INPL: Query Parsing Task for GeoCLEF2007 Report

Due on Friday, April 9, 2010

Delivery 1 - Lab Group 4

Axel Brando Guillaumes
Vicent Roig Ripoll

Contents

Introduction

Geo-query parsing task is a sub-task in **GeoCLEF2007** and it is run by Microsoft Research Asia. We have provided a query set of 800.000 real queries (in English) from MSN search. The proposed task requires that, based on the provided query set, the participants first identify the local queries and then analyse the different components for the local queries. We have provided a sample labelled query set of 100 queries for training.

The training corpus consists of 100 queries and is attached *(GQ_Tr_100.xml)*. The test corpus consists also of 100 queries and will be provided one week before the deadline of the laboratory project *(GQ_Test_100.xml)*. The only difference in the format of both files is that in the later the answers fields are obviously not included (after the deadline a file, *GQ_Test_with_answers_100.xml*, with answers will be provided).

Description of the problem

The aim of this delivery consists of writing in python a **corpus accessor** for accessing the .xml file and building a more suitable representation of all the queries (the accessor could be used for accessing both training (with answers) and test (without) files).

Approach used for solving the problem

We decided to develop **two different approaches** for the problem as we thought that each one has its own benefits depending on a design point of view. Therefore, we provide a module composed by one version implemented using procedural programming and another version that uses Object-oriented programming (OOP).

The project directory tree has been structured in multiple folders as follows:

- /lib: includes all classes and functions source files.
- /scripts: includes all usage script files.
- /data: includes xml data files.
- /doc: including task and delivery documents.

Object Oriented version

On the one hand, in order to solve the problem, firstly we considered to use the object-oriented paradigm (OOP). We defined a Query class that contains all the attributes that identify and are characteristic of a subset of words (which we called *Query*). Afterwards, we created a list of objects of this type. Given that the access to the variables in Python language is never private, it makes no sense to define a class with getters and setters. However, it is possible to write a typical OOP implementation, as it can be seen in the file which not contains *reduced* in it's name. Therefore, we made a more adequate implementation of a corpus accessor to use it in Python where the variables will have to be accessed directly.

The files involved are:

- /lib/query_classes.py
- /lib/query_reduced_classes.py
- /lib/query_functions.py

- /lib/query_reduced_functions.py
- /scripts/query_scripts.py
- /scripts/query_reduced_scripts.py

Procedural version

On the other hand, the solution provided in *xmlToDict* version has been developed in a way that most people can feel comfortable using the basics of Python language.

The files involved are:

- /lib/xmlToDict_functions.py
- /scripts/xmlToDict_scripts.py

Once the xml has been parsed, a python dictionary type is retrieved instead of the previous version, which provides an OOP instance. In addition, we provide some formatted functions *dumpClean*, *showQuerieNos* in order to represent all sorted dictionary data in a more suitable way. We assume that dictionaries are made to storing key-value relationships and use directly a structure of dictionaries may be interesting and fast to work with.

Discussion and Evaluation performed

What is more efficient in Python in terms of memory usage and CPU consumption - Dictionary or Object?

Attribute access in a Python object uses dictionary access behind the scenes - so by using attribute access we are adding extra overhead. Plus in the object case, we are incurring additional overhead because of e.g. additional memory allocations and code execution.

Programming is all about managing complexity, and the maintaining the correct abstraction is very often one of the most useful way to achieve such result, that's why we have preferred to write both versions instead of choosing one single abstraction.