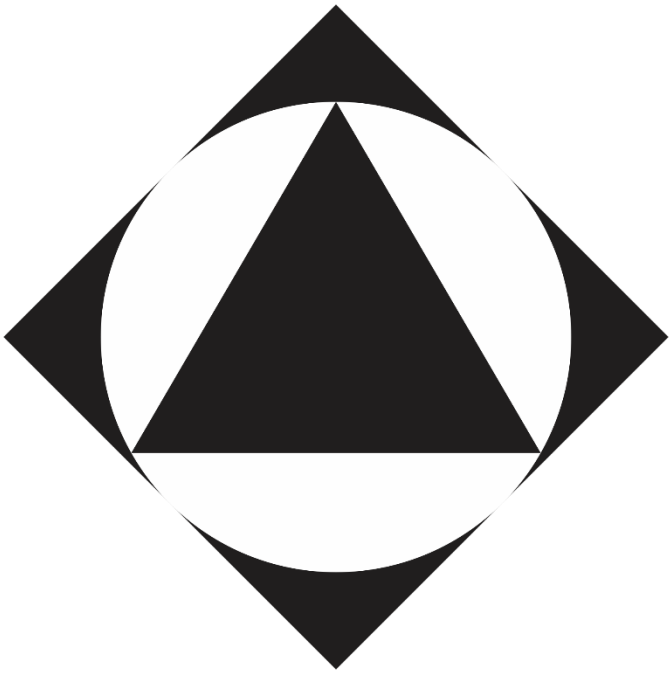
TUGAS BESAR

IMPLEMENTASI TEMU BALIK INFORMASI DENGAN ALGORITMA VSM

Diajukan untuk Memenuhi Tugas Mata Kuliah Data Mining and Information Retrieval

Dosen Pengampu : Jasman Pardede, Dr., S.Si., M.T.



Disusun Oleh Kelompok 1 :

152021140 Muhamad Dicky Ardiansah
152021149 Mohamad Hafiz Daffa
152021169 Tegar Subagdja

PRODI INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI NASIONAL BANDUNG

2023

KATA PENGANTAR

Puji dan Syukur penulis panjatkan kepada Tuhan yang maha esa karena berkat dan karunianya penulis dapat mengerjakan dan menyelesaikan makalah ini. Makalah ini disusun sebagai bagian dari tugas mata kuliah Data Mining dan Information Retrieval.

Pemilihan topik ini didasarkan pada keinginan untuk mengembangkan pengetahuan dan keterampilan dalam penerapan program Python untuk membaca dokumen teks. Fokus utama makalah ini adalah pada implementasi program sederhana yang memanfaatkan bahasa pemrograman Python untuk membaca, memproses, dan mengeksplorasi dokumen teks.

Penulis menyadari bahwa makalah ini mungkin mengandung kesalahan tata bahasa, kosa kata, dan isi. Oleh karena itu, penulis sangat mengharapkan masukan, komentar, dan saran dari pembaca untuk perbaikan dan pengembangan lebih lanjut.

Makalah ini diharapkan dapat memberikan wawasan tentang implementasi program membaca dokumen teks dengan menggunakan bahasa pemrograman Python, serta menjadi panduan praktis bagi pembaca yang tertarik dalam pengolahan teks menggunakan teknologi ini. Semoga makalah ini bermanfaat dan dapat menjadi referensi yang berguna.

Bandung, 1 Januari 2024

Penulis

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1. Latar Belakang

Pertumbuhan Pertumbuhan data dokumen teks yang cepat memunculkan tantangan dalam pengelolaan dan pengambilan informasi yang relevan. Information Retrieval (IR), sebagai bidang ilmu yang menangani hal ini, memerlukan teknik-teknik efektif, seperti algoritma Vector Space Model (VSM).

Dalam konteks IR, pembangunan program yang mampu membaca dokumen teks dengan efisien menjadi krusial. Python, sebagai bahasa pemrograman serbaguna, menyediakan pustaka dan modul yang mendukung implementasi program IR, menjadikannya pilihan yang ideal untuk menangani pertumbuhan pesat data dokumen.

Keefektifan dan efisiensi dalam proses pembacaan dokumen teks semakin penting, mendorong penelitian dan pengembangan teknik seperti stemming. Stemming adalah proses mencari bentuk dasar kata, dan dalam konteks IR, hal ini dapat meningkatkan efisiensi program dengan mengelompokkan kata-kata yang memiliki akar kata dan arti serupa. Sastrawi menjadi salah satu algoritma stemming yang populer dan dapat diimplementasikan dengan mudah menggunakan Python.

Tujuan dari penggunaan Python dalam Information Retrieval, khususnya dengan mengadopsi algoritma stemming Sastrawi, adalah untuk memberikan panduan praktis bagi pengembang dan praktisi. Python memanfaatkan kekuatannya untuk mengatasi tantangan pengambilan informasi dari dokumen teks yang semakin kompleks.

Dengan terus mengeksplorasi potensi Python dan menerapkan teknologi seperti stemming, implementasi program IR dapat menjadi lebih efektif. Melalui pendekatan ini, diharapkan dapat memahami peran Python sebagai alat yang efektif dalam mendukung Information Retrieval dan meningkatkan pemahaman terhadap algoritma VSM, khususnya saat diterapkan dalam temu balik informasi.

1.2. Rumusan Masalah

1. Bagaimana implementasi Information Retrieval dapat ditingkatkan melalui penggunaan teknik Case Folding untuk mengatasi masalah variasi huruf kapital?
2. Apa dampak penggunaan Tokenizing dalam proses Information Retrieval terhadap akurasi dan efisiensi dalam pemrosesan teks?

