Machine Learning Engineer Nanodegree

Capstone Proposal: Building a Game Recommender System

Robert Lizatovic

August 23, 2019

Proposal

Domain Background

Recommender systems attempt to predict a “rating” or a “score” users would give to items, and based on these predictions recommend new items to new or existing users[1]. Recommender systems are quite ubiquitous today and many enterprises are utilizing them regularly to offer more personalized and targeted content to their customers (i.e. Amazon, Netflix, Spotify etc.). Research in recommender systems is very active and broad with different types of general approaches employed and many different types of algorithms developed and used[2]. The classical approaches to developing recommender systems include collaborative filtering (CF) and content-based filtering (CBF) methods, as well as various hybrid techniques combining the two.

In this project I will build a recommender system for computer games using the game’s review dataset from Steam (a platform for distributing, playing, and discussing games)[3]–[5]. The recommender system will function by first predicting the ratings (in the case of Steam game reviews, the ratings are a binary: “recommend” or “not recommend” labels) that players would give to games they haven’t seen/interacted within in the past, and then constructing a list of games to recommend based on these predicted ratings. The idea to build a recommender system for games is motivated by my work as a data scientist in the gaming industry where I am trying to develop effective algorithms that present players with more personalized/targeted content in the game (which would hopefully result in higher player satisfaction and in turn higher player acquisition and retention).

Problem Statement

Given past player-game ratings, predict player ratings for games that they haven’t purchased/played before and based on these predictions recommend games to users that they would probably like. To fully evaluate such a recommender system in production, one would need to set up online A/B tests and compare a metric like the conversion-rate (i.e. how many recommended games are actually purchased by the player). As this is not possible in this case, the recommender performance will be evaluated on a carefully constructed hold-out (test) set extracted from the original reviews data (player-game ratings data) that will not be used during training.

Datasets and Inputs

For this project I will use the player-game reviews/recommendation data collected from the Steam gaming platform[3]–[5].A link to the dataset can be found here: https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data. The dataset is comprised of two files:

- A player-game reviews file (in JSON format) that contains the ratings (binary “recommend”/”not recommend”), and reviews of players for various games distributed and sold on Steam, in addition to dates, number of times the reviews were upvoted by other users and other metadata.
- A game information file (in JSON format) that contains information about the games on Steam such as their title, genre, pricing, launch date etc.

The goal of the project is to predict as accurately as possible the ratings the players have given to particular games in the test set based on the reviews/ratings and game attributes in the training set. Besides the explicit ratings given by the users, the dataset contains a plethora of additional information that can provide further context for modelling player-game preference/interaction. These are for example the reviews posted by the players (their titles, main text bodies, and the number of times they were voted helpful by other users) and the general game information (game title, genre, etc.) which can be used to learn better representations (models) of how players interact with games[6].

Solution Statement

My proposed solution for the game recommender system is to learn a parametrized hybrid model of how players interact with and rate games on Steam by combining both the previous ratings data (in a CF-like approach) with game attribute data (in a CBF-like approach) that can predict with high accuracy whether a particular player will “recommend” or “not recommend” a particular game. The models will be trained on the training set and evaluated on a holdout (test) set which will be carefully constructed at the beginning of the project.

Benchmark Model

For this project, I will use two benchmark models. The first is a simple random predictor that assigns a +1 (“recommend”) or a -1 (“not recommend”) to a game based only on the distribution of positive and negative ratings in the training set (in other words, the recommendations are not personalized at all and only follow the general rating frequencies in the entire dataset). To ensure consistency/reproducibility of this baseline model, I will fix the random number generator seed.

The second benchmark model will be a model-based CF algorithm that attempts to learn latent representations of both users and games in a lower-dimensional feature space based solely on user-game ratings[2]. This will be accomplished using matrix factorization where the ratings matrix - R (representing the available ratings of every game by every user) is factorized into matrices U and V (representing the mapping of users and games onto the latent feature space respectively) whose product approximates the original ratings matrix well. This algorithm is a standard practice in recommender system development.

Evaluation Metrics

While there are numerous evaluation metrics for recommender systems, for simplicity's sake and ease of reproduction, I will use the classification accuracy of ratings in the test set as a primary model evaluation and comparison tool (as the ratings are binary in nature, i.e. "recommend"/"not recommend"). In case the exploratory data analysis (EDA) phase indicates that there is a significant class imbalance (most reviews are positive for example), I will instead switch to using the F1 score.

