A supervised learning approach to build a recommendation system for user-generated content in a casual game

Paulo de Freitas Coleta Neto[1]* Túlio Braga Moreira Pinto[1]†

[1]Universidade Federal de Minas Gerais, DCC, Brasil

Abstract

There is a growing tendency in games that made use of user-generated content, artifacts created for a game by the players instead of game developers. This type of content may allow games to have an extension of playable hours and then a greater longevity. However, allowing users to provide their own content may result in low-quality game experience to other players, caused by the difficulty to find a content that delight the user base that will consume the user-generated content. In this paper, we present an approach to creating pleasant content recommendations to the users. We propose the use of Supervised Learning to build a model capable of predicting positive ratings from a user over a specific content and use this information to build the recommendation system. The recommendation system is capable to handle each user individually and create their own content recommendation list. To validate and measure our strategy performance, we collected user ratings of stages created by others user in a mobile puzzle game called Mr. Square, during a period of 5 months. Our results show that we can improve the percentage of positively rated content, and then the user satisfaction, to 96.2%, while the current strategy implemented in Mr. Square holds a satisfaction of only 70.1%.

Keywords: Recommendation, Classification, User-Generated Content, Games

1 Introduction

One of the hardest and most important challenges of game development is how to keep your players base active and interested in your game. A possible solution to this problem is to use what is usually called *user-generated content* [1]. User-generated content is any kind of resource provided by regular people while users of a system or service. This content is most likely provided voluntarily and presented to others users in an entertaining way. Examples of such content can be a location rating, a wiki page or even a video. The use of User-Generated Content is growing fast because it is not expensive to obtain, as most users will provide this content seeking only recognition for their contributions [2].

In video games, the user-generated content can vary from a wide range of possibilities. Examples of this type of content are: (1) new stages, common in games that provided level editor and similar tools; (2) cosmetic items, that are items that provided visual modifications in game but no changes to the game mechanics or balance, and (3) mods, that are advanced modifications capable of creating new game modes of even a entirely new game. The level of dependency and use of User-Generated Content will change with each game. There are games designed to be totally dependent on such content[3].

Some examples of games that use user-generated content are: (1) *Counter-Strike*, an FPS game created from a Mod of the game *Half-Life* in 1999; (2) *Defense of the Ancients*, an MOBA game created as a custom map of the game *Warcraft III* using his built-in level editor; (3) *Dota 2*, a more recent example that comes with workshop tools that allow users to create cosmetic items, custom maps, and mods; (4) *Super Mario Maker*, a game entirely focused on levels created by other users with the built-in level editor.

Adoption of user-generated content in a system or game have some benefits but also points a challenge, the bad content can provide an unfair experience to users [1]. Such experience can influence users to quit the game instead of extending its longevity.

Systems with user-generated content commonly implement a rating methodology to curate the resources. This may be a simple *like* and *dislike* system, a star rating or a number grading representing the user approval or disapproval. These evaluations represent what the whole community thinks about a specific content. Sometimes this strategy may fail to reflect the preferences of each individual user.

Recommendation systems are able to list a specific set of items most likely to please each user individually. Build such systems is a work already done in other contexts such as presented by Davidson et al. [4] in *Youtube*, by Linden et al. [5] in Amazon or by Gomez-Uribe and Hunt [6] in *Netflix*.

Recommendation systems have already been suggested [7] as a way to solve the disadvantages of the usage of user-generated content [1]. Such tools can provide to each user an individual list of content that the system believes will please him. Constructing this list, the system is able to minimize the high ratio of bad content. Our users will not interact with these undesirable resources and by so having a good experience.

In this paper, we present a strategy to build a recommendation system using Supervised Machine Learning techniques. We evaluate our results using data collected from Mr. Square, a mobile casual puzzle game with a built-in level editor. At the end of this paper, we also discuss the applicability in a real world with time constraints.

2 Related Work

Researches involving recommendation systems exist in several contexts. One of the most famous examples is the work of Linden et al. [5], that uses a *Collaborative Filtering* variation to create personalized product recommendations to each user in a virtual store. In a work presented by Davidson et al. [4] a *recommendation system* was created and evaluated over the platform of videos Youtube.

