Week 4: Optimizing the Labeling Process

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra
IIT Gandhinagar

Part 1: The Labeling Cost Problem

Why we need smarter approaches

Previously on CS 203...

Week 1: Collected movie data from APIs

Week 2: Validated and cleaned the data

Week 3: Learned how to label data with quality control

```
# We labeled 1,000 movies... but we need 100,000!
labeled_movies = 1000
total_needed = 100000
remaining = total_needed - labeled_movies  # 99,000 more!
```

Problem: At $0.30/movie and 5 min/movie, this would cost $30,000 and 8,333 hours!

The Labeling Bottleneck

```
     TRADITIONAL APPROACH

    +-----------------------------------------------------------+
    |                                                           |
    |   Unlabeled        Label            Labeled         Train |
    |     Data     -->   ALL of    -->     Data      --> Model  |
    |   (100,000)        them!           (100,000)              |
    |                                                           |
    |   Cost: $$$$$      Time: Months                           |
    |                                                           |
    +-----------------------------------------------------------+
```

Can we do better?

Three Strategies to Reduce Labeling Cost

ACTIVE LEARNING	WEAK SUPERVISION	LLM LABELING
Label SMARTER Pick the most informative examples	Label with CODE Write rules & heuristics	Label with AI Use GPT/Claude as annotators
Human labels fewer items	Noisy labels many items	AI labels many items

Today's Mission

Learn techniques to reduce labeling effort by 10x or more.

Technique	Effort Reduction	Best When
Active Learning	2-10x	Limited budget for human labels
Weak Supervision	10-100x	Patterns can be encoded as rules
LLM Labeling	10-50x	Task is well-defined, cost-sensitive
Noisy Label Handling	1.5-2x	Labels already exist but noisy

Part 2: Active Learning

Label smarter, not harder

What is Active Learning?

Core Idea: Let the model choose which examples to label.

```
PASSIVE LEARNING (Traditional)

Random sample --> Human labels --> Train model


ACTIVE LEARNING

Model picks "hard" examples --> Human labels --> Train model
      ^                                              |
      |______________________________________________|
                         (repeat)
```

Why it works: Not all examples are equally informative!

The Teaching Analogy

Imagine you're learning to drive:

```
PASSIVE LEARNING:
Instructor randomly picks roads
- 50 trips on straight highways  (easy, repetitive)
- 3 trips in parking lots        (never practiced!)
- 2 trips in rain                (rare but important!)

ACTIVE LEARNING:
You tell instructor what you struggle with
- 10 trips on straight highways  (got it!)
- 20 trips in parking lots       (need practice)
- 15 trips in rain               (challenging!)
```

Active learning focuses effort where it helps most!

Active Learning: The Intuition

```
                    Easy Examples
                 (Model is confident)

  "I loved this movie! Best film ever!"
  Model: 99% POSITIVE  --> Don't need to label this


                    Hard Examples
                 (Model is uncertain)

  "The movie was... interesting."
  Model: 48% POS, 52% NEG  --> LABEL THIS ONE!
```

Hard examples teach the model the most!

Movie Review Example: Why Uncertainty Matters

```
# Our movie review classifier after training on 100 examples

reviews = [
    "Best movie ever! 10/10!",                  # Model: 99% POS - Already knows this
    "Terrible waste of time.",                   # Model: 98% NEG - Already knows this
    "It was okay, I guess.",                     # Model: 52% POS - UNCERTAIN!
    "Interesting but flawed.",                   # Model: 55% NEG - UNCERTAIN!
    "Not bad, not great.",                       # Model: 49% POS - VERY UNCERTAIN!
]

# Which should we label next?
# The uncertain ones! They define the decision boundary.
```

The model already "knows" extreme reviews. Label the ambiguous ones!

The Decision Boundary Intuition

```
     Positive Reviews                                    Negative Reviews
          +                                                    -
        + + +                                                - - -
      + + + + +            ?  ?  ?                         - - - - -
    + + + + + + +        ? ? ? ? ?                       - - - - - - -
      + + + + +            ?  ?  ?                         - - - - -
        + + +                                                - - -
          +                                                    -

     Easy to classify     THE BOUNDARY         Easy to classify
     (don't need labels)  (need labels!)       (don't need labels)
```

Active learning samples from the decision boundary where the model is confused!

