BAB I
PENDAHULUAN

1.1 Latar Belakang

Dengan semakin melimpahnya informasi yang mengalir setiap hari, ringkasan menjadi sesuatu yang penting. Tujuan dari ringkasan adalah memberikan ekstraksi dari dokumen dengan menyajikan inti dokumen secara singkat namun memenuhi keperluan pembaca. Dibanding membaca keseluruhan dokumen, pembaca akan cepat mempelajari isi dari sebuah dokumen dari sebuah ringkasan.

Besarnya jumlah dan ukuran dokumen yang menjadi sumber ringkasan mendorong penelitian tentang otomasi peringkasan dokumen. Dipelopori oleh Luhn pada tahun 1958, metode dan pendekatan dalam otomasi peringkasan dokumen semakin beragam dan berkembang dari tahun ke tahun (Erkan, 2004). Salah satu metode yang sering digunakan adalah *Latent Sematics Analysis*. Metode ini memanfaatkan analisis persamaan semantik antar kata dengan menggunakan sumber dokumen berskala besar. Metode *Latent Semantics Analysis* dikembangkan dan dipatenkan oleh Scott Deerwester, Susan Dumais, George Furnas, Richard Harshman, Thomas Landauer, Karen Lochbaum dan Lynn Streeter pada tahun 1988 (Wikipedia, 2006).

Dengan berkembangnya metode dan pendekatan dalam otomasi peringkasan dokumen, muncul tantangan untuk menyediakan hasil ringkasan yang informatif dan mememenuhi keinginan pembaca. Memanfaatkan *user defined query,* muncul metode-metode untuk menjawab tantangan tersebut. Salah satu metode yang dikembangkan secara itensif adalah EMBRA System yaitu penelitian oleh Ben Hachey, Gabriel Murray dan David Reitter yang diperkenalkan dalam DUC-2005 (*Document Understanding Conference*) pada tahun 2005 (Hatchey, 2005). EMBRA System adalah suatu metode peringkasan dokumen dengan mengacu pada *user query* dan menggunakan data semantik yang sangat besar (Hatchey, 2005).

Dari sekian banyak metode yang diteliti, sangat jarang atau bahkan tidak ada yang melibatkan sumber berbahasa indonesia untuk dijadikan bahan. Oleh karena itu akan dilakukan penelitian

penerapan peringkasan (*summarization*) untuk mengekstrak informasi dari sumber berita berbahasa indonesia.

Dengan menerapkan modifikasi metode yang diperkenalkan oleh Ben Hachey, Gabriel Murray dan David Reitter dan menggunakan metode *latent semantic analysis* sebagai analisa semantik diharapkan hasil ekstraksi informasi dapat mendekati hasil buatan manusia, sehingga judul yang diambil untuk tugas akhir ini adalah **"Ekstraksi informasi melalui *Query Oriented Multi Document Summarization* dengan *Latent Semantic Analysis* "**.

1.2 Perumusan Masalah

Berdasarkan uraian pada latar belakang masalah, maka dalam tugas akhir ini dapat dirumuskan permasalahan sebagai berikut:

1. Apakah metode peringkasan dokumen dapat diterapkan dalam ekstraksi informasi?
2. Berapakah tingkat akurasi dari metode yang diterapkan?

1.3 Batasan Masalah

Dari perumusan masalah, diberikan batasan masalah untuk menghindari melebarnya masalah yang akan diselesaikan:

1. Penelitian menggunakan sumber dokumen berbahasa Indonesia .
2. Mengabaikan dampak kesalahan hasil proses stemming dalam evaluasi ringkasan.
3. Mengabaikan urutan kata dalam *query* dalam perhitungan ekstraksi.

1.4 Tujuan Penelitian

Tujuan penelitian yang ingin dicapai adalah:

1. Mengetahui hasil penerapan metode peringkasan dokumen dalam ekstraksi informasi.
2. Mengetahui tingkat keakuratan hasil metode yang diterapkan.

1.5 Manfaat Penelitian

Memberikan gambaran penerapan metode peringkasan dokumen untuk mengekstrak informasi dan memberikan analisa tingkat keakuratan hasil dari metode yang diterapkan.

1.6 Metodologi Pemecahan Masalah

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah:

1. Studi Literatur
   Mempelajari teori-teori dan penelitian terdahulu yang berhubungan dengan konsep *Text Summarization, Multi-document Summarization, Information Retrieval, Latent Semantic Analysis* dari berbagai referensi.
2. Pendefinisian dan analisis masalah
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
   Membuat perancangan perangkat lunak untuk mendukung penelitian dengan analisis terstruktur dan mengimplementasikan hasil rancangan.
4. Uji coba dan analisa hasil implementasi
   Menguji perangkat lunak yang dibangun, dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievaluasi dan disempurnakan.

1.7 Sistematika Penulisan

Tugas akhir ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN
   Berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi pemecahan masalah, dan sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Menguraikan teori-teori yang berhubungan dengan *Information Retrieval, Text Summarization,* dan *Latent Semantics Analysis.*
3. BAB III METODOLOGI DAN PERANCANGAN SISTEM
   Pada bab ini akan dijelaskan mengenai metode-metode yang digunakan dalam penelitian penerapan metode peringkasan dokumen untuk mengekstrak informasi.

4. BAB IV IMPLEMENTASI DAN UJI COBA SISTEM
   Pada bab ini akan dilakukan implementasi sistem, pengujian dan analisa sistem yang dibangun, yaitu apakah dengan menerapkan metode peringkasan memberikan hasil ekstraksi yang dapat mendekati hasil buatan manusia.

5. BAB V KESIMPULAN DAN SARAN
   Berisi kesimpulan dari seluruh rangkaian penelitian serta saran kemungkinan pengembangannya.

BAB II
DASAR TEORI

2.1 *Text Mining*

Text mining dapat diartikan sebagai penemuan informasi yang baru dan tidak diketahui sebelumnya oleh komputer. Hal ini dilakukan secara otomatis dengan mengambil informasi dari sumber yang berbeda-beda (Hearst, 2003). Penemuan informasi ini bertujuan untuk menemukan fakta baru atau hipotesa baru untuk dieksplorasi lebih jauh menggunakan percobaan lebih lanjut.

Text mining adalah varian dari *data mining,* yang mencoba menemukan pola yang menarik dari basis data yang besar. Perbedaan antara *data mining* dan *text mining* terletak pada sumber data yang digunakan, *text mining* menggunakan data teks yang tidak terstruktur sedangkan *data mining* menggunakan data yang terstruktur dari basis data (Hearst, 2003).

Text mining, kadang-kadang disebut sebagai *text data mining*, secara umum bisa disebut sebagai proses untuk mendapatkan informasi yang berkualitas dari teks (Wikipedia, 2006). Proses mendapatkan informasi yang berkualitas dapat dilakukan dengan cara meramalkan pola dan kecenderungan menggunakan pembelajaran pola statistik.

Tujuan dari *text mining* adalah untuk memproses informasi yang tidak terstruktur berupa data tekstual, kemudian mengambil informasi dari teks itu untuk dirubah kedalam data numerik yang lebih berarti. Informasi yang ada dalam teks itu dapat digunakan untuk meramalkan pola dan kecenderungan yang dimiliki berdasarkan pada kata yang dikandungnya. Data numerik yang dihasilkan tersebut berasal dari kata-kata yang dikandung oleh suatu dokumen. Sehingga dengan menganalisa kata yang sudah diubah kedalam data numerik ini maka pembelajaran pola statistik bisa dilakukan misalnya untuk mengklasifikasikan dokumen, mengelompokkan dokumen ataupun mencari kemiripan antar dokumen.

Untuk memenuhi tujuan itu maka dalam *text mining* ada beberapa proses yang harus dilalui yang biasanya meliputi proses penyusunan teks masukan (biasanya *parsing*, dilanjutkan dengan penghilangan kata-kata yang tidak penting yaitu *stop word* dan

stemming, dan kemudian dimasukkan ke basis data). Dari basis data yang dihasilkan maka dapat diambil pola dan dilakukan evaluasi untuk menentukan keluaran.

2.2 *Text Summarization*

Salah satu cabang dari *text mining* adalah *text summarization* atau dalam bahasa indonesia lazim disebut peringkasan. Ide dasar dari sebuah ringkasan adalah penyingkatan. Ringkasan adalah versi singkat dari sebuah dokumen aslinya. Menurut Maybury dan Mani, ringkasan merupakan proses dari pembuatan intisari informasi terpenting dari suatu sumber untuk menghasilkan versi yang lebih ringkas bagi penggunanya (Many dan Mayburry, 1999).

Terdapat dua tipe dari pembuatan suatu ringkasan yang mengambil bagian terpenting dari teks aslinya, yaitu :

- Abstraktif.

  Abstraktif menghasilkan sebuah interpretasi terhadap teks aslinya. Dimana sebuah kalimat akan ditransformasikan menjadi kalimat yang lebih singkat dan kalimat baru yang tidak terdapat dalam dokumen yang asli.

- Ekstraktif

  Ekstraktif menghasilkan ringkasan dengan memilih sebagian dari kalimat yang ada dalam dokumen asli. Metode ini menggunakan *statistical*, *linguistical*, dan *heuristic* atau kombinasi dari semuanya dalam menetapkan ringkasan suatu teks.

Meskipun ringkasan yang dihasilkan oleh manusia bersifat tidak ekstraktif, tapi kebanyakan penelitian mengenai peringkas ini adalah ekstraktif. Secara teori ringkasan ekstraktif memberikan hasil yang lebih baik dibandingkan peringkas abstraktif. Hal ini dikarenakan masalah dalam abstraktif, seperti representasi semantik, *inferens*, dan pembangun *natural language*, relatif lebih sulit, dibandingkan pendekatan *data-driven*, seperti ekstraksi kalimat (Erkan dan Radev, 2004).

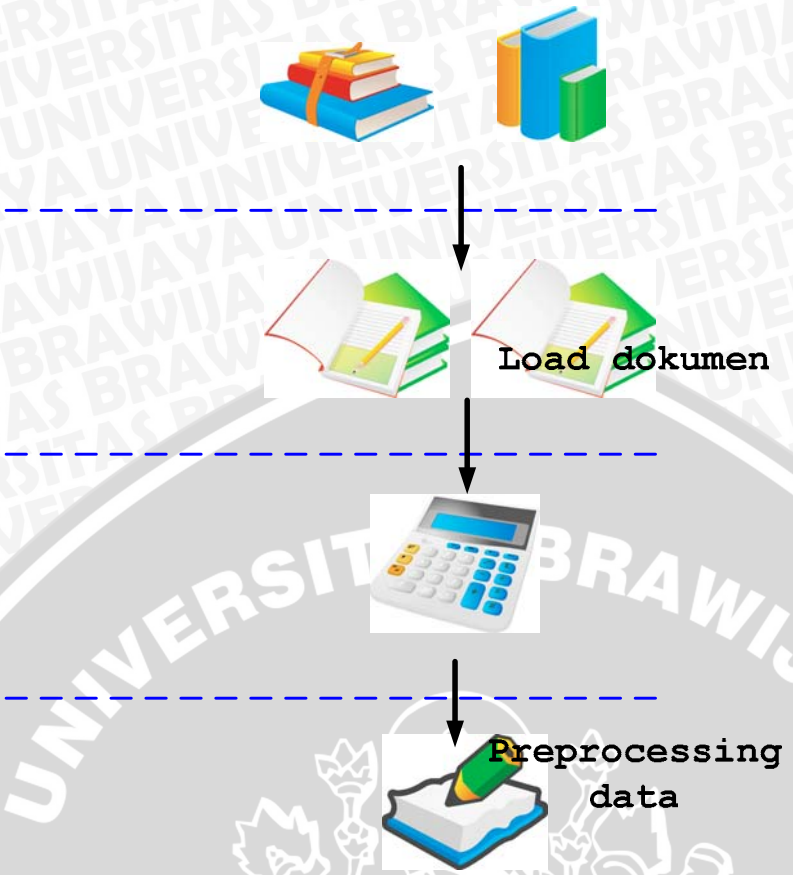
Tujuan dari *text summarizer* dapat diketegorikan berdasarkan maksud, fokus, dan cakupannya (Firmin dan Chrzanowski, 1999), adalah :

- Informatif, Indicatif, dan Evaluatif
  - Informatif, ringkasan ini menyatakan informasi-informasi penting yang terdapat pada dokumen asal.
  - Indikatif, tujuan dari ringkasan ini adalah untuk dijadikan sebuah referensi, yang membantu pembaca untuk mengetahui isi dari teks daripada membaca keseluruhan teks yang ada. Ringkasan ini meliputi topik kunci dari teks asal.
  - Evaluatif, atau ringkasan yang melibatkan pembuatan sebuah pertimbangan pada teks asal, seperti suatu tinjauan ulang atau opini.
- *User-focused* dan *Generic*
  - *User-focused* (*query-relevant*), ringkasan yang dibuat berdasarkan topik yang dipilih oleh user, sering merupakan jawaban dari *query* yang dimiliki oleh user.
  - *Generic*, disebut juga *author-focused*, sifatnya lebih umum dan berdasarkan pada teks aslinya.
- Dokumen tunggal (*single document*) dan Banyak Dokumen (*multi document*)
  - Dokumen tunggal, ringkasan merupakan ringkasan dari satu dokumen
  - Banyak Dokumen, ringkasan merupakan hasil ringkasan dari beberapa dokumen. Pembuatan peringkas multi dokumen lebih sulit dibanding satu dokumen.

Otomasi *text Summarization* adalah serangkaian proses pengolahan data tekstual melalui komputasi numerik. Data tekstual tidak secara langsung dapat diolah melalui komputasi numerik. Sebelumnya data tekstual melalui *preprocess* sehingga data numerik, selanjutnya dijelaskan secara rinci pada Subbab 2.2.1.

Setelah data numerik didapatkan, proses selanjutnya adalah pengolahan data. Dalam proses ini, banyak metode yang dapat dilakukan. Seperti yang dipaparkan dalam latar belakang dalam Bab 1 proses yang akan diterapkan adalah *Latent Semantics Analysis* yang akan dijelaskan secara rinci dalam Subbab 2.2.2.

Proses yang terakhir adalah pembentukan hasil ringkasan. Metode pembentukan ringkasan akan dijelaskan dalam Subbab 2.2.3. Secara umum otomasi peringkasan dokumen dapat dilihat dalam Gambar 2.1



Gambar 2.1 Ilustrasi otomasi peringkasan dokumen

Dari Gambar 2.1, tahap-tahap proses *text Summarization* secara umum dapat dijabarkan:

1. *Preprocessing* data.
2. Pengolahan data.
3. Pembentukan ringkasan

Penjelasan secara rinci mengenai proses *text Summarization* dijelaskan pada Subbab 2.2.1, Subbab 2.2.2 dan Subbab 2.2.3.

2.2.1 *Preprocessing* data

Preprocessing data adalah proses yang dilakukan untuk mempersiapkan dokumen mentah baik dokumen latih maupun dokumen tes sebelum siap diolah. Dilakukannya *preprocessing* data ini bertujuan untuk memindahkan dokumen teks menjadi representasi data yang rapi dan siap dihitung untuk proses peringkasan. Sebagaimana dinyatakan oleh Gonde guo dkk, bahwa *preprocessing* data mengimplementasikan fungsi untuk memindahkan dokumen awal kedalam representasi yang rapi, biasanya diimplementasikan pada dokumen latih dan dokumen tes (Guo, 2005).

Preprocessing data dilakukan karena dokumen teks tidak dapat diolah secara langsung untuk mendapatkan hasil jika tidak diterjemahkan terlebih dahulu. Penerjemahan ini bertujuan untuk menghasilkan data numerik yang mudah diakses, karena data numeriklah yang dapat digunakan untuk perhitungan lebih lanjut. Hal ini juga didukung oleh Sebastiani yang menyatakan bahwa alasan dilakukannya *preprocessing* data menjadi representasi dokumen yang rapi adalah karena dokumen teks tidak dapat diterjemahkan secara langsung oleh algoritma pembentuk hasil pencarian (Sebastiani, 2002).

Langkah-langkah dalam *preprocess* biasanya meliputi beberapa tahapan, yaitu:

1. pemisahan kalimat,
2. *tokenization,*
3. *filtration,*
4. *stemming* dan
5. pembobotan.

2.2.1.1 Pemisahan kalimat

Agar proses ekstraksi dan pembentukan ringkasan dapat dilakukan, dokumen dipisah menjadi kalimat-kalimat. Pemisahan kalimat dilakukan berdasarkan tanda baca titik (.), tanda seru (!) dan tanda tanya (?).

2.2.1.2 *Tokenization*

Tokenization adalah proses untuk mengambil kata dan istilah sederhana dari sebuah dokumen, pendapat itu dikemukakan oleh Baldi (Baldi, 2003). Kata dan istilah sederhana itu berupa potongan-potongan kata tunggal yang menyusun suatu dokumen.

Selama fase ini proses yang dialami oleh suatu dokumen meliputi penguraian teks dalam dokumen menjadi potongan-potongan kata. Kemudian dirubah menjadi huruf kecil dan semua tanda baca dihilangkan. Penghilangan tanda baca inilah yang menyebabkan suatu dokumen hanya terdiri dari kata dan istilah sederhana.

2.2.1.3 *Filtration*

Filtration adalah proses untuk menentukan kata penting yang digunakan untuk merepresentasikan dokumen. Kata-kata penting itu dianggap dapat menggambarkan isi dokumen dan untuk membedakan dokumen yang satu dengan dokumen yang lain. Proses yang dialamai pada tahapan ini adalah dilakukannya penghapusan kata-kata yang tidak penting seperti kata penghubung, kata sapaan dsb. Pendapat ini juga didukung oleh Garcia yang menyatakan bahwa dalam fase ini biasa dilakukan penghapusan kata-kata yang bukan merupakan ciri (kata unik) dari suatu dokumen seperti kata sambung misalnya dan, yang, dsb, yang dikenal dengan *stopword,* (Garcia, 2005).

2.2.1.4 *Stemming*

Stemming adalah proses untuk mereduksi kata ke bentuk dasarnya (Garcia, 2005). Bisa dikatakan juga bahwa *Stemming* merupakan proses yang menyediakan suatu pemetaan antara berbagai kata dengan morfologi yang berbeda menjadi satu bentuk dasar (*stem*) (Tala, 2003). Hal ini dilakukan untuk menghilangkan bermacam-macam bentuk kata yang berbeda kedalam bentuk tunggal. Misalnya sebuah dokumen berisi beberapa kejadian kata seperti tabrak, menabrak, dan tabrakan. Sedangkan dalam *query* yang dinginkan adalah kata dengan kata kunci ditabrak maka kata ini tidak pernah terjadi dalam dokumen (Baldi, 2003). Disamping berguna dalam proses mendapatkan kembali informasi *stemming*

juga bisa mengurangi ukuran indeks. Salah satu *stemmer* yang terkenal untuk bahasa inggris adalah *Porter Stemmer* yang dikembangkan oleh Porter pada tahun 1980 (Baldi, 2003).

Proses Stemming hanya dapat diterapkan pada kata kerja berimbuhan. Untuk menghindari terjadinya stemming pada kata-kata yang bukan kata kerja, perlu diberikan daftar *block word. Block word* adalah daftar kata benda atau kata asing yang mengandung unsur imbuhan misalnya difusi (mengandung imbuhan di dan i), perang (mengandung imbuhan per) dan kata-kata lain yang dapat dilihat dalam lampiran.

Untuk melakukan proses *stemming* pada bahasa indonesia, dibutuhkan pemahaman tentang morfologi bahasa indonesia itu sendiri.

2.2.1.4.1 Struktur morfologi kata bahasa Indonesia

Morfologi adalah bagian dari ilmu bahasa yang membicarakan atau yang mempelajari seluk beluk kata serta pengaruh perubahan-perubahan bentuk kata terhadap golongan dan arti kata, atau dengan kata lain dapat dikatakan bahwa morfologi mempelajari seluk-beluk bentuk kata serta fungsi perubahan-perubahan bentuk kata itu (Tala, 2003).

Morfologi kata bahasa Indonesia bisa terdiri dari struktur *infleksional* dan *derivasional*. *Infleksional* adalah struktur yang paling sederhana yang dinyatakan dalam penambahan sufiks dimana tidak mempengaruhi arti sebenarnya dari kata dasar yang dilekati (Tala, 2003). Sufiks *infleksional* dapat dibagi menjadi 2 jenis :

1. Sufiks *-lah, -kah, -pun, -tah*. Sufiks ini sebenarnya adalah partikel yang tidak mempunyai arti. Keberadaannya pada suatu kata adalah untuk penekanan. Contoh :

   dia + kah → diakah
   duduk + lah → duduklah

2. Sufiks *-ku, -mu, -nya*. Sufiks ini berfungsi sebagai kata ganti kepunyaan. Contoh :

   tas + ku → tasku
   buku + mu → bukumu

Sufiks-sufiks diatas dapat melekat pada kata dasar secara bersama-sama. Adapun aturan urutannya adalah sufiks pada jenis kedua selalu diletakkan sebelum sufiks jenis pertama. Sehingga struktur morfologi pada kata *infleksional* adalah :

Infleksional = *(kata dasar +kata ganti)* | *(kata dasar + partikel)* | *(kata dasar + kata ganti + partikel)*

Penambahan sufiks *infleksional* tidak akan merubah bentuk dasar dari kata berimbuhan (Tala, 2003). Dengan kata lain, tidak ada penghilangan atau peleburan kata dasar pada kata berimbuhan. Kata dasar dapat ditentukan dengan mudah pada struktur *infleksional*.

Struktur *derivasional* dalam bahasa Indonesia terdiri dari prefiks, sufiks dan kombinasi dari keduanya. Prefiks yang sering dipakai adalah : *ber-, di-, ke-, meng-, peng-, per-, ter-*. Contoh penggunaan prefiks adalah :

ber	+	lari	→	berlari
di	+	makan	→	dimakan
ke	+	kasih	→	kekasih
meng	+	ambil	→	mengambil
peng	+	atur	→	pengatur
per	+	lebar	→	perlebar
ter	+	baca	→	terbaca

Beberapa prefiks seperti *ber-*, *meng-*, *peng-*, *per-*, *ter-* mungkin akan berubah menjadi beberapa bentuk yang berbeda. Bentuk dari setiap prefiks bergantung pada karakter pertama dari kata dasar yang dilekatinya. Tidak seperti struktur *infleksional*, pada struktur *derivasional* pengucapan kata mungkin berubah setelah adanya penambahan prefiks. Seperti contoh menyapu yang terdiri dari prefiks meng- dan kata dasar sapu. Prefiks meng- berubah menjadi meny- dan karakter pertama dari kata dasar mengalami peleburan.

Sufiks *derivasional* adalah *-i, -kan, -an* (Tala, 2003). Contoh penggunaan sufiks *derivasional* adalah :

gula	+	i	→	gulai

makan	+ an	→	makanan
sampai	+ kan	→	sampaikan

Berbeda dengan penggunaan prefiks, penambahan sufiks tidak akan mengubah bentuk dasar dari suatu kata.

Seperti disebutkan sebelumnya, struktur *derivasional* juga terdiri dari konfiks, yaitu gabungan dari prefiks dan sufiks yang melekat secara bersama-sama pada suatu kata. Contoh :

per + main + an	→ permainan
ke + kalah + an	→ kekalahan
ber + jatuh + an	→ berjatuhan
meng + ambil + i	→ mengambili

Tidak semua prefiks dan sufiks dapat dikombinasikan menjadi sebuah konfiks. Ada beberapa kombinasi prefiks dan sufiks yang tidak diperbolehkan. Kombinasi tersebut adalah :

Tabel 2.1 Pasangan Konfiks yang tidak diperbolehkan

Prefiks	Sufiks
ber	i
di	an
ke	i | kan
meng	an
peng	i | kan
ter	an

Prefiks/konfiks dapat ditambahkan pada suatu kata yang telah terdapat konfiks/prefiks, yang menghasilkan struktur prefiks ganda. Seperti pada pembentukan sebuah konfiks, pada pembentukan prefiks ganda, tidak semua prefiks/konfiks dapat ditambahkan pada kata yang telah mendapatkan prefiks/konfiks. Ada beberapa aturan dalam urutan pembentukan prefiks ganda. Aturan-aturan tersebut adalah :

Tabel 2.2 Urutan prefiks ganda (Tala, 2003)

Prefiks 1	Prefiks 2
meng	per

di ter ke	ber

Struktur morfologi pada kata derivasional adalah :

Derivasional = (prefiks + kata dasar) | (kata dasar + sufiks) | (prefiks + kata dasar + sufiks) | (prefiks 1 + prefiks 2 + kata dasar) | (prefiks 1 + prefiks 2 + kata dasar + sufiks).

Struktur lain yang mungkin terjadi dalam morfologi bahasa Indonesia adalah penambahan sufiks *infleksional* pada struktur *derivasional*, yang dinamakan multiple sufiks.

