Axiomatic Retrieval Experimentation with ir_axioms

Alexander Bondarenko[1,⋆] Maik Fröbe[1,⋆] Jan Heinrich Reimer,[1,⋆]
Benno Stein[2] Michael Völske[2] Matthias Hagen[1]

[1]Martin-Luther-Universität Halle-Wittenberg
<first>.<last>@informatik.uni-halle.de

[2]Bauhaus-Universität Weimar
<first>.<last>@uni-weimar.de

ABSTRACT

Axiomatic approaches to information retrieval have played a key role in determining basic constraints that characterize good retrieval models. Beyond their importance in retrieval theory, axioms have been operationalized to improve an initial ranking, to "guide" retrieval, or to explain some model's rankings. However, recent open-source retrieval frameworks like PyTerrier and Pyserini, which made it easy to experiment with sparse and dense retrieval models, have not included any retrieval axiom support so far.

To fill this gap, we propose ir_axioms, an open-source Python framework that integrates retrieval axioms with common retrieval frameworks. We include reference implementations for 25 retrieval axioms, as well as components for preference aggregation, re-ranking, and evaluation. New axioms can easily be defined by implementing an abstract data type or by intuitively combining existing axioms with Python operators or regression. Integration with PyTerrier and ir_datasets makes standard retrieval models, corpora, topics, and relevance judgments—including those used at TREC—immediately accessible for axiomatic experimentation. Our experiments on the TREC Deep Learning tracks showcase some potential research questions that ir_axioms can help to address.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**.

KEYWORDS

Axiomatic Thinking for Information Retrieval; Software Framework; Software Toolkit; Evaluation

ACM Reference Format:
Alexander Bondarenko, Maik Fröbe, Jan Heinrich Reimer, Benno Stein, Michael Völske, 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), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3477495.3531743

⋆These authors contributed equally to the paper and are listed alphabetically.

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531743

1 INTRODUCTION AND BACKGROUND

In the field of information retrieval, many open-source frameworks and toolkits are available that provide a convenient way to experiment with all kinds of retrieval models on standard corpora. One of the early examples is the Terrier platform [29]—open-sourced in 2004—that supports retrieval and learning-to-rank experiments on most of the document collections used at TREC; the recent PyTerrier framework [30] builds on it. Also the Lucene-based Java toolkit Anserini [45, 46] and its Python interface Pyserini [23] were developed to enable experiments with sparse retrieval models and easy access to the majority of the data used at TREC. Tightly integrated with Pyserini, PyGaggle [34] is a toolkit for neural ranking and question answering (e.g., using BERT-based [12] or T5-based [37] pairwise and pointwise rankers). Other toolkits and libraries that implement sparse and dense retrieval include MatchZoo [19], OpenNIR [26], Capreolus [47], OpenMatch [24], and Tevatron [17].

However, besides the many implemented retrieval models and evaluation metrics, so far, none of these frameworks, libraries, and toolkits include components for retrieval axioms. Proposed in the more theoretical field of axiomatic thinking for information retrieval, axioms are constraints that a "good" retrieval model should fulfill—often in the form of ranking preferences between documents. Recently, axioms were also practically applied in re-ranking experiments [20], to meta-learn how to combine the scores of different retrieval models [2], to regularize neural retrieval models [40], or to analyze and explain rankers [8, 16, 27, 38, 42]. To close the gap of axiom support in major retrieval toolkits, we develop ir_axioms: a Python framework to conduct retrieval experiments with axioms.[1]

Following the best practices of existing retrieval frameworks and toolkits, ir_axioms is a modular software that allows to effortlessly build pipelines and combine axioms using a set of special operators. Tightly integrated with PyTerrier and ir_datasets [28], ir_axioms can be used with a wide range of retrieval models, indexers, collections, and evaluation functions. All axioms are implemented as class objects that can be parameterized (e.g., to change an axiom's preconditions). Further axioms can easily be defined by extending the Axiom class. A KwikSortReranker module helps to implement axiomatic re-ranking experiments, and the AxiomaticExperiment module provides functionality for axiomatic analyses. An interesting feature of ir_axioms is to localize document pairs in some ranking that are incorrectly ordered according to manual relevance judgments along with the respective axiom preferences. This allows for a deeper analysis and better understanding of a retrieval model's incorrect ranking decisions. We demonstrate the functionality of ir_axioms on such example use cases in the scenario of the TREC 2019 and 2020 Deep Learning tracks [10, 11].

[1]GitHub: https://github.com/webis-de/ir_axioms/
Python package: https://pypi.org/project/ir_axioms/

2 AXIOMS FOR INFORMATION RETRIEVAL

Retrieval axioms are basic constraints that a good ranking function should fulfill. For instance, the axiom TFC1 states that documents with more query term occurrences should be ranked higher [13]. This constraint obviously can be "wrong" (e.g., if a document with more occurrences is a near-duplicate of another document already high in the ranking, a document with fewer occurrences might be preferable). The term 'retrieval axiom' thus needs to be interpreted a bit different than, for instance, axioms in geometry or probability theory. Still analyzing how well ranking functions satisfy specific retrieval axioms has led to improved ranking functions [25]—but also to the identification of incompatible axiom sets [18].

Table 1 shows a selection of retrieval axioms from the literature. Making use of term frequency, term discrimination power (e.g., idf), or term similarity (e.g., WordNet-based), many of these axioms induce a partial order on documents: a preference $d_1 >_A d_2$ implies that d_1 should be ranked higher than d_2 according to axiom A.

More recent studies have tried to integrate axiomatic ideas directly into the retrieval process: (1) improving an initial retrieval result via re-ranking according to weighted axiom preferences [6, 20], (2) using axioms as regularization loss in neural models [40], and (3) explaining neural rankings [8, 38, 42]. For an easier setup of such studies, `ir_axioms` implements 25 axioms and makes them available for experiments with standard retrieval toolkits.

Table 2 shows the axioms currently implemented in `ir_axioms`. Following Hagen et al. [20] and Völske et al. [42], our implementations try to incorporate an axiom's formalized idea while also making it applicable in practical settings. We thus reformulate the axioms to work for arbitrary queries (e.g., the original TFC1 requires single-term queries), to express pairwise preferences (e.g., DIV), etc.[2] Also, we include parameters to relax or strengthen some pre- and filter conditions (e.g., allowing for some small relative document length difference in TFC1). For axioms employing term similarity (REG and STMC1/2), we provide two variants based on WordNet synsets [32] or on fastText embeddings [4]. Axioms that cannot be easily reformulated to yield practically usable pairwise ranking preferences are not included (e.g., TFC2 or LB2).

Most recent among the currently implemented axioms are the argumentativeness axioms [5, 6] that aim at queries that probably require more argumentative results (e.g., `mercy killing`). The axiom ArgUC favors documents with more argument units (premises and claims), the axioms QTArg and QTPArg favor documents where the query terms appear closer to the argument units and closer to the beginning of the document, while the axiom aSLDoc favors more "readable" documents (average sentence length: 12–20 words).

Finally, we also include the axioms ORIG [20] and ORACLE. The preferences of ORIG simply follow some original ranking (e.g., a ranking returned by BM25 [39] or any other retrieval model). This enables weighting comparisons of an initial ranking to some weighted combinations of other axioms. The preferences of ORACLE are intended to follow human judgments if available (e.g., preferences between any documents from different relevance groups in TREC qrel files). This allows to incorporate "ground-truth" relevance information into experimental axiomatic settings.

[2] Complete list of changes to the original axiom formulations: https://github.com/webis-de/ir_axioms/blob/main/documentation/axioms.md

Table 1: Selected axioms from previous studies (intuition, pre- (in gray) and filter conditions, concluded ranking preference).

