NLP4Free 🔠⚡🤖🧠😀

A Free Natural Language Processing Microcourse

Part 2 - Data Acquisition and Preprocessing

Myles Harrison
http://www.mylesharrison.com/nlp4free

Contents

This work is licensed under a Creative Commons License.

Credit Where Due

Though this work is based upon my experiences consulting and teaching data and machine learning, including that for natural language, all materials within have been compiled or created by myself.

When I have included or relied upon others' materials such as images, code, or text, I have done my best to cite as appropriate and provide links to the source.

Data Acquisition

Structured and Unstructured Data

First we note some particulars about how text data differs from other kinds, for example, much of that present in relational database systems (RDBMS).

Though they do not have hard-and-fast definitions, ***structured data*** usually refers to anything with a set number of columns of particular data types - so anything you could put into Excel.

On the other hand, ***unstructured data*** refers to other data types: images, video, audio, specialty file formats, etc. and what we are interested in, free-form text.

The way to transform data from unstructured to structured varies; we'll see that turning text data into that with columns and rows is a requirement before performing machine learning but not for storage.

Structured Data

Unstructured Data

Data Acquisition

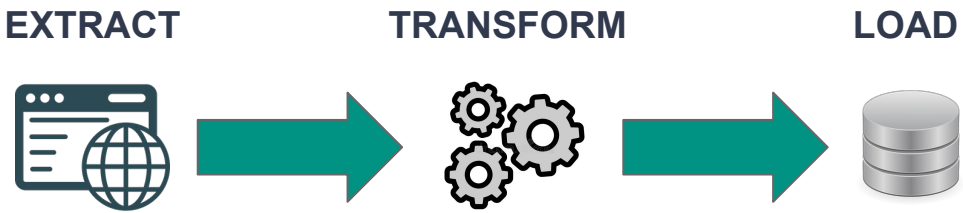
Data acquisition refers to, unsurprisingly, acquiring the data with which we wish to work.

In addition to the data itself, this usually also involves considering necessary other processes, tools, and platforms for working with the data - *e.g.* consolidating different data sources and how the result will be loaded and stored.

When working with relational databases (*i.e.* SQL) this is usually put in terms of the *Extract, Load,* and *Transform* processes:

- **Extract:** Getting data from source systems
- **Transform:** Consolidating, aggregating, filtering, cleaning
- **Load:** Put the data into a destination system (database)

This exists analogously for text data, where we might be extracting text from customer invoices or scraping from web pages, transforming or cleaning it in some way, and then loading into a type of database that can store text.



Storing Text Data

While traditional databases can store text, there are also specialized data platforms for doing so.

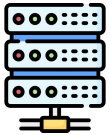
Examples of potential storage solutions are depicted on the right.

In this course, we will be working with smaller datasets stored in files which can be loaded directly into memory in Python.

Relational Databases
PostgreSQL, MySQL, Microsoft SQL.

Document Databases
ElasticSearch, MongoDB, Redis

Cloud Object Storage
AWS S3, Google Cloud Storage, Azure Blob



Internal Data
Traditional In-House

Organizations will have large amounts of data already stored, much of which may take the form of text.

This can include things like call transcripts, invoices, customer feedback, documentation, and more.

Websites and APIs
Online Sources

Supplementary data may be acquired from web pages or made available via web services (APIs).

This usually requires writing code or using software or a third-party service designed for this task.

User Data
Constantly Generated

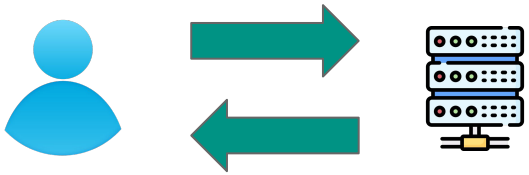
Most organizations' users will be generating a constant stream of data through interactions via phone calls, websites, and apps.

These may be stored internally, with another provider, or may even have to be acquired from another platform.

Web Services and APIs

A *web service* is an application running on a computer which can provide data or perform transactions when you interact with over the wire.

The machine hosting the service is referred to as a *server* and a machine interacting with it a *client*.



Technically, an *Application Programming Interface (API)* is the software enabling communication between two computers, allowing them to exchange information. There are many different kinds of APIs, with a modern and ubiquitous standard being the *Representational state transfer*, more commonly referred to as REST.