Sehingga dapat disimpulkan secara umum struktur morfologi kata bahasa Indonesia adalah :

Sturtur morfologi = [prefiks 1] + [prefiks 2] + kata dasar + [sufiks] + [kata ganti] + [partikel].

keterangan :

[...] menunjukkan opsi/pilihan.

2.2.1.4.2 Proses *stemming* bahasa Indonesia

Berdasarkan analisa morfologi yang telah dibahas sebelumya, maka terdapat 5 aturan tahapan pada proses *stemming* dalam bahasa Indonesia (Tala, 2003). Aturan tersebut adalah :

- Aturan tahap pertama menangani partikel infleksional

Tabel 2.3 Menangani partikel infleksional

Sufiks	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
kah	NULL	2	NULL	diakah→ dia
lah	NULL	2	NULL	adalah → ada
tah*	NULL	2	NULL	apatah → apa
pun**	NULL	2	NULL	Buku pun→buku

* tah → improduktif

** pun → menurut EYD terpisah dengan kata mengikutinya

- Aturan tahap kedua menangani kata ganti infleksional

Tabel 2.4 Menangani kata ganti infleksional

Sufiks	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
ku	NULL	2	NULL	bukuku→buku
mu	NULL	2	NULL	bukumu→buku
nya	NULL	2	NULL	bukunya→buku

- Aturan tahap ketiga menangani urutan prefiks derivasional pertama.

Tabel 2.5 Menangani urutan prefiks derivasional pertama

Prefiks	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
meng	NULL	2	NULL	mengukur→ukur
meny	s	2	V...*	menyapu→sapu
men	t	2	V...	menuduh→tuduh
men	NULL	2	NULL	menduga→duga
mem	p	2	V...	memukul→pukul
mem	NULL	2	NULL	membakar→bakar
me	NULL	2	NULL	merusak→rusak
peng	NULL	2	NULL	pengukur→ukur
peny	s	2	V...	penyelam→selam
pen	t	2	V...	penari→tari
pen	NULL	2	NULL	Penduga→duga
pem	P	2	V...	Pemandu→pandu
pem	NULL	2	NULL	Pembaca→baca
di	NULL	2	NULL	diukur→ukur
ter	NULL	2	NULL	tersipu→sipu
ke	NULL	2	NULL	kekasih→kasih

* kata dasar dimulai huruf vokal

- Aturan tahap keempat menangani urutan prefiks derivasional kedua.

Tabel 2.6 Menangani urutan prefiks derivasional kedua

Prefiks	Pengganti	Kondisi	Kondisi	Contoh

		Ukuran	Tambahan	
ber	NULL	2	NULL	berlari→lari
bel	NULL	2	ajar	belajar→ajar
be	NULL	2	kerja	bekerja→kerja
per	NULL	2	NULL	perjelas→jelas
pel	NULL	2	ajar	pelajar→ajar
pe	NULL	2	NULL	pekerja→kerja

- Aturan tahap kelima menangani sufiks derivasional

Tabel 2.7 Menangani urutan sufiks derivasional

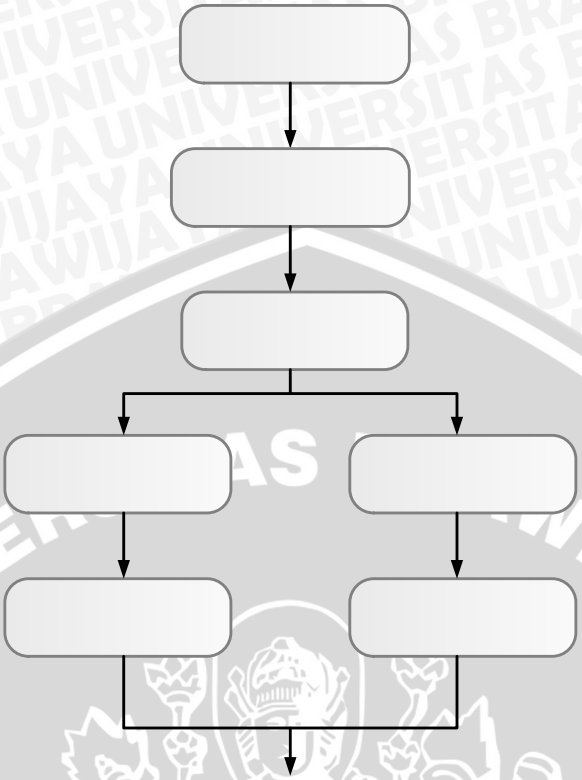
Sufiks	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
kan	NULL	2	prefiks∉ {ke,peng}	tarikkan→tarik (meng)ambilkan→ ambil
an	NULL	2	prefiks∉ {di,meng,ter }	makanan→makan (per)janjian→janji
i	NULL	2	V|K...c1c2,c1≠s, c2≠i dan prefiks∉{ber,ke,peng}	tandai→tanda (men)dapati→dapat

Kondisi ukuran adalah jumlah mininum suku kata dalam sebuah kata. Karena dalam bahasa Indonesia, kata dasar setidaknya mempunyai 2 suku kata. Maka kondisi ukuran dalam proses *stemming* bahasa Indonesia adalah dua. Adapun suku kata didefinisikan memiliki satu vokal.

2.2.1.4.3 Algoritma

Algoritma Stemming didasarkan ide sufiks pada bahasa inggris yang sebagian besar terdiri dari kombinasi dari sufiks yang lebih kecil dan lebih simpel yang mensimulasikan perubahan kata kerja pada sebuah kata. Setiap langkah setiap sufiks dihilangkan melalui kondisi tertentu (Tala, 2003).

Desain algoritma Stemming untuk bahasa indonesia dalam sistem yang akan dibangun dapat dilihat dalam Gambar 2.2



Gambar 2.2 Skema Stemming Bahasa Indonesia

2.2.1.5 Pembobotan

Pembobotan adalah proses untuk menentukan nilai numerik suatu kata dalam dokumen. Skema pembobotan yang digunakan ada bermacam-macam yaitu pembobotan *tf,* dan pembobotan *tfidf* yang dikemukakan oleh Garcia (Garcia, 2005).

Definisi masing-masing dari pembobotan itu adalah, pembobotan *tf* adalah penghitungan jumlah masing-masing kata yang muncul dalam dokumen. *Idf* adalah inverse dari *document frequency* dimana *document frequency* merupakan banyaknya dokumen yang ada dalam proses baik dokumen latih maupun dokumen tes yang

mengandung kata tertentu. Jadi dokumen yang dilibatkan dalam perhitungan *idf* adalah keseluruhan dokumen latih dan satu dokumen yang akan dikategorikan. Dua skema ini memunculkan terobosan baru yang menggabungkan dua skema pembobotan ini yaitu skema pembobotan *tfidf*. Pembobotan *tfidf* adalah kombinasi antara pembobotan *tf (term ferquency)* dan *idf* (*inverse document frequency*).

Skema pembobotan yang digunakan dalam penelitian ini adalah skema pembobotan *tfidf* .Disamping sering digunakan, pembobotan ini levelnya lebih umum karena melibatkan dokumen itu sendiri dan juga kesuluruhan dokumen yang ada.

Dalam proses pembobotan ini ada beberapa langkah yang harus dilakukan yaitu :

- Melakukan perhitungan jumlah masing-masing kata di tiap-tiap dokumen untuk mendapatkan nilai *tf* yang ditunjukkan oleh $\#(t_k, d_j)$ pada Persamaan 2.1 .
- Membentuk kata menjadi representasi vektor kata yang dikenal sebagai *vector space model*. Pembahasan lebih lanjut mengenai *vector space model* terdapat pada Subbab 2.4.1.6
- Melakukan perhitungan nilai *df*-nya yang ditunjukkan dengan $Tr(t_k)$ pada Persamaan 2.1
- Melakukan perhitungan nilai *idf*-nya dengan memanfaatkan vektor kata yang telah dibentuk
- Melakukan perhitungan *tfidf* untuk mendapatkan bobot masing-masing kata ditiap-tiap dokumen menggunakan persamaan 2.1.

$$tfidf(t_k, d_j) = \#(t_k, d_j).\log\frac{|Tr|}{\#Tr(t_k)}, \qquad (2.1)$$

Keterangan :

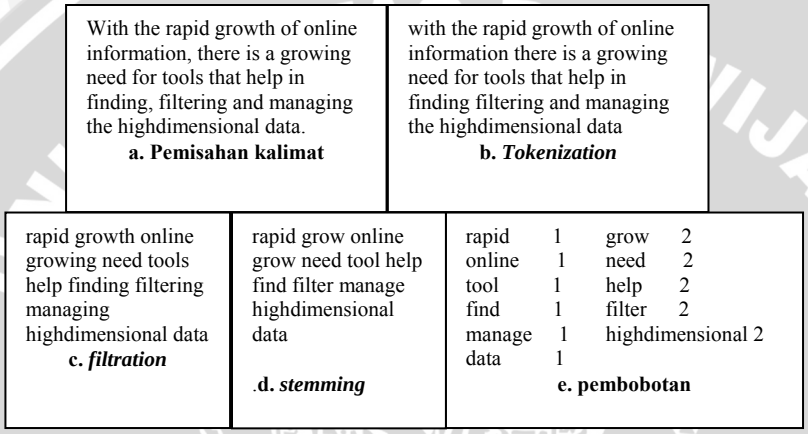
- $\#(t_k, d_j)$ menyatakan jumlah kejadian t_k dalam dokumen d_j,
- $|Tr|$ menyatakan jumlah keseluruhan dokumen yaitu keseluruhan dokumen
- $Tr\ (t_k)$ menyatakan jumlah dokumen dalam *Tr* yang mengandung kejadian kata t_k.

Fungsi 2.1 menyatakan bahwa semakin sering kata muncul dalam dokumen maka semakin menggambarkan isinya, dan semakin banyak dokumen yang mengandung kata yang sama maka semakin kecil perbedaannya, pendapat ini diungkapkan oleh Sebastiani

(Sebastiani, 2002). Contoh perhitungan *tfidf* ditunjukkan pada Tabel 2.8.

Tabel 2. 8 Perhitungan *tfidf*

t_k	$\#(t_k, d_j)$	$Tr\ (t_k)$	$\lvert Tr \rvert$	$\log \frac{\lvert Tr \rvert}{\#Tr(t_k)}$	$tfidf(t_k, d_j)$
t_1	312	28799	30000	0.017744	5.53608
t_2	179	26452	30000	0.054663	9.784631
t_3	136	179	30000	2.224268	302.5005
t_4	131	231	30000	2.113509	276.8697
t_5	63	98	30000	2.485895	156.6114
t_6	45	142	30000	2.324833	104.6175
t_7	37	227	30000	2.121095	78.48053



Gambar 2.3 Proses penghapusan *markup* dan *format* (a), *Tokenization* (b), *filtration*(c), *stemming* (d), diikuti dengan pembobotan (e).

Sumber : (Garcia, 2006)

2.2.1.6 *Vector space model*

Vector space model adalah gambaran dokumen dalam bentuk vektor kata, definisi ini dikemukakan oleh Liao Yihua [LIA02]. Dalam representasinya akan dibagi menjadi dua bagian yaitu baris

dan kolom seperti sebuah matrik dengan ukuran *N* dimensi. Baris merupakan kumpulan urutan dokumen, pada Gambar 2.4 ditunjukkan dengan $d_1, d_2, \ldots, d_N$ dan q. Kolom adalah kumpulan urutan kata atau istilah yang ditunjukkan oleh t_1, t_2, t_3 **...***tn* pada Gambar 2.4. Sedangkan isi dari vektor itu adalah jumlah tiap-tiap kata dalam masing masing dokumen yang ditunjukkan oleh angka-angka pada Gambar 2.4. Angka-angka yang ditunjukkan pada gambar tersebut masih berupa bobot *tf*. Bobot *tf* inilah yang nantinya juga digunakan untuk menentukan bobot *tfidf* dari suatu kata pada sebuah dokumen yang sifatnya umum. Hasil dari perhitungan pembobotan akan digunakan untuk membentuk Vektor dengan dimensi N, $R^{|N|}$.

	t_1	t_2	t_3	...	t_n
d_1	14	6	1		0
d_2	0	1	3	...	1
d_3	0	1	0	...	2
...	...	...	...	...	...
d_N	4	7	0	...	5
q	0	1	0	...	1

Gambar 2.4 *Vector Space Model*

2.2.2 Pengolahan Data

Dalam tahap ini, data *Vector Space Model* yang didapatkan dari *preprocess* kemudian diolah menggunakan metode *Latent Semantics Analysis*. Penjelasan mengenai metode *Latent Semantics Analysis,* proses dan hasilnya akan dijelaskan pada Subbab 2.2.2.1.

2.2.2.1 Latent Semantics Analysis

Latent Semantics Analysis (LSA) adalah sebuah teknik yang terdapat pada *Natural Language Processing*, terutama pada *vectorial semantic*, mengenai analisa hubungan dari beberapa dokumen dan term yang dimiliki dengan membentuk konsep mengenai kondisi dan term (Wikipedia, 2006). Hal ini didapatkan dengan merepresentasikan similaritas antar kata dengan kata , kalimat dengan kalimat dan kata dengan kalimat

melalui penghitungan statistika pada sekumpulan besar dokumen teks (Wikipedia, 2006). Representasi kata yang dihasilkan oleh *LSA* terbukti mampu mensimulasikan berbagai fenomena pendekatan oleh manusia. Mulai dari pendekatan kategorisasi daftar kata hingga penilaian kualitas essai.

LSA dipatenkan pada tahun 1988 oleh Scott Deerwester, Susan Dumais, George Furnas, Richard Harshman, Thomas Landauer, Karen Lochbaum dan Lynn Streeter. Jika konteks aplikasinya mengarah ke *information retrieval* biasanya disebut *Latent Semantics Indexing* (Wikipedia, 2006). *LSA* menerapkan dekomposisi matrix yang disebut *Singular Value Decomposition.*

2.2.2.2 Singular Value Decomposition

Singular Value Decomposition (SVD) adalah metode yang terdapat dalam aljabar linier. *SVD* banyak digunakan dalam analisa faktor (Wikipedia, 2006). *SVD* mengolah matrik dengan dekomposisi yang disebut *eigen analysis* dan menghasilkan dua matrik yang memiliki *eigen values* dan satu matriks yang mengandung *eigen vector* (Deerweister, 1990).

Misalkan yang akan didekomposisi adalah matriks X berukuran m x n yang elemen (i,j) menunjukkan kemunculan term i pada dokumen j, dimana i dimulai dari 1 hingga m dan j dimulai dari 1 hingga n.

$$\begin{bmatrix} X_{i,j} & \cdots & X_{i,n} \\ \vdots & \ddots & \vdots \\ X_{m,j} & \cdots & X_{m,n} \end{bmatrix}$$

Gambar 2.5 *Term-document matrix*

Sebuah baris dari matriks diatas akan menjadi vektor yang berhubungan dengan term, memberikan relasi antar dokumen.

$$t_j^T = \begin{bmatrix} X_{i,j} & \cdots & X_{i,n} \end{bmatrix}$$

Gambar 2.6 *Term to term vector*

Sebuah kolom dari matriks menjadi vektor yang menghubungkan dokumen, memberikan relasi antar term.

$$d_j = \begin{bmatrix} X_{1,j} \\ \vdots \\ X_{m,j} \end{bmatrix}$$

Gambar 2.7 *Document to Document vector*

Sekarang diasumsikan terdapat dekomposisi dari matriks X berupa U dan V yang merupakan *orthonomal matrix* dan $\sum$ adalah *diagonal matrix*. Inilah yang disebut *Singular Value Decomposition.*[8]

$$\underset{X}{\begin{bmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{m,1} & \cdots & X_{m,n} \end{bmatrix}} = \underset{U}{\begin{bmatrix} \begin{bmatrix} u_1 \end{bmatrix} & \cdots & \begin{bmatrix} u_l \end{bmatrix} \end{bmatrix}} \cdot \underset{S}{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_l \end{bmatrix}} \cdot \underset{V^T}{\begin{bmatrix} \begin{bmatrix} v_1 \end{bmatrix} \\ \begin{bmatrix} v_l \end{bmatrix} \end{bmatrix}}$$

Gambar 2.8 *Singular Value Decomposition*

Matriks U adalah matriks (t x r), dimana r adalah nilai min antara t dan d. Matriks T didapatkan dari nilai *eigen vector* dari matriks simetrik k XX^T.

Matriks V^T adalah matriks (t x r), dimana r adalah nilai min antara t dan d. Matriks D didapatkan dari nilai *eigen vector* dari matriks simetrik coovariance X^TX.

2.2.2.3 Penerapan Latent Semantics Analisis

Salah satu contoh pembentukan LSA yang diambil dari jurnal *"Introduction To Latent Semantics Analysis"*. Dokumen teks didalam contoh ini adalah beberapa judul memo.

Tabel 2.9 Contoh daftar dokumen

Kode	Judul Memo
C1	Human machine interface for ABC computer applications
C2	A survey of user opinion of computer system response time
C3	The EPS user interface management system
C4	System and human system engineering testing of EPS
C5	Relation of user perceived response time to error measurement
M1	The generation of Random, binary, ordered trees
M2	The intersection graph of paths in trees
M3	Graph Minors IV:Widths of trees and well-quasi-ordering
M4	Graph Minors:A survey

dari daftar dokumen diatas dibentuk matriks term-dokumen dengan menghitung banyaknya term dalam satu dokumen.

Tabel 2.10 Contoh *term-document matrix*

	C1	C2	C3	C4	C5	M1	M2	M3	M4
Human	1	0	0	1	0	0	0	0	0
Interface	1	0	1	0	0	0	0	0	0
Computer	1	1	0	0	0	0	0	0	0
User	0	1	1	0	1	0	0	0	0
System	0	1	1	2	0	0	0	0	0
Response	0	1	0	0	1	0	0	0	0
Time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
Survey	0	1	0	0	0	0	0	0	1
Trees	0	0	0	0	0	1	1	1	0
Graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

dari matriks *term-document* dioperasikan *Singular Value Decomposition* menjadi tiga matriks dekomposisi seperti terlihat dalam persamaaan 2.2

$$X = U \quad S \quad V^T \qquad (2.2)$$

Matriks U (*term to term matrix*) didapatkan dari kumpulan *eigen vector* dari matriks XX^T (Wikipedia, 2006) seperti yang dapat dilihat dalam Gambar 2.8

$$U = \begin{bmatrix} 0.22 & -0.11 & 0.29 & -0.41 & -0.11 & -0.34 & 0.52 & -0.06 & -0.41 \\ 0.20 & -0.07 & 0.14 & -0.55 & 0.28 & 0.50 & -0.07 & -0.01 & -0.11 \\ 0.24 & 0.04 & -0.16 & -0.59 & -0.11 & -0.25 & -0.30 & 0.06 & 0.49 \\ 0.40 & 0.06 & -0.34 & 0.10 & 0.33 & 0.38 & 0.00 & 0.00 & 0.01 \\ 0.64 & -0.17 & 0.36 & 0.33 & -0.16 & -0.21 & -0.17 & 0.03 & 0.27 \\ 0.27 & 0.11 & 0.43 & 0.07 & 0.08 & -0.17 & 0.28 & -0.02 & -0.05 \\ 0.27 & 0.11 & 0.43 & 0.07 & 0.08 & -0.17 & 0.28 & -0.02 & -0.05 \\ 0.30 & -0.14 & 0.33 & 0.19 & 0.11 & 0.27 & 0.03 & -0.02 & -0.17 \\ 0.21 & 0.27 & -0.18 & -0.03 & 0.54 & 0.08 & -0.47 & -0.04 & -0.58 \\ 0.01 & 0.49 & 0.23 & 0.03 & 0.59 & -0.39 & -0.29 & 0.25 & -0.23 \\ 0.04 & 0.62 & 0.22 & 0.00 & -0.07 & 0.11 & 0.16 & -0.68 & 0.23 \\ 0.03 & 0.45 & 0.14 & -0.01 & -0.03 & 0.28 & 0.34 & -0.68 & 0.18 \end{bmatrix}$$

Gambar 2.9 *Term to term matrix*

Sedangkan matriks V^T (*document to document matrix*) didapatkan dari kumpulan *eigen vector* dari matriks X^TX (Wikipedia, 2006) seperti terlihat pada Gambar 2.9

$$V^T = \begin{bmatrix} 0.20 & 0.61 & 0.46 & 0.54 & 0.28 & 0.00 & 0.01 & 0.02 & 0.08 \\ -0.06 & 0.17 & -0.13 & -0.23 & 0.11 & 0.19 & 0.44 & 0.62 & 0.53 \\ 0.11 & -0.50 & 0.21 & 0.57 & -0.51 & 0.10 & 0.19 & 0.25 & 0.08 \\ -0.95 & -0.03 & 0.04 & -0.21 & 0.15 & 0.02 & 0.02 & 0.01 & -0.03 \\ 0.05 & -0.21 & 0.38 & -0.37 & 0.33 & 0.39 & 0.35 & 0.15 & -0.60 \\ -0.08 & -0.26 & 0.72 & 0.26 & 0.03 & -0.30 & -0.21 & 0.00 & 0.36 \\ 0.18 & -0.43 & -0.24 & 0.26 & 0.67 & -0.34 & -0.15 & 0.25 & 0.04 \\ -0.01 & 0.05 & 0.01 & -0.02 & -0.06 & 0.45 & -0.76 & 0.45 & -0.07 \\ -0.06 & 0.24 & 0.02 & -0.08 & -0.26 & -0.62 & 0.02 & 0.52 & -0.45 \end{bmatrix}$$

Gambar 2.10 *Document to document matrix*

Matriks S (*Singular Matrix*) didapatkan dari kumpulan nilai *eigen value* dari matriks XX^T (Wikipedia, 2006). Hasil matriks S dapat dilihat dalam gambar 2.10

$$S = \begin{bmatrix} 3.34 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 2.54 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 2.35 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.64 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.5 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.31 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.85 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.56 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.36 \end{bmatrix}$$

Gambar 2.11 *Singular Matrix*

Matriks-matriks yang diperoleh dari pengolahan data dapat menggambarkan representasi semantik dari dokumen. Seperti yang dikemukakan oleh Deerweister, hubungan antar *term* dan hubungan antar dokumen didapatkan dengan mengoperasikan matriks U, matriks S dan V^T (Deerweister, 1990).

2.2.3 Pembentukan ringkasan

Tahap terakhir dalam proses *Summarization* adalah pembentukan ringkasan. Dalam tahap ini, dilakukan ekstraksi kalimat-kalimat yang memenuhi syarat menjadi ringkasan. Penentuan kalimat yang akan diekstrak dalam sistem ini adalah modifikasi algoritma yang dikemukakan oleh Ben Hachey, Gabriel Murray dan David Reitter dalam EMBRA System yang diperkenalkan dalam DUC-2005 pada tahun 2005 (Hatchey, 2005).

2.2.3.1 Algoritma Ekstraksi

Seperti yang telah dikemukakan dalam latar belakang pada bab I, sistem yang akan dibangun menerapkan modifikasi algoritma ekstraksi pada EMBRA System yaitu penelitian oleh Ben Hachey, Gabriel Murray dan David Reitter yang diperkenalkan dalam DUC-2005 pada tahun 2005 (Hatchey, 2005) dan dapat dilihat pada gambar 2.11

```
for each sentence in document:
        for each word in sentence:
                get word vector from semantic model
        average word vectors to form sentence vector
        sim1 = cossim(sentence vector, query vector)
        sim2 = highest(cossim(sentence vector, all extracted vectors))
        score = λ*sim1 - (1-λ)*sim2
        if sentence contains multiple named entities:
                if granularity == 'specific':
                        weight score higher
                else if granularity == 'general':
                        weight score lower
        else:
                do not weight score
extract sentence with highest score
repeat until desired length
```

Gambar 2.12 Algoritma Ekstraksi EMBRA System

Modifikasi yang dilakukan dititikberatkan pada skor pembobotan dan penghitungan similaritas *extracted sentence* dan

sentence vector (*sim2*). Hal ini dikarenakan diperlukan penghitungan similaritas seluruh *extracted sentence* dengan *sentence vector*, yang mana dalam EMBRA System hanya diwakili similaritas terbesar. Sedangkan untuk skor pembobotan yang digunakan dalam EMBRA System diubah menjadi penambahan similaritas, karena penghitungan similaritas seluruh *extracted sentence* dengan *sentence vector* diambil rata-rata similaritasnya. Sehingga algoritma yang diterapkan dapat dilihat dalam Gambar 2.12

```
while summary not reached desired length
  for each sentence in document
    sim1 = cossim(sentence vector, query vector)
    sim2= average(cossim(sentence vector, extracted vector))
    score = λ.sim1 + (1-λ).sim2
  extracted sentence with highest score
```

Gambar 2.13 Modifikasi algoritma ekstraksi

Penghitungan similaritas dalam sistem dapat dilihat pada Subbab 2.2.3.2

2.2.3.2 Penghitungan simililaritas dua dokumen

Secara umum, penghitungan similaritas antara dua dokumen didapatkan dengan menghitung nilai kosinus dari dua vector dokumen (Deerweister, 1990). Dalam *Latent Semantic Analysis* penghitungan similaritas antara dua dokumen dapat dilakukan dengan mengoperasikan

$$Sim(di, dj) = \sum_{m=1}^{k} V_{im} V_{jm} s_m^2 \qquad (2.3)$$

dimana Di adalah dokumen pertama dalam Matriks V, dan Vj adalah dokumen kedua. Sedangkan S adalah nilai singular (Deerweister, 1990).

