HackerRank Machine Learning CodeSprint:
Challenge Recommendation

Name: Caio Taniguchi
Email: caiotaniguchi@gmail.com
Date: September 4th, 2016

Introduction

The *HackerRank: Challenge Recommendation* is a challenge to develop a recommendation system that indicates a set of HackerRank challenges that a given hacker is likely to solve that is part of a target contest. To develop the model we have two datasets at our disposal. The first one is *challenges.csv*, which contains information about each challenge available, like its number of submissions, the contest which it was part of, its domain, degree of difficulty. The second is *submissions.csv*, which contains the interactions between a hacker and a contest-challenge pair, including whether the challenge was solved or not.

Data Exploration and Preprocessing

Recommendation systems often implement one of two techniques (or a combination of both): collaborative filtering and content-based filtering. The former uses past hacker-challenge interactions to make recommendations, while the latter uses metadata to group items and users by some measure of similarity.

Due to its simplicity, I began my exploration with a collaborative filtering approach in mind. To do so, I transformed the data in the *submissions.csv* file to a hacker-challenge matrix, with each cell *[i , j]* containing the number of submissions of hacker *i* to challenge *j*. One problem that became apparent was the high sparsity of the interaction matrix (99.27%). Given the setting of the problem, it makes sense that we have very few unique hacker-challenge interactions, since a hacker is likely to try the same challenge many times, while submitting to only a limited set of challenges.

The sparse issue made me move on to a content-based approach, so I began analysing the *challenges.csv* dataset. The first problem that appeared was that the challenges were only unique within a contest, but not unique in the dataset. Another problem was that challenges that were not part of the target contest did not have any information about its domain or subdomain.

To deal with the missing data, I assumed that a given challenge is always part of the same domain-subdomain, regardless of its hosting contest. With this in mind, all challenges that were also part of the target contest could have their values imputed. The imputation for challenges that were not part of the target contest was trickier. I used the fact that most languages have only a few popular usages and merged it to the *challenge.csv* dataset, then

each missing domain-subdomain pair was filled the most frequent domain-subdomain for the language used by the hacker for the challenge. Some challenges still could not have its domains inferred this way and were assigned to an “Unknown” domain.

Having solved the missing data issue, I turned to the challenge duplicates problem. I simply summed the duplicates feature counts to the main entry (the one from the target contest) to keep the count information and removed the duplicates.

This takes care of the data preprocessing. The previous steps are also summarized in table 1.

Problem	Solution
High sparsity (99.27%) of the hacker-challenge interaction matrix	Discard pure collaborative filtering solutions and focus on content-based filtering
Challenge duplicates in the *challenge.csv* dataset	Sum submission counts to the main entry, the one that is part of the target contest and discard the duplicates
Missing domains and subdomains	Use language usage from users that participated in the challenge with missing data to impute values. Assign ‘Unknown’ to challenges with insufficient data.

Table 1. Dataset problems and implemented solutions

Modeling

As stated in the previous section, a pure collaborative filtering approach was discarded due to the high sparsity of the hacker-challenge interaction, so that left me with hybrid or pure content-based solutions. Since this challenge was my first contact with recommender systems and the dataset is not so straightforward (there is no explicit interaction ranks and past challenges are discouraged through the evaluation metric), at first I decided to go with a pure content-based filter, but ended up with a hybrid solution, even though it’s a simple one.

The design I came up with for the recommendation system is mostly based on the assumption that if a hacker participates in a challenge that is part of a certain domain/subdomain, he is interested in that domain/subdomain as a whole, so he is likely to at least submit for some of the other challenges in the same scope. On top of that, we already know which challenges a hacker already tried and not has not solved, so we can also add that to the recommendations. Within each context, the candidates are ordered by number of solution submitted, using the rationale that if a challenge has been solved more often than others, then it is also more likely to be solved in the future.

To summarize, the model implements the following steps to compute the recommendations of each hacker:

1. Sort challenges by total number of solved submissions in decreasing order;
2. Select every challenge attempted but not solved by the hacker;
3. Select every challenge not solved by the hacker that is part of the challenges' subdomains that the hacker attempted and add them to the end of the recommendations queue from (2);
4. Select every challenge not solved by the hacker that is part of the challenges' domains that the hacker attempted and add them to the end of the recommendations queue from (3);
5. If the number of recommendations exceed 10, throw away the excess. If it is below 10, add the most solved overall challenges

For the actual implementation I chose to use Python mainly due to my familiarity with the language and the fact that the Pandas package makes data manipulation easy and it handles it efficiently, although the model is simple enough that it could be easily ported to other languages.

Conclusion and results

As a first attempt at recommendation systems, the results were quite satisfactory. Despite its simplicity, the model was able to achieve a score of 0.142, placed 19th in the public leaderboard at the time of this writing. For future attempts I would like to try some black-box models that were not possible for this competitions due to time constraints. It would also be interesting to explore other forms of ranking that could include the difficulty values, which I was not able to implement successfully for this challenge, and leverage the hacker metadata and interaction data more efficiently.