Data Labeling & Annotation

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

The Labeling Bottleneck

The Reality:

- Data is abundant (unlabeled).
- Labels are scarce (expensive).
- 80% of AI project time is Data Prep.

Why Labeling Matters:

- **Supervised Learning**: Needs ground truth (y).
- **Evaluation**: Even unsupervised methods need a test set to verify.
- **Ambiguity**: Labeling forces you to define your problem clearly.

Types of Annotation: Overview

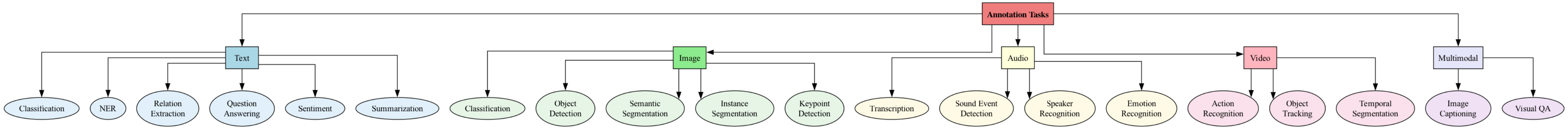
By Modality:

- **Text**: Classification, NER, QA, Summarization
- **Images**: Classification, Detection, Segmentation, Keypoints
- **Audio**: Transcription, Speaker ID, Event Detection
- **Video**: Action Recognition, Tracking, Temporal Segmentation
- **Multimodal**: Image Captioning, VQA, Audio-Visual

By Task Complexity:

- **Simple**: Binary classification (spam/not spam)
- **Medium**: Multi-class, bounding boxes
- **Complex**: Segmentation, relationship extraction

Annotation Taxonomy



We'll cover:

1. Text annotation tasks (6 types)
2. Image annotation tasks (5 types)
3. Audio annotation tasks (4 types)
4. Video annotation tasks (3 types)
5. Metrics for each task type

Text Annotation: Classification

Task: Assign one or more labels to entire text.

Examples:

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

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

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

Annotation Interface:

- Radio buttons (single-label)

Text Classification: Metrics

Inter-Annotator Agreement:

- **Cohen's Kappa**: Binary/multi-class
- **Fleiss' Kappa**: Multiple annotators
- **Krippendorff's Alpha**: General case

Model Evaluation:

```
from sklearn.metrics import classification_report, confusion_matrix

# Metrics per class
print(classification_report(y_true, y_pred,
                            labels=['POSITIVE', 'NEGATIVE', 'NEUTRAL']))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Shows which classes are confused
```

Text Annotation: Named Entity Recognition (NER)

Task: Identify and classify spans of text.

Example:

```
Text: "Apple CEO Tim Cook announced iPhone 15 in Cupertino on Sep 12."

Entities:
- "Apple" [0:5] → ORGANIZATION
- "Tim Cook" [10:18] → PERSON
- "iPhone 15" [29:38] → PRODUCT
- "Cupertino" [42:51] → LOCATION
- "Sep 12" [55:61] → DATE
```

Label Format (JSON):

```
{
  "text": "Apple CEO Tim Cook...",
  "entities": [
```

NER: Annotation Challenges

1. **Boundary Ambiguity**:

```
"New York City Mayor"
Should we tag "New York" or "New York City"?
```

2. **Nested Entities**:

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

3. **Discontinuous Entities**:

```
"Pick the phone up"
→ "Pick ... up" is a phrasal verb
```


Solution: Clear guidelines with examples

NER: Metrics

Span-Level Metrics (exact match):

```
from seqeval.metrics import classification_report

# Format: List of token-level tags
y_true = [['B-PER', 'I-PER', 'O', 'B-LOC']]
y_pred = [['B-PER', 'I-PER', 'O', 'O']]

print(classification_report(y_true, y_pred))
# Output: Precision, Recall, F1 per entity type
```

Token-Level vs Span-Level:

- **Token-Level**: Each word tagged correctly (lenient)
- **Span-Level**: Entire entity span must match (strict)

Example:

Text Annotation: Relation Extraction

Task: Identify relationships between entities.

Example:

```
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)

Relation Extraction: Metrics

Evaluation:

```
# Relation is correct if:
# 1. Both entities are correct (exact span match)
# 2. Relation type is correct

def evaluate_relations(gold, pred):
    tp = 0  # True positives
    fp = 0  # False positives
    fn = 0  # False negatives

    for rel in pred:
        if rel in gold:
            tp += 1
        else:
            fp += 1

    fn = len(gold) - tp

    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
```

Text Annotation: Question Answering

Task: Find answer span in passage given question.

Example (SQuAD format):

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

Challenges:

- **Extractive QA**: Answer is substring of context
- **Abstractive QA**: Answer must be generated
- **Unanswerable**: Question has no answer in context

QA Annotation: Metrics

Extractive QA Metrics:

Exact Match (EM):

```
def exact_match(pred, gold):
    """Strict: pred must exactly match at least one gold answer."""
    return int(normalize(pred) in [normalize(g) for g in gold])

def normalize(text):
    """Remove articles, punctuation, fix whitespace."""
    return ' '.join(text.lower().split())
```

F1 Score (token overlap):

