An NLP-Driven analysis on research papers in the field of computer science

Master Thesis

Navid Shahbazi
# 390236

Submission Date: —

Supervisor: Dr. Tatiana Ermakova,
Prof. Dr. Benjamin Fabian

Technische Universität Berlin
School of Electrical Engineering and Computer Science
Department of Telecommunication systems
Chair of Telecommunication Networks Group

Statutory Declaration

I herewith formally declare that I have written the submitted dissertation independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper.

I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content.

I am aware that the violation of this regulation will lead to failure of the thesis.

Berlin, den 17th June 2021

Abstract

Natural language processing (NLP) is rapidly becoming as integral to the workplace as communication itself. From chatbots to language translation and voice-activated platforms, we have watched the world embark on an artificial intelligence (AI) arms race as players large and small successfully embrace NLP technology.

In this thesis, it was investigated how to make the process of finding the relevant research papers slightly more convenient and accurate by using NLP-driven methods. Multiple methods and algorithms were tested and compared to make it happen.

The first step to conduct a research paper is finding the relevant prior research papers. In order to search for a specific topic, there are already many criteria and filter selections on each of the common digital paper libraries, like ACM, IEEE, Arxiv, or Springer, that can be used. For example the published year is one of those filter selections by which you can filter the recent published papers or even olds, or another filter selection is the publishing journal name by which you can easily filter the papers that were published in the journal you consider. Despite having all those available filter selections, finding relevant papers to a research criteria is still difficult.

By using the final suggested approach in this thesis, researchers will be able to find more relevant prior research papers in the field they are researching about. The idea behind the suggested method is based on the theory of knowledge (ToK), which investigate on how a human can claim to know about something.

ToK looks to pursue more conceptual ideas of what it takes to acquire knowledge and how to apply that to real-world scenarios. ToK is heavily associated with epistemology. A branch of philosophy, epistemology studies the nature of knowledge, belief and truth.

According to the research paper "Theory of Knowledge for Literature Reviews: An Epistemological Model, Taxonomy and Empirical Analysis of IS Literature" published in December 2015, each research paper which is published in the real world can contribute in six different categories. Those six contributions are:

1. Synthesis
2. Adopting a new perspective
3. Theory building
4. Testing theories
5. Identification of research gaps
6. Providing a research agenda

This thesis aimed to provide a framework, that can help researchers to be able to classify the papers before they start investigating on prior done works. For example, if a researcher searches

for a research paper that has tested a theory, can easily use this framework instead of searching for it in the digital libraries, that is usually inconvenient and does not provide the accurate results.

At first, some crawlers have been programmed, that each of them crawls a different digital library. In each of the crawlers, a keyword should be used to crawl and in other words query the digital library for the specific keyword. By using these crawlers, the train data is collected. The next step is cleaning and preparing the data for the training process.

Contents

List of Figures

List of Tables

Symbols

List of Abreviations

ToK	Theory of knowledge
AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing

1 Introduction

2 Technical Basics

2.1 Machine Learning

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Because of new computing technologies, machine learning today is not like machine learning of the past. It was born from pattern recognition and the theory that computers can learn without being programmed to perform specific tasks; researchers interested in artificial intelligence wanted to see if computers could learn from data. The iterative aspect of machine learning is important because as models are exposed to new data, they are able to independently adapt. They learn from previous computations to produce reliable, repeatable decisions and results. It's a science that's not new – but one that has gained fresh momentum.

While many machine learning algorithms have been around for a long time, the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development. Here are a few widely publicized examples of machine learning applications you may be familiar with:

1. The heavily hyped, self-driving Google car? The essence of machine learning.
2. Online recommendation offers such as those from Amazon and Netflix? Machine learning applications for everyday life.
3. Knowing what customers are saying about you on Twitter? Machine learning combined with linguistic rule creation.
4. Fraud detection? One of the more obvious, important uses in our world today.

Why is machine learning important?

Resurging interest in machine learning is due to the same factors that have made data mining and Bayesian analysis more popular than ever. Things like growing volumes and varieties of available data, computational processing that is cheaper and more powerful, and affordable data storage.

All of these things mean it's possible to quickly and automatically produce models that can analyze bigger, more complex data and deliver faster, more accurate results – even on a very large scale. And by building precise models, an organization has a better chance of identifying profitable opportunities – or avoiding unknown risks.

