Obtaining, Cleaning, and Exploring Corpora

Elliott Ash

September 6, 2018

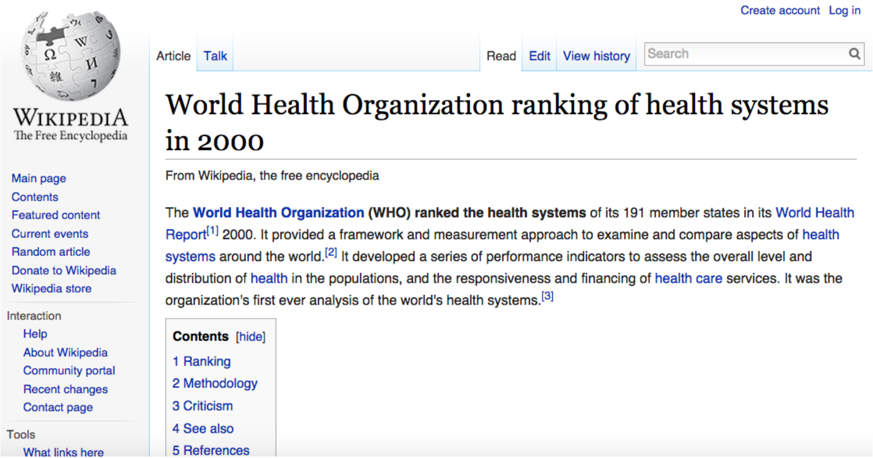
Publicly Available Corpora

- There is already a vast amount of data out there that has already been compiled (e.g. CourtListener, Twitter, New York Times, Reuters, Google, Wikipedia).
- Chris Bail curates a list of these datasets:
  - `https://docs.google.com/spreadsheets/d/1I7cvuCBQxosQK2evTcdL3qtglaEPc0WFEs6rZMx-xiE/edit`
- Some interesting corpora described in NLTK Book Chapter 2.
- Many proprietary corpora are becoming available for research:
  - Lexis
  - Web of Science

Screen Scraping

- A screen scraper is a computer program that:
  - loads/reads in a web page
  - finds some information on it
  - grabs the information
  - stores it in a dataset
- Once upon a time you could collect virtually any piece of information from the internet by screen scraping.
  - But now web sites make it difficult with restrictive terms of use, bot-blockers, javascript, etc.
  - Still, a little creativity goes a long way.

What a web site looks like to us



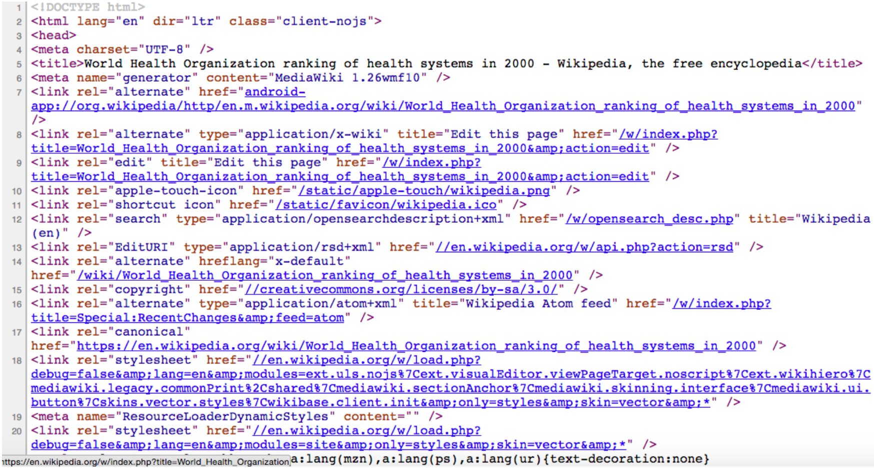
World Health Organization ranking of health systems in 2000

