COMP3009J – Information Retrieval 2021-2022
Information Retrieval System for R&B

Design of Search Engine for Law Firm

Final Assessment: Yang Liuxin
Word Count: 2999
UCD Student Number: *19206207*
BJUT Student Number: *19372226*

Contents

1 Introduction

This report describes the design of a keyword-based Legal Documents Retrieval system for an Irish law firm (R&B). Here are the requirements.

- Design an Information Retrieval (IR) system for R&B to search for over 1 million legal documents of different types written in English.
- The IR system should not overlook any important document when searching.
- Users enter keyword queries to express their information needs.
- Users should be convinced of the IR system's effectiveness.

Specifically, the report explains the components of the IR system and the reasons for chosen techniques in detail. Moreover, I include tradeoffs to show critical thinking. Now this report will explore these in detail.

2 Legal Documents Retrieval Pipeline

Classical IR pipeline has drawbacks such as lack of advanced technology and vocabulary mismatch, particularly for polysemy and synonym (see more in Figure 2). To solve problems, I design an improved pipeline (see Figure 1).

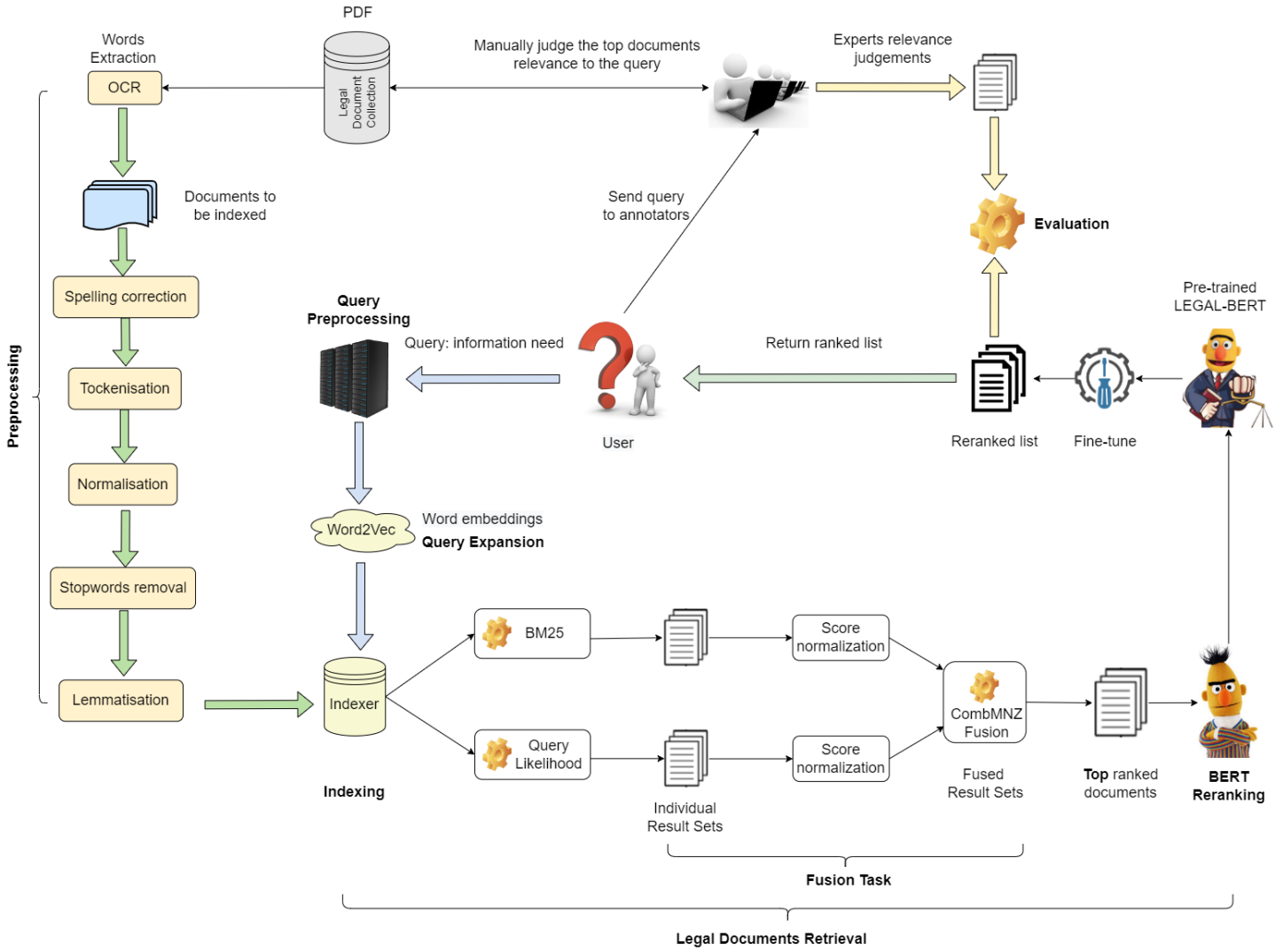

Figure 1: Legal Documents Retrieval Pipeline

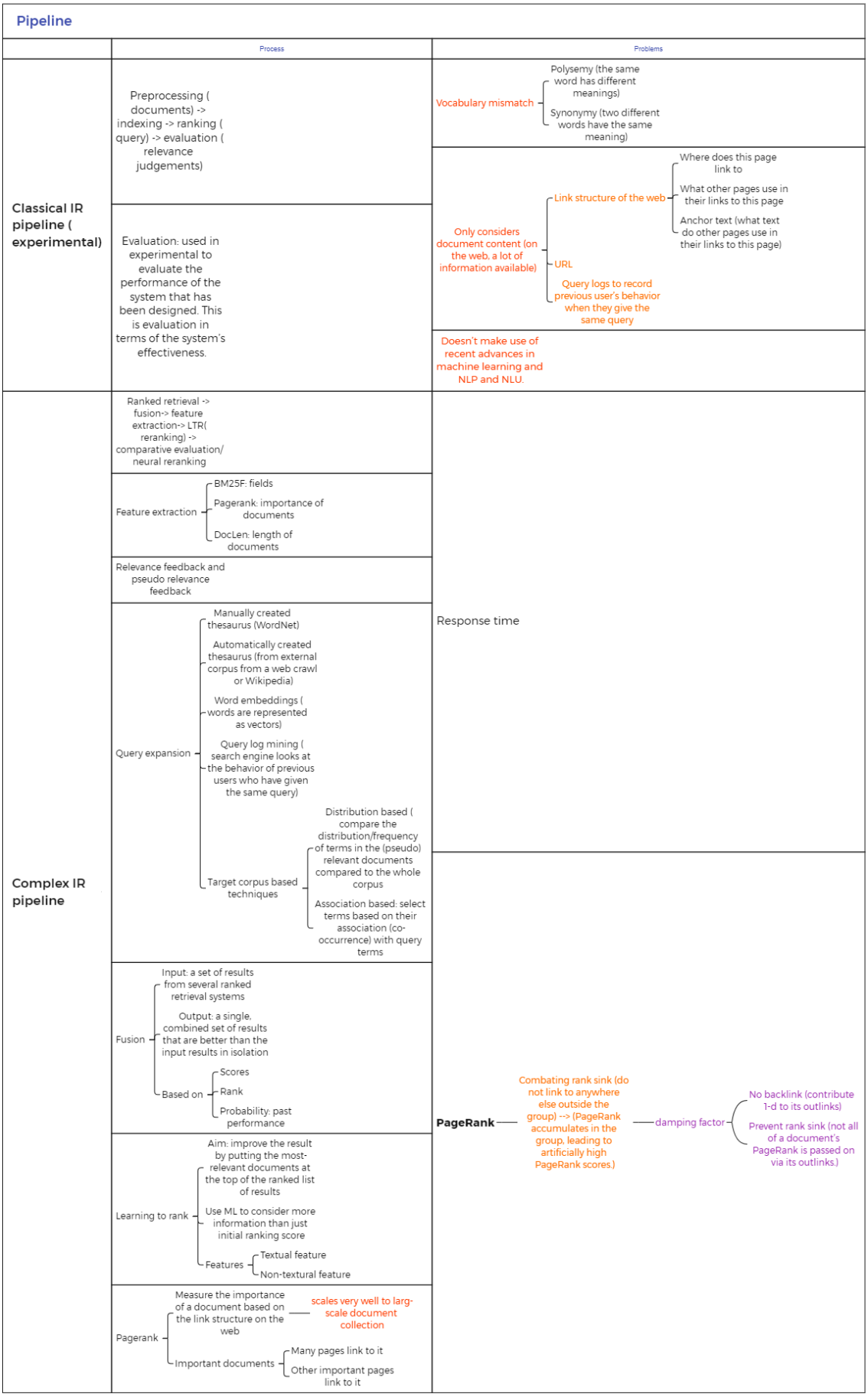

Figure 2: Comparison between Traditional and Complex IR Pipeline

3 Preprocessing

3.1 Text Extraction

My system extracts texts from scanned PDF documents using a text recognition technique known as Optical Character Recognition (OCR). OCR enables the automatic conversion of many document types, such as scanned paper documents, PDF files, and photographs, into searchable data (Haponik 2021).

3.2 Spelling Correction

My system performs isolated word correction and context-sensitive correction to rectify typos in documents and user queries. Document correction is beneficial for **OCR'ed documents** to reduce misspellings in the dictionary without changing documents (Manning, Raghavan & Schütze 2008;2012;).

3.3 Tokenisation

Tokeniser splits character sequences from the previous section into tokens, which are different from terms included in the index. Specifically, I remove whitespace and specific characters, including punctuation within and between words. Here are some of my design choices (Manning et al. 2008;2012;).

- I split words by **apostrophe**: so "Finland's Capital" is processed as "Finland" and "Capital".
- I do not split words by **hyphens** because they may not work alone.
- I do **not split (clients or businesses) names** because they may be searched as keywords.
- For **white spaces**, I include some words as a single term candidate, such as "San Francisco," in the index. Users seldom search for San or Francisco because they do not work alone.
- I **include numbers** in the index because computing storage and power are much greater. Excluding numbers from the index to expedite searches for older IR systems on the assumption that numbers are less likely to be sought for. However, people do search for numbers in the law IR, such as the year a crime was committed and the offence code. Therefore, my system **normalises different date representations**, such as 14/4/2018 and April/14/2018, to the same format as a single term for better retrieval.
- **No language-specific issues**: because files are written in English.

3.4 Normalisation (Equivalence Classing of Terms)

