Indian Statistical Institute

Explanability of Query Expansions

M.Tech Disseration

Computer Science

Aditya Dutta

Indian Statistical Institute

Kolkata, 2025-06-11

Supervised by
Prof. Mandar Mitra

Abstract

Query Expansion (QE) techniques aim to mitigate vocabulary mismatch in Information Retrieval by augmenting user queries with related terms. However, their effectiveness varies across queries. This work investigates the explainability of QE by introducing the concept of an Ideal Expanded Query (IEQ): a hypothetical query yielding near-perfect retrieval performance, measured via Average Precision (AP). We hypothesize that the closer a QE variant is to the IEQ, the higher its AP. We generate multiple QE variants using methods like RM3, SPL, CEQE, and Log-Logistic, and compare them to IEQs constructed using Oracle Rocchio tuning and Logistic Regression.

The codebase for this project is located at: https://github.com/mrishu/py-qe-explain

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Information Retrieval

When we type a few words into a search bar: "best sci-fi movies", "laptop overheating", or "how to train a neural network", we expect the system to understand what we mean and return the most useful documents. **Information Retrieval (IR)** is the field of computer science that deals with this task: retrieving relevant information from large collections of unstructured data, typically textual documents.

IR systems are used everywhere, from web search engines to digital libraries, recommendation systems, and various databases. Unlike structured databases like SQL where answers to the queries as well as the queries themselves are explicit and exact, IR systems aim to **rank** documents based on their **relevance** to a given query.

1.1 The Retrieval Process

At the heart of IR is the simple interaction between **queries** and **documents**. A **query** is a short string of text representing a user's *information need*. This could be as concise as "black hole evaporation" or as vague as "best movies". The system's job is to retrieve and rank documents (e.g., web pages, articles, papers) from a large **corpus** so that the most relevant ones appear at the top.

This is done in a few core steps:

(i) **Indexing**: Preprocess the corpus (tokenization, stopword removal, stemming/lemmatization), and build an inverted index mapping terms to documents.

(ii) **Scoring**: Given a query, compute a **relevance score** for each document in the corpus.

(iii) **Ranking**: Return the top-k documents based on those scores.

The effectiveness of an IR system depends crucially on the **scoring model** used. We discuss two such models below, namely TF-IDF and BM25.

1.2 Vector Space Model

One of the foundational models in IR is the **Vector Space Model**. In the Vector Space Model, both queries and documents are represented as vectors in a high-dimensional space, where each dimension corresponds to a term in the vocabulary. The relevance score between a query and a document is then typically computed using **dot product** or **cosine similarity** (which is the angle between their respective vectors).

Definition 1.1. (Cosine Similarity): Let $\boldsymbol{A} = (A_1, A_2, ..., A_n)$ and $\boldsymbol{B} = (B_1, B_2, ..., B_n)$ be vectors. Then,

$$\text{CosineSimilarity}(\boldsymbol{A}, \boldsymbol{B}) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\|\boldsymbol{A}\| \, \|\boldsymbol{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

1.3 Term Weighting Schemes

1.3.1 tf-idf

To improve on just term frequency matching, **term weighting schemes** like **tf-idf** are used:

- **tf** reflects how often a term occurs in a document.

- **idf** reflects how rare the term is across the corpus.

The **term frequency** (tf) of a term t in a document D is typically defined as:

$$\mathrm{tf}(t, D) = f(t, D),$$

where:
- $f(t, d)$: Raw count of how many times term t appears in document d.

The **inverse document frequency** (idf) is defined as:

$$\mathrm{idf}(t) = \log\left(\frac{N - n_t + 0.5}{n_t + 0.5} + 1\right),$$

where:
- $\mathrm{idf}(t)$: Inverse document frequency of term t
- N: Total number of documents in the collection
- n_t: Number of documents in which term t appears

And the relevance score of a document D for a query Q is given by:

$$\text{tf-idf}(D, Q) = \sum_{t \in Q} \mathrm{tf}(t, D) \cdot \mathrm{idf}(t).$$

However, tf-idf has limitations and may not perform very well for retrieval.

1.3.2 BM25

This led to the development of **BM25**, a retrieval model that has become one of the standard baselines in IR. BM25 scores a document D for a query Q as:

$$\mathrm{BM25}(D, Q) = \sum_{t \in Q} \mathrm{idf}(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\mathrm{avgdl}}\right)},$$

where:
- $f(t, D)$: term frequency of term t in document D
- $|D|$: length of document D
- avgdl: average document length in the corpus
- k_1, b: hyperparameters (commonly $k_1 = 1.2$, $b = 0.75$).

