Tosan Iqbal Kurniawan
A11.2023.15409
UTS STKI

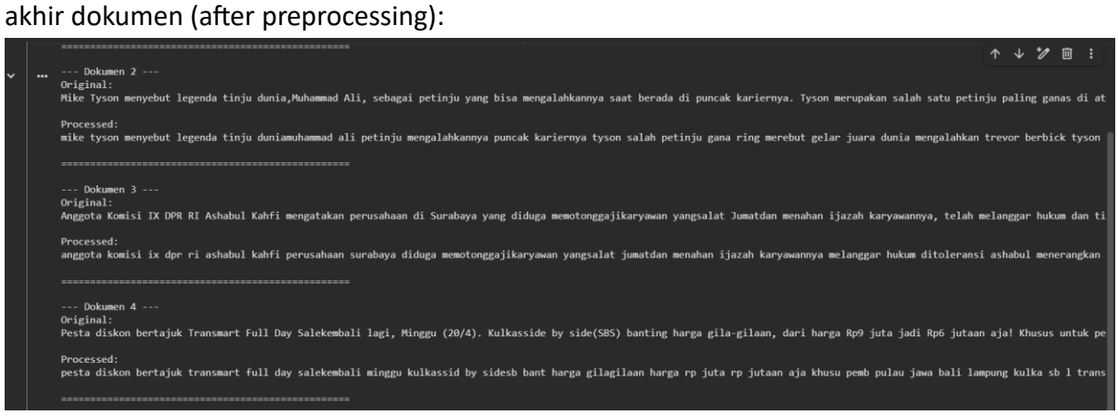
Pendahuluan

Proyek ini bertujuan membangun sebuah mini project sistem temu kembali informasi (STKI). Saya memfokuskan untuk mencari dan mengklasifikasi berita dari dataset berita CNN. STKI sendiri merupakan sistem yang dirancang khusus untuk menangani data tidak terstruktur seperti teks berita. Dengan memproses koleksi dokumen dan mengembalikannya berdasarkan query pengguna. Elemen kunci dalam sistem ini adalah penggunaan indeks dan mekanisme ranking untuk mengurutkan dokumen yang paling relevan.

Untuk mengimplementasikannya, proyek ini menggunakan dua pendekatan utama: Boolean Retrieval Model dan Vector Space Model (VSM). Model Boolean bekerja dengan mencocokkan kata kunci menggunakan operator logika (AND, OR, NOT), sedangkan VSM menggunakan representasi vektor untuk menghitung kesamaan (similarity) antara query dan dokumen. Seluruh alur kerja sistem, mulai dari preprocessing data hingga tahap ranking dan presentasi hasil, akan divisualisasikan menggunakan diagram arsitektur search engine klasik.

Data & Preprocessing

Untuk tahap preprocessing, saya menggunakan data berupa kumpulan berita yang diambil dari CNN. Sebanyak 9 dokumen dipilih untuk diproses dalam proyek ini. Setiap dokumen berita berisi teks yang cukup panjang dengan berbagai informasi terkait topik terkini. Dokumen-dokumen ini kemudian diubah ke dalam format .txt untuk mempermudah pemrosesan lebih lanjut. Pada langkah pertama dalam preprocessing, dilakukan tokenisasi untuk memecah setiap dokumen menjadi token-token yang lebih kecil, seperti kata atau frasa. Langkah selanjutnya adalah case-folding, yang dilakukan untuk mengubah seluruh teks menjadi huruf kecil. Hal ini bertujuan untuk menghindari perbedaan antara kata-kata yang memiliki kapitalisasi yang berbeda. Setelah case-folding, langkah berikutnya adalah stopword removal, di mana kata-kata yang tidak membawa makna signifikan dalam pencarian, seperti “dan”, “di”, “yang”, dihapus. Setelah menghapus stopwords, dilakukan stemming untuk mengubah kata-kata turunan menjadi bentuk dasar. Langkah terakhir dalam preprocessing adalah normalisasi angka dan tanda baca. Semua angka dan tanda baca yang tidak relevan dihapus atau diganti untuk memudahkan analisis lebih lanjut. Setelah kesembilan dokumen tersebut melalui kelima tahapan preprocessing, data teks mentah telah berhasil ditransformasi menjadi kumpulan token yang bersih, seragam, dan siap untuk dianalisis lebih lanjut. Untuk menunjukkan secara konkret efektivitas dari proses ini, berikut disajikan perbandingan antara kondisi dokumen asli (before preprocessing) dengan hasil akhir dokumen (after preprocessing):



