Contents

1 Migration Guide: `tokenizer` → `extraction_mode` 1

1.1 Overview 1
1.2 Why the Change? 1
1.3 Migration Steps 2
1.3.1 Simple Replacement 2
1.4 Parameter Values 2
1.5 Examples 2
1.5.1 Training with Line-Level Extraction 2
1.5.2 Prediction with Section-Level Extraction 3
1.5.3 Training from Separate Database 3
1.6 Backwards Compatibility 4
1.6.1 Deprecated Property 4
1.7 Updated Documentation 4
1.8 Code Search and Replace 4
1.9 Terminology Changes 4
1.10 Implementation Details 5
1.11 Testing 5
1.12 Related Changes 5

1 Migration Guide: `tokenizer` → `extraction_mode`

1.1 Overview

The `tokenizer` parameter in `SkolClassifierV2` has been renamed to `extraction_mode` to better reflect its purpose. This parameter controls the **granularity of text extraction** (line, paragraph, or section level), not tokenization which happens during feature extraction.

Date: 2025-12-24 **Breaking Change**: Yes (parameter name changed) **Backwards Compatibility**: Deprecated parameter still supported

1.2 Why the Change?

The term `tokenizer` was misleading because: - It doesn't control tokenization (which is handled by Spark's ML Tokenizer during feature extraction) - It actually controls **how text is chunked/extracted** from documents before processing - The name suggested a lower-level operation than what it actually does

The new name `extraction_mode` more accurately describes the parameter's function: controlling the mode/granularity of text extraction.

1.3 Migration Steps

1.3.1 Simple Replacement

Replace `tokenizer=` with `extraction_mode=` in all `SkolClassifierV2` instantiations:

Before:

```
classifier = SkolClassifierV2(
    spark=spark,
    input_source='couchdb',
    tokenizer='section',  # OLD NAME
    use_suffixes=True
)
```

After:

```
classifier = SkolClassifierV2(
    spark=spark,
    input_source='couchdb',
    extraction_mode='section',  # NEW NAME
    use_suffixes=True
)
```

1.4 Parameter Values

The allowed values remain the same:

Value	Description
`'line'`	Extract and process text line-by-line
`'paragraph'`	Extract and process text by paragraphs (default)
`'section'`	Extract sections from PDFs with section name features

1.5 Examples

1.5.1 Training with Line-Level Extraction

Before:

```
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/*.txt.ann'],
    tokenizer='line',
    model_path='models/line_model.pkl'
)
```

After:

```
classifier = SkolClassifierV2(
    input_source='files',
    file_paths=['data/*.txt.ann'],
    extraction_mode='line',
    model_path='models/line_model.pkl'
)
```

1.5.2 Prediction with Section-Level Extraction

Before:

```
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_database='articles',
    tokenizer='section',
    section_name_vocab_size=50,
    output_dest='couchdb'
)
```

After:

```
classifier = SkolClassifierV2(
    input_source='couchdb',
    couchdb_database='articles',
    extraction_mode='section',
    section_name_vocab_size=50,
    output_dest='couchdb'
)
```

1.5.3 Training from Separate Database

Before:

```
classifier = SkolClassifierV2(
    couchdb_database='skol_dev',
    couchdb_training_database='skol_training',
    tokenizer='section',
    use_suffixes=True
)
```

After:

```
classifier = SkolClassifierV2(
    couchdb_database='skol_dev',
    couchdb_training_database='skol_training',
    extraction_mode='section',
    use_suffixes=True
)
```

1.6 Backwards Compatibility

1.6.1 Deprecated Property

The `line_level` property still works for backwards compatibility:

```
classifier = SkolClassifierV2(extraction_mode='line')
assert classifier.line_level == True  # Still works

classifier = SkolClassifierV2(extraction_mode='paragraph')
assert classifier.line_level == False  # Still works
```

1.7 Updated Documentation

The following documentation has been updated to use `extraction_mode`:

- TRAINING_DATABASE_SETUP.md
- PDF_TXT_ATTACHMENT_SUPPORT.md
- TXT_ATTACHMENT_IMPLEMENTATION.md
- CLASSIFIER_V2_TOKENIZER_UPDATE.md
- PDF_YEDDA_ANNOTATION_SUPPORT.md
- PDF_YEDDA_IMPLEMENTATION_SUMMARY.md

1.8 Code Search and Replace

To migrate your codebase, use this search and replace pattern:

Search: `tokenizer=` **Replace:** `extraction_mode=`

Search (regex): `tokenizer\s*[:=]\s*` **Replace:** `extraction_mode=`

1.9 Terminology Changes

Old Term	New Term	Context
"tokenizer mode"	"extraction mode"	Parameter description
"line tokenization"	"line extraction"	Processing description
"paragraph tokenization"	"paragraph extraction"	Processing description
"section tokenization"	"section extraction"	Processing description

1.10 Implementation Details

What Changed: - Parameter name: `tokenizer` → `extraction_mode` - Instance variable: `self.tokenizer` → `self.extraction_mode` - Documentation: Updated to use "extraction mode" terminology

What Didn't Change: - Allowed values (`'line'`, `'paragraph'`, `'section'`) - Default value (`'paragraph'`) - Behavior and functionality - Feature extraction pipeline - Model training and prediction

1.11 Testing

All test files have been updated: - test_training_db_access.py - test_train_with_section_names.py - test_parser_section_names.py

Run tests to verify migration:

```
python3 test_training_db_access.py
python3 test_train_with_section_names.py
```

1.12 Related Changes

This refactoring is part of the section name feature implementation: - TRAINING_DATABASE_SETUP.md - Training database setup - Section name detection in `AnnotatedTextParser` - NULL handling in `FeatureExtractor`

Status: ✅ Complete **Date**: 2025-12-24 **Breaking Change**: Parameter renamed (`tokenizer` → `extraction_mode`) **Migration**: Simple search and replace