Data Labeling & Annotation

Week 3 · CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar

Part 1: The Motivation

From raw data to supervised learning

Previously on CS 203...

Week 1: We collected movie data from the OMDB API

```
movies = []
for title in movie_list:
    response = requests.get(OMDB_API, params={"t": title})
    movies.append(response.json())
```

Week 2: We validated and cleaned the data

```
# Validated schema, fixed types, removed duplicates
clean_movies = validate_and_clean(raw_movies)
print(f"Clean dataset: {len(clean_movies)} movies")
```

Now: We have 10,000 clean movies. Time to build a model!

The Missing Ingredient

```python
# Our clean movie data
movies = [
    {"title": "Inception", "year": 2010, "plot": "A thief who..."},
    {"title": "The Room", "year": 2003, "plot": "Johnny is a..."},
    {"title": "Parasite", "year": 2019, "plot": "Greed and..."},
    # ... 9,997 more movies
]

# We want to predict: Is this a good movie?
# But wait... what makes a movie "good"?
```

The data doesn't tell us the answer. We need LABELS.

The Labeling Bottleneck

```
THE AI REALITY CHECK

Unlabeled Data:   ABUNDANT   (web, sensors, logs, databases)

Labeled Data:     SCARCE     (expensive, time-consuming)

Time on Labeling: 80% of AI project effort
```

This is the bottleneck that slows down most AI projects.

The Teacher Analogy

Machine learning is like teaching a child: You show examples ("this is a cat, this is a dog"), and the child learns to recognize the pattern. Without examples, there's nothing to learn from.

What labeled data provides:

- **Definition**: What exactly are we trying to predict?
- **Examples**: What does "correct" look like?
- **Boundaries**: Where does one category end and another begin?
- **Ground truth**: How do we know if the model is right?

No labels = No supervision = No learning direction

Why Do We Need Labels?

1. Supervised Learning requires ground truth

```
Input: "This movie was terrible!"  ⟶  Model  ⟶  Output: ???

Without labels, model can't learn what "terrible" means for the task.
```

2. Evaluation needs a test set

- Even unsupervised methods need labels to verify quality

3. Labeling forces you to define the problem

- What exactly is "spam"? What counts as "positive sentiment"?
- Ambiguity in labeling = ambiguity in your model

Today's Mission

Learn to transform unlabeled data into labeled training data.

```
TODAY'S JOURNEY

1. Where does unlabeled data come from?
2. Types of labeling tasks (text, image, audio, video)
3. How to label: tools and platforms
4. How to measure label quality (IAA, Cohen's Kappa)
5. Quality control and guidelines
6. Managing annotation teams
```

Part 2: Where Does Data Come From?

Sources of unlabeled data

The Data Landscape

```
┌─────────────────────────────────────────────┐
│          SOURCES OF UNLABELED DATA          │
├─────────────────────────────────────────────┤
│                                             │
│  PUBLIC                 PRIVATE             │
│  ──────                 ───────             │
│  - Web scraping         - Company databases │
│  - Public APIs          - User uploads      │
│  - Open datasets        - Internal logs     │
│  - Government data      - Sensor streams    │
│  - Social media         - Transaction records│
│                                             │
└─────────────────────────────────────────────┘
```

Unlabeled data is everywhere. The challenge is getting labels.

Public Data Sources

Web Scraping (Week 1 topic):

```
# News articles, product listings, reviews
soup = BeautifulSoup(response.text)
articles = soup.find_all('article')
```

Public APIs:

- Twitter/X API - millions of tweets
- Reddit API - discussions and comments
- Wikipedia API - encyclopedic text

Open Datasets:

- Common Crawl - petabytes of web pages
- ImageNet - millions of images (but WITH labels!)
- The Pile - 800GB of diverse text

Private/Enterprise Data

User-Generated Content:

```
# E-commerce reviews
reviews = db.query("SELECT * FROM product_reviews")
# No sentiment labels!
```

Operational Logs:

```
# Server logs, user behavior
logs = parse_log_files("/var/log/app/")
# No "normal" vs "anomaly" labels!
```

Sensor Data:

```
# IoT devices, cameras, microphones
sensor_stream = read_sensor(device_id)
# No event annotations!
```

The Label Gap

```
                UNLABELED                     LABELED

Common Crawl    [════════════════]
(400TB)
                                              ImageNet
YouTube         [══════════]                  [=]
(500 hrs/min)                                 (14M images)

Company Logs    [════════]                    SQuAD
(varies)                                      [.]
                                              (100K QA pairs)
```

The gap between available data and labeled data is enormous.

From Unlabeled to Labeled

```
┌─────────────┐      ┌─────────────┐      ┌─────────────┐
│  Unlabeled  │  →   │  Labeling   │  →   │   Labeled   │
│    Data     │      │  Process    │      │    Data     │
│  (10,000)   │      │             │      │  (10,000)   │
└─────────────┘      └─────────────┘      └─────────────┘
                            |
                            v
                     ┌───────────┐
                     │   Cost:   │
                     │ $$$$$     │
                     │   Time    │
                     │  Effort   │
                     └───────────┘
```

The labeling process is where the real work happens.

Part 3: Types of Labeling Tasks

Different problems, different annotation needs

Annotation Task Taxonomy

TEXT	IMAGES
Classification	Classification
Named Entity Recognition	Object Detection (bbox)
Sentiment Analysis	Segmentation (pixel)
Question Answering	Keypoint Detection
Relation Extraction	Instance Segmentation

AUDIO	VIDEO
Transcription	Action Recognition
Speaker Identification	Object Tracking
Event Detection	Temporal Segmentation
Emotion Recognition	Dense Captioning

Task Complexity Spectrum

```
SIMPLE                                                        COMPLEX
  |                                                              |
  v                                                              v

Binary          Multi-class        Sequence        Pixel-level
Classification  Classification     Labeling        Segmentation

[Spam/Not]      [Cat/Dog/Bird]     [NER Tags]      [Every Pixel]

  2 min/item     5 min/item        10 min/item     30+ min/item
```