Boolean Retrieval Model

Saya mengimplementasikan Boolean Retrieval Model dengan memproses query menggunakan operator logika AND, OR, dan NOT. Langkah fundamental untuk model ini adalah membangun Vocabulary, yaitu kamus yang berisi semua kata unik dari 9 dokumen berita yang telah diproses. Dari *vocabulary* tersebut, saya membangun dua struktur data inti:

1. Incidence Matrix: Sebuah matriks biner (sparse) yang menunjukkan ada (1) atau tidaknya (0) sebuah kata pada setiap dokumen.
2. Inverted Index: Struktur data yang lebih efisien yang memetakan setiap kata (*term*) ke daftar dokumen yang mengandung kata tersebut.

Berikut adalah cuplikan dari *Vocabulary* (10 kata pertama), *Incidence Matrix* (untuk beberapa dokumen dan kata pertama), dan *Inverted Index* (5 *term* pertama) yang berhasil dibangun dari korpus berita CNN:

```
Vocabulary (10 kata pertama): ['abdul', 'advertis', 'agam', 'agama', 'agamanya', 'aguero', 'agung', 'aja', 'akar', 'akibat']
Incidence Matrix (untuk beberapa dokumen dan kata pertama):
[[1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
Inverted Index (5 term pertama): {'abdul': [1], 'advertis': [1, 2, 3, 4, 5, 6, 7, 9], 'agam': [1], 'agama': [3, 7], 'agamanya': [3]}
```

Selain itu, saya juga menguji dan mendemonstrasikan model pencarin Boolean serta melakukan uji wajib menghitung precision/recall pada Boolean result set dengan gold set dengan hasil berikut:

```
    --- Query 3: dpr ri gaji karyawan ---
    Query diproses: ['dpr', 'ri', 'gaji', 'karyawan']
    Ditemukan di dokumen ID: [3]
      Cuplikan Dokumen 3: anggota komisi ix dpr ri ashabul kahfi perusahaan surabaya diduga memotonggajikaryawan yangsalat jumatdan menahan ijazah karyawannya mela

    --- Query 4: transmart diskon sepeda ---
    Query diproses: ['transmart', 'diskon', 'sepeda']
    Ditemukan di dokumen ID: [6]
      Cuplikan Dokumen 6: transmart full day salebalik kasih diskon gede pelanggan berbelanja gelaran bawa pulang sepeda diobral rp jutaan aneka sepeda sepeda gunu

    --- Query 5: liverpool gelar liga ---
    Query diproses: ['liverpool', 'gelar', 'liga']
    Ditemukan di dokumen ID: [5, 9]
      Cuplikan Dokumen 5: moham salahberpeluang menciptakan rekor unik saatliverpoolmelawan leicest citi laga krusial perebutan gelar liga inggri stadion king powe
      Cuplikan Dokumen 9: arsen peluang merebut gelar juaraliga championsmusim berat mengejar liverpool liga inggri rotasi pemain arsen arsen tertingg angka liverp
```

```
--- Evaluating Boolean Retrieval Model Performance ---

--- Scenario 1: Query = 'Transmart AND diskon' ---
  Gold Relevant Docs (IDs): [3, 6]
  Retrieved Docs (IDs): [4, 6]
  Precision: 0.5000
  Recall: 0.5000

--- Scenario 2: Query = 'Liverpool OR tinju' ---
  Gold Relevant Docs (IDs): [2, 5, 7]
  Retrieved Docs (IDs): [2, 5, 9]
  Precision: 0.6667
  Recall: 0.6667

--- Scenario 3: Query = 'korupsi NOT Surabaya' ---
  Gold Relevant Docs (IDs): [1, 4]
  Retrieved Docs (IDs): [1, 7]
  Precision: 0.5000
  Recall: 0.5000
```

