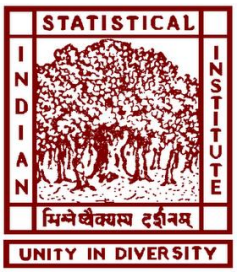
Enhancing Axiomatic Explanations for Pairwise Ranking



A dissertation submitted in partial fulfilment of

therequirements for the M. Tech. (Computer

Science) degree of the Indian Statistical Institute.

By

Swastik Mohanty

under the supervision of

Dr. Debapriyo Majumdar

Indian Statistical Institute

203, B.T. Road. Kolkata: 700108

CERTIFICATE

I certify that I have read the thesis titled “Enhancing Axiomatic Explanations for Pairwise Ranking”, prepared under my guidance by Swastik Mohanty, and in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Master of Technology in Computer Science of the Indian Statistical Institute.

_____

Dr. Debapriyo Majumdar
Assistant Professor
CVPRU

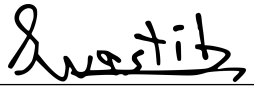
KOLKATA
JULY, 2024

Declaration of Authorship

I, Swastik Mohanty, declare that this thesis titled "Enhancing Axiomatic Explanations for Pairwise Ranking" and the work presented within it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Swastik Mohanty

Signed: 

Date: 17/06/24

Acknowledgements

To Dr. Debapriyo Majumdar whose expert guidance, infinite patience and constant encouragement has endowed me with a strong foundation in Information Retrieval and made this dissertation possible.

To Sourav Saha who was always available to answer all my queries and helped mein using the various tools required for my project. He supplied me with a variety of research papers and libraries to enhance my knowledge on explainable IR and helped me build solid foundation before diving deep into pairwise explanations. Thanks for always being available to answer all my doubts and always pointing me in the right direction when I was stuck.

To Dr. Mandar Mitra for always being there to provide me with the domain knowledge and key insights regarding my project.

Swastik Mohanty

June, 2024.

Table of Contents

List of Tables and Figures

Chapter 1 - Introduction to Explainability

1.1 Explainable AI

Explainable AI (XAI) is a critical framework designed to enhance human comprehension and confidence in the outcomes produced by artificial intelligence (AI) and machine learning (ML) models. It addresses the inherent "black box" nature of many AI systems by offering clear, understandable, and transparent explanations for their decisions and actions. This transparency is particularly crucial in high-stakes domains such as healthcare, finance, law enforcement, and autonomous systems, where the ability to interpret AI reasoning is essential for ensuring accountability, fairness, and trust. Key aspects of XAI include –

1. **Transparency**: This involves making the internal workings of AI systems visible and understandable. It includes documenting how the AI model was developed, the data it was trained on, the algorithms it uses, and the logic behind its predictions or decisions.

2. **Interpretability**: The extent to which a human can understand the cause of a decision made by an AI system. Models that are interpretable can provide insights into how they arrive at specific outputs, making it easier to identify potential biases or errors.

3. **Justifiability:** This aspect ensures that the decisions made by AI can be justified with respect to domain knowledge and societal norms. The explanations provided should make sense to domain experts and be aligned with human reasoning.

4. **Reliability and Robustness**: An explainable AI system should not only provide understandable explanations but also be reliable and robust in its predictions. This involves ensuring that the system behaves consistently and can handle a variety of inputs without producing unexpected or incorrect results.

5. **User Control and Feedback**: Allowing users to interact with the AI system, provide feedback, and have control over its outputs. This helps in refining the system and aligning it more closely with user expectations and requirements.

1.2 Explainability for IR

Explainability in Information Retrieval (IR) is a crucial area that focuses on making the decision-making processes of IR systems understandable and transparent to users. This is particularly important given the complex nature of modern IR systems, which often involve sophisticated neural models and large-scale data processing.

1.3 Axiomatic Approaches for Explainable IR

In the realm of information retrieval (IR), retrieval axioms play a pivotal role as formal principles that define the desirable behavior of retrieval models. These axioms are designed to ensure that IR systems rank documents in a manner that aligns with users' expectations and needs. By establishing clear, theoretical guidelines for how documents should be ranked in response to queries, retrieval axioms help to improve the effectiveness and reliability of IR systems.

Retrieval axioms provide a structured way to evaluate and enhance the performance of retrieval models. They serve as a theoretical foundation that guides the development and refinement of these models, ensuring that they produce rankings that are both intuitive and effective. For instance, axioms can specify that if a document is more relevant to a query than another, the more relevant document should be ranked higher. Such principles help in creating models that better understand and fulfill user intents, leading to more satisfactory search experiences.

To evaluate and explain the adherence of IR models to these axioms, specialized tools such as ir_axioms [1] and ir_explain [2] have been developed. These tools facilitate the implementation and assessment of retrieval axioms by comparing pairs of documents and generating scores that indicate preference according to specific axioms.
In the subsequent sections, we will examine the application of these tools.

Chapter 2 - Related Work

2.1 Information Retrieval Toolkits

The Information Retrieval (IR) community has a well-established tradition of providing open-source libraries and resources that make it convenient for researchers to experiment with, evaluate, and understand different retrieval models. Among these, Pyserini and PyTerrier stand out as prominent modern IR toolkits that support both sparse and dense retrieval models, facilitating a wide range of IR tasks and experiments.

