Web Scraping

With Using Scrapy library

gathered by Ramin Kohandel

WEB MINING

Web mining is the process of using data mining techniques to discover and extract useful information from the web. It involves analyzing data from web pages, social media, and other online sources to extract insights and trends. Web mining can be used for a wide range of purposes, such as market research, sentiment analysis, and fraud detection.

Here is an outline of the main steps involved in web mining:

1. Data collection
2. Data preprocessing
3. Data analysis
4. Data visualization
5. Data interpretation

Web mining involves a wide range of techniques and tools, and it can be a complex process. Familiarity with data mining algorithms and techniques, as well as programming skills, is usually necessary to effectively perform web mining.

1.DATA COLLECTION STEP

At this point, you are responsible for identifying the target website and determining the specific data that you want to extract. This involves understanding the structure and organization of the website, as well as identifying any potential barriers to data extraction (such as login requirements or rate limiting).

There are a few different approaches you can take when collecting data from a website:

- **Web scraping:** This involves using a tool or writing code to extract data from the HTML source code of a website. This is useful when the data is structured in a predictable way and is easily accessible from the HTML.
- **API access:** Many websites provide APIs that allow you to access data directly, rather than having to scrape it from the HTML source code. This can be a more efficient and reliable way to access data, but it requires you to follow the API's specific guidelines and protocols.
- **Downloading data directly:** In some cases, the data you want may be available for direct download from the website, either as a file (such as a CSV or Excel spreadsheet) or as a structured API endpoint.

Once you have identified the data you want to collect and the method you will use to collect it, you can proceed to the next step of writing the code to extract the data.

2.DATA PROCESSING STEP

The data processing step in data mining is the process of preparing data for analysis. It involves a series of tasks that are designed to clean, transform, and integrate data from various sources in order to make it more suitable for analysis.

Here are the main steps involved in the data processing of data mining:

- **Data cleaning:** Data cleaning involves identifying and correcting or removing data that is incorrect, incomplete, or inconsistent. This may involve tasks such as identifying and filling in missing values, identifying and correcting errors, and standardizing data formats.
- **Data transformation:** Data transformation involves converting data from one format to another, or aggregating data from multiple sources. This may involve tasks such as merging data sets, pivoting data, or aggregating data by certain criteria.
- **Data integration:** Data integration involves combining data from multiple sources into a single, consistent dataset. This may involve tasks such as de-duplicating data, reconciling data from different sources, and resolving conflicts between data elements.
- **Data reduction:** Data reduction involves reducing the size of the dataset by selecting a subset of the data or summarizing the data in some way. This may involve tasks such as selecting a random sample of the data, or aggregating data by certain criteria.
- **Data discretization:** Data discretization involves converting continuous data into a set of discrete values or bins. This may be necessary when working with data that is too large or complex to be analyzed directly.

Data processing is an important step in the data mining process, as it helps to ensure that the data is clean, consistent, and ready for analysis. It is usually necessary to perform data processing before running data mining algorithms on the data.

3.DATA ANALYSIS STEP

Data analysis is the process of examining and interpreting data to draw insights and conclusions. In the context of web mining, this step involves analyzing the data that you have collected from the web to gain a deeper understanding of the trends, patterns, and relationships present in the data.

There are many different approaches you can take when analyzing web mining data, depending on the goals of your analysis and the characteristics of the data. Some common techniques include:

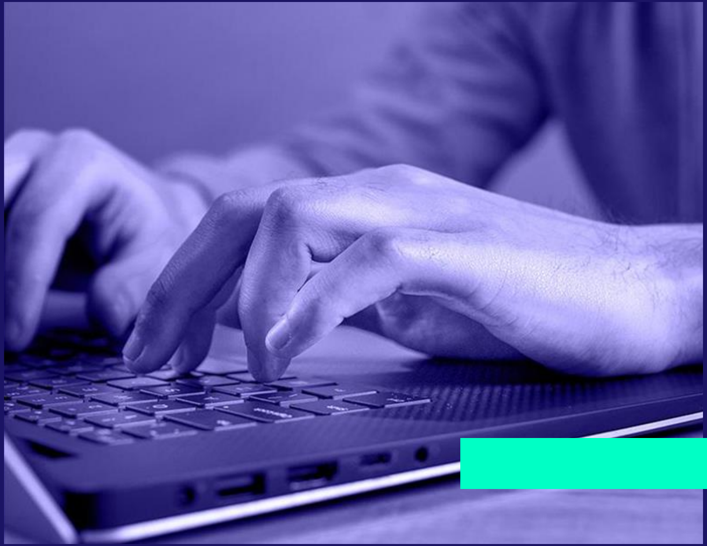
- Descriptive analysis: This involves summarizing the main characteristics of the data, such as calculating averages, frequencies, and standard deviations.
- Exploratory analysis: This involves examining the data in more detail to discover patterns and relationships that might not be immediately apparent. This can involve using visualizations to help identify trends and outliers.
- Inferential analysis: This involves using statistical techniques to make inferences about a larger population based on a sample of data.
- Predictive analysis: This involves using machine learning techniques to build models that can predict future outcomes based on historical data.

