Computational Linguistics

Project 1 - Recipe Interpreter

Student: Lucas Moscovicz

Date: 14/10/2014

1. Introduction

The main objective of this project was to design and develop, using linguistic approaches, a program capable of extracting basic information from a given corpus (internet cooking recipes) by using regular expressions and context awareness.

2. Main Architecture

The solution provided was developed using Java 7.0. The runnable file is a .jar and it should be run using the following command:

```
> java -jar RecipeInterpreter.jar <input_file> <output_file>
```

The given corpus was written in HTML code, which provided extra information besides the actual written text for the recipe. One first approach was to use that information and, combined with the information extracted from the text itself, it was possible to build an interpreter that provided a satisfactory answer to the problem.

The approach used for the program was a Multi-Layer filtering scheme, where each layer consists on a series of filters based on regular expressions which are applied to the whole text in different ways.

Each time a set of filters is applied, the parts of the text that are left are those who could get through **at least** one of the filters.

2.1 Starting filters over the whole text

The first set of filters that is applied to the text are regular expressions that work over a big set of text in order to extract smaller parts of it based on singular data points (numbers, measures, etc). With that, it is possible to split the text and keep only the parts that are of interest when looking for certain kinds of data.

<u>Example:</u> `([0-9]+\\/[1-9]{1}[0-9]*|([0-9]+\\.[0-9]*)|([0-9]*\\.[0-9]+)|([0-9]+))(\\s[a-z,.;:]+){0,5}`
This regular expression looks for a number in different formats, followed by 5 more words at most.

2.2 Filters over lists of strings

Once the first set of filters is applied, another sets of filters are applied but now over the list of strings resulting from applying the first set. This filters now work looking for different keywords or patterns to be able to discard every part of the text that is not of interest.

Example: .\\b(part(s)?|slice(s)?|dash(es)?)\\b.**
This regular expression looks for some specific keywords which are usually found in certain recipes

2.3 Exclusive filters

Another set of filters that is applied is the exclusive set of filters, this set is applied over the list of each relevant part of the text that went through the other filters and does not allow to pass any of them that matches an exclusive filter.

Example: .\\b(jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may| ... |dec(ember)?)\\b.**
This regular expression looks for months on the text since many of the recipes have comments and that is an easy way to discard them

3. Solution created

Many resources were used to find the best possible solution to this problem. One first approach was to, using the HTML information, find all elements contained in lists (<li>) and put them through the filters which usually returned the ingredients list.

As for the filters, there are two main kinds of filters which were designed.

1. Keyword filters: They look for certain keywords in the text and, should they find them, they let the text pass onto the next filter layer.
2. Structure filters: They look for a certain kind of structure in the text and, should they find it, they let the text pass onto the next filter layer.

Both kinds of filters are combined in layers with filters of their same kind.

For the time and servings parser, keyword filters were mainly used whereas in order to parse the ingredients, both kinds of filters were necessary to get better results.

After applying all the filters over both the HTML information and the text itself, the solution is merged to avoid repeated elements. This is easily done using an own class for each part of the solution (Ingredient, Time, Servings), and correctly implementing the method *equals* to compare when two elements are similar enough to be considered the same.

4. Considerations made

Along the project some considerations had to be made in order to improve the performance and the completeness/correctness of the program.

- Jsoup (Java Framework) was used to parse HTML and use it's tags to provide extra information

- All the text is converted to lowercase before processing

- The US Metric standard units were used for conversions

5. Conclussion

For this project it was possible to extract the requested data in a pure linguistic way. However, some information is probably missing since it's impossible to consider every single scenario. For example, Ingredients that don't have either a number or a measurement attached, they are very hard to find. One possible way of doing this is considering for every word the context it appears in (whether it contains many ingredients surrounding it or nothing of relevance may mean it's an ingredient or not) but still it's a very variable problem.

For this kind of information another kind of approach than the linguistic one would probably be more useful, either automatic learning from the corpus or other approaches that are more computational than linguistic.