CAPP 30122, Final Project

Group Members: Syeda Jaisha, Onel Abreu, James Midkiff, Domingo Carbone
CNetID: *dcarbone-jmidkiff-oabreu-syedajaisha*

Project Overview.

Our work intends to tackle two problems related to tabletop board games: obtaining personalized recommendations for consumers interested in new titles and, predicting success rates for games in development and making recommendations to improve the same.

To build these systems, we used data from BoardGameGeek, www.boardgamegeek.com, a website that aggregates user ratings and game information for 100,000+ boardgames. We scraped a list of the top 5,000 titles ordered by total number of ratings. Afterwards, we used the site's own API to build a database of these games and their characteristics, listing variables such as type of game, recommended number and age of players, and game difficulty, among others.

Based on this data, we implemented a search functionality through which users describe the board games they're interested in, as well as their preference for popular vs. critically acclaimed titles, and receive a personalized list of the top five releases that meet their criteria, as well as some of their characteristics.

Finally, we created two predictive systems which, based on a number of inputs (unique to each model with some overlaps) foresee either the number of BoardGameGeek user ratings a future title might have, or its projected aggregate rating. In order to contextualize these results, we also provide the relative rank such a game would take within BoardGameGeek's popularity and rating metrics.

Original plan and final implementation.

We initially set out to build two structures: A recommendation system for titles based on the user's preferences, and a predictor on the likelihood of success in a crowdfunding campaign on Kickstarter, www.kickstarter.com, for game producers planning on future projects.

In order to build these, we intended to use two sources of information: A base composed of BoardGameGeek's top 5,000 games, ordered by user ratings, and a collection of Kickstarter's most successful board game campaigns, ordered by funding.

As work began, we ran into challenges scraping Kickstarter, particularly regarding automatically identifying successful campaigns, as well as matching them with their Game Board Geek counterparts. Simultaneously, we realized Game Board Geek already provided a good indicator for the success of a game, in the number of user reviews each title had, indicative of overall adoption rates.

Finally, we decided to drop Kickstarter, and rely on both scraping BoardGameGeek and using its own API as sources of information. Using that data, we implemented the recommendation system and a pair of predictive models (which forecast adoption rates and ratings, rather than funding). We also provided a Django based front-end interface to interact with all systems more easily.

Software Structure.

Our project begins with the file *main.py*. This opens an interactive terminal session with our three project components: the ‘Game Recommendation’ system, the ‘Regression Prediction’ system, and an application to test the web scraping and API extraction scripts. For the first two components, *main.py()* opens the corresponding Django site. For the third component, the file runs the *bgg_api.py* script for a limited number of boardgames adding a suffix to the new filenames to ensure the new data does not overwrite the data used in the main systems. Finally, *main.py* also includes an ‘About the Project’ screen.

The data used in the main systems is the product of two functions: The first, *bgg_crawler*, is a web scraper that traverses BoardGameGeek and extracts game id’s and short descriptions for a number of games, that are ranked by popularity and then exported into an csv. The function uses a for loop, BeautifulSoup, and a counter to iterate through webpages a given number of times.

Given that csv, *bgg_api.py* pulls data from BoardGameGeek’s API, analyzes it, and exports it. The API is called in batches, extracting each game’s information, and exporting the full dataset as a JSON file. As there are too many board game categories and mechanics to use in a meaningful regression, *bgg_api.py* determines the ten most common of each, flags games that use them, and exports this revised dataset as a CSV. The script also exports additional CSVs with the total game type, category, and mechanics counts for further analysis.

Our front end utilizes Django and some of the base code provided from CS122’s 3rd programming assignment (credit given to Gustav Larsson who wrote the interface and Kartik Singhal who ported it to Django 2). We adapted *index.html* and *main.css* to include stock images and style changes, but the majority of the program work occurs from the *views.py* function in each folder. In this function, we created the appropriate Django form inputs (multiple choice, dropdowns, etc.) for users based on the function needs, extracted form data into dictionary formats to serve as input to game_search or regression functions, and then rendered the outputs to webpage form using Django.

The “Game Recommendation” system lies in *game_search.py*, which is structured around a main function, *find_best_match*. This function takes as sole input a dictionary generated by the Django interface from user input, and opens the game data csv file, to then call on its helper functions.

These begin by generating a search dictionary, used to query the .csv file, and then iterate through said dictionary’s keys in a predefined order, intended to block and filter out the largest amount of data as fast as possible (starting with difficulty, which is reduced from a 50-point score system to one of three values). The remaining data is then sorted according to the user’s preference for popular games, highly ranked games, or neither (in which case a new column normalizing and averaging both is created, only to then sort). Given this sorting, *find_best_match* returns the top five results in descending order.

As for the “Regression Prediction”, the main function *predict* accepts a dictionary generated by the Django interface containing user inputs and a boolean that determines which regression model to run. It then processes the game data CSV, and constructs the required *x* and *y* matrices. The regression is then run, and the resultant β vector is used to predict a value given the user input. Each model’s R^2 is also obtained, as well as the predicted rank of the user-defined game in the dataset. Then, it generates a list of top 5 games in our dataset, sorted by the parameter of interest to the user. Finally, based on our regression results and given user inputs as constraints, the function recommends simple changes user can make to improve their game and and the expected improvements associated with these changes.

File Locations and Authors

Repository: capp30122-win-21-dcarbone-jmidkiff-oabreu-syedajaisha/

- *main.py* - James Midkiff
- *bgg_crawler.py* - Onel Abreu
- *bgg_api.py* - James Midkiff
- *regression.py* - Syeda Jaisha
- *search.py* - Domingo Carbone
- *front_end/regression_ui/search/views.py* - Onel Abreu
- *front_end/game_search_ui/search/views.py* - Onel Abreu
- *front_end/regression_ui/search/templates/index.html* - Onel Abreu
- *front_end/game_search_ui/search/templates/index.html* - Onel Abreu
- *front_end/regression_ui/static/main2.css* - Onel Abreu
- *front_end/game_search_ui/static/main.css* - Onel Abreu