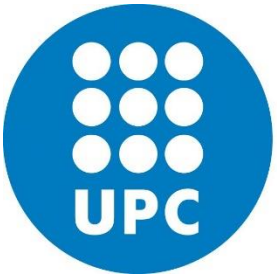


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

GEOCLEF-2007 Challenge

Introduction to Natural Language Processing

Students:
Deividas Skiparis
Ilmira Terpugova
Andrei Polzounov
Andrei Mihai

Table of Contents

Introduction

The problem of this project was based on GEOCLEF-2007 challenge. The task was to create a set of grammars that will be the basis of Geographical Information Retrieval (GIR) system. GIR systems usually deal with unstructured text and aim at solving a query that contains geographical information. The main issue is to separate the thematic and geographical part and work with both of them.
In our case the data is the set of relatively short queries that could or not contain some location. The queries that have location part are composed of four components <WHAT>, <WHAT-TYPE>, <GEO-RELATION> and <WHERE>. A set of 100 training and 100 testing queries have been provided. The goal was to build a system that will extract or resolve the aforementioned parts from queries. This report outlines the description of our system, rules used and results obtained on the test dataset "GC_Test_Golden_100.xml"

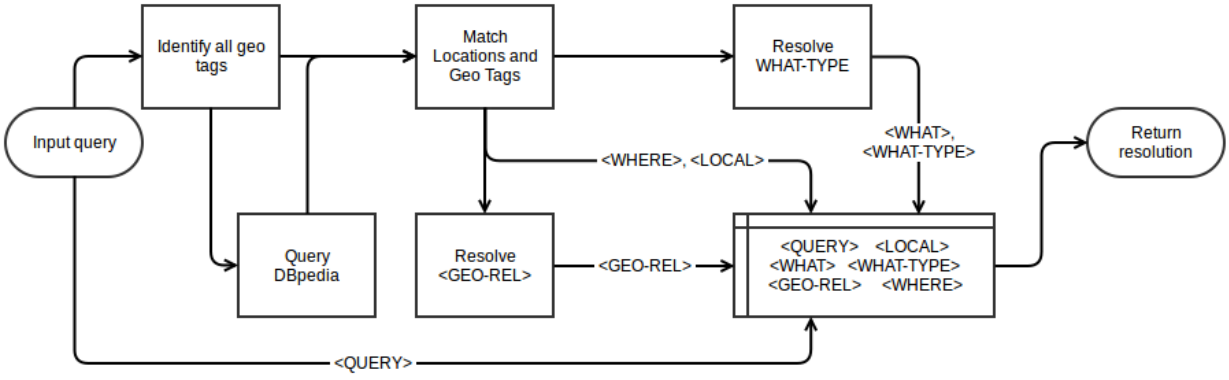
Acknowledgements

For writing XML files we have used XML writer provided by Group 17, Jordi Ganzer and Nil Sanz.
For XML evaluation we have used the scorer provided by Daniel Duato Catalan, Luis Fabregues de los Santos and Javier Selva Castello.

The system: Description

Overview

High-level flowchart of our query parser



System description

Our system to parse geographically oriented queries consists of 5 stages:

1. Geo tag search
2. Location search
3. Geo tag and Location merge
4. <GEO-RELATION> resolution
5. <WHAT TYPE> resolution

We have decided not to utilize Freeling or any other morphological analyzers, Part Of Speech (POS) taggers or Named Entity Recognizers (NER). The reason for this is we discovered that they give unreliable results if queries are all lowercase/uppercase or each word is capitalized. We have tested Freeling as well as Stanford NER and we found they both give wrong results in such cases, therefore we decided to build our system without these methods.
We do not use any punctuations or letter capitalization as a clue for locations or named entities. For this reason each query before processing is made lowercase and any punctuation like commas, dots, semi-colons, colons are removed with the exception of dash ('-') as it could be part of a named entity.