3. Bagaimana penerapan teknik Filtering pada data teks dapat mempengaruhi relevansi hasil Information Retrieval?
4. Sejauh mana penerapan stemming dengan menggunakan library Sastrawi dapat meningkatkan akurasi Information Retrieval pada bahasa Indonesia?
5. Bagaimana penggabungan Case Folding, Tokenizing, Filtering, dan Stemming dapat memberikan kontribusi signifikan terhadap peningkatan kualitas hasil Information Retrieval?

1.3. Tujuan

1. Mengimplementasikan information retrieval untuk penggunaan Teknik Case Folding untuk mengatasi masalah variasi huruf kapital.
2. Mengetahui dampak penggunaan Tokenizing dalam proses Information Retrieval terhadap akurasi dan efisiensi dalam pemrosesan teks.
3. Menerapkan Teknik filtering pada data teks untuk dapat mempengaruhi relevansi hasil information retrieval.
4. Menerapkan stemming dengen menggunakan library sastrawi dapat meningkatkan akurasi information retrieval pada Bahasa Indonesia.
5. Mengetahui penggabungan case folding, tokenizing, Filtering, dan Stemming untuk memberikan kontribusi signifikan terhadap peningkatan kualitas hasil information retrieval.

1.4. Manfaat

Implementasi program algoritma VSM Sastrawi dengan Python diharapkan memberikan dampak positif terhadap efisiensi dan efektivitas temu balik informasi pada koleksi dokumen berbasis teks, khususnya dalam konteks bahasa Indonesia. Dengan memanfaatkan metode VSM yang dioptimalkan oleh Sastrawi, sistem dapat lebih akurat dalam merepresentasikan hubungan antara dokumen dan query dalam ruang vektor multidimensional. Fitur Sastrawi, seperti stemming, memungkinkan sistem untuk mengatasi variasi kata dan mengidentifikasi akar kata yang seragam, sehingga meningkatkan konsistensi hasil temu balik. Penggunaan VSM Sastrawi diharapkan mampu memberikan peningkatan signifikan dalam kemampuan sistem untuk menemukan informasi yang relevan, sehingga meningkatkan kepuasan pengguna dalam mencari dan mengakses dokumen berbasis teks.

BAB II

LANDASAN TEORI

2.1. Data Mining

Data Mining adalah proses penggalian informasi dan pola yang bermanfaat dari suatu data yang sangat besar. Proses data mining terdiri dari pengumpulan data, ekstraksi data, analisa data, dan statistik data. Ia juga umum dikenal sebagai knowledge discovery, knowledge extraction, data/pattern analysis, information harvesting, dan lainnya (Arhami dan Nasir, 2020). Empat proses dalam data mining ini akan menghasilkan model atau pengetahuan yang sangat berguna. Menurut Muflikhah (2018), data mining dapat didefinisikan sebagai penguraian kompleks dari sekumpulan data menjadi informasi yang memiliki potensi secara implisit (tidak nyata/jelas) yang sebelumnya belum diketahui. Ia juga dapat didefinisikan sebagai penggalian dan analisis dengan menggunakan peralatan otomatis atau semi otomatis, dari sebagian besar data yang memiliki tujuan yaitu menemukan pola yang memiliki arti atau maksud. Data mining termasuk ke dalam knowledge discovery di dalam database (KDD).

2.2. Temu Balik Informasi

Temu kembali informasi (information retrieval) adalah suatu proses pencarian informasi di dalam kumpulan dokumen, baik itu dokumen itu sendiri, metadata yang menjelaskannya, atau di dalam database. Tujuan utamanya adalah untuk menemukan kembali informasi yang relevan terhadap kebutuhan pengguna. Sistem temu kembali informasi didesain untuk menemukan dokumen atau informasi yang diperlukan oleh pengguna, dengan membuang dokumen yang tidak relevan. Proses ini melibatkan dua komponen utama, yaitu sistem pengindeksan yang menghasilkan indeks dokumen, dan sistem temu kembali yang merupakan gabungan dari user interface dan look-up-table. Sistem temu kembali informasi bertujuan untuk menjawab kebutuhan informasi pengguna dengan sumber informasi yang tersedia, sambil meminimalkan jumlah dokumen yang tidak relevan. Terdapat berbagai model pendekatan yang digunakan dalam sistem temu kembali informasi, seperti model ruang vektor, model Boolean, dan model probabilistik.

2.3. Stemming

Stemming adalah proses dalam pengolahan bahasa alami yang bertujuan untuk mengubah kata-kata ke bentuk dasar atau akar kata. Proses ini dilakukan dengan menghilangkan semua imbuhan (affixes) baik yang terdiri dari awalan (prefixes), sisipan (infixes), maupun akhiran (suffixes) dari kata. Tujuan utama dari proses stemming adalah untuk menyederhanakan variasi morfologis kata, sehingga kata-kata dengan akar yang sama dapat dikenali sebagai bentuk yang sama. Hal ini memungkinkan untuk memperkecil jumlah indeks yang berbeda dari suatu dokumen, serta untuk melakukan pengelompokan kata-kata lain yang memiliki kata dasar dan arti yang serupa namun memiliki bentuk atau form yang berbeda karena mendapatkan imbuhan yang berbeda.