1.3.3 Back to the vector space model

In both of these models, the expression that occurs inside the summation can be regarded as the weight of then term t in document D, which will be important in our work.

$$\mathrm{BM25}(D, t) = \mathrm{idf}(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\mathrm{avgdl}}\right)}.$$

Here, $\mathrm{BM25}(D, t)$ represents the weight of term t in document D when document D is considered as a vector in the Vector Space Model.

1.4 Evaluation

1.4.1 Unranked Evaluation

In **unranked retrieval**, the system returns a **set** of documents without any particular order. Two standard metrics used are:

- **Precision**: Measures the fraction of retrieved documents that are actually relevant.

$$\text{Precision} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Retrieved}|}$$

- **Recall**: Measures the fraction of relevant documents that were successfully retrieved.

$$\text{Recall} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Relevant}|}$$

These metrics are important when assessing systems that produce binary outputs (relevant or not), but they don't capture ranking information.

1.4.2 Ranked Evaluation

In most real-world scenarios, retrieval systems return a **ranked list** of documents. For such systems, **Mean Average Precision (MAP)** is a preferred evaluation metric.

Average Precision (AP)

For a single query, **Average Precision (AP)** is defined as the average of the precision values at the ranks where relevant documents appear.

Definition 1.2. (Average Precision (AP)): Let us assume that the top-N retrieved documents were listed using a information retrieval model. Now let,

- $R \subseteq \{1, 2, ..., N\}$ be the subset of ranks where relevant documents are found.
- $P@k$ denote the precision at rank k, which is the precision considering only top-k retrieved documents.

Then, Average Precision is defined as:

$$\text{AP} = \frac{1}{|R|} \sum_{k \in R} P@k.$$

Mean Average Precision (MAP)

For a set of queries, **Mean Average Precision (MAP)** defined as the mean of the APs across all queries.

Definition 1.3. (Mean Average Precision (MAP)): Given a set of queries $Q = \{q_1, q_2, ..., q_M\}$, the **MAP** is the mean of the APs across all queries:

$$\text{MAP} = \frac{1}{M} \sum_{i=1}^{M} \text{AP}(q_i).$$

Chapter 2

Dataset Used

The **Robust 2004** dataset is a benchmark collection which consists of a large set of news articles and government documents designed to evaluate the robustness of information retrieval models.

The dataset includes:
(i) 250 topics (queries and their descriptions),
(ii) a corpus of 528,155 documents,
(iii) relevance information for each query (in a `qrel` file).

> The TREC Robust retrieval task focuses on "improving the consistency of retrieval technology by focusing on poorly performing topics."

Remark 2.1. (Dataset Link): https://ir-datasets.com/trec-robust04.html

Chapter 3

Vocabulary Mismatch Problem and Query Expansion

3.1 Vocabulary Mismatch Problem

The **vocabulary mismatch problem** occurs when users and relevant documents use different words to express the same concept. This is a major hurdle in information retrieval. Even if a document is relevant, it may not be retrieved simply because it doesn't share the same vocabulary as the query.

Example 3.1. (Vocabulary Mismatch): A user searches for: `laptop overheating`.

But the relevant documents might use terms like: `thermal throttling`, `cooling issues`, or `fan problems`.

Since these terms don't exactly match the user's query, the documents may not appear in search results. This hurts the system's recall.

3.2 What is Query Expansion?

Query Expansion (QE) aims to solve this problem by adding related words or phrases to the user's original query. The goal is to capture different ways the same idea might be expressed in the document collection, improving recall and precision.

Example 3.2. (Query Expansion): Original Query: `laptop overheating`

After Query Expansion: `laptop overheating thermal throttling fan problems cooling issues`.

This reformulated query is more likely to match relevant documents.

3.3 Query Expansion Techniques

Several algorithms exist for expanding queries. These techniques can be broadly categorized into:

3.3.1 Relevance Feedback (RF)

Relevance Feedback methods use **user input**. The user manually marks some retrieved documents as relevant or not relevant. The system then uses this feedback to improve the query.

Pros:
- High-quality feedback
- Personalized improvements

Cons:
- Requires user interaction
- Slower in real-world systems

Definition 3.3. (Rocchio Relevance Feedback): The **Rocchio algorithm** is a classic relevance feedback technique used to improve search queries in vector space models. It updates the original query vector based on user-marked relevant and non-relevant documents:

$$\boldsymbol{q}_{\text{new}} = \alpha \boldsymbol{q}_{\text{orig}} + \frac{\beta}{|D_r|} \sum_{\boldsymbol{d}_i \in D_r} \boldsymbol{d}_i - \frac{\gamma}{|D_{\text{nr}}|} \sum_{\boldsymbol{d}_j \in D_{\text{nr}}} \boldsymbol{d}_j$$

Here:
- $\boldsymbol{q}_{\text{new}}$ is the modified query vector.
- $\boldsymbol{q}_{\text{orig}}$ is the original query vector.
- D_r and D_{nr} are sets of relevant and non-relevant documents.
- α, β, and γ are hyperparameters that balance the contributions.