There are also works regarding *recommendations systems* in contexts more related to electronic games, as published by Sifa et al. [8] and Skocir et al. [9]. In the first work, the authors recommended new games to a determined user using the data from the platform *Steam*, that focus in computer games. In that paper, classification models are compared using the neighborhood of each item, the games, and the *Factor Oriented Model*. The work of Skocir et al. [9] creates their own new model called *MARS, Multi-Agent Recommendation System*, that generates a profile with each user's abilities using the historic of games played before. The work presented in our research differ from those cited above, since the content we recommend is related to game levels, in this case, *puzzles* inside a

*e-mail: paulo.freitas@dcc.ufmg.br
†e-mail: tuliobraga@dcc.ufmg.br

unique game. The other papers focus in recommend new games to each user inside a unique platform.

A different approach to the problem of the low ratio of good user-generated content is presented by Hicks et al. [10]. The authors deal with this problem in the context of the game called BOTS, in which the players can create puzzles and share with others users. To handle this issue, they added a process of levels' self-validation, in which each player needs to solve its own puzzle before submitting it to the server. The authors of the paper reported that this process reduced the number of low-quality levels created by the users.

In comparison, our work differs from the others since our recommendation system is based on content. Also, our data source is uncommon. In the mentioned studies, the authors had user details available to build a representation. In our work, we created an individual user representation taking into consideration the rating history over the user-generated content consumed. Each puzzle the gamer played interfere into the representation. This makes our recommendation system different than the other since we have applied a technic called *Content-Based Recommendation System* in the context of games without using any additional data, what also minimize any questions about loss of privacy.

Our research uses a totally different approach than that proposed by Hicks et al. [10]. Even though we have the most similar goals, we handle the same problem with completely different solutions. Their work focus in reducing the creation of what is called low-quality content, while our study is able to send a set of recommendations to each user regarding predictions of what our system believes will be considered high-quality content. This difference represents a big advantage since we do not change the whole spectrum of available levels, and then we keep a large database of levels that can be enjoyable to different sets of users, respecting each user own preferences of what is low-quality or high-quality content.

3 Puzzle Recommendation

The puzzle recommendation inside a game is the task to only show to each user the set of puzzles we predict that he will rate positively. The success of this task can be easily measured by the changes in the value of what we call satisfaction for each user. The satisfaction of a given user can be defined by the ratio of puzzles that the user rated as good over the total amount of puzzles he rated, or alternatively the percent of positively rated puzzles.

Accomplish this task is hard because we need to understand the profile of each user. Different users have different criteria and reasons to approve or disapprove a given content, and then they will like or dislike different subsets of levels. A *Recommendation System* must be capable of understanding this difference between groups of users and the various features the users will notice on each content, the puzzles.

In our approach to building a *Content Based Recommendation System*, we must create a model representation of each level, that will be the content to be recommended, and a representation of each player, that will be the users to consume the recommendations. Also, there is a representation for the user-level pair, that relates to a user's content rating over that level in the past. Our task in building such *Recommendation System* can be seen as learning if each of those possible user-level pairs will hold a positive or negative rating. It is equivalent to predict whether the user approves or disapproves that level he just played.

Commonly, the profile's representation of each user shows data that highlights his preferences and history in the game or system. The representation of levels, content, is a much complex task since it depends on which data are already available. Guyon et al. [11] discuss the process of constructing and selecting features to represent items for the task of machine learning and classification models.

After building our rating's representations, our user-level pairs, we are able to use *Machine Learning* models to train over the data and classify each of these pairs. These classification models will learn how to predict the perception of users about each specific level, that are the *puzzles* created by others users. This task is normally measured by *accuracy* in its prediction of each rating. Additionally, we adopted another key metric we call *precision score*, in which its value for the positive class is equivalent to the average value among all users, what we previously called user satisfaction metric. Therefore, an increase in the positive class' *precision score* causes an equivalent increase in the user satisfaction.

4 Methodology

Through our work, we considered a dataset that consists of a collection of user ratings over user-generated levels under the mobile puzzle game Mr. Square. With this dataset, we trained some *classification models* based on three techniques: (1) SVM, *Support Vector Machine*; (2) Decision Trees; and (3) *Random Forests*. Then, using the built classifiers we were able to predict the rating of a given user over game's specific levels. This information enabled us to provide recommendations to each user taking into consideration only the positive class predictions.

In this section, we present the algorithms and detail the required tasks in order to build the classifier models. First, we present the classification algorithms evaluated in this work. Then, we present the process of collecting the dataset and the process of constructing the representation of each level and user, what we call *Feature Engineering*. Finally, we discuss our process of sampling and disclose the metrics to evaluate our results.

4.1 Classification Models

In this work, we evaluate the use of different types of classifiers to use in our task of building a content-based recommendation system. In this subsection, we make a brief description of the algorithms used to train our classifiers: Support Vector Machines, Decision Trees, and Random Forests. Then, we introduce the key metrics to evaluate the results of our classifiers: *accuracy* and *class precision*.

The classification models presented here can be described as supervised learning models. These methods adjust their answers basing in training datasets. Therefore, we consider the training data as a set of n points in a d-dimensional space, and for each point, we assign a label $y_i \in \{+1, -1\}$ representing the class that the point refers to. In our case, there are two different classes: like or dislike. In this way, we defined the *dataset* as $D = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$.

4.1.1 Support Vector Machines

In this work, we used the *Support Vector Machine*, SVM, a binary classification method based on maximum margin linear discriminants. The goal of the SVM model is to find the best hyperplane that maximizes the margin or space between points of different classes. Additionally, we can employ a method known as *Kernel Trick* to find that optimal nonlinear discriminant between the classes, corresponding to a hyperplane in a high-dimensional "nonlinear" space. This process is detailed in Zaki and Wagner Meira [12, Chapter 21]. The SVM model is also considered to be optimal in binary classification, as said in Boser et al. [13].

A d-dimensional hyperplane is given by the set of points $\mathbf{x} \in \mathbb{R}^d$ that satisfy the equation $h(\mathbf{x}) = 0$, in which $h(\mathbf{x})$ is the hyperplane's function as below:

$$h(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b = w_1x_1 + w_2x_2 + ... + w_dx_d + b \qquad (1)$$

Where $\mathbf{w}$ is a *vector* of d-dimensional weights and b is a scalar value known as bias. The hyperplane's function $h(\mathbf{x})$ can be used as a linear classifier or a linear discriminant and can predict the label y for any given point $\mathbf{x}$ following the decision rule below:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0, \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases} \tag{2}$$

Given the above classification model, the SVM is an algorithm that allows to refine the value of the *vector* **w**, that represents the hyperplane $h((x))$, maximizing the distance margin between the points **x** that belongs to different values of labels, y. In the training stage, the algorithm adjusts the values of weights **w**, with d dimensions, relative to each *features* of each item of the training dataset. At the end of the training process, the algorithm provides a hyperplane $h(x)$ capable of split the data. The detailed version of the SVM algorithm employing the *Kernel Trick* can be found in Boser et al. [13]. In our paper, we evaluate three different *Kernels*: polynomial with degree 2, polynomial with degree 3, and the Radial Base Function, RBF. Details about the RBF kernel can be found in the work of Musavi et al. [14].

Finding the hyperplane $h(\mathbf{x})$ that split perfectly the points of different labels can become an impossible task if the points can not be correctly split, or can lead to very complex model, a problem known as *overfitting*. To avoid these undesired cases it was introduced a regulation parameter C in the SVM model, that needs to be empirically chosen with the goal of minimizing the validation error.

$$\text{Objective function: } \min_{\mathbf{w},b,\xi_i} \left\{ \frac{\|w\|^2}{2} + C \sum_{i=1}^{n} \xi_i^k \right\} \tag{3}$$

$$\begin{aligned} \text{Linear restrictions: } y_i(w^t x_i + b) &\geq 1 - \xi_i, \forall x_i \in \mathbf{D} \\ \xi_i &\geq 0, \forall x_i \in \mathbf{D} \end{aligned} \tag{4}$$

In the equation 3, the expression $\sum_{i=1}^{n} \xi_i^k$ represents the classification errors in the training stage, while the expression $\frac{\|w\|^2}{2}$ represents the goal to maximize the margin. The parameter C controls the *trade-off* between maximizing the margin and minimizing the error during the training stage. The constant k defines the type of loss function considered. When $k = 1$ is said that we use the *hinge loss* and when $k = 2$ we use the *quadratic loss* [12, Chapter 21].

4.1.2 Decision Trees

The *Decision Tree* classifier is a model that creates a tree that partitions the dataset space recursively until achieve the prediction of the class y_i of a given point x_i. The decision tree applies an axis-parallel hyperplane with the intent to split the space, R, of the dataset into two resulting half-spaces or regions, also partitioning the points of the dataset in two partitions. Each of these regions will repeat recursively this process of splitting until the partitions are stated as *pure*. We can say that a region of the Decision Tree is *pure* if it just contains points of the same class.

The resulting hierarchy of this process is a tree model in which the leaves are the last partitions. We assign a class to each leaf according