# Recommender Systems for Online Video Game Platforms: the Case of STEAM

Germán Cheuque
IMFD &
Pontificia Universidad Católica
Santiago, Chile
gacheuque@uc.cl

José Guzmán
Pontificia Universidad Católica
Santiago, Chile
jaguzman6@uc.cl

Denis Parra
IMFD &
Pontificia Universidad Católica
Santiago, Chile
dparra@ing.puc.cl

## ABSTRACT

The world of video games has changed considerably over the recent years. Its diversification has dramatically increased the number of users engaged in online communities of this entertainment area, and consequently, the number and types of games available. This context of information overload underpins the development of recommender systems that could leverage the information that the video game platforms collect, hence following the trend of new games coming out every year. In this work we test the potential of state-of-the-art recommender models based respectively on Factorization Machines (FM), deep neural networks (DeepNN) and one derived from the mixture of both (DeepFM), chosen for their potential of receiving multiple inputs as well as different types of input variables. We evaluate our results measuring the ranking accuracy of the recommendation and the diversity/novelty of a recommendation list. All the algorithms achieve better results than a baseline based on implicit feedback (Alternating Least Squares model). The best performing algorithm is DeepNN, the high order interactions are more important than the low order ones for this recommendation task. We also analyze the effect of the sentiment extracted directly from game reviews, and find that it is not as relevant for recommendation as one might expect. We are the first in studying the aforementioned recommender systems over the context of online video game platforms, reporting novel results which could be used as baseline in future works.

## KEYWORDS

Recommender System, Factorization Machines, Deep Neural Networks, Deep Factorization Machines, Novelty, Diversity

**ACM Reference Format:**
Germán Cheuque, José Guzmán, and Denis Parra. 2019. Recommender Systems for Online Video Game Platforms: the Case of STEAM. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3308560.3316457

## 1 INTRODUCTION

According to the European Mobile Game Market, in 2016 over 2.5 billion people spent part of their time playing video games. The huge number of adepts have made the video game industry one of the most valued in the world. In 2017 alone this industry grew by 10.7% in earnings, achieving over $116 billion. One of the biggest reasons behind this success is the high diversification that this industry has had over latest years. We can now find multiple platforms dedicated to games, an enormous number of genres and game' categories and even platforms that provide social interaction between players. This adaptability has generated a huge number of items with different attributes that are able to attract diverse users. An example of adaptation is the STEAM[1] platform, a company dedicated to the digital distribution of video games which allows users to buy, see, share opinions, play on-line, play off-line, compete and cooperate across the platform and games. Over 10 million people connect to the STEAM server each hour to play video games. However, the fact of having such a variety of products and so many users, makes it difficult to choose a particular new game some user like. Also, according to STEAM registries of 2014, about 37% of games purchased have never been played by the users who bought them. This context creates a need for recommender systems, which are systems able to make relevant personalized suggestions [8], alerting users to unknown games as well as new releases.

Playing video games is a recurrent activity, in this sense that is it more similar to music listening than to movie watching. A preferred game is played many times, but users also want to discover new games. This represents a double challenge for the industry: the need for video games which encourage users to return, as well as helping users find novel games which will be consumed as much as those already liked. Our intuition is that the great number of features available in the STEAM platform allows us to explore interactions between items' features, as well as between users and their preferences. These relations make it reasonable to argue that it is possible to face the second challenge. Being able to develop an algorithm which addresses the aforementioned challenges could generate great benefits for the industry, the community of users and even for game developers, by predicting what the users want the most as well as promoting new releases.

To achieve these goals, we test three state-of-the-art algorithms based on different recommendation paradigms that can be used with different types of input data:

- Recommender System (RecSys) based on Collaborative Filtering (CF) that uses the ALS (Alternating Least Squares) algorithm for making recommendation based on implicit feedback [4]. We use it as a baseline.

[1] https://store.steampowered.com/


*WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA*

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3316457

- RecSys based on Factorization Machines [10] that benefits from a huge amount of data, context and features from users, items and purchases.
- RecSys based on FM and deep learning techniques (DeepNN and DeepFM [3]), where we introduce sentiment analysis to decode information contained on reviews.

We argue that feeding our models solely with the item preference history of users (implicit feedback [4, 7]) should be enough to make a good baseline recommendation model. Moreover, we think that adding content and contextual information could be effectively used by the FM and DeepNN models to improve ranking and novelty performance performance, due to interaction effects. However, the combination of both in a joint model (DeepFM) could either produce an improvement or rather be just redundant with respect to DeepNN.

Our contributions are the following: (i) We are the first in studying the aforementioned state-of-the-art recommender systems on the context of video games, reporting novel results to be used as baselines in future works, (ii) in addition to ranking accuracy, we report and analyze results about novelty and diversity of recommendations, which is critical in the online video game industry, and (iii) we are the first in analyzing the impact of using sentiment analysis of the textual reviews over the quality, novelty and diversity of recommendations of online video games.