The Rocchio method effectively pushes the query vector closer to relevant documents and away from non-relevant ones in the vector space.

3.3.2 Pseudo-Relevance Feedback (PRF)

Pseudo-Relevance Feedback assumes that the **top-ranked documents** returned by an initial query are likely relevant. It uses them to find expansion terms, without requiring explicit user feedback.

Pros:
- Fully automatic
- Works in most real-world systems

Cons:
- May propagate errors if many documents in the top results are non-relevant

3.3.3 Thesaurus-Based Expansion

This method uses resources like WordNet or domain-specific thesauri to expand a query with synonyms or related terms.

For example, a query containing `car` may be expanded to include `automobile`, `vehicle`, or `motorcar`.

Pros:
- Fully automatic after the thesaurus has been created
- Simple and interpretable

Cons:
- May introduce noise due to context-insensitive synonyms
- Limited by the coverage of the thesaurus

3.3.4 Co-Occurrence Analysis

This technique expands queries using terms that frequently co-occur with the query terms in a large corpus. It assumes that terms that appear together in similar contexts are semantically related.

Example: If `virus` often co-occurs with `infection`, `flu`, and `symptom` in documents, these terms may be good candidates for expansion.

Pros:
- Fully automatic
- Reflects corpus-specific language usage

Cons:
- May include spurious associations

3.4 QE Techniques Used in This Work

Here are the query expansion methods analyzed in this dissertation:

• **RM3 (Relevance Model 3)** A probabilistic **pseudo-relevance feedback** method: it takes the top search results, finds commonly occurring terms with the original query, and adds those terms to improve retrieval.

• **SPL (Smooth Power-Law)** A statistical method that models how often words appear using a **smoothed power-law distribution**. It selects useful terms that follow this natural frequency pattern, helping to catch important but less frequent words.

• **Log-Logistic Model** Uses a **log-logistic distribution** to analyze word frequency patterns. It chooses expansion terms whose occurrence matches the statistical behavior of relevant terms.

• **CEQE (Contextualized Embeddings for Query Expansion)** A modern neural method using models like **BERT** to understand the query's meaning. It finds context-aware terms for expansion, capturing semantic nuances and improving results.

Chapter 4

Problem Statement and Hypothesis

4.1 Problem Statement

As we saw, many query expansion (QE) algorithms exist, each with its own term selection and weighting schemes. It has been observed that different methods perform better for some queries and worse for others. For certain queries, average precision (AP) improves after expansion using a particular algorithm, while for others, it decreases.

The **goal of this work** is to understand, why certain methods perform well for some queries but not others, in a clear and interpretable way.

4.2 Hypothesis

We propose that for each query, there exists an **Ideal Expanded Query (IEQ)** that yields nearly perfect performance (Average Precision close to 1).

If a real QE method produces a query that is **close** to this IEQ (in some quantitative sense), then it will have higher AP. Conversely, methods that produce queries farther from the IEQ will perform worse.

4.3 Overall Setup

4.3.1 QE variants

(i) We generated **80 QE variants** ($4 \times 4 \times 5 \times 1 \times 1 = 80$) using this parameter grid:

- **Expansion methods** (`expansion_method`): `[rm3, ceqe, loglogistic, spl]`
- **Number of top documents** (`num_top_docs`): `[10, 20, 30, 40]`: how many retrieved documents are used for feedback
- **Number of expansion terms** (`num_exp_terms`): `[15, 25, 35, 45, 55]`: how many new terms are added to the query
- **Mixing parameter** (`mixing_param`): `0.5`: fixed weight for mixing the original query with expansion terms

Each method also uses a specific `tuning_parameter`:

Method	Parameter Name	Value
`rm3`	Feedback weight β	0.6
`ceqe`	Embedding-context weight	0.6
`loglogistic`	Distribution shape/scale	2
`spl`	Smooth power-law factor	8

Each variant is identified using the format which will be it's `runid`:

```
<expansion_method>-<num_exp_terms>-<num_top_docs>-<tuning_parameter>
```

(ii) For each variant,

- We generated the `term_weights` file which contains the term with their corresponding weights for each query in
`<qid> <term> <weight>` format.

- We generated the run file using top-1000 BM25 retrieval in `<qid> Q0 <docid> <rank> <score> <runid>` format.
- Using trec_eval we generated an ap file which contains the AP achieved by all queries in `<qid> <AP>` format.

