Contents

1 SkolClassifierV2 Tokenizer Parameter Update

1.1 Overview

Updated `SkolClassifierV2` to replace the boolean `line_level` parameter with a more flexible `extraction_mode` parameter that supports three text processing modes.

Date: 2025-12-22 **Version**: 2.1 **Breaking Changes**: Parameter rename (backwards compatible via property)

1.2 Changes Made

1.2.1 1. Parameter Replacement

Old Parameter:

```
line_level: bool = False  # True = line-by-line, False = paragraph
```

New Parameter:

```
extraction_mode: Literal['line', 'paragraph', 'section'] = 'paragraph'
```

1.2.2 2. Tokenizer Modes

Mode	Description	Equivalent to Old	Use Case
`'line'`	Process text line-by-line	`line_level=True`	Short text, precise alignment
`'paragraph'`	Process text by paragraphs	`line_level=False`	Natural language processing
`'section'`	Use PDF section extraction with section name features	*New*	Scientific PDFs with structure

1.2.3 3. Section Mode Features

When `extraction_mode='section'`: - Uses `PDFSectionExtractor` to parse PDF documents - Automatically extracts section names (Introduction, Methods, Results, etc.) - Enables section name TF-IDF features in `FeatureExtractor` - Requires `input_source='couchdb'` (PDFs must be in CouchDB) - DataFrame includes `section_name` column with rich metadata

1.2.4 4. Feature Extraction Integration

The FeatureExtractor now automatically enables section name features:

```
# When extraction_mode='section'
use_section_names = (self.extraction_mode == 'section')

feature_extractor = FeatureExtractor(
    use_suffixes=self.use_suffixes,
    use_section_names=use_section_names,  # Auto-enabled for section mode
    min_doc_freq=self.min_doc_freq,
    word_vocab_size=self.word_vocab_size,
    suffix_vocab_size=self.suffix_vocab_size
)
```

1.3 Usage Examples

1.3.1 Line Mode (Old: line_level=True)

```
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/train/*.txt.ann'],
    extraction_mode='line',  # Process line-by-line
    use_suffixes=True,
    model_type='logistic'
)
```

1.3.2 Paragraph Mode (Old: line_level=False)

```
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/train/*.txt.ann'],
    extraction_mode='paragraph',  # Default, process by paragraphs
    use_suffixes=True,
    model_type='logistic'
)
```

1.3.3 Section Mode (New)

1.3.3.1 Auto-Discovery (Recommended)

```
# Automatically discovers all PDF documents in database
classifier = SkolClassifierV2(
    input_source='couchdb',  # Required for section mode
    couchdb_url='http://localhost:5984',
    couchdb_database='taxonomic_articles',
```

```
    couchdb_username='admin',
    couchdb_password='password',
    # No couchdb_doc_ids needed - auto-discovers PDFs!
    extraction_mode='section',  # Use PDF section extraction
    use_suffixes=True,
    model_type='logistic'
)

# Extract sections from all PDFs in database
raw_df = classifier.load_raw()
# Output:
# [Classifier] Auto-discovering PDF documents in database: taxonomic_articles
# [Classifier] Found 15 documents with PDF attachments
# [Classifier] Database: taxonomic_articles
```

1.3.3.2 Explicit Document IDs (Optional)

```
# Or specify exact documents to process
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='taxonomic_articles',
    couchdb_username='admin',
    couchdb_password='password',
    couchdb_doc_ids=['doc1', 'doc2', 'doc3'],  # Process only these
    extraction_mode='section',
    use_suffixes=True,
    model_type='logistic'
)

# raw_df will have columns:
# - value: Section text
# - doc_id: Document ID
# - attachment_name: PDF filename
# - paragraph_number: Section number
# - line_number: Line number in PDF
# - page_number: PDF page number
# - empirical_page_number: Document page number
# - section_name: Standardized section name (e.g., "Introduction", "Methods")

# Train with section name features
classifier.fit(raw_df)
```

1.3.4 Section Mode with Filtering (New)

```
# Only process specific sections from PDFs
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='taxonomic_articles',
    couchdb_username='admin',
    couchdb_password='password',
    couchdb_doc_ids=['doc1', 'doc2', 'doc3'],
    extraction_mode='section',
    section_filter=['Introduction', 'Methods', 'Results'],  # Only these section
    use_suffixes=True,
    model_type='logistic'
)

# Only sections matching the filter will be loaded
raw_df = classifier.load_raw()
```

