Information Retrieval
Books Search Engine

William Manik[1], Jaden Panggabean[2], and Marthin Hutauruk[3] Information System, Del Institute of Technology, Indonesia
{iss21018, iss21038, iss21044}@students.del.ac.id

Abstrak

Fokus utama dari studi ini adalah untuk mengevaluasi dan mem- bandingkan empat teknik pencarian utama yaitu Boolean Search, Phrase Search, Proximity Search, dan Ranked Retrieval, serta dua metode per- ankingan, yaitu Jelinek-Mercer Smoothing dan SMART lnc.ltc. Melalui penerapan teknik-teknik ini, tujuan utama penelitian adalah untuk meningkatkan akurasi dan relevansi hasil pencarian, yang pada gilirannya dapat meningkatkan efektivitas sistem pencarian informasi. Penelitian ini mengimplemen- tasikan berbagai metode pemrosesan seperti tokenisasi, pengindeksan terbalik, dan penggunaan vektor embedding dalam pengambilan informasi semantik. Hasil dari penelitian ini menunjukkan bahwa teknik perankingan SMART lnc.ltc menghasilkan hasil yang lebih relevan dalam pencarian berbasis frekuensi kata, sedangkan Jelinek-Mercer Smoothing lebih efek- tif dalam mengatasi masalah distribusi kata yang jarang muncul dalam dokumen. Dengan menggunakan metode evaluasi berbasis TF-IDF dan cosine similarity, penelitian ini memberikan rekomendasi metode pencar- ian yang lebih sesuai dengan konteks dan kebutuhan aplikasi pencarian dokumen. Temuan ini bermanfaat bagi pengembangan sistem pencarian informasi yang lebih efisien dan efektif di masa depan.

Keywords: Information Retrieval, TF-IDF, Nearest Neighbors, To- kenisasi, Pencarian Semantik, Spell Checking, Dataset Buku, Evaluasi Kinerja, Pemrosesan Bahasa Alami.

1. Pendahuluan

Pencarian informasi merupakan proses penting dalam dunia digital saat ini, di mana data dan informasi tersebar luas di berbagai platform dan aplikasi. Dalam konteks ini, relevansi menjadi faktor yang sangat menentukan kualitas hasil pencarian. Relevansi mengacu pada sejauh mana hasil pencarian sesuai dengan kebutuhan atau niat pengguna. Tanpa relevansi yang tepat, hasil pencarian yang ditampilkan dapat menjadi tidak berguna, bertele-tele, atau bahkan menyesatkan, yang pada akhirnya menghambat pengambilan keputusan yang akurat dan efisien. Oleh karena itu, sistem pencarian yang mampu memberikan hasil yang relevan sangat dibutuhkan dalam meningkatkan kualitas pengambilan informasi.

Seiring dengan perkembangan teknologi, jumlah data yang tersedia juga semakin meningkat, sehingga penting untuk memiliki sistem pencarian yang mampu menyaring informasi yang tidak relevan dan menampilkan hasil yang paling sesuai dengan kebutuhan pengguna. Pengembangan algoritma pencarian yang efektif dan akurat menjadi salah satu langkah penting dalam meningkatkan kinerja sistem pencarian. Algoritma pencarian yang baik tidak hanya mengandalkan pencocokan kata kunci, tetapi juga harus mampu memahami konteks dan niat dari pengguna untuk menghasilkan hasil yang relevan dan bermanfaat.

Presentasi ini membahas tiga algoritma utama yang sering digunakan dalam meningkatkan relevansi pencarian informasi, yaitu Algoritma Rocchio, BM25, dan Learning to Rank (LTR). Setiap algoritma memiliki pendekatan dan keunggulannya masing-masing dalam menangani masalah relevansi pencarian. Algoritma Rocchio, sebagai algoritma klasik, menggunakan umpan balik dari pengguna untuk memodifikasi query pencarian berdasarkan dokumen yang relevan dan tidak relevan. Dengan demikian, sistem ini dapat belajar dan meningkatkan kualitas hasil pencarian melalui interaksi dengan pengguna.

Di sisi lain, BM25 merupakan pengembangan dari algoritma TF-IDF yang dirancang untuk

menangani masalah frekuensi term dan panjang dokumen dengan lebih baik. BM25 memberikan skor relevansi yang lebih akurat dengan menyesuaikan bobot kata kunci berdasarkan frekuensi kemunculannya dalam dokumen dan panjang dokumen tersebut. Algoritma ini banyak digunakan dalam mesin pencari untuk memberikan hasil pencarian yang lebih sesuai dengan niat pengguna. Sementara itu, Learning to Rank (LTR) menggunakan pendekatan machine learning untuk meningkatkan hasil pencarian. LTR memanfaatkan data dan fitur relevansi untuk melatih model yang dapat memprediksi peringkat hasil pencarian berdasarkan relevansi. Pendekatan ini lebih adaptif dan dapat disesuaikan dengan preferensi dan perilaku pengguna, sehingga memungkinkan hasil pencarian yang lebih personal dan optimal.

Dengan perkembangan algoritma-algoritma ini, sistem pencarian informasi dapat lebih efisien, akurat, dan relevan. Masing-masing algoritma memiliki keunggulannya tergantung pada konteks dan tujuan pencarian, dan pemilihan algoritma yang tepat akan sangat berpengaruh pada kualitas hasil pencarian yang diperoleh oleh pengguna. [1]

2. Studi Literatur

2.1 Information Retrieval

Information Retrieval (IR) adalah proses pencarian informasi yang relevan dari kumpulan data atau dokumen berdasarkan query atau permintaan yang dia- jukan oleh pengguna. Proses ini bertujuan untuk menemukan dokumen yang paling sesuai dengan kata kunci atau frasa yang dimasukkan oleh pengguna, yang pada akhirnya dapat memberikan informasi yang berguna sesuai dengan kebutuhan pencarian. Dalam konteks ini, IR tidak hanya terbatas pada pen- carian teks biasa, tetapi juga mencakup berbagai jenis data dan media, seperti gambar, video, atau data terstruktur lainnya.

Pencarian informasi ini banyak diterapkan dalam berbagai bidang dan ap- likasi teknologi yang kita gunakan sehari-hari. Salah satu contoh paling umum adalah mesin pencari (search engines) seperti Google, Bing, dan Yahoo, yang memungkinkan pengguna untuk menemukan informasi yang relevan di internet. Selain itu, IR juga digunakan dalam sistem rekomendasi, yang memberikan saran produk atau layanan kepada pengguna berdasarkan preferensi mereka, seperti yang ditemukan dalam platform e-commerce atau layanan streaming. Sistem IR juga diterapkan dalam analisis teks, seperti dalam penelitian ilmiah atau bisnis, di mana analisis terhadap data teks besar diperlukan untuk menarik kesimpulan atau menemukan pola tertentu.

Teknologi IR telah berkembang pesat, dengan teknik-teknik yang semakin canggih untuk meningkatkan relevansi dan keakuratan hasil pencarian. Misal- nya, metode peringkat dan teknik pencarian berbasis kata kunci yang lebih kompleks telah diterapkan untuk menyaring hasil yang paling relevan. Hal ini penting karena volume data yang terus berkembang membuat pencarian yang efektif menjadi lebih menantang. Oleh karena itu, pemahaman mendalam tentang berbagai teknik dan algoritma dalam IR sangat penting untuk men- goptimalkan hasil pencarian dan mempermudah pengguna dalam menemukan informasi yang mereka butuhkan dengan lebih cepat dan akurat.

.