```
def f1_score(pred, gold):
    """Partial credit for word overlap."""
    pred_tokens = normalize(pred).split()
    gold_tokens = normalize(gold).split()
```

Text Annotation: Sentiment Analysis

Task: Classify opinion/emotion in text.

Levels of Granularity:

1. **Document-Level**:

```
{"text": "This phone is amazing!", "sentiment": "POSITIVE"}
```

2. **Sentence-Level**:

```
{
  "text": "Great camera. Poor battery.",
  "sentences": [
    {"text": "Great camera.", "sentiment": "POSITIVE"},
    {"text": "Poor battery.", "sentiment": "NEGATIVE"}
  ]
}
```

Sentiment Analysis: Guidelines

Common Issues:

1. Sarcasm:

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

2. Mixed Sentiment:

```
"Good product, terrible delivery"
→ Label as MIXED or separate aspects
```

3. Neutral vs No Opinion:

```
"The phone is blue"  → NEUTRAL (factual)
"I don't care"       → NEUTRAL (no opinion)
```

Text Annotation: Text Summarization

Task: Create concise summary of document.

Types:

- **Extractive**: Select important sentences
- **Abstractive**: Write new summary

Annotation Format (extractive):

```
{
  "document": "Long article with multiple paragraphs...",
  "summary_sentences": [0, 3, 7, 12],  # Sentence indices
  "rationale": "These sentences contain key points"
}
```

Annotation Format (abstractive):

Summarization: Quality Metrics

Automatic Metrics:

ROUGE (Recall-Oriented Understudy for Gisting Evaluation):

```
from rouge import Rouge

rouge = Rouge()
scores = rouge.get_scores(
    hyps=["the cat sat on the mat"],
    refs=["cat on mat"]
)

# Output: ROUGE-1, ROUGE-2, ROUGE-L
# Measures n-gram overlap
```

Human Evaluation Criteria:

1. **Relevance**: Contains important information

Image Annotation: Classification

Task: Assign label(s) to entire image.

Examples:

Single-Label:

```
{"image": "cat.jpg", "label": "CAT"}
```

Multi-Label:

```
{
  "image": "outdoor_scene.jpg",
  "labels": ["OUTDOOR", "PEOPLE", "DAYTIME", "URBAN"]
}
```

Fine-Grained:

Image Classification: Metrics

Inter-Annotator Agreement:

```python
from sklearn.metrics import cohen_kappa_score

# Two annotators label 100 images
annotator1 = ['cat', 'dog', 'cat', ...]  # 100 labels
annotator2 = ['cat', 'cat', 'cat', ...]  # 100 labels

kappa = cohen_kappa_score(annotator1, annotator2)
print(f"Cohen's Kappa: {kappa:.3f}")

# Interpretation:
# > 0.8: Excellent agreement
# 0.6-0.8: Substantial agreement
# < 0.6: Need to improve guidelines
```

Model Metrics: Accuracy, Top-5 Accuracy, Confusion Matrix

Calibration: Are model confidences reliable?

Image Annotation: Object Detection

Task: Locate and classify objects with bounding boxes.

Annotation Format:

```
{
  "image": "street.jpg",
  "width": 1920,
  "height": 1080,
  "objects": [
    {
      "class": "car",
      "bbox": [100, 200, 400, 500],  # [x, y, width, height]
      "confidence": 1.0
    },
    {
      "class": "person",
      "bbox": [800, 150, 120, 350]
    }
  ]
```

Object Detection: Common Issues

1. Bounding Box Tightness:

```
Too Loose:  [    |--object--|     ]
❌

Too Tight:  [--object--]
❌
  (cuts off parts)
Just Right: [  |--object--|  ]
✅
  (small margin)
```

2. Overlapping Objects:

```
Person behind car - label both?
→ Yes, even if heavily occluded (>20% visible)
```


3. Ambiguous Objects:

Object Detection: Metrics

Intersection over Union (IoU):

```python
def iou(box1, box2):
    """Compute IoU between two boxes [x, y, w, h]."""
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2

    # Intersection area
    xi = max(x1, x2)
    yi = max(y1, y2)
    wi = max(0, min(x1 + w1, x2 + w2) - xi)
    hi = max(0, min(y1 + h1, y2 + h2) - yi)
    intersection = wi * hi

    # Union area
    union = w1 * h1 + w2 * h2 - intersection

    return intersection / union if union > 0 else 0

# Example
box1 = [100, 100, 50, 50]  # Ground truth
```

Object Detection: mAP Metric

mean Average Precision (mAP):

Steps:

1. For each class, compute Average Precision (AP)
2. Average AP across all classes → mAP

AP Calculation:

```
# For each predicted box:
# - If IoU > threshold (e.g., 0.5) → True Positive
# - Otherwise → False Positive
# Plot Precision-Recall curve, compute area under curve

from sklearn.metrics import average_precision_score

# Sort predictions by confidence
predictions_sorted = sorted(predictions, key=lambda x: x['confidence'], reverse=True)
```

Image Annotation: Semantic Segmentation

Task: Classify every pixel in image.

Example:

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

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

Annotation Format:

