

Week 1 Lab: Data Collection for ML

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Prof. Nipun Batra & Teaching Assistants

Lab Overview

Goal: Build a movie data collection system for our Netflix recommendation problem

What you'll build:

- curl commands to test OMDb API
- Python scripts to collect movie data
- Web scraper for additional movie information
- Complete dataset of 50+ movies with features

Skills practiced:

- Using command-line tools (curl, jq)
- Python requests library
- BeautifulSoup for HTML parsing

Setup Check

Verify your environment:

```
# Check Python version (need 3.8+)
python --version

# Install required packages
pip install requests beautifulsoup4 \
python-dotenv pandas

# Create project directory
mkdir movie-collector
cd movie-collector
```

Part 1: Command Line Tools

Getting started with curl and jq

Exercise 1.1: Your First API Call

Task: Get information about "Inception" using curl

Get an API key first:

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

Save the key for later use!

Exercise 1.1: Solution

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

Output (unformatted JSON):

```
{"Title":"Inception","Year":"2010","Rated":"PG-13",  
  "Released":"16 Jul 2010","Runtime":"148 min",  
"Genre":"Action, Sci-Fi, Thriller","Director":"Christopher Nolan",  
  "imdbRating":"8.8","imdbVotes":"2,535,646",...}
```

Problem: Hard to read!

Exercise 1.2: Format with jq

Task: Make the JSON readable

Install jq (if not already installed):

```
# Mac
brew install jq

# Linux
sudo apt-get install jq

# Windows: download from stedolan.github.io/jq/
```

Use it:

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

Exercise 1.2: Output

Now it's nicely formatted:

```
{
  "Title": "Inception",
  "Year": "2010",
  "Rated": "PG-13",
  "Released": "16 Jul 2010",
  "Runtime": "148 min",
  "Genre": "Action, Sci-Fi, Thriller",
  "Director": "Christopher Nolan",
  "imdbRating": "8.8",
  "imdbVotes": "2,535,646",
  "BoxOffice": "$292,587,330"
}
```

Much better!

Exercise 1.3: Extract Specific Fields

Task: Get only title, year, and rating

```
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Inception" | \
jq '{title: .Title, year: .Year, rating: .imdbRating}'
```

Output:

```
{
  "title": "Inception",
  "year": "2010",
  "rating": "8.8"
}
```

jq is powerful for filtering JSON!

Exercise 1.4: Multiple Movies

Task: Get data for 3 movies using a loop

```
# Create a file with movie titles
echo -e "Inception\nThe Matrix\nInterstellar" > movies.txt

# Loop through and fetch each
while read movie; do
    echo "Fetching: $movie"
    curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=$movie" | \
    jq '{title: .Title, rating: .imdbRating}'
    sleep 1 # Be polite to the API
done < movies.txt
```

Part 1 Checkpoint

What you've learned:

- Using curl to make HTTP GET requests
- Formatting JSON with jq
- Filtering JSON fields with jq syntax
- Batch processing with shell loops

Why this matters:

- Quick API testing without writing code
- Understanding API responses before coding
- Command-line tools are fast and powerful

Part 2: Python Data Collection

Building a production collector

Exercise 2.1: Basic API Call

Task: Fetch movie data in Python

Create file: `get_movie.py`

```
import requests

api_key = "YOUR_KEY" # We'll fix this soon!
title = "Inception"

url = "http://www.omdbapi.com/"
params = {
    "apikey": api_key,
    "t": title
}

response = requests.get(url, params=params)
print(response.json())
```

Exercise 2.1: Run It

```
python get_movie.py
```

Output:

```
{
  'Title': 'Inception',
  'Year': '2010',
  'imdbRating': '8.8',
  'Genre': 'Action, Sci-Fi, Thriller',
  ...
}
```

Success! But hardcoding the API key is bad practice.

