Steam Game Recommendations

Christopher Gomes,[1] Steven Horn[2]

Northeastern University[1,2]
gomes.chri@northeastern.edu,[1] horn.s@husky.neu.edu[2]

Abstract

With the prevalence of online shopping and increased content on the web, there has been a significant focus on tailoring these online experiences for users. Recommendation systems have grown to become one of the prominent ways that this is done. We took multiple implicit recommendation models and applied them to the task of video game recommendation based off of user play history. These models include: Alternate Least Squares, Bayesian Personalized Ranking, Logistic Matrix Factorization, and Neural Collaborative Filtering. These models were compared based on how accurately they recommended games and the diversity of their recommendation set. We found that some of the models tested provided good recommendations for the users in the dataset. With more modifications, data, and tests, these models could have the ability to provide even better recommendations for a wider variety of users and games.

Introduction[1]

Similar to other machine learning models, recommendation systems are present in most of the services that customers use everyday. There is just so much content online now and recommendation systems provide a great way for organizations to tailor this content to each user that they interact with. This has led to a surge in demand for recommendation systems, which in turn has led to a significant increase in research being done on these models. All of this research has led to the development of many recommandation models that are used by a variety of companies.

Our goal was to implement and apply some of these models to see how they would perform in recommending games to Steam users. In our comparison, we included four different recommendation system models: Alternate Least Squares (ALS), Bayesian Personalized Ranking (BPR), Logistic Matrix Factorization (LMF), and Neural Collaborative Filtering (NCF). ALS and LMF are two models that have been popularized by major streaming platforms, such as Spotify and Netflix, and have been used extensively as a result. NCF is a more recent model that tries to improve on ALS and LMF by using neural networks. Hopefully, our implementations of and experiments on these models on Steam user play history can provide useful insights into incorporating recommendation systems into new domains.

Background

Recommendation systems in their basic form consist of users, items, and an interaction/rating metric, where the user provides a rating for the item or there is a measure for how the user used the item. Therefore, datasets used to train these models will typically be a table where each row is an instance of a user identifier, an item identifier, and a metric to quantify the relationship between the two. The goal of the recommendation system is to then learn from these instances and use that learning to provide more items for a given user that they have not tried yet.

There are two types of values that can represent the feedback between the user and item: explicit and implicit. Explicit feedback is when a user provides a response about the item that clearly states their thoughts on the item. There is no inference about how the user feels and the user is making a cognisant declaration about the item. Common examples of explicit interactions include a user's review of an item or a like/dislike system, where the user can report whether they liked or disliked something they bought, watched, or listened to. This type of feedback is extremely useful, because it provides an authentic representation of the user/item interaction, and it allows for positive and negative feedback. Unfortunately, this type of feedback is

more difficult to come by since it requires the user to take additional action to provide this response to the system.

Implicit feedback on the other hand is any interaction between the user and item that is measurable. In this case, the user is not openly telling the system what it likes or doesn't like. The system is just using information about the actions that the user has with the items to infer if the user liked or disliked an item. Some examples of implicit feedback include reports on the amount of time that a user watched, played or listened to something. This type of feedback is normally more common than explicit feedback, because it does not require any additional effort by the user. Unfortunately, implicit feedback can suffer from the fact that the response being reported may not completely reflect whether a user likes an item or not, and typically there is a lack of negative feedback, since you cannot have negative amounts of interaction. There are models that have been created to address some of the shortcomings of implicit feedback. Some examples of these models include: Alternate Least Squares, Bayesian Personalized Ranking, and Logistic Matrix Factorization.

The data we worked with was implicit feedback. It contained instances that provided how long a Steam user played a specific game.[2] We used this data to train and compare each of the models mentioned above. In addition, we also implemented a Neural Collaborative Filtering model (He et. al. 2017).

Implicit Models

For the first three implicit models, we used a Python library called Implicit.[3]

Alternate Least Squares

Alternating Least Squares (ALS) is a matrix factorization model based on the algorithm described in "Collaborative Filtering for Implicit Feedback Datasets" (Hu et. al. 2008), with some optimizations described in "Applications of the conjugate gradient method for implicit feedback collaborative filtering" (Takacs et. al. 2011).

The general idea of matrix factorization is that we are given a large user-item matrix, R, where each cell represents the user's implicit feedback on the item. This could be whether the user interacted with the item, how many times the user interacted with the item, or how long the user interacted with the item for, depending on the dataset. In our case, it was how much time the user spent playing the game.

