dense_retriever

April 27, 2025

[1]:
```
# %%
import os
import json
import pickle
import numpy as np
from tqdm import tqdm
import torch
from transformers import AutoTokenizer, AutoModel
import faiss
from huggingface_hub import login
from config import config
login(config['HF_API_KEY'])

class DenseRetriever:
    """
    Dense retriever implementation using a pre-trained language model
    for encoding documents and queries into vector representations.
    """

    def __init__(self, model_name="nlpaueb/legal-bert-base-uncased",␣
 ↪index_name="legal_dense_index"):
        """
        Initialize the dense retriever with a pre-trained language model.

        Args:
            model_name: Name of the pre-trained model to use for encoding
            index_name: Name for the index directory
        """
        self.model_name = model_name
        self.index_name = index_name
        self.index_dir = os.path.join(os.getcwd(), index_name)
        os.makedirs(self.index_dir, exist_ok=True)

        # Initialize model and tokenizer
        self.tokenizer = None
        self.model = None
        self.index = None
        self.documents = None
        self.doc_ids = None
        self.embedding_dim = None

    def _initialize_model(self):
        """Load the model and tokenizer if not already loaded"""
        if self.tokenizer is None:
            print(f"Loading tokenizer for {self.model_name}...")
            self.tokenizer = AutoTokenizer.from_pretrained(self.model_name)

        if self.model is None:
            print(f"Loading model {self.model_name}...")
            self.model = AutoModel.from_pretrained(self.model_name)
            self.model.eval()  # Set to evaluation mode

    def get_embedding(self, text, max_length=512):
        """Generate embedding for a single text"""
        # Ensure model and tokenizer are loaded
        self._initialize_model()

        # Tokenize input
        inputs = self.tokenizer(text, return_tensors="pt",␣
↪max_length=max_length,
                               padding="max_length", truncation=True)

        # Generate embeddings
        with torch.no_grad():
            outputs = self.model(**inputs)
            # Use CLS token embedding as text representation
            embedding = outputs.last_hidden_state[:, 0, :].cpu().numpy()

        return embedding[0]  # Return as 1D array

    def index_corpus(self, documents, doc_ids, batch_size=8):
        """
        Generate embeddings for all documents and build a search index.

        Args:
            documents: List of document texts
            doc_ids: List of document IDs corresponding to the documents
            batch_size: Batch size for processing documents
        """
        print(f"Building dense index with {len(documents)} documents...")

        # Store document texts and IDs
        self.documents = documents
        self.doc_ids = doc_ids
        # Ensure model and tokenizer are loaded
        self._initialize_model()

        # Generate embeddings for all documents
        print("Generating document embeddings...")
        all_embeddings = []

        for i in tqdm(range(0, len(documents), batch_size), desc="Processing␣
↪document batches"):
            batch_docs = documents[i:i+batch_size]
            batch_inputs = self.tokenizer(batch_docs, padding=True,␣
↪truncation=True,
                                        return_tensors="pt", max_length=512)

            with torch.no_grad():
                outputs = self.model(**batch_inputs)
                # Use CLS token embedding
                batch_embeddings = outputs.last_hidden_state[:, 0, :].cpu().
↪numpy()
                all_embeddings.append(batch_embeddings)

        # Concatenate all batch embeddings
        document_embeddings = np.vstack(all_embeddings)
        self.embedding_dim = document_embeddings.shape[1]

        # Normalize embeddings for cosine similarity
        faiss.normalize_L2(document_embeddings)

        # Build FAISS index for fast similarity search
        print(f"Building FAISS index with {document_embeddings.shape[1]}␣
↪dimensions...")
        self.index = faiss.IndexFlatIP(document_embeddings.shape[1])  # Inner␣
↪product for cosine similarity
        self.index.add(document_embeddings)

        # Save the index and metadata
        self.save_index()

        print("Dense index built successfully")
        return self

    def save_index(self):
        """Save the index and associated data to disk"""
        print(f"Saving index to {self.index_dir}...")

        # Save FAISS index
        if self.index is not None:
            faiss.write_index(self.index, os.path.join(self.index_dir, "faiss.
↪index"))

        # Save metadata
        metadata = {
            "model_name": self.model_name,
            "embedding_dim": self.embedding_dim,
            "num_documents": len(self.documents) if self.documents else 0
        }

        with open(os.path.join(self.index_dir, "metadata.json"), 'w') as f:
            json.dump(metadata, f, indent=2)

        # Save documents and doc_ids
        with open(os.path.join(self.index_dir, "documents.json"), 'w') as f:
            json.dump(self.documents, f)

        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 index and associated data"""
        index_path = os.path.join(self.index_dir, "faiss.index")
        if not os.path.exists(index_path):
            raise ValueError(f"Index not found at {index_path}. Build index␣
↪first with index_corpus()")

        print(f"Loading index from {self.index_dir}...")

        # Load FAISS index
        self.index = faiss.read_index(index_path)

        # Load metadata
        with open(os.path.join(self.index_dir, "metadata.json"), 'r') as f:
            metadata = json.load(f)
            self.model_name = metadata["model_name"]
            self.embedding_dim = metadata["embedding_dim"]

        # Load documents and doc_ids
        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 dense index with {len(self.documents)} documents")
        return self

    def retrieve(self, query, k=100):
        """
        Retrieve top-k documents for a query.

        Args:
            query: Query text
            k: Number of documents to retrieve

        Returns:
            List of dictionaries with document ID, score, and text
        """
        if self.index is None:
            self.load_index()

        # Generate query embedding
        query_embedding = self.get_embedding(query)
        query_embedding = query_embedding.reshape(1, -1)

        # Normalize query embedding for cosine similarity
        faiss.normalize_L2(query_embedding)

        # Search index
        scores, indices = self.index.search(query_embedding, k)

        # Format results
        results = []
        for i, idx in enumerate(indices[0]):
            if idx < len(self.documents):  # Safety check
                results.append({
                    "id": self.doc_ids[idx],
                    "score": float(scores[0][i]),
                    "text": self.documents[idx]
                })

        return results

    def batch_retrieve(self, queries, k=100):
        """
        Retrieve top-k documents for multiple queries.

        Args:
            queries: List of query texts
            k: Number of documents to retrieve per query

        Returns:
            Dictionary mapping query index to list of results
        """
        if self.index 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}")

# %%
# Usage example with your generated corpus
if __name__ == "__main__":
    # Load the generated corpus
    corpus_file = "legal_dummy_corpus.json"
    queries_file = "legal_sample_queries.json"

    # Check if corpus file exists
    if not os.path.exists(corpus_file):
        print("Corpus file not found. Please generate the corpus first.")
        exit(1)
    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. Please generate the queries first.")
        exit(1)
    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 Dense retriever with a legal-domain model
    # Using nlpaueb/legal-bert-base-uncased - specialized for legal text
    retriever = DenseRetriever(
        model_name="nlpaueb/legal-bert-base-uncased",
        index_name="legal_dense_retr2"
    )

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

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

    # Save results
    output_file = "dense_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)
```

```
/home/bandham/miniconda3/envs/llm692_venv/lib/python3.13/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
Loading corpus from legal_dummy_corpus.json…
Loading queries from legal_sample_queries.json…
```

```
Corpus has 120 documents
Query set has 15 queries
Building new dense index…
Building dense index with 120 documents…
Loading tokenizer for nlpaueb/legal-bert-base-uncased…
Loading model nlpaueb/legal-bert-base-uncased…
Generating document embeddings…

Processing document batches: 100%|     | 15/15 [00:39<00:00,  2.61s/it]

Building FAISS index with 768 dimensions…
Saving index to /home/bandham/Documents/StonyBrook_CourseWork/Spring 2025/LLM-
AMS692.02/Legal-Mind/legal_dense_retr2…
Dense index built successfully

Processing queries: 100%|     | 15/15 [00:36<00:00,  2.46s/it]

Saved retrieval results to dense_retrieval_results.json

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

Query: What are the essential elements of a valid contract?
--------------------------------------------------------------------------------
Document 1: (Score: 0.9432)
ID: criminal_law_030
Text: SEARCH WARRANT application states that probable cause exists to believe
evidence of possession with intent to distribute will be found at Northern
University Campus based on DNA analysis observed by O…
----------------------------------------
Document 2: (Score: 0.9333)
ID: administrative_law_018
Text: AGENCY DECISION: Securities and Exchange Commission hereby approves/denies
{party}'s application for operating license based on findings that the party
demonstrated financial responsibility. This deci…
----------------------------------------

Query: How is negligence defined in tort law?
--------------------------------------------------------------------------------
Document 1: (Score: 0.5427)
ID: contract_law_028
Text: EMPLOYMENT CONTRACT: Smith Corp. agrees to employ Commonwealth of
Jefferson as Chief Financial Officer commencing on September 9, 2022 for three
years. Compensation shall be $300,000 per annum with be…
----------------------------------------
Document 2: (Score: 0.4883)
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: What constitutes probable cause for a search warrant?
--------------------------------------------------------------------------------
Document 1: (Score: 0.5912)
ID: contract_law_028
Text: EMPLOYMENT CONTRACT: Smith Corp. agrees to employ Commonwealth of
Jefferson as Chief Financial Officer commencing on September 9, 2022 for three
years. Compensation shall be $300,000 per annum with be…
----------------------------------------
Document 2: (Score: 0.5856)
ID: criminal_law_013
Text: INDICTMENT: The Grand Jury charges that on July 8, 2022, defendant Richard
Taylor did knowingly and intentionally misrepresented material facts,
constituting the offense of money laundering under §9.2…
----------------------------------------
```