This work is organized as follows: Section 2 presents a discussion of the state of the art, the studies that inspired our work. In section 3 we present the Materials (datasets) and the methods (recommendation models) used in this work. Section 4 shows the experimental methodology and evaluation metrics to measure the performance of our systems, while in section 5 we present a sensitivity analysis of parameters. Section 6 shows our most relevant results and finally, section 7 discusses our conclusions and ideas for future work.

## 2 RELATED WORK

Numerous are the studies that address the recommendation problem, however just a few have emerged in the video game industry. The video game purchase platforms or in the video game communities an interesting places to develop and test the potential of existing recommender systems. Despite this, there are some notable studies whose application could be relevant for the introduced context.

Bertens et al. 2018 [1], for example, tries to measure the performance of two recommender systems based on the idea of prediction of the next most likely item to buy. The paper works over the behavior of Japaneses card players of the game *Age of Ishtaria* taking into consideration their play time, daily progress and purchase history. They test two models: the first one is called Extremely Randomized Trees (ERT), which is a randomized version of the decision tree algorithm. This method has the advantage of being computationally efficient due to its high parallelization ability, and it also has the property of preventing overfitting. The second one corresponds to an algorithm based on deep neural networks, with the particularity of being built as a recurrent network because of the sequential property of data. Both algorithms show very similar results, with ERT being the one with better performance in both train time and scalability.

Quadrana et al. 2018 [9] propose a recommender system based on attention to a sequence of events as one of the best approximations to solve the recommendation task on the introduced context. They survey multiple developed models, considering different tasks and goals to achieve. On their discussion, they highlight the potential of neural networks for the recommendation task. Although their discussion is relevant to the video game industry context, they emphasize their implementations to be tested on the *in session* attention context. One relevant approximation of this type of application can be found in Wan et al. 2018 [13] where the authors study the so called *monotonically behavior chains* concept that is understood as a sequence of events that account for a more explicit preference that a user has about an item. This chain is constructed from implicit and abundant information to a more explicit and limited information. The novelty of the publication lies on the capability of introducing a new paradigm to face the recommendation task and being able to prove its potential over different datasets, including one about video games from STEAM. Their improvement in performance went from 1% to 28% with respect to *Most Popular* method, depending on the dataset.

Due to the context of networks and interactions that exist in video game platforms, recommender systems based on graphs could be very useful as a modeling approach. In this topic, Shams et al. (2016) [11] is one of the most recent studies. In this publication they introduce the method GRank to correctly model the priorities of users and to discriminate better the relevant connections between nodes.

In this work, we propose to develop a recommender system for games based on state-of-the-art techniques, by comparing the results of implicit feedback collaborative filtering [4], factorization machines [10] and deep neural networks [3]. Unlike previous work, our system leverages interactions between users and items by mixing up implicit information from users and features from items, considering implicit information as playtime and explicit information like opinions, within reviews. In addition, we evaluate beyond prediction or ranking accuracy, we use the metrics proposed by Vargas and Castells [12] to measure novelty and diversity, since these metrics are critical for the continuous and diverse releases in the game industry.

## 3 MATERIALS AND METHODS

### 3.1 Materials: The Dataset

To carry out the implementation of the mentioned recommender systems (discussed in detail on the next section) and evaluate its predictive ability we work over three datasets, obtained from the collection shared by J. McAuley[2]. The first one consists of the purchase history of Australian users of the STEAM platform, a database sorted by user, indicating for each one the list of purchased items with a small collection of metadata as the playing time. The second dataset contains the opinion that different users on the platform have about the available items, this opinion serves as a review for

[2]http://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data

the whole community. The third and last dataset gives us detailed information about different features of video games, such as the genres they belong to or their platform availability (as discussed in [5]). The data is collected between October 2010 to January 2018.

The different datasets are joined with the purpose of creating tuples of data associated with each pair of item and user. Table 1 shows a summary of final descriptive statistics. Each tuple consist of 32 features where we find abundant implicit information, as the playtime, and scarce explicit information as the *recommend* feature. As we can see, there is no general rating measure that could be used as a measure of how much a user likes or prefers an item. Because of this, our analysis and the model we consider is based on measures of implicit feedback or made from a combination of information that we consider useful in making recommendations. The wealth of information gives us a unique chance of testing different kinds of data and their explicit and implicit interaction, as well as their non-linearity.

Special attention is given to *playtime*, considering the importance of this variable reported in previous research on recommender systems [7, 14].

