POLITEKNIK NEGERI BANDUNG – POLBAN
JL. Gegerkalong Hilir, Ciwaruga Bandung, Tel. (022)200.7651, Fax. (022)201.3889
E-mail: jtmpolban_semnas_st_2010@yahoo.com , URL: www.polban.ac.id

Sistem Manajemen Dokumen: Pengorganisasian dan Temu Kembali Dokumen Artikel Ilmiah Elektronik

Urip T Setijohatmo, Setiadi Rachmat, Irwan Setiawan

Jurusan Teknik Komputer dan Informatika, Politeknik Negeri Bandung
Jl. Gegerkalong Hilir, Ds Ciwaruga, Bandung, Telp dan Fax (022) 2013789 dan 2013788

e-mail: angki@polban.ac.id

Abstrak

Perkembangan teknologi jaringan dan bermunculannya mesin pencari (search engine) dilingkungan web, memungkinkan setiap orang untuk memperoleh informasi sebanyak-banyaknya dan menyimpannya pada media penyimpanan (baik pada komputer standalone atau yang terhubung ke suatu jaringan). Hal ini menimbulkan masalah baru pada saat pencarian kembali dokumen yang dibutuhkan karena pada umumnya dokumen elektronik yang telah didapat tidak disimpan dengan pengorganisasian file yang baik. Sehingga semakin banyak dokumen yang tersimpan akan menyebabkan semakin sulit pula mencari informasi yang spesifik pada suatu dokumen yang telah dimiliki. Hal ini disebabkan oleh masih minimnya search engine di lingkungan offline/standalone dan intranet, dimana arsitekturnya dan kebutuhannya berbeda dengan Web. Sehingga memanfaatkan search engine untuk Web tidak efektif bila diimplementasikan untuk lingkungan offline/standalone dan intranet. Atas dasar hal-hal tersebut, maka perlu dikembangkan suatu mekanisme pengorganisasian dan temu kembali dokumen dalam lingkungan intranet dalam hal ini di lingkungan Politeknik Negeri Bandung. Sistem yang akan dikembangkan untuk itu melingkupi pengorganisasian, mekanisme document aquisition dan memiliki fasilitas pencarian berdasarkan kriteria tertentu. Proses pencarian atau penelusuran informasi (information retrieval) akan menghasilkan dokumen-dokumen yang relevan dengan keywords yang diminta. Untuk membantu user mendapatkan suatu dokumen yang dicari diantara dokumen-dokumen yang relevan tersebut ditampilkan abstraksi dan similar documents dari suatu dokumen yang sedang dipilih. Karenanya mekanisme penyimpanan mempersiapkan relevansi dokumen dan similaritas dokumen. Mendahului mekanisme tersebut, dipersiapkan ekstraksi dokumen menjadi term-term melalui tokenizing, minimasi jumlah token yang dihasilkan dan stemming. Jika document relevance berkaitan dengan keseringan kemunculam term pada dokumen yang merepresentasikan kekentalannya pada dokumen, document similarity berkenaan dengan kemiripan topik sehingga kesamaan semantik dipertimbangkan. Sistem yang dikembangkan menggunakan tools Java Netbeans 6.5.1 dengan dukungan java database yang disediakan. Penggunaan tools ini cukup memadai dikarenakan dalam penanganan database tidak memerlukan fitur-fitur DBMS seperti concurrency control dan integrity constraint.

Kata kunci : *document retrieval, similarity document, relevant document.*

1. PENDAHULUAN

Keberadaan World Wide Web merepresentasikan era informasi. Menurut [4] lautan informasi ini mengandung 2.3 milyard dokumen digital dan para analis memprediksi jumlah dokumen pada Web akan berkembang delapan kali pada tahun 2000 - dan 100 kali pada dekade berikutnya. Hal tersebut mendapat tanggapan dari peneliti Teknologi Informasi (IT) dengan melakukan banyak penelitian yang bertujuan untuk mengoptimalisasi *search engine* agar dapat memperoleh suatu informasi yang tepat dan sesuai dengan kebutuhan seseorang dari jumlah dokumen yang banyak. Menurut [5], bermunculan search engine baru atau optimasinya untuk mengantisipasi pertumbuhan jumlah dokumen pada Web.