2.2.4 Evaluasi

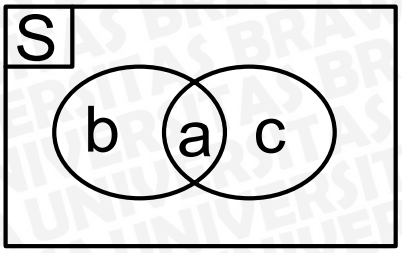
Evaluasi dalam peringkasan teks lebih berhubungan dengan percobaan dari pada secara analisis. Karena untuk mengevaluasi sistem secara analisis diperlukan rincian permasalahan yang dicoba diselesaikan oleh sistem. Evaluasi percobaan pada *summarizer* biasanya lebih diutamakan untuk mengukur keefektifannya daripada efisiensinya, yaitu kemampuannya untuk melakukan peringkasan secara benar

Untuk melakukan proses evaluasi diperlukan suatu matriks hasil pengkategorian yang disebut dengan matrik *confusion*. Matriks *confusion* berisi informasi mengenai klasifikasi sebenarnya dan prediksi klasifikasi yang dilakukan oleh sistem. Hal ini juga didukung oleh pendapat yang menyatakan bahwa *performance* sistem biasanya dievaluasi menggunakan data dalam matriks (Oregina, 2006). Tabel 2.11 ini menunjukkan matriks *confusion* untuk hasil pencarian.

Tabel 2. 11 Matriks *Confusion*

Peringkasan oleh sistem	Peringkasan sebenarnya	
	Ya	Bukan
Ya	a	b
Bukan	c	d

- a adalah jumlah peringkasan yang benar terhadap dokumen test yang dilakukan oleh sistem, peringkasan sebenarnya ya dan peringkasan oleh sistem ya.
- b adalah jumlah peringkasan yang salah terhadap dokumen test yang dilakukan oleh sistem, peringkasan sebenarnya ya dan peringkasan oleh sistem bukan.
- c adalah jumlah peringkasan yang salah, peringkasan sebenarnya ya dan peringkasan oleh sistem bukan .
- d adalah jumlah peringkasan yang benar, peringkasan sebenarnya bukan dan peringkasan oleh sistem juga bukan.



Gambar 2.14 Diagram Matriks *Confusion*

Diagram yang ditujukkan oleh Gambar 2.7 yang dirumuskan dari matriks *confusion* akan memudahkan perhitungan evaluasi. Evaluasi standar yang digunakan dalam peringkasan teks ada dua macam, yang biasanya digunakan untuk mengukur efektifitas algoritma yang digunakan yaitu *precision*, dan *recall* sedangkan F_1 *measure* merupakan kombinasi dari kedua evaluasi tersebut Untuk menghitung tingkat kesalahan sistem digunakan *Fall-out.*

Recall di definisikan sebagai perbandingan antara peringkasan yang benar yang dilakukan oleh sistem dibagi dengan jumlah keseluruhan peringkasan yang benar. Persamaan 2.5 merupakan persamaan yang digunakan untuk mengukur *recall.*

$$recall = \frac{a}{a+c}. \qquad (2.4)$$

Precision adalah perbandingan antara jumlah peringkasan yang benar yang dilakukan oleh sistem dibagi dengan jumlah keseluruhan peringkasan yang dilakukan oleh sistem. Persamaan 2.6 merupakan persamaan yang digunakan untuk mengukur *precision.*

$$precision = \frac{a}{a+b}. \qquad (2.5)$$

Sedangkan F_1 *measure* merupakan kombinasi antara *recall* (2.5) dan *precision* (2.6) *y*ang bisa didefinisikan dengan Persamaan 2.7.

$$F_1\ measure = \frac{2 \times recall \times precision}{recall + precision} \qquad (2.7)$$

Fall-out adalah proporsi antara hasil yang salah oleh terhadap keseluruhan hasil *irrelevant* yang tersedia.

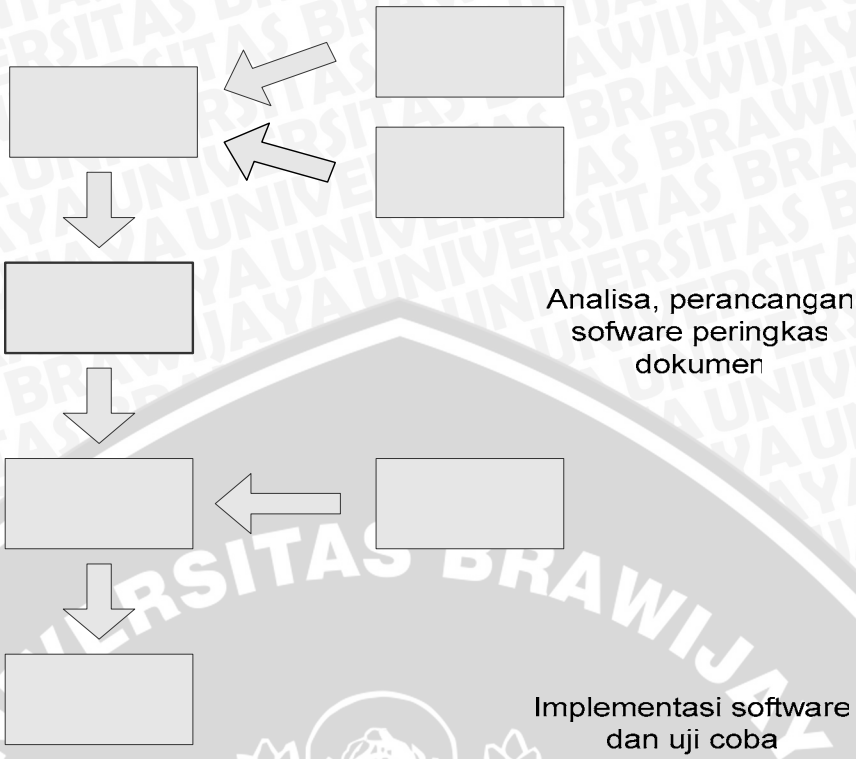
$$fall - out = \frac{b}{b+d}.$$

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab metodologi dan perancangan akan dibahas metode, rancangan yang digunakan dan langkah-langkah yang dilakukan dalam penelitian Ekstraksi informasi melalui *Query Oriented Multi Document Summarization* menggunakan *Latent Semantics Analysis.*.
Penelitian dilakukan dengan tahapan – tahapan:

1. Menganalisa dan merancang metode ekstraksi informasi berdasarkan penelitian sebelumnya mengenai *latent semantics analysis* oleh Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Launder, Richard Harshman dan *Query Oreinted multi-document summarization* oleh Ben Hachey, Gabriel Murray, David Reitter.
2. Membuat perangkat lunak pendukung penelitian berdasarkan analisis dan perancangan yang telah dilakukan.
3. Uji coba perangkat lunak pengekstrak informasi, dengan menggunakan sejumlah dokumen berita berbahasa indonesia yang memiliki topik yang berbeda-beda. Dokumen berita yang dijadikan sumber diambil dari *http://ilps.science.uva.nl/ Resources/ BI/index.html*
4. Evaluasi hasil ekstraksi yang dibuat oleh sistem, berdasarkan perbandingkan kemunculan kalimat dalam ringkasan sistem dengan kalimat hasil ideal. Karena sebenarnya tidak ada hasil yang benar-benar ideal, maka hasil buatan manusia dianggap sebagai hasil yang ideal.

Langkah – langkah penelitian ini dapat dilihat pada Gambar 3.1 :



Gambar 3.1 Diagram alir penelitian

3.1 Analisa Sistem

Pada subbab analisa sistem akan dibahas berbagai hasil analisis terhadap sistem dan elemen-elemen yang terkait, seperti *user*, *evaluator,* dan semua yang diperlukan dalam proses ekstraksi informasi.

3.1.1 Deskripsi Umum Sistem

Pengekstrak informasi yang akan dibuat merupakan sistem yang menghasilkan hasil ekstraksi yang informatif dari sekumpulan dokumen berdasarkan *query* yang dimasukkan oleh user. Untuk pemilihan kalimat yang akan diekstrak digunakan perhitungan similaritas antar kalimat melalui analisa semantik. Untuk

mendapatkan analisa semantik dari dokumen-dokumen yang menjadi sumber digunakan *latent semantic analysis.*

Ketika ekstraksi informasi dijalankan, proses yang dilakukan adalah:

- *User* memberikan *query*, menentukan ukuran hasil ekstraksi (jumlah kalimat yang ingin diekstrak dari ekstraksi informasi) dan menentukan dokumen-dokumen yang menjadi sumber.
- Mendaftar seluruh dokumen dan *term*. *Term* didapatkan dengan cara melakukan proses *stemming* yaitu proses mereduksi kata berimbuhan menjadi kata awal (kata kerja), *term* juga berupa kata benda. Sebelumnya dilakukan proses pemecahan kalimat dan penghilangan *stop word.*
- Membentuk *term*-dokumen matriks (matriks X) atau *vector space model*, yaitu matriks yang merepresentasikan jumlah *term* dalam sebuah dokumen (ukuran matrik adalah n term x n dokumen.
- Mengoperasikan *Singular Value Decomposition* pada matriks *term-sentence* (matriks X), sehingga dihasilkan matriks *term-to-term* (matriks U), matriks *singular* (matriks S), dan matriks *sentence-to-sentence* (matriks V^T). Dimana mattriks X adalah representasi *term* dalam tiap dokumen, matrix U adalah representasi hubungan *term* dengan *term*, dan matriks V^T adalah representasi hubungan *kalimat* dengan *kalimat*.
- Pembentukan ringkasan dilakukan dengan mengekstrak kalimat nilai skor terbesar. Skor dihitung dari rumus yang dimodifikasi dari penelitian Ben Hachey, Gabriel Murray dan David Reitter yaitu

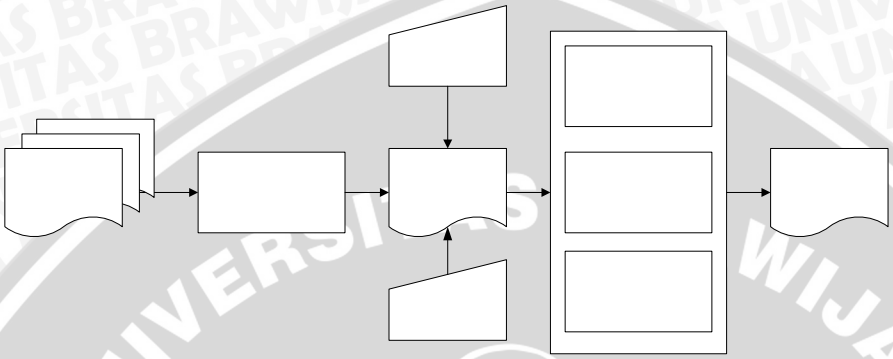
  $$Skor = \lambda \cdot sim1 + (1- \lambda) \cdot sim2$$

  Dimana sim1 adalah nilai similaritas antara kalimat dalam dokumen dengan kalimat *query* sedangkan sim2 adalah nilai similaritas rata-rata antara kalimat dalam dokumen dengan kalimat-kalimat yang telah diekstrak (dalam ringkasan). Nilai awal λ adalah 1 dan terus berkurang seiring dengan bertambahnya ukuran hasil ekstraksi (Hatchey, 2005).
- Penghitungan similaritas antar dua kalimat dapat dihitung dengan mencari nilai kosinus perpotongan vektor keduanya.Sedangkan pada dengan metode *Latent Semantic Analysis* dapat dirumuskan seperti pada rumus 3.1

$$Sim(di, dj) = \sum_{m=1}^{k} V_{im} V_{jm} s_m^2 \text{ [2]} \qquad (3.1)$$

- Analisa hasil ekstraksi sistem dengan memasukkan hasil ideal pada bagian *analysis*. Sistem kemudian menghitung jumlah kesamaan kalimat dan menghtung nilai *precision, recall, dan f-measure* serta *fall-out.*

Blok diagram dari sistem ditunjukkan oleh Gambar 3.2



Gambar 3.2 Blok Diagram Proses Umum Sistem

3.1.2 Batasan Sistem

Batasan dari sistem yang akan dikembangkan adalah :

1. Proses penghilangan *stopword* (kata umum) tergantung pada bahasa yang digunakan, karena itu sebagai batasan bahasa yang dicakup oleh sistem ini adalah bahasa Indonesia.
2. Daftar *blockword* (kata yang dilewatkan stemmer) yang digunakan sistem tergantung pada bahasa indonesia.
3. Proses Stemming tidak melibatkan kata-kata yang mempunyai *infix* (sisipan).

Dokumen-
dokumen asal

Penggabungan
dokumer

3.1.3 Analisis Kebutuhan Sistem

Analisis kebutuhan ini didasarkan pada pembangunan sistem pengekstrak dokumen yang dapat menerapkan metode yang digunakan dan hasil yang didapatkan mewakili hasil buatan manusia (dimana hasil buatan manusia dianggap hasil ideal). Kebutuhan perangkat ini meliputi hal pokok yang harus ada dan tambahan.

Hal pokok :

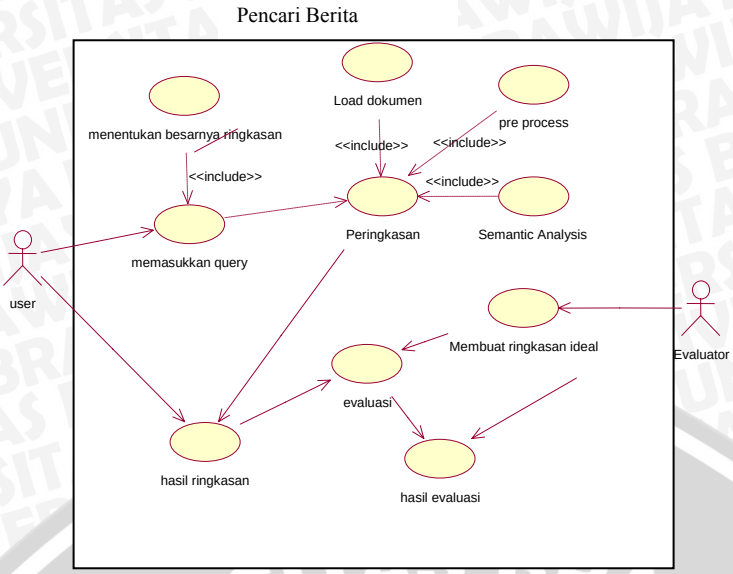
- Sistem dapat membaca multi-dokumen.
- Sistem dapat menerima *query* dan ukuran hasil ekstraksi.
- Sistem dapat menghasilkan hasil ekstraksi sesuai dengan algoritma yang telah ditentukan.
- Sistem dapat menghasilkan informasi yang mewakili hasil buatan manusia.

Tambahan :

- Interaksi pengguna dengan sistem akan melalui GUI (*Graphical User Interface*).

3.1.4 Analisis *Use Case*

Dalam sistem pengekstrak informasi ini *user* yang akan mendapatkan hasil ekstraksi berupa ringkasan dari beberapa dokumen yang sesuai dengan *query* yang dimasukkannya, selanjutnya *user* juga harus menentukan besarnya hasil ekstraksi Sedangkan untuk mengevaluasi hasil pencairan yang dihasilkan sistem dibutuhkan beberapa *evaluator*. Gambaran sistem dari sisi *user* dapat dilihat dalam Gambar 3.3:



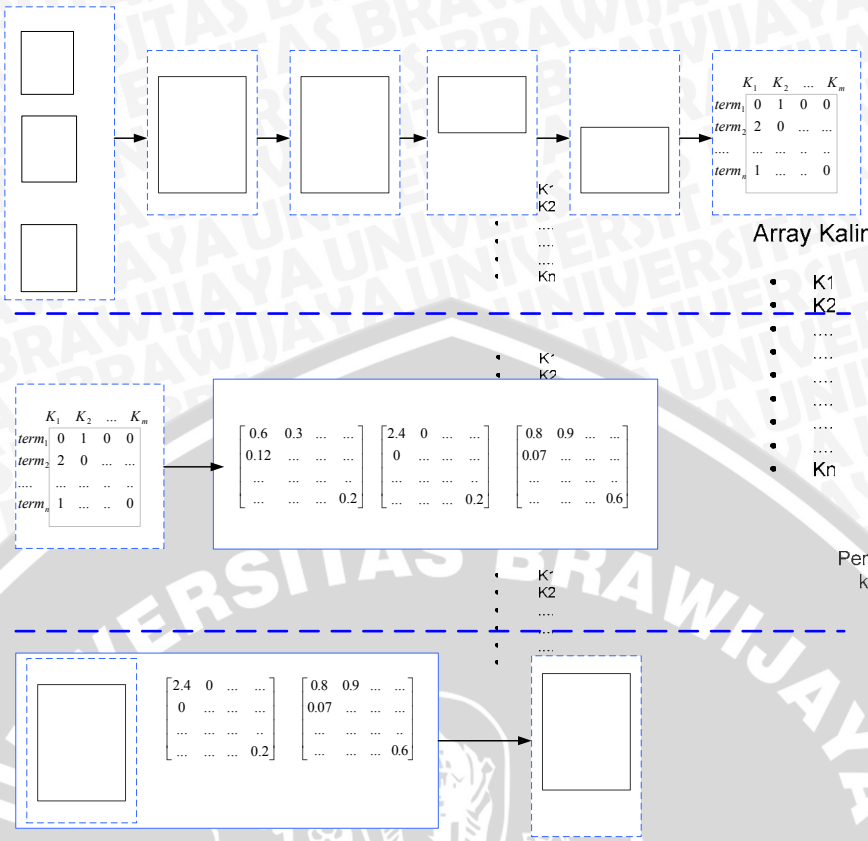
Gambar 3.3 Diagram *Use Case*

3.2 Perancangan Sistem

Berdasarkan analisis yang telah dilakukan, pada Subbab 3.2.1 akan dibahas mengenai arsitektur dan proses yang terjadi pada sistem pengekstrak informasi yang akan dibangun ini.

3.2.1 Perancangan Proses

Seperti yang telah dijelaskan dalam blok diagram sistem pada Gambar 3.2, sistem yang akan dibangun terdiri dari proses-proses yang lebih kecil. Proses-proses yang ada dalam sistem dapat diilustrasikan sebagai dalam Gambar 3.4.



Gambar 3.4 Ilustrasi Proses pada Sistem

Dokumen-dokumen yang akan diringkas, pada awalnya dipecah menjadi kalimat dan *term*, dimana *term* adalah kata yang telah mengalami proses *stemming* (reduksi kata menjadi kata dasar), proses ini terdapat dalam *preprocessing*. Seperti yang sudah dijelaskan dalam Subbab 2.2.1 bahwa data teks yang akan diproses tidak dapat secara langsung mengalami pengolahan data. Oleh karena itu data text ditransformasi menjadi data numerik melalui proses-proses :

3.2.1.1 Preprocess

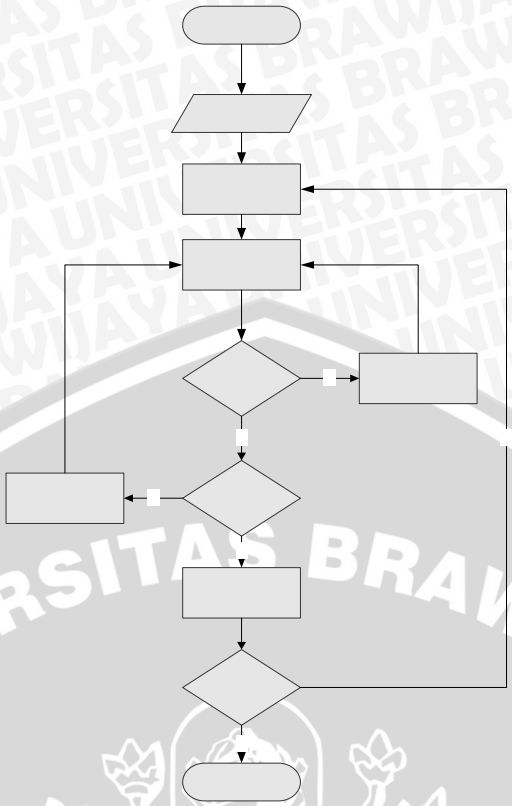
Dikarenakan metode *Latent Semantics Analysis* tidak memperhitungkan pembobotan judul maupun kata kunci, maka semua bagian dari dokumen dianggap sama. Pada *preprocess*, dokumen dipecah menjadi kalimat-kalimat dan kata-kata. Daftar kata yang diperoleh diolah lebih lanjut dalam proses *stemming* untuk mendapatkan *term*.

1. Pemecahan Kalimat

Untuk memecah (*parsing*) teks yang masuk menjadi kalimat digunakan beberapa batas penentu sebuah kalimat (*sentence delimiter*). Beberapa hal yang bisa digunakan untuk menentukan akhir dari sebuah kalimat :

- Sebuah kalimat diakhiri dengan 1 atau lebih titik (.), tanda seru (!), tanda tanya (?).
- *Sentence delimiter* biasanya diikuti oleh tanda petik dua, seperti pada contoh : *Ibu berkata : "Jangan pulang sore-sore!"*
- Delimiter terakhir harus diikuti oleh satu atau lebih *white space*(spasi, tab, *newline*).
- Kata pertama kalimat berikutnya seharusnya dimulai dengan huruf kapital.
- *Sentence delimiter* tidak seharusnya menjadi bagian dari singkatan (*abbreviation*).

Perancangan algoritma pemecahan kalimat dapat dilihat pada Gambar 3.5



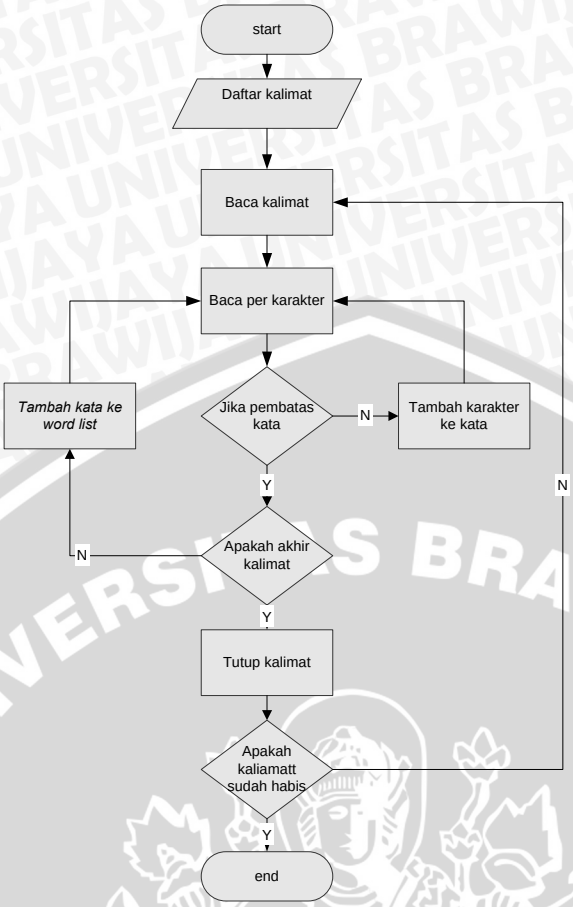
Gambar 3.5 Flow Chart Pemecahan Kalimat

2. tokenization.

Didalam proses ini dilakukan pemisahan dokumen menjadi kata-kata. Mengikuti pemecahan kalimat, pemecahan kalimat menjadi kata-kata tunggal juga dilakukan, yaitu dengan men-*scan* kalimat dengan pemisah *white space* (spasi, tab, *newline*). Sebuah kata didefinisikan sebagai sebuah runtutan karakter - karakter ([A-Z], [a-z] dan / atau [0,9]).. Perancangan algoritma dalam proses ini dapat dilihat dalam gambar 3.6

Tambah kalimat ke array

N



Gambar 3.6 Flow Chart *tokenization*

6. *filtration*

Pemilihan kosa kata dilakukan dalam tahapan ini. Yang dilakukan adalah dengan menghilangkan kata-kata yang tidak penting seperti kata penghubung, kata tanya dan lain sebagainya yang dikenal dengan *stopword*.