4.3.2 Ideal Query Generation

We then generate the IEQ for each query (using algorithms described later). It also has its corresponding term_weights, run and ap files.

4.3.3 Similarity Measuring and Correlation Computation

(i) Firstly, for each query, we measure the similarity between IEQ and the QE variants (using similarities described later).
(ii) For each query we make two lists of length 80,
   - one containing the similarity of each QE variant to the IEQ,
   - the other containing the corresponding APs achieved by each QE variant.

```
qid:
  similarity_list = [sim_variant_1, sim_variant_2, ..., sim_variant_80]
  ap_list = [ap_variant_1, ap_variant_2, ..., ap_variant_80]
```

(iii) Then, we find the Pearson, Kendall and Spearman correlation between these lists.
(iv) Finally, we take the average of these similarities across all 250 queries and report the average Pearson, Kendall and Spearman correlations.

According to our hypothesis, we expect high correlations between these two lists.

Chapter 5

Similarity Measures

We now describe the similarity measures that we have used in our work to find the similarity between IEQ and a QE. From now, we will assume that both IEQ and QE are vectors in the Vector Space Model.

5.1 Cosine Similarity

Definition 5.1. (Cosine Similarity):

$$\mathrm{L2}(\mathbf{IEQ}, \mathbf{QE}) = \frac{\mathbf{IEQ} \cdot \mathbf{QE}}{\|\mathbf{IEQ}\|_2 \; \|\mathbf{QE}\|_2}$$

Henceforth, this will be referred to as `l2_similarity`.

Remark 5.2. (Dot product): Note that,

$$\mathbf{IEQ} \cdot \mathbf{QE} = \sum_{t \in \, \mathbf{IEQ} \, \cap \, \mathbf{QE}} \mathrm{IEQ}_t \cdot \mathrm{QE}_t,$$

where IEQ_t and QE_t denote the weights of the term t in **IEQ** and **QE** respectively.

Remark 5.3. (L2 norm): Note that,

$$\|\mathbf{QE}\|_2 = \sum_{t \in \, \mathbf{QE}} \sqrt{\mathrm{QE}_t^2}$$

Similarly for **IEQ**.

5.2 Cosine Similarity variant normalized by L1 norm

Definition 5.4. (Cosine similarity variant normalized by L1 norm):

$$\mathrm{L1}(\mathbf{IEQ}, \mathbf{QE}) = \frac{\mathbf{IEQ} \cdot \mathbf{QE}}{\|\mathbf{IEQ}\|_1 \; \|\mathbf{QE}\|_1}$$

Henceforth, this will be referred to as `l1_similarity`.

Remark 5.5. (L1 norm): Note that,

$$\|\mathbf{QE}\|_1 = \sum_{t \in \, \mathbf{QE}} |\mathrm{QE}_t|, \text{where } |\cdot| \text{ denotes the absolute value.}$$

Similarly for **IEQ**.

5.3 Jaccard Similarity

Definition 5.6. (Jaccard Similarity):

$$J(\mathbf{IEQ}, \mathbf{QE}) = \frac{|\mathbf{IEQ} \cap \mathbf{QE}|}{|\mathbf{IEQ} \cup \mathbf{QE}|}.$$

Here, $|\cdot|$ denotes the cardinality of a set.

Henceforth, this will be referred to as `jaccard_similarity`.

5.4 Modified nDCG similarity

Assume that **IEQ** and **QE** are ranked by weights and arranged in descending order.

Definition 5.7. (nDCG similarity):
Let,

$$\mathrm{DCG} = \sum_{t \in \mathbf{IEQ} \cap \mathbf{QE}} \frac{\mathrm{IEQ}_t \times 1000}{1000 + \mathrm{Rank}_{\mathbf{QE}}(t) + 1}$$

and

$$\mathrm{IDCG} = \sum_{i=1}^{|\mathbf{QE}|} \frac{\mathbf{IEQ}[i] \times 1000}{1000 + i + 1}.$$

Then,

$$\mathrm{nDCG}(\mathbf{IEQ}, \mathbf{QE}) = \frac{\mathrm{DCG}}{\mathrm{IDCG}}.$$

Here, $\mathrm{Rank}_{\mathbf{QE}}(t)$ denotes the rank of term t in **QE**.
And $\mathbf{IEQ}[i]$ denotes the weight of the i^{th} term in **IEQ**.

Henceforth, this will be referred to as `n2_similarity`.