Kombinasi dari dua hal tersebut memungkinkan setiap orang untuk memperoleh informasi dalam bentuk dokumen elektronik sebanyak-banyaknya. Masalahnya adalah semakin banyaknya dokumen yang terkumpul (disimpan pada *offline/standalone* komputer atau intranet) bukan berarti semakin mudah orang tersebut *meretrieve*/mendapatkan kembali dokumen yang dibutuhkan, tetapi justru akan berdampak pada sulitnya pencarian informasi

yang spesifik pada suatu dokumen. Hal ini disebabkan oleh masih minimnya *search engine* di lingkungan *offline/standalone* dan intranet, dimana arsitekturnya dan kebutuhannya berbeda dengan Web. Sehingga memanfaatkan *search engine* untuk Web tidak efektif bila diimplementasikan untuk lingkungan *offline/standalone* dan intranet.

Atas dasar hal-hal tersebut, maka perlu dikem bangkan suatu mekanisme pengorganisasian dan

temu kembali dokumen dalam lingkungan intranet. Dengan aplikasi sistem manajemen dokumen diharapkan akan memberikan hal-hal:

- Tidak ada duplikasi dokumen
- Terdapat mekanisme sharing kekayaan dokumen pada setiap jurusan
- Kemudahan akses
- Memfasilitasi bila user ingin mendapatkan dokumen yang mirip

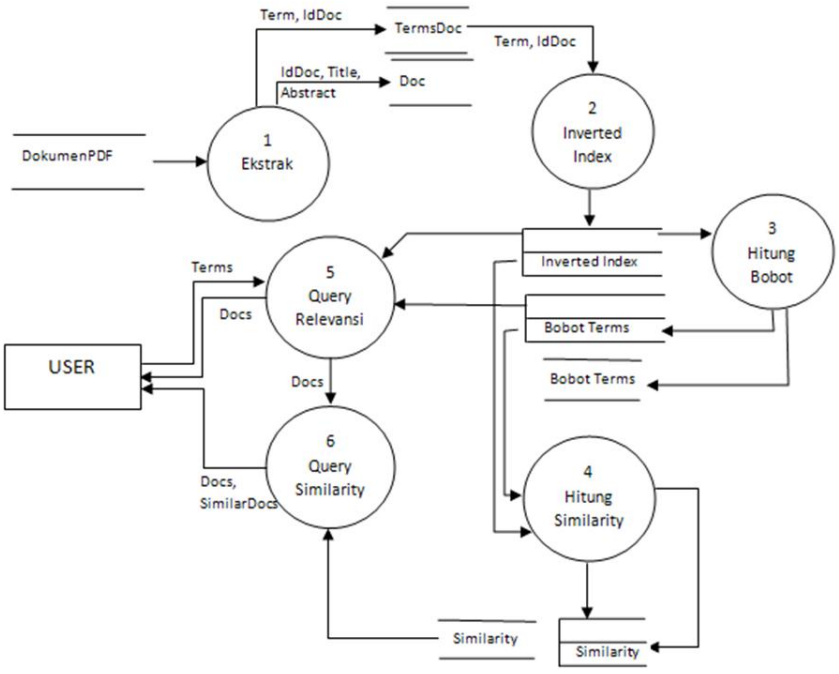
Dokumen yang menjadi perhatian merupakan dokumen penelitian berformat pdf. Dengan fasilitas internet, dosen setiap jurusan mendapatkan dokumen yang dicarinya dengan cara googling dan mendownload yang kemudian dokumen sebagai hasilnya disimpan masing-masing. Pembaca dokumen biasanya mengamati abstrak untuk memastikan isinya sesuai atau tidak dengan yang dicarinya. Bila sesuai dokumen dibaca, dan bila belum puas pembaca ini akan mencari dokumen yang mirip. Intensitas yang tinggi mengakibatkan penumpukan dokumen yang tidak efisien, pengaksesan kembali yang sulit, dan sharing yang belum dimungkinkan. Dari uraian di atas beberapa hal requirement adalah sebagai berikut:

- Ekstraksi dokumen penelitian. Selain memperhatikan aturan pembentukan kata, juga section sebagai pemisah isi sebagai aturan format penelitian.