```
{
  "image": "street.jpg",
```

Semantic Segmentation: Tools & Speed

Annotation Tools:

- **Polygon Tool**: Draw polygon around object
- **Brush Tool**: Paint over pixels
- **Magic Wand**: Auto-select similar pixels
- **SAM (Segment Anything)**: AI-assisted segmentation

Annotation Time:

```
Simple object (car):       2-5 minutes
Complex object (person):   5-10 minutes
Full street scene:         30-60 minutes
Medical scan:              1-4 hours (high precision required)
```

Speed-up Techniques:

Semantic Segmentation: Metrics

Pixel Accuracy:

```
correct_pixels = (pred_mask == true_mask).sum()
total_pixels = pred_mask.size
accuracy = correct_pixels / total_pixels
```

Problem: Dominated by large classes (e.g., background)

Intersection over Union (IoU) per class:

```
def iou_per_class(pred, true, class_id):
    pred_mask = (pred == class_id)
    true_mask = (true == class_id)

    intersection = (pred_mask & true_mask).sum()
    union = (pred_mask | true_mask).sum()

    return intersection / union if union > 0 else 0
```

Image Annotation: Instance Segmentation

Task: Segment and identify each object instance.

Difference from Semantic Segmentation:

```
Semantic: All "person" pixels labeled as 1
Instance: Person #1 → 1, Person #2 → 2, Person #3 → 3
```

Annotation Format (COCO style):

```
{
  "image": "crowd.jpg",
  "annotations": [
    {
      "id": 1,
      "category_id": 1,  # Person
      "segmentation": [[x1, y1, x2, y2, ...]],  # Polygon points
      "bbox": [100, 200, 50, 80],
      "area": 4000
```

Instance Segmentation: Metrics

Mask AP (Average Precision):

- Similar to object detection mAP
- But uses mask IoU instead of bbox IoU

```
def mask_iou(mask1, mask2):
    """Compute IoU between two binary masks."""
    intersection = np.logical_and(mask1, mask2).sum()
    union = np.logical_or(mask1, mask2).sum()
    return intersection / union if union > 0 else 0

# Match predicted instances to ground truth
# Using Hungarian algorithm (bipartite matching)
from scipy.optimize import linear_sum_assignment

# Compute cost matrix (1 - IoU for each pred-gt pair)
cost_matrix = np.zeros((len(pred_instances), len(gt_instances)))
for i, pred in enumerate(pred_instances):
```

Image Annotation: Keypoint Detection

Task: Locate specific points on objects (e.g., body joints, facial landmarks).

Example (Human Pose):

```
{
  "image": "person.jpg",
  "keypoints": [
    {"name": "nose", "x": 120, "y": 80, "visible": 1},
    {"name": "left_eye", "x": 110, "y": 75, "visible": 1},
    {"name": "right_eye", "x": 130, "y": 75, "visible": 1},
    {"name": "left_shoulder", "x": 100, "y": 150, "visible": 1},
    {"name": "right_shoulder", "x": 140, "y": 150, "visible": 0}  # occluded
  ],
  "skeleton": [[0, 1], [0, 2], [0, 3], [0, 4]]  # Connections
}
```

Visibility Flags:

- `0`: Not visible (occluded)

Keypoint Detection: Metrics

Object Keypoint Similarity (OKS):

```
def oks(pred_kpts, gt_kpts, bbox_area, kpt_sigmas):
    """
    Similar to IoU but for keypoints.

    Args:
        pred_kpts: Predicted keypoints [(x, y, v), ...]
        gt_kpts: Ground truth keypoints [(x, y, v), ...]
        bbox_area: Bounding box area (for normalization)
        kpt_sigmas: Per-keypoint standard deviation (how precise to be)
    """
    distances = []
    for (px, py, pv), (gx, gy, gv), sigma in zip(pred_kpts, gt_kpts, kpt_sigmas):
        if gv == 0:  # Skip if not labeled
            continue

        # Euclidean distance
        d = np.sqrt((px - gx)**2 + (py - gy)**2)

        # Normalized by bbox size and keypoint precision
        distances.append(np.exp(-(d**2) / (2 * bbox_area * sigma**2)))
```

Audio Annotation: Speech Transcription

Task: Convert speech to text.

Annotation Format:

```
{
  "audio": "interview.wav",
  "duration": 120.5,
  "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."
```

Speech Transcription: Challenges

1. Speaker Diarization:

```
"Who is speaking when?"
→ Label each segment with speaker ID
```

2. Overlapping Speech:

```
[0.0–2.0] Speaker A: "I think that—"
[1.5–3.0] Speaker B: "No wait, listen..."
→ Overlap at 1.5–2.0
```

3. Accents & Dialects:

```
Non-standard pronunciation → Transcribe what was said, not standard form
Example: "gonna" vs "going to"
```


4. Code-Switching:

Speech Transcription: Metrics

Word Error Rate (WER):

```python
from jiwer import wer

reference = "hello world how are you"
hypothesis = "hello word how you"

error_rate = wer(reference, hypothesis)
print(f"WER: {error_rate:.2%}")  # 40% (2 errors / 5 words)

# WER = (Substitutions + Deletions + Insertions) / Total Words
```

Character Error Rate (CER):

- Similar to WER but at character level
- Better for languages without clear word boundaries

Audio Annotation: Sound Event Detection

Task: Identify and timestamp sound events.

Example (Smart home):

```
{
  "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"},
    {"start": 15.0, "end": 45.0, "label": "music", "overlap": true}
  ]
}
```

Applications:

- **Surveillance**: Gunshot, glass break, scream

Sound Event Detection: Metrics

Event-Based Metrics:

1. Onset/Offset tolerance:

```
def event_based_f1(pred_events, true_events, tolerance=0.2):
    """
    Allow tolerance in start/end times.