This matrix is then factored out into a user factor X and an item factor Y. We essentially want to find a vector $x_u \in R^f$ and a vector $y_i \in R^f$ for each user u and item i, such that $p_{ui} = x_u^T y_i$, which represents the user's preference for the item. It is worth noting that the paper defines r_{ui} as the amount that the user consumed the item (in our case, the amount of time the user spent playing the game), and p_{ui} as 1 if $r_{ui} > 0$ and 0 if $r_{ui} = 0$, with a confidence of $c_{ui} = 1 + \alpha r_{ui}$.

We want to optimize these values not only for the user-item interactions that we have, but also for those that we have not observed, given our confidence levels. Thus, we compute the factors by minimizing the cost function:

$$min_{x*,y*} \sum_{u,i} c_{ui}(r_{ui} - x^T_u * y_i) + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

This cost function is non-convex and contains a huge amount of terms (the number of users times the number of items). The paper proposes an alternating-least-squares optimization process where we alternate between fixing the item factor and just using the user factor, then fixing the user factor and just using the item factor. This makes the cost function quadratic, which makes it easier to find the global minimum.

Finally, we recommend to the user the items that have the largest values of $p = x^T y$, taking confidence into account.

Bayesian Personalized Ranking

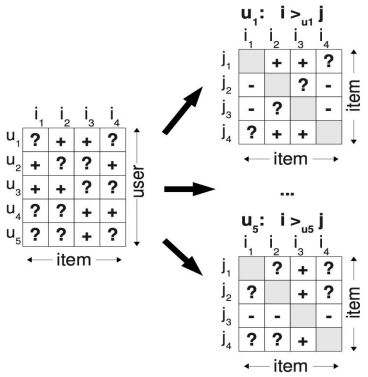

Figure 1: How to find (u, i, j) for each user and item-pair (Rendle et. al. 2014).

Bayesian Personalized Ranking (BPR) is a model based on the algorithm described in "BPR: Bayesian Personalized Ranking From Implicit Feedback" (Rendle et. al. 2009). BPR is different from most collaborative filtering models in that, rather than finding relationships between users and

[2] https://www.kaggle.com/tamber/steam-video-games/data
[3] https://implicit.readthedocs.io/en/latest/index.html

items (e.g. p_{ui} in ALS), it finds relationships between a user and two items. It works to find triples (u, i, j) such that user u prefers item i over item j.

In Figure 1 above, you can see how a user's preferences are first found. If a user has interacted with item 1 and not item 2, we say that the user prefers item 1. If the user has interacted with both or neither, we cannot say whether the user prefers one over the other.

A Bayesian formulation is then used in order to optimize the parameter vector of an arbitrary model class:

$$p(\Theta| >u) \propto p(>u |\Theta)\, p(\Theta)$$

where Θ is the parameter vector and >u is the desired but latent preference structure for user u. Without going into extreme detail, p(>u |Θ) can be found by $\sum_{i,\ j}$ p(i >u j | Θ), and p(i >u j | Θ) := σ(x_{uij}(Θ)), where σ is the logistic sigmoid function. Finally, p(Θ) is a normal distribution: p(Θ) ~ N(0, $\sum_{\Theta}$).

Using the arbitrary model class (Implicit uses ALS), we find x_{uij} for each triple: x_{uij} := x_{ui} – x_{uj}. We use this information to recommend the most preferred items to the user.

Logistic Matrix Factorization

Logistic Matrix Factorization (LMF) is based on the algorithm described in "Logistic Matrix Factorization for Implicit Feedback Data" (Johnson 2014). As another matrix factorization model, the setup for LMF is very similar to that of ALS: we factor out a matrix R into user factor X and item factor Y. The big difference here is that we are trying to calculate the probability that user u interacts with item i:

$$p(l_{ui} |x_u, y_i, \beta_i, \beta_j) = \frac{exp(xi\ yi + \beta i + \beta j)}{1 + exp(xu\ yi + \beta i + \beta j)}$$

β_i and β_j are biases for the user and the item in order to normalize behavior across users and items. For example, maybe some users only interact with a very small portion of the items, or maybe some items are very popular with many users. This model also uses a confidence metric of $c = \alpha r_{ui}$.

To get the values of X, Y, and β, we want to maximize the log posterior: $\text{argmax}_{X,\ Y,\ \beta}$ log p(X, Y, β | R). The right-hand side of this equation is very long and simply unnecessary to detail in this report.

