaitan dengan representasi, penyimpanan, dan akses terhadap representasi *file*. *File* yang ditemukan harus relevan dengan kebutuhan informasi yang dinyatakan dalam *query*. Terkadang apa yang dicari tidak relevan dengan apa yang diinginkan, terdapat beberapa penyebab yang mengakibatkan apa yang dicari tidak relevan dengan apa yang diinginkan, salah satunya adalah kata kunci (*keyword*) yang ambigu[13]. Solusi yang dapat digunakan untuk mengoptimalkan kata kunci (*keyword*) adalah dengan mengimplementasikan algoritma lesk. Algoritma lesk merupakan algoritma yang digunakan untuk menemukan makna kata yang berkaitan dengan kata kunci utama, dengan kata lain terdapat kata yang tidak dimasukan kedalam *query* tapi masuk kedalam kata pencarian.

Berdasarkan latar belakang yang telah diuraikan, tujuan dari tugas akhir ini yaitu untuk menilai efektifitas algoritma lesk dalam melakukan pencarian kata atau kalimat yang ambigu. Penelitian sebagain topik untuk tugas akhir ini mengangkat judul "IMPLEMENTASI METODE GENERALIZED VECTOR SPACE MODEL (GVSM) MENGGUNAKAN ALGORITMA LESK PADA SISTEM TEMU KEMBALI".

1.1 *Information Retrieval* (IR)

Sistem Temu Kembali (*Information Retrieval System*) adalah bagaimana menemukan suatu dokumen dari dokumen-dokumen tidak terstruktur yang memberikan informasi yang dibutuhkan dari koleksi dokumen yang sangat besar yang tersimpan dalam komputer[5]. Tujuan dari Sistem Temu

Kembali (*Information Retrieval System*) adalah untuk memenuhi kebutuhan informasi pengguna dengan me*retrieve* semua dokumen yang mungkin relevan, pada waktu yang sama me-*retrieve* sesedikit mungkin dokumen yang tidak relevan. Sistem Temu Kembali (*Information Retrieval System*) yang baik memungkinkan pengguna menentukan secara cepat dan akurat apakah isi dari dokumen yang diterima memenuhi kebutuhannya.

Tujuan yang harus dipenuhi adalah bagaimana menyusun dokumen yang telah didapatkan tersebut ditampilkan terurut dari dukumen yang memiliki tingkat relevansi tinggi ke tingkat relevansi yang lebih rendah. Penyusunan dokumen tersebut disebut sebagai perangkingan dokumen.

1.2 *Generalized Vector Space Model* (GVSM)

Generalized Vector Space Model (GVSM) merupakan perluasan dari *Vector Space Model* (VSM) yaitu dengan menambahkan jenis informasi tambahan, disamping term, dalam merepresentasikan dokumen[6]. Sistem Temu Kembali (*Information Retrieval System*) dengan *Generalized Vector Space Model* (GVSM) merepresentasikan dokumen dengan similiaritas vektor terhadap semua dokumen yang ada.

Pada tahun 1985, Wong et al[6], menyajikan suatu alternatif terhadap Sistem Temu Kembali (*Information Retrieval System*) *Vector Space Model* (VSM), yang disebut *Generalized Vector Space Model* (GVSM). Deskripsi ringkas mengenai *Generalized Vector Space Model* (GVSM) diberikan oleh Carbonell dkk. Asumsikan term dari *Vector Space Model* (VSM) adalah *liniearly independent. Generalized Vector Space Model* (GVSM) menghindari pengasumsian dengan penggunaan dokumen-dokumen sebagai dasar ruang vektor dari pada term. Dalam "*Dual Space*" suatu dokumen direpresentasikan oleh suatu vektor dimana dimensinya merujuk terhadap dokumen.

Algoritma *Generalized Vector Space Model* yang dibahas menggunakan konsep ruang vektor. Masukan dari pengguna dan kumpulan dokumen diterjemahkan menjadi vektor-vektor, kemudian vektor-vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan pengguna (*query*) terhadap kumpulan dokumen.

Ada beberapa langkah atau proses untuk mendapatkan hasil dari *query* yang dimasukkan, yang disebut algoritma *Generalized Vector Space Model*[6]:

1. Membuang kata depan dan kata penghubung.
2. Menggunakan *stemmer* pada kumpulan dokumen dan *query*, yaitu aplikasi yang digunakan untuk menghilangkan imbuhan (awalan, akhiran). Contoh: ketampanan: tampan, kesalahan: salah.
3. Menentukan *minterm* untuk menentukan kemungkinan pola frekuensi kata. Panjang *minterm* ini didasarkan pada banyak kata yang diinput pada *query*. Kemudian diubah menjadi vektor orthogonal sesuai dengan pola *minterm* yang muncul. Kemungkinan pola yang akan muncul adalah :

