Analysis of historical sound change via the *Index Diachronica*

Wilson Biggs

April 20, 2023

Introduction

What is the *Index Diachronica?*

- A hand-compiled reference index of historical sound changes, intended for use in creating constructed language families
- Originally a collection of posts in a thread on the *Zompist Bulletin Board*; compiled into one index by Galen Buttitta
- "The Index is not an academic source, nor is it perfect – far from it."
    - Notation is only *mostly* consistent
    - A fair number of errors and dubious inclusions
    - Can still draw insights, but important to keep the caveats in mind
- Adapted from PDF to a more convenient and easily searchable HTML version, the *Searchable Index Diachronica*, by chri d. d.

What is the *Index Diachronica?*

17.7.2.1.1 Anglo-Frisian to Old English

Pogostick Man, from Wikipedia contributors (2011), "Phonological history of English". *Wikipedia, the Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Phonological_history_of_English&oldid=453796112>

ɑ̃: → õ:

V[+nas] → V[-nas]

{i,u} → Ø/ _# ! V[-long]C_#

k ɣ g → tʃ ʝ dʒ / "in certain complex circumstances"

What were my goals?

- What are the most common sound changes?
- How are sound changes influenced by their environments?
    - More specifically: how do neighboring consonants affect vowel changes?

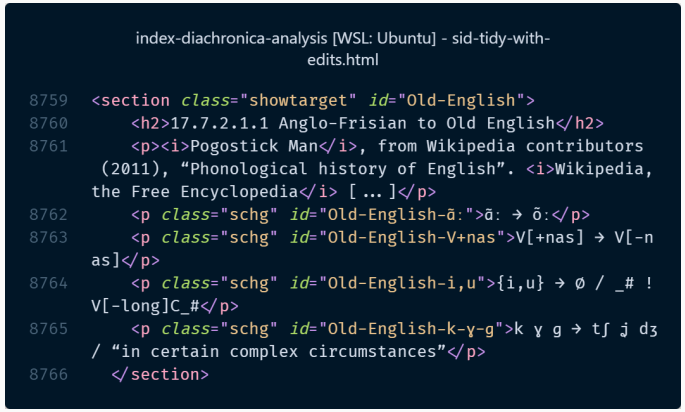
Parsing the data

Parsing the data

- The majority of my work
- Used a separate parsing script
- Lots of trial and error, running the parsing script over and over and fixing issues until nothing broke
- Had to manually edit the data a bit to remove things that would have made it impossible to parse
- Lots of complexity – my explanation of the steps will be leaving out some of the edge cases I had to handle

Pulling the content from HTML

The first step was to pull sound changes from the HTML and clean up any extra HTML tags. This was done using **Beautiful Soup**. The data I wanted was the name of the branch, the branches' and sounds' IDs, and the full text of the sound change rule.



Pulling out the environment

{i,u} → ∅ / _# ! V[-long]C_#

↓

 # ! V[-long]C#

index-diachronica-analysis [WSL: Ubuntu] - data_parsing_script.py

```python
# Split the rule up. This will inevitably include stuff I don't want but we can work out how to remove that stuff later
    # First split by the environment separator
    env_split = rule_string.split(" / ", 1)

    environment = ''

    if len(env_split) > 1:
        environment = env_split[1]
    else:
        # If no environment, but rule ends with some text in parentheses or quotes, consider that the environment
        parens_match = re.search(r'(.+[^→]) (\(.+\)|".+")$', rule_string)
        if parens_match:
            env_split[0] = parens_match.group(1)
            environment = parens_match.group(2)

    split_rules: list[Tuple[str,list[str],str]] = []
```