    Args:
        tolerance: Allowed time difference (seconds)
    """
    tp = 0
    for pred in pred_events:
        for true in true_events:
            # Check label match
            if pred['label'] != true['label']:
                continue

            # Check temporal overlap within tolerance
            if (abs(pred['start'] - true['start']) < tolerance and
                abs(pred['end'] - true['end']) < tolerance):
                tp += 1
                break
```

Audio Annotation: Speaker Recognition

Task: Identify who is speaking.

Types:

1. **Speaker Identification** (closed-set):

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

2. **Speaker Verification** (is this person X?):

```
{
```

Audio Annotation: Emotion Recognition

Task: Classify emotion in speech.

Labels (common taxonomy):

```
emotions = [
    "neutral",
    "happy",
    "sad",
    "angry",
    "fearful",
    "surprised",
    "disgusted"
]
```

Annotation Format:

```
{
  "audio": "utterance.wav",
```

Video Annotation: Action Recognition

Task: Classify actions in video clips.

Annotation Format:

```
{
  "video": "sports.mp4",
  "fps": 30,
  "actions": [
    {
      "start_frame": 0,
      "end_frame": 90,  # 3 seconds at 30fps
      "label": "running"
    },
    {
      "start_frame": 90,
      "end_frame": 150,
      "label": "jumping"
    }
  ]
```

Video Annotation: Object Tracking

Task: Follow objects across frames.

Annotation Format (MOT - Multiple Object Tracking):

```
{
  "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]}
      ]
    },
    {
      "track_id": 2,
      "category": "person",
```

Video Tracking: Metrics

CLEAR MOT Metrics:

1. **MOTA (Multiple Object Tracking Accuracy)**:

```
MOTA = 1 - (FP + FN + IDSW) / GT

where:
- FP: False positives (wrong detections)
- FN: False negatives (missed objects)
- IDSW: ID switches (track lost then found with new ID)
- GT: Total ground truth objects
```

2. **MOTP (Multiple Object Tracking Precision)**:

```
MOTP = Σ IoU(matched_pairs) / |matched_pairs|
# Average IoU of correctly matched detections
```

3. **IDF1** (ID F1 Score):

Video Annotation: Temporal Segmentation

Task: Divide video into meaningful segments.

Example (Cooking video):

```
{
  "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**: Play-by-play analysis
- **Surveillance**: Activity monitoring

Multimodal Annotation: Image Captioning

Task: Generate textual description of image.

Annotation Format:

```json
{
  "image": "beach.jpg",
  "captions": [
    "A person walking on the beach at sunset",
    "Someone enjoying a peaceful evening walk by the ocean",
    "A solitary figure strolls along the sandy shore"
  ]
}
```

Why Multiple Captions?

- Captures different aspects
- Handles ambiguity

Multimodal Annotation: Visual Question Answering

Task: Answer questions about images.

Example:

```
{
  "image": "kitchen.jpg",
  "qa_pairs": [
    {"question": "How many people are in the image?", "answer": "2"},
    {"question": "What color is the refrigerator?", "answer": "white"},
    {"question": "Are they cooking?", "answer": "yes"}
  ]
}
```

Answer Types:

- **Number**: Counting questions
- **Yes/No**: Binary questions

Annotation Metrics: Summary by Task

Task	Primary Metric	IAA Metric	Typical IAA
Text Classification	F1-Score	Cohen's κ	0.7-0.9
NER	Span F1	Entity-level κ	0.6-0.8
Object Detection	mAP@0.5	Mean IoU	> 0.7
Segmentation	mIoU	Pixel agreement	> 0.8
Keypoints	OKS	Mean distance	< 5 pixels
Speech Transcription	WER	WER between annotators	< 5%
Sound Events	Event F1	Temporal IoU	> 0.7
Action Recognition	Clip accuracy	Temporal IoU	0.6-0.8
Object Tracking	MOTA/IDF1	Track IoU	> 0.6

Annotation Speed: Benchmarks

Text (per hour):

- Classification: 200-500 examples
- NER: 50-150 sentences
- Relation extraction: 20-50 documents

Image (per hour):

- Classification: 100-300 images
- Bounding boxes: 20-50 images (5-10 objects each)
- Segmentation: 5-15 images (high complexity)

Audio (per hour of annotation work):

- Transcription: 15-30 min of audio

Cost Estimation for Annotation

Example Project: Label 10,000 images for object detection

Scenario 1: In-house team

```
images = 10000
time_per_image = 3  # minutes
hourly_rate = 20  # USD

total_hours = (images * time_per_image) / 60  # 500 hours
total_cost = total_hours * hourly_rate  # $10,000
```

