Implementasi *Semantic Search* pada *Open Library* menggunakan Metode *Latent Semantic Analysis* (Studi Kasus: *Open Library* Universitas Telkom)

Ivana Azharyani[1], Dana Sulistyo Kusumo[2],

[1,2,]Fakultas Informatika, Universitas Telkom, Bandung
[1]ivanaazharyani@students.telkomuniversity.ac.id, [2]danakusumo@telkomuniversity.ac.id,
[3]pembimbing2@telkomuniversity.ac.id, [4]pembimbingluar@telkom.co.id

Abstrak

Open Library Telkom University mengalami pertumbuhan yang pesat baik dari sisi jumlah maupun kekayaan kontennya. Sebagai konsekuensinya, dibutuhkan metode pencarian yang mampu memberikan hasil yang lebih akurat. Pencarian berbasis metadata sudah memberikan tambahan opsi, namun juga masih memiliki kelemahan tidak dapat menemukan dokumen yang memiliki kemiripan. Kelemahan ini bisa diatasi oleh pencarian semantik dengan memahami maksud dari pencari dan makna kontekstual istilah, seperti yang ditampilkan dalam data pencarian dengan mengkombinasiakan Latent Semantic Analysis (LSA) dan weighted tree similarity. Berdasarkan hasil pengujian, sistem yang dibagun mampu memberikan informasi relevan dengan rata-rata nilai precision dan recall yang baik. Nilai rata-rata precision 57.1181868% dan nilai rata-rata recall 85.0848178%. Hasil nilai rata-rata recall sudah baik karena hampir mendekati 100% artinya tingkat keberhasilan sistem dalam menemukan dokumen yang relevan. Sehingga dapat disimpulkan, metode LSA dan weigted tree mampu memberikan dokumen yang relevan kepada penggunanya serta ketepatan antara kueri masukkan dengan hasil pencarian dokumen.
Kata Kunci: ***latent semantic analysis, weighted tree similarity***

Abstract

The Telkom University Open Library promotes rapid growth in terms of both the amount and wealth of its content. As a consequence, a search method is needed that is able to provide more accurate results. Search based on metadata has provided additional options, but also has the disadvantage of not being able to find documents that have a similarity. This weakness can be overcome by semantic search with the intent and purpose of contextual terms, as referred to in search data by combining Latent Semantic Analysis (LSA) and weighted tree similarity. Based on the test results, the built system is able to provide relevant information with an average value of precision and good memory. The average precision value is 57.1181868% and the average recall value is 85.0848178%. The results of the average value of 100% withdrawal means the level of success in finding relevant documents. Detachable, the LSA method and the weigted tree provide relevant documents for its users as well as the accuracy between the queries provided with the document search results.
Keywords: ***latent semantic analysis, weighted tree similarity***

1. Pendahuluan

1.1 Latar Belakang

Open library Telkom University merupakan perpustakaan yang menyimpan buku, skripsi, jurnal dalam bentuk fisik maupun *online*. *Open Library Telkom University* menyediakan fasilitas website untuk mempermudah pengguna melakukan pencarian data buku. Pencarian pada website ini masih menggunakan pencarian teks penuh berdasarkan judul, kode, pengarang, penerbit, subjek atau jenis yang dimasukkan oleh pengguna. Pencarian teks penuh merupakan metode pencarian yang paling pertama dikembangkan [1]. Pencarian dilakukan dengan mencocokkan setiap kata yang ada pada judul buku ataupun tugas akhir dengan kata yang diberikan oleh pengguna [1]. Sementara itu, Banyak mahasiswa *Telkom University* ingin mencari jurnal tugas akhir bukan hanya memunculkan dokumen sesuai inputan, tetapi juga keterkaitan antara dokumen tersebut dengan dokumen yang lain. Contoh kasus yang dialami adalah ketika pengguna memasukkan kata kunci *"*Sistem Pendukung Keputusan untuk pemilihan Supplier menggunakann metode *elimination et choix traduisant la realite(ELECTRE)",* pada pencarian tersebut output yang dihasilkan sudah sesuai dengan inputan user tetapi, tidak memberikan dokumen yang berkaitan dengan inputan user. Dapat disimpulkan bahwa *Open Library* sudah optimal dalam melakukan proses pencarian tetapi belum menerapkan korelasi antar dokumen. Sehingga pada tugas akhir ini dibuat sistem pencarian untuk mendapatkan informasi yang relevan dalam merepresentasikan hubungan antar kata dan dokumen yang berkaitan. Metode yang ditawarkan dalam proses pencarian untuk mengatasi kekurangan pada sistem *open library* ini adalah pencarian semantik. *Semantic search* adalah pencarian berdasarkan makna kata atau kalimat [1]. pencarian semantik memberikan saran bagi pengguna berdasarkan perhitungan dan penarikan kesimpulan yang dilakukan sistem. Sebuah pemodelan data dibutuhkan untuk mendukung pencarian semantik. Bentuk–bentuk pemodelan data untuk semantic *search* antara lain *weighted tree similarity*. Dalam tugas akhir ini, model dibangun dengan menghitung pencarian kesamaan menggunakan metode *latent semantic analysis* (LSA) dan *weigted tree similarity*. *Latent Semantic Analysis* (LSA) adalah teori dan metode untuk mengekstraksi dan merepresentasikan makna kontekstual dari kata tersebut [2]. Tugas akhir ini menggunakan data *open library telkom university* khusus tugas akhir yang berbahasa Indonesia. Sistem akan merespon dengan memberikan hasil pencarian yang relevan dan keterkaitan antar dokumen dalam bahasa Indonesia.

1.2 Topik dan Batasannya

Topik yang diangkat dalam tugas akhir ini adalah memberikan informasi yang relevan bagi penggunanya dan menampilkan hasil keterkaitan dokumen. Pada tugas akhir ini mengolah abstrak dan judul dalam database menggunakan *preprocessing*, abstrak menggunakan metode LSA. Untuk judul, penulis, editor menggunakan *cosine similarity*. Kemudian untuk menghitung keseluruhan menggunakan *weighted tree.*

Terdapat beberapa hal yang dijadikan batasan pada tugas akhir ini yaitu, data yang digunakan sebanyak 4127 data yang diberikan oleh open library Telkom university. Data tersebut berupa judul, abstrak, penulis, dan editor S1 Informatika Telkom University terbitan tahun 2014-2018 yang berbahasa Indonesia.

1.3 Tujuan

Tujuan dari tugas akhir pada tugas akhir ini adalah sistem pencarian untuk mendapatkan informasi yang relevan dalam merepresentasikan hubungan antar kata dan dokumen yang berkaitan.

1.4 Organisasi Tulisan

Jurnal TA disusun sebagai berikut: Bagian 2 menunjukan tugas akhir-tugas akhir terkait dengan tugas akhir ini. Bagian 3 merupakan arsitektur sistem yang dibangun. Bagian 4 menjelaskan hasil pengujian dan evaluasi sistem. Bagian 5 kesimpulan. Bagian terakhir adalah daftar pustaka yang merupakan daftar dari setiap referensi yang digunakan dalam penyelesaian tugas akhir.

2. Studi Terkait

2.1 Open Library Telkom University

Telkom University Open Library adalah brand untuk Unit Sumber Daya Keilmuan & Perpustakaan (SDK & Perpustakaan) Telkom University yang berada di bawah Wakil Rektor III. Telkom University Open Library memiliki visi "Menjadi *leader* dari pusat ilmu dan pengetahuan berbasis teknologi informasi". Telkom University Open Library memiliki lebih dari 80.500 judul koleksi dengan jumlah eksemplar 123.937 eksemplar, dan telah mendapatkan akreditasi "A" dari Perpustakaan Nasional RI pada tahun 2015.

Dalam pelaksanaan operasionalnya, konsep "*Open Library*" didukung oleh pengembangan teknologi informasi sistem perpustakaan, untuk senantiasa meningkatkan layanan, database, dan koleksi perpustakaan. *Telkom University* juga ingin perpustakaannya dapat bermanfaat sebesar mungkin dan sebanyak mungkin bagi sivitas dan warga *Telkom University* serta masyarakat luas. Oleh karena itu, *Telkom University Open Library* menggagas juga kegiatan-kegiatan untuk menggerakkan literasi seperti *"Telkom University Literacy Event*", *"Library Open Discussion",* penggalangan donasi buku, program perpustakaan binaan di beberapa daerah terpencil, serta mengedukasi anti-plagiarisme [2].

2.2 Preprocessing

Pemrosesan / pengolahan dokumen yang dilakukan mencakup pada proses *casefolding, cleanening, stopword, stemming* [3].

a. Case folding

Dalam *text preprocessing* proses *case folding* bertujuan untuk mengubah semua huruf dalam sebuah dokumen teks menjadi huruf kecil (*lowercase*).

b. Cleaning

Dalam *text preprocessing* proses *cleaning* bertujuan untuk dilakukan pembersihan data dengan cara menangani noisy (data yang masih mengandung error dan outliers) serta redudansi data yang ada pada data mentah.

c. Stopword

Dalam *text preprocessing* proses stopword bertujuan untuk dilakukan proses penghilangan kata-kata umum yang biasanya muncul namun tidak memiliki makna atau akan mengambil kata-kata yang dianggap penting.

d. Stemming

Dalam *text preprocessing* proses stemming pada tugas akhir ini menggunakan library Sastrawi Stemming, library stemming bahasa Indonesia yang berbasis Python yang berbasis algoritma Nazief dan Adriani.

2.3 Latent Semantic Analysis

Latent Semantic Analysis (LSA) adalah teori dan metode untuk mengekstraksi dan merepresentasikan makna kontekstual dari kata tersebut. Ini dilakukan melalui perhitungan statistik yang diterapkan pada korpus dokumen teks[2]. Ide dasarnya adalah bahwa konteks keseluruhan dari kata yang diberikan tidak muncul. Ini adalah hal yang sangat penting dalam arti umum kata-kata dan satu set kata-kata satu sama lain [4]. Singkatnya, LSA adalah metode di mana dekomposisi nilai singular digunakan untuk membentuk suatu generalisasi dari semantik tekstual. LSA menggunakan fakta bahwa kata-kata tertentu muncul dalam konteks yang sama untuk membangun hubungan antara makna kata[5]. Ini memungkinkan bagian-bagian teks untuk dibandingkan satu sama lain daripada membandingkan kata-kata itu secara langsung. Kata-kata yang tidak pernah muncul bersama dapat dibandingkan dengan makna [5]. Dalam tugas akhir ini, hubungan semantik dinilai dari kemunculan kata-kata pada sebuah dokumen dan pada query.

Cara Menghitung LSA, langkah pertama yang harus dilakukan adalah merepresentasikan teks ke dalam matriks *term-document* (TDM). Dalam hal ini, setiap baris mewakili istilah unik dan setiap kolom mewakili dokumen. Kemudian sel-sel matriks diisi dengan frekuensi kemunculan istilah untuk setiap dokumen. Kemudian, TDM akan diproses menggunakan teknik *Single Value Decomposition* (SVD) [6]. SVD didasarkan pada teorema dalam aljabar linier yang menyatakan bahwa matriks persegi dapat dipecah menjadi perkalian tiga matriks: matriks ortogonal U, matriks S diagonal, dan matriks transpose matriks V. Teorema dinyatakan sebagai berikut:

$$A_{mn} = U_{mm} \cdot S_{mn} \cdot V_{nn}^{T} \quad (1)$$

Dimana $U^T U$ = I; $V^T V$= I; kolom U merupakan eigenvektor ortonormal dari AA^T , kolom V merupakan eigen vektor ortonormal dari AA^T , dan S merupakan matriks diagonal yang berisi akar nilai eigen dari U atau V dalam urutan bawah [7].

Matriks S dihitung melalui prosedur-prosedur seperti berikut [8]:

1. A^T dan $A.A^T$ dihitung
2. Mencari eigenvalue dari hasil perkalian matriks dan mengurutkan nilai dari yang terbesar ke yang terkecil. Hasil atau nilai dari matriks S adalah nonnegatif matriks, yang merupakan akar dari nilai pengurutan dan disebut dengan nilai singular dari A.
3. S dibangun dengan menempatkan nilai yang singular dimana nilai diurutkan dari yang terbesar ke yang terkecil pada setiap diagonalnya. Dari persamaan 1 dapat dibentuk:

$$A.V = U.S.V^T.V = U.S \quad (2)$$
$$U^T A.V = S$$

Dimana, A = matriks A yang dibangun dari TDM pembobotan ternormalisasi pada corpus, V = matriks V hasil dekomposisi SVD matriks A, S = matriks singular hasil dekomposisi SVD matriks A, V^T = matriks V transpose, U = matriks U hasil dekomposisi SVD matriks A,U^T = matriks U transpose hasil dekomposisi SVD matriks A.

4. U dan V adalah ortoghonal, dimana matriks orthogonal merupakan sebuah matriks yang jika dikalikan dengan transposenya akan menghasilkan matriks identitas. Misalkan matriks M adalah orthogonal maka dapat ditulis bahwa $MM^T = M^T M = I$ =1. Perkalian M dan M^T bersifat komutatif [8]. Sehingga berdasarkan prosedur yang telah dijelaskan sebelumnya, nilai eigenvalue dapat dicari melalui persamaan berikut:

$$|A^T . A = cI| = 0 \quad (3)$$

Dimana, $|A^T . A = cI|$ merupakan determinan nilai $A^T . A - cI$, A merupakan matriks A: dari TDM pembobotan corpus, A^T merupakan matriks A t*ranspose* dari TDM pembobotan *corpus*, c merupakan vaiable nilai eigen, I merupakan matriks indentitas.

5. Nilai c merupakan nilai eigen yang akan dihasilkan oleh persamaan tersebut. Akar dari nilai *eigen* disebut dengan nilai singular. Nilai *singular* tersebut disusun berurutan dari nilai yang terbesar ke yang terkecil dan pada akhirnya membentuk matriks diagonal yang disebut dengan matriks *S*. Dari nilai *eigen* akan didapatkan *eigenvector* untuk membentuk matriks *V*, sesuai dengan persamaan berikut:

$$(A^T.A - cI)x = 0 \quad (4)$$

Dimana, *A* merupakan matriks *A*: dari TDM pembobotan corpus, A^T merupakan matriks *A* transpose dari TDM pembobotan *corpus*, *c* merupakan merupakan variable nilai *eigen*, *I* merupakan matriks indentitas, *x* merupakan variable *x*.

6. Untuk mencari nilai dari matriks *U* maka digunakan persamaan berikut:

$$U = A.V.S^{-1} \quad (5)$$

Dimana, *U* merupakan hasil dari perkalian, *A* merupakan matriks *A*: dari TDM pembobotan *corpus*, *V* merupakan matriks *V* hasil dari nilai *eigenvector*, S^{-1} merupakan *inverse* dari matriks singular.

7. Proses dekomposisi dari reduksi dimensi SVD kemudian dapat digunakan untuk pengembangan sistem temu kembali. Untuk menghitung *query* vektor dari SVD maka dapat dihitung dengan menggunakan persamaan berikut:

$$q' = q^T.U_k.S_k^{-1} \quad (6)$$

Dimana, q' merupakan *query vector* representasi dari LSA, q^T merupakan *transpose* TDM dari pembobotan ternormalisasi TF-IDF query, U_k merupakan reduksi dimensi k dari matriks *U*, S_k^{-1} merupakan *inverse* dari reduksi dimensi *k* matriks *S*.

2.3.1 Pengindekan dengan LSA

Proses pengindekan dengan LSA dilakukan dalam beberapa tahap sebagai berikut [8]:

Tahap 1. Gunakan algoritma LSA untuk mendapatkan matrik A' dan B' representasi masing-masing document latih dan dokumen uji tereduasi k-rank.

Tahap 2. Hitung cosine similaritas antar dokumen latih dan dokumen uji hasil reduksi berdasarkan matrik tereduksi A' dan B', dalam hal ini X merepresentasikan dokumen uji dan dj representasi dokumen latih ke j.

$$sim(X, d_j) = \frac{\sum_{j=1}^{m} X_{j.d_{ij}}}{\sqrt{\left[\sum_{j=1}^{m} X_j\right]^2}.\sqrt{\left[\sum_{j=1}^{m} X_j\right]^2}} \quad (7)$$

2.4 Pembobotan Term Frequency-Invers Document Frequency

Tf-Idf yaitu perhitungan yang menggambarkan seberapa pentingnya kata (term) dalam sebuah dokumen. Proses ini digunakan untuk menilai bobot relevansi term dari sebuah dokumen terhadap seluruh dokumen. Term frequency adalah ukuran seringnya kemunculan sebuah term dalam sebuah dokumen. IDF merupakan banyaknya istilah tertentu dalam keseluruhan dokumen, dapat dihitung dengan persamaan (8) :[9]

$$idf_j = \log \frac{n}{n^j} \quad (8)$$

Dimana n merupakan jumlah dokumen yang di gunakan, nj merupakan hasil df (document frequency), log digunakan untuk memperkecil pengaruhnya relative terhadap tf. Bobot dari term dihitung menggunakan ukuran $tf - idf$ dalam persamaan (9) :

$$w = tf \times idf \quad (9)$$

Dimana tf merupakan kemunculan term dari setiap dokumen, dan w merupakan bobot dokumen terhadap kata atau bobot dari key terhadap dokumen.

