## Contents

# 1 PDF Section Extractor - YEDDA Annotation Support

## 1.1 Overview

The `PDFSectionExtractor` now automatically parses YEDDA (Yet Another Entity Detection and Annotation) format annotations from PDF text and includes the active label for each section in the output DataFrame.

**Date**: 2025-12-22 **Breaking Changes**: None (backward compatible -

new nullable field)

## 1.2 YEDDA Format

YEDDA annotations use the format:

```
[@ text content
#Label*]
```

### 1.2.1 Features Supported

- **Nested Annotations**: When annotations are nested, the inner-most label takes precedence
- **Multi-line Annotations**: Annotations can span multiple lines
- **Cross-Page Support**: Annotations can cross page boundaries within the same PDF file
- **File Boundary**: Annotations do NOT cross file boundaries (each PDF is processed independently)

## 1.3 Usage

### 1.3.1 Basic Example

```python
from pyspark.sql import SparkSession
from pdf_section_extractor import PDFSectionExtractor

spark = SparkSession.builder.appName("PDFExtractor").getOrCreate()
extractor = PDFSectionExtractor(spark=spark)

# Extract from PDF with YEDDA annotations
df = extractor.extract_from_document(
    database='skol_dev',
    doc_id='document-id'
)

# DataFrame now includes 'label' column
df.select("value", "section_name", "label").show()
```

### 1.3.2 Example Output

Given a PDF with YEDDA annotations:

```
--- PDF Page 1 ---

[@ This is the introduction section.
#Introduction*]
```

```
[@ Glomus mosseae Nicolson & Gerdemann, 1963.
#Nomenclature*]

[@ Spores formed singly or in clusters.
#Description*]

This is unann otated conclusion text.
```

The resulting DataFrame:

| value | line_number | section_name | label |
|---|---|---|---|
| This is the introduction section. | 3 | NULL | Introduction |
| Glomus mosseae Nicolson & Gerdemann, 1963. | 6 | NULL | Nomenclature |
| Spores formed singly or in clusters. | 9 | NULL | Description |
| This is unannotated conclusion text. | 11 | NULL | NULL |

## 1.4 Nested Annotations

YEDDA annotations can nest, and the innermost label takes precedence:

```
[@ Outer annotation text
[@ Inner nested annotation
#InnerLabel*]
Back to outer annotation
#OuterLabel*]
```

Result: - Lines in inner annotation: label = “InnerLabel” - Lines in outer annotation only: label = “OuterLabel”

### 1.4.1 Example

```
Line 1: [@ This is nomenclature         # OuterLabel = Nomenclature
Line 2: [@ This is a description        # InnerLabel = Description
Line 3: of the species.                 # InnerLabel = Description
Line 4: #Description*]                  # InnerLabel = Description
Line 5: Back to nomenclature.           # OuterLabel = Nomenclature
Line 6: #Nomenclature*]                 # OuterLabel = Nomenclature
```

Labels assigned: - Lines 1: Nomenclature - Lines 2-4: Description (innermost wins!) - Lines 5-6: Nomenclature

## 1.5 DataFrame Schema

The DataFrame schema now includes a `label` field:

```
StructType([
    StructField("value", StringType(), False),
    StructField("doc_id", StringType(), False),
    StructField("attachment_name", StringType(), False),
    StructField("paragraph_number", IntegerType(), False),
    StructField("line_number", IntegerType(), False),
    StructField("page_number", IntegerType(), False),
    StructField("empirical_page_number", IntegerType(), True),  # Nullable
    StructField("section_name", StringType(), True),  # Nullable
    StructField("label", StringType(), True)  # Nullable - YEDDA annotation
])
```

## 1.6 Working with Labels

### 1.6.1 Filter by Label

```
# Get all nomenclature sections
nomenclature_df = df.filter(df.label == "Nomenclature")

# Get all annotated sections
annotated_df = df.filter(df.label.isNotNull())

# Get unannotated sections
unannotated_df = df.filter(df.label.isNull())
```

### 1.6.2 Label Statistics

```
from pyspark.sql.functions import count

# Count sections by label
label_counts = df.groupBy("label").agg(count("*").alias("count"))
label_counts.show()

# Output:
# +-------------+-----+
# |        label|count|
# +-------------+-----+
# |         NULL|  150|
# |Nomenclature |   45|
# | Introduction|   12|
# |  Description|   78|
# +-------------+-----+
```

### 1.6.3 Combine with Section Names

```
# Sections with both labels and section names
df.filter(
    df.label.isNotNull() & df.section_name.isNotNull()
).select("value", "section_name", "label").show()
```

## 1.7 Implementation Details

### 1.7.1 Parsing Algorithm

1. **Line-by-Line Parsing**: Text is split into lines (1-indexed)
2. **Stack-Based Tracking**: Annotation markers are tracked using a stack
3. **Label Assignment**: When an annotation closes, all lines in its range get the label
4. **Nesting Resolution**: Innermost labels are assigned first (overwriting outer labels)

