Laporan Proyek Mini Search Engine: Implementasi dan Evaluasi Model Boolean & Vector Space

Disusun Oleh :

Nama : Zekki Maulana Rahman

NIM : A11.2023.14989

FAKULTAS ILMU KOMPUTER

UNIVERITAS DIAN NUSWANTORO

2025

Pendahuluan

A. Tujuan Proyek

Tujuan utama dari proyek ini adalah untuk merancang, mengimplementasikan, dan mengevaluasi sistem temu kembali informasi (IR) skala kecil, atau "Mini Search Engine". Sistem ini dibangun untuk memenuhi beberapa tujuan utama:

1. Mengimplementasikan alur kerja *preprocessing* teks Bahasa Indonesia yang standar, mulai dari data mentah hingga corpus yang siap diolah.
2. Mengimplementasikan dua model retrieval fundamental: **Boolean Retrieval** untuk *query* eksak dan **Vector Space Model (VSM)** untuk *query* yang di-ranking.
3. Melakukan evaluasi kuantitatif terhadap performa model VSM dengan membandingkan dua skema *term-weighting* (TF-IDF) yang berbeda.
4. Menyajikan keseluruhan sistem dalam sebuah antarmuka web interaktif untuk demonstrasi fungsionalitas.

B. Ruang Lingkup Proyek

1. **Data:** Penggunaan sebuah corpus kustom (10 dokumen) berisi ulasan makanan dan kafe dalam Bahasa Indonesia.
2. **Preprocessing:** Implementasi *case folding*, penghapusan tanda baca, *tokenisasi*, *stopword removal* (menggunakan Sastrawi), dan *stemming* (menggunakan Sastrawi).
3. **Model Boolean:** Implementasi *retrieval* berbasis operator AND, OR, dan NOT.
4. **Model VSM:** Implementasi TF-IDF dengan dua varian skema TF:
   a. **Standard:** Frekuensi mentah (raw count).
   b. **Sublinear:** Frekuensi yang di-skala-kan secara logaritmik ($1 + \log(tf)$).
5. **Evaluasi:** Penggunaan truth_set (gold standard) untuk mengukur metrik performa VSM, termasuk Precision@k, Recall@k, F1-Score, nDCG@k, dan Mean Average Precision (MAP).
6. **Antarmuka:** Pembuatan aplikasi web menggunakan Streamlit.

C. Kontribusi Proyek terhadap Sub-CPMK

Proyek ini (yang merupakan *capstone* PjBL, sesuai pemetaan Soal 5) secara langsung berkontribusi pada pencapaian CPL10 melalui beberapa Sub-CPMK yang di-mapping dari materi 2, 3, 4, dan 5:

1. **Sub-CPMK 10.1.1 (Konsep Dasar IR):** Dicapai melalui perancangan arsitektur sistem secara keseluruhan dan pemahaman konseptual tentang cara kerja *search engine* (Materi 1).

2. **Sub-CPMK 10.1.2 (Document Preprocessing):** Dicapai secara penuh melalui implementasi preprocess.py (Soal 2, Materi 2). Ini mencakup perubahan data mentah (raw) menjadi token bersih (processed) melalui tahapan *cleaning, tokenizing, stopword removal,* dan *stemming*.
3. **Sub-CPMK 10.1.3 (Pemodelan IR):** Dicapai secara penuh melalui implementasi dua model fundamental (Soal 3, Materi 3): *Boolean Retrieval Model* (boolean_ir.py) dan *Vector Space Model* (vsm_ir.py).
4. **Sub-CPMK 10.1.4 (Term Weighting & Evaluasi):** Dicapai melalui implementasi *term weighting* (TF-IDF Standard vs Sublinear) (Soal 4, Materi 4) dan implementasi modul eval.py untuk mengukur metrik performa (Precision, Recall, MAP, nDCG) secara kuantitatif (Soal 5, Materi 5/6/7).

Data dan Preprocessing

A. Deskripsi Corpus

Corpus yang digunakan adalah kumpulan 10 dokumen teks (.txt), di mana setiap dokumen berisi ulasan singkat mengenai pengalaman di sebuah restoran atau kafe. Teks ulasan ini bersifat informal, mengandung bahasa sehari-hari, dan memiliki panjang yang bervariasi (rata-rata 35-40 term per dokumen setelah diproses).

