Penerapan *Singular Value Decomposition* dan *Cosine Similarity* untuk Akurasi Mesin Pencari Teks

Anella Utari Gunadi 13523078[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523078@std.stei.itb.ac.id, [2]anellautari@gmail.com

Abstrak—Pada era digital, pencarian dokumen berbasis teks menjadi kebutuhan penting bagi pelajar, peneliti, dan profesional. Teknologi mesin pencari teks memudahkan pencarian dokumen yang relevan dilakukan secara efisien dengan bantuan metode matematika dan algoritma yang canggih. Dalam makalah ini, diimplementasikan metode *Singular Value Decomposition* (SVD) dan *cosine similarity* untuk meningkatkan relevansi hasil pencarian dokumen berbasis teks. Hasil pengujian menunjukkan bahwa metode ini mampu menghasilkan pencarian yang lebih akurat dengan menganalisis hubungan antar kata secara laten. Dengan pendekatan ini, *query* "Teknik Informatika di Ganesha" menghasilkan nilai *cosine similarity* tertinggi pada dokumen yang memiliki relevansi paling besar, yaitu dokumen D3 yang berisi "Jurusan Teknik Informatika berada di kampus Ganesha dan Jatinangor" dengan nilai 0.892. Penerapan metode ini menunjukkan potensi besar dalam membuat dan mengembangkan mesin pencari teks yang efektif dan efisien untuk berbagai kebutuhan.

Kata kunci—*Singular Value Decomposition, cosine similarity,* laten.

Abstract— In the digital era, text-based document retrieval has become essential for students, researchers, and professionals. Text search engine technology simplifies the process of finding relevant documents efficiently using advanced mathematical methods and algorithms. In this paper, the Singular Value Decomposition (SVD) and cosine similarity methods are implemented to enhance the relevance of text-based document search results. Testing results show that these methods effectively produce more accurate searches by analyzing latent relationships between words. With this approach, the query "Teknik Informatika di Ganesha" yielded the highest cosine similarity value with the document most relevant to it, namely document D3, which contains "Jurusan Teknik Informatika berada di kampus Ganesha dan Jatinangor," with a value of 0.892. This implementation demonstrates significant potential for creating and developing effective and efficient text search engines for various needs.

Keywords—Singular Value Decomposition, cosine similarity, latent.

I. PENDAHULUAN

Pada era digital ini, kebutuhan akan mesin pencari teks semakin meningkat. Pengguna dari berbagai kalangan, mulai dari siswa, mahasiswa, hingga profesional membutuhkan mesin pencari teks untuk mencari suatu dokumen tertentu dalam konteks yang spesifik. Terkadang, untuk mencari sebuah dokumen dalam konteks yang diinginkan terasa sangat sulit. Semakin bertambahnya jumlah dokumen seperti jurnal, artikel, buku, dan sumber informasi lainnya menyebabkan semakin sulit pencarian suatu dokumen dengan konteks tertentu.

Dengan adanya mesin pencari teks, pencarian dokumen yang relevan menjadi lebih mudah. Hanya dengan mengetikkan kata kunci sesuai konteks yang diinginkan, pengguna dapat memperoleh berbagai dokumen yang cocok dan relevan dengan kebutuhan mereka. Teknologi ini dapat memudahkan pelajar, peneliti, pekerja profesional, maupun berbagai kalangan lainnya dalam mencari suatu dokumen sesuai kebutuhan mereka secara efisien.

Teknologi mesin pencari teks telah banyak digunakan dalam berbagai aplikasi, seperti Google, platform *e-commerce*, dan perpustakaan digital. Berbagai metode telah dikembangkan untuk meningkatkan akurasi mesin pencari teks, di antaranya *Boolean Retrieval Model*, *Latent Dirichlet Allocation*, *Fuzzy Matching*, serta pendekatan *Singular Value Decomposition* dan *cosine similarity*. Dalam makalah ini, akan dibahas dan diimplementasikan penerapan *Singular Value Decomposition* dan *cosine similarity* sebagai metode utama untuk meningkatkan akurasi pencarian dokumen berbasis teks.

