# Interlude: Museums and Libraries

In this interlude, we are going to use the wonderful, beautiful open data from the Cleveland Museum of Art to generate new, fake art pieces. The idea here is to think about the power and limitations of open data, of historical bias, and of the problems of language in the public space.

This chapter comes with a BIG warning: A lot of art is racist, developed in patently racist societies using racist language. The same trigger applies to sexism, colonialism, homophobia, antisemitism, ableism, and other forms of oppression. We feel confident in saying that this is not the fault of the Cleveland Museum of Art. To remove all art created in racist societies is to remove all art. That said, it is shocking to see how quickly the simple statistical trick shown here will reproduce, say, sexist modes of language, simply because they are so overrepresented through the historical artifacts and their descriptions.

As with all interludes, we start by bringing in all the LIBRARY modules that we will need. In this case, we also need to download a separate toolkit of language information about English as a spoken/written ("natural") language; the use of the term *natural* for spoken/written English language could be another whole book, perhaps.

```
import json # to parse the data format
import urllib.request # to get data from the web
import pickle # the storage format for python
import nltk # "natural language toolkit"
import re # regular expression library lets us search text
from random import choice, random
from os.path import exists # this asks "does this file exist?"
from bs4 import BeautifulSoup # this can turn web text into human-readable
text
from collections import Counter # helpful to create tallies
import string

# the first time we use the natural language toolkit (nltk)
# we need to download the data
nltk.download('punkt')
>>> True
```

Now we need to download the information from the Cleveland Museum of Art's database. That said, we do not want to download all of it every time we want to use it, so we save it to disk in a format that Python (the computer language) prefers; this is called PICKLE (and the file ending on a pickle file is, at least for me, .PKL).

```
# we would like to only ask for data from the lovely museum once,
# so we need to store it locally
data_filename = "./cleveland_data.pkl"

# if we have not already made a data file, we need to make one
if not exists(data_filename):

    # the wonderful people of the cleveland art museum
    # make a lot of data open and free.
    # read more here: https://openaccess-api.clevelandart.org/
    CLEVELAND_URL = "https://github.com/ClevelandMuseumArt/openaccess
    /raw/master/data.json"
```

```
# we need to get the data from them, and this is how they suggest to do it:
with urllib.request.urlopen(CLEVELAND_URL) as cleveland_json:

    # json is the simple and human-readable data format they use
    cleveland_raw_data = json.loads(cleveland_json.read().decode("utf-8"))

        # pkl.dump puts all that data in a file we can use locally
        pickle.dump(cleveland_raw_data,open(data_filename,"wb"))

# just to verify we're using local data, we will only use data loaded from disk
art_data = pickle.load(open(data_filename,"rb"))

# this tells us how many artworks were in that query above
print(f"DATA LOADED: {len(art_data)} artworks.")
>>> DATA LOADED: 64137 artworks.
```

Now that we have loaded our artworks, we need to get the descriptions of the art. We are not going to look at the visual components of the art—just their descriptions.

```
# we are just using the descriptions of all the artworks
# no titles, no artists, nothing.
all_descriptions = [elt['wall_description'] for elt in art_data if 'wall_description' in elt
and None!= elt['wall_description']] + \
            [elt['digital_description'] for elt in art_data if 'digital_description' in elt
            and None!= elt['digital_description']]

# beautiful soup takes the "HTML-like" descriptions and makes them raw text
soup = BeautifulSoup(" ".join(all_descriptions))
descriptions = "".join([c for c in soup.text])

# nltk does the hard job of turning all the words into a nice sequential list
# you'd think that simply separating by space does the job
# but that, again, is a bias toward simple sentences.
# for instance, it's easy to see how: "You have a cup." becomes
# ["You","have","a","cup","."] but sentences
# like "Bob's cup is at St. Germain" are harder to parse.
art_words = nltk.word_tokenize(descriptions)
```

Now that we have the descriptions, we need to find a simple way to statistically encode those descriptions. We are using a simple, but effective, idea called the *bigram*; this is how autocorrect historically makes its guesses. What's the most likely word to follow the current warp? Sorry, I meant *word*, not *warp*.

```
# strip out punctuation that makes parsing and generating harder
# note ALL the bias implicit here
# in many English AND non-English texts and names, these punctuation are crucial
# but we are just throwing them away.
letters = set(descriptions)
# print(letters)
unwanted_punctuation = set([c for c in letters if not (re.match("\w",c,flags=re.A) or
c in ".'")])
# print(unwanted_punctuation)
def is_punctuated(word):
    for letter in word:
        if letter in unwanted_punctuation:
            return True
    return False
art_words = [word for word in art_words if not is_punctuated(word)]

# The key to our algorithm is something called a bigram
# Our bigrams are every set of two words in a row.
# "A cat sits." -> ("a","cat"),("cat","sits")
# 'zip' is a command in Python that takes two lists
# and make a new list that groups each of the same-located items,
# e.g., zip([1,2],[3,4]) -> [(1,3),(2,4)]
# so we passed in two equally sized lists of all words in order,
# with the second list offset from the first list by one word.
art_bigrams = zip(art_words[:-1],art_words[1:])
```

