Contents

1 Extraction Mode Object Hierarchy

This document describes the object-oriented hierarchy for extraction modes in the SKOL classifier.

1.1 Overview

The extraction mode system has been refactored from string-based if statements to an object-oriented hierarchy with two parallel class hierarchies:

1. **AnnotatedTextParser** - for parsing YEDDA annotations
2. **ExtractionMode** - for mode-specific behavior

1.2 Class Hierarchies

1.2.1 AnnotatedTextParser Hierarchy

Located in `skol_classifier/extraction_modes/`

```
AnnotatedTextParser (ABC)
├── LineAnnotatedTextParser
```

```
├── ParagraphAnnotatedTextParser
└── SectionAnnotatedTextParser
```

Purpose: Parse YEDDA-annotated text into structured DataFrames

Key methods: - `parse(df: DataFrame) -> DataFrame` - Parse annotations from input DataFrame - `extraction_mode` property - Returns 'line', 'paragraph', or 'section' - `line_level` property - Backwards compatibility for line mode detection

Files: - `base.py` - Abstract base class with `_get_section_name()` static method - `line_mode.py` - Line-level parser (includes line_number in output) - `paragraph_mode.py` - Paragraph-level parser - `section_mode.py` - Section-level parser (handles both pre-segmented and plain text)

1.2.2 ExtractionMode Hierarchy

Located in `skol_classifier/extraction_modes/mode.py`

```
ExtractionMode (ABC)
├── LineExtractionMode
├── ParagraphExtractionMode
└── SectionExtractionMode
```

Purpose: Encapsulate mode-specific loading and behavior logic

Key methods: - `load_from_files(classifier) -> DataFrame` - Load from local files - `load_from_couchdb(classifier) -> DataFrame` - Load from CouchDB - `name` property - Returns 'line', 'paragraph', or 'section' - `is_line_mode` property - Returns True for line mode - `__eq__(other)` - Allows comparison with strings for backwards compatibility

Behavior differences by mode:

Method	Line/Paragraph	Section
`load_from_files()`	Uses RawTextLoader	Raises NotImplementedError
`load_from_couchdb()`	Loads .txt.ann attachments	Uses PDFSectionExtractor

1.3 Factory Functions

1.3.1 get_parser()

```python
from skol_classifier.extraction_modes import get_parser
```

```
parser = get_parser(mode='section', collapse_labels=True)
result_df = parser.parse(df)
```

Creates the appropriate AnnotatedTextParser subclass.

1.3.2 get_mode()

```
from skol_classifier.extraction_modes import get_mode

mode = get_mode('section')
df = mode.load_from_couchdb(classifier)
```

Creates the appropriate ExtractionMode subclass.

1.4 Backwards Compatibility

1.4.1 AnnotatedTextParser Factory Function

In `preprocessing.py`, the original `AnnotatedTextParser` class has been replaced with a factory function that maintains backwards compatibility:

```
def AnnotatedTextParser(
    extraction_mode: str = 'paragraph',
    collapse_labels: bool = True,
    line_level: Optional[bool] = None
):
    """Factory function for creating AnnotatedTextParser instances."""
    from .extraction_modes import get_parser

    # Handle deprecated line_level parameter
    if line_level is not None:
        import warnings
        warnings.warn(
            "line_level parameter is deprecated, use extraction_mode='line'",
            DeprecationWarning,
            stacklevel=2
        )
        extraction_mode = 'line' if line_level else 'paragraph'

    return get_parser(mode=extraction_mode, collapse_labels=collapse_labels)
```

Existing code continues to work:

```
# Old API still works
parser = AnnotatedTextParser(extraction_mode='section')
parser = AnnotatedTextParser(line_level=True)  # Deprecated but functional
```

1.4.2 String Comparison

ExtractionMode instances can be compared with strings:

```
mode = get_mode('section')
if mode == 'section':  # This works due to __eq__ override
    print("Section mode detected")
```

1.5 Usage Pattern

1.5.1 Before (String-based)

```
class TaxaClassifier:
    def __init__(self, extraction_mode: str = 'paragraph'):
        self.extraction_mode = extraction_mode

    def _load_raw_from_files(self):
        if self.extraction_mode == 'section':
            return self._load_sections_from_files()
        else:
            # Load traditional way
            loader = RawTextLoader(self.spark)
            df = loader.load_files(
                self.file_paths,
                line_level=(self.extraction_mode == 'line')
            )
            return self.load_raw_from_df(df)
```

1.5.2 After (Object-oriented)

```
from skol_classifier.extraction_modes import get_mode

class TaxaClassifier:
    def __init__(self, extraction_mode: str = 'paragraph'):
        self.extraction_mode = get_mode(extraction_mode)

    def _load_raw_from_files(self):
        # Mode object handles the dispatch
        return self.extraction_mode.load_from_files(self)
```

1.6 Benefits

1. **Single Responsibility**: Each mode class handles its own behavior
2. **Extensibility**: Easy to add new extraction modes
3. **Type Safety**: Better IDE support and type checking

4. **Testability**: Each mode can be tested independently
5. **Backwards Compatibility**: Existing code continues to work

1.7 Migration Path

The system is designed for incremental migration:

1. **Phase 1** (Complete): Create class hierarchies, maintain backwards compatibility
2. **Phase 2** (In Progress): Replace if statements in `classifier_v2.py` with mode object methods
3. **Phase 3** (Future): Update all code to use object-oriented API
4. **Phase 4** (Future): Deprecate and eventually remove string-based API

1.8 Files Modified/Created

1.8.1 Created:

- `skol_classifier/extraction_modes/__init__.py` - Module exports and factories
- `skol_classifier/extraction_modes/base.py` - AnnotatedTextParser ABC
- `skol_classifier/extraction_modes/line_mode.py` - LineAnnotatedTextParser
- `skol_classifier/extraction_modes/paragraph_mode.py` - ParagraphAnnotatedTextParser
- `skol_classifier/extraction_modes/section_mode.py` - SectionAnnotatedTextParser
- `skol_classifier/extraction_modes/mode.py` - ExtractionMode hierarchy

1.8.2 Modified:

- `skol_classifier/preprocessing.py` - Replaced AnnotatedTextParser class with factory function

1.9 Next Steps

To complete the migration, replace remaining if statements in `classifier_v2.py`:

```
# Find and replace patterns like:
if self.extraction_mode == 'section':
    ...
else:
```

```
    ...

# With:
self.extraction_mode.method_name(self)
```

Key locations to update: - `_load_raw_from_files()` at line 774 - `_load_raw_from_couchdb()` at line 791 - Section name TF-IDF logic at line 379