The Active Learning Loop

```
1. Start with small labeled set (seed data)
                      |
                      v
2. Train model on labeled data
                      |
                      v
3. Model scores unlabeled examples
                      |
                      v
4. Select most informative examples (query strategy)
                      |
                      v
5. Human labels selected examples
                      |
                      v
6. Add to labeled set, repeat from step 2
```

Query Strategies: How to Pick Examples

1. **Uncertainty Sampling** - Pick examples where model is least confident

```
# For classification, pick where max probability is lowest
uncertainty = 1 - max(model.predict_proba(x))
```

2. **Margin Sampling** - Pick where top two classes are closest

```
probs = sorted(model.predict_proba(x), reverse=True)
margin = probs[0] - probs[1]  # Small margin = uncertain
```

3. **Entropy Sampling** - Pick where prediction distribution is most spread

```
entropy = -sum(p * log(p) for p in model.predict_proba(x))
```

Query Strategy Comparison

```
UNCERTAINTY SAMPLING                 MARGIN SAMPLING

Probs: [0.34, 0.33, 0.33]     Probs: [0.50, 0.49, 0.01]
Max: 0.34 (low confidence)     Margin: 0.50 - 0.49 = 0.01

Both would select this         Very uncertain between
example - very uncertain!      top 2 classes

ENTROPY SAMPLING

Probs: [0.34, 0.33, 0.33]
Entropy = -3 * (0.33 * log(0.33)) = 1.58 (high)

Probs: [0.98, 0.01, 0.01]
Entropy = -(0.98*log(0.98) + 2*0.01*log(0.01)) = 0.12 (low)
```

Active Learning with modAL

```python
from modAL.models import ActiveLearner
from modAL.uncertainty import uncertainty_sampling
from sklearn.ensemble import RandomForestClassifier

# Start with small seed set
X_initial, y_initial = X_labeled[:10], y_labeled[:10]
X_pool = X_unlabeled

# Create active learner
learner = ActiveLearner(
    estimator=RandomForestClassifier(),
    query_strategy=uncertainty_sampling,
    X_training=X_initial,
    y_training=y_initial
)
```

Active Learning Loop in Practice

```python
n_queries = 100

for i in range(n_queries):
    # Query for the most uncertain example
    query_idx, query_instance = learner.query(X_pool)

    # Get label from human (or oracle in experiments)
    y_new = get_human_label(query_instance)

    # Teach the model
    learner.teach(query_instance, y_new)

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

    # Track performance
    accuracy = learner.score(X_test, y_test)
    print(f"Query {i+1}: Accuracy = {accuracy:.2%}")
```

Active Learning: Typical Results

```
    Accuracy
        ^
   1.0  |
        |                                    ___________
   0.9  |                               ____/
        |                          ____/
   0.8  |                     ____/
        |                ____/   <-- Active Learning
   0.7  |           ____/
        |      ____/      _______________________
   0.6  |   ___/     _____/
        | /    ____/  <-- Random Sampling
        |/____/
   0.5  +------------------------------------------>
        0    100   200   300   400   500
                Number of Labels

    Active learning reaches 90% accuracy with 200 labels
    Random sampling needs 400+ labels for same accuracy
```

Batch Active Learning

Problem: Querying one example at a time is slow.

Solution: Select a batch of examples at once.

```
from modAL.batch import uncertainty_batch_sampling

learner = ActiveLearner(
    estimator=RandomForestClassifier(),
    query_strategy=uncertainty_batch_sampling,
    X_training=X_initial,
    y_training=y_initial
)

# Query 10 examples at once
query_idx, query_instances = learner.query(X_pool, n_instances=10)
```

Challenge: Top-10 uncertain examples might be very similar!

Diversity in Batch Selection

```python
from modAL.batch import ranked_batch

def diversity_uncertainty_sampling(classifier, X_pool, n_instances=10):
    # Get uncertainty scores
    uncertainty = 1 - np.max(classifier.predict_proba(X_pool), axis=1)

    # Cluster similar examples
    from sklearn.cluster import KMeans
    kmeans = KMeans(n_clusters=n_instances).fit(X_pool)

    # Pick most uncertain from each cluster
    selected = []
    for cluster_id in range(n_instances):
        cluster_mask = kmeans.labels_ == cluster_id
        cluster_uncertainty = uncertainty[cluster_mask]
        best_idx = np.argmax(cluster_uncertainty)
        selected.append(np.where(cluster_mask)[0][best_idx])

    return selected
```

Active Learning: Practical Considerations

1. Cold Start Problem

- Initial model is bad, uncertainty estimates unreliable
- Solution: Start with diverse random sample or stratified sample

2. Stopping Criteria

- When to stop labeling?
- Options: Budget exhausted, accuracy plateau, uncertainty threshold

3. Class Imbalance

- Uncertainty sampling may neglect rare classes
- Solution: Add diversity constraint or stratified sampling

4. Batch vs Sequential

Active Learning Tools

Tool	Description	Best For
modAL	Python library, sklearn-compatible	Research, prototyping
Label Studio ML	ML backend for Label Studio	Production annotation
Prodigy	Commercial, built-in active learning	NLP tasks
BaaL	Bayesian active learning	Deep learning

```
# Install modAL
pip install modAL-python

# Install Label Studio with ML backend
pip install label-studio
pip install label-studio-ml
```

Part 3: Weak Supervision

Label with code, not clicks

What is Weak Supervision?

Core Idea: Write labeling functions (code) instead of labeling examples.

```python
# Traditional: Label 10,000 examples by hand

# Weak Supervision: Write 10 labeling functions
def lf_contains_love(text):
    return "POSITIVE" if "love" in text.lower() else None

def lf_contains_terrible(text):
    return "NEGATIVE" if "terrible" in text.lower() else None

def lf_exclamation_count(text):
    if text.count("!") > 3:
        return "POSITIVE"
    return None
```

Trade-off: Labels are noisier, but you get many more of them!

The Expert Knowledge Intuition

You're a movie critic. How do you know a review is positive?

```
YOUR BRAIN'S "LABELING FUNCTIONS":

1. Contains "amazing", "loved", "masterpiece" --> Positive
2. Contains "boring", "waste", "terrible"     --> Negative
3. Rating mentioned > 8/10                    --> Positive
4. Multiple exclamation marks                 --> Probably positive
5. Mentions "Oscar" or "award"                --> Probably positive
6. Very short review                          --> Often negative (rant)
```

Weak supervision = encoding your expert intuition as code!

Labeling Functions: Netflix Movie Example

```python
# Real labeling functions for our Netflix movie dataset

@labeling_function()
def lf_high_rating(movie):
    """Movies rated > 8 on IMDB are usually good."""
    if movie.imdb_rating and movie.imdb_rating > 8.0:
        return POSITIVE
    return ABSTAIN

@labeling_function()
def lf_oscar_winner(movie):
    """Oscar winners are good movies."""
    if "Oscar" in str(movie.awards) and "Won" in str(movie.awards):
        return POSITIVE
    return ABSTAIN

@labeling_function()
def lf_low_box_office(movie):
    """Very low box office often means bad movie."""
    if movie.box_office and movie.box_office < 1_000_000:
        return NEGATIVE
    return ABSTAIN

@labeling_function()
def lf_sequel_fatigue(movie):
    """Sequels numbered > 3 are often worse."""
    if re.search(r'\b[4-9]\b|10|11|12', movie.title):
        return NEGATIVE
    return ABSTAIN
```

Labeling Functions: Characteristics

```
                    COVERAGE vs ACCURACY

    High Coverage,          Low Coverage,
    Low Accuracy            High Accuracy
         ^                       ^
         |                       |
    ┌─────────┐             ┌─────────┐
    │ Keyword │             │ Pattern │
    │ Match   │             │ Match   │
    │         │             │         │
    │ "good"  │             │ rating  │
    │ in text │             │ > 9/10  │
    │ -> POS  │             │ -> POS  │
    └─────────┘             └─────────┘

    Matches 40%             Matches 5%
    of data                 of data

    70% accurate            95% accurate
```

Labeling Functions: Types

1. Keyword/Pattern-based

```python
def lf_keyword_positive(text):
    keywords = ["amazing", "excellent", "loved", "great"]
    return "POS" if any(k in text.lower() for k in keywords) else None
```

2. Heuristic-based

```python
def lf_short_reviews_negative(text):
    # Short reviews tend to be complaints
    return "NEG" if len(text.split()) < 10 else None
```

3. External Knowledge

```python
def lf_known_good_movie(text, movie_title):
    top_movies = load_imdb_top_250()
    return "POS" if movie_title in top_movies else None
```

Labeling Function Conflicts

Problem: LFs often disagree!

```
text = "I love how terrible this movie is!"

lf_contains_love(text)     # Returns: "POSITIVE"
lf_contains_terrible(text) # Returns: "NEGATIVE"

# Which one is right?
```

Solution: Use a **Label Model** to combine LF outputs.

Snorkel: The Weak Supervision Framework

```python
from snorkel.labeling import labeling_function, LFAnalysis

@labeling_function()
def lf_contains_good(x):
    return 1 if "good" in x.text.lower() else -1  # 1=POS, 0=NEG, -1=ABSTAIN

@labeling_function()
def lf_contains_bad(x):
    return 0 if "bad" in x.text.lower() else -1

@labeling_function()
def lf_rating_based(x):
    if hasattr(x, 'rating') and x.rating is not None:
        return 1 if x.rating > 7 else 0
    return -1
```

Applying Labeling Functions

```python
from snorkel.labeling import PandasLFApplier

# Define all LFs
lfs = [lf_contains_good, lf_contains_bad, lf_rating_based]

# Apply to data
applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df_train)

# L_train is a matrix: (n_examples, n_lfs)
# Each cell is 1 (POS), 0 (NEG), or -1 (ABSTAIN)

print(L_train[:5])
# [[-1,  0, -1],   # Only lf_contains_bad fired -> NEG
#  [ 1, -1,  1],   # lf_contains_good and lf_rating agree -> POS
#  [-1, -1, -1],   # No LF fired -> unlabeled
#  [ 1,  0, -1],   # Conflict! good vs bad
#  [-1, -1,  0]]   # Only lf_rating -> NEG
```

Analyzing Labeling Functions

```
from snorkel.labeling import LFAnalysis

analysis = LFAnalysis(L=L_train, lfs=lfs).lf_summary()
print(analysis)
```

LF	Polarity	Coverage	Overlaps	Conflicts
lf_contains_good	[1]	0.25	0.12	0.05
lf_contains_bad	[0]	0.18	0.08	0.05
lf_rating_based	[0, 1]	0.60	0.15	0.02

Coverage: Fraction of data the LF labels

Overlaps: Fraction where LF agrees with another

Conflicts: Fraction where LF disagrees with another

The Label Model

Goal: Combine noisy LF outputs into probabilistic labels.

```python
from snorkel.labeling.model import LabelModel

# Train label model
label_model = LabelModel(cardinality=2, verbose=True)
label_model.fit(L_train=L_train, n_epochs=500)

# Get probabilistic labels
probs = label_model.predict_proba(L_train)
# probs[i] = [P(NEG), P(POS)] for example i

# Get hard labels (for training downstream model)
preds = label_model.predict(L_train)
```

How the Label Model Works

```
LF Outputs                         Label Model                Probabilistic
(noisy votes)                      (learns weights)           Labels

lf_good:   [1, -1, 1, 0]                                      [0.85, 0.2, 0.9, 0.3]
lf_bad:    [0, -1, 0, 1]  -->   Learns which    -->
lf_rating:[1, -1, 1, 1]         LFs are accurate              P(POSITIVE|LF outputs)

Key Insight:
- LFs that often agree are probably accurate
- LFs that often conflict might be noisy
- Model learns accuracy WITHOUT ground truth labels!
```

The Voting Intuition

Think of LFs as a jury voting on each example:

```
Movie: "The Godfather" (1972)

    LF_high_rating:     POSITIVE  (IMDB: 9.2)
    LF_oscar_winner:    POSITIVE  (Won Best Picture)
    LF_classic_year:    POSITIVE  (Before 1980, acclaimed)
    LF_sequel_fatigue:  ABSTAIN   (Not a sequel)
    LF_low_budget:      ABSTAIN   (No data)

    Jury Vote: 3 POSITIVE, 0 NEGATIVE, 2 ABSTAIN

    Label Model Output: 95% POSITIVE
```

LFs that agree with each other get higher weight!

When LFs Disagree: The Label Model Resolves

```
Movie: "Sharknado 5" (2017)

    LF_high_rating:     NEGATIVE  (IMDB: 3.5)
    LF_cult_classic:    POSITIVE  (Has devoted fanbase)
    LF_sequel_fatigue:  NEGATIVE  (5th sequel!)
    LF_social_buzz:     POSITIVE  (Trending on Twitter)

    Jury Vote: 2 POSITIVE, 2 NEGATIVE

    But LF_high_rating has 90% accuracy historically
    And LF_cult_classic only has 60% accuracy

    Label Model Output: 65% NEGATIVE
    (Weighs LFs by learned accuracy)
```

The label model learns which LFs to trust!

Training Downstream Model

```python
from snorkel.labeling import filter_unlabeled_dataframe

# Filter out examples where no LF fired
df_train_filtered, probs_filtered = filter_unlabeled_dataframe(
    df_train, probs, L_train
)

# Train your actual model on probabilistic labels
from sklearn.linear_model import LogisticRegression

# Use soft labels (probabilistic)
model = LogisticRegression()
model.fit(
    df_train_filtered['text_features'],
    probs_filtered[:, 1]  # P(POSITIVE)
)

# Or use hard labels
hard_labels = (probs_filtered[:, 1] > 0.5).astype(int)
model.fit(df_train_filtered['text_features'], hard_labels)
```

Weak Supervision: Complete Example

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

# 1. Load unlabeled data
df = pd.read_csv("movie_reviews.csv")

# 2. Define labeling functions
@labeling_function()
def lf_awesome(x):
    return 1 if "awesome" in x.text.lower() else -1

@labeling_function()
def lf_boring(x):
    return 0 if "boring" in x.text.lower() else -1

lfs = [lf_awesome, lf_boring, ...]  # Add more LFs
```

Weak Supervision: Complete Example (cont.)

```
# 3. Apply LFs to data
applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df)

# 4. Train label model
label_model = LabelModel(cardinality=2)
label_model.fit(L_train, n_epochs=500)

# 5. Get probabilistic labels
probs = label_model.predict_proba(L_train)

# 6. Train downstream model
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['text'])
model = MultinomialNB()
model.fit(X, (probs[:, 1] > 0.5).astype(int))
```

When to Use Weak Supervision

Good candidates:

- Patterns can be encoded as rules
- You have domain knowledge
- Labels have clear heuristics
- Data is too large for manual labeling

Bad candidates:

- Task requires human judgment (e.g., humor detection)
- No clear patterns or heuristics
- Very small dataset (just label it manually)
- High precision required (weak labels are noisy)

Part 4: LLM-Based Labeling

AI labeling your data

The LLM Labeling Revolution

2022-2024: Large Language Models became viable annotators.

```
# Before: Hire annotators
cost_per_label = 0.30  # USD
human_labels = 10000
total_cost = 3000  # USD

# Now: Use GPT-4 / Claude
cost_per_label = 0.002  # USD (roughly)
llm_labels = 10000
total_cost = 20  # USD

# 150x cost reduction!
```

But: Are LLM labels as good as human labels?

Why LLMs Can Label Data

LLMs are trained on the entire internet - they've "seen" everything:

```
GPT-4 has read:
- Millions of movie reviews
- IMDB, Rotten Tomatoes, Metacritic
- Professional film criticism
- Reddit discussions about movies
- Academic papers on sentiment analysis

So when you ask: "Is this review positive or negative?"
It can often answer correctly!
```

LLMs = Crowdsourced human knowledge, distilled into a model

LLM Labeling: Movie Review Example

```python
import openai

def label_movie_review(review):
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": """
                You are a movie critic. Classify reviews as:
                - POSITIVE: Reviewer enjoyed the movie
                - NEGATIVE: Reviewer did not enjoy the movie
                - NEUTRAL: Mixed feelings or no clear opinion
            """},
            {"role": "user", "content": f'Review: "{review}"\n\nClassification:'}
        ]
    )
    return response.choices[0].message.content

# Examples from our Netflix dataset
print(label_movie_review("Mind-blowing visuals! Nolan does it again!"))
# Output: POSITIVE

print(label_movie_review("Meh. Seen better, seen worse."))
# Output: NEUTRAL

print(label_movie_review("Two hours of my life I'll never get back."))
# Output: NEGATIVE
```

LLM Labeling: Basic Approach

```python
import openai

def label_with_gpt(text, task_description):
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[
            {"role": "system", "content": task_description},
            {"role": "user", "content": f"Text: {text}\n\nLabel:"}
        ],
        max_tokens=10
    )
    return response.choices[0].message.content.strip()

# Example
task = "Classify movie reviews as POSITIVE or NEGATIVE."
label = label_with_gpt("This movie was incredible!", task)
print(label)  # "POSITIVE"
```

Prompt Engineering for Annotation

Bad Prompt:

```
Classify this: "The movie was okay I guess"
```

Good Prompt:

```
You are an expert movie critic annotating reviews for sentiment.

