Data Collection and Labeling

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar

Module Overview

What We'll Learn Today

1. **Data Collection** — Where does AI data come from?
2. **Data Validation** — How do we ensure quality?
3. **Data Labeling** — How do we add ground truth?
4. **Data Augmentation** — How do we create more data?

> 💡 **Tip**
> **Key Idea:** 80% of AI work is data! Good data = Good models

Why Does Data Matter?

```
# Bad data example
model.train(data=[1, 2, None, "three", 999, -5])
# Result: Garbage predictions! 🚫

# Good data example
model.train(data=[1.2, 2.3, 3.1, 4.5, 5.2, 6.1])
# Result: Reliable predictions! ✅
```

Your model is only as good as your data!

Part 1: Data Collection

Real-World Example: Building a Chatbot

What you need:

- User messages
- Timestamps
- Response times
- Error logs
- Click events

Where it comes from:

- Web app logs
- Mobile app analytics
- Server logs
- Database queries
- User feedback forms

Python Logging: Your First Tool

```python
import logging
import datetime

# Set up logging
logging.basicConfig(
    filename='chatbot.log',
    level=logging.INFO,
    format='%(asctime)s - %(message)s'
)

# Log user interactions
def handle_message(user_id, message):
    logging.info(f"User {user_id}: {message}")
    response = generate_response(message)
    logging.info(f"Bot response: {response}")
    return response

# Example usage
handle_message("user123", "What's the weather?")
```

Exercise: Add Logging to Your Code

```
# Try this yourself!
def calculate_score(answers):
    # TODO: Add logging here
    correct = sum(1 for a in answers if a.is_correct)
    # TODO: Log the result
    return correct / len(answers)

# Better version:
def calculate_score(answers):
    logging.info(f"Calculating score for {len(answers)} answers")
    correct = sum(1 for a in answers if a.is_correct)
    score = correct / len(answers)
    logging.info(f"Final score: {score:.2%}")
    return score
```

Structured Logging with JSON

```
import json
import logging

def log_event(event_type, user_id, data):
    event = {
        "timestamp": datetime.datetime.now().isoformat(),
        "type": event_type,
        "user_id": user_id,
        "data": data
    }
    logging.info(json.dumps(event))

# Example: Track video watching
log_event("video_start", "user456", {
    "video_id": "python_basics_01",
    "duration": 600
})

log_event("video_complete", "user456", {
```

```
    "video_id": "python_basics_01",
    "watch_time": 580
})
```

> **i Note**
> **Why JSON?** Easy to parse later with `json.loads()` for analysis!

Web Analytics: Google Analytics Example

```
<!-- Add to your website -->
<script async src="https://www.googletagmanager.com/gtag/js?id=GA_ID"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'GA_MEASUREMENT_ID');

  // Track custom events
  gtag('event', 'button_click', {
    'button_name': 'download_report',
    'page': 'dashboard'
  });
</script>
```

Track: Page views, clicks, time on page, user flows

Web Scraping: Collecting Public Data

```python
import requests
from bs4 import BeautifulSoup

# Scrape quotes from a public website
url = "http://quotes.toscrape.com"
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Extract quotes
quotes = soup.find_all('span', class_='text')
for quote in quotes[:3]:  # First 3 quotes
    print(quote.text)

# Extract authors
authors = soup.find_all('small', class_='author')
for author in authors[:3]:
    print(f"  - {author.text}")
```

Web Scraping: Collecting Public Data

> ⚠️ **Warning**
> Always check `robots.txt` and respect rate limits!

Exercise: Scrape a Simple Website

```
# Try scraping this practice website
url = "http://books.toscrape.com"

# TODO:
# 1. Get all book titles
# 2. Get all prices
# 3. Save to a CSV file

import csv

def scrape_books():
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    books = []
    for book in soup.find_all('article', class_='product_pod'):
        title = book.h3.a['title']
        price = book.find('p', class_='price_color').text
        books.append({"title": title, "price": price})
```

Exercise: Scrape a Simple Website

```
    return books
```

API Data Collection

```python
import requests

# Example: GitHub API
url = "https://api.github.com/repos/python/cpython"
response = requests.get(url)
data = response.json()

print(f"Repository: {data['name']}")
print(f"Stars: {data['stargazers_count']}")
print(f"Forks: {data['forks_count']}")
print(f"Language: {data['language']}")

# Track repository statistics over time
def collect_repo_stats(repo_name):
    url = f"https://api.github.com/repos/{repo_name}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        return {
```

