|  |
| --- |
| BI&A 660 Web Analytics |
| **Selecting Complementary Courses on MOOC - List.com** |
| Final Project Report |

|  |
| --- |
| Team 3: Changshuo Gao, Fangzhou Yan, Fei Feng, Hui Zheng, Yi Wang, Xin Sun  5/5/2014 |

**Problem Motivation**

MOOC(Massive Open Online Course) is a recent development in [distance education](http://en.wikipedia.org/wiki/Distance_education), which refers to an [online course](http://en.wikipedia.org/wiki/Online_course) aimed at unlimited participation and open access via the [web](http://en.wikipedia.org/wiki/World_Wide_Web). In addition to traditional course materials such as videos, readings, and [problem sets](http://en.wikipedia.org/wiki/Problem_set), MOOCs provide interactive user forums that help build a community for students, professors, and [teaching assistants](http://en.wikipedia.org/wiki/Teaching_assistant). Many websites provide different MOOCs, while “MOOC-List” (http://www.mooc-list.com/) is an aggregator (directory) of MOOCs from different providers.

Since there are more than one thousand courses on MOOC-List, it is difficult for people who desire a free education and have the motivation to find an ideal course from various MOOCs. The main objective of our project is to assist the users to find courses suit for them.

**Problem Definition**

Users may face two problems when making decision on which course to take. On the one hand, they have to choose the subject to narrow the choices. If a user has no interest on a specific area, he only wants to obtain knowledge on some popular subjects which may benefit his career life. However, there is no information about which area or subject is popular on MOOC-List. In order to address this problem, we have analyzed the data extracted from MOOC-List and visualized our analysis result to show the popular subjects. On the other hand, after choosing the subject users need to search a course in a specific area. When a user plans to learn multiple subjects within a time period, the existing search function on MOOC-List cannot find a course satisfying the user’s requirements. For example, if a program developer intends to spend 3 months learning Python and Hadoop form online courses, it may take him several hours to select a course or a course set on MOOC-List. Therefore, we tried to create a function which is able to help users easily select complementary courses on MOOC-List.

**Data Description**

Looking at the webpage, there are 18 items of information for each course. We have extracted all the information for each USA course in midterm project, but for this project we only focus on discovering tags, course length and estimated effort. Thus, we filtered data to keep four items, including course name, tags, length and effort, and cleaned data by deleting the courses on which webpage these three items are shown as “No Information”. We stored the data as a text file for analysis and also created an output like query formula so that we can import the dataset through reading this sql file. To import the data into MySQL local database, we used a package named MySQLdb which is specialized in manipulating database. Through creating database, creating table, and reading sql file, we accomplished data import procedure.

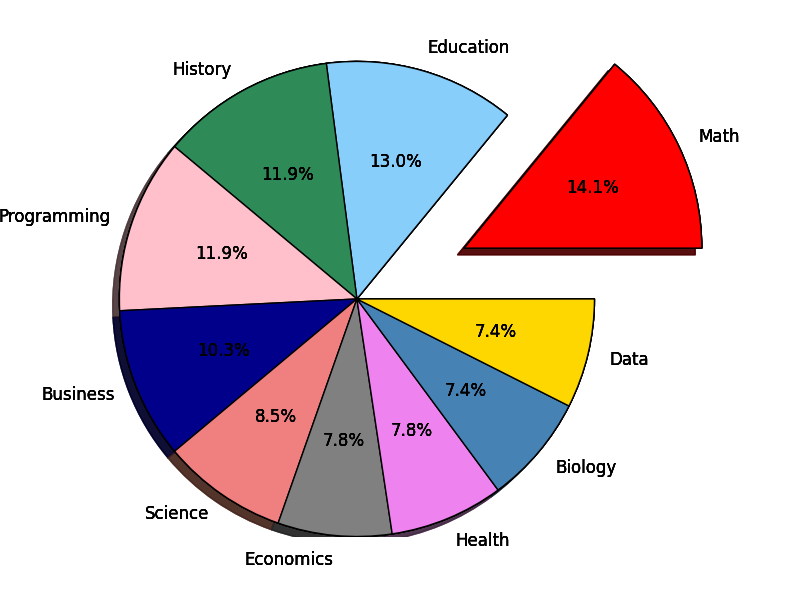
**Data Analysis**

Before selecting a course, users may have several questions: What are relatively popular subjects (tags)? How frequently do the popular tags show up on MOOC-List? If I want to learn these popular areas (tags), what is the appropriate time they may cost? Based on these questions, we analyzed the data and visualized the analysis result.

