Week 1 Lab: Data Collection for ML

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Prof. Nipun Batra & Teaching Assistants

Your Mission 🎬

You're building the Netflix movie recommendation dataset.

Today you'll:

- ✅ Explore data sources with DevTools
- ✅ Test APIs with curl
- ✅ Build a movie data collector
- ✅ Handle errors like a pro
- ✅ Create a clean dataset ready for ML

End goal: CSV file with 100+ movies, ready for Week 2!

Lab Structure

Part 1: Tool Mastery (60 min)

- curl for API testing
- Chrome DevTools exploration
- HTTP basics hands-on

Part 2: API Data Collection (60 min)

- OMDb API integration
- Error handling
- Building the dataset

Part 3: Web Scraping (Optional) (45 min)

- BeautifulSoup basics

Setup (10 minutes)

Check Python version:

```
python --version  # Need 3.8+
```

Install packages:

```
pip install requests beautifulsoup4 pandas python-dotenv
```

For scraping challenges (optional):

```
pip install playwright
playwright install chromium
```

Get Your API Key

OMDb API (The Open Movie Database)

1. Visit: http://www.omdbapi.com/apikey.aspx
2. Select "FREE" (1,000 requests/day)
3. Enter your email
4. Check email for API key

Save it! You'll need it in 5 minutes.

Part 1: Tool Mastery

Master curl and Chrome DevTools

Exercise 1.1: curl Basics (10 min)

Test the OMDb API from terminal:

```
# Replace YOUR_KEY with your actual key
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Inception"
```

You should see JSON data!

Exercise 1.2: Pretty JSON with jq (5 min)

Install jq (JSON processor):

```
# macOS
brew install jq

# Ubuntu/Debian
sudo apt install jq

# Windows (Git Bash)
# Download from: https://jqlang.github.io/jq/download/
```

Try it:

```
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Inception" | jq
```

Much prettier! 🎨

Exercise 1.3: Explore the Response (10 min)

Extract specific fields with jq:

```
# Just the title
curl "..." | jq '.Title'

# Title and rating
curl "..." | jq '{title: .Title, rating: .imdbRating}'

# Check if response is valid
curl "..." | jq '.Response'
```

Task: Get the genre and year for "The Matrix"

Exercise 1.4: Search by IMDb ID (5 min)

```
# Search by title (can be ambiguous)
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Avatar"

# Search by IMDb ID (precise!)
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&i=tt0499549"
```

Find: What's the IMDb ID for "Shawshank Redemption"?

Hint: Visit IMDb page, look at URL

Exercise 1.5: Chrome DevTools (15 min)

Task: Inspect IMDb's website

1. Visit: https://www.imdb.com/title/tt0111161/
2. Right-click → Inspect → Network tab
3. Refresh page (Cmd+R / Ctrl+R)
4. Look for API calls (Fetch/XHR filter)

Questions:

- How many HTTP requests does the page make?
- Are there any JSON responses?
- What status codes do you see?

Exercise 1.6: Find the Data (15 min)

Task: Locate the rating on IMDb page

1. Inspect tab → Elements
2. Right-click on the rating → Inspect
3. Note the HTML structure:
   ```
   <span class="rating-value">9.3</span>
   ```

Goal: Understand where data lives in HTML

This is what we'd scrape if there was no API!

🎯 Checkpoint 1

You've mastered:

- ✅ Testing APIs with curl
- ✅ Formatting JSON with jq
- ✅ Chrome DevTools for inspection
- ✅ Understanding HTTP requests

Next: Build a Python data collector!

Part 2: API Data Collection

Build the Netflix dataset

Exercise 2.1: First API Call in Python (10 min)

Create: `movie_collector.py`

```python
import requests

API_KEY = 'your_key_here'  # Replace!

def get_movie(title):
    """Get movie data from OMDb API."""
    response = requests.get('http://www.omdbapi.com/', params={
        'apikey': API_KEY,
        't': title
    })

    return response.json()

# Test it!
movie = get_movie('Inception')
print(movie['Title'])
print(movie['imdbRating'])
```

Exercise 2.2: Safe API Keys (10 min)

Never hardcode API keys!

Create: `.env` file

```
OMDB_API_KEY=your_actual_key_here
```

Update code:

```python
import os
from dotenv import load_dotenv

load_dotenv()  # Load .env file

API_KEY = os.getenv('OMDB_API_KEY')

if not API_KEY:
    raise ValueError("API key not found! Check .env file")
```

Exercise 2.3: Error Handling (15 min)

Improve the function:

```python
def get_movie(title):
    """Get movie data with error handling."""
    try:
        response = requests.get('http://www.omdbapi.com/',
            params={'apikey': API_KEY, 't': title},
            timeout=10
        )
        response.raise_for_status()  # Raise exception for 4xx/5xx

        data = response.json()

        if data.get('Response') == 'False':
            print(f"Movie not found: {title}")
            return None

        return data

    except requests.exceptions.Timeout:
        print(f"Timeout fetching: {title}")
        return None
    except requests.exceptions.RequestException as e:
```

