NLP Project - Query Parsing

Introduction to Natural Language Processing

Universitat Politècnica de Catalunya

January 18, 2016

Authors:
Kazancli, Erol
Kazimi, Mohammad Bashir
Amin, Khizer

Contents

1 Description

The task at hand is parsing an xml file containing possible queries in search of a location, a query object, geographical relation between the object and the location and query type, and returning this extracted information as an output xml file. The aim is to to obtain as much accuracy as possible as to the locations and related information contained in the query.

2 Our Approach

We started by analyzing the general structures of the queries to figure out the general characteristics of a query that is categorized as a location in the training data. We then identified the general structure and the grammars detailed in the section below. We followed these grammars to extract the necessary information. We mainly used regular expressions to be able to represent these grammars, however we also needed to use some heuristics and extra coding.

To parse, our approach was first to retrieve the locations using our dbpedia search method isLocation. This method returns all the location possibilities in the query. As a heuristic, we chose the first location returned by the method, which is the most comprehensive one. This approach worked most of the time, however if there are several locations in the query this approach sometimes failed. To resolve this problem we checked the series of words coming after the detected location in the query. If this series contained a word like 'in', this meant that our first choice might be wrong. Therefore we searched for another location that is situated after this 'in' word, because it is highly probable that the main location comes after this 'in'-like word. Once the location is found, the country it belongs to is also searched in dbpedia using the method getCountry.

After we found the most probable location candidate, we started searching for the 'what' in the query. If the location is situated at the beginning of the query, we chose the succeeding series of words as 'what'. If the location is at the end, we looked for the 'what' among the preceding words. However, these words may also contain the geo-relation information. Therefore before extracting the 'what' information, we searched for the geo-relation, checking the presence of certain keywords like in, on, near, south.... After we found the specific keyword if it exists at all, this word becomes our geo-relation information and the rest of the words form the 'what'. The problem is when the location is in the middle of the query. In this case, we add the words coming after this location to the location itself, thinking that it is highly likely that they are a part of 'where' information. And the preceding words form our 'what' in this case.

One problem we encountered was, sometimes even if there exists a location in the query, this location is not found by the dbpedia search. In such cases, it is very difficult to find the location itself automatically. There might be clues like certain words such as 'in', 'to', etc. but these words have also other functions apart from indicating a place. Therefore counting on these words alone may be misleading. However there are some words like 'South West' which are highly likely to indicate a location. Therefore when we found one of these words, we assumed the query contained a location, which is the series of words starting by this keyword, otherwise we chose to miss the possible locations which were not detected by dbpedia, and this led to false negatives from time to time. We could have adjusted our code to the specific examples in the training data but this would lead to overfitting and a worse test data performance.

To find the 'what' type, we checked for specific keywords in the 'what' information we found. Words like 'hotel', 'repair', 'food', 'hospital' etc., which may hint at an address search indicated a high possibility for a yellow page search. Words like 'info', 'trip', 'discount', 'flight', etc. indicated information search. If the 'what' info we found is an empty string, then the 'what-type' is most probably a 'map'. There may also be some proper nouns indicating yellowpages, like specific car brands, however, it is very difficult to capture all these specific words and they may also be misleading, since they can be used for other purposes as well. Therefore, we chose to ignore them at the cost of misclassifying. If the what type is not found using our set of keywords, we leave it blank. We could have guessed it as 'Yellow Page' because it seems from the training data that it is the most common query type however this would be misleading because we do not have enough information implying this in general. Besides, in case of false positives this would damage the reliability of the parser.

To find the coordinates, we made another dbpedia search. This search was successful most of the times comparing with the training data. However, there might be slight differences which is not significant but may affect the scoring even if the numbers are very close. There is no way of getting perfect numbers in this case therefore we ignored such a possibility. We expect this should be handled by the scorer.

3 Grammars Used

To find 'where', 'what', 'geo-relation' information we used the following CFG grammar:

```

Query -> what relation where
Query -> what where
Query -> where what
where -> Location ',' Country
where -> Location NP ',' Country
Location -> dpedia_locations
Country -> All possible countries
what -> NP
NP -> NP N
NP -> NP PP
N -> all possible nouns
relation -> PP
PP -> in |on | at |of | near | near | to | to | south_in | north_of | west_in | ...
    east_at | southwest_of | northwest_of
```

To find what-type information we used the following CFG grammar:

```

what -> NP
what -> None
NP -> NP yellowpage_keyword
NP -> NP info_keyword
NP -> NP N
NP -> PP N
PP -> PP Prep
Prep -> all possible prepositions
N -> all possible nouns
yellowpage_keyword -> hotel | bookshop | auto | rent | cleaner | hostel | food | ...
    hospital | park | doctor | law | police | shop | cinema | house | buy | inn ...
    | mall | restaurant | car | dealer | apartment | school | plumber | training ...
    | cafe | gym | airport | store | theater | center
info_keyword -> info(rmation) | news | festival | trip | discount | income | ...
    flight | to | blog | questionnaire | tour | weather | thing | tip
```

