SemLink Code Documentation

Contents

1 data_loader

1.1 DataLoader Objects

Class: `DataLoader()`
Handles loading and preprocessing of CSV/TSV files, with optional JSON metadata. It generates a standardized data lake representation (list of dictionaries) including basic stats and initial LLM-based descriptions.

`__init__`

```
def __init__(openai_client: OpenAI = None)
```

Initializes DataLoader with optional OpenAI client. If OpenAI client is provided, it also loads the tiktoken tokenizer for generating LLM-based descriptions.

- **Arguments:**
  - `openai_client` *OpenAI* - An initialized OpenAI client instance. If no OpenAI client is provided, it prints a warning message and disables generation of LLM descriptions.

`load_and_describe_datalake`

```
def load_and_describe_datalake(
    data_directory: str,
    metadata_directory: str | None = None,
    llm: bool = True,
    sample_size: int = 100,
    max_prompt_tokens: int = 8000,
    output_directory: str | None = None) -> list[dict]
```

Loads data from CSV/TSV files in the provided data directory and creates a data lake list with descriptions generated using a Large Language Model (LLM).

- **Arguments:**
  - `data_directory` *str* - Path to the directory containing CSV/TSV files.
  - `metadata_directory` *str — None* - Path to the directory containing JSON metadata files matching the data files (optional).
  - `llm` *bool* - Whether to use an LLM to generate descriptions for files and columns.
  - `sample_size` *int* - Number of distinct values to sample from each column for statistics.
  - `max_prompt_tokens` *int* - Maximum number of tokens to use in the LLM prompt.
  - `output_directory` *str — None* - Directory to save the generated data lake JSON file to.
- **Returns:**
  - `list[dict]` - A list of dictionaries, each representing a data file with its columns, containing statistical information and descriptions generated by the LLM.

2 schema_generator

2.1 SchemaGenerator Objects

Class: `SchemaGenerator()`
Generates a LinkML schema from a processed data lake representation. It collects column data, unifies descriptions for common columns, and then generates the LinkML YAML schema using an LLM.

`__init__`

```
def __init__(openai_client: OpenAI)
```

Initializes the SchemaGenerator with an OpenAI client.

- **Arguments:**
  - `openai_client` *OpenAI* - An initialized OpenAI client instance.

`generate_linkml_schema`

```
def generate_linkml_schema(data_lake_list: Union[list[dict], str],
                           output_directory: str | None = None) -> dict
```

Generates a LinkML schema from a processed data lake representation. This method collects column data, unifies descriptions for common columns, and then generates the LinkML YAML schema using an LLM.

- **Arguments:**
  - `data_lake_list` *list[dict] — str* - The list of dictionaries representing the data lake, as generated by DataLoader.load_and_describe_datalake.
  - `output_directory` *str — None* - If specified, the generated LinkML schema will be saved to a file in this directory, named "linkml_schema.yaml".
- **Returns:**
  - `dict | None` - The generated LinkML schema in dictionary format, or None if an error occurs.

`prune_schema`

```
def prune_schema(data_lake_list: Union[list[dict], str],
                 yaml_schema: Union[str, dict],
                 output_directory: str | None = None) -> str | None
```

Prunes the given LinkML schema (`yaml_schema`) to include only the classes that are relevant to the datasets in the data lake (`data_lake_list`).

- **Arguments:**
  - `data_lake_list` *list[dict] or str* - The list of dictionaries representing the data lake, or a path to a JSON file containing the data lake information.
  - `yaml_schema` *str or dict* - The LinkML schema as a dictionary, or a path to a YAML file containing the LinkML schema.
  - `output_directory` *str, optional* - The directory where the pruned schema should be saved. If not provided, the pruned schema is returned as a dictionary.
- **Returns:**
  - `str or None` - The path to the pruned schema YAML file if `output_directory` is provided, otherwise the pruned schema as a dictionary.

3 semantic_annotation

`archetype_annotation`

```
def archetype_annotation(data_lake_list: Union[list[dict], str],
                         yaml_schema: Union[str, dict],
                         sample_size: int = 20,
                         output_directory: str | None = None) -> list[dict]
```

Annotates the given data lake representation with semantic annotations using the Archetype library.

- **Arguments:**
  - `data_lake_list` *list[dict] — str* - The list of dictionaries representing the data lake, as generated by DataLoader.load_and_describe_datalake.
  - `yaml_schema` *str — dict* - The LinkML schema as a dictionary, or a path to a YAML file containing the LinkML schema.
  - `sample_size` *int* - The number of rows to sample from each file for annotation.
  - `output_directory` *str — None* - If specified, the annotated data lake JSON will be saved to a file in this directory, named "data_lake_annotated.json".
- **Returns:**
  - `list[dict]` - The annotated data lake representation.