Scenario 2: Crowdsourcing (MTurk)

```
cost_per_image = 0.50  # USD (cheaper but lower quality)
redundancy = 3  # annotations per image

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

Annotation Best Practices by Modality

Text:

- Provide clear label definitions with examples
- Handle edge cases explicitly in guidelines
- Use consensus for ambiguous cases

Images:

- Zoom tools for precise boundaries
- Grid overlay for alignment
- Pre-annotation with models to speed up

Audio:

- High-quality headphones required

Annotation Interfaces: Label Studio

Label Studio: Open-source, flexible, web-based tool.

Configuration (XML):

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

Why usage XML?

- Allows custom UI layouts.
- Can mix modalities (Image + Text + Audio).

Quality Control: The Gold Standard

How do we know if labels are "correct"?

1. **Gold Standard (Ground Truth)**:
   - Experts label a small subset (e.g., 100 items).
   - Annotators are tested against this set.
2. **Consensus (Majority Vote)**:
   - 3 annotators label the same item.
   - If 2 say "Cat" and 1 says "Dog", label is "Cat".

Inter-Annotator Agreement (IAA)

Measure of reliability. Do annotators agree with each other?

Percent Agreement:

$$ext{Agreement} = \frac{\text{Agreed Items}}{\text{Total Items}}$$

Problem: Doesn't account for chance agreement.

Cohen's Kappa (κ):

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

- p_o: Observed agreement.
- p_e: Expected agreement by chance.

Cohen's Kappa Example

Scenario: 2 Annotators, 100 items (Yes/No).

- Both say Yes: 45
- Both say No: 45
- Disagree: 10
- $p_o = 0.90$

Chance Calculation:

- A says Yes 50% of time.
- B says Yes 50% of time.
- Chance they agree on Yes = $0.5 \times 0.5 = 0.25$.
- Chance they agree on No = 0.25.

Managing Labeling Teams

Workflow:

1. **Guidelines**: Write detailed instructions (e.g., "Does a reflection count as a car?").
2. **Pilot**: Label 50 items, calculate Kappa. Refine guidelines.
3. **Production**: Label large batch.
4. **Review**: Spot check 10% of labels.

Human-in-the-Loop:

- Use Model to pre-label (Predictions).
- Humans verify/correct (faster than starting from scratch).

Active Learning: Smart Sampling

Problem: Labeling all data is expensive.

Solution: Label only the most informative examples.

Uncertainty Sampling:

```python
def uncertainty_sampling(model, unlabeled_pool, n=100):
    """Select examples where model is least confident."""
    predictions = model.predict_proba(unlabeled_pool)

    # Entropy-based uncertainty
    entropy = -np.sum(predictions * np.log(predictions + 1e-10), axis=1)

    # Select top-n most uncertain
    top_indices = np.argsort(entropy)[-n:]

    return unlabeled_pool[top_indices]
```


Benefit: Can achieve 90% accuracy with 20-30% of labeled data

Active Learning Strategies

1. **Uncertainty Sampling**:
    - **Least Confidence**: $1 - P(\hat{y}|x)$
    - **Margin**: $P(y_1|x) - P(y_2|x)$ (difference between top 2)
    - **Entropy**: $-\sum P(y|x) \log P(y|x)$
2. **Query-by-Committee**:
    - Train ensemble of models
    - Select examples with highest disagreement
3. **Expected Model Change**:
    - Select examples that would change model most if labeled
4. **Diversity Sampling**:

Active Learning Implementation

```
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import entropy

class ActiveLearner:
    def __init__(self, model, X_pool, y_pool=None):
        self.model = model
        self.X_pool = X_pool
        self.y_pool = y_pool
        self.X_labeled = []
        self.y_labeled = []

    def query(self, n_samples=10, strategy='entropy'):
        """Query most informative samples."""
        if strategy == 'entropy':
            probs = self.model.predict_proba(self.X_pool)
            scores = entropy(probs.T)
        elif strategy == 'margin':
            probs = self.model.predict_proba(self.X_pool)
            probs_sorted = np.sort(probs, axis=1)
            scores = probs_sorted[:, -1] - probs_sorted[:, -2]

        # Select top-n uncertain samples
        query_idx = np.argsort(scores)[-n_samples:]
        return query_idx

    def teach(self, idx, labels):
        """Add labeled samples to training set."""
        self.X_labeled.extend(self.X_pool[idx])
        self.y_labeled.extend(labels)

        # Remove from pool
        self.X_pool = np.delete(self.X_pool, idx, axis=0)

        # Retrain model
        self.model.fit(self.X_labeled, self.y_labeled)
```

Weak Supervision: Programmatic Labeling

Idea: Write labeling functions instead of manually labeling.

Example (Spam detection):

```
# Labeling Function 1: Check for money mentions
def lf_contains_money(text):
    if re.search(r'\$\d+|\bmoney\b', text, re.I):
        return 1  # SPAM
    return -1  # NOT_SPAM