2.2 Tokenisasi

Tokenisasi adalah proses yang digunakan untuk memecah teks menjadi unit- unit terkecil yang disebut token. Token ini bisa berupa kata, frasa, atau bahkan kalimat tergantung pada tujuan analisis. Proses tokenisasi adalah langkah per- tama yang penting dalam banyak aplikasi pemrosesan bahasa alami (NLP), seperti analisis teks, penerjemahan otomatis, dan pencarian informasi. Den- gan membagi teks menjadi unit-unit yang lebih kecil, sistem dapat lebih mudah memahami struktur dan makna dari teks tersebut. Misalnya, dalam tokenisasi berbasis kata, setiap kata dalam teks akan diperlakukan sebagai token terpisah, sementara dalam tokenisasi berbasis kalimat, teks akan dipecah menjadi be- berapa kalimat. Tokenisasi memungkinkan pemrosesan lebih lanjut, seperti penghapusan kata-kata yang tidak relevan (stopwords) atau pemrosesan seman- tik yang lebih kompleks, sehingga meningkatkan efektivitas model-model NLP dalam memahami dan menghasilkan bahasa. [2]

2.3 Information Retrieval Methods

Berbagai teknik dalam pengambilan informasi telah diuji coba dalam konteks RAG (Retrieval-Augmented Generation). Salah satunya adalah pencarian se- mantik, yang memanfaatkan representasi vektor (embedding) dari dokumen dan query untuk menilai kedekatan semantik antar keduanya. Dengan pendekatan ini, dokumen dapat diambil berdasarkan kesamaan makna, bukan hanya berdasarkan kecocokan kata kunci. Metode ini biasanya menggunakan model penyematan seperti BERT atau Sentence-BERT yang mampu memahami konteks serta hubun- gan semantik antar kata. Selain itu, ada juga pencarian kedekatan, yang men- gukur relevansi dokumen berdasarkan seberapa dekat kata-kata dalam query muncul bersama dalam teks. Asumsi dasarnya adalah jika kata-kata dalam query sering muncul berdekatan dalam dokumen, maka dokumen tersebut memi- liki relevansi yang lebih tinggi. Teknik ini efektif dalam mengidentifikasi bagian- bagian dokumen yang membahas topik-topik terkait dalam konteks yang serupa. Metode lainnya adalah ranked retrieval, yang mengandalkan algoritma per- ingkat seperti TF-IDF atau BM25 untuk menentukan relevansi dokumen. Al- goritma ini menilai dokumen berdasarkan frekuensi kemunculan kata dan sig- nifikansinya dalam keseluruhan koleksi dokumen, sehingga memungkinkan pen- gurutan dokumen sesuai dengan tingkat relevansinya terhadap query yang diberikan. [3]

2.4 BM25

BM25 adalah algoritma peringkat yang sangat populer dalam bidang Information Retrieval (IR). Algoritma ini merupakan salah satu metode yang digunakan untuk menghitung skor relevansi antara kueri dan dokumen dengan pendekatan probabilistik. BM25 merupakan pengembangan dari model Term Frequency-Inverse Document Frequency (TF-IDF), namun dengan beberapa penyesuaian untuk menangani masalah-masalah yang sering ditemukan pada pencarian informasi. Salah satu aspek utama dari BM25 adalah kemampuannya untuk menangani frekuensi term (TF) dan panjang dokumen yang bervariasi, serta inverse document frequency (IDF) yang menunjukkan seberapa penting sebuah term dalam keseluruhan koleksi dokumen.

BM25 bekerja dengan memberikan bobot yang lebih tinggi pada term yang sering muncul dalam dokumen, namun tidak bersifat linear—artinya, semakin sering sebuah term muncul, semakin kecil tambahan bobot yang diberikan. Selain itu, BM25 juga mengoreksi panjang dokumen, dengan memberikan normalisasi pada dokumen panjang agar tidak selalu dianggap lebih relevan hanya karena ukurannya yang lebih besar. Parameter utama yang digunakan dalam BM25 adalah k1, yang mengontrol pengaruh frekuensi term pada skor relevansi, dan b, yang digunakan untuk mengatur normalisasi panjang dokumen.

Keunggulan utama BM25 adalah kesederhanaannya dan efisiensinya dalam memberikan hasil pencarian yang relevan. Algoritma ini banyak digunakan dalam berbagai mesin pencari dan sistem informasi karena kemampuannya untuk menghitung skor relevansi yang cepat dan efektif. Namun, BM25 tidak memperhitungkan umpan balik pengguna atau preferensi khusus, yang mungkin membatasi efektivitasnya dalam pencarian yang lebih kompleks.[4]

2.5 Learning to Rank (LTR)

Learning to Rank (LTR) adalah pendekatan yang lebih canggih dalam sistem pencarian informasi yang menggunakan pembelajaran mesin untuk meningkatkan hasil pencarian. Berbeda dengan algoritma seperti BM25, yang mengandalkan fitur statistik seperti frekuensi term dan panjang dokumen, LTR memanfaatkan data historis dan teknik machine learning untuk melatih model yang dapat memprediksi relevansi dokumen terhadap kueri yang diberikan. Dalam LTR, berbagai fitur yang dihasilkan dari dokumen dan kueri, seperti TF-IDF, BM25, kecocokan judul, atau kecocokan penulis, digunakan sebagai input untuk model pembelajaran mesin.

Salah satu model yang paling umum digunakan dalam LTR adalah RandomForestRegressor, yang merupakan model berbasis pohon keputusan. Model ini dilatih untuk mempelajari hubungan antara fitur-fitur yang ada dan skor relevansi dokumen. Setelah model dilatih dengan data pelatihan, model ini dapat digunakan untuk memprediksi relevansi dokumen terhadap kueri baru dengan cara yang lebih adaptif dan dinamis. Keuntungan utama dari LTR adalah kemampuannya untuk menghasilkan model yang lebih akurat dan dapat menyesuaikan diri dengan preferensi pengguna atau pola-pola dalam data.

Namun, kelemahan dari LTR adalah bahwa ia membutuhkan data yang cukup besar dan waktu pelatihan yang cukup panjang untuk menghasilkan model yang akurat. Selain itu, LTR juga memerlukan pemeliharaan yang terus-menerus untuk menyesuaikan dengan perubahan dalam data atau preferensi pengguna yang mungkin terjadi seiring waktu.

2.6 Proximity Search

Proximity Search adalah teknik pencarian informasi yang mengidentifikasi kata kunci yang berada dalam jarak tertentu satu sama lain dalam sebuah doku- men. Dalam metode ini, pencarian tidak hanya mempertimbangkan keber- adaan kata kunci, tetapi juga memperhitungkan seberapa dekat posisi kata-kata tersebut satu sama lain. Pengguna dapat menentukan batasan jarak antara kata kunci yang relevan, sehingga memungkinkan sistem pencarian untuk men- emukan dokumen yang tidak hanya mengandung kata-kata tertentu, tetapi juga menyusun kata-kata tersebut dalam konteks yang lebih bermakna.

Keunggulan utama dari Proximity Search adalah kemampuannya untuk mengi- dentifikasi hubungan antara kata-kata yang muncul dekat satu sama lain, yang sering kali menunjukkan relevansi yang lebih tinggi dalam konteks pencarian. Teknik ini sangat berguna ketika kata-kata tertentu harus muncul bersama dalam sebuah kalimat atau paragraf untuk membentuk konsep yang lebih kom- pleks. Dengan menggunakan Proximity Search, sistem pencarian dapat lebih cerdas dalam mengidentifikasi konteks yang lebih dalam, yang pada akhirnya menghasilkan hasil pencarian yang lebih relevan dan bermakna.

Namun, Proximity Search juga memiliki beberapa keterbatasan. Salah sat- unya adalah ketergantungan pada jarak antara kata-kata. Jika jarak antara kata-kata dalam dokumen terlalu jauh atau terlalu dekat dari yang ditentukan dalam kueri, dokumen tersebut mungkin tidak akan muncul meskipun kata-kata yang relevan ada di dalamnya. Selain itu, pengaturan jarak yang terlalu ketat dapat menyebabkan pencarian menjadi terlalu selektif, sementara jarak yang terlalu longgar dapat mengurangi ketepatan hasil pencarian.

Meskipun demikian, Proximity Search adalah metode yang sangat efektif ketika pencarian memerlukan pemahaman konteks atau hubungan antar kata dalam dokumen. Teknik ini sering digunakan dalam aplikasi yang membutuhkan analisis teks yang lebih mendalam,

seperti dalam penelitian ilmiah atau anali- sis konten, di mana hubungan antara kata-kata dalam konteks tertentu sangat mempengaruhi relevansi dan makna informasi yang ditemukan. [2]

Algoritma **Rocchio** adalah metode yang lebih klasik dalam **Information Retrieval** yang memanfaatkan umpan balik relevansi untuk meningkatkan hasil pencarian. Dalam metode ini, setelah pencarian awal dilakukan, pengguna memberikan umpan balik mengenai dokumen yang relevan dan tidak relevan. Berdasarkan umpan balik tersebut, sistem kemudian memodifikasi kueri asli dengan cara mengubah vektor kueri untuk lebih mencocokkan dokumen yang relevan dan mengurangi pengaruh dokumen yang tidak relevan. Ini dilakukan dengan cara menambahkan vektor dokumen relevan dan mengurangi vektor dokumen yang tidak relevan dari vektor kueri asli.

Rumus matematis untuk memodifikasi kueri dalam algoritma Rocchio adalah:

$$\text{Qnew} = \alpha \cdot \text{Qoriginal} + \beta \cdot (|\ \text{Dr}\ |\ 1\text{dr} \in \text{Dr} \sum \text{dr}) - \gamma \cdot (|\ \text{Dnr}\ |\ 1\text{dnr} \in \text{Dnr} \sum \text{dnr})$$

Di mana QoriginalQ_{original}Qoriginal adalah vektor kueri asli, DrD_rDr adalah himpunan dokumen yang relevan, DnrD_{nr}Dnr adalah himpunan dokumen yang tidak relevan, dan α\alphaα, β\betaβ, serta γ\gammaγ adalah bobot yang mengatur pengaruh masing-masing komponen.

Keuntungan dari metode Rocchio adalah kemampuannya untuk mempersonalisasi hasil pencarian berdasarkan umpan balik pengguna. Hal ini memungkinkan sistem untuk memberikan hasil yang lebih relevan sesuai dengan preferensi pengguna. Namun, kelemahan metode ini adalah ketergantungannya pada umpan balik yang cukup dari pengguna untuk bekerja dengan baik. Selain itu, sistem ini mungkin tidak efisien untuk digunakan pada sistem pencarian yang tidak memiliki banyak interaksi pengguna atau pada kasus di mana umpan balik terbatas.

3. Metodologi

Penelitian ini menggunakan dataset buku yang berisi informasi terkait buku- buku yang tersedia di Amazon. Dataset ini mencakup beberapa atribut seperti ISBN, judul buku, pengarang, penerbit, dan deskripsi buku.

3.1 Pemuatan dan Konfigurasi Data

3.1.1 Pemuatan dan Konfigurasi Data BM25

Tahap fundamental dalam metodologi BM25, seperti yang terimplementasi dalam *file* BM25 (1).ipynb, dimulai dengan konfigurasi dan pemuatan data secara cermat. Kode program secara eksplisit mendefinisikan variabel CSV_FILE_PATH yang menjadi penunjuk akurat ke lokasi *file* data_books_updated.csv, yang merupakan repositori data buku. Selanjutnya, dilakukan penetapan variabel TEXT_COLUMNS, yang berisi daftar nama kolom krusial seperti 'Book-Title', 'Book-Author', 'Publisher', dan 'description'. Kolom-kolom inilah yang kontennya akan digabungkan untuk membentuk representasi tekstual dari setiap entitas buku. Tidak kalah penting, variabel DISPLAY_COLUMNS juga dikonfigurasi untuk menentukan informasi apa saja, misalnya 'Book-Title', 'Book-Author', dan 'Publisher', yang akan disajikan kepada pengguna sebagai output dari proses pencarian. Proses pemuatan data aktual dijalankan oleh fungsi load_data(filepath) yang memanfaatkan *library* pandas dengan perintah pd.read_csv(filepath). Sebagai bagian dari upaya menjaga kebersihan dan konsistensi data, kode program juga melakukan pemeriksaan terhadap keberadaan kolom 'Unnamed: 0', yang seringkali muncul sebagai artefak dari proses penyimpanan *file* CSV. Jika kolom tersebut ditemukan, maka akan dihapus melalui perintah df = df.drop(columns=['Unnamed: 0']). Langkah penanganan data selanjutnya adalah mengatasi nilai yang hilang atau NaN pada kolom-kolom teks yang telah ditentukan dalam TEXT_COLUMNS. Untuk setiap kolom tersebut, nilai NaN akan diisi dengan *string* kosong

menggunakan df[col] = df[col].fillna('').astype(str), sebuah tindakan preventif agar proses penggabungan teks dan tokenisasi di tahap berikutnya dapat berjalan lancar tanpa terinterupsi oleh data yang tidak lengkap.

3.1.2 Pemuatan dan Konfigurasi Data Learning to Rank (LTR)

Untuk model Learning to Rank (LTR) yang detail implementasinya terdapat dalam file Learning To Rank (1).ipynb, persiapan data awal memiliki beberapa kekhususan untuk mengakomodasi kebutuhan ekstraksi fitur yang lebih beragam dan kompleks. Kelas BookSearchEngine diinisialisasi dengan parameter csv_path yang menunjukkan lokasi file data buku. Di dalam metode load_data(self) pada kelas tersebut, pemuatan data dari file CSV dilakukan menggunakan pd.read_csv(self.csv_path). Sebuah langkah pembersihan data yang signifikan pada tahap ini adalah self.df = self.df.dropna(subset=['Book-Title', 'Book-Author', 'description']). Perintah ini akan menghapus setiap baris data yang tidak memiliki informasi lengkap pada salah satu dari kolom-kolom kunci tersebut ('Book-Title', 'Book-Author', atau 'description'). Tindakan eliminasi ini bertujuan untuk meningkatkan kualitas keseluruhan data yang akan digunakan sebagai dasar pelatihan model LTR, karena fitur-fitur yang akan diekstraksi nantinya diharapkan menjadi lebih representatif dan bermakna. Setelah pembersihan baris, konten dari kolom 'Book-Title', 'Book-Author', dan 'description' – setelah terlebih dahulu nilai NaN di dalamnya diisi dengan string kosong melalui metode .fillna('') – akan digabungkan menjadi satu kolom teks tunggal baru yang diberi nama self.df['combined_text']. Kolom combined_text inilah yang akan menjadi input utama untuk serangkaian proses pra-pemrosesan teks di tahap selanjutnya.

3.1.3 Pemuatan dan Konfigurasi Data Algoritma TF-IDF dengan Umpan Balik Relevansi Rocchio

Dalam implementasi algoritma Rocchio yang terdapat pada *file* Rocchio (Relevance Feedback) (2).ipynb, kelas RocchioInformationRetrieval juga memulai operasinya dengan menerima csv_path sebagai parameter inisialisasi. Metode load_data(self) di dalam kelas ini bertanggung jawab untuk memuat data dari *file* CSV menggunakan *library* pandas. Penanganan nilai kosong dilakukan secara spesifik pada kolom-kolom utama yaitu 'Book-Title', 'Book-Author', dan 'description', di mana setiap nilai NaN pada kolom-kolom tersebut akan digantikan dengan *string* kosong, misalnya melalui df['Book-Title'].fillna(''). Langkah ini memastikan bahwa semua dokumen nantinya memiliki representasi teks, meskipun beberapa atribut individualnya mungkin awalnya tidak terisi. Selanjutnya, dalam metode preprocess_documents(self), untuk setiap entri buku (baris data), konten teks dari row['Book-Title'], row['Book-Author'], dan row['description'] akan digabungkan menjadi satu *string* tunggal yang disebut combined_text. *String* gabungan inilah yang kemudian akan menjadi subjek dari tahap pra-pemrosesan teks lebih lanjut sebelum akhirnya ditransformasikan menjadi representasi vektor menggunakan skema TF-IDF. Dokumen Word yang Anda lampirkan menyebutkan penggunaan dataset buku yang mencakup atribut seperti ISBN, judul buku, pengarang, penerbit, dan deskripsi buku, yang sepenuhnya selaras dengan kolom-kolom data yang dimanfaatkan dan diproses dalam kode.