Neural Collaborative Filtering

Neural Collaborative Filtering (NCF) is the combination of matrix factorization and a deep neural network to make item recommendations for a user. The concept and original structure is described in a paper titled "Neural Collaborative Filtering" (He et. al. 2017). There are two parts to this model: a neural network and the matrix factorization. The neural network and matrix factorization sections are trained separately on the user and item instances with the output being if there was or wasn't an interaction between the two. The outputs of these two parts are then combined and an output layer is trained to make the final predictions.

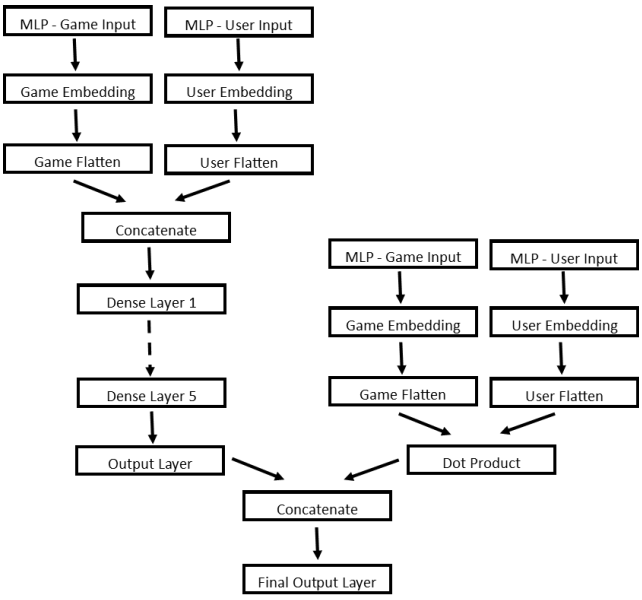

Figure 2: NCF structure for Steam game recommendations.

The neural network portion of our NCF model consisted of ten layers as shown on the left side of Figure 2. The first three layers create and format the latent vectors for each user and each item. The creation of the latent vectors resides in the embedding layer. The length of the vectors (number of latent factors) is chosen when the model is created. This length will be referred to as "k" for the remainder of this section. After the latent vectors for the users and the items are created and flattened, they are concatenated together into one vector.

The resulting concatenated vector is then the start of a neural network of five dense layers, which start at a length of 4*k neurons and are consistently halved until a length of (¼)*k neurons. An output layer at the end of these dense

layers gives the final output. The first dense layer is different then the following four in that it contains a batch normalization layer, which rescales the data. The five dense layers are L2 normalized and have a 20% dropout to help reduce overfitting. All of the dense layers also use a relu activation function and the final output layer uses a sigmoid activation function. As recommended by Xiangnan He (2017), this neural network was trained with an adam optimizer.

The matrix factorization part of the NCF model is similar to the Logistic Matrix Factorization and Alternate Least Square models and is shown on the right side of Figure 2. Its first three layers are identical in structure to the neural network section, and the model is designed so that the number of latent factors in the matrix factorization side can be different from the neural network side. The dot product of the user latent vectors and the item latent vectors is then found. The result of the dot product is the output.

Lastly, the NCF model concatenates the outputs of the neural network and matrix factorization parts. The concatenated values are then the input to one final output layer that gives the final prediction. This model is then trained again to get weights for this final layer using the standard SGD optimizer (He et. al. 2017). In addition, to help with implementing this model in TensorFlow, we referenced one tutorial to help with the neural network[4] and another one for the matrix factorization.[5]

Steam Data

Source

We obtained the data for our experiments from Kaggle.[6] The original data set contained 200,000 instances of user id, game name, whether it was a play or purchase behavior between the two, and the amount if it was a play behavior. We focused on just the play interactions in the dataset. Most games were played by less than 100 users and very few games were played by more than 300 users as shown in Figure 3.

Preprocessing

After filtering the data to only include interactions that were plays, we then did a few more cleaning steps. First, we kept games that were played by at least ten users and no more than 300 users. The idea being that we didn't want to recommend games that we didn't have much information about, but we also didn't want to recommend over popular games. The latter was done for two reasons: extremely popular games can overshadow other games in model training, which can cause the model to predict the most popular games instead of the games the user will like the most, and there really is no point recommending really popular games, since most people would know about them anyway. The 300 person threshold was chosen, since games above it were outliers compared to the rest of the data. We then removed any user that hasn't played at least two games, since this is the minimum number needed to be able to train and recommend.

The rest of the cleaning steps focused on normalizing and formatting the data. Specifically, we min-max normalized the play amounts across each game. The goal was to have this keep games that have longer story lines from being overvalued compared to games with shorter story lines. The user and game identifiers were then replaced with new ids that started at zero and incremented by 1. To be able to get specific game names later, we saved a CSV that could convert the new game ids to the original game names. At the end of the preprocessing stage, we used a leave-one-out strategy to create a test set by randomly choosing one game per user, and the rest of the games went into the training set.