$M_1 = (0,0,0)$
$M_2 = (1,0,0)$
.....
$M_n = (0,0,0)$

4. Menghitung banyaknya frekuensi atau kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query*
5. Menghitung *index term* yang dapat dinyatakan dengan :

$$\overrightarrow{Ki} = \frac{\sum_{\forall r.gi(Mr)=1} Ci,r.\overrightarrow{Mr}}{\sqrt{\sum_{\forall r.gi(Mr)=1} Ci,r^2}} \quad \ldots. (1)$$

Dimana :
$\overrightarrow{Ki}$: Indeks term ke-i
$\overrightarrow{Mr}$: Vektor ortogonal sesuai pola minterm yang terpakai
Ci,r : Faktor korelasi antara indeks term ke-i dengan minterm r
Sedangkan faktor korelasi sebagai berikut:

$$C_{i,r} = \sum_{dj|gi(\overrightarrow{dj})=gi(Mr)} W_{i,j} \quad \ldots. (2)$$

Dimana:
$W_{i,j}$: Vektor dokumen ke-j
$gi(Mr)$: Bobot indeks term K_i dalam minterm M_r

6. Mengubah dokumen dan *query* menjadi vector

$$\overrightarrow{d_j} = \sum_{i=1}^{n} W_{i,j} \times \overrightarrow{K_t} \quad \ldots. (3)$$

$$\vec{q} = \sum_{i=1}^{n} q_i \times \overrightarrow{K_i} \ldots. (4)$$

Dimana:
$\overrightarrow{d_j}$: Vektor dokumen ke-j
$\vec{q}$: Vektor *query*
$W_{i,j}$: Berat indeks *term* i pada dokumen j
q_i : Berat indeks *term* pada *query* i
n : Jumlah indeks *term*

7. Mengurutkan dokumen berdasarkan similaritas, dengan menghitung perkalian vector

$$Sim(\overrightarrow{d_j}.\vec{q}) = \frac{\overrightarrow{d_j}.\vec{q}}{|\overrightarrow{d_j}|\,|\vec{q}|} \ldots. (5)$$

Dimana :
$\overrightarrow{d_j}$: Vektor dokumen ke-J
$\vec{q}$: Vektor *query*

1.3 Algoritma Lesk

Algoritma *lesk* adalah algoritma yang digunakan untuk menghilangkan ambiguitas makna kata. Algoritma *lesk* merupakan salah satu algoritma untuk menyelesaikan masalah ambigu atau kata yang memiliki lebih dari satu arti (*word sense disambiguation*) dengan berbasis kamus. Algoritma ini bekerja dengan membandingkan definisi dari kata yang berambigu dengan definisi. Definisi dari kata tetangganya berdasarkan definisi kamus[13].

Algoritma *lesk* merupakan implementasi dari pendekatan *unsupervised* atau jaringan saraf tiruan yang digunakan untuk menghilangkan ambiguitas makna kata. Algoritma ini berdasarkan intuisi bahwa kata yang bermakna ambigu yang terdapat bersamaan dalam kalimat, digunakan untuk merujuk topik yang sama dan makna yang berhubungan dengan topik tersebut didefinisikan di dalam kamus dengan menggunakan kata yang sama. Algoritma *lesk* memiliki fungsi *relatedness* (keterhubungan) akan mengembalikan jumlah kata yang tumpang tindih antara definisi dari dua kata yang dimasukan.

1.4 *Preprocessing*

Proses yang terjadi pada *Generalized Vector Space Model* (GVSM) terbagi menjadi dua yaitu tahapan *preprocessing* yang terdiri dari *reading text* (*.doc) menggunakan *indexing*, *tokenizing*, *filtration* atau *stop word removal*, *stemming*, sedangkan proses yang kedua adalah menghitung relevansi antara kumpulan dokumen yang telah di-*preprocess* dengan *query* yang diinginkan pengguna. Banyaknya kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query* akan dihitung.

Query yang dimasukan oleh pengguna dan dokumen diterjemahkan menjadi vektor-vektor kemudian vektor-vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan pengguna (*query*) terhadap kumpulan dokumen.

1.4.1 *Indexing*

Mencari sebuah informasi yang relevan sangat tidak mungkin dapat dilakukan oleh sebuah komputer, meskipun dilakukan oleh sebuah komputer yang memiliki spesifikasi yang canggih. Agar komputer dapat mengetahui sebuah dokumen itu relevan terhadap sebuah informasi, komputer memerlukan sebuah model yang mendeskripsikan bahwa dokumen tersebut relevan atau tidak. Salah satu caranya adalah dengan menggunakan indeks istilah.