Geo tag search

In this part of the system, the program finds possible candidates for geo tags ('in the south of', 'in', etc.). We have used regular expressions in several stages to find appropriate tags. Firstly, the system looks for longest geo tags, such as 'in and around', 'in the south west of', etc. The short geo tags such as 'in' or 'of' are only matched in the absence of longs geo tags.
All the matched geo tags are then removed from the query and the sentence is divided into several smaller parts. For example, performing geo tags search for a query *'accommodation near fort william'* returns geotag = 'near' and remaining sentence divided into ['accommodation', 'fort william']. If no geo tags are found, sentence is returned as is.

If no locations are found in the sentence in further steps, the geo tag search is repeated, omitting previously found tags. For example *'imaging in west covina'* firstly matches 'in west' as a geo tag. Since neither 'imaging' nor 'covina' are locations, the geo tag search is repeated. The second time, the matched geo tag is *'in'*, and consequently 'west covina' is found as a location.

Location search

The returned sentence from geo tag search is then queried in Dbpedia to determine if is contains a location. We have used our own script from part 1 of this project, and provided additional functionality to the code to return 1) the probability P of a string being a location and 2) full name of location (including state and country if applicable). Dbpedia accessor functionality is detailed in Appendices. A minimum threshold of P=50% has been set. Only strings with more than 50% certainty are treated as locations.

The system uses the sentence returned from Geo Tag matching to build all possible combinations of up to 4 consecutive words. If sentence was split, then each split is treated individually. All these combinations are passed to Dbpedia one-by-one, starting with the ones having most words. If a location is found, it is removed from the original sentence and the whole procedure is repeated until no further removals are made. For example, ['accommodation', 'fort william'] is built to 'accommodation', 'fort william', 'fort' and 'william'. Using Dbpedia accessor, 'fort william' is used as a first choice and matched as a location. It is then removed from sentence and remaining ['accommodation'] is then queried. In total - Dbpedia accessor is used twice for this query.

Geo tag and Location merge

In most sentences, there are more geo tags than there are locations, however not all geo tags correspond to a location. Therefore in this stage, the system matches locations with their appropriate geo tags. Only tags, which correspond to a location, are then treated as actual geo tags in further steps.

<GEO-RELATION> resolution

Once the actual geo tags have been identified, they are classed into categories given in the task specification. Since there is always only one location per query, it results into a single geo tag class. The logic between categorization is the following:

a) If the geo tag is a single word, this word is the name of the category in capitals. Ex. 'IN'
b) If there is a number followed by distance measure, the category is 'DISTANCE'
c) For directional geo tags ('south of', 'to the north west of'), the category is the direction ('south', 'north west') followed by leading preposition (or trailing, if leading is absent). Ex. 'SOUTH_OF', 'NORTH_WEST_TO'.
d) If words 'in' and 'around' are present, the category is 'IN_NEAR'
e) If combinations like 'next to' or 'near to' are present, the category is 'NEAR'
f) If geo tag is 'within', the category is simple 'IN'
g) Otherwise, it is 'UNDEFINED'

<WHAT TYPE> resolution

Once the location and its corresponding tag have been removed from the sentence, the remaining string is treated as a 'WHAT' tag. In order to correctly identify the type of this tag ('Map', 'Yellow page', 'Information') the system uses the following grammar:

1. If WHAT is empty, this means the whole string is a location, therefore 'Map'
2. Check if the string is a possible name of the street. We have used a list of all possible english street types and abbreviations[1], such as STREET; STRT; ST; STR; STREETS; WY; WAY; WAYS and many others. Check *streetAbbrs.txt* text file for the full list. If any of these abbreviations are matched at the end of the 'what' tag, the whole 'what' tag is treated as an address and 'WHERE' is the whole original query. Therefore, <WHAT-TYPE> = 'Map'
3. If it is not an address, the system checks if the 'what' tag is a Yellow Pages query. To check this, we have used a rudimentary list of categories, for which one would consult yellow pages. This includes 'Housing' (flats, houses, apartments), 'Service' (rent, buy, sell), 'Professions'[2] (plumbers, repairers, lawyers) and 'Businesses and Institutions' (banks, police, schools). The full list can be found in *categoriesYP.txt*. If any of these categories are matched in the 'what' tag, then we treat this as a 'Yellow Page' query.
4. If the category is not 'Map' and not 'Yellow page', it is considered as 'Information'.