More complex tasks = more time = more cost

Part 3a: Text Annotation Tasks

Text: Classification

Task: Assign label(s) to entire text.

```
# Binary Classification
{"text": "This movie was terrible!", "label": "NEGATIVE"}

# Multi-class Classification
{"text": "How do I reset my password?", "label": "ACCOUNT_SUPPORT"}

# Multi-label Classification
{"text": "Great phone with poor battery",
 "labels": ["POSITIVE_FEATURE", "NEGATIVE_FEATURE"]}
```

Annotation Interface: Radio buttons, checkboxes, or dropdown

Speed: 200-500 examples/hour

Text Classification: Annotation Diagram

```
TEXT CLASSIFICATION INTERFACE

Text: "The movie had stunning visuals but a weak plot."

  Select sentiment:

    ( ) Positive
    (o) Mixed            ⟵ Selected
    ( ) Negative
    ( ) Neutral

                          [Submit]
```

Text: Named Entity Recognition (NER)

Task: Identify and classify spans of text.

```
Text: "Apple CEO Tim Cook announced iPhone 15 in Cupertino."

Entities:
  [Apple]       @ 0:5   → ORGANIZATION
  [Tim Cook]    @ 10:18 → PERSON
  [iPhone 15]   @ 29:38 → PRODUCT
  [Cupertino]   @ 42:51 → LOCATION
```

Annotation Format (JSON):

```json
{
  "text": "Apple CEO Tim Cook ... ",
  "entities": [
    {"start": 0, "end": 5, "label": "ORG"},
    {"start": 10, "end": 18, "label": "PERSON"}
  ]
}
```

NER: Annotation Diagram

```
┌─────────────────────────────────────────────────────────┐
│ NER ANNOTATION INTERFACE                                │
├─────────────────────────────────────────────────────────┤
│                                                         │
│ [Apple] CEO [Tim Cook] announced [iPhone 15] in [Cupertino]. │
│  ────        ───────               ───────     ───────   │
│  ORG         PERSON                PRODUCT     LOCATION │
│                                                         │
│ Labels:                                                 │
│ ┌────────┬────────┬────────┬────────┐                   │
│ │  ORG   │ PERSON │PRODUCT │  LOC   │                    │
│ └────────┴────────┴────────┴────────┘                    │
│                                                         │
│ Instructions: Highlight text, then click label          │
│                                                         │
└─────────────────────────────────────────────────────────┘
```

NER: Common Challenges

1. Boundary Ambiguity:

```
"New York City Mayor"
Tag "New York" or "New York City"?
```

2. Nested Entities:

```
"MIT AI Lab director"
- "MIT AI Lab" → ORGANIZATION
- "MIT" → ORGANIZATION (nested inside!)
```

3. Overlapping Context:

```
"Bank of America" - ORG or LOC?
"Washington" - PERSON or LOC?
```

Solution: Clear guidelines with examples for every edge case.

Text: Sentiment Analysis

Task: Classify opinion/emotion in text.

```
# Document-level: One label per document
{"text": "Great movie! Loved every moment.", "sentiment": "POSITIVE"}

# Sentence-level: Label each sentence
{"text": "Great visuals. Poor story.", "sentences": [
    {"text": "Great visuals.", "sentiment": "POSITIVE"},
    {"text": "Poor story.", "sentiment": "NEGATIVE"}
]}
```

Sentiment: Aspect-Based Analysis

```
# Aspect-based: Sentiment per feature/aspect
{"text": "Great camera but poor battery", "aspects": [
    {"aspect": "camera", "sentiment": "POSITIVE"},
    {"aspect": "battery", "sentiment": "NEGATIVE"}
]}
```

Granularity Spectrum:

- **Document-level**: Simplest, fastest
- **Sentence-level**: More nuanced
- **Aspect-based**: Most detailed, slowest

Sentiment: The Ambiguity Problem

Sarcasm:

```
"Yeah, great service... 2 hour wait!"
Literal: POSITIVE  |  Actual: NEGATIVE
```

Mixed Sentiment:

```
"Good product, terrible delivery"
What's the overall label?
```

Neutral vs No Opinion:

```
"The phone is blue"     → NEUTRAL (factual)
"I received the phone" → NEUTRAL (no sentiment)
```

These require clear guidelines!

Text: Question Answering

Task: Find answer span in passage.

```json
{
  "context": "The Apollo program landed 12 astronauts on
              the Moon between 1969 and 1972.",
  "question": "When did Apollo land astronauts?",
  "answers": [
    {"text": "between 1969 and 1972", "start": 54}
  ]
}
```

```
┌─────────────────────────────────────────────────────────────────────┐
│  Context: The Apollo program landed 12 astronauts on the Moon      │
│           [between 1969 and 1972].  ⟵ Highlighted answer            │
│                                                                     │
│  Question: When did Apollo land astronauts?                         │
│                                                                     │
│  [ ] No answer in text                                              │
└─────────────────────────────────────────────────────────────────────┘
```

Text: Relation Extraction

Task: Identify relationships between entities.

```
Text: "Steve Jobs founded Apple in 1976."

Entities:
  - "Steve Jobs" → PERSON
  - "Apple" → ORGANIZATION
  - "1976" → DATE

Relations:
  - (Steve Jobs, FOUNDED, Apple)
  - (Apple, FOUNDED_IN, 1976)
```

Annotation Process:

1. First pass: Mark entities (NER)
2. Second pass: Draw relations between entities

Relation Extraction: Diagram

```
RELATION EXTRACTION INTERFACE

Text: [Steve Jobs] founded [Apple] in [1976].

        PERSON                ORG      DATE

              FOUNDED
               ─────────────>
                               FOUNDED
                                 IN
                                ─────────>

Relation Types: [FOUNDED] [WORKS_AT] [BORN_IN] [LOCATED_IN]
```

Part 3b: Image Annotation Tasks

Image: Classification

Task: Assign label(s) to entire image.

```
// Single-label
{"image": "photo.jpg", "label": "CAT"}

// Multi-label
{"image": "scene.jpg", "labels": ["OUTDOOR", "PEOPLE", "DAYTIME"]}

// Fine-grained
{"image": "dog.jpg", "breed": "GOLDEN_RETRIEVER", "category": "DOG"}
```