It's important to carefully consider the goals of your analysis and choose the appropriate techniques to achieve those goals. It can also be useful to involve domain experts in the analysis process, as they may be able to provide valuable insights and context.

WEB SCRAPING

Web scraping refers to the extraction of data from a website. It is a way to extract large amounts of data from websites, fast. It allows you to get specific information from a website in a structured format, such as a spreadsheet or a database. There are many tools available for web scraping, including web browser extensions and specialized software.

Web scraping can be used for a variety of purposes, such as data mining, data analysis, and automated testing. It is important to respect the terms of service of a website and the privacy of its users when using web scraping techniques.



WEB SCRAPING VS WEB MINING

Web scraping and web mining are related but distinct concepts.

Web scraping refers to the process of extracting specific data from a website. It involves making a request to a website's server and receiving a response, which is typically in the form of HTML or XML. The data is then extracted from the response and stored in a structured format, such as a spreadsheet or a database. Web scraping is often used to collect large amounts of data from websites, fast.

Web mining, on the other hand, is the process of using data mining techniques to discover and extract useful information from the World Wide Web. It involves the use of algorithms to analyze and extract patterns and trends from large datasets. Web mining can be used to discover insights about customer behavior, predict outcomes, and identify relationships.

In summary, web scraping is the act of extracting specific data from a website, while web mining is the use of data mining techniques to discover useful information from the World Wide Web.

WEB SCRAPING VS WEB CRAWLING

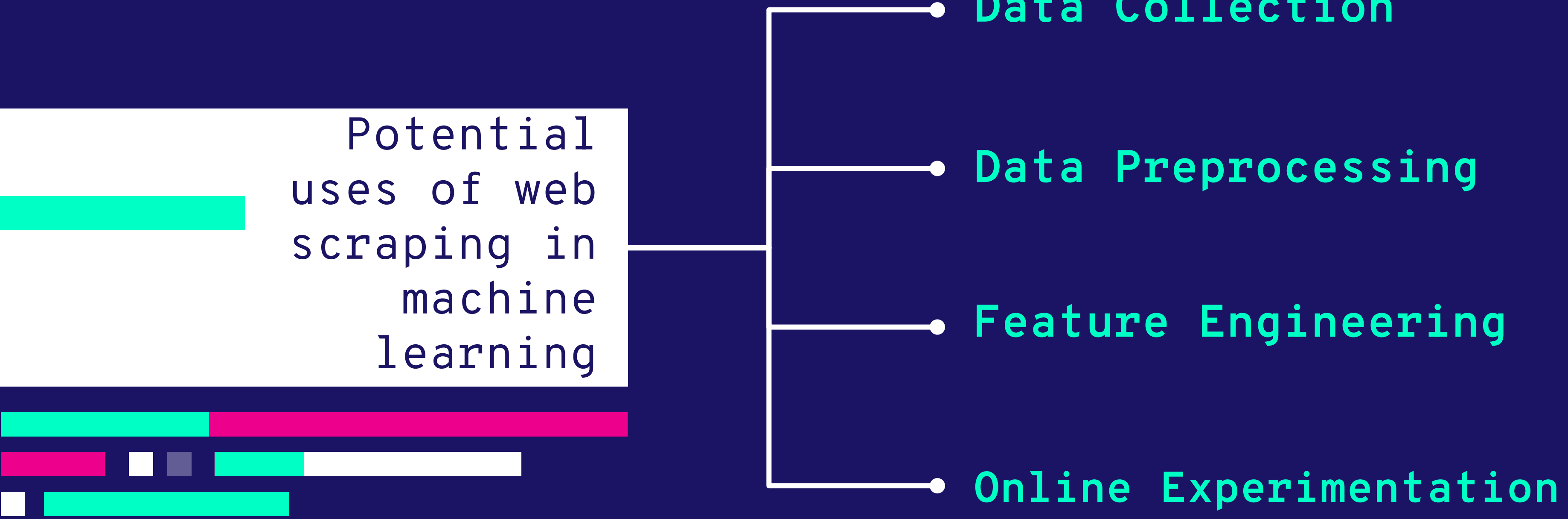
Web scraping and web crawling are often used interchangeably, but they are not exactly the same thing.