Penyediaan mekanisme penyimpanan data yang menghindari duplikasi, kemudahan akses dan memungkinkan sharing tanpa harus mengetahui posisi fisik dokumen.

2. PEMODELAN SISTEM

2.1. Analisis Kebutuhan



Gambar-1. Data Flow Diagram Level-1 Kebutuhan Engine

- Penyediaan kemampuan (fitur) menentukan relevansi terms pada dokumen dokumen.
- Penyediaan kemampuan (fitur) menentukan kemiripan dokumen.

- Penyediaan kemampuan (fitur) query mendapatkan dokumen sesuai istilah (terms) yang diinginkan.
- Penyediaan kemampuan (fitur) query mendapatkan dokumen yang mirip dengan dokumen tertentu.

3. HASIL PENGUKURAN DAN DISKUSI

3.1. Spesifikasi Kebutuhan

Proses yang terlibat adalah ekstraksi dari dokumen pdf menjadi term-term, lalu proses membentuk inverted index, dilanjutkan dengan

proses menghitung bobot term dan similaritas dokumen dan akhirnya adalah pemrosesan query relevan dengan term query dan query dokumen yang mirip.

3.1.1. Proses Ekstraksi Dokumen

Intinya, masalah ekstraksi kalimat menjadi term-term terdiri dari beberapa subproses, yaitu: tokenizing, verifikasi stopword dan stemming. Dalam ilmu dan bahasa komputer, seperti pada [1], parsing (atau lebih umumnya disebut sebagai analisis sintatik) adalah suatu proses menganalisa suatu rangkaian *token* untuk menentukan struktur gramatikal dengan menjunjung *formal grammar* atau susunan kalimat dan aturan-aturan dalam membentuk suatu kalimat. Namun dalam pengembangan sistem manajemen dokumen ini *parsing* merujuk pada proses mengidentifikasi unit terkecil *(token*) dari suatu struktur kalimat atau disebut sebagai *tokenizing*.

Tujuan dilakukannya *parsing* atau tepatnya *tokenizing* ini adalah untuk mendapatkan *term-term* yang nantinya akan diindeks. Pengidentifikasian *token* dilakukan untuk teks yang dipisah dengan ”spasi” atau “enter” dan mengabaikan *term-term* ataupun karakter yang dianggap tidak penting dalam suatu dokumen. Selain itu beberapa kasus yang mungkin terjadi pada komposisi kata pada kalimat *english written* yang harus ditangani pada proses *parsing* adalah : penanganan karakter khusus berbagai jenis seperti akronim (misalnya: IBM, SQL), singkatan (misalnya: U.S.A), model (F-16), mata uang (misalnya $123.92). Selain itu juga phrase, seperti nama tempat (misalnya: new york), kepemilikan (my book), adjective (big book).

Struktur kata dalam tata bahasa Inggris yang mungkin seharusnya melingkupi hal-hal berikut:

- Kata diapit oleh tanda apposthrope, tunggal atau ganda
- Kata diapit oleh tanda kurung, atau tanda kurung kotak atau kurung kurawal
- Kata diahiri oleh tanda titik, atau titik-koma, atau koma, atau titik ganda
- Sedangkan kata itu sendiri adalah kumpulan abjad atau angka. Bila diantara huruf ada titik, maka setelah titik tersebut harus langsung huruf karena bila setelah titik adalah white space maka setelah titik tsb dianggap sebagai milik dari kata berbeda. Hal ini mengadopsi kemungkinan kata seperti : U.S.A atau $10,500.50

Maka, dari kesimpulan tersebut di atas, BNF kata dalam bahasa natural Inggris:

<Token> ::= <punct1> <term> [<punct1> | <term> <punctuation2>

<punct1> ::= <kurung> | <appost>

<punct2> ::= <ttk_koma1> | <tsertany>

<ttk_koma1> ::= "."<wspace> | ","<wspace>| ";" | ":"

<wspace> ::= " " | "\t" | “\n”

<kurung> ::= "(" | ")" | "[" | "]" | "{"| "}"

<tsertany> ::= "?" | "!"