Stop word merupakan kata- kata yang tidak berpengaruh atau tidak memiliki arti penting untuk menentukan topik dari suatu teks. Yang termasuk ke dalam *stop word* adalah kata ganti orang

(saya, kamu, dia, anda, engkau,...), kata hubung (dan, tidak, atau,...), kata depan (di, ke, pada,...).

Daftar kata *stop word* yang digunakan pada sistem peringkas dokumen ini diambil dari paper Fadillah Z Tala dengan judul *A Study of Stemming effect on Information Retrieval in Bahasa Indonesia*. Dalam sistem ini penghilangan *stop word* dilakukan ketika memecah kalimat menjadi kata-kata. Daftar *stopword* yang dihilangkan terdapat pada lampiran.

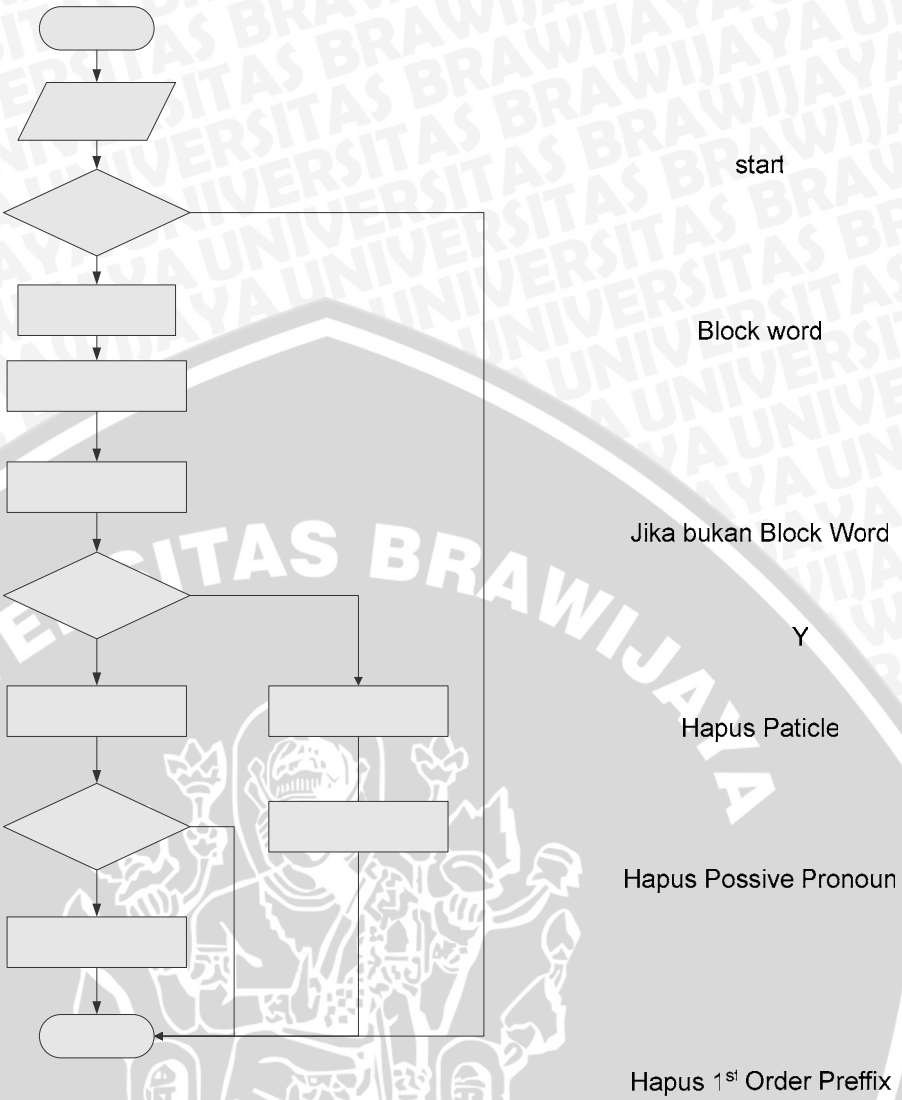
4. Stemming

Proses ini berfungsi untuk mengolah kata yang berimbuhan, baik *suffix* (akhiran), *preffix* (awalan) maupun kombinasi keduanya. Kata-kata yang diproses dalam stemming adalah kata kerja bukan kata benda atau kata dari bahasa asing yang memiliki unsur *preffix* atau *inffix* (*block word*)

Peraturan proses stemming dalam bahasa indonesia telah dijelaskan dalam daftar pustaka (dengan metode *purely rule-based stemming*). Proses stemming memiliki langkah-langkah:

1. Pemeriksaan *block word*
2. Pemeriksaan adanya *affix* (imbuhan).
3. Penghapusan *particle*.
4. Penghapusan *possesive pronoun*.
5. Penghapusan *preffix*.
6. Penghapusan *suffix*, *2nd order suffix* atau *2nd prefix*.

Gambar 3.7 menunjukkan Flow Chart Stemming pada bahasa Indonesia.

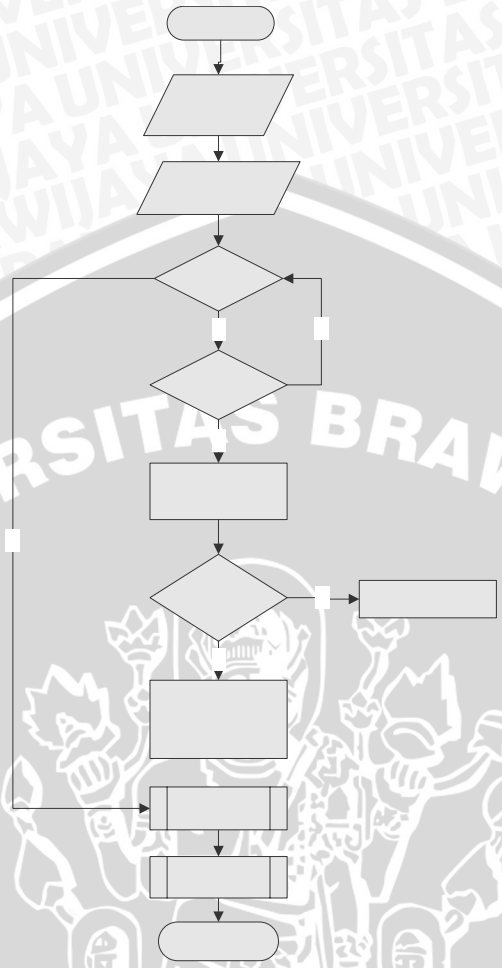


Gambar 3.7 Flow Chart stemming

5. Pembobotan

Tahap awal yang dilakukan dalam pembobotan adalah menghitung jumlah masing-masing kata sehingga diperoleh nilai *tf*. Semua kata yang ada dalam basis data diambil untuk menyusun *vector space model*. Kemudian dari vektor itu dapat

digunakan untuk menghitung pembobotan lebih lanjut, yaitu perhitungan *idf* yang dilanjutkan dengan perhitungan bobot *tfidf*. *Flow chart* pembobotan ditunjukkan pada Gambar 3.8.



Gambar 3.8 Flowchart Proses pembobotan .

3.2.1.2 Pengolahan Data

Melalui *preprocess* telah didapatkan *vector space model* berupa matriks representasi *term-sentence*. Dalam pengolahan data akan diolah matriks representasi *term-sentence* untuk mendapatkan analisa semantik dari data. Proses pengolahan data yang dilakukan

adalah *Latent Semantic Analysis* dengan cara mengopersikan *Singular Value Decomposition* pada matriks representasi *term-sentence* yang selanjutnya disebut matriks X.

Untuk mengoperasikan *Singular Value Decomposition* pada matriks X (m x n), diperlukan syarat $m >= n$. Dari proses ini matriks X(m x n) dipecah menjadi matriks U (m x m), S (n x n) dan V^T (n x n). Dimana matriks U didapatkan dari nilai eigen vektor pada matriks simetrik kovarian XX^T, matriks V adalah nilai dari eigen vektor pada matriks simetrik kovarian X^TX.

Perancangan *Singular Value Decomposition* diperoleh dari *http://unt.edu./~scott/LSA/SingularValueDecomposition.java*

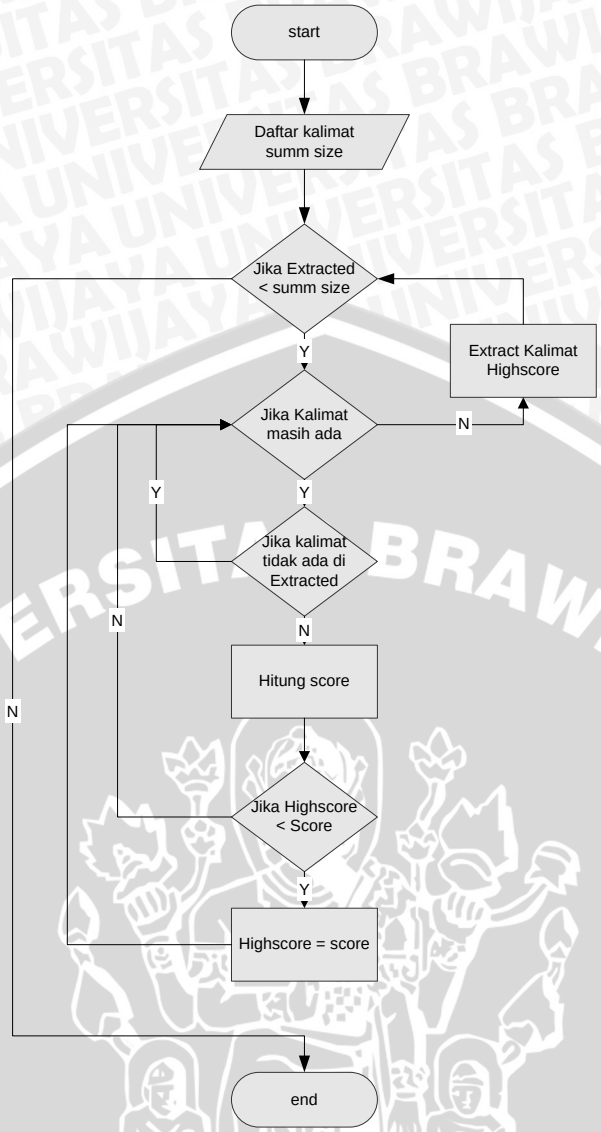
3.2.1.3 Pembentukkan Ringkasan

Ringkasan diperoleh dari mengekstrak kalimat yang memiliki kesesuaian dengan *query* dan rata-rata similaritas dengan kalimat yang telah diekstrak. Penentuan skor seperti yang dijelaskan dalam Subbab 2.2.3.1 yaitu dengan rumus *score=λ*sim1 + (1- λ)*sim2*. Dimana sim1 adalah similaritas antara kalimat dengan *query* dan sim2 adalah rata-rata similaritas kalimat dengan kalimat lain yang sudah terekstrak (Hatchey, 2005).

Nilai similaritas dua vektor umumnya dihitung dengan mencari nilai cosinus dari dua vektor. Dalam *latent semantic analysis* penghitungan similaritas dua vektor dapat dirumuskan dengan

$$Sim(di, dj) = \sum_{m=1}^{k} V_{im} V_{jm} s_m^2 \qquad (3.1)$$

dimana D adalah matriks V (perbandingan dokumen) dan S (nilai singular) (Deerweister, 1990). Perancangan algoritma ekstraksi pembetukan hasil dapat dilihat dalam Gambar 3.9

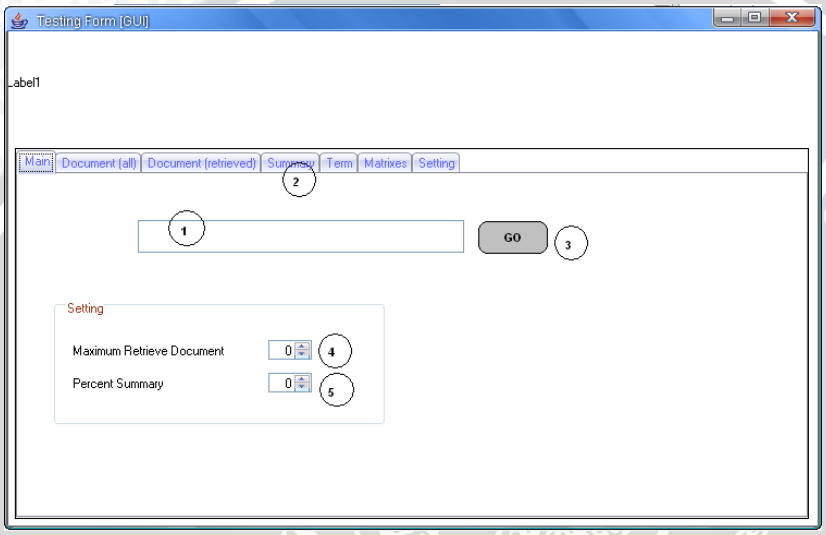


Gambar 3.9 Flow Chart Pembentukan hasil

3.2.2 Perancangan Antarmuka

Agar pelaksanaan ujicoba dengan menggunakan sistem dapat berjalan dengan baik, maka dibutuhkan antar muka yang *user friendly*. Antar muka dirancang agar pengguna dapat mudah mengerti cara penggunaannya. Untuk antar muka dalam sistem yang akan dibangun membutuhkan perangkat-perangkat :

1. *input text* untuk memasukkan *query*.
2. *text area* untuk menampilkan hasil ekstraksi
3. tombol untuk menjalankan proses.
4. *spin edit* untuk mengatur besarnya hasil (jumlah kalimat)
5. *list* untuk memunculkan daftar *term, stop word, block word* dan kalimat-kalimat yang diproses.

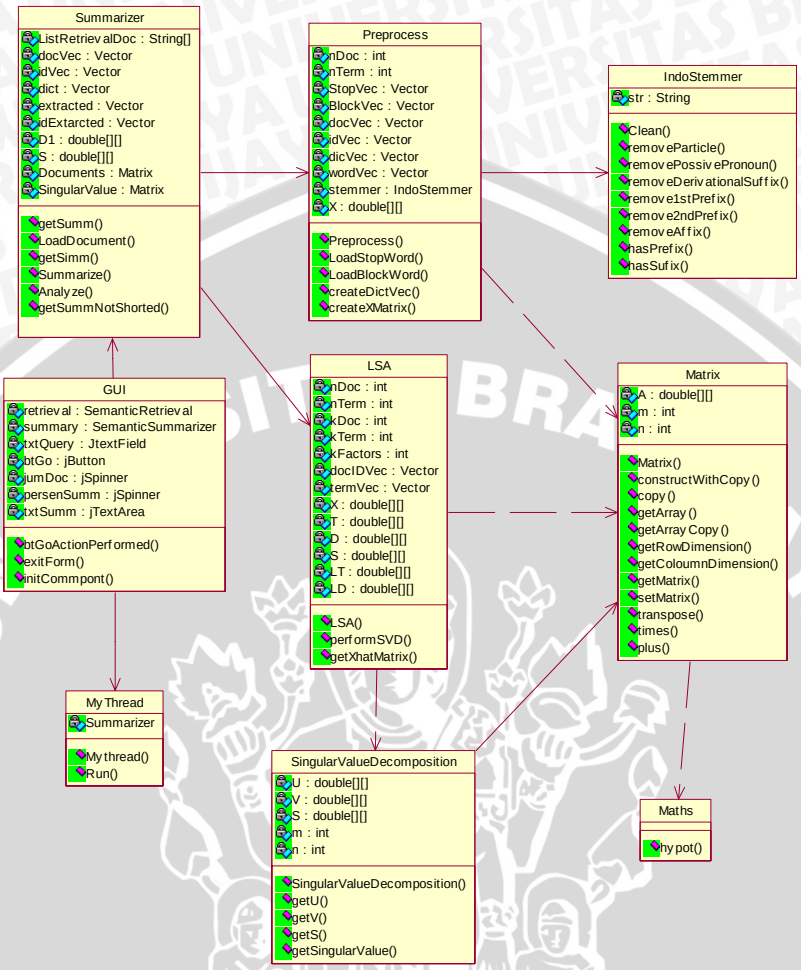


Gambar 3.10 Rancangan Graphical User Interface

3.2.3 Perancangan Kelas

Dari perancangan proses dan perancangan antar muka, kemudian dirancang kelas – kelas menampung semua kebutuhan baik dalam proses maupun antar muka. Perancangan kelas terdiri dari tiga kelas utama yaitu kelas yang menangani *preprocessing* yaitu kelas *preprocess,* kelas yang menangani pengolahan data yaitu kelas *LSA*

dan kelas untuk membentuk ringkasan yaitu kelas *Summarizer*. Perancangan kelas secara keseluruhan dapat diilustrasikan pada gambar 3.11



Gambar 3.11 Diagram Kelas

1. Kelas IndoStemmer
   Kelas ini berfungsi untuk menghasilkan *term* (*stemmeed word*) dengan cara mengolah kata yang mengandung awalan

(prefik) dan akhiran (sufik) dan tidak termasuk daftar *blockword* (kata benda atau kata kerja baku yang mengandung akhiran dan awalan). Kelas IndoStemmer hanya digunakan pada bahasa indonesia dan menerapkan metode *Purely Based Stemmer.*

2. Kelas Matrix
   Kelas ini memiliki fungsi untuk menyimpan data berupa array (2 dimensi) dan memiliki operasi-operasi dasar pada matriks seperti *transpose*, perkalian, penambahan.
3. Kelas SingularValueDecomposition
   Kelas ini berfungsi untuk mengoperasikan *Singular Value Decomposition* pada matriks masukan (matriks X) untuk menghasilkan matriks U, matriks S, dan matriks V.
4. Kelas Maths
   Kelas ini berisi fungsi matematika tambahan yang tidak disediakan oleh *library math* pada java.
5. Kelas Preprocess
   Kelas ini berfungsi untuk melakukan *preprocess* pada dokumen yang akan diolah lebih lanjut. Proses yang terdapat pada kelas diantaranya penghilangan *stopword*, pemecahan kalimat dan kata, eliminasi tanda baca dan pembentukan matriks representasi *term*-dokumen.
6. Kelas LSA
   Kelas ini digunakan untuk melakukan proses *Latent Semantic Analysis* dimana akan didapatkan matriks-matriks singular. Matriks-matriks tersebut yang nantinya digunakan kelas SemanticSummarizer.
7. Kelas Mythread
   Berfungsi menjalankan proses peringkasan dokumen sebagai thread.
8. Kelas Summarizer
   Kelas yang berfungsi meringkas dokumen-dokumen menjadi ringkasan tunggal dengan pembobotan *query* menggunakan kelas preprocess dan LSA. Hasil ringkasan akan dikirim ke *text Area* dalam kelas GUI.
9. Kelas GUI
   Kelas antar muka yang digunakan untuk berinteraksi dengan user. Merupakan class *Graphical User Interface* implementasi class *Jframe*. Kelas ini menerima input query

dari user dan besarnya hasil kemudian meringkas dokumen-dokumen dengan Summarizer.

3.3 Perancangan Uji Coba

Pada subbab ini akan dilakukan perancangan uji coba dari sistem pengekstrak informasi ini, baik pengujian terhadap sistem apakah telah sesuai dengan analisis dan perancangan, maupun evaluasi hasil ekstraksi yang dihasilkan. Hasil ekstraksi akan dievaluasi, meliputi evaluasi berdasarkan hasil ideal (hasil buatan manusia).

3.3.1 Bahan Pengujian

Bahan yang akan digunakan sejumlah dokumen berita berbahasa indonesia yang memiliki topik yang berbeda-beda. Dokumen berita yang dijadikan sumber diambil dari *http://ilps.science.uva.nl/ Resources/ BI/index.html*

3.3.2 Tujuan Pengujian

Beberapa hal yang menjadi tujuan dari pelaksanaan pengujian terhadap sistem pengekstrak informasi ini, yaitu :

1. Memeriksa kesesuaian hasil implementasi dengan hasil analisis dan perancangan.
2. Memeriksa perangkat lunak apakah telah berjalan baik (tidak terjadi *error*).
3. Mengevaluasi hasil sistem, dengan menghitung nilai *recall, precision* dan *f-measure* serta *fall-out*..

3.3.3 Skenario dan Kriteria Pengujian

Pengujian yang dilaksanakan pada meliputi pengujian proses dan pengujian hasil sistem. Hasil sistem kemudian dianalisa untuk menentukan baik tidak implementasi metode yang diterapkan

3.3.3.1 Pengujian Implementasi Kelas

Sesuai dengan tujuan pengujian pertama dan kedua maka pengujian bagian pertama ini befungsi untuk memeriksa fungsionalitas sistem dan kesesuaian dengan hasil analisis dan perancangan.

3.3.3.2 Evaluasi hasil

Untuk mengetahui kualitas hasil ekstraksi sistem, hasil ekstraksi akan dibandingkan dengan ringkasan ideal (dengan menitikberatkan pada *Query* pencarian). Seperti yang telah dituliskan pada tinjauan pustaka, bahwa tidak ada satupun ringkasan yang ideal (karena hasil ekstraksi merupakan hasil peringkasan), baik yang dibuat oleh sistem peringkas yang telah ada, bahkan yang dibuat oleh manusia sekalipun. Tetapi, dikarenakan terbatasnya kemampuan untuk memperoleh suatu model ideal, maka yang akan dipakai pada penelitian ini adalah ringkasan hasil buatan manusia. Oleh karena itu dokumen yang akan diringkas oleh sistem ini, terlebih dahulu diringkas oleh beberapa *abstractor*. Pihak *abstractor* yang digunakan dalam penelitian ini :

- Objek : 3 orang mahasiswa atau sarjana.
- Lingkungan : Universitas Brawijaya dari beberapa Fakultas dan Jurusan yang berbeda.
- Metode : Para mahasiswa ini diberi beberapa dokumen, kemudian dengan menitikberatkan pada *Query* mahasiswa tersebut akan memilih kalimat-kalimat mana saja dalam dokumen tersebut yang akan menjadi ringkasan. Ukuran (jumlah kalimat) dari ringkasan diusahakan mendekati ukuran yang telah ditentukan, sehingga pembandingan dengan hasil peringkasan sistem dapat dilakukan.

Dari Perbandingan hasil sistem dan *abstractor* tersebut akan disajikan dalam tabel 3.1 :

Tabel 3.1 Rancangan Tabel Hasil Perbandingan

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out

BAB IV
IMPLEMENTASI DAN UJI COBA

Implementasi merupakan proses transformasi representasi rancangan ke bahasa pemrograman yang dapat dimengerti oleh komputer. Pada bab 4 akan dibahas hal-hal yang berkaitan dengan implementasi sistem pengekstrak informasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dipaparkan dalam bab 4 meliputi lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem pengekstrak informasi adalah :

1. Prosesor AMD Athlon XP 1800 Mhz
2. RAM 512 MB
3. *Harddisk* dengan kapasitas 40 GB
4. Monitor 15”
5. Keyboard
6. Mouse

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem pengekstrak informasi adalah :

1. Sistem Operasi Windows XP SP2
2. *Java Development Kit* (JDK) versi 1.5.0.1
3. *Java Editor* NetBeans IDE versi 3.6
4. *Java Editor* Jcreator Pro versi 3.10.009

4.2 Implementasi Kelas

Berdasarkan hasil analisis kelas pada subbab 3.2 , mengenai analisa kelas-kelas dalam sistem pengekstrak informasi, maka pada subbab 4.2 akan dijelaskan implementasi kelas-kelas tersebut.

4.2.1 Kelas IndoStemmer

Untuk memperoleh kata dasar *(Stem)* dari kata yang diberikan. Untuk setiap penghilangan affiks diperiksa apakah kata tersebut termasuk *block word* atau bukan. Jika termasuk *block word* kata langsung dikembalikan tanpa penghilangan affiks terakhir.