2.1.1 Pyserini

Pyserini is a Python interface to the Anserini information retrieval toolkit, which is built on top of the Apache Lucene search library. Designed to bridge the gap between robust, production-grade search capabilities and the needs of the research community, Pyserini provides a powerful yet accessible platform for IR experimentation. One of its core strengths lies in its seamless integration with Lucene, allowing users to leverage Lucene's high-performance, full-featured text search engine capabilities. This integration ensures that Pyserini can handle large-scale text retrieval tasks efficiently.

Moreover, Pyserini is designed with reproducible research in mind. It offers pre-built indices for major datasets, ensuring that researchers can reproduce results and compare their work against established benchmarks. The Python API provided by Pyserini makes it easy for researchers to interact with the toolkit, facilitating the use of Python's rich ecosystem for data analysis and machine learning. This accessibility, combined with its state-of-the-art retrieval methods, makes Pyserini a valuable tool for advancing IR research.

2.1.2 PyTerrier

PyTerrier, or Python Terrier, is another influential toolkit in the IR domain. Built on top of the Terrier search engine platform, PyTerrier provides an efficient and flexible environment for conducting IR research.

It integrates seamlessly with Terrier, a well-known platform celebrated for its extensible and powerful IR capabilities. This integration allows PyTerrier to benefit from the robust search and indexing functionalities that Terrier offers.

One of PyTerrier's key features is its emphasis on rapid experimentation with retrieval models. Researchers can prototype and evaluate new IR methods quickly, thanks to the modular and Pythonic design of PyTerrier. This modularity not only facilitates easy customization and extension of retrieval pipelines but also enhances the toolkit's adaptability to various research needs. Additionally, PyTerrier supports the standard retrieve-and-rerank pipeline on TREC collections, ensuring that researchers can work with widely recognized benchmarks and datasets.

2.2 Ir_axiom

ir_axioms is an open-source Python framework designed to integrate axiomatic principles into information retrieval (IR) systems [1]. The primary goal of ir_axioms is to incorporate retrieval axioms into modern retrieval frameworks to enhance their performance and interpretability. Retrieval axioms are fundamental principles or rules that define the behavior and properties of effective retrieval models. By integrating these axioms, ir_axioms aims to enhances the quality of initial search result rankings thereby improving the overall retrieval performance. It ensures that the outcomes are more effective and aligned with fundamental retrieval principles. Additionally, it provides insights and explanations for the rankings produced by different retrieval models.

ir_axioms offers several key features that enhance its utility in information retrieval. It includes reference implementations for several predefined retrieval axioms, each representing a specific principle or rule pertinent to effective information retrieval. The framework provides tools for preference aggregation, allowing users to incorporate and apply user preferences to refine retrieval models.

One fundamental example is the Term Frequency Constraint (TFC1) axiom, which asserts that documents containing more occurrences of query terms should be ranked higher. This principle posits that a document that mentions the query term more frequently is more relevant to the user's search intent. By incorporating the TFC1 axiom, an IR system ensures that documents which are likely to be more informative and relevant to the user's query are prioritized, leading to a more effective and satisfying search experience.

Additionally, ir_axioms includes methods for reranking search results based on axiomatic principles, ensuring that the output aligns with the defined axioms. To evaluate the performance of models enhanced by these axioms, ir_axioms offers robust evaluation tools. Moreover, it facilitates the easy definition of new axioms; users can either create new axioms by implementing an abstract data type or combine existing axioms using Python operators or regression techniques, offering flexibility and extensibility in experimentation and application.

2.2.1 Implementation details

ir_axioms seamlessly integrates with popular retrieval frameworks such as PyTerrier and Pyserini, facilitating the application and experimentation of axiomatic principles within various retrieval models. This integration ensures that users can easily implement and test retrieval axioms without the need for extensive adjustments to their existing workflows.

Furthermore, ir_axioms provides comprehensive access to ir_datasets, which includes standard retrieval models, corpora, topics, and relevance judgments. This access is particularly valuable for facilitating experimentation with well-known datasets, such as those used in the Text Retrieval Conference (TREC). By integrating with ir_datasets, ir_axioms allows users to work with established benchmarks and datasets, ensuring that their experiments are grounded in widely-recognized and reliable data sources. This integration not only streamlines the process of setting up experiments but also enhances the credibility and comparability of the results, making it an invaluable tool for advancing research in information retrieval.

2.2.2 Re-ranking using ir_axioms

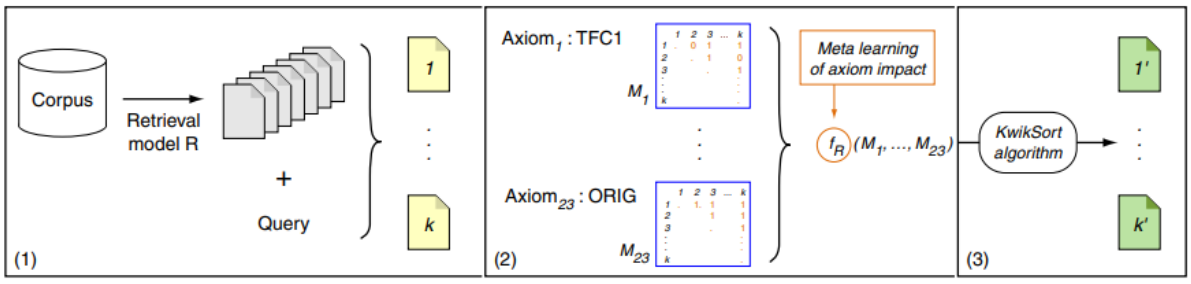


Figure 1: Re-reanking using ir_axioms taken from Hagen 2016 [3]

This figure illustrates the process of axiomatic re-ranking in three main steps. Let's break down each part in detail:

Step 1: Initial Retrieval:

1. **Corpus and Retrieval Model:** The process begins with a corpus of documents. A retrieval model *R* (e.g., BM25, a common information retrieval model) is used to retrieve documents from the corpus based on a given query.

2. **Initial Ranking:** The retrieval model produces an initial ranking of the top-k documents relevant to the query. These documents are labeled 1 through k.

Step 2: Axiomatic Evaluation:

1. **Application of Axioms:** Multiple axioms are applied to evaluate the initial rankings. Examples include TFC1 and ORIG. Each axiom M_i generates a matrix where the rows represent the top-k documents and the entries indicate the preferences or scores assigned by the axiom to each document.

2. **Meta Learning:** The outputs of these axiomatic evaluations are fed into a meta-learning component. Meta-learning helps to understand and estimate the impact of each axiom on the ranking process, producing a function $f_R(M_1, ..., M_{23})$ that aggregates the preferences or impacts of all the axioms.

Step 3: KwikSort Algorithm:

1. **Re-ranking with KwikSort**: The aggregated preferences from the meta-learning process are input into the KwikSort algorithm. KwikSort [4] then re-ranks the top-k documents based on the combined axiomatic preferences, resulting in a new, potentially more effective ranking of documents labeled 1' through k'.

This process highlights how axiomatic principles can be systematically integrated into retrieval models to refine and enhance the quality of search results.

2.2 Ir_explain

ir_explain is an open-source Python library that offers a range of well-established techniques for Explainable Information Retrieval (ExIR) within a unified and extensible framework. It is designed to simplify the reproduction of state-of-the-art ExIR baselines on standard test collections and to enable the exploration of new methods for explaining IR models. To encourage its use, ir_explain is seamlessly integrated with popular toolkits like Pyserini and ir_datasets.

2.2.1 Understanding the library

The most common Information Retrieval (IR) task involves retrieving the top-k ranked documents for a query Q from a large collection C and presenting them to the end-user. Modern Neural Information Retrieval (neuIR) typically uses a two-stage pipeline for this process.

1. **First Stage**: An initial ranker, such as BM25 or language models with Jelinek-Mercer (LMJM) or Dirichlet (LMDir) smoothing, retrieves a larger set of top k' documents (where k' is much larger than k but still small compared to the entire collection C). For example, if k is 100, k' might be 1000.

2. **Second Stage**: Dense retrieval techniques are then used to re-rank this initial set of k' documents. This re-ranking is manageable for complex neural models.

The first-stage ranker is denoted as M_1, and the second-stage ranker as M_2. Explainable IR (ExIR) aims to clarify different aspects of these two ranking stages. Explanation methods are generally categorized into three types:

- **Pointwise Explanations**: Focus on individual document scores.

- **Pairwise Explanations**: Compare pairs of documents to explain ranking differences.

- **Listwise Explanations**: Consider the entire list of ranked documents to provide explanations.

2.2.2 Pairwise Explanation

A pairwise explanation seeks to clarify why one document D_i is preferred over another document D_j for a given query Q. These explanations are typically based on retrieval axioms, which are formalizations of intuitive retrieval heuristics outlining the constraints a good ranking method should satisfy. Various sets of axioms have been developed in the literature, and some relaxations have been proposed to make these axioms practical. Hagen et al. [3] demonstrated that the axiomatic framework can also serve as the foundation for the second-stage ranker M_2. While this use of the axiomatic framework is not an explanation per se, it helps illustrate how the intuitive, pairwise preferences indicated by axioms can be integrated into a final ranking.

This is our contribution to the library and will be discussed in detail in the upcoming chapters.

2.2.3 Installation

The ir_explain library can be installed via GitHub repository:

- git clone https://github.com/souravsaha/ir_explain

beir==1.0.1	gensim==4.3.1	nltk==3.8.1	scipy==1.10.1
captum==0.6.0	h5py==3.9.0	numpy==1.24.4	torch==2.0.1
cvxpy==1.3.2	ir_datasets==0.5.5	pyserini==0.21.0	torchtext==0.15.2
Datasets==2.13.1	ir_measures==0.3.3	pytorch_lightning==2.0.5	tqdm==4.65.0
genosolver==0.1.0.6	ipython==7.29.0	scikit_learn==1.3.0	transformers==4.30.2

Table 1: Requirements to run ir_explain

Chapter 3 - Our contribution to IR_Explain

Inspired from IR_axioms, pairwise component of ir_explain library implements such retrieval axioms for experimentation with standard retrieval toolkits. These axioms are adapted for practical use, such as reformulating them to work with arbitrary queries and expressing pairwise preferences. Parameters are included to adjust the conditions of the axioms, and term similarity axioms are provided in variants using WordNet synsets or fastText embeddings.