API Data Collection

```
        "date": datetime.datetime.now().isoformat(),
        "stars": data['stargazers_count'],
        "forks": data['forks_count']
    }
```

Database Logging Example

```
import sqlite3
from datetime import datetime

# Create database
conn = sqlite3.connect('user_activity.db')
cursor = conn.cursor()

# Create table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS events (
        id INTEGER PRIMARY KEY,
        timestamp TEXT,
        user_id TEXT,
        event_type TEXT,
        details TEXT
    )
''')

# Log an event
```

Database Logging Example

```
def log_to_db(user_id, event_type, details):
    cursor.execute(
        'INSERT INTO events VALUES (NULL, ?, ?, ?, ?)',
        (datetime.now().isoformat(), user_id, event_type, details)
    )
    conn.commit()

log_to_db("user789", "login", "successful")
```

Sampling Strategies

Random Sampling

```python
import random

# Sample 10% of data
def sample_data(data, rate=0.1):
    return [x for x in data
            if random.random() < rate]

large_dataset = range(10000)
sample = sample_data(large_dataset)
print(len(sample))  # ~1000
```

Stratified Sampling

```python
from collections import defaultdict

# Sample equally from each class
def stratified_sample(data, n=10):
    by_class = defaultdict(list)
    for item in data:
        by_class[item['class']].append(item)

    sample = []
    for cls_data in by_class.values():
        sample.extend(random.sample(cls_data, n))
    return sample
```

Part 2: Data Validation

Why Validate Data?

Real story: A medical AI misdiagnosed patients because:

- Some ages were entered as “99” (meaning unknown)
- Heights were in cm AND inches (mixed units)
- Blood pressure had typos: “210/80” entered as “21080”

⚠ Warning

Bad data ⟶ Bad predictions ⟶ Real harm!

Common Data Problems

```
# Problem 1: Missing values
data = [1, 2, None, 4, 5]  # What to do with None?

# Problem 2: Wrong types
age = "twenty-five"  # Should be: 25

# Problem 3: Out of range
temperature = -500  # Impossible!

# Problem 4: Inconsistent format
dates = ["2024-01-15", "15/01/2024", "Jan 15, 2024"]

# Problem 5: Duplicates
users = ["alice", "bob", "alice"]  # alice appears twice!
```

Data Validation with Python

```python
def validate_age(age):
    """Validate age is a reasonable number"""
    if not isinstance(age, (int, float)):
        raise TypeError(f"Age must be a number, got {type(age)}")

    if age < 0 or age > 150:
        raise ValueError(f"Age {age} is out of valid range (0-150)")

    return True

# Test it
try:
    validate_age(25)    # ✅ OK
    validate_age(-5)    # ❌ Raises ValueError
    validate_age("old") # ❌ Raises TypeError
except Exception as e:
    print(f"Validation error: {e}")
```

Pydantic: Automatic Validation

```python
from pydantic import BaseModel, Field, validator
from typing import Optional

class User(BaseModel):
    name: str = Field(..., min_length=1, max_length=100)
    age: int = Field(..., ge=0, le=150)
    email: str
    height_cm: float = Field(..., gt=0, lt=300)

    @validator('email')
    def validate_email(cls, v):
        if '@' not in v:
            raise ValueError('Invalid email')
        return v

# Automatic validation!
user = User(
    name="Alice",
    age=20,
```

Pydantic: Automatic Validation

```
    email="alice@example.com",
    height_cm=165.5
)  # ✅ Works!

# This fails automatically
try:
    bad_user = User(name="", age=200, email="bad", height_cm=-10)
except Exception as e:
    print("Validation failed:", e)
```

Exercise: Create Your Own Validator

```
# TODO: Create a validator for student records
class Student(BaseModel):
    roll_number: str  # Format: "2024CS001"
    name: str
    gpa: float  # 0.0 to 10.0
    courses: list[str]  # At least 1 course

    # Add validators:
    # 1. roll_number should match pattern "YYYYCS\\d{3}"
    # 2. gpa should be between 0 and 10
    # 3. courses list should not be empty
    # 4. name should be at least 2 characters

# Hint: Use @validator decorator
```