1.4 Backwards Compatibility

1.4.1 Property Access

A backwards-compatible `line_level` property is provided:

```
classifier = SkolClassifierV2(extraction_mode='line')
print(classifier.line_level)  # True

classifier = SkolClassifierV2(extraction_mode='paragraph')
print(classifier.line_level)  # False

classifier = SkolClassifierV2(extraction_mode='section')
print(classifier.line_level)  # False
```

1.4.2 Migration Guide

Old Code:

```
# Old API
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/*.ann'],
    line_level=True,  # Old parameter
    use_suffixes=True
)
```

New Code:

```python
# New API
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/*.ann'],
    extraction_mode='line',  # New parameter
    use_suffixes=True
)
```

1.5 Model Metadata Changes

1.5.1 Version Update

Model metadata version updated from `2.0` to `2.1`:

```json
{
  "label_mapping": {...},
  "config": {
    "extraction_mode": "section",
    "use_suffixes": true,
    "min_doc_freq": 2,
    "model_type": "logistic",
    "model_params": {...}
  },
  "version": "2.1"
}
```

Old Version (2.0):

```json
{
  "config": {
    "line_level": true,
    ...
  },
  "version": "2.0"
}
```

1.6 Implementation Details

1.6.1 Files Modified

classifier_v2.py: - Line 88: Added `couchdb_doc_ids` to docstring - Line 104-107: Updated docstring for `extraction_mode` parameter - Line 108-109: Added `section_filter` to docstring - Line 168: Added `couchdb_doc_ids` parameter - Line 186: Added `section_filter` parameter - Line 185: Changed `line_level: bool` to `extraction_mode: Literal['line', 'paragraph', 'section']` - Line 216: Store `self.couchdb_doc_ids` - Line 234:

Store `self.extraction_mode` instead of `self.line_level` - Line 235: Store `self.section_filter` - Line 266-269: Added `line_level` property for backwards compatibility - Line 361: Enable section name features when `extraction_mode='section'` - Line 747-762: Updated `_load_raw_from_files()` to handle section mode - Line 764-780: Updated `_load_raw_from_couchdb()` to handle section mode - Line 887-899: Added `_load_sections_from_files()` (raises NotImplementedError) - Line 903-972: Added `_load_sections_from_couchdb()` (full implementation) - Uses PDFSectionExtractor with document IDs - Applies section filtering - Shows detailed progress and statistics - Line 1173, 1331: Updated metadata to use `extraction_mode` and version `2.1`

1.6.2 New Parameters

1. **`couchdb_doc_ids`**: List of document IDs for section mode (line 168)
2. **`section_filter`**: Optional list of section names to include (line 186)

1.6.3 New Methods

1. **`_load_sections_from_files()`**: Raises error (section mode requires CouchDB)
2. **`_load_sections_from_couchdb()`**: Full implementation with PDFSectionExtractor
   - Extracts sections from multiple documents
   - Applies section filtering
   - Shows section distribution with verbosity
3. **`line_level` property**: Backwards compatibility helper

1.7 Limitations

1.7.1 Section Mode Constraints

1. **CouchDB Required**: Section mode only works with `input_source='couchdb'`
2. **PDF Attachments**: Documents must have PDF attachments in CouchDB
3. **Section Names**: Only sections with recognized names can be filtered
4. **Auto-Discovery Performance**: Queries all documents in database (may be slow for large databases)

1.7.2 Error Messages

```
# Using section mode with files
classifier = SkolClassifierV2(
    input_source='files',
    extraction_mode='section'  # ERROR
)
# Raises: NotImplementedError - section mode requires CouchDB

# Using section mode when no PDFs found
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='empty_db',
    extraction_mode='section'
)
classifier.load_raw()
# Raises: ValueError - No PDF documents found in database
```

1.8 Benefits

1.8.1 1. Clearer API

- More descriptive parameter name (`extraction_mode` vs `line_level`)
- Supports more than two modes
- Self-documenting code

1.8.2 2. Section-Based Classification

- Leverage document structure for better classification
- Section name features improve model accuracy
- Ideal for scientific/academic documents

1.8.3 3. Feature Extraction Integration

- Automatic section name TF-IDF when appropriate
- Combines word, suffix, and section features
- Flexible vocabulary sizes for each feature type

1.8.4 4. Extensible Design

- Easy to add new tokenization modes (e.g., ‘sentence’, ‘semantic’)
- Mode-specific data loading logic
- Decoupled from feature extraction

1.9 Future Enhancements

1.9.1 Completed ✅

1. ✅ **Section Loading**: Full implementation of `_load_sections_from_couchdb()` with document IDs
2. ✅ **Section Filtering**: Filter by specific section names via `section_filter` parameter
3. ✅ **Automatic Document Discovery**: Auto-discovers all PDFs in database when `couchdb_doc_ids` not provided