TFC1 [13] "Prefer documents with more query term occurrences."
Given $q = \{t\}$ and d_1, d_2 with $|d_1| = |d_2|$.
If $\mathrm{tf}(t, d_1) > \mathrm{tf}(t, d_2)$ then $d_1 >_{\text{TFC1}} d_2$

TFC2 [14] "Additional query term occurrences yield smaller retrieval score improvements."
Given $q = \{t\}$ and d_1, d_2, d_3 with $|d_1| = |d_2| = |d_3|$ and $\mathrm{tf}(t, d_1) > 0$, let $\Delta s_{i,j} = \mathrm{score}(t, d_i) - \mathrm{score}(t, d_j)$.
If $\mathrm{tf}(t, d_2) - \mathrm{tf}(t, d_1) = 1 \quad \wedge \quad \mathrm{tf}(t, d_3) - \mathrm{tf}(t, d_2) = 1$ then $\Delta s_{2,1} >_{\text{TFC2}} \Delta s_{3,2}$

TFC3 [14] "Prefer documents with occurrences of more query terms."
Given $q = \{t_1, t_2\}$ and d_1, d_2 with $|d_1| = |d_2|$ and $\mathrm{idf}(t_1) = \mathrm{idf}(t_2)$.
If $\mathrm{tf}(t_1, d_2) = \mathrm{tf}(t_1, d_1) + \mathrm{tf}(t_2, d_1) \quad \wedge$ $\mathrm{tf}(t_2, d_2) = 0 \quad \wedge \quad \mathrm{tf}(t_1, d_1) \neq 0 \quad \wedge \quad \mathrm{tf}(t_2, d_1) \neq 0$ then $d_1 >_{\text{TFC3}} d_2$

TDC [14] "Prefer documents with more discriminative query terms."
Given $q = \{t_1, t_2\}$ and d_1, d_2 with $|d_1| = |d_2|$, let d be some document.
If $\mathrm{idf}(t_1) > \mathrm{idf}(t_2) \quad \wedge \quad t_1 \in d_1 \quad \wedge \quad t_2 \notin d_1$ $\wedge \quad t_1 \notin d_2 \quad \wedge \quad t_2 \in d_2$ then $d \circ d_1 >_{\text{TDC}} d \circ d_2$

LNC1 [14] "Penalize longer documents for non-relevant terms."
Given q and d_1, d_2, let $t \notin q$ and t' denote some arbitrary terms.
If $\mathrm{tf}(t, d_1) + 1 = \mathrm{tf}(t, d_2) \quad \wedge \quad \forall_{t' \neq t}\, \mathrm{tf}(t', d_1) = \mathrm{tf}(t', d_2)$ then $d_1 \geq_{\text{LNC1}} d_2$

LNC2 [14] "Do not prefer shorter documents when matched query term ratio is the same."
Given q, term $t \in q$, and d_1, d_2, let t' denote some arbitrary term.
If $\exists_{c>1}(\ |d_1| = c \cdot |d_2| \quad \wedge \quad \mathrm{tf}(t, d_2) > 0 \quad \wedge$ $\forall_{t'}\, \mathrm{tf}(t', d_1) = c \cdot \mathrm{tf}(t', d_2)\ )$ then $d_1 \geq_{\text{LNC2}} d_2$

TF-LNC [14] "Reward additional query terms more than document length is penalized."
Given $q = \{t\}$ and d_1, d_2.
If $\mathrm{tf}(t, d_1) > \mathrm{tf}(t, d_2) \wedge |d_1| = |d_2| + \mathrm{tf}(t, d_1) - \mathrm{tf}(t, d_2)$ then $d_1 >_{\text{TF-LNC}} d_2$

LB1 [25] "Do not override the term presence–absence gap with length normalization."
Given $t \in q$ and d_1, d_2 with $\mathrm{score}(q \setminus \{t\}, d_1) = \mathrm{score}(q \setminus \{t\}, d_2)$.
If $\mathrm{tf}(t, d_1) > 0 \wedge \mathrm{tf}(t, d_2) = 0$ then $d_1 >_{\text{LB1}} d_2$

LB2 [25] "Repeated query term occurrence is less important than first occurrence."
Given $q = \{t_1, t_2\}$ and d_1, d_2 with $\mathrm{idf}(t_1) = \mathrm{idf}(t_2)$, $\mathrm{tf}(t_2, d_1) = \mathrm{tf}(t_2, d_2) = 0$, $\mathrm{tf}(t_1, d_1) > 0$, $\mathrm{tf}(t_1, d_2) > 0$, and let $t \in d_1$ and $t' \in d_2$ be terms not in q.
If $\mathrm{score}(q, d_1) = \mathrm{score}(q, d_2)$ then $d_1 \cup \{t_2\} \setminus \{t\} >_{\text{LB2}} d_2 \cup \{t_1\} \setminus \{t'\}$

REG [48] "Prefer documents covering more different query aspects."
Given $q = \{t_1, t_2, t_3\}$ and d_1, d_2 with $|d_1| = |d_2|$, $\mathrm{idf}(t_2) = \mathrm{idf}(t_3)$, $\mathrm{sim}(t_1, t_2) > \mathrm{sim}(t_1, t_3)$, and $\mathrm{sim}(t_1, t_2) > \mathrm{sim}(t_2, t_3)$.
If $\mathrm{tf}(t_1, d_1) = \mathrm{tf}(t_1, d_2) > 0 \quad \wedge \quad \mathrm{tf}(t_3, d_1) =$ $\mathrm{tf}(t_2, d_2) > 0 \quad \wedge \quad \mathrm{tf}(t_2, d_1) = \mathrm{tf}(t_3, d_2) = 0$ then $d_1 >_{\text{REG}} d_2$

AND [43] "Prefer documents containing all query terms."
Given $q = \{t_1, t_2\}$ with $\mathrm{idf}(t_1) \geq \mathrm{idf}(t_2)$.
If $\mathrm{tf}(t_1, d_1) = 1 \quad \wedge \quad \mathrm{tf}(t_2, d_1) = 1 \quad \wedge$ $\mathrm{tf}(t_1, d_2) > 1 \quad \wedge \quad \mathrm{tf}(t_2, d_2) = 0$ then $d_1 >_{\text{AND}} d_2$

STMC1 [15] "Prefer documents with terms more similar to query terms."
Given $q = \{t\}$, $d_1 = \{t_1\}$, and $d_2 = \{t_2\}$ with $t \neq t_1$, $t \neq t_2$.
If $\mathrm{sim}(t, t_1) > \mathrm{sim}(t, t_2)$ then $d_1 >_{\text{STMC1}} d_2$

STMC2 [15] "Do not reward similar terms more than exact matches."
Given $q = \{t\}$ and d_1, d_2 with $|d_1| = 1$, $|d_2| = c > 0$, and term t' with $\mathrm{sim}(t, t') > 0$.
If $\mathrm{tf}(t, d_1) = 1 \quad \wedge \quad \mathrm{tf}(t', d_2) = c$ then $d_1 \geq_{\text{STMC2}} d_2$

PROX1 [20] "Prefer documents with shorter distance between query term pairs."
Given q and d_1, d_2 with $d_1 \cap q = d_2 \cap q = q$, let $P = \{(t, t') : t, t' \in q, t \neq t'\}$ and let $\delta(d, t, t')$ denote the average number of words between the terms t and t' in d.
If $\sum_{(t,t') \in P} \delta(d_1, t, t') < \sum_{(t,t') \in P} \delta(d_2, t, t')$ then $d_1 >_{\text{PROX1}} d_2$

PROX2 [20] "Prefer documents with earlier query term occurrences."
Given q and d_1, d_2 with $d_1 \cap q = d_2 \cap q = q$, let $\mathrm{first}(t, d)$ denote the position of the first occurrence of t in d.
If $\sum_{t \in q} \mathrm{first}(t, d_1) < \sum_{t \in q} \mathrm{first}(t, d_2)$ then $d_1 >_{\text{PROX2}} d_2$