Pandas Data Validation

```
import pandas as pd

# Load data
df = pd.read_csv('students.csv')

# Check for missing values
print(df.isnull().sum())

# Check data types
print(df.dtypes)

# Check for duplicates
print(df.duplicated().sum())

# Check value ranges
print(df['age'].describe())

# Find outliers
mean = df['score'].mean()
```

```
std = df['score'].std()
outliers = df[abs(df['score'] - mean) > 3 * std]
print(f"Found {len(outliers)} outliers")
```

Handling Missing Data

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
    'name': ['Alice', 'Bob', None, 'Dave'],
    'age': [25, None, 30, 28],
    'score': [85, 90, None, 88]
})

# Option 1: Drop rows with missing values
df_clean = df.dropna()

# Option 2: Fill with default value
df_filled = df.fillna({
    'name': 'Unknown',
    'age': df['age'].mean(),
    'score': 0
})
```

Handling Missing Data

```
# Option 3: Forward fill
df_ffill = df.fillna(method='ffill')

# Check result
print(df_filled)
```

Data Type Conversion

```
# Convert strings to numbers
df['age'] = pd.to_numeric(df['age'], errors='coerce')

# Convert strings to dates
df['date'] = pd.to_datetime(df['date'], errors='coerce')

# Convert to categories (saves memory!)
df['grade'] = df['grade'].astype('category')

# Handle errors gracefully
def safe_convert(value):
    try:
        return float(value)
    except (ValueError, TypeError):
        return None

df['score'] = df['score'].apply(safe_convert)
```

Part 3: Data Labeling

What is Data Labeling?

Before labeling:

- Image of a dog
- Text: “Great product!”
- Audio clip
- Video frame

After labeling:

- Image of a dog → “dog”
- Text → “positive sentiment”
- Audio → “speaker_id: person1”
- Video → “car detected at (100, 200)”

> 💡 **Tip**
> Labels are the “answers” we want our AI to learn!

Image Classification: Labeling Cats vs Dogs

```python
# Simple labeling script
import os
from PIL import Image

def label_images(image_folder):
    labels = []

    for filename in os.listdir(image_folder):
        if filename.endswith('.jpg'):
            img_path = os.path.join(image_folder, filename)
            img = Image.open(img_path)
            img.show()

            # Ask user for label
            label = input(f"Label for {filename} (cat/dog): ")
            labels.append({
                'filename': filename,
                'label': label
            })
```

Image Classification: Labeling Cats vs Dogs

```
    # Save labels
    import json
    with open('labels.json', 'w') as f:
        json.dump(labels, f, indent=2)

# Usage
label_images('animal_images/')
```

Text Classification Example

```
# Sentiment labeling
reviews = [
    "This product is amazing!",
    "Worst purchase ever.",
    "It's okay, nothing special.",
    "Absolutely love it!",
    "Terrible quality."
]

# Manual labeling
labels = []
for review in reviews:
    print(f"\nReview: {review}")
    sentiment = input("Sentiment (positive/negative/neutral): ")
    labels.append({
        'text': review,
        'sentiment': sentiment
    })
```

Text Classification Example

```
# Create dataset
import pandas as pd
df = pd.DataFrame(labels)
df.to_csv('sentiment_labels.csv', index=False)
print(df)
```

Label Studio: Professional Tool

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

# Start server
label-studio start

# Open browser: http://localhost:8080
```

Features:

- Image annotation (boxes, polygons, keypoints)
- Text annotation (NER, classification)
- Audio annotation
- Video annotation
- Team collaboration

Inter-Annotator Agreement

```
# When multiple people label the same data
from sklearn.metrics import cohen_kappa_score

# Two annotators label 10 images
annotator1 = ['cat', 'dog', 'cat', 'dog', 'cat',
              'dog', 'cat', 'dog', 'cat', 'dog']
annotator2 = ['cat', 'dog', 'cat', 'cat', 'cat',
              'dog', 'dog', 'dog', 'cat', 'dog']

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

# Interpret:
# 0.0-0.20: Poor
# 0.21-0.40: Fair
# 0.41-0.60: Moderate
# 0.61-0.80: Good
# 0.81-1.00: Excellent
```

Active Learning: Smart Labeling

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier

def active_learning(X_unlabeled, model, n_samples=10):
    """Select most uncertain samples to label"""
    # Get prediction probabilities
    probs = model.predict_proba(X_unlabeled)

    # Calculate uncertainty (entropy)
    entropy = -np.sum(probs * np.log(probs + 1e-10), axis=1)

    # Select most uncertain samples
    uncertain_idx = np.argsort(entropy)[-n_samples:]

    return X_unlabeled[uncertain_idx]