Implementasi mesin pencari teks juga memiliki berbagai tantangan, salah satunya adalah keambiguan kata kunci. Keambiguan ini sering kali menyebabkan hasil pencarian yang kurang relevan. Dengan menerapkan metode *Singular Value Decomposition* dan *cosine similarity*, masalah tersebut dapat diminimalkan karena metode ini mampu mengenali hubungan atau keterkaitan antar kata. Tantangan lainnya adalah pencarian dokumen dengan konteks yang kompleks. Melalui penerapan dua metode tersebut, mesin pencari teks dapat bekerja lebih efisien dan menghasilkan hasil pencarian yang relevan walaupun harus melakukan pencarian dari berbagai dokumen dengan konteks yang rumit.

Tujuan dari penulisan makalah ini adalah untuk memberikan pemahaman mengenai peran *Singular Value Decomposition* dan *cosine* similarity dalam meningkatkan relevansi pencarian dokumen berbasis teks. Dengan

demikian, diharapkan pembaca dapat memahami peran kedua metode tersebut dalam menghasilkan pencarian yang lebih relevan dan akurat.

II. KAJIAN TEORI

2.1. Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF adalah metode yang digunakan untuk memberikan nilai yang lebih besar pada kata-kata yang dianggap lebih relevan atau signifikan dalam sebuah dokumen atau kumpulan dokumen[1]. Metode ini memiliki 2 cara dalam memberikan bobot nilai tersebut, yaitu dengan *Term Frequency* dan *Inverse Document Frequency*.

Term Frequency (TF) adalah menghitung banyaknya suatu kata muncul dalam sebuah dokumen[1]. Cara menghitungnya adalah dengan membagi jumlah kemunculan kata tersebut dengan total kata yang ada dalam dokumen.

$$TF(t,d) = \frac{f(t,d)}{\sum_k f(k,d)}$$

Dengan, t = kata tertentu,
d = dokumen tertentu,
$f(t,d)$ = jumlah kemunculan t dalam d,
$\sum_k f(k,d)$ = total kata dalam dokumen d.

Inverse Document Frequency (IDF) digunakan untuk menurunkan bobot kata-kata yang sering muncul di berbagai dokumen sekaligus memberikan bobot yang lebih besar pada kata-kata yang jarang ditemukan[1]. Cara menghitungnya adalah dengan membagi total dokumen dalam korpus dengan jumlah dokumen yang mengandung kata yang sedang dicari.

$$IDF(t) = \log\frac{N}{df(t)}$$

Dengan, t = kata tertentu,
N = total dokumen dalam korpus,
$df(t)$ = jumlah dokumen yang mengandung t.

Namun, rumus tersebut memiliki kekurangan jika digunakan dalam sebuah dokumen yang tidak kompleks. Jika dalam suatu dokumen tidak terdapat kata sesuai *query*, maka hasil perhitungannya akan error karena $df(t)$ nya adalah 0. Maka, terdapat rumus modifikasi yang sering digunakan dalam implementasi praktis untuk menghindari error pembagian nol.

$$IDF(t) = \log\frac{1+N}{1+\ df(t)} + 1$$

Dengan, t = kata tertentu,
N = total dokumen dalam korpus,
$df(t)$ = jumlah dokumen yang mengandung t.

Menghitung TF-IDF adalah mengalikan hasil dari TF dan IDF.

$$TFIDF = TF(t,d) \times IDF(t)$$

Setelah menghitung TF-IDF, diperlukan normalisasi TF untuk memastikan bahwa bobot setiap kata dalam dokumen tidak dipengaruhi oleh panjang dokumen tersebut. Normalisasi dilakukan dengan membagi setiap nilai TF-IDF dalam dokumen dengan panjang vektor TF-IDF. Panjang vektor dihitung dengan rumus L2 Norm sebagai berikut.

$$Norm(d) = \sqrt{\sum_{t\in d}\left(TF\ IDF\ (t,d)\right)^2}$$

Rumus Normalisasi TF-IDF untuk sebuah kata adalah sebagai berikut.

