STEAM GAMES: PERSONALIZED RECOMMENDATION ENGINE

DHIVYA R

Source Code: https://github.com/dhivyar/SteamGames

Context: Steam is the world's most popular PC Gaming hub with over 6000 games and a community of many millions of users across the world. Steam's gaming collection includes games from many different genres from blockbusters to small Indie titles.

Problem Statement: Steam provides basic discovery tools for games like filtering by genre, by platform, by free to play option, by VR and so on. Steam also showcases a list of featured games (The developer or the publisher could pay a premium to get their games on the featured list for a specific period) and Steam also shows games with offers and deals. These are available to both Steam visitors and Steam users. The problem statement is as follows:

a) There is an inherent need for recommendation engines because there are over 6000 titles and it's impossible for users to pin- point and discover games without tools aiding them. These days, more emphasis is placed on the product to find relevant games for the user than the user searching for it. Study has also shown that users' attention span on the internet is ever decreasing, so there is an increasing risk of drop off (from the product) if targeted recommendations are not dynamically given to the user.
b) There is also an inherent need for personalized recommendation engines, with an added emphasis on personalized because there are many millions of users on Steam and each with different tastes and preferences. Having stale and untargeted recommendations (like basic filters) for all users will result in poor conversions.
c) Enabling personalized recommendations will also improve the product experience for the users and result in higher user engagement.

Approach:

Summary: Here, the first step is to identify where the recommendation engine will run in the product lifecycle. The product flow starts with visitors or users (visitors who have created accounts). Users could have purchased games earlier and could have also played those games earlier, but it should be assumed that we may or may not have this information depending on the user's lifecycle with Steam. So, the recommendation engine should run at the very beginning of the product flow to cater to all users irrespective of their purchase and play history.

The best approach to solve this problem would be to build a learning to rank model which will predict the probability of a user playing the games in the hub. So, for each user, there will be as many probabilities as the number of games. The underlying assumption with this approach is that Steam wants to maximize people playing the games and not just purchasing them. This probability will be used to rank the games and offer as recommendations. Additional conditions need to be written on top of the model, like do not recommend what is obvious or what the user will find anyways (As in: Not recommend Counter Strike:

Global Offensive game to users who have played Counter Strike: Source) and do not recommend if already the game is already purchased by the user.

Collaborative filtering is another technique, although it is easier and requires less engineering to implement in production, it has the following deal breakers: On an average, users only purchased (and therefore interacted) with only 0.019% of games this makes the interactions sparse and also for every new game and every new user, the interactions grow and hence the algorithms change.

Data Manipulation, Feature Engineering and Bi-Variate Analyses: The dataset available has user events which include user IDs, game titles purchased, play/ purchase behavior and hours played. The initial step is to create purchase and play flags by aggregating user along each game and then expanding the dataset to include games from the hub not purchased by the user.

The signal to noise ratio is approaching 0 for this data set, as user/ game combinations with an associated purchase or play are very rare events. So, to correct the class imbalances on this dataset, under sampling technique is employed, where 100% of signal data (minority event) is taken and a random sample of noise data (majority event) is taken. So, this resulting dataset has the following base rates:

Purchase Rate: 22.09%; Play Rate: 13.19% and Play when Purchase Rate: 59.69%.

User profile variables like PII, gender, country, avatar, channel, device, platform of the user, amount in Steam wallet are needed but are not available in this study. So, **user behavior variables** and **game level variables** are to engineered. The following is the list of top games in terms of hours spent on playing the game:

	game_title	**HoursSpent**
1	dota2	981684.6
2	counterstrikeglobaloffensive	322771.6
3	teamfortress2	173673.3
4	counterstrike	134261.1
5	counterstrikesource	96075.5

Using these, behaviors like: Number of purchases made, PurchasedDota2, PurchasedCounterStrikeGlobalOffensive, PurchasedTeamFortress2 and so on are engineered.

Steam API renders information about various features of all the games, the variables thus engineered are: NoAgeRestriction, SixteenPlusAge, MetacriticScore, ScreenshotCount, SteamSpyOwnersInMillions, IsFree, PurchaseAvail, Platform, CategorySinglePlayer, CategoryMultiplayer, CategoryCoop, CategoryMMO, CategoryIncludeLevelEditor, GenreIsIndie, GenreIsAction, GenreIsAdventure, GenreIsCasual, GenreIsStrategy, GenreIsSimulation, GenreIsAction, GenreIsEarlyAccess, GenreIsFreeToPlay, GenreIsSports, GenreIsRacing, GenreIsMassivelyMultiplayer, PriceFinal, PriceInitial, French, German, Italian, Spanish, Russian, Korean, Japanese. The following is a partial bi-variate study for these variables:

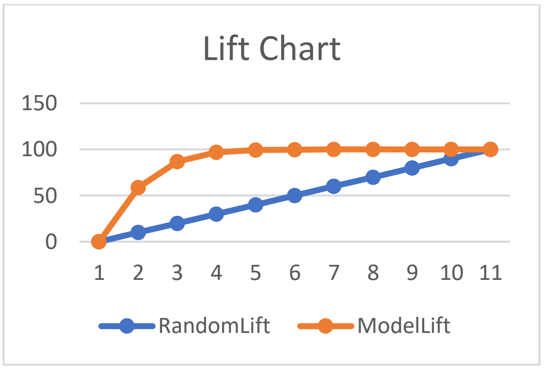
Sample Bi-Variates	**Levels**	**Play Rate**	**Description**
NoAgeRestriction	0	26.52%	Age restricted games are generally played more
	1	11.50%	
Metacritic Bucket	[0,50]	6.27%	Games with high Metacritic score are played more
	(50,100]	21.93%	
Screenshot Bucket	[0,5]	7.56%	Games with more screenshots are played more
	(5,10]	8.11%	
	(10,15]	18.30%	
	(15,200]	20.76%	
Platform	Only Windows	9.25%	Games available in all 3 platforms are played more
	Windows+Linux	18.38%	
	Windows+Linux+Mac	21.91%	

	Windows+Mac	8%	
	0	11.50%	Games with level editores are played more
IncludesLevelEditor	1	29.30%	
	0	7.25%	Games with action genre are played more
GenreIsAction	1	18.62%	
	Free	22.45%	Free games have high patronage and games priced less than $10 have less patronage (Possibly indicative of the quality of the game)
	(1,5]	3.16%	
	(5,10]	9.22%	
	(10,20]	18.44%	
	(20,50]	19.38%	
PriceBucket	(50,100]	25.41%	
	0	6.64%	Games available in multiple languages are played more
InFrenchToo	1	20.43%	

Modeling: All variables are used as inputs and a first cut model is built using Logistic Regression or XGBoost Machine Learning Algorithms. XGBoost scores highly on accuracy while Logistic Regression models score highly on ease of implementation. Here's a sample XGBoost model performance: The model captures 96.82% of all plays by 30% user game combinations as seen in the Rank Ordering Table. There is also a lift in slope for the first few deciles due to the model compared to the random lift as seen in the Lift Chart.

Confusion Matrix		ActualPlays			
		1	0		
PredictedPlays	1	17881	5773	Pos Pred Value	0.9445
	0	10669	181675	Neg Pred Value	0.7559
		Sensitivity	Specificity	Accuracy	
		0.9692	0.6263	0.9239	

Samples	CountOfUserGameCombinations	CumulativeCountOfUserGameCombinations	CountOfPlays	CumulativeCountOfPlays	UserGamesComboShare	CumulativeUserGamesComboShare	PlaysShare	CumulativePlaysShare
sample.1	21600	21600	16774	16774	10	10	58.75	58.75
sample.2	21600	43200	8063	24837	10	20	28.24	86.99
sample.3	21600	64800	2804	27641	10	30	9.82	96.82



Implementation: The model needs to be iterated to produce the same results over the top 12/15 most important predictors and it also needs to be validated over an out of time dataset to observe overfitting and regularized accordingly. The manual prediction scores for each node (as a spreadsheet) in the XGBoost model can be handed over to Dev team for implementation in production. On top of the recommender model, it is essential to write conditions where the model will be over-ridden. Few of these conditions are mentioned in the exceptions to rules column below.

Sample Recommendation for One User: Let's say Steam wants to recommend 4 games to this user, here we rank the games using the prediction probability to play and adjust this ranking basis the exceptions to this ranking. So, the recommendations for this user will be the games under adjusted recommendations.

Game	Prediction	Purchased	Rank	AdjustedRecommendations	Exception to Rule
counterstrikeglobaloffensive	0.6435	1	1	-	Do not recommend if already purchased
teamfortress	0.4635	0	2	1	None
counterstrike	0.2198	0	3	-	Do not recomment what is obvious (since he has already purchased counterstrikeglobaloffensive)
portal	0.1938	0	4	2	None
portal2	0.1763	0	5	-	Do not recommend very similar games: portal and portal 2
alflife2lostcoast	0.1153	0	6	3	None
totalwarshogun2	0.0937	0	7	4	None

Model Improvement: Now this model is also prone to becoming stale if only user profile level and game level variables are used, so in order to avoid this situation more and more user behavior level variables need to engineered for better models. This can come users' browsing history, web events associated with users' navigation between web pages and subsequent actions.

Evaluation Metrics: After implementation, an A/B experiment can be conducted in 70:30 or 90:10 ratio (risk averse) of users for a specified period and conversions can be observed. If the conversions and ratios for the recommendation engine are statistically higher than that of the default treatment, then the approach can be implemented for 100% of users.

Treatment	Visits or Logins	Purchases	Plays	Plays/ Visits	Plays/ Puchases
Default					
Variation					

Infrastructure Required: RStudio Server hosted on a 32 GB RAM (or a minimum of 16 GB, but 32 GB preferred) Ubuntu/ Debian server box running remotely (Assuming 2/3 people would be parallelly using the box) and accessed through a browser or RStudio running on a AWS EC2 instance with 32 GB memory like m5.2xlarge (or a minimum of 16 GB).

Packages Needed (Optional- Can be done at user-level): plyr, dplyr, data.table, lubridate, XGBoost, caret, tidyr, stringr, httr, jsonlite, RMarkdown..

GitHub enterprise version with private team repos for code sharing and documentation.