3.1 Implementing the Axioms

While ir_axioms is closely integrated with the Pyterrier retrieval pipeline and lacks the capability to compare arbitrary document pairs D_i and D_j , this feature is included in ir_explain.

Similar to ir_axioms, ir_explain allows users to use binary and unary operators to combine and aggregate different axioms, and also provides an easy way to define new axioms. The primary function of an axiom object A is to determine, given a query Q and two documents D_i and D_j, whether D_i is preferred over D_j for the query Q. This is done by returning a preference score $pref_A(Q, D_i, D_j) \in \{-1, 0, 1\}$ defined as follows:

$$pref_A(Q, D_i, D_j) = \begin{cases} 1, & \text{if } D_i \text{ is preferred over } D_j \\ -1, & \text{if } D_i \text{ is preferred over } D_j \\ 0, & \text{if no preferrence given by A} \end{cases}$$

Here is how you can use the pairwise class within the ir_explain framework to explain the ranking of two documents (doc1 and doc2) for a given query (query) using a set of predefined axiom classes:

```
pairwise = Pairwise(query, doc1, doc2, index_path)
axiom_classes = [TFC1(), STMC1(), ...]
pairwise.explain(axiom_classes)
```

Here, a Pairwise object is created, which represents the pairwise comparison between two documents (doc1 and doc2) for a given query (query). The index_path variable represents the path to the index or dataset used for retrieval.

A list axiom_classes is defined by user, where the user can specify which axioms it wants to use for the given pair of documents. The explain method of the Pairwise class computes and provides an explanation for the ranking of doc1 and doc2 for the given query based on the specified axiom classes.

3.1.1 List of Axioms

In order to fulfil our requirements, we had to code the axioms from ir_axioms from scratch following the logic proposed by the original authors of the axioms (along with the recent modifications made to them) [3]. Here is a detailed summary of each axiom in the library -

TFC1:

The Term Frequency Constraint 1 (TFC1) is a fundamental axiom in information retrieval (IR) that guides how documents should be ranked based on the frequency of a query term within them [5]. Specifically, TFC1 asserts that when a query consists of a single term t, documents containing higher occurrences of t should receive higher scores. In essence –

> TFC1 (as given by Hagen et 2016 [3]): Let $q = \{ t \}$ be the query with one term t. Assume $|d_1| = |d_2|$. If $tf(t, d_1) > tf(t, d_2)$ then $score(d_1, q) > score(d_2, q)$.
>
> We transform TFC1 to our triple notation by setting (10% relaxation)
>
> Precondition:= $length\ (d_1) \approx length\ (d_2)$
>
> Filter := $length\ (d_1) \approx length\ (d_2)$
>
> Conclusion:= $d_1 > d_2$

When dealing with queries that consist of more than one term, a generalized version of TFC1 expands the principle to incorporate the sum of individual term frequencies. This approach ensures that documents containing higher combined frequencies of all query terms are ranked more favorably.

TFC3:

This axiom is designed to provide rules for scoring documents when the query contains two terms. It considers two main factors: term frequency and term importance (inverse document frequency, IDF).

Term Frequency (TF): As seen before, it refers to how often each term in the query appears in the document. A document scores better if it includes the query terms more frequently, as this indicates a stronger relevance to the query.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}}$$

$f_{t,d}$ is the raw count of a term in a document, i.e., the number of times that term t occurs in document d. Note the denominator is simply the total number of terms in document d (counting each occurrence of the same term separately).

Term Importance (IDF): IDF measures how unique or rare a term is across the entire document collection. A higher IDF value indicates that a term is more important or distinctive. A document scores better if it contains terms with higher IDF values, as these terms contribute more to the uniqueness of the document.

$$idf(t,D) = log \frac{N}{|\{d: d \epsilon D \ and \ t \epsilon d\}|}$$

N is the total number of documents in the corpus. $|\{d \epsilon D: t \epsilon d\}|$ is the number of docs where the term t appears (i.e. $tf(t,d) \neq 0$). However, there is a potential issue with this axiom i.e. if a query term is not present in the corpus, it can lead to a division by zero error. To overcome this, we can adjust the formula to avoid such situations by changing the numerator and denominator to $1 + N$ and $1 + |\{d: d \epsilon D \ and \ t \epsilon d\}|$ respectively.

To extend these scoring rules to longer queries (queries with more than two terms), the axiom suggests applying the rules to every possible pair of terms within the query. [5]

PROX1:

It ranks documents by how closely query terms appear to each other on average. We define $\pi(q, d)$ which measures the average distance between pairs of query terms in a document.

$$\pi(q, d) = \frac{1}{|P|} \sum_{(i,j)\ \epsilon\ P} \delta(d, i, j)$$

Where $P = \{(i, j) | i, j\ \epsilon\ q, i \neq j\}$ is the set of all possible query term pairs and $\delta(d, i, j)$ calculates the average number of words between the query terms t_i and t_j in the document d based on all positions of t_i and t_j . Finally whichever doc has a smaller $\pi(q, d)$ is preferred. [6]

PROX2:

It follows the logic that a document where query terms appear earlier is likely more relevant, as it suggests that the document addresses the query terms sooner, making it easier for the searcher to find relevant information quickly. It ranks documents based on how early the query terms first appear. If the first occurrences of query terms in document d_1 happen earlier than in document d_2, then d_1 is considered better. [6]

PROX3:

It ranks documents based on how early the entire query phrase appears. For each document, find where the entire query q first appears as a single phrase. Identify the position where this phrase first appears in the document. This position is called $\tau(d, q)$. If the query does not appear as a single phrase, assign $\tau(d, q) = \infty$. Prefer the document with lower value of $\tau(d, q)$. [6]

PROX4:

It ranks documents based on how closely the query terms appear together with the fewest non-query words in between and how frequently this close grouping occurs. We define:

$\omega(d, q)$: Represented as a pair (a, b) where:

- a is the number of non-query words in the closest grouping of all query terms.
- b is how often this gap value occurs.

Compare the pairs $\omega(d_1, q)$ and $\omega(d_2, q)$ for two documents d_1 and d_2. The one with lower $\omega(d, q)$ is preferred. [6]

PROX5:

It ranks documents based on the average size of the smallest text spans that contain all query terms around each occurrence of each query term. We define $\bar{s}(d, q)$:

- For each query term t in the document, determine the smallest text span that includes all query terms.
- Measure the size (number of words) of this text span.
- Compute the average size of these smallest text spans for all occurrences of all query terms in the document.

Compare the pairs $\bar{s}(d_1, q)$ and $\bar{s}(d_2, q)$ for two documents d_1 and d_2. The one with lower $\bar{s}(d, q)$ is preferred. [6]

LNC1:

In this axiom, when we're comparing two documents that exhibit the same term frequency for all the query terms (with the consideration of the 10% softening), a preference is given to the shorter document. This means that if two documents have an equal number of occurrences of the terms in the query, the document with fewer words overall is considered more favorable. [5]

LNC2:

This axiom originally checks if one document is an m-times copy of another. In this scenario, the original (shorter) document is favored. However, this condition is unlikely in real-world documents. So, the axiom is modified:

1. It calculates the Jaccard coefficient of the documents' vocabularies (terms they share).
2. If this coefficient is at least 80%, it derives the value of m based on the ratio of the minimum document length to the maximum.

This modification makes the axiom more practical and adaptable to real-world scenarios. [5]

TF_LNC:

Axiom TF-LNC combines term frequency and document length for single-term queries. When comparing two documents without the term and having the same length (with a 10% softening allowance), the one with the higher term frequency is preferred. This principle is generalized for multi-term queries by summing the term frequencies, similar to the LNC1 generalization. In essence, TF-LNC prioritizes documents with higher term frequencies, considering document length as a tiebreaker when frequencies are equal. [5]

LB1:

This axiom examines documents that have the same retrieval score score(q,d). This retrieval score represents the relevance of a document d to a given query q. There's a 10%-softening allowance, meaning small variations in the retrieval score are permitted. If a term in the query appears in document d_2 but not in document d_1, it suggests that document d_2 might be more relevant to the query than d_1. [7]

STMC1:

Matching words that are similar in meaning rather than exactly matching the query terms can be beneficial in situations where there's a mismatch in vocabulary or to improve small result sets. WordNet is utilized to find these semantically similar terms. Axioms like STMC1 and STMC2 are analyzed in this context to leverage this semantic matching approach.

For each word w in a document d and each query term t in the query q, the semantic similarity between w and t is calculated using WordNet. These individual similarity values are then averaged to derive the overall semantic similarity σ(d,q) between document d and query q. From two documents d_1 and d_2, the one with the larger average semantic similarity value σ(d,q) is preferred. [8] [9]

STMC2:

This axiom is generalized to compare pairs of documents d_1 and d_2 based on semantic similarity between non-query terms and query terms. From either document, the non-query term t that is most similar to any query term t' identified using WordNet. It ranks d_1 higher than d_2 if the ratio of their lengths is about 20% of the ratio of t's frequency in d_2 to t' frequency in d_1. [9] [8]

$$\frac{|d2|}{|d1|} = approx\ 20\% \ \frac{tf(t,d2)}{tf(t0,d1)}$$

REG:

In this axiom first, identify the query term t that is most similar to the other query terms. This similarity can be determined using WordNet. If both documents d_1 and d_2 contain all the other query terms (except for t), compare the term frequency (tf) values of t in d_1 and d_2. The document with a higher tf-value for t is preferred. [10] [11]

AND:

In this axiom, given a query, if there's a pair of documents d_1 and d_2 where only d_1 contains all the query terms, then d_1 is favored. [10] [11]

DIV:

In this axiom we again use the Jaccard coefficient, denoted as $J(d,q)$, which measures the similarity between the set of terms in a document d and the set of query terms q. If the Jaccard coefficient $J(d,q)$ for documentd_1 is lower than $J(d,q)$ for document d_2, then d_1 is preferred. [12]

(Note – We haven't included the Argumentativeness axioms as of now, since they do not apply to the ad-hoc retrieval settings used in our experiments)

3.2.2 Combining and Adding new axioms

You can add a new axiom in the following way –

Class NewAxiom (Axiom):
Name = "NEW"
 # Initialization/parameters

def explain(self, C_{index} , query, doc1, doc2) :
 # Rules for preferring doc1 over doc2 for the given query
 # C_{index} for accessing term frequencies in an index