From Wikipedia, the free encyclopedia

The **World Health Organization (WHO) ranked the health systems** of its 191 member states in its World Health Report[1] 2000. It provided a framework and measurement approach to examine and compare aspects of health systems around the world.[2] It developed a series of performance indicators to assess the overall level and distribution of health in the populations, and the responsiveness and financing of health care services. It was the organization's first ever analysis of the world's health systems.[3]

Contents [hide]

What a web site looks like to a computer



```
<!DOCTYPE html>
<html lang="en" dir="ltr" class="client-nojs">
<head>
<meta charset="UTF-8" />
<title>World Health Organization ranking of health systems in 2000 - Wikipedia, the free encyclopedia</title>
<meta name="generator" content="MediaWiki 1.26wmf10" />
<link rel="alternate" href="android-
app://org.wikipedia/http/en.m.wikipedia.org/wiki/World_Health_Organization_ranking_of_health_systems_in_2000"
/>
<link rel="alternate" type="application/x-wiki" title="Edit this page" href="/w/index.php?
title=World_Health_Organization_ranking_of_health_systems_in_2000&amp;action=edit" />
<link rel="edit" title="Edit this page" href="/w/index.php?
title=World_Health_Organization_ranking_of_health_systems_in_2000&amp;action=edit" />
<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png" />
<link rel="shortcut icon" href="/static/favicon/wikipedia.ico" />
<link rel="search" type="application/opensearchdescription+xml" href="/w/opensearch_desc.php" title="Wikipedia
(en)" />
<link rel="EditURI" type="application/rsd+xml" href="//en.wikipedia.org/w/api.php?action=rsd" />
<link rel="alternate" hreflang="x-default"
href="/wiki/World_Health_Organization_ranking_of_health_systems_in_2000" />
<link rel="copyright" href="//creativecommons.org/licenses/by-sa/3.0/" />
<link rel="alternate" type="application/atom+xml" title="Wikipedia Atom feed" href="/w/index.php?
title=Special:RecentChanges&amp;feed=atom" />
<link rel="canonical"
href="https://en.wikipedia.org/wiki/World_Health_Organization_ranking_of_health_systems_in_2000" />
<link rel="stylesheet" href="//en.wikipedia.org/w/load.php?
debug=false&amp;lang=en&amp;modules=ext.uls.nojs%7Cext.visualEditor.viewPageTarget.noscript%7Cext.wikihiero%7C
mediawiki.legacy.commonPrint%2Cshared%7Cmediawiki.sectionAnchor%7Cmediawiki.skinning.interface%7Cmediawiki.ui.
button%7Cskins.vector.styles%7Cwikibase.client.init&amp;only=styles&amp;skin=vector&amp;*" />
<meta name="ResourceLoaderDynamicStyles" content="" />
<link rel="stylesheet" href="//en.wikipedia.org/w/load.php?
debug=false&amp;lang=en&amp;modules=site&amp;only=styles&amp;skin=vector&amp;*" />
a:lang(mzn),a:lang(ps),a:lang(ur){text-decoration:none}
```

ttps://en.wikipedia.org/w/index.php?title=World_Health_Organization

Screen Scraping in Python

```
# package to access web pages
from urllib.request import urlopen
url = 'https://goo.gl/VRF8Xs' # shortened URL
page = urlopen(url) # open page

html = page.read() # read page as string
print(html[:400])  # print first 400 characters
print(html[-400:]) # print last 400 characters
print(len(html))   # print length of string
```

Browser Automation

- Many web sites are designed to be difficult to scrape.
- Python has many solutions for simulating a human browser:
  - robobrowser
  - selenium (chromedriver, phantomjs)
- Other solutions if all else fails:
  - DownThemAll! plug-in for Firefox
  - Hire mechanical turkers to manually download data.

API's

- API = Application Programming Interface
  - These are developer-oriented tools that provide access to cleaner data.
- Chris Bail's list of API's that could be interesting for research:
  - `https://docs.google.com/spreadsheets/d/1ZEr3okdlb0zctmX0MZKo-gZKPsq5WGn1nJ0xPV7al-Q/edit`
- The example data set was obtained from the CourtListener API (`courtlistener.com/api`).