Vector Space Model (VSM)

Pada bagian ini, kami mengimplementasikan Vector Space Model (VSM) untuk perankingan dokumen berdasarkan kesamaan antara query dan dokumen. VSM merepresentasikan dokumen dan query sebagai vektor, di mana setiap elemen vektor mewakili suatu kata dalam koleksi dokumen. Untuk menghitung Term Frequency (TF), kami pertama-tama menggunakan CountVectorizer untuk menghitung frekuensi kemunculan setiap kata dalam dokumen. Selanjutnya, untuk menghitung TF-IDF (Term Frequency-Inverse Document Frequency), kami menggunakan TfidfVectorizer, yang memperhitungkan bobot kata berdasarkan frekuensinya dalam dokumen dan seberapa jarang kata tersebut muncul di seluruh koleksi dokumen. Proses ini memungkinkan model untuk menilai kata-kata yang lebih relevan dalam konteks pencarian, dengan memberi bobot lebih pada kata-kata yang sering muncul dalam dokumen tetapi jarang muncul di dokumen lainnya. Dengan menggunakan TF-IDF, model dapat memberikan ranking yang lebih akurat berdasarkan kesamaan antara query dan dokumen.

Sebagai langkah awal, CountVectorizer menghasilkan matriks Term Frequency (TF) yang berisi frekuensi mentah (raw count) dari setiap kata dalam setiap dokumen. Berikut adalah cuplikan dari matriks TF untuk 5 dokumen pertama dan 10 *term* pertama:

```
--- Term Frequency (TF) Matrix (Raw Counts - Cuplikan) ---
       abdul  advertis  agam  agama  agamanya  aguero  agung  aja  akar  akibat
Doc 1      1         1     1      0         0       0      2    0     0       0
Doc 2      0         1     0      0         0       0      0    0     0       0
Doc 3      0         1     0      1         2       0      0    0     0       0
Doc 4      0         1     0      0         0       0      0    1     0       0
Doc 5      0         1     0      0         0       1      0    0     0       0

Penjelasan: Setiap angka dalam tabel menunjukkan berapa kali suatu 'term' muncul di 'dokumen' tersebut.
```

Setelah mendapatkan frekuensi mentah (TF), langkah selanjutnya adalah menghitung Inverse Document Frequency (IDF). IDF memberikan bobot lebih tinggi pada kata-kata yang unik (jarang muncul di seluruh dokumen) dan bobot rendah pada kata-kata yang umum. IDF dihitung dari Document Frequency (DF), yaitu jumlah dokumen di mana sebuah term muncul.
Untuk menganalisis korpus saya, saya mengidentifikasi term yang paling umum (DF tinggi) dan yang paling unik (IDF tinggi). Seperti yang terlihat di bawah, term seperti 'advertis' dan 'scroll' sangat umum (muncul di 8 dari 9 dokumen), sementara term seperti 'zona' dan 'abdul' sangat unik (IDF tinggi).

```
--- Top 10 Terms by IDF (paling unik/jarang) ---
       Term    IDF
       zona 2.6094
      abdul 2.6094
  zinchenko 2.6094
       agam 2.6094
     virgil 2.6094
      villa 2.6094
        van 2.6094
      utama 2.6094
      uskup 2.6094
      usaha 2.6094

--- Top 10 Terms by DF (paling sering muncul di dokumen berbeda) ---
       Term  DF
     scroll   8
    advertis   8
    content   8
    continu   8
     minggu   5
      orang   5
  memiliki   4
         rp   4
     pidana   3
      khusu   3
```

Tabel berikut memberikan ringkasan lebih rinci dari nilai DF dan IDF untuk beberapa term yang dihitung dari korpus saya:

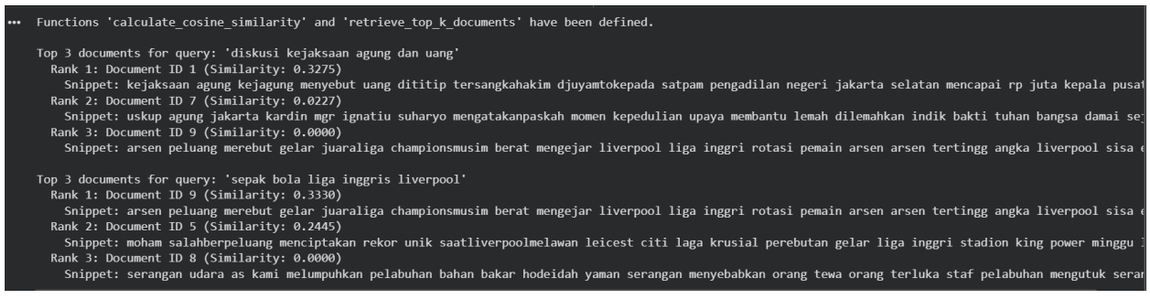
```
--- Ringkasan DF dan IDF untuk Beberapa Term (Tabel) ---
       Term  DF    IDF
    advertis   8 1.1054
     content   8 1.1054
     continu   8 1.1054
      scroll   8 1.1054
      minggu   5 1.5108
       orang   5 1.5108
   memiliki   4 1.6931
         rp   4 1.6931
       berat   3 1.9163
     dikutip   3 1.9163
       gelar   3 1.9163
       juara   3 1.9163
        juta   3 1.9163
       kalah   3 1.9163
       khusu   3 1.9163
```

Terakhir, kami menggunakan TfidfVectorizer untuk menggabungkan nilai TF dan IDF menjadi satu matriks pembobotan TF-IDF. Matriks inilah yang menjadi representasi VSM untuk korpus 9 berita CNN kami. Ukuran (shape) dari matriks *sparse* yang dihasilkan menunjukkan bahwa korpus kami terdiri dari 9 dokumen dan 1015 *term* unik (kata):

```
TF-IDF Sparse Matrix Shape: (9, 1015)
```

Dengan matriks TF-IDF ini, sistem siap menerima *query*, mengubah *query* tersebut menjadi vektor TF-IDF, dan menghitung Cosine Similarity untuk menemukan dokumen yang paling relevan.

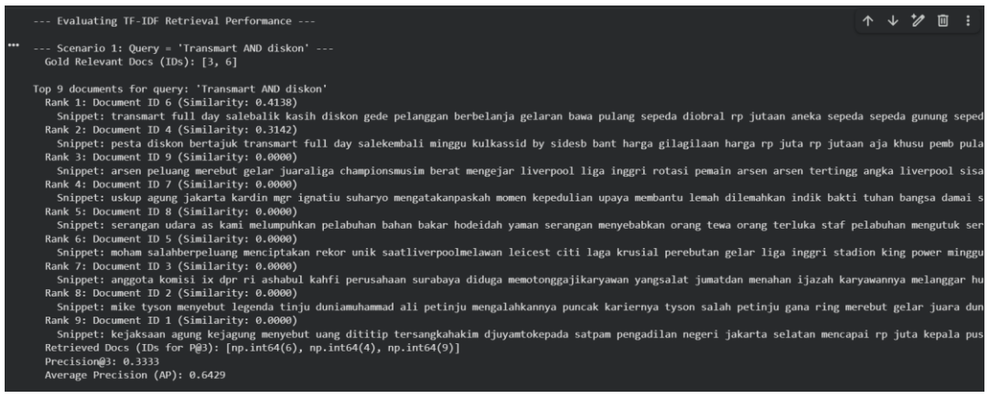
Setelah matriks TF-IDF siap, kami menghitung Cosine Similarity antara vektor *query* dan vektor dokumen untuk mendapatkan *ranking* relevansi. Berikut adalah hasil retrieve top k_documents (dengan k=3) untuk dua query pengujian.