0.1 USAGE

[3]:
```python
# from dense_retriever import DenseRetriever  # Import your class

retriever = DenseRetriever(
        model_name="nlpaueb/legal-bert-base-uncased",
        index_name="legal_dense_retr2"
    )

retriever.load_index()

# 3. Search with any new query
results = retriever.retrieve("What are the essential elements of a valid␣
 ↪contract?", k=5)

# 4. Process the results
for i, result in enumerate(results):
    print(f"Result {i+1}: {result['text'][:100]}... (Score: {result['score']:.
 ↪4f})")
```

```
Loading index from /home/bandham/Documents/StonyBrook_CourseWork/Spring
2025/LLM-AMS692.02/Legal-Mind/legal_dense_retr2…
Loaded dense index with 120 documents
Loading tokenizer for nlpaueb/legal-bert-base-uncased…
Loading model nlpaueb/legal-bert-base-uncased…
Result 1: SEARCH WARRANT application states that probable cause exists to
believe evidence of possession with … (Score: 0.9432)
Result 2: AGENCY DECISION: Securities and Exchange Commission hereby
approves/denies {party}'s application for… (Score: 0.9333)
Result 3: PLEA AGREEMENT: Defendant Jennifer Lee, charged with criminal
```

```
negligence, agrees to plead guilty to … (Score: 0.9310)
Result 4: In In re Wilson Estate (2022), the Court held that religious freedom
protected under the Sixth Amend… (Score: 0.9307)
Result 5: CONSTITUTIONAL ANALYSIS: The Senate Bill 247 must be subjected to
heightened scrutiny under the Four… (Score: 0.9304)
```