Exercise 2.2: Use Environment Variables

Task: Secure your API key with .env file

Step 1: Install python-dotenv

```
pip install python-dotenv
```

Step 2: Create `.env` file

```
OMDB_API_KEY=your_actual_key_here
```

Step 3: Add to `.gitignore`

```
.env
```

Exercise 2.2: Updated Code

Update `get_movie.py` :

```
import requests
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()
api_key = os.getenv("OMDB_API_KEY")

title = "Inception"
url = "http://www.omdbapi.com/"
params = {"apikey": api_key, "t": title}

response = requests.get(url, params=params)
data = response.json()

print(f"Title: {data['Title']}")
print(f"Year: {data['Year']}")
```


Exercise 2.3: Add Error Handling

Task: Handle network errors and API failures

Create file: `collector.py`

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

def get_movie_data(title):
    api_key = os.getenv("OMDB_API_KEY")
    url = "http://www.omdbapi.com/"
    params = {"apikey": api_key, "t": title}

    try:
        response = requests.get(url,
                                params=params,
                                timeout=10)
```

Exercise 2.3: Error Handling (continued)

```
        # Check if movie was found
        if data.get("Response") == "False":
            print(f"Error: {data.get('Error')}")
            return None

        return data

    except requests.exceptions.Timeout:
        print("Request timed out")
        return None
except requests.exceptions.RequestException as e:
    print(f"Request failed: {e}")
    return None

# Test it
movie = get_movie_data("Inception")
if movie:
    print(f"Found: {movie['Title']}")
```

Exercise 2.4: Collect Multiple Movies

Task: Build a list of movie data

```
import requests
import os
import time
from dotenv import load_dotenv

load_dotenv()

def get_movie_data(title):
    # [Previous function code here]
    pass

# List of movies to collect
movies_to_fetch = [
    "Inception", "The Matrix", "Interstellar",
    "The Dark Knight", "Pulp Fiction",
    "The Shawshank Redemption"
]
```

Exercise 2.4: Collection Loop

```
collected_movies = []

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

    movie_data = get_movie_data(title)

    if movie_data:
        collected_movies.append(movie_data)
        print(f"Success! Rating: {movie_data['imdbRating']}")
    else:
        print(f"Failed to fetch {title}")

    # Be polite to the API
    time.sleep(1)

print(f"\nCollected {len(collected_movies)} movies")
```

Exercise 2.5: Save to JSON

Task: Save collected data for later use

```
import json

# Save raw data
with open('movies_raw.json', 'w') as f:
    json.dump(collected_movies, f, indent=2)

print("Saved to movies_raw.json")
```

Why save raw data?

- Can reprocess without re-fetching
- Debugging is easier
- Keeps original data intact

Exercise 2.6: Extract Features

Task: Create a clean dataset with only needed features

```
import pandas as pd

# Extract features we need
features_list = []

for movie in collected_movies:
    features = {
        'title': movie.get('Title'),
        'year': movie.get('Year'),
        'genre': movie.get('Genre'),
        'director': movie.get('Director'),
        'rating': movie.get('imdbRating'),
        'votes': movie.get('imdbVotes'),
        'runtime': movie.get('Runtime'),
        'box_office': movie.get('BoxOffice')
    }
    features_list.append(features)
```

Exercise 2.6: Save to CSV

```
# Create DataFrame
df = pd.DataFrame(features_list)

# Save to CSV
df.to_csv('movies.csv', index=False)

# Display
print(df.head())
```

Output:

	title	year	genre	director	rating
0	Inception	2010	Action, Sci-Fi, Thriller	Christopher Nolan	8.8
1	The Matrix	1999	Action, Sci-Fi	The Wachowskis	8.7
2	Interstellar	2014	Adventure, Drama, Sci-Fi	Christopher Nolan	8.7

Part 2 Checkpoint

What you've learned:

- Making API calls with requests library
- Handling errors gracefully
- Using environment variables for secrets
- Collecting data in batches
- Saving data to JSON and CSV

Your progress: You now have a working movie data collector!

Part 3: Web Scrapping

Getting data not available via API

Exercise 3.1: Inspect HTML

Task: Understand website structure

Website: <http://quotes.toscrape.com>

1. Open in Chrome
2. Right-click on a quote
3. Select "Inspect"
4. See the HTML structure

Observations:

- Each quote is in a `<div class="quote">`
- Text is in ``
- Author is in `<small class="author">`

Exercise 3.2: Your First Scraper

Task: Scrape quotes from the website

Create file: `scrape_quotes.py`