2.5 Cosine Similarity

Secara umum, fungsi similarity adalah fungsi yang menerima dua buah objek dan mengembalikan nilai kemiripan (*similarity*) antara kedua objek tersebut berupa bilangan riil. Umumnya, nilai yang dihasilkan oleh ungsi similarity berkisar pada interval [0...1]. Namun ada juga beberapa fungsi *similarity* yang menghasilkan nilai yang berada di luar interval tersebut. Untuk memetakan hasil fungsi tersebut pada interval [0...1] dapat dilakukan normalisasi [10].

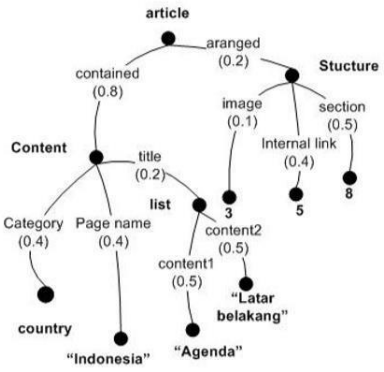
Cosine similarity adalah perhitungan kesamaan antara dua vector n dimensi dengan mencari kosinus dari sudut diantara keduanya dan sering digunakan untuk membandingkan dokumen dalam text mining [11]. Metode cosine similarity ini menghitung similarity antara dua buah objek (misalkan D1 dan D2) yang dinyatakan dalam dua buah vector dengan menggunakan keywords (kata kunci) dari sebuah dokumen sebagai ukuran. [12].

$$CosSim(d_i, q_i) = \frac{q_{i.d_i}}{|q_i||d_i|} = \frac{\sum_{j=1}^{t}(q_{ij}.d_{ij})}{\sqrt{\sum_{j=1}^{t}(q_{ij})^2.\sum_{j=1}^{t}(d_{ij})^2}} \quad (10)$$

Dimana, q_{ij} *merupakan bobot istilah j pada dokumen* $i = tf_{ij}$. idf_j,
d_{ij} *merupakan bobot istilah j pada dokumen* $i = tf_{ij}.idf_j$

Pang-Ning Tan, menjelaskan bahwa semakin besar hasil fungsi similarity, maka kedua objek yang dievaluasi diannggap semakin mirip [10]. Jika sebaliknya, maka semakin kecil hasil fungsi similarity, maka kedua objek tersebut dianggap semakin berbeda [10]. Pada fungsi yang menghasilkan nilai pada jangkauan [0…1], nilai 1 melambangkan kedua objek persis sama, sedangkan nilai 0 melambangkan kedua objek sama sekali berbeda [10].

2.6 Weighted Tree Similarity



Gambar 2.1 Contoh representasi tree

Dokumen yang akan dihitung kemiripannya direpresentasikan dalam sebuah *tree* yang memiliki karakteristik node berlabel, cabang berlabel dan cabang berbobot. Contoh representasi *tree* bisa dilihat pada Gambar 1. Gambar 1 menggambarkan representasi query pengguna terhadap sebuah artikel Wikipedia [13]. Keunikan dari *weighted tree* ini adalah cabang yang berlabel dan berbobot. Pada contoh Gambar 2 pengguna lebih menekankan pencari untuk menemukan ketepatan pencarian berdasarkan cabang content(berbobot 0,8) dibandingkan strukturnya (berbobot 0,2). Penentuan tingkat kepentingan cabang ini terdapat dalam representasi *weighted tree*.

Representasi *tree* dalam suatu *weighted tree* mengikuti aturan sebagai berikut [14]:

1. Nodenya berlabel
2. Cabang berlabel
3. Cabang berbobot
4. Label dan bobot ternormalkan. Label terurut sesuai abjad dari kiri ke kanan. Jumlah bobot dalam cabang setingkat subtree yang sama adalah 1.

Untuk merepresentasikan tree tersebut digunakan Rule-ML versi Object Oriented Modelling, yang disebut Weighted Object Oriented RuleML (WOORuleML) yang mengacu pada standarisasi XML.Contohnya bisa dilihat pada gambar 3.

```
<cterm>
<_ope><ctor>Article</ctor><_ope>
<_rn="Content" w="0.8">
        <cterm>
        <_opc><ctor>Content</ctor></opc>
        <_rn="category" w="0.4"><ind>Indonesia</ind></_r>
        <_rn="page name" w="0.4"><ind>Indonesia</ind></_r>
        <_rn="title" w="0.2">
                <cterm>
                <_ope><ctor>List</ctor></_ope>
                 <_rn="content1" w="0.5"><ind>"Agenda"</ind></_r>
                 <_rn="content2" w="0.5"><ind>"Latar Belakang"</ind></_r>
                </cterm>
                </_r>
        </cterm>
</r>
<_rn="Structure" w=0.2">
        <cterm>
        <_ope><ctor>Structure</ctor></_ope>
         <_rn="image" w="0,1"><ind>3</ind</_r>
         <_rn="intenallink" w="0.4"><ind>5</ind></_r>
         <_rn="section" w="0.5"><ind>8</ind></_r>
        </cteerm>
</r>
</cterm>
```

Gambar 2.2 Representasi tree dalam WOORuleML

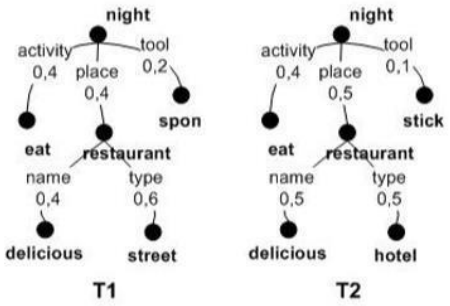
Pada Gambar3 terdapat beberapa symbol adapun keterangan dari simbol-simbol ini antara lain:

1. <cterm> = keseluruhan tree
2. <opc> = root dari tree
3. <ctor> = node label dari root
4. <_r> = role dari setiap arch/edge dan memiliki beberapa atribut yaitu n mewakili label dan w yang mewakili bobot/weight
5. <ind> = label untuk role

Subtree dari sebuah role memiliki struktur yang sama atau indentik yang diawali dengan<cterm>dan seterusnya seperti pada Gambar 3.