Pada kelas IndoStemmer terdapat beberapa metode- metode:

- Clean ()
  Metode Clean berfungsi untuk menghilangkan angka dari kata yang diberikan.

```
private String Clean( String str ) {
     int last = str.length();
     Character ch = new Character( str.charAt(0) );
     String temp = "";
     for ( int i=0; i < last; i++ ) {
         if ( ch.isLetterOrDigit( str.charAt(i) ) )
            temp += str.charAt(i);
     }
     return temp;
  }
```

Gambar 4.1 *procedure* Clean

- hasSuffix ()
  Metode ini berfungsi untuk memeriksa apakah kata yang diberikan mengandung akhiran tertentu.

```
private boolean hasSuffix( String word, String suffix,
NewString stem ) {
     String tmp = "";
     if ( word.length() <= suffix.length() )
        return false;
     if (suffix.length() > 1)
     if ( word.charAt( word.length()-2 ) !=    suffix.charAt(
suffix.length()-2 ) )
           return false;
         ....
         ....
  }
```

Gambar 4.2 *procedure* hasSuffix

- hasPreffix ()
  Metode ini berfungsi untuk memeriksa apakah kata yang diberikan mengandung akhiran tertentu

```
private boolean hasPreffix( String word, String preffix,
NewString stem ) {
     String tmp = "";
     if ( word.length() <= preffix.length() )
        return false;
     if (preffix.length() > 1)
        if ( word.charAt( 1 ) != preffix.charAt( 1 ) )
           return false;
     stem.str = "";
   for ( int i=preffix.length(); i<word.length(); i++ )
         stem.str += word.charAt( i );
     tmp = stem.str;
     for ( int i=preffix.length()-1; i>=0; i-- )
         tmp = preffix.charAt( i ) + tmp;
     if ( tmp.compareTo( word ) == 0 )
        return true;
     else
        return false;
  }
```

Gambar 4.3 *procedure* hasPreffix

- removeParticle ()
  Metode ini berfungsi untuk menghilangkan *particle* dalam kata. Akhiran yang termasuk *particle* adalah akhiran lah, kah, tah, pun .

```
public String removeParticle (String str)
{
   NewString stem = new NewString();
   String[] particle = {"lah","kah","tah","pun"};
   for (int i=0;i<particle.length;i++)
  {
    if (hasSuffix(str,particle[i],stem))    {
         str = str.substring(0,(str.length()-
               particle[i].length()));
        return str;     }  }
  return str;
  }
```

Gambar 4.4 *procedure* removeParticle

- removePossivePronoun ()
  Metode ini berfungsi untuk menghilangkan *Possive Pronoun* dalam kata. Akhiran yang termasuk *Possive Pronoun* adalah akhiran ku, mu, nya.

```
public String removePossivePronoun (String str)
  {
        NewString stem = new NewString();
         String[] PossivePronoun = {"ku","mu","nya"};
         for (int i=0;i<PossivePronoun.length;i++)
         {
         if (hasSuffix(str,PossivePronoun[i],stem))         {

           str = str.substring(0,(str.length()-
           PossivePronoun[i].length()));
          return str;
         }
         }
         return str;
  }
```

Gambar 4.5 *procedure* removePossesivePronoun

- removeDerivationalSuffix ()
  Metode ini berfungsi untuk menghilangkan *Derivational Suffix* dalam kata. Akhiran yang termasuk *Derivational Suffix* adalah i, kan, an.

```
public String removeDerivationalSuffix (String str)
  {
     NewString stem = new NewString();
     String[] DerivationalSuffix = {"i","kan","an"};
         int hasDerivationalSuffix = 0;
          for (int i=0;i<DerivationalSuffix.length;i++)
          {
          if (hasSuffix(str,DerivationalSuffix[i],stem))     {

          str = str.substring(0,(str.length()-
         DerivationalSuffix[i].length()));
          return str;
          }
          }
          return str;
  }
```

Gambar 4.6 *procedure* removeDerivationalSuffix

- removePreffix ()
  Metode ini berfungsi untuk menghilangkan *preffix* dalam kata. Masukan yang diberikan adalah kata yang akan diproses, *preffix* yang akan dihilangkan dan huruf *replacement*.

```
public String removePreffix(String str, String preffix, char
replacement)
{
 NewString stem = new NewString();
 if (hasPreffix(str,preffix,stem))
 {
 if (replacement!='!')
 {
 str = str.substring(preffix.length(),str.length());
 str = replacement + str;
 }
 else
 str = str.substring(preffix.length(),str.length());
 }

 return str;
 }
```

Gambar 4.7 *procedure* removePreffix

- remove remove1stPreffix ()
  Metode ini berfungsi untuk menghilangkan *1st Preffix* dalam kata. Awalan yang termasuk *1st Preffix* adalah awalan me, pe, di, ter, ke dan peleburannya. Peleburan yang misalnya perubahan me menjadi men, meng dan meny.

```
public String remove1stPreffix (String str)
  {
         NewString stem = new NewString();
         String[] FirstPreffix = {"me","pe","di","ter","ke"};
         for (int i=0;i<FirstPreffix.length;i++)
         {
          if (hasPreffix(str,FirstPreffix[i],stem))
          {
           if (FirstPreffix[i].equals("me"))
           {
            if(hasPreffix(str,"men",stem))
            {
             if (hasPreffix(str,"meny",stem))
             {  str = removePreffix(str,"meny",'s');  }
             else if (hasPreffix(str,"meng",stem))
             {  str = removePreffix(str,"meng",'!');  }
             else
             { switch(str.charAt(3))
             {
        case 'a' : case 'i' : case 'u' : case 'e' : case 'o' :
         {str = removePreffix(str,"men",'t');}
         default :
         {str = removePreffix(str,"men",'!');}
              } }         }
         ....
         ....
         return str;
  }
```

Gambar 4.8 *procedure* remove1stPreffix

- removeremove2ndPreffix ()
  Metode ini berfungsi untuk menghilangkan *2nd Preffix* dalam kata. Awalan yang termasuk *2nd Preffix* adalah awalan ber, per dan peleburannya.

```
public String remove2ndPreffix (String str)
{
   NewString stem = new NewString();
   String[] SecondPreffix ={"ber","bel","be","per","pel","pe"};

  for (int i=0;i<SecondPreffix.length;i++)
  {
         str = removePreffix(str,SecondPreffix[i],'!');
         }

         return str;
  }
```

Gambar 4.9 *procedure* remove2ndPreffix

- Remove cekVowel ()
  Metode ini berfungsi untuk memeriksa berapa *vowel* yang dimiliki kata yang diberikan. *Vowel* adalah jumlah huruf vokal (a,i,u,e,o).

```
public int cekVowel(String str)
  {
    int countVowel = 0;
    for (int i=0;i<str.length();i++)
    {
        switch(str.charAt(i))
        {
            case 'a' :
            case 'i' :
            case 'u' :
            case 'o' :
            case 'e' :
                countVowel++;
        }
    }
    return countVowel;
  }
```

Gambar 4.10 *procedure* cekVowel

- Remove removeAffix ()
  Metode ini berfungsi untuk mengilangkan imbuhan pada kata yang diberikan. Algoritma yang digunakan dapat dilihat pada Gambar 3.6

```
public String removeAffix(String str, Vector blocword)
  {
        String prevString = "";
       String base = str;
       str = Clean(str);
       if(cekVowel(str)<2) str = base;
       else      base = str;
       if (!blocword.contains(str)) str = removeParticle(str);
       if(cekVowel(str)<2)  str = base;
       else    base = str;
       if (!blocword.contains(str))
        str = removePossivePronoun(str);
        ....
        ....
  }
```

Gambar 4.11 *procedure* removeAffix

4.2.2 Kelas Preprocess

Untuk menyiapkan data yang akan diproses lebih lanjut dengan memecah dokumen menjadi kalimat-kalimat, menghilangkan *stop word* , membentuk daftar *term* (menggunakan kelas IndoStemmer) dan menghitung frekuensi *term* pada tiap-tiap kalimat.
Atribut yang terdapat dalam kelas Preprocess ini adalah:

- ndocs : digunakan untuk menyimpan jumlah total kalimat.
- nterms : berfungsi menyimpan jumlah *term.*
- stopVec : digunakan untuk menyimpan daftar kata *stop word*, bertipe data Vector.
- stopfile : digunakan untuk menunjukkan alamat file *stop word.*
- blockVec : digunakan untuk menyimpan daftar kata *block word*, bertipe data Vector.
- blockfile : digunakan untuk menunjukkan alamat file *block word.*
- docVec : digunakan untuk menyimpan daftar kalimat, bertipe data Vector.
- dictVec: digunakan untuk menyimpan daftar *term*, kalimat, jumlah *term* **i** dalam kalimat **j** dan total *term* **i** dalam daftar *term*, bertipe data Vector

- docIndexVec : berfungsi untuk menyimpan *index* kalimat, bertipe data Vector.
- docIDVec : digunakan untuk menyimpan ID kalimat, bertipe data Vector.
- wordVector : digunakan untuk menyimpan *term*, bertipe data Vector.
- tfVec : berfungsi untuk menyimpan pembobotan tf tiap-tiap *term* dalam kalimat, bertipe data Vector.
- docNormWeightVec : berfungsi untuk menyimpan data normalisasi dari pembobotan kalimat, bertipe data Vector.
- Stemmer : stemmer yang digunakan, bertipe data IndoStemmer.
- docWeightingMethod : berfungsi untuk melakukan pilihan penghitungan bobot kalimat
- activatePorterStemmer : berfungsi untuk melakukan pilihan pengaktifan stemmer, bertipe data boolean.
- eliminateUniqueTerms : berfungsi untuk melakukan pilihan mengeliminasi *term* yang unik (hanya muncul di satu kalimat saja), bertipe data boolean.
- X : digunakan untuk menyimpan matriks *term*-kalimat. Bertipe data double[][].
- Xtranspose : digunakan untuk menyimpan transpose matriks *term*-kalimat. Bertipe data double[][].

Pada kelas Preprocess terdapat metode-metode :

- Preprocess()
  Merupakan konstruktor yang berisi fungsi menginisialisasi data-data.

```
public Preprocess() throws
FileNotFoundException,IOException{
      ndocs = 0;
      stopVec = new Vector();
      stopfile = new String();
      blockVec = new Vector();
      blockfile = new String();
      stemmer = new IndoStemmer();
      docWeightingMethod = new String();
      queryWeightingMethod = new String();
      activatePorterStemmer = true;
      eliminateUniqueTerms = true;
   }
```

Gambar 4.12 *class* Preprocess

- loadStopwords()
  Metode ini berfungsi untuk meload daftar stopword dari file teks dan disimpan ke dalam stopVec.

```
public void loadStopwords(String sf) throws
FileNotFoundException, IOException{
      stopfile = sf;
      FileReader fr3 = new FileReader(sf);
      BufferedReader br3 = new BufferedReader(fr3);
      stopVec = readLines(br3);
      fr3.close();
   }
```

Gambar 4.13 *procedure* loadStopwords

- loadBlockWord()
  Metode ini berfungsi untuk meload daftar stopword dan disimpan ke dalam stopVec.

```
public void loadStopwords(String sf) throws
FileNotFoundException, IOException{
      stopfile = sf;
      FileReader fr3 = new FileReader(sf);
      BufferedReader br3 = new BufferedReader(fr3);
      stopVec = readLines(br3);
      fr3.close();
   }
```

Gambar 4.14 *procedure* loadBlockWord

- setAutoDocIDs()
  Metode ini berfungsi untuk membuat ID Doc secara otomatis berdasarkan urutan. *Precedure* digunakan jika pada pembacaan docVec tidak mendefinisikan docIDVec.

```
public void setAutoDocIDs(){
      String textID = new String();
      for(int i=1; i < docVec.size(); i++){
         textID = Integer.toString(i);
         docIDVec.addElement(textID);
      }
   }
```

Gambar 4.15 *procedure* setAutoDocIDs

- getTokens ()
  Metode ini berfungsi untuk memproses kalimat menjadi kata-kata dan memisahkan dari daftar kata *stopword* yang kemudian melalui proses *stemming* . Hasil dari proses ini disimpan dalam Token Vector.

```
private Vector getTokens(String s){
      Vector tokenVec = new Vector();
      int minusIndex = -1;
      int pointIndex = -1;
      int tagStart, tagEnd = 0;
      int tagFlag = 0;
      String ts = new String();
      Character pointChar = new Character('.');

      if(s.length() > 0){
         s = s.toLowerCase();
         ....
         ....
      StringTokenizer st = new StringTokenizer(s," ");
      while(st.hasMoreTokens()){
        String st2 = new String();
        st2 = st.nextToken(" ");
        Character lastOfToken = new
                    Character(st2.charAt(st2.length() - 1));
        if(lastOfToken.equals(pointChar)) st2 =
                          st2.substring(0,st2.length() - 1);
        if(st2.length() > 0){
          if(!(Character.isDigit(st2.charAt(0))) &&
                                !(stopVec.contains(st2))){
          if(activatePorterStemmer) st2 =
                          stemmer.removeAffix(st2,blockVec);
          if((st2.length() > 0) &&
                      !(Character.isDigit(st2.charAt(0))))
          tokenVec.addElement(st2);
               }
            }
}
```

Gambar 4.16 *procedure* getTokens

- createDictPost()
  Digunakan untuk memproses dokumen menjadi data *dictionary* yaitu dengan menghitung frekuensi *term* . implentasi dari metode createDictPost dapat dilihat pada gambar 4.17.

```
public void createDictPost(){
      Vector mergedVec = new Vector();
      String text = new String();


      if(ndocs > 0){
         text = (String) docVec.elementAt(0);
         Vector tokenVec1 = getTokens(text);
         Vector sortedVec1 = quickSort(tokenVec1);
         Vector freqVec1 = freqCount(sortedVec1, 0);

         for(int i=1; i < docVec.size(); i++){
            text = (String) docVec.elementAt(i);
            int pos = 0;
            Vector tokenVec2 = getTokens(text);
            Vector sortedVec2 = quickSort(tokenVec2);
            Vector freqVec2 = freqCount(sortedVec2, i);
            mergedVec = mergeSort(freqVec1, freqVec2);
            freqVec1 = mergedVec;
         }
          if(eliminateUniqueTerms){
            int i = 0;
            int remaining = mergedVec.size();
            boolean streak = false;
            String ls1 = new String();
            String ls2 = new String();
            String term1 = new String();
            String term2 = new String();

            while(remaining > 0){
               if(remaining == 1){
                  if(!streak) mergedVec.removeElementAt(i);
                  remaining--;
               }
               else{
                  ls1 = (String)mergedVec.elementAt(i);
                  StringTokenizer st1 = new
                                    StringTokenizer(ls1);
                  term1 = st1.nextToken();
                  ls2 = (String)mergedVec.elementAt(i + 1);
                  StringTokenizer st2 = new
                                    StringTokenizer(ls2);
                  term2 = st2.nextToken();
                  if(term1.equals(term2)){
                     i++;
                     remaining--;
                     streak = true;
                  }
                  else
                  if(!streak) mergedVec.removeElementAt(i);
                     else i++;
                     streak = false;
                     remaining--;
                  } }
            }  }
```

Gambar 4.17 *procedure* createDictPost

- quickSort()
  Metode ini berfungsi untuk melakukan sorting pada *Vector* dengan metode q*uick sort.*

```
private Vector quickSort(Vector inputVec){
      Vector sortedVec = new Vector();
      Object [ ] a = inputVec.toArray();
      RecursiveQuicksort( a, 0, a.length - 1 );
      sortedVec.removeAllElements();
      for(int i=0; i < a.length; i++)
         sortedVec.addElement(a[i]);
      return sortedVec;
   }
```

Gambar 4.18 *procedure* quickSort

- freqCount()
  Metode ini berfungsi untuk melakukan penghitungan banyaknya frekuensi pada *Vector* dengan cara menghitung duplikasinya.

```
private Vector freqCount(Vector v, int docIndex){
      Vector result = new Vector();
      String ws = new String();
      String wfs = new String();
      int freq = 1, index = 0, pos;
      int n = v.size();
      for(pos=0; pos<n; pos = pos+freq){
         freq = 1;
         while(freq + pos != n){

if(v.elementAt(pos).equals(v.elementAt(pos+freq)))
               freq++;
            else break;
         }
         ws = (String) v.elementAt(pos);
         wfs = ws + " " + docIndex + " " + freq;
         result.insertElementAt(wfs, index);
         index++;
      }
      return result;
   }
```

Gambar 4.19 *procedure* freqCount

- mergeSort()
  Metode ini berfungsi untuk melakukan *merge sort* pada dua *vector*.

```
private Vector mergeSort(Vector v1, Vector v2){
      int pos1 = 0;
      int pos2 = 0;
      int comp;
      String s1 = new String();
      String s2 = new String();
      Vector v3 = new Vector();

      while(pos1 < v1.size() && pos2 < v2.size()){
         s1 = (String)v1.elementAt(pos1);
         s2 = (String)v2.elementAt(pos2);
         comp = s1.compareTo(s2);
         if(comp < 0){
            v3.addElement((String)v1.elementAt(pos1));
            pos1++;
         }
         else{
            v3.addElement((String)v2.elementAt(pos2));
            pos2++;
         }
      }

      if(pos1 == v1.size()){
         while(pos2 < v2.size()){
            v3.addElement((String)v2.elementAt(pos2));
            pos2++;
         }
      }
      else if(pos2 == v2.size()){
         while(pos1 < v1.size()){
            v3.addElement((String)v1.elementAt(pos1));
            pos1++;
         }
      }
      return v3;
   }
```

Gambar 4.20 *procedure* mergeSort

- create splitMerged ()
  Metode ini berfungsi untuk memisahkan mergedVec menjadi *Vector* melalui normalisasi dan melakukan pembobotan. mergedVec adalah vektor hasil *sorting* pada gambar 4.20, sedangkan implementasi dari metode splitMerged dapat dilihat pada gambar 4.21

```
private void splitMerged(Vector mv, Vector dv, Vector
docIndexVector, Vector tfVector, Vector wVector,
 int N){
      int mvIndex = 0, dictIndex = 0;
      int docCount = 1, wordFreq, postIndex, docIndex,
wordDocFreq1, wordDocFreq2;
      double idf = 0.0, weight, pij, Gi, realN = 0.0, di =
0.0, ni = 0.0;
      String ls1 = new String();
      ....
      ....
      String w2 = new String();
      int n = mv.size();
      for(mvIndex = 0; mvIndex < n; mvIndex = mvIndex +
docCount){
         docCount = 1;
         wordFreq = 0;

         ls1 = (String)mv.elementAt(mvIndex);
         StringTokenizer st1 = new StringTokenizer(ls1);
         w1 = st1.nextToken();
         docIndex = Integer.parseInt(st1.nextToken());
         wordDocFreq1 = Integer.parseInt(st1.nextToken());
         wordFreq = wordFreq + wordDocFreq1;
         postIndex = mvIndex;
docIndexVector.addElement(Integer.toString(docIndex));
tfVector.addElement(Integer.toString(wordDocFreq1));

 while(mvIndex + docCount != n){
         ls2 = (String)mv.elementAt(mvIndex + docCount);
StringTokenizer st2 = new StringTokenizer(ls2);
        w2 = st2.nextToken();
        docIndex = Integer.parseInt(st2.nextToken());
         wordDocFreq2 = Integer.parseInt(st2.nextToken());

         if(w1.equals(w2)){

docIndexVector.addElement(Integer.toString(docIndex));

tfVector.addElement(Integer.toString(wordDocFreq2));
      wordFreq = wordFreq + wordDocFreq2 ;
docCount++;                                              }
      else break;
}
 if(docWeightingMethod.equals("idf")){
realN = Double.parseDouble(String.valueOf(N));
di = Double.parseDouble(String.valueOf(docCount));
ni = Double.parseDouble(String.valueOf(wordFreq));
idf = (Math.log(realN/ni))/(Math.log(2));
for(int p = postIndex; p < postIndex + docCount; p++){
weight=idf* Integer.parseInt((String)tfVector.elementAt(p));
            wVector.addElement(Double.toString(weight));
           }
 }
....
....
```

Gambar 4.21 *procedure* splitMerged

- createXmatrix()
  Metode ini berfungsi untuk menciptakan *vector space model* berupa matrix *term-sentence* .

```
public void createXmatrix() {
      X = new double[nterms][ndocs];
      Xtranspose = new double[ndocs][nterms];
      int docs, d1, postIndexOffset;
      double weight;

      for(int t=0; t < nterms; t++){
         for (int d=0; d < ndocs; d++){
            X[t][d] = 0.0;
            Xtranspose[d][t] = 0.0;
         }
      }
```

Gambar 4.22 *procedure* mergeSort

4.2.3 Kelas LSA

Atribut yang terdapat dalam kelas LSA ini adalah:

- ndocs : untuk menyimpan jumlah dokumen yang diproses.
- nterms : untuk menyimpan jumlah *term* yang terdapat dalam dokumen.
- kfactors : *keep factor* jika tidak diinisialisasi bernilai nilai minimum dari nterm dan ndocs
- kterms : *keep factor* untuk *term*.
- kdocs : *keep factor* untuk dokumen.
- docIDVec : Vector untuk menyimpan ID dokumen.
- termVec : Vector untuk mnyimpan *term.*
- X : matriks *term*-dokumen.
- T : matriks *term*.
- D : matriks dokumen.
- S : matriks *Singular*
- LT : matriks *Loading term.*
- LD : matriks *Loading* dokumen.

```
public class LSA{
   private int ndocs;
   private int nterms;
   private int kfactors;
   private int kterms;
   private int kdocs;
   private Vector docIDVec;
   private Vector termVec;
   private double[][] X;
   private double[][] T;
   private double[][] D;
   private double[][] S;
   private double[][] LT;
   private double[][] LD;
```

Gambar 4.23 *class* LSA

Pada kelas LSA terdapat metode-metode :

- LSA ()
  Merupakan konstruktor yang berisi inisialisasi data-data.

```
public LSA(double[][] xm, Vector tv, Vector didv){
      X = xm;
      termVec = tv;
      docIDVec = didv;
      nterms = tv.size();
      ndocs = didv.size();
      kfactors = 1;
   }
```

Gambar 4.24 *constructor* LSA

- performSVD()
  Metode ini berfungsi untuk mengoperasikan *Singular Value Decomposition.* Dan menghasilkan matrix D, S, dan T.

```
public void performSVD(){
      Matrix X0, X1;
      X1 = new Matrix(X);
      if (ndocs > nterms) {
         X0 = new Matrix(ndocs, nterms);
         X0 = X1.transpose();
         kfactors = nterms;
}
      else {
         X0 = new Matrix(nterms, ndocs);
         X0 = X1;
         kfactors = ndocs
          }
SingularValueDecomposition svd = new
SingularValueDecomposition(X0);
      Matrix D0 = svd.getV();
      Matrix T0 = svd.getU();
      if (ndocs > nterms) {
         T = D0.getArray();
         D = T0.getArray();
      }
      else {
         D = D0.getArray();
         T = T0.getArray();
      }
      Matrix S0 = svd.getS();
      S = S0.getArray();
   }
```

Gambar 4.25 *procedure* performSVD

- keepKFactors ()
  Merupakan *procedure* menyimpan k *factor*. Jika k *factor* tidak didefinisikan maka nilainya adalah nilai minimal antara jumlah *term* dan jumlah kalimat. K faktor mempengaruhi ukuran matriks singular (matriks singular berukuran k x k). Implementasi dari *prosedure* keepKFactors dapat dilihat pada gambar 4.26

```
public void keepKfactors(int k)
      throws NullPointerException{

      if (ndocs>nterms) {
         if (k>nterms) kfactors = nterms;
         else kfactors = k;
      }
      else {
         if (k>ndocs) kfactors = ndocs;
         else kfactors = k;
      }
      Matrix D1 = new Matrix(ndocs,kfactors);
      double[][] D2 = D1.getArray();
      try {
         for (int i = 0; i < ndocs; i++) {
            for (int j = 0; j < kfactors; j++) {
               D2[i][j] = D[i][j];
            }
         }
      } catch(ArrayIndexOutOfBoundsException e) {
         throw new ArrayIndexOutOfBoundsException("Submatrix
indices");
      }
      D = D2;
      Matrix T1 = new Matrix(nterms,kfactors);
      double[][] T2 = T1.getArray();
      try {
         for (int i = 0; i < nterms; i++) {
            for (int j = 0; j < kfactors; j++) {
               T2[i][j] = T[i][j];
            }
         }
      } catch(ArrayIndexOutOfBoundsException e) {
         throw new ArrayIndexOutOfBoundsException("Submatrix
indices");
      }
      T = T2;
      Matrix S1 = new Matrix(kfactors,kfactors);
      double[][] S2 = S1.getArray();
      try {
         for (int i = 0; i < kfactors; i++) {
            for (int j = 0; j < kfactors; j++) {
               S2[i][j] = S[i][j];
            }
         }
      } catch(ArrayIndexOutOfBoundsException e) {
         throw new ArrayIndexOutOfBoundsException("Submatrix
indices");
      }
      S = S2;
   }
```

Gambar 4.26 *procedure* keepKfactors

4.2.4 Kelas Matrix

Kelas ini berisi operasi-operasi matriks dasar. Seperti transpose, perkalian, penambahan dan lain-lain. Kelas ini didownload dari *http://unt.edu/LSA/Matrix.java*

4.2.5 Kelas Singular Value Decomposition

Kelas ini berisi pengoperasian *Singular Value Decomposition.* Kelas ini didownload dari *http://unt.edu/LSA/SingularValue Decomosition.java*

4.2.6 Kelas Summarizer

Kelas ini merupakan kelas yang berfungsi menghasilkan *summary* dari dokumen-dokumen yang diberikan. Kelas ini memiliki atribut-atribut :

