Tutorial: Analyzing Medical Transcripts with DocETL Python API

This tutorial will guide you through the process of using DocETL's Python API to analyze medical transcripts and extract medication information. We'll create a pipeline that identifies medications, resolves similar names, and generates summaries of side effects and therapeutic uses.

Prerequisites and Setup

For installation instructions, API key setup, and data preparation, please refer to the YAML-based tutorial. The prerequisite steps are identical for both the YAML and Python API approaches.

> **Common Setup**
>
> Both this Python API tutorial and the YAML-based tutorial share the same:
>
> - Installation requirements
> - API key configuration
> - Data format and preparation steps
>
> Once you've completed those steps from the main tutorial, you can continue with this Python API implementation.

Creating the Pipeline with Python API

Now, let's create a DocETL pipeline using the Python API to analyze this data. We'll use a series of operations to extract and process the medication information:

1. **Medication Extraction**: Analyze each transcript to identify and list all mentioned medications.
2. **Unnesting**: The extracted medication list is flattened, such that each medication (and associated data) is a separate document.
3. **Medication Resolution**: Similar medication names are resolved to standardize the entries. This step helps in consolidating different variations or brand names of the same medication.

4. **Summary Generation**: For each unique medication, generate a summary of side effects and therapeutic uses based on information from all relevant transcripts.

Create a Python script named `medical_analysis.py` with the following code:

```
from docetl.api import Pipeline, Dataset, MapOp, UnnestOp, ResolveOp,
ReduceOp, PipelineStep, PipelineOutput

# Define the dataset - JSON file with medical transcripts
dataset = Dataset(
    type="file",
    path="medical_transcripts.json"
)

# Define operations
operations = [
    # Extract medications from each transcript
    MapOp(
        name="extract_medications",
        type="map",
        prompt="""
        Analyze the following transcript of a conversation between a doctor
and a patient:
        {{ input.src }}
        Extract and list all medications mentioned in the transcript.
        If no medications are mentioned, return an empty list.
        """,
        output={
            "schema": {
                "medication": "list[str]"
            }
        }
    ),
    # Unnest to create separate items for each medication
    UnnestOp(
        name="unnest_medications",
        type="unnest",
        unnest_key="medication"
    ),
    # Resolve similar medication names
    ResolveOp(
        name="resolve_medications",
        type="resolve",
        blocking_keys=["medication"],
        blocking_threshold=0.6162,
        comparison_prompt="""
        Compare the following two medication entries:
        Entry 1: {{ input1.medication }}
        Entry 2: {{ input2.medication }}
        Are these medications likely to be the same or closely related?
        """,
        embedding_model="text-embedding-3-small",
        output={
            "schema": {
                "medication": "str"
            }
```

```
        },
        resolution_prompt="""
        Given the following matched medication entries:
        {% for entry in inputs %}
        Entry {{ loop.index }}: {{ entry.medication }}
        {% endfor %}
        Determine the best resolved medication name for this group of entries.
The resolved
        name should be a standardized, widely recognized medication name that
best represents
        all matched entries.
        """
    ),
    # Summarize side effects and uses for each medication
    ReduceOp(
        name="summarize_prescriptions",
        type="reduce",
        reduce_key=["medication"],
        prompt="""
        Here are some transcripts of conversations between a doctor and a
patient:

        {% for value in inputs %}
        Transcript {{ loop.index }}:
        {{ value.src }}
        {% endfor %}

        For the medication {{ reduce_key }}, please provide the following
information based on all the transcripts above:

        1. Side Effects: Summarize all mentioned side effects of {{ reduce_key
}}.
        2. Therapeutic Uses: Explain the medical conditions or symptoms for
which {{ reduce_key }} was prescribed or recommended.

        Ensure your summary:
        - Is based solely on information from the provided transcripts
        - Focuses only on {{ reduce_key }}, not other medications
        - Includes relevant details from all transcripts
        - Is clear and concise
        - Includes quotes from the transcripts
        """,
        output={
            "schema": {
                "side_effects": "str",
                "uses": "str"
            }
        }
    )
]

# Define pipeline step
step = PipelineStep(
    name="medical_info_extraction",
    input="transcripts",
    operations=[
        "extract_medications",
```

```
        "unnest_medications",
        "resolve_medications",
        "summarize_prescriptions"
    ]
)

# Define output
output = PipelineOutput(
    type="file",
    path="medication_summaries.json",
    intermediate_dir="intermediate_results"
)

# Define system prompt (optional but recommended)
system_prompt = {
    "dataset_description": "a collection of transcripts of doctor visits",
    "persona": "a medical practitioner analyzing patient symptoms and
reactions to medications"
}

# Create the pipeline
pipeline = Pipeline(
    name="medical_transcript_analysis",
    datasets={"transcripts": dataset},
    operations=operations,
    steps=[step],
    output=output,
    default_model="gpt-4o-mini",
    system_prompt=system_prompt
)

# Run the pipeline
cost = pipeline.run()
print(f"Pipeline execution completed. Total cost: ${cost:.2f}")
```

Running the Pipeline

To execute the pipeline, run the Python script:

```
python medical_analysis.py
```

This will process the medical transcripts, extract medication information, resolve similar medication names, and generate summaries of side effects and therapeutic uses for each medication. The results will be saved in `medication_summaries.json`.

Pipeline Performance

When running this pipeline on a sample dataset, we observed the following performance metrics using `gpt-4o-mini`:

- Total cost: $0.10
- Total execution time: 49.13 seconds

If you want to run it on a smaller sample, you can add a `sample` parameter to the `extract_medications` operation:

```
MapOp(
    name="extract_medications",
    type="map",
    sample=10,  # Process only 10 random transcripts
    # ... other parameters
)
```

Optimizing the Pipeline

If you want to optimize the pipeline configuration (such as finding the best blocking threshold for medication resolution):

```
# Create the pipeline
pipeline = Pipeline(
    # ... pipeline configuration as before
)

# Optimize the pipeline before running
optimized_pipeline = pipeline.optimize()

# Run the optimized pipeline
cost = optimized_pipeline.run()
print(f"Optimized pipeline execution completed. Total cost: ${cost:.2f}")
```

Further Questions

What if I want to focus on a specific type of medication or medical condition?

You can modify the prompts in the `extract_medications` and `summarize_prescriptions` operations to focus on specific types of medications or medical conditions. For example, you could update the `extract_medications` prompt to only list medications related to cardiovascular diseases.

How can I improve the accuracy of medication name resolution?

The `resolve_medications` operation uses a blocking threshold and comparison prompt to identify similar medication names. You can adjust the `blocking_threshold` parameter to control the sensitivity of the matching. Lower values will match more aggressively, while higher values require closer matches. You can also customize the comparison and resolution prompts.

Can I process other types of medical documents with this pipeline?

Yes, you can adapt this pipeline to process other types of medical documents by modifying the input data format and adjusting the prompts in each operation. For example, you could use it to analyze discharge summaries, clinical notes, or research papers by updating the extraction and summarization prompts accordingly.

How can I use the pandas integration?

DocETL provides a pandas integration for several operators (map, filter, merge, agg, split, gather, unnest). Here's an example of how to use it with the medication analysis:

```
import pandas as pd
from docetl import SemanticAccessor

# Load data as DataFrame
df = pd.read_json("medical_transcripts.json")

# Use semantic map operation to extract medications
result_df = df.semantic.map(
    prompt="""
    Analyze the following transcript:
    {{ input.src }}
    Extract all medications mentioned.
    """,
    output_schema={"medications": "list[str]"}
)

# Continue processing with other pandas operations
```

Learn more about the pandas integration in the pandas documentation.