<appost> ::= "“"| "’" ;

<term> ::= <kata> | <abjad> | <nmrik>

<abjad> ::= <huruf> [<ttk_koma2>] <huruf>

<huruf> ::= “A” |“ B” |...|“Z” | “a” | “b”| |”z”

<nmrik> ::= [<mata_uang>]<angka>

<mt_uang> ::= “$” | ”¥” | “£”

<angka> ::= “0” | “1” | ... | “9”

Namun untuk keperluan aplikasi sistem manajemen dokumen semua huruf dijadikan huruf kecil, sehingga huruf menjadi:

<huruf> ::= “a” | “b” | ... |”z”

Sebelum melakukan ekstraksi dilakukan terlebih dahulu pengecekan apakah dokumen mempunyai abstrak. Jika mempunyai abstrak, apakah sudah pernah diakuisisi dengan cara membandingkan abstraknya. Hal ini dilakukan untuk menghindari duplikasi dokumen.

Hal-hal tersebut di atas direpreresentasikan pada algoritme di bawah ini. Pada prosedur baca_dok, dilakukan pembacaan perbaris sambil dilakukan tokenizing dan pengecekan apakah token adalah “abstract”. Bila ya, lakukan pengambilan isi abstract, yaitu diambil semua token setelah itu sampai bertemu *tag* introduction. Bila berhasil

abstract dibandingkan dengan abstract-abstract yang ada untuk memastikan tidak terjadi duplikasi.

```
procedure baca_dok(f: FILE)
        baris: String; token: String[] ; dapat, lanjut: boolean; count: integer;
        i, j, n : integer; intro: boolean; abstract: String[]; stem: String[];
begin
    dapat := FALSE; count := 0; lanjut := TRUE;
    /* mencari awal abstrak
    openfile (f);
    baris := readln(f);
    while(not eof(f) and lanjut)
    begin
        token := tokenize(baris);
        while(count < = 200 and not dapat)
        begin
            i := 1;
            while(i <= token.length and not dapat)
            begin
            if (token[i] = "abstract")
                dapat := TRUE;
                else
                begin
                  i := i + 1;
                  count := count + 1;
                end
            end
        end
        if (count <=200 and dapat)
            lanjut := FALSE;
    end
 if(dapat)
         begin
                intro := FALSE;
                baris := readln(f);
                while(not EOF(f) and not intro)
                 begin
                        token := tokenize(baris);
                        i := 1;
                        while(i <= token.length and not intro)
                         begin
if(token[i] = "Introduction"or token[i] = "1. Introduction" or token[i] = "I. Introduction")
                                intro := TRUE;
                                else
                                i := i + 1;
            end
                        if (not intro)
                         begin
                                abstrct := concat(abstract, baris);
                                baris := readln(f);
                        end
                end
        end
        if(not CekDouble(abstract))
         begin
                baris := readln(f);
                while(not EOF(f))
                 begin
                        token := tokenize(baris)
                        i := 1; n := 1;
                        while(i <= token.length)
                         begin
                                if(not adaDiStopList(token[i]))
                                 begin
                                        stem[n] := stemming(token[i]);
                                        n := n + 1;
                                 end
                                i := i + 1;
                         end
                        baris := readln(f);
                end
        end
end
function tokenize(s: string): String[]
        token: String[] ; tok: String;
        i, j, k : integer;
begin
        i,k := 1;
        while(i <= s.length)
         begin
if(not isWhiteSpace(s[i]) and not isStpWord(s[i]) )
                 begin
                        tok[j] := s[i];
                        j := j + 1;
                 end
                if(isWhiteSpace(s[i]))
                begin
                        token[k] := tok;
                        k := k + 1;
                        j := 1;
                 end
                i := i + 1;
         end
        return token;
end

function isWhiteSpace(s: string): boolean
begin
        if(s = " " or s = "\t" or s = "\n")
                return TRUE
```

```
        else
                return FALSE
end
function isStpWord(s: string): boolean
begin
        if(s = "." or s = ","or s = "-" or s = "_" or
s = "-")
                return TRUE
        else
                return FALSE
end
```