Then my system normalises tokens to match terms effectively despite superficial variances in character sequences (Manning et al. 2008;2012;). I will construct **equivalence classes** by converting all letters to **lowercase**. Moreover, I remove all punctuation except hyphens (U.S.A –>usa). The mapping rules enable users to retrieve documents containing either the exact query term or another with the same normalised term.

Fed vs. fed
March vs. march
Turkey vs. turkey
US vs. us

Alternatively, I can **maintain relations between unnormalised tokens**, such as synonyms and homonyms. I can achieve this term relationship in two ways. The first method maintains the index with unnormalised tokens and a **query expansion list** with several vocabulary entries for a particular query word. The second way is to **index "automobile" under "car" and vice versa**. However, both two methods are considered less effective than equivalence class construction. This is because the first method, employing a query expansion dictionary, increases query processing complexity, while the second method demands additional storage space for postings. I employ the second method using posting lists owing to its improved flexibility (Manning et al. 2008;2012;). The second way is superior to building an equivalence class since the first method would map "C.A.T" to "cat", which is undesirable because C.A.T is an abbreviation and not an animal! See more in Section 5.2.

3.5 Stopwords Removal (Dropping Common Terms)

Stopwords exist commonly within the collection but provide little information about documents. Due to much processing power required to handle those too common but relative useless words, some IR systems remove them from documents by identifying them using Zipf's Law (see Figure 3).

Stopwords removal helps my IR system focus on essential keywords (Sarica & Luo 2021;2020;), save processing power and expedite searching by **reducing corpus size** (Ladani & Desai 2020). There is a **tradeoff** that

Algorithm Zip's Law

Require: vocabulary V for $\mathcal{C}$; threshold θ
  for all $t \in V$ **do**
    if $f(t) \geq \theta$ **then**
      remove t as a stopword
    end if
  end for

Figure 3: Zip's Law (Lo et al. 2005), (Ferilli 2021)

stopwords removal can make some queries worse because queries may include "Flights **to** Ireland" where "to" is often included in the stopwords list. Although good compression and optimisation reduce the need for stopwords removal (Schweighofer 2010), I choose to **remove stopwords but keep useful word combinations as terms**.

3.6 Lemmatisation

Since stemming or suffix stripping might **overstem unrelated words to the same word**, and it simply applies some simple rules to words, I use lemmatisation, a Natural Language Processing (NLP) technique that converts words into **lemmas** by analysing words and extracting information. Although overstemming could be alleviated by using a dictionary, the difficulty with homographs (same spelling but different meaning) still exists. However, there is a **tradeoff** that lemmatisation is more **effective** than stemming but **slower** due to more text analysis.

4 Indexing

My IR system employs **inverted index**, which maps a **smaller** number of keywords to their occurring documents, rather than a sequential document scan. Each term has a posting list with the **ordered doc ID** as the posting. Inverted index creation is shown in Figure 4 and then **compressed** to save I/O operations and storage overheads (Manning et al. 2008;2012;).

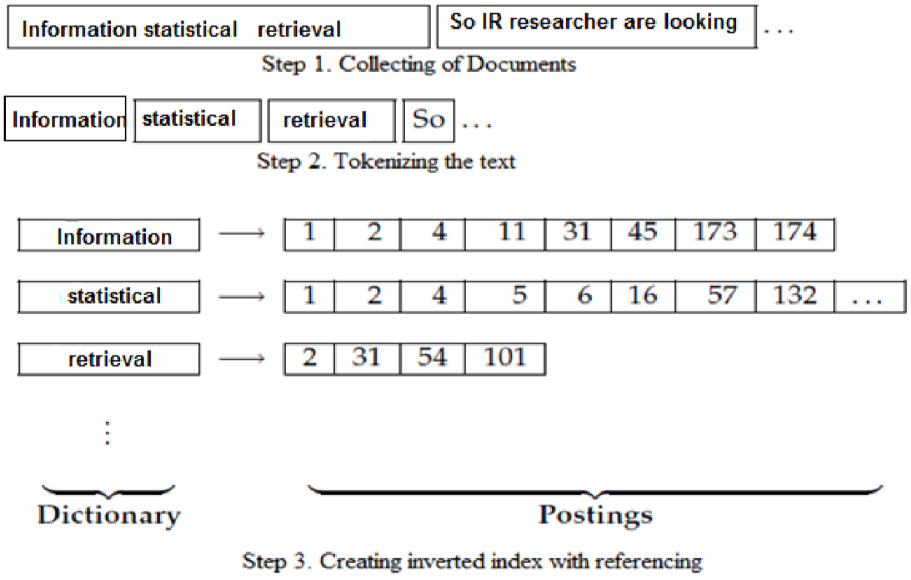

Figure 4: Inverted Index Creation (Thakur et al. 2012)

Figure 5 shows the traditional pipeline of my BM25 index construction.