You can combine the axioms using the following binary and unary operators –

+	Add	Add axiom preferences or constants
-	Subtract	Subtract axiom preferences or constants
*****	Multiply	Multiply axiom preferences by a weight
/	Divide	Divide axiom preferences by a weight
&	Conjuction	Return preferance if all axioms agree
%	Majority	Take majority vote of axioms

Table 2: Binary operators in ir_explain

-	Negate	Negate axiom preference
+	Normalize	Normalise axiom preference to (-1,0,1)

Table 3: Unary operators in ir_explain

3.3 Explain_More

For more complex axioms such as PROX1, PROX2, and PROX3, the simplistic use of preference scores alone falls short in adequately explaining why one document is preferred over another. To address this limitation, we have the class explain_more which contains the function *`explain_details()`*, offering a deeper understanding of the preference computation process.

This tool serves a crucial role in enhancing our comprehension of the preferences derived from axioms, offering a deeper dive into their underlying mechanics. It enables us to scrutinize intricate details that govern document prioritization.

For instance, if TFC1 indicates that document 1 is preferred over document 2 for a specific query, `explain_details()` allows us to delve deeper. By examining the exact differences in term frequency (TF) values, we can ascertain the extent to which this axiom influences the ranking decision. If the TF difference is minor, it suggests that TFC1 may not accurately reflect the documents' relevance for that query, prompting a reassessment of its significance in such cases.

Similarly, consider LNC1, which asserts that two documents have an identical frequency of query terms. Utilizing `explain_details()`, we gain insights into the documents' respective lengths. This information aids in identifying whether discrepancies in document ranking could be attributed to variations in document sizes. Such detailed analysis provided by the tool enhances our understanding of how different factors, beyond simple term frequencies, contribute to document relevance and ranking outcomes. This capability is essential for refining and optimizing information retrieval strategies with a more nuanced approach.

The tool can be implemented in the following manner –

```
from explain_more import explain_details
pairwise.explain_details(query, doc1, doc2, axiomName)
```

We'll see many use cases of the Pairwise class in the next section for better understanding about how to use both the pairwise class and explain_more class

Chapter 4 - Experiments and Use Cases

4.1 The Dataset

We are using a collection of datasets known as the MS MARCO (Microsoft MAchine Reading COmprehension) dataset for the TREC Deep Learning (TREC-DL) track in 2019, which is a widely used benchmark dataset in the field of natural language processing (NLP) and information retrieval (IR). It is designed specifically for the task of machine reading comprehension and question answering, with a focus on real-world relevance and user intent understanding.

Here are the key details and components of the MS MARCO datasets:

1. **Size and Scope**: The dataset consists of a large collection of real user queries and corresponding passages extracted from web documents. It contains over 1 million (1,010,916) unique queries along with their associated passages. The passages are relatively short segments of text extracted from web pages, typically ranging from a few sentences to a short paragraph in length.

2. **Query Types**: The queries in the dataset cover a wide range of topics and query intents, reflecting the diversity of real user search queries encountered on the web. These queries vary in terms of length, complexity, and specificity, representing different aspects of information needs and user intents.

3. **Passage Selection**: The passages in the dataset are selected based on their relevance to the corresponding queries. Each query is associated with a set of relevant passages, which are manually annotated by human judges. These passages are intended to provide relevant information or answer the query to the best extent possible.

4. **Annotation and Evaluation**: The relevance of passages to queries is assessed using human judgments. Human annotators are tasked with evaluating the relevance of passages to each query based on their understanding of the query intent and the information provided in the passage. The judgments are typically binary (relevant or not relevant) or graded on a relevance scale.

5. **Challenges**: Despite its richness and size, the MS MARCO TREC-DL dataset poses several challenges for machine learning models. These challenges include understanding complex queries, identifying relevant information from noisy passages, handling ambiguity and diversity in query intents, and generalizing to unseen queries and passages.

4.2 Experiments

To effectively demonstrate the diverse applications of our library and underscore its role in explaining document rankings for specific queries, we will begin by selecting a sample query. Subsequently, we will compile a list of the top-ranked documents relevant to this query. Through the application of our axioms and explanations to these documents, we aim to enhance our understanding of their respective rankings.

4.2.1 Use Case 1

The chosen query for this demonstration is:

query id: "183378"
query Text: "exons definition biology"

We take a list of the top 20 passages from the *"msmarco-passage/trec-dl-2019"* dataset for the given query. After that we take the first and fifth document and see whether we can explain the difference in their ranking using our library.

First rank document is as follows:

Body of Document ID 7135097:
Herbivory: Definition & Examples 4:44 Next Lesson.....

Fifth rank document is as follows:

Body of Document ID 719550:
Chitin (biology) definition,meaning online encyclopedia.....

We use the following axiom classes -

$$axiom\_classes = [TFC1(), PROX1(), DIV()]$$

Before we apply the axioms, we apply stemming and stop words removal as part of pre-processing.

We take document 1 as the top ranked document (doc id: 7135097) and document 2 as the fifth ranked document (doc id: 719550). We apply our library to get the following results –

(Note: As mentioned, a value of 1 indicates a preference for Document 1, -1 indicates a preference for Document 2, and 0 indicates no preference between the documents.)