3.2 Pra-pemrosesan Teks dan Tokenisasi

3.2.1 Pra-pemrosesan Teks dan Tokenisasi BM25

Setelah data berhasil dimuat dan konten teks dari kolom-kolom yang relevan telah digabungkan menjadi satu *string* tunggal per dokumen (yang dilakukan oleh fungsi create_corpus_and_tokenize dalam *file* BM25 (1).ipynb), setiap *string* dokumen tersebut, dan nantinya juga kueri yang dimasukkan oleh pengguna, akan melewati serangkaian langkah pra-pemrosesan yang ditangani oleh fungsi preprocess_text(text). Langkah pertama adalah konversi seluruh teks menjadi huruf kecil melalui text = text.lower(). Standardisasi ini penting untuk memastikan konsistensi, sehingga kata yang sama namun memiliki kapitalisasi berbeda

(contohnya, "Search" dan "search") akan diperlakukan sebagai token yang identik oleh sistem. Langkah berikutnya adalah penghapusan karakter non-alfanumerik. Dengan menggunakan ekspresi reguler melalui perintah text = re.sub(r'\W+', ' ', text), semua urutan karakter yang bukan huruf atau angka (termasuk tanda baca, simbol, dan lain-lain) akan digantikan dengan satu karakter spasi. Proses ini bertujuan untuk membersihkan teks dari elemen-elemen yang umumnya dianggap sebagai *noise* dan tidak membawa makna signifikan untuk keperluan pencocokan kata kunci dalam konteks temu kembali informasi. Akhirnya, *string* teks yang telah melalui pembersihan tersebut dipecah menjadi unit-unit individual yang disebut token. Ini dilakukan dengan perintah tokens = text.split(), yang memisahkan *string* berdasarkan spasi menjadi sebuah daftar kata-kata. Hasil dari aplikasi proses ini ke seluruh koleksi dokumen adalah tokenized_corpus. Dokumen Word Anda secara akurat menjelaskan bahwa tokenisasi adalah proses fundamental yang digunakan untuk memecah teks menjadi unit-unit terkecil yang disebut token, yang bisa berupa kata, frasa, atau bahkan kalimat, dan merupakan langkah pertama yang penting dalam banyak aplikasi pemrosesan bahasa alami (NLP). Proses yang identik ini juga akan diterapkan pada kueri yang dimasukkan pengguna saat melakukan pencarian untuk memastikan bahwa perbandingan antara kueri dan dokumen dilakukan secara konsisten.

3.2.2 Pra-pemrosesan Teks dan Tokenisasi Learning to Rank (LTR)

Dalam implementasi Learning to Rank pada *file* Learning To Rank (1).ipynb, metode preprocess_text(self, text) yang terdapat di dalam kelas BookSearchEngine bertanggung jawab untuk melakukan pra-pemrosesan pada setiap item teks yang ada dalam kolom 'combined_text'. Langkah-langkah yang dilakukan meliputi: pertama, penanganan khusus untuk nilai NaN melalui if pd.isna(text): return "", yang memastikan bahwa input teks yang mungkin kosong akan menghasilkan *string* kosong sehingga tidak menyebabkan error pada proses selanjutnya. Kedua, konversi teks menjadi huruf kecil dilakukan dengan text = str(text).lower(), sebuah langkah standardisasi yang umum. Ketiga, penghapusan semua karakter tanda baca dari teks dilakukan secara efektif menggunakan text = text.translate(str.maketrans('', '', string.punctuation)). Keempat, untuk memastikan kerapian teks, spasi ganda atau spasi yang tidak perlu di awal atau akhir teks dihilangkan dengan text = ' '.join(text.split()). Hasil dari keseluruhan proses pra-pemrosesan ini kemudian disimpan sebagai self.processed_texts. Meskipun pada tahap ini tokenisasi eksplisit (pemecahan teks menjadi daftar token) tidak disimpan secara terpisah (karena TfidfVectorizer yang akan digunakan di tahap berikutnya akan menangani proses tokenisasi secara internal sesuai dengan konfigurasinya), langkah-langkah pembersihan teks ini merupakan bentuk pra-tokenisasi yang sangat krusial untuk menjamin kualitas input yang akan diberikan kepada *vectorizer*. Proses pembersihan dan persiapan teks ini sejalan dengan deskripsi dalam dokumen Word yang menyatakan bahwa tokenisasi memungkinkan dilakukannya pemrosesan lebih lanjut, seperti penghapusan kata-kata yang tidak relevan (*stopwords*) atau pemrosesan semantik yang lebih kompleks, sehingga meningkatkan efektivitas model-model NLP dalam memahami dan menghasilkan bahasa.

3.2.3 Pra-pemrosesan Teks dan Tokenisasi Algoritma TF-IDF dengan Umpan Balik Relevansi Rocchio

Implementasi algoritma Rocchio, seperti yang terdapat dalam *file* Rocchio (Relevance Feedback) (2).ipynb, juga memanfaatkan metode preprocess_text(self, text) dengan fungsionalitas yang serupa untuk pra-pemrosesan teks. Metode ini melakukan beberapa langkah penting: pertama, ia menangani kasus di mana input teks mungkin berupa NaN atau *string* kosong dengan if pd.isna(text) or text == '': return ''. Kedua, seluruh teks dikonversi menjadi format huruf kecil melalui text = str(text).lower(). Ketiga, semua karakter tanda baca dihilangkan dari teks menggunakan text = text.translate(str.maketrans('', '', string.punctuation)). Keempat, spasi putih yang berlebih atau tidak perlu dihilangkan dengan text = ' '.join(text.split()). Metode preprocess_documents(self) kemudian menerapkan fungsi preprocess_text ini ke setiap dokumen

yang teksnya telah digabungkan dari beberapa kolom pada tahap pemuatan data. Teks yang telah bersih dan terstandardisasi ini menjadi input langsung untuk TfidfVectorizer pada tahap selanjutnya. Serupa dengan pendekatan LTR, TfidfVectorizer akan melakukan proses tokenisasi internalnya sendiri. Dokumen Word yang Anda sediakan secara konsisten menekankan bahwa tokenisasi adalah langkah awal yang penting untuk memecah teks menjadi unit-unit yang lebih kecil dan dapat dikelola, yang selanjutnya siap untuk dianalisis lebih lanjut. Kueri yang dimasukkan oleh pengguna juga akan melewati fungsi preprocess_text yang identik untuk memastikan konsistensi dalam representasi teks antara kueri dan dokumen.\

3.3 Pembangunan Model dan Ekstraksi Fitur

3.3.1 Model BM25 (Okapi BM25)

Setelah tokenized_corpus (yang merupakan hasil dari keseluruhan proses pra-pemrosesan dan tokenisasi untuk semua dokumen dalam koleksi) berhasil dibuat, *file* BM25 (1).ipynb melanjutkan ke tahap pembangunan model BM25. Ini dilakukan dengan menginisialisasi objek dari kelas BM25Okapi yang berasal dari *library* rank_bm25. Inisialisasi model ini secara spesifik dilakukan dengan perintah bm25 = BM25Okapi(tokenized_corpus) yang terdapat di dalam fungsi build_bm25_model. Pada saat inisialisasi ini, model BM25Okapi akan secara internal melakukan kalkulasi dan penyimpanan berbagai statistik penting yang diperlukan dari korpus yang diberikan. Statistik tersebut meliputi frekuensi kemunculan setiap term dalam setiap dokumen individual, panjang total dari setiap dokumen (jumlah token), jumlah total dokumen dalam korpus, dan frekuensi dokumen untuk setiap term unik (yaitu, dalam berapa banyak dokumen sebuah term muncul). Semua statistik ini merupakan komponen fundamental yang akan digunakan dalam formula perhitungan skor BM25 untuk menentukan tingkat relevansi antara sebuah kueri dan setiap dokumen dalam koleksi. Deskripsi dalam dokumen Word Anda mengenai BM25 selaras dengan ini, yang menyatakan bahwa BM25 adalah algoritma peringkat yang sangat populer dalam bidang Information Retrieval (IR) dan merupakan pengembangan dari model Term Frequency-Inverse Document Frequency (TF-IDF) dengan pendekatan probabilistik. Dokumen tersebut juga menyoroti kemampuan BM25 dalam menangani frekuensi term (TF), panjang dokumen yang bervariasi, serta *inverse document frequency* (IDF).