2.2 Natural Language Processing (NLP)

Natural language processing deals with humongous amount of text to perform different range of tasks like clustering, classification, Machine Translation, and etc.. Humans can deal with text format quite intuitively but provided we have millions of documents being generated in a single day, we cannot have humans performing the above the three tasks. It is neither scalable nor effective. So, how do we make computers of today perform clustering, classification etc on a text data since we know that they are generally inefficient at handling and processing strings or texts for any fruitful outputs? A computer can match two strings and tell you whether they are same or not. But how do we make computers tell you about football or Ronaldo when you search for Messi? How do you make a computer understand that “Apple” in “Apple is a tasty fruit” is a fruit that can be eaten and not a company?

The answer to the above questions lie in creating a representation for words that capture their meanings, semantic relationships and the different types of contexts they are used in.

And all of these are implemented by using NLP techniques (such as Word Embeddings or numerical representations of texts) so that computers may handle them.

In nutshell, NLP contains multiple methods that helps us analyzing text inputs. For conducting NLP analysis on texts any programming language can be used but in this thesis Python has been used. There are couple of libraries in python which are tremendously useful: such as SpaCy, NLTK, and gensim.

2.3 Web Scraping

Nowadays, it is estimated that there are over 1.7 billions active websites daily on the internet. These websites are made for many purposes. Some have a specific content to share, some are just online shopping websites, and some are made for entertaining purposes and etc..

Behind all of these websites lie an html piece of code. This code contains many tags that each of them is written to perform a specific purpose. Some are design related but many of them contain some sort of informations. These informations can be retrieved from any website by web scraping techniques.

Web scraping is a programming technique, by which the user can trigger any tag within html source code of any website and store the information behind the tag in any data format, for instance tabular and JSON.

For web scraping, many languages can be used but in this thesis Python was used. There are multiple python libraries for web scraping in python such as scrapy, beautiful soup, and selenium.

In this thesis, it has been tried to use each of these libraries and finding out the best among them (see section ***).

3 Web Scraping

3.1 Introduction

Web scraping, also known as screen scraping, generally refers to the process of extracting, copying, saving and reusing third-party content on the internet. In addition to manual scraping, where content is copied by hand, a number of tools for the automated copying of websites have also become established. A positive use case of web scraping is the indexing of websites by Google or other search engines. In most cases, this indexing is welcome, since it is the only way for users to find the company pages they are looking for on the internet. Malicious screen scraping intended to unlawfully misappropriate intellectual property, on the other hand, violates copyright law and is therefore illegal.

3.2 How does web scraping work?

A variety of technologies and tools are employed in web scraping:

1. Manual scraping: It is a fact that both content and parts of the source code of websites are sometimes copied by hand. Criminals on the internet resort to this method particularly when bots and other scraping programs are blocked by the robots.txt file.
2. Software tools: Web scraping tools such as Scraper API, ScrapeSimple and Octoparse enable the creation of web scrapers even by those with little to no programming knowledge. Developers also use these tools as the basis for developing their own scraping solutions.
3. Text pattern matching: Automated matching and extraction of information from web pages can also be accomplished by using commands in programming languages such as Perl or Python.
4. HTTP manipulation: Content can be copied from static or dynamic websites using HTTP requests.
5. Data mining: Data mining can also be used for web scraping. To do this, web developers rely on an analysis of the templates and scripts in which website content is embedded. They identify the content they are looking for and use a “wrapper” to display it on their own site.
6. HTML parser: HTML parsers familiar from browsers are used in web scraping to extract and parse the content sought after.
7. Copying microformats: Microformats are a frequently used component of websites. They may contain metadata or semantic annotations. Extracting this data enables conclusions to be drawn about the location of special data snippets.

3.3 Use and areas of application

Web scraping is employed in many different areas. It is always used for data extraction, often for completely legitimate purposes, but misuse is also common practice.

1. Search engine web crawlers: Indexing websites is the basis for how search engines like Google and Bing work. Only by using web crawlers, which analyze and index URLs, is it possible to sort and present search results. Web crawlers are “bots”, which are automated programs that perform defined and repetitive tasks.
2. Replacement for web services: Screen scrapers can be used as a replacement for web services. This is of particular interest to companies that want to provide their customers with specific analytical data on a website. However, using a web service for this involves high costs. For this reason, screen scrapers, which extract the data, are the more cost-effective option.
3. Remixing: Remixing or mashup combines content from different web services. The result is a new service. Remixing is often done via interfaces, but if no such APIs are available, the screen scraping technique is also used here.