Query	Document 1	Document 2	TFC1()	PROX1()	DIV()
exon definit bilog	herbivori definit....	chitin (biolog).....	1	0	-1

Table 4: pairwise.explain() in Use Case 1

4.2.2 Use Case 2

The results returned by our library in the previous use case were quite intriguing because TFC1() aligns with our ranking criteria, while DIV() diverges from our ranking, and PROX1() doesn't exhibit any discernible preference. Let's delve deeper by taking a look at the internal functioning to gain a clearer understanding of why these results are occurring using "*explain_details()*" -

Using explain_details on **TFC1()** for the given query document pair –

Term Frequency of query terms in document1 is 4
Term Frequency of query terms in document2 is 2

As we can see, document 1 exhibits a higher cumulative term frequency for all query terms compared to document 2 thus demonstrating that in this case higher term frequency of query terms results in higher relevance to the query (might not be true always).

Using explain_details on **DIV()** for the given query document pair –

Jaccard Co-efficient of doc1 is:0.09090909090909091
Jaccard Co-efficient of doc2 is:0.07692307692307693

Here since Jaccard Co-efficient of document 1 is higher than that of document 2 with the qu ery terms according to the definition of the axiom, document 2 is preferred (which contradic ts the ranking). However, we can clearly see that the difference is very small (less than 0.02). This suggests that the Jaccard coefficient alone may not be a decisive factor in determining t he ranking of these two documents according to the axiom.

Using explain_details on **PROX1()** for the given query document pair –

Metric	Document 1	Document 2
tf(exon)	0	0
tf(definit)	3	1
tf(bilog)	1	1
avg_dist(exon, definit)	0	0
avg_dist(exon, biolog)	0	0
avg_dist(definit, biolog)	8.5	0
num pairs	3	3
Total_avg_dist	2.83	0

Table 5: explain_details() on Prox1 in Use case 2

Here we can see that the term frequency of the first query term is 0 for both documents indicating not all query terms are present in both documents thus as per rules of the axiom, PROX1() would not be able to give any preferences.

4.2.2 Use Case 3

We can combine and weight the axioms using binary and unary operators. We have defined the library functions such that after any linear combinations the scores would be normalized to $\{-1,0,1\}$ using the unary "+".

$$pref_{+A}(q, d_i, d_j) = \begin{cases} 1, & if\ \ pref_A(q, d_i, d_j) > 0 \\ -1, & if\ \ pref_A(q, d_i, d_j) < 0 \\ 0, & if\ \ pref_A(q, d_i, d_j) = 0 \end{cases}$$

For demonstration let us take use case 1 and define new axiom class by combining the axioms in the following way –

$$axiom\_classes = [3*LNC1() + 4*TFC1(), -DIV(), PROX1() + PROX2() + PROX3()]$$

we get the following results (keeping the query, document1 and document 2 same as use case 1) by applying our library –

Query	Document 1	Document 2	4*TFC1() + 3*LNC1()	PROX1()+PROX2()+PROX3()	- DIV()
exon definit bilog	herbivori definit....	chitin (biolog).....	1	0	1

Table 6: Pairwise.explain() on Use Case 3

Here we have combined the axioms as per the following logic:

TFC1() prefers documents with higher term frequencies of query terms whereas LNC1() prefers documents with shorter lengths, we have combined them so that our new axiom gives preference to both (kind of like TF_LNC) however, we give slightly more weightage to tf values over document lengths. We have combined 3 proximity axioms to ensure documents containing query terms near to one other and in the beginning are preferred. Finally we have negated the DIV() axiom to ensure preference is given to less diverse documents.

Thus, this approach allows us to experiment by logically combining multiple axioms to determine document preferences based on our specific criteria. By synthesizing these axioms, we can effectively gauge the relevance of documents to a given query according to our tailored logic.

4.2.4 Use Case 4

To fully explore the capabilities of proximity axioms, let us use another dataset within MS Marco with larger documents where all query terms are present. We'll then execute the *"explain_details"* function to analyze the preferences generated by the axioms involved. This will provide comprehensive insights into how proximity-based considerations influence document rankings based on the given query.

So, we take the same query:

query id: "183378"
query Text: "exons definition biology"

We take the *"msmarco-document/trec-dl-2019"* dataset. MS Marco also has documents assigned a particular relevance level based on how relevant they are for a particular query. The relevance levels are defined as follows:

- 3: Perfectly relevant
- 2: Highly Relevant
- 1: Relevant
- 0: Irrelevant

We take doc id: D1077802 (which has been assigned a relevance of 3 for the given query) as document 1 and doc id: D1806793 (relevance 1) as document 2.