Terdapat beberapa teknik stemming yang dapat dikategorikan menjadi 3, yaitu berdasarkan aturan dalam bahasa tertentu, berdasarkan kamus, dan berdasarkan kemunculan bersama. Salah satu tujuan utama dilakukan proses stemming adalah meningkatkan efisiensi dengan cara memilah isi dokumen menjadi unit-unit kecil yang akan menjadi penciri, misalnya berupa kata, frase, atau kalimat.

Proses stemming memiliki peran yang penting dalam Information Retrieval (IR) karena memungkinkan untuk mencari kata dasar dari bentuk kata yang berimbuhan, sehingga kata yang dimasukkan ke dalam indeks adalah dalam bentuk umum, sehingga dapat menghasilkan dokumen yang lebih relevan. Proses ini juga dapat meningkatkan efisiensi pemrosesan teks dengan mengurangi variasi kata yang tidak perlu, sehingga ukuran kosakata dapat dikurangi, serta waktu dan sumber daya yang diperlukan untuk pemrosesan teks dapat berkurang.

2.4. Algoritma VSM

Model Ruang Vektor (Vector Space Model/VSM) adalah sebuah model yang digunakan dalam temu balik informasi untuk merepresentasikan dokumen dan istilah sebagai vektor dalam ruang multidimensi. Setiap dimensi dalam ruang vektor tersebut sesuai dengan sebuah istilah dalam kosakata. Dokumen dan query direpresentasikan sebagai vektor dalam ruang tersebut. VSM didasarkan pada asumsi bahwa makna dari sebuah dokumen dapat disimpulkan dari distribusi istilah-nya, dan bahwa dokumen-dokumen dengan konten yang mirip akan memiliki distribusi istilah yang mirip pula. Untuk menghitung tingkat kesamaan antara dokumen dan query, berbagai metode

pembobotan istilah telah dikembangkan, salah satunya adalah pembobotan tf-idf. Dengan VSM, pencocokan parsial dimungkinkan karena bobot non-biner diberikan pada istilah-istilah dalam query dan dokumen. Hasilnya, dokumen yang relevan dengan query dapat diurutkan berdasarkan tingkat kemiripannya, sehingga memungkinkan pengambilan dokumen yang lebih akurat sesuai dengan informasi yang dibutuhkan oleh pengguna.

VSM memungkinkan penggunaan operasi vektor untuk membandingkan dokumen dengan query. Model ini telah banyak digunakan dalam sistem temu balik informasi, filtrasi informasi, pengindeksan, dan peringkat relevansi. VSM juga dapat diimplementasikan dalam berbagai aplikasi, termasuk dalam membangun mesin pencari dan sistem temu balik dokumen. Selain itu, VSM juga dapat diintegrasikan dengan berbagai alat dan teknik matematika lainnya, seperti analisis semantik laten dan basis data leksikal, untuk mengatasi berbagai kesulitan yang mungkin muncul dalam implementasinya.

2.5. Library Sastrawi

Sastrawi adalah library Python yang dirancang khusus untuk mempermudah pemrosesan bahasa Indonesia, terutama dalam konteks analisis teks dan temu balik informasi. Dengan Sastrawi, pengguna dapat dengan mudah mengimplementasikan algoritma stemming, sebuah langkah kunci dalam pemrosesan teks yang bertujuan memotong kata-kata menjadi bentuk dasarnya. Library ini menyediakan fungsi dan alat yang intuitif, memungkinkan pengembang atau peneliti untuk membersihkan dan merapikan teks bahasa Indonesia dengan efisien, meningkatkan kemampuan analisis dan pencarian informasi yang relevan dalam koleksi dokumen.

Sastrawi memberikan solusi yang praktis dalam mengatasi kompleksitas bahasa Indonesia dalam pemrosesan teks, sehingga dapat digunakan dalam berbagai aplikasi seperti analisis sentimen, ekstraksi informasi, dan pengelolaan konten berbasis teks. Dengan dukungan Sastrawi, implementasi algoritma pemotongan kata-kata dalam bahasa Indonesia menjadi lebih sederhana dan mudah diintegrasikan ke dalam proyek-proyek pengembangan perangkat lunak yang melibatkan teks berbahasa Indonesia.

BAB III

PENJELASAN

3.1. Formula

3.1.1. Term Frequency

Term Frequency (TF) mengukur seberapa sering sebuah kata muncul dalam suatu dokumen. Rumusnya adalah:

$$Similarity(q, dj) = \sum_{n=1}^{n} Wiq.Wij$$

3.1.2. Cosine Similarity

Cosine Similarity mengukur kedekatan antara dua vektor dokumen.

$$Similarity(dokumen, query) = \frac{dj.q}{|dj|.|q|} = \frac{\sum_{n=1}^{n}(wij, wiq)}{\sqrt{\sum_{n=1}^{n} Wij^2 . \sum_{n=1}^{n} Wiq^2}}$$

3.2. Algoritma

1. Pre-processing

   Metode untuk membentuk Kumpulan kata dasar yang terindeks. Beberapa tahapan pre-processing.

   a. Case Folding

      Mengubah seluruh karakter pada dokumen menjadi huruf kecil. Hal ini dilakukan agar tidak ada perbedaan huruf besar ke huruf kecil yang dapat mempengaruhi proses selanjutnya.

   b. Tokenizing

      Proses untuk memecah (mem-parsing) kalimat kalimat menjadi kata-kata pembentuknya, baik yang penting maupun tidak penting. Pada proses Tokenizing pemisahan kata-kata dalam kalimat ini dilakukan berdasarkan adanya spasi.

   c. Filtering

      Filtering atau stopword removal untuk menghilangkan kata-kata yang tidak memiliki arti penting. Contoh kata yang tidak memiliki arti penting dalam Bahasa Indonesia adalah “yang”, “dan”, “di”, dan lainnya.

   d. Stemming

Proses mencari kata dasar dari tiap kata hasil filltering. Pada tahap ini dilakukan proses pengembalian berbagai bentuk kata ke dalam suatu representasi ke dalam suatu representasi yang sama.