$$TFIDF_{Normalized}(t,d) = \frac{TFIDF(t,d)}{Norm(d)}$$

2.2. Singular Value Decomposition (SVD)

SVD adalah salah satu metode dekomposisi matriks yang memfaktorkan matriks A berukuran $m \times n$ menjadi matriks U, Σ, dan V^T[2]. Metode SVD digunakan untuk mengidentifikasi kesamaan antar segmen kata. Metode ini berfungsi sebagai komponen pemrosesan yang merangkum informasi dalam jumlah besar ke dalam dimensi yang lebih kecil sehingga dapat menghilangkan informasi yang kurang signifikan terhadap hasil akhir tanpa mengurangi kualitas analisis[3]. Berikut adalah bentuk umum dari SVD.

$$A = U\Sigma V^T$$

Dengan, A = matriks asal
U = matriks orthonormal
Σ = matriks diagonal
V^T = transpose dari matriks orthogonal V

Matriks U adalah matriks orthonormal berukuran $m \times n$, di mana m adalah jumlah kata unik dan n adalah jumlah dimensi dari hasil dekomposisi. Matriks ini merepresentasikan informasi tentang kata-kata yang dapat dianggap sebagai vektor dengan n dimensi. Jumlah dimensi bergantung pada ukuran data yang digunakan.

Matriks Σ adalah matriks diagonal yang setiap elemen pada diagonalnya menunjukkan kekuatan hubungan antara dimensi tertentu dengan data. Matriks ini dapat mengabaikan dimensi-dimensi yang kurang relevan.

Matriks V^T adalah matriks berukuran $m \times n$ yang menyimpan informasi tentang dokumen. Setiap kolom pada matriks ini mewakili sebuah dokumen, sehingga informasi dokumen dapat dinyatakan sebagai vektor dalam n dimensi.

2.3. Latent Semantic Analysis (LSA)

LSA adalah metode dalam pemrosesan teks yang digunakan untuk melihat hubungan antara dokumen dan kata-kata di dalamnya. LSA membantu menemukan konsep-konsep yang menghubungkan kata-kata dan dokumen, sehingga membantu memahami arti tersembunyi dari teks[4].

Prinsip dari LSA adalah menemukan kemiripan makna pada kata-kata yang sering muncul bersamaan. Contohnya,

dalam artikel mengenai teknologi, sering dijumpai kata "teknologi", "inovasi", "AI", "*digital transformation"* sering ditemukan bersama. LSA mengenali pola-pola ini untuk memahami konteks dan arti dari kata-kata tersebut.

Proses LSA dimulai dengan membuat matriks yang menunjukkan seberapa sering kata muncul dalam dokumen. Lalu, metode SVD digunakan untuk memperkecil ukuran matriks, sambil tetap mempertahankan hubungan penting antara kata dan dokumen. Ini membuat analisis hubungan antar kata menjadi lebih mudah.

Langkah-langkah melakukan LSA adalah yang pertama membuat matriks dengan cara TF-IDF, melakukan dekomposisi matriks dengan metode SVD, mengurangi dimensi agar lebih fokus pada hubungan utama antara kata dan dokumen, sehingga hanya informasi penting yang ditampilkan, serta menghitung kemiripan dengan *cosine similarity*.

2.4. Cosine Similarity

Cosine similarity adalah metode untuk mengukur tingkat kemiripan antara dua vektor dengan melihat arah keduanya, tanpa memedulikan besarannya[5]. Metode ini menghitung kosinus dari sudut antara dua vektor, sehingga cocok digunakan untuk mesin pencari teks. Rumus dari *cosine similarity* antara dua vektor A dan B adalah sebagai berikut.

$$similarity = \cos(\theta) = \frac{A \,.\, B}{|A|.|B|}$$

Dengan, $A \,.\, B$ = hasil *dot product* dari A dan B,
$|A|$ = panjang (magnitudo) vektor A
$|B|$ = panjang (magnitudo) vektor B

Nilai hasil perhitungan *cosine similarity* berada di antara -1 dan 1[5].

- Nilai 1: kedua vektor mengarah ke arah yang sama (sangat mirip).
- Nilai 0: kedua vektor tidak memiliki hubungan arah (tidak mirip sama sekali).
- Nilai -1: kedua vektor berlawanan arah.