Though technically the server machine and software running on it is a web service, colloquially they are often also referred to in their entirety as an API. (*e.g.* "I got my data from the XYZ Bank REST API")

HTTP Requests

HTTP requests are what your web browser makes when you navigate to a given URL. Web pages are hosted by web servers and requests are made by client machines (for example, your laptop).

A server machine does not just have to serve web pages as part of a web service, it can also provide other services, such as returning data from a database or performing transactions.

There are a number of HTTP status codes that are returned as part of the response to a request. Some commonly encountered ones are detailed on the right.

"OK" - The request succeeded and the response was returned successfully.

"Not Found" - There is nothing at the path where the request was made (resource does not exist)

"Unavailable" - The server is unable to return a response to the request (usually because the server is down)

Client

Uses software to make requests over the wire (web browser, other program or with code)

Client initiates request

Server returns response

Server

Hosts web service via REST or other standard, performs actions and/or returns data

Python Libraries for Collecting Online Data

There are a number of helpful libraries in Python for collecting data in this way, which are typically used in conjunction.

We will focus on the two on the left for getting data from APIs and scraping web sites.

Requests - A python library that allows making HTTP requests to online services

Scrapy - A python library for advanced web scraping and building web crawlers (spiders)

Beautiful Soup - A library for parsing semi-structured data, such as that in HTML (web pages)

Selenium - Web automation framework for advanced web scraping tasks (JS, auth, etc.) with Python connector

The `requests` library

`requests` is a Python library that allows making HTTP requests programmatically.

One of the nice things about the requests library is that it is very simple code to write and very well documented.

https://pypi.org/project/requests/

Making your first request

In `requests`, getting data from a web service, whether a web server or REST API, is as simple as a few lines of code.

On the right, we make a request to get Google's homepage, and our response is the HTML code that our browser would render.

In the Jupyter notebook for this part of the course, we will go through an example in more detail, getting data from an API.

```
[1]: import requests

[10]: # Make the request
r = requests.get("https://www.google.com")

[3]: # Check out the response status code
r

[3]: <Response [200]>

*[5]: # Print the results as text (first 2000 characters)
r.text[0:2000]

[5]: '<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang
="en-CA"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Ty
pe"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.pn
g" itemprop="image"><title>Google</title><script nonce="x82VAMD51ra8XS_UUIuwv
g">(function(){var _g={kEI:\'DK2ZZIXcLuOv0PEPiuya8A4\',kEXPI:\'0,1359409,6059,
206,4804,2316,383,246,5,1129120,1197778,623,224,379865,16115,28684,22430,1362,
12314,17585,4998,17075,41316,2891,3926,214,4209,3405,606,50059,13245,13721,101
4,1,16916,2652,4,1528,2304,38933,3193,13659,4437,9358,13228,6651,7596,1,11943,
30211,2,16737,21269,1755,5679,1020,31122,4568,6256,23421,1252,5835,14968,4332,
7484,445,2,2,1,26632,8155,7381,2,1399,14569,873,19633,7,1922,9779,42459,20199,
927,19209,14,82,7651,12555,4070,4307,18988,2277,3098,3030,5630,480,9706,1804,7
```

Web scraping

Web scraping refers to the process of gathering data programmatically from web sites.

Since most web pages are intended to be rendered by a browser, performing web scraping usually requires a fair bit of investigative and trial-and-error kinds of work to extract the desired data, depending on the page design.