Web scraping refers to the extraction of data from a website. It involves making a request to a website's server and receiving a response, which is typically in the form of HTML or XML. The data is then extracted from the response and stored in a structured format, such as a spreadsheet or a database.

Web crawling, on the other hand, is the process of automatically visiting web pages and following links to other pages within the same website or to other websites. It is a way to discover new and updated content on the internet. Web crawlers, also known as spiders or bots, are used by search engines to index websites and by online marketplaces to discover new products.

In summary, web scraping is the act of extracting specific data from a website, while web crawling is the act of automatically visiting and discovering new and updated content on the internet.

WEB SCRAPING IN MACHINE LEARNING



WEB SCRAPING OUTLINE

Step 1

Identify the website from which you want to extract data.

Step 2

Inspect the website's structure and identify the data you want to extract.

Step 3

Write the code to send an HTTP request to the website's server and receive a response.

Step 4

Parse the response to extract the data you are interested in

Step 5

Store the data in a structured format, such as a spreadsheet or a database

Step 6

you can also write code to clean and organize the data, or to perform additional analysis on the data.

Some Popular Libraries for Performing Web Mining Tasks

PYTHON	scikit-learn	Gensim	NLTK
R	Caret	randomForest	xgboost
JAVA	Weka	Apache Mahout	

Popular Web Scraping Libraries

PYTHON	Beautiful Soup	Scrapy	PySpider
JAVASCRIPT	Cheerio	Puppeteer	
JAVA	JSoup	Apache HttpComponents	
PHP	Goutte	Simple HTML DOM	
C#	HtmlAgilityPack		

SCRAPY

Scrapy is a popular Python library for web scraping and crawling. It provides a simple, extensible, and easy-to-use interface for crawling websites and extracting structured data.

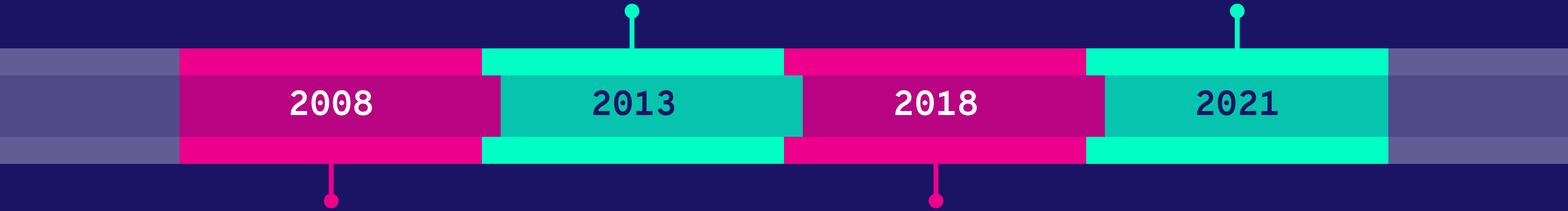
SCRAPY TIMELINE

New API and
A more modular design

Scrapy 1.0

support for Python 3.9, improved
performance and memory usage and
Support for Parallel requests

Scrapy 3.0



First Release

It was developed by Scrapinghub, a
company that provides web scraping
and data processing services.

Scrapy 2.0

support for Python 3, performance
and stability improvement and as
support for JavaScript rendering

Pros and Cons of Using Scrapy

Pros

- easy to use
- fast and scalable
- Extensible
- well-documented

Cons

- can have a steep learning curve
- is a more complex solution than some other web scraping libraries
- is not always the best choice

Using Scrapy Library

```
pip install scrapy
```

```
conda install -c conda-forge scrapy
```

You can then define your spider by subclassing the scrapy. Spider class and defining the following methods:

- name: A string that identifies the spider.
- start_urls: A list of URLs that the spider will start by visiting.
- parse(): A method that defines how the spider should parse the response and extract the data it is interested in.

But Before that, Once Scrapy is installed, you need to create a new Scrapy project using the scrapy startproject command. This will create a new directory with the basic file structure and settings for your Scrapy project.

```
scrapy startproject myproject
```

Creating Spider

This will create a new directory called myproject, which will contain the following files and directories:

myproject/: the main directory for your Scrapy project

myproject/scrapy.cfg: the configuration file for your Scrapy project

myproject/myproject/: the Python package for your Scrapy project

myproject/myproject/items.py: a file for defining the items that your Scrapy spider will extract

myproject/myproject/middlewares.py: a file for defining middlewares (custom components that can process requests and responses)

myproject/myproject/pipelines.py: a file for defining pipelines (components that process extracted items)

myproject/myproject/settings.py: the settings for your Scrapy project

myproject/myproject/spiders/: a directory for your Scrapy spiders

Once you have created a new Scrapy project, you can create a new spider by creating a new Python file in the myproject/myproject/spiders directory. This file should define a class that subclasses scrapy.Spider and has at least one method, parse(), which will be called when the spider starts crawling.