PROX3 [20] "Prefer documents where the query occurs earlier as a phrase."
Given q and d_1, d_2 with $d_1 \cap q = d_2 \cap q = q$, let $\tau(q, d)$ denote the first position of the whole query q as one phrase in d (set to ∞ if q does not occur as a phrase).
If $\tau(d_1, q) < \tau(d_2, q)$ then $d_1 >_{\text{PROX3}} d_2$

PROX4 [20] "Prefer documents that contain all query terms in a shorter substring."
Given q and d_1, d_2 with $d_1 \cap q = d_2 \cap q = q$, let $\omega(d, q)$ denote the length of the shortest substring of d that contains all query terms.
If $\omega(d_1, q) < \omega(d_2, q)$ then $d_1 >_{\text{PROX4}} d_2$

PROX5 [20] "Prefer documents where the query terms are closer together on average."
Given q and d_1, d_2 with $d_1 \cap q = d_2 \cap q = q$, let $\bar{s}(d, q)$ denote the avg. length of the shortest substrings of d that contain all query terms (average over all occurrences of query terms in d).
If $\bar{s}(d_1, q) < \bar{s}(d_2, q)$ then $d_1 >_{\text{PROX5}} d_2$

Table 2: The axioms in `ir_axioms` grouped by their objectives; the similarity-based axioms (*) are each implemented in two variants using WordNet synsets or fastText embeddings.

Objective	Implemented Axioms
Term frequency	TFC1 [13], TFC3 [14], M-TDC [41]
Document length	LNC1, TF-LNC [13]
Lower-bounding term freq.	LB1 [25]
Query aspects	REG* [48], AND [43], DIV [18]
Semantic similarity	STMC1*, STMC2* [15]
Term proximity	PROX1–PROX5 [20]
Argumentativeness	ArgUC, QTArg, QTPArg, aSLDoc [5, 6]
Other	ORIG [20], ORACLE

3 THE `ir_axioms` FRAMEWORK

The `ir_axioms` framework has a modular architecture. Each axiom is implemented as a Python class that extends the abstract base class `Axiom`. The `Axiom` class ensures type safety when defining (new) axioms and is a core element for computing axiom preference matrices. Each subclass of `Axiom` models the preference $>_A$ of an axiom A by returning a numeric preference value $pref_A(q, d_i, d_j)$ for a pair of documents d_i, d_j given a query q as follows:

$$\begin{aligned} pref_A(q, d_i, d_j) > 0 &\iff d_i >_A d_j\,, \\ pref_A(q, d_i, d_j) < 0 &\iff d_i <_A d_j\,, \\ pref_A(q, d_i, d_j) = 0 &\iff d_i \not>_A d_j \;\wedge\; d_i \not<_A d_j\,. \end{aligned}$$

Note that a 'zero' preference is returned when an axiom does not express a preference (e.g., if the axiom's preconditions are not met).

Each axiom is registered in a central registry of `ir_axioms` that provides a convenient way to search for a specific axiom by its name. The built-in `to_axiom()` helper and the `AutoAxiom` class take care of parsing the name and create an axiom instance that then can be directly used for axiomatic experimentation.

Since many axioms rely on statistics like term frequencies (e.g., TFC1) or document length (e.g., LNC1), we allow an `Axiom` instance to directly access a document index and its statistics (cf. Section 3.1). When an axiom requires access to a document's content (i.e., text), `ir_axioms` offers three ways in the following order of priority until available: (1) via a `TextDocument` object, (2) via a document index, or (3) via an `ir_datasets` [28] corpus storage. The index structure and the required document collection are customizable.

3.1 Integration with Existing IR Frameworks

Deriving an axiom's ranking preferences often relies on document collection properties captured in an inverted index. Hence, we expose an `IndexContext` interface that specifies all required methods (e.g., tokenization, term frequencies, etc.) to integrate `ir_axioms` with Pyserini [23] and PyTerrier [30]. Both Pyserini's underlying Lucene index [3] and PyTerrier's underlying Terrier index [29] can be directly accessed by axioms through a unified interface.

Architecturally, `ir_axioms` largely follows PyTerrier and is fully compatible allowing to directly call PyTerrier functions that implement retrieval pipelines in Python ranging from sparse retrieval (e.g., BM25) to learning to rank (e.g., LambdaMART [7]) and to dense retrieval (e.g., ColBERT [22]) and neural re-ranking (e.g., monoT5 and duoT5 [36]). Through PyTerrier, `ir_axioms` also has easy access to a variety of benchmark datasets and pre-built indices from `ir_datasets` [28]. PyTerrier's data model uses a pandas `DataFrame` [31] to represent a set of documents, or a set of queries, or the results retrieved for each query, etc. Functions (called transformers in PyTerrier) can be implemented to transform a `DataFrame` (e.g., from queries to ranked documents). Special operators can be used to combine transformers in a pipeline, passing the output from one transformer to the next.

```
# Linear combination of TFC1 and TFC3.
tfc = TFC1() + (TFC3() * 2)

# Conjunction of the PROX axioms.
prox = PROX1() & PROX2() & PROX3() & PROX4() & PROX5()

# Fallback to the ORIG axiom if STMC1 returns 0.
stmc1_orig = STMC1() | ORIG()
```

Listing 1: Examples of combining multiple axioms using the 'add', 'multiply', 'conjunction', and 'cascade' operators.

Since PyTerrier allows to extend transformers by user-defined transformer classes, in `ir_axioms` we include custom axiom-specific transformers that can directly be used in retrieval pipelines (cf. Table 3). For instance, a transformation $R \longrightarrow R'$ means to re-rank a ranking R by applying some modification, whereas $R \longrightarrow R_f$ denotes some feature extraction from R. This way, `ir_axioms` deeply integrates with PyTerrier's declarative definitions of retrieval pipelines and its transparent data model. As a result, axioms can be conveniently applied to the rankings produced by any of the many retrieval models implemented in PyTerrier.

3.2 Schemes to Combine and Weight Axioms

Each retrieval axiom serves as a proxy for a single important constraint that a good ranking function should fulfill. Combining different axioms (i.e., their preference decisions) into ensembles then can be effective [20]. In `ir_axioms`, configurable axiom groupings enable such ensembles and can act as axioms themselves, which can in turn be combined with other axioms. Combination or manipulation of axioms is governed by arithmetic and logic Python operators that are overloaded in `ir_axioms` to allow for declarative axiom expressions and intuitive experimentation (cf. Table 4). The examples in Listing 1 show how axioms can be combined. One possibility is a weighted linear combination of the preference values using point-wise addition and scalar multiplication:

$$\begin{aligned} pref_{A+B}(q, d_i, d_j) &= pref_A(q, d_i, d_j) + pref_B(q, d_i, d_j)\,, \\ pref_{x \cdot A}(q, d_i, d_j) &= x \cdot pref_A(q, d_i, d_j)\,. \end{aligned}$$

Besides linear combinations, axioms can also be combined "conjunctively" in various flavors. By using the `&`-operator or the aggregation class `AndAxiom`, a non-zero preference is returned iff all axioms return the same non-zero preference (all axioms agree). By using the `VoteAxiom` class, the conjunctive agreement can be relaxed so that each axiom "votes" for the preference individually. The majority vote is returned iff it reaches a specified threshold (passed as an argument to the `VoteAxiom` class; e.g., 50% for the absolute majority as in the second example in Table 5). Hence, `AndAxiom` is equivalent

Table 3: Axiom-specific transformer classes, their types, and description. Following the PyTerrier transformer notations, R denotes ranking and f denotes feature extraction.

Transformer Class	Type	Description
`AggregatedPreferences`	$R \rightarrow R_f$	Aggregate axiom preferences for each document (cf. Section 4.3.2).
`EstimatorKwikSortReranker`	$R \rightarrow R'$	Train estimator for ORACLE, use it to re-rank with KwikSort (cf. Section 4.3.3).
`KwikSortReranker`	$R \rightarrow R'$	Re-rank using axiom preferences aggregated by KwikSort (cf. Section 4.3.1).
`PreferenceMatrix`	$R \rightarrow (R \times R)_f$	Compute an axiom's preference matrix (cf. Section 4.1).