Algoritma Stemming Sastrawi

1. Melakukan Pemeriksaan apakah kata yang akan destemming ada dalam kamus kata dasar atau tidak. Jika ada, maka proses stemming berhenti pada Langkah ini.
2. Jika kata tidak ada dalam kamus, artinya kata tersebut merupakan kata berimbuhan. Kemudian menghilangkan kata akhiran "-lah", "-kah", "-ku", "-mu", "-nya", "-tah" atau "-pun".
3. Menghilangkan kata imbuhan akhiran "-i", "-kan", "-an", kemudian hapus kata imbuhan awalan "be-", "di-", "ke-", "me-", "pe-", "-se", dan "te-".
4. Jika kata dasar yang dihasilkan dari Langkah sebelumnya tidak terdapat di kamus, maka kata tersebut dicek apakah termasuk pada table keambiguan atau tidak.
5. Jika seluruh proses Langkah 1-4 gagal dilakukan, maka algoritma akan mengembalikan kata aslinya.

2. Pembobotan

Merupakan perhitungan jumlah kemunculan kata pada dokumen. Semakin sering sebuah kata muncul dalam dokumen, semakin dianggap penting kata tersebut bagi dokumen tersebut.

3. Similarity

Metode similarity digunakan untuk menghitung tingkat kemiripan antara dokumen. Ini memungkinkan penentuan sejauh mana suatu dokumen mirip dengan dokumen lainnya berdasarkan representasi kata-kata yang telah dihasilkan melalui tahapan pre-processing dan pembobotan.

3.3. Studi Kasus

1. Dokumen dan Query

Dok1 : Air pasang melanda sangat cepat

Dok2 : Terjadi banjir akibat air pasang

Dok3 : Banjir disebabkan oleh intensitas air pasang yang tinggi

Query : banjir air pasang

2. Prepocessing
    a. Case Folding

        Dok1: air pasang melanda sangat cepat

        Dok2: terjadi banjir akibat air pasang

        Dok3: banjir disebabkan oleh intensitas air pasang yang tinggi

        Query : banjir air pasang

    b. Tokenizing

        Dok1: air, pasang, melanda, sangat, cepat

        Dok2: terjadi, banjir, akibat, air, pasang

        Dok3: banjir, disebabkan, oleh, intensitas, air, pasang, yang, tinggi

        Query : banjir, air, pasang

    c. Filtering

        Dok1: air, pasang, melanda, sangat, cepat

        Dok2: terjadi, banjir, akibat, air, pasang

        Dok3: banjir, disebabkan, intensitas, air, pasang, yang, tinggi

        Query : banjir, air, pasang

    d. Stemming (Arifin dan Setiono)

        Dok1: air, pasang, landa, sangat, cepat

        Dok2: jadi, banjir, akibat, air, pasang

        Dok3: banjir, sebab, intensitas, air, pasang, tinggi

        Query : banjir, air, pasang

3. Pembobotan

	1	2	3	4	5	6	7
	air	akibat	banjir	cepat	intensitas	jadi	landa
dok 1	1	0	0	1	0	0	1
dok 2	1	1	1	0	0	1	0
dok 3	1	0	1	0	1	0	0
query	1	0	1	0	0	0	0

	8	9	10	11
	pasang	sangat	sebab	tinggi
dok 1	1	1	0	0
dok 2	1	0	0	0
dok 3	1	0	1	1
query	1	0	0	0

4. Similarity (VSM)

   a. Dokomen 1

      Let d1 = (1,0,0,1,0,0,1,1,1,0,0) and d' = (1,0,1,0,0,0,0,1,0,0,0)

      D1xd' = (1x1) + (0x0) + (0x1) + (1x0) + (0x0) + (0x0) + (1x0) + (1x1) + (1x0) + (0x0) + (0x0) = 2

      $$|d1| = \sqrt{1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2} = \sqrt{5}$$

      $$= 2.236$$

      $$|d'| = \sqrt{1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2} = \sqrt{3}$$

      $$= 1.732$$

      Similarity = $\frac{2}{2.236 \; x \; 1.732} = 0.516$

   b. Dokumen 2

      Let d2 = (1,1,1,0,0,1,0,1,0,0,0) and d' = (1,0,1,0,0,0,0,1,0,0,0)

      D2xd' = (1x1) + (1x0) + (1x1) + (0x0) + (0x0) + (1x0) + (0x0) + (1x1) + (0x0) + (0x0) + (0x0) = 3

      $$|d2| = \sqrt{1^2 + 1^2 + 1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2} = \sqrt{5}$$

      $$= 2.236$$

      $$|d'| = \sqrt{1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2} = \sqrt{3}$$

      $$= 1.732$$

      Similarity = $\frac{3}{2.236 \; x \; 1.732} = 0.774$

   c. Dokumen 3

      Let d3 = (1,0,1,0,1,0,0,1,0,1,1) and d' = (1,0,1,0,0,0,0,1,0,0,0)