B. Alur Preprocessing

Untuk mengubah data mentah (raw) menjadi data terstruktur yang siap diindeks, alur *preprocessing* (didefinisikan dalam preprocess.py) diterapkan pada setiap dokumen. Alur ini terdiri dari 4 langkah sekuensial:

1. **Cleaning (Pembersihan):** Melakukan *case folding* (mengubah semua teks menjadi huruf kecil) serta menghapus angka dan seluruh tanda baca (termasuk emoji).
2. **Tokenizing (Tokenisasi):** Memecah teks yang sudah bersih menjadi daftar kata (token) individual berdasarkan spasi.
3. **Stopword Removal (Filtrasi):** Menghapus kata-kata umum (stopwords) Bahasa Indonesia yang tidak membawa makna signifikan (mis. "di", "dan", "tapi") menggunakan daftar *stopwords* dari library Sastrawi.
4. **Stemming (Pencarian Kata Dasar):** Mengubah setiap token ke bentuk kata dasarnya (mis. "pelayanannya" -> "layan") menggunakan *stemmer* dari library Sastrawi.

C. Contoh Hasil Preprocessing

Berikut adalah contoh perbandingan *before/after* dari dua frasa yang diambil dari corpus:

Tahapan	**Contoh 1 (Raw)**	**Contoh 2 (Raw)**
Teks Asli	pelayanannya agak lama pas rame	minumnya agak kemanisan sih menurutku
1. Cleaning	pelayanannya agak lama pas rame	minumnya agak kemanisan sih menurutku

2. Tokenizing	['pelayanannya', 'agak', 'lama', 'pas', 'rame']	['minumnya', 'agak', 'kemanisan', 'sih', 'menurutku']
3. Stopword Removal	['pelayanannya', 'lama', 'pas', 'rame']	['minumnya', 'kemanisan', 'sih', 'menurutku']
4. Stemming	['layan', 'lama', 'pas', 'rame']	['minum', 'manis', 'sih', 'turut']
Hasil Akhir (Proses)	layan lama pas rame	minum manis sih turut

Hasil akhir dari tahap ini adalah 10 file .txt bersih di dalam folder data/processed/ yang menjadi dasar untuk *indexing* model.

Metode Information Retrieval

A. Boolean Retrieval

Model Boolean memproses *query* yang mengandung operator logika AND, OR, dan NOT. Model ini bekerja dengan memanipulasi himpunan (set) dokumen:

1. **Inverted Index:** Model ini bergantung pada *inverted index* (diimplementasikan dalam boolean_ir.py) yang memetakan setiap *term* dalam *vocabulary* ke daftar dokumen yang mengandung *term* tersebut (dikenal sebagai *postings list*).

2. **Operasi Logika:**
   - A AND B: Melakukan irisan (intersection) dari *postings list* A dan B.
   - A OR B: Melakukan gabungan (union) dari *postings list* A dan B.
   - NOT A: Mengambil himpunan semua dokumen, lalu dikurangi (difference) dengan *postings list* A.

3. **Hasil:** Hasil dari model ini bersifat biner; sebuah dokumen dianggap relevan (muncul di hasil) atau tidak sama sekali.

B. Vector Space Model (VSM)

VSM merepresentasikan dokumen dan *query* sebagai vektor dalam ruang multidimensi di mana setiap dimensi mewakili satu *term* dari *vocabulary*. Relevansi diukur berdasarkan kedekatan (proximity) antara vektor *query* dan vektor dokumen, biasanya menggunakan Cosine Similarity.

1. **Formula TF (Term Frequency)**
   TF mengukur seberapa sering sebuah *term* t muncul dalam dokumen d.
   Proyek ini mengimplementasikan dua skema TF (didefinisikan di vsm_ir.py):
   - **Standard (Raw Count):** Bobot TF adalah frekuensi mentah.
   - **Sublinear (Log-scaled):** Bobot TF di-skala-kan secara logaritmik untuk meredam efek *term* yang muncul sangat sering.

2. **Formula IDF (Inverse Document Frequency)**
   IDF mengukur seberapa penting sebuah *term* (t) di seluruh corpus. *Term* yang langka akan memiliki IDF tinggi, dan *term* yang umum (muncul di banyak dokumen) akan memiliki IDF rendah. (N) adalah jumlah total dokumen dan (df_t) adalah jumlah dokumen yang mengandung *term* (t).

3. **Formula Pembobotan TF-IDF**

   Bobot akhir w untuk *term* t dalam dokumen d adalah hasil perkalian dari bobot TF dan IDF-nya.

4. **Formula Cosine Similarity**

   Untuk me-ranking dokumen, kemiripan antara vektor *query* $\vec{q}$ dan vektor dokumen $\vec{d}$ dihitung menggunakan Cosine Similarity. Skor yang lebih tinggi (mendekati 1) menandakan relevansi yang lebih tinggi.