```
import requests
from bs4 import BeautifulSoup

url = 'http://quotes.toscrape.com/'
response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

# Find all quote containers
quotes = soup.find_all('div', class_='quote')

print(f"Found {len(quotes)} quotes")
```

Exercise 3.2: Extract Quote Data

```
    for quote in quotes:
        # Extract text
        text = quote.find('span', class_='text').text

        # Extract author
        author = quote.find('small', class_='author').text

        # Extract tags
        tag_elements = quote.find_all('a', class_='tag')
        tags = [tag.text for tag in tag_elements]

        print(f"\nQuote: {text}")
        print(f"Author: {author}")
        print(f"Tags: {' , '.join(tags)}")
```

Exercise 3.3: Handle Multiple Pages

Task: Scrape quotes from all pages

```
import requests
from bs4 import BeautifulSoup
import time

base_url = 'http://quotes.toscrape.com'
all_quotes = []
page = 1

while True:
    url = f'{base_url}/page/{page}/'
    print(f"Scraping page {page}...")

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    quotes = soup.find_all('div', class_='quote')
```

Exercise 3.3: Pagination Loop

```
# If no quotes found, we're done
    if not quotes:
        print("No more quotes found")
        break

# Process quotes on this page
    for quote in quotes:
        text = quote.find('span', class_='text').text
        author = quote.find('small', class_='author').text
        all_quotes.append({
            'quote': text,
            'author': author
        })

    page += 1
    time.sleep(1) # Be polite

print(f"Total quotes scraped: {len(all_quotes)}")
```

Exercise 3.4: Handle Missing Elements

Task: Make scraper robust to missing data

```
def extract_quote_data(quote_element):  
    # Safely extract text  
    text_elem = quote_element.find('span', class_='text')  
    text = text_elem.text if text_elem else "No text"  
  
    # Safely extract author  
    author_elem = quote_element.find('small', class_='author')  
    author = author_elem.text if author_elem else "Unknown"  
  
    # Safely extract tags  
    tag_elements = quote_element.find_all('a', class_='tag')  
    tags = [tag.text for tag in tag_elements] if tag_elements else []  
  
    return {  
        'text': text,  
        'author': author,  
        'tags': tags
```

Part 3 Checkpoint

What you've learned:

- Inspecting HTML structure with DevTools
- Using BeautifulSoup to parse HTML
- Finding elements by tag, class, and id
- Handling pagination
- Defensive programming (missing elements)

Next: Apply to real movie data!

Part 4: Mini Project

Build your own movie data collector

Project Requirements

Goal: Collect data for 50+ movies from OMDb API

Required features:

- Title, year, genre, director
- IMDb rating, number of votes
- Runtime, box office revenue

Bonus features:

- Plot summary
- Main actors
- Awards
- Multiple language support

Project Structure

Create this file structure:

```
movie-collector/
├── .env                # API keys (don't commit!)
│   ├── .gitignore     # Ignore .env
│   ├── collector.py   # Main collection script
│   ├── utils.py       # Helper functions
│   ├── requirements.txt # Dependencies
│   └── data/
│       ├── raw/        # Raw API responses
│       └── processed/  # Cleaned CSV
└── README.md          # Documentation
```

Step 1: Movie List

Create `movies.txt` with movie titles (one per line):

```
The Shawshank Redemption
The Godfather
The Dark Knight
Pulp Fiction
Forrest Gump
Inception
Fight Club
The Matrix
Goodfellas
The Lord of the Rings: The Return of the King
...
```

Tip: Use IMDb Top 250 list for ideas

Step 2: Collector Function

Create `utils.py` :

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

def get_movie_by_title(title):
    """Fetch movie data from OMDb API"""
    api_key = os.getenv("OMDB_API_KEY")
    url = "http://www.omdbapi.com/"

    params = {
        "apikey": api_key,
        "t": title,
        "plot": "full" # Get full plot
    }

    try:
        response = requests.get(url,
                                params=params,
                                timeout=10)
```