Exercise 2.4: Extract Useful Fields (10 min)

Clean up the response:

```python
def parse_movie(data):
    """Extract relevant fields from API response."""
    if not data:
        return None

    return {
        'title': data.get('Title'),
        'year': data.get('Year'),
        'rating': data.get('imdbRating'),
        'genre': data.get('Genre'),
        'director': data.get('Director'),
        'actors': data.get('Actors'),
        'plot': data.get('Plot'),
        'box_office': data.get('BoxOffice', 'N/A'),
        'imdb_id': data.get('imdbID')
    }

# Test
data = get_movie('Inception')
```

Exercise 2.5: Batch Collection (15 min)

Collect multiple movies:

```python
def collect_movies(titles):
    """Collect data for multiple movies."""
    movies = []

    for i, title in enumerate(titles, 1):
        print(f"Fetching {i}/{len(titles)}: {title}")

        data = get_movie(title)
        movie = parse_movie(data)

        if movie:
            movies.append(movie)

    return movies

# Test with a few movies
titles = ['Inception', 'The Matrix', 'Interstellar', 'Shawshank Redemption']
movies = collect_movies(titles)
```

Exercise 2.6: Save to CSV (10 min)

Store the dataset:

```python
import pandas as pd

def save_to_csv(movies, filename='movies.csv'):
    """Save movies to CSV file."""
    df = pd.DataFrame(movies)
    df.to_csv(filename, index=False)
    print(f"✓ Saved {len(movies)} movies to {filename}")

# Save your data
save_to_csv(movies)
```

Check the file:

```
head movies.csv
```

Exercise 2.7: IMDb Top 100 (Challenge!)

Goal: Collect IMDb's top rated movies

Starter code:

```
# IMDb IDs for top 100 movies
top_100_ids = [
    'tt0111161',  # Shawshank Redemption
    'tt0068646',  # The Godfather
    'tt0468569',  # The Dark Knight
    # ... add more IDs
]

def get_movie_by_id(imdb_id):
    """Get movie by IMDb ID (more reliable than title)."""
    response = requests.get('http://www.omdbapi.com/',
        params={'apikey': API_KEY, 'i': imdb_id}
    )
    return response.json()
```

🎯 Checkpoint 2

You've built:

- ✅ API client with error handling
- ✅ Data parser for movie info
- ✅ Batch collector
- ✅ CSV export functionality

You have a real dataset now! 🎉

Part 3: Web Scraping (Optional)

For when there's no API

Exercise 3.1: Why Scrape? (5 min)

Discussion:

- OMDb API is great, but limited (1,000 requests/day)
- Some sites don't have APIs
- Sometimes you need data API doesn't provide

Today's challenge:

Scrape IMDb directly (educational purposes only!)

Exercise 3.2: Fetch HTML (10 min)

Create: `scraper.py`

```python
import requests
from bs4 import BeautifulSoup

def fetch_movie_page(imdb_id):
    """Fetch IMDb movie page HTML."""
    url = f'https://www.imdb.com/title/{imdb_id}/'

    headers = {
        'User-Agent': 'Mozilla/5.0 (Educational purposes)'
    }

    response = requests.get(url, headers=headers)
    response.raise_for_status()

    return response.text

# Test
```

Exercise 3.3: Parse with BeautifulSoup (15 min)

```python
def parse_movie_page(html):
    """Extract movie info from IMDb HTML."""
    soup = BeautifulSoup(html, 'html.parser')

    # Find title (inspect page to get correct selectors)
    title_elem = soup.find('h1')
    title = title_elem.text.strip() if title_elem else None

    # Find rating
    rating_elem = soup.find('span', class_='rating-value')
    rating = rating_elem.text if rating_elem else None

    return {
        'title': title,
        'rating': rating
    }

# Test
html = fetch_movie_page('tt0111161')
movie = parse_movie_page(html)
print(movie)
```

Exercise 3.4: Compare Approaches (10 min)

Create a comparison:

```python
import time

# API approach
start = time.time()
api_data = get_movie('Inception')
api_time = time.time() - start

# Scraping approach
start = time.time()
html = fetch_movie_page('tt1375666')
scrape_data = parse_movie_page(html)
scrape_time = time.time() - start

print(f"API: {api_time:.2f}s")
print(f"Scraping: {scrape_time:.2f}s")
```

Exercise 3.5: Playwright (Advanced, 10 min)

For JavaScript-heavy sites:

```python
from playwright.sync_api import sync_playwright

def scrape_dynamic_site(url):
    """Scrape sites that load content with JavaScript."""
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()

        page.goto(url)
        page.wait_for_selector('.movie-title')  # Wait for content

        html = page.content()
        browser.close()

        return html
```

When to use: React/Vue/Angular sites, infinite scroll

🎯 Checkpoint 3

You've learned:

- ✅ Web scraping basics
- ✅ BeautifulSoup for parsing
- ✅ When to scrape vs use APIs
- ✅ Playwright for dynamic sites

Key takeaway: APIs > Scraping (when available)