- vsm : Kelas Preproses.
- docVec : Vector kalimat.
- idVec : Vector ID Kalimat.
- outputVec : Vector Output.
- dict : Vector Dictionary.
- Summ : Vector Summary
- idSumm : Vector ID Summary
- idDocSumm : Vector ID Doc Summary
- Extracted : Vector Extracted.
- IDExtracted : Vector ID Extracted.
- D1 : Matriks relasi kalimat.
- S : Matriks Singular.
- k : keep factor.
- Percent : persentase summary
- nDoc : jumlah kalimat.
- posQuery : posisi Query pada matriks term-sentence
- Process : proses yang dijalankan
- Stage : urutan dari proses

```
public class Summarizer {
      Preprocess vsm;
      Vector docVec = new Vector();
      Vector idVec = new Vector();
      Vector outputVec = new Vector();
      Vector dict = new Vector();
      Vector Summ = new Vector();
      Vector idSumm = new Vector();
      Vector idDocSumm = new Vector();
      Vector Extracted = new Vector();
      Vector IDExtracted = new Vector();
      double[][] D1;
      double[][] S;
      int k = 0;
      int percent;
      int nDoc;
      int posQuery;
      public String Process;
      public int Stage;
```

Gambar 4.27 *constructor* Summarizer

Metode-metode yang digunakan pada kelas ini

- CekFileType()
  Metode ini berfungsi untuk memeriksa tipe file yang diload. Hal ini untuk menghindari jenis file yang tidak dapat diparsing.

```
protected boolean CheckFileType(String NamaFile, String FileType)
        {
                String tmp = "";
                String CleanNamaFile = "";
        if ( NamaFile.length() <= FileType.length() )
        return false;
        if (FileType.length() > 1)
        if ( NamaFile.charAt( NamaFile.length()-2 ) !=
FileType.charAt( FileType.length()-2 ) )
                return false;

        for ( int i=0; i<NamaFile.length()-FileType.length(); i++
)
                CleanNamaFile  += NamaFile.charAt( i );
        tmp = CleanNamaFile;

        for ( int i=0; i<FileType.length(); i++ )
                tmp += FileType.charAt( i );
        if ( tmp.compareTo( NamaFile ) == 0 )
        return true;
        else
        return false;
        }
```

Gambar 4.28 *procedure* cekFileType

- LoadAllDocument ()
  Metode ini berfungsi untuk mengambil file-file yang berada pada folder tertentu.

```
protected void LoadAllDocument (String FileFolder) throws IOException
    {
        File Direktori = new File(FileFolder);
        String[] ListFile = Direktori.list();
        File Files;
        for (int i=0;i<ListFile.length;i++)
        {

                String FileName = FileFolder+(char)92+ListFile[i];
                Files = new File(FileName);

                if(Files.isDirectory())
                        LoadAllDocument(FileName);
                else
                {
                   String s, teks = "";
                  if(CheckFileType(FileName,"txt"))
                  {

                FileReader input  = new FileReader(FileName);
                BufferedReader streamInput = new
                                        BufferedReader(input);

                while ((s=streamInput.readLine())!=null)
                 {
                    if (s!=null)
                      teks += s + " ";
                }
                  input.close();
         }

        if(teks.length()>0)
        {
        BreakIterator bi = BreakIterator.getSentenceInstance();
        bi.setText(teks);
        int index = 0;
        String Kalimat;
        while (bi.next() != BreakIterator.DONE)
        {
           Kalimat = teks.substring(index, bi.current());
           docVec.addElement(Kalimat);
           idVec.addElement(ListFile[i]);
           index = bi.current();
        }
 }
```

Gambar 4.29 *procedure* LoadAllDocument

- getSimm ()
  Metode ini berfungsi mencari simiaritas antar dua kalimat

```
protected double getSimm(int Vector1, int Vector2)
{
   double SValue = 0;
    double Value = 0;

    for(int j=0;j<Documents.getColumnDimension();j++)
      {
         SValue = (Math.pow(S[j][j],2));
          Value += (D1[Vector1][j] * D1[Vector2][j] * SValue);
      }
      return Value;
}
```

Gambar 4.30 *procedure* getSimm

- Summarize ()

Metode utama pada kelas ini. Berfungsi untuk mencari hasil ringkasan dari Query yang diberikan. Langkah awal dari metode Summarize dapat dilihat pada gambar 4.31

```
public void Summarize(int percentage, String Query, String
FileFolder,String SrcPath,String weighting, boolean write)
throws IOException
       {
                 Stage = 0;
                 percent = percentage;
                 .....
                 .....
                 IDExtracted.removeAllElements();
                 if(docVec.isEmpty())
                 { LoadAllDocument(FileFolder); }
                 else
                 { docVec.removeElementAt(posQuery);}
                 ....
                 ....
                 docVec.addElement(Query);
                 posQuery = docVec.indexOf(Query);

         vsm = new Preprocess();
         vsm.loadStopwords(SrcPath+"/data/stopword.txt");
         vsm.loadBlockwords(SrcPath+"/data/blockword.txt");
         vsm.setDocumentVector(docVec);
         vsm.setAutoDocIDs();
         vsm.setDocWeightingMethod(weighting);
         ....
         vsm.createDictPost();
         vsm.createXmatrix();
         lsa1 = new LSA(vsm.getXMatrix(), vsm.getTermVec(),
vsm.getDocIDVec());
       lsa1.performSVD();
```

Gambar 4.31 langkah awal *procedure* Summarize

Setelah persiapan data telah selesai, proses seanjutnya adalah membuat ringkasan dokumen. Untuk mengeliminasi hasil yang kurang tepat, ekstraksi kalimat dibatasi dengan skor lebih dari 0,1 dimana skor maksimal adalah 1. Pengambilan nilai 0,1 dikarenakan jika nilainya kurang dari 0,1 lebih mengarah ke nilai 0, dimana nilai 0 adalah nilai minimal dan tidak mewakili data.

```
Lambda = 1;
double LambdaMinus = (double)1 / (double)k;

  Hashtable prev = new Hashtable();
  prev.clear();

  while (count<k)
  {
    HighScore = 0;
    posScore = -1;
    for (int i=0;i<Documents.getRowDimension()-1;i++)
    {
        if(!prev.containsKey(i))
         {
         Sim1 = getSimm(i,posQuery);
         Sim2 = 0;
         if(!idSumm.isEmpty())
         {
           double HighScoreStc = 0;
           double Value;
         for(int a=0;a<idSumm.size();a++){
         Value =
getSimm(i,Integer.parseInt(idSumm.elementAt(a).toString()));}
Sim2 = HighScoreStc / (double)idSumm.size(); }
Score = Lambda*Sim1 + (1-Lambda)*Sim2;
if (HighScore<Score) {
   HighScore = Score;
   posScore = i; }
}
}
if (HighScore>0.1)
{
idSumm.addElement(posScore);
idDocSumm.addElement(idVec.elementAt(posScore));
Summ.addElement(docVec.elementAt(posScore));
prev.put(posScore,1);
}
count++;
Lambda = Lambda - LambdaMinus;
}
```

Gambar 4.32 pembentukan ringkasan *procedure* Summarize

- getSumm()
  Metode mengambil hasil ringkasan dan memilah-milah berdasarkan dokumen-dokumennya.

```
public String getSumm()
 {
   Vector summSorted = new Vector();
   Vector summIDSorted = new Vector();
   String buffer = "";
   Hashtable prev = new Hashtable();
   prev.clear();
   for (int i=0;i<Summ.size();i++)
   {
    int lower = 999;
    for (int j=0;j<Summ.size();j++)
    {

if((Integer.parseInt(idSumm.elementAt(j).toString())<lower) &&
(!prev.containsKey(Integer.parseInt(idSumm.elementAt(j).toStri
ng()))))
         lower =
         Integer.parseInt(idSumm.elementAt(j).toString());
   }

....
....
```

Gambar 4.33 *procedure* getSumm

- getSummNotShorted()
  Metode mengambil hasil ringkasan tanpa memilah-milah berdasarkan dokumen-dokumennya.

```
public String getSummNotShorted()
{
    String bufferNotShorted = "";
    for (int i=0;i<Summ.size();i++)
    {
        bufferNotShorted += Summ.elementAt(i);
    }
    return bufferNotShorted;
}
```

Gambar 4.33 *procedure* getSummNotShorted

- Analyze ()
  Metode ini berfungsi untuk menganalisa hasil ringkasan dengan membandingkannya dengan hasil ideal.

```
int IrrelevantDoc = 0; //hasil sistem yang tak ada di hasil
manual
int RetrievedDoc = idSumm.size(); //hasil sistem
int ManRetrieveDoc = ManSummID.size(); //hasil manual
int RelevantDoc = 0; //hasil sistem yang ada di hasil manual
     for (int i=0;i<idSumm.size();i++)
     {
         if (ManSummID.contains(idSumm.elementAt(i)))
                         RelevantDoc++;
     }

IrrelevantDoc = RetrievedDoc - RelevantDoc;

 //Penghitungan

double Precision = Double.valueOf(RelevantDoc).doubleValue()/
Double.valueOf(RetrievedDoc).doubleValue();

double Recall = Double.valueOf(RelevantDoc).doubleValue()/
Double.valueOf(ManRetrieveDoc).doubleValue();

 double FMeasure = 2 * Double.valueOf(Precision).doubleValue()
* Double.valueOf(Recall).doubleValue() /
(Double.valueOf(Precision).doubleValue()+Double.valueOf(Recall
).doubleValue());

double Fallout = Double.valueOf(IrrelevantDoc).doubleValue() /
(Double.valueOf(docVec.size()).doubleValue() -
Double.valueOf(ManRetrieveDoc).doubleValue());
```

Gambar 4.35 *procedure* Analyze

4.2.7 MyThread

Kelas ini adalah extend dari kelas *thread* dan digunakan untuk menjalankan proses menjadi thread. Kelas ini memiliki atribut-atribut :

- MySumm : digunakan untuk menjalankan proses peringkasan.
- sumSize : digunakan untuk menyimpan besarnya hasil ringkasan.
- txtQuery : digunakan untuk menyimpan kalimat query yang diberikan *user*.
- DocFolderPath : berfungsi menyimpan alamat dokumen berita.
- folderPath : berfungsi menyimpan alamat aplikasi.
- Weighting : pembobotan yang dipilih.
- Write : pilihan untuk menulis matrix hasil atau tidak.

```
class MyThread extends Thread
{
        Summarizer MySumm = new Summarizer();
        int sumSize;
        String txtQuery;
        String DocFolderPath;
        String folderPath;
        String weighting;
        boolean write;

        ....
        ....
}
```

Gambar 4.36 *class* MyThread

Metode-metode yang digunakan

- MyThread ()
  Metode ini berfungsi sebagai kelas konstruktor kelas MyThread.

```
public MyThread (Summarizer summ,int sumSize1,String
txtQuery1,String DocFolderPath1,String folderPath1,String
weighting1, boolean write1)
        {
                MySumm = summ;
                sumSize = sumSize1;
                txtQuery = txtQuery1;
                DocFolderPath = DocFolderPath1;
                folderPath = folderPath1;
                weighting = weighting1;
                write = write1;
        }
```

Gambar 4.37 *constuctor* Mythread

- Run ()
  Metode ini berfungsi untuk overide metode Run() pada kelas *thread.*

```
public void run()
{
try
{MySumm.Summarize(sumSize,txtQuery,DocFolderPath,folderPath,we
ighting,write);
                }catch (Exception e)
                {
                }

}
```

Gambar 4.38 *procedure* Run

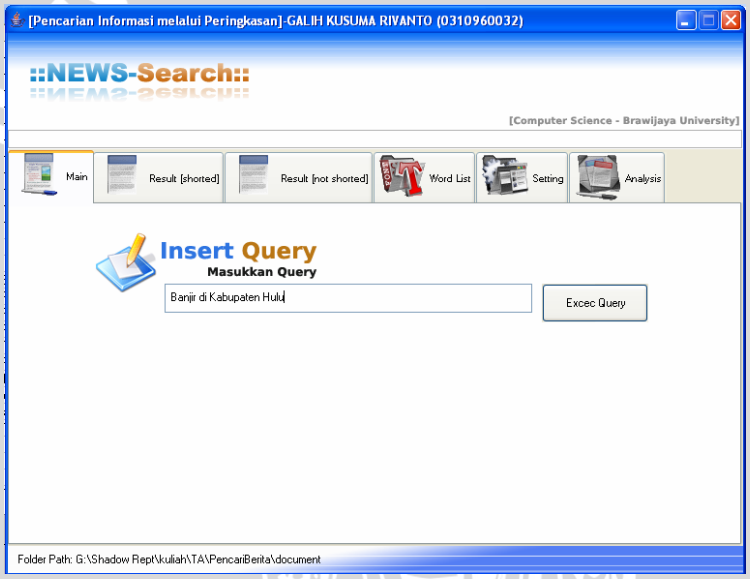
4.2.8 Kelas GUI

Kelas ini berisi atribut- atribut yang berupa komponen untuk membangun antarmuka Pencari Berita . Komponen yang dipergunakan antara lain

6. *input text* untuk memasukkan *query*.
7. *text area* untuk menampilkan hasil ringkasan
8. tombol untuk menjalankan proses.
9. *spin edit* untuk mengatur jumlah dokumen yang diambil untuk ringkasan
10. *spin edit* untuk mengatur besarnya ringkasan (jumlah kalimat)
11. *list* untuk memunculkan daftar *term, stop word, block word* dan kalimat-kalimat yang diproses.

4.3 Implementasi Antarmuka

Berdasarkan rancangan antarmuka pada subbab 3.2.2 dihasilkan antarmuka seperti terlihat pada Gambar 4.39.



Gambar 4.39 **Antarmuka Peringkas Dokumen**

4.4 Implementasi Uji Coba

Pada subbab ini akan dilakukan pembahasan mengenai pengujian yang telah dilakukan pada sistem dan hasil evaluasi dari hasil sistem.

4.4.1 Uji Fungsionalitas Sistem

Untuk mengetahui sistem dapat dijalankan dengan baik, dilakukan uji fungsionalitas modul-modul yang dibuat. Uji fungsionalitas dilakukan hanya pada modul-modul tertentu, terutama modul yang mempunyai fungsi yang vital. Modul-modul yang diuji antara lain.

4.4.1.1 Fungsi Pemecahan Kalimat

Untuk mengetahui apakah fungsi pemecahan kalimat dapat berjalan dengan baik, dilakukan uji coba dengan menggunakan dokumen pendek dan kode seperti pada Gambar 4.40

```
public static void main(String[] args) throws IOException
{
        Summarizer summ = new Summarizer();
        Vector kalimat = new Vector();

        String file_folder = "test doc";

        summ.LoadAllDocument(file_folder);


        kalimat = summ.showDoc();

        for(int i=0;i<kalimat.size();i++)
        {
           System.out.println("Kalimat  <"+(i+1)+"> :
"+kalimat.elementAt(i)+"");
        }
```

Gambar 4.40 Uji coba Pemecahan Kalimat

Dokumen yang digunakan untuk percobaan diatas adalah file teks "1.txt" yang isinya adalah

Suasana pergantian tahun diwarnai peristiwa tragis di jalur selatan arus balik. Sepuluh penumpang minibus Daihatsu HiJet tewas seketika, tujuh diantaranya tewas terbakar, setelah minibus itu ditabrak oleh bus Sinar Jaya, persis di atas Jembatan Sungai Ciberung, Kecamatan Ajibarang, sekitar 18 kilometer arah barat Kota Purwokerto, Minggu (31/12) sekitar pukul 15.10.

Tidak hanya menabrak minibus Daihatsu HiJet, bus Sinar Jaya dengan nomor polisi B 7375 PV yang melaju kencang dari arah utara itu juga menghantam sebuah mobil Toyota Kijang dan sepeda motor Honda Grand. Namun,

penumpang mobil Kijang maupun pengendara motor hanya mengalami cedera ringan.

Usai tabrakan beruntun itu, pengemudi bus tersebut langsung melarikan diri. "Yang bersangkutan masih buron. Tapi, polisi sudah mengantungi jati diri sopir bus itu. Jadi lebih baik kalau ia menyerahkan diri saja daripada menjadi buronan," ujar Kepala Kepolisian Resor (Polres) Banyumas Superintendent Imam Basuki.

Tiga hari sebelumnya, sepuluh penumpang minibus Daihatsu Zebra yang merupakan rombongan pemain band tewas akibat kendaraan yang mereka tumpangi tertabrak bus PO Dwimarta antara Kota Prembun-Kebumen. Peristiwa tabrakan yang menimpa rombongan pemusik lokal itu juga terjadi di tengah jembatan. (Kompas, 29/12)

Sejumlah saksi mata menuturkan, bus Sinar Jaya yang melaju dengan kecepatan 80-90 kilometer per jam sudah tampak oleng saat akan menyalip sedan Mitsubishi Lancer di depannya. Akibatnya, sopir bus tak mampu lagi mengendalikan kendaraannya saat menyalip dan menabrak minibus Daihatsu HiJet serta dua kendaraan lainnya yang melaju dari arah selatan.

Akibat benturan keras, badan minibus Daihatsu hancur total, bahkan terperangkap di kolong bus Sinar Jaya dan terseret hingga beberapa meter. Benturan dan gesekan kedua kendaraan tersebut menimbulkan percikan api yang menyebabkan terjadinya kobaran api yang menghanguskan bus Sinar Jaya, minibus Daihatsu HiJet, mobil Kijang, serta sepeda motor bebek Honda Grand.

Tujuh dari 10 penumpang minibus Daihatsu terjebak dalam kendaraan yang hancur, dan hangus terbakar. Sedangkan tiga orang lainnya, termasuk sopir minibus itu, Sukaryadi, terlempar ke luar kendaraan, juga tewas.

Evakuasi korban yang terjebak dan terbakar dalam kendaraan dipimpin langsung oleh Kepala Kepolisian Wilayah (Polwil) Banyumas Senior Superintendent Carel Risacotta dan Kepala Kepolisian Resor Superintendent Imam Basuki memerlukan waktu lebih dari tiga jam. Untuk mengangkat badan minibus yang berada di kolong bus, petugas harus menariknya dengan mobil derek. Ketujuh korban tewas terbakar sulit dikenali lagi.

Data dari kepolisian setempat, korban meninggal akibat kecelakaan itu adalah Sukaryadi (38) dan istrinya Ny Munjiah, serta dua anaknya Aziz (10) dan Dewi (1,5); kemudian Subandi (25) dan istrinya, Ny Djuju (20) serta anaknya, Resa (3); serta Djumadi (48) dan anaknya, Yunus (7) dan Dwi Novi Kurniawati (18), mahasiswi Fakultas Ekonomi Universitas Soedirman, Purwokerto. Mereka bermaksud berlibur ke Jakarta.

Dari isi teks diatas dihasilkan kalimat-kalimat yang dapat dilihat pada Gambar 4.41

Gambar 4.41 Hasil pemecahan kalimat

4.4.1.2 Fungsi *tokenization, filtration* dan *stemming*

Fungsi *stemming* mempunyai peranan yang penting untuk memperoleh analisa semantik dari dokumen. Seperti yang telah dijelaskan pada Subbab 2.2.1.4 bahwa ada kemungkinan ada kata-kata yang memiliki makna sama walaupun dalam bentuk yang berbeda. Untuk menguji fungsi *stemming* dijalankan kode seperti dalam Gambar 4.43

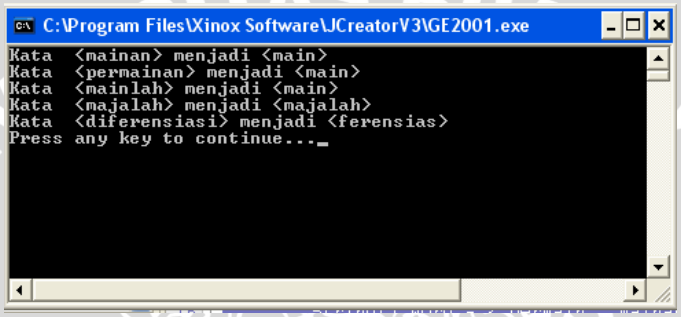
```
public static void main(String[] args)
{
  IndoStemmer test = new IndoStemmer();
  Vector block = new Vector();

  block.addElement("majalah");

  String[] word =
{"bermain","mainan","permainan","mainlah","majalah","diferensi
asi"};
   for(int i=1;i<word.length;i++)
   {
         System.out.println("Kata  <"+word[i]+"> menjadi
<"+test.removeAffix(word[i],block)+">");
   }
}
```

Gambar 4.42 Uji coba *Stemming*

Dari kode pada Gambar 4.42 akan didapatkan hasil seperti pada Gambar 4.43.



Gambar 4.43 Hasil uji coba *Stemming*

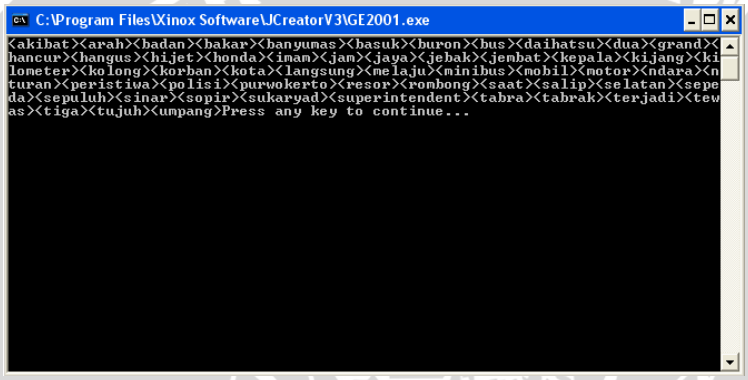
Dari hasil diatas dapat disimpulkan bahwa proses *stemming* dapat berjalan dengan baik dari lima kata yang diberikan empat kata yang berhasil. Kata diferensiasi tidak berhasil melalui proses *stemming* karena tidak dimasukkan kedalam daftar *block word,* diferensiasi adalah kata dasar namun memiliki imbuhan i dan di.

Untuk menguji fungsi *tokenization,filtration* dan *stemming* dilakukan secara bersamaan. Kode yang digunakan untuk menguji ketiganya dapat dilihat pada Gambar 4.44

```
public static void main(String[] args) throws IOException
{
        Summarizer summ = new Summarizer();
        Preprocess vsm = new Preprocess();
        Vector kata = new Vector();
        String file_folder = "test doc";
        summ.LoadAllDocument(file_folder);
        vsm.loadStopwords("data/stopword.txt");
        vsm.loadBlockwords("data/blockword.txt");
        vsm.setDocumentVector(summ.showDoc());
        vsm.setAutoDocIDs();
        vsm.setDocWeightingMethod("idf");
        vsm.setActivatePorterStemmer(true);
        vsm.setEliminateUniqueTerms(true);
         vsm.createDictPost();
        kata = vsm.getTermVec();
        for(int i=0;i<kata.size();i++)
        {
          System.out.print("<"+kata.elementAt(i)+">");
        }
}
```

Gambar 4.44 kode *test tokenization, filtration* dan *stemming*

Dari kode yang dijalankan akan muncul hasil seperti pada Gambar 4.45

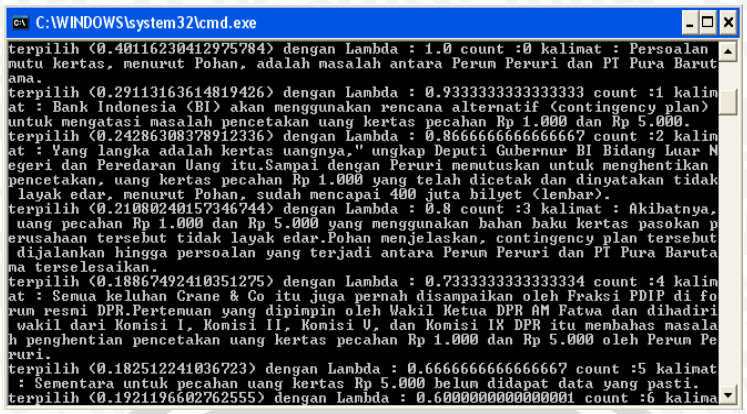


Gambar 4.45 hasil *test tokenization, filtration* dan *stemming*

Dari Gambar 4.45 dapat dilihat bahwa proses *Tokenization, filtration* dapat berjalan dengan baik, sedangkan untuk *stemming* masih ada kesalahan, seperti pada *test stemming* tersendiri bahwa *block word* yang digunakan masih belum lengkap.

4.4.1.3 Fungsi pembentuk ringkasan

Untuk mengetahui bagaimana penghitungan skor dan pemilihan ringkasan diadakan *test* untuk menguji penghitungan kalimat. Hasil dari pengujian dapat dilihat pada Gambar 4.46



Gambar 4.46 hasil pembentukan ringkasan

4.4.2 Hasil Evaluasi

Dalam Subbab 4.4.1 memuat hasil implementasi sistem yang telah dibangun mulai dari proses memasukkan *query* sampai pembentukan ringkasan dan analisa oleh sistem.