Speed: 100-300 images/hour

Image Classification: Diagram

```
IMAGE CLASSIFICATION INTERFACE

    [  Image of a cat  ]

What animal is in this image?

[  Cat  ]  [ Dog ]  [ Bird ]  [ Other ]
    ^
    Selected
```

Image: Object Detection

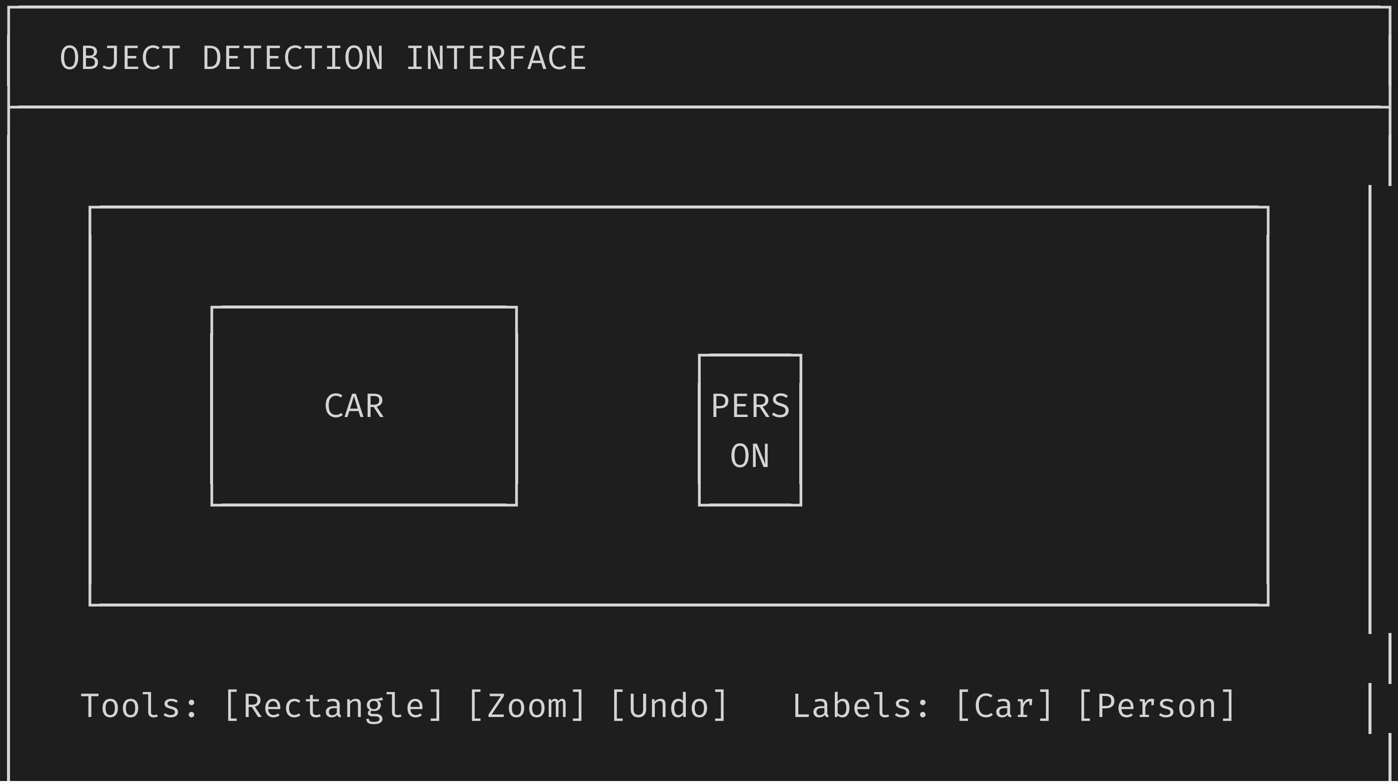
Task: Locate and classify objects with bounding boxes.

```json
{
  "image": "street.jpg",
  "width": 1920, "height": 1080,
  "objects": [
    {"class": "car", "bbox": [100, 200, 400, 300]},
    {"class": "person", "bbox": [800, 150, 100, 350]}
  ]
}
```

bbox format: `[x, y, width, height]` or `[x1, y1, x2, y2]`

Speed: 20-50 images/hour (5-10 objects each)

Object Detection: Diagram



Object Detection: Best Practices

Bounding Box Tightness:

```
Too Loose:  [    ├──object──┤      ]  Bad
Too Tight:  [--objec]                Bad (cuts off)
Just Right: [ ├──object──┤ ]         Good (small margin)
```

Occlusion Rules:

- Label partially visible objects? (>20% visible = yes)
- How to handle overlapping boxes?

Edge Cases:

- Reflections in mirrors?
- Objects in pictures on walls?
- Tiny/distant objects?

Image: Semantic Segmentation

Task: Classify every pixel in image.

```
Input:  RGB image (1920×1080×3)
Output: Label mask (1920×1080) where each pixel in {0,1,2,...}

Pixel values:
  0 → Background
  1 → Person
  2 → Car
  3 → Road
   ...
```

Speed: 5-15 images/hour (very time-consuming!)

Segmentation: Diagram

```
┌──────────────────────────────────────────────────────────────┐
│ SEGMENTATION INTERFACE                                       │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│   Original Image             Segmentation Mask               │
│   ┌─────────────────┐        ┌─────────────────┐             │
│   │   ████████      │        │   ████████      │ ← Person (red) │
│   │   ████████      │   →    │   ████████      │             │
│   │ ████████████    │        │ ▓▓▓▓▓▓▓▓▓▓▓▓    │ ← Car (blue) │
│   │▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ │       │░░░░░░░░░░░░░░░░│ ← Road (gray) │
│   └─────────────────┘        └─────────────────┘             │
│                                                              │
│   Tools: [Brush] [Polygon] [Magic Wand] [SAM]                │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

Instance vs Semantic Segmentation

```
SEGMENTATION COMPARISON

SEMANTIC SEGMENTATION          INSTANCE SEGMENTATION

All "person" pixels            Person #1 → Instance 1
get class ID = 1               Person #2 → Instance 2
                               Person #3 → Instance 3

█ █ █                          █ ▓ ░
█ █ █                          █ ▓ ░
all same                       different
```

Image: Keypoint Detection

Task: Locate specific points (joints, landmarks).