Chapter 6

Ideal Query Generation

6.1 Oracle Rocchio Vector tuning

We present the first algorithm for generating an **Ideal Expanded Query (IEQ)**:

(i) This method first constructs the *oracle* **Rocchio Vector** using the ground truth, which is obtained from the `qrel` file. See Definition 3.3. The document vectors are assumed to have BM25 weights.

(ii) We sort the terms of the Rocchio vector by their weights in decreasing order and trim the Rocchio vector upto top `num_expansion_terms`.

(iii) We then select a `tweak_magnitude` from a `list_of_magnitudes`.

(iv) We iteratively go over the terms of the Rocchio vector one by one and tweak its weight by `new_weight` = (1 + `tweak_magnitude`) × `current_weight`.
If the AP after tweaking is higher than the AP before tweaking, we accept the tweak. Otherwise we revert it.

(v) After we have gone over all the terms, we select the next `tweak_magnitude` and repeat this process.

Algorithm 1: Oracle Rocchio Tuning

```
Input: qid, list_of_magnitudes, num_expansion_terms, ground_truth
# The ground_truth is the qrel information.
# It can be a dictionary which maps qid → dictionary(docid → relevance).
Construct oracle Rocchio Vector rocchio_vector for query ID qid using ground_truth.
# rocchio_vector is a dictionary which maps term → weight.
rocchio_vector.sort_by_weight(reverse=True) # sort by weights in decreasing order
rocchio_vector.trim(num_expansion_terms) # trim upto top num_expansion_terms terms
current_AP = computeAP(rocchio_vector, ground_truth)
for tweak_magnitude in list_of_magnitudes:
    for term in rocchio_vector:
        current_weight = rocchio_vector[term]
        rocchio_vector[term] = (1 + tweak_magnitude) × current_weight
        nudged_AP = computeAP(rocchio_vector, ground_truth)
        if nudged_AP >= current_AP:
            current_AP = nudged_AP
        else:
            rocchio_vector[term] = current_weight
        endif
    end for
end for
return rocchio_vector
```

Remark 6.1. (AP Computation): The `computeAP(rocchio_vector, ground_truth)` function first retrieves the top 1000 results for the given `rocchio_vector` query using BM25 retrieval. It then creates a temporary `run` file and compares the retrieved results to the `ground_truth` using a tool like `trec_eval` to compute and return the AP.

The hyperparameters mentioned were taken to be:
- For *oracle* Rocchio vector construction, $\alpha = 2.0, \beta = 64.0, \gamma = 64.0$.
- `list_of_magnitudes` $= [4.0, 2.0, 1.0, 0.5]$.
- `num_expansion_terms` $= 200$.

Henceforth, this method will be referred to as **IEQ0**.

Remark 6.2. (IEQ0 is computationally expensive): Since, generation of IEQ0 involves retrieval multiple times, it is computationally expensive. Hence, we limit it to `num_expansion_terms`=200.

6.2 Logistic Regression based Ideal Query Generation

We now present the second algorithm for generating IEQs:

(i) Read the `qrel` file to collect the relevant and non-relevant documents for query ID `qid`.
(ii) Construct document vectors for each of the relevant and non-relevant documents using BM25 weights.
(iii) Construct design matrix $\boldsymbol{X}$ by stacking all the relevant and non-relevant document vectors.
(iv) Construct target label $\boldsymbol{y}$, which is a vector of ones and zeros, where $\boldsymbol{y}_i = 1$ if the ith document is relevant and $\boldsymbol{y}_i = 0$ otherwise.
(v) Do feature selection using Variance Thresholding (where we eliminate terms/features having variance less than `variance_threshold`). After that, we again do feature selection using χ^2 test and select the `num_after_chi2_terms` - best terms.
(vi) Fit a Logistic Regression model on the data $\boldsymbol{X}$ and $\boldsymbol{y}$.
(vii) Finally, we use the Logistic Regression model coefficients as weights and their corresponding terms in our IEQ.

Remark 6.3. (Negative coefficients): Since the model can have negative coefficients, we simply ignore them during retrieval. We select the highest positive `num_expansion_terms` number of terms and use that during retrieval.