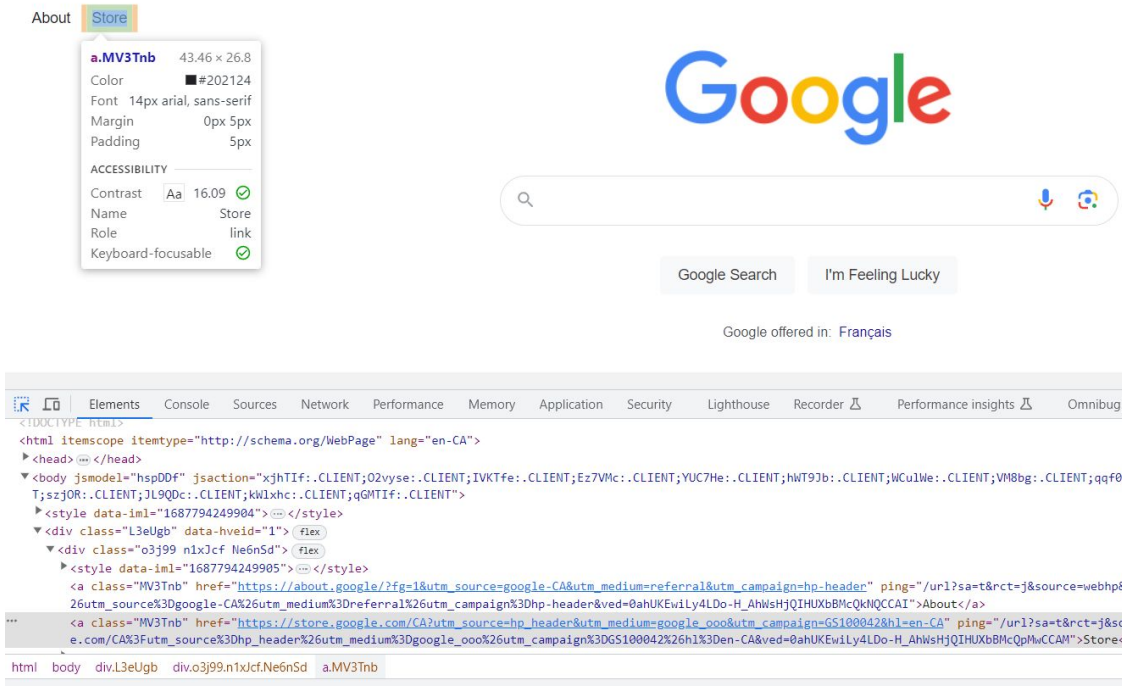
In Python, our two step process will be getting the page source using `requests`, and then extracting the desired data from the page structured using Beautiful Soup.

Extracting text data from websites

Getting the exact data from a website usually requires some investigation into the page's structure using your browser's web developer tools (Chrome, Firefox, Edge)

While a background in web development is not required, some familiarity with HTML is helpful.

We will see a detailed example in the Jupyter notebook.



Preprocessing Text

Talking About Text

Natural language processing has fairly specific terminology of its own.

In NLP, the term *document* refers to a single unit of collected text: this could be something as large as a book, or something as short as a single sentence movie review or tweet.

The term corpus (from the Latin for body) refers to a collection of many documents comprising an entire dataset.

The same preprocessing steps must be applied to all documents in a given corpus to ensure consistency.

Document

Corpus

What is Preprocessing?

Preprocessing refers to cleaning and transforming the original text data to structured data to make it suitable for machine learning (or other uses).

While some preprocessing steps may need to be considered carefully as they may be specific to your use case, there are general approaches that work well and are standard.

BIAS ALERT: The vast majority of the NLP is based on working with the English language.

Though there are approaches for other languages, and some preprocessing techniques work equally well for both English and other languages based upon the Latin alphabet, working with non-English text is typically categorized under the domain of multilingual NLP or machine translation applications.

Preprocessing Text in Python

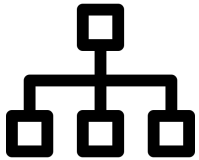
Base python has built-in string manipulation capabilities and is able to work with regular expressions.

NLTK

The **Natural Language Toolkit (NLTK)** is a fully-featured NLP library and de facto standard for such in python, essential for many tasks including preprocessing.

Scikit-learn (sklearn) is the standard open source machine learning library in Python and includes modules for preprocessing text in addition to machine learning.

Preprocessing Steps in NLP

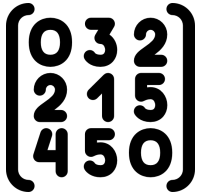


Tokenization

Break free-form text documents down into tokens: constituent units of language (usually words)

Normalization

Apply techniques to reduce the noise and variance in the language data and standardize



Vectorization

Convert text data to numeric features: structured data suitable for machine learning or analytics

Tokenization

There are many different ways to tokenize. In its simplest application, tokenizing could be breaking a string of text into words separated by whitespace.

```
The quick brown fox jumped over the lazy dog
```

Different applications may require more sophisticated methods of tokenization or use-case specific ones, for example, OpenAI has their own customer tokenizer and there is a class for tokenizing tweets in nltk.