Table 4: Operators in `ir_axioms` that allow to combine, modify, and cache preferences of retrieval axioms.

Op.	Name	Description
Binary		
`+`	*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.
`|`	*cascade*	Fallback if an axiom preference is 0.
`&`	*conjunction*	Return preference if all axioms agree.
`%`	*majority*	Absolute majority vote of involved axioms.
Unary		
`~`	*cache*	Cache axiom preferences on disk.
`-`	*negate*	Negate axiom preferences.
`+`	*normalize*	Normalize axiom preferences to (-1, 0, 1).

to `VoteAxiom` with a required majority of 100%. When an axiom returns no preference, a cascade axiom can define a fallback:

$$pref_{A|B}(q, d_i, d_j) = \begin{cases} pref_A(q, d_i, d_j) & \text{if } \; pref_A(q, d_i, d_j) \neq 0\,, \\ pref_B(q, d_i, d_j) & \text{otherwise}\,. \end{cases}$$

The last example in Listing 1 uses the 'cascade' operator to return the ORIG preference in case that STMC1 returns a 'zero' preference.

Bondarenko et al. [5] proposed three weighting schemes by which some axiom set may "overrule" an ORIG preference (i.e., the initial ranking): equal weights, majority voting, and total agreement. Table 5 shows formulations of these schemes with the combination operators and aggregation axiom classes of `ir_axioms`.

The aggregation of axioms in `ir_axioms` is optimized for efficiency by early stopping the preference computation in a cascade (`CascadeAxiom`) or a logical conjunction (`AndAxiom`). For example, if in a conjunction A_1 & ... & A_n the A_1-preference is 0, then the conjunctive preference $pref_{A_1 \,\&\, \ldots \,\&\, A_n}(q, d_i, d_j)$ must also be 0, and the preference computation can be skipped for all other axioms.

3.3 Preference Normalization and Caching

Besides the above binary operators, `ir_axioms` also includes unary operators: negation, normalization, and caching (cf. Table 4). Negating an axiom (unary '-') simply inverts its preferences:

$$pref_{-A}(q, d_i, d_j) = -\, pref_A(q, d_i, d_j)\,.$$

Table 5: Weighting schemes for a set of axioms $A_1, \ldots, A_n$ to overrule ORIG; `ir_axioms` classes and short operator notation.

Scheme	Usage examples
Equal weights	`SumAxiom([`A_1`,...,`A_n`,ORIG()])`
(A_i as import. as ORIG)	A_1 `+ ... +` A_n `+ ORIG()`
Majority vote	`VoteAxiom([`A_1`,...,`A_n`], 0.5) | ORIG()`
(>50% need to agree)	`(`A_1 `% ... %` A_n`) | ORIG()`
Total agreement	`AndAxiom([`A_1`,...,`A_n`]) | ORIG()`
(all A_i need to agree)	`(`A_1 `& ... &` A_n`) | ORIG()`

```
# Normalizing the combined STMC-preferences.
normalized_stmc = +(STMC1() + STMC2())

# Caching the preferences of ArgUC.
cached_arguc = ~ArgUC()
```

Listing 2: Normalize and cache axiom preferences.

Since preference values are not restricted (e.g., $pref_A(q, d_i, d_j) = 42$ and $pref_A(q, d_i, d_j) = 0.815$ both express $d_i >_A d_j$), it can be advisable to sometimes re-calibrate them (e.g., after linear combinations). In `ir_axioms`, the preference values of any axiom or combination can be normalized to $1, 0, -1$ using the unary '+':

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

The first example in Listing 2 normalizes an STMC-combination.

A final unary operator '~' offers some convenience functionality for experiments in which the same axioms are applied more than once on similar document sets (e.g., rankings of different retrieval models for the same query with many overlapping documents). To avoid re-computing the same axiom preferences over and over in such scenarios, `ir_axioms` provides a caching mechanism using the `diskcache` library[3] to store preferences in an SQLite database. The second example in Listing 2 caches ArgUC preferences—quite a costly axiom due to calls of the external `TARGER` API [9]. But also other axioms benefit from caching. For example, in our experiments, enabling caching speeds up the computation of STMC1 preferences for the top-20 results of runs from the TREC 2020 Deep Learning track by 52% for the second run file.

[3] https://pypi.org/project/diskcache/

```python
@dataclass
class NewAxiom(Axiom):
  name = "NEW"
  # Initialization / parameters.

  def preference(self, C_index, q, d_i, d_j) → float:
    # Preference for d_i and d_j, given a query q.
    # C_index is an IndexContext object that allows to
    # access the term frequencies in an index.
```

Listing 3: Defining a new axiom by extending the `Axiom` class.

3.4 Defining New Axioms

Adding new axioms to `ir_axioms` is a straightforward task: the abstract base class `Axiom` can be extended as shown in Listing 3. Assigning a `name` (optional) registers the new axiom in the `ir_axioms` registry and overriding the `preference()` method ensures that the new axiom's preference values $pref_{\text{NEW}}(q, d_i, d_j)$ are returned. Python's built-in `__repr__()` method can be implemented to support caching. A new axiom can also be parameterized—similar to, for instance, STMC1 that allows to choose the similarity function or the argumentative axioms that allow to choose the pre-trained `TARGER` model. For newly defined axioms, customizability can be achieved by adding all parameters as class properties and decorating the class with the `@dataclass` decorator. This automatically adds a constructor with the required parameters and implements the `__repr__()` method for caching preference values. All axioms in `ir_axioms` are implemented as data classes.

4 EXPERIMENTS AND USE CASES

We demonstrate the functionality of `ir_axioms` on three use cases: (1) post-hoc axiomatic analyses of rankings and judgments from shared retrieval tasks, (2) example-based analyses of rankings with respect to axiom violations, and (3) axiomatic result re-ranking in several variants. For our experiments, we use the setup of the TREC 2019 and 2020 Deep Learning tracks [10, 11]. All (in-)equalities in the axioms' pre- and filter conditions are parameterized with a 10% margin so that equality conditions are "softened" to allow for some slight differences and inequality conditions are "strengthened" to require differences of at least 10% to express a preference.

4.1 Post-Hoc Axiomatic Analyses

Post-hoc axiomatic analyses of a shared retrieval task could ask questions like how consistent the relevance judgments and the submitted rankings are with axiom preferences. Such analyses are supported by a range of utility functions in `ir_axioms`. The `AxiomaticExperiment` class provides the entry-point to a post-hoc analysis. Listing 4 shows how an `AxiomaticExperiment` is instantiated by passing as parameters the rankings (from run files), the topics, the relevance judgments, an index location, and the axioms to use; similar to how an `Experiment` is instantiated in PyTerrier [30]. The `preferences` member of an `AxiomaticExperiment` then provides access to all axiom preferences in a pandas `DataFrame`.

4.1.1 Axiom Preference Distributions. To calculate the distribution of axiom preferences for all rankings that were submitted in the Deep Learning tracks, we use the `preference_distribution` method

```python
experiment = AxiomaticExperiment(
  [bm25, monot5, ...],
  dataset.get_topics(),
  dataset.get_qrels(),
  index,
  axioms=[ArgUC(), QTArg(), QTPArg(), ...]
)

# Pairwise axiom preferences.
experiment.preferences
```

Listing 4: Example of an experiment returning pairwise axiom preferences with retrieval models passed in a list.

from `AxiomaticExperiment` that returns the distribution of all preferences in a pandas `DataFrame`. Analyzing such distributions provides an overview of which axioms might be interesting in the context of a shared task. Table 6 shows the absolute numbers of pairs in the top-10 results of the submitted runs for which an axiom had no preference or matched / did not match the preference in the ranking (i.e., ORIG). Interestingly, most axioms express relatively few preferences (e.g., TFC1 only for 4–13% of the pairs; reason: precondition ensures that the axiom is applied only to documents of about the same length). Still, when an axiom expresses a preference, this preference tends to agree with a ranking more often than not (exception: DIV). In the extreme cases of TFC3 and M-TDC with hardly any preferences, their restrictive pre- and filter conditions are the cause (e.g., hardly any query has terms of approximately equal inverse document frequency). On the other end of the spectrum are the axioms REG, DIV, STMC1, PROX1, and PROX2 that all rather more often express a preference than not.