# Labeling Function 2: All caps
def lf_all_caps(text):
    if text.isupper() and len(text) > 10:
        return 1  # SPAM
    return -1  # NOT_SPAM

# Labeling Function 3: Urgency words
def lf_urgency(text):
    urgent_words = ['urgent', 'act now', 'limited time']
```

Snorkel Framework

Snorkel: Framework for weak supervision.

Workflow:

1. Write labeling functions (LFs)
2. Apply LFs to unlabeled data
3. Learn generative model to denoise labels
4. Train final classifier on probabilistic labels

```python
from snorkel.labeling import labeling_function, PandasLFApplier
from snorkel.labeling.model import LabelModel

@labeling_function()
def lf_keyword_spam(x):
    keywords = ['free', 'win', 'click here']
    return 1 if any(k in x.text.lower() for k in keywords) else -1
```

Weak Supervision: Benefits and Challenges

Benefits:

- **Scalability**: Label thousands of examples in minutes
- **Flexibility**: Encode domain knowledge
- **Iteration**: Easy to update rules vs re-labeling
- **Cost**: Much cheaper than manual annotation

Challenges:

- **Quality**: LFs may be noisy or conflicting
- **Coverage**: LFs may abstain on many examples
- **Bias**: Rules encode human biases
- **Complexity**: Need to balance precision vs coverage

Semi-Supervised Learning

Setting: Small labeled set + Large unlabeled set.

Self-Training:

1. Train model on labeled data
2. Predict on unlabeled data
3. Add high-confidence predictions to labeled set
4. Repeat

```
def self_training(model, X_labeled, y_labeled, X_unlabeled, threshold=0.9):
    """Iterative self-training."""
    for iteration in range(10):
        # Train on current labeled set
        model.fit(X_labeled, y_labeled)

        # Predict on unlabeled
        probs = model.predict_proba(X_unlabeled)
        max_probs = np.max(probs, axis=1)
        preds = np.argmax(probs, axis=1)
```

Co-Training: Multi-View Learning

Idea: Use different views of same data.

Example (Web page classification):

- View 1: Page text
- View 2: Anchor text from links to page

Algorithm:

1. Train classifier on each view independently
2. Each classifier labels unlabeled data
3. Add high-confidence predictions from one view to other view's training set

```
def co_training(X1, X2, y, X1_unlabeled, X2_unlabeled, n_iter=10):
    """Co-training with two views."""
    model1 = RandomForestClassifier()
    model2 = RandomForestClassifier()
```

Crowdsourcing Platforms

Major Platforms:

Platform	Use Case	Cost	Quality
Amazon MTurk	Generic tasks	Low	Variable
Scale AI	High-quality CV/NLP	High	High
Labelbox	Enterprise labeling	Medium	Medium-High
Hive	Image/video annotation	Medium	High
Appen	Multilingual, global	Medium	Medium

Key Considerations:

- **Task complexity**: Simple (MTurk) vs Expert (Scale AI)
- **Quality control**: Gold standard questions, redundancy

Crowdsourcing: Quality Control

Challenge: Workers may be inattentive or malicious.

Solutions:

1. **Gold Standard Questions** (Test questions):

```
# Mix 10% gold questions into tasks
gold_questions = [
    {"text": "The sky is blue", "label": "POSITIVE"},  # Known
]

# Check worker accuracy on gold questions
if worker_accuracy < 0.8:
    reject_worker()
```

2. **Redundancy** (Multiple workers per item):

- Assign same task to 3-5 workers

Advanced IAA: Fleiss' Kappa

Fleiss' Kappa: Extends Cohen's Kappa to >2 annotators.

Formula:

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e}$$

Where:

- $\bar{P}$: Overall agreement
- $\bar{P}_e$: Expected agreement by chance

Implementation:

```
from statsmodels.stats.inter_rater import fleiss_kappa

# Data format: (n_items, n_categories)
```

Krippendorff's Alpha

Most general IAA metric:

- Works with any number of annotators
- Handles missing data
- Works with different data types (nominal, ordinal, interval, ratio)

Formula:

$$\alpha = 1 - \frac{D_o}{D_e}$$

Where:

- D_o: Observed disagreement
- D_e: Expected disagreement by chance

Labeling Bias: Types and Impact

Types of Bias:

1. **Selection Bias**:
   - Annotators label non-representative subset
   - Example: Only labeling easy cases
2. **Confirmation Bias**:
   - Annotators favor pre-existing beliefs
   - Example: Political bias in sentiment labeling
3. **Anchoring Bias**:
   - First label influences subsequent labels
   - Example: Seeing "spam" first makes next emails look more like spam

Mitigating Labeling Bias

Strategies:

1. **Blind Annotation**:

```
# Don't show annotators previous labels or predictions
# Randomize order to prevent patterns
def randomize_annotation_order(tasks):
    return random.sample(tasks, len(tasks))
```

2. **Calibration Sessions**:

- Train annotators together
- Discuss edge cases and establish consensus

3. **Regular Audits**:

```
def audit_annotator_quality(annotations, gold_standard):
```

Label Noise: Detection and Handling

Sources of Noise:

- Genuine ambiguity in data
- Annotator mistakes
- Poorly defined guidelines

Confident Learning (cleanlab):