Part 4: Challenge Projects

Extend your skills

Challenge 1: Multi-Source Dataset ⭐

Combine data from multiple APIs:

```
# OMDb for basic info
omdb_data = get_movie('Inception')

# TODO: Add TMDb API for additional data
# (Budget, revenue, popularity)

# TODO: Add News API for recent mentions
# (Buzz score)

# Combine into rich dataset
```

APIs to try:

- TMDb: https://www.themoviedb.org/documentation/api
- News API: https://newsapi.org/

Challenge 2: Rate Limit Handler ⭐⭐

Handle rate limits gracefully:

```python
import time

def get_movie_with_retry(title, max_retries=3):
    """Retry on rate limit (429) or server errors."""
    for attempt in range(max_retries):
        response = requests.get(url, params=params)

        if response.status_code == 429:
            wait_time = 2 ** attempt  # Exponential backoff
            print(f"Rate limited. Waiting {wait_time}s...")
            time.sleep(wait_time)
            continue

        return response.json()

    raise Exception("Max retries exceeded")
```

Challenge 3: Data Quality Checks ⭐⭐

Validate your dataset:

```
def validate_dataset(movies):
    """Check data quality."""
    issues = []

    for movie in movies:
        # Check for missing critical fields
        if not movie.get('title'):
            issues.append(f"Missing title: {movie}")

        # Check rating is valid
        rating = movie.get('rating', 'N/A')
        if rating != 'N/A':
            try:
                r = float(rating)
                if not (0 <= r <= 10):
                    issues.append(f"Invalid rating: {rating}")
            except ValueError:
                issues.append(f"Non-numeric rating: {rating}")

    return issues

# Run validation
```

Challenge 4: Genre Analysis ⭐⭐⭐

Analyze your dataset:

```python
import pandas as pd
import matplotlib.pyplot as plt

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

# Split genres (they're comma-separated)
df['genre_list'] = df['genre'].str.split(', ')

# Count genre occurrences
from collections import Counter
all_genres = []
for genres in df['genre_list'].dropna():
    all_genres.extend(genres)

genre_counts = Counter(all_genres)

# Plot
pd.Series(genre_counts).sort_values().plot(kind='barh')
plt.title('Movie Genres in Dataset')
```

Challenge 5: Async Collection ⭐⭐⭐⭐

Speed up with async requests:

```python
import asyncio
import httpx

async def get_movie_async(client, title):
    """Async version of get_movie."""
    response = await client.get('http://www.omdbapi.com/',
        params={'apikey': API_KEY, 't': title}
    )
    return response.json()

async def collect_movies_async(titles):
    """Collect multiple movies in parallel."""
    async with httpx.AsyncClient() as client:
        tasks = [get_movie_async(client, title) for title in titles]
        results = await asyncio.gather(*tasks)
    return results

# Run it
```

Best Practices Checklist ✅

Before you submit:

- [ ] API keys in `.env` (not in code!)
- [ ] `.env` in `.gitignore`
- [ ] Error handling for network issues
- [ ] Timeouts on all requests
- [ ] Respect rate limits
- [ ] Validate data before saving
- [ ] Document your code
- [ ] Test with small dataset first

Submission Requirements

Submit on Moodle:

1. **Code**: `movie_collector.py`
2. **Dataset**: `movies.csv` (minimum 50 movies)
3. **README**: Explain your approach
4. **Analysis** (optional): Genre distribution plot

Bonus points:

- Multi-source data (OMDb + others)
- Async implementation
- Data quality validation
- Creative analysis/visualization

Common Issues & Solutions

Issue: "Invalid API key"

- Solution: Check `.env` file, verify key on OMDb website

Issue: Rate limit (429)

- Solution: Implement backoff, or wait before retrying

Issue: Movie not found

- Solution: Use IMDb ID instead of title

Issue: Connection timeout

- Solution: Add `timeout=10` parameter

Issue: SSL certificate error

- Solution: `requests.get(..., verify=False)` (not recommended for production!)

Resources

Documentation:

- requests: https://requests.readthedocs.io/
- BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/
- pandas: https://pandas.pydata.org/
- python-dotenv: https://pypi.org/project/python-dotenv/

APIs:

- OMDb: http://www.omdbapi.com/
- TMDb: https://www.themoviedb.org/documentation/api

Practice:

- JSONPlaceholder: https://jsonplaceholder.typicode.com/

What's Next?

Week 2: Data Validation with Pydantic

- Clean your messy data
- Type checking
- Schema validation
- Handle missing values

Preparation:

- Keep your `movies.csv`
- Think about data quality issues you found
- What fields are missing or inconsistent?

Lab Summary

What you built:

- ✅ Movie data collector using OMDb API
- ✅ Error handling and retry logic
- ✅ CSV dataset export
- ✅ Data quality checks

Skills gained:

- HTTP/API fundamentals
- Python requests library
- Data processing with pandas
- Real-world ML data collection

Questions?

TAs are here to help!

Start with Exercise 1.1 and work your way through.

Good luck! 🎬