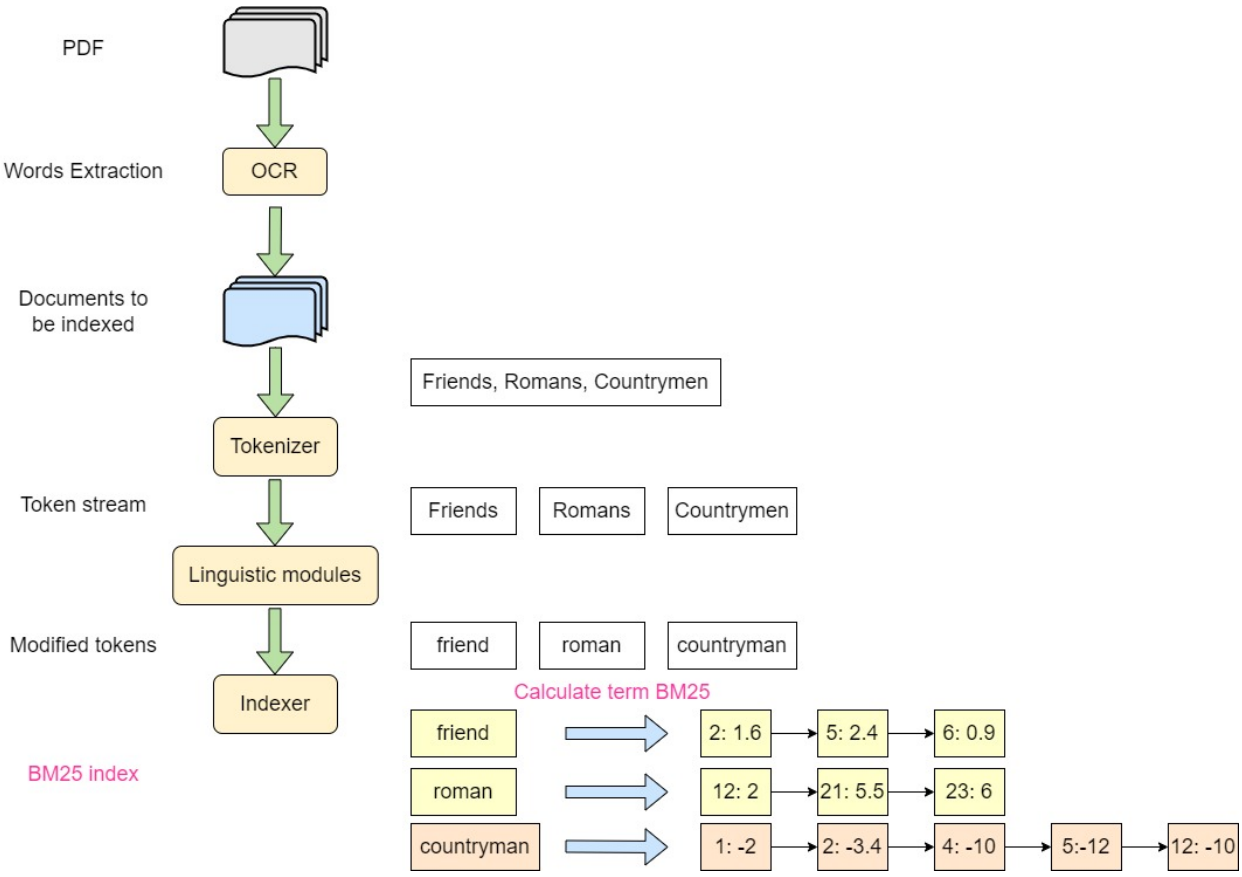

Figure 5: BM25 Index Construction

My appropriate index - dictionary is critical for query speed (see algorithm 6 for index construction).

Algorithm 1 Algorithm of Index Construction

Input: Distinct terms: inverted index called indexes
  removed = [ ]
  for each *term* ∈ *indexes* **do**
    doc_dic = indexes[term]
    for each *doc* ∈ *doc_dic* **do**
      calculate the ratio of document length and average document length
      calculate the term BM25
    end for
    if *term_BM25* > 0 **then**
      indexes[term][doc] = term_BM25
    else
      add the term to the removed list
      break
    end if
  end for
  for each *term* ∈ *removed* **do**
    remove that term from indexes (consider it as stopwords)
  end for

Figure 6: Algorithm of Index Construction

I pre-calculate BM25 for each term in the established inverted index to preprocess as much as possible. The key of my index is the term, and the value is another dictionary with doc ID as the key and BM25 value that the term contributes to this doc as the value. Figure 7 shows the change from inverted index to BM25 index.

Furthermore, my **index does not include terms with negative BM25 value** because they are too common and do not add meaning to differentiating documents by appearing in more than half the documents in the collection! My system stores the index file after calculating each term's BM25 score in each document. This

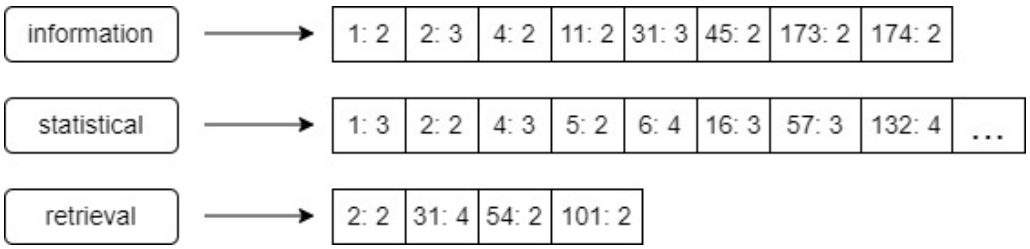

Figure 7: Term BM25 in Inverted Index

pre-calculating can save manual querying and evaluation time because it does not vary across queries.

5 Query Processing

The keyword queries entered by users will undergo the same **preprocessing** steps as documents. Moreover, users need to **select document types**, such as laws and judgments. This is because different **domain-specific terms** are used in various kinds of legal documents (Kanapala, Jannu & Pamula 2019).

5.1 Graphical User Interface

Figure 8 shows the search prototype. Section 6 explains how retrieval satisfies information needs.