```
from cleanlab.classification import CleanLearning

# Wrap any sklearn classifier
cl = CleanLearning(RandomForestClassifier())

# Automatically identify and remove noisy labels
cl.fit(X_train, y_train)

# Get indices of likely label errors
```

Consensus Mechanisms: Dawid-Skene Model

Problem: Annotators have different error rates.

Dawid-Skene Model:

- Probabilistic model for aggregating labels
- Learns confusion matrix per annotator
- Estimates true labels given noisy annotations

```
from crowdkit.aggregation import DawidSkene

# Data format: DataFrame with columns ['task', 'worker', 'label']
annotations = pd.DataFrame([
    {'task': 1, 'worker': 'A', 'label': 'spam'},
    {'task': 1, 'worker': 'B', 'label': 'spam'},
    {'task': 1, 'worker': 'C', 'label': 'not_spam'},
    {'task': 2, 'worker': 'A', 'label': 'spam'},
    {'task': 2, 'worker': 'B', 'label': 'not_spam'},
```

Cost-Quality Tradeoffs

Annotation Budget: B dollars

Decision Variables:

- n: Number of samples to label
- k: Annotators per sample
- c: Cost per annotation

Constraint: $n \times k \times c \leq B$

Quality vs Quantity:

- **High n, low k**: More data, less reliable
- **Low n, high k**: Less data, more reliable

Optimal strategy depends on task:

Annotation Guidelines: Best Practices

Key Elements:

1. Clear Definitions:

```
# Spam Classification Guidelines

## Definition
Spam: Unsolicited commercial email sent in bulk.

## Examples
✅
 Spam: "Buy cheap meds now! Click here!"
❌
 Not Spam: Newsletter you subscribed to
⚠️
 Edge case: Promotional email from company you bought from
```

2. Decision Trees:

Measuring Annotation Productivity

Metrics:

1. **Throughput**:

```
throughput = annotations_completed / time_spent_hours
# Target: 50-100 simple labels/hour
```

2. **Learning Curve**:

```
def plot_learning_curve(annotations_df):
    """Plot annotator speed over time."""
    annotations_df['time_per_label'] = (
        annotations_df.groupby('annotator')['timestamp']
        .diff()
        .dt.total_seconds()
    )

    # Group by batch
```

Data Programming: Advanced Patterns

Labeling Function Patterns:

1. **Pattern Matching:**

```
@labeling_function()
def lf_regex_email(x):
    """Detect emails in text."""
    pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    return 1 if re.search(pattern, x.text) else -1
```

2. **External Knowledge:**

```
from nltk.corpus import wordnet

@labeling_function()
def lf_wordnet_positive(x):
    """Use WordNet to detect positive sentiment."""
    positive_words = set(['good', 'great', 'excellent', 'amazing'])
```

Few-Shot Learning for Labeling

Goal: Train model with very few examples (5-10 per class).

Meta-Learning Approach:

```
from sentence_transformers import SentenceTransformer

# Few-shot classifier using embeddings
class FewShotClassifier:
    def __init__(self, model_name='all-MiniLM-L6-v2'):
        self.model = SentenceTransformer(model_name)
        self.support_embeddings = None
        self.support_labels = None

    def fit(self, support_texts, support_labels):
        """Train on few examples."""
        self.support_embeddings = self.model.encode(support_texts)
        self.support_labels = np.array(support_labels)

    def predict(self, query_texts, k=3):
        """Predict using k-nearest neighbors."""
        query_embeddings = self.model.encode(query_texts)

        predictions = []
        for query_emb in query_embeddings:
            # Compute cosine similarity
            similarities = np.dot(self.support_embeddings, query_emb) / (
                np.linalg.norm(self.support_embeddings, axis=1) *
                np.linalg.norm(query_emb)
            )

            # Get top-k neighbors
            top_k_idx = np.argsort(similarities)[-k:]
```

Prompt-Based Labeling with LLMs

Modern Approach: Use GPT-4 or Claude for labeling.

```
import anthropic

def llm_label(text, task_description, examples):
    """Use LLM for zero/few-shot labeling."""
    client = anthropic.Anthropic()

    prompt = f"""Task: {task_description}

Examples:
{examples}

Now classify this text:
Text: {text}

Classification:"""

    message = client.messages.create(
        model="claude-3-5-sonnet-20241022",
        max_tokens=10,
        messages=[{"role": "user", "content": prompt}]
    )

    return message.content[0].text.strip()

# Example usage
task = "Classify sentiment as POSITIVE or NEGATIVE"
examples = """
Text: "I love this product!" → POSITIVE
Text: "Terrible experience." → NEGATIVE
"""
```

Synthetic Data Generation

Goal: Generate labeled data programmatically.

Text Augmentation:

```
import nlpaug.augmenter.word as naw

# Synonym replacement
aug_synonym = naw.SynonymAug(aug_src='wordnet')
text = "The movie was great"
augmented = aug_synonym.augment(text)
# Output: "The film was excellent"