Hasilnya menunjukkan:

- Untuk *query* 'diskusi kejaksaan agung dan uang', Dokumen 1 adalah yang paling relevan (Skor: 0.3275).
- Untuk *query* 'sepak bola liga inggris liverpool', Dokumen 9 adalah yang paling relevan (Skor: 0.3380).
- Skor 0.0000 berarti tidak ada *term* relevan yang ditemukan antara *query* dan dokumen tersebut.

Saya juga membandingkan VSM dengan Gold Set yang saya buat dengan hasil seperti berikut:



Analisis Singkat:

- Gold Set untuk query ini adalah dokumen [3, 6].
- Sistem saya berhasil menempatkan Dokumen 6 di Peringkat 1.
- Karena hanya 1 dari 3 dokumen teratas yang dikembalikan ([6, 4, 9]) yang ada di Gold Set, Precision@3 adalah 0.3333.
- Average Precision (AP) untuk query ini adalah 0.6429.

DISKUSI

Disini saya ingin membahas refleksi atas keseluruhan proyek, mengidentifikasi kelebihan dan keterbatasan dari implementasi yang telah dilakukan, serta memberikan saran untuk pengembangan di masa depan.

Kelebihan :

1. Implementasi Model Inti Berhasil: Proyek ini berhasil mengimplementasikan dua model STKI fundamental dari awal: Boolean Retrieval (dengan inverted index) dan Vector Space Model (dengan TF-IDF dan cosine similarity).
2. Preprocessing Efektif: Pipeline preprocessing (tokenisasi, case folding, stopword removal, stemming) berhasil diterapkan untuk membersihkan 9 dokumen berita CNN, yang terbukti dari vocabulary dan matriks TF-IDF yang dihasilkan.
3. Kemampuan Ranking: VSM yang dibangun terbukti mampu memberikan ranking relevansi (bukan hanya kecocokan Biner), yang divalidasi melalui skor cosine similarity dan evaluasi MAP.
4. Evaluasi Kuantitatif: Proyek ini berhasil dievaluasi secara kuantitatif menggunakan metrik standar (P@3, AP, MAP), memberikan baseline kinerja yang jelas.

Keterbatasan:

1. Skala Korpus Sangat Kecil: Keterbatasan terbesar adalah Saya hanya mengambil sedikit korpus yang terdiri dari 9 dokumen. Ukuran sekecil ini membuat statistik TF-IDF (terutama IDF) kurang andal dan membuat evaluasi menjadi sangat sensitif.
2. Subjektivitas Gold Set: Evaluasi sangat bergantung pada Gold Set yang dibuat secara manual. Pada korpus yang kecil, penentuan relevansi bisa menjadi sangat subjektif.
3. Query Parser Sederhana: Implementasi parser untuk query Boolean tidak ditunjukkan, dan query VSM (seperti 'Transmart AND diskon') masih diproses sebagai bag-of-words sederhana oleh TfidfVectorizer tanpa logika Boolean yang eksplisit.

Saran Pengembangan:
Berdasarkan keterbatasan yang telah diidentifikasi, berikut adalah beberapa saran pengembangan untuk meningkatkan sistem ini di masa depan:

1. Peningkatan Skala Korpus: Langkah paling krusial adalah memperbesar dataset. Menggunakan ratusan atau ribuan dokumen berita akan menghasilkan bobot TF-IDF yang jauh lebih bermakna dan hasil evaluasi yang lebih stabil.
2. Integrasi Modul (Orkestrasi): Dalam pembuatan search engine saya mengalami kesulitan sehingga tidak menambahkan kedalam mini project UTS.
3. Implementasi Skema Pembobotan Lain: Membandingkan kinerja TF-IDF dengan skema pembobotan yang lebih canggih seperti BM25 (Okapi), yang seringkali memberikan hasil lebih baik untuk koleksi dokumen standar.