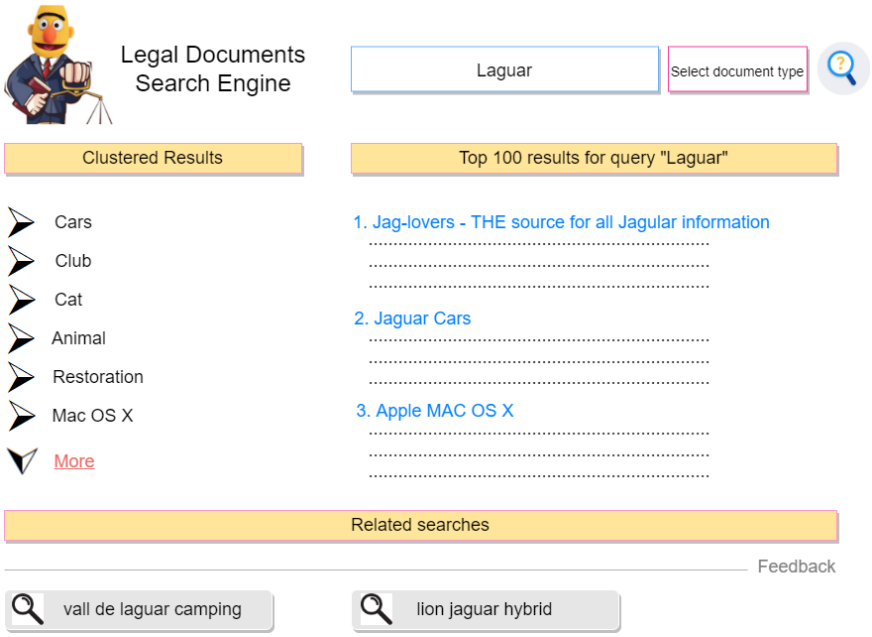

Figure 8: GUI of Search Engine

5.2 Query Expansion and Clustering

My system employs query expansion to mitigate the vocabulary mismatch between queries and relevant documents caused by lost information underlying brief queries and term independence in BM25 and query likelihood. Specifically, I use **Word2Vec's Continuous Bag-of-Words**, which represents terms in a vector space based on their co-occurrence (Mikolov, Chen, Corrado & Dean 2013). Moreover, I combine the terms identified using **word embeddings** with the **relevance model** based on pseudo-feedback (Mikolov et al. 2013). Here are more reasons for using Word2vec.

- Word2vec considers contextual information.
- Word2vec has fewer dimensions and, therefore, faster.
- Similarities between the term vectors represent semantic similarities, which are easy to calculate. In comparison, the creation of a manual thesaurus is laborious and costly.

- Although BERT (later in pipeline) provides more powerful embeddings, embeddings for query expansion are also needed here because BERT only reranks top documents from the fused result set. The final performance will be worse if relevant documents are ignored here due to no query expansion.
- It **improves recall** and **expedites search**.

However, query expansion may lead to **false positives**. Moreover, Word2vec **cannot solve polysemy**, such as "action" meaning "lawsuit"; therefore **clustering** after embedding based on polysemy is necessary (Schweighofer 2010). BERT can alleviate it.

6 Document Retrieval

My search engine uses **two-stage ranking**. This section describes first-stage retrieval ranking using BM25 and query likelihood for without ground truth labels. The second stage (see Section 8) re-ranks first-stage top documents using Bidirectional Encoder Representation from Transformers (BERT).

6.1 BM25

Probabilistic model may require user involvement (not just keywords) with the benefits of **relevance feedback**, **ranked** and **not one-off** results. However, users do not like to provide feedback, and the alternative pseudo relevance feedback may result in **query drift**, guessing initial probabilistic, binary weights and independent terms.

To solve that, I use BM25 (see Figure 9) that can be calculated efficiently **without relevance information** from users and allow **partial matching** and **ranking**. It is based on inverse document frequency, term frequency, and document length normalisation. My system sums all BM25 terms for each document to get the total BM25 of a document.

IR models

	Weight	Index	Disadvantage	Better representation	Notes	Advantage	Results
Boolean model	Binary 0,1 (it uses the term weights - binary weighting scheme)	Term-document incidence matrix	Incidence matrix is sparse Incidence matrix does not scale up large collection Either relevant or non-relevant, rather than based on users' needs No partial matching No ranking	Only store 1 Inverted index	Based on set theory and boolean algebra	Very efficient Predictable Structured queries works well when searchers know exactly what is wanted.	One-off
Inverted index		**Posting lists** Steps: Token sequence; Sort; Create dictionary and postings O(n) Process query O(logN) - ordered Merge two posting lists (AND): O(x+y); Worst case: distinct two lists	Large storage overhead. High maintenance cost on update, delete and insert.	Query optimization: Process in increasing order; Skip pointer (sorted): Easy for static ! / Only help memory-based	Ordered: Alphabetically / docID	Fast full text searches Easy to develop Popular and used on a large scale	One-off
Biword index		Index every consecutive pair of terms in the text as a phrase.	False positives Index blow up due to bigger memory	Combine biword and positional queries	Phrase queries	Two-word phrases can now easily be answered.	One-off
Positional index		{term: posting list...} Postings lists in a positional index: each posting is a docID and a list of positions (offsets)	Position index expands posting storage 2-4 times non-positional index Caveat: only for English-like languages	Combine biword and positional queries	Phrase queries	More effective to biword index for phrase queries.	One-off
Vector space model	TF-IDF	{term: {docID: weight...}...}	Independent		Represent text documents as vectors of identifiers	Partial matching Ranking Non-binary weighing	One-off
Probabilistic model	Binary 0,1		Guess initial probabilistic Binary weights - ignore frequency Terms are assumed to be independent	Automatic probabilistic retrieval: Avoid user interaction / Return a single list	Extended by BM25	Ranked User interaction	Not one-off Continue interacting
BM25	**TF-IDF**: If: rare across a collection carry more weight; tf: common within a document carry more weight; Document length normalization: larger documents do not get an unfair advantage	My index {term: {docID: bm25, ...}, ...}	Full of hacks/heuristics. This fact makes it hard to extend this framework.	BM25F: give different fields different importance BM25+: avoid short documents have too high scores	More words common with query is good Common words in a collection are less important Repetition of query words is good Repetitions are good, but less important than different query words Long documents do not get an unfair advantage	Not require relevance information (like vector space model that takes a query and returns the ranked list)	Patial ranking

Figure 9: BM25 Advantages over Other Models

6.2 Query Likelihood Model (QLM)

Here are reasons for my choice (Manning et al. 2008;2012;).

- QLM (language modelling) matches scores between queries and documents differently (so fusion can be more comprehensive) and assumes probabilistic modelling enhances weights and model performance.
- It is precise, simple, tractable, and intuitive.
- QLM is similar to TF-IDF but better at probabilistic intuitions.

QLM ranks documents based on the probability of their corresponding query. To overcome the data sparseness, I use **smoothing**, which transfers probability mass from the probability associated with a query term appearing in the document to that query term appearing in the collection. Therefore the predicted probability of a query term not appearing in the document is not zero; therefore, the document's overall score is non-zero (Zhang, Nulty & Lillis 2022).