1.9.2 Planned Improvements

1. **Local PDF Support**: Add `_load_sections_from_files()` for local PDFs
2. **Optimized Discovery**: Use CouchDB views/queries for faster PDF document discovery
3. **Hybrid Modes**: Support combinations like `extraction_mode='section+line'`
4. **Custom Section Patterns**: User-defined section recognition patterns
5. **Section Annotations**: Support for annotated sections in training data

1.9.3 Example Future API

```
# Optimized discovery with CouchDB views (planned)
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='mydb',
    extraction_mode='section',
    use_views=True,  # Use _all_docs with attachment filtering for speed
    section_filter=['Introduction', 'Methods'],
    ...
)

# Custom section patterns (planned)
classifier = SkolClassifierV2(
    input_source='couchdb',
    extraction_mode='section',
    section_patterns={'custom_section': r'^Custom\s+Section'},  # Custom pattern
    ...
)
```

1.10 Testing

1.10.1 Test Coverage

- ✅ Parameter replacement and storage
- ✅ Backwards compatibility property
- ✅ FeatureExtractor integration for section mode
- ✅ Model metadata save/load with new version
- ✅ Full section loading from CouchDB
- ✅ Section filtering
- ✅ Automatic PDF document discovery
- □ End-to-end classification with section mode (requires real data)

1.10.2 Manual Testing

```
from pyspark.sql import SparkSession
from skol_classifier.classifier_v2 import SkolClassifierV2

spark = SparkSession.builder.appName("Test").getOrCreate()

# Test 1: Line mode
clf_line = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=['test.ann'],
    extraction_mode='line'
)
assert clf_line.tokenizer == 'line'
assert clf_line.line_level == True

# Test 2: Paragraph mode
clf_para = SkolClassifierV2(
    spark=spark,
    input_source='files',
    file_paths=['test.ann'],
    extraction_mode='paragraph'
)
assert clf_para.tokenizer == 'paragraph'
assert clf_para.line_level == False

# Test 3: Section mode with files (should fail)
try:
    clf_section = SkolClassifierV2(
        spark=spark,
```

```
        input_source='files',
        extraction_mode='section'
    )
    clf_section.load_raw()
except NotImplementedError as e:
    print(f"Expected error: {e}")

# Test 4: Section mode with CouchDB (requires document IDs)
try:
    clf_section = SkolClassifierV2(
        spark=spark,
        input_source='couchdb',
        couchdb_url='http://localhost:5984',
        couchdb_database='mydb',
        extraction_mode='section'
        # Missing: couchdb_doc_ids
    )
    clf_section.load_raw()
except ValueError as e:
    print(f"Expected error: {e}")

# Test 5: Section mode with explicit doc IDs (requires real CouchDB)
clf_section = SkolClassifierV2(
    spark=spark,
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='skol_dev',
    couchdb_doc_ids=['doc1', 'doc2', 'doc3'],
    extraction_mode='section',
    section_filter=['Introduction', 'Methods'],
    verbosity=2
)
# sections_df = clf_section.load_raw()
# print(f"Loaded sections: {sections_df.count()}")

# Test 6: Section mode with auto-discovery (requires real CouchDB)
clf_auto = SkolClassifierV2(
    spark=spark,
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='skol_dev',
    # No couchdb_doc_ids - auto-discovers all PDFs
    extraction_mode='section',
    verbosity=2
)
# sections_df = clf_auto.load_raw()
```

```
# Output:
# [Classifier] Auto-discovering PDF documents in database: skol_dev
# [Classifier] Found 15 documents with PDF attachments
# [Classifier] Total sections: 450
```

1.11 See Also

- feature_extraction.py - Section name TF-IDF implementation
- pdf_section_extractor.py - PDF section extraction
- PDF_SECTION_EXTRACTOR_SUMMARY.md - Section extractor features
- PDF_FIGURE_CAPTION_EXTRACTION.md - Figure caption handling

Status: ✅ Complete and Tested **Version**: 2.1 **Breaking Changes**: None (backwards compatible) **Features Implemented**: - ✅ Tokenizer parameter with 3 modes (line, paragraph, section) - ✅ Section-based PDF extraction from CouchDB - ✅ Automatic PDF document discovery (no doc IDs required!) - ✅ Section name TF-IDF features - ✅ Section filtering by name - ✅ Backwards compatibility via `line_level` property - ✅ Model metadata versioning (2.1)

Optional Enhancements: - Local PDF support (requires filesystem access to PDFs) - Optimized discovery using CouchDB views (for large databases) - Custom section patterns (requires pattern configuration)