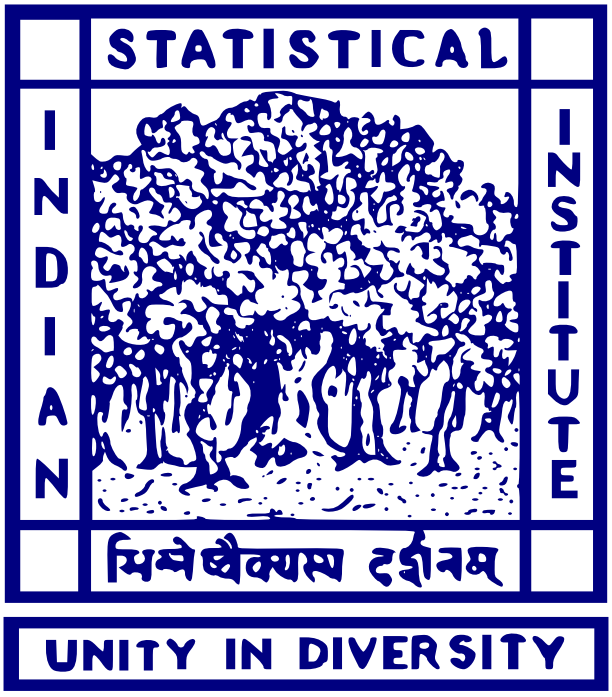

Indian Statistical Institute

Explaining Query Expansion Algorithms

M.Tech. Dissertation

Computer Science

Aditya Dutta

Indian Statistical Institute

Kolkata, 2025-06-14

Supervised by
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 leveraging 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 an Expanded Query (EQ) variant is to the IEQ, the higher its AP.

Our approach consists of three major components:

(i) generating an IEQ,
(ii) measuring the similarity between an EQ and an IEQ, and
(iii) computing the correlation between the above similarities and the corresponding AP values.

For component (i), we use Oracle Rocchio tuning and Logistic Regression to construct IEQs.
For component (ii), the following similarity measures were used:

- Cosine similarity metric,
- Modified cosine similarity normalized by L_1 norm,
- Jaccard index, and
- a modified nDCG similarity.

For component (iii), we use Pearson, Kendall's Tau and Spearman's Rho correlation coefficients.

We generate multiple EQ variants using methods like RM3, SPL, CEQE, and Log-Logistic, and compare them to constructed IEQs.

The main findings are:

(i) Using Logistic Regression to classify relevant and non-relevant documents, and then using the trained model's coefficients as expanded query weights resulted in very high MAP. This shows that this method can be potentially used to generate IEQs. One benefit of using this method is that the IEQ generation is very fast and takes only a few seconds per query.
(ii) There are multiple IEQs for any query. Various IEQs can be constructed via different procedures described in the dissertation like *IEQ0* and *IEQ1* construction. *Pruning* also modifies the input IEQ, producing a different IEQ.
(iii) Only moderate correlations were observed between the AP achieved by an EQ and it's similarity to an IEQ. The highest correlation was the Pearson correlation coefficient, at 0.4858. It was found using L_2 similarity between IEQs (from IEQ0) and EQs.
(iv) It was seen as a general trend that the AP achieved by an IEQ (generated by any method) decreased as the number of relevant documents for that particular query increased.
(v) Pruning helped in getting rid of unhelpful terms in the IEQ, and resulted in much shorter IEQs. It was seen that there was a strong positive correlation between the number of relevant documents and the number of terms in the pruned IEQ.

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

Acknowledgement

I would like to express my heartfelt gratitude to my supervisor, **Prof. Mandar Mitra**, for his constant guidance, encouragement, and insightful feedback throughout the course of this dissertation. His expertise and support have been invaluable.

I am also deeply thankful to my senior **Sourav Saha**, Senior Research Fellow at ISI Kolkata, for his generous help in clarifying numerous technical doubts during this work. The first algorithm for Ideal Expanded Query (IEQ0) generation was originally written by him in Java using Lucene. I have rewritten the entire algorithm in PyLucene as part of this dissertation, based on his implementation.