3.1.2 Proses Membentuk Inverted Index

Indexing merupakan sebuah proses untuk mendukung *efficient retrieval* dari seluruh *record* data dengan suatu nilai *key* yang diberikan. Sistem Manajemen Dokumen ini menggunakan struktur indeks yang disebut sebagai *inverted index*. Dengan melihat tujuan dari *retrieval machine*, *inverted index* membantu mencari dokumen yang dibutuhkan sesuai dengan *query* atau *keyword* yang dimasukkan pengguna dari kata-kata atau *term-term* yang dikumpulkan dan terkandung dalam dokumen yang telah disimpan. Kata-kata tersebut menunjuk pada dokumen yang mengandungnya.

Inverted Index	
Word	**Documents**
cool	Document-1
cow	Document 2, Document 3, Document 4
look	Document 1, Document 3, Document 4
moo	Document 1, Document-4
say	Document-2

Gambar-2. Inverted Index

```
type isi = record
        jumlah: integer;
        bobot: float;
end

inverted = record
        aki: integer;
        term: integer;
        aka: integer;
        termdoc: array[] of isi;
end

termdocdb = record
        doc: string;
        jumlah: integer;
end
terms, docs: String[];
invNdx: array[][] of inverted;

function uploadinvertedndx(): array[][] of inverted
        td: termdocDB;
        k,m, posisi: integer;
    begin
        terms := π terms(termsdoc);
        docs := π docs(termsdoc);
        invNdx:=                    new
inverted[terms.length][docs.length]
        for(i = 1 to docs.length)
         begin
                invNdx[][i] := docs[i];
         end
        for(i = 1 to terms.length)
         begin
                k := insertBST(terms[i];
                td := π docs, jumlah(σ terms =
terms[j] (termsdoc));
                for(m = 1 to td.length)
                 begin
                        posisi          :=
cariPosisi(td[m].doc)

        invNdx[k][posisi].isi.jumlah:=
td[m].jumlah;
                 end
         end
        return invNdx;
    end
function insertBST(s: String): integer
        i: integer;
 begin
        i := 1;
        while(invNdx[i].simbol!="    "and
s!= invNdx[i].simbol)
         begin
                if(s < invNdx[i])
                        i               :=
invNdx[i].aki;
                else
                if(s > invNdx[i].simbol)
                        i := invNdx.aka;
         end
        return i;
 end
function cariPosisi(s: String) : integer
    i: integer; dapat: boolean;
 begin
    i := 1; dapat := FALSE;
    while(i < docs.length)
     begin
        if(s = docs[i])
                dapat := TRUE;
        else
                i := i + 1;
     end
```

```
        if(dapat)
                return i;
        else
                return 0;
end
```

3.1.3 Proses Pembobotan *Term*

Pembobotan *term* (*term weighting*) merupakan salah satu operasi yang dibutuhkan untuk membantu suatu proses *information retrieval* yaitu dengan menghitung kemunculan frekuensi suatu kata atau *term* pada sebuah dokumen.

Pembobotan *term* dibutuhkan dalam menentukan peringkat dokumen (*document ranking*) – pada operasi pembobotan dokumen, dimana hal itu dilakukan untuk mencari besarnya kemiripan atau relevansi antara dokumen dengan *query*, proses ini dilakukan dengan cara melihat relevansi antara *term-term* atau kata-kata yang terdapat di dalam suatu dokumen dengan *query* pencarian.