Now we can figure out the specific likelihood that any particular word is following another word. We create a big "dictionary" which is a bit like a spreadsheet where our code can "look up" these likelihoods.

```
# now we want to make a big lookup "dictionary" of
# all the bigrams such that if we look up the first
# word of the two, we get a list of every word that might
# follow it.
art_chances = {k.lower(): [] for k in art_words}
for b in art_bigrams:
    art_chances[b[0].lower()].append(b[1])
```

At points, we should check our work. What words follow, say, "ancient"? There's a handy tool in Python (COUNTER) to count lists and show us the most common ones.

```
Counter(art_chances["ancient"]).most_common(5)
>>> [('Greek', 50), ('Roman', 41), ('tombs', 41), ('Egyptian', 22), ('India', 21)]
```

The output (everything after >>>) means that: the word GREEK follows ANCIENT the most often (seven times in our data); ROME follows it six times; and so on. This makes sense, so we can move on to generate some new art descriptions!

The idea in the following code is that we want to generate sentences based on those likelihoods.

```
# now we just need to generate some new descriptions
# as you might guess from our very simple algorithm,
# the results will be weird. there's a chance they'll be
# racist, homophobic, and/or any number of other terrible
# things. there's some question of whether that means we
# should allow them to be randomly generated and potentially
# upsetting/damaging or whether specifically excising
# problematic language is erasing the real, colonialist history of art.

# let's make no more than 10 sentences, this is easily changed
num_sentences = 10
for _ in range(num_sentences):

    # we need to start our sentence with a random word!
    prev_word = choice([word for word in art_words if re.match("\w",word) and
    len(word) > 3])
    word_number = 0
    sentence = ""
    # end sentences at a '.'
    while "."!= prev_word:

        # this is the crux
        # the next word will be randomly chosen from the
        # list of words/punctuation ("tokens") that
        # follow the current word in the text
        # if "of" follows "book" 80% of the time,
        # then that list (art_chances["book"]) includes 80% "of"s
        # and that should be the word we choose 80% of the time.
        new_word = choice(art_chances[prev_word.lower()])

        # capitalize the first word of a sentence
        if (0 == word_number):
            prev_word = prev_word.title()
        word_number += 1

        # don't put a space between a word and its possessive
        # but do put space between most other tokens.
        sentence += f"{prev_word}"
```

```
        if not (re.match("\W",new_word) or "s" == new_word): #punctuation
            sentence += " "

        # move to the next word
        prev_word = new_word

    # a lot of the generated sentences are junk!
    # throw away very short ones.
    if word_number > 4: # throw away junk sentences
        print(f"{sentence}.")
>>>
```

Planning and beauty, including rites, surrounded by the sculpture as a decidedly personal visual terms" da Firenze on a place, halfway between 1389 and Erlangs focus for Cowan's magnificent decoration of contemplative, but deviated from Heike Monogatari, while her husband is not the empty and the 1880s.
Textile with the inclusion of hours perhaps Cardinal Francisco Jimenez was an established and his head, marches, Genesis, Mexico's frontispiece.
Multicolored acanthus leaves, and enrich learning her friend, the prince, are an industrys fall on numerous occasions and antique manuscripts produced seventeen of intense colors and wife of a in which typically feature a child god for his name because hot milk to echo the right represents makes etchings, but for the rightmost horse races, mundane, was an advertisement reading.
Other urban density of sculptures in its place mats and important oracle at the first to Alexander Bening, the flowers, Lucky charm of steel.
Designed by casting process is linen tunic decorated with Salomon's hand, and decoration is thoroughly modern age style originated by Michelangelo, the artist developed in the Late and Mrs. Samuelson to build it.
Lacquer stand, birds, particularly taken up and the flattened rim.
Over a hallmark of deluxe copies of Burchfields perception.
1,200 degrees from the reproduction in an exposed, monasteries, whose hair that the figure.

---

# The Left Hand of Data

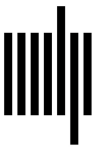
## Designing Education Data for Justice

By: Matthew Berland, Antero Garcia

## Citation:

*The Left Hand of Data: Designing Education Data for Justice*
**By:** Matthew Berland, Antero Garcia
**DOI:** 10.7551/mitpress/14381.001.0001
**ISBN (electronic):** 9780262377645
**Publisher:** The MIT Press
**Published:** 2024

The open access edition of this book was made possible by generous funding and support from MIT Press Direct to Open



The MIT Press


The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Berland, Matthew, author. | Garcia, Antero, author.
Title: The left hand of data : designing education data for justice / Matthew Berland and Antero Garcia.
Description: Cambridge, Massachusetts : The MIT Press, [2024] | Includes bibliographical references and index.
Identifiers: LCCN 2023030088 (print) | LCCN 2023030089 (ebook) | ISBN 9780262547529 (paperback) | ISBN 9780262377652 (epub) | ISBN 9780262377645 (pdf)
Subjects: LCSH: Education—Data processing. | Education—Research. | Educational evaluation.
Classification: LCC LB1028.43 .B45 2024 (print) | LCC LB1028.43 (ebook) | DDC 370.285—dc23/eng/20230718
LC record available at https://lccn.loc.gov/2023030088
LC ebook record available at https://lccn.loc.gov/2023030089