retriever_BM25

April 27, 2025

[1]:
```python
# %%
import json
import os
import pickle

import numpy as np
from rank_bm25 import BM25Okapi
from tqdm import tqdm

from generate_dummy_data import generate_legal_corpus


class SimpleBM25Retriever:
    """
    A simple BM25 retriever implementation using rank_bm25 package.
    No Java dependencies required.
    """

    def __init__(self, index_name="legal_bm25_index"):
        self.index_name = index_name
        self.index_dir = os.path.join(os.getcwd(), index_name)
        os.makedirs(self.index_dir, exist_ok=True)
        self.bm25 = None
        self.tokenized_corpus = None
        self.documents = None
        self.doc_ids = None

    def tokenize(self, text):
        """Simple whitespace tokenization"""
        return text.lower().split()

    def index_corpus(self, documents, doc_ids):
        """Build BM25 index from documents"""
        print(f"Building BM25 index with {len(documents)} documents...")

        self.documents = documents
        self.doc_ids = doc_ids
```

```
        # Tokenize corpus
        print("Tokenizing documents...")
        self.tokenized_corpus = [self.tokenize(doc) for doc in tqdm(documents,␣
↪total=len(documents))]

        # Build BM25 index
        print("Building BM25 index...")
        self.bm25 = BM25Okapi(self.tokenized_corpus)

        # Save the index
        self.save_index()

        print("BM25 index built successfully")
        return self

    def save_index(self):
        """Save the index to disk"""

        # bm25.pkl: The BM25 scoring object with term frequencies and IDF values
        with open(os.path.join(self.index_dir, "bm25.pkl"), 'wb') as f:
            pickle.dump(self.bm25, f)

        # tokenized_corpus.pkl: The tokenized versions of all documents
        with open(os.path.join(self.index_dir, "tokenized_corpus.pkl"), 'wb')␣
↪as f:
            pickle.dump(self.tokenized_corpus, f)

        # documents.json: The original document texts
        with open(os.path.join(self.index_dir, "documents.json"), 'w') as f:
            json.dump(self.documents, f)

        # doc_ids.json: The document IDs
        with open(os.path.join(self.index_dir, "doc_ids.json"), 'w') as f:
            json.dump(self.doc_ids, f)

    def load_index(self):
        """Load pre-built BM25 index"""
        index_path = os.path.join(self.index_dir, "bm25.pkl")
        if not os.path.exists(index_path):
            raise ValueError(f"Index not found at {index_path}. Build index␣
↪first with index_corpus()")

        with open(index_path, 'rb') as f:
            self.bm25 = pickle.load(f)
```

```
        with open(os.path.join(self.index_dir, "tokenized_corpus.pkl"), 'rb')␣
↪as f:
            self.tokenized_corpus = pickle.load(f)

        with open(os.path.join(self.index_dir, "documents.json"), 'r') as f:
            self.documents = json.load(f)

        with open(os.path.join(self.index_dir, "doc_ids.json"), 'r') as f:
            self.doc_ids = json.load(f)

        print(f"Loaded BM25 index with {len(self.documents)} documents")
        return self

    def retrieve(self, query, k=100):
        """Retrieve top-k documents for a query"""
        if self.bm25 is None:
            self.load_index()

        # Tokenize query
        tokenized_query = self.tokenize(query)

        # Get scores
        scores = self.bm25.get_scores(tokenized_query)

        # Get top k document indices
        top_indices = np.argsort(scores)[::-1][:k]

        # Format results
        results = []
        for i in top_indices:
            if scores[i] > 0:  # Only include documents with non-zero scores
                results.append({
                    "id": self.doc_ids[i],
                    "score": float(scores[i]),
                    "text": self.documents[i]
                })

        return results

    def batch_retrieve(self, queries, k=100):
        """Retrieve top-k documents for multiple queries"""
        if self.bm25 is None:
            self.load_index()

        all_results = {}
        for i, query in enumerate(tqdm(queries, desc="Processing queries")):
            all_results[str(i)] = self.retrieve(query, k=k)
```

```
        return all_results

    def save_results(self, results, output_file):
        """Save retrieval results to file"""
        with open(output_file, 'w') as f:
            json.dump(results, f, indent=2)
        print(f"Saved retrieval results to {output_file}")
```

```
Generated 120 legal documents across 6 domains.
Saved to legal_dummy_corpus.json
Saved 15 sample queries to legal_sample_queries.json
```