Mesin pencari teks memanfaatkan *cosine similarity* untuk memberikan hasil yang relevan serta mengurutkan hasil pencarian. *Query* yang diinput oleh pengguna dan dokumen diubah menjadi vektor, lalu dihitung tingkat kesamaannya dengan *cosine similarity*. Dokumen yang paling mirip dengan *query* pengguna akan muncul di urutan teratas, sehingga pengguna mendapatkan hasil yang paling sesuai.

III. HASIL DAN PEMBAHASAN

Misalkan kita memiliki tiga buah dokumen sebagai berikut.

- D1 = Institut Teknologi Bandung adalah multikampus yang berada di empat tempat
- D2 = Teknik Informatika adalah salah satu jurusan yang ada di Institut Teknologi Bandung
- D3 = Jurusan Teknik Informatika berada di kampus Ganesha dan Jatinangor

Pengguna ingin mencari suatu dokumen dengan masukan *query* sebagai berikut.

- *Query* = Teknik Informatika di Ganesha

Langkah pertama dalam implementasi mesin pencari teks adalah dengan menerapkan metode TF-IDF.

3.1. Menghitung dan Membuat Tabel TF-IDF

Proses pertama dalam menganalisis dokumen-dokumen tersebut adalah dengan melakukan *lemmatization. Lemmatization* merupakan tahap *preprocessing* teks yang bertujuan untuk menyederhanakan kata-kata ke bentuk dasarnya. Proses ini dilakukan dengan mempertimbangkan konteks gramatikal sehingga hasilnya lebih akurat dan sesuai dengan makna asli kata.

Contoh kata yang berubah setelah dilakukan *lemmatization:*

Tabel 1. Perubahan Kata Sebelum dan Setelah Dilakukan Lemmatization

Sebelum *Lemmatization*	Setelah *Lemmatization*
berada	ada

Sumber: Data diolah sendiri

Terdapat total 19 kata dari tiga dokumen tersebut setelah dilakukannya *lemmatization*. Setiap kata dihitung bobot nilainya dengan menggunakan rumus TF dan IDF. Contoh perhitungan untuk kata "institut" pada setiap dokumen adalah sebagai berikut.

Diketahui:
Jumlah kemunculan kata "institut" pada D1= 1
Jumlah kemunculan kata "institut" pada D2= 1
Jumlah kemunculan kata "institut" pada D3= 0
Banyaknya dokumen = 3
Banyaknya dokumen yang memiliki kata "institut" = 2
Total seluruh kata pada D1 = 10
Total seluruh kata pada D2 = 12
Total seluruh kata pada D3 = 9

- TF-IDF kata "institut" terhadap dokumen D1

$$TF(t, D1) = \frac{f(t, D1)}{\sum_k f(k, D1)} = \frac{1}{10} = 0.1$$

$$IDF(t) = \log\frac{1+N}{1+\ df(t)} + 1 = \log\frac{1+3}{1+2} + 1 = 1.12$$

$$\begin{aligned} TFIDF(t, D1) &= TF(t, D1) \times IDF(t) \\ &= 0.1 \times 1.12 \\ &= 0.112 \end{aligned}$$

Lakukan normalisasi TF-IDF:

$$Norm(D1) = \sqrt{\sum_{t \in D1} \left(TF\ IDF\ (t, D1)\right)^2} = 0.88$$

$$TFIDF_{Normalized}(t, D1) = \frac{TFIDF(t, D1)}{Norm(D1)}$$

$$= \frac{0.112}{0.88}$$

$$= 0.128$$

Jadi, diperoleh $TFIDF(\text{"institut"}, D1) = 0.128$.

- TF-IDF kata "institut" terhadap dokumen D2

$$TF(t, D1) = \frac{f(t, D2)}{\sum_k f(k, D2)} = \frac{1}{12} = 0.083$$

$$IDF(t) = \log \frac{1 + N}{1 + df(t)} + 1 = \log \frac{1 + 3}{1 + 2} + 1 = 1.12$$

$$TFIDF(t, D2) = TF(t, D2) \times IDF(t)$$
$$= 0.1 \times 1.12$$
$$= 0.112$$

Lakukan normalisasi TF-IDF:

$$Norm(D2) = \sqrt{\sum_{t \in D2} \left(TF\ IDF\ (t, D2)\right)^2} = 1.043$$

$$TFIDF_{Normalized}(t, D2) = \frac{TFIDF(t, D2)}{Norm(D2)}$$

$$= \frac{0.112}{1.043}$$

$$= 0.107$$

Jadi, diperoleh $TFIDF(\text{"institut"}, D2) = 0.107$.