Algorithm 2: IEQ USING LOGISTIC REGRESSION

Input: `qid`, `qrel`, `variance_threshold`, `num_after_chi2_terms`, `num_expansion_terms`

Extract relevant and non-relevant documents for `qid` from `qrel` and construct BM25-weighted document vectors

Build design matrix $\boldsymbol{X}$ by stacking all document vectors

Build label vector $\boldsymbol{y}$: $y_i = 1$ if document i is relevant, else 0

 # Apply feature selection using variance thresholding and χ^2 test

Apply `VarianceThreshold(threshold=variance_threshold)` on $\boldsymbol{X}$

Apply `SelectKBest(chi2, k=num_after_chi2_terms)` on the reduced $\boldsymbol{X}$

*# Fit **LogisticRegression** model*

`model = LogisticRegression(solver="liblinear", penalty="l2").fit(`$\boldsymbol{X}, \boldsymbol{y}$`)`

Let `coef = model.coef_` *# extract the coefficients of trained logistic regression model*

Select top `num_expansion_terms` terms with highest positive coefficients

return `query_vector` with selected terms and corresponding weights from `coef`

Remark 6.4. (Keeping care of terms during feature selection): During feature selection, the terms which are being eliminated and rearranged need to be taken care of. The order of coefficients must correspond to their terms, otherwise it will be incorrect.

Note 6.5. (Motivation): In logistic regression, the classification decision is based on the value of $\sigma(w \cdot x + b)$. Since relevant documents are labeled 1, the model ideally pushes $w \cdot x + b \gg 0$ for relevant instances. My intuition was:

- If we substitute $x = w$, then the model output becomes $\sigma(w \cdot w + b) = \sigma(\|w\|^2 + b)$, which should be close to 1 if $\|w\|^2$ is large.
- Hence, w can be interpreted as a pseudo-relevant document, or a good representation of the relevant set, thereby making it a candidate for an *Ideal Query*.
- $\sigma(b)$ gives the probability of an empty document being relevant (as $\sigma(w \cdot x + b) = \sigma(b)$ when $x = 0$).
  I expected $b \approx 0$ or $\sigma(b) \approx 0.5$, because this means that the model cannot distinguish an empty document to be relevant or non-relevant.
  Empirically, I observed that $|b|$ was small for all queries.

This motivation aligned with the results but might be flawed.

The hyperparameters mentioned were taken to be:

- `variance_threshold` = 10^{-4}.
- `num_after_chi2_terms` = 10000.
- `num_expansion_terms` = 1000 or 200.

Henceforth, this method will be referred to as **IEQ1**.

6.3 MAP comparison

Table 1: MAP of IEQs

IEQ	`num_expansion_terms`	MAP across 250 queries
Untweaked Oracle Rocchio	200	0.5121
Untweaked Oracle Rocchio	1000	0.5465
IEQ0	200	0.8919
IEQ1	1000	0.9026
IEQ1	200	0.8197

We see both IEQ0 and IEQ1 achieve very high MAPs.

In the figure below, we have plotted,
AP achieved on individual queries *vs.* ln(No. of relevant documents for the query)
for both IEQ0 and IEQ1 (with `num_expansion_terms=1000`).

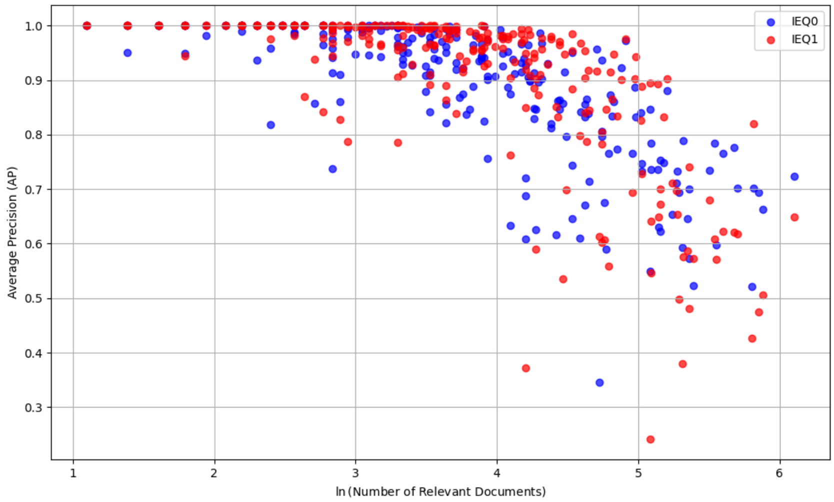

Figure 1: AP *vs.* ln(No. of relevant documents) for IEQs

The main observation is that in both cases:
IEQ performs worse as the number of relevant documents increases.

Chapter 7

Correlation results for IEQ0 and IEQ1

The average Pearson, Kendall and Spearman correlation coefficients across 250 queries are listed below for each similarity type; for both IEQ0 and IEQ1 (with `num_expansion_terms=1000`).

