ChunkEmbeddings

1. Extracting text from pdf

```
# PDF Text Processing and Embedding Generation

This code snippet demonstrates how to extract text from a PDF, split it into small

## 1. PDF Text Extraction

```python
import fitz  # PyMuPDF

def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = "\n".join([page.get_text() for page in doc])
    return text

pdf_text = extract_text_from_pdf("/Users/vinod/Desktop/mike/sample.pdf")
```

- **Import `fitz`:** This line imports the PyMuPDF library, which is used for working with PDF files. Make sure you have it installed (`pip install pymupdf`).
- `extract_text_from_pdf(pdf_path)` **function:** This function takes the path to a PDF file as input.
- `fitz.open(pdf_path)`: Opens the PDF file.
- `"\n".join([page.get_text() for page in doc])`: Iterates through each page in the PDF, extracts the text content of the page using `page.get_text()`, and joins all the extracted text together into a single string, separated by newline characters (`\n`). This preserves paragraph breaks.
- `pdf_text = ...`: Calls the function to extract the text from your specified PDF file and stores the result in the `pdf_text` variable. **Remember to replace `/Users/vinod/Desktop/mike/sample.pdf` with the actual path to your PDF.**
```

2. Text Chunking

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

def split_text(text, chunk_size=500, chunk_overlap=50):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size, chunk_overlap=chunk_overlap
    )
    return text_splitter.split_text(text)

chunks = split_text(pdf_text)
```

- **Import `RecursiveCharacterTextSplitter`:** Imports the text splitter from the `langchain` library. Install it if you don't have it: `pip install langchain`.
- `split_text(text, chunk_size=500, chunk_overlap=50)` **function:** This function takes the extracted text as input, along with `chunk_size` (the desired size of each chunk) and `chunk_overlap` (how many characters should overlap between consecutive chunks). Overlapping helps to maintain context across chunks.
- `RecursiveCharacterTextSplitter(...)`: Creates an instance of the text splitter. This splitter attempts to break text into chunks while respecting sentence and paragraph boundaries.
- `text_splitter.split_text(text)`: Splits the input text into chunks and returns a list of strings (the chunks).
- `chunks = ...`: Calls the `split_text` function to split the `pdf_text` into chunks and stores the result in the `chunks` list.

3. Embedding Generation

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")
chunk_embeddings = model.encode(chunks)
```

```
print(f"Generated {len(chunk_embeddings)} embeddings!")
```

- **Import** `SentenceTransformer`: Imports the necessary class from the `sentence-transformers` library. Install it: `pip install sentence-transformers`.
- `model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")`: Loads a pre-trained Sentence Transformer model. `all-MiniLM-L6-v2` is a good balance between speed and performance. You can explore other models if needed. This line downloads the model if it's not already cached.
- `chunk_embeddings = model.encode(chunks)`: This is the core step. The `encode()` method takes the list of text chunks (`chunks`) and converts each chunk into a vector (embedding). These vectors capture the semantic meaning of the text.
- `print(...)`: Prints the number of embeddings generated. This should be equal to the number of text chunks.

Now you have `chunk_embeddings`, a NumPy array (or list of arrays, depending on the Sentence Transformer version) where each element is a vector representing the semantic meaning of a corresponding text chunk. You can then use these embeddings for various downstream tasks.