Arsitektur Search Engine

Sistem ini memiliki dua alur kerja utama: alur *offline* (Indexing) dan alur *online* (Querying), yang diatur oleh modul search_engine.py dan disajikan melalui main.py (Streamlit).

A. Alur Kerja Offline (Indexing)

Proses ini berjalan satu kali di awal untuk mempersiapkan sistem.

1. **Mulai:** Pengguna menjalankan sistem.
2. **Preprocessing:** Skrip preprocess.py membaca semua file dari data/raw/ dan menjalankan alur (Cleaning, Tokenizing, Stopword Removal, Stemming).
3. **Simpan Corpus Bersih:** Hasil disimpan di data/processed/.
4. **Indexing:** Modul vsm_ir.py dan boolean_ir.py di-load.
5. **Build Model:**
    - vsm_ir.py membaca corpus bersih untuk membangun **Vocabulary**, menghitung **DF** dan **IDF** untuk setiap *term*.
    - vsm_ir.py kemudian menghitung **Matriks TF-IDF** (satu untuk Standard, satu untuk Sublinear).
    - boolean_ir.py membangun **Inverted Index**.
6. **Selesai:** Sistem siap menerima *query*.

B. Alur Kerja Online (Querying via Streamlit)

Proses ini terjadi secara *real-time* saat pengguna berinteraksi dengan aplikasi Streamlit.

1. **Input Pengguna:** Pengguna memilih mode (mis. "VSM Search") dan memasukkan *query* (mis. "ayam enak sambal") di main.py.
2. **Query Preprocessing:** *Query* pengguna juga melalui **langkah *preprocessing* yang sama** (cleaning, tokenizing, stemming, dll) agar sesuai dengan *vocabulary*
3. **Kalkulasi Vektor Query:** Vektor TF-IDF untuk *query* dihitung berdasarkan *vocabulary* dan bobot IDF yang sudah ada.
4. **Kalkulasi Similarity:** Vektor *query* dibandingkan dengan **semua** vektor dokumen dalam Matriks TF-IDF menggunakan **Cosine Similarity**.

5. **Ranking:** Dokumen diurutkan (ranking) berdasarkan skor similaritasnya, dari yang tertinggi ke terendah.
6. **Tampilkan Hasil:** main.py (Streamlit) menampilkan daftar dokumen yang telah di-ranking (Top-K) kepada pengguna.

Eksperimen dan Evaluasi

A. Skenario Eksperimen

- **Tujuan:** Menentukan skema *weighting* TF-IDF mana yang menghasilkan ranking paling relevan untuk corpus ulasan makanan.
- **Truth Set:** Sebuah truth_set (gold standard) didefinisikan secara manual di dalam vsm_ir.py. truth_set ini berisi 6 *query* sampel dan daftar dokumen yang dianggap relevan secara manual untuk setiap *query* tersebut.
- **Proses:** Modul eval.py (dipanggil oleh Streamlit) menjalankan keenam *query* tersebut pada kedua skema (Standard dan Sublinear).
- **Parameter:** Evaluasi dilakukan dengan k=5 (Top-5 dokumen).

B. Metrik Evaluasi

1. **Precision@k (P@k):** Proporsi dokumen yang relevan di antara K dokumen teratas yang diambil.
2. **Recall@k (R@k):** Proporsi dokumen relevan yang berhasil diambil di dalam K dokumen teratas.
3. **F1-Score@k:** Rata-rata harmonik dari P@k dan R@k.
4. **nDCG@k (Normalized Discounted Cumulative Gain):** Mengukur kualitas ranking, di mana dokumen relevan yang muncul di posisi lebih atas mendapat skor lebih tinggi.
5. **MAP (Mean Average Precision):** Rata-rata dari Average Precision (AP) untuk semua *query*. Metrik ini memberikan gambaran umum performa *ranking* di seluruh set *query*.

C. Hasil Eksperimen

Hasil eksekusi perbandingan skema (dari eval.py dan tab "Compare Schemes" di Streamlit) dirangkum di bawah ini. Data direkonstruksi dari *screenshot* hasil, karena adanya *bug* visual pada kolom 'Standard'.

Metrik	Standard (TF Raw)	Sublinear (TF Log)	Delta	Improvement (%)
Mean P@5	0.0	0.4	0.4	0%*
Mean R@5	0.0	1.0	1.0	0%*

Mean F1@5	0.0	0.5516	0.5516	0%*
MAP@5	**0.9537**	**0.9722**	**0.0185**	**1.94%**
Mean nDCG@5	**0.9790**	**0.9892**	**0.0102**	**1.05%**

Peningkatan 0% pada P, R, dan F1 disebabkan oleh nilai 'Standard' yang nol, sehingga (Delta/Standard) tidak terdefinisi (division by zero) dan diatur ke 0.

D. Analisis Hasil

1. Observasi 1: Sublinear Unggul Secara Konsisten

- Berdasarkan tabel di atas, skema **Sublinear (TF Log)** menunjukkan performa yang **secara signifikan lebih baik** pada metrik-metrik utama yang mengukur kualitas *ranking*, yaitu **MAP** dan **nDCG@k**.
- Skema Sublinear menghasilkan nilai **MAP@5 sebesar 0.9722**, yang **1.94% lebih tinggi** dibandingkan skema Standard (MAP: 0.9537).
- Peningkatan juga terlihat pada **Mean nDCG@5 (naik 1.05%)**, yang menunjukkan bahwa skema Sublinear tidak hanya menemukan dokumen relevan, tetapi juga menempatkannya di *ranking* yang lebih atas.

2. Observasi 2: Kegagalan Skema Standard (pada P, R, F1)

- Hasil P@5, R@5, dan F1@5 untuk skema Standard adalah 0.0. Ini menunjukkan bahwa untuk *query* dalam truth_set, skema Standard (raw count) **gagal total** menempatkan *satu pun* dokumen relevan di 5 besar.
- **Interpretasi:** Ini adalah temuan yang sangat kuat. Ini mengindikasikan bahwa corpus ulasan ini kemungkinan memiliki *term* yang sangat dominan (mis. "enak" atau "ayam"). Skema Standard (raw count) memberikan bobot yang **terlalu besar** pada *term* ini, menyebabkan dokumen yang tidak relevan namun mengandung banyak *term* umum menjadi *over-weighted* dan masuk ke 5 besar.
- **Kesimpulan:** Skema Sublinear ($1 + \log(tf)$) **berhasil** meredam (dampen) efek negatif dari *term* yang sangat frekuen ini. Dengan "menghukum" *term* yang muncul berlebihan, skema Sublinear memberikan kesempatan bagi *term* yang lebih langka (namun lebih spesifik dan relevan, mis. "kremes" atau "lumer") untuk berkontribusi pada skor, sehingga berhasil menaikkan dokumen yang benar-benar relevan ke *ranking* atas.

Diskusi

A. Kelebihan

1. **Sistem Lengkap:** Proyek berhasil mengimplementasikan alur kerja IR dari awal (data mentah) hingga akhir (antarmuka web fungsional).
2. **Evaluasi Objektif:** Adanya modul eval.py dan truth_set memungkinkan perbandingan performa yang kuantitatif, bukan hanya "terlihat bagus".
3. **Fleksibilitas Model:** Menyediakan dua model (Boolean dan VSM) melayani dua kasus penggunaan yang berbeda (pencarian eksak vs. pencarian relevansi).

B. Keterbatasan:

1. **Ukuran Corpus:** Corpus yang hanya terdiri dari 10 dokumen sangat kecil. Hal ini menyebabkan nilai IDF mungkin tidak terlalu bermakna dan hasil evaluasi sangat sensitif terhadap truth_set yang juga kecil.
2. **Ketergantungan Sastrawi:** Kualitas *preprocessing* sangat bergantung pada *stemmer* Sastrawi. Seperti terlihat pada contoh ("menurutku" -> "turut"), *stemmer* tidak selalu sempurna dan kesalahan ini akan terbawa ke *indexing*.
3. **Subjektivitas truth_set:** truth_set dibuat secara manual dan bersifat subjektif. Performa sistem diukur berdasarkan relevansi menurut pembuat truth_set saja.

C. Saran Pengembangan

1. **Corpus yang Lebih Besar:** Menggunakan corpus yang jauh lebih besar (ratusan atau ribuan dokumen) untuk mendapatkan statistik (TF, IDF) yang lebih stabil dan hasil evaluasi yang lebih valid.
2. **Model Lanjutan:** Mengimplementasikan model *weighting* yang lebih canggih seperti **BM25 (Okapi-BM25)**, yang seringkali mengungguli TF-IDF standar.
3. **Query Expansion:** Menambahkan fitur *query expansion* (mis. menggunakan sinonim) untuk menangani *query* yang tidak memiliki kata yang sama persis dengan dokumen (mis. *query* "lezat" tetap bisa menemukan dokumen "enak").
4. **Perbaikan Preprocessing:** Membuat daftar *stopwords* kustom atau memperbaiki *output stemmer* secara manual untuk *term* yang sering salah.