Metode pembobotan *term* yang digunakan pada[4] adalah TF-IDF. Bobot TF–IDF (*term frequency–inverse document frequency*) adalah suatu bobot yang sering digunakan dalam *information retrieval* dan *text mining*. Metode TF-IDF merupakan suatu cara untuk memberikan bobot hubungan suatu kata (*term*) terhadap dokumen. Metode ini menggabungkan dua konsep untuk perhitungan bobot yaitu, frekuensi kata yaitu *term tj*) kemunculan sebuah kata di dalam sebuah dokumen tertentu dan *inverse* frekuensi dokumen yang mengandung kata tersebut. Frekuensi kemunculan kata di dalam dokumen yang diberikan menunjukkan seberapa penting kata tersebut di dalam dokumen tersebut. Frekuensi dokumen yang mengandung kata tersebut menunjukkan seberapa umum kata tersebut. Sehingga bobot hubungan antara sebuah kata dan sebuah dokumen akan tinggi apabila frekuensi kata tersebut tinggi di dalam dokumen dan frekuensi keseluruhan dokumen yang mengandung kata tersebut yang rendah pada kumpulan dokumen (*database*). Variasi skema pembobotan *term* sering digunakan oleh *search engine* sebagai *central tool* dalam proses *scoring* dan *ranking* sebuah relevansi dokumen yang diberikan sebuah *user query*.

Menggunakan rumus TF/IDF yang di bawah

ini diberikan ilustrasi bagaimana proses menghitung bobot dengan data yang digunakan kembali Di sebelah kiri adalah term, sedangkan disebelah kanan adalah dokumen yang mengandung term tsb disertai jumlah term tsb pada dokumen yang bersangkutan

term1={[doc1,1],[doc3,1]}

term2={[doc1,1],[doc3,2],[doc4,1],[doc5,1]}

term3={[doc2,1], [doc4,1]}

term4={[doc1,1], [doc4,1], [doc5,4]}

term5={[doc2,1], [doc4,3]}

term6={[doc3,2]}

term7={[doc1,1], [doc3,1], [doc5,2]}

term8={[doc4,2]}

term9={[doc2,1]}

term10={[doc3,1], [doc5,3]}

Fakta lain dari data di atas untuk kepentingan TF/IDF adalah jumlah dokumen yang mengandung *term*.

term1 = 2 term2 = 4 term3 = 2 term4 = 3

term5 = 2 term6 = 1 term7 = 3 term8 = 1 term9 = 1 term10 = 2

Dari data dokumen-dokumen di atas, juga dapat diketahui bahwa total dokumen yang ada adalah 5 buah. Menghitung bobot keterhubungan *term* dengan dokumen untuk setiap *term* pada dokumen dapat dilakukan sebagai berikut:

$$W_{ij} = tf_{ij} \times \left(\left(\log\frac{N}{n}\right) + 1\right)$$

W_{ij} = bobot kata term t_j terhadap dokumen d_i

tf_{ij} = jumlah kemunculan kata / term t_j dalam d_i

N = jumlah semua dokumen yang ada

n = jumlah dokumen yang mengandung kata / term t_j (Minimal ada satu

Contoh:

$$W_{1,1} = 1 \times \left(\left(log\frac{5}{2}\right) + 1\right) = 1.3979$$

Algoritme menghitung bobot sebagai berikut:

```
hitungJmlDokUtkTermtj(bariske: int, nkolom: int) : int /* jumlah dokumen = nkolom
        jml, j : int;
begin
                jml := 0;
                for j = 1 to nkolom
                        if(tfidf[bariske, j].jml > 0)
                                jml := jml + 1;
                return jml;
end
```

```
hitungbobot(nbaris: int, nkolom: int)
        n : int          /* jumlah dokumen mengandung term tj
begin
                for i = 1 to nkolom
                        for j = 1 to nbaris
                                n := hitungJmlDokUtkTermtj(j,nkolom)

        tfidf[i,j].weight:= tfidf[i,j].tf *(log(nkolom/n)+1)
end
```

3.1.4. *Similarity*

Similaritas dokumen dapat diukur melalui dua pendekatan, pengukuran jarak diantara dokumen (vectorial model) dan similaritas semantik (semantic similarity). Pendekatan pertama, vectorial model, selanjutnya meluas ke model bahasa n-gram. Pendekatan kedua mempunyai nama berbeda diantaranya adalah corpus based, dan content based. Sejauh ini *good guess* yang bisa diberikan adalah bahwa akan lebih baik mengukur similaritas menggunakan model semantic similarity, karena dengan maksud/ide yang sama tapi menggunakan kata/istilah yang secara sintaks berbeda bila menggunakan model vektorial menghasilkan similaritas yang rendah. Menurut [8], terdapat beberapa metode perhitungan similaritas semantik dokumen. Salah satu dari metode yang ditinjau adalah LSA(Latent Semantic Analysis). Latent Semantic Analisis adalah sebuah teori dan metode untuk mengekstrak dan merepresentasikan konteks yang digunakan sebagai sebuah arti kata dengan memanfaatkan komputasi statistik untuk sejumlah corpus yang besar dari teks.