Indeks adalah bahasa yang digunakan di dalam sebuah buku konvensional untuk mencari informasi berdasarkan kata atau istilah yang mengacu ke dalam suatu halaman. Dengan menggunakan indeks, pencari informasi dapat dengan mudah menemukan informasi yang diinginkannya. Pada sistem temu-kembali informasi, indeks ini nantinya yang digunakan untuk merepresentasikan informasi di dalam sebuah dokumen.

Elemen dari indeks adalah istilah indeks (*index term*) yang didapatkan dari teks yang dipecah di dalam sebuah dokumen. Elemen lainnya adalah bobot istilah (*term weighting*) sebagai penentuan rangking dari kriteria relevan sebuah dokumen yang memiliki istilah yang sama.

Baeza-Yates dan Ribeiro-Neto[7] menjelaskan tentang proses pembuatan indeks dari sebuah dokumen teks atau dikenal dengan proses analisis teks (*automatic teks analysis*) melalui beberapa tahap:

a. Proses penghapusan digit, tanda hubung, tanda baca dan penyeragaman dari huruf yang digunakan.
b. Penyaringan kata meliputi penghilangan bukan kata kunci yang disebut *filtering* atau *stopword removal*.
c. Penghilangan imbuhan kata, baik awalan maupun akhiran kata. Penghilangan imbuhan kata ini dikenal dengan *stemming*.
d. Pemilihan istilah untuk menentukan kata atau *stem* (kelompok kata) yang akan digunakan sebagai elemen indeks.
e. Pembentukan kategori istilah terstruktur seperti kelompok persamaan kata yang digunakan untuk perluasan dari *query* dasar yang diberikan oleh pengguna sistem temu-kembali informasi dengan istilah lain yang sesuai.

Pengindeksan dapat dilakukan dengan dua cara yaitu manual dan otomatis. Idealnya, untuk mendapatkan indeks istilah yang sempurna sebuah pengindeksan dilakukan secara manual (konvensional). Akan tetapi, menurut Salton[7] sistem pencarian dan analisa teks yang sepenuhnya otomatis tidak menghasilkan kinerja temu-kembali yang lebih buruk dibandingkan dengan sistem konvensional yang menggunakan pengindeksan dokumen manual dan formulasi pencarian manual.

1.4.2 *Tokenizing*

Sebelum indeks dibandingkan dengan dokumen, dilakukan *tokenization* terlebih dahulu, yaitu mencacah kalimat kedalam bagian-bagian. Contohnya “dia bernama rahman”, setelah kalimat dimasukan maka tugas token adalah memecah kedalam bagian-bagian menjadi “dia”, “bernama”, “rahman”.

Terlihat dari contoh diatas terdapat kalimat “dia bernama Rahman” kemudian proses *tokenization* dilakukan dengan memecah kata dalam kalimat tersebut menjadi 3 pecahan yaitu dia, bernama, dan rahman.

1.4.3 *Filtering*

Dalam sebuah dokumen terdapat banyak kata yang bukan kata kunci di dalam dokumen atau kata-

kata tambahan hanya untuk menghubungkan kata, contohnya adalah kata penghubung dan juga terdapat tanda-tanda baca. Dalam proses *indexing* dilakukan proses untuk menghilangkan kata-kata tersebut untuk mengurangi proses peng-*index*-an dan mengurangi kata-kata dan tanda baca yang nantinya tidak berkaitan langsung dengan kata kunci.

Selain untuk mengungari proses *indexing* proses tersebut dilakukan agar penerapan perhitungan kesamaan dokumen dengan dokumen yang dicari terdapat kesesuaikan karena berkurangnya *noise* kata penghubung dan tanda baca yang jika tidak dihilangkan akan masuk kedalam perhitungan.

1.4.4 *Stemming*

Stemming merupakan bagian yang tidak terpisahkan dalam Sistem Temu Kembali (*Information Retrieval System*). Algoritma Nazief & Adriani sebagai algoritma *stemming* untuk teks berbahasa Indonesia yang memiliki kemampuan prosentase keakuratan (presisi) lebih baik dari algoritma lainnya. Algoritma ini sangat dibutuhkan dan menentukan dalam proses *preprocessing* dalam dokumen Indonesia.

