institut national d'histoire de l'art

INHA 20 ans

ENRICHING RICH DATA

INTERNSHIP

FOR MASTER OF SCIENCE IN DIGITAL HUMANITIES

Ravinithesh Annapureddy

EPFL

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

September, 2021 - February, 2022

ABSTRACT

The tools of digitization have brought a change in the ways to interact with cultural heritage. Especially manuscripts and old books that are difficult and not in great shape to access physically can be accessed through their digitized versions. This is taken one step ahead by transcribing the text in these books through Optical Character Recognition (OCR) processes. The OCR is imperfect and misinterprets the text leading to spelling errors. Nevertheless, correcting them is important to access textual data. This work proposed a pipeline to clean the text obtained by OCRizing the city directories of 19^{th} Paris. The pipeline uses predominantly the OCRized data with a bit of external data and some human intervention to correct the misspelled words. Experiments carried out revealed a significant improvement in the OCR error correction rate. While future work can improve the pipeline to use more human intelligence to correct rare mistakes, the cleaned data itself can be used to carry out other types of research on the dataset about the people of Paris.

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 CONTEXT

The *Richelieu. Histoire du quartier* was a joint project of the Institut national d'histoire de l'art, Centre allemand d'histoire de l'art, Bibliothèque nationale de France, École nationale des chartes, Paris 1 Panthéon-Sorbonne université and Sorbonne Université [1].

In the venture to study the history of the Richelieu district, the previous project utilized the scanned city directories and designed an automated data extraction process to produce data sets of the population during the 19^{th} century [2].

The dataset contained the Name, Profession/Activity, and Address (street name and number) of the people in Paris between 1839 and 1922. The extraction pipeline processed 27,000 scanned pages from 56 available directories in 89 years. The extraction resulted in approximately 4.5 million lines of data, which is nearly 80,000 addresses per year.

In addition, a subset of the data with addresses (200,000 addresses) belonging to the Richelieu district was extracted [1]. A post-processing correction was performed on the addresses in this subset to remove the mistakes during the transcribing process and add the geographical coordinates (latitude and longitude).

Although the data about people in the Richelieu district was extracted and geo-referenced, the profession/activity information has not been processed after transcribing owing to spelling mistakes. At the same time, abbreviations such as *fabr., fab., entrepr.* were used to save the printing space. Both of these conditions hinder the researchers from searching for people based on a profession whose spellings are not interpreted correctly.

This internship was built on the previous work and improved the shortcomings of the profession/activity aspects of the dataset.

1.2 GOALS

The internship aimed to normalize and clean the professions of the people in the dataset by removing the spelling errors and filling the full forms of abbreviated words. Secondly, make available the data for the Richelieu district with the cleaned professions and Geo-referencing to the public by publishing it to a data repository.

CHAPTER 2

METHODS

2.1 TAGS FOR PROFESSION

The city directories used to create the dataset were produced in the 19th century by various companies under different conditions. Between 1839 and 1922, not only the language but the activities of the people have transformed. As the technologies boomed, new activities came into existence. In some cases, one activity was further divided into multiple activities over time or one activity acted as an umbrella activity for multiple sub-activities. At the same time, individuals described their profession non-uniformly based on their specializations. Besides, as the directories were intended to be read (in contrast to being searched through computer/text), the words in the professions (or street names) were abbreviated based on the sentence.

Keeping in view this varied background, the professions/activities in the directories are not uniform and do not have consistent strings (sentences). Thus it is impractical to correct the profession or activity strings and while searching for the data, the users are prone to use keywords than sentences. Hence, the profession strings were cleaned/corrected by creating tags to navigate through the above-mentioned requirements and constraints. The tags were created based on the words in the profession string for each entry.

The tags are single-word keywords that provide the essential meaning to the sentence. In this context, the keywords are those words in the profession that provide a meaningful understanding of the profession without the stop words and connecting words that are required to build an expressive sentence. However, the words from the profession string itself will not suffice as the words contain several mistakes (presented in 2.4) and need to be corrected to be uniform across the dataset.

In designing the process of creating the tags, the following assumptions were made.

1. Most of the words have a correct spelling
2. The correct spelling appears more frequently than the misspelled one.

Based on these assumptions, the word having mistakes were corrected by using the same words in the dataset that were in a list of correct words (see 2.3) and those that appear very frequently.

In the remainder of the section, the types of mistakes in the words are described. Then the pipeline to create tags from the profession strings is presented.

2.2 SIMILARITY OF WORDS (TOKENS)

In the pipeline designed to create the tags, at various occasions a pair of words are compared to check if they are similar. In this section, a brief description of how *similarity* was used in the project is provided.

The token (or words) similarity is the Levenshtein similarity between the tokens. The Levenshtein distance between two strings is the "minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other" [3]. The distance is normalized between 0 and 1 and converted into similarity. For this project, the similarity is calculated using the RapidFuzz library's *fuzz.ratio* [4] function without any processing of the strings.

The reason for choosing this library (while other similar libraries are available) is that the function to calculate the similarity between strings accepts a threshold for similarity. It uses the threshold as an early stopping criterion in calculating the similarity between the strings (based on the lengths of the strings). It is not possible to obtain a high similarity between the strings when the length of them is significantly different. It returns zero as the similarity when the similarity is less than the given threshold.

2.3 CORRECTLY SPELLED WORDS

The key idea behind the pipeline lies in the process of correcting the spelling of wrongly spelled words. To ascertain which words are already correct spelled, three external sources of data were used to curate this list of correctly spelled words.

For a lexicon of the French language, a set of words is provided by ortolang as Morphalau3 [5] that has 159,271 lemmas and 954,690 inflected forms of modern French was used.

For proper nouns, the proper nouns dictionary is sourced from prolex-unitex [6].

For the names of the streets, the table of streets of Paris is obtained from [7].

2.4 PROBLEMS IN PROFESSION STRINGS

In order to comprehend the pipeline designed to create tags for each profession string, it is crucial to understand various types of mistakes/issues in the profession string in the current state of the data. Four categories of problems were identified by iteratively combing the dataset manually.