- TF-IDF kata "institut" terhadap dokumen D3

$$TF(t, D3) = \frac{f(t, D3)}{\sum_k f(k, D3)} = \frac{0}{9} = 0$$

$$IDF(t) = \log \frac{1 + N}{1 + df(t)} + 1 = \log \frac{1 + 3}{1 + 2} + 1 = 1.12$$

$$TFIDF(t, D3) = TF(t, D3) \times IDF(t)$$
$$= 0 \times 1.12$$
$$= 0$$

Karena hasil dari TF-IDF kata "institut" pada dokumen D3 adalah 0, maka perhitungan normalisasi TF-IDF seperti sebelumnya tidak diperlukan. Sehingga, diperoleh $TFIDF(\text{"institut"}, D3) = 0$.

Dengan cara yang sama, hitung setiap kata untuk ketiga dokumen tersebut. Berikut adalah tabel hasil TF-IDF dari setiap kata pada ketiga dokumen tersebut.

Tabel 2. Hasil Perhitungan TF-IDF dari Setiap Kata pada Ketiga Dokumen

Kata\Dokumen	D1	D2	D3
institut	0.128	0.107	0
teknologi	0.128	0.107	0
bandung	0.128	0.107	0
adalah	0.128	0.107	0
multikampus	0.169	0	0
yang	0.128	0.107	0
ada	0.1	0.083	0.11
di	0.1	0.083	0.11
empat	0.169	0	0
tempat	0.169	0	0
teknik	0	0.107	0.143
informatika	0	0.107	0.143
salah	0	0.141	0
satu	0	0.141	0
jurusan	0	0.107	0.143
kampus	0	0	0.188
ganesha	0	0	0.188
dan	0	0	0.188
jatinangor	0	0	0.188

Sumber : Data diolah sendiri

Pada python, TF-IDF dapat dihitung dengan membuat beberapa fungsi. Fungsi `compute_tf` berfungsi untuk memisahkan kata dalam dokumen dan disimpan dalam daftar kata lalu menghitung jumlah kemunculan setiap kata dengan melakukan iterasi melalui daftar kata kemudian dinormalisasi. Fungsi `compute_idf` berfungsi untuk menggabungkan semua kata dari seluruh dokumen lalu dilakukan iterasi pada setiap kata untuk menghitung nilai IDF dengan rumus sama seperti yang telah dijelaskan sebelumnya. Fungsi `compute_tf_idf` berfungsi untuk menghitung nilai TF-IDF dengan mengalikan hasil TF dan IDF. Implementasi pada python adalah sebagai berikut.

```python
def compute_tf(doc):
    tf = {}
    words = doc.split()
    total_words = len(words)
    for word in words:
        tf[word] = tf.get(word, 0) + 1
    for word in tf:
        tf[word] /= total_words
    return tf

def compute_idf(docs):
    import math
    idf = {}
    total_docs = len(docs)
    all_words = set(word for doc in docs for word in doc.split())
    for word in all_words:
        doc_count = sum(1 for doc in docs if word in doc.split())
        idf[word] = math.log((1 + total_docs) / (1 + doc_count)) + 1
    return idf

def compute_tf_idf(tf, idf):
    tf_idf = {}
    for word, tf_value in tf.items():
        tf_idf[word] = tf_value * idf.get(word, 0)
    return tf_idf
```

Gambar 1. Implementasi TF-IDF pada python
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Output yang dihasilkan berupa matriks TF-IDF:

```
Langkah 1: Matriks TF-IDF
    Bandung   Ganesha  Informatika  ...      satu    tempat      yang
D1  0.128768  0.000000     0.000000  ...  0.000000  0.169315  0.128768
D2  0.107307  0.000000     0.107307  ...  0.141096  0.000000  0.107307
D3  0.000000  0.188127     0.143076  ...  0.000000  0.000000  0.000000
```

Gambar 2. Output Matriks TF-IDF
Sumber: https://github.com/anellautari/Makalah-Algeo.git

3.2. Implementasi SVD dan LSA

Setelah menghitung TF-IDF, matriks yang dihasilkan akan dipakai sebagai matriks A untuk menghitung SVD. Matriks U diperoleh dengan menghitung perkalian matriks A dengan A^T kemudian hitung nilai eigen dari matriks AA^T tersebut. Setelah dapat nilai eigen, hitung vektor-vektor eigen berdasarkan nilai eigen tidak nol dari AA^T. Vektor-vektor eigen tersebut dinormalisasi dan menghasilkan matruks U.