2.7 Perhitungan Kemiripan

Algoritma penghitungan kemiripan antara dua *weighted tree* ini terdapat di dalam makalah [14]. Gambar 3 menunjukkan contoh dua buah tree T1 dan T2 yang dihitung kemiripannya. Nilai kemiripan tiap pasangan subtree berada diantara interval [0,1]. Nilai 0 bermakna berbeda sama sekali sedangkan 1 bermakna identik. Kedalaman (depth) dan lebar (breadth) tree tidak dibatasi. Algoritma penghitung kemiripan tree secara rekursif menjelajahi tiap pasang tree dari atas kebawah mulai dari kiri kekanan. Algoritma mulai menghitung kemiripan dari bawah keatas ketika mencapai *leaf node*. Nilai kemiripan tiap pasang subtree di level atas dihitung berdasar kepada kemiripan subtree di level bawahnya.



Gambar 2.3 Contoh kemiripan penghitungan dasar

Sewaktu penghitungan, kontribusi bobot cabang juga diperhitungkan. Bobot dirata-rata menggunakan rata-rata aritmatika (w_i+w_i)/2 Nilai rata-rata bobot sebuah cabang dikalikan dengan kemiripan S_i yang diperoleh secara rekursif. Nilai S_i pertama diperoleh berdasar kemiripan *leaf node* dan dapat diatur nilainya menggunakan fungsi $A(S_i)$. Pada awalnya algoritma *weighted tree similarity* hanya memberi nilai 1 bila *leaf node*-nya sama dan 0 bila berbeda [14]. Perumusan penghitungan kemiripan tree ini terdapat di dalam persamaan berikut:

$$\sum(A(S_i)(w_{i+}w_i)/2) \quad (11)$$

dengan $A(S_i)$ adalah nilai kemiripan *leaf node*, dan adalah bobot pasangan *arc weighted tree*. Penilaian $A(S_i)$ analog dengan logika *AND* sedangkan penilaian bobot pasangan analog dengan logika *OR*.Di dalam contoh pada Gambar 4 perilaku algoritma dapat dijelaskan sebagai berikut. Pada awalnya dihitung kemiripan *node* cabang *activity* yang diperoleh 1. Kemiripan ini dikalikan rata-rata bobot cabang activity (0,4+0,4)/2 menghasilkan kemiripan cabang. Algoritma kemudian mencari kemiripan *node* cabang berikutnya, place. Karena *node* ini bukan *leaf* maka algoritma akumulasi kemiripan dengan cabang lain yang setingkat menghasilkan kemiripan total.

2.8 Evaluasi performasi

Confussion Matrix

Confusion Matrix adalah sebuah metode yang biasa digunakan untuk perhitungan akurasi. Dalam pengujian keakuratan hasil pencarian akan dievaluasi nilai recall, precision. Dimana precision mengevaluasi kemampuan sistem untuk menemukan peringkat yang paling relevan, dan didefinisikan sebagai presentase dokumen yang diretrieve dan benar benar relevan terhadap query. Recall mengevaluasi kemampuan sistem untuk menemukan semua item yang relevan dari koleksi dokumen dan didefinisikan sebagai presentase dokumen yang relevan terhadap query [15].

Metode ini memiliki keluaran yaitu:

a. Recall

Recall adalah perhitungan yang berguna untuk mengetahui seberapa besar presentase kasus true yang diidentifikasi true. Persamaan untuk menghitung *recall* adalah :

$$Recall = \frac{fp}{tn+fp} \quad (12)$$

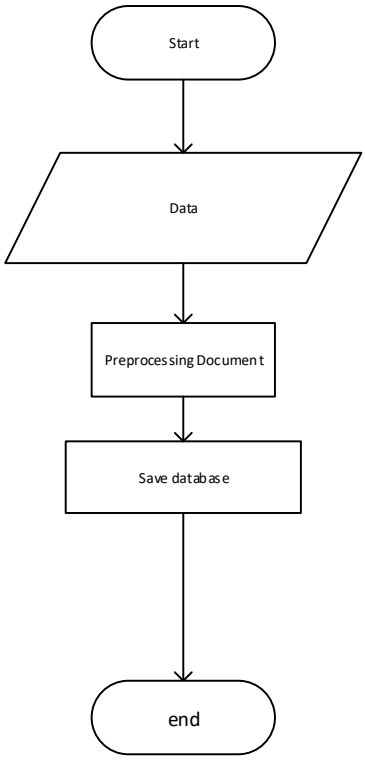
b. Precision

Precision adalah perbandingan antara kasus true yang dididentifikasi true dengan semua hasil prediksi berniilai positif. Persamaan untuk menghitung *precision* adalah :

$$Precission = \frac{fp}{fn+fp} \quad (13)$$

Dimana, tp merupakan hasil prediksi positif dan data sebenarnya positif, fn merupakan hasil prediksi negatif dan data sebenarnya positif, tn merupakan hasil prediksi positif dan data sebenarnya negative, fp merupakan hasil prediksi negatif dan data sebenarnya negatif.

