RMIT University
School of Computer Science & IT
COSC 1165 / 1167 – Intelligent Web Systems
Assignment 2

Web Crawler and Mining

You may complete this assignment **individually or in a group of two.**

In this assignment, you will implement a crawler to fetch and store web pages. The "Intelligent Web Systems" blackboard contains further announcements, a list of frequently asked questions, and a discussion board for this assignment. You are expected to check the discussion board on daily basis. Login through `http://my.rmit.edu.au`.

Plagiarism

All assignments will be checked with plagiarism-detection software; any student found to have plagiarised will be subject to disciplinary action as described in the course guide. Plagiarism includes submitting code that is not your own or submitting text that is not your own. Allowing others to copy your work is also plagiarism. All plagiarism will be penalised; there are no exceptions and no excuses. For further information, please see the items under "Academic Integrity" at `http://www.rmit.edu.au/compsci/current.`

1 - Web Crawler (25%)

Your task is to build a simple web crawler. When provided with a seed page URL, the crawler should fetch the page, and then fetch pages linked to from the seed page, until either there are no more pages to fetch, or another termination criterion is reached (see below).

Each page that is fetched should be stored under the current working directory from which the crawler is being executed. Your program should not try to open URLs that are not of type text/html (such as images and videos). Furthermore, your crawler will only follow links to pages that exist on the same host as that provided as the seed URL.

Your crawler must be executable from the command-line as follows:

```
./crawl [-politeness <seconds>] [-maxpages <pages>] seed_url
```

That is, the executable name of this program must be "crawl" (or crawl.py, crawl.php etc) and it must be supplied with a starting point `seed_url`. In addition, it must be able to accept (and correctly act on) the following optional arguments:

```
    -politeness <seconds>
```

Time, in seconds, that the crawler must wait before attempting to fetch another page from the same host. If this item is not specified, your program should use a default delay of 10 seconds.

```
     -maxpages <pages>
```

Limit the total number of pages that your crawler will fetch and store. If this option is not specified, your program should use a default maximum of 20 pages.

To implement your crawler, there are certain key design decisions that you need to make, including the choice of appropriate data structures to:

- Track the pages to be visited.
- Track pages already visited.
- Follow standard politeness protocols.

2 - Data Mining (25%)

Your task is to use the web crawler you built for Part 1 to collection data for training a crawler which is only interested in text relevant to mobile.

Your crawler can go to `http://www.theage.com.au/digital-life/mobiles` and collect articles from there. You can assume the articles under that URL such as `http://www.theage.com.au/digital-life/mobiles/Mobiles/iPhone` are all relevant to mobile and pages from outside are all irrelevant to mobiles. You need to extract features that could differentiate mobile articles from non-mobile articles such as the sport news, a financial report, new release of DELL PC etc. Design a suitable set of features is a part of this exercise. Note, other than mobile articles your crawler should collect roughly equal amount of non-mobile articles as well to facilitate the training.

After the crawling and feature extraction, your program should be able to generate an arff file that can be trained in WEKA. There are only two classes, “Mobile” and “Not Mobile” in your data set. The class label can be generated automatically based on the URL. (You may manually label the data. However that would be a very time-consuming task.)

When provided with a seed page URL, the crawler should fetch the page, and then fetch pages linked to from the seed page, until either there are no more pages to fetch, or another termination criterion is reached (see below).

The generated arff file should be tested on WEKA. Now you can experiment with different classifiers available on WEKA and different parameter settings for each classifier. Note “Cross Validation” option should be used during the training, so the classification accuracy would be more realistic. Otherwise the trained classifier may achieve high accuracy on training set but perform very poorly on test or unseen data set.

3 - Smart Web Crawler (30%)

Your task is to use the classification model (classifier) generated from the previous section to collect mobile-related article online from a given url, e.g.
http://www.heraldsun.com.au/technology

The improved crawler may collect an article or a set of articles, extract features, and classify them into “Mobile” and “Not Mobile”. Ideally this crawler would only present “Mobile” articles back to the user. For the sake of this exercise, each article is marked either “Mobile” or “Not Mobile” by the crawler.

As the extension of this exercise, your program may provide an opportunity for users to correct the return results from the crawler. That means user can mark an irrelevant article “Not Mobile” or mark a genuine “Mobile” article as positive if it is indeed marked wrongly by the crawler. Based on the user correction, you crawler may re-train the model to improve the performance of the crawler.

NOTE: there is no need to find the “perfect” feature set or train a “perfect” classifier that may achieve 100% accuracy. Your task is mainly to implement the whole process of mining useful information, providing intelligent solutions and improving the performance overtime.

Notes on programming

- Your programs must be well written, using good coding style such as appropriate use of comments and meaningful variable names. Coding style is a part of the assessment.

- You may use any programming language of your choice to implement the crawler. For example, Java, C, Perl, or PHP are all suitable choices.

- Your programs may be developed on any machine, but must compile and run on RMIT laboratory machines.

- Please ensure that your submission exactly follows the file naming convention specified in this assignment specification.

4 - Report (20%)

In addition to the source code of your program that follows the above guidelines, you need to write a short report. This file should be in pdf format, and should be named “report crawler.pdf”. The report should include the following (please number each section and use the headings indicated below):

1. Your name and student number (group information).

2. Operating instructions: a short paragraph giving clear instructions on how to compile and run your code.

3. Implementation. A report that details:
    - an overview of how your crawler is implemented, and how it operates, including politeness policies;
    - any data structures that you have used, including to keep track of pages to visit, and pages already visited;
    - a brief explanation of any language-specific libraries or functions that you have used as part of your crawler;
    - your design of features for classification;
    - the details of your arff file for training;
    - the classifiers that you have experimented with and your findings through the exercise;
    - the classifier or classifiers you selected for the smart crawler;
    - the structure of your smart crawler, e.g. how does it make decisions based on the trained model?
    - the performance on test or unseen data (websites);
    - how user feedback (if any) is used to re-train the classifier?
    - the performance of the re-trained model on test or unseen (websites);
    - your conclusions, comments on this mining exercise;
    - possible future works on how to improve the smart crawler in terms of accuracy and efficiency.

Your report should not be more than 3 pages in length.

What to Submit, When, and How

The assignment is due on **Tuesday, 8th May 2012, 11:59pm**.

You should submit your source code, reports, and README file using weblearn. (Do not submit executables, object files, or data.)

Your weblearn submission must have a timestamp showing Tuesday, 8th May 2012, 11:59pm (or earlier). Late submissions should be submitted using the same weblearn procedure, but be aware that these will suffer a mark deduction as explained in the course guide.