Other Languages

- All of the tools that we discuss in this class are available in many languages.
- spaCy has full functionality in English, German, Spanish, Portuguese, French, Italian, and Dutch.
  - beta functionality in dozens of other languages including Chinese and Arabic
  - See `https://spacy.io/usage/models`.

The machine learning models are language-independent.

Translation

- Can also translate languages before running an analysis

```
from googletrans import Translator
translator = Translator()
lang = translator.detect(korean_string).lang
eng = translator.translate(korean_string,
                           src=lang,
                           dest='en')
eng.text
```

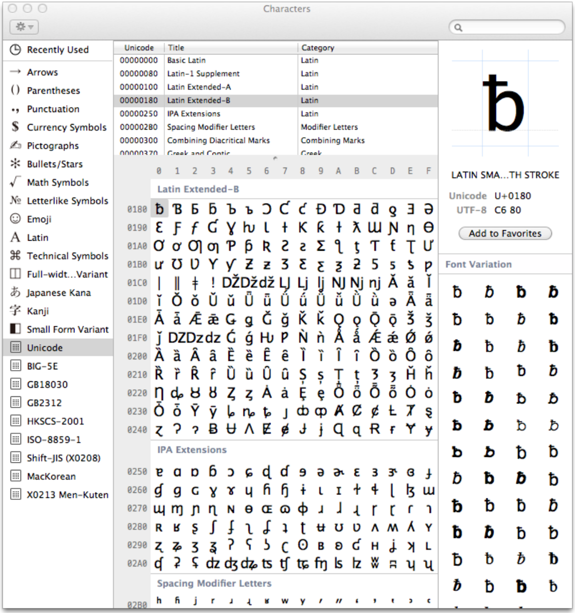
HTML Parsing

```
from bs4 import BeautifulSoup # HTML parser
soup = BeautifulSoup(html, 'lxml') # parse page
print(soup.title)

# extract text
text = soup.get_text() # remove HTML markup
lines = text.splitlines() # split by line
print(len(lines)) # print number of lines

# drop empty lines
lines = [line for line in lines if line != '']
print(len(lines))
print(lines[:20]) # print first 20 lines
```

Character Encoding



Unicode	Title	Category
00000000	Basic Latin	Latin
00000080	Latin-1 Supplement	Latin
00000100	Latin Extended-A	Latin
00000180	Latin Extended-B	Latin
00000250	IPA Extensions	Latin
00000280	Spacing Modifier Letters	Modifier Letters
00000300	Combining Diacritical Marks	Combining Marks
00000370	Greek and Coptic	Greek

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Latin Extended-B																
0180	ƀ	Ɓ	Ƃ	ƃ	Ƅ	ƅ	Ɔ	Ƈ	ƈ	Ɖ	Ɗ	Ƌ	ƌ	ƍ	Ǝ	Ə
0190	Ɛ	Ƒ	ƒ	Ɠ	Ɣ	ƕ	Ɩ	Ɨ	Ƙ	ƙ	ƚ	ƛ	Ɯ	Ɲ	ƞ	Ɵ
01A0	Ơ	ơ	Ƣ	ƣ	Ƥ	ƥ	Ʀ	Ƨ	ƨ	Ʃ	ƪ	ƫ	Ƭ	ƭ	Ʈ	Ư
01B0	ư	Ʊ	Ʋ	Ƴ	ƴ	Ƶ	ƶ	Ʒ	Ƹ	ƹ	ƺ	ƻ	Ƽ	ƽ	ƾ	ƿ
01C0	ǀ	ǁ	ǂ	ǃ	Ǆ	ǅ	ǆ	Ǉ	ǈ	ǉ	Ǌ	ǋ	ǌ	Ǎ	ǎ	Ǐ
01D0	ǐ	Ǒ	ǒ	Ǔ	ǔ	Ǖ	ǖ	Ǘ	ǘ	Ǚ	ǚ	Ǜ	ǜ	ǝ	Ǟ	ǟ
01E0	Ǡ	ǡ	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ
01F0	ǰ	Ǳ	ǲ	ǳ	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ
0200	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
0210	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
0220	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
0230	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ƚ	Ⱦ	ȿ
0240	ɀ	Ɂ	ɂ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ
IPA Extensions																
0250	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ
0260	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ
0270	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ
0280	ʀ	ʁ	ʂ	ʃ	ʄ	ʅ	ʆ	ʇ	ʈ	ʉ	ʊ	ʋ	ʌ	ʍ	ʎ	ʏ
0290	ʐ	ʑ	ʒ	ʓ	ʔ	ʕ	ʖ	ʗ	ʘ	ʙ	ʚ	ʛ	ʜ	ʝ	ʞ	ʟ
02A0	ʠ	ʡ	ʢ	ʣ	ʤ	ʥ	ʦ	ʧ	ʨ	ʩ	ʪ	ʫ	ʬ	ʭ	ʮ	ʯ
Spacing Modifier Letters																
02B0	ʰ	ʱ	ʲ	ʳ	ʴ	ʵ	ʶ	ʷ	ʸ	ʹ	ʺ	ʻ	ʼ	ʽ	ʾ	ʿ