Ide yang melandasi adalah melakukan agregat dari semua konteks kata yang diberikan, baik yang ada ataupun tidak dalam menyediakan batasan untuk menentukan kesamaan arti dari kata terhadap set kata yang lainnya.

LSA yang memadai akan merefleksikan pengetahuan manusia yang dinyatakan dalam berbagai cara. Dengan metode LSA, semantic dari teks dapat ditentukan melalui proses perhitungan *semantic similarity* yang merupakan proses yang memerlukan keterlibatan beberapa ilmu seperti bahasa, komputer, matematika logik dan domain yang bersangkutan. Langkah awal perhitungan kesamaan semantik adalah mengacu kepada kesamaan secara terminologi. Terminologi yang dimaksud dapat meliputi *class*, *property* hingga *instances*. Menurut Euzenat [2], pendekatan secara terminologi ada yang berdasarkan *string based* dan *language based*. Pada penelitian ini akan ditinjau pendekatan untuk *languange based* dengan menggunakan Latent Semantic.

Tahapan LSA meliputi 3 tahap utama:

1. Parsing dan pembobotan

Parsing meliputi *tokenizing*, *filtering* dan *stemming* dan pembobotan terhadap teks yang terkandung dalam *corpus* dimana salah satu metode pembobotan teks adalah tf-idf. Tahap ini telah dilakukan. Jadi LSA melingkupi pembobotan tf/idf.

2. SVD (Singular Value Decomposition)

Perhitungan dekomposisi matriks menjadi 3 bagian matriks baru dan pengerucutan matriks tersebut menjadi matriks baru

3. Vector manipulation

Manipulasi terhadap matriks vector kolom yang ada pada matriks sehingga menjadi matriks dengan bobot antar vector.

Secara sederhana proses dari LSA adalah:

- Merepresentasikan teks dalam matriks, baris menujukkan terms dan kolom adalah dokumen yang mengandung term tsb. Setiap cell berisi freq kata pada setiap dokumen.
- Lakukan perhitungan bobot dari frekuensi yang ada pada cell baru dalam matriks.
- Selanjutnya lakukan singular value decomposition (SVD) terhadap matriks ini. Matriks yang direpresentasikan menggunakan SVD akan diurai menjadi tiga komponen matriks: matriks vektor singular kiri, martiks nilai singular, dan matriks vektor singular kanan sebagai berikut:

$A_{mn} = U_{mm}\, S_{mn}\, V_{nn}^{T}$ dimana:

A = matriks yang didekomposisi

U = matriks ortogonal U (matriks vektor singular kiri) berdimensi mxm

S = matriks diagonal S (matriks nilai singular) berdimensi mxn dengan nilai terurut menurun

V_{nn}^{T} = transpose matriks orthogonal V (matriks vektor singular kanan) berdimensi nxn

m = jumlah baris matriks

n = jumlah kolom matriks

Salah satu keperluan penggunaan SVD adalah penghitungan nilai *similarity* untuk *term* dan dokumen yang dilakukan melalui perkalian matriks U, S dan V^t hasil reduksi. Vektor baris digunakan untuk mencari nilai *similarity term* dan vektor kolom untuk *similarity* dokumen. Namun, untuk melakukan penghitungan salah satu *similarity* (*term*

atau document *similarity* saja) dapat dilakukan melalui perkalian yang lebih sederhana yaitu:

- penghitungan *term similarity*: mengalikan matriks U dengan S (UxS) dan perhitungan vektor barisnya sebagai nilai *similarity*nya
- sedangkan untuk pencarian dokumen *similarity* mengalikan matriks V dengan S (VxS) dan perhitungan vektor barisnya sebagai nilai *similarity*nya.