2.4.1 NON-ALPHA-NUMERIC CHARACTERS

During the OCRization of the directories, some characters and symbols in the text were misinterpreted as special characters. For example, the symbols used for medals were interpreted as **#** or ***** or **¥**. In another case, **l** was interpreted as **!** or **œ** as **@**.

2.4.2 SINGLE-WORD BROKEN INTO MULTIPLE WORDS

During the OCRization of the directories, some words were broken into multiple words. The most common reason for such a behavior is the presence of the text in two lines in the directory itself. For example, **l'agriculture** was broken into **l'a** and **griculture** as the the sentence was present into two lines.

2.4.3 WORDS WITH SPELLING MISTAKES

The most significant problem is the issue of words having spelling mistakes. The most probable reason for these spelling errors is the scan quality of the directory page. The spelling errors range from the

misinterpretation of letters such as **l** for **i** to **m** interpreted as **ln**.

2.4.4 ABBREVIATIONS FOR WORDS

After the spelling mistakes, the next major problem is shortened words. The words in the profession entry were abbreviated to save space in printing. However, these abbreviations are not consistent and also contain spelling mistakes. The examples include *fabr., fab., jabr., propriét., ingén., ing.*

2.5 TAG GENERATION PIPELINE

The pipeline to create tags is composed of six steps as shown in Figure 2.1. Each component of the pipeline deals with one of the problems mentioned above. This section further illustrates each step of the pipeline with examples.

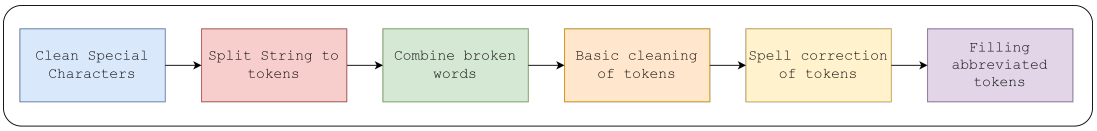

FIGURE 2.1
A high level view of the tag generation pipeline

2.5.1 CLEANING SPECIAL CHARACTERS

The OCR process misinterpreted the symbols and letters as special characters (which are neither alphabets nor digits). The presence of these characters will hamper the users searching in the data.

Broadly, two types of cleaning - generic and specific - were performed on the data. The generic steps were rule-based and applied uniformly irrespective of the special character. In the specific step, the entries were corrected individually. Below, the idea behind this cleaning is presented. Refer to the Jupyter notebook (in the code accompanying this report) to check the individual steps.

2.5.1.1 GENERIC CLEANING OF SPECIAL CHARACTERS

Generic cleaning also has two categories. The first type of generic cleaning was on the special characters present at the start (followed by a space), at the end (preceded by a space), and/or surrounded by space. The characters were removed from the profession string when present in the mentioned conditions.

Example: The profession *manufacture de porcelaines ; dépôt* is changed to *manufacture de porcelaines dépôt* or *actrice & l'Opéra-Comique* is changed to *actrice l'Opéra-Comique*. In the second example, although **&** can be modified to **et**, it is removed because the stopwords such as *et* are anyway removed at a later stage.

The second generic step is to replace the same mistakes across the dataset. This generic step is specific to the special characters but applied across the dataset instead of individual entries.

For example, the *d'* in some cases appears as *d*' or ®. All such cases were replaced with *d'*.

2.5.1.2 SPECIFIC CLEANING OF SPECIAL CHARACTERS

The second category of cleaning was for individual words containing the special characters i.e. they are present in the word. For each case, the entry was printed, compared manually with the scanned document (through Gallica) and the changes were made.

Profession String	Tokens	Comments
a.p. moller-maersk	[a.p., moller, maersk]	Split at space and hyphen
aire-sur-l'adour	[aire, sur, adour]	Split at hyphen and removed l as it is a complementary pronoun
clin d'œil	[clin, œil]	Split at space and removed d as it is a complementary pronoun
chien de mer	[chien, de, mer]	Split at space
fab.de voitures	[fab., de, voitures]	Split at space and split at dot (de is stop word and present in list of correctly spelled words)

TABLE 2.1
Examples of splitting a string

The examples would be changing from *ancien consul du grandduch& de hesse* to *ancien consul du grand-duché de hesse* or *dépositaire de mar)quinerie* to *dépositaire de maroquinerie*.

2.5.2 STRING TO TOKENS

As mentioned earlier, instead of cleaning the strings (sentences) describing the profession from the dataset, tags (single words) will be created for each entry. These tags will contain meaningful words that help in understanding the occupation and exclude the stop words and connecting words.

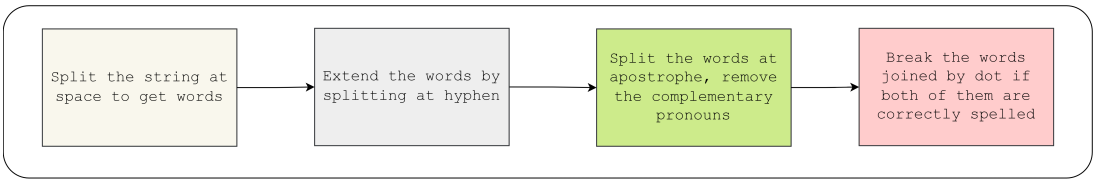

FIGURE 2.2
Steps in splitting a string to tokens

The first step to move to tags is to break the string into a list of words. The list of words - referred to as tokens - was obtained by splitting the string at space, a hyphen, apostrophe, and dot [Figure 2.2].

While space is an apparent choice, the reason for choosing the other delimiters are

- **Hyphen**: Generally, the hyphens indicate a link between words or the overflow of the line, and the em dashes are used to provide emphasis. However, the observation of the dataset has revealed that their usage is not consistent i.e. the hyphen and the em dash were used interchangeably by the OCR process. To standardize the process, the strings are split at the hyphen.
- **Apostrophe**: The apostrophe is frequently used to combine the complementary pronoun and a word starting with a vowel. Thus, if the string has a complementary pronoun before the apostrophe, it is split at apostrophe, and the part that doesn't contain the complementary pronoun was retained.
- **Dot**: In the dataset, the words were abbreviated by placing a dot after the first few characters. Nevertheless, there were cases where a word with a dot at the end and the next word in a string were combined (the OCR system ignored the space between them). Such tokens were split at the dot and replaced the token if the resultant tokens were belonging to a set of correctly spelled words.