Step 2: Error Handling

```
if data.get("Response") == "False":  
    print(f"    Error: {data.get('Error')}")  
    return None  
  
    return data  
  
except requests.exceptions.Timeout:  
    print("    Request timed out")  
    return None  
except requests.exceptions.RequestException as e:  
    print(f"    Request failed: {e}")  
    return None
```

Step 3: Main Collection Script

Create `collector.py` :

```
import json
import time
from pathlib import Path
from utils import get_movie_by_title

# Setup directories
Path("data/raw").mkdir(parents=True, exist_ok=True)
Path("data/processed").mkdir(parents=True, exist_ok=True)

# Read movie titles
with open('movies.txt') as f:
    titles = [line.strip() for line in f if line.strip()]

print(f"Collecting data for {len(titles)} movies...\n")
```

Step 3: Collection Loop

```
        collected = []
        failed = []

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

        movie_data = get_movie_by_title(title)

        if movie_data:
            collected.append(movie_data)
            print(f"    Success! Rating: {movie_data.get('imdbRating')}")

            # Save individual JSON for debugging
            filename = title.replace(' ', '_').replace(':', '') + '.json'
            with open(f'data/raw/{filename}', 'w') as f:
                json.dump(movie_data, f, indent=2)
        else:
            failed.append(title)

    time.sleep(1) # Rate limiting
```


Step 3: Save Results

```
# Save all collected data
with open('data/raw/all_movies.json', 'w') as f:
    json.dump(collected, f, indent=2)

    print(f"\n{'='*50}")
    print(f"Collection complete!")
print(f"Successful: {len(collected)}")
print(f"Failed: {len(failed)}")

    if failed:
        print(f"\nFailed movies:")
        for title in failed:
            print(f"    - {title}")

# Save failed titles for retry
with open('failed_movies.txt', 'w') as f:
    f.write('\n'.join(failed))
```

Step 4: Data Processing

Create `process_data.py` :

```
import json
import pandas as pd

# Load raw data
with open('data/raw/all_movies.json') as f:
    movies = json.load(f)

# Extract features
features = []
for movie in movies:
    features.append({
        'title': movie.get('Title'),
        'year': movie.get('Year'),
        'genre': movie.get('Genre'),
        'director': movie.get('Director'),
        'actors': movie.get('Actors'),
        'runtime': movie.get('Runtime'),
```

Step 4: Feature Extraction

```
        'rating': movie.get('imdbRating'),
        'votes': movie.get('imdbVotes'),
        'box_office': movie.get('BoxOffice'),
        'awards': movie.get('Awards'),
        'plot': movie.get('Plot'),
        'language': movie.get('Language'),
        'country': movie.get('Country')
    })

    # Create DataFrame
    df = pd.DataFrame(features)

    # Save to CSV
    df.to_csv('data/processed/movies.csv', index=False)

    print(f"Processed {len(df)} movies")
    print(f"\nDataset shape: {df.shape}")
    print(f"\nMissing values:\n{df.isnull().sum()}")
```

Step 5: Data Validation

Add validation to `process_data.py`:

```
# Check for missing critical fields
critical_fields = ['title', 'year', 'rating']

for field in critical_fields:
    missing = df[field].isnull().sum()
    if missing > 0:
        print(f"Warning: {missing} movies missing {field}")

# Check data types
print("\nData types:")
print(df.dtypes)

# Check rating range
df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
print(f"\nRating range: {df['rating'].min()} to {df['rating'].max()}")
```

Step 6: Exploratory Analysis

Create `analyze.py` :

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

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

# Basic statistics
print("Dataset Statistics:")
print(f"Total movies: {len(df)}")
print(f"Average rating: {df['rating'].mean():.2f}")
print(f"Year range: {df['year'].min()} – {df['year'].max()}")