I also thank the faculty and my peers at the Indian Statistical Institute for providing an intellectually stimulating environment throughout my time here.

Aditya Dutta

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 (e.g. SQL) where both the queries and the answers are explicit and exact, IR systems aim to **rank** documents based on their **relevance** estimated for 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.

Thus, if the vocabulary of the corpus (set of distinct words contained in all documents in the corpus) is $(t_1, t_2, ..., t_n)$, then a document D in this corpus is represented by an n-dimensional vector

$$\boldsymbol{D} = (d_1, d_2, .., d_n),$$

where d_i, the weight of t_i in D, is a real number indicating how strongly t_i is related to the content of D.

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 compute the weight of a term t_i in D, **term weighting schemes** like **tf-idf** are typically used.

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

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

where:
- $f(t, d)$: Raw count of how many times term t appears in document d.
- **tf** reflects how often a term occurs in a document.

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

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

where:
- $\text{idf}(t)$: Inverse document frequency of term t
- N: Total number of documents in the collection
- n_t: Number of documents in which term t appears
- **idf** reflects how rare the term is across the corpus.

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

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

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:

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

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: parameters (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 the term t in document D, which will be important in our work.

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

Here, $\text{BM25}(D,t)$ 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 an information retrieval model. Now let

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

Then, Average Precision is defined as:

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

- Here, $|\text{Rel}|$ denotes the total number of relevant documents.

Mean Average Precision (MAP)

For a set of queries, **Mean Average Precision (MAP)** is 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

Vocabulary Mismatch Problem and Query Expansion

2.1 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 traditional/keyword-based 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 2.1. (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.

2.2 What is Query Expansion?

Query Expansion (QE) adresses this issue by augmenting the query with related words or phrases. The goal is to capture different ways the same idea might be expressed in the document collection, improving recall and precision.

Example 2.2. (Query Expansion): Original Query: `laptop overheating`

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

This reformulated query increases the likelihood of retrieving relevant documents.

2.3 Query Expansion Techniques

Several algorithms exist for expanding queries [1]. These techniques can be broadly categorized into:

2.3.1 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 2.3. (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 parameters 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.

2.3.2 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

2.3.3 Thesaurus-Based Expansion

This method uses resources like WordNet [2] 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

2.3.4 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

2.4 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. [3]

• **Log-Logistic Model**: This method uses a special formula (log-logistic distribution) to find terms that appear more often in some documents than expected. Such terms are considered important and used to expand the query. [4]

• **SPL (Smooth Power-Law)**: SPL is similar to LogLogistic but uses a different math formula (smoothed power-law) to find "bursty" terms i.e. words that tend to repeat a lot in relevant documents. These are added to improve the query. [4]

• **CEQE (Contextualized Embeddings for Query Expansion)**: A modern neural method using models like **BERT** [5] to understand the query's meaning. It finds context-aware terms for expansion, capturing semantic nuances and improving results. [6]

Chapter 3

Problem Statement and Hypothesis

3.1 Problem Statement

As mentioned previously, many query expansion algorithms exist, each with its own term selection and weighting schemes. It has been observed that different methods perform differently for different queries. 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.

3.2 Hypothesis

This work builds on the earlier studies by **Snehasish Mukherjee** [7] and **Soumajit Pramanik** [8].

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 query expansion method produces a query that is **close** to this IEQ (in some quantitative sense), then that EQ is expected to yield higher AP. Conversely, methods that produce queries farther from the IEQ will perform worse.

3.3 Overall Setup

3.3.1 Expanded Query (EQ) variants

(i) For each query in our dataset (See Section 6.1), we generated **80 expanded queries** ($4 \times 4 \times 5 \times 1 \times 1 = 80$) 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 EQ variant is identified by its `qid` and its `runid` which is a string of the form:

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

which describes the expansion method and parameters used to construct that particular variant.