```json
{
  "image": "person.jpg",
  "keypoints": [
    {"name": "nose", "x": 120, "y": 80, "visible": 1},
    {"name": "left_eye", "x": 110, "y": 75, "visible": 1},
    {"name": "left_shoulder", "x": 100, "y": 150, "visible": 1},
    {"name": "right_shoulder", "x": 140, "y": 150, "visible": 0}
  ]
}
```

Visibility flags: 0=occluded, 1=visible, 2=outside image

Keypoint Detection: Diagram

```
┌──────────────────────────────────────────────────────┐
│  KEYPOINT ANNOTATION (Human Pose)                    │
├──────────────────────────────────────────────────────┤
│                                                      │
│                   o  ← nose                          │
│                 /   \                                │
│                o     o  ← eyes                       │
│                 \ /                                  │
│                   |                                  │
│              o───+───o  ← shoulders                  │
│              |    |    |                             │
│              o    |    o  ← elbows                   │
│              |    |    |                             │
│              o    |    o  ← wrists                   │
│                   |                                  │
│              o───+───o  ← hips                       │
│              |         |                             │
```

Part 3c: Audio Annotation Tasks

Audio: Transcription

Task: Convert speech to text with timestamps.

```
{
  "audio": "interview.wav",
  "transcription": [
    {"start": 0.0, "end": 3.2, "speaker": "A",
     "text": "Hello, how are you?"},
    {"start": 3.5, "end": 5.8, "speaker": "B",
     "text": "I'm doing well, thank you."}
  ]
}
```

Speed: 15-30 min of audio per hour of work (3-4x real-time)

Audio Transcription: Diagram

```
AUDIO TRANSCRIPTION INTERFACE

Waveform:

├─────────┼────────────┼─────────┤
0:00      0:03         0:06      0:09

Segments:
[Speaker A: "Hello, how are you?"]  [0:00 - 0:03]
[Speaker B: "I'm doing well."]      [0:03 - 0:06]

Controls: [Play] [Pause] [0.5x] [1x] [1.5x] [Loop]
```

Transcription Challenges

1. Speaker Diarization - Who is speaking?

```
[0:00-0:02] Speaker A: "I think that--"
[0:01-0:03] Speaker B: "No wait, listen..."  ← Overlap!
```

2. Background Sounds

```
"The cat [dog barking] jumped over the fence"
```

3. Accents & Dialects

```
"gonna" vs "going to" - Transcribe verbatim?
```

4. Filler Words

```
"So, um, I was thinking, like, maybe we could, uh..."
Include or remove?
```

Audio: Sound Event Detection

Task: Identify and timestamp sound events.

```
{
  "audio": "home_audio.wav",
  "events": [
    {"start": 2.3, "end": 3.1, "label": "door_slam"},
    {"start": 5.0, "end": 8.2, "label": "dog_bark"},
    {"start": 10.5, "end": 11.0, "label": "glass_break"}
  ]
}
```

Applications:

- Surveillance: gunshot, glass break
- Healthcare: cough, snore
- Environment: bird species, vehicles

Audio: Speaker & Emotion Recognition

Speaker Recognition:

```json
{
  "audio": "meeting.wav",
  "segments": [
    {"start": 0, "end": 5, "speaker": "Alice"},
    {"start": 5, "end": 12, "speaker": "Bob"}
  ]
}
```

Emotion Recognition:

```json
{
  "audio": "utterance.wav",
  "emotion": "angry",
  "arousal": 7,    // 1-9 scale (calm to excited)
  "valence": 2     // 1-9 scale (negative to positive)
}
```

Challenge: Emotion is subjective - expect lower IAA

Part 3d: Video Annotation Tasks

Video: Action Recognition

Task: Classify actions in video clips.

```
{
  "video": "sports.mp4",
  "fps": 30,
  "actions": [
    {"start_frame": 0, "end_frame": 90, "label": "running"},
    {"start_frame": 90, "end_frame": 150, "label": "jumping"},
    {"start_frame": 150, "end_frame": 200, "label": "landing"}
  ]
}
```

Types:

- **Clip-level**: One label per clip
- **Temporal**: Start/end for each action
- **Dense**: Label every frame

Video: Object Tracking

Task: Follow objects across frames.

```json
{
  "video": "traffic.mp4",
  "tracks": [
    {
      "track_id": 1,
      "category": "car",
      "bboxes": [
        {"frame": 0, "bbox": [100, 200, 50, 80]},
        {"frame": 1, "bbox": [105, 202, 50, 80]},
        {"frame": 2, "bbox": [110, 204, 50, 80]}
      ]
    }
  ]
}
```

Challenges: Occlusion, re-identification after disappearing

Video Tracking: Diagram

```
┌──────────────────────────────────────────────────────────┐
│ VIDEO TRACKING INTERFACE                                 │
├──────────────────────────────────────────────────────────┤
│                                                          │
│ Frame 1          Frame 2          Frame 3          Frame 4
│ ┌───────┐        ┌───────┐        ┌───────┐        ┌───────┐
│ │ [1]   │   →    │  [1]  │   →    │   [1] │   →    │    [1]│
│ │   [2] │        │   [2] │        │   [2] │        │   [2] │
│ └───────┘        └───────┘        └───────┘        └───────┘
│                                                          │
│ Track 1: Car (ID=1) - continuous path                    │
│ Track 2: Person (ID=2) - stationary                      │
│                                                          │
│ Timeline: [═══╪════╪════╪════╪════╪════╪════╪════]       │
│           0    1    2    3    4    5    6    7    8       │
```

Video: Temporal Segmentation

Task: Divide video into meaningful segments.