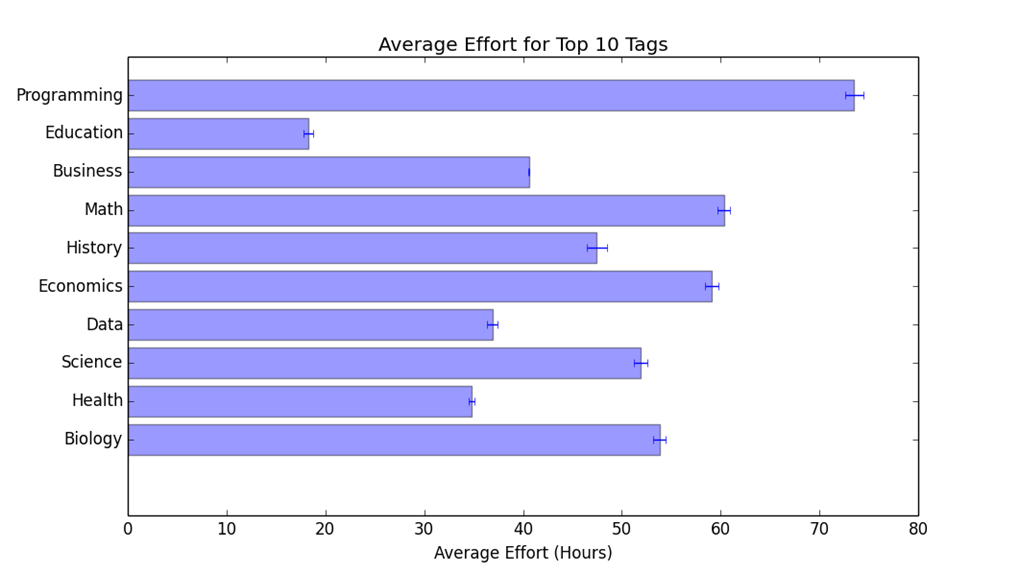
Tag clouds have become popular as a summarized representation for text data, so it is our preferred visualization tool. We created a tag cloud by using Pytagcloud Library, so as to address the first question. As shown below, the importance of each tag is shown with font size or color. This format is useful for quickly perceiving the most prominent terms and for locating a term alphabetically to determine its relative prominence [1]. The bigger size the word displays on this tag cloud, the higher frequency the tag appears on MOOCs. Obviously, math, education, programing and business are very popular subjects, as many MOOCs include these tags.



Besides that, we created a pie chart to show the percentage of frequency for popular tags. By splitting the tag file as tokens, we get the length of token is 4722, which means tag data consists of 4722 items. Then we use Counter and defaultdict functions to count how many times each tag appears and rank them by frequency. According to the result, top 10 popular tags are Math (89 times), Education (82 times), History (75 times), Programming (75 times), Business (65 times), Science (54 times), Economics (49 times), Health (49 times), Biology (47 times) and Data (47 times). Compared with the total number of 4722, the total frequency of these popular tags is small. If the pie chart consists of all the tags, top 10 tags would only occupy a small piece. Thus, our pie chart shows the percentage of frequency for top 10 tags, rather than for all the tags.



For the third question, we calculated the average study time for each top 10 tag. At first we got the study time for each MOOC by multiplying course length with estimated effort. After we summed the study time for all the courses which contain the tag, we reached the average study time through dividing the total time by the frequency of the tag. Finally, we created a horizontal bar chart to show the result. Observing the bar chart, programing seems to be the most time consuming subject, followed by math, economics and biology, while education is likely to be the easiest subject. Recommendations could be put forward, based on this chart. For example, if a user has limit time, we will not encourage him to take a course which covers programming.