**Table 1: Summary of features.**

| Feature | Type | Description |
|---|---|---|
| *user_id* | Str | Unique identifier of a user |
| *item_id* | Str | Unique identifier of item |
| *count* | Int | Number of games purchased by the user |
| *playtime* | Int | Period of time that the game has been played by a user, in hours |
| *RecCount* | Int | Times that an item has been recommended |
| *Metacritic* | Int | Valuation of game, agreed to specialized critic. |
| *GenreIs* \|G\| = 13 | [Bool] | True if the item belongs to one of the 13 different game genres. |
| *Category* \|C\| = 8 | [Bool] | True if the game belongs to one of the 8 game categories, e.g.: multi player |
| *Platform* \|P\| = 3 | [Bool] | True if the game is supported by one of the 3 listed operative systems. |
| *recommend* | [Bool] | True if user recommends the item. |
| *review* | Str | Free text which reports the user opinion. |

About 5, 153, 209 of tuples are contained in our dataset with a total of 70, 912 different users and 10, 978 different items (video games) to choose from. These numbers tell us about a large sparsity of 0.66% seen registries. A large sparsity is not ideal for training a model of interactions between users and items. Because of this, we filter the dataset; we believe that densities in the order of 10% could be useful for our objectives. Our final dataset considers only items purchased at least 200 times and users with at least 100 purchased items. A summary of our final database is presented in Table 2. With the data filtering, we obtained a new dataset with a density close to 10%, reducing its original size to a half, but still of a considerable size to carry out experiments. We also analyze the distribution

**Table 2: Summary of statistics of the final database after cut.**

| Descriptive statistic | |
|---|---|
| Number of registries | 2,149,858 |
| Number of different users | 8,183 |
| Number of different items | 2,872 |
| Number of reviews | 9,823 |
| Dataset density | 9.14% |
| Average purchases per user | 262.72 |
| Average purchases per item | 748.55 |
| Average hours played per user | 161,317.9 |
| Average hours played per item | 476,727.9 |

of consumption in the dataset, in order to dismiss potential issues due to imbalance bias: too few users accounting for too many of the total transactions. Figure 1 shows that user bias is small: around 67% of them explain the 80% of tuples. However, Figure 2 shows that a small number of items (40%) are responsible for most records (80%). This situation encourages to further analyze our systems to ensure they recommend different kinds of items. To address this concern, we are going to use novelty and diversity metrics. Further details are discussed in section 4.

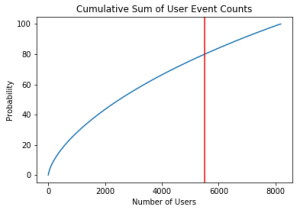


**Figure 1: Cumulative sum of contribution of users to the number of registries. Sixty-seven percent of users explain the 80% of data.**

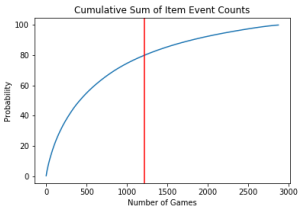


**Figure 2: Cumulative sum of contribution of items to the number of registries. Forty percent of items explain the 80% of data.**

**Implicit feedback data**. We will work on implicit feedback to learn user preferences, specifically, using playtime as our proxy measure. After preliminary analysis of the whole playtime distribution (see Figure 3), we proposed a limit of 5 hours as enough to differentiate between preference or not. This is indeed a strong

assumption, since the ideal would be knowing the usual playtime of each game as a more precise measure. But for the sake of simplicity, we leave this idea for future work. Figures 3 and 4 show distributions of playtime for the most active users and most purchased games.

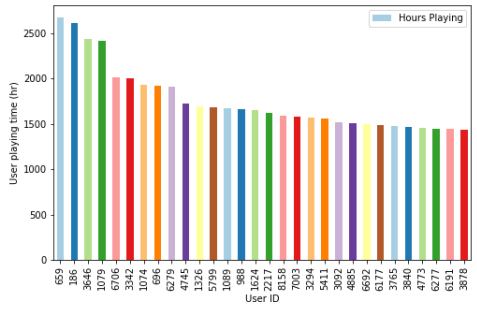


**Figure 3: Playtime of most active users.**

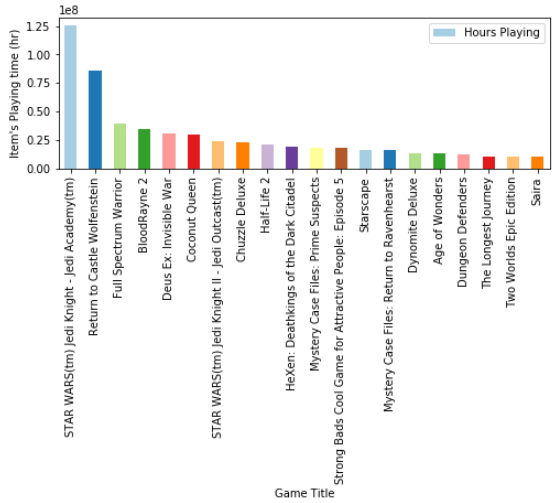


**Figure 4: Playtime of most purchased games.**

Figure 3 shows small differences on playtime among the most active users playing video games on STEAM. Meanwhile, most purchased games show bigger variations on playtime, where we find games widely played across users and also, games not played at all. When we see the average playtimes in table 2 we can infer another characteristic about the distribution of user and games. About 161,317.9 hours on average are played per user, while a total a 476,727.85 hours is the average time that an item is played on record history. Again we find that users have a more homogeneous distribution compared to items, for which a rather small fraction of items represent most interactions in the dataset. Although these numbers could suggest us that a low potential for making both accurate and diverse recommendation lists, when we apply over our records a threshold of 5 hours playtime, we find that 46% of database are over this threshold and, therefore, correspond to relevant items for recommendation task.