[2]:
```
# %%
# Modified usage to work with your generated data
if __name__ == "__main__":
    # Load the generated corpus
    corpus_file = "legal_dummy_corpus.json"
    queries_file = "legal_sample_queries.json"

    # Check if corpus file exists, otherwise run data generation
    if not os.path.exists(corpus_file):
        print("Corpus file not found. Generating dummy legal corpus...")
        corpus = generate_legal_corpus(120, corpus_file)
        documents = corpus["documents"]
        doc_ids = corpus["doc_ids"]
    else:
        # Load existing corpus
        print(f"Loading corpus from {corpus_file}...")
        with open(corpus_file, 'r') as f:
            corpus_data = json.load(f)
            documents = corpus_data["documents"]
            doc_ids = corpus_data["doc_ids"]

    # Check if queries file exists
    if not os.path.exists(queries_file):
        print("Queries file not found. Using default queries...")
        queries = [
            "What are the essential elements of a valid contract?",
            "How is negligence defined in tort law?",
            "What constitutes probable cause for a search warrant?",
            "What rights are protected under the First Amendment?",
            "How does adverse possession work in property law?",
            "What is the standard for proving defamation?",
            "What are the remedies for breach of contract?",
            "How does the Fourth Amendment limit police searches?",
            "What is the process for appealing an administrative decision?",
```

```
            "What constitutes insider trading under securities regulations?",
            "How are easements created and terminated?",
            "What is the difference between murder and manslaughter?",
            "What are the requirements for a valid will?",
            "How does eminent domain work?",
            "What constitutes workplace discrimination?"
        ]
    else:
        # Load existing queries
        print(f"Loading queries from {queries_file}...")
        with open(queries_file, 'r') as f:
            queries = json.load(f)

    print(f"Corpus has {len(documents)} documents")
    print(f"Query set has {len(queries)} queries")

    # Initialize BM25 retriever
    retriever = SimpleBM25Retriever(index_name="legal_bm25_retr1")

    # Check if index already exists
    if os.path.exists(os.path.join(retriever.index_dir, "bm25.pkl")):
        print("Index already exists. Loading...")
        retriever.load_index()
    else:
        print("Building new index...")
        retriever.index_corpus(documents, doc_ids)

    # Retrieve results for queries
    results = retriever.batch_retrieve(queries, k=10)

    # Save results
    output_file = "bm25_retrieval_results.json"
    retriever.save_results(results, output_file)

    # Print sample results
    print("\nSample Retrieval Results:")
    print("========================")

    for i, query in enumerate(queries[:3]):  # Show results for first 3 queries
        print(f"\nQuery: {query}")
        print("-" * 80)
        results_for_query = results[str(i)]

        for j, doc in enumerate(results_for_query[:2]):  # Show top 2 documents
            print(f"Document {j+1}: (Score: {doc['score']:.4f})")
            print(f"ID: {doc['id']}")
```

```
            print(f"Text: {doc['text'][:200]}..." if len(doc['text']) > 200␣
↪else f"Text: {doc['text']}")
            print("-" * 40)
```

```
Loading corpus from legal_dummy_corpus.json…
Loading queries from legal_sample_queries.json…
Corpus has 120 documents
Query set has 15 queries
Index already exists. Loading…
Loaded BM25 index with 120 documents
Processing queries: 100%|      | 15/15 [00:00<00:00, 1814.30it/s]
Saved retrieval results to bm25_retrieval_results.json

Sample Retrieval Results:
========================

Query: What are the essential elements of a valid contract?
--------------------------------------------------------------------------------
Document 1: (Score: 4.4936)
ID: property_law_119
Text: EASEMENT: TransAmerica Logistics grants to Commonwealth of Jefferson a
perpetual easement for conservation over the property described as a 20-foot
wide strip along the western edge of the property. T…
----------------------------------------
Document 2: (Score: 4.1827)
ID: property_law_092
Text: EASEMENT: MediCorp grants to PacificRoute Services a perpetual easement
for conservation over the property described as a 20-foot wide strip along the
western edge of the property. This easement shall…
----------------------------------------

Query: How is negligence defined in tort law?
--------------------------------------------------------------------------------
Document 1: (Score: 5.5100)
ID: tort_law_073
Text: The tort claim filed by Sarah Williams asserts that Omega Corporation's
ignored industry safety standards constituted negligence and breached the duty
of care owed to plaintiff, causing severe physica…
----------------------------------------
Document 2: (Score: 3.2071)
ID: tort_law_083
Text: NEGLIGENCE CLAIM: Robert Johnson alleges that on August 30, 2022, Thomas
Anderson failed to {duty} resulting in emotional distress. The standard of care
required defendant to exacting, which was breac…
----------------------------------------

Query: What constitutes probable cause for a search warrant?
```

```
--------------------------------------------------------------------------------
Document 1: (Score: 7.2187)
ID: criminal_law_116
Text: SEARCH WARRANT application states that probable cause exists to believe
evidence of insider trading will be found at Central Park based on witness
statements observed by Officer C. Wilson on March 3, …
----------------------------------------
Document 2: (Score: 7.2187)
ID: criminal_law_068
Text: SEARCH WARRANT application states that probable cause exists to believe
evidence of bribery will be found at Downtown Financial District based on
digital communications observed by Officer D. Williams…
----------------------------------------
```