(ii) For each variant,

- We generated a `term_weights` file containing the terms with their corresponding weights for each query in `<qid> <term> <weight>` format.
- We retrieved the top-1000 documents for each query and stored the results in a `run` file containing lines of the form: `<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.

Note 3.1. (Boosting Queries): Earlier we saw that the BM25 score of a document D with respect to a query Q is given by the formula (1). Here Q is considered as a zero-one vector which just signifies the presence or absence of a term in the query.

In general, the query Q can have non-negative real valued weights for each term. This is known as **Boosting**. In this case, the BM25 score of a document D with respect to a query Q is given by the formula

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

- Here w_t denotes the weight of term t in the query Q.

3.3.2 Ideal Query Generation

We then generate the IEQ for each query (using algorithms described later in Chapter 5). Ideal queries also have their corresponding `term_weights`, `run` and `ap` files.

3.3.3 Similarity Measuring and Correlation Computation

(i) Firstly, for each query, we measure the similarity between the IEQ and each of the 80 expanded versions of that query (using similarity formulae described later in Chapter 4).

(ii) For each query we make two lists of length 80,
- one containing the similarity of each EQ variant to the IEQ,
- the other containing the corresponding APs achieved by each EQ 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 correlations 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.

3.4 Software Used

- We used **PyLucene 10.0.0** (with Python 3.11) as the underlying search engine library for indexing and retrieval.
- We used **pytrec_eval** for evaluation of the retrieval results [9].
- We also used **trec_eval** for evaluating retrieval results. Link: https://github.com/usnistgov/trec_eval.
- Other Python libraries include **NumPy**, **SciPy**, **Matplotlib**.
- We used **GNU Parallel** for parallelization.

Chapter 4

Similarity Measures

We now describe the formulae that we have used in our work to measure the similarity between IEQ and an EQ. From now, we will assume that both IEQ and EQ are vectors in the Vector Space Model.

4.1 Cosine Similarity

Definition 4.1. (Cosine Similarity):

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

We refer to this as `l2_similarity` throughout.

Remark 4.2. (Dot product): Note that

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

where IEQ_t and EQ_t denote the weights of the term t in **IEQ** and **EQ** respectively.

Remark 4.3. (L2 norm): Note that

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

and for **IEQ**.

4.2 Cosine Similarity variant normalized by L_1 norm

Definition 4.4. (Cosine similarity variant normalized by L1 norm):

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

We refer to this as `l1_similarity` throughout.

Remark 4.5. (L1 norm): Note that,

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

and for **IEQ**.

4.3 Jaccard Similarity

Definition 4.6. (Jaccard Similarity):

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

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

We refer to this as `jaccard_similarity` throughout.

4.4 Modified nDCG similarity

Assume that terms in $\mathbf{IEQ}$ and $\mathbf{EQ}$ are ranked by weights and arranged in descending order.

Definition 4.7. (nDCG similarity):
Let,

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

and

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

Then,

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

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

We refer to this as `n2_similarity` throughout.

Chapter 5

Ideal Query Generation

5.1 Oracle Rocchio Vector tuning

We present the first algorithm for generating an **Ideal Expanded Query (IEQ)**. This is based on the **Dynamic Feedback Optimization (DFO)** scheme as described in [10]:

(i) Construct the *oracle* **Rocchio Vector** using the ground truth, which is obtained from the `qrel` file. See Definition 2.3. The document vectors are assumed to have BM25 weights.

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

(iii) Select a `tweak_magnitude` from a `list_of_magnitudes`.

(iv) 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 going over all the terms, 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 5.1. (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 parameters 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 5.2. (IEQ0 is computationally expensive): Since generating IEQ0 involves repeated retrieval steps, it is computationally expensive. Hence, we limit it to `num_expansion_terms`=200.