## 3.2 Methods: Recommender Systems Models

### 3.2.1 *Alternating Least Squares (ALS).*.

Under the diversity of models of matrix factorization, the ALS model stands out for being capable of working using implicit feedback [4].

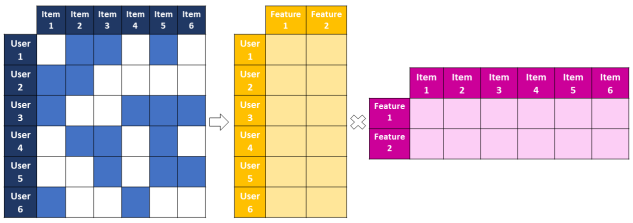


**Figure 5: Matrix factorization model representation.**

Figure 5 shows the idea under the factorization method, This is, to find a new representation for user ($x_u$) and items ($y_i$) interactions, so that a particular preference of an user about an item is given by the dot product of the latent factor representation of each of them. An innovation that introduces this algorithm analyzes the implicit feedback measure through the insertion of two new measures that represent the preference and confidence that exist behind the uptake of an item. Preference tells us if an item is consumed or not.

$$p_{ui} = \begin{cases} 1 & \text{if} r_{ui} > 0 \\ 0 & \text{if} r_{ui} = 0 \end{cases} \quad (1)$$

Where $r_{ui}$ is some measure of implicit feedback. The second measure stands for the confidence of that preference,

$$c_{ui} = 1 + \alpha * r_{ui} \quad (1)$$

Where $\alpha$ is a linear scale factor that set more importance to relevant items, above never played ones. The value $\alpha = 40$ is usually used as a result of the original paper [4]. Under these definitions, the search of latent factor for users and items is made by the optimization of the following loss function,

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

the optimization via least squares that give its name to the algorithm is visible. In this way, subsequent updates of latent factors are given by,

$$x_u = (V^T V + V^T(C^u - I)V + \lambda I)^{-1} V^T C^u p(u) \quad (3)$$

$$y_i = (U^T U + U^T(C^i - I)U + \lambda I)^{-1} U^T C^i p(i) \quad (4)$$

Some relevant parameters like the number of latent factors to use are selected through analysis in many iterations. In this article, we

used the ALS implementation found in the pyreclab[3] recommender library.

### 3.2.2 *Factorization Machines (FM)* .

Factorization Machines (FMs) are a type of model for recommender systems making latent factor models as easy to use as regression or SVM models [10]. FMs can deal with different types of inputs, from continuous to discrete variables, and more importantly, latent factor models. FMs can model interactions of different orders (order-n) between these variables [10]. In our case, $n = 2$ was defined because it has been widely used in this way as it delivers good results while maintaining a reasonable training time. With this configuration, the output is a prediction resulting from the linear interaction of the inputs (order-1) plus that of the latent factors, defined by the relationships between pairs of inputs (order-2), as observed in the equation 5.

$$y(x) := w_0 + \sum_{i=0}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j \qquad (5)$$

We decided to choose this model because it is a type of system that can scale quickly, has shown excellent performance when working with sparse datasets, and supports numerical input data of any type. These are all features that contribute to the recommendation process in a company like video games, where product variety and growth are critical factors in the process.

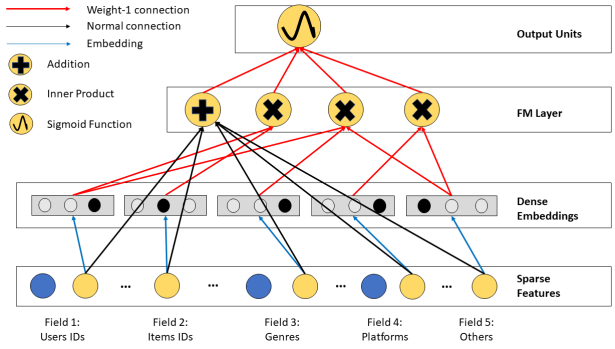


**Figure 6: Factorization Machine Architecture. Image adapted from [3].**

Figure 6 (as in [3] ) shows an example of the architecture of a factorization machine model and the Figure 7 (as in [10]) an example of the inputs and outputs that can receive.