Task: Classify the sentiment of movie reviews.

Labels:
- POSITIVE: The reviewer liked the movie
- NEGATIVE: The reviewer disliked the movie
- NEUTRAL: The reviewer has mixed or no strong feelings

Review: "The movie was okay I guess"

Classification (respond with only the label):
```

Few-Shot Prompting

```
prompt = """Classify movie review sentiment.

Examples:
Review: "Absolutely loved it! Best movie of the year!"
Label: POSITIVE

Review: "Waste of time. Don't bother watching."
Label: NEGATIVE

Review: "It was fine. Nothing special but not bad."
Label: NEUTRAL

Review: "{review_text}"
Label:"""

label = label_with_gpt(prompt.format(review_text=text))
```

Few-shot examples significantly improve accuracy!

Getting Confidence Scores

```python
def label_with_confidence(text):
    prompt = f"""Classify the sentiment and provide confidence.

Text: "{text}"

Respond in JSON format:
{{"label": "POSITIVE/NEGATIVE/NEUTRAL", "confidence": 0.0-1.0}}
"""
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}]
    )

    import json
    result = json.loads(response.choices[0].message.content)
    return result["label"], result["confidence"]

label, conf = label_with_confidence("Great movie!")
print(f"Label: {label}, Confidence: {conf}")
# Label: POSITIVE, Confidence: 0.95
```

Batch Processing for Cost Efficiency

```python
import asyncio
import aiohttp

async def batch_label(texts, batch_size=10):
    results = []

    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]

        # Format as single prompt
        prompt = "Classify each review:\n\n"
        for j, text in enumerate(batch):
            prompt += f"{j+1}. {text}\n"
        prompt += "\nRespond with labels (one per line):"

        response = await async_gpt_call(prompt)
        labels = response.strip().split('\n')
        results.extend(labels)

    return results
```

LLM Labeling Quality Control

```python
def validate_llm_labels(texts, llm_labels, sample_size=100):
    # Random sample for human validation
    indices = random.sample(range(len(texts)), sample_size)

    human_labels = []
    for idx in indices:
        print(f"Text: {texts[idx]}")
        print(f"LLM Label: {llm_labels[idx]}")
        human = input("Your label (or 'agree'): ")
        if human.lower() == 'agree':
            human_labels.append(llm_labels[idx])
        else:
            human_labels.append(human)

    # Calculate agreement
    agreement = sum(h == llm_labels[i] for i, h in
                    zip(indices, human_labels)) / sample_size
    print(f"LLM-Human Agreement: {agreement:.1%}")

    return agreement
```

When LLMs Struggle

1. Subjective Tasks

```
"This movie is so bad it's good"
LLM: NEGATIVE (wrong - it's ironic praise!)
```

2. Domain-Specific Knowledge

```
"The mise-en-scene was pedestrian but the diegetic sound..."
LLM: ? (needs film theory knowledge)
```

3. Nuanced Categories

```
5-point scale: Very Negative, Negative, Neutral, Positive, Very Positive
LLM accuracy drops significantly with more categories
```

4. Ambiguous Guidelines

```
What exactly counts as "slightly negative"?
```

Hybrid Approach: LLM + Human

```python
def hybrid_labeling(texts, confidence_threshold=0.8):
    llm_labels = []
    human_queue = []

    for i, text in enumerate(texts):
        label, confidence = label_with_confidence(text)

        if confidence >= confidence_threshold:
            llm_labels.append((i, label, "llm"))
        else:
            human_queue.append(i)

    print(f"LLM labeled: {len(llm_labels)}")
    print(f"Need human: {len(human_queue)}")

    # Send human_queue to annotation platform
    return llm_labels, human_queue
```

Use LLMs for easy examples, humans for hard ones!

LLM Labeling: Cost Comparison

Method	Cost/1000	Quality	Speed
Expert humans	$300-500	Highest	Slow
Crowdsourcing	$50-100	Medium	Medium
GPT-4	$20-50	Good	Fast
GPT-3.5	$2-5	Moderate	Very Fast
Claude Haiku	$1-3	Moderate	Very Fast
Open source LLM	~$0 (compute)	Varies	Depends

Sweet spot: GPT-3.5/Claude for first pass, humans for validation

Part 5: Handling Noisy Labels

Garbage in, garbage out?

Sources of Label Noise