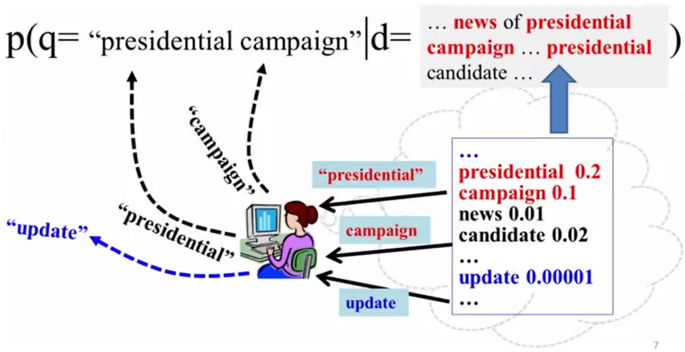

Figure 10: Query Likelihood

7 Score-based Fusion - CombMNZ

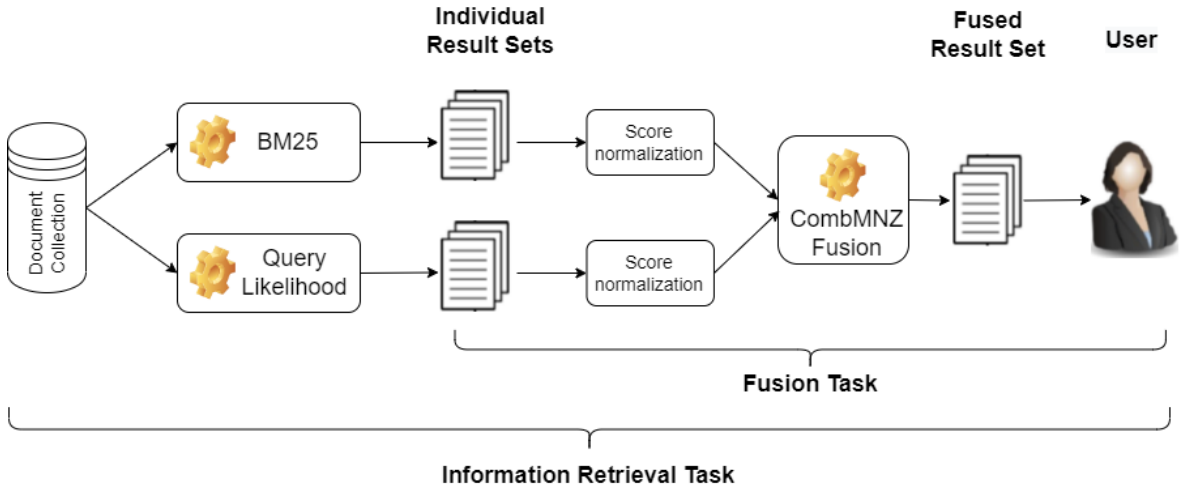

Figure 11: Data Fusion Process (Lillis 2020)

Since there is **no perfect retrieval model** for every query, my system combines results from BM25 and Query Likelihood model into a single ranked result of **better quality** than the individual input result sets (see Figure 11). My system focuses on **data fusion**. CombMNZ outperforms other fusion techniques here because:

- Rank-based fusion ignores score information and equals systems' qualities.
- Linear combination is not applicable because the system's weights (quality) are unavailable.
- Probability-based fusion is not applicable because past performance is unavailable.

After careful analysis (see Figure 12), I choose CombMNZ, which is simple but effective. It utilises *Skimming Effect* by ranking highly relevant documents near the top and **enhances *Chorus Effect*** by multiplying each document's CombSUM score by the number of result sets containing it. The *Dark Horse effect* is ignored. Taking advantage of the document's redundancies gives an edge to CombMNZ outperforming other models. However, it suffers from the failure of **single weight** to reflect useful characteristics. CombMNZ involves the following steps.

- **Score normalization**: make each document's score in a **comparable range**.

$$normalised\_score = \frac{unnormalised\_score - min\_score}{max\_score - min\_score} \tag{1}$$

- Adding individual normalized score from each input system.
- Multiply the sum by the number of result sets that a document returned in.

Fusion

	Idea	Effectiveness	Effect	Assumption	Application	Motivation	Phase	Distribution of Probabilities
Interleaving	Take one document from the top of each input set in a round robin fashion and add it to the fused result set. (Highest-ranked document that is not yet inclued in the fused result set)	Poor	Skimming effect	Every result set is of equal quality (better result sets are diluted by being merged with non-relevant documents from poorer systems)	collection fusion and data fusion			Many ranks with 0 probability
			No Chorus effect					Jagged distribution
Borda-Fuse	Based on election system		Skimming effect	Equal quality	Data fusion			Many ranks with 0 probability
	A few voters (input systems) vote for many candidates (documents)		Chorus effect					Jagged distribution
Reciprocal rank fusion		Simple and effective	Skimming effect	Equal quality	Data fusion			Many ranks with 0 probability
			Chorus effect					Jagged distribution
Other rank based	Use historical data to estimate which input systems tend to perform better.							
	Weighted version of interleaving so more documents are taken from better systems)							
	Weighted version of Borda-fuse							
CombSUM	Perform fusion using the relevance scores that each document is assigned by the input system.		Skimming effect: high scores in the input result set are carried to the fused result set	Equal quality	Data fusion			
			Chorus effect: because scores are added documents appearing in multiple result sets will receive a boost					
CombMNZ	F本組 10	Simple and effective	Skimming effect	Equal quality	Data fusion	Emphasize Chorus effect further	Normalize scores from each input system	
			Emphasizing Chorus effect further by multiplying by the number of result sets				Sum up all normalized scores	
							Multiply the sum by the number of result sets that the document occurs	
Linear combination	Similar to CombSUM, except that each normalized scores multiplying by the associated weighting before adding together		Skimming effect	Weight (but a single weight for each input system)	Data fusion			
			Choruses effect					
ProbFuse	Reflect ranks/positions that tend to return relevant		Skimming effect: divide by the segment number, giving highly-ranked documents an additional boost	series of weights	generally data fusion	Using a single weight for each system only reflects that the overall average performance of one system is better (or worse) than another.	Training — Probabilities for each input system estimated based on historical queries for which relevance judgements are available	High probability in early positions is lost
	Past results are analyzed during the training phase to build a series of weights, depending on the position (probabilities are calculated by examining historical data from each input system)							Segments smooth out " jagged" distribution
	Equal sized		Choruses effect: add each individual score				Fusion	Big difference in probability between adjacent ranks
SegFuse	Variations to ProbFuse — Instead of constant segment sizes, the size of the segments increase exponentially as going down the result set. / Normalized scores used to boost the Skimming effect rather than dividing by the segment number.		Skimming effect: multiplied by the normalised scores boosts the Skimming Effect, rather than dividing by the segment number.)	series of weights	generally data fusion	ProbFuse loses information since relevant documents are more likely to be found in early positions.	Training — Same as ProbFuse except that the size of the segment varies	Capture high early probability better
			Chorus effect: a document's final fused ranking score is based on adding some scores it receives from the individual input systems (all the data fusion algorithms basically use it in the same way)					Smoothing effect is still apparent
							Fusion	Probability drops between adjacent ranks are generally smaller, but are still apparent.
SlideFuse	Just use the probability at each rank.	4/5 runs measured with MAP	Skimming effect: probability of relevance is higher at the beginning	series of weights	generally data fusion	For SegFuse, adjacent (side-by-side) documents could be treated very different.	Training — Calculate the probability of relevance at each rank r	Capture high early probability like SegFuse
	Not dividing the result sets into segments. Use each rank's neighbours.	3/5 runs measured with bpref	Choruses effect: add each individual score				Fusion — For each input system, get the average probability in the sliding window that surrounds the document. / Average the sum of probabilities from all input systems	Smooth out zero probabilities
	Sliding window	5/5 runs measured with P@10	Smooth the probability graph — Avoid jadgged peaks seen in "rank only" distribution / Avoid sudden drops in ProbFuse and SegFuse					Large drops between adjacent documents eliminated

Figure 12: Fusion Comparison Analysis

8 Re-ranking - BERTopic

There are some issues concerning the above conventional models. Firstly, parameter tuning is difficult for a particular model's ranking calculation. Secondly, models perfectly tuned to the validation set may fail on test queries (over-fitting) when comparing two models. Thirdly, combining multiple models is not efficient (Liu 2009). Learning to Rank is also not applicable because expensive labelled data is unavailable.