3.3.2 Model Learning to Rank (LTR) dengan Fitur Gabungan

Dalam alur kerja model Learning to Rank yang diimplementasikan pada *file* Learning To Rank (1).ipynb, setelah self.processed_texts (kumpulan teks dokumen yang telah dipra-pemroses) siap, langkah krusial pertama adalah pembangunan indeks Term Frequency-Inverse Document Frequency (TF-IDF). Ini diawali dengan inisialisasi objek TfidfVectorizer melalui self.tfidf_vectorizer = TfidfVectorizer(max_features=10000, stop_words='english', ngram_range=(1, 2)). Parameter max_features=10000 berfungsi untuk membatasi jumlah fitur (kata atau n-gram) yang akan dipertimbangkan hanya pada 10.000 fitur yang memiliki frekuensi kemunculan tertinggi di seluruh korpus. Parameter stop_words='english' menginstruksikan *vectorizer* untuk mengabaikan kata-kata umum dalam bahasa Inggris (seperti "the", "is", "in") yang cenderung tidak membawa makna diskriminatif. Pengaturan ngram_range=(1, 2) menandakan bahwa baik kata tunggal (unigram) maupun pasangan kata yang muncul berurutan (bigram) akan diekstraksi dan dipertimbangkan sebagai fitur potensial. Matriks TF-IDF kemudian dibangun dengan menjalankan perintah self.tfidf_matrix = self.tfidf_vectorizer.fit_transform(self.processed_texts). Selanjutnya, dalam metode extract_features(self, query), serangkaian fitur yang beragam diekstraksi untuk setiap dokumen dalam koleksi relatif terhadap kueri yang diberikan. Proses ini penting karena LTR, seperti yang disebutkan dalam dokumen Word, memanfaatkan berbagai fitur yang dihasilkan dari dokumen dan kueri. Fitur-fitur tersebut meliputi:

- **Skor TF-IDF**: Kueri yang telah melalui pra-pemrosesan (query_processed) pertama-

tama diubah menjadi representasi vektor TF-IDF menggunakan tfidf_vectorizer yang telah di-*fit* sebelumnya (query_tfidf = self.tfidf_vectorizer.transform([query_processed])). Setelah itu, skor kemiripan kosinus dihitung antara vektor TF-IDF kueri ini dan setiap vektor dokumen dalam self.tfidf_matrix menggunakan tfidf_scores = cosine_similarity(query_tfidf, self.tfidf_matrix).flatten().

- **Skor BM25**: Skor ini dihitung oleh fungsi terpisah calculate_bm25_scores(self, query, k1=1.5, b=0.75). Fungsi ini mengimplementasikan formula standar BM25 secara rinci. Langkah-langkahnya melibatkan tokenisasi kueri (query_terms = self.preprocess_text(query).split()), perhitungan doc_freqs (frekuensi dokumen untuk setiap term dalam kueri), dan perhitungan avg_doc_len (panjang rata-rata dokumen dalam korpus). Kemudian, untuk setiap dokumen dalam korpus, dihitung tf (frekuensi term kueri dalam dokumen tersebut), df (frekuensi dokumen untuk term kueri), dan idf (menggunakan formula logaritmik: math.log((N - df + 0.5) / (df + 0.5)), di mana N adalah jumlah total dokumen). Skor BM25 untuk setiap term kueri dalam dokumen dihitung menggunakan formula idf * (tf * (k1 + 1)) / (tf + k1 * (1 - b + b * doc_len / avg_doc_len)), dan skor total BM25 untuk dokumen tersebut adalah penjumlahan skor dari semua term kueri. Parameter k1 (mengontrol saturasi frekuensi term) dan b (mengontrol normalisasi panjang dokumen) adalah parameter standar BM25 yang nilainya dapat disesuaikan.
- **Fitur Tambahan**: Selain skor TF-IDF dan BM25, kode juga mengekstraksi fitur-fitur lain yang lebih bersifat heuristik atau berbasis aturan, seperti title_exact_match = [1 if query.lower() in title.lower() else 0 ...] (mengecek apakah kueri persis ada dalam judul buku), author_match = [1 if any(term in author.lower() for term in query_terms) else 0 ...] (mengecek apakah ada term dari kueri yang cocok dengan nama penulis), dan title_term_count = [sum(1 for term in query_terms if term in title.lower()) ...] (menghitung berapa banyak term dari kueri yang muncul di judul buku). Penggunaan beragam fitur ini, termasuk TF-IDF dan BM25, adalah ciri khas dari pendekatan LTR.

3.3.3 Model TF-IDF dengan Umpan Balik Relevansi Rocchio

Dalam implementasi algoritma Rocchio yang terdapat pada *file* Rocchio (Relevance Feedback) (2).ipynb, pembangunan indeks TF-IDF merupakan langkah sentral yang terjadi pada saat inisialisasi objek dari kelas RocchioInformationRetrieval. Objek TfidfVectorizer diinisialisasi dengan perintah self.vectorizer = TfidfVectorizer(max_features=5000, stop_words='english', lowercase=True, ngram_range=(1, 2)). Parameter max_features=5000 membatasi jumlah fitur unik yang dipertimbangkan, stop_words='english' menghilangkan kata-kata umum, lowercase=True memastikan semua teks diproses dalam huruf kecil (meskipun ini mungkin redundan jika pra-pemrosesan sebelumnya sudah mencakupnya), dan ngram_range=(1, 2) memungkinkan penggunaan unigram dan bigram. Setelah inisialisasi, matriks TF-IDF untuk seluruh koleksi dokumen dibangun menggunakan self.doc_vectors = self.vectorizer.fit_transform(self.documents), di mana self.documents adalah daftar dokumen yang telah melalui tahap pra-pemrosesan. self.vectorizer yang telah di-*fit* ini menjadi sangat penting karena ia menyimpan seluruh kosakata (vocabulary) yang dipelajari dari korpus beserta bobot *Inverse Document Frequency* (IDF) untuk setiap term dalam kosakata tersebut. Informasi ini akan digunakan secara konsisten nantinya untuk mentransformasi kueri baru yang dimasukkan oleh pengguna ke dalam ruang vektor yang sama dengan dokumen.

3.4 Proses Pencarian dan Perankingan

3.4.1 Pencarian dan Perankingan dengan BM25

Ketika seorang pengguna memasukkan kueri pencarian dalam sistem yang diimplementasikan pada *file* BM25 (1).ipynb, fungsi search(query, bm25_model, df, k=5) akan dieksekusi untuk memproses kueri tersebut dan menghasilkan daftar dokumen yang relevan. Langkah pertama adalah pra-pemrosesan kueri itu sendiri. Kueri mentah dari pengguna akan dilewatkan ke fungsi preprocess_text(query), yang akan melakukan langkah-langkah pembersihan dan tokenisasi yang sama persis seperti yang telah diterapkan pada dokumen dalam korpus, menghasilkan tokenized_query. Konsistensi dalam pra-pemrosesan ini sangat penting. Selanjutnya, model BM25Okapi yang telah dibangun sebelumnya (bm25_model) digunakan untuk menghitung skor relevansi. Ini dilakukan melalui pemanggilan metode doc_scores = bm25_model.get_scores(tokenized_query). Metode ini akan mengkalkulasi skor BM25 untuk setiap dokumen dalam tokenized_corpus terhadap tokenized_query berdasarkan statistik internal model (TF, IDF, panjang dokumen). Setelah semua skor dokumen diperoleh, langkah berikutnya adalah pengurutan. Perintah top_k_indices = np.argsort(doc_scores)[::-1][:k] menggunakan NumPy untuk mendapatkan indeks dari k dokumen yang memiliki skor BM25 tertinggi (pengurutan dilakukan secara menurun, [::-1], dan [:k] mengambil k elemen pertama). Terakhir, untuk setiap indeks dalam top_k_indices, jika skor dokumen pada indeks tersebut (doc_scores[idx]) lebih besar dari 0 (menandakan adanya tingkat relevansi minimal), informasi detail dokumen (diambil dari df.iloc[idx] menggunakan kolom-kolom yang didefinisikan dalam DISPLAY_COLUMNS) beserta skor BM25-nya akan dikumpulkan dan kemudian ditampilkan kepada pengguna sebagai hasil pencarian. Algoritma BM25, seperti yang dijelaskan dalam dokumen Word, memberikan skor relevansi dengan menyesuaikan bobot kata kunci berdasarkan frekuensi kemunculannya dalam dokumen dan panjang dokumen tersebut.