7.1 `l2_similarity`

Table 2: Average Correlations for IEQ0 and IEQ1 (using `l2_similarity`)

Correlation Name	**IEQ0**	**IEQ1**
Pearson	0.4858	0.4697
Kendall	0.3762	0.3607
Spearman	0.4765	0.4521

7.2 `l1_similarity`

Table 3: Average Correlations for IEQ0 and IEQ1 (using `l1_similarity`)

Correlation Name	**IEQ0**	**IEQ1**
Pearson	−0.2100	0.3472
Kendall	−0.1643	0.2594
Spearman	−0.2079	0.3351

7.3 `jaccard_similarity`

Table 4: Average Correlations for IEQ0 and IEQ1 (using `jaccard_similarity`)

Correlation Name	**IEQ0**	**IEQ1**
Pearson	0.3285	0.2550
Kendall	0.2526	0.2119
Spearman	0.3334	0.2831

7.4 `n2_similarity`

Table 5: Average Correlations for IEQ0 and IEQ1 (using `n2_similarity`)

Correlation Name	**IEQ0**	**IEQ1**
Pearson	0.2746	0.2897
Kendall	0.2280	0.2238
Spearman	0.2975	0.3000

- We observe weak and weakly moderate correlations.
- Except for `l1_similarity`, correlations for both IEQ0 and IEQ1 are almost identical for all other similarities.
- It is unclear why correlations using `l1_similarity` are so different in IEQ0 and IEQ1.

Chapter 8

Using restricted ground truth for IEQ0 generation

8.1 Problem of overfitting

After looking at the individual terms of IEQ0, we realise that it contains many terms that do not occur in any of the QEs.

- It might be that the algorithm is **overfitting** to the relevant documents.
- To mitigate this, we decide to form the initial *oracle* Rocchio vector using **restricted ground truth**.

Construction of **restricted ground truth**:

(i) To construct restricted ground truth, we first generate the `run` file using simple top-1000 BM25 retrieval. We use the original queries given in the dataset for this step.

(ii) We then form a `restricted_qrel` file by going over the original `qrel` file and for each query ID `qid`, we add the docid and its relevance to the `restricted_qrel` file, only if it occurs in the top-1000 retrieved results in the `run` file.

We follow the exact same procedure as Algorithm 1, except for Step 1, where we use `restricted_qrel` to construct the initial *oracle* Rocchio Vector.

Algorithm 3: Oracle Rocchio Tuning on Restricted Ground Truth

Input: `qid`, `list_of_magnitudes`, `num_expansion_terms`, `ground_truth`, `restricted_ground_truth`

1 Construct *oracle* Rocchio Vector `rocchio_vector` for query ID `qid` using `restricted_ground_truth`.

2 ... the rest of the algorithm remains the same ...

Henceforth, this method will be referred to as **IEQ0Restricted**.

8.2 MAP achieved

The MAP achieved by IEQ0Restricted is **0.8256**, which is lower than the MAP achieved by IEQ0 but still very high.

8.3 Correlation results

8.3.1 `l2_similarity`

Table 6: Average Correlations for IEQ0 and IEQ0Restricted (using `l2_similarity`)

Correlation Name	IEQ0	IEQ0Restricted
Pearson	0.4858	0.4246
Kendall	0.3762	0.3302
Spearman	0.4765	0.4182

8.3.2 `l1_similarity`

Table 7: Average Correlations for IEQ0 and IEQ0Restricted (using `l1_similarity`)

Correlation Name	IEQ0	IEQ0Restricted
Pearson	−0.2100	−0.207
Kendall	−0.1643	−0.1601
Spearman	−0.2079	−0.2050

8.3.3 `jaccard_similarity`

Table 8: Average Correlations for IEQ0 and IEQ0Restricted (using `jaccard_similarity`)

Correlation Name	IEQ0	IEQ0Restricted
Pearson	0.3285	0.2775
Kendall	0.2526	0.2254
Spearman	0.3334	0.2938

8.3.4 `n2_similarity`

Table 9: Average Correlations for IEQ0 and IEQ0Restricted (using `n2_similarity`)

Correlation Name	IEQ0	IEQ0Restricted
Pearson	0.2746	0.2001
Kendall	0.2280	0.2051
Spearman	0.2975	0.2582

Conclusion- We see that this doesn't result in improvement in the correlations.

Chapter 9

Pruning IEQ0 and IEQ1

We prune the Ideal Queries to remove terms which have no or detrimental impact on the AP. This helps to reduce noise and focus only on terms that help the AP positively.

9.1 Pruning IEQ0

We saw that while generating IEQ0, we have a `list_of_magnitudes` from where we pick a `tweak_magnitude` and tweak the weights of the terms in the query vector by:

```
new_weight = (1 + tweak_magnitude) * old_weight
```

If the `new_weight` for the term results in an increase in AP, we accept the tweak otherwise we revert it.