# Example
model = RandomForestClassifier()
model.fit(X_labeled, y_labeled)
```

Active Learning: Smart Labeling

```
# Get 10 samples that model is most uncertain about
samples_to_label = active_learning(X_unlabeled, model)
print(f"Label these {len(samples_to_label)} samples next!")
```

> 💡 **Tip**
> Active learning can reduce labeling effort by 50-90%!

Weak Supervision Example

```
# Use rules instead of manual labels
def label_review(text):
    """Weak labeling with rules"""
    positive_words = ['great', 'amazing', 'excellent', 'love']
    negative_words = ['bad', 'terrible', 'awful', 'hate']

    text_lower = text.lower()

    pos_count = sum(1 for word in positive_words if word in text_lower)
    neg_count = sum(1 for word in negative_words if word in text_lower)

    if pos_count > neg_count:
        return 'positive'
    elif neg_count > pos_count:
        return 'negative'
    else:
        return 'neutral'

# Test
```

Weak Supervision Example

```
review = "This product is amazing and excellent!"
print(label_review(review))  # Output: positive
```

Part 4: Data Augmentation

Why Augment Data?

Problem: You have 100 labeled images, but need 10,000 for training

Solution: Create variations of existing data!

- Flip images horizontally
- Rotate images slightly
- Add noise
- Change brightness
- Crop and resize

> 💡 **Tip**
> More data → Better models (usually!)

Image Augmentation with PIL

```python
from PIL import Image, ImageEnhance
import random

def augment_image(img_path, output_folder):
    img = Image.open(img_path)

    # 1. Flip horizontally
    img_flipped = img.transpose(Image.FLIP_LEFT_RIGHT)
    img_flipped.save(f"{output_folder}/flipped.jpg")

    # 2. Rotate
    angle = random.randint(-15, 15)
    img_rotated = img.rotate(angle)
    img_rotated.save(f"{output_folder}/rotated.jpg")

    # 3. Adjust brightness
    enhancer = ImageEnhance.Brightness(img)
    img_bright = enhancer.enhance(random.uniform(0.8, 1.2))
    img_bright.save(f"{output_folder}/brightness.jpg")
```

Image Augmentation with PIL

```
    # 4. Adjust contrast
    enhancer = ImageEnhance.Contrast(img)
    img_contrast = enhancer.enhance(random.uniform(0.8, 1.2))
    img_contrast.save(f"{output_folder}/contrast.jpg")

# Now you have 5x more images!
```

Image Augmentation with OpenCV

```
import cv2
import numpy as np

def augment_opencv(img_path):
    img = cv2.imread(img_path)
    augmented = []

    # 1. Gaussian blur
    blurred = cv2.GaussianBlur(img, (5, 5), 0)
    augmented.append(('blur', blurred))

    # 2. Add noise
    noise = np.random.normal(0, 25, img.shape).astype(np.uint8)
    noisy = cv2.add(img, noise)
    augmented.append(('noise', noisy))

    # 3. Crop and resize
    h, w = img.shape[:2]
    crop = img[h//4:3*h//4, w//4:3*w//4]
```

Image Augmentation with OpenCV

```
    resized = cv2.resize(crop, (w, h))
    augmented.append(('crop', resized))

    return augmented

# Save all versions
for name, aug_img in augment_opencv('cat.jpg'):
    cv2.imwrite(f'cat_{name}.jpg', aug_img)
```

Text Augmentation: Synonym Replacement

```python
import random

# Simple synonym dictionary
synonyms = {
    'good': ['great', 'excellent', 'nice', 'wonderful'],
    'bad': ['terrible', 'awful', 'poor', 'horrible'],
    'happy': ['joyful', 'cheerful', 'pleased', 'content']
}

def augment_text(text, n_aug=3):
    words = text.split()
    augmented = []

    for _ in range(n_aug):
        new_words = words.copy()
        # Replace 20% of words with synonyms
        for i, word in enumerate(new_words):
            if word.lower() in synonyms and random.random() < 0.2:
                new_words[i] = random.choice(synonyms[word.lower()])
```

Text Augmentation: Synonym Replacement

```
            augmented.append(' '.join(new_words))

        return augmented

# Example
text = "This is a good product"
for aug in augment_text(text):
    print(aug)
# Output:
# This is a great product
# This is a excellent product
# This is a wonderful product
```

Text Augmentation: Back Translation