Stemming adalah salah satu cara yang digunakan untuk meningkatkan performa Sistem Temu Kembali (*Information Retrieval System*) dengan cara mentransformasi kata-kata dalam sebuah dokumen teks ke bentuk kata dasarnya. Algoritma *stemming* untuk bahasa yang satu berbeda dengan algoritma *stemming* untuk bahasa lainnya. Sebagai contoh bahasa Inggris memiliki morfologi yang berbeda dengan bahasa Indonesia sehingga algoritma *stemming* untuk kedua bahasa tersebut juga berbeda. Proses *stemming* pada teks berbahasa Indonesia lebih rumit/kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan kata dasar (*root word*) dari sebuah kata. Pada umumnya kata dasar pada bahasa Indonesia terdiri dari kombinasi:

Prefiks 1 + Prefiks 2 + Kata dasar + Sufiks 3 + Sufiks 2 + Sufiks 1

Algoritma Nazief & Adriani yang dibuat oleh Bobby Nazief dan Mirna Adriani ini memiliki tahap-tahap sebagai berikut:

1. Pertama cari kata yang akan diistem dalam kamus kata dasar. Jika ditemukan maka diasumsikan kata adalah *root word*, maka algoritma berhenti.
2. *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”) dibuang. Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hapus *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan di kamus, maka algoritma berhenti. Jika tidak maka ke langkah 3a

a. Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 3b.
b. Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah 4.

4. Hapus *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka pergi ke langkah 4a, jika tidak pergi ke langkah 4b.

a. Periksa tabel kombinasi awalan-akhiran yang tidak diijinkan. Jika ditemukan maka algoritma berhenti, jika tidak
b. For i = 1 to 3, tentukan tipe awalan kemudian hapus awalan. Jika root word belum juga ditemukan lakukan langkah 5, jika sudah maka algoritma berhenti. Catatan: jika awalan kedua sama dengan awalan pertama algoritma berhenti.

5. Melakukan Recoding.
6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai *root word*, proses selesai.

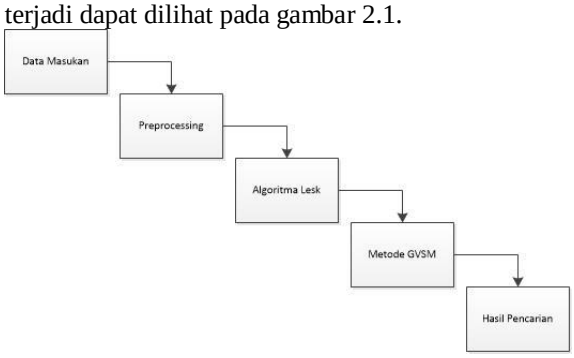
Tipe awalan ditentukan melalui langkah-langkah berikut:

1. Jika awalannya adalah: “di-”, “ke-”, atau “se-” maka tipe awalannya secara berturut-turut adalah “di-”, “ke-”, atau “se-”.
2. Jika awalannya adalah “te-”, “me-”, “be-”, atau “pe-” maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan “di-”, “ke-”, “se-”, “te-”, “be-”, “me-”, atau “pe-” maka berhenti.

Jika tipe awalan adalah “none” maka berhenti. Jika tipe awalan adalah bukan “none” maka hapus awalan jika ditemukan.

2. ISI PENELITIAN

Pada bagian ini penulis menjelaskan analisis metode yang terjadi pada implementasi metode *generalized vector space model* (GVSM) menggunakan algoritma lesk pada sistem temu kembali (*information retrieval system*). Proses yang terjadi dapat dilihat pada gambar 2.1.



Gambar 2.1. Proses Utama Sistem

2.1 Data Masukan

Terdapat dua jenis data yang menjadi masukan kedalam sistem, yang pertama yaitu kata atau kalimat (*query*) menggunakan bahasa Indonesia dengan Ejaan Yang Disempurnakan (EYD) berdasarkan

Keputusan Menteri Pendidikan dan Kebudayaan tanggal 27 Agustus 1975 Nomor 0196/U/1975[5]. Data masukan yang kedua adalah dokumen didalam komputer kemudian dilakukan proses ekstraksi text menggunakan *library* di .net yaitu *Microsoft.Office.Interop.Word*.

Sebagai contoh, terdapat sebuah *query* (Q), dan 5 buah dokumen yaitu dokumen 1 (D1), dokumen 2 (D2), dokumen 3 (D3), dokumen 4 (D4), dokumen 5 (D5) sebagai berikut:

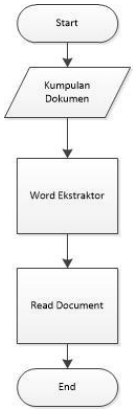
Q : Faktor kepala cabang dalam mempengaruhi kinerja karyawan
Judul D1: UNIKOM_AI KARTINI_BABIII
Judul D2: UNIKOM_FERY TRI LAKSANA_BAB2
Judul D3: UNIKOM_Fujiutama_Bab 2
Judul D4: UNIKOM_Putri Famawati_Abstrak
Judul D5: UNIKOM_Wupi Ocktavia K_Bab 5

2.2 Preprocessing

Pada tahapan ini, data yang telah dimasukan akan dilakukan *preprocessing* yang terdiri dari *reading text* (*.doc) dengan *tokenizing*, *filtration*, *stemming* dan algoritma lesk.