Didalam Gambar 4.39 telah ditunjukkan antar muka awal sistem yang dibangun. Kemudian dimasukkan *query* pada *text field* yang telah disediakan dan kemudian tombol proses dijalankan. Ketika proses dijalankan, sistem mengambil dan meload seluruh dokumen yang berada pada *path* yang ditinjukkan pada Folder Path dalam sistem. Fungsi yang digunakan untuk mengambil seluruh dokumen adalah *procedure* LoadAllDocument yang ada pada kelas Summarizer.

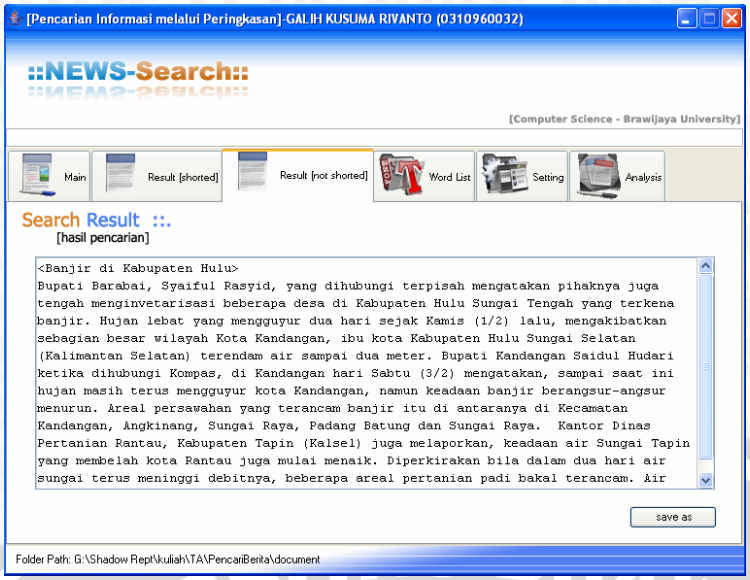
Setelah seluruh dokumen telah diambil dan dipecah menjadi kalimat, selanjutnya ditambahkan *query* pada *Vector* kalimat pengambilan *stop word* dan *block word* kemudian dilakukan yang dilanjutkan dengan penentuan pembobotan. Setelah itu seluruh data di*passing* ke CreateDictPost.

Di dalam CreateDictPost seluruh proses *preprocessing* dilakukan sampai dihasilkan matriks *vector space model* yang disimpan dalam matriks X. Proses selanjutnya adalah melakukan proses *Singular Value Decomposition* dengan menggunakan *procedure* performSVD yang ada pada kelas LSA.

Setelah data yang diperlukan didapatkan, proses yang paling akhir adalah pembentukan ringkasan yang dapat dilihat dalam Gambar 4.48. Gambar 4.47 adalah tampilan sistem ketika menjalankan proses.

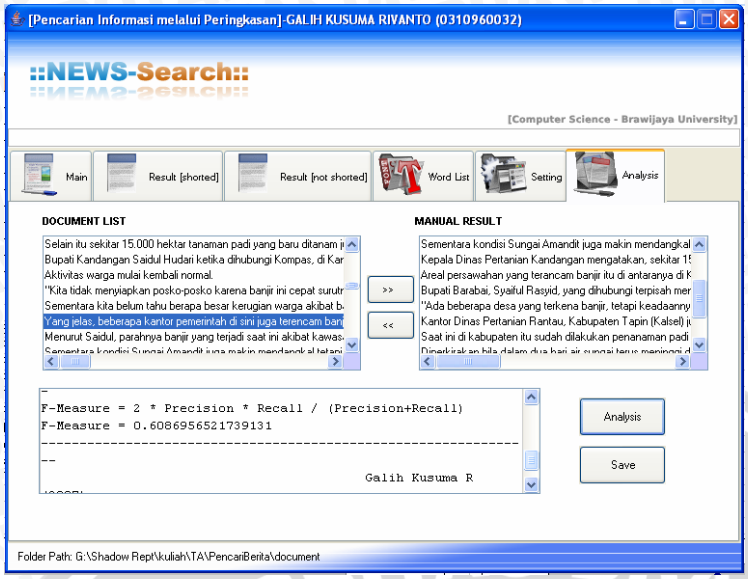
Gambar 4.47 Hasil tampilan sistem saat proses dijalankan

Hasil peringkasan yang telah didapatkan selanjutnya dimasukkan ke dalam *text area.*Ada dua macam hasil peringkasan yang dihasilkan, yang pertama hasil ringkasan yang ditampilkan tanpa memilah-milah berdasarkan dokumen seperti yang terlihat pada Gambar 4.48 .



Gambar 4.48 Hasil ekstraksi tanpa mengalami *sorting*

Untuk melakukan analisa, *Abstractor* memilih kalimat-kaliamat yang sesuai dengan hasil ringkasan manual pada *list* dokumen (kalimat-kalimat dalam dokumen) ke dalam *manual result,* kemudian dilakukan analisa. Sistem memeriksa berapa jumlah kalimat sistem dan manual yang sama dan yang tidak sama yang selanjutnya dihitung *precision, recall,* dan *F-Measure*-nya. Hasil analisa dapat dilihat pada Gambar 4.49



Gambar 4.49 Hasil analisa

Dari sepuluh *Query* yang diberikan ke sistem maupun ke *Abstractor,* tabel 4.1, sampai tabel 4.10 adalah tabel hasil penghitungan *precision, recall* dan *F-Measure* serta *fall-out* berdasarkan jumlah kata kunci dalam *query* .

Untuk hasil analisa dengan menggunakan kata kunci dalam *query* berjumlah 1 dapat dilihat dalam tabel 4.1, tabel 4.2 dan tabel 4.3

Tabel 4.1 Hasil Analisa 1 kata kunci oleh *Abstractor*1

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	4	1	0.7143	0.8333	0
2	Q2	3	0	11	1	0.2143	0.3529	0
3	Q3	13	2	2	0.8667	0.86677	0.8667	0.0054
4	Q4	10	1	5	0.9091	0.6667	0.7692	0.0027
5	Q5	7	0	7	1	0.5	0.6667	0
6	Q6	12	3	5	0.8	0.7059	0.75	0.0080
7	Q7	12	0	4	1	0.75	0.8571	0
8	Q8	8	1	6	0.8889	0.5714	0.6957	7
9	Q9	11	0	5	1	0.6875	0.8148	0
10	Q10	13	2	3	0.8667	0.8125	0.8387	0.0054
rata-rata					0.9331	0.6489	0.7445	0.0024

Tabel 4.2 Hasil Analisa 1 kata kunci oleh *Abstractor*2

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	3	0	10	1	0.2308	0.375	0
3	Q3	13	2	4	0.8667	0.7647	0.8125	0.0054
4	Q4	11	0	3	1	0.7857	0.88	0
5	Q5	6	1	8	0.8571	0.4286	0.5714	0.0027
6	Q6	13	2	4	0.8667	0.7647	0.8125	0.0054
7	Q7	10	2	6	0.8333	0.625	0.7143	0.0054
8	Q8	9	0	5	1	0.6429	0.7826	0
9	Q9	9	2	7	0.8181	0.5625	0.6667	0.0054
10	Q10	13	2	5	0.8667	0.7222	0.7879	0.0054
rata-rata					0.9109	0.61934	0.7203	0.0010

Tabel 4.3 Hasil Analisa 1 kata kunci oleh *Abstractor*3

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	3	0	10	1	0.2308	0.375	0
3	Q3	13	2	4	0.8667	0.7647	0.8125	0.0054
4	Q4	11	0	3	1	0.7857	0.88	0
5	Q5	7	0	7	1	0.5	0.6667	0
6	Q6	13	2	4	0.8667	0.7647	0.8125	0.0054
7	Q7	10	2	6	0.8333	0.625	0.7143	0.0054
8	Q8	9	0	5	1	0.6429	0.7826	0
9	Q9	11	0	5	1	0.6875	0.8148	0
10	Q10	13	2	5	0.8667	0.7222	0.7879	0.0054
rata-rata					0.9433	0.6390	0.7446	0.0023

Dari ketiga hasil analisa dari table 4.1, table 4.2 dan table 4.3 didapatkan rata-rata nilai *precicion, recall, f-measure dan fall-out* untuk *query* 1 kata kunci seperti yang terlihat pada tabel 4.4

Tabel 4.4 Rata-rata Hasil Analisa 1 kata kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.9331	0.6489	0.7445	0.0024
2	0.9109	0.61934	0.7203	0.001
3	0.9433	0.639	0.7446	0.0023
rata-rata	0.9291	0.6357	0.7365	0.0019

Keterangan *Query* :

Q1 Banjir
Q2 demonstrasi
Q3 ajinomoto
Q4 saham
Q5 Tabrakan
Q6 PERURI
Q7 granat
Q8 bentrokan
Q9 MUI
Q10 sensus

Tabel 4.5 Hasil Analisa 2 kata kunci Oleh *Abstractor*1

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	11	2	3	0.8462	0.7857	0.8148	0.0053
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	10	5	6	0.6667	0.625	0.6452	0.0134
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7608	0.6652	0.7068	0.00917

Tabel 4.6 Hasil Analisa 2 kata kunci Oleh *Abstractor*2

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	9	4	5	0.6923	0.6429	0.6667	0.0107
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	8	7	8	0.5333	0.5	0.5161	0.0188
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7321	0.6384	0.6791	0.0102

Tabel 4.7 Hasil Analisa 2 kata kunci Oleh *Abstractor*3

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	11	2	3	0.8462	0.7857	0.8148	0.0053
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	10	5	6	0.6667	0.625	0.6452	0.0134
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7608	0.6652	0.7068	0.00917

Dari ketiga hasil analisa dari table 4.5, table 4.6 dan table 4.7 didapatkan rata-rata nilai *precicion, recall, f-measure dan fall-out* untuk *query* 2 kata kunci seperti yang terlihat pada tabel 4.8

Tabel 4.8 Rata-rata Hasil Analisa 2 kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.7608	0.6667	0.7068	0.00917
2	0.7321	0.6384	0.6791	0.0102
3	0.7608	0.6667	0.7068	0.00917
rata-rata	0.7512	0.6652	0.6976	0.0095

Keterangan *Query* :

Q1 Banjir di Hulu
Q2 massa demonstrasi
Q3 direksi ajinomoto
Q4 Perdagangan saham
Q5 Tabrakan minibus
Q6 uang PERURI
Q7 granat KA
Q8 bentrokan mahasiswa
Q9 tanggapan MUI

Q10 Sensus penduduk

Tabel 4.5 Hasil Analisa 2 kata kunci Oleh *Abstractor*1

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	11	2	3	0.8462	0.7857	0.8148	0.0053
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	10	5	6	0.6667	0.625	0.6452	0.0134
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7608	0.6652	0.7068	0.00917

Tabel 4.6 Hasil Analisa 2 kata kunci Oleh *Abstractor*2

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	9	4	5	0.6923	0.6429	0.6667	0.0107
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	8	7	8	0.5333	0.5	0.5161	0.0188
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7321	0.6384	0.6791	0.0102

Tabel 4.7 Hasil Analisa 2 kata kunci Oleh *Abstractor*3

No	Query	A	B	C	Precision	Recall	Fmeasure	Fall-out
1	Q1	10	0	5	1	0.6667	0.8	0
2	Q2	11	2	3	0.8462	0.7857	0.8148	0.0053
3	Q3	13	1	4	0.9286	0.7647	0.8387	0.0027
4	Q4	8	4	6	0.6667	0.5714	0.6154	0.0107
5	Q5	9	1	4	0.9	0.6923	0.7826	0.0027
6	Q6	9	6	7	0.6	0.5625	0.5806	0.0161
7	Q7	10	5	6	0.6667	0.625	0.6452	0.0134
8	Q8	8	7	6	0.5333	0.5714	0.5517	0.0187
9	Q9	13	2	3	0.8667	0.8125	0.8387	0.0054
10	Q10	9	6	6	0.6	0.6	0.6	0.0161
rata-rata					0.7608	0.6652	0.7068	0.00917

Dari ketiga hasil analisa dari table 4.5, table 4.6 dan table 4.7 didapatkan rata-rata nilai *precicion, recall, f-measure dan fall-out* untuk *query* 2 kata kunci seperti yang terlihat pada tabel 4.8

Tabel 4.8 Rata-rata Hasil Analisa 2 kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.7608	0.6667	0.7068	0.00917
2	0.7321	0.6384	0.6791	0.0102
3	0.7608	0.6667	0.7068	0.00917
rata-rata	0.7512	0.6652	0.6976	0.0095

Keterangan *Query* :

Q1 Banjir di Hulu
Q2 massa demonstrasi
Q3 direksi ajinomoto
Q4 Perdagangan saham
Q5 Tabrakan minibus
Q6 uang PERURI
Q7 granat KA

Q8 bentrokan mahasiswa
Q9 tanggapan MUI

Rata-rata hasil analisa untuk 3, dan 4 kata kunci dapat dilihat pada tabel 4.9 dan tabel 4.10

Tabel 4.9 Rata-rata Hasil Analisa 3 kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.7567	0.578	0.6554	0.0092
2	0.7473	0.6265	0.6816	0.0091
3	0.7535	0.5817	0.6433	0.0089
rata-rata	0.7523	0.5954	0.6601	0.009

Keterangan *Query* :

Q1 Banjir di Hulu
Q2 massa demonstrasi
Q3 direksi ajinomoto
Q4 Perdagangan saham
Q5 Tabrakan minibus
Q6 uang PERURI
Q7 granat KA
Q8 bentrokan mahasiswa
Q9 tanggapan MUI
Q10 Sensus penduduk

Tabel 4.9 Rata-rata Hasil Analisa 3 kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.74081	0.6465	0.6874	0.0099
2	0.7321	0.6384	0.6791	0.0102
3	0.7408	0.6465	0.6874	0.0099
rata-rata	0.7379	0.6652	0.6846	0.0100

Keterangan *Query* :

Q1 Banjir di Kabupaten Hulu
Q2 Pengerahan massa demonstrasi
Q3 Penangkapan direksi ajinomoto

Q4 Perdagangan saham dan bursa
Q5 Korban tabrakan minibus
Q6 uang kertas PERURI
Q7 granat di rel KA
Q8 bentrokan mahasiswa pendemo
Q9 tanggapan MUI terhadap ajinomoto
Q10 Sensus penduduk indonesia

Tabel 4.10 Rata-rata Hasil Analisa 4 kunci

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.7567	0.5760	0.6554	0.0102
2	0.7321	0.6384	0.6821	0.0102
3	0.7535	0.5817	0.6433	0.0089
rata-rata	0.7474	0.5994	0.6603	0.0096

Keterangan *Query* :

Q1 Banjir di Kabupaten Hulu Kalimantan
Q2 Masalah pengerahan massa demonstrasi
Q3 Penangkapan direksi ajinomoto oleh polisi
Q4 Perdagangan saham dan bursa di awal tahun
Q5 tabrakan minibus menelan korban
Q6 Permasalahan uang kertas PERURI
Q7 Ditemukan granat di rel KA
Q8 Bentrokan BEM mahasiswa pendemo
Q9 Tanggapan MUI terhadap penyedap ajinomoto
Q10 Sensus kepadatan penduduk indonesia

Tabel 4.11 Rata-rata Hasil Analisa

Abstraktor	Precision	Recall	Fmeasure	Fall-out
1	0.9291	0.6357	0.7365	0.0019
2	0.7512	0.6652	0.6976	0.0095
3	0.7379	0.6652	0.6846	0.01
4	0.7474	0.5994	0.6603	0.0096
rata-rata	0.7914	0.6413	0.6948	0.0078

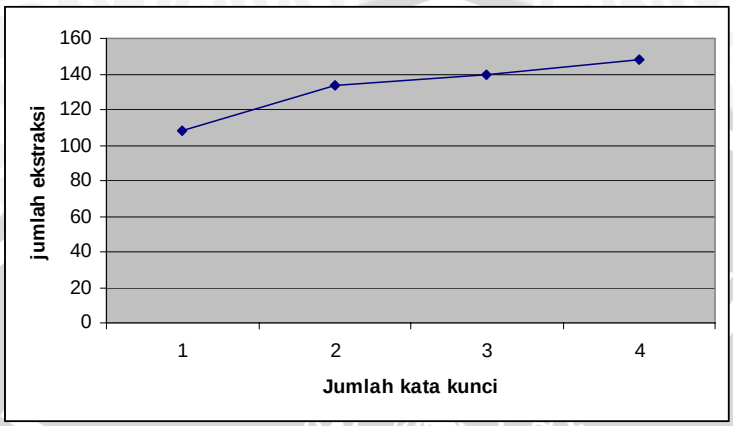
Keterangan Notasi :

A : Hasil sistem yang sama dengan hasil ideal

B : Hasil sistem yang tidak ada namun hasil ideal ada

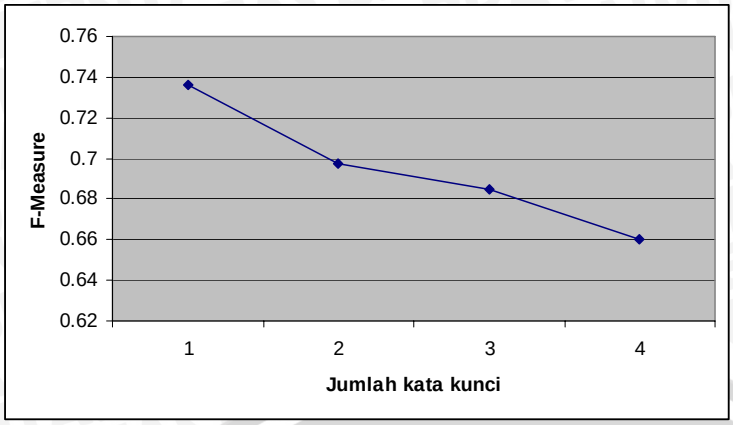
C : Hasil sistem ada namun hasil ideal tidak ada

Gambar 4.50 menunjukkan pengaruh jumlah kata kunci dalam menentukan jumlah ekstraksi sistem.



Gambar 4.51 Grafik pengaruh jumlah kata kunci terhadap jumlah ekstraksi sistem

Dari beberapa percobaan penggunaan jumlah kata kunci pada query didapatkan grafik pengaruh kata kunci terhadap presisi hasil sistem, seperti terlihat pada Gambar 4.51



Gambar 4.51 Grafik pengaruh jumlah kata kunci terhadap presisi sistem

4.4.3 Analisa Hasil

Pengujian fungsionalitas aplikasi dapat dilihat dalam Subbab 4.4.1, hasilnya bahwa aplikasi telah sesuai dengan metode yang digunakan dan dapat menjalankan algoritma yang digunakan. Sedangkan untuk hasil evaluasi efektifitas dari hasil yang dihasilkan oleh sistem, diperoleh nilai rata-rata *recall* sebesar 64 %, *precision* 79%, dan *f-measure* sebesar 69% serta *fall-out* 0,7%. Dari hasil uji efektifiitas dapat disimpulkan bahwa rata-rata hasil sistem dapat merepresentasikan 69% hasil buatan manusia.

Dari beberapa percobaan jumlah kata kunci yang diberikan, terdapat perbedaan jumlah ekstraksi sistem. Semakin banyak kata kunci sistem yang diberikan, maka semakin besar jumlah ekstraksi sistem seperti yang terlihat pada gambar 4.50. Hal ini dipengaruhi oleh setiap kata kunci memiliki relasi dengan kata-kata tertentu. Misalnya *query* "sensus penduduk indonesia", kata "sensus" memiliki arti yang spesifik, namun kata "indonesia" dapat diambil dari berberapa dokumen yang tidak berhubungan sama sekali dengan kata "sensus".

Dari pengaruh jumlah kata kunci juga didapatkan hasil presisi sistem yang berbeda-beda seperti pada gambar 4.51. Alasannya sama dengan dengan pengaruh jumlah kata kunci terhadap jumlah ekstraksi. Karena dengan sedikitnya kata kunci pada *query* yang

diberikan maka hasil ekstraksi sistem menjadi semkin fokus demikian sebaliknya.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang didapat selama pengerjaan Tugas Akhir ini adalah :

1. Penerapan metode peringkasan dokumen sebagai ekstraksi informasi dapat berjalan sesuai dengan perancangan dan dapat dikatakan mewakili hasil yang dibuat manusia.
2. Ekstraksi informasi dengan menerapkan metode peringkasan dokumen ini menghasilkan rata-rata *recall* sebesar 64%, rata-rata *precision* sebesar 79%, dan rata-rata *f-measure* sebesar 69% serta rata-rata *fall-out* sebesar 0,7%.
3. Jumlah kata kunci dalam *query* berpengaruh terhadap hasil ekstraksi. Penambahan kata kunci membuat jumlah hasil ekstraksi yang lebih besar namun memiliki presisi yang lebih kecil. Sedangkan semakin sedikit kata kunci yang diberikan membuat hasil ekstraksi lebih fokus namun dengan jumlah hasil yang sedikit.

5.2 Saran

Beberapa saran pengembangan lebih lanjut yang dapat diberikan oleh penulis adalah :

1. Karena hasil stemming sangat berpengaruh pada penghitungan *vector space model* sebaiknya dilakukan perbaikan proses *stemming* pada kata yang ada dalam dokumen, supaya untuk fitur frekuensi dapat lebih akurat.
2. Pembuatan hasil ideal, diserahkan kepada orang yang benar-benar ahli.

Daftar Pustaka

Baldi, P, P.Frasconi dan P.Smyth. 2003. ***Modelling the internet and the web***. Diakses pada tanggal 29 september 2006.

Bergo, Alexander. 2001. ***Text Categorization and Prototypes***. http://www.illc.uva.nl/Publications/ResearchReports/MoL-2001-08.text.pdf. Diakses pada tanggal 16 januari 2007.

Deerwester, S., Dumais, Susan T., Furnas, George W., Launder, Thomas K., dan Harshman, R. ***Indexing by Latent Semantics Analysis***. 1990.

Erkan, G. and D.R Radev. 2004. Lexrank : ***Graph-based centrality as salience in text summarization***. *JAIR*.

Firmin, T. and M.J. Chrzanowski. 1999. ***An Evaluation of Automatic Text Summarization Systems***. The MIT Press : Cambridge

Garcia, Dr. E. 2005. ***Document Indexing Tutorial for Information Retrieval Students and Search Engine Marketers.*** http://www.miislita.com/information-retrieval-tutorial/indexing.html. Diakses pada tanggal 29 september 2006.

Guo, Gonde, Hui Wang, David Bell, Yaxin Bi dan Kieran Greer. 2004. ***An knn model based approach and its application in text categorization***. Northern Ireland, UK. http://citeseer.ist.psu.edu/guo04knn.html. Diakses pada tanggal 16 januari 2007.

Hatcey, B., Murray, G., dan Reitter, D. 2005. ***Query Based Multi-Document Summarization using very large Semantics Space*** : School of Informatics University of Edinburgh.

Hearst, Marti. 17 Oktober 2003. ***What is text mining?***. http://www.sims.berkeley.edu/~hearst/text-mining.html.

H.P. Edmudson. 1969. ***New Methods in Automatic Extracting. Journal of Assosiation for Computing Machinery***, 16(2):264-285.

http://wikipedia.org/wiki/information_retrieval.html, tanggal akses : 16 Maret 2007.

http://faure.iei.pi.cnr.it/~fabrizio/Publications/ACMCS02.pdf. tanggal akses : 16 januari 2007.

http://en.wikipedia.org/wiki/document__summarization.html , tanggal akses : 16 Maret 2007

http://en.wikipedia.org/wiki/svd.html , tanggal akses : 16 Maret 2007

http://en.wikipedia.org/wiki/Text_mining.html, tanggal akses :16 Maret 2007

http://www.usenix.org/events/sec02/full_papers/liao/liao_html/. Diakses pada tanggal 16 januari 2007

http://en.wikipedia.org/wiki/latent_sematics analisis.html , tanggal akses : 16 Maret 2007

http://ilps.science.uva.nl/Resources/BI/index.html, tanggal akses : 13 Desember 2006.

http://www2.cs.uregina.ca/_dbd/cs831/notes/confusion_matrix/confusion_matrix.html. Diakses pada tanggal 17 november 2006.

Jen-Yuan Yeh, Hao-Ren Ke, Wei-Pang Yang, dan I-Heng Meng. 2004. ***Text summarization using trainable summarizer and Latent Semantics Analysis***.

Jones, K.S. dan Galliers,J.R. 1996. ***Evaluating natural language processing system : An analysis and review***. New York : Springer.

Launder, Thomas K., Foltz, Peter W. dan Laham Darell. 1998. ***Introduction to Latent Semantics Analysis***. Discourses Processes.

Yihua, Liao, 2002. ***Review of K-Nearest Neighbor Text Categorization Method.***

Mani, I. and M.T. Maybury. 1999. ***Advance in Automatic Text Summarization. The MIT***, Press :Cambridge

Sebastiani, F. 2002. ***Machine Learning In Automated Text Categorization***. ACM Computing Surveys, Vol34, No.1, March 2002, pages 1-47.