```
LABEL NOISE SOURCES

1. ANNOTATOR ERROR
   - Fatigue, lack of attention
   - Misunderstanding guidelines

2. TASK AMBIGUITY
   - Inherently subjective tasks
   - Unclear category boundaries

3. WEAK SUPERVISION
   - Noisy labeling functions
   - Heuristic errors

4. DATA ENTRY ERRORS
   - Mislabeled due to typos
   - Wrong column/field mapping
```

Detecting Label Errors

Approach 1: Confident Learning

```python
from cleanlab import Datalab

# Your data with possibly noisy labels
X, y = load_data()

# Find label issues
lab = Datalab(data={"X": X, "y": y}, label_name="y")
lab.find_issues(features=X)

# Get indices of likely mislabeled examples
issues = lab.get_issues()
mislabeled = issues[issues['is_label_issue'] == True].index
print(f"Found {len(mislabeled)} potential label errors")
```

Confident Learning: How It Works

```
Model predicts P(class|x) for each example

Example:  Label = "POSITIVE"
          P(NEGATIVE|x) = 0.92
          P(POSITIVE|x) = 0.08

This is likely mislabeled!

┌─────────────────────────────────────────────┐
│  High confidence predictions that           │
│  disagree with given labels = suspicious    │
└─────────────────────────────────────────────┘
```

The Wisdom of the Crowd Intuition

Imagine 100 students grade an essay. 95 say "B", 5 say "A".

```
If the official grade is "A"... something's wrong!

Either:
1. The grading key was wrong
2. The teacher made a mistake
3. Those 5 students are unusually generous

Confident Learning = Train a model on all data
                     Model becomes the "crowd"
                     When crowd disagrees with label = suspicious
```

The model learns the data distribution and spots outliers!

Real Example: Catching Label Errors

```
# Our Netflix movie reviews - some were mislabeled by tired annotators

review = "This movie was not good. I didn't enjoy it at all."
original_label = "POSITIVE"  # Annotator mistake!

# Model prediction after training
model_prediction = {
    "POSITIVE": 0.05,
    "NEGATIVE": 0.95   # Model is VERY confident this is negative
}

# cleanlab flags this as a likely error
# Confidence of label being wrong: 95%

# Upon review: Yes, this was mislabeled!
corrected_label = "NEGATIVE"
```

Cleanlab found the annotator's mistake automatically!

Cleanlab: Practical Example

```python
from cleanlab import Datalab
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_predict

# Get out-of-fold predictions
clf = RandomForestClassifier()
pred_probs = cross_val_predict(clf, X, y, cv=5, method='predict_proba')

# Find label issues using predictions
lab = Datalab(data={"features": X, "labels": y}, label_name="labels")
lab.find_issues(pred_probs=pred_probs)

# Examine issues
print(lab.get_issue_summary())
print(lab.get_issues().head(10))

# Get clean indices
clean_indices = lab.get_issues()[
    lab.get_issues()['is_label_issue'] == False
].index
```

What to Do With Noisy Labels?

Option 1: Remove them

```
clean_X = X[clean_indices]
clean_y = y[clean_indices]
model.fit(clean_X, clean_y)
```

Option 2: Re-label them

```
for idx in mislabeled_indices:
    new_label = get_human_label(X[idx])
    y[idx] = new_label
```

Option 3: Train with noise-robust methods

```
# Use label smoothing, mixup, or noise-robust losses
```

Learning With Noisy Labels

Label Smoothing: Soften hard labels

```
# Instead of y = [1, 0, 0] (one-hot)
# Use y = [0.9, 0.05, 0.05] (smoothed)

def label_smoothing(y, num_classes, epsilon=0.1):
    smoothed = np.full((len(y), num_classes), epsilon / num_classes)
    for i, label in enumerate(y):
        smoothed[i, label] = 1 - epsilon + epsilon / num_classes
    return smoothed
```

Mixup: Interpolate between examples

```
# Create synthetic training examples
alpha = np.random.beta(0.2, 0.2)
x_mixed = alpha * x1 + (1 - alpha) * x2
y_mixed = alpha * y1 + (1 - alpha) * y2
```

Noise Transition Matrix

Idea: Model how labels get corrupted

```
True Label ->  Observed Label

              POS    NEG
True POS    [ 0.85   0.15 ]   15% of true POS are labeled NEG
True NEG    [ 0.10   0.90 ]   10% of true NEG are labeled POS
```

```
# Estimate transition matrix from data
from cleanlab.count import estimate_cv_predicted_probabilities
from cleanlab.count import compute_confident_joint

pred_probs = estimate_cv_predicted_probabilities(X, y, clf)
confident_joint = compute_confident_joint(labels=y, pred_probs=pred_probs)
transition_matrix = confident_joint / confident_joint.sum(axis=1, keepdims=True)
```

Part 6: Combining Approaches

The best of all worlds

Decision Tree: Which Technique?