The examples of splitting a string is shown in Table 2.1.

2.5.3 COMBINING BROKEN WORDS

During the OCR process, a few words are split into parts because of the wrong interpretation of the text and/or low quality of the scan and/or presence of text in multiple lines. Combining these words to a single word before spell correction is relevant because the split words might change to a different word in the process of correction.

1. Split the profession string at space, a hyphen, apostrophe to create a token set for each unique profession[1].
2. Create a counter for each unique token.
3. Go over each token set, if either of the consecutive entries is not in the list of correct words and either of them has a frequency less than a threshold then try to concatenate these entries.
4. Check if the concatenated word or a word that is similar with a certain threshold exists in the already existing tokens.
    - If the exact concatenated word is not present in the existing tokens, then the **process**[2] module of RapidFuzz library [4] is used to extract the closet word from a list of words.
5. If such a word exists and if the frequency of such a word is greater than or equal to all the individual consecutive tokens used to obtain it, then replace the consecutive tokens with the found word.
6. Update the token set for the profession using the words obtained after combining.

Few examples from the dataset are,

- *fabr. d'éta lages et d'armures* is updated to *fabr. étalages et armures*
- *à la bibliothèque impé\ riale* is updated to *à la bibliothèque impériale*
- *ancien chef de bur. au minist. des financ.* is updated to *ancien chef de bureau minist. des financ.*

2.5.4 BASIC CLEANING OF TOKENS

In the previous step 2.5.3, the profession strings were split at space, a hyphen, apostrophe. In this step, the words are also split at the dot. In addition, the words having less than 3 alphanumeric characters and those in the stop words list are removed[3]. Then the words that are potentially indicating a number are removed. Following the numbers, the words that are not in the list of correct words are converted into their normal form and replaced with the most frequent non-normalized form. Lastly, the tokens containing a dot in the middle are split into separate tokens when the part of the word represents an abbreviation. This step will finally result in a set of tokens for each profession.

2.5.4.1 CLEANING THE WORDS SHORTHANDED FOR NUMBERS

The 3 letter words with *re* or *er* were short hands for writing numbers like *première* or *troisième* or *quatrième*. These will be removed. The entries with *re* or *er* present are *lre*, *fer*, *ire*, *ler*, *ɟre*, *tre*, *ser*, *pre*, *fre*, *!re*, *ère*, *are*, *ier*, *bre*, *jre*, *jer*, *der*, *mer*, *her*, *(re*, *nre*, *gre*, *ere*, *īre*, *cer*, *ter*, *per*, *ger*, *ìre*, *ver*, *yre*, *qre*, *cre*, *vre*, *mre*, *ner*, *íre*, *dre*, *rer*, *ber*, *ure*, *îre*.

[1] Here the complete pipeline to split the string into tokens is not applied (specifically splitting at dot). As the word can be split arbitrarily, the small words or stop words are not ignored when splitting the profession string into parts.

[2] The *process.extractOne* function is used for this purpose. It accepts the arguments in the following order: The seed word (concatenated word), The list of words to search in (The list of unique tokens), A processor to pre-process the strings, A scorer to calculate the similarity between strings (fuzz.ratio) and The minimum similarity between the seed word and the word in the list of words is to be selected and returned.

[3] The list of stop words in French language provided by [8] was used

However, some of these three-letter words are present in the list of correct words or do not indicate a number. The rows containing these words were studied and only the following were replaced

- All the three-letter words starting with *l* or *q* or *any digit* and ending with *re* or *er*.
- All the three-letter words starting with *i* or *f* or *m* or *(* or *ī* or *!* or *ì* or *į* or *j* or *î* or *í* and ending with *re*.
- All the three-letter words starting with *g* or *a* or *p* or *b* or *y* and ending with *re* and followed by a space and *i*
- All *ter* words
- All *jer* words followed by a space and *i* or *a*.
- All *tre* words followed by a space and *ins* or *cl* or *iss*.
- All *fer* words followed by a space and *inst* or *ar*.

The idea of checking if the word is followed by a certain character comes from the fact that most of the time the numbers are used in the content of mentioning the *arrondissement* or the *instances*. Hence the entries are checked for *i* or *a* or *ins* or *iss* or *inst* or *ar* etc.

2.5.4.2 NORMALIZING (AND DENORMALIZING) TOKENS

Some tokens in the dataset have non-french alphabets and some words that have ligature are sometimes broken into individual characters and sometimes not. For example, eggs as *œufs* and *oeufs*, sister as *sœur* or *soeur*, variants of the words *vins*: *vinš, vïns, viñs, vîns, víns, vinş, viṇs, vìns, viń́s, vīns*.

For tokens of each profession after removing the words related to the numbers, those that are not in the set of correctly spelled words were converted to normal form (i.e. converted into ASCII form) using the unidecode [9]. If the normal form was different from the non-normal form then the token was temporarily converted into a normal form. Then all the tokens having the same normal form were replaced with the non-normal form of the token (having the same normal form) that appeared the highest number of times in the dataset.

2.5.4.3 SEPARATE MERGED ABBREVIATIONS

In the data, sometimes the profession has two abbreviations to describe it and they are not separated by space during OCR. For example *imprimeur.lithogr.* or *négoc.commiss.*. In this step after normalizing the tokens, these types of tokens where they were joined by a dot were split into multiple tokens. The algorithm is as follows:

- For each token that is not in the list of correctly spelled words and contains a dot (The tokens that have only one dot and that one at the end are ignored as they are potential abbreviations.)
    1. The token is split at the dot and checked
    2. If the majority (more than half plus one) of the resultant split has a length greater than one (Otherwise, the token could be a spelling mistake that has dot instead of alphabets and such token can be ignored), then for each sub token (the result of the split):
        (a) Split the token at apostrophe if it contains an apostrophe (this provides a chance to split the tokens that have multiple apostrophes which were not split earlier)
        (b) If the token is not in the stop words, add a dot to the token at the end, if the token with a dot at the end has a higher frequency than the token without the dot.

3. If the result of splitting at dots is not the same as the result after updating tokens based on the frequency, then the token is updated to the list of new tokens obtained from the split.