Tala, Fadillah Z. 2003. ***A Study of Stemming Effects on Information Retrieval in*** **Bahasa Indonesia**

Yi-Hong Gong dan Xin Liu. 2001. Generic ***Text Summarization Using Relevance Measure and Latent Semantic Analysis***

Lampiran 1

Daftar Stopword

1	ada	36	awalnya	71	benarlah
2	adalah	37	bagai	72	berada
3	adanya	38	bagaikan	73	berakhir
4	adapun	39	bagaimana	74	berakhirlah
5	agak	40	bagaimanakah	75	berakhirnya
6	agaknya	41	bagaimanapun	76	berapa
7	agar	42	bagi	77	berapakah
8	akan	43	bagian	78	berapalah
9	akankah	44	bahkan	79	berapapun
10	akhir	45	bahwa	80	berarti
11	akhiri	46	bahwasanya	81	berawal
12	akhirnya	47	baik	82	berbagai
13	aku	48	bakal	83	berdatangan
14	akulah	49	bakalan	84	beri
15	amat	50	balik	85	berikan
16	amatlah	51	banyak	86	berikut
17	anda	52	bapak	87	berikutnya
18	andalah	53	baru	88	berjumlah
19	antar	54	bawah	89	berkali
20	antara	55	beberapa	90	berkata
21	antaranya	56	begini	91	berkehendak
22	apa	57	beginian	92	berkeinginan
23	apaan	58	beginikah	93	berkenaan
24	apabila	59	beginilah	94	berlainan
25	apakah	60	begitu	95	berlalu
26	apalagi	61	begitukah	96	berlangsung
27	apatah	62	begitulah	97	berlebihan
28	artinya	63	begitupun	98	bermaksud
29	asal	64	bekerja	99	bermula
30	asalkan	65	belakang	100	bersama
31	atas	66	belakangan	101	bersiap
32	atau	67	belum	102	bertanya
33	ataukah	68	belumlah	103	berturut
34	ataupun	69	benar	104	bertutur
35	awal	70	benarkah	105	berujar

106	berupa	141	demikian	176	dilalui
107	besar	142	demikianlah	177	dilihat
108	betul	143	dengan	178	dimaksud
109	betulkah	144	depan	179	dimaksudkan
110	biasa	145	di	180	dimaksudkannya
111	biasanya	146	dia	181	dimaksudnya
112	Bila	147	diakhiri	182	diminta
113	bilakah	148	diakhirinya	183	dimintai
114	Bisa	149	dialah	184	dimisalkan
115	bisakah	150	diantara	185	dimulai
116	boleh	151	diantaranya	186	dimulailah
117	bolehkah	152	diberi	187	dimulainya
118	bolehlah	153	diberikan	188	dimungkinkan
119	buat	154	diberikannya	189	dini
120	bukan	155	dibuat	190	dipastikan
121	bukankah	156	dibuatnya	191	diperbuat
122	bukanlah	157	didapat	192	diperbuatnya
123	bukannya	158	didatangkan	193	dipergunakan
124	bulan	159	digunakan	194	diperkirakan
125	bung	160	diibaratkan	195	diperlihatkan
126	cara	161	diibaratkannya	196	diperlukan
127	caranya	162	diingat	197	diperlukannya
128	cukup	163	diingatkan	198	dipersoalkan
129	cukupkah	164	diinginkan	199	dipertanyakan
130	cukuplah	165	dijawab	200	dipunyai
131	cuma	166	dijelaskan	201	diri
132	dahulu	167	dijelaskannya	202	dirinya
133	dalam	168	dikarenakan	203	disampaikan
134	dan	169	dikatakan	204	disebut
135	dapat	170	dikatakannya	205	disebutkan
136	dari	171	dikerjakan	206	disebutkannya
137	daripada	172	diketahui	207	disini
138	datang	173	diketahuinya	208	disinilah
139	dekat	174	dikira	209	ditambahkan
140	demi	175	dilakukan	210	ditandaskan

211	ditanya	246	hendaknya	281	juga
212	ditanyai	247	hingga	282	jumlah
213	ditanyakan	248	ia	283	jumlahnya
214	ditegaskan	249	ialah	284	justru
215	ditujukan	250	ibarat	285	kala
216	ditunjuk	251	ibaratkan	286	kalau
217	ditunjuki	252	ibaratnya	287	kalaulah
218	ditunjukkan	253	ibu	288	kalaupun
219	ditunjukkannya	254	ikut	289	kalian
220	ditunjuknya	255	ingat	290	kami
221	dituturkan	256	ingin	291	kamilah
222	dituturkannya	257	inginkah	292	kamu
223	diucapkan	258	inginkan	293	kamulah
224	diucapkannya	259	ini	294	kan
225	diungkapkan	260	inikah	295	kapan
226	dong	261	inilah	296	kapankah
227	dua	262	itu	297	kapanpun
228	dulu	263	itukah	298	karena
229	empat	264	itulah	299	karenanya
230	enggak	265	jadi	300	kasus
231	enggaknya	266	jadilah	301	kata
232	entah	267	jadinya	302	katakan
233	entahlah	268	jangan	303	katakanlah
234	guna	269	jangankan	304	katanya
235	gunakan	270	janganlah	305	ke
236	hal	271	jauh	306	keadaan
237	hampir	272	jawab	307	kebetulan
238	hanya	273	jawaban	308	kecil
239	hanyalah	274	jawabnya	309	kedua
240	hari	275	jelas	310	keduanya
241	harus	276	jelaskan	311	keinginan
242	haruslah	277	jelaslah	312	kelamaan
243	harusnya	278	jelasnya	313	kelihatan
244	hendak	279	jika	314	kelihatannya
245	hendaklah	280	jikalau	315	kelima

316	keluar	351	luar	386	memperbuat
317	kembali	352	macam	387	mempergunakan
318	kemudian	353	maka	388	memperkirakan
319	kemungkinan	354	makanya	389	memperlihatkan
320	kemungkinannya	355	makin	390	mempersiapkan
321	kenapa	356	malah	391	mempersoalkan
322	kepada	357	malahan	392	mempertanyakan
323	kepadanya	358	mampu	393	mempunyai
324	kesampaian	359	mampukah	394	memulai
325	keseluruhan	360	mana	395	memungkinkan
326	keseluruhannya	361	manakala	396	menaiki
327	keterlaluan	362	manalagi	397	menambahkan
328	ketika	363	masa	398	menandaskan
329	khususnya	364	masalah	399	menanti
330	kini	365	masalahnya	400	menantikan
331	kinilah	366	masih	401	menanya
332	kira	367	masihkah	402	menanyai
333	kiranya	368	masing	403	menanyakan
334	kita	369	mau	404	mendapat
335	kitalah	370	maupun	405	mendapatkan
336	kok	371	melainkan	406	mendatang
337	kurang	372	melakukan	407	mendatangi
338	lagi	373	melalui	408	mendatangkan
339	lagian	374	melihat	409	menegaskan
340	lah	375	melihatnya	410	mengakhiri
341	lain	376	memang	411	mengapa
342	lainnya	377	memastikan	412	mengatakan
343	lalu	378	memberi	413	mengatakannya
344	lama	379	memberikan	414	mengenai
345	lamanya	380	membuat	415	mengerjakan
346	lanjut	381	memerlukan	416	mengetahui
347	lanjutnya	382	memihak	417	menggunakan
348	lebih	383	meminta	418	menghendaki
349	lewat	384	memintakan	419	mengibaratkan
350	lima	385	memisalkan	420	mengibaratkannya

421	mengingat	456	misalnya	491	pertama
422	mengingatkan	457	mula	492	pertanyaan
423	menginginkan	458	mulai	493	pertanyakan
424	mengira	459	mulailah	494	pihak
425	mengucapkan	460	mulanya	495	pihaknya
426	mengucapkannya	461	mungkin	496	pukul
427	mengungkapkan	462	mungkinkah	497	pula
428	menjadi	463	nah	498	pun
429	menjawab	464	naik	499	punya
430	menjelaskan	465	namun	500	rasa
431	menuju	466	nanti	501	rasanya
432	menunjuk	467	nantinya	502	rata
433	menunjuki	468	nyaris	503	rupanya
434	menunjukkan	469	nyatanya	504	saat
435	menunjuknya	470	oleh	505	saatnya
436	menurut	471	olehnya	506	saja
437	menuturkan	472	pada	507	sajalah
438	menyampaikan	473	padahal	508	saling
439	menyangkut	474	padanya	509	sama
440	menyatakan	475	pak	510	sambil
441	menyebutkan	476	paling	511	sampai
442	menyeluruh	477	panjang	512	sampaikan
443	menyiapkan	478	pantas	513	sana
444	merasa	479	para	514	sangat
445	mereka	480	pasti	515	sangatlah
446	merekalah	481	pastilah	516	satu
447	merupakan	482	penting	517	saya
448	meski	483	pentingnya	518	sayalah
449	meskipun	484	per	519	se
450	meyakini	485	percuma	520	sebab
451	meyakinkan	486	perlu	521	sebabnya
452	minta	487	perlukah	522	sebagai
453	mirip	488	perlunya	523	sebagaimana
454	misal	489	pernah	524	sebagainya
455	misalkan	490	persoalan	525	sebagian

526	sebaik	561	sekadar	596	semua
527	sebaiknya	562	sekadarnya	597	semuanya
528	sebaliknya	563	sekali	598	semula
529	sebanyak	564	sekalian	599	sendiri
530	sebegini	565	sekaligus	600	sendirian
531	sebegitu	566	sekalipun	601	sendirinya
532	sebelum	567	sekarang	602	seolah
533	sebelumnya	568	sekecil	603	seolah-olah
534	sebenarnya	569	seketika	604	seorang
535	seberapa	570	sekiranya	605	sepanjang
536	sebesar	571	sekitar	606	sepantasnya
537	sebetulnya	572	sekitarnya	607	sepantasnyalah
538	sebisanya	573	sekurangnya	608	seperlunya
539	sebuah	574	sela	609	seperti
540	sebut	575	selain	610	sepertinya
541	sebutlah	576	selaku	611	sepihak
542	sebutnya	577	selalu	612	sering
543	secara	578	selama	613	seringnya
544	secukupnya	579	selamanya	614	serta
545	sedang	580	selanjutnya	615	serupa
546	sedangkan	581	seluruh	616	sesaat
547	sedemikian	582	seluruhnya	617	sesama
548	sedikit	583	semacam	618	sesampai
549	sedikitnya	584	semakin	619	sesegera
550	seenaknya	585	semampu	620	sesekali
551	segala	586	semampunya	621	seseorang
552	segalanya	587	semasa	622	sesuatu
553	segera	588	semasih	623	sesuatunya
554	seharusnya	589	semata	624	sesudah
555	sehingga	590	semata-mata	625	sesudahnya
556	seingat	591	semaunya	626	setelah
557	sejak	592	sementara	627	setempat
558	sejauh	593	semisal	628	setengah
559	sejenak	594	semisalnya	629	seterusnya
560	sejumlah	595	sempat	630	setiap

631	setiba	667	tegasnya	703	tiap
632	setibanya	668	telah	704	tiba
633	setidaknya	669	tempat	705	tidak
634	setinggi	670	tengah	706	tidakkah
635	seusai	671	tentang	707	tidaklah
636	sewaktu	672	tentu	708	tiga
637	siap	673	tentulah	709	tinggi
638	siapa	674	tentunya	710	toh
639	siapakah	675	tepat	711	tunjuk
640	siapapun	676	terakhir	712	turut
641	sini	677	terasa	713	tutur
642	sinilah	678	terbanyak	714	tuturnya
643	soal	679	terdahulu	715	ucap
644	soalnya	680	terdapat	716	ucapnya
645	suatu	681	terdiri	717	ujar
646	sudah	682	terhadap	718	ujarnya
647	sudahkah	683	terhadapnya	719	umum
648	sudahlah	684	teringat	720	umumnya
649	supaya	685	terjadi	721	ungkap
650	tadi	686	terjadilah	722	ungkapnya
651	tadinya	687	terjadinya	723	untuk
652	tahu	688	terkira	724	usah
653	tahun	689	terlalu	725	usai
654	tak	690	terlebih	726	waduh
655	tambah	691	terlihat	727	wah
656	tambahnya	692	termasuk	728	wahai
657	tampak	693	ternyata	729	waktu
658	tampaknya	694	tersampaikan	730	waktunya
659	tandas	695	tersebut	731	walau
660	tandasnya	696	tersebutlah	732	walaupun
661	tanpa	697	tertentu	733	wong
662	tanya	698	tertuju	734	yaitu
663	tanyakan	699	terus	735	yakin
664	tanyanya	700	terutama	736	yakni
665	tapi	701	tetap	737	yang
666	tegas	702	tetapi		

Lampiran 2

Daftar Blockword

1	alangkah	36	diagram	71	dikara
2	alami	37	diaken	72	dikotil
3	alhamdulillah	38	diakoni	73	dikotomi
4	barakah	39	diakritik	74	diktator
5	bismillah	40	diakronis	75	diktatorial
6	ejawantah	41	dialek	76	dilatasi
7	istilah	42	dialektik	77	dilatometer
8	majalah	43	dialektika	78	dilematik
9	makalah	44	dialektologi	79	diletan
10	masalah	45	dialisis	80	dimensi
11	memutah	46	dialog	81	dimorfik
12	mentelah	47	dialogis	82	dimorfisme
13	pelalah	48	diameter	83	dinamika
14	bus	49	diapositif	84	dinamis
15	bank	50	diare	85	dinamisator
16	belas	51	diastole	86	dinamisme
17	abdurrahman	52	diatermi	87	dinamit
18	afiliasi	53	diatesis	88	dinamo
19	balikpapan	54	diatom	89	dinamometer
20	pemerintah	55	diatonak	90	dinati
21	pepatah	56	diatorsi	91	dinosaurus
22	perintah	57	diatosia	92	diode
23	sebelah	58	didaktik	93	dioksida
24	sedekah	59	didaktis	94	diorama
25	sekolah	60	didaktius	95	diploid
26	serakah	61	diensefalon	96	diploma
27	silsilah	62	diferensial	97	diplomasi
28	diabetes	63	diferensiasi	98	diplomat
29	diadem	64	difluensi	99	diplomatik
30	diaforetik	65	difteri	100	diplomatis
31	diafragma	66	difusi	101	dipsomania
32	diagnosa	67	digdaya	102	direksi
33	diagnosis	68	digital	103	direktorat
34	diagnostik	69	digraf	104	direktorium
35	diagonal	70	digresi	105	direktris

106	direktur	141	disparitas	176	keleneng
107	dirgahayu	142	dispensasi	177	kelengkeng
108	dirgantara	143	dispenser	178	keliar
109	dirigen	144	disposisi	179	keliling
110	disagio	145	dispresi	180	kelola
111	disakarida	146	dispreunia	181	kelompok
112	disekuilibrium	147	distikon	182	kelontong
113	disentri	148	distilasi	183	keluar
114	disertasi	149	distilator	184	keluarga
115	disfungsi	150	distingsi	185	kemarau
116	disharmoni	151	distingtif	186	kemarin
117	disimilasi	152	distribusi	187	kembali
118	disinfeksi	153	distributor	188	kemeja
119	disinfektan	154	disuasi	189	kemenyan
120	disinformasi	155	ditserse	190	kemilau
121	disinsentif	156	diversifikasi	191	kemudi
122	disintegrasi	157	diversitas	192	kemudian
123	disiplin	158	divestasi	193	kendala
124	diskonto	159	dividen	194	kendali
125	diskotek	160	divisi	195	kendara
126	diskredit	161	kebaya	196	kendari
127	diskrepansi	162	kecambah	197	kendati
128	diskresi	163	kecewa	198	kenduri
129	diskriminasi	164	kecoa	199	kental
130	diskriminatif	165	kecuali	200	kepala
131	diskualifikasi	166	kecubung	201	kepalang
132	diskulpasi	167	kedai	202	keparat
133	diskusi	168	kedawung	203	kepingin
134	dislokasi	169	kedelai	204	kepiting
135	dismembrasio	170	keladi	205	kepundan
136	dismutasi	171	kelahi	206	kerabat
137	disorder	172	kelambu	207	keramat
138	disorganisasi	173	kelapa	208	keramik
139	disorientasi	174	keledai	209	kerangka
140	disosiasi	175	kelelawar	210	keranjang

211	kerawang	246	megaspora	281	mempelam
212	kerbau	247	megasporangium	282	mempelas
213	kereta	248	meiosis	283	mempelasari
214	kerikil	249	mejana	284	mempening
215	keringat	250	mekanik	285	mempitis
216	keritik	251	mekanika	286	mempurung
217	keriut	252	mekanisasi	287	menaga
218	keroncong	253	mekanisme	288	menantu
219	kerongkongan	254	mekonium	289	menara
220	keropok	255	melabuai	290	menceret
221	keruan	256	melamin	291	mendira
222	kerubung	257	melanesia	292	mendoan
223	kerunyut	258	melanin	293	mendura
224	kerupuk	259	melankolia	294	mengerawan
225	ketika	260	melankolis	295	mengerna
226	ketimun	261	melanodermia	296	menggusta
227	ketoprak	262	melarat	297	mengiang
228	ketumbar	263	melase	298	mengkara
229	ketupat	264	melati	299	mengkaras
230	medali	265	melawah	300	mengkelan
231	medallion	266	melela	301	mengkirai
232	media	267	melempem	302	mengkirik
233	median	268	melinjo	303	mengkudu
234	medicinal	269	melodi	304	meningitis
235	meditasi	270	melodius	305	meniran
236	mediterania	271	melodrama	306	meniskus
237	medium	272	melompong	307	menkuang
238	medula	273	melukut	308	menopause
239	medusa	274	melulu	309	menoragia
240	megafon	275	memoar	310	menserendahi
241	megalit	276	memorandum	311	mensiang
242	megalitikum	277	memorat	312	menstruasi
243	megalomania	278	memori	313	mensurasi
244	megalosit	279	memorial	314	mentalitas
245	megapolis	280	mempelai	315	mentari

316	mentaruh	351	meringis	386	metafisika
317	mentaus	352	meristem	387	metafora
318	mentega	353	merkantilisme	388	metaforis
319	mentifakta	354	merkubang	389	metalik
320	mentigi	355	merkuri	390	metalinguistik
321	mentilau	356	merkurius	391	metalografi
322	mentimun	357	merkuro	392	metaloid
323	mentolo	358	merkurokrom	393	metalurgi
324	mentora	359	merlilin	394	metalurgis
325	menuet	360	merlimau	395	metamorf
326	meracang	361	merogoni	396	metamorfisme
327	meraga	362	merpati	397	metamorfosis
328	meragai	363	merpitis	398	metana
329	merakan	364	mertajam	399	metanol
330	merambung	365	mertapal	400	metari
331	merana	366	mertelu	401	metastasis
332	merangsi	367	mertua	402	metatesis
333	merangu	368	merubi	403	metazoa
334	meranti	369	merunggai	404	meteograf
335	merapu	370	mesiu	405	meteogram
336	merasi	371	mesomorf	406	meteor
337	merawal	372	mesopause	407	meteorit
338	merawan	373	mesotoraks	408	meteorologi
339	merbau	374	mesozoa	409	meterai
340	merbulan	375	mesozoikum	410	metode
341	merdangga	376	mestika	411	metrologi
342	merdeka	377	metabolik	412	metromini
343	merdesa	378	metabolis	413	metronom
344	merembung	379	metabolisme	414	metronomis
345	meriam	380	metafisik	415	metropolis
346	meriang	381	meringis	416	metropolisasi
347	merica	382	meristem	417	metropolitan
348	meridian	383	merkantilisme	418	metroragia
349	merikan	384	merkubang	419	meunasah
350	merinding	385	merkuri	420	mezosopran

421	penaka	456	penjuru	491	teriak
422	penalti	457	penomah	492	terigu
423	penanggah	458	pensiun	493	terikit
424	penaram	459	pentagin	494	terima
425	penasaran	460	pentagor	495	terindil
426	penatu	461	pentagram	496	teripang
427	penatua	462	pentameter	497	terista
428	pencalang	463	pentatonik	498	teritik
429	pendaga	464	pentolan	499	teritip
430	pendahan	465	terada	500	teritis
431	pendapa	466	terajang	501	terjemah
432	pendekar	467	teraju	502	terlanjur
433	pendeta	468	terakota	503	terlantar
434	pendongkok	469	terakup	504	terlentang
435	pendulum	470	terala	505	terminal
436	penembahan	471	terali	506	terminologi
437	penetrasi	472	teraling	507	termoelektrik
438	penewu	473	teralogi	508	termograf
439	penganan	474	terampil	509	termogram
440	pengantin	475	teranas	510	termohigrograf
441	pengapuh	476	terapang	511	termoklin
442	pengaruh	477	terapi	512	termolabil
443	pengawinan	478	terasi	513	termometer
444	pengerih	479	teraso	514	termoplastik
445	penggawa	480	terasul	515	termostat
446	penghulu	481	teratai	516	terobos
447	penguin	482	teratak	517	terombol
448	pengulun	483	teratu	518	teromol
449	peniaram	484	terau	519	terompah
450	penisilin	485	terawang	520	terompet
451	penitensi	486	teraweh	521	terondol
452	peniti	487	terenang	522	terongko
453	penjajab	488	terendak	523	teropong
454	penjalin	489	terentang	524	teroris
455	penjara	490	Teretet	525	terorisme

526	permisif	561	pertusis	596	sampai
527	permutasi	562	peruak	597	santai
528	pernikel	563	peruan	598	santiaji
529	perogol	564	peruang	599	saudari
530	perohong	565	perubalsem	600	sejati
531	peroi	566	perudang	601	sekian
532	peroksida	567	perumpung	602	selatan
533	peroksidase	568	perunggu	603	seledri
534	peroksisoma	569	perunjung	604	selesai
535	peromeal	570	perupuk	605	selokan
536	perompak	571	perusa	606	sembilan
537	perosok	572	perversi	607	sembunyi
538	perpatih	573	perwara	608	senapan
539	persada	574	perwira	609	senapati
540	persangga	575	percaya	610	sendiri
541	persegi	576	radiasi	611	senopati
542	persekot	577	ramadan	612	seperei
543	persentase	578	ramadhan	613	seperti
544	persentil	579	ramai	614	serai
545	persepsi	580	rambutan	615	sertifikasi
546	perseptif	581	rampai	616	sesuai
547	perseus	582	rangkai	617	situasi
548	perseverasi	583	rantai	618	solusi
549	persneling	584	reaksi	619	spekulasi
550	personafikasi	585	redaksi	620	suami
551	personal	586	referensi	621	sulawesi
552	personalia	587	refleksi	622	sungai
553	personel	588	regenerasi	623	survei
554	perspektif	589	reparasi	624	suplai
555	persuasi	590	resensi	625	teknologi
556	persuasif	591	resepsi	626	tradisi
557	pertai	592	retribusi	627	usai
558	pertama	593	revisi	628	wartawan
559	pertiwi	594	revolusi	629	wawasan
560	pertubasi	595	rohani	630	yahudi

Lampiran 3

Aturan derivasional untuk prefiks

Prefiks (alomorf)	Variasi (morf)	Aturan
meng	meng	+ vokal | k | g | h ... , contoh : ambil → mengambil ikat → mengikat hilang → menghilang
	meny	+ s... , contoh : sapu → menyapu sisir → menyisir
	mem	+ b | f | p... , contoh : beku → membeku fitnah → memfitnah pukul → memukul
	men	+ c | d | j | t… , contoh : cuci → mencuci darat → mendarat jual → menjual tukar → menukar
	me	+ l | m | n | r | y |w… , contoh : lintas → melintas makan → memakan nikah → menikah rusak → merusak wabah → mewabah yakin → meyakin(kan)
peng	peng	+ vokal | k | g | h... , contoh : ikat → pengikat urus → pengurus ganggu → penggangu halus → penghalus
	peny	+ s... , contoh : saring → penyaring
	pem	+ b | f | p… , contoh : baca → pembaca fitnah → pemfitnah pukul → pemukul

	pen	+ c | d | j | t… , contoh : cuci → pencuci datang → pendatang jual → penjual tukar → penukar
	pe	+ l | m | n | r | y |w… , contoh : lintas → pelintas makan → pemakan rusak → perusak warna → pewarna
ber	bel	+ ajar , contoh : ajar → belajar
	be	+ r | KVr... , contoh : rencana → berencana kerja → bekerja
	ber	+ seluruh huruf selain morf bel dan ber , contoh : tamu → bertamu
per	pel	+ ajar , contoh : ajar → pelajar
	pe	+ r | KVr... , contoh : ramal → peramal
	per	+ seluruh huruf selain morf pel dan per , contoh : kaya → perkaya
ter	te	+ r... , contoh : rasa → terasa
	ter	+ K | V… , dimana K≠r , contoh : atur → teratur