### 1.7.2 Annotation Markers

- **Start marker**: `[@` - Pushes a new annotation onto the stack
- **End marker**: `#Label*]` - Pops from stack and assigns label to all lines in range

### 1.7.3 Label Lookup

For each section/paragraph: - The `line_number` field indicates the first line of the section - The `label` field contains the YEDDA label active at that line - If no annotation is active, `label` is NULL

## 1.8 Examples

### 1.8.1 Example 1: Training Data with Labels

```
# Load annotated training data
training_df = extractor.extract_from_document(
    database='training_data',
    doc_id='annotated_article_001'
)

# Filter to only labeled data for training
labeled_df = training_df.filter(training_df.label.isNotNull())

# Train classifier
from skol_classifier.classifier_v2 import SkolClassifierV2
```

```
classifier = SkolClassifierV2(
    spark=spark,
    input_source='dataframe',  # Hypothetical future feature
    use_suffixes=True,
    model_type='logistic'
)

classifier.fit(labeled_df.select("value", "label"))
```

#### 1.8.2 Example 2: Quality Control

```
# Check annotation coverage
total = df.count()
labeled = df.filter(df.label.isNotNull()).count()
coverage = (labeled / total) * 100

print(f"Annotation coverage: {coverage:.1f}%")
print(f"Labeled sections: {labeled}/{total}")
```

#### 1.8.3 Example 3: Export Annotations

```
# Export to JSON for review
annotations = df.filter(df.label.isNotNull()).select(
    "value", "label", "page_number", "line_number"
).collect()

import json
with open('annotations.json', 'w') as f:
    json.dump([row.asDict() for row in annotations], f, indent=2)
```

### 1.9 Integration with SkolClassifierV2

YEDDA labels can be used as training labels for the classifier:

```
from skol_classifier.classifier_v2 import SkolClassifierV2

# Extract with YEDDA annotations
classifier = SkolClassifierV2(
    spark=spark,
    input_source='couchdb',
    couchdb_url='http://localhost:5984',
    couchdb_database='training_docs',
    extraction_mode='section',  # Use PDF section extraction
    use_suffixes=True,
    model_type='logistic'
)

# Load sections with YEDDA labels
sections_df = classifier.load_raw()

# Filter to labeled sections for training
training_df = sections_df.filter(sections_df.label.isNotNull())

# Rename 'label' to match classifier expectations
training_df = training_df.withColumnRenamed("label", "label_from_yedda")

# Train on labeled data
# (Implementation depends on classifier API)
```

## 1.10 Compatibility

### 1.10.1 Backward Compatibility

- **Fully compatible**: Existing code continues to work
- **New field**: The `label` column is nullable and won't affect existing queries
- **No breaking changes**: All existing DataFrame operations work as before

### 1.10.2 Migration

No migration required. Existing code will see `NULL` values in the new `label` column for PDFs without YEDDA annotations.

## 1.11 Testing

Run the test suite:

```
python test_yedda_pdf_integration.py
```

Tests cover: - Basic annotation parsing - Nested annotation handling (innermost wins) - Multi-line annotations - DataFrame integration - NULL labels for unannotated text

## 1.12 Limitations

1. **Format**: Only supports YEDDA format `[@ text #Label*]`
2. **File Boundaries**: Annotations cannot cross file boundaries
3. **Validation**: No validation that closing labels match opening markers

4. **Error Handling**: Malformed annotations are silently ignored

## 1.13 Future Enhancements

Possible improvements: 1. **Validation**: Warn about unclosed or mismatched annotations 2. **Alternative Formats**: Support other annotation formats (XML, JSON) 3. **Annotation Metadata**: Include annotation confidence or source 4. **Label Hierarchies**: Support hierarchical label structures 5. **Annotation Conflicts**: Handle overlapping annotations more explicitly

## 1.14 See Also

- yedda_parser/README.md - YEDDA parser module
- PDF_SECTION_EXTRACTOR_SUMMARY.md - Complete feature summary
- PDF_FIGURE_CAPTION_EXTRACTION.md - Figure caption handling
- CLASSIFIER_V2_TOKENIZER_UPDATE.md - Section mode classifier

---

**Status**: ✅ Complete and Tested **Version**: Added 2025-12-22 **Breaking Changes**: None **New Features**: - ✅ YEDDA annotation parsing - ✅ Nested annotation support (innermost label wins) - ✅ Multi-line annotation support - ✅ Cross-page annotation support - ✅ Nullable `label` field in DataFrame schema - ✅ Full test coverage

**Known Limitations**: - Annotations cannot cross file boundaries - No validation of annotation format - Malformed annotations silently ignored