      D3xd' = (1x1) + (0x0) + (1x1) + (0x0) + (1x0) + (0x0) + (0x0) + (1x1) + (0x0) + (1x0) + (1x0) = 3

      $$|d3| = \sqrt{1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 1^2 + 1^2} = \sqrt{6}$$

      $$= 2.449$$

      $$|d'| = \sqrt{1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2} = \sqrt{3}$$

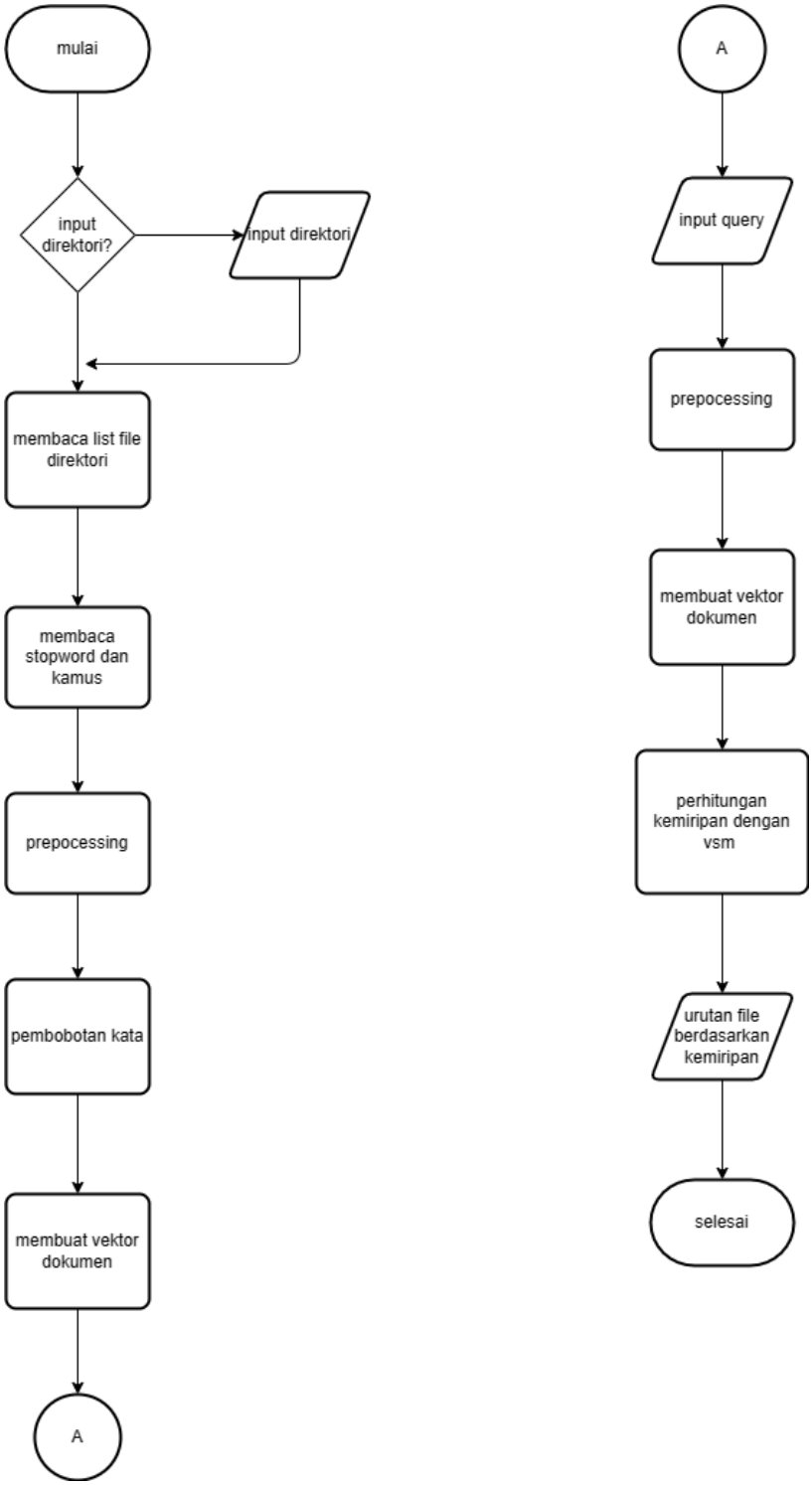
      $$= 1.732$$

      Similarity = $\frac{3}{2.449 \; x \; 1.732} = 0.707$

BAB IV

IMPLEMENTASI

4.1. FlowChart



Gambar 4.1.1 flow

4.2. Source Code

```
import os
import PyPDF2
from docx import Document
from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
import streamlit as st
from collections import Counter
from math import sqrt

def read_txt(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        content = file.read()
    return content

def read_pdf(file_path):
    with open(file_path, 'rb') as file:
        pdf_reader = PyPDF2.PdfReader(file)
        text = ""
        for page_num in range(len(pdf_reader.pages)):
            text += pdf_reader.pages[page_num].extract_text()
        return text

def read_docx(file_path):
    doc = Document(file_path)
    text = ""
    for paragraph in doc.paragraphs:
        text += paragraph.text + " "
    return text

def read_stop_words_sastrawi():
    stopword_factory = StopWordRemoverFactory()
    return stopword_factory.get_stop_words()

def read_dicti_sastrawi(file_path):
    dicti = {}
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) == 2:
                dicti[parts[0]] = parts[1]
            # else:
                # print(f"Ignoring invalid line: {line}")
    return dicti

def preprocess_text(text, stop_words, dicti):
    text_lower = text.lower()
    text_cleaned = ''.join(char for char in text_lower if char.isalpha() or char.isspace())
    tokens = text_cleaned.split()
    filtered_tokens = [token for token in tokens if token not in stop_words]
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    stemmed_tokens = [stemmer.stem(token) if token not in dicti else dicti[token] for token
in filtered_tokens]

    return text_lower, text_cleaned, filtered_tokens, tokens, stemmed_tokens

def preprocess_directory(directory_path, stop_words, dicti):
    results = []
    for filename in os.listdir(directory_path):
        file_path = os.path.join(directory_path, filename)
        if file_path.endswith(".txt"):
            content = read_txt(file_path)
        elif file_path.endswith(".pdf"):
```

```
            content = read_pdf(file_path)
        elif file_path.endswith(".docx"):
            content = read_docx(file_path)
        else:
            print(f"Ignoring unsupported file: {file_path}")
            continue

        original_lower, original_cleaned, original_filter, original_tokens, stemmed_tokens
= preprocess_text(content, stop_words, dicti)
        word_count = len(set(stemmed_tokens))
        results.append((filename, content, original_lower, original_cleaned,
original_filter, original_tokens, stemmed_tokens, word_count))
    return results

def calculate_similarity(query, documents, stop_words, dicti, all_words):
    query_stemmed_tokens = preprocess_text(query, stop_words, dicti)[3]
    query_vector = build_doc_vector(Counter(query_stemmed_tokens), all_words)
    document_vectors = [build_doc_vector(Counter(doc), all_words) for _, _, _, _, _, _,
doc, _ in documents]
    similarity_scores = [cosine_similarity(query_vector, doc_vector) for doc_vector in
document_vectors]
    return similarity_scores

def build_doc_vector(doc, all_words):
    doc_vector = [doc.get(word, 0) for word in all_words]
    return doc_vector

def cosine_similarity(vec1, vec2):
    dot_product = sum(x * y for x, y in zip(vec1, vec2))
    magnitude_vec1 = sqrt(sum(x**2 for x in vec1))
    magnitude_vec2 = sqrt(sum(x**2 for x in vec2))

    if magnitude_vec1 == 0 or magnitude_vec2 == 0:
        return 0

    return dot_product / (magnitude_vec1 * magnitude_vec2)

