**CHAPTER 1**

**INTRODUCTION**

**Business is changing based on the emergences of the Technology including and every People need to buy something else that would be useful in their lives. It’s included clothing, foods, places to stay, accessories and so on. Traditional marketing is to go to the market and buy items that are need for us. But most people do not buy the product of the shop that they see first. People go around the entire market and check every shop that are selling the products that they need to buy. They are checking suitable price, quality of product, discount, specifications, brand and warranty before they buy.**

**Since the internet and technology is tremendously increase in every situation. So, most people need to buy laptops to use in their use cases such as studying purposes, work, entertainment, simulations, research, and development. They do not buy a laptop that they see first at the shop that they reach first time. Every people go around and check for suitable products for them. Nowadays, E-Commerce websites are available for buying products. They do not need to go physically located shops. They just need to open a Web Browser and buy it. But there are 50 billion of websites that sales laptops all over the world. People can’t find or retrieve the suitable product they need to buy.**

* 1. **Objectives**
  2. **Field Background**

The need to scrape websites came with the popularity of the Internet, where you share your content and a lot of data. The first widely known scrapers were invented by search engine developers (like Google or AltaVista). These scrapers go through (almost) the whole Internet, scan every web page, extract information from it, and build an index that you can search. Everyone can create a scraper. Few of us will try to implement such a big application, which could be new competition to Google or Bing. But we can narrow the scope to one or two web pages and extract information in a structured manner—and get the results exported to a database or structured file (JSON, CSV, XML, Excel sheets).

Nowadays, digital transformation is the new buzzword companies use and want to engage. One component of this transformation is providing data access points to everyone (or at least to other companies interested in that data) through APIs. With those APIs available, you do not need to invest time and other resources to create a website scraper. Even though providing APIs is something scraper developers won’t benefit from, the process is slow, and many companies don’t bother creating those access points because they have a website and it is enough to maintain.

There are a lot of use cases where you can leverage your knowledge of website scraping. Some might be common sense, while others are extreme cases. In this section you will find some use cases where you can leverage your knowledge.

The main reason to create a scraper is to extract information from a website. This information can be a list of products sold by a company, nutrition details of groceries, or NFL results from the last 15 years. Most of these projects are the groundwork for further data analysis: gathering all this data manually is a long and error-prone process.

Sometimes you encounter projects where you need to extract data from one website to load it into another—a migration. I recently had project where my customer moved his website to WordPress and the old blog engine’s export functionality wasn’t meant to import it into WordPress. I created a scraper that extracted all the posts (around 35,000) with their images, did some formatting on the contents to use WordPress short codes, and then imported all those posts into the new website.

A weird project could be to download the whole Internet! Theoretically it is not impossible: you start at a website, download it, extract and follow all the links on this page, and download the new sites too. If the websites you scrape all hav links to each other, you can browse (and download the whole Internet. I don’t suggest you start this project because you won’t have enough disk space to contain the entire Internet, but the idea is interesting. Let me know how far you reached if you implement a scraper like this.

* 1. **Overview of the System**
  2. **Organization of the Thesis**

**Thesis is formed of**

**CHAPTER 2**

**THEORY BACKGROUND**

**2.1 WEB CRAWLING**

**2.2 TERM FREQUENCY \* INVERSE DATA FREQUENCY**

introduction to Vector Space Model (VSM)

In information retrieval or text mining, the [term frequency – inverse document frequency](http://en.wikipedia.org/wiki/Tf%E2%80%93idf) (also called **tf-idf**), is a well know method to evaluate how important is a word in a document. tf-idf are is a very interesting way to convert the textual representation of information into a [Vector Space Model](http://en.wikipedia.org/wiki/Vector_space_model) (VSM), or into sparse features.