```json
{
  "video": "cooking_recipe.mp4",
  "segments": [
    {"start": 0, "end": 15, "label": "gather_ingredients"},
    {"start": 15, "end": 45, "label": "chop_vegetables"},
    {"start": 45, "end": 90, "label": "cook_in_pan"},
    {"start": 90, "end": 120, "label": "plate_and_serve"}
  ]
}
```

Applications: Sports analysis, surgical videos, tutorials

Annotation Speed Benchmarks

Task Type	Speed (per hour)	Complexity
Text Classification	200-500 items	Low
Sentiment Analysis	150-300 items	Low
NER	50-150 sentences	Medium
Image Classification	100-300 images	Low
Bounding Boxes	20-50 images	Medium
Segmentation	5-15 images	High
Audio Transcription	15-30 min audio	Medium
Video Tracking	5-10 min video	High

Use these to estimate labeling time and cost!

Part 4: Labeling Tools & Platforms

Software for annotation

Labeling Tool Landscape

Open Source	Commercial
Label Studio (flexible)	Labelbox (enterprise)
CVAT (video/CV focused)	Scale AI (full service)
Doccano (NLP focused)	V7 (auto-annotation)
VGG Image Annotator	Prodigy (active learning)
	Amazon SageMaker GT

Start open source, scale to commercial when needed.

Label Studio: The Swiss Army Knife

Open-source, web-based, highly flexible

```
# Installation
pip install label-studio

# Start server
label-studio start

# Access at http://localhost:8080
```

Key Features:

- Supports all modalities (text, image, audio, video)
- Customizable interfaces via XML config
- Export to many formats (JSON, CSV, COCO, YOLO)
- ML-assisted labeling
- Multi-user support

Label Studio: Text Classification Config

```xml
<View>
  <Text name="text" value="$text"/>
  <Choices name="sentiment" toName="text" choice="single">
    <Choice value="Positive"/>
    <Choice value="Negative"/>
    <Choice value="Neutral"/>
  </Choices>
</View>
```

Result: Text displayed with radio buttons for selection.

Label Studio: NER Config

```xml
<View>
  <Labels name="ner" toName="text">
    <Label value="PERSON" background="#FF0000"/>
    <Label value="ORG" background="#00FF00"/>
    <Label value="LOCATION" background="#0000FF"/>
    <Label value="DATE" background="#FFFF00"/>
  </Labels>
  <Text name="text" value="$text"/>
</View>
```

Result: Highlight text to assign entity labels.

Label Studio: Object Detection Config

```xml
<View>
  <Image name="img" value="$image"/>
  <RectangleLabels name="bbox" toName="img">
    <Label value="Car" background="red"/>
    <Label value="Person" background="blue"/>
    <Label value="Bicycle" background="green"/>
  </RectangleLabels>
</View>
```

Supports: Rectangles, polygons, brush, keypoints

CVAT: Video & CV Focus

Computer Vision Annotation Tool (Intel)

Strengths:

- Excellent for video annotation
- Automatic tracking (interpolation)
- Semi-automatic annotation with models
- 3D cuboid support

```
# Docker installation
docker-compose up -d
```

Best for: Video object tracking, autonomous driving datasets

Crowdsourcing Platforms

Platform	Cost	Quality	Best For
Amazon MTurk	\$0.01-0.10/task	Variable	Simple tasks
Scale AI	\$1-5/task	High	Complex CV
Labelbox	Subscription	Med-High	Enterprise
Prolific	\$0.10-0.50/task	Higher	Research
Appen	Variable	Medium	Multilingual

Key Insight: You get what you pay for.

Tool Selection Guide

Use Case	Recommended Tool
Prototyping / Learning	Label Studio
Video / Tracking	CVAT
NLP / Text Focus	Prodigy or Doccano
Enterprise Scale	Labelbox
Full-Service	Scale AI
Research Crowdsourcing	Prolific
Budget Crowdsourcing	Amazon MTurk

Start with Label Studio (free), scale up as needed.

Part 5: Label Quality & IAA

How do we know labels are correct?

The Quality Problem

Labels are created by humans. Humans make mistakes.

```
Annotator 1: "This movie was okay" → POSITIVE
Annotator 2: "This movie was okay" → NEUTRAL
Annotator 3: "This movie was okay" → NEGATIVE
```

Who is right?

We need a way to measure agreement and quality.

Why Agreement Matters

> **If humans can't agree, how can we expect a model to learn?** The ceiling for model performance is roughly human-level agreement. Low agreement = noisy labels = confused model.

The chain reaction:

```
Ambiguous task definition
       ↓
Annotators interpret differently
       ↓
Inconsistent labels in training data
       ↓
Model learns conflicting patterns
       ↓
Poor and unpredictable performance
```

Fix it at the source: Clear guidelines → High agreement → Clean labels → Better models

Types of Agreement Metrics

Data Type	Metrics
Categorical	Percent Agreement, Cohen's Kappa (2 raters), Fleiss' Kappa (3+ raters), Krippendorff's Alpha
Continuous/Ordinal	Pearson Correlation, Spearman Correlation, Intraclass Correlation
Spatial (Images)	IoU (Intersection over Union), Dice Coefficient

Choose metric based on your data type and number of annotators!

Percent Agreement: The Naive Approach

```
agreed = sum(ann1 == ann2 for ann1, ann2 in zip(labels1, labels2))
agreement = agreed / total_items
```

Problem: Doesn't account for chance!

Example:

- Binary task (Yes/No)
- Both annotators guess randomly
- Expected agreement by chance: 50%
- 60% agreement sounds okay, but it's barely better than random!

Cohen's Kappa

Accounts for chance agreement

```
          observed_agreement - chance_agreement
kappa = ─────────────────────────────────────────
                1 - chance_agreement
```

In notation:

```
      P_observed - P_expected
k = ─────────────────────────
         1 - P_expected
```

Why Kappa, Not Just Percent Agreement?

The coin-flip problem: Two annotators randomly guessing on a binary task will agree 50% of the time. That's not skill - that's chance. Kappa tells you how much better than chance your agreement is.

Intuition with examples:

Scenario	% Agreement	Kappa	Interpretation
Both guess randomly (binary)	50%	0.0	No better than chance
Slight improvement	60%	0.2	Barely better than chance
Real agreement	80%	0.6	Moderate agreement
Almost perfect	95%	0.9	Excellent

Kappa normalizes for chance, giving a fairer picture of true agreement.

Cohen's Kappa: Python