2.5.5 SPELL CORRECTION OF TOKENS

In the fifth step of the pipeline, the tokens that do not have a correct spelling (identified through the list of correct words) were merged with the closest correctly spelled word iteratively until there were no more possible merges. The merge was performed in three rounds (see Figure 2.3. In the first two rounds, the context of the tokens (that is considering the other tokens in the profession string) was taken into account. In the last round, the spellings were merged ignoring the context.

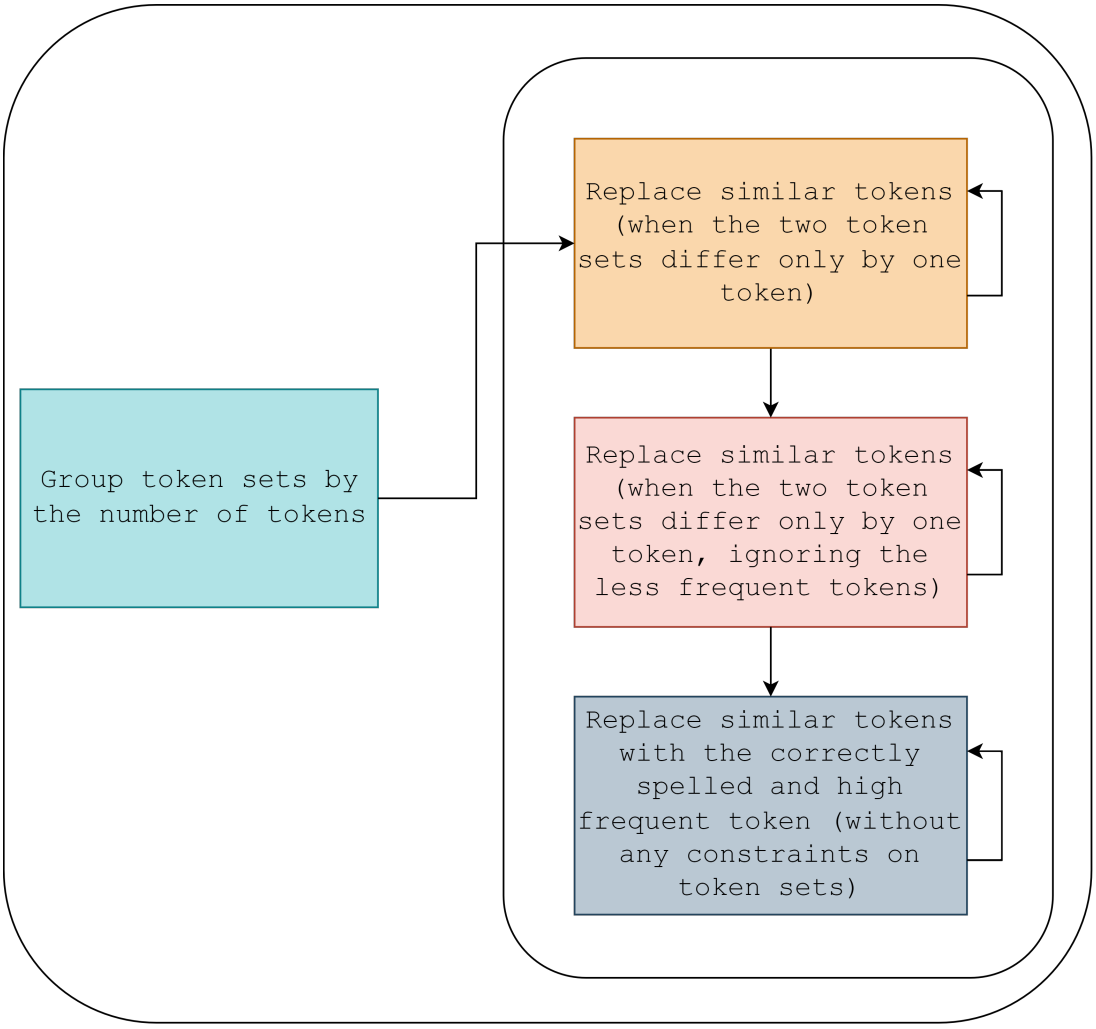

FIGURE 2.3
A high level view of the spelling correction step of the tag generation pipeline

2.5.5.1 ALGORITHM FOR MERGING TOKENS

- In the first round, the tokens are merged according to the steps described in Section 2.5.5.2
- In the second round, the less frequent tokens (after round 1) are removed temporarily in each token set per profession, and the same process as round 1 is repeated.
  - The count to determine if a token is frequent or not is defined beforehand.

  - For now, any token that appears less than 50 times is treated as a low frequent token. The choice of 50 comes from the fact that 56 years of city directories were OCRized.

- In the third round, the low frequent tokens per token set are added back to the per profession token sets (after round 2). However, this time the tokens are considered without the context, and the same process as round 1 is repeated.
- After round 3, the per profession token sets (after round 2) are updated to change the tokens that are merged.

2.5.5.2 ALGORITHM FOR CONTEXTUAL MERGING

1. While the tokens can be merged, continue the iteration
2. Create a counter for each unique token present in the dataset at this step
3. Group the per profession token sets based on the length of the sets.
4. For each length of the token set
   (a) For each token set of the considered token set length
      i. Compare with all other token sets of the same length, if the two token sets differ only by one token (i.e. all the tokens are the same except one) and if the tokens are mergeable (2.5.5.3)
         A. With the same tokens in both the token sets as a base, store all such pairs of mergeable tokens as values (Intuitively this means that the pairs of tokens are probably the same and they can be merged with confidence).
            - For the tokens of length one, there is no specific base and a dummy string is considered as the base and the merge is continued.
5. For each base of tokens set produce a base-wise update mapping that stores the token and the token to which it should be updated (2.5.5.4).
6. Combine the base-wise update mapping's produced per base of token sets to generate a token update mapping for the iteration.
   (a) This step is performed to combine the updates of tokens from the same token set that were merged under different bases.
7. Using the token update mapping for the iteration, update the profession string to token sets mapping.
8. If the profession string to token sets mapping is unchanged from the previous iteration (i.e. if no token is updated in the iteration) then stop the merging, else continue the merging from *2*.