Parsing “optionals”

e(ː)j → i

↓

ej → i

eːj → i

index-diachronica-analysis [WSL: Ubuntu] - data_parsing_script.py

```
168  # if there are any optional bits, run the split with all possible combinations of with and without them
169      optionals = [(match.start(), match.end(), match.group(0)) for match in re.finditer(r'(\(.*?\))', env_split[0])]
170      if (optionals):
171          combinations = list(itertools.chain.from_iterable(itertools.combinations(optionals, l) for l in range(len(optionals) + 1)))
172          for combo in combinations:
173              combo_string = env_split[0]
174              # print(combo)
175              combo_string = remove_combos(combo_string, combo)
176              combo_string = combo_string.replace('(','').replace(')','')
177              # print(combo_string)
178              split_rules += parse_rule_steps(combo_string)
179      else:
180          split_rules += parse_rule_steps(env_split[0])
181
182      # only uniques
183      for split_rule in [sr for i, sr in enumerate(split_rules) if sr not in split_rules[:i]]:
184          rule = Rule()
185
186          rule.id = rule_id
187          rule.branch_id = branch.id if branch else None
188          rule.branch_index = branch.index if branch else None
189          rule.original_text = decoded
190          rule.environment = environment
191          rule.from_sound, rule.intermediate_steps, rule.to_sound = split_rule
192
193          rules.append(rule)
```

Parsing steps and multi-sound rules

aː ɔː → ɛː oː → eː ow → ej əw

↓

from_sound = "aː", intermediate_steps=["ɛː", "eː"], to_sound = "ej"

from_sound = "ɔː", intermediate_steps=["oː", "ow"], to_sound = "əw"

index-diachronica-analysis [WSL: Ubuntu] - data_parsing_script.py

```python
def parse_rule_steps(steps: str) -> list[Tuple[str, list[str], str]]:
    """Parse out the steps of a rule"""
    rules: list[Tuple[str, list[str], str]] = []

    steps = re.sub(r'^— ', '', steps)

    # split by the rule step separator
    rule_split = [s.strip() for s in steps.split("→")]

    if len(rule_split) < 2:
        print(f'Too few steps in rule: {steps} - skipping')
        return []

    if len(rule_split) > 2:
        from_sounds = split_sounds(rule_split[0])
        intermediates = [split_sounds(substr) for substr in rule_split[1:-1]]
        to_sounds = split_sounds(rule_split[-1])
        if (not (
                (len(from_sounds) == len(intermediates[0]) == len(to_sounds))
                and all(len(i) == len(intermediates[0]) for i in intermediates)
            )):
            print(f'Warning: mismatched lengths for rule: {steps} ({from_sounds}, {intermediates}, {to_sounds}) ({len(from_sounds)}, {[len(im) for im in intermediates],}, {len(to_sounds)})')

        for index, from_sound in enumerate(from_sounds):
            unbracketed_froms = handle_brackets(from_sound)
            unbracketed_tos = handle_brackets(to_sounds[index])
            for unb_from in unbracketed_froms:
                for unb_to in unbracketed_tos:
                    rules.append((unb_from, [im[index] for im in intermediates], unb_to))
    else:
        from_sounds = split_sounds(rule_split[0])
        to_sounds = split_sounds(rule_split[1])
        if (len(from_sounds) != len(to_sounds)):
            print(f'Warning: mismatched lengths for rule: {steps} ({from_sounds}, {to_sounds}) ({len(from_sounds)}, {len(to_sounds)})')

        for index, from_sound in enumerate(from_sounds):
            unbracketed_froms = handle_brackets(from_sound)
            unbracketed_tos = handle_brackets(to_sounds[index])
            for unb_from in unbracketed_froms:
                for unb_to in unbracketed_tos:
                    rules.append((unb_from, [], unb_to))

    return rules
```

Parsing brackets

{i,u} → ∅

↓

from_sound = "i", to_sound = "∅"

from_sound = "u", to_sound = "∅"

index-diachronica-analysis [WSL: Ubuntu] - data_parsing_script.py

```python
def handle_brackets(sound: str) -> set[str]:
    """Handles bracketed sounds"""
    sounds: set[str] = set()

    # find innermost bracket and recurse
    bracket_start: int = 0
    bracket_end: int = 0

    # Gotta do bracket matching. Ugh
    for (index, character) in enumerate(sound):
        if character == '{':
            bracket_start = index
        elif character == '}':
            bracket_end = index
            break

    if (bracket_start == 0 and bracket_end == 0):
        return set([sound])

    if (bracket_end == 0):
        bracket_end = len(sound)

    # print(sound[bracket_start : bracket_end+1])

    inner_sounds = sound[bracket_start + 1 : bracket_end].split(',')

    for inner_sound in inner_sounds:
        repl_sound = sound[:bracket_start] + inner_sound + sound[bracket_end + 1:]
        sounds |= handle_brackets(repl_sound)

    return sounds
```

What are the most common sound changes?