3.2.Query Relevansi Dokumen

Proses ini diawali ketika user menginginkan dokumen-dokumen yang mengandung term-term yang dimaksudkan. Term-term tersebut dimasukkan melalui form inputan searching. Kompleksitas yang mungkin adalah kombinasi term bila jumlahnya lebih dari sebuah. Seperti halnya dalam teori himpunan, kombinasi yang mungkin adalah sesuai aturan operasi OR:

a OR b : dokumen mengandung term a atau term b atau mengandung keduanya.

Ditinjau dari sudut kekentalan arti, dokumen-dokumen yang mengandung term a dan term b mempunyai nilai relevansi lebih tinggi dari pada dokumen-dokumen yang hanya mengandung term a atau hanya term b. Dasar pemikirannya adalah ketika yang dimaksudkan adalah user menginginkan lebih dari satu term, maka berarti menginginkan yang spesifik. Misalkan user meginginkan dokumen mengandung term "computer" dibandingkan dengan bila menginginkan dokumen mengandung dua term "computer vision", maka dokumen yang pertama akan menghasilkan dokumen-dokumen apapun yang mengandung kata "computer" sedangkan yang kedua hanya dokumen-dokumen yang mengandung term "computer vision" dimana hasilnya lebih spesifik.

Maka bila query termnya adalah t1 t2 … tn, operasi yang dilakukan adalah

set dok mengandung t1 and t2 and … and tn **union** set dok mengandung t1 and t2… and tn-1

union set dok mengandung t2 and t3… and tn

union set dok mengandung t2 and t4… and tn1

union set dok mengandung tn-1 and tn

union set dok mengandung t1 or t2 or … or tn

dimana operasi awal adalah **and** untuk semua term query, operasi akhir adalah **or** untuk semua term query sedangkan diantara keduanya adalah operasi **and** semua subset mulai panjang dua sampai n-1.

3.3. Query Dokumen Mirip

Dari hasil pencarian sesuai term, dokumen-dokumen yang relevan dengan term-term inilah yang menjadi inputan untuk mendapatkan dokumen-dokumen lain yang mirip. Proses pencarian dilakukan pada database Dokumen dimana nilai similarity (direkam sebagai hasil perhitungan document similarity) dipilih berdasarkan yang paling besar menurun (descending).

4. KESIMPULAN

Pengorganisasian dokumen artikel ilmiah elektronik meliputi ekstraksi dokumen menjadi term-term, pembentukan inverted index untuk menghitung bobot term menggunakan metode TF/IDF dan menghitung similarity menggunakan LSA. Semua hasilnya disimpan dalam database, sedemikian sehingga mekanisme temu kembali dokumen sesuai term-term dan dokumen yang mirip dapat dilakukan.

Dari beberapa sampel data, pengukuran similaritas dokumen menggunakan LSA cukup baik walaupun demikian sampel data ini belum mewakili testing yang lengkap.

DAFTAR PUSTAKA

[1] A. Aho, Ravi Setha, J. D. Ullman, Compilers: Principles, Techniques, Tools, Addison-Wesley Company, 1987.

[2] http://snowball.tartarus.org/algorithms/english/stemmer.html, 25 Maret 2009.

[3] Marie-Francine Moens, Automatic Indexing And Abstracting Of Document Texts, Kluwer Academic Publisher, The Kluwer International Series On Information Retrieval, 2002.

[4] http://www.tdan.com/view-articles/4917, 23 Maret 2009.

[5] http://www.articlesbase.com/business-articles/sempo-search-engine-marketing-301029.html, 23 Maret 2009.

[6] Lecture Notes, CS601R, Brigham Young University, Maret 2008.

[7] Rada Mihalcea, Courtney Corley, Carlo Strapparava, Corpus-based and Knowledge-based Measures of Text Semantic Similarity, in Proceedings of the American Association for Artificial Intelligence (AAAI 2006), Boston, July 2006.