VSM has a very confusing past, that the history behind the ghost cited paper which in fact never existed; in sum, VSM is an algebraic model representing textual information as a vector, the components of this vector could represent the importance of a term (tf–idf) or even the absence of presence ([Bag of Words](http://en.wikipedia.org/wiki/Bag_of_words_model)) of it in a document; it is important to note that the classical VSM proposed by Salton incorporates local and global parameters/information (in a sense that it uses both the isolated term being analyzed as well the entire collection of documents). VSM is a space where text is represented as a vector of numbers instead of its original string textual representation; the VSM represents the features extracted from the document.

The first step in modeling the document into a vector space is to create a dictionary of terms present in documents. To do that, you can simple select all terms from the document and convert it to a dimension in the vector space, but we know that there are some kind of words (stop words) that are present in almost all documents, and extracting important features from documents, features do identify them among other similar documents, so using terms like “the, is, at, on”, etc.. isn’t going to help us, so in the information extraction, we’ll just ignore them.

Let’s take the documents below to define our document space:

Train Document Set:

d1: The sky is blue.

d2: The sun is bright.

Test Document Set:

d3: The sun in the sky is bright.

d4: We can see the shining sun, the bright sun.

Now, what we have to do is to create an index vocabulary (dictionary) of the words of the train document set, using the documents and from the document set, we’ll have the following index vocabulary denoted as where the t is the term:

1, if t is “blue”

E(t) = 2, if t is “sun”

3, if t is “bright”

4, if t is “sky”

Note that the terms like “is” and “the” were ignored as cited before. Now that we have an index vocabulary, we can convert the test document set into a vector space where each term of the vector is indexed as our index vocabulary, so the first term of the vector represents the “blue” term of our vocabulary, the second represents “sun” and so on. Now, we’re going to use the **term-frequency** to represent each term in our vector space; the term-frequency is nothing more than a measure of how many times the terms present in our vocabulary are present in the documents or, we define the term-frequency as a counting function:

where the is a simple function defined as:

So, what the returns are how many times is the term is present in the document. An example of this, could be since we have only two occurrences of the term “sun” in the document. Now you understood how the term-frequency works, we can go on into the creation of the document vector, which is represented by:

Each dimension of the document vector is represented by the term of the vocabulary, for example, the represents the frequency-term of the term 1 or (which is our “blue” term of the vocabulary) in the document .

Let’s now show a concrete example of how the documents and are represented as vectors:

which evaluates to:

As you can see, since the documents  and  are:

1. d3: The sun in the sky is bright.
2. d4: We can see the shining sun, the bright sun.

The resulting vector shows that we have, in order, 0 occurrences of the term “blue”, 1 occurrence of the term “sun”, and so on. In the, we have 0 occurrences of the term “blue”, 2 occurrences of the term “sun”, etc.

But wait, since we have a collection of documents, now represented by vectors, we can represent them as a matrix with shape, where is the cardinality of the document space, or how many documents we have and there is the number of features, in our case represented by the vocabulary size. An example of the matrix representation of the vectors described above is:

As you may have noted, these matrices representing the term frequencies tend to be very [sparse](http://en.wikipedia.org/wiki/Sparse_matrix) (with majority of terms zeroed), and that’s why you’ll see a common representation of these matrix as sparse matrices.

**2.3 COSINE SIMILARITY**

**2.4 WEB SCRAPING**

The automated gathering of data from the internet is nearly as old as the internet

itself. Although *web scraping* is not a new term, in years past the practice has been

more commonly known as *screen scraping*, *data mining*, *web harvesting*, or similar

variations. General consensus today seems to favor *web scraping*, so that is the term I

use throughout the book, although I also refer to programs that specifically traverse

multiple pages as *web crawlers* or refer to the web scraping programs themselves as

*bots*.

In theory, web scraping is the practice of gathering data through any means other

than a program interacting with an API (or, obviously, through a human using a web

browser). This is most commonly accomplished by writing an automated program

that queries a web server, requests data (usually in the form of HTML and other files

that compose web pages), and then parses that data to extract needed information.