Matriks V dihasilkan dari menghitung perkalian matriks A^T dengan A kemudian dengan cara yang sama, hitung nilai eigen dari matriks A^TA dan hitung vektor-vektor eigen berdasarkan nilai eigen tidak nol dari A^TA. Normalisasikan vektor-vektor eigen tersebut lalu lakukan transpose matriks dan dihasilkanlah matriks V.

Matriks singular (Σ) dihasilkan dari nilai singular yang didapat dari perhitungan akar kuadrat nilai eigen tidak nol dari matriks A^TA.

Implementasi program python untuk menghitung SVD memanfaatkan *library* "scikit-learn". Pada program ini, digunakan `TruncatedSVD` untuk melakukan dekomposisi matriks TF-IDF (A) menjadi tiga komponen, yaitu U, Σ, dan V^T. Matriks U dihitung dengan memanggil fungsi `svd.fit_transform`. Matriks yang dihasilkan berukuran $m \times k$, dengan m adalah jumlah dokumen dan k adalah jumlah komponen laten. Pada program, jumlah komponen laten dipilih 3 karena data set hanya terdiri dari 3 dokumen, sehingga pengurangan dimensi menjadi 3 komponen laten tetap menjaga informasi penting dari data. Matriks sigma (Σ) dihitung dengan menyimpan nilai singular dengan memanggil fungsi `svd.singular_values_`. Matriks V^T dihitung dengan menggunakan fungsi svd.components_ lalu diperoleh matriks berukuran $k \times n$, dengan k adalah jumlah komponen laten dan n adalah jumlah kata unik. Berikut adalah implementasi dalam program python.

```
svd = TruncatedSVD(n_components=3)
U_k = svd.fit_transform(tf_idf_df.values)
Sigma_k = svd.singular_values_
V_k = svd.components_
```

Gambar 3. Implementasi SVD pada python
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Berikut adalah matriks U yang dihasilkan:

```
Matriks U :
    Component 1  Component 2  Component 3
D1     0.298431    -0.260228    -0.196750
D2     0.277123    -0.168139     0.249223
D3     0.371934     0.334078    -0.027825
```

Gambar 4. Output matriks U
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Matriks Σ:

```
Matriks Sigma:
          Value
Sigma 1  0.551537
Sigma 2  0.455629
Sigma 3  0.318742
```

Gambar 4. Output matriks Σ
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Dan matriks V^T:

```
Matriks V^T:
             Bandung   Ganesha  Informatika  ...      satu    tempat      yang
Component 1  0.224087  0.230022     0.272695  ...  0.128540  0.166108  0.224087
Component 2 -0.248324  0.302746     0.143336  ... -0.114277 -0.212240 -0.248324
Component 3  0.013861 -0.051524     0.224045  ...  0.346116 -0.327891  0.013861
```

Gambar 5. Output matriks V^T
Sumber: https://github.com/anellautari/Makalah-Algeo.git

3.3. Menghitung *Cosine Similarity*

Untuk menghitung *cosine similarity*, langkah pertama adalah menghitung proyeksi *query* dan dokumen D1, D2, D3 ke ruang laten yang dihasilkan oleh LSA. Rumus perhitungan proyeksi *query* dan setiap dokumen ke ruang laten adalah sebagai berikut.

$$Q_{lat} = Q \,.\, V \,.\, \Sigma^{-1}$$
$$D_{lat} = U[i] \,.\, \Sigma$$

Dengan, Q = matiks TF-IDF dari *query*
V = transpose dari matriks V_k
Σ^{-1} = invers dari matriks singular Σ_k
$U[i]$ = vektor baris ke-i dari matriks U
Σ = matriks diagonal yang berisi nilai singular

