ust

July 1, 2025

[62]:
```
!pip install scikit-learn transformers
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (1.6.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
packages (4.52.4)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.33.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-
packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (2025.3.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (4.14.0)
```

```
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from huggingface-
hub<1.0,>=0.30.0->transformers) (1.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->transformers)
(2025.6.15)
```

[63]:
```python
from typing import List
from dataclasses import dataclass, field
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import pipeline
```

[64]:
```python
@dataclass(eq=True, frozen=True)
class TreeNode:
    name: str
    children: List['TreeNode'] = field(default_factory=list, compare=False,␣
 ↪hash=False)
    documents: List[str] = field(default_factory=list, compare=False,␣
 ↪hash=False)

    def add_child(self, child: 'TreeNode') -> None:
        self.children.append(child)

    def add_document(self, doc: str) -> None:
        self.documents.append(doc)
```

[65]:
```python
class DocumentEncoder:
    def __init__(self):
        self.vectorizer = TfidfVectorizer()

    def fit_transform(self, docs: List[str]):
        return self.vectorizer.fit_transform(docs).toarray()

    def transform(self, doc: str):
        return self.vectorizer.transform([doc]).toarray()
```

[66]:
```python
class HierarchyTree:
    def __init__(self, root_name: str):
        self.root = TreeNode(name=root_name)
```

```
    def add_topic(self, path: List[str]) -> TreeNode:
        node = self.root
        for name in path:
            match = next((child for child in node.children if child.name ==␣
↪name), None)
            if not match:
                match = TreeNode(name=name)
                node.add_child(match)
            node = match
        return node

    def classify_and_assign(self, doc: str, topic_paths: List[List[str]]) ->␣
↪None:
        for path in topic_paths:
            node = self.add_topic(path)
            node.add_document(doc)

    def get_all_documents_under(self, node: TreeNode) -> List[str]:
        docs = list(node.documents)
        for child in node.children:
            docs.extend(self.get_all_documents_under(child))
        return docs
```

[67]:
```
class SearchModule:
    def __init__(self, tree: HierarchyTree):
        self.tree = tree

    def search(self, query: str, top_k: int = 3) -> List[TreeNode]:
        candidates = []
        node_doc_map = {}

        def collect_nodes(node: TreeNode):
            if node.documents:
                text = ' '.join(node.documents)
                node_doc_map[id(node)] = (node, text)
                candidates.append(node)
            for child in node.children:
                collect_nodes(child)

        collect_nodes(self.tree.root)

        corpus = [val[1] for val in node_doc_map.values()]
        if not corpus:
            return []

        encoder = DocumentEncoder()
        corpus_vectors = encoder.fit_transform(corpus)
```

```
        query_vector = encoder.transform(query)

        sims = cosine_similarity(query_vector, corpus_vectors).flatten()
        top_indices = sims.argsort()[::-1][:top_k]

        return [node_doc_map[list(node_doc_map.keys())[i]][0] for i in␣
 ↪top_indices]
```

[68]:
```
class RAGModel:
    def __init__(self, tree: HierarchyTree):
        self.tree = tree
        self.searcher = SearchModule(tree)
        self.generator = pipeline("text-generation", model="gpt2")

    def query(self, question: str) -> str:
        relevant_nodes = self.searcher.search(question)
        context_docs = []
        for node in relevant_nodes:
            context_docs.extend(self.tree.get_all_documents_under(node))

        context = " ".join(context_docs)[:1000]
        prompt = f"Context: {context}\nQuestion: {question}\nAnswer:"


        result = self.generator(prompt, max_length=150,␣
 ↪num_return_sequences=1)[0]['generated_text']
        return result.split("Answer:")[-1].strip()
```

[69]:
```
# Sample documents
documents = [
    "Neural networks and deep learning techniques",
    "Transformers in NLP applications",
    "Classical machine learning models like SVM",
    "Database indexing and SQL queries",
    "NoSQL systems and distributed databases"
]

# Build the tree and classify documents
tree = HierarchyTree("AI")
tree.classify_and_assign(documents[0], [["ML", "Deep Learning"]])
tree.classify_and_assign(documents[1], [["ML", "NLP"]])
tree.classify_and_assign(documents[2], [["ML", "Classical ML"]])
tree.classify_and_assign(documents[3], [["Databases", "SQL"]])
tree.classify_and_assign(documents[4], [["Databases", "NoSQL"]])

#  Create the model
rag = RAGModel(tree)
```

```
Device set to use cpu
```

[70]:
```python
#Ask a test question
question = "What are deep learning techniques?"
print("\nBot:", rag.query(question))
```

```
Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Both `max_new_tokens` (=256) and `max_length`(=150) seem to have been set.
`max_new_tokens` will take precedence. Please refer to the documentation for
more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)


Bot: Deep learning techniques are the tools that allow you to learn and learn
from a large number of topics.
Deep learning is the technology that allows you to use neural networks or
machine learning techniques to generate, process, and store information in real-
time.
Deep learning methods are the tools that allow you to learn and learn from large
number of topics.
Some deep learning techniques are:
SVDT (Simultaneous Neural Networks)
SVDT is a new approach to deep learning that allows you to make deep learning
decisions using the inputs and outputs of a deep learning model. This model can
be used to automatically classify, model, and predict a variety of data sources.
This technique uses deep learning data to teach and learn about the social,
cultural, and other dimensions of a person's behavior. It does not apply to non-
social networks like Facebook, LinkedIn, or Google+.
SVDT is a new approach to deep learning that allows you to make deep learning
decisions using the inputs and outputs of a deep learning model. This model can
be used to automatically classify, model, and predict a variety of data sources.
SVDT is a new approach to deep learning that allows you to make deep learning
decisions using the inputs and outputs of a deep learning
```

[70]:

[70]: