Background

Alphabet compression problem

Let L be a language consists of words from an alphabet Σ, and (for simplicity) we assume that all words are non-empty and less than length n, i.e.

$$L \subset \bigcup_{i=1}^{n} \Sigma^i = \{c_1 c_2 \dots c_i | c \in \Sigma, i \leq n\}$$

In case that L is sparse, say $|L| \ll |\Sigma|^n$, we might want to compress that alphabet for better encoding.

Consider another alphabet σ with $|\sigma| \ll |\Sigma|$, and a mapping $f : \Sigma \to \sigma$. Under such mapping we define a smaller language l

$$l_f = \{f(c_1)f(c_2)\dots f(c_i) | w = c_1 c_2 \dots c_i \in L\}$$

The goal is to find the mapping that minimize information loss caused by some words in L being mapped to the same word in l:

$$f = \arg\min D[f] = \arg\min \left(|L| - |l_f|\right)$$

Where does this problem come from?

Chinese character set is enormous (90,000+ in Unicode, and still adding more). Before 1980s, Chinese typists and publishers use keyboards with thousands of keys to input Chinese, which is called "大键盘", *lit.* "large keyboard".

Some better input methods make use of the fact that Chinese characters are composed of "radicals" (components), and designed keyboards where each key is assigned a radical; i.e. "汉" = "氵" + "又", "字" = "宀" + "子", so you can use radicals "氵" "又" "宀" "子" to input them. This approach is called "中键盘", *lit.* "medium-sized keyboard". However, 中键盘 still contains 500+ keys, which is hard to use.

Input method with 中键盘 can be viewed as a sparse language L where each "letter" in the alphabet Σ is a radical, and each Chinese character is a sequence of radicals (with length truncated at n). The reason that L is sparse is because $500^3 = 125,000,000 \gg |L|$, and most characters have $n \geq 3$ radicals.

The hardness to use 中键盘 naturally raises a problem: can we compress Σ of that sparse language L to find a much smaller alphabet σ that could still distinguish characters?

The resulting method is called "小键盘", *lit.* "small keyboard" which allows us to input Chinese with general purpose keyboards with latin alphabets. A lot of input method with 小键盘 were designed and published 1980s and 1990s in China, and novel input methods are still being invented nowadays.

Method

Settings and dataset

We assume $|\sigma| = 26$, i.e. the latin alphabet $\sigma = \{\mathrm{A}, \mathrm{B}, \dots, \mathrm{Z}\}$. Ways to decompose Chinese character is not unique, so we will use this file https://github.com/tansongchen/c42/blob/main/assets/decomposition.dat as a reference, which specifies a 3-radical sequence ($n = 3$) for each character in character set GB/T 2312-1980 which contains 6763 characters. Characters that have less than 3 radicals will be postfixed with at most two latin letters from its phonetic romanization sequence (i.e. 汉语拼音), which is also available in this file.

Finding approximate solutions

Solving $\min_f D[f]$ is in general NP-hard. However, one might start from some simple considerations.

Let c, c' be two letters ("radicals") from Σ, and consider the following mapping:

$$f^{(c,c')}(x) = \begin{cases} * & x = c \text{ or } x = c' \\ x & \text{otherwise} \end{cases}$$

This is an "elementary" compression that reduces the alphabet size from $|\Sigma|$ to $|\Sigma| - 1$, coalescing c and c'. We define a correlation matrix M indexed by c, c',which is

$$M(c, c') = |L| - |l_{f^{(c,c')}}| = D[f^{(c,c')}]$$

The algorithm to construct such a matrix is:

```
for (i, ei) in enumerate(elems)
    for (j, ej) in enumerate(elems)
        if i >= j
            continue
        end
        merged = map(language) do word
            map(word) do elem
                (elem == ei || elem == ej) ? "*" : elem
            end
        end
        correlation = length(language) - length(Set(merged))
        matrix[i, j] = correlation
        matrix[j, i] = correlation
    end
end
```

Some radicals really don't like each other, for example, in character set GB/T 2312-1980 which contains 6763 characters

$$M(扌, 氵) = 75$$

which means that 75 pairs of characters will become indistinguishable if we map "扌" and "氵" onto the same letter in a smaller alphabet. To name a few: "抄" and "沙", "打" and "汀", "挂" and "洼", ...

However, some radicals are happy to live with each other

$$M(一, 讠) = 1$$

So M really contains some information for input method designers to use.

Greedy algorithm for generating f by sampling from M

1. Rank the radicals from mostly used to seldomly used;
2. Map top 26 mostly used radicals respectively to A ~ Z, say 口, 木, 日, 氵, 艹, 一, 扌, 土, 亻, 钅, ...;
3. For each of the remaining radicals c,
    1. For each of $\mathrm{A} \sim \mathrm{Z} \in \sigma$, sum all $M(c, c')$ for each c' already mapped to that key
    2. Sample a letter x from $\mathrm{A} \sim \mathrm{Z}$ which gives the minimum conflict
    3. Map c to x

This is a randomized method, and we run this 100 times to get an averaged result of $D[f] \approx 1200$. Not bad, but we can still improve.

```
function arrange_with_matrix(elems, matrix)
    assign = Dict{String, Char}()
    alphabet = collect("abcdefghijklmnopqrstuvwxyz")
    reverse = Dict{Char, Vector{Int}}()
    k = length(alphabet)
    for (i, elem) in enumerate(elems)
        if i <= k
            assign[elem] = alphabet[i]
            reverse[alphabet[i]] = [i]
        else
            f = c -> sum(other -> matrix[i, other], reverse[c])
            m = map(f, alphabet)
            indices = alphabet[m .== minimum(m)]
            char = rand(indices)
            assign[elem] = char
            push!(reverse[char], i)
        end
    end
    assign
end
```

M does contain some pattern but only capture first-order interactions between the radicals. However since $n = 3$, two characters with very different radical sequences can be mapped to the same thing because of compression. e.g. "明" = "日" + "月", "杜" = "木" + "土". if $f(日) = f(木) = a$, and $f(月) = f(土) = b$, then they will also be indistinguishable even if they are very different. Therefore, we need a better matrix to capture some structure within radical sequences, not just single radicals.

Improve estimation M to M^*

We consider the empirical correlation matrix given by sampling a large number of different f's. For each of them, we define

$$M^*(c, c') = \{\#\text{ of characters become indistinguishable because of } c, c' \text{ merged }\}$$

And let M^* to be the average of all of them. This is purely statistical quantity which don't have great physical meaning like M, but this quantity is calculated from real-world mappings $f : |\Sigma| \to |\sigma| = 26$, not some hypothetical $f : |\Sigma| \to |\Sigma| - 1$.

Using this matrix for the greedy algorithm, we got $D[f] \approx 1100$; not improving by a great deal :)

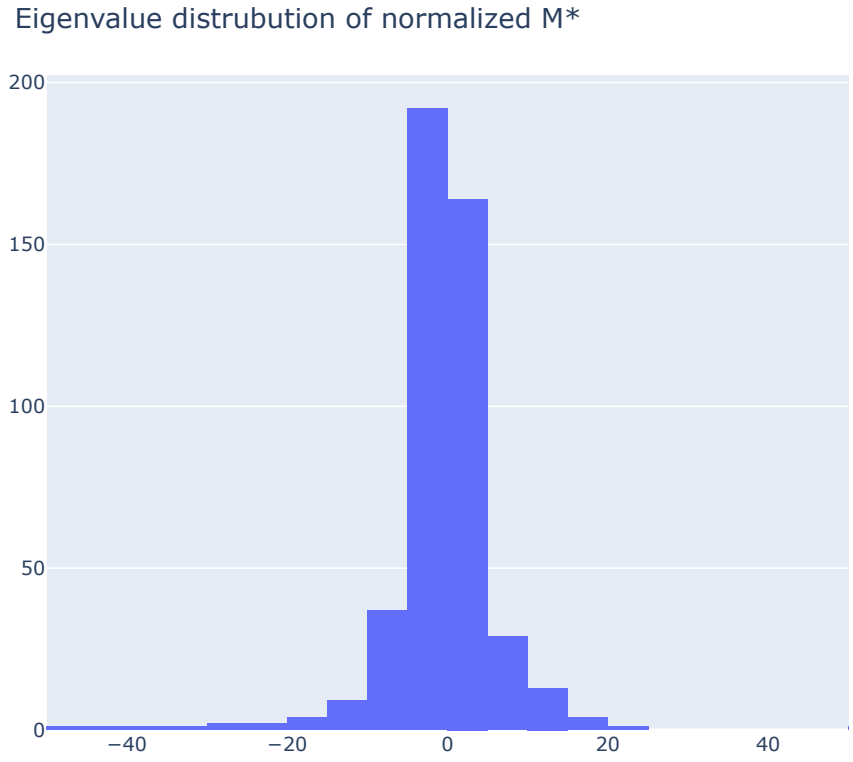
Denoising M^*

M^* does contain more information than purely M, but the noise is also higher. We would like to discover some real patterns from M^* without being immersed in noise.

Let $\tilde{M}$ to be properly normalized M^* and spectral decomposed as

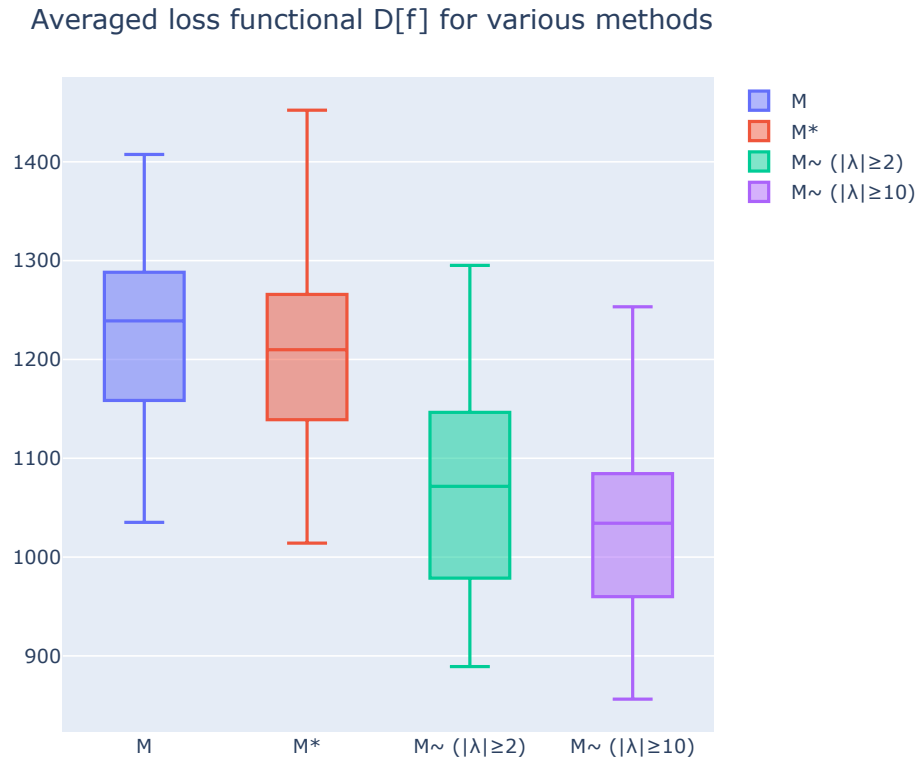
$$\tilde{M} = \sum_i \lambda_i v_i v_i^T$$

such that portion with $|\lambda| \leq 2$ can be safely discarded. In practice, we found that discarding $|\lambda| \leq 10$ altogether gives better results, many significant eigenvalues can be as large as 50.



Result summary

For each of the methods mentioned above, we run 100 trials of the greedy algorithm with the correlation matrix provided by that method, and the result is shown below:



The result is a bit fluctuating, but at least statistically we can conclude that using de-noised matrix $\tilde{M}$ can improve the result given by the greedy algorithm.

Further thoughts

M^* or de-noised $\tilde{M}$ might be used to sample "thoughtful" random moves in meta-heuristic algorithms, like simulated annealing or genetic algorithm, instead of naive greedy algorithm.

Reference

- 汉字电脑输入法形码设计三原理. 王永民. 两岸中文电脑输入研讨会论文 (1993)
- 汉字编码输入法综述. 戴石麟. 重庆大学硕士学位论文 (2003)
- Edelman, A., & Wang, Y. (2013). Random matrix theory and its innovative applications. In *Advances in Applied Mathematics, Modeling, and Computational Science* (pp. 91-116).