2.5.5.3 ALGORITHM FOR CHECKING TOKENS MERGEABILITY

Two tokens are mergeable if the tokens have a similarity greater than a certain threshold (defined apriori) and

1. If the high frequent token is in the list of correctly spelled words and the low frequent token is not in the list of correctly spelled words.
2. If both the tokens have the same frequency and only one of them is in the list of correctly spelled words

In both cases, the token that is not in the list of correctly spelled words is merged to the word that is in the list of correctly spelled words.

2.5.5.4 ALGORITHM FOR MERGING TOKEN PAIRS

For a given base and the pairs of tokens that are mergeable, a two-step process is followed to determine to which token the pairs are changed.

1. Loop over all the pairs to create a mapping with key as the token and value as a list of tuples containing that token that it can be merged with and the similarity between the two tokens.
2. For each of the tokens to be merged,
   (a) If there is only one token that it can be merged with then the token to be merged is changed to the token it can be merged with.
   (b) else if there are more than one possible tuples, then the token to be merged is changed to the token with the highest similarity score.
      i. If there are multiple tokens with the same similarity score, then the token to be merged is changed to the token that has appeared more frequently among those that have the same frequency.
   (c) If the token to merge with is not found (i.e. multiple tokens with same similarity and frequency), then the token is left as it is without merging.

2.5.6 FILLING ABBREVIATED TOKENS

In the previous step, the tokens that do not have correct spelling were merged with the closest correctly spelled word iteratively. In this last step of the pipeline, the tokens that are classified as abbreviations are completed into full words. Any token containing a dot (.) is classified as an abbreviation.

The idea of completing the abbreviations is drawn from the previous step i.e. to use the contextual tokens to decide the abbreviations of the words. After the abbreviations are filled with contextual information, for the second time, the remaining ones are filled based on only frequency and similarity.

2.5.6.1 ALGORITHM FOR COMPLETING THE ABBREVIATIONS

1. The current token sets for all the entries in the dataset are considered.
2. The abbreviations are completed using the contextual tokens (see 2.5.7)
3. Collected the tokens that have a dot as potential abbreviations after *2*.
4. The abbreviations are completed without contextual tokens (see 2.5.8)

2.5.7 ALGORITHM FOR COMPLETING THE ABBREVIATIONS WITH SUPPORT

1. While the abbreviations can be filled, continue the iteration for the current list of token sets per profession (The token sets with only one token are ignored in this sub-step as they do not have contextual information)
2. Get the co-occurrence frequency using the complete dataset with all possible combination of tokens for a given token set.
   - For a set of 3 tokens T1, T2 and T3, the possible combinations are *{T1, T2}, {T2, T3}, {T1, T3}, {{T1, T2}, T3}, {{T1, T3}, T2} and {{T2, T3}, {T1}}*.

3. The unique token sets are clustered based on the length.
4. For each length of the token set
   (a) For each token set of the considered token set length
      i. If not all the tokens are in the list of correctly spelled words and there is any token with a dot
         A. Compare with all other token sets of the same length that have at least one token in the list of correctly spelled words
            - If the two token sets differ only by one token (i.e. all the tokens are the same except one), only one token has a dot (abbreviation) and the other token is in the list of correctly spelled words (full form), the length the full form is greater than the abbreviation and lastly the 2-gram Jaccard similarity of the abbreviation without a dot and the full form (reduced to the length of the abbreviation without dot) (called modified Jaccard score) is greater than the set threshold
              – With the same tokens in both the token sets as a base, store all such pairs of abbreviation and full form tokens along with the modified Jaccard score and the frequency of the full form with the common tokens.
5. For each base of token sets,
   (a) For each abbreviation and its possible full forms, select the full form with the highest modified Jaccard score and then the frequency of the full form with the common tokens to produce a base wise update mapping that stores the token and the token to which it should be updated.
6. Using the base-wise abbreviation full forms the tokens sets containing those abbreviations are updated. While updating the abbreviations, a counter indicating the number of times an abbreviation is replaced for a particular full form is created. To keep in mind, this counter is created over unique token sets rather than the full dataset.
7. If there aren't anymore full form suggestions from *5* then stop the iteration, else continue from *2*.

2.5.8 ALGORITHM FOR COMPLETING THE ABBREVIATIONS WITHOUT SUPPORT

1. For each possible abbreviation
   (a) If the same abbreviation is filled while using the support, then the full form that the abbreviation is replaced with full form that was used most number of times using the support.
   (b) If the abbreviation is not filled while using the support, then the closest abbreviation that was filled using the support is obtained by using the *fuzz.ratio* similarity. If the multiple filled abbreviations have the same similarity as the unfilled one, the filled abbreviation with high frequency is considered and the unfilled abbreviation is filled with the full form that the filled abbreviation is replaced with for the most number of times using the support.

2.5.9 TAGS

At the end of the pipeline, each profession has a set of words (referred to as tags) that represent the profession obtained through the OCR process.

These tags are added to each entry of the dataset.

CHAPTER 3

PIPELINE EXECUTION AND RESULTS

3.1 PIPELINE EXECUTION

The pipeline described in section 2.1 is executed to generate the tags for the data. The pipeline requires setting some hyper-parameters at various steps in the process. The hyper-parameters are

1. **Minimum Token Length**: The minimum number of alphanumeric characters to be present in a token to be considered valid.
    - This value is set to 3.
2. **Minimum Token Frequency**: The minimum frequency for a token to be not considered as a low frequent token.
    - This value is set to 50. The choice of 50 comes from the fact that 56 years of city directories are OCRized.
3. **Minimum Threshold To Replace Broken Word**: The minimum similarity for a word to be considered as a replacement for a set of consecutive low frequent and non-correct tokens.
    - This value is set to 80. The value was chosen based on the quality of the results and to enforce a high threshold to avoid combining meaningful tokens.
4. **Minimum Token Similarity**: The minimum similarity (*fuzz.ratio*) between tokens to be considered as similar.
    - This value is set to 75. The value was chosen based on the quality of the results and some of it is discussed in 3.2.
5. **Minimum Abbreviation Fullform Similarity**: The minimum threshold for the modified Jaccard similarity score between the full form and the abbreviation.
    - This value is set to 50. The choice of 50 comes primarily from the idea of using a 2 gram Jaccard score and the tokens are short.
6. **Minimum Inter Abbreviation Similarity**: The minimum threshold between the filled and unfilled abbreviations to be considered similar.
    - This value is set to 70. The value was chosen based on the quality of the results and it is lower as shorter strings are compared.