XML evaluation

As mentioned in Acknowledgements section, we have used XML scorer provided by another group. However, we have slightly modified the code to give a fairer evaluation:

- For <WHAT> and <WHERE> comparison, we have lowered the case for golden and test queries.
- We have removed any unnecessary whitespace (double spaces, leading/trailing spaces, tabs, etc.) from golden and test queries.

This way, the scorer is not sensitive to string capitalization or whitespace.

1 http://pe.usps.gov/text/pub28/28apc_002.htm
2 http://www.occupationsguide.cz/en/abecedni/abecedni.htm

Results

The total results obtained by the parser is

Precision	0.6
Recall	0.7792
F1-Score	0.6780

We have also analyzed the results in terms of correctly classifying whether a query represents a location. The confusion matrix with metrics is below:

Query represents Location?		PREDICTED		
		True	False	Precision = 65/68 = 0.956
ACTUAL	True	65	12	Recall = 65/77=0.844
	False	3	20	Accuracy = (65+20)/100=0.850

The overall results are quite high, both precision Recall and Accuracy are ~85%, while Precision is nearly 96%. This compliments our results obtained in the first part of the project, where Dbpedia accessor was able to perform with over 90% accuracy.

Moreover, the <GEO-RELATION> identification was almost perfect. There were 17 location queries with geo relations tag. Our code has made 1 mistake in identifying it, because it failed to find a location in the string. <GEO-RELATION> accuracy = 16/17 = 0.9411

We have also analyzed how good was the code in predicting one of the WHAT type labels (Map, Information of Yellow Page). The confusion matrices are below for all three types:

Yellow pages?		PREDICTED		
		YP=True	YP=False	Precision = 21/23 = 0.913
ACTUAL	YP=True	21	20	Recall = 21/41=0.512
	YP=False	2	57	Accuracy = (21+57)/100=0.780

Information?		PREDICTED		
		Info=True	Info=False	Precision = 9/34 = 0.265
ACTUAL	Info=True	9	20	Recall = 9/29=0.310
	Info=False	25	66	Accuracy = (9+66)/100=0.75

Map?		PREDICTED		
		Map=True	Map=False	Precision = 11/11 = 1
ACTUAL	Map=True	11	16	Recall = 11/27=0.407
	Map=False	0	73	Accuracy = (11+73)/100=0.84

Conclusions

After careful comparison of misclassified queries with golden set, we mostly failed in the following situations:

1. Typos. If there are spelling errors in the sentence, it is most likely going to fail. For example 'accomodation south africa' or 'bismark state college'
2. Foreign (non-english) queries. Dbpedia was set-up to only search english site and english labels. Any other language is not supported and gives erroneous results
3. Queries for particular companies or businesses. In our Dbpedia accessor, any object, which does not have a population, is said to be a non-location. Therefore any particular companies or businesses are omitted. For example, 'Baltimore Zoo', 'Bronx botanical garden', 'Allen Tate realtors' and others.
4. Disambiguates. Some locations in Dbpedia have more non-location disambiguates than location. Therefore P(location) is less than 0.5 for theses queries, meaning they are incorrectly treated as non-locations. For example, 'Banes', 'Britannia', 'Fort William' or others

We also have some disagreements with the golden dataset. We believe there are errors and discrepancies in golden query labels, which would result in even human evaluators mislabeling.

1. Some queries with company names and businesses are labeled as 'Local', some are 'Not local'. E.g., Local – 'baltimore zoo', 'bellaire high school', BUT Not Local – 'bbc'
2. We disagree that 'Caverns in Alabama' is a 'Yellow Page' type query. Caverns are natural places, therefore it should be 'Map' or at least 'Information'.
3. We also disagree, that 'About Australia' is a 'Map' type query. It should be 'Information'.

Ideas for improvements

There are a few things in the system, which could be fixed in order to improve the performance.