4 Our Algorithm

We parsed the XML file, using one of the solutions developed by our fellow students and posted by the professor. We needed to make some adjustments to the code. To find the possible locations we used our own algorithm, namely isLocation(), which we developed in the first phase of the project. This method returns all possible locations in the string starting from the most comprehensive one. To find the 'what', 'where' and 'geo-relation' information we developed a method named findWhereWhatRelation(query), which takes a query and using the heuristics and grammars explained above and extracts the 'where', 'what' and 'geo-relation' information. To find the ‘what-type’ information we developed a method named findWhatType(what), which takes a series of words and returns a possible answer as to the type of this search using regular expressions. getLattLong(term) is a function we use to find the lattitudes. We also found the country to which the location belongs using a new dbpedia method, getCountry.

A general outlook of the code is as follows:

```

XMLItems = xml.read()

for x in XMLItems:
    q = x.Query
    found, what, relation, where, country = dbpedia.findWhereWhatRelation(q)
    if found:
        x.Local = 'YES'
        x.What = what
        x.Where = where + ', ' + country
        x.Geo_relation = relation
        x.What_type = dbpedia.findWhatType(what)
        x.LattLong = dbpedia.getLattLong(where)
    else:
        x.Local = 'NO'

xml.write(XMLItems, output.xml')
```

5 Evaluation and Results

Using the training set we tried to extract grammar rules as accurate as possible by using dbpedia, identifying the relevant keywords and the probable positions of locations in the query. When we check the output file using our algorithm for the training data, we see quite good results. We were successful in parsing many queries especially when they contained a location detected by dbpedia and when the queries are regular in structure. We saw that we failed in some cases where even though the query contains a location, it cannot be detected by dbpedia. We also failed in some situations when the query contained several possible locations. For whattype information, since we used the keywords contained in the training set, we were successful in most cases with the training set. We scored our outputfile for training set by using one of the scorers provided and got 0.33 for precision and recall. This did not look a good performance, however when we checked the scorer we realized it was case sensitive and did not take into account some of the scoring rules. We adapted it to be case-insensitive and we got a score of 0.6 for the training case. This is expected because we trained our algorithm using the training set.

For the test set, in addition to the ones explained for the training set we came across some situations quite new to us and not taken into account for the training. Therefore we failed in some of these situations. To give an example, to find the whattype information, we realized that the keyword set we used using the training set was not enough. Whattype information was where we failed the most. For the where and what information, our heuristics gave good results most of the time. If the query contained several locations or several keywords indicating locations, our algorithm sometimes failed. But checking with the naked eye, we saw we performed quite well finding the correct where and what information. For the geo-relation information we sometimes failed especially if the query contained several keywords indicating a geo-relation. Using the scorer we used before we got 0.32 for the precision and the recall values. However, when we made it case-insensitive, we got a better value, which is 0.39. With a more accurate scorer, we expect we would have better results.

6 Possible Improvements

We could have used other resources when dbpedia search failed to find the location contained in the query, since our dbpedia search sometimes failed to find the location. We could also check for spelling errors and correct them before parsing, which would help with better results. In cases of multiple locations in the query, we could have improved our heuristics to choose the correct one, using the structural or semantic information. In the case of what-type information we should have included more keywords or we could have used other information(eg.structural), or maybe even use an outside source. For the geo-relation information we could prioritize the keywords or take better advantage of the structural information. Since we tested with only two datasets, we do not have enough variety to learn enough cases, therefore we could have searched other sources for possible location queries. We did not do error checks for possible invalid xml structures assuming the given XML file is in a correct format. It would be better if we checked the xml structure first. One last point, the existing scorers checked if the whole query parsing is correct. They could be adapted so that they calculate partial accuracies as well, because it is very hard and unnecessary to get a 100 percent match to the last detail.

7 How to Execute the Code

Extract the .zip file in a directory. The directory should contain the following files.

XMLInterpreter.py XMLParse.py XMLItem.py dbpedia.py scorer.py and the xml input and output files

To run the code:

XMLInterpreter.Interpret("GC_Test_Not_Solved_100.xml")

This will create a file named fileName_Output.xml in the same directory as inputfile.

To run our adapted scorer:

scorer.score("GC_Test_Not_Solved_100_Output.xml", "GC_Test_solved_100.xml")

The difference of this scorer from the original one is that it is case-insensitive. Please take care that your golden set does not have blank values, instead it should have 'None' as in the test golden xml file.

References

[1] Query Parsing Task for GeoCLEF2007 Report, Zhisheng Li1, Chong Wang2, Xing Xie2, Wei-Ying Ma2