In practice, web scraping encompasses a wide variety of programming techniques

and technologies, such as data analysis, natural language parsing, and information

security. Because the scope of the field is so broad, this book covers the fundamental

basics of web scraping and crawling in Part I and delves into advanced topics in

Part II. I suggest that all readers carefully study the first part and delve into the more

specific in the second part as needed.

If the only way you access the internet is through a browser, you’re missing out on a

huge range of possibilities. Although browsers are handy for executing JavaScript,

displaying images, and arranging objects in a more human-readable format (among

other things), web scrapers are excellent at gathering and processing large amounts of

data quickly. Rather than viewing one page at a time through the narrow window of a

monitor, you can view databases spanning thousands or even millions of pages at

once.

In addition, web scrapers can go places that traditional search engines cannot. A

Google search for “cheapest flights to Boston” will result in a slew of advertisements

and popular flight search sites. Google knows only what these websites say on their

content pages, not the exact results of various queries entered into a flight search

application. However, a well-developed web scraper can chart the cost of a flight to

Boston over time, across a variety of websites, and tell you the best time to buy your

ticket.

You might be asking: “Isn’t data gathering what APIs are for?” (If you’re unfamiliar

with APIs, see Chapter 12.) Well, APIs can be fantastic, if you find one that suits your

purposes. They are designed to provide a convenient stream of well-formatted data

from one computer program to another. You can find an API for many types of data

you might want to use, such as Twitter posts or Wikipedia pages. In general, it is preferable

to use an API (if one exists), rather than build a bot to get the same data. However,

an API might not exist or be useful for your purposes, for several reasons:

• You are gathering relatively small, finite sets of data across a large collection of

websites without a cohesive API.

• The data you want is fairly small or uncommon, and the creator did not think it

warranted an API.

• The source does not have the infrastructure or technical ability to create an API.

• The data is valuable and/or protected and not intended to be spread widely.

Even when an API *does* exist, the request volume and rate limits, the types of data, or

the format of data that it provides might be insufficient for your purposes.

This is where web scraping steps in. With few exceptions, if you can view data in your

browser, you can access it via a Python script. If you can access it in a script, you can

store it in a database. And if you can store it in a database, you can do virtually anything

with that data.

There are obviously many extremely practical applications of having access to nearly

unlimited data: market forecasting, machine-language translation, and even medical

diagnostics have benefited tremendously from the ability to retrieve and analyze data

from news sites, translated texts, and health forums, respectively.

Even in the art world, web scraping has opened up new frontiers for creation. The

2006 project “We Feel Fine” by Jonathan Harris and Sep Kamvar scraped a variety of

English-language blog sites for phrases starting with “I feel” or “I am feeling.” This led

to a popular data visualization, describing how the world was feeling day by day and

minute by minute.

Regardless of your field, web scraping almost always provides a way to guide business

practices more effectively, improve productivity, or even branch off into a brand-new

field entirely.

**2.5 NAÏVE BAYES CLASSIFICATION**

**2.6**

**TFIDF** stands for term frequency- inverse document frequency.

The TFIDF weight is used in text mining and IR. The weight is a measure used to evaluate how important a word is to a document in a collection of documents.

When using a simple technique like a frequency table of the terms in the document, we remove stop words, punctuation and stem the word to its root. And then, the importance of the word is measured in terms of its frequency; higher the frequency, more important the word.

In case of TFIDF, the only text pre-processing is removing punctuation and lower casing the words. We **do not** have to worry about the stop words.

TFIDF is the product of the TF and IDF scores of the term.

**TF** = number of times the term appears in the doc/total number of words in the doc

**IDF** = ln(number of docs/number docs the term appears in)

Higher the TFIDF score, the rarer the term is and vice-versa.

TFIDF is successfully used by search engines like Google, as a ranking factor for content.

The whole idea is to weigh down the frequent terms while scaling up the rare ones.

<https://medium.com/@shivangisareen/tfidf-7b29017dcdd>