```
                  START
                    │
                    v
          ┌──────────────────────┐
          │ How much data do     │
          │ you need to label?   │
          └──────────────────────┘
                    │
         ┌──────────┼──────────┐
         v          v          v
      <1000      1k-10k      >10k
         │          │          │
         v          v          v
    Manual     Active     Weak Supervision
    Labeling   Learning   or LLM Labeling
                  │
                  v
            ┌─────────────┐
            │ Can write   │──Yes──> + Weak Supervision
            │ heuristics? │
            └─────────────┘
                  │No
                  v
            ┌─────────────┐
            │ Budget for  │──Yes──> + LLM Labeling
            │ LLM API?    │
            └─────────────┘
```

Hybrid Pipeline Example

```python
# Step 1: Weak supervision for bulk labels
weak_labels = apply_labeling_functions(unlabeled_data)

# Step 2: LLM for high-uncertainty examples
uncertain = get_low_confidence_examples(weak_labels)
llm_labels = batch_label_with_gpt(uncertain)

# Step 3: Active learning for remaining hard cases
learner = ActiveLearner(estimator=model)
for round in range(n_rounds):
    query_idx = learner.query(hard_examples)
    human_labels = get_human_labels(hard_examples[query_idx])
    learner.teach(hard_examples[query_idx], human_labels)

# Step 4: Clean noisy labels
all_labels = combine_labels(weak_labels, llm_labels, human_labels)
clean_labels = cleanlab_filter(all_labels)

# Step 5: Train final model
model.fit(data, clean_labels)
```

Cost-Benefit Analysis

Approach	Setup Cost	Per-Label Cost	Quality
Manual only	Low	$0.30	High
+ Active Learning	Medium	$0.30 (fewer)	High
+ Weak Supervision	High	~$0	Medium
+ LLM Labeling	Low	$0.002	Medium-High
+ Noise Cleaning	Medium	~$0	Improved

Typical savings: 5-20x cost reduction with hybrid approach

Part 7: Key Takeaways

Key Takeaways

1. **Active Learning** - Let model pick what to label (2-10x savings)
2. **Weak Supervision** - Write labeling functions (10-100x savings)
3. **LLM Labeling** - Use GPT/Claude as annotators (10-50x cost reduction)
4. **Noisy Labels** - Detect and handle with cleanlab
5. **Combine approaches** - Hybrid pipelines give best results
6. **Quality matters** - Validate with human spot-checks
7. **Tools exist** - modAL, Snorkel, cleanlab, OpenAI API

Part 8: Lab Preview

What you'll build today

This Week's Lab

Hands-on Practice:

1. **Active Learning with modAL**
    - Implement uncertainty sampling
    - Compare to random sampling
    - Visualize learning curves
2. **Weak Supervision with Snorkel**
    - Write labeling functions
    - Train label model
    - Analyze LF quality
3. **LLM Labeling**
    - Prompt engineering for annotation

Lab Setup Preview

```
# Install required packages
pip install modAL-python
pip install snorkel
pip install cleanlab
pip install openai

# Verify installations
python -c "import modAL; print('modAL OK')"
python -c "import snorkel; print('Snorkel OK')"
python -c "import cleanlab; print('cleanlab OK')"
```

You'll implement a complete labeling optimization pipeline!

Next Week Preview

Week 5: Data Augmentation

- Why augmentation improves models
- Text augmentation techniques
- Image augmentation with Albumentations
- Audio and video augmentation
- When (not) to augment

More data from existing data - without labeling!

Resources

Libraries:

- modAL: https://modal-python.readthedocs.io/
- Snorkel: https://snorkel.ai/
- cleanlab: https://cleanlab.ai/
- OpenAI API: https://platform.openai.com/

Papers:

- "Data Programming" (Snorkel paper)
- "Confident Learning" (cleanlab paper)

Reading:

- Snorkel tutorials: https://www.snorkel.org/use-cases/

Questions?

Thank You!

See you in the lab!