Categorization Experiments

Set up categorization experiments based on a taxonomy of items (eg: shopping) and post mechanical turk tasks to collect user responses for evaluation.

Overall Design of the system and its various components:

The taxonomy has been hand-crafted (based on Amazon's shopping catalog) in the form of a mind map file (.mm). Mindmap format is easy to view, expand, collapse nodes and also useful during debugging. Use FreeMind software to open this file to view the taxonomy.

A python (bottle-based) web application has been used to create the questions by sampling different nodes in the taxonomy. This application starts off by reading the taxonomy from an xml file (an xml file was created for the same .mm file) and storing it into a sqlite database table. Another table called tasks_distribution stores the various combinations of the products, items, etc that form the task. It also records the # of times it was shown to the user along with the response.

Mechanical Turk allows creation of various types of tasks. In this case, we point the mechanical task experiment to use an External web site as the source for interaction. Then, on starting the mechanical turk experiment, our external website loads up inside a frame in the mechanical turk environment itself. Python's Boto library has been used to create a HIT programatically as an ExternalQuestion type. Although I was able to create a HIT and see that our website gets loaded as an embedded page, the frames were messed up badly. The website was not fully visible and buttons got hidden below. Basically UI needed to be fixed. I have not got past that stage.

Description of the Code/Resources in the archive:

The source code root directory looks like below:

categorize -

- app (contains the bootle web application project)
    - db (contains all sqlite db related stuff)
        - turk.db (this is the database binary file)
        - taxonomy.xml (this is xml equivalent of the mindmap file which is loaded into the db). I had to remove duplicate nodes at some point.
    - templates (contains all the html files for the interfaces used. We mainly looked at three types of interfaces, them should explain themselves when viewed, but if not, please contact me)
    - create_tables.py (to create the sqlite db and tables from scratch. this is done only once and then tables are updated. **It is also useful if you accidently delete the turk.db sqlite file**).

- views.py (this is the MAIN source code file that contains most of the logic. **I have described the important methods in this file in a separate section below!**
- taxonomy_tree.py (contains a custom definitoin of TreeNode class used by views.py. I could not find a python based tree implementation, so defined one myself. Advantage of this was that I could write my own methods to traverse the tree in ways to suit our experiment).

- mechanical_turk (contains boto library and createHIT.py - which is the python boto code to create an ExternalQuestion type task in mechanical turk). This works, but the frame issue **has to be fixed**.
- media (contains the css files used, javascript and importantly all the images for the products etc which are pulled by the html files when displayed). I think at this point, we restricted ourselves to only 20 products (leaf level nodes).
- packages (contains bottle which is the web framework, jinja which is the templating engine for all the UI stuff)
- app.yaml, index.yaml (are bottle specific files that need to be present)

High-level overview of the main.py file (contains most of the logic):

landing_page() is the starting point of the application. Thereafter, depending on the type of next task we want to create, we call load_interface1(), load_interface2() or load_interface3() methods. Only interface1 has been fully implemented currently. Rest two methods are just stubs for now.

On loading the main html page, populate_tree() method reads the taxonomy.xml file and loads all items into a the custom TreeNode object. After that this tree object is navigated to set up all tasks as required.

Task Distribution Table is used to keep track of the types of questions we have asked already and how we create the questions (combination of items and +ve, -ve buckets, etc). More details on how this distribution table looks like and is used, type: http://localhost:8080/db when running the application. This /db url basically displays the current state of the sqlite db (for debugging purposes only).

There are a bunch of utility methods like find_uplist(), find_nodes_in_subtree() etc which I have written to extract/filter nodes from the taxonomy tree as needed.

I had added lot of debug (print) statements in the code and they will show up in the output as well. My code verification process consisted of examining this output and matching it against the taxonomy mindmap file (.mm file opened up in FreeMind) to see if my logic was correct.

How to run the application:

1. Install bottle web framework.
2. Go to the source code directory and type:

$ python main.py

3. Navigate to the url: http://localhost:8080 on the browser. You should be able to see the application and complete the tasks according to the instructions.
4. I have set the maximum # of tasks to be 5 in a config file, so after 5 tasks the db is updated. At that point, you can also view the contents of the database and the format of data in the tables by typing: http://localhost:8080/db .