4.1.2 Consistency of Rankings or Judgments with Axioms. Contradictions between axiom preferences and rankings or relevance judgments can be identified by calling the `preference_consistency` function of the `AxiomaticExperiment` class. Table 7 shows the consistency of some selected axioms' positive preferences with the preferences of the nDCG@10-wise best Deep Learning track runs from different categories (columns 'NNLM' (neural networks with language models), 'NN' (neural networks), and 'Trad' (traditional); categories from the Deep Learning track organizers) and with the relevance judgments of a task (column 'Qrel'). From the axiom groups with the same objectives, we selected the axioms with the most non-zero preferences in Table 6. In all tasks of the Deep Learning tracks, the best language model-based neural runs (NNLM) were more effective than the best neural runs (NN) and the best neural runs were more effective than the best traditional runs (Trad).

The positive axiom preferences agree with the relevance judgments in 66–81% of the cases on all tasks, while the agreement with the best runs usually is lower. This hints at some potential for possible ranking improvements by deeper axiomatic analyses of a run's decisions (cf. Section 4.2). For instance, the REG, DIV, STMC1, PROX1, and PROX2 axioms have substantially higher agreement with the judgments than with the best runs and they are the axioms that most often express preferences (cf. Table 6). It thus seems that further diversifying a run's top results and matching the query

Table 6: Absolute numbers of how often an axiom had no preference (A = 0) or returned a preference matching (A = ORIG) or not matching (A ≠ ORIG) the ranking preference of the pairs in the top-10 results of the runs submitted to the TREC 2019 and 2020 Deep Learning tracks (passage and document retrieval; a ranking with 10 results contributes 45 pairs).

Axiom	Passage'19 (43 topics, 37 runs)			Document'19 (43 topics, 38 runs)			Passage'20 (54 topics, 59 runs)			Document'20 (45 topics, 64 runs)		
	$A=0$	$A=$ORIG	$A\neq$ORIG	$A=0$	$A=$ORIG	$A\neq$ORIG	$A=0$	$A=$ORIG	$A\neq$ORIG	$A=0$	$A=$ORIG	$A\neq$ORIG
TFC1	52,692	3,681	2,834	60,193	1,250	1,178	104,568	8,582	6,406	104,133	2,633	2,068
TFC3	59,185	16	6	62,621	0	0	119,473	69	14	108,834	0	0
M-TDC	58,963	124	120	62,621	0	0	119,188	204	164	108,824	7	3
LNC1	57,239	1,106	862	61,862	375	384	115,659	2,131	1,766	108,047	440	347
TF-LNC	56,298	1,594	1,315	62,477	54	90	112,730	3,684	3,142	108,555	155	124
LB1	45,773	8,444	4,990	51,826	6,190	4,605	86,437	20,499	12,620	91,574	10,296	6,964
REG (WordNet)	30,935	15,730	12,542	28,784	19,050	14,787	54,445	36,900	28,211	39,629	39,797	29,408
REG (fastText)	7,524	26,587	25,096	4,699	30,458	27,464	22,273	52,659	44,624	9,368	55,644	43,822
AND	46,165	9,076	3,966	44,394	11,669	6,558	93,102	19,119	7,335	79,073	19,137	10,624
DIV	2,528	24,783	31,896	736	28,625	33,260	4,124	47,922	67,510	1,007	48,856	58,971
STMC1 (WordNet)	3,103	29,493	26,611	3,541	30,386	28,694	2,922	61,163	55,471	3,240	53,386	52,208
STMC1 (fastText)	347	31,820	27,040	647	32,585	29,389	674	65,157	53,725	842	56,278	51,714
STMC2 (WordNet)	54,165	2,710	2,332	60,853	894	874	107,298	6,795	5,463	105,022	1,770	2,042
STMC2 (fastText)	51,904	4,081	3,222	59,222	1,749	1,650	104,647	7,760	7,149	102,819	3,176	2,839
PROX1	30,059	17,035	12,113	26,207	21,664	14,750	69,860	28,647	21,049	38,876	41,558	28,400
PROX2	28,401	18,054	12,752	26,070	21,878	14,673	67,369	30,383	21,804	38,751	41,897	28,186
PROX3	56,610	1,675	922	55,565	4,768	2,288	116,254	1,992	1,310	99,443	6,438	2,953
PROX4	43,212	9,591	6,404	34,231	17,440	10,950	94,600	14,678	10,278	46,255	38,273	24,306
PROX5	40,258	10,565	8,384	33,633	17,266	11,722	91,564	15,587	12,405	45,555	37,678	25,601
ArgUC	51,316	4,145	3,746	58,140	2,357	2,124	100,771	9,890	8,895	100,318	4,141	4,375
QTArg	52,538	4,192	2,477	58,674	2,435	1,512	103,448	9,705	6,403	100,512	5,239	3,083
QTPArg	50,710	5,001	3,496	58,428	2,456	1,737	99,778	11,919	7,859	100,434	5,020	3,380
aSL	54,105	2,649	2,453	61,030	860	731	107,544	6,343	5,669	105,638	1,471	1,725

terms or some semantically similar terms earlier in result documents that contain the query terms closer to each other might be good ideas to further improve the best runs' top ranks.

Among the individual axioms, AND has the highest agreement rates with the Deep Learning track's best runs and AND, PROX1, PROX2, and QTArg have the highest agreement rates with the judgments (i.e., matching all query terms close to each other and close to argumentative units seems to be good). The axiom with the lowest agreement rates is DIV. Still, also for DIV, the difference between the agreement with the judgments compared to the agreement with the best runs is substantial (24–34 percentage points).

Among the runs, the best traditional runs seem to agree more with most axioms, while the best neural and language model-based runs are less consistent with the axioms but achieved better effectiveness values at the Deep Learning tracks. This might hint at gaps in the current axiom categories. The few axioms and axiom categories studied so far are definitely not covering all aspects of relevance. The identification of axioms for other angles of relevance thus is an interesting direction for future work; also concluded earlier by, for instance, Hagen et al. [20], Rennings et al. [38], Câmara and Hauff [8], and Völske et al. [42] in their studies on axiomatically improving, diagnosing, or explaining rankings.

4.2 Example-based Analyses of Rankings

Besides general agreement of some system's rankings with axioms, also a deeper inspection of particular ranking errors can be interesting to better understand the underlying decisions. One could focus such an analysis on pairs where a ranking violates a particular axiom or where a ranking disagrees with the "ground-truth" relevance judgments. The `inconsistent_pairs` method of the `AxiomaticExperiment` class can be used to identify pairs that a retrieval system ranked incorrectly according to the judgments (i.e., a less relevant document is ranked higher than a more relevant one). This is not exactly the diagnostic setup of Rennings et al. [38] or Câmara and Hauff [8] who created artificial datasets for specific axioms but can be viewed as close to their underlying idea of using axioms to diagnose some system's decisions or problems.

Table 8 shows an example pair along with axioms that express preferences for it. Such a case-by-case analysis could be a starting point to better understand causes of ranking errors. In the example, the run's underlying BERT-based model (type NNLM) also generally is not that consistent with TFC1, REG, and STMC1 (50–56%; cf. column 'NNLM' for Passage'19 in Table 7). Hence, specifically fine-tuning the model on pairs where its ranking violates TFC1, REG, and STMC1 might be an idea for some effectiveness improvement.

