LINGI2263 : COMPUTATIONAL LINGUISTICS

Group 2 : Project 1

Authors:
Crochelet Martin (2236-10-00)
Baugnies Benjamin (6020-10-00)

Professor:
Pierre Dupont
Cédric Fairon

2013 - 2014

1 Program Architecture

For this project, we chose a Python3 implementations. We provided two Python (.py) files: `exporter.py` and `mt_extractor.py`. The first is used to output the results to the tab-separated values or to an html file for better readability. The second file, `mt_extractor.py` will extract the data of a given file and output it to the desired file and format. We will now describe the architecture of this second file in more detail.

We first define all the necessary regular expressions using the python `re` library. Each attribute we wish to extract will have its own regular expression(s) which attempt to define how the value is expressed as generally as possible, but without accepting too many false values.

In the following part of the program, we define the utility functions and classes we will need. The first one is the `getNumber` function. This function is used to convert the many different notations for numbers into floating point values. We also define a `TranscriptGen` class. This class will provide us with the ability to extract individual transcripts from the corpus, therefore allowing us to deal with one patient at a time. This class's `extract` function will provide an iterable over all the transcripts by using the Python `yield` statement.

The third part of the program contains the methods for extracting each type of data we need. These methods search a given transcript for the appropriate regular expressions and isolate the value and correct unit. In these methods, we will sometimes apply regexes in a specific order to create a priority, as we will show later on. The methods will also apply certain 'filter' rules in order to discard unusual values.

Finally, we end the program applying the methods on each transcript of the file and outputting the obtained values.

2 Usage

The program can be used with two modes:

- the first mode is the raw mode that outputs the results as a "\t" tab-separated table into the output file specified the command is :
  `python3 extract.py` *`inputFile outputFile`*
- the second mode is a more readable mode that will output the results as a html table that can be sorted by clicking onto the column names. The generated file have the name given by the user as outputFile and has the extension html. The corresponding command is
  `python3 extract.py --fine` *`inputFile outputFile`*

3 Data extraction

In this section, we will go over how certain attributes are extracted from the corpus. Specifically, we will concentrate on the age, weight, and temperature. However, it is worth noting that other attributes are extremely similar (weight and height, temperature and pulse, for example).

Age

In order to extract the age of patients, we first had to go through the data to isolate the different ways in which this value is expressed. In almost every case (those that do not are covered in the Improvements section), the age contains at least one number. The regex used to identify an age therefore starts with a number. We used two string to describe such numbers: `float_string` for decimal numbers and `fraction` for decimals. The second part of the regex lists the most common ways we found for expressing an age in the corpus.

All the expressions used in this case are quite explicit; it is unlikely that "X-years-old" refers to anything other than an age. Additionally, they are also equal priority-wise. That is, "X y/o" is not more likely to describe an age than "X years of age". For this reason, we only use one regex which will match to any of the expressions by using the or ('—') operator.

The next step was to choose which match within the given transcript would most likely be the patient's current age. Indeed, many transcripts contain several age values. These range from the ages of relatives to younger ages at which a procedure or injury took place in the patient's history. By exploring the data, we found the first mention of an age most often corresponds to the desired value. We therefore chose to only exploit the first match to our regex.

The last step is to calculate the decimal value of the age in years. Since many units are used (years, months, weeks, and days), we need to match the age expressions to the correct unit and then convert the obtained value. The decimal number is obtained via the `getNumber` function if order to account for various formats (e.g: 2-1/2-years-old = 2.5).

Height

The regex for identifying a Height is built in two parts. First, `length` is a string that describes an isolated length value. To unambiguously mention a length, it is necessary to have a value and an appropriate unit. In this case, the main units used were feet and inches, which occasional uses of centimetres. We found no case of meters being used for the height of a patient. Additionally, when referring to the height of a person, centimetres were only used in the "X cm" format, not fully typed-out. In the case of feet and inches, several different notations are used (such a X'Y" or "X feet Y inches").