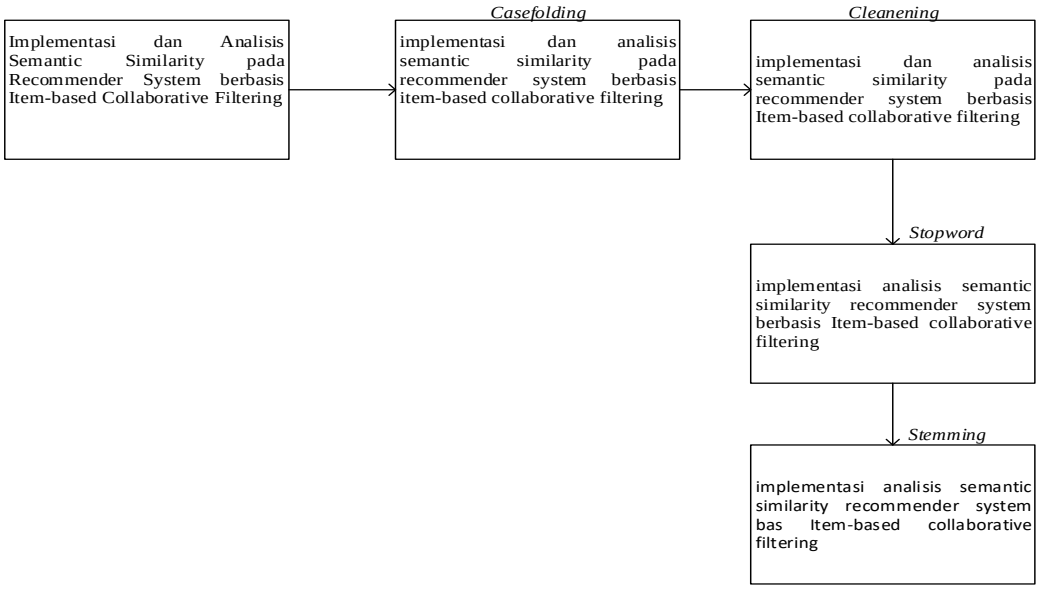
3. Sistem yang Dibangun



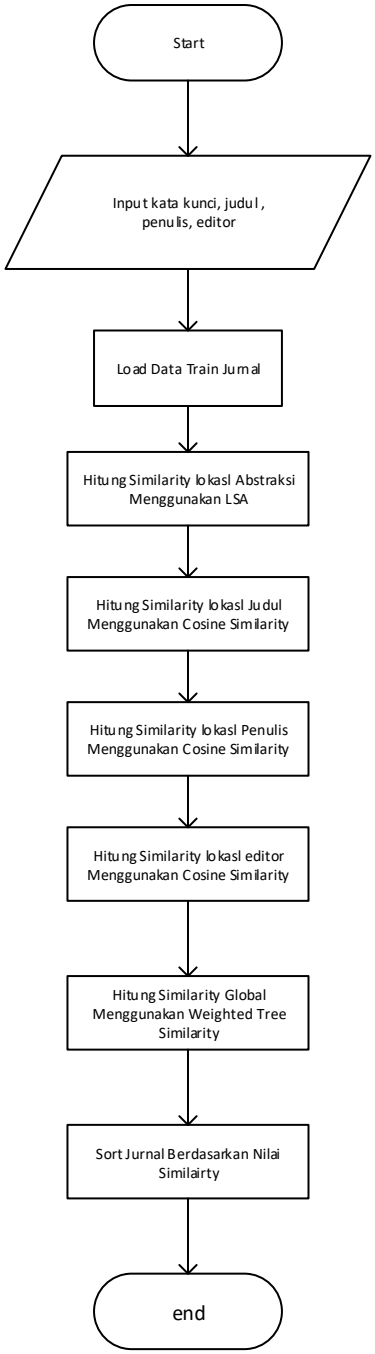
Gambar 3.1 Pembuatan data train

3.1 Preprocessing Document

Data yang di gunakan abstrak, judul, penulis, editor. Format dokumen file yang digunakan berupa csv. Pada tahan preprocessing dokumen ini adalah case folding, cleaning, stemming, dan stopword removal, preprocessing dilakukan pada dokumen judul dan abstrak. Proses case folding dimana semua huruf diubah menjadi huruf kecil. Proses cleaning dimana proses membersihkan dokumen dari komponen-komponen yang tidak memiliki hubungan dengan informasi yang ada pada dokumen, seperti tag html, link, dan script, dan sebagainnya. Proses stopword removal merupakan kata-kata yang tidak deskriptif (tidak *penting*) yang dapat dibuang, contoh stopwords adalah untuk, sang, sudah, adalah, dan seterusnya . Proses stemming merupakan tahap mengubah kata menjadi bentuk kata dasar. Contoh preprocessing ditunjukan pada Gambar 6.



Gambar 3.2 Contoh Preprocessing Dokumen



Gambar 3.3 Proses pencarian

Data yang diinput kata kunci atau abstrak, judul, penulis, editor. Setelah itu input data. Hitung similarity abstrak menggunakan LSA, hitung similarity judul, penulis, editor menggunakan cosine similarity. Hitung similarity global menggunakan weigted tree similarity. Sort nilai similarity, nilai similaritas tertinggi ditempatkan paling atas.