def calculate_unique_vector(docs):
    all_words = set(word for _, _, _, _, _, _, doc, _ in docs for word in doc)
    sorted_all_words = sorted(all_words)
    return list(sorted_all_words)

def word_count_table(stemmed_tokens):
    word_counts = Counter(stemmed_tokens)
    data = {'Word': [], 'Count': []}
    for word, count in word_counts.items():
        data['Word'].append(word)
        data['Count'].append(count)
    return data

if "button_clicked" not in st.session_state:
    st.session_state.button_clicked = False

def callback():
    st.session_state.button_clicked = True

def open_file(file_path):
    try:
        os.startfile(file_path)
        st.success('File successfully opened')
    except:
        st.warning('An error occurred while opening the file')

def main():
    st.header('Implementing the :orange[VSM] (Vector Space Model) Algorithm for Information
Retrieval', divider='orange')
```

```
    path_option = st.radio("Select Path Option:", ["Use Current Path", "Enter Path
Manually"])

    if path_option == "Use Current Path":
        directory_path = "file_test_dicky"
    else:
        directory_path = st.text_input("Enter the directory path:")

    if directory_path and os.path.exists(directory_path):
        stop_words = read_stop_words_sastrawi()
        dicti = read_dicti_sastrawi("kata-dasar.txt")

        st.subheader("List of Files in the Directory:")
        file_list = os.listdir(directory_path)
        st.text("\n".join(file_list))

        results = preprocess_directory(directory_path, stop_words, dicti)

        st.subheader("Results:")
        st.header('', divider='orange')
        for filename, content, original_lower, original_cleaned, original_filter,
original_tokens, stemmed_tokens, word_count in results:
            st.subheader(f"**File:** :orange[{filename}]")
            st.write("**Original Content:**", content)
            st.write("**After Case Folding:**", f":orange[{original_lower}]")
            st.write("**After Cleaned:**", f":orange[{original_cleaned}]")
            st.write("**After Tokenize:**", original_tokens)
            st.write("**After Filtered:**", original_filter)
            st.write("**After Stemmed:**", stemmed_tokens)
            st.write(f"**Total Words:** {word_count}")
            st.write("Word Count Table:")
            word_count_data = word_count_table(stemmed_tokens)
            st.table(word_count_data)

            st.header('', divider='orange')

        all_words = calculate_unique_vector(results)
        st.subheader("Vector Unique Word:")
        st.table([all_words])
        st.header('', divider='orange')

        st.subheader("Query:")
        user_query = st.text_input("Enter your query:")

        if st.button("Search", on_click=callback) or st.session_state.button_clicked:
            if user_query:
                similarity_scores = calculate_similarity(user_query, results, stop_words,
dicti, all_words)
                query_stem = preprocess_text(user_query, stop_words, dicti)
                st.write("**Original Content:**", user_query)
                st.write("**After Case Folding:**", f":orange[{query_stem[0]}]")
                st.write("**After Cleaned:**", f":orange[{query_stem[1]}]")
                st.write("**After Tokenize:**", query_stem[2])
                st.write("**After Filtered:**", query_stem[3])
                st.write("**After Stemmed:**", query_stem[4])
                st.write(f"**Total Words:** {len(set(query_stem[4]))}")
                st.write("Word Count Table:")
                query_count_data = word_count_table(query_stem[4])
                st.table(query_count_data)

                st.subheader("Search Results:")
                for (filename, _, _, _, _, _, _, _), score in sorted(zip(results,
similarity_scores), key=lambda x: x[1], reverse=True):
                    st.write(f"**Similarity Score** :orange[{filename}] :
:red[{score:.4f}]")
```