However, when talking about web scraping there are always some cases of misuses. The misuse of web scraping or web harvesting can have a variety of objectives:

1. Price grabbing: One special form of web scraping is called price grabbing. Here, retailers use bots to extract a competitor’s product prices in order to intentionally undercut them and acquire customers. Thanks to the great transparency of prices on the internet, customers quickly end up going to the next-cheapest retailer, which results in greater price pressure.
2. Content/product grabbing: Instead of prices or price structures, in content grabbing bots target the content of the website. Attackers copy intricately designed product pages in online shops true to the original and use the expensively created content for their own e-commerce portals. Online marketplaces, job exchanges and classified ads are also popular targets of content grabbing.
3. Increased loading times: Web scraping wastes valuable server capacities: Large numbers of bots constantly update product pages while searching for new pricing information. This results in slower loading times for human users—especially during peak periods. If it takes too long to load the desired web content, customers quickly turn to the competition.
4. Phishing: Cyber criminals use web scraping to grab email addresses published on the internet and use them for phishing. In addition, criminals can recreate the original site for phishing activities by making a deceptively realistic copy of the original site.

3.4 How can companies block web scraping?

There are a few measures that prevent a website from being subject to scraping:

1. Bot management: Using bot management solutions, companies are able to fine-tune which bots are allowed to access information on the website and which to treat as malware.
2. robots.txt: Using the robots.txt file, site operators can specify which areas of the domain

may be crawled and exclude specific bots from the outset.

3. Captcha prompts: Captcha prompts can also be integrated into websites to provide protection against bot requests.
4. Appropriate integration of telephone numbers and e-mail addresses: Site operators protect contact data from scraping by putting the information behind a contact form. The data can also be integrated via CSS.
5. Firewall: Strict firewall rules for web servers also protect against unwanted scraping attacks.

3.5 Legal framework: is screen scraping legal?

Many forms of web scraping are covered by law. This applies, for example, to online portals that compare the prices of different retailers. A ruling to this effect by the German Federal Court of Justice in 2014 clarifies this: As long as no technical protection to prevent screen scraping is overcome, there is no anti-competitive obstacle. However, web scraping becomes a problem when it infringes copyright law. Anyone who integrates copyrightable material into a website without citing the source is therefore acting unlawfully. In addition, if web scraping is misused, for phishing for example, the scraping itself may not be unlawful, but the activities that follow may be.

3.6 Web Scraping: Things you need to know

Web scraping is an integral part of the modern internet. Many popular services such as search engines or price comparison portals would not be possible without the automated retrieval of information from websites. Abuse of it, however, also poses serious risks for companies, such as unscrupulous competitors extracting expensively produced content from an online shop and copying it to their own. The load from traffic caused by bots acting autonomously is also not negligible. Currently, bots generate about half of website traffic. That is why effective bot management is a crucial factor in protecting a company website from scraping attacks.

3.7 How does web scraping work in practice in this thesis?

As mentioned before, you can do web scraping in almost every programming language, but the selected programming language in this thesis is python. There are dozens of libraries in python for web scraping but the top libraries are:

1. Scrapy
2. Requests
3. Selenium
4. Beautiful Soup

Each of these libraries has its own features and hence its own advantage and disadvantages. Table below shows the difference between these web scraping libraries, in case you are interested:

Table 3.1: Comparison of different web scraping libraries

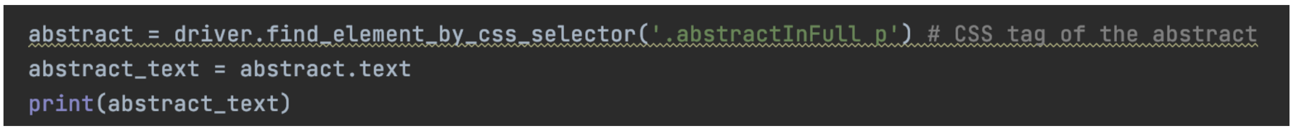
Specification	Scrapy	Requests	Beautiful Soup	Selenium
What is it?	Web scraping framework	Library	Library	Library
Purpose	Complete web scraping solution	Simplifies making HTTP requests	Data parser	Scriptable web browser to render javascript
Ideal use case	Development of recurring or large scale web scraping projects	Simple non-recurring web scraping tasks	Simple non-recurring web scraping tasks	Small-scale web scraping of javascript heavy websites
Built-in Data Storage Supports	JSON, JSON lines, XML, CSV	Need to develop your own	Need to develop your own	Customizable
Available selectors	JCSS, Xpath	N/A	CSS	CSS, Xpath
Asynchronous	Yes	No	No	No
Javascript support	Yes, via Splash library	N/A	No	Yes
Documentation	Excellent	Excellent	Excellent	Good
Learning curve	Easy	Very easy	Very easy	Easy
Ecosystem	Large ecosystem of developers contributing projects and support on Github and StackOverflow	Few related projects or plugins	Few related projects or plugins	Few related projects or plugins
Learning curve	32,690	34,727	-	14,262

The final library which was used in this thesis is Selenium. It is primarily for automating web applications for testing purposes, but is certainly not limited to just that. you can trigger any element on any website by its CSS or Xpath.