# Most common genres
print("\nTop 5 Genres:")
genres = df['genre'].str.split(', ').explode()
print(genres.value_counts().head())
```

Step 6: Visualizations

```
# Rating distribution
plt.figure(figsize=(10, 5))
df['rating'].hist(bins=20)
plt.xlabel('IMDb Rating')
plt.ylabel('Number of Movies')
plt.title('Distribution of Movie Ratings')
plt.savefig('data/processed/rating_distribution.png')
plt.close()

# Movies per year
plt.figure(figsize=(12, 5))
df['year'].value_counts().sort_index().plot(kind='bar')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.title('Movies per Year')
plt.savefig('data/processed/movies_per_year.png')
plt.close()

print("\nVisualizations saved!")
```

Step 7: Documentation

Create `README.md` :

Movie Data Collector

Collects movie data from OMDb API for machine learning.

Setup

1. Get API key from <http://www.omdbapi.com/apikey.aspx>
2. Create `.env` file with `OMDB_API_KEY=your_key`
3. Install dependencies: `pip install -r requirements.txt`

Usage

1. Add movie titles to `movies.txt`
2. Run collector: `python collector.py`
3. Process data: `python process_data.py`
4. Analyze: `python analyze.py`

Dataset

– **Size**: 50+ movies

Step 8: Requirements File

Create `requirements.txt` :

```
requests==2.31.0  
beautifulsoup4==4.12.2  
python-dotenv==1.0.0  
pandas==2.1.0  
matplotlib==3.7.2
```

Install with:

```
pip install -r requirements.txt
```


Testing Your Collector

Run the complete pipeline:

```
# 1. Collect data
python collector.py

# 2. Process data
python process_data.py

# 3. Analyze data
python analyze.py

# 4. Check results
cat data/processed/movies.csv
```

Expected output: CSV file with 50+ movies and all features

Common Issues and Solutions

Issue: API returns 401 Unauthorized

Solution: Check your API key in .env file

Issue: Some movies not found

Solution: Check spelling in movies.txt, some titles need exact match

Issue: Missing box office data

Solution: Not all movies have this field, that's OK

Issue: Rate limit errors

Solution: Increase sleep time between requests

Extending the Project

Ideas to make it better:

1. **Add TMDb API:** Get additional features like budget
2. **Web scraping:** Get critic reviews from Rotten Tomatoes
3. **Async collection:** Use `httplib` for faster collection
4. **Caching:** Don't re-fetch movies you already have
5. **Resume capability:** Continue from where you stopped
6. **Logging:** Better tracking of collection progress
7. **Data cleaning:** Handle missing values systematically

Lab Wrap-up

What you've accomplished:

- Built command-line data collection tools
- Created a production Python collector
- Learned web scraping with BeautifulSoup
- Collected a complete movie dataset
- Validated and analyzed the data

Next week: Data Validation

- Clean and validate the collected data
- Use Pydantic for schema validation
- Analyze with csvkit and pandas
- Handle missing values and outliers

Homework

Before next class:

1. **Extend your dataset:** Collect 100+ movies
2. **Add features:** Get plot summaries, awards
3. **Explore the data:** Find interesting patterns
4. **Document:** Update README with findings

Bonus challenges:

- Collect data from TMDb API
- Scrape critic reviews from a website
- Create visualizations of your dataset
- Share your dataset with classmates

Resources

APIs:

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

Documentation:

- requests: <https://requests.readthedocs.io/>
- BeautifulSoup: <https://www.crummy.com/software/BeautifulSoup/>
- pandas: <https://pandas.pydata.org/>

Tools:

- curl: <https://curl.se/>
- jq: <https://stedolan.github.io/jq/>

Questions?

Remember:

- Start small, test with 5-10 movies first
- Save raw data before processing
- Handle errors gracefully
- Respect API rate limits
- Document your code

Get help:

- Teaching assistants during lab
- Discussion forum for questions
- Office hours this week