4 join_discoverer

4.1 JoinDiscoverer Objects

Class: `JoinDiscoverer()`
This class provides a streamlined workflow to:

1. Generate embeddings for a data lake's columns using an LLM.
2. Calculate various distance metrics between these embeddings.
3. Use the distances to generate a Neo4j-compatible CSV for nodes and relationships.

`__init__`

```
def __init__(openai_client: OpenAI)
```

Initializes the DataJoinerPipeline.

- **Arguments:**
  - `openai_client` *OpenAI* - An initialized OpenAI client instance.

`generate_embeddings`

```
def generate_embeddings(data_lake_list: Union[list[dict], str],
                        prompt_mode: str = 'semantic_mode',
                        embedding_model: str = 'text-embedding-3-small',
                        output_directory: str | None = None) -> list[dict]
```

Generates a list of embeddings for each column in the data lake representation.

- **Arguments:**
  - `data_lake_list` *list[dict] — str* - The list of dictionaries representing the data lake, as generated by DataLoader.load_and_describe_datalake.
  - `prompt_mode` *str* - One of 'semantic_mode' or 'data_profiling_mode'.
  - `embedding_model` *str* - The name of the OpenAI embedding model to use.
  - `output_directory` *str — None* - If specified, the generated embeddings will be saved to a file in this directory, named "embeddings.json".
- **Returns:**
  - `list[dict]` - A list of dictionaries, each containing the embedding for a column and its file name, as well as the semantic annotation for the column (if available).

`compute_distances_and_export_neo4j`

```
def compute_distances_and_export_neo4j(
    embeddings: Union[list[dict], str],
    cosine_sim_threshold: float = 0.5,
    anns_threshold: float = 0.2,
    output_directory: str | None = None
) -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]
```

Computes cosine similarity, euclidean distance, and ANNS distances between all pairs of column embeddings, and exports the results as CSV files for Neo4j nodes and edges.

- **Arguments:**

  - `embeddings` - A list of dictionaries, each containing the embedding for a column and its file name, as well as the semantic annotation for the column (if available). Alternatively, a string path to a JSON file containing the embeddings.
  - `cosine_sim_threshold` *float* - The minimum cosine similarity required for a pair of columns to be considered joinable.
  - `anns_threshold` *float* - The maximum ANNS distance required for a pair of columns to be considered joinable.
  - `output_directory` *str* - The directory where the output CSV files will be saved.

- **Returns:**
  - `tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]` - The computed distances DataFrame, the nodes DataFrame, and the edges DataFrame.

5 utils

`color_text`

```
def color_text(text: str, color: str = 'white') -> str
```

Colors text using colorama colors.

- **Arguments:**
  - `text` *str* - Text to be colored.
  - `color` *str* - Color name to use, defaults to 'white'. Valid colors: 'black', 'red', 'green', 'yellow', 'blue', 'magenta', 'cyan', 'white'.
- **Returns:**
  - `str` - Colored text string with reset style appended.

`add_timestamp`

```
def add_timestamp(text: str) -> str
```

Adds a timestamp before the provided text.

- **Arguments:**
  - `text` *str* - Text to prepend timestamp to.
- **Returns:**
  - `str` - Text with timestamp prepended in format [HH:MM:SS].

`create_directory_if_not_exists`

```
def create_directory_if_not_exists(path: str)
```

Creates a directory if it does not already exist.

- **Arguments:**
  - `path` *str* - The path to the directory to create.

`load_datalake_json`

```
def load_datalake_json(json_file_path: str) -> list[dict]
```

Loads a standardized data lake list of dictionaries from a JSON file.

- **Arguments:**
  - `json_file_path` *str* - The full path to the data lake JSON file.
- **Returns:**
  - `list[dict]` - The loaded list of dictionaries representing the data lake.

`yaml_parser`

```
def yaml_parser(data: dict) -> tuple[list[str], dict[str, str]]
```

Parses a YAML schema file to extract classes, attributes, and their descriptions. It handles inheritance for attributes with special handling for root classes.

- **Arguments:**
  - `data` *dict* - The loaded YAML schema data.
- **Returns:**
  - `tuple[list[str], dict[str, str]]`
    - A list of strings in the format "ClassName.attributeName" for all classes and their attributes, including inherited ones.
    - A dictionary mapping "ClassName" or "ClassName.attributeName" to their descriptions.