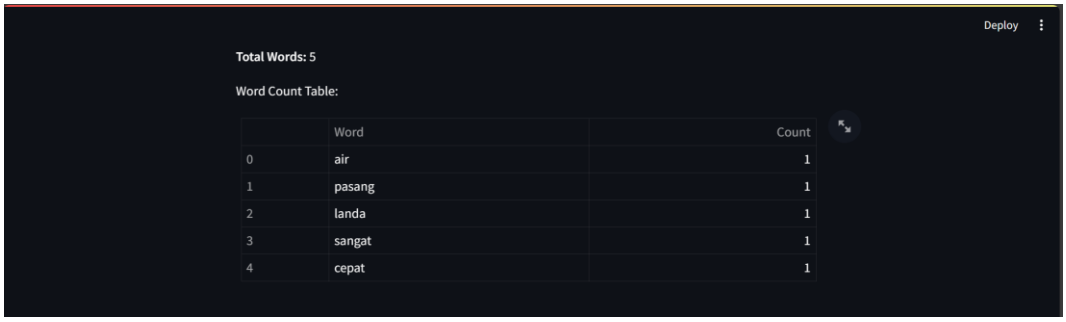
```
                    if st.button(f"Open :orange[{filename}]", key=str(filename)):
                        file_path = os.path.join(directory_path, filename)
                        open_file(file_path)
    elif directory_path:
        st.warning("Directory does not exist.")

    st.markdown("---")

if __name__ == "__main__":
    main()
```

4.3. Output

Gambar 4.2.1 Output List Directory

Gambar 4.3.1 Output Original Content dan preprocessing dokumen 1

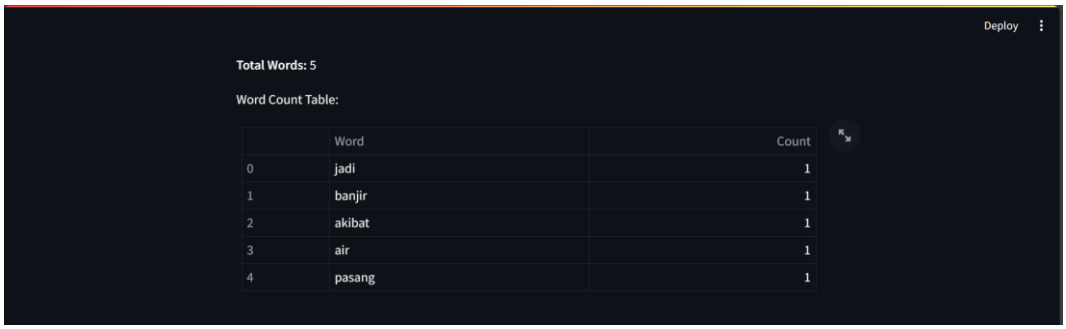
Gambar 4.3.2 Output After Stemmed dokumen 1



Gambar 4.3.3 Output Total Words dokumen 1

Gambar 4.3.4 Output Original Content dan preprocessing dokumen 2

Gambar 4.3.5 Output After Stemmed dokumen 2



Gambar 4.3.6 Output Total Words dokumen 2

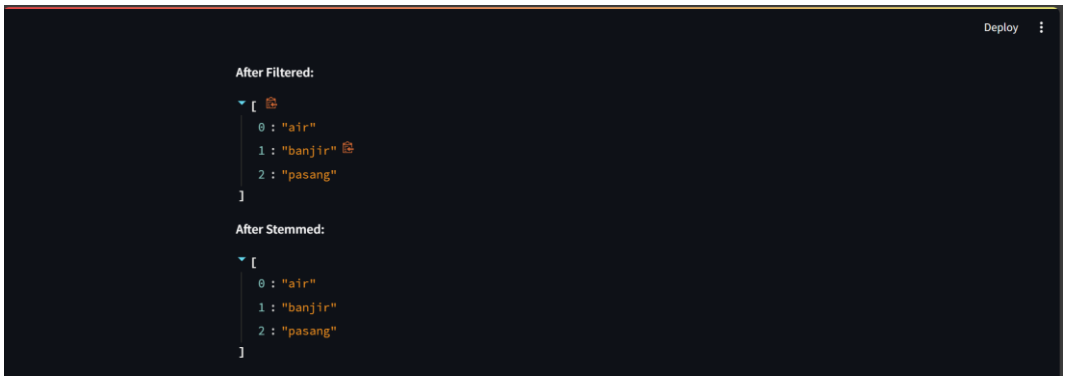
Gambar 4.3.7 Output Original Content dan Preprocessing dokumen 3

Gambar 4.3.8 Output After Stemmed dokumen 3

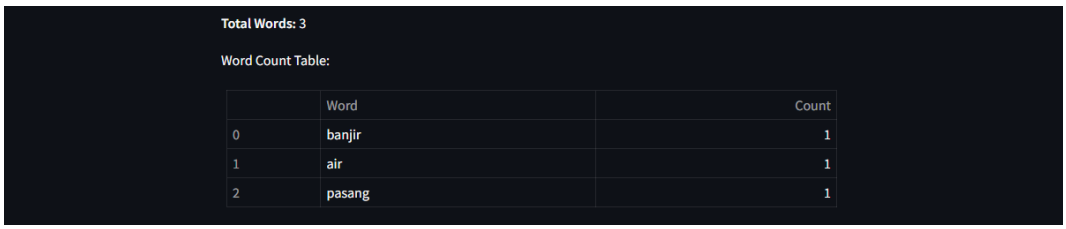
Gambar 4.3.9 Output Total Words dokumen 3

Gambar 4.3.10 Output Kata dalam vektor

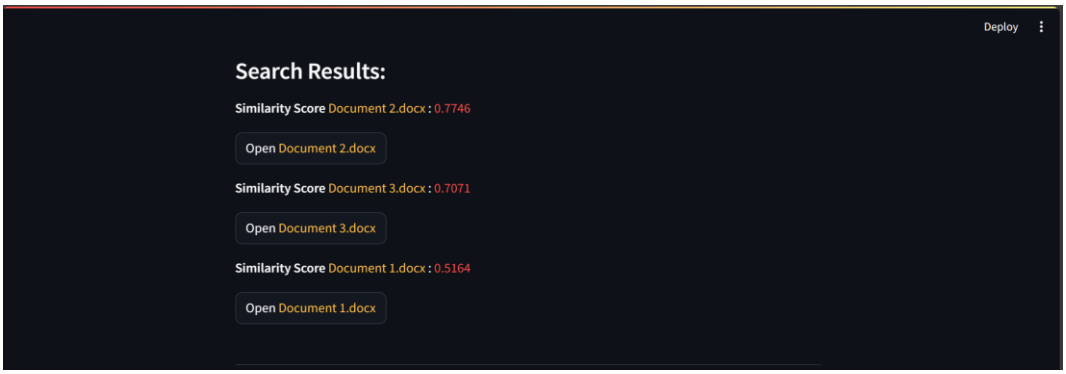
Gambar 4.3.11 Input dan Hasil Preprocessing Query



Gambar 4.3.12 Output After Filtered dan After Stemmed Query



Gambar 4.3.13 Output Jumlah kata unik dan frequensi kemunculan kata



Gambar 4.3.14 Hasil perhitungan similarity untuk setiap document

BAB V

PENUTUP

5.1. Kesimpulan

Kesimpulan dari laporan ini mengindikasikan bahwa penerapan Information Retrieval dengan memanfaatkan teknik Case Folding, Tokenizing, Filtering, dan Stemming menggunakan library Sastrawi memberikan hasil yang positif dan menguntungkan. Pertama, melalui implementasi Case Folding, konsistensi hasil pencarian ditingkatkan dengan menangani variasi huruf kapital, memastikan kata-kata dengan kapitalisasi yang berbeda tetap setara. Selanjutnya, penggunaan teknik Tokenizing berhasil mengoptimalkan proses pemrosesan teks dengan memecahnya menjadi unit-token, meningkatkan efisiensi dan kecepatan pencarian. Teknik Filtering, yang digunakan untuk menghilangkan kata-kata tidak penting, secara efektif meningkatkan relevansi hasil pencarian dengan memfokuskan pada konten yang benar-benar relevan. Selanjutnya, penerapan Stemming menggunakan library Sastrawi terbukti membawa manfaat dalam meningkatkan akurasi pencarian, mengatasi variasi kata dalam bahasa Indonesia. Akhirnya, integrasi komprehensif dari semua teknik preprocessing, yaitu Case Folding, Tokenizing, Filtering, dan Stemming, membentuk pendekatan yang sangat efektif untuk meningkatkan kualitas keseluruhan sistem Information Retrieval. Hasilnya menggambarkan potensi peningkatan signifikan dalam efektivitas dan efisiensi pencarian, khususnya dalam konteks bahasa Indonesia, dan menunjukkan bahwa pendekatan ini dapat diandalkan dalam memperoleh hasil pencarian yang lebih akurat dan relevan.

5.2. Saran

Untuk meningkatkan akurasi hasil pencarian, disarankan untuk memastikan bahwa query yang dimasukkan tidak mengandung kesalahan ketik atau typo. Cek kembali setiap kata dalam query agar sesuai dengan terminologi yang benar guna meminimalkan kemungkinan kegagalan pencarian. Dengan memperhatikan ketelitian dalam input query, pengguna dapat memastikan bahwa hasil yang diperoleh lebih sesuai dengan kebutuhan mereka. Kesalahan kecil dalam penulisan query dapat berdampak signifikan pada validitas dan relevansi hasil temu balik informasi.

DAFTAR PUSTAKA

Amin, F. (2012). Sistem Temu Kembali Informasi dengan Metode Vector Space Model. *Jurnal Sistem Informasi Bisnis*, 79-82.

Dewi, D. S. (2022). PENERAPAN ALGORITMA STEMMING SASTRAWI DAN COSINE SIMILARITY PADA INFORMATION RETRIEVAL SYSTEM AL-QURAN DENGAN QUERY TREN TWITTER. 8-20.

Irmawati. (2017). SISTEM TEMU KEMBALI INFORMASI PADA DOKUMEN DENGAN METODE VECTOR SPACE MODEL . *JURNAL ILMIAH FIFO*, 74-76.

Sucipto, A. (2021). Temu Kembali Informasi Menggunakan Metode Vector Space Model Pada Majalah Suara Muhammadiyah Periode 2010 - 2015. *Jurnal Teknik Elektro*, 103-106.