Contents

1 Data Loaders Refactoring

1.1 Overview

The data loading logic has been moved from `data_loaders.py` into the extraction mode class hierarchy. This properly separates concerns by extraction mode and eliminates code duplication.

1.2 Changes Made

1.2.1 Before

data_loaders.py (~382 lines): - `AnnotatedTextLoader` class with complex line/paragraph logic - `RawTextLoader` class with complex line/paragraph logic - Duplicated code between line and paragraph modes - if/else branches for line_level parameter

1.2.2 After

extraction_modes/ directory structure:

```
extraction_modes/
├── __init__.py          # Exports and factory functions
├── base.py              # AnnotatedTextParser ABC
├── mode.py              # ExtractionMode ABC
├── line.py              # LineExtractionMode with data loading
├── paragraph.py         # ParagraphExtractionMode with data loading
├── section.py           # SectionExtractionMode with data loading
├── line_mode.py         # LineAnnotatedTextParser
├── paragraph_mode.py    # ParagraphAnnotatedTextParser
└── section_mode.py      # SectionAnnotatedTextParser
```

data_loaders.py (~178 lines): - Thin wrapper classes for backwards compatibility - Delegates to appropriate extraction mode

1.3 ExtractionMode Methods

Each `ExtractionMode` subclass now implements:

1.3.1 Raw Data Loading

- `load_raw_from_files(spark, file_paths)` - Load unannotated text from files
- `load_raw_from_couchdb(spark, couchdb_url, database, username, password, pattern)` - Load from CouchDB

1.3.2 Annotated Data Loading

- `load_annotated_from_files(spark, file_paths, collapse_labels)` - Load annotated text from files
- `load_annotated_from_couchdb(spark, couchdb_url, database, username, password, pattern, collapse_labels)` - Load from CouchDB

1.4 Implementation Details

1.4.1 LineExtractionMode (line.py)

Implements line-level extraction: - Splits content into individual lines - Adds `line_number` column for ordering - Preserves line order within files/documents

1.4.2 ParagraphExtractionMode (paragraph.py)

Implements paragraph-level extraction: - Uses `ParagraphExtractor.extract_heuristic_parag` for raw text - Uses `ParagraphExtractor.extract_annotated_paragraphs()` for annotations - Adds `row_number` for ordering

1.4.3 SectionExtractionMode (section.py)

Implements section-level extraction: - **Files**: Raises `NotImplementedError` (requires CouchDB) - **CouchDB**: Uses `PDFSectionExtractor` for PDF processing - Returns sections with metadata (page_number, line_number, section_name)

1.5 Backwards Compatibility

The old API still works:

```
# Old code - still works
from skol_classifier.data_loaders import AnnotatedTextLoader, RawTextLoader

loader = AnnotatedTextLoader(spark)
df = loader.load_from_files(file_paths, line_level=True)

raw_loader = RawTextLoader(spark)
df = raw_loader.load_from_couchdb(
    couchdb_url, database, username, password,
    pattern="*.txt", line_level=False
)
```

1.6 New Object-Oriented API

Direct usage of extraction modes:

```
# New code - preferred
from skol_classifier.extraction_modes import get_mode

mode = get_mode('line')
df = mode.load_annotated_from_files(
    spark=spark,
    file_paths=file_paths,
    collapse_labels=True
)

df = mode.load_raw_from_couchdb(
    spark=spark,
    couchdb_url=couchdb_url,
    database=database,
    username=username,
    password=password,
    pattern="*.txt"
)
```

1.7 Benefits

1. **Separation of Concerns**: Each mode handles its own loading logic
2. **Reduced Duplication**: No more if/else branches for line_level
3. **Extensibility**: Easy to add new extraction modes
4. **Type Safety**: Each mode has strongly-typed methods
5. **Testability**: Can test each mode independently
6. **Backwards Compatibility**: Old code continues to work

1.8 Files Modified

1.8.1 Created:

- `skol_classifier/extraction_modes/line.py` - Line mode with data loading
- `skol_classifier/extraction_modes/paragraph.py` - Paragraph mode with data loading
- `skol_classifier/extraction_modes/section.py` - Section mode with data loading
- `skol_classifier/extraction_modes/mode.py` - Updated with abstract data loading methods

1.8.2 Modified:

- `skol_classifier/data_loaders.py` - Converted to thin wrapper (382 → 178 lines)
- `skol_classifier/extraction_modes/__init__.py` - Updated imports

1.9 Code Reduction

- **Before**: ~382 lines of complex loading logic in data_loaders.py
- **After**: ~178 lines of delegation in data_loaders.py
- **Savings**: ~200 lines removed from data_loaders.py
- **Added**: ~600 lines in extraction_modes/ (better organized)

The code is now more maintainable because: - Each mode's logic is isolated in its own file - No complex conditional logic - Clear separation between line/paragraph/section behavior

1.10 Testing

All existing tests continue to pass because the wrapper classes maintain the old API:

```
# Old tests still work
python -m pytest tests/test_data_loaders.py
```

New tests can directly use extraction modes:

```
from skol_classifier.extraction_modes import LineExtractionMode

def test_line_mode_loading():
    mode = LineExtractionMode()
    df = mode.load_raw_from_files(spark, file_paths)
    assert 'line_number' in df.columns
```

1.11 Migration Path

1. **Phase 1** (Complete): Move logic to extraction modes, maintain wrappers
2. **Phase 2** (Future): Update classifier_v2.py to use extraction modes directly
3. **Phase 3** (Future): Deprecate data_loaders.py wrapper classes
4. **Phase 4** (Future): Remove data_loaders.py entirely