Alexander Bondarenko Maik Fröbe Jan Heinrich Reimer Benno Stein Michael Völske Matthias Hagen

Axiomatic Retrieval Experimentation with `ir_axioms`

Axiomatic Thinking

Successful retrieval scoring functions share similar properties:

$$BM25(q,d) = \sum_{i=1}^{n} IDF(t_i) \cdot \frac{TF(t_i,d) \cdot (k_1+1)}{TF(t_i,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)}$$

IDF weighting — TF weighting — Length normalization

Axioms formally capture such properties.

TFC1: prefer documents with more query terms [Fang et al., SIGIR'04]
LNC1: penalize non-query terms in longer doc's [Fang et al., TOIS 29(2)]

Axiom applications

- Improving an initial retrieval result via re-ranking [Hagen et al., CIKM'16]
- Using axioms as regularization loss in neural models [Rosset et al., SIGIR'19]
- Learning how to combine retrieval models [Arora and Yates, AMIR@ECIR'19]
- Analyzing / explaining neural rankers [Völske et al., ICTIR'21; Formal et al., ECIR'21]

The `ir_axioms` Framework

- Python framework for experiments with IR axioms
- Implements 25 axioms (parameterizable preconditions, multi-term queries, etc.)
- Access to retrieval models and test collections in PyTerrier and `ir_datasets`
- Caching and parallelization

Examples

Implemented axioms:

Objective	Axioms
Term frequency	TFC1, TFC3, TDC, M-TDC
Document length	LNC1, TF-LNC
Lower-bound TF	LB1
Query aspects	REG, AND, DIV
Semantic similarity	STMC1, STMC2
Term proximity	PROX1–PROX5
Argumentativeness	ArgUC, QTArg, QTPArg, aSLDoc
Other	ORIG, ORACLE

Implementing the TFC1 axiom:

```python
class TFC1(Axiom):
  name = "TFC1"
  def preference(self,c,q,d_i,d_j) → float:
    # Length precondition.
    if not approx_same_length(c,d_i,d_j,0.1):
      return 0
    # Count query terms.
    tf_i = sum(c.term_frequency(d_i,t)
      for t in c.terms(q))
    tf_j = sum(c.term_frequency(d_j,t)
      for t in c.terms(q))
    if approx_equal(tf_i,tf_j,0.1):
      return 0
    return 1 if tf_i > tf_j else -1
```

Combining axioms with operators:

```python
# Linear combination of TFC axioms.
tfc = TFC1() + (TFC3() * 2)

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

# Combine STMC in majority vote.
stmc = (STMC1() % STMC2()) | ORIG()

# Normalize combined preferences.
normalized_arg = +(QTArg() + QTPArg())

# Cache preferences of ArgUC.
cached_arguc = ~ArgUC()
```

Post-hoc Analysis

```python
bm25 = BatchRetrieve(index, "BM25")
monot5 = bm25 >> ...
experiment = AxiomaticExperiment(
  [bm25, monot5, ...],    # Retrieval systems
  dataset.get_topics(),   # Topics
  dataset.get_qrels(),    # Judgments
  index,                  # Document index
  axioms=[ArgUC(), QTArg(), QTPArg(), ...])
```

- Interface similar to PyTerrier's `Experiment`
- Pairwise axiomatic preferences: `experiment.preferences`
- Consistency with judgments: `experiment.preference_consistency`
- Analyzing inconsistent pairs: `experiment.inconsistent_pairs`

Incorrectly ranked documents (most effective run idst_bert_p1 at TREC 2019 DL passage retrieval): less relevant doc. at rank 3, more relevant at rank 5; violates TFC1 and STMC1, but consistent with PROX1.

Query: 207786 how are some sharks warm blooded				**Selected Axioms**		
Rank	**Doc. ID**	**Rel.**	**Content**	**TFC1**	**STMC1**	**PROX1**
3	7941579	1	Great white sharks are some of the only…	↓	↓	↑
5	2763917	2	These sharks can raise their temperature…	↑	↑	↓

Axiomatic Re-ranking

```python
# Re-rank top-20 BM25 results.
kwiksort = bm25 % 20 >> KwikSortReranker(
  (ArgUC() & QTArg() & QTPArg()) | ORIG(), index)

# Train LambdaMART with axiomatic features from top-10.
features = bm25 % 10 >> AggregatedAxiomaticPreferences(
  [ArgUC(), QTArg(), ...],
  index,
  [mean, median])
ltr = features >> apply_learned_model(LGBMRanker(...))
ltr.fit(train_topics,train_qrels,dev_topics,dev_qrels)
```

- Re-rank with KwikSort [Ailon et al., J. ACM 55(5)]
- Generate axiomatic features for LTR
- Estimate ORACLE preferences with axioms

Effectiveness for re-ranking top-20 of BM25 (TREC 2020 DL passages).

(Re-)Ranker	**nDCG@5**	**nDCG@10**
BM25 (initial ranking)	0.497	0.494
KwikSort with majority voting	0.496	0.492
KwikSort with a rand. forest estimator for ORACLE prefs.	0.516	**0.498**
LambdaMART with axiom preference features	**0.517**	**0.498**

Conclusions

`ir_axioms`:

- Implementation of 25 axioms
- Post-hoc analysis of rankings
- Axiomatic re-ranking pipeline
- Axiom preferences as features for LTR
- Caching and parallelization

Future work: further axioms, integration with other IR frameworks

Resources and Installation

Contributions and feedback are welcome!

`webis-de/ir_axioms`

`pip install ir_axioms`

`10.1145/3477495.3531743`

Thanks to the SIGIR for a student travel grant.

This work has been partially supported by the DFG through the project "ACQuA 2.0: Answering Comparative Questions with Arguments" (project number 376430233) as part of the priority program "RATIO: Robust Argumentation Machines" (SPP 1999).