CSS stands for Cascading Style Sheets. it describes how HTML elements are to be displayed on screen, paper, or in other media. Furthermore, it saves a lot of work. It also can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files. CSS selectors are used to "find" (or select) the HTML elements you want to style.
We can divide CSS selectors into five categories:

1. Simple selectors (select elements based on name, id, class)
2. Combinator selectors (select elements based on a specific relationship between them)
3. Pseudo-class selectors (select elements based on a certain state)
4. Pseudo-elements selectors (select and style a part of an element)
5. Attribute selectors (select elements based on an attribute or attribute value)

Figure 3.1 shows an element that is triggered from the ACM digital library by its CSS selectors:



```
abstract = driver.find_element_by_css_selector('.abstractInFull p') # CSS tag of the abstract
abstract_text = abstract.text
print(abstract_text)
```

Figure 3.1: CSS selector

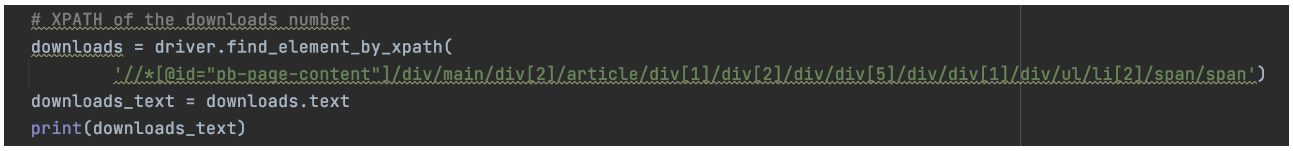
This CSS selector is related to the abstract text of each research paper when searching for a specific keyword on ACM digital library and the complete abstract can be extracted from each search result by using this piece of code.

Another approach for triggering elements on a website for web scraping is using Xpath. The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. In popular use (though not in the official specification), an XPath expression is often referred to simply as "an XPath".

Originally motivated by a desire to provide a common syntax and behavior model between XPointer and XSLT, subsets of the XPath query language are used in other W3C specifications such as XML Schema, XForms and the Internationalization Tag Set (ITS).

XPath has been adopted by a number of XML processing libraries and tools, many of which also offer CSS Selectors, another W3C standard, as a simpler alternative to XPath.

Figure 3.2 shows an element that is triggered from the ACM digital library by its Xpath:



```
# XPATH of the downloads number
downloads = driver.find_element_by_xpath(
        '//*[@id="pb-page-content"]/div/main/div[2]/article/div[1]/div[2]/div/div[5]/div/div[1]/div/ul/li[2]/span/span')
downloads_text = downloads.text
print(downloads_text)
```

Figure 3.2: Xpath

In this thesis, it was tried to retrieve all the possible elements out of the under observation digital libraries such as; ACM, Arxiv, and IEEE. The extracted data points might be useful as a separate feature for the training process in the machine learning algorithms that will be explained in its respective section later.

The extracted data points from ACM digital library are:

1. Title: Title of the research paper.
2. type: Indicating the type of the research paper. It can be research article, short paper, abstract, or etc.
3. downloads: Indicating total number of downloads on that specific research paper.
4. citations: Indicating total number of citations on that specific research paper.
5. date: It shows the publishing date of the research paper.
6. author1: The first and last name of the first author.
7. author2: The first and last name of the second author if applicable.
8. link: The URL link of the research paper.
9. abstract: The abstract of the research paper in text format.

The extracted data points from ARXIV digital library are:

1. Title: Title of the research paper.
2. type: Indicating the type of the research paper. It can be research article, short paper, abstract, or etc.
3. date: It shows the publishing date of the research paper.
4. link: The URL link of the research paper.
5. author1: The first and last name of the first author.
6. author2: The first and last name of the second author if applicable.
7. abstract: The abstract of the research paper in text format.