5.2 Logistic Regression based Ideal Query Generation

We now present a high level overview of the second algorithm for generating IEQs:

Algorithm 2: IEQ using Logistic Regression

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

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

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

3 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

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

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

*# Fit **LogisticRegression** model*

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

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

8 Select top `num_expansion_terms` terms with highest positive coefficients

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

Remark 5.3. (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.

Note 5.4. (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. The 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 $x = 0$).
  It was expected that $b \approx 0$ or $\sigma(b) \approx 0.5$, because this means that the model cannot distinguish an empty document to be relevant or non-relevant.
  Empirically, it was observed that $|b|$ was small for all queries.

The parameters 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**. We will use $\textbf{IEQ1}_{200}$ or $\textbf{IEQ1}_{1000}$ to denote IEQ1 with the number of expansion terms.

Chapter 6

Experimental Results

6.1 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 249 of the queries (in a `qrel` file).

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

Remark 6.1. (Dataset Link): https://ir-datasets.com/trec-robust04.html

6.2 MAP comparison of IEQs

Table 1: MAP of IEQs

IEQ	`num_expansion_terms`	MAP
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_{1000}.

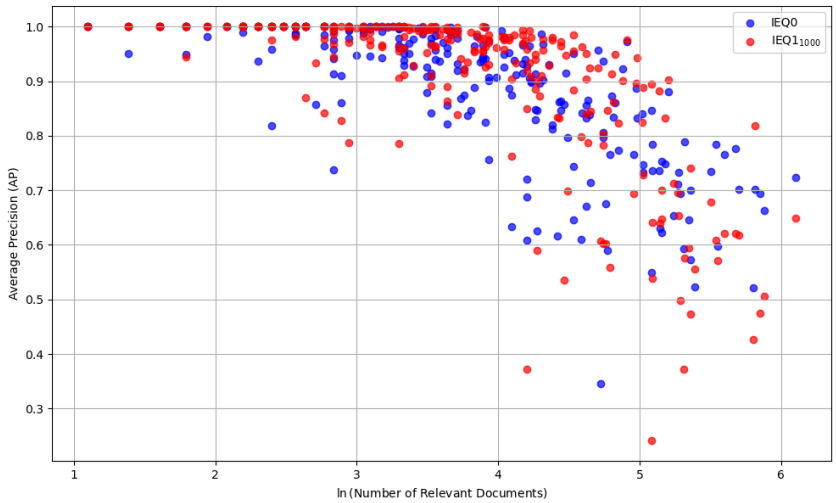

Figure 1: AP *vs.* ln(No. of relevant documents) for IEQs

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

6.3 Correlation Results

The average Pearson, Kendall and Spearman correlation coefficients across all queries are listed below for each similarity type; for both IEQ0 and IEQ1_{1000} ideal queries.

6.3.1 `l2_similarity`

Table 2: Average Correlations for IEQ0 and IEQ1_{1000} (using `l2_similarity`)

Correlation Name	IEQ0	IEQ1_{1000}
Pearson	**0.4858**	0.4697
Kendall	0.3762	0.3607
Spearman	0.4765	0.4521

6.3.2 `l1_similarity`

Table 3: Average Correlations for IEQ0 and IEQ1_{1000} (using `l1_similarity`)

Correlation Name	IEQ0	IEQ1_{1000}
Pearson	−0.2100	0.3472
Kendall	−0.1643	0.2594
Spearman	−0.2079	0.3351

6.3.3 `jaccard_similarity`

Table 4: Average Correlations for IEQ0 and IEQ1_{1000} (using `jaccard_similarity`)

Correlation Name	IEQ0	IEQ1_{1000}
Pearson	0.3285	0.2550
Kendall	0.2526	0.2119
Spearman	0.3334	0.2831

6.3.4 `n2_similarity`

Table 5: Average Correlations for IEQ0 and IEQ1_{1000} (using `n2_similarity`)

Correlation Name	IEQ0	IEQ1_{1000}
Pearson	0.2746	0.2897
Kendall	0.2280	0.2238
Spearman	0.2975	0.3000

- We observe moderate and weakly moderate correlations. For `l2_similarity`, we observe the highest correlation coefficients, among which the highest is the Pearson correlation coefficient for IEQ0, which is 0.4858.
- Except for `l1_similarity`, correlations for both IEQ0 and IEQ1_{1000} are almost identical for all other similarities.
- It is unclear why correlations using `l1_similarity` are so different in IEQ0 and IEQ1_{1000}.

Chapter 7

Using restricted ground truth for IEQ0 generation

7.1 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 EQs.

- We hypothesize that the algorithm is **overfitting** to the relevant documents. This means that the IEQ might contain terms that occur in only very few 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 retrieve the top-1000 documents for each query using BM25 and generate `a run` file. 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`, and each relevance information for that `qid`, we add the `docid` and its relevance to the `restricted_qrel` file, only if the `docid` occurs in the top-1000 retrieved results in the `run` file.

Note 7.1. (Rationale for using Restricted Ground Truth): Suppose a term t is very peculiar and rare and occurs in only a very few relevant documents. As per our hypothesis, due to overfitting, there is a good chance that term t will be assigned a high weight in the IEQ as it helps in increasing the AP of the IEQ. But since t is very rare, and QE algorithms first retrieve documents using the initial queries and then build the EQs using the vocabulary of retrieved documents, t has very low chances of appearing in any EQ with significant weight. We want to use the same *retrieved set* for IEQ construction, thus, reducing the chances of such rare terms appearing in the IEQ.

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**.

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

7.2 Correlation results

7.2.1 `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

7.2.2 `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

7.2.3 `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

7.2.4 `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. This is counter-intuitive because we expected that the restricted ground truth will produce IEQs with better intersection with the EQs.

Chapter 8

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.

8.1 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 or remains same. If so, we eliminate the term.

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.

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

8.2 Pruning IEQ1_{200}

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 retain it.

MAP achieved: Compared to IEQ1_{200}, which had a MAP of 0.8197, pruning helped the MAP significantly in this case, increasing it to **0.9055**. Refer Table 1.

Henceforth, this method will be referred to as $\textbf{IEQ1Pruned}_{200}$.

8.3 Pruning IEQ1_{1000}

Although very computationally expensive, we similarly prune IEQ1 queries containing 1000 terms.

MAP achieved: This method achieves an incredibly high MAP of **0.9760**. This is still very high compared to the MAP achieved by IEQ1_{1000} which was 0.9026. Refer Table 1.

Henceforth, this method will be referred to as $\textbf{IEQ1Pruned}_{1000}$.

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

IEQ0 and IEQ1_{200} queries had a constant size of 200 terms.

Pruning had a significant effect 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. On the other hand, for some queries, only a small number of terms were pruned.

Size of Pruned IEQs found to be highly correlated to the number of relevant documents: Expectedly, the size of IEQ0Pruned and IEQ1Pruned$_{200}$ queries were highly positively correlated to the number of relevant documents as we see in this plot:

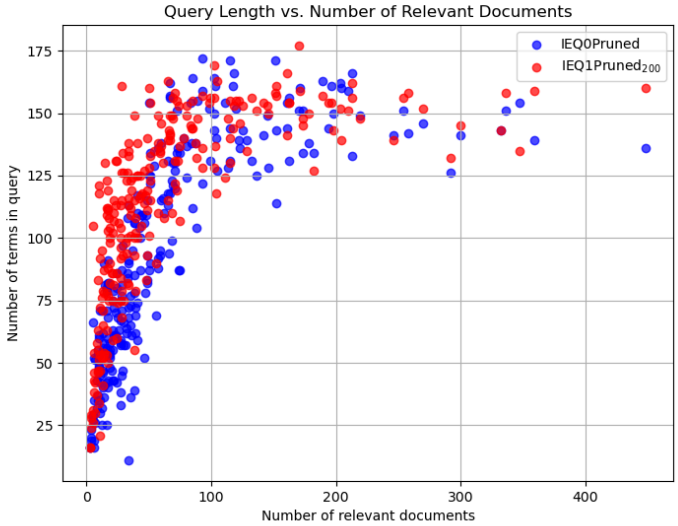

Figure 2: No. of terms in query *vs.* No. of relevant documents

This is expected, as an increase in the number of relevant documents typically introduces a greater diversity of terms. To retrieve all such documents effectively, the IEQ must incorporate more of these terms, leading to longer queries. In other words, the broader the set of relevant documents, the more terms are needed in the IEQ to maintain retrieval effectiveness.