Here are my reasons for employing **BERTopic** to **re-rank the top documents**.

- BERTopic takes a ranked list of documents as input from fused ranking models and produces a more reasonable re-ranked list based on the expanded query (Zheng, Hui, He, Han, Sun & Yates 2020).

- BERTopic keeps the **efficiency** and **lexical matching** of sparse representation(BM25), but also considers **contextual relevance** between query and document (Zhiying, Raphael, Ji & Jimmy 2021).
- BERTopic, a pre-trained transformer model, **performs well even with partial annotations** (Devlin, Chang, Lee & Toutanova 2018).
- BERTopic can identify the documents' topic, known as topic modelling (Silveira et al. 2021).

Here are the phases.

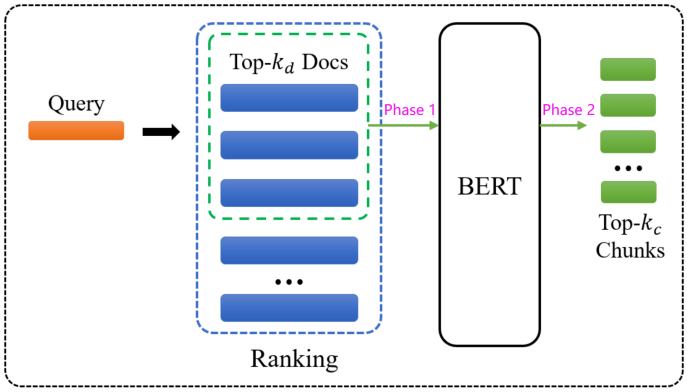

Figure 13: Fuse Ranked List into BERT

- **Phase 1**:

  My system uses the pre-trained model **LEGAL-BERT** which is pre-trained from legal data to incorporate **domain-specific** knowledge (Zhang et al. 2022). This is because the pre-trained contextual embeddings for the specific domain with **jargon (specialised word)** provide a more sophisticated and semantically rich text representation (Silveira et al. 2021) and uncover hidden information for better document retrieval (Rosso, Correa & Buscaldi 2011).

  Initial embeddings derived from token inputs is constructed by three vectors: *token embeddings* (pretraining embeddings), *segment embeddings* (sentence number decoded into a vector), and *position embeddings* (position of a word within a sentence decoded into a vector to order inputs). Concatenate these three vectors as initial embeddings to BERT simultaneously.

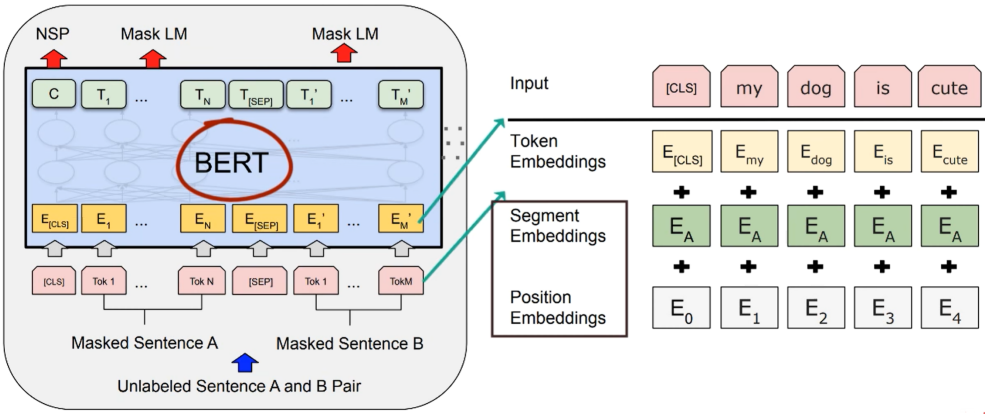

Figure 14: Pretraining BERT

- **Phase 2**: fine-tune the model.
- **Phase 3**: rerank documents for a query using fine-tuned BERT.
- **Phase 4**: extract relevant chunks from top-ranked documents based on a sliding window.
- **Phase 5**: use the selected expansion chunk for final reranking.

I will return a ranked list of **most** documents to users because any potentially useful documents should not be overlooked.

9 Evaluation

In this section, the report describes how to evaluate the system's effectiveness.

9.1 Evaluation Organization

My evaluation is organised in two stages (see Figure 15).

- **Stage one** (Section 9.2):

  Initially, I evaluated my system on TREC's legal and patent collection for IR evaluation (Buscaldi, Rosso, Gómez-Soriano & Sanchis 2009;2010;). This is because I want to convince the company that my system is worth their resources (i.e. the time employees use for evaluation) to further evaluate whether it works well on their dataset.

- **Stage two** (Section 9.3):

  Complete relevance judgements for huge collection is infeasible because it is laborious and costly. Fortunately, few documents are relevant to most queries, and my system exploits the Skimming Effect to rank relevant documents at the top. So next, I ask company staff to judge the **graded relevance** of the **top** returned documents to their proposed queries using **depth-100 pooling** (Yilmaz & Aslam 2006), reducing the number of documents to be judged while maintaining evaluation scores.

  Admittedly, there is a **tradeoff** because I cannot ensure all relevant documents are on top; thus, some are ignored. However, my evaluation decision is still reasonable because my goal is to evaluate the top documents, not all returned documents.

  Then compare the ranking of those documents run by my system against their given relevance. To evaluate the effectiveness of my system comprehensively, I **average the scores among all queries because they are of varying difficulties**.