**Methodology and Interface Design**

Since users may have more than one requirement, we use greedy algorithm to enable the function to select complementary courses on MOOC-List. Greedy algorithm is an [algorithm](http://www.wikipedia.org/wiki/Algorithm) that follows the [problem solving](http://www.wikipedia.org/wiki/Problem_solving) [heuristic](http://www.wikipedia.org/wiki/Metaheuristic) of making the locally optimal choice at each stage with the hope of finding the global optimum [[2]](http://en.wikipedia.org/wiki/Greedy_algorithm#cite_note-NISTg-0). The idea behind a greedy algorithm is to perform a single procedure in the recipe over and over again until it can't be done any more and see what kind of results it will produce. It may not completely solve the problem, or, if it produces a solution, it may not be the very best one, but it is one way of approaching the problem and sometimes yields very good (or even the best possible) results. Although greedy algorithm does not guarantee an optimum result, for this case it is an efficient way to find satisfying result.

To implement the algorithm, we read the input as target set. If the user chooses “self-paced” option, we will search for self-paced courses firstly. When all self-paced courses cannot come up with a result, we will turn to courses with fixed course length. The course which has most tags in common with the target is added into result set. Once a tag in target set has been satisfied, we will remove it. The above procedure will be repeated until no tag leave in the target set. When the target set is empty we get the appropriate result for user, nonetheless we may not find the courses for the user when the target set is not empty and no more tags can be removed from the target set. The result will be shown to the user, if the total course length is not more than the user input, otherwise we will notice the user no result with this time budget. As we can see from above, the greedy algorithm is implemented in 2 tiers: looking for most tags and then seeking less cost (course length) in former process.

The search function is implemented as a webpage with backend in PHP and mysql database. To improve the user experience, we designed a user friendly interface. In the interface, user can type in the tags and choose either fixed course length or “self-paced” option. When the option is checked, the time input area will be disabled. The looking appears clear with notice on the page. The page style is made from a framework called Bootstrap. Since some user may not know what tag can be chosen, we made all the tags displayed by alphabetical order on the bottom of interface. We wrote several JavaScript functions so that the user can easily click on the tag then it will appear in the input area. Furthermore, we implemented the hint showing function to provide the user with more convenience by using AJAX. When the user types some character or word in the input area, the page will show the tags that contains the character or word.

**Conclusions**

Greedy algorithm has several advantages, including efficient performance, easy to design and implement, etc. Nevertheless, greedy algorithm should be used with great care, since for many optimization problems its usage seems impractical even for generating a starting solution (that will be improved by a local search or another heuristic) [3]. In additional, it cannot promise to output best result. Fortunately, we only need to meet all the requirements from users but don’t need to provide them with an optimal course set, hence greedy algorithm works well on our project.

For future work, our process would be improved by calculating cost for each tag in the target set. For instance, if we have course I with 4 tags in target set and with a cost of 20 weeks, the cost for each tag should be 5 weeks; Course II with 3 tags in target set and with a cost of 9 weeks, the cost for each tag should be 3. Based on this idea, we will choose course I because it has more tags but less cost.

**References**

[1] Dimitrios Skoutas and Mohammad Alrifai, “Tag Clouds Revisited”, *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 221-230, 2011.

[2] “Greedy algorithm” [online] *Available at* [*http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy\_algorithm.html*](http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html) [Accessed 5 May 2014].

[3] [JøRgen Bang-Jensen](http://dl.acm.org/author_page.cfm?id=81410595801&coll=DL&dl=ACM&trk=0&cfid=452873440&cftoken=93270227), [Gregory Gutin](http://dl.acm.org/author_page.cfm?id=81341490894&CFID=452873440&CFTOKEN=93270227) and [Anders Yeo](http://dl.acm.org/author_page.cfm?id=81100426392&CFID=452873440&CFTOKEN=93270227), “When the greedy algorithm fails”, *Discrete Optimization, Vol.1, Issue 2*, PP. 121-127, November 2004.