A similar trend was seen for IEQ1Pruned$_{1000}$, although the number of terms in IEQ1Pruned$_{1000}$ queries was generally much higher.

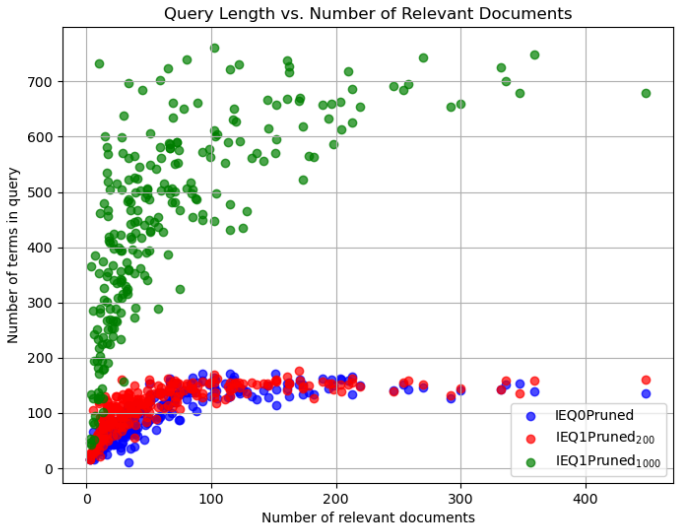

Figure 3: No. of terms in query *vs.* No. of relevant documents

The **Spearman Correlation** (which measures the monotonic relationship between two ranked variables) between the number of relevant documents and the number of terms in IEQ for IEQ0Pruned, IEQ1Pruned$_{200}$ and IEQ1Pruned$_{1000}$ are: 0.8907, 0.8523 and 0.7951 respectively.

8.5 Correlation Results

8.5.1 `l2_similarity`

Table 10: Average Correlations for IEQ0, IEQ1, IEQ0Pruned$_{200}$ and IEQ1Pruned$_{1000}$ (using `l2_similarity`)

Correlation Name	IEQ0	IEQ1	IEQ0Pruned	IEQ1Pruned$_{200}$	IEQ1Pruned$_{1000}$
Pearson	0.4858	0.4697	0.3451	0.4082	0.4163
Kendall	0.3762	0.3607	0.2743	0.3121	0.3283
Spearman	0.4765	0.4521	0.3480	0.3931	0.4114

8.5.2 `l1_similarity`

Table 11: Average Correlations for IEQ0, IEQ1, IEQ0Pruned$_{200}$ and IEQ1Pruned$_{1000}$ (using `l1_similarity`)

Correlation Name	IEQ0	IEQ1	IEQ0Pruned	IEQ1Pruned$_{200}$	IEQ1Pruned$_{1000}$
Pearson	−0.2100	0.3472	−0.1942	0.2847	0.3142
Kendall	−0.1643	0.2594	−0.1472	0.2070	0.2311
Spearman	−0.2079	0.3351	−0.1898	0.2722	0.3033

8.5.3 `jaccard_similarity`

Table 12: Average Correlations for IEQ0, IEQ1, IEQ0Pruned$_{200}$ and IEQ1Pruned$_{1000}$ (using `jaccard_similarity`)

Correlation Name	IEQ0	IEQ1	IEQ0Pruned	IEQ1Pruned$_{200}$	IEQ1Pruned$_{1000}$
Pearson	0.3285	0.2550	0.3239	0.2591	0.2732
Kendall	0.2526	0.2119	0.2424	0.2015	0.2180
Spearman	0.3334	0.2831	0.3226	0.2667	0.2960