1. Language detector – this system would really benefit from a module, which could give reliable language identification. We have tried using Python's langdetect package, however the language predictions gave us more problems than solutions. The predictions were not very accurate, thus it could not be used to decisions in our system. There is an option in Freeling, which enables to detect language, however we have not tested it.
2. Better results would have been achieved if queries with Businesses and Organizations were classed as 'Local' queries. Dbpedia accessor could be modified to account for this. However we have to be careful, as there could be many normal word combinations, which could be interpreted as names of companies. A burlesque query 'How to eat an apple' best imitates the concern. So there is a risk that this could result in finding more locations/companies than needed. A possible way around this would be to implement a semantic analysis of a query.
3. Improvements also have to be made to <WHAT-TYPE> resolution. As mentioned, at the moment we are using a rudimentary list of keywords, which imply a 'Yellow Page' query. However this method is not reliable, not self-updating and generally messy. A better solution, which would most likely give a better results, would be application of machine learning algorithms or even possibly neural networks. This, however, would require huge amount of labeled training data.

How to run the code

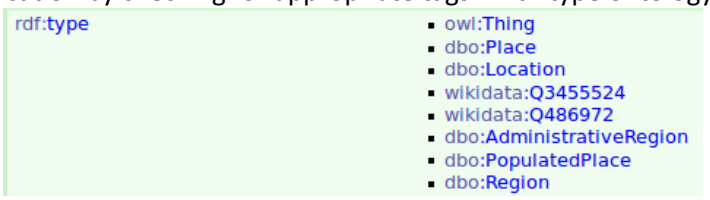
In order to run the query parser, use main.py file. There are 3 options provided, so uncomment the only one required.
Option 1 – reads and writes to specified XML files.
Option 2 – lets to select read/write files using file selection dialog
Option 3 – just a simple demo, which writes results to terminal

Appendices

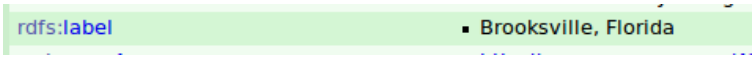
DBpedia accessor

Just to recall our Dbpedia accessor - we are using SPARQL querying language to access this service and the querying is done in several stages. Input of Dbpedia accessor – a string.
Output – tuple of (P, Label, Type), where P – probability, that the string is a location; Label – full name of the location, including state and country if applicable; Type – irrelevant for this paper, see DbpediaAccessor.py for more info). Querying process:

1. Find resource with direct URL query with 'http://dbpedia.org/page/' + name. For example for Berlin, the accessor would try http://dbpedia.org/page/Berlin. If resource is found, the system checks if it is a location by checking for appropriate tags in rdf:type ontology.



2. If there are no direct resources, the system performs a supervised keyword search in *rdfs:label* property. Supervised in the sense, that it does not return all found resources with given keywords, but instead only very similar in content and length. It mainly accounts for additional punctuation and capitalization. For example, searching 'brooksville florida' would only match 'Brooksville, Florida'. As per step 1, the resource is checked if it is a location



3. If nothing is found, it looks for page redirections in *dbo:wikiPageRedirects*

This could sometimes account for frequent typos for popular locations or postal codes, for example querying 'Berlib' would return no results in previous steps, but would return a redirection. As per step 1, the check for location is performed.

4. Lastly if there are no direct resources or redirections, the accessor checks if the page contains 'dbo:Disambiguates', meaning there are many possible Dbpedia pages with such name. For example 'Bellevue' has 58 disambiguates

The system then accesses each of these disambiguates and checks if they correspond to a location. It then returns a probability, how likely that the query is a location.
P = (#disamb. which are locations)/(Total # of disamb.)

Dealing with geographical queries - if there are more than 1 disambiguate in the query, the one with the highest score is the only one treated as location. For example, in the query 'Bellevue shopping center in Bluefield', Bellevue and Bluefield are both disambiguates with P=0.52 and 1.00 respectively, therefore Bluefield is the only one treated as location.

To resolve the correct Label from the list of disambiguates, the script selects the location with the highest population. For example 'Bluefield' has 3 disambiguates, which are all locations. The selected label is 'Bluefields, Nicaragua' as it has higher population than the other two.