### 3.2.3 *DeepFM*.

DeepFM is a newer model that seeks to take advantage of the versatility of Factorization Machines to model low-order interactions between input variables, with the ability to model deeper interactions of Deep Neural Networks (DNN) [3]. To do this, a DNN is implemented in parallel to the FM and the result of both is joined

[3]https://github.com/gasevi/pyreclab

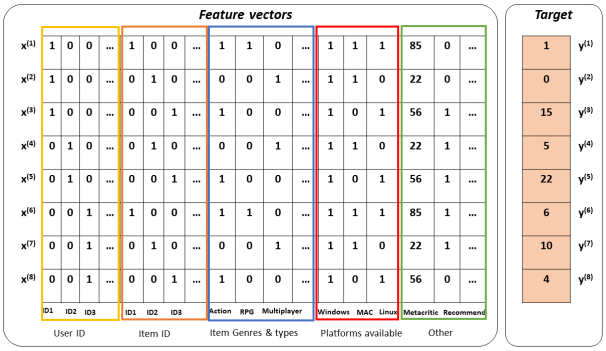


**Figure 7: Input example of features received by FM. Image adapted from [10].**

on a last node using a sigmoid function, making the output a prediction resulting from the interaction of several orders of inputs that represents the probability of whether or not a user belongs to a particular class defined with training tags. In our case, this class referred to whether the user liked the game or not.

We decided to classify the entries according to their type (user, item, category, platform, etc.) and to use an embedding of the entries that gives the same weight to each group, so that the effect of the "sparse" variables did not interfere in a significant way against the effect of the other variables. This embedding consisted in generating several matrices that allowed transforming the $N_i$ variables of each input class into a fixed K number defined as a model parameter. This idea was taken from the same paper [3] from where the model information was obtained, providing us with a large part of its implementation [4], which, for the most part, is made using the tensorflow library. Figure 8 (as in [3]) shows a representation of all the components of the model and how they are connected.

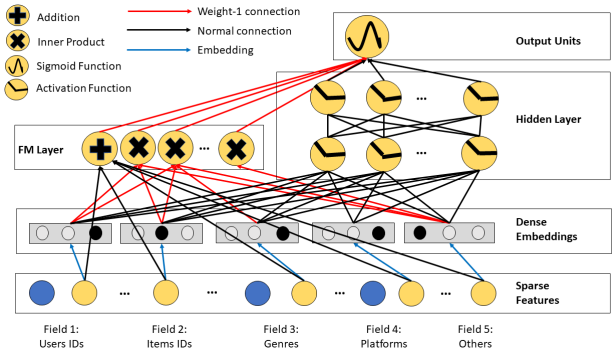


**Figure 8: Example of the DeepFM Architecture. Image adapted from [3].**

### 3.2.4 *DeepNN*.

The novel component of DeepFM is the deep neural network (also

[4]https://github.com/ChenglongChen/tensorflow-DeepFM

referred to by us as DeepNN) which analyzes in parallel the interactions between users and items. The deep component is a feed-forward neural network and is used to learn high-order interactions. A summary of this architecture is shown in Figure 9.

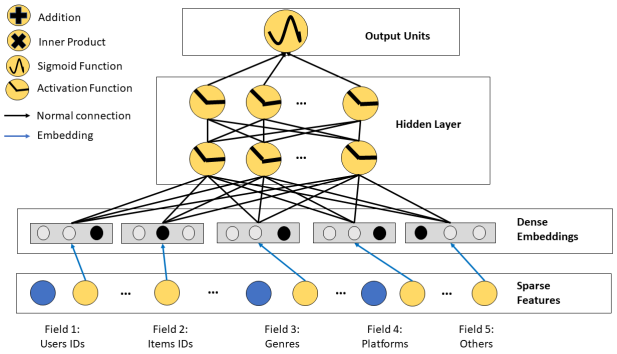


**Figure 9: Example of the DeepNN component of the DeepFM Architecture. Image adapted from [3].**

Its first component is an embedding layer that compresses the input fields to a low dimensional and dense real-valued vector. Notice that these fields can be of different lengths while embedding always returns a codification of same $K$ units of length. We denote the output of the embedding layer as:

$$a(0) = [e_1, e_2, ..., e_m], \tag{6}$$

where $e_i$ is the i-th field embedded from a total of m fields. These inputs are fed then to the hidden layers where the features interact. The forward process is:

$$a^{(l+1)} = \sigma(W^{(l)}a^{(l)} + b^{(l)}), \tag{7}$$

where $l$ is the layer depth and $\sigma$ is an activation function. The variables $a^{(l)}, W^{(l)}, b^{(l)}$ are the output, model weight, and bias of the l-th layer. Then a last dense vector is generated and these results are passed throughout a sigmoid function that returns a final output for the recommendation task.

### 3.3 Sentiment analysis

One of the most interesting features in our dataset corresponds to reviews. A review is a piece of text that encloses the opinion that a user has and gave. However, a piece of text is not useful itself to feed our models. To solve this, we propose sentiment analysis techniques to transform this opinion to a numeric measure that can give us a representation about how positive or negative the expressed idea is. Using it could help the confidence degree obtained from the explicit *recommend* feature, being also able to impute it when those values do not match. We experiment with three different tools.