For the models that were implemented using Implicit[7], the training data was then converted to a matrix of users by items. Values that were in the training set stayed the same, and any interactions that were not present in training were filled in with a zero to represent that there was no interaction between the user-item pair. The resulting matrix had a sparsity of about 97%, which means only around 3% of the user-item pairs had an interaction. For the NCF model, user-item pairs were randomly generated and added to the training set, if they were not already present. These randomly generated user-item pairs became the non-interacted instances for the training dataset. This resulted in a similar sparsity of about 97%.

[4] https://petamind.com/mlp-for-implicit-binary-collaborative-filtering/

[5] https://petamind.com/build-a-simple-recommender-system-with-matrix-factorization/

[6] https://www.kaggle.com/tamber/steam-video-games/data

[7] https://implicit.readthedocs.io/en/latest/index.html

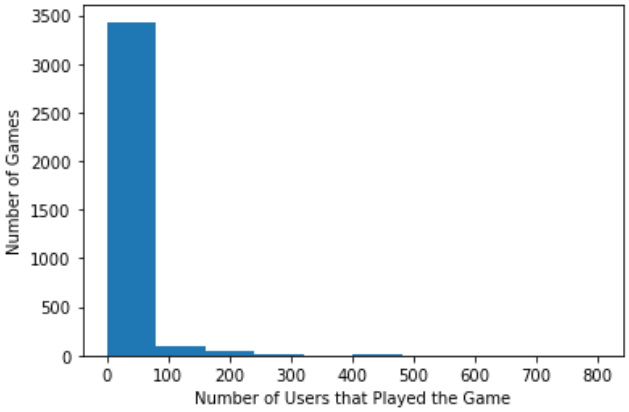

Figure 3: Histogram of number of users that played different Steam games.

Evaluation

Methodology

There were two metrics that we tracked to compare the four models. The first consisted of having each of the models recommend five games per user. The recommendation was considered successful if the game that the user played in the test set was one of the five games that the model recommended for that user. We then calculated the percent of users that had their game successfully predicted in the test set. As a baseline, if the model just randomly picked five games to recommend to each user, the probability that one of the games is the game that the user played in the test set would be 5 out of 1,033 games or 0.48%.

The second metric was to measure the diversity of the games that each of the models were recommending. We measured this by taking the number of unique games that the model recommended to all of the users and divided it by the number of unique games in the training dataset. This metric gave us some insight into whether models were only recommending a small number of games. If any of the models were, this could be a sign that those models were biased toward just recommending popular games. For all of the models except for NCF, the model training and testing was performed 100 times and the average for each metric was reported. Due to hardware limitations and training times, the NCF runs were not done in multiple runs.

Results

Each model had parameters that were fine-tuned to ensure that the most accurate results were being acquired. All of the models performed better than the baseline, most of them by a very large margin.

The ALS model was able to accurately predict a game that a user played from five recommended games for 22.79% of the users. It was also able to recommend 69.5% of the games that were in the training set. This is obviously much better than the previously defined baseline. For this and the other implicit models, we reached these numbers by training the data one hundred times, calculating the accuracy each time, and calculating the average results. For the ALS model, we tuned the parameters to have 128 latent factors, a regularization constant of 0.05, and ran for 50 iterations while training.

The BPR model was able to accurately predict a game that a user played from the five recommended games for 23.86% of the users. It was also able to recommend 95.3% of the games in the training set. Again, this was much better than the baseline, but this model also performed slightly better than ALS for accuracy and significantly better in percentage of games recommended. For this model, the parameters chosen were 128 latent factors, a 0.01 learning rate, a regularization constant of 0.05, and 200 iterations.

The LMF model accurately predicted a game that a user played from the five recommended games for only 1.58% of the users and only recommended 0.48% of the games in the training set. This model did not perform significantly better than the baseline, and it performed significantly worse than the other models. We suspect that for this model to run correctly in Implicit, the data must be set up in a specific way that was not clearly defined, because by all accounts this should be a rather accurate model. The only parameter that was tuned for this model was the amount of latent factors being 128 like the other models.

The NCF model that was created and tested performed well compared to the other models that were evaluated. The NCF model was able to be configured to accurately predict a game that a user played from five recommended games for 26.03% of the users. For this same configuration, the model recommended 53.82% of the games that were in the training set. Again, to the baseline that we set for these experiments, these are significant improvements, and is even an improvement in accuracy compared to the other models. Due to limitations in the hardware that we had we were not able to do as many test runs as we would like, but these are solid preliminary numbers.