1. *Reading text*

Pada tahapan ini, *reading text* menggunakan metode *multithread* untuk meningkatkan kecepatan sistem dalam membaca dokumen dengan cara bersamaan. Berikut adalah langkah-langkah untuk melakukan *reading text* pada kumpulan dokumen dapat dilihat pada gambar 3.2. dibawah ini:



Gambar 2.2. Flowchart *Reading Text*

2. *Case Folding*

Dalam proses ini dilakukan pengecekan terhadap huruf-huruf kapital yang berada di tiap-tiap kalimat. Jika ditemukan huruf kapital tersebut, maka akan dilakukan *lowercase*, yaitu mengubah menjadi huruf kecil. Berikut adalah langkah langkah untuk melakukan *case folding* pada kumpulan dokumen dan *query* dapat dilihat pada gambar 2.3. dibawah ini:



Gambar 2.3. Flowchart *Case Folding*

Pada kasus ini, *query* dirubah menjadi huruf kecil menjadi "faktor pemimpin dalam mempengaruhi kinerja karyawan".

3. *Tokenizing*

Dalam proses ini dilakukan penghapusan tanda baca dan angka. Setelah proses tersebut, dokumen dipecah menjadi *token-token* dengan memotongnya menjadi kata (*term*). Berikut adalah langkah-langkah untuk melakukan *tokenizing* pada kumpulan dokumen dan *query* dapat dilihat pada gambar 2.4. dibawah ini:



Gambar 2.4. Flowchart *Tokenizing*

Pada kasus ini, *query* dibagi kedalam 6 bagian yang terdapat pada tabel 2.1.

Tabel 2.1. Hasil *Tokenizing Query*

faktor
kepala
cabang
dalam
mempengaruhi
kinerja
karyawan

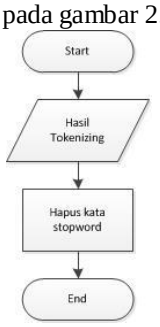
4. *Filtering*

Proses *filtering* merupakan proses menghilangkan kata-kata tidak penting yang terdapat dari hasil *tokenizing*. Untuk melakukan *filtering* bisa menggunakan *stoplist* atau *word list* atau *stopword*. Data hasil *tokenizing* akan dibandingkan dengan kamus, jika terdapat dalam kamus maka kata tersebut akan dihapus. Kata-kata yang tersisa merupakan kata yang dianggap penting. Untuk lebih jelasnya tahapan proses *filtering* adalah sebagai berikut :

1. Kata hasil proses *tokenizing* dibandingkan dengan kata *filtering* / *stopword*.
2. Jika data hasil *tokenizing* sama dengan kata di tabel *stopword* maka akan dihapus.

3. Jika tidak sama dengan tabel 2.1. kata *filtering* / *stopword* maka kata tersebut akan disimpan.

Berikut adalah langkah-langkah untuk melakukan *tokenizing* pada kumpulan dokumen dan *query* dapat dilihat pada gambar 2.5. dibawah ini:



Gambar 2.5. Flowchart *Filtering*

Pada kasus ini, kata "dalam" termasuk kedalam kelompok *stopword* maka kata "dalam" dihapus. Tabel 3.2. menunjukan perubahan *query* hasil *stopword*.

Tabel 2.2. hasil *stopword*

faktor
kepala
cabang
mempengaruhi
kinerja
karyawan

5. *Stemming*

Setelah proses *filtering*, dokumen dan *query* masuk ke proses *stemming*. Proses *stemming* yaitu menghilangkan kata depan dan kata belakang sehingga menjadi kata dasar. Penulis menggunakan Algoritma Stemming Bahasa Indonesia Nazief dan Adriani. Untuk kata dasar penulis mengambil dari Kamus Besar Bahasa Indonesia (KBBI).

Pada kasus ini, terdapat kata "mempengaruhi" yang memiliki imbuhan mem- dan -i menjadi pengaruh. Tabel 3.3. menunjukan perubahan kata yang telah di *stemming*.

Tabel 3.3. hasil *stemming*

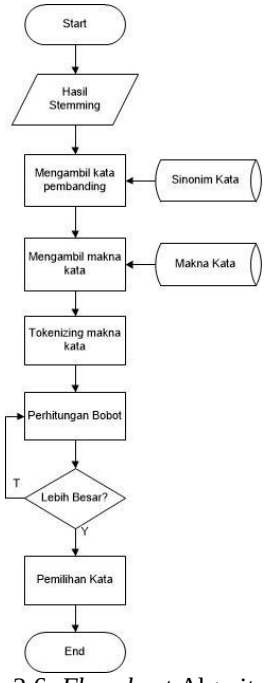
faktor
kepala
cabang
pengaruh
kinerja
karyawan