First we use the open source algorithm *Tweetment*. This algorithm exists as a Python library and is capable of making binary sentiment classification to labels positively or negatively, represented by values 1 or 0, respectively. This algorithm is trained over a tweet database, achieving a F1-score of 69.02% on binary classification [6]. Another consulted algorithm was *SentiWordNet*, developed as an opinion mining application able to classify text in a continuous range [-1, 1], indicative of how positive or negative is the expressed idea. This algorithm works using a pre-trained dictionary that synthesize n-grams relations, giving a score about its positiveness, negativeness and objectiveness, without being exclusive [2]. Finally, we tried a model base on Neural networks and a word2vec embedding. This algorithm tries to pay attention on text over the game in question. However encountered two drawbacks in its development: we were able to make just binary classification and also, the implementation achieved a F1-score lower than Tweetment. We finally decided to work with SentiWordNet because of the advantage of continuous measure for the sentiment appreciation. This could be more appropriate for different comment context (jokes, etc).

## 4 EXPERIMENTAL METHODOLOGY

The experiments presented the following sections consist of the training and evaluation of the recommendation models. To ensure robustness we proceed to make randomized folds of the dataset (k-folding), applying cross validation techniques to average these scores. To train ALS algorithm, we use five folds while on FM and DeepFM we use only three. The testing set was also randomly obtained, however, because of our previous cut on dataset, we have items that are purchased at least 100 times, so we verify that at least ten of these purchases stay on test set so we try to predict the user's preferences is preference about them and verify what recommendations our models suggest.

To measure our algorithms performance we use several metrics. Training time is considered as a relevant measure to give us an idea about the scalability of the models. Also, we want to measure the Mean Average Precision (MAP) at the 10th top position in a list. So, Average Precision is defined as,

$$AP = \frac{\sum_k P@k * rel(k)}{\#relevant\_items} \tag{8}$$

and the Mean average precision measure is given by,

$$MAP = \frac{\sum_{u=1}^{n} AP(u)}{\#users} \tag{9}$$

Similarly we want to calculate the nDCG measure at the 10th position of the list. This measure tries to standardize the gain and utility of a recommendation list through the consideration of the relevance of items and its ranking position on the list (introducing a logarithmic discount). The definition of this parameter is a ratio between the Discounted Cumulative Gain at position k, given by:

$$DCG_k = \sum_{i=1}^{k} \frac{rel(i)}{log_2(1+i)} \tag{10}$$

and its ideal value at position k, also known as Ideal DCG. So the nDCG measure is given by,

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{11}$$

Finally we are interested in measuring how diverse and new are the recommendation made by our systems. As we have already discussed, a large number of new games are released each year, making these metrics very relevant. To measure these quantities we use similarity based relations between items from a recommendation list. Details are presented in [12].

$$Novelty(R|u) = \sum_{n,j \in u} disc(n)p(rel|i_n, u)p(rel|j_n, u)d(i, j) \quad (12)$$

$$Diversity(R|u) = 2 \sum_{k<n} disc(n)disc(k)p(rel|i_n, u)d(i, k) \quad (13)$$

The definitions introduce multiple discount factors, used as logarithmic ones. The d(i,k) term refers to the distance between pairs of items, this was calculated as a cosine distance. Finally the terms $p(rel|i_n, u)$ refer to the probability that a user $u$ prefers the item $i_n$, a factor that could be understood as a relevance. This measure is obtained directly from the output of recommender systems.

## 5 PARAMETERS SENSITIVITY ANALYSIS

With the purpose of obtaining the best results, we proceed to analyze the different tunable parameter. ALS algorithm has three parameters that we can change they: The number of latent factors used to represent the factorization, the number of epochs of training also refereed to as iterations; finally the $\lambda$ factor, used for regularization on equations 3 and 4. The sensitivity analysis is shown on the next figures.

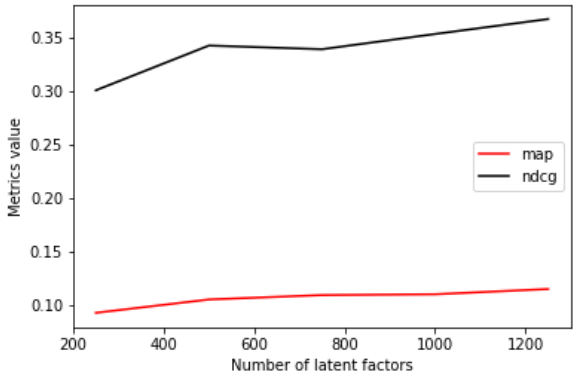


**Figure 10: Training with different number of latent factors.**

We finally work with the combination of the best found parameters. We set regularization to 0.01, a number of 500 latent factors privileging the training time against low improvements over 800. Also 300 iterations were used for training.