Project Design

Phase I: EDA

In this exploratory phase, I will extract and format the data from the 2 JSON files and compute some basic statistics such as player/game count, dataset density (how many player-game ratings are present out of the full possible set), and the distribution of positive and negative ratings. Additionally, I will produce and inspect the *long tail* plot which is standard practice when inspecting recommender system data (essentially a histogram showing the distribution of ratings over players and games). Finally, I will perform a data truncation step by extracting a k-core (a subset of the reviews in which every player rated at least k games and every game was rated by at least k players). This avoids the cold-start problem in which we do not have enough data points for a new game or a new player in order to make meaningful recommendations. I will initially set k to 5, but this might change depending on subsequent results.

Phase II: Train/Test split

In this phase I will use stratified sampling to construct a test set for evaluating final model performance. Stratified sampling is used in order to ensure that reviews from all users are equally represented in both sets and that no user is accidentally dropped from the training set entirely (many of the algorithms I will employ in this project necessitate that all users/items have at least a single data point present in the training set). Additionally, some adjustments might be necessary to ensure that no games are completely dropped from the training set either.

Phase III: Data Processing

In this step I will encode various dataset features (such as categorical features and the various text-based features such as title names) and the ratings themselves. Textual data will be encoded and potentially also embedded.

Phase IV: Benchmarking

In this phase I will use the processed training data to build the two benchmark models discussed above and evaluate their performance on the test set. The obtained values will be used as reference points against which to compare more elaborate models.

Phase V: Model Building/Evaluation

In this project stage, I will construct several more advanced models of the recommender system by utilizing more of the available data and combining several approaches.

i) As a first step, I will build and test the performance of the CBF algorithm by extracting features related to the games themselves and then building a game similarity matrix (using a similarity measure such as cosine

similarity or the correlation coefficient) that will be used for computing the predicted rating for a game by a player.

ii) I will then try to extend the basic matrix factorization CF approach by modelling user-game interactions in a non-linear fashion. Instead of the simple dot product of user and game latent feature vectors, I will model their interaction via a multilayer perceptron (MLP) by first embedding the users and games as trainable feature vectors and subsequently passing and combining them through deeper layers of the network. The model will be trained using SGD to minimize the binary cross-entropy (CE) between the predicted and real ratings in the training set.

iii) I will then experiment with weighted hybrid recommender systems that combine the predictions of both CF and CBF models (constructed separately above) using for example simple logistic regression or more elaborate techniques such as SVMs, decision trees, or various ensemble models (random forests, AdaBoost etc.). Here the predictions of the individual models will be used as features of a meta-classifier that will be trained on the training set.

iv) Furthermore, I will try to use the outputs of the CF model (the user/game interaction feature vectors) as additional features in the CBF model to compute the game similarity matrix and subsequently calculate predicted ratings as a weighted average of previous user ratings (where the weights are the similarity scores between the target game and the games that particular user has rated).

v) Finally, I will create an integrated CF/CBF hybrid model that will be trained simultaneously. This will be accomplished by first embedding the users and games as interaction feature vectors (just like in model ii) and also adding the extracted game features (from model i). These will be passed through a neural net, that will be trained to minimize the binary CE between the predicted and true ratings in the training set.

References

[1] ‘Recommender system - Wikipedia’. [Online]. Available: https://en.wikipedia.org/wiki/Recommender_system. [Accessed: 21-Aug-2019].

[2] ‘Toward the Next Generation of Recommender Systems’. [Online]. Available: https://dl.acm.org/citation.cfm?id=1070611.1070751. [Accessed: 21-Aug-2019].

[3] ‘Item recommendation on monotonic behavior chains’. [Online]. Available: https://dl.acm.org/citation.cfm?id=3240369. [Accessed: 24-Aug-2019].

[4] ‘[1808.09781] Self-Attentive Sequential Recommendation’. [Online]. Available: https://arxiv.org/abs/1808.09781. [Accessed: 24-Aug-2019].

[5] ‘Generating and Personalizing Bundle Recommendations on Steam’. [Online]. Available: https://dl.acm.org/citation.cfm?id=3077136.3080724. [Accessed: 24-Aug-2019].

[6] ‘TopicMF’. [Online]. Available: https://dl.acm.org/citation.cfm?id=2893874. [Accessed: 22-Aug-2019].