3.2 Hitung Similarity Abstraksi Menggunakan LSA

Abstraksi yang telah di preprocessing, selanjutnya proses pengindeksan untuk membuat matriks term-document(TDM) bertujuan untuk menghasilkan istilah signifikan dari setiap dokumen. Matriks dokumen yang terdiri dari baris mewakili semua dokumen yang telah berhasil diindeks,kolom mewakili dokumen. merepresentasikan teks ke dalam matriks *term-document* (TDM). Dalam hal ini, setiap baris mewakili istilah unik dan setiap kolom mewakili dokumen. Kemudian sel-sel matriks diisi dengan frekuensi kemunculan istilah untuk setiap dokumen. Kemudian,

TDM akan diproses menggunakan teknik *Single Value Decomposition* (SVD). SVD didasarkan pada teorema dalam aljabar linear yang menyatakan bahwa matriks persegi dapat dipecah menjadi perkalian tiga matriks: matriks ortogal U, matriks diagonal S diagonal, dan matriks orthogonal transpose dari matriks V. selanjutnya, SVD akan menghasilkan tiga matriks yaitu matriks skala, matriks vector jangka dan matriks vector doc . Setelah mendapatkan SVD, akan dilakukan tahapan berikutnya yaitu untuk pengurangan ukuran dimensi, sebanyak k. nilai k=2[8]. Selanjutnya,menghitung koefisien similarity menggunakan cosinus antara vector query dengan tiap vektor dokumen untuk mendapatkan nilai dokumen sebagai nilai kesamaan.Selanjutnya menghitung koefisien similarity menggunakan cosinus antara query dengan tiap dokumen.

3.3 Hitung Similarity Judul Menggunakan Cosine Similarity

Judul yang telah di preprocessing, selanjutnya dihitung pembobotan term frequency-invers document frequency seperti yang ditunjukan pada Tabel 1. Selanjutnya hitung kemiripan vector [dokumen] query Q dengan setiap dokumen yang ada. Kemiripan antar dokumen dapat menggunakan cosine similarity untuk mendapatkan nilai kesamaan lokal.

Tabel 3.1 Contoh Pembobotan TF/IDF Judul

		tf				idf	Wdt=tf.idf			
Term	Q	D1	D2	D3	df	log(n/df)+1	Q	D1	D2	D3
fuzzy	1	2	0	1	3	1.12493874	1.12493874	2.24987748	0	1.12493874
inference	1	1	0	0	2	1.30103	1.30103	1.30103	0	0
sistem	1	1	1	1	4	1	1	1	1	1

3.4 Hitung Similarity Penulis Menggunakan Cosine Similarity

Pada bagian ini penulis dihitung pembobotan term frequency-invers document frequency seperti yang ditunjukan pada Tabel 2. Selanjutnya hitung kemiripan vector [dokumen] query Q dengan setiap dokumen yang ada. Kemiripan antar dokumen dapat menggunakan cosine similarity untuk mendapatkan nilai kesamaan lokal.

Tabel 3.2 Contoh Pembobotan TF/IDF Penulis

		Tf				Idf	Wdt=tf.idf			
Term	Q	D1	D2	D3	df	log(n/df)	Q	D1	D2	D3
Hendrik	1	1	0	0	2	1.30103	1.30103	1.30103	0	0
Puasa	1	1	0	0	2	1.30103	1.30103	1.30103	0	0

3.5 Hitung Similarity Editor Menggunakan Cosine Similarity

Pada bagian ini penulis dihitung pembobotan term frequency-invers document frequency seperti yang ditunjukan pada Tabel 3. Selanjutnya hitung kemiripan *vecto*r [dokumen] *query* Q dengan setiap dokumen yang ada. Kemiripan antar dokumen dapat menggunakan cosine similarity untuk mrnghitung nilai kesamaan lokal.

Tabel 3.3 Contoh Pembobotan TF/IDF Editor

		tf				Idf	Wdt=tf.idf			
Term	Q	D1	D2	D3	df	log(n/df)	Q	D1	D2	D3
Ririn	1	1	1	0	3	1.12493874	1.12493874	1.12493874	1.12493874	0
Dwi	1	1	2	0	3	1.12493874	1.12493874	1.12493874	2.24987748	0

3.6 Hitung Similarity Global

Untuk menghitung Similarity global menggunakan algoritma weigted tree similarity. Bandingkan setiap tree document dengan tree query untuk mendapatkan nilai kesamaan setiap dokumen. Bobot yang digunakan untuk menghitung masing-masing similarity. Bobot untuk query dan dokumen pada abstrak 0,4. Bobot untuk query dan dokumen pada judul 0,3. Bobot untuk query dan dokumen pada penulis 0,2. Bobot untuk query dan dokumen pada editor 0,1 [14]. Seteleh itu hasil similarity abstrak,judul,penulis,author dijumlahkan, kemudian dari hasil tersebut,dibagi 4 dan di dapatkan hasil perhitungan kesamaan dokumen. Setelah itu diurutkan nilai similaritas, dokumen dengan nilai similaritas tertinggi ditempatkan paling atas, yang berarti dokumen memiliki relevansi terbesar terhadap query.

4. Evaluasi

4.1 Hasil Pengujian

Pengujian dilakukan dengan cara menghitung nilai rata-rata *precision* dan rata-rata *recall* dari lima query uji dengan menggunakan metode L*aten Semantic Analysis.* Pengalaman Nielsen mengindikasikan bahwa tiga sampai lima evaluator biasanya dapat mengidentifikasi sekitar 75 persen dari masalah kegunaan dari desain tertentu [16]. Oleh sebab itu tugas akhir ini menggunakan lima query. *Precision* digunakan sebagai alat untuk mengukur ketepatan

pencarian pada sistem dan *recall* digunakan untuk mengukur kualitas seberapa lengkap hasil relevan yang ditampilkan oleh sistem. Hasil pengujian sistem ditunjukan pada Tabel 4.1.