3.4.2 Pencarian dan Perankingan dengan LTR

Dalam sistem Learning to Rank yang diimplementasikan pada *file* Learning To Rank (1).ipynb, metode search(self, query, top_k=10) bertanggung jawab atas keseluruhan proses pencarian dan perankingan. Terdapat mekanisme *fallback*: jika model LTR (self.learning_model) belum dilatih (yaitu, self.learning_model is None), maka sistem akan secara otomatis memanggil return self.search_tfidf(query, top_k). Fungsi search_tfidf ini akan melakukan pencarian standar berbasis TF-IDF, di mana kueri diubah menjadi vektor TF-IDF dan kemiripan kosinus dihitung terhadap semua dokumen untuk perankingan. Namun, jika model LTR telah berhasil dilatih, prosesnya adalah sebagai berikut: pertama, untuk kueri baru yang dimasukkan oleh pengguna, fitur-fitur yang sama persis seperti yang digunakan pada tahap pelatihan akan diekstraksi dari semua dokumen dalam dataset relatif terhadap kueri tersebut. Ini dilakukan melalui pemanggilan features = self.extract_features(query). Setelah matriks fitur untuk kueri saat ini diperoleh, model LTR (self.learning_model, yang merupakan objek RandomForestRegressor yang telah dilatih) digunakan untuk menghasilkan prediksi skor relevansi. Ini dilakukan dengan perintah relevance_scores = self.learning_model.predict(features). Setiap baris dalam features (merepresentasikan satu dokumen) akan mendapatkan satu skor relevansi dari model. Langkah selanjutnya adalah pengurutan dokumen berdasarkan skor prediksi LTR ini. top_indices = np.argsort(relevance_scores)[::-1][:top_k] digunakan untuk mendapatkan indeks dari k dokumen dengan skor tertinggi. Akhirnya, untuk setiap indeks teratas, informasi detail buku (judul, penulis), skor LTR yang diprediksi, deskripsi singkat, dan ISBN akan diambil dari self.df.iloc[idx] dan ditampilkan kepada pengguna sebagai hasil pencarian. Pendekatan LTR, sebagaimana disebutkan dalam dokumen Word, memanfaatkan data dan fitur relevansi untuk melatih model yang dapat memprediksi peringkat hasil pencarian berdasarkan relevansi, yang bertujuan menghasilkan hasil yang lebih personal dan optimal.

3.5.3 Pencarian Awal dan Umpan Balik Relevansi Rocchio

Proses pencarian dalam sistem Rocchio yang diimplementasikan pada *file* Rocchio (Relevance Feedback) (2).ipynb dapat terdiri dari dua tahap utama, terutama jika pengguna memanfaatkan fitur umpan balik relevansi.

Tahap Pencarian Awal (Tanpa Umpan Balik): Ketika pengguna pertama kali memasukkan kueri, fungsi search(self, query, top_k=10) akan dipanggil. Di dalam fungsi ini, kueri mentah dari pengguna akan melalui proses pra-pemrosesan menggunakan processed_query = self.preprocess_text(query). Kueri yang telah bersih ini kemudian ditransformasikan menjadi representasi vektor TF-IDF menggunakan vectorizer yang telah di-*fit* sebelumnya pada seluruh korpus: query_vector = self.vectorizer.transform([processed_query]). Setelah mendapatkan vektor kueri, skor kemiripan kosinus dihitung antara query_vector dan setiap vektor dokumen yang tersimpan dalam self.doc_vectors (matriks TF-IDF seluruh dokumen). Ini dilakukan dengan similarities = cosine_similarity(query_vector, self.doc_vectors).flatten(). Hasil kemiripan ini kemudian diurutkan secara menurun, dan top_k dokumen dengan skor tertinggi akan ditampilkan kepada pengguna sebagai hasil pencarian awal.

Tahap Umpan Balik Relevansi dan Pencarian Ulang: Jika pengguna memutuskan untuk memberikan umpan balik terhadap hasil pencarian awal (yaitu, dengan mengidentifikasi dokumen mana yang dianggap relevan, relevant_docs, dan mana yang tidak relevan, non_relevant_docs, dari daftar yang ditampilkan), maka fungsi rocchio_feedback(self, original_query_vector, relevant_docs, non_relevant_docs) akan dipanggil. Dokumen Word secara eksplisit menjelaskan bahwa Algoritma Rocchio adalah metode yang memanfaatkan umpan balik dari pengguna untuk memodifikasi *query* pencarian berdasarkan dokumen yang relevan dan tidak relevan. Implementasi dalam kode mengikuti prinsip ini dengan cermat. Vektor kueri asli (original_query_vector, yang merupakan query_vector dari pencarian awal) akan dimodifikasi. Jika ada relevant_docs yang diberikan oleh pengguna, vektor TF-IDF dari dokumen-dokumen relevan tersebut akan diambil dari self.doc_vectors[relevant_docs], dan centroid (rata-rata vektor) dari dokumen-dokumen relevan ini (centroid_relevant = np.mean(relevant_vectors, axis=0)) akan dihitung. Demikian pula, jika ada non_relevant_docs, centroid dari dokumen-dokumen tidak relevan (centroid_non_relevant) juga akan dihitung. Vektor kueri baru (new_query_vector) kemudian dibentuk menggunakan formula Rocchio yang juga dikutip dalam dokumen Word: new_query_vector = self.alpha * original_query_vector. Selanjutnya, jika ada dokumen relevan, kontribusinya ditambahkan: new_query_vector += self.beta * centroid_relevant. Jika ada dokumen tidak relevan, kontribusinya dikurangkan: new_query_vector -= self.gamma * centroid_non_relevant. Parameter self.alpha, self.beta, dan self.gamma adalah bobot yang mengontrol pengaruh dari kueri asli, dokumen relevan, dan dokumen tidak relevan terhadap kueri yang dimodifikasi. Setelah new_query_vector terbentuk, fungsi search_with_feedback akan melakukan proses pencarian ulang. Skor kemiripan kosinus akan dihitung kembali, kali ini antara new_query_vector dan semua self.doc_vectors. Hasil pencarian yang baru, yang diharapkan lebih sesuai dengan preferensi pengguna berkat modifikasi kueri, kemudian akan diurutkan dan ditampilkan.

3.5 Evaluasi Sistem

3.5.1 Evaluasi Model Learning to Rank (LTR)

Dalam *file* Learning To Rank (1).ipynb, setelah menyajikan hasil pencarian kepada pengguna, terdapat sebuah blok kode yang didedikasikan untuk "EVALUASI METRICS". Pada bagian ini, sistem melakukan simulasi evaluasi kinerja model LTR. *Ground truth* atau data referensi relevansi dibuat secara artifisial dengan mengasumsikan bahwa tiga dokumen teratas yang dikembalikan oleh sistem LTR (results[:3]) adalah dokumen yang relevan (relevant_items = [str(r['isbn']) for r in results[:3]]). Daftar semua dokumen yang diambil oleh sistem pada pencarian tersebut dijadikan sebagai retrieved_items (berdasarkan ISBN mereka). Selanjutnya, kelas EvaluationMetrics (yang didefinisikan dalam kode yang sama) digunakan untuk menghitung beberapa metrik evaluasi peringkat standar. Metrik-metrik tersebut meliputi precision_5 = EvaluationMetrics.precision_at_k(relevant_items, retrieved_items, 5) (presisi pada 5 dokumen teratas), recall_5 = EvaluationMetrics.recall_at_k(relevant_items, retrieved_items, 5) (recall pada 5 dokumen teratas), ap_5 = EvaluationMetrics.average_precision_at_k(relevant_items, retrieved_items, 5) (yang merupakan komponen untuk menghitung Mean Average Precision, MAP, pada 5 dokumen teratas), dan ndcg_5 = EvaluationMetrics.ndcg_at_k(relevant_items, retrieved_items, 5) (Normalized Discounted Cumulative Gain pada 5 dokumen teratas). Meskipun abstrak dokumen Word Anda menyebutkan penggunaan metode evaluasi berbasis TF-IDF dan *cosine similarity*, implementasi evaluasi dalam kode LTR ini lebih fokus pada penilaian kualitas perankingan akhir yang dihasilkan oleh model LTR itu sendiri, menggunakan metrik standar IR dengan asumsi *ground truth* yang disimulasikan.

3.5.2. Evaluasi Model Rocchio

Serupa dengan pendekatan LTR, *file* Rocchio (Relevance Feedback) (2).ipynb juga menyediakan opsi menu bernomor "3. Evaluate system performance" untuk mendemonstrasikan evaluasi sistem.

Definisi Kueri Sampel dan Ground Truth: Bagian evaluasi ini dimulai dengan mendefinisikan sample_queries. Ini adalah sebuah daftar yang berisi beberapa pasangan, di mana setiap pasangan terdiri dari sebuah *string* kueri contoh (misalnya, "python programming") dan sebuah daftar indeks dokumen yang dianggap relevan untuk kueri tersebut (misalnya, [100, 150, 200]). Daftar indeks dokumen relevan ini berfungsi sebagai *ground truth* sintetis untuk tujuan evaluasi.