```python
from sklearn.metrics import cohen_kappa_score

annotator1 = ['pos', 'neg', 'pos', 'pos', 'neg', 'pos', 'neg', 'neg']
annotator2 = ['pos', 'neg', 'neg', 'pos', 'neg', 'pos', 'neg', 'pos']

kappa = cohen_kappa_score(annotator1, annotator2)
print(f"Cohen's Kappa: {kappa:.2f}")  # 0.50
```

Kappa Interpretation

KAPPA INTERPRETATION

Kappa Value	Agreement Level	Action
< 0.00	Poor	Redesign task
0.00 - 0.20	Slight	Major guideline revision
0.21 - 0.40	Fair	Significant revision
0.41 - 0.60	Moderate	Minor revision
0.61 - 0.80	Substantial	Guidelines working
0.81 - 1.00	Almost Perfect	Excellent!

Target: kappa >= 0.8 for most production tasks

If kappa < 0.6: Improve your guidelines before proceeding!

Kappa Example: Step by Step

Scenario: 2 annotators label 100 items (Spam / Not Spam)

```
                     Annotator B
                     Spam     Not Spam     Total
    Annotator A  Spam     45          5           50
                 Not      5           45          50
                 Total    50          50          100
```

Step 1: Observed Agreement

```
    P_o = (45 + 45) / 100 = 0.90
```

Kappa Example: Step by Step (cont.)

Step 2: Expected Agreement by Chance

```
P(A says Spam) = 50/100 = 0.5
P(B says Spam) = 50/100 = 0.5

P(both say Spam by chance) = 0.5 * 0.5 = 0.25
P(both say Not Spam by chance) = 0.5 * 0.5 = 0.25

P_e = 0.25 + 0.25 = 0.50
```

Step 3: Calculate Kappa

```
        P_o - P_e      0.90 - 0.50
k  =  ───────────  =  ─────────────  =  0.80
        1 - P_e        1 - 0.50
```

Result: Substantial agreement!

Fleiss' Kappa: Multiple Annotators

When you have more than 2 annotators:

```python
from statsmodels.stats.inter_rater import fleiss_kappa
import numpy as np

# Data format: (n_items, n_categories)
# Each cell = count of annotators who chose that category
data = np.array([
    [3, 0, 0],  # Item 1: all 3 chose category 0
    [1, 2, 0],  # Item 2: 1 chose cat 0, 2 chose cat 1
    [0, 1, 2],  # Item 3: 1 chose cat 1, 2 chose cat 2
])

print(f"Fleiss' Kappa: {fleiss_kappa(data):.3f}")
```

IoU for Spatial Annotations

For bounding boxes and segmentation:

```
                 Area of Overlap
IoU = ─────────────────────────────────
       Area of Union (both boxes)


      ┌─────────────┐
      │    ┌────────┼───────┐
      │    │////////│       │
      │    │//Ovrlp/│       │
      └────┼────────┘       │
           │                │
           └────────────────┘


IoU = Overlap / (Box1 + Box2 - Overlap)
```

IoU > 0.5: Generally considered a match

IoU > 0.7: Good agreement for detection tasks

IoU: Python Implementation

```python
def calculate_iou(box1, box2):
    """Calculate IoU between two boxes [x1, y1, x2, y2]."""
    # Intersection coordinates
    x1, y1 = max(box1[0], box2[0]), max(box1[1], box2[1])
    x2, y2 = min(box1[2], box2[2]), min(box1[3], box2[3])

    if x2 < x1 or y2 < y1:
        return 0.0  # No overlap

    intersection = (x2 - x1) * (y2 - y1)
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])

    return intersection / (area1 + area2 - intersection)
```

IoU: Usage Example

```
box1 = [100, 100, 200, 200]  # Ground truth
box2 = [150, 150, 250, 250]  # Prediction

iou = calculate_iou(box1, box2)
print(f"IoU: {iou:.2f}")  # 0.14

# Thresholds
if iou > 0.7:
    print("Good match!")
elif iou > 0.5:
    print("Acceptable match")
else:
    print("Poor match - boxes don't align well")
```

Typical IAA by Task Type

Task	Metric	Typical Good IAA
Text Classification	Cohen's Kappa	> 0.8
Sentiment Analysis	Cohen's Kappa	> 0.7
NER	Span F1	> 0.85
Object Detection	Mean IoU	> 0.7
Segmentation	Mean IoU	> 0.8
Transcription	WER < 5%	Between annotators
Emotion Recognition	Cohen's Kappa	> 0.6 (subjective)

Part 6: Quality Control

Ensuring consistent, accurate labels

The Quality Control Framework

Pillar	Description
1. GUIDELINES	Clear, detailed annotation instructions
2. TRAINING	Calibration sessions with annotators
3. GOLD STANDARD	Known-correct examples for testing
4. REDUNDANCY	Multiple annotators per item
5. MONITORING	Ongoing IAA and accuracy tracking
6. ADJUDICATION	Expert resolution of disagreements

All six pillars work together for quality labels!

1. Guidelines: The Foundation

Good guidelines include:

Clear Definitions:

```
SPAM: Unsolicited commercial email sent in bulk.
NOT SPAM: Personal email, newsletters you subscribed to.
```

Positive and Negative Examples:

```
SPAM examples:
  - "Buy cheap meds now! Click here!"
  - "You've won $1,000,000!"