8.5.4 `n2_similarity`

Table 13: Average Correlations for IEQ0, IEQ1, IEQ0Pruned$_{200}$ and IEQ1Pruned$_{1000}$ (using `n2_similarity`)

Correlation Name	IEQ0	IEQ1	IEQ0Pruned	IEQ1Pruned$_{200}$	IEQ1Pruned$_{1000}$
Pearson	0.2746	0.2897	0.2941	0.2606	0.2660
Kendall	0.2280	0.2238	0.2346	0.1989	0.2045
Spearman	0.2975	0.3000	0.3078	0.2622	0.2733

Conclusion: As we see, this also did not result in an improve in terms of correlations. A possible reason could be that due to pruning, IEQs became too small and minimal due to which they had very small intersections with the EQs. This might have lead to unreliable correlations.

8.6 MAP comparison of IEQs (final)

We finally present the MAPs achieved by all IEQ methods discussed so far:

Table 14: MAP of IEQs (Final)

IEQ	`num_expansion_terms`	MAP
Untweaked Oracle Rocchio	200	0.5121
Untweaked Oracle Rocchio	1000	0.5465
IEQ0	200	0.8919
$IEQ1_{1000}$	1000	0.9026
$IEQ1_{200}$	200	0.8197
IEQ0Restricted	200	0.8256
IEQ0Pruned	11-172*	0.9060
$IEQ1Pruned_{200}$	16-177*	0.9055
$IEQ1Pruned_{1000}$	42-761*	**0.9760**

* Represented in the format `minimum_number_of_terms-maximum_number_of_terms`

Bibliography

[1] C. Carpineto and G. Romano, "A Survey of Automatic Query Expansion in Information Retrieval," *ACM Computing Surveys*, vol. 44, no. 1, pp. 1–50, Jan. 2012, doi: 10.1145/2071389.2071390.

[2] G. A. Miller, "WordNet: A Lexical Database for English," in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. [Online]. Available: https://aclanthology.org/H94-1111/

[3] Victor Lavrenko and W. Bruce Croft, "Relevance-based language models," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 120–127. doi: 10.1145/383952.383972.

[4] Stéphane Clinchant and Eric Gaussier, "Information-Based Models for Ad Hoc IR," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2010, pp. 234–241. doi: 10.1145/1835449.1835490.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *CoRR*, vol. abs/1810.04805, 2018.

[6] Shahrzad Naseri, Jeffrey Dalton, Andrew Yates, and James Allan, "CEQE: Contextualized Embeddings for Query Expansion." pp. 467–482, 2021. doi: 10.1007/978-3-030-72113-8_31.

[7] Snehasish Mukherjee, "Relative Query Performance Prediction using Ideal Expanded Query," 2012.

[8] Soumajit Pramanik, "Understanding the Performance of Expanded Queries," 2013.

[9] Van Gysel, Christophe and de Rijke, and Maarten, *Pytrec_eval: An Extremely Fast Python Interface to trec_eval*. (2018). doi: 10.1145/3209978.3210065.

[10] Chris Buckley and Gerard Salton, "Optimization of relevance feedback weights," in *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, 1995, pp. 351–357. doi: 10.1145/215206.215383.

Declaration

I hereby declare that the dissertation titled **"Explaining Query Expansion Algorithms"** submitted by me to the **Indian Statistical Institute, Kolkata**, in partial fulfillment of the requirements for the degree of **Master of Technology in Computer Science**, is my own work carried out under the supervision of **Prof. Mandar Mitra**.

This dissertation has not been submitted to any other university or institution for the award of any degree or diploma. All sources of information and data used have been appropriately acknowledged.

I certify that the work presented is free from plagiarism and is a result of my own independent research and analysis.

Kolkata, 2025-06-14

Aditya Dutta