2.3 Algoritma Lesk

Setelah proses *preprocessing*, maka tahapan selanjutnya untuk mengoptmalkan kata kunci (*query*) agar tidak ambigu yaitu menggunakan algoritma lesk. Proses algoritma lesk yaitu membandingkan makna kata pada kata pembanding dengan dengan makna kata masukan (*query*) untuk menemukan sinonim kata yang tepat dengan *query*. Seluruh makna kata mengambil dari *website* kamus besar bahasa Indonesia dan untuk kata pembanding diambil dari *website* sinonim bahasa Indonesia. Untuk lebih jelasnya tahapan proses *stemming* adalah sebagai berikut:

1. Pengambilan kata (*query*) hasil stemming
2. Menentukan sinonim kata dari *query* yang nantinya akan menjadi kata pembanding
3. Pengambilan makna kata dari *query* dan kata pembanding
4. Melakukan proses *tokenizing* pada makna *query* dan kata pembanding
5. Menghitung bobot kata pembanding berdasarkan perbandingan makna kata *query* dengan makna kata pembanding
6. Memilih kata pembanding berdasarkan bobot yang paling besar

Berikut adalah langkah-langkah untuk melakukan proses algoritma lesk pada *query* dapat dilihat pada gambar 2.6. dibawah ini:



Gambar 2.6. *Flowchart* Algoritma Lesk

Pada kasus ini terdapat 6 *query* yang akan dibandingkan dengan kata pembanding. Proses algoritma lesk dapat dilihat pada tabel 2.4.

Tabel 2.4. Algoritma lesk

Kata (*query*)	Makna	Kata Pembanding	Makna	Bobot
Kepala	bagian tubuh yang di atas leher (pada manusia, beberapa jenis hewan	akal	daya pikir, jalan cara melakukan sesuatu, daya upaya,	1

	merupakan tempat otak, pusat jaringan saraf, dan beberapa pusat indra)		ikhtiar	
	pemimpin, ketua (kantor, pekerjaan, perkumpulan)	pemimpin	orang yang memimpin	2

Berdasarkan hasil perhitungan algoritma lesk, kata (*query*) "kepala" memiliki 2 kata pembanding yaitu "akal" yang memiliki bobot 0 dan "pemimpin" yang memiliki bobot 2, maka kata pembanding yang diambil sebagai hasil dari perhitungan algoritma lesk adalah pemimpin karena memiliki nilai bobot yang lebih besar. Hasil dari algoritma lesk nantinya akan ditambahkan kedalam *query* agar hasil pencarian lebih optimal. Tabel 2.5. merupakan hasil dari perhitungan algoritma lesk

Tabel 2.5. hasil algoritma lesk

aspek
pemimpin
filial
akibat
prestasi
buruh

2.4 *Generalized Vector Space Model* (GVSM)

Ada beberapa langkah atau proses untuk mendapatkan hasil dari *query* yang dimasukkan, yang disebut algoritma *Generalized Vector Space Model*[6]:

1. Membuang kata depan dan kata penghubung.
2. Menggunakan *stemmer* pada kumpulan dokumen dan *query*, yaitu aplikasi yang digunakan untuk menghilangkan imbuhan (awalan, akhiran). Contoh: ketampanan: tampan, kesalahan: salah.
3. Menentukan *minterm* untuk menentukan kemungkinan pola frekuensi kata. Panjang *minterm* ini didasarkan pada banyak kata yang diinput pada *query*. Kemudian diubah menjadi vektor orthogonal sesuai dengan pola *minterm* yang muncul.
4. Menghitung banyaknya frekuensi atau kemunculan kata dalam kumpulan dokumen yang sesuai dengan *query*
5. Menghitung *index term*
6. Mengubah dokumen dan *query* menjadi vector
7. Mengurutkan dokumen berdasarkan similaritas, dengan menghitung perkalian vector

2.4.1 *Generalized Vector Space Model* (GVSM) Menggunakan Algoritma Lesk

Tabel 2.6. Hasil perhitungan GVSM menggunakan algoritma lesk

Dokumen	Bobot Similiaritas
D1	0.999702951479197
D2	0.986850140318568
D3	0.913581007337747
D4	0
D5	0

Berdasarkan hasil similiaritas antara dokumen dengan *query* maka dapat disimpulkan bahwa urutan dokumen yang sesuai dengan *query* adalah:

1. Dokumen 1 (D1) = 0.999702951479197
2. Dokumen 2 (D2) = 0.986850140318568
3. Dokumen 3 (D3) = 0.913581007337747
4. Dokumen 4 (D4) = 0
5. Dokumen 5 (D5) = 0

Dikarenakan nilai similiaritas dokumen 2 lebih besar dibandingkan dengan nilai similiaritas dokumen yang lainnya maka ($Sim(\vec{d_1}.\vec{q}) > Sim(\vec{d_2}.\vec{q}) > Sim(\vec{d_3}.\vec{q}) > Sim(\vec{d_4}.\vec{q}) > Sim(\vec{d_5}.\vec{q})$). Berdasarkan contoh kasus diatas dapat disimpulkan bahwa *Generalized Vector Space Model* (GVSM) menghitung korelasi antar *query* dan dokumen dengan cara menghitung semua term dijadikan vektor ortogonal untuk menghitung *Index term* dan setelah itu setiap *term* pada dokumen digeneralisasi menjadi vektor ortogonal dengan mengkalikan hasil *Index term* dengan *term* dokumen dan *query*, yang kemudian setiap vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan (*query*) terhadap kumpulan dokumen.