3.2 RESULTS

In this section, an quantitative analysis on the tokens before and after processing through the pipeline is presented.

3.2.1 BEFORE PROCESSING

Before processing the tokens through the cleaning and abbreviation filling - after removing words with less than 3 characters, cleaning the special characters, and combining the mistakenly broken words - 87,979 unique tokens represented the professions. Only 18,882 out of 87,979 are in the list of correctly spelled words, which is **21.46%**.

In Figure 3.1, the X-axis is the frequency of a token in the dataset i.e. number of times it has appeared in the 4M lines. The Y-axis is the proportion of tokens that have a given frequency on the X-axis. From this plot, it can be interpreted that more than half of the unique tokens occur only once and the next highest proportion of tokens (around 15%) of them occur only twice. The set of tokens that appear less than 6 times together composes 80% of the tokens and 89% of them are not in the list of correctly spelled words.

At the other end, only a few tokens have very high frequency and they are also present in the list of correctly spelled words. The tokens with low frequency and not in the list of correctly spelled words represent that they are wrongly spelled or they contain foreign words that are not in the french dictionary or they have abbreviated forms of longer words and few have an old spelling or the words is currently not in use.

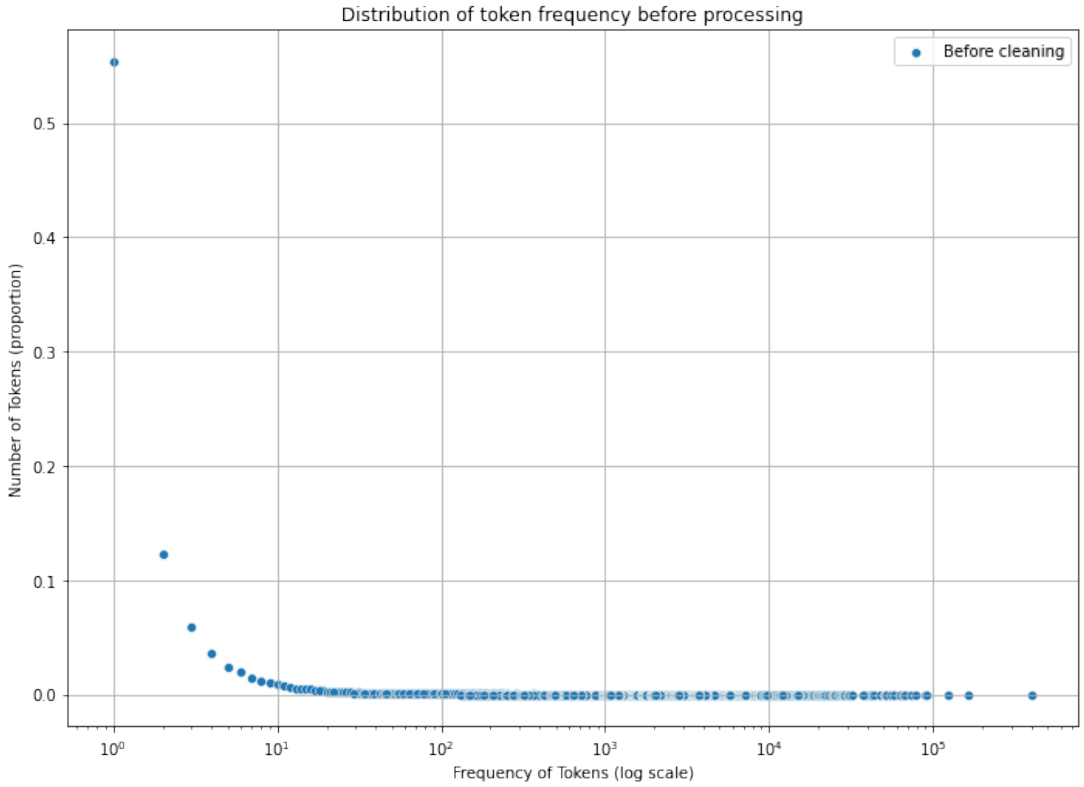

FIGURE 3.1
Distribution of token frequency before processing

3.2.2 AFTER PROCESSING

The hyper-parameter that has the greatest impact on the creation of clean tags is the threshold to determine if two words are similar/close or not. For this project, after qualitative analysis of the data, a 75% similarity threshold seems to be appropriate. When using the similarity threshold of 75, the number of unique tokens reduces to 24,314 from 87,979. 18,882 out of them are in the list of correct words, i.e. **77.66%** of the (new) tokens have the correct spelling. For using the 85% similarity threshold, the number of unique tokens was 43,206 and only 18,882(**43.7%**) out of them are in the list of the correctly spelled words.

