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?

> **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

```
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)

# Solution:
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(),
        "event_type": event_type,
        "user_id": user_id,
        "data": data
    }
    logging.info(json.dumps(event))

# Usage
log_event("video_watch", "user456", {
    "video_id": "abc123",
    "watch_time": 580
})
```

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

Web Scraping: Collecting Public Data

```
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]:
    print(quote.text)

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

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

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']}")

# Track over time
def track_repo_stats(repo_url):
    response = requests.get(repo_url)
    if response.status_code == 200:
        data = response.json()
        return {
            "date": datetime.datetime.now().isoformat(),
            "stars": data['stargazers_count'],
            "forks": data['forks_count']
        }
```

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"

⚠️ **Bad data → Bad predictions → Real harm!**

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=120)
    email: str
    height_cm: Optional[float] = Field(None, gt=0, lt=300)

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

# This works
user = User(name="Alice", age=25, email="alice@example.com")

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

Part 3: Data Labeling

What is Data Labeling?

Before labeling:

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

After labeling:

- Image → "dog"
- Text → "positive sentiment"
- Audio → "speech, female"
- Video → "car detected at (100, 200)"

> **Tip:** Labels are the "answers" we want our AI to learn!

Label Studio: Professional Tool

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

# Start server
label-studio start
```

Features:

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

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

    # Get indices of most uncertain samples
    uncertain_indices = np.argsort(uncertainty)[-n_samples:]

    return X_unlabeled[uncertain_indices]

# Usage
samples_to_label = active_learning(X_unlabeled, model)
print(f"Label these {len(samples_to_label)} samples next!")
```

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

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 slightly
- Add noise
- Change brightness
- Crop and resize

💡 **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. Horizontal flip
    img_flip = img.transpose(Image.FLIP_LEFT_RIGHT)
    img_flip.save(f"{output_folder}/flip.jpg")

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

    # 3. Adjust brightness
    enhancer = ImageEnhance.Brightness(img)
    img_bright = enhancer.enhance(random.uniform(0.7, 1.3))
    img_bright.save(f"{output_folder}/bright.jpg")

    # Now you have 4x more images!
```

Tabular Data: SMOTE

```
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, 0, 1, 1])  # Only 2 samples of class 1!

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

print(f"Original: {len(X)} samples")
print(f"After SMOTE: {len(X_resampled)} samples")
```

ℹ️ **SMOTE creates synthetic samples between existing minority class samples**

Summary & Best Practices

Key Takeaways

1. **Data Collection**
    - Use logging everywhere
    - Structure your data (JSON)
    - Respect privacy and robots.txt
2. **Data Validation**
    - Always validate at system boundaries
    - Use Pydantic for automatic checking
    - Monitor data quality continuously
3. **Data Labeling**
    - Use professional tools (Label Studio)
    - Use active learning to be efficient
    - Measure inter-annotator agreement

Tools Summary

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

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

# Data Labeling
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

⚠️ **Watch out for:**

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:

- 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