Tabel 4.1 Precision dan Recall dari lima *query* uji

		koleksi	hasil pencarian			performa	
query	waktu pemrosesan	relevan |R|	Total |A|	Relevan |Ra|	Tidak relevan	*precision* |Ra|/|A|	*recall* |Ra|/|R|
search	13. 710873365402222	58	125	57	68	0,456	0,98275862
fuzzy	13. 554869890213000	195	228	108	120	0,47368421	0,55384615
k-nearest neighbor	13. 191831827163600	41	70	31	39	0,44285714	0,75609756
latent semantic	12. 580224752426100	21	26	21	5	0,80769231	1
ontology	12. 993074178695600	26	37	25	12	0,67567568	0,96153856
					Rata rata	0,571181868	0,850848178

4.2 Analisis Hasil Pengujian

Pada tabel 4.1 menunjukan hasil pengujian precision dan recall, dimana analisis dilakukan dengan memilih lima query yang dimasukkan oleh pengguna. Hasil rata-rata precision dari lima query uji adalah 57.1181868% dan rata-rata recall 85.0848178%. rata-rata precision dari pengujian tersebut rendah dalam mengukur ketepatan pencarian pada sistem. Berdasarkan waktu pemrosesan query "latent semantic" memiliki waktu pemrosesan yang lebih cepat yaitu 12. 580224752426100 detik dan query "search" memiliki waktu pemroresan yang lama yaitu 13. 710873365402222 detik, menunjukkan bahwa dalam proses mengukur ketepatan dan mengukur kualitas hasil relevan sistem tidak membutuhkan waktu yang lama.

Dilihat dari koleksi relevan menunjukkan bahwa koleksi relevan yang tertinggi dimiliki oleh query "fuzzy" dengan 195 data dan query "latent semantic" memiliki koleksi relevan terendah yaitu 21 data. Hasil pencarian menunjukkan total pencarian tertinggi terdapat pada query "fuzzy" yaitu 228 data dan terendah adalah query "latent semantic" yaitu 26 data, dimana menunjukkan similarity kata tersebut memiliki relevansi besar terhadap query masukan. Selain itu pencarian relevan query "fuzzy" memiliki hasil yang tinggi yaitu 108 data. Untuk pencarian kata yang tidak relevan query "search" dan query "ontology" memiliki hasil yang baik yaitu 1, hal tersebut menunjukkan bahwa query tersebut memiliki sedikit makna dari penggunaan query yang tidak sesuai dengan query masukan. Sehingga LSA dan weighted tree baik digunakan untuk pencarian informasi yang relevan pada data Open Library.

5. Kesimpulan

Dalam tugas akhir yang telah dilakukan untuk mendapatkan informasi yang relevan dalam mempresentasikan antar kata dan dokumen dengan mengunakan penggabungan LSA dan *weighted tree* didapatkan rata-rata *precision* 57.1181868% dan rata-rata *recall* 85.0848178%. Sehingga dapat disimpulkan bahwa pengguna system pencarian menggunakan penggabungan LSA dan *weighted tree* dapat digunakan untuk mendapatkan informasi yang relevan dalam merepresentasikan hubungan antar kata dan dokumen yang berkaitan, namun rendah dalam menentukan dalam mengukur ketepatan pencarian.

Daftar Pustaka

[1] Y. A. R. F. Riyanarto Sarno, Semantic Search: Pencarian Berdasarkan Konten, Yogyakarta, 2012.

[2] R. S. U. L. Y. Umi Sa'adah, "LATENT SEMANTIC ANALYSIS AND WEIGHTED TREE SIMILARITY FOR SEMANTIC SEARCH IN DIGITAL LIBRARY," 2012.

[3] "Open Library Telkom University," [Online]. Available: https://openlibrary.telkomuniversity.ac.id/home/aboutus.html. [Accessed 5 agustus 2019].

[4] I. Monica, "Analisis Sentimen level Aspek pada Ulasan Produk," *Universitas Telkom*, 2017.

[5] P. W. F. D. L. Thomas K Landauer, "An Introduction to Latent Semantic Analysis," *researchgate.net*, pp. 259-284, 2014..

[6] G. Gorrell, "Latent Semantic Analysis: How does it work, and what is it good for?," January 2007. [Online]. Available: https://staffwww.dcs.shef.ac.uk/people/G.Gorrell/lsa_tutorial.htm.

[7] D. S. E. Y. U. L. Y. D. S. Diana Purwitasari, "SISTEM PEMBANGKIT ANOTASI PADA ARTIKEL," *JUTI*, vol. 9, pp. 21-28, 2011.

[8] K. Baker, "Singular Value Decomposition Tutorial," *Researchgate*, 2014.

[9] D. E. Garcia, "Latent Semantic Indexing (LSI) A Fast Track Tutorial," 2006.

[10] M. Fitri, "PERANCANGAN SISTEM TEMU BALIK INFORMASI DENGAN METODE PEMBOBOTAN KOMBINASI TF-IDF UNTUK PENCARIAN DOKUMEN BERBAHASA INDONESIA," *Justin*, vol. 1, 2013.

[11] A. Karhendana, "PEMANFAATAN DOCUMENT CLUSTERING PADA AGREGATOR BERITA," *Semantic Scholar*, 2008.

[12] L. Zhiqiang, S. Werimin and Y. Zhenhua, "Measuring Semantic Similarity between Words Using Wikipedia," *IEEE*, 2009.

[13] N. A. S. E. Gede Aditra Pradnyana, "PERANCANGAN DAN IMPLEMENTASI AUTOMATED DOCUMENT INTEGRATION DENGAN MENGGUNAKAN ALGORITMA COMPLETE LINKAGE AGGLOMERATIVE HIERARCHICAL CLUSTERING," *Jurnal Ilmu Komputer*, vol. 5, 2012.

[14] F. R. Riyanarto Sarno, "Penerapan Algoritma Weighted Tree Similarity untuk Pencarian Semantik," *researchgate*, 2008.

[15] S. Ferdinandus, "Sistem Information Retrieval Layanan Kesehatan Untuk Berobat dengan Metode Vector Space Model berbasis WebGis. Jurnal. Teknik Informatika STIKI Malang.," 2015.

[16] T. WIJAYA, "PERANCANGAN ALAT UKUR INDEKS USABILITAS PADA MESIN PENCARI (SEARCH ENGINE)," 2011