Setelah menghitung proyeksi *query* dan dokumen ke ruang laten, hitung *cosine similarity* dengan rumus yang sama seperti pada Bab Kajian Teori dengan mengganti vektor A dengan vektor Q_{lat} dan vektor B dengan vektor D_{lat}. Rumusnya akan menjadi seperti ini.

$$Cosine\ Similarity = \frac{Q_{lat} \,.\, D_{lat}}{||Q_{lat}|| \times ||D_{lat}||}$$

Implementasi pada program python adalah menghitung representasi TF-IDF dari *query* dengan menggunakan fungsi `compute_query_tf_idf`. Setelah itu, hitung proyeksi *query* ke ruang laten dengan menggunakan fungsi `np.dot` dari *library* NumPy. Hitung juga proyeksi dokumen ke ruang laten dengan fungsi `np.dot` dari *library* NumPy. Langkah terakhir adalah menghitung *cosine similarity* antara Q_{lat} dengan setiap dokumen D_{lat}.

```
query = "Teknik Informatika di Ganesha"
def compute_query_tf_idf(query, idf):
    query_tf = compute_tf(query)
    query_tf_idf = compute_tf_idf(query_tf, idf)
    return np.array([query_tf_idf.get(word, 0) for word in all_words])

query_tfidf = compute_query_tf_idf(query, idf)

query_latent_space = np.dot(query_tfidf, np.dot(V_k.T, np.linalg.pinv(np.diag(Sigma_k))))

document_latent_space = np.dot(U_k, np.diag(Sigma_k))

cosine_similarities = cosine_similarity([query_latent_space], document_latent_space)
```

Gambar 6. Implementasi *cosine similarity* pada python
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Hasil dari Q_{lat} dan D_{lat} dari program tersebut adalah sebagai berikut.

```
Q_lat (Query di Ruang Laten):
[0.63532816 0.47609804 0.36870872]

D1_lat (Dokumen 1 di Ruang Laten):
[ 0.16459564 -0.11856722 -0.06271244]

D2_lat (Dokumen 2 di Ruang Laten):
[ 0.15284348 -0.07660878  0.07943779]

D3_lat (Dokumen 3 di Ruang Laten):
[ 0.20513533  0.15221561 -0.00886904]
```

Gambar 7. Output hasil Q_{lat} dan D_{lat}
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Dari hasil Q_{lat} dan D_{lat} tersebut, kita dapat menghitung *cosine similarity* dengan rumus yang telah dijelaskan sebelumnya.

- Untuk Dokumen D1

$$Cosine\ Similarity = \frac{Q_{lat} \cdot D[1]_{lat}}{||Q_{lat}|| \times ||D[1]_{lat}||} = \frac{0.025}{0.874 \times 0.211} = 0.135$$

- Untuk Dokumen D2

$$Cosine\ Similarity = \frac{Q_{lat} \cdot D[2]_{lat}}{||Q_{lat}|| \times ||D[2]_{lat}||} = \frac{0.089}{0.874 \times 0.187} = 0.544$$

- Untuk Dokumen D3

$$Cosine\ Similarity = \frac{Q_{lat} \cdot D[3]_{lat}}{||Q_{lat}|| \times ||D[3]_{lat}||} = \frac{0.199}{0.874 \times 0.255} = 0.892$$

Dari hasil perhitungan *cosine similarity* untuk setiap dokumen dengan *query* "Teknik Informatika di Ganesha", diperoleh hasilnya sebagai berikut:

- *Cosine similarity query* dan D1 = 0.135
- *Cosine similarity query* dan D2 = 0.544
- *Cosine similarity query* dan D3 = 0.892

Berikut adalah output dari program pada gambar 7 yang merupakan hasil dari perhitungan *cosine similarity* antara *query* dengan setiap dokumen.

```
Langkah 3: Cosine Similarity
            D1        D2        D3
Query  0.134508  0.5449  0.891792
```

Gambar 8. Output hasil *cosine similarity*
Sumber: https://github.com/anellautari/Makalah-Algeo.git

Dapat dilihat, hasil *cosine similarity* yang paling mendekati angka 1 (yang paling akurat) adalah pada perhitungan antara *query* dengan dokumen D3, yaitu bernilai 0.892 (hasil pembulatan). Jadi, dapat disimpulkan bahwa *query* "Teknik Informatika di Ganesha" memiliki kesamaaan cosinus paling dekat dengan Dokumen D3 yang berisi "Jurusan Teknik Informatika berada di kampus Ganesha dan Jatinangor".

