Annotation Tools and Data Formats II

Data Conversion and Pipelines

Jin Zhao

Brandeis University

March 11, 2025

Today's Agenda

1. Review of Label Studio basics
2. Data format conversion
3. Working with BIO format
4. Export formats and pipelines
5. Batch processing annotations
6. Quality control setup
7. From annotations to ML training

Data Format Review

Common annotation formats:

- **JSON:** Flexible, human-readable, native to Label Studio
- **Standoff:** Separate from text (brat format)
- **BIO/CONLL:** Token-per-line for sequence labeling
- **CSV/TSV:** Tabular, good for classification
- **XML:** Inline annotation (less common now)

Key consideration: What format does your ML pipeline expect?

Label Studio JSON Output

Example export from Label Studio:

```
{
  "id": 1,
  "data": {"text": "Apple released the iPhone."},
  "annotations": [{
    "result": [
      {"from_name": "ner", "to_name": "text",
       "type": "labels",
       "value": {"start": 0, "end": 5, "text": "Apple",
                 "labels": ["ORG"]}},
      {"from_name": "ner", "to_name": "text",
       "type": "labels",
       "value": {"start": 18, "end": 24, "text": "iPhone",
                 "labels": ["PRODUCT"]}}
    ]
  }]
```

Converting JSON to BIO Format

BIO format for sequence labeling models:

```
Apple    B-ORG
released O
the      O
iPhone   B-PRODUCT
.        O
```

Conversion requires:

1. Tokenize the text
2. Map character offsets to token indices
3. Assign B-/I-/O labels
4. Handle multi-token entities

Python Conversion Script

Basic conversion logic:

```
def convert_to_bio(text, annotations):
    tokens = text.split()  # Simple tokenization
    labels = ['O'] * len(tokens)

    for ann in annotations:
        start, end = ann['start'], ann['end']
        label = ann['labels'][0]

        # Find token indices (simplified)
        # ... mapping logic ...

        labels[token_idx] = f'B-{label}'
        for i in range(token_idx+1, end_idx):
            labels[i] = f'I-{label}'
```

Tokenization Alignment

Critical challenge: Character offsets to tokens

Problems:

- Entity spans may not align with token boundaries
- Different tokenizers produce different results
- Subword tokenization (BERT) adds complexity

Solutions:

1. Tokenize before annotation (recommended)
2. Use character-to-token mapping
3. Handle partial overlap carefully
4. Use spaCy or NLTK for consistent tokenization

Export for Classification Tasks

Simpler than sequence labeling:

Label Studio JSON → CSV:

text	label
“Great product!”	Positive
“Terrible service.”	Negative
“It was okay.”	Neutral

Direct export from Label Studio UI
Or use Python to parse JSON and create DataFrame

Non-Consuming Tags Export

Document-level labels to TSV:

```
import json
import csv

with open('export.json') as f:
    data = json.load(f)

with open('output.tsv', 'w') as f:
    writer = csv.writer(f, delimiter='\t')
    writer.writerow(['text', 'label'])

    for item in data:
        text = item['data']['text']
        label = item['annotations'][0]['result'][0]
                    ['value']['choices'][0]
        writer.writerow([text, label])
```

Batch Processing

When you have many files:

Pipeline steps:

1. Read all annotation files
2. Validate format consistency
3. Convert to target format
4. Combine into single dataset
5. Split into train/dev/test

Best practices:

- Use pathlib for cross-platform paths
- Avoid hard-coded file paths
- Use argparse for script arguments
- Log processing steps

Avoiding Hard-Coded Paths

Bad practice:

```
filepath = "/Users/jin/project/data/file.txt"
```

Good practice:

```
import argparse
from pathlib import Path

parser = argparse.ArgumentParser()
parser.add_argument('--input', type=Path, required=True)
parser.add_argument('--output', type=Path, required=True)
args = parser.parse_args()

# Now use args.input and args.output
```

Benefits: Other users don't need to edit your code

Quality Control in Pipelines

Validation checks to include:

1. **Format validation:** JSON schema, required fields
2. **Completeness:** All items annotated?
3. **Consistency:** Same labels used throughout?
4. **Span validity:** Offsets within text bounds?
5. **Label validity:** Only expected labels used?

Automated checks:

- Raise errors on invalid data
- Log warnings for suspicious patterns
- Generate quality report

Dataset Splitting

Train/Dev/Test splits:

Typical ratios: 80/10/10 or 70/15/15

Considerations:

- **Random split:** Simple, works for most cases
- **Stratified split:** Maintain label distribution
- **Document split:** Keep documents together
- **Time split:** Train on older, test on newer

Scikit-learn:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=0.2)
```

From Annotations to Features

Using annotations in ML models:

Classification:

- Text $\rightarrow$ features (bag of words, embeddings)
- Label $\rightarrow$ target variable
- Standard scikit-learn or transformers workflow

Sequence labeling:

- Tokens $\rightarrow$ features (word embeddings, context)
- BIO labels $\rightarrow$ target sequence
- CRF, BiLSTM-CRF, or BERT-based models

Key: Match annotation format to model input requirements

Useful Python Tools

For data processing:

- `pandas`: DataFrames, CSV/JSON I/O
- `pathlib`: Cross-platform path handling
- `json`: JSON parsing
- `argparse`: Command-line arguments

For NLP:

- `nltk`: Tokenization, stemming, stopwords
- `spacy`: Industrial-strength NLP
- `transformers`: BERT, GPT models
- `scikit-learn`: ML utilities

NLTK Utilities

Useful for preprocessing:

Tokenization:

- `nltk.tokenize.word_tokenize`
- `nltk.tokenize.sent_tokenize`
- `NLTKWordTokenizer` with span offsets

Text normalization:

- `nltk.stem.PorterStemmer`
- `nltk.corpus.stopwords`

Remember: Consistent preprocessing between annotation and training

Next Class: Tools Advanced

Lecture 16 (Mar 16): Annotation Tools Advanced — Feature Engineering

Topics:

- Argilla for RLHF data collection
- Custom annotation interfaces
- From annotations to features
- Representation learning from annotated data

Project: Draft 2 guidelines due
Assignment: HW 2 due

Key Takeaways

1. **Format conversion** is essential for ML pipelines
2. **BIO format** is standard for sequence labeling
3. **Tokenization alignment** is a critical challenge
4. **Avoid hard-coded paths** – use argparse
5. **Validate data** at each pipeline stage
6. **Consistent preprocessing** between annotation and training

Questions?

Questions?

Office Hours: Wednesdays 1-3pm, Volen 109

✉ jinzhao@brandeis.edu