NOT SPAM examples:
  - "Meeting tomorrow at 3pm"
  - "Your order has shipped"
```

Guidelines: Edge Cases

The real value is in edge cases:

```
## Edge Cases

Q: Promotional email from a store I bought from?
A: NOT SPAM (established commercial relationship)

Q: Newsletter I don't remember subscribing to?
A: NOT SPAM if it has unsubscribe link, SPAM otherwise

Q: Email from unknown sender asking for information?
A: SPAM (even if not selling anything - potential phishing)
```

Guidelines: Decision Trees

```
Is the email from a known sender?
├── Yes: Is the content relevant?
│   ├── Yes: NOT SPAM
│   └── No: Does it have unsubscribe link?
│       ├── Yes: NOT SPAM (newsletter)
│       └── No: SPAM
└── No: Does it ask for money or personal info?
    ├── Yes: SPAM
    └── No: Is it selling something?
        ├── Yes: SPAM
        └── No: Probably NOT SPAM (review manually)
```

Decision trees reduce annotator uncertainty.

2. Training: Calibration Sessions

Before Production Labeling:

1. **Study guidelines** - All annotators read same document
2. **Practice round** - Label 20-50 items independently
3. **Group discussion** - Review disagreements together
4. **Update guidelines** - Add clarifications based on discussion
5. **Second practice** - Label another 20-50 items
6. **Measure IAA** - Must reach threshold before production

```python
# Target: Kappa > 0.8 before production
if kappa < 0.8:
    print("Need more calibration!")
    revise_guidelines()
    run_another_practice_round()
```

3. Gold Standard Questions

Mix known-correct items into the task:

```python
# Create gold standard set (obvious examples)
gold_items = [
    {"text": "FREE MONEY CLICK NOW!!!", "label": "SPAM"},
    {"text": "Meeting at 3pm tomorrow", "label": "NOT_SPAM"},
    {"text": "Your Amazon order shipped", "label": "NOT_SPAM"},
]

# Mix 10% gold items into annotation queue
all_items = real_items + random.sample(gold_items, k=n//10)
random.shuffle(all_items)
```

Gold Standard: Monitoring Accuracy

```python
def evaluate_annotator(annotations, gold_answers):
    correct = sum(a == g for a, g in zip(annotations, gold_answers))
    return correct / len(gold_answers)

# Check annotator performance on gold items
accuracy = evaluate_annotator(annotator_labels, gold_labels)

if accuracy < 0.9:
    flag_annotator_for_review()
    print(f"Annotator accuracy: {accuracy:.0%} - needs retraining")
```

Target: >90% accuracy on gold questions

4. Redundancy

Multiple annotators per item:

```python
# Assign each item to 3 annotators
annotations = {
    "item_1": ["SPAM", "SPAM", "NOT_SPAM"],
    "item_2": ["NOT_SPAM", "NOT_SPAM", "NOT_SPAM"],
    "item_3": ["SPAM", "NOT_SPAM", "SPAM"],
}

# Majority vote aggregation
def majority_vote(labels):
    return max(set(labels), key=labels.count)

final_labels = {item: majority_vote(lbls) for item, lbls in annotations.items()}
```

Redundancy: Handling Disagreements

```
# Flag items with low agreement for expert review
for item, labels in annotations.items():
    if len(set(labels)) > 1:  # Any disagreement
        send_to_expert(item)

# Example output
# item_1: 2/3 agree → majority vote = SPAM
# item_2: 3/3 agree → unanimous = NOT_SPAM
# item_3: 2/3 agree → majority vote = SPAM (flagged for review)
```

Typical redundancy: 2-5 annotators per item

5. Monitoring: Track Quality Over Time

ANNOTATOR QUALITY DASHBOARD

Annotator	Gold Acc.	IAA	Speed	Status
Alice	95%	0.85	120/hr	Good
Bob	92%	0.82	145/hr	Good
Charlie	78%	0.65	180/hr	REVIEW NEEDED
Diana	89%	0.78	100/hr	OK

Overall IAA: 0.81 (Substantial)
Items completed: 4,523 / 10,000

6. Adjudication

When annotators disagree, who decides?

Option 1: Majority Vote

```python
labels = ["spam", "spam", "not_spam"]
final = max(set(labels), key=labels.count)  # "spam"
```

Option 2: Expert Adjudication

```python
if len(set(labels)) > 1:  # Disagreement
    final = expert_decides(item, labels)
```

Option 3: Weighted Vote

```python
# Weight by annotator historical accuracy
weights = [0.95, 0.88, 0.75]  # Alice, Bob, Charlie
final = weighted_vote(labels, weights)
```

The Quality Control Workflow

1. PILOT (50-100 items)

- Multiple annotators label same items
- Calculate IAA (target: kappa > 0.8)
- Discuss disagreements, refine guidelines

2. PRODUCTION

- Label large batch
- 10-20% overlap for ongoing IAA
- Gold standard checks (10% of items)

3. REVIEW

- Spot check 5-10% of labels
- Identify problematic annotators

Part 7: Cost Estimation

Planning your annotation budget

Cost Factors

Factor	Low Cost	High Cost
Task Complexity	Classification: $0.01-0.05/item	Segmentation: $1-5/item
Quality	Single annotator: 1x	3x redundancy: 3x
Domain	General: $10-20/hr	Expert: $50-200/hr
Platform	MTurk: variable	Scale AI: premium

Key insight: Cost = Complexity x Redundancy x Expertise

Cost Example: Image Object Detection

Project: Label 10,000 images with bounding boxes (5 objects/image)

Scenario 1: In-house team

```
images = 10000
time_per_image = 3  # minutes
hourly_rate = 25    # USD

total_hours = (images * time_per_image) / 60  # 500 hours
total_cost = total_hours * hourly_rate        # $12,500
```

Scenario 2: Crowdsourcing (MTurk)

```
cost_per_image = 0.30    # USD
redundancy = 3           # annotators per image

total_cost = images * cost_per_image * redundancy  # $9,000
```

Cost Example (continued)

Scenario 3: Professional service (Scale AI)

```
cost_per_image = 1.50     # USD (high quality)
redundancy = 1            # They handle QC internally

total_cost = images * cost_per_image  # $15,000
```

Trade-off Summary:

Approach	Cost	Quality	Speed
In-house	$12,500	High (control)	Slow
MTurk	$9,000	Variable	Fast
Scale AI	$15,000	High (guaranteed)	Medium

Budget Planning Formula