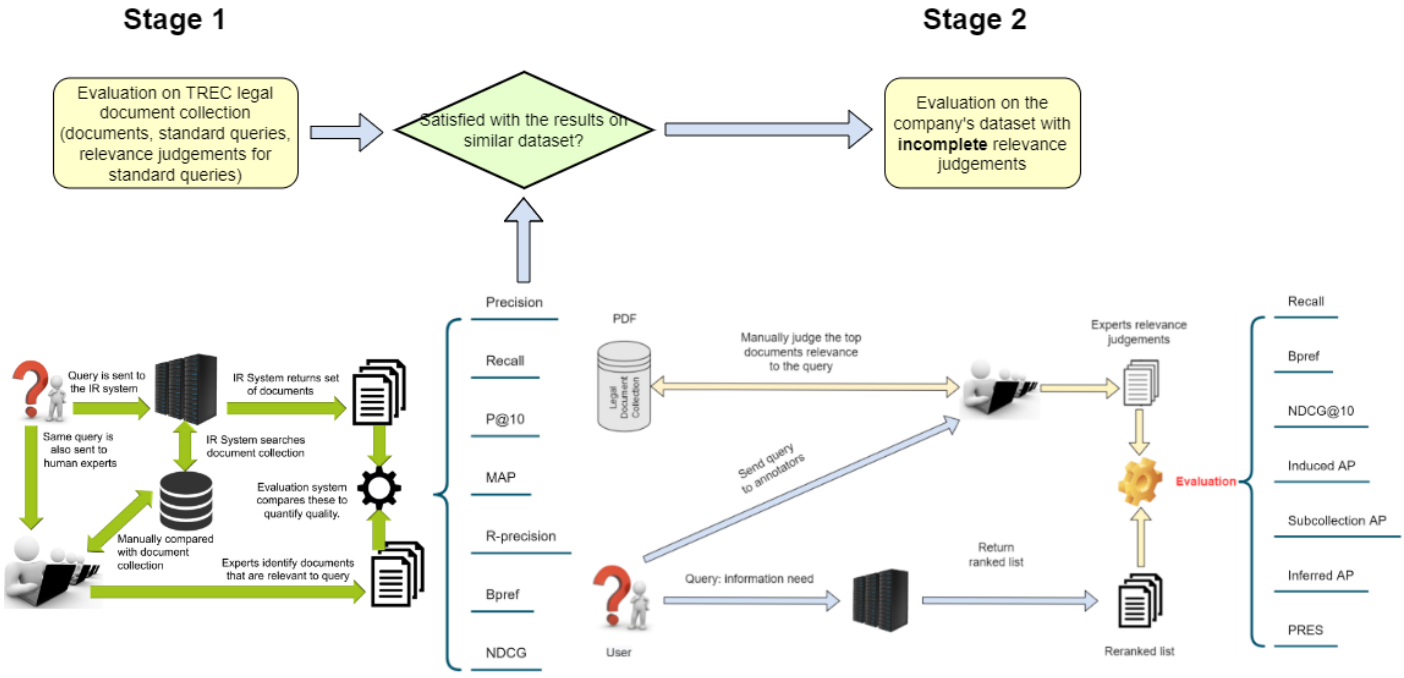

Figure 15: Evaluation Organization

9.2 Evaluation Using TREC

Since I have relatively complete relevance to the TREC dataset, I can evaluate the performance using precision, recall, p@10, map, R-precision, bpref and NDCG (see Figure 16).

Evaluation (the purpose of evaluating the effectiveness of an IR technique is to evaluate the quality of this ranked list)								
	Formula	Notes	Benefits	High	Application	Ideal	Disadvantage	Improvement and Step
Precision	Retrieved relevant/ relevant	Inversely related to recall	High when good at avoid non-relevant documents	High precision when returning very few documents	Web (high precision and recall does not need to be high)	Both precision and recall is 100% —- answer set = relevant set	Set-based Unranked Require two metrics	
Recall	Retrieved relevant/ retrieved	Inversely related to precision	High when finds many relevant documents	100% recall by retrieving all documents in the collection	**Patent lawyer (high recall and tolerate low precision - read through some non-relevant documents)**	Both precision and recall is 100% —- answer set = relevant set	Huge documents collection makes impossible to calculate recall accurately Set-based Unranked Require two metrics	
Precision at n (P@n)	Retrieved relevant before n/n	N is fixed for all queries Single value metric	Easy to interpret Easy to compute	Top n has many relevant documents		Users will only examine the top-K results	Performance is affected by number of relevant documents available (10, p@10 is difficult; 1000, p@10 is easy) Ranking of documents within the top K is inconsequential How to choose n?	Combination of easy queries and difficult queries
R-precision	Retrieved relevant before relevant number /relevant number	N is not fixed for all queries because the number of relevant documents will be different for each query. Single value matric						
Mean average map (MAP)		Single value metric based on precision	Reward system that rank relevant documents at te beginning of the returned results —- do not treat documents equally Continue to examine later stages of the ranked list				Still need to know the number of relevant documents for retrieval (not clear)	Calculate the precision at each recall point where a relevant document is found Mean average precision: calculate the mean of the average query precision (for multiple queries) Average precision: divide the sum the above precision by total number of relevant documents

Evaluation						
	Size and Relevance judgements	Unjudged documents	Problem	Idea	Unusual/Features	Reward
Traditional evaluation metrics	Small collection Complete relevance judgements	Assumption: unjudged documents are non-relevant	1. Unjudged documents lowers the evaluation score. But it does not mean the retrieval is worse: whether a document is relevant or not is not affected by whether someone has judged it. This is a problem because evaluation scores no longer accurately reflect the effectiveness of retrieval. 2. Binary relevance judgements		MAP: give less weight to relevant documents found late in the ranking (similar to NDCG, but bpref cannot do that)	
Bpref (binary preference)	Small/large collection Incomplete relevance judgements	Ignore unjudged documents. The only documents considered are those that were judged relevant or judged non-relevant.	Binary relevance judgements	Unjudged documents should not impact so largely on the evaluation score.	If R is small (there are only one or two relevant documents) it fails. Solution: use bpref10	
NDCG		Assumption: unjudged documents are non-relevant	CG: documents late in the list can add as much as to the gain as documents early in the list. -> DCG DCG: not suitable 1. Too many data points. 2. Difficult to compare. 3. Most evaluation metrics give a single value in range between 0 and 1 -> NDCG	Graded relevance judgements CG: each relevant documents should add to the gain More highly ranked documents add more to the gain than less relevant documents DCG: later documents add less to the gain than earlier ones DCG: 1. More relevant documents contribute more to gain. 2. Relevant documents found earlier in the ranked list also contribute more to gain. NDCG is calculated by comparing DCG vector against an Ideal DCG vector IDCG: same length as the DCG vector (with 0 relevance values inserted at the end if there are not enough relevant documents)	Combines document ranks and graded relevance judgements. Single measure of quality at any rank without needing to know recall. NDCG@n only considers relevant documents found to that point: not affected by many relevant documents found very late. Give less weight to relevant documents found late in the ranking. (similar to MAP, but bpref cannot do not)	Rank highly-relevant documents ahead of mildly relevant ones Position relevant documents in early positions in the ranking

Figure 16: Evaluation Metrics

9.3 Evaluation Metrics for Incomplete Relevance Judgements

9.3.1 Recall

This system aims at higher recall owing to the importance of identifying all documents relevant to a query. Recall reflects some system's behaviour that MAP does not. For instance, even if many relevant documents have been retrieved, MAP rewards system returns relevant documents at the top but does not consider those ranked later. While the high recall is easy to achieve by returning most documents, I aim to **maximise recall** with the **minor user effort** (Magdy & Jones 2010). Therefore, I use recall jointly with other evaluation metrics to evaluate my system more comprehensively because recall itself cannot indicate the quality of ranking of the retrieved results.

9.3.2 Bpref

Since complete relevance judgements are not possible for **large-scale** and **dynamic** document collection, unjudged documents are regarded as irrelevant, which may lower the evaluation score. However, this does not indicate worse retrieval because the relevance is unaffected by whether it is judged. To avoid failure when few documents are relevant to a query, I use **bpref@10**. Figure 17 shows bpref is relatively stable with incomplete relevance judgements by ignoring unjudged documents, which is consistent with this task.