```python
# Translate to another language and back
from googletrans import Translator

translator = Translator()

def back_translate(text, intermediate_lang='fr'):
    """Augment by translating to French and back"""
    # English → French
    translated = translator.translate(text, dest=intermediate_lang)

    # French → English
    back = translator.translate(translated.text, dest='en')

    return back.text

# Example
original = "Machine learning is fascinating"
augmented = back_translate(original)
print(f"Original: {original}")
```

Text Augmentation: Back Translation

```
print(f"Augmented: {augmented}")
# Might get: "Machine learning is captivating"
```

Tabular Data: SMOTE

```python
from imblearn.over_sampling import SMOTE
import numpy as np

# Original imbalanced data
X = np.array([[1, 2], [2, 3], [3, 4], [1, 3], [2, 2]])
y = np.array([0, 0, 1, 0, 0])  # Mostly class 0!

print(f"Original: {len(X)} samples")
print(f"Class 0: {sum(y == 0)}, Class 1: {sum(y == 1)}")

# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

print(f"\nAfter SMOTE: {len(X_resampled)} samples")
print(f"Class 0: {sum(y_resampled == 0)}, Class 1: {sum(y_resampled == 1)}")
```

Tabular Data: SMOTE

i Note
SMOTE creates synthetic samples between existing minority class samples

Exercise: Create Your Augmentation Pipeline

```
# TODO: Build a pipeline that:
# 1. Loads images from a folder
# 2. Applies 3 different augmentations to each
# 3. Saves all augmented images
# 4. Creates a CSV mapping filenames to labels

import os
from PIL import Image

def augmentation_pipeline(input_folder, output_folder):
    # Your code here
    pass

# Test it
augmentation_pipeline('original_images/', 'augmented_images/')
```

Summary & Best Practices

Key Takeaways

1. **Data Collection**
   - Use logging everywhere
   - Structure your data (JSON)
   - Respect privacy and robots.txt
2. **Data Validation**
   - Validate early, validate often
   - Use Pydantic for automatic checks
   - Handle missing data carefully
3. **Data Labeling**
   - Start small, iterate
   - Use active learning to be efficient
   - Measure inter-annotator agreement
4. **Data Augmentation**
   - Create variations thoughtfully
   - Don’t overaugment (keep it realistic)

Key Takeaways

- Test if augmentation helps your model

Tools Summary

```
# Data Collection
import logging           # Basic logging
import requests         # API calls
from bs4 import BeautifulSoup  # Web scraping

# Data Validation
from pydantic import BaseModel  # Type validation
import pandas as pd             # Data analysis

# Data Labeling
# label-studio (web-based tool)
from sklearn.metrics import cohen_kappa_score

# Data Augmentation
from PIL import Image           # Image processing
import cv2                      # Computer vision
from imblearn.over_sampling import SMOTE  # Tabular data
```

Common Pitfalls to Avoid

⚠ Warning

1. **Not validating data** → Models learn from bad data
2. **Over-augmenting** → Unrealistic variations
3. **Ignoring class imbalance** → Model ignores minority class
4. **Poor labeling guidelines** → Inconsistent labels
5. **Not versioning datasets** → Can't reproduce results

Your Assignment

Build a complete pipeline:

1. Collect 100 images from the web (with permission)
2. Create a validation script to check image sizes and formats
3. Manually label 20 images
4. Augment to create 100 labeled images total
5. Save everything in organized folders with a README

Deliverables:

- Python scripts
- Labeled dataset
- Documentation

Resources for Further Learning

Documentation:

- Pydantic: https://docs.pydantic.dev
- Label Studio: https://labelstud.io
- BeautifulSoup: https://beautiful-soup-4.readthedocs.io

Tutorials:

- Kaggle Learn: Data Cleaning
- Google's ML Crash Course: Data Preparation
- Fast.ai: Practical Deep Learning (Ch. 2)

Practice:

- Label images on Label Studio
- Try web scraping on practice sites
- Experiment with augmentation libraries

Questions?

Thank You!

Remember:

> "Data is the new oil, but only if refined properly!"

- Collect systematically
- Validate rigorously
- Label carefully
- Augment thoughtfully

Office Hours: Mon/Wed 2-3 PM

Email: nipun.batra@iitgn.ac.in

Next Class: Model Training & Evaluation