```python
def estimate_annotation_budget(n_items, complexity, quality, domain):
    # Base rates per item (USD)
    base_rates = {"simple": 0.05, "medium": 0.30, "complex": 2.00}

    # Quality multipliers (redundancy factor)
    quality_mult = {"low": 1, "medium": 2, "high": 3}

    # Domain expertise multipliers
    domain_mult = {"general": 1, "expert": 5}

    cost = n_items * base_rates[complexity] * quality_mult[quality] \
           * domain_mult[domain]
    return cost
```

Budget Planning: Example

```python
# Project: 10,000 items, medium complexity, high quality, general domain
budget = estimate_annotation_budget(
    n_items=10000,
    complexity="medium",
    quality="high",
    domain="general"
)
print(f"Estimated budget: ${budget:,.0f}")  # $9,000

# Same project with expert annotators (medical/legal)
expert_budget = estimate_annotation_budget(10000, "medium", "high", "expert")
print(f"Expert budget: ${expert_budget:,.0f}")  # $45,000
```

Formula: `items * base_rate * redundancy * expertise`

Part 8: Managing Annotation Teams

People, not just tools

Team Structures: Small & Medium Projects

```
SMALL PROJECT (<1000 items)

    ┌──────────┐
    │   You    │  →  Do it yourself or with 1-2 helpers
    └──────────┘


MEDIUM PROJECT (1k-10k items)

    ┌──────────┐
    │  Lead    │  →  1 lead + 3-5 annotators
    └────┬─────┘
         │
    ┌────┼────┐
    v    v    v
  [A1] [A2] [A3]
```

Team Structures: Large Projects

```
LARGE PROJECT (>10k items)

          ┌─────────────────┐
          │  Project Lead   │
          └────────┬────────┘
                   │
     ┌─────────────┼─────────────┐
     v             v             v
 [QC Lead]     [QC Lead]     [QC Lead]
     |             |             |
     v             v             v
 [Team A]      [Team B]      [Team C]
 (5 ann)       (5 ann)       (5 ann)
```

Roles:

- **Project Lead**: Guidelines, training, escalations
- **QC Lead**: Monitor quality, adjudicate, retrain
- **Annotators**: Label items according to guidelines

Annotator Selection

What to look for:

1. **Attention to detail** - Test with ambiguous examples
2. **Consistency** - Same answer for same item on different days
3. **Speed vs Quality balance** - Not too fast, not too slow
4. **Communication** - Asks questions when unsure
5. **Domain knowledge** (if needed) - Medical, legal, technical

Red flags:

- Suspiciously fast completion
- Random-looking answers on gold questions
- Never asks clarifying questions
- Quality degrades over time

Common Annotator Issues

Issue 1: Fatigue

```
Quality degrades over long sessions

Solution:
- Limit sessions to 2-3 hours
- Insert mandatory breaks
```

Issue 2: Anchoring Bias

```
First few items influence later decisions

Solution:
- Randomize order
```

Issue 3: Label Leakage

```
Annotator sees previous labels or predictions

Solution:
- Hide previous annotations
```

Annotator Feedback Loop

```
┌──────────────┐
│  Annotator   │   ←─ Individual feedback on quality
│   Labels     │
└──────┬───────┘
       │
       v
┌──────────────┐      ┌──────────────┐
│   Quality    │ ──→  │   Feedback   │  ──→ Weekly quality reports
│   Check      │      │   to Team    │
└──────┬───────┘      └──────────────┘
       │
       v
┌──────────────┐
│   Update     │   ──→ FAQ, edge case clarifications
│  Guidelines  │
└──────────────┘
```

Continuous improvement through regular feedback!

Part 9: Key Takeaways

Interview Questions

Common interview questions on data labeling:

1. **"How would you ensure label quality in a large annotation project?"**
    - Use multiple annotators (redundancy)
    - Calculate inter-annotator agreement (Cohen's Kappa)
    - Include gold standard questions for monitoring
    - Write clear guidelines with edge cases
    - Run calibration sessions before production
2. **"What is Cohen's Kappa and why use it instead of percent agreement?"**
    - Kappa accounts for chance agreement
    - Two random guessers on binary task agree 50% by chance
    - Kappa measures how much better than chance your agreement is
    - Target: Kappa >= 0.8 for production-ready labels

Key Takeaways

1. **Labels are the bottleneck** - 80% of AI project time
2. **Different tasks, different challenges** - NER != Classification != Segmentation
3. **Start with Label Studio** - Free, flexible, supports all modalities
4. **Measure agreement** - Cohen's Kappa >= 0.8 before production
5. **Invest in guidelines** - Decision trees, examples, edge cases
6. **Quality control is ongoing** - Gold questions, redundancy, monitoring
7. **Budget realistically** - complexity * redundancy * domain expertise

Part 10: Lab Preview

What you'll build today

This Week's Lab

Hands-on Practice:

1. **Install Label Studio** - Set up local annotation server
2. **Create annotation project** - Configure for sentiment analysis
3. **Write guidelines** - Clear definitions and examples
4. **Label data** - Annotate 30 movie reviews
5. **Calculate IAA** - Measure Cohen's Kappa with a partner
6. **Discuss disagreements** - Refine guidelines based on edge cases

Lab Setup Preview

```bash
# Install Label Studio
pip install label-studio

# Start the server
label-studio start

# Access at http://localhost:8080
```

You'll create a sentiment analysis project and experience the full annotation workflow!

Next Week Preview

Week 4: Optimizing Labeling

- Active Learning - smart sampling to reduce labeling effort
- Weak Supervision - programmatic labeling with rules
- LLM-based labeling - using GPT/Claude as annotators
- Noisy label detection and handling

The techniques to make labeling 10x more efficient!

Resources

Tools:

- Label Studio: https://labelstud.io/
- CVAT: https://cvat.ai/
- Prodigy: https://prodi.gy/

Metrics:

- sklearn.metrics.cohen_kappa_score
- statsmodels fleiss_kappa

Reading:

- Annotation guidelines (Google, Amazon, Microsoft examples online)

Questions?

Thank You!

See you in the lab!