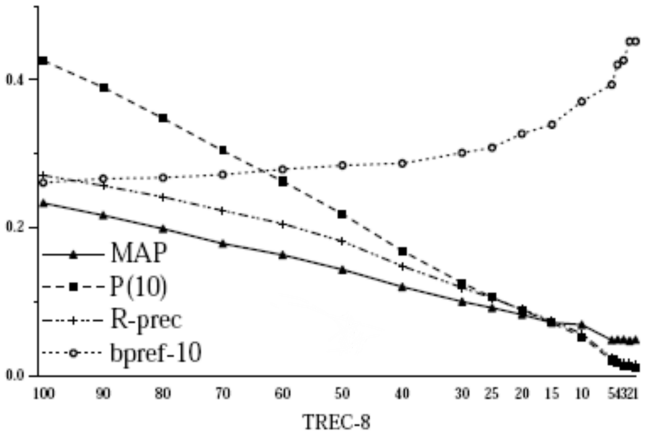

Figure 17: Stable bpref

However, unlike MAP, bpref does not explicitly give less weight to relevant documents found late in the ranking. With bpref, the lower score of later documents is not due to the actual rank itself but to the higher likelihood of non-relevant documents above it.

9.3.3 Normalized Discounted Cumulative Gain (NDCG@10)

Here are reasons to use NDCG@10.

- Graded relevance judgement rewards my system for ranking highly relevant documents ahead of partially relevant ones (Sakai & Kando 2008) by giving less weight to documents appearing late in the ranking (similar to MAP).
- Rank relevant documents in the early position.
- A single quality measure at rank without knowing recall.
- Only considers relevant documents found to that point unaffected by many relevant documents ranked late.

Specifically, the DCG is calculated up to rank ten as usual, and IDCG is calculated within the top 100 returned documents because they have been judged.

9.3.4 Induced Average Precision (indAP)

Induced AP is calculated similarly to average precision, except unjudged documents are removed from the original ranked list to form a condensed list and therefore not evaluated. (Yilmaz & Aslam 2006).

9.3.5 Subcollection AP (subAP)

However, induced AP and bpref overlook unjudged document information that may be indicative. I use subAP and infAP to alleviate it. SubAP exploits extra information on the documents in the depth-100 pool (even if they are not assessed) and the proportion p of the depth-100 pool that is judged. However, subAP performs better for partial relevance judgements than indAP and bpref, and it approximates actual AP more closely (Yilmaz & Aslam 2006). That is why I use it as an evaluation metric.

9.3.6 Inferred Average Precision (infAP)

The whole collection average precision is directly estimated from the pool subsample using inferred AP. InfAP is very stable with various degrees of incompleteness and does not fluctuate greatly (Yilmaz & Aslam 2006), so it has become my choice.

9.3.7 PRES

Although indAP, subAP, infAP and bpref are relatively robust to incomplete judgments, they emphasise retrieval precision, which is not the goal of the recall-oriented retrieval task. Consequently, I will also employ a patent retrieval evaluation score (PRES) **specially designed for recall-oriented IR tasks with incomplete relevance judgements**. Since PRES perform well even with few relevant judgements (Magdy & Jones 2010), I choose PRES.

9.4 Quick Query Retrieval Guarantee

Experiments of different queries show that retrieval within two seconds (efficiency) can be guaranteed.

10 Dynamic Document Collection

Since the collection will be continuously updated (relevance judgements for removed documents remain **imperfect** (Nuray & Can 2003), terms must be updated with the dictionary and posted lists. (StanfordGroupNLP n.d.)

Since legal staff must not ignore useful documents, periodic index reconstruction with delay is inappropriate. Fortunately, the previously used **inverted index with sorted doc ID and alphabetic** makes index reconstruction more **efficient**. My system maintains two indexes to update the index rapidly: a large **main index** and a **small supplementary** index for updated terms in new documents. Searches across both indexes are **merged**. Moreover, a **bit vector** stores deleted documents to be removed from the ranking results. When the supplementary index becomes excessively huge, it is **combined** with the main index. Admittedly, dynamic indexing suffers from its **complexity** (StanfordGroupNLP n.d.).

11 Conclusion

In conclusion, this report describes the overall design with justified reasons and tradeoffs of a Computer-Based Legal Information Retrieval System. Firstly, documents are preprocessed before being retrieved using BM25 and query likelihood. Then my system combines the result from each input system into a single ranked list using score-based fusion CombMNZ. Subsequently, by analysing the contextual information, I apply BERTopic with legal-domain knowledge to rerank the top documents in the fused list. Finally, I explain how the evaluation is organised and how to handle incomplete relevance judgements.

References

Buscaldi, D., Rosso, P., Gómez-Soriano, J. M. & Sanchis, E. (2009;2010;), 'Answering questions with an n-gram based passage retrieval engine', Journal of intelligent information systems **34**(2), 113–134.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding'.

Ferilli, S. (2021), 'Automatic multilingual stopwords identification from very small corpora', Electronics (Basel) **10**(17), 2169.

Haponik, A. (2021), 'Text extraction from images using machine learning?'. `https://addepto.com/text-extraction-from-images-using-machine-learning/`.

Kanapala, A., Jannu, S. & Pamula, R. (2019), 'Passage-based text summarization for legal information retrieval', Arabian journal for science and engineering (2011) **44**(11), 9159–9169.

Ladani, D. J. & Desai, N. P. (2020), Stopword identification and removal techniques on tc and ir applications: A survey, IEEE, pp. 466–472.

Lillis, D. (2020), On the evaluation of data fusion for information retrieval, pp. 54–57.

Liu, T. (2009), 'Learning to rank for information retrieval'.

Lo, R. T., He, B. & Ounis, I. (2005), 'Automatically building a stopword list for an information retrieval system', Journal of digital information management **3**(1), 3.

Magdy, W. & Jones, G. J. F. (2010), Examining the Robustness of Evaluation Metrics for Patent Retrieval with Incomplete Relevance Judgements, Vol. 6360 of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 82–93.

Manning, C. D., Raghavan, P. & Schütze, H. (2008;2012;), Introduction to information retrieval, Cambridge University Press, New York.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), 'Efficient estimation of word representations in vector space'.

Nuray, R. & Can, F. (2003), Automatic ranking of retrieval systems in imperfect environments, ACM, pp. 379–380.

Rosso, P., Correa, S. & Buscaldi, D. (2011), 'Passage retrieval in legal texts', The journal of logic and algebraic programming **80**(3-5), 139–153.

Sakai, T. & Kando, N. (2008), 'On information retrieval metrics designed for evaluation with incomplete relevance assessments', Information retrieval (Boston) **11**(5), 447–470.

Sarica, S. & Luo, J. (2021;2020;), 'Stopwords in technical language processing', PloS one **16**(8), e0254937–e0254937.

Schweighofer, E. (2010), Semantic Indexing of Legal Documents, Vol. 6036 of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 157–169.

Silveira et al., R. (2021), 'Topic modelling of legal documents via legal-bert'.

StanfordGroupNLP (n.d.), 'Dynamic indexing'. `https://nlp.stanford.edu/IR-book/html/htmledition/dynamic-indexing-1.html`.

Thakur, N., Mehrotra, D. & Bansal, A. (2012), 'Information retrieval system assigning context to documents by relevance feedback', International journal of computer applications **58**(20), 37–47.

Yilmaz, E. & Aslam, J. (2006), Estimating average precision with incomplete and imperfect judgments, ACM, pp. 102–111.

Zhang, G., Nulty, P. & Lillis, D. (2022), 'Enhancing legal argument mining with domain pre-training and neural networks'.

Zheng, Z., Hui, K., He, B., Han, X., Sun, L. & Yates, A. (2020), 'Bert-qe: Contextualized query expansion for document re-ranking'.

Zhiying, J., Raphael, T., Ji, X. & Jimmy, L. (2021), 'How does bert rerank passages? an attribution analysis with information bottlenecks'.