IV. KESIMPULAN

Makalah ini membahas mengenai penerapan *Singular Value Decomposition* (SVD) dan *cosine similarity* untuk membuktikan keakuratan dalam mesin pencari teks. Berdasarkan hasil implementasi dan pengujian, metode ini terbukti mampu meningkatkan relevansi hasil pencarian dengan cara:

1. Mengidentifikasi hubungan laten antar kata melalui dekomposisi matriks TF-IDF.
2. Mengurangi dimensi data tanpa menghilangkan informasi penting, sehingga mempermudah analisis dokumen.
3. Memberikan hasil pencarian yang lebih akurat berdasarkan *cosine similarity*.

Hasil pengujian didapatkan *query* "Teknik Informatika di Ganesha" memiliki kesamaan tertinggi dengan dokumen D3, yaitu 0,892. Dengan demikian, metode SVD dan *cosine similarity* dapat digunakan untuk membuat dan mengembangkan mesin pencari teks yang efisien dan relevan, meskipun terdapat tantangan seperti kompleksitas dan keambiguan kata kunci. Penerapan lebih lanjut dapat menggunakan dataset yang lebih besar dan kompleks untuk meningkatkan kinerja mesin pencari teks.

V. SARAN

Berdasarkan hasil penelitian, penulis memberikan beberapa saran. Pertama, penelitian lebih lanjut dapat menggunakan dataset yang lebih besar dan beragam untuk menguji performa metode *Singular Value Decomposition* (SVD) dan *cosine similarity*. Selain itu, sistem dapat dikembangkan dengan mengintegrasikan model pembelajaran mesin untuk meningkatkan akurasi pencarian dokumen.

VI. LAMPIRAN

Tautan repository:
https://github.com/anellautari/Makalah-Algeo.git

Tautan video youtube:
https://youtu.be/2yjxU6Ut-1k

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas rahmat dan kemudahan yang diberikan sehingga makalah ini dapat diselesaikan. Penulis juga mengucapkan terima kasih kepada keluarga tercinta atas dukungan dan doanya, serta kepada Bapak Dr. Ir. Rinaldi Munir, M.T. sebagai dosen mata kuliah Aljabar Linier dan Geometri yang telah memberikan ilmu dan arahan selama perkuliahan. Penulis berharap makalah ini dapat memberikan manfaat bagi pembaca.

Referensi

[1] E. Sairina, "Mengenal Term Frequency-Inverse Document Frequency (TF-IDF) pada Model NLP," *Medium*, 5 Februari 2024. [Online]. Tersedia: https://esairina.medium.com/mengenal-term-frequency-inverse-document-frequency-tf-idf-pada-model-nlp-e0cc571f7e37. [Diakses: 31-Des-2024].

[2] R. Munir, "Singular Value Decomposition Bagian 1," *Program Studi Teknik Informatika, Institut Teknologi Bandung*, 2023. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf. [Diakses: 31-Des-2024].

[3] W. Valle, "Singular Value Decomposition for developers—Selecting what matters from text data," *Medium*, 10 Januari 2023. [Online]. Tersedia: https://medium.com/artificial-intelligence-ai/fordev-singular-value-decomposition-selecting-what-matters-from-text-data-part-1-77bec0748691. [Diakses: 31-Des-2024].

[4] "Apa yang dimaksud dengan Latent Semantic Analysis (LSA)," *Ranktracker*, [Online]. Tersedia: https://www.ranktracker.com/id/seo/glossary/latent-semantic-analysis-lsa/. [Diakses: 31-Des-2024].

[5] TiDB Team, "Understanding the Cosine Similarity Formula," *PingCAP*, 16 Juli 2024. [Online]. Tersedia: https://www.pingcap.com/article/understanding-the-cosine-similarity-formula/. [Diakses: 31-Des-2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Januari 2025

Anella Utari Gunadi 13523078