4.3 Axiomatic Result Re-Ranking

Nowadays, retrieval pipelines often re-rank the top-k results of a baseline model (e.g., BM25) by using learning-to-rank or neural methods. Also axiom combinations have been applied as re-rankers to improve the retrieval effectiveness [20].

4.3.1 KwikSort to Aggregate Rankings from Axiom Preferences. In their axiomatic re-ranking experiments, Hagen et al. [20] used the

Table 7: Consistency (in percent) of some selected axioms' positive preferences with the preferences in the top-10 results of the best submitted run based on neural networks (NN) or on neural networks and large language models (NNLM), the best traditional run (Trad), and the respective non-negative relevance judgments from the TREC 2019 and 2020 Deep Learning tracks (Qrel); fastText embeddings used for REG and STMC1, each top-10 ranking contributes 45 preference pairs. A non-negative judgment preference counts as a match for a positive axiom preference (i.e., axiom preferences within a relevance level are acceptable). The best runs from the respective tasks are: Passage'19: idst_bert_p1 (NNLM), TUW19-p3-f (NN), srchvrs_ps_run3 (Trad); Document'19: idst_bert_v3 (NNLM), TUW19-d3-re (NN), srchvrs_run1 (Trad); Passage'20: pash_r3 (NNLM), TUW-TK-Sparse (NN), bl_bcai_mdl1_vt (Trad); Document'20: d_d2q_duo (NNLM), ndrm3-orc-full (NN), bl_bcai_multfld (Trad).

Axiom	Passage'19 (43 topics)				Document'19 (43 topics)				Passage'20 (54 topics)				Document'20 (45 topics)			
	NNLM	NN	Trad	Qrel	NNLM	NN	Trad	Qrel	NNLM	NN	Trad	Qrel	NNLM	NN	Trad	Qrel
TFC1	56	56	57	78	48	54	66	73	60	56	56	69	56	54	52	76
LB1	66	63	73	78	50	49	60	75	71	65	65	72	59	53	62	76
REG	50	54	57	76	51	53	54	74	55	52	54	71	57	57	56	73
AND	68	78	82	81	62	61	66	75	74	79	77	80	64	66	68	76
DIV	45	42	39	73	46	43	44	73	40	42	38	66	44	45	44	71
STMC1	53	55	56	76	54	52	55	70	57	53	55	71	54	53	51	71
PROX1	58	58	57	79	67	60	59	76	61	57	58	76	59	63	62	78
PROX2	58	59	58	80	65	61	61	76	61	58	60	78	59	63	62	79
ArgUC	47	51	57	75	52	46	59	75	53	52	48	71	46	46	41	71
QTArg	55	65	64	81	57	51	75	81	62	53	61	73	69	61	70	78

Table 8: A pair from a ranking of the most effective run that participated in the TREC 2019 Deep Learning passage retrieval task. A highly relevant document is returned at rank 5, lower than a less relevant document at rank 3. This pair, for instance, violates TFC1, REG, and STMC1 but is consistent with PROX1, PROX2, and ArgUC; fastText embeddings used for REG and STMC1.

Run: idst_bert_p1			**Query:** 207786 how are some sharks warm blooded	Selected axioms with preferences					
Rank	**Doc. ID**	**rel**	**Content**	**TFC1**	**REG***	**STMC1***	**PROX1**	**PROX2**	**ArgUC**
3	7941579	1	Great white sharks are some of the only warm blooded…	⇓	⇓	⇓	⇑	⇑	⇑
5	2763917	2	These sharks can raise their temperature about the…	⇑	⇑	⇑	⇓	⇓	⇓

KwikSort method [1] from the field of computational social choice to derive a final ranking from axiom preferences. In `ir_axioms`, KwikSort is implemented as a PyTerrier transformer operation. This way, any KwikSort re-ranking can be directly evaluated on test collections using the PyTerrier `Experiment` class [30].

Note, however, that a KwikSort-aggregated ranking can contradict individual axiom preferences (e.g., when combining several axioms' preferences, they might differ on some document pair). The axiom combination and weighting schemes implemented in `ir_axioms` (cf. Section 3.2) can help to avoid some such situations. For example, in Listing 5, three axioms are conjunctively combined to re-rank the top-20 results of BM25 with the ORIG axiom as a fallback when the three axioms do not agree. However, some circular aggregation issues are still possible. For instance, in the example of Listing 5, the three axioms could favor d_i over d_j and d_j over d_k but may not agree on a preference for d_i and d_k for which the fallback ORIG might then favor d_k over d_i. In such cases, a KwikSort aggregation will still contradict at least one of the preferences.

```
bm25 = BatchRetrieve(index, "BM25")

# If ArgUC, QTArg, and QTPArg don't agree, use ORIG.
axiom = (ArgUC() & QTArg() & QTPArg()) | ORIG()

# Re-rank the BM25 top-20 using KwikSort.
kwiksort = bm25 % 20 >> \
  KwikSortReranker(axiom, index)

pipeline = kwiksort ^ bm25
```

Listing 5: Re-ranking BM25 based on axiomatic preferences.

4.3.2 Axiom Preferences as Features for Learning to Rank. Besides directly aggregating rankings from (weighted) pairwise axiom preferences via KwikSort, the preferences could also be interesting features for arbitrary learning-to-rank approaches like LambdaMART [44]. An axiom A's preferences for the top-k results of some baseline ranker form a $k \times k$ preference matrix $P_A = [p_{ij}] \in \mathbb{R}^{k \times k}$ with $p_{ij} = pref_A(q, d_i, d_j)$. To create a characteristic preference feature for each document d_i in a to-be-re-ranked top-k result list, `ir_axioms` allows to combine the preferences for d_i (i.e., the entries from the i-th matrix row) using simple functions like the

```
bm25 = BatchRetrieve(index, "BM25")

# Aggregate ArgUC, QTArg, etc. by mean/median.
axioms = [ArgUC(), QTArg(), ...]
aggs = [mean, median]
# Compute features for top-10 results.
features = bm25 % 10 >> \
  AggregatedAxiomaticPreferences(axioms, index, aggs)

# Train LambdaMART re-ranker.
lmart = LGBMRanker(objective="lambdarank")
ltr = features >> apply_learned_model(lmart, "ltr")
ltr.fit(train_topics, train_qrels,
        dev_topics, dev_qrels)

pipeline = ltr ^ bm25
```

Listing 6: Re-ranking BM25 with LambdaMART using `mean`- and `median`-aggregated axiom preferences as features.

median or the arithmetic mean:

$$f_{\text{mean}}(d_i) = \frac{\sum_{j=1}^{k} p_{ij}}{k}.$$

In a feature vector, aggregated preferences from multiple axioms and/or multiple aggregation functions can be combined.

In `ir_axioms`, the aggregation of preferences to learning-to-rank features is implemented as a PyTerrier transformer operator. Listing 6 shows an example in which a LambdaMART re-ranker is trained on `mean`- and `median`-aggregated axiom preferences as features to re-rank the top-10 results of BM25.

4.3.3 Re-Ranking by Estimating the ORACLE Axiom. The ORACLE axiom represents the ranking preferences implicitly stated in the human relevance judgments of a shared retrieval task. Formally, if $\text{rel}(q, d_i) > \text{rel}(q, d_j)$ then $d_i >_{\text{ORACLE}} d_j$; informally, documents with better relevance judgments should be preferred. Re-ranking a baseline's top-k results using KwikSort on the ORACLE preferences would yield a perfect ranking of the k documents (but not necessarily an overall perfect ranking since the top-k might miss some highly relevant documents). However, in practical scenarios without relevance judgments, ORACLE preferences are not available. Still, the ORACLE preferences from shared tasks and test collections could be used to train an estimator for unseen document pairs by combining the preferences of other axioms [20].

In `ir_axioms`, a special class `EstimatorAxiom` is implemented that can be used to predict an arbitrary target axiom's preferences via a classification or regression method from scikit-learn [35] with the preferences of a pre-defined set of other axioms as features. During training, the target axiom preferences need to be available (e.g., judgments for some test collections in case of the ORACLE axiom) but in the later estimation phase for unseen pairs, only the preferences of the other axioms are needed as input to the estimator. On the estimated preferences, KwikSort can be run to generate a final ranking. Listing 7 shows an example using the `EstimatorKwikSortRanker` that combines ORACLE estimation and KwikSort re-ranking in a single PyTerrier module.

4.3.4 Axiomatic Re-Ranking Experiments on MS MARCO. To evaluate axiomatic re-ranking with `ir_axioms`, we conduct experiments

```
bm25 = BatchRetrieve(index, "BM25")

# Estimate ORACLE based on ArgUC, QTArg, etc.
axioms = [ArgUC(), QTArg(), ...]
rf = RandomForestClassifier()

# Train Random Forest classifier on top-20 results.
kwiksort_rf = bm25 % 20 >> \
  EstimatorKwikSortReranker(axioms, rf, index)
kwiksort_rf.fit(train_topics, train_qrels)

pipeline = kwiksort_rf ^ bm25
```

Listing 7: Re-ranking BM25 using KwikSort aggregation on a trained estimator for the ORACLE preferences.

on the MS MARCO passage retrieval dataset [33] using the relevance judgments from the TREC 2019 Deep Learning track [11] for training and the ones from the TREC 2020 Deep Learning track [10] for testing. In the experiments, we re-rank the top-20 BM25 results using three different strategies: (1) KwikSort with all axioms, (2) KwikSort on an estimated ORACLE axiom, and (3) LambdaMART with preferences from all axioms as features.

For the first KwikSort re-ranker, all 25 axioms (cf. Table 2) are combined in an absolute majority voting scheme with a fallback option to the ORIG axiom (like in the second example in Table 5). In the second KwikSort re-ranker, the ORACLE axiom's preferences are estimated by a random forest classifier that uses all 25 axioms as features (training on the TREC 2019 Deep Learning track for 100 iterations, tree depth set to 3).

For the LambdaMART re-ranker, we aggregate the preferences of each axiom A for each document using three aggregation functions:

$$f_{\text{hi}}(d_i) = \frac{\left|\{d_j : \; d_i \geq_A d_j, \;\; j = 1, \ldots, k\}\right|}{k},$$

$$f_{\text{lo}}(d_i) = \frac{\left|\{d_j : \; d_i \leq_A d_j, \;\; j = 1, \ldots, k\}\right|}{k},$$

$$f_{\text{eq}}(d_i) = \frac{\left|\{d_j : \; d_i =_A d_j, \;\; j = 1, \ldots, k\}\right|}{k}.$$

These features represent the number of documents above or below which d_i could be ranked according to A, and how often A does not express a preference. The LambdaMART re-ranker is trained on the topics of the TREC 2019 Deep Learning track for 1000 iterations (no tuning of the LambdaMART parameters, though).

Table 9 shows the retrieval effectiveness of the baseline and the re-rankers as measured by normalized discounted cumulative gain (nDCG) [21] that was also used at the TREC 2020 Deep Learning track. Since KwikSort and LambdaMART are not deterministic, we let the re-rankers each create 10 rankings per query and apply 5-fold cross-validation to obtain average evaluation results.

Our basic experiments are meant to demonstrate the functionality of `ir_axioms` (i.e., no parameter tuning, etc.). Still, the results of the KwikSort-RF and LambdaMART re-rankers show the potential of axiom preferences for retrieval effectiveness improvements—none of the differences are statistically significant, though. Using aggregated axiom preferences as learning-to-rank features and combining them with other common features could be an interesting

Table 9: Retrieval effectiveness on the TREC 2020 Deep Learning track (passage retrieval task) when re-ranking the top-20 passages of BM25 with the three axiomatic re-ranking strategies: (1) KwikSort with majority voting (KwikSort-MV), (2) KwikSort with a random forest estimator for the ORACLE axiom (KwikSort-RF), and (3) LambdaMART with axiom preference features.

(Re-)Ranker	nDCG@5	nDCG@10
BM25 (init. rank.)	0.497	0.494
KwikSort-MV	0.496	0.492
KwikSort-RF	0.516	**0.498**
LambdaMART	**0.517**	**0.498**

direction for further axiomatic experiments. Additionally, formulating new axioms that capture different angles of relevance also seems to be a very promising direction. With `ir_axioms` at their fingertips, researchers working on any of these topics can now simply focus on expressing their respective axiomatic ideas and then rely on the framework for conducting their experiments.

5 CONCLUSIONS AND FUTURE WORK

With the open-source framework `ir_axioms`, we provide tools to experiment with retrieval axioms—basic constraints that are meant to characterize good ranking functions. Implementations of 25 commonly used and well-understood retrieval axioms provide a good starting point to experiment with axiomatic re-ranking and evaluation. Since `ir_axioms` is tightly integrated with PyTerrier, it can be combined with a wide range of retrieval models, test collections, and evaluation functions. This makes it easy to incorporate axiomatic approaches into state-of-the-art retrieval pipelines.

Our use cases show the potential of conducting axiomatic retrieval experiments with `ir_axioms` in a declarative, easy-to-understand way. Further use cases could be to actually reproduce some of the results of recent axiomatic studies like the re-ranking experiments of Hagen et al. [20] and Bondarenko et al. [6], the meta-learning experiments of Arora and Yates [2], the regularization experiments of Rosset et al. [40], or the diagnosis and explanation experiments of Rennings et al. [38], Câmara and Hauff [8], Formal et al. [16], Völske et al. [42], and MacAvaney et al. [27]. However, so far, not all of these experiments are supported in `ir_axioms` (exception: re-ranking). Still, other types of experiments can further be added to `ir_axioms`, and we gratefully accept contributions.[4]

Previous studies and also our experiments indicate that the current axioms are somewhat limited in their expressiveness. Axiomatically capturing further angles of relevance is an interesting direction for future work. Similar to new experiments, also new axioms can be directly added to `ir_axioms`. On the technical side, we plan to add parallelization to speed up axiomatic analyses (e.g., for computing axiom preference matrices), and we plan to simplify axiomatic consistency checks at various points in multi-stage retrieval pipelines.

[4] GitHub: https://github.com/webis-de/ir_axioms/
Python package: https://pypi.org/project/ir_axioms/

ACKNOWLEDGMENTS

This work has been partially supported by the DFG through the project "ACQuA 2.0: Answering Comparative Questions with Arguments" (grant HA 5851/2-2) as part of the priority program "RATIO: Robust Argumentation Machines" (SPP 1999).

REFERENCES

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM* 55, 5 (2008), 23:1–23:27. https://doi.org/10.1145/1411509.1411513

[2] Siddhant Arora and Andrew Yates. Investigating retrieval method selection with axiomatic features. In *Proceedings of the 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval co-located with the 41st European Conference on IR Research, AMIR@ECIR 2019 (CEUR Workshop Proceedings, Vol. 2360)*. CEUR-WS.org, 18–31. http://ceur-ws.org/Vol-2360/paper4Axiomatic.pdf

[3] Andrzej Bialecki, Robert Muir, and Grant Ingersoll. Apache Lucene 4. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval, OSIR@SIGIR 2012*. University of Otago, 17–24.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146. https://transacl.org/ojs/index.php/tacl/article/view/999

[5] Alexander Bondarenko, Maik Fröbe, Vaibhav Kasturia, Matthias Hagen, Michael Völske, and Benno Stein. Webis at TREC 2019: Decision track. In *Proceedings of the Twenty-Eighth Text REtrieval Conference, TREC 2019 (NIST Special Publication, Vol. 1250)*. NIST, 4 pages. https://webis.de/publications.html#bondarenko_2019

[6] Alexander Bondarenko, Michael Völske, Alexander Panchenko, Chris Biemann, Benno Stein, and Matthias Hagen. Webis at TREC 2018: Common Core track. In *Proceedings of the Twenty-Seventh Text REtrieval Conference, TREC 2018 (NIST Special Publication, Vol. 500-331)*. NIST, 3 pages. https://webis.de/publications.html#bondarenko_2018

[7] Christopher J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. *Learning* 11, 23-581 (2010), 81.