We run both TFC1() and PROX1() and find that where TFC1() prefers document 2, PROX1() prefers document 1 To further understand this phenomenon we run "explain_details" on **PROX1()** and get the following table –

Metric	Document 1	Document 2
tf(exon)	23	21
tf(definit)	7	56
tf(bilog)	1	25
avg_dist(exon, definit)	174.43	2728.07
avg_dist(exon, biolog)	354.71	3287.24
avg_dist(definit, biolog)	315.04	2864.24
num pairs	3	3
Total_avg_dist	281.39	2959.85

Table 7: explain_details on PROX1 in Use Case 4

We also run PROX2() and PROX3() on the given setup. We find that PROX2() prefers document 1 whereas PROX3() doesn't return any preference. Using "explain_details" on PROX2() returns the following –

Query Term	Document	First Occurrence
exon	1	27
definit	1	38
biolog	1	51
exon	2	1604
definit	2	24
biolog	2	82

Table 8: explain_details on PROX2() Use Case 4

Whereas "explain_details" on PROX3() returns that the entire query phrase doesn't appear in either of the two documents thus helping us understand the preferences returned by our two axioms.

Therefore, a key insight emerges: despite document 2 having higher term frequencies for query terms, these terms are dispersed throughout the document. This sparse distribution contrasts with document 1, where query terms are closely clustered together. Additionally, query terms in document 1 appear earlier compared to document 2. These proximity factors significantly influence relevance determination. This underscores that the mere presence of more query terms doesn't automatically translate to higher relevance. Proximity and placement of query terms are crucial criteria in assessing document relevance alongside term frequency.

Chapter 5 - Future Work

In future one direction we can move into is to incorporate the IR axioms into the pre-training process of neural ranking models. There are studies have indicated that incorporating these axioms can enhance the effectiveness and interpretability of IR models. However, there has been little effort to integrate these axioms into pre-training methods for IR.

One such method proposed was a novel pre-training method called Axiomatic Regularization for ad hoc Search (ARES) is proposed [13]. ARES uses a set of re-organized IR axioms to generate training samples that guide neural rankers in learning desirable ranking properties during the pre-training process. This approach makes ARES more intuitive and explainable compared to existing pre-training methods.

5.1 ARES (Axiomatic Regularization for ad-hoc Search)

The Axiomatic Regularization for ad hoc Search (ARES) framework addresses the challenge of incorporating IR axioms into the pre-training of neural ranking models, given the difficulty of directly applying axioms that require pairwise document comparisons when only documents are available. ARES consists of three main stages:

1. Pseudo Query Sampling (PQS): Generates pseudo queries from documents using a contrastive sampling strategy. This stage ensures that each document in the corpus can be associated with a set of pseudo queries, facilitating the next steps.

2. Preference Predictor Constructing (PPC): Collects ordered query pairs through four different sampling settings. Extracts axiomatic features for each query pair, which are essential for training. Trains a preference predictor (an axiomatic binary decision model) using the axiomatic feature map and weak preference labels. This predictor helps in understanding which query-document pairs should be ranked higher based on the axioms.

3. Axiomatically Regularized Pre-training (ARP): Integrates query pairs regularized with axioms into the pre-training process. Instead of relying on document pairs, ARES uses these query pairs to guide the neural ranking model, embedding the axiomatic knowledge directly into the model's training.

However, testing this model on standard datasets revealed only marginal improvements in test metrics. This indicates a need for a more refined approach to enhance its performance further. Future work should focus on developing a more elegant solution that builds upon ARES, incorporating pairwise explanations to improve interpretability and effectiveness.

Bibliography

[1] *Alexander Bondarenko, Maik Fröbe, Jan Heinrich Reimer, Benno Stein, Michael Völske, and Matthias Hagen. 2022. Axiomatic Retrieval Experimentation with ir_axioms. In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22). Association for Computing Machinery, New York, NY, USA, 3131–3140. https://doi.org/10.1145/3477495.3531743*

[2] *Sourav Saha et al. "ir_explain: a Python Library of Explainable IR Methods". In: arXiv preprint arXiv:2404.18546 (2024).*

[3] *Matthias Hagen, Michael Völske, Steve Göring, and Benno Stein. Axiomatic result re-ranking. In Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016. ACM, 721–730. https: //doi.org/10.1145/2983323.2983704*

[4] *N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. J.ACM, 55(5), 2008.*

[5] *H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In Proceedings of SIGIR 2004, pp. 49–56.*

[6] *T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In Proceedings of SIGIR 2007, pp. 295–302.*

[7] *Y. Lv and C. Zhai. Lower-bounding term frequency normalization. In Proceedings of CIKM 2011, pp. 7–16*

[8] *H. Fang and C. Zhai. Semantic term matching in axiomatic approaches to information retrieval. In Proceedings of SIGIR 2006, pp. 115–122.*

[9] *H. Fang. A re-examination of query expansion using lexical resources. In Proceedings of ACL 2008, pp. 139–147.*

[10] *W. Zheng and H. Fang. Query aspect based term weighting regularization in information retrieval. In Proceedings of ECIR 2010, pp. 344–356.*

[11] *H. Wu and H. Fang. Relation based term weighting regularization. In Proceedings of ECIR 2012, pp. 109–120.*

[12] *S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In Proceedings of WWW 2009, pp. 381–390.*

[13] *Jia Chen, Yiqun Liu, Yan Fang, Jiaxin Mao, Hui Fang, Shenghao Yang, Xiaohui Xie, Min Zhang, and Shaoping Ma. 2022. Axiomatically Regularized Pre-training for Ad hoc Search. In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22). Association for Computing Machinery, New York, NY, USA, 1524–1534. https://doi.org/10.1145/3477495.3531943*