Questions to ask first

- What does it mean for two sound changes to be ‘the same’?
  - What if they have the same ‘from’ and ‘to’ sound, but different environments or intermediate steps?
  - I looked at both options - considering these the same, and considering these different.
  - Not much new information was gained by including environments or intermediate steps – the results were just the ‘most common sound changes with no environment or intermediate steps’
- Lots of sound changes are copied between different daughter languages in a single branch. Let’s count those as just 1 sound change.

The most common sound changes

Looking at only the `from_sound` and `to_sound`...

from_sound	to_sound	count
h	∅	46
w	∅	44
ʔ	∅	40
k	∅	37
j	∅	34
e	i	34
a	e	32
i	e	30
u	o	29
o	u	29

- So many deletions! Makes sense, since what can be deleted is less limited than 'what can become /i/'
- /h/ being removed is most common, which makes some sense – i.e. British English "history"
- Nothing really surprising here

The most common sound changes

What if we ignore deleted sounds?

from_sound	to_sound	count
e	i	34
a	e	32
i	e	30
u	o	29
o	u	29
ts	s	27
s	ʃ	26
g	k	24
k	g	23
a	o	23

- Still nothing too surprising here
- Mostly just simple vowel changes
- /ts/ → /s/ is the most interesting thing here

How neighboring consonants affect vowel changes

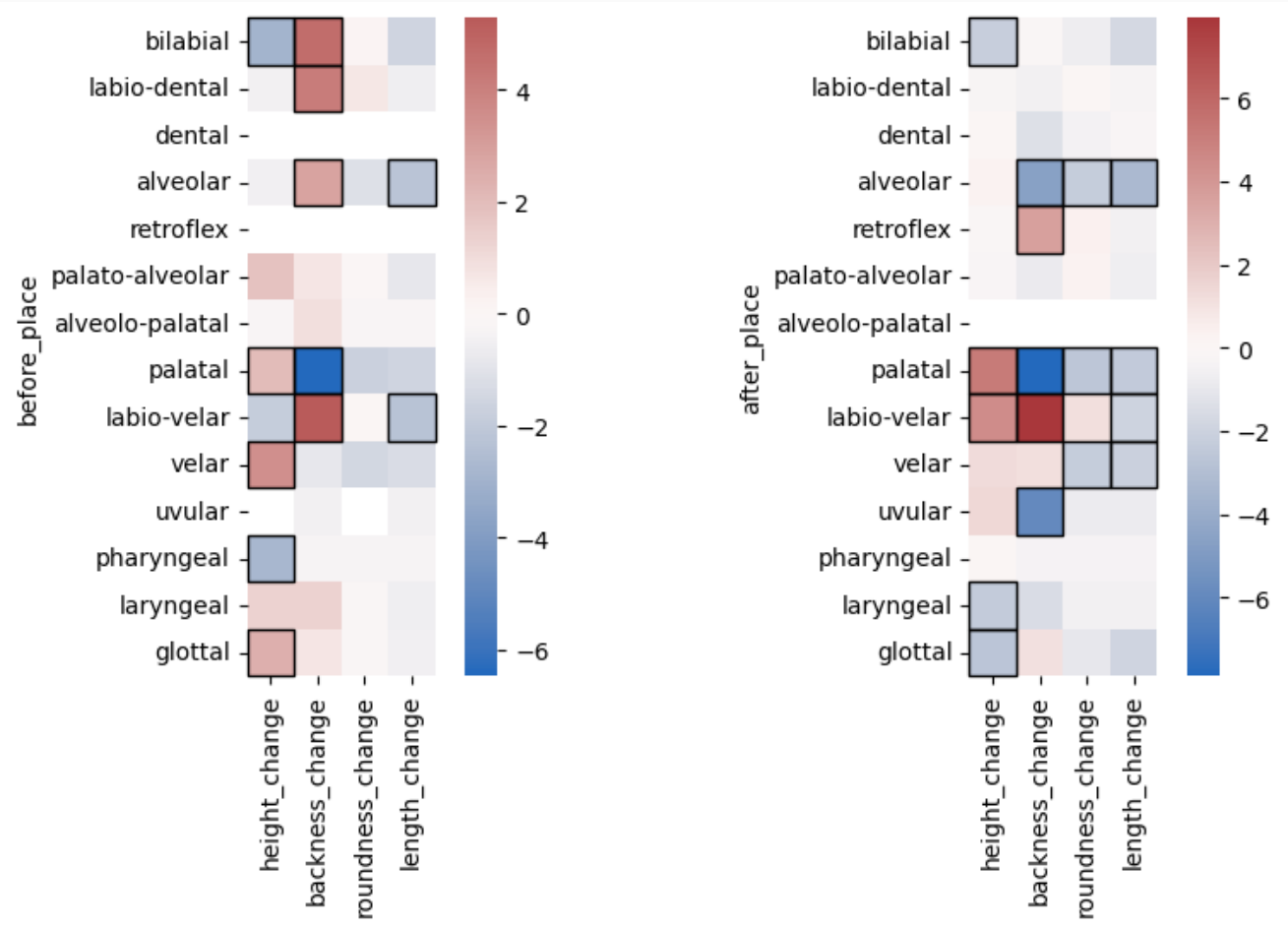
More data parsing!