2.4.2 *Generalized Vector Space Model* (GVSM) Tanpa Menggunakan Algoritma Lesk

Tabel 2.7. Hasil perhitungan GVSM tanpa menggunakan algoritma lesk

Dokumen	Bobot Similiaritas
D1	0.816474545780421
D2	0.996230230720323
D3	0.653918516245756
D4	0
D5	0

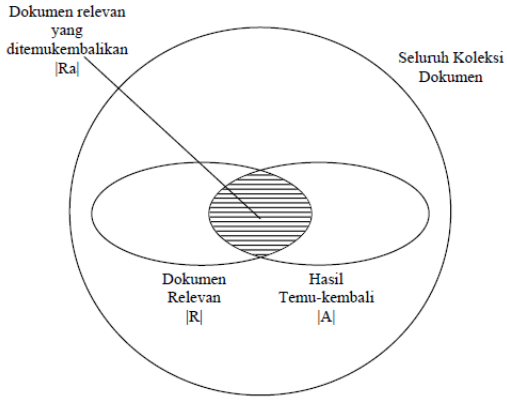
Berdasarkan hasil similiaritas antara dokumen dengan *query* maka dapat disimpulkan bahwa urutan dokumen yang sesuai dengan *query* adalah:

1. Dokumen 2 (D2) = 0.996230230720323
2. Dokumen 1 (D1) = 0.816474545780421
3. Dokumen 3 (D3) = 0.653918516245756
4. Dokumen 4 (D4) = 0
5. Dokumen 5 (D5) = 0

Dikarenakan nilai similiaritas dokumen 2 lebih besar dibandingkan dengan nilai similiaritas dokumen yang lainnya maka ($Sim(\vec{d_2}.\vec{q}) > Sim(\vec{d_1}.\vec{q}) > Sim(\vec{d_3}.\vec{q}) > Sim(\vec{d_4}.\vec{q}) > Sim(\vec{d_5}.\vec{q})$). Berdasarkan contoh kasus diatas dapat disimpulkan bahwa *Generalized Vector Space Model* (GVSM) menghitung korelasi antar *query* dan dokumen dengan cara menghitung semua term dijadikan vektor ortogonal untuk menghitung *Index term* dan setelah itu setiap *term* pada dokumen digeneralisasi menjadi vektor ortogonal dengan mengkalikan hasil *Index term* dengan *term* dokumen dan *query*, yang kemudian setiap vektor tersebut dikenakan operasi perkalian titik dan hasilnya menjadi acuan dalam menentukan relevansi masukan (*query*) terhadap kumpulan dokumen.

2.5 Pengujian *Recall* dan *Precission*

Evaluasi dari sistem temu kembali (*Information Retrieval System*) dipengaruhi oleh dua parameter utama yaitu *recall* dan *precision*. *Recall* adalah rasio antara dokumen relevan yang berhasil ditemukembalikan dari seluruh dokumen relevan yang ada di dalam sistem, sedangkan *precision* adalah rasio dokumen relevan yang berhasil ditemukembalikan dari seluruh dokumen yang berhasil ditemu-kembalikan[10].



Gambar 2.7. *Recall* dan *Precision* pada contoh hasil temu-kemabali informasi[7]

Berdasarkan Gambar 2.7. *recall* dan *precision* dapat dinyatakan sebagai berikut:

$$\text{Re}call = \frac{|Ra|}{|R|}$$

$$\text{Pr}ecision = \frac{|Ra|}{|A|} \quad \text{..... (6)}$$

Untuk menguji ketepatan dan keakuratan aplikasi, dilakukan pengujian dengan menghitung nilai *recall* dan *precission* dimulai dari dokumen yang berjumlah 10 hingga 69.