# Back-translation
from googletrans import Translator
translator = Translator()

def back_translate(text, intermediate_lang='fr'):
    """Translate to intermediate language and back."""
    # English -> French
    translated = translator.translate(text, dest=intermediate_lang).text
    # French -> English
```

Transfer Learning to Reduce Labeling

Pre-trained Models: Already learned general features.

Fine-tuning Strategy:

```
from transformers import AutoModelForSequenceClassification, Trainer

# Load pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=2
)

# Fine-tune on small labeled set (100-1000 examples)
trainer = Trainer(
    model=model,
    train_dataset=small_labeled_dataset,
    eval_dataset=validation_dataset,
)
```

Annotation Tools Comparison

Tool	Type	Strengths	Weaknesses	Cost
Label Studio	Open-source	Flexible, customizable	Setup required	Free
CVAT	Open-source	Video annotation, tracking	CV-focused only	Free
Labelbox	Commercial	Enterprise features, QC	Expensive	$$$
V7	Commercial	Auto-annotation, versioning	Learning curve	$$$
Prodigy	Commercial	Active learning built-in	License per user	$$
Scale AI	Service	Full-service labeling	Least control	$$$$

Recommendation:

- **Prototyping**: Label Studio
- **Production CV**: CVAT or Labelbox

Multi-Task Annotation

Scenario: Annotate multiple attributes simultaneously.

Example (Product reviews):

```json
{
  "text": "Great phone but battery life is poor",
  "annotations": {
    "overall_sentiment": "MIXED",
    "aspects": [
      {"aspect": "phone", "sentiment": "POSITIVE"},
      {"aspect": "battery", "sentiment": "NEGATIVE"}
    ],
    "rating": 3,
    "would_recommend": false
  }
}
```


Benefits:

Handling Edge Cases and Ambiguity

Define Ambiguity Threshold:

```python
# In annotation interface, allow "UNCERTAIN" label
labels = ["POSITIVE", "NEGATIVE", "NEUTRAL", "UNCERTAIN"]

# Later, handle uncertain labels
def resolve_uncertain_labels(annotations):
    """Send uncertain cases to expert annotators."""
    uncertain = annotations[annotations['label'] == 'UNCERTAIN']

    # Send to expert pool
    expert_annotations = expert_annotate(uncertain)

    # Merge back
    annotations.loc[uncertain.index, 'label'] = expert_annotations

    return annotations
```


Escalation Protocol

Annotation Audit Trails

Track Everything:

```
annotation_log = {
    "id": "ann_12345",
    "task_id": "task_789",
    "annotator_id": "user_42",
    "timestamp": "2024-01-15T10:30:00Z",
    "label": "SPAM",
    "confidence": 0.9,  # Annotator confidence
    "time_spent_seconds": 12,
    "annotations_history": [  # If label was changed
        {"timestamp": "2024-01-15T10:29:50Z", "label": "NOT_SPAM"},
        {"timestamp": "2024-01-15T10:30:00Z", "label": "SPAM"}
    ],
    "notes": "Unclear sender but obvious commercial intent"
}
```

Benefits:

Label Taxonomy Design

Hierarchical Labels:

```
Product Review
├── Electronics
│   ├── Smartphones
│   │   ├── Android
│   │   └── iOS
│   └── Laptops
│       ├── Gaming
│       └── Business
└── Clothing
    ├── Men's
    └── Women's
```

Considerations:

- **Depth**: Too many levels = confusion
- **Balance**: Similar number of examples per category

Domain Adaptation for Labeling

Problem: Labels from one domain don't transfer well.

Example: Sentiment in product reviews vs movie reviews.

Solution: Active learning + Transfer learning.

```
# Train on source domain (movie reviews)
model.fit(X_source, y_source)

# Active learning on target domain (product reviews)
for iteration in range(10):
    # Select uncertain examples from target
    uncertain_idx = uncertainty_sampling(model, X_target_unlabeled)

    # Label selected examples
    y_selected = manual_label(X_target_unlabeled[uncertain_idx])

    # Add to target training set
    X_target_labeled = np.vstack([X_target_labeled,
```

Summary: Annotation Best Practices

Before Labeling:

1. Define clear taxonomy and guidelines
2. Set up quality control (gold standard, IAA)
3. Choose appropriate tools and platform
4. Budget allocation (n samples, k annotators)

During Labeling:

1. Monitor IAA and quality metrics
2. Regular calibration sessions
3. Handle edge cases and ambiguity
4. Track productivity and fatigue

Advanced Techniques Summary

Technique	Use Case	Effort Reduction
Active Learning	Limited budget	50-80%
Weak Supervision	Domain expertise available	70-90%
Semi-Supervised	Large unlabeled pool	40-60%
Few-Shot Learning	Very few examples	90-95%
Transfer Learning	Pre-trained models exist	80-90%
LLM Labeling	High budget, quality needed	60-80%
Synthetic Data	Augmentation tasks	30-50%

Combine techniques for maximum efficiency!

Lab Preview

Today's Goals:

1. **Install Label Studio**.
2. **Config**: Set up a Sentiment Analysis project.
3. **Label**: Annotate 10 examples.
4. **Export**: Get JSON/CSV data.
5. **Analysis**: Write a Python script to calculate Cohen's Kappa between two "simulated" annotators.

Let's start labeling!