Earlier we had, `list_of_magnitudes` $= [4.0, 2.0, 1.0, 0.5]$.
We modify this list to contain a -1.0 at the end so that our
new `list_of_magnitudes` $= [4.0, 2.0, 1.0, 0.5, -1.0]$.

This has the effect that after normal IEQ0 generation, we tweak the weights of the terms by making them 0 (essentially eliminating them), and seeing if the AP increases.

Henceforth, this method will be referred to as **IEQ0Pruned**.

9.1.1 MAP achieved

Since, this process (by construction) can only improve the MAP, we achieve a slightly better MAP of **0.9060**, compared to IEQ0 which was 0.8919. Refer Table 1.

9.2 Pruning IEQ1

Pruning IEQ1 is similar:
We load the IEQ1 terms and weights and sort them by weight in decreasing order. Then, we go term by term and tweak the weight to 0. If this causes an increase in AP or the AP remains same, we accept the tweak, i.e. we eliminate the term, otherwise we revert it.

Henceforth, this method will be referred to as **IEQ1Pruned**.

> **Remark 9.1.** (Computationally Expensive): Just like the generation and pruning of IEQ0, pruning IEQ1 is also computationally expensive. Hence, we limit IEQ1 too to `num_expansion_terms`=200 for pruning.

9.2.1 MAP achieved

Compared to IEQ1 (with `num_expansion_terms`=200), which had a MAP of 0.8197, pruning helped the MAP significantly in this case, increasing it to **0.9055**. Refer Table 1.

9.3 Effect on sizes of the Ideal Expanded Queries (IEQs)

IEQ0 and IEQ1 (with `num_expansion_terms`=200) queries had a constant size of 200 terms.

Pruning had significant effects on the sizes of many of the IEQs. Many of them became very short, some of them even containing only 10-20 terms while still giving very high AP. While still some of them also had very small decreases in size.

Size of Pruned IEQs found to be highly correlated to the number of relevant documents: Interestingly, the size of IEQ0Pruned and IEQ1Pruned queries were highly positively correlated to the number of relevant documents as we see in this plot:

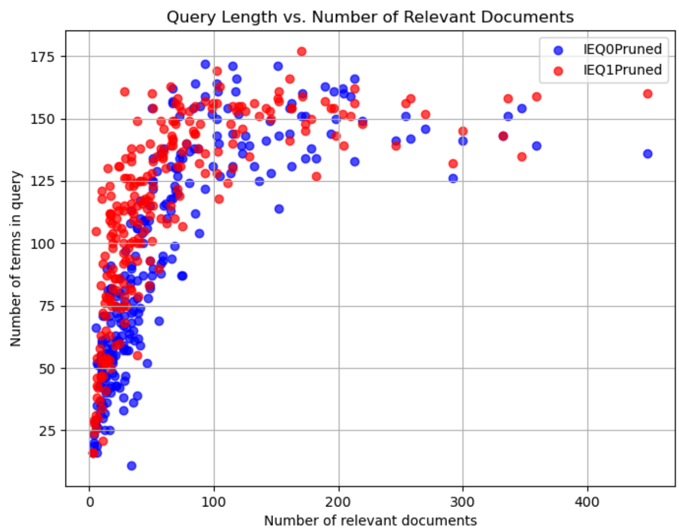

Figure 2: No. of terms in query *vs.* No. of relevant documents

9.4 Correlation Results

9.4.1 `l2_similarity`

Table 10: Average Correlations for IEQ0Pruned and IEQ1Pruned (using `l2_similarity`)

Correlation Name	IEQ0Pruned	IEQ1Pruned
Pearson	0.3451	0.4082
Kendall	0.2743	0.3121
Spearman	0.3480	0.3931

9.4.2 `l1_similarity`

Table 11: Average Correlations for IEQ0Pruned and IEQ1Pruned (using `l1_similarity`)

Correlation Name	IEQ0Pruned	IEQ1Pruned
Pearson	−0.1942	0.2847
Kendall	−0.1472	0.2070
Spearman	−0.1898	0.2722

9.4.3 `jaccard_similarity`

Table 12: Average Correlations for IEQ0Pruned and IEQ1Pruned (using `jaccard_similarity`)

Correlation Name	IEQ0Pruned	IEQ1Pruned
Pearson	0.3239	0.2591
Kendall	0.2424	0.2015
Spearman	0.3226	0.2667

9.4.4 `n2_similarity`

Table 13: Average Correlations for IEQ0Pruned and IEQ1Pruned (using `n2_similarity`)

Correlation Name	IEQ0Pruned	IEQ1Pruned
Pearson	0.2941	0.2606
Kendall	0.2346	0.1989
Spearman	0.3078	0.2622

Conclusion: As we see, this also did not result in an improve in correlations.

Bibliography