Proses Evaluasi Per Kueri: Untuk setiap kueri sampel dalam sample_queries, sistem akan menjalankan proses pencarian awal (yaitu, tanpa adanya umpan balik relevansi) menggunakan results, _ = ir_system.search(query, top_k=10). Dari hasil pencarian ini, daftar indeks dokumen yang berhasil diambil oleh sistem (retrieved_docs = [r['index'] for r in results]) akan dikumpulkan.

Penghitungan Metrik Evaluasi: Daftar retrieved_docs ini kemudian dibandingkan dengan relevant_docs (yang merupakan *ground truth* dari kueri sampel saat itu). Kelas EvaluationMetrics (yang juga didefinisikan dalam kode ini) digunakan untuk menghitung berbagai metrik. Secara spesifik, kode menghitung map_5 = metrics.map_at_k(queries_results, k=5) (di mana queries_results adalah daftar yang berisi pasangan (retrieved_docs, relevant_docs) untuk semua kueri sampel). Selain itu, untuk setiap kueri sampel, dihitung skor presisi (precision_scores.append(metrics.precision_at_k(...))), recall (recall_scores.append(metrics.recall_at_k(...))), dan NDCG (ndcg_scores.append(metrics.ndcg_at_k(...))). Nilai rata-rata dari skor-skor ini untuk semua kueri sampel kemudian ditampilkan. Evaluasi ini memberikan gambaran mengenai kinerja dasar dari sistem temu kembali informasi berbasis TF-IDF sebelum adanya intervensi umpan balik dari pengguna melalui algoritma Rocchio.

4. HASIL DAN PEMBAHASAN

Bab ini menyajikan analisis hasil yang diharapkan dari implementasi ketiga metodologi temu

kembali informasi—BM25, Learning to Rank (LTR), dan TF-IDF dengan Umpan Balik Rocchio—serta membahas implikasi dan perbandingan antar metodologi tersebut berdasarkan implementasi kode dan studi literatur yang relevan dari dokumen yang dilampirkan.

4.1 Hasil Implementasi Algoritma

4.1.1 Hasil Pencarian dengan Algoritma BM25

Implementasi algoritma BM25, seperti yang terdapat dalam *file* BM25 (1).ipynb, dirancang untuk menghasilkan peringkat dokumen berdasarkan skor relevansi probabilistik. Hasil yang diharapkan dari sistem ini adalah daftar buku yang diurutkan, di mana buku-buku yang paling relevan dengan kueri pengguna akan muncul di posisi teratas. Skor BM25, yang dihitung oleh bm25_model.get_scores(tokenized_query), akan merefleksikan pentingnya term kueri dalam setiap dokumen, dengan mempertimbangkan frekuensi term (TF), *inverse document frequency* (IDF), dan normalisasi panjang dokumen. Dokumen Word menyebutkan bahwa BM25 merupakan pengembangan dari TF-IDF yang dirancang untuk menangani masalah frekuensi term dan panjang dokumen dengan lebih baik, serta memberikan skor relevansi yang lebih akurat. Oleh karena itu, hasil pencarian dari implementasi ini diharapkan dapat memberikan peringkat yang cukup baik, terutama untuk kueri yang mengandung term-term yang memiliki distribusi yang jelas dalam korpus. Keunggulan BM25 adalah kesederhanaan dan efisiensinya dalam memberikan hasil pencarian yang relevan.

4.1.2 Hasil Pencarian dengan Algoritma Learning to Rank (LTR)

Model Learning to Rank (LTR) yang diimplementasikan dalam *file* Learning To Rank (1).ipynb bertujuan untuk meningkatkan hasil pencarian dengan memanfaatkan pendekatan berbasis *machine learning*. Hasil dari sistem ini adalah daftar buku yang diperingkat berdasarkan skor yang diprediksi oleh model RandomForestRegressor (self.learning_model.predict(features)). Skor ini didasarkan pada kombinasi berbagai fitur, termasuk skor TF-IDF, skor BM25 (dihitung melalui calculate_bm25_scores), kecocokan judul persis, kecocokan penulis, dan jumlah term kueri di judul. Dokumen Word menyatakan bahwa LTR memanfaatkan data dan fitur relevansi untuk melatih model yang dapat memprediksi peringkat hasil pencarian berdasarkan relevansi. Dengan menggunakan skor relevansi sintetis untuk pelatihan, hasil yang diharapkan adalah model LTR yang mampu menangkap pola kompleks antara fitur-fitur dokumen dan relevansinya terhadap kueri, yang berpotensi menghasilkan peringkat yang lebih adaptif dan optimal dibandingkan metode peringkat tradisional. Jika model LTR tidak dilatih, sistem akan kembali menggunakan pencarian TF-IDF standar, yang akan menghasilkan peringkat berdasarkan kemiripan kosinus. Evaluasi metrik seperti Precision@5, Recall@5, MAP@5, dan NDCG@5, meskipun menggunakan *ground truth* sintetis dalam kode, memberikan indikasi awal mengenai kemampuan model LTR dalam menempatkan dokumen relevan di posisi atas.

4.1.3 Hasil Pencarian dengan Algoritma TF-IDF dan Umpan Balik Relevansi Rocchio

Implementasi algoritma TF-IDF dengan umpan balik Rocchio dalam *file* Rocchio (Relevance Feedback) (2).ipynb dirancang untuk menghasilkan dua set hasil: hasil pencarian awal dan hasil pencarian yang disempurnakan setelah umpan balik pengguna.

Hasil Pencarian Awal: Hasil awal didasarkan pada skor kemiripan kosinus antara vektor TF-IDF kueri dan vektor TF-IDF setiap dokumen (similarities = cosine_similarity(query_vector, self.doc_vectors).flatten()). Peringkat ini mencerminkan relevansi berdasarkan model ruang vektor standar.

Hasil Pencarian Setelah Umpan Balik Rocchio: Setelah pengguna memberikan umpan balik dengan menandai dokumen relevan dan tidak relevan, vektor kueri dimodifikasi menggunakan formula Rocchio (new_query_vector = self.alpha * original_query_vector + self.beta * centroid_relevant - self.gamma * centroid_non_relevant). Dokumen Word menjelaskan bahwa Algoritma Rocchio menggunakan umpan balik dari pengguna untuk memodifikasi *query* pencarian. Hasil pencarian yang baru, yang dihitung menggunakan kemiripan kosinus dengan new_query_vector, diharapkan menjadi lebih personal dan relevan dengan preferensi spesifik

pengguna tersebut. Efektivitas penyempurnaan ini sangat bergantung pada kualitas dan kuantitas umpan balik yang diberikan oleh pengguna. Contoh evaluasi dalam kode, yang menghitung metrik seperti MAP@5 dan NDCG@5 untuk pencarian awal, memberikan *baseline* kinerja sebelum umpan balik diterapkan.

4.2 Pembahasan

4.2.1 Perbandingan Kinerja Algoritma

Ketiga algoritma yang diimplementasikan—BM25, LTR, dan TF-IDF dengan Rocchio—memiliki pendekatan yang berbeda dalam menentukan relevansi dan menghasilkan peringkat.
BM25 menawarkan pendekatan probabilistik yang kuat dan telah terbukti efektif dalam banyak skenario pencarian teks. Kelebihannya terletak pada kemampuannya untuk menangani frekuensi term dan panjang dokumen secara lebih canggih dibandingkan TF-IDF murni. Implementasi dalam BM25 (1).ipynb langsung memanfaatkan *library* rank_bm25 yang sudah teroptimasi.
Learning to Rank (LTR), seperti yang diimplementasikan dalam Learning To Rank (1).ipynb, adalah pendekatan yang lebih canggih karena menggunakan *machine learning* untuk "belajar" cara memperingkat dokumen. Dengan menggabungkan berbagai fitur (TF-IDF, BM25, fitur berbasis aturan seperti kecocokan judul), LTR berpotensi menghasilkan peringkat yang lebih akurat dan adaptif terhadap nuansa data dan preferensi pengguna (meskipun dalam kode ini preferensi pengguna disimulasikan melalui skor relevansi sintetis). Namun, LTR memerlukan data pelatihan (dalam kasus ini, data sintetis) dan proses pelatihan model yang bisa jadi lebih kompleks dibandingkan BM25. Dokumen Word juga menyoroti bahwa LTR memerlukan pemeliharaan untuk menyesuaikan dengan perubahan data atau preferensi pengguna.
TF-IDF dengan Umpan Balik Rocchio, diimplementasikan dalam Rocchio (Relevance Feedback) (2).ipynb, menawarkan mekanisme personalisasi melalui interaksi pengguna. Pencarian awal berbasis TF-IDF memberikan *baseline* yang solid. Kekuatan utama Rocchio adalah kemampuannya untuk memperbaiki kueri berdasarkan dokumen yang secara eksplisit ditandai relevan atau tidak relevan oleh pengguna, sehingga hasil pencarian berikutnya dapat lebih terarah. Namun, efektivitasnya sangat bergantung pada partisipasi aktif pengguna dalam memberikan umpan balik yang berkualitas. Dokumen Word menyebutkan bahwa algoritma ini mungkin tidak efisien jika interaksi pengguna terbatas.