The number of latent factors was the main thing that was configured and experimented with. There were two ways that different variations of latent vectors were tested. The first involved keeping both the neural network and matrix factorization latent vector sizes equal and increasing them together, see Figure 4A. The second involved keeping the neural network latent vector size at 16 and increasing the latent vector size for the matrix factorization, see Figure 4B. In both tests, the metrics increased with an increase in the number of latent factors. Increasing the size of just the matrix factorization latent vectors, compared to increasing the neural network ones as well, produced marginally better results in the model recommending games to users. Moreover, it recommended a wider variety of games as well compared to increasing both.

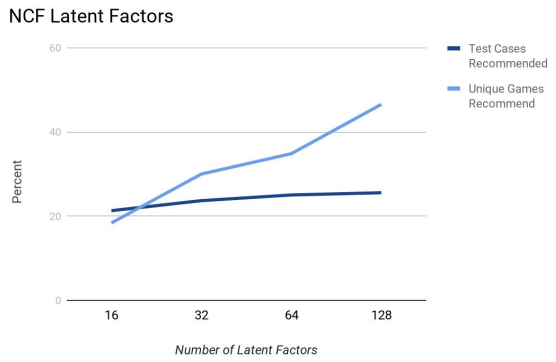

Figure 4A: Results from increasing the number of latent factors for both the neural network and matrix factorization

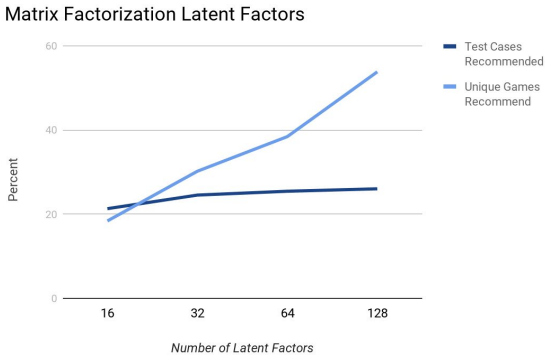

Figure 4B: Results from increasing the number of latent factors for just the matrix factorization.

As a final note, we also combined the results of ALS and BPR, the two best performing Implicit models, into a set of 10 recommended games and calculated the accuracy of that set in predicting the user's test set game. We found that it had an accuracy of 36.48%. We thought that this was noteworthy because it shows that while BPR performed slightly better than ALS on their own, it is not necessarily because BPR accurately predicted the games for the same users as ALS, plus a few more. This calculation shows that it is more likely that ALS and BPR recommended different games for each of the users.

Conclusion

After running tests with all of the models, we determined that the Neural Collaborative Filtering model was the most accurate, followed closely by Bayesian Personalized Ranking and Alternating Least Squares, with Logistic Matrix Factorization far behind. In terms of percentage of games in the training set recommended by the model, BPR performed the best by a wide margin, leading ALS by almost close to 30%, followed by NCF and with LMF again far behind.

There is more that we would have liked to do with this project given more time and resources. One aspect of the data cleaning that we could look into is the issue of overlapping game names. For example, there is one listing for Call of Duty: Black Ops, and one for Call of Duty: Black Ops - Multiplayer. We could also try to factor purchase data into the training set. As stated above, the original data set had a Behavior column of either Purchase or Play, where Purchase indicated that the user had bought the game but had never played it. We took out these rows and only considered games that users had played, but there may be some way to factor the Purchase behavior into our calculations. Finally, if we had more resources in terms of hardware, we could run more tests and get more accurate results, especially with the NCF model.

References

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T. 2017. Neural Collaborative Filtering. Paper presented at the 26th International Conference on World Wide Web. Perth, Australia, April 3-7.

Hu Y.; Koren Y.; and Volinsky C. 2008. Collaborative Filtering for Implicit Feedback Datasets. Paper presented at the 2008 Eighth IEEE International Conference on Data Mining. Pisa, Italy, December 15-19.

Takacs, G.; Pilaszy, I.; and Tikk, D. 2011. Applications of the conjugate gradient method for implicit feedback collaborative filtering. Paper presented at the 5th ACM Conference on Recommender Systems. Chicago, IL, October 23-27.

Rendle, S: Freudenthaler, S.; Gantner, Z; and Schmidt-Thieme, L. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. Paper presented at the 25th Conference on Uncertainty in Artificial Intelligence. Montreal, Canada, June 18-21.

Johnson, C. 2014. Logistic Matrix Factorization for Implicit Feedback Data.