The deep learning models have two tunable parameters. The first one directly influences the net architecture, referring to the number of neurons in each layer. The second one is the batch size, that encloses the number of tuples that are analyzed together. The last one is crucial due its responsibility on the over-fitting and for being

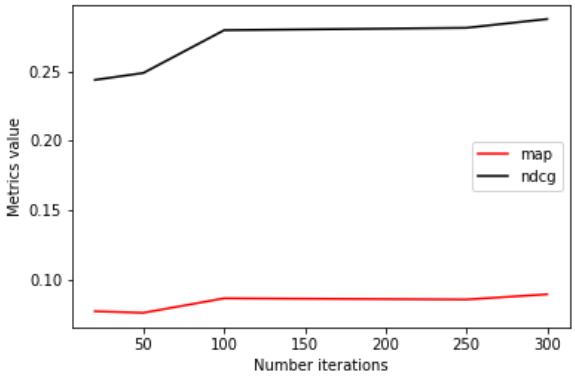


**Figure 11: Training with different number of iterations.**

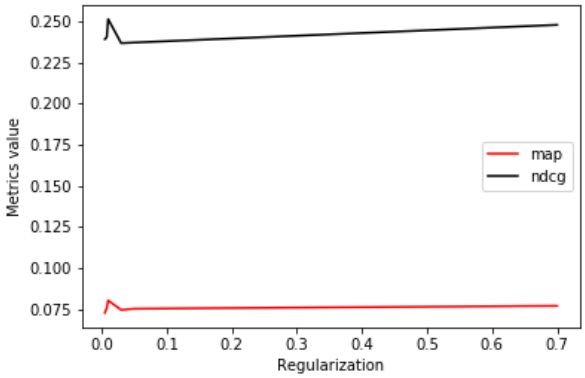


**Figure 12: Training with different regularization parameter.**

capable to change the learning ratio. Our training produces graphics like the ones shown in figures 13 and 14. The presented score is measured in terms of the Gini norm, a good metric for binary classification that is equal to ROC-AUC but with a bigger resolution spectrum.

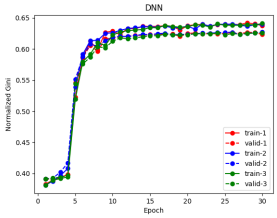


**Figure 13: DNN training curve.**

Using those curves and the defined metrics, we produced a sensitivity analysis shown on table 3.

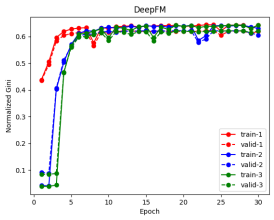


**Figure 14: DeepFM training curve.**

**Table 3: Sensitivity analysis for DeepFM model. We vary the number of neurons per layer ($H_{ixj}$) and the batch size ($B_n$). We use as a base a default configuration with a batch size of 1024 and 32 neurons by layer.**

| Configuration | Time (sec) | MAP @10 | NDCG @10 | Novelty | Diversity |
|---|---|---|---|---|---|
| $B_{1024}$ $HL_{32x32}$ | 2027,52 | 0,8911 | 0,9437 | 0,1667 | 0,4245 |
| $B_{512}$ | 2186,77 | 0,8940 | 0,9456 | 0,1845 | 0,4967 |
| $B_{256}$ | 2692,96 | 0,8919 | 0,9440 | 0,1847 | 0,4996 |
| $HL_{16x16}$ | 1815,70 | 0,8939 | 0,9454 | 0,1815 | 0,4775 |
| $HL_{8x8}$ | **1708,57** | 0,8941 | 0,9455 | 0,1837 | 0,4869 |
| $B_{512}$ $HL_{8x8}$ | 2180,6 | **0,8945** | **0,9458** | **0,1857** | **0,5149** |

We can see that when batch size is decreased from 1024 to 512, we get better results under all metrics. However, if we continue decreasing this value to 256 the results are slightly worse except for the Novelty that increased a little. That can be explained because when we decrease the batch size we generate an over-training on the training set, and relations are quickly learned with a few number of records. Also, when we decrease the number of neurons by layer from 32 to 8, we get better results over all metrics. This effect can be explained because of the number of features on each tuple (32), so when we use a smaller number of neurons we are forcing an embedding that benefits from interactions. Finally, we combine those configurations to create a model with the best performance for use in the test set.

## 6 RESULTS

Table 4 shows the results of evaluating all recommender systems models discussed previously.

**Table 4: Test results for every model presented, including results with and without (WS) sentiment analysis considered.**

| Configuration | Time (seg) | MAP @10 | NDCG @10 | Novelty | Diversity |
|---|---|---|---|---|---|
| *ALS* | 1421,8 | 0,107 | 0,332 | - | - |
| *FM* | 1450,86 | 0,893 | 0,944 | 0,176 | 0,45 |
| *DeepNN* | 1889,58 | 0,897 | 0,947 | 0,186 | 0,49 |
| *DeepFM* | 2027,52 | 0,891 | 0,943 | 0,167 | 0,43 |
| *FM(WS)* | **1416,04** | 0,894 | 0,945 | 0,18 | 0,46 |
| *DeepNN(WS)* | 1866,35 | **0,897** | **0,948** | **0,197** | **0,54** |
| *DeepFM(WS)* | 1984,17 | 0,892 | 0,944 | 0,19 | 0,53 |