What is Spider in Scrapy?

In the context of web scraping, a spider is a program that visits web pages and follows links to other pages within the same website or to other websites. Spiders are also known as web crawlers or bots, and they are used by search engines to index websites and by online marketplaces to discover new products.

Spiders are an important part of the web scraping process, as they are responsible for visiting and discovering new and updated content on the internet. When a spider visits a website, it typically starts at a specific page and follows links to other pages within the same website. As it visits each page, it extracts the data it is interested in and stores it in a structured format, such as a database or a spreadsheet.

There are many algorithms and techniques that can be used to design and implement web spiders, and the choice of algorithm may depend on the specific requirements of the project. Some common techniques used in web scraping include breadth-first search, depth-first search, and incremental recrawling.

What is Parsing in Scrapy?

In the context of web scraping, parsing refers to the process of extracting data from a response returned by a website's server. A web scraper sends a request to a website's server, and the server responds with a response that includes the data the scraper is interested in. The scraper then needs to parse the response to extract the specific data it is looking for.

There are many ways to parse a response, and the specific method will depend on the format of the response and the data you are trying to extract. Some common formats for web scraping responses include HTML and XML, and there are many libraries and tools available for parsing these formats. For example, you can use a library such as Beautiful Soup to parse an HTML response and extract the data you are interested in, or you can use a library such as lxml to parse an XML response.

In summary, parsing is the process of extracting specific data from a website's response. It is an important step in the web scraping process, as it allows you to extract the data you are interested in from the raw response returned by the server

Web Scraping With Scrapy

```
import scrapy

class ArticleSpider(scrapy.Spider):
name = 'article'
start_urls = ['https://www.example.com/articles']

def parse(self, response):

    for article in response.css('article'):

     yield {

      'title':article.css('.article-title::text').extract_first(),
      'author':article.css('.article-author::text').extract_first(),
      'date':article.css('.article-date::text').extract_first(),
      'body':article.css('.article-body::text').extract_first(),

          }
```

Output Generated by Scrapy Spider

```
{'title': 'Article 1', 'author': 'John Smith', 'date': '2022-01-
01', 'body': 'Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.'},

{'title': 'Article 2', 'author': 'Jane Doe', 'date': '2022-01-02',
'body': 'Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.'},

{'title': 'Article 3', 'author': 'Bob Johnson', 'date': '2022-01-
03', 'body': 'Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur.'}

, ...
```

You can export your output in CSV, JSON, XML, PDF, XLS, XLSX formats.

Web Scraping With Beautiful Soup

```
import requests from bs4
import BeautifulSoup

# Send an HTTP request to the website's server
response = requests.get('https://www.example.com/%27)

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

# Find all the elements with the class 'article-title'
article_titles =soup.findall(class='article-title')

# Print the text of each article title for title in
article_titles:
 print(title.text)
```

Skills and Knowledge that can be helpful in Web Scraping

Familiarity with programming: Web scraping typically involves writing code to make HTTP requests, parse responses, and extract data. Familiarity with a programming language such as Python or Ruby is usually necessary to implement web scrapers.

Understanding of HTML and CSS: Web pages are typically structured using HTML and styled using CSS. Understanding these technologies and how they work is essential for correctly extracting data from web pages.