4.2.2 Relevansi Hasil dan Konteks Penggunaan

Relevansi adalah faktor kunci dalam kualitas hasil pencarian.
BM25 cenderung menghasilkan hasil yang relevan untuk pencarian berbasis kata kunci di mana pencocokan frekuensi dan distribusi term penting. Ini efektif untuk pencarian umum di mana pemahaman semantik yang mendalam tidak selalu menjadi prioritas utama.
LTR berpotensi memberikan hasil yang paling relevan dalam skenario di mana banyak sinyal atau fitur dapat diekstraksi dan dipelajari. Kemampuannya untuk menggabungkan berbagai sumber informasi (TF-IDF, BM25, fitur struktural) memungkinkannya menangkap relevansi yang lebih multifaset. Dokumen Word menyatakan bahwa pendekatan LTR lebih adaptif dan dapat disesuaikan dengan preferensi dan perilaku pengguna, sehingga memungkinkan hasil pencarian yang lebih personal dan optimal.
Rocchio sangat berguna ketika preferensi relevansi pengguna bersifat dinamis atau sulit ditangkap oleh model statis. Dengan memodifikasi kueri secara langsung berdasarkan umpan balik, sistem dapat dengan cepat beradaptasi dengan kebutuhan informasi spesifik pengguna pada sesi pencarian tertentu.
Metode pengambilan informasi lainnya seperti *Boolean Search*, *Phrase Search*, dan *Proximity Search*, serta penggunaan vektor *embedding* untuk pencarian semantik. Meskipun ketiga *file code* utama tidak secara langsung mengimplementasikan semua teknik ini (misalnya, *Boolean Search*

atau *Phrase Search* seperti dalam contoh-contoh di bagian akhir dokumen Word), pemahaman bahwa BM25 dan TF-IDF (yang digunakan dalam Rocchio dan sebagai fitur LTR) adalah dasar dari *ranked retrieval* tetap penting. Implementasi LTR dalam kode bahkan sudah mulai memasukkan fitur-fitur yang lebih dari sekadar pencocokan kata kunci murni, seperti kecocokan judul dan penulis, yang merupakan langkah menuju pemahaman konteks yang lebih baik, sejalan dengan tujuan meningkatkan relevansi.

4.2.3 Keterbatasan Implementasi

Penting untuk dicatat bahwa implementasi LTR dalam kode menggunakan skor relevansi sintetis. Dalam aplikasi dunia nyata, LTR akan mendapat manfaat besar dari data relevansi yang dihasilkan oleh manusia (human judgments) untuk pelatihan, yang akan menghasilkan model yang lebih akurat. Untuk Rocchio, efektivitasnya di dunia nyata bergantung pada antarmuka pengguna yang intuitif untuk memberikan umpan balik dan kemauan pengguna untuk berinteraksi. Model BM25, meskipun kuat, tidak secara inheren menangani sinonim atau pemahaman semantik yang lebih dalam seperti yang bisa dilakukan oleh model berbasis *embedding* yang disebutkan dalam dokumen Word.

Secara keseluruhan, ketiga pendekatan yang diimplementasikan menunjukkan berbagai strategi untuk meningkatkan relevansi pencarian informasi. Pemilihan algoritma yang paling sesuai akan sangat bergantung pada konteks spesifik aplikasi, ketersediaan data pelatihan, dan tingkat interaksi pengguna yang diharapkan. Temuan ini, sejalan dengan kesimpulan dokumen Word, bermanfaat bagi pengembangan sistem pencarian informasi yang lebih efisien dan efektif di masa depan.

5. Kesimpulan

Penelitian ini telah berhasil mengimplementasikan dan menganalisis tiga metodologi utama dalam sistem temu kembali informasi untuk pencarian buku, yaitu algoritma BM25, Learning to

Rank (LTR), dan TF-IDF dengan Umpan Balik Relevansi Rocchio. Setiap algoritma menunjukkan karakteristik dan potensi kinerja yang berbeda dalam upaya meningkatkan akurasi dan relevansi hasil pencarian, sejalan dengan tujuan utama studi untuk mengevaluasi dan membandingkan teknik pencarian serta metode perankingan.

Implementasi BM25 menunjukkan pendekatan probabilistik yang solid, mampu memberikan peringkat relevansi yang baik dengan memperhitungkan frekuensi term dan panjang dokumen secara efektif. Algoritma ini, seperti yang dijelaskan dalam literatur, menawarkan keseimbangan antara kesederhanaan implementasi dan efektivitas hasil, terutama untuk pencarian berbasis kata kunci standar.

Pendekatan Learning to Rank (LTR) menghadirkan metode yang lebih canggih dengan memanfaatkan *machine learning* untuk "belajar" bagaimana cara terbaik memperingkat dokumen. Dengan menggabungkan beragam fitur seperti skor TF-IDF, skor BM25, serta fitur-fitur heuristik lainnya (kecocokan judul dan penulis), model LTR yang dilatih menggunakan `RandomForestRegressor` berpotensi menangkap nuansa relevansi yang lebih kompleks. Meskipun implementasi ini menggunakan skor relevansi sintetis untuk pelatihan, ia mendemonstrasikan potensi LTR untuk menghasilkan peringkat yang lebih adaptif dan optimal, sejalan dengan klaim bahwa LTR dapat disesuaikan dengan preferensi pengguna.

Algoritma TF-IDF dengan Umpan Balik Relevansi Rocchio menawarkan mekanisme personalisasi yang dinamis. Sistem ini dimulai dengan peringkat berbasis TF-IDF dan kemiripan kosinus, kemudian memungkinkan pengguna untuk secara aktif memperbaiki hasil pencarian melalui umpan balik terhadap dokumen relevan dan tidak relevan. Modifikasi vektor kueri menggunakan formula Rocchio memungkinkan sistem untuk menyesuaikan diri dengan kebutuhan informasi spesifik pengguna pada sesi pencarian tertentu, yang berpotensi meningkatkan relevansi hasil secara signifikan.

Perbandingan antara ketiga metode ini menunjukkan bahwa tidak ada satu solusi tunggal yang superior dalam semua kondisi. BM25 cocok untuk skenario pencarian umum yang efisien. LTR menjanjikan hasil yang sangat optimal jika tersedia data pelatihan yang berkualitas dan beragam fitur dapat diekstraksi. Rocchio unggul dalam situasi yang memerlukan adaptasi cepat terhadap preferensi pengguna melalui interaksi langsung. Pemilihan metode yang paling tepat akan sangat bergantung pada konteks aplikasi, ketersediaan sumber daya (seperti data pelatihan), dan tingkat interaktivitas yang diinginkan dengan pengguna.

Secara keseluruhan, penelitian ini menggarisbawahi pentingnya pemilihan algoritma yang tepat dan konfigurasi yang cermat dalam membangun sistem pencarian informasi yang efektif. Baik BM25, LTR, maupun Rocchio, masing-masing memberikan kontribusi berharga terhadap upaya untuk menyajikan informasi yang paling relevan kepada pengguna, yang pada akhirnya dapat meningkatkan kualitas pengambilan keputusan dan efisiensi pencarian dalam volume data yang terus meningkat. Temuan ini mendukung pengembangan sistem pencarian informasi yang lebih efisien dan efektif di masa depan.