First of all, we see that our baseline model ALS is the fastest to train, but is the one with the poorest performance in all metrics discussed. The performance of FM, DNN and DeepFM is close to eight times better in terms of MAP@10, indicating that the content and contextual data are very important for this problem, and implicit co-ocurrances are not enough to yield good results. Another important result is that contrary to our assumptions, the DeepNN model achieves better results on all metrics outperforming FM and even DeepFM that includes this architecture on it. This could indicate that the high order interactions contributed from the DeepNN model provide more information than the lower order ones from the FM, so these interactions are generating noise to the final result of DeepFM. Although the differences in terms of MAP@10 and NDCG@10 are small, they are consistently better for DeepNN. Moreover, novelty and diversity are also better for DeepNN, showing that even with similar ranking results, the resultant recommendations are more novel and diverse, which are very important aspects in the game industry.
Another interesting result is that when we do not consider the sentiment analysis values generated from reviews, all models show even better performances against themselves when considering this feature. A reasonable cause for this is the small proportion of reviews compared to the total user-game interaction records. The metrics of Novelty and Diversity also benefit from not including sentiment, reaching up to 0.54 in terms of diversity for DeepNN without using sentiment. As we have commented, on in the proposed online video game problem, it is always better to reach improvements in diversity and novelty in addition to accurate predictions as assessed with MAP and NDCG.

## 7 CONCLUSION

In this work we experimented with different options of recommendation systems within the context of video game recommendation, using a dataset from the company STEAM as a test platform. We joined three databases that contained information of user purchases, the hours dedicated to play each item, their social interaction (critics) and the characteristics of video games. The ALS model was chosen as the base model for comparison, and tests were carried out with the models of Factorization Machines (FM), DeepNN and DeepFM. The latter model uses a deep neural network (DeepNN) that works in parallel to a layer composed of a FM, to introduce higher-order interactions between inputs, aiming to improve the performance of our prediction in terms of factors of novelty, diversity and accuracy.

All the models studied outperformed the ALS baseline model. DeepNN stood out from the rest. Despite being a simpler model than DeepFM, it managed to exploit item-user relationships better, and although it takes longer than FM and DeepFM to achieve competitive results, it achieves consistent results over different datasets. Also, it obtains better results with the evaluated metrics, which implies that the higher order interactions provided by the DeepNN model provide more information than the lower order interactions provided by the FM model. As we argued the non-linear interaction from items

features is leveraged by deep-learning based methods, but interestingly, this difference is better perceived in novely and diversity improvements over the lists of recommendations.
In order to obtain additional information, we used the user reviews with different methods of sentiment analysis, finding a continuous measure that could further abstract the user's opinion for the video game. We conclude that the proportion of available reviews was not large enough in order to contribute to an improvement of the results, being translated instead in an effect of noise. Metrics reduce their value as much in MAP@10 as in NDCG@10 and even in the measures of Novelty and Diversity.

In future work we plan to perform tests of parameter analysis on the DNN model that gave the best results to see if they can be improved. We might also conduct a user study to validate the offline results on a real system, where other variables can have a strong impact on the final results. Finally, we have recently found that the STEAM database has been updated by its author. In its last version, a new dataset of 7,793,069 reviews is present, so we expect for future work to leverage them for a larger text analysis. This will help us test whether sentiment analysis and other textual features can indeed be a useful tool for recommending video games.

## 8 ACKNOWLEDGEMENTS

This work have been partially funded by Millennium Institute for Foundational Research on Data (IMFD). We have also been partially supported by NIC Chile.

## REFERENCES

[1] Paul Bertens, Anna Guitart, Pei Pei Chen, and África Periáñez. 2018. A Machine-Learning Item Recommendation System for Video Games. *arXiv preprint arXiv:1806.04900* (2018).
[2] Andrea Esuli and Fabrizio Sebastiani. 2007. SentiWordNet: a high-coverage lexical resource for opinion mining. *Evaluation* 17 (2007), 1–26.
[3] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 1725–1731.
[4] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. Ieee, 263–272.
[5] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.
[6] Saif M Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*. 321–327.
[7] Denis Parra and Xavier Amatriain. 2011. Walk the talk: analyzing the relation between implicit and explicit feedback for preference elicitation. In *Proceedings of the 19th international conference on User modeling, adaption, and personalization*. Springer-Verlag, 255–268.
[8] Denis Parra and Shaghayegh Sahebi. 2013. Recommender systems: Sources of knowledge and evaluation metrics. In *Advanced techniques in web intelligence-2*. Springer, 149–175.
[9] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4, Article 66 (July 2018), 36 pages. https://doi.org/10.1145/3190616
[10] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.
[11] Bita Shams and Saman Haratizadeh. 2017. Graph-based collaborative ranking. *Expert Systems with Applications* 67 (2017), 59–70.
[12] Saúl Vargas and Pablo Castells. 2011. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 109–116.
[13] Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 86–94.
[14] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. 2014. Beyond clicks: dwell time for personalization. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 113–120.