Tabel 2.8. Hasil Penelitian *Recall* dan *Precission*

No.	Jumlah Dokumen	Algoritma Lesk		Tanpa Algoritma Lesk	
		Recall (%)	Precission (%)	Recall (%)	Precission (%)
1.	10	100	60	100	60
2.	20	100	65	100	65
3.	30	100	80	79.166	63.333
4.	40	100	82.5	96.96	80
5.	50	100	94	95.744	90
6.	60	100	85	98.039	83.333
7.	69	100	82.608	94.736	78.26

3. PENUTUP

Berdasarkan hasil penelitian yang telah dilakukan dapat dilihat bahwa penambahan algoritma lesk untuk sistem temu kembali (*Information Retrieval System*) bisa diimplementasikan dan mampu menghilangkan kata ambigu di dalam *query*. Algoritma lesk dapat bekerja sesuai yang diharapkan sebagai masukan sistem temu kembali (*Information Retrieval System*) dalam menghilangkan ambiguitas.

Nilai yang dihasilkan oleh algoritma lesk sangat berpengaruh terhadap bobot similiaritas. Berdasarkan pengujian di bab sebelumnya, bobot similiaritas menggunakan algoritma lesk sebesar. Tidak hanya bobot similiaritas yang bertambah, tetapi nilai *recall* dan *precission* pun ikut meningkat. Pada *folder* yang berisi 30 dokumen menghasilkan nilai *recall* dan

precission tanpa menggunakan algoritma lesk sebesar 94.736% dan 78.26%, sedangkan menggunakan algoritma lesk sebesar 100% dan 82.608%. Terlihat bahwa algoritma lesk dapat meningkatkan keakuratan pencarian dokumen dengan baik.

Berdasarkan hasil penelitian, analisis, perancangan hingga pembuatan implementasi metode *Generalized Vector Space Model* (GVSM) menggunakan algoritma lesk pada Sistem Temu Kembali (*Information Retrieval System*), maka dapat diberikan saran sebagai berikut:

1. Tahapan *preprocessing* seperti *tokenizing, filtering*, *stemming*, dan penambahan algoritma lesk yang relatif lama diharapkan dapat dioptimalkan kembali.
2. Kelengkapan basis data seperti meenggunakan *Wordnet* bahasa Indonesia sebagai acuan dalam menghilangkan ambiguitas, akan tetapi untuk Wordnet bahasa Indonesia sampai saat ini belum ada dan masih dikembangkan.
3. Pengambilan makna kata dan sinonim kata yang dapat dilakukan otomatis oleh sistem
4. Untuk pengembangan lebih lanjut, dapat dilengkapi dengan kata ambigu berupa frasa ambigu, kata berimbuhan yang ambigu.
5. Pengujian dapat dilakukan pada koleksi dokumen yang lebih banyak.

DAFTAR PUSTAKA

[1] Zainal A. Hasibuan , Yofi Andri : Penerapan Berbagai Teknik Sistem Temu-Kembali Informasi Berbasis Hiperteks, Fakultas Ilmu Komputer, Universitas Indonesia
[2] Chairul Furqon, Konsep Informasi (diakses tanggal 5/10/2015)
[3] http://widuri.raharja.info/index.php?title=BAB_II_ROOSTER_(KKP) diakses tgl 5/10/2015 pukul 20.26
[4] Sahirul Alim Tri Bawono, *Information Retrieval* Meningkatkan Pencarian Data yang Relevan, Universitas Gadjah Mada.
[5] Jasman Pardede, Mira Musrini Barmawi, Wildan Denny Pramonor "IMPLEMENTASI METODE GENERALIZED VECTOR SPACE MODEL PADA APLIKASI INFORMATION RETRIEVAL", Institut Teknologi Nasional Bandung 2013
[6] Baeza, Ricardo, B. Ribeiro. 1999. *Modern Information Retrieval*. ACM Press. United States of America. 1999.
[7] Nazief, B. d. Approach to Stemming Algorithm. *Confix-Stripping*.
[8] Salton, G. (1989). *Automatic Text Processing,The Transformation, Analysis, andRetrieval of Information by Computer*. United States of America: Addison – Wesly Publishing Company,Inc. All rights reserved.Nadirman, S, 2006,
[9] Sistem Temu-Kembali Informasi dengan Metode Vector Space Model pada Pencarian File Dokumen Berbasis Teks, Yogyakarta, Universitas Gadjah Mada.
[10] Jasman Pardede, IMPLEMENTASI MULTITHREADING UNTUK MENINGKATKAN KINERJA INFORMATION RETRIEVAL DENGAN METODE GVSM", Jurnal Sistem Komputer Vol. 4 No. 1, Mei 2014
[11] Abdul Rouf, Pengujian Perangkat Lunak Dengan Menggunakan Metode White box dan Black box, Sistem Informasi – STMIK HIMSYA Semarang
[12] SQL Server Official Site (www.microsoft.com/sql/default.asp) diakses tanggal 28/11/2015
[13] Stanjeev Banerjee and Ted Pedersen, "An Adapted Lesk Algorithm For Word Sense Disambiguition Using WordNet", University of Minnesota, Duluth, MN 55812 USA