- Filter down my rules to just vowel changes
- Pull neighboring consonants from the environment and/or rule
  - Parse environments – consonants before/after the underscore
  - e.g. `n_mV#` → `before = "n", after = "m"`
- Used **Gruut IPA** to split strings of IPA characters into individual phones (`/tʃuːz/` → `/tʃ/ + /uː/ + /z/`)
- Used **ipapy** to break down phones into their component features:
  - Consonants: voicing, place, manner, modifiers (palatalized, etc.)
  - Vowels: length, height, backness, roundness, modifiers (centralized, etc.)

How I performed the analysis

- Assigned a number to the values of each vowel feature (0 = open, 3 = mid, 6 = close) and found the difference between start and end sound to get that feature's "change"
- Looked at preceding and following consonants separately
- Did statistical t-tests to determine if the 'average changes' for different consonant features were significantly different from each other, and in what direction they differed
- Visualized these results as a heatmap
- Outlined squares signify statistical significance
- Excluded duplicate sound changes shared by daughter languages within the same branch, like before

Place of articulation



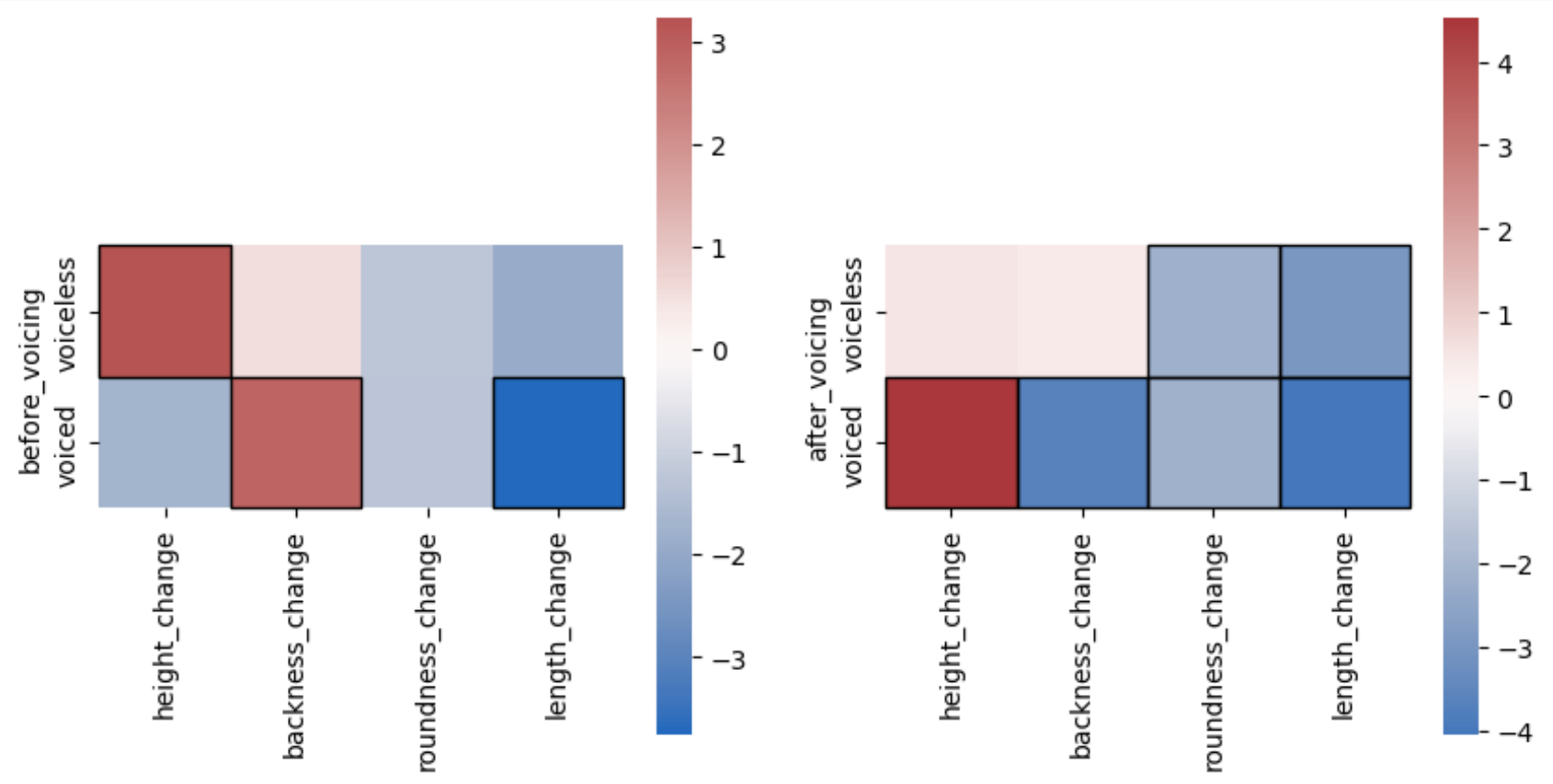
Height: negative = lowering (/u/ → /o/), positive = raising (/o/ → /u/)

Backness: negative = fronting (/u/ → /i/), positive = backing (/i/ → /u/)

Roundness: negative = unrounding, positive = rounding

Length: negative = shortening, positive = lengthening

Voicing



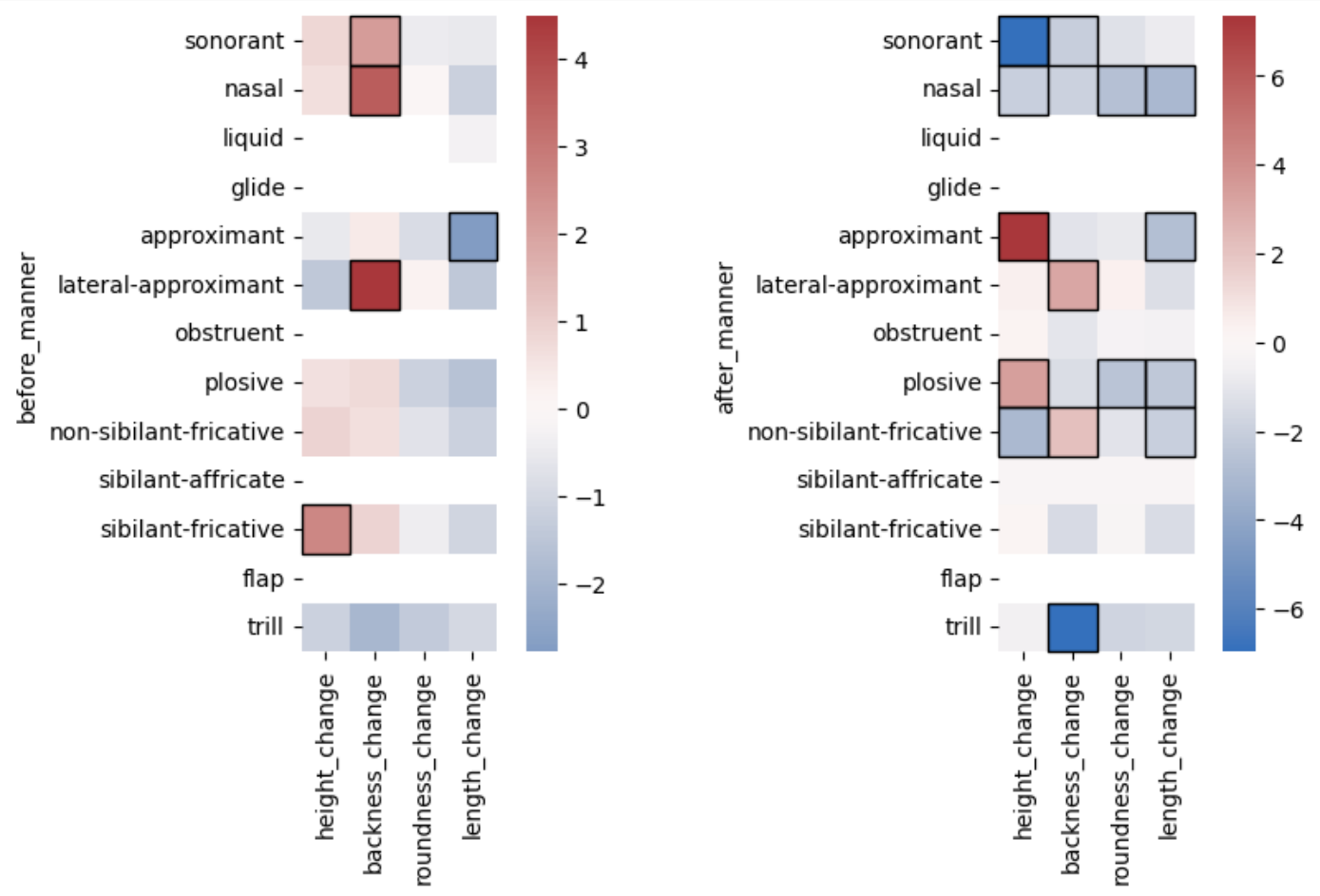
Height: negative = lowering (/u/ → /o/), positive = raising (/o/ → /u/)

Backness: negative = fronting (/u/ → /i/), positive = backing (/i/ → /u/)

Roundness: negative = unrounding, positive = rounding

Length: negative = shortening, positive = lengthening

Manner



Height: negative = lowering (/u/ → /o/), positive = raising (/o/ → /u/)

Backness: negative = fronting (/u/ → /i/), positive = backing (/i/ → /u/)

Roundness: negative = unrounding, positive = rounding

Length: negative = shortening, positive = lengthening

Takeaways

- Some results lined up with what I expected (palatals are associated with fronting, /w/ is associated with backing) but I couldn't explain most results
- Each combination of features was not evenly distributed, so they might be contaminating each other's results
- I tried using linear regression to try and untangle this, but those results were even messier - not included here

Conclusion

Future ideas

- Untangle the effects of these different variables to try and determine which features are actually having which effects
- Do the same analysis, but for consonant changes
- Fix some of the errors in my data – there is a big PDF of corrections to the *Index Diachronica* that someone has compiled
- Re-write the *Index* using more standard notation, like PhoMo (a sound-change notation intended to be parsed by software and used to automatically apply sound changes to words)
- Interactive tools – e.g. put in a word or IPA string, get the most likely ways that word could evolve

Questions?