Hugging Face has a series of videos breaking down different tokenization approaches here.

Normalization

Normalization is the standardization of the text data to remove unwanted variation and noise, and address any possible issues the data.

This usually includes steps such as:

- Removing extra whitespace
- Removing punctuation
- Expanding contractions
- Standardizing case
- Addressing spelling errors

There are also separate important steps which may be performed as part of normalization: *removing stop words*, *stemming,* and *lemmatization*.

Stop words

Stop words are words that are removed as they are not important in the analysis. In standard practice, these are usually just the syntactic "glue" which holds together language, and in English includes terms like *the*, *and*, *or*, *is*, and so on.

These are removed as they don't contain any particular meaning on their own, and so looking at language as data, we are generally only interested in words like nouns, adjectives, and verbs.

NLTK includes lists of stop words in different languages, as seen here.

```
import nltk

stopwords = nltk.corpus.stopwords.words('english')

print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
'you', "you're", "you've", "you'll", "you'd", 'your', 'your
s', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their', 'theirs', 'themselve
s', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
l", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be',
'been', 'being', 'have', 'has', 'had', 'having', 'do', 'doe
s', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'o
r', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'fo
r', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from',
'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'aga
in', 'further', 'then', 'once', 'here', 'there', 'when', 'whe
re', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'onl
y', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'ca
n', 'will', 'just', 'don', "don't", 'should', "should've", 'n
ow', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "ar
en't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "does
n't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't",
'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "must
n't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't",
'wouldn', "wouldn't"]
```

Stemming

Stemming is a rules-based method for normalizing text by collapsing tokens which have the same meaning but different forms by truncation, for example, conjugated verbs or plural forms of nouns.

Stemming is not perfect - if one of our rules was to remove *-ing* suffixes, this would successfully convert *falling* to *fall*, but *running* would become *runn*.

falling running

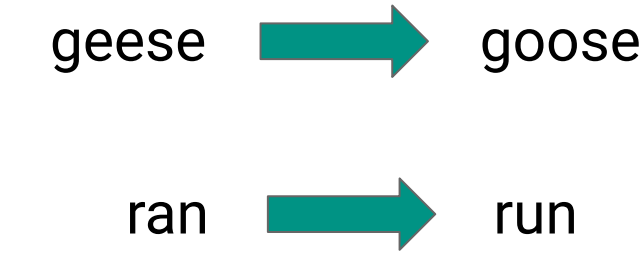
Different methods of stemming exist based upon rules and heuristics which have been historically developed.

In NLTK, the Porter Stemmer and Snowball stemmer are commonly used.

Lemmatization

Lemmatization seeks to resolve tokens which are part of the same concept without the shortcomings of stemming.

It uses a more complex rules-based approach, which requires keeping track of a vocabulary of all the related forms of a word and their base or root token.



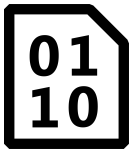
Vectorization

Vectorizing text is the important final step where we turn the unstructured text data into structured data with rows, columns, and numeric values, that can be used for machine learning.

There are different approaches to vectorization, but traditional approaches create the very important *document-term matrix*, so called because the observations of collected text (documents) lie along the rows and the tokens (terms) become our features.

Vectorization Methods

So what goes into the matrix? How do we quantify the text? There are a number of different methods.



Binary (One-Hot) Encoding

A binary flag (0/1) for if the term appears in each document. This is generally less common in use.

TF-IDF

Term Frequency-Inverse Document Frequency

What proportion of all tokens each token is for each document, divided by the log of the number of documents in which it appears in the corpus.

Count Vectorization

A raw count of the number of times each token appears per document. That's it!

Embeddings

Advanced methods that learn statistical representations of words based on a given corpus.

There is no single "right" answer - though embeddings are state of the art at the cost of additional complexity.

Acquiring and Preprocessing Text in Python

Let's see what we covered in action! Download the Jupyter Notebook to run locally, or open in Colab.

End of Part 2

Data Acquisition and Preprocessing