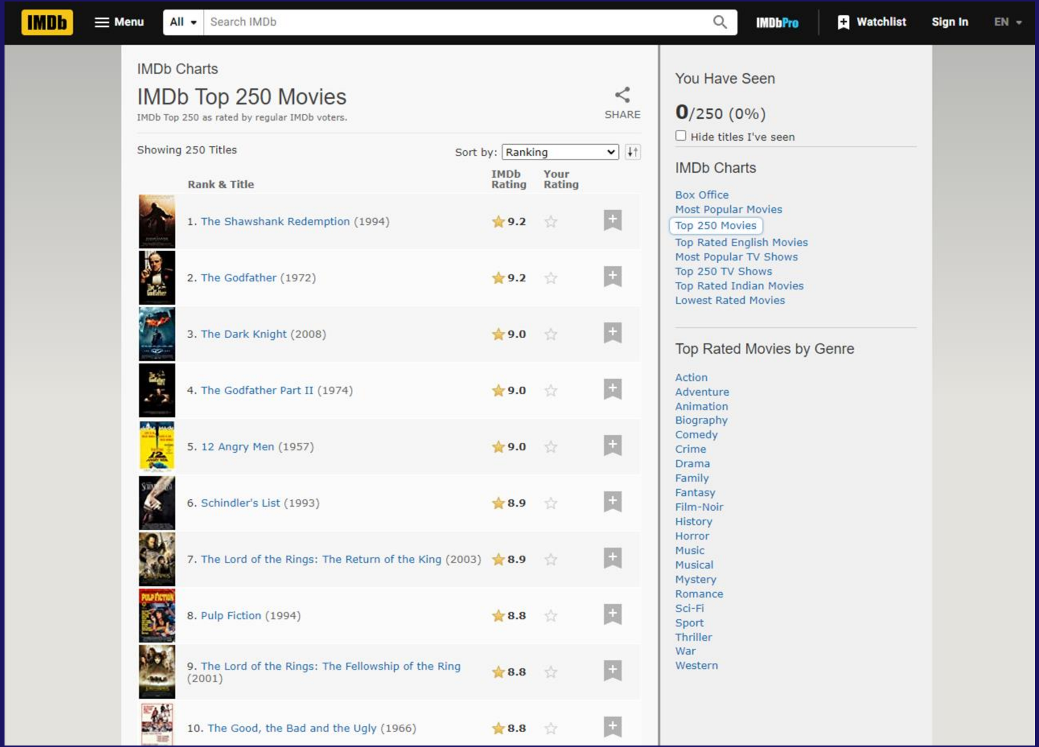
Familiarity with HTTP and web protocols: Web scraping involves making HTTP requests to a website's server and receiving responses. Knowing how HTTP works and how to craft HTTP requests can be helpful when debugging web scrapers and working with websites that require specific headers or cookies.

Experience with data manipulation and storage: Web scraping often involves collecting and processing large amounts of data. Knowledge of how to manipulate and store data using tools such as databases, CSV files, and JSON is useful when working with web scraping projects.

Familiarity with web scraping libraries and frameworks: There are many libraries and frameworks available that can make it easier to implement web scrapers. Familiarity with these tools can save time and effort when working on web scraping projects.

Understanding of ethical and legal issues: Web scraping can raise ethical and legal issues, such as privacy concerns and terms of service violations. It is important to be aware of these issues and to respect the terms of service of websites and the privacy of their users when using web scraping techniques.

IMDB



IMDb Charts

IMDb Top 250 Movies

IMDb Top 250 as rated by regular IMDb voters.

Showing 250 Titles

Sort by: Ranking

Rank & Title	IMDb Rating	Your Rating
1. The Shawshank Redemption (1994)	9.2	
2. The Godfather (1972)	9.2	
3. The Dark Knight (2008)	9.0	
4. The Godfather Part II (1974)	9.0	
5. 12 Angry Men (1957)	9.0	
6. Schindler's List (1993)	8.9	
7. The Lord of the Rings: The Return of the King (2003)	8.9	
8. Pulp Fiction (1994)	8.8	
9. The Lord of the Rings: The Fellowship of the Ring (2001)	8.8	
10. The Good, the Bad and the Ugly (1966)	8.8	

You Have Seen

0/250 (0%)

Hide titles I've seen

IMDb Charts

- Box Office
- Most Popular Movies
- Top 250 Movies
- Top Rated English Movies
- Most Popular TV Shows
- Top 250 TV Shows
- Top Rated Indian Movies
- Lowest Rated Movies

Top Rated Movies by Genre

- Action
- Adventure
- Animation
- Biography
- Comedy
- Crime
- Drama
- Family
- Fantasy
- Film-Noir
- History
- Horror
- Music
- Musical
- Mystery
- Romance
- Sci-Fi
- Sport
- Thriller
- War
- Western

Web Scrapy on IMDB

```
import scrapy

class IMDbSpider(scrapy.Spider):
    name = "imdb_spider"
    start_urls = [
        "https://www.imdb.com/chart/top/?ref_=nv_mv_250"
                    ]

    def parse(self, response):
        # Find the container with the list of movies
        movie_container = response.css("tbody.lister-list")

        # Iterate through each movie in the container
        for movie_div in movie_container.css("tr"):
            # Find the title of the movie
            title = movie_div.css("td.titleColumn a::text").get()
            print(title)
```

THANKS!

gathered by Ramin Kohandel

For Course of Machine Learning, DR. BORNA