[8] Arthur Câmara and Claudia Hauff. Diagnosing BERT with retrieval heuristics. In *Proceedings of the 42nd European Conference on IR Research, ECIR 2020 (Lecture Notes in Computer Science, Vol. 12035)*. Springer, 605–618. https://doi.org/10.1007/978-3-030-45439-5_40

[9] Artem N. Chernodub, Oleksiy Oliynyk, Philipp Heidenreich, Alexander Bondarenko, Matthias Hagen, Chris Biemann, and Alexander Panchenko. TARGER: Neural argument mining at your fingertips. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019*. ACL, 195–200. https://doi.org/10.18653/v1/p19-3031

[10] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 Deep Learning track. *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020 (NIST Special Publication, Vol. 1266)*. NIST. https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf

[11] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. Overview of the TREC 2019 Deep Learning track. *Proceedings of the Twenty-Eighth Text REtrieval Conference, TREC 2019 (NIST Special Publication, Vol. 1250)*. NIST. https://trec.nist.gov/pubs/trec28/papers/OVERVIEW.DL.pdf

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*. ACL, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[13] Hui Fang, Tao Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2004*. ACM, 49–56. https://doi.org/10.1145/1008992.1009004

[14] Hui Fang, Tao Tao, and ChengXiang Zhai. Diagnostic evaluation of information retrieval models. *ACM Transactions on Information Systems* 29, 2 (2011), 7:1–7:42. https://doi.org/10.1145/1961209.1961210

[15] Hui Fang and ChengXiang Zhai. Semantic term matching in axiomatic approaches to information retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2006*. ACM, 115–122. https://doi.org/10.1145/1148170.1148193

[16] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. A white box analysis of ColBERT. In *Proceedings of the 43rd European Conference on IR Research, ECIR 2021 (Lecture Notes in Computer Science, Vol. 12657)*. Springer, 257–263. https://doi.org/10.1007/978-3-030-72240-1_23

[17] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Tevatron: An efficient and flexible toolkit for dense retrieval. *CoRR* abs/2203.05765 (2022). https://doi.org/10.48550/arXiv.2203.05765

[18] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*. ACM, 381–390. https://doi.org/10.1145/1526709.1526761

[19] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. MatchZoo: A learning, practicing, and developing system for neural text matching. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019*. ACM, 1297–1300. https://doi.org/10.1145/3331184.3331403

[20] 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

[21] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446. https://doi.org/10.1145/582415.582418

[22] Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020*. ACM, 39–48. https://doi.org/10.1145/3397271.3401075

[23] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021*. ACM, 2356–2362. https://doi.org/10.1145/3404835.3463238

[24] Zhenghao Liu, Kaitao Zhang, Chenyan Xiong, Zhiyuan Liu, and Maosong Sun. OpenMatch: An open source library for Neu-IR research. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021*. ACM, 2531–2535. https://doi.org/10.1145/3404835.3462789

[25] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011*. ACM, 7–16. https://doi.org/10.1145/2063576.2063584

[26] Sean MacAvaney. OpenNIR: A complete neural ad-hoc ranking pipeline. In *Proceedings of the 13th ACM International Conference on Web Search and Data Mining, WSDM 2020*. ACM, 845–848. https://doi.org/10.1145/3336191.3371864

[27] Sean MacAvaney, Sergey Feldman, Nazli Goharian, Doug Downey, and Arman Cohan. ABNIRML: Analyzing the behavior of neural IR models. *Transactions of the Association for Computational Linguistics* 10 (03 2022), 224–239. https://doi.org/10.1162/tacl_a_00457

[28] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. Simplified data wrangling with `ir_datasets`. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2021*. ACM, 2429–2436. https://doi.org/10.1145/3404835.3463254

[29] Craig Macdonald, Richard McCreadie, Rodrygo L. T. Santos, and Iadh Ounis. From puppy to maturity: Experiences in developing Terrier. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval, OSIR@SIGIR 2012*. University of Otago, 60–63.

[30] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. PyTerrier: Declarative experimentation in Python from BM25 to dense retrieval. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management, CIKM 2021*. ACM, 4526–4533. https://doi.org/10.1145/3459637.3482013

[31] Wes McKinney. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference, SciPy 2010*. 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

[32] George A. Miller. WordNet: A lexical database for English. *Communications of the ACM* 38, 11 (1995), 39–41. https://doi.org/10.1145/219717.219748

[33] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated MAchine Reading COmprehension dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016) (CEUR Workshop Proceedings, Vol. 1773)*. CEUR-WS.org. https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf

[34] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. ACL, 708–718. https://doi.org/10.18653/v1/2020.findings-emnlp.63

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. https://dl.acm.org/doi/10.5555/1953048.2078195

[36] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *CoRR* abs/2101.05667 (2021). https://arxiv.org/abs/2101.05667

[37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21 (2020), 140:1–140:67. https://jmlr.org/papers/v21/20-074.html

[38] Daniël Rennings, Felipe Moraes, and Claudia Hauff. An axiomatic approach to diagnosing neural IR models. In *Proceedings of the 41st European Conference on IR Research, ECIR 2019 (Lecture Notes in Computer Science, Vol. 11437)*. Springer, 489–503. https://doi.org/10.1007/978-3-030-15712-8_32

[39] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994 (NIST Special Publication, Vol. 500-225)*. NIST, 109–126. https://trec.nist.gov/pubs/trec3/papers/city.ps.gz

[40] Corby Rosset, Bhaskar Mitra, Chenyan Xiong, Nick Craswell, Xia Song, and Saurabh Tiwary. An axiomatic approach to regularizing neural ranking models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019*. ACM, 981–984. https://doi.org/10.1145/3331184.3331296

[41] Shuming Shi, Ji-Rong Wen, Qing Yu, Ruihua Song, and Wei-Ying Ma. Gravitation-based model for information retrieval. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2005*. ACM, 488–495. https://doi.org/10.1145/1076034.1076117

[42] Michael Völske, Alexander Bondarenko, Maik Fröbe, Benno Stein, Jaspreet Singh, Matthias Hagen, and Avishek Anand. Towards axiomatic explanations for neural ranking models. In *Proceedings of the 2021 ACM SIGIR International Conference on the Theory of Information Retrieval, ICTIR 2021*. ACM, 13–22. https://doi.org/10.1145/3471158.3472256

[43] Hao Wu and Hui Fang. Relation based term weighting regularization. In *Proceedings of the 34th European Conference on IR Research, ECIR 2012 (Lecture Notes in Computer Science, Vol. 7224)*. Springer, 109–120. https://doi.org/10.1007/978-3-642-28997-2_10

[44] Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270. https://doi.org/10.1007/s10791-009-9112-1

[45] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of Lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2017*. ACM, 1253–1256. https://doi.org/10.1145/3077136.3080721

[46] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using Lucene. *ACM Journal of Data and Information Quality* 10, 4 (2018), 16:1–16:20. https://doi.org/10.1145/3239571

[47] Andrew Yates, Kevin Martin Jose, Xinyu Zhang, and Jimmy Lin. Flexible IR pipelines with Capreolus. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM 2020*. ACM, 3181–3188. https://doi.org/10.1145/3340531.3412780

[48] Wei Zheng and Hui Fang. Query aspect based term weighting regularization in information retrieval. In *Proceedings of the 32nd European Conference on IR Research, ECIR 2010 (Lecture Notes in Computer Science, Vol. 5993)*. Springer, 344–356. https://doi.org/10.1007/978-3-642-12275-0_31