ƀ

LATIN SMA...TH STROKE

Unicode U+0180

UTF-8 C6 80

Removing unicode characters

```
# package for removing unicode
from unidecode import unidecode
fixed = unidecode('Visualizations\xa0')
print(fixed) # print cleaned string
```

Corpus cleaning

- What we've already done:
  - removed HTML markup, extra white space, and unicode
- But HTML markup is often valuable:
  - HTML markup for section header names.
  - Legal database web sites often have HTML tags for citations to other cases.
- Other cleaning steps:
  - page numbers
  - hyphenations at line breaks
  - table of contents, indexes, etc.
- These are all corpus-specific, so inspect ahead of time.

Regular Expressions

- Regular Expressions, implemented in the Python package **re**, provide a powerful string matching tool.
  - A systematic string matching protocal – can match arbitrary string patterns
  - e.g., use utilit* to match utility, utilities, utilitarian, ...
  - Important for identifying speaker names (in political documents) section headers (in statutes), citations (in judicial opinions), etc.
- Also quite tedious, so we will not cover it here.
  - See NLTK book Chapter 3.4-3.5 for an introduction.

OCR (Optical Character Recognition)

- Your data might be in PDF's or images. Needs to be converted to text
- The best solution (that I know of) is ABBYY FineReader, which is expensive but might be available at your university library.
- My colleague Joe Sutherland at Columbia has a nice open-source package for OCR:
  - `https://github.com/jlsutherland/doc2text`

Should you run a spell checker?

- The short answer is no:
  - Most corpora have important specialized vocabulary that would be flagged by standard spell-checkers.
  - They are also very slow to run on large corpora.
  - In most empirical contexts, it's safe to assume that spelling errors (especially OCR errors) are uncorrelated with treatment assignment.
- Better solutions:
  - drop short (one or two letters) and long words (over 12 letters).
  - get doc frequencies for each word and filter out rare words
    - or use word embeddings (Lecture 13) and trust that misspellings will be nearby the true word.
- But:
  - There are cases where spelling errors could be correlated with treatment (for example, increasing legislator salaries might change both policy priorities and spelling error rates)
  - Check out the **enchant** module in Python.

Collect Key Metrics

```
df1 = df1[['state','snippet']]
# Number of documents
len(df1['snippet'])
# Number of label categories (e.g. states)
df1['state'].describe()
# Number of samples per class
counts_per_class = df1.groupby('state').count()
counts_per_class.head()
# Words per sample
def get_words_per_sample(txt):
    return len(txt.split())
df1['num_words'] = df1['snippet'].apply(get_words_per_sample)
df1['num_words'].describe()
# Frequency distribution over words
from collections import Counter
freqs = Counter()
for i, row in df1.iterrows():
    freqs.update(row['snippet'].lower().split())
freqs.most_common()[:20]
# (Number of samples) / number of words per sample)
len(df1['snippet']) / df1['num_words'].mean()
```