The second part are the regular expressions themselves. Obviously, an isolated length expression can represent the height of a patient, but it remains very ambiguous. Indeed, it could also refer to the size of a tumour or the length of a needle, for example. To help differentiate, we create a second, more complex regex. In this case, we look for explicit mentions to height near the length expression.

In the height extraction method, we apply these two regexes in a specific order: the more constraining one first. This creates a priority between expressions. While a simple "6 feet" can represent a height, "167 cm tall" almost certainly does. In this way, we avoid relying on the more ambiguous expressions if we can find a better match. Within a same priority, we favour early matches as the height of the patient is more likely to be mentioned earlier than previous heights (in a growth history for example).

Since there is still a fair chance of getting an incorrect value (for example, if the patient's height is never mentioned but a needle's length is), we want to try to remove impossible values. This is complicated, however, because there is a large range of values (from newborns to adults). We chose to apply a rather simple rule. We first check the patient's age. If it over 18 years, we impose a minimum of 50 cm (going through the file showed few intermediate values, there were mostly either heights or very small lengths). If the patient's age is under 18 or if is unavailable, we simply accept any value. Note however that values under centimetres/inches (namely, millimetres) are not considered by our regex.

The last step is computing the decimal value of the height in meters. To do this, we check the length expression for units and apply the proper conversions.

Temperature

The regex that searches for the temperature is a little bit more complex and is composed of three parts:

- The first one searches for expressions of the pattern *temperature [zero to five words] number* that can be interrupted at any place by *:* or a "enter" ($\backslash n$). This allow us to be generic and handle cases from *temperature is 98.3* to *temperature yesterday was 98.3*
- The second part takes into account the abbreviations for the temperature that are often used in the transcript such as *T 98*, *TEMP 80*, etc. For this regex, it is important to match only when *T* is a single word and not the end of another word ; for this, we use the \b functionality of the python3 regexes that matches only at the beginning and end of a word
- The last one analysis pattern such as *number [zero to three words] temperature* but is a little bit more complicated: the regex only matches when the text corresponds to the pattern above but isn't followed by a pattern that would have been matched by the two other parts. This is important because the pattern *number temperature* can be easily found in a part such as VITAL SIGNS: where the characteristics are written one after the other (e.g. *RR: 20. temperature 98.*). We do not want to match the "20. temperature" pattern in the previous example while we have the pattern "temperature 98".

Finally, once we have matched the temperature we extract the corresponding number and convert it to float. We then convert the number in Fahrenheit if it is superior than 45 degrees Celsius.

4 Possible Improvements

An general improvement would be to look for context further away from the specific strings we identified. By this, we mean that our current implementation only looks for patterns spanning a few characters. This allows us to get context from the units, since they are usually close to the value. However, the corpus' structure is such that some context can be derived from the paragraph header. For example, paragraphs starting with "PHYSICAL EXAMINATION:", "Hx:", or similar keywords are more likely to contain information about the patient. In a similar way, paragraphs where words like "patient" occured are more likely relevant, whereas paragraphs with "father" or "sister" are likely to contain irrelevant information (unless we were trying to extract a family history, of course).

There are also improvements we could have brought to specific methods. For identifying the gender, for example, we could have counted the occurrence of gender-specific pronouns and determined the sex based on which occurred most. There is also a problem with the way we eliminate unlikely values. In the weight for example, if a value is discarded, we return 'NA' whereas we could go back and see if there is another match which might be correct. Finally, we could also use the BMI to cross-check the values of weight and height. If the three values to not satisfy the relation $BMI = \frac{mass}{height^2}$, we know there is a problem with at least one of the values.

More computer oriented improvements are also possible such as optimizing the exporter in order to write the result for each transcript directly instead of storing everything in memory and write once the analysis is finished. This would allow the program to handle heavier files while actually we are limited by the size of the RAM available. Another consideration is the following: the python3 regex implementation will parse the entire transcript every time a method search is invoked onto it. This lead us to believe that it would be more efficient to compile all the regexes into one big regex (disjunction of single regexes) then parse the transcript and finally parse the results in order to avoid the overheat of parsing the transcript multiple times.