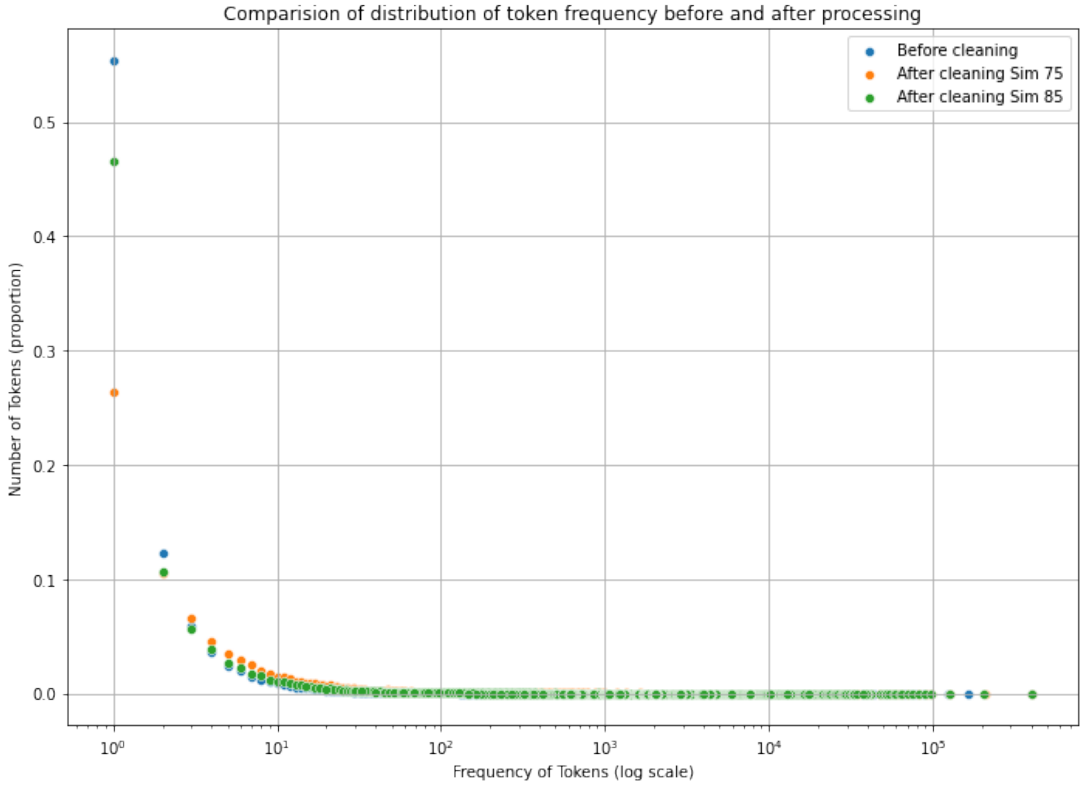

FIGURE 3.2
Comparison of distribution of token frequency before and after processing

Figure 3.2 shows the comparison of the distribution of token frequency before and after processing. The blue dots correspond to the distribution before the cleaning and filling of the abbreviations. Which is same as Figure 3.1. The green dots represent the distribution of tokens after using the similarity score threshold of 75 and orange represent the distribution when using the similarity score threshold of 85.

From Figure 3.2, we see the proportion of the number of tokens having a low frequency is reduced. The reduction when using a 75% similarity score is significant and almost 80% of the tokens after cleaning are present in the list of correctly spelled words.

The major step in the pipeline is the spell correction of words by merging the less frequent and not correctly spelled tokens with high frequent correctly spelled tokens. The average similarity of the tokens merged in this step was 82% when the minimum similarity threshold was 75% and 88% when the minimum similarity threshold was 85%. The distribution of average similarity per iteration in each of the three rounds is shown in Figure 3.3. It can be seen from Figure 3.3 that although the threshold is set to 75%, the average of merges is higher.

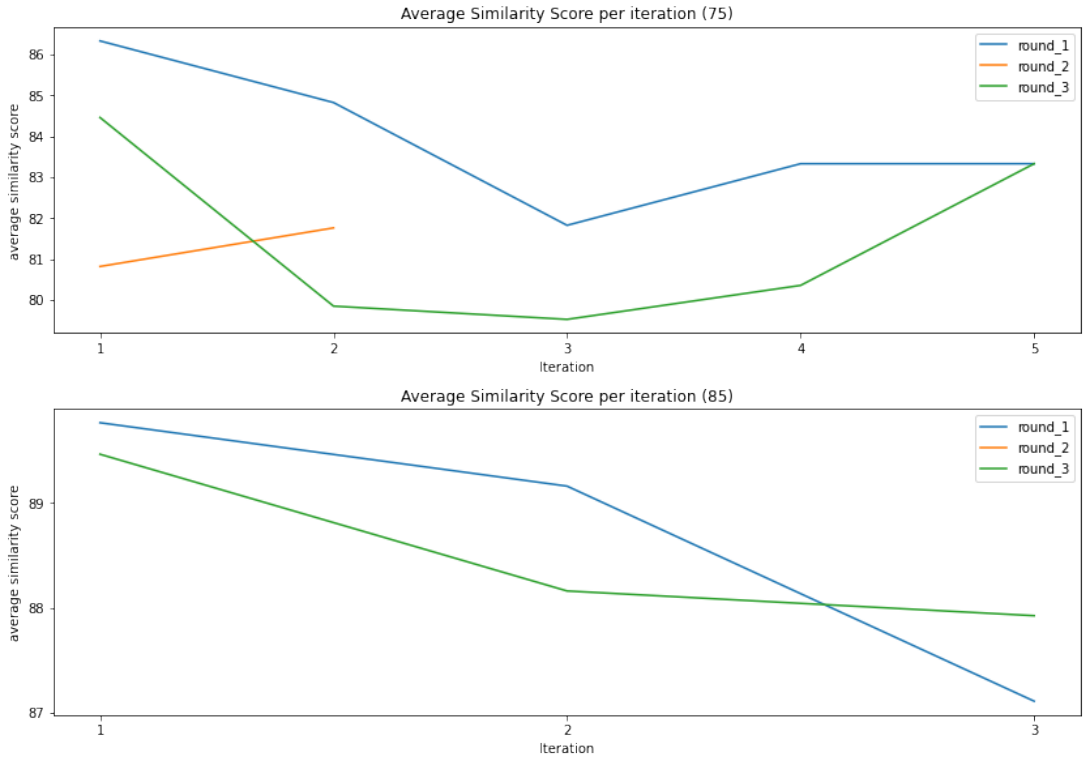

FIGURE 3.3
Distribution of average similarity per iteration per round

3.2.2.1 ERROR RATE

Before performing the spelling correction and filling the full forms of words in the tokens, the average error rate per entry was *12.5%*. The error rate is defined as the fraction of tokens with wrong spellings (after removing short and stop word tokens) per total number of tokens present for a given entry. It can be interpreted as, if a profession has 100 tokens, then 12 or 13 of them are not in the list of correctly spelled tokens. After, processing these tokens through the pipeline using the 75% similarity threshold, the average error rate per entry reduces to *0.6%* making it that only 1 out 100 tokens for a profession string may not be in the list of correctly spelled words. The average error rate was reduced by **96%**.

3.3 DATA FOR RICHELIEU DISTRICT

The above-mentioned pipeline is applied to the dataset extracted from the previous project with all addresses from Paris and generated tags to all possible entries.

As established earlier, this (and the previous project) aimed to study the Richelieu district of Paris. Additionally, the previous project produced a subset of data belonging to the Richelieu district and cleaned the address entries. Keeping in view the clean address and the clean tags for professions, the storage of this subset of the dataset was transformed from CSV files to JSON.

As the spatial coordinates of a place remain constant, the JSON data is produced pivoted around the addresses. The primary key of the JSON file is the name of the street (*rue*) and the details of the street are stored in a list. Each list is again a JSON dictionary with the number (*numéro*) as the key and the geographical coordinates and the people information as values. The information about the people at the given address is stored based on the year. Each year has an entry for the person. In each entry, the name,

profession (as obtained from the OCR process), the tags (as a list), and the link to the page on the scanned document are stored. Apart from these, empty fields are created to store any multimedia information that would be acquired for the person/business.

The format of the JSON file is:

```
{
    "<rue name>": {
        "details": [
            {
                "<rue number>": {
                    "location": {
                        "geo coordinates": {
                            "latitude": "<>",
                            "longitude": "<>"
                        }
                    },
                    "people": {
                        "year": [
                            {
                                "Person Name": "<>",
                                "Profession": "<>",
                                "Tags": [<>],
                                "Related Data": {
                                    "Multimedia URLs": [<>],
                                    "Other URLs": [<>]
                                },
                                "gallica link": "<>"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

CHAPTER 4

DISCUSSION

This work of cleaning the profession strings in the data extracted about the Richelieu district (or Paris) in general provides a closure to the first phase of the project [1]. While the previous project extracted the data from the scanned directories, this work has cleaned the mistakes and provided a searchable and useable dataset (especially the professions or activities of the people). With the result of this internship, the data can be searched based on the street name, house number, and/or profession of the people. Although it is desirable to search the people based on their name, it is not trouble-free to clean the names. The spellings for names of the people can vary and involve individual preference to write it. Hence, it could be naive to clean them.

On the highest level, this work has used the idea of single word tags for strings instead of correcting the complete strings. The concept of creating tags was driven by two factors. First, the complete sentences use connecting words to make the text legible and thus such words are not important. Secondly, the users interested in the data would use keywords to search. The keywords matching can be performed by attributing tags to each entry of the dataset.

Another core concept used in designing this pipeline was the use of words within the dataset to correct the mistakes. Only to identify these mistaken words, some external data is used. This is essential to not corrupt the dataset with unrelated modern words, which is of utmost importance in this case of dealing with 19^{th} century text.

While the pipeline has been able to reduce the average error rate by 95% or to make 77% of the tokens to be present in the dictionary, there are some limitations. Although most of the limitations could correspond back to the design of the pipeline, most of them were not dealt with due to the time constraint.

The most influential hyper-parameter in the pipeline is the threshold to check if two strings are similar. Although two values are compared in Section 3.2, a comprehensive study on pitfalls and advantages of various similarity thresholds was not performed. In addition to the plots provided in 3.2, other comparisons are available in the plots Jupyter notebook accompanying this report.

The second limitation is about the words joined without any special characters or space. Although a probabilistic model was developed to break such words, it was not included in the pipeline as it was not tested.

Lastly, only those abbreviations that were similar to the most frequent abbreviations were transformed into complete words. There are some abbreviations left that were not similar or a complete word was never used for them.

The algorithm and the steps in the individual components of the pipeline developed in this work are

generic and can be extended to post-processing correction of other OCRizied documents. To facilitate the researchers to use the dataset and the algorithm, both of them, including the python code is published in a public repository.

4.1 FUTURE WORK

With the usage of the tags generated for each entry of the dataset, multiple avenues of research are possible. In this section, two such ideas are presented.

4.1.1 SEMANTIC CLUSTERING OF PROFESSIONS

In the dataset, different keywords are used to represent similar professions, and searching for one word might not show up all related or close professions. In addition, the present-day classification of professions is different from the 19^{th} century. It is also difficult for the users to know upfront which keywords to use to search the data and they cannot search for people for the concept of the profession such as fashion or waterworks etc.

An idea in this direction is to create a bridge between the current idea of a profession and the data extracted from the directory. Semantic clustering of the tags can be created and manually annotating the clusters could provide themes of professions.

4.1.2 SPATIAL VISUALIZATION

To understand the growth both physically and socially, economically the data set can be visualized on a map of the time. Such visualization would aid in tracing the growth of the district/city. A text-based search platform can also be added to that visualization.

BIBLIOGRAPHY

[1] Isabella Di Lenardo. *Richelieu. histoire du quartier*. URL: https://quartier-richelieu.fr/ (visited on 5th Sept. 2021).

[2] Isabella di Lenardo et al. *Repopulating Paris: massive extraction of 4 Million addresses from city directories between 1839 and 1922*. Version V2. 2019. DOI: 10.34894/MNF5VQ. URL: https://doi.org/10.34894/MNF5VQ.

[3] Wikipedia. *Levenshtein distance*. Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Levenshtein_distance (visited on 5th Nov. 2021).

[4] *Rapid fuzzy string matching in Python and C++ using the Levenshtein Distance*. URL: https://pypi.org/project/rapidfuzz/ (visited on 28th Feb. 2022).

[5] *Morphalou 3*. URL: https://repository.ortolang.fr/api/content/morphalou/2/LISEZ_MOI.html#idp37913792 (visited on 28th Feb. 2022).

[6] *prolex-unitex*. URL: https://tln.lifat.univ-tours.fr/version-francaise/ressources/prolex-unitex (visited on 28th Feb. 2022).

[7] *Dénominations des emprises des voies actuelles*. URL: https://opendata.paris.fr/explore/dataset/denominations-emprises-voies-actuelles/table/?disjunctive.siecle&disjunctive.statut&disjunctive.typvoie&disjunctive.arrdt&disjunctive.quartier&disjunctive.feuille&sort=typo_min (visited on 28th Feb. 2022).

[8] Steven Bird, Ewan Klein and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[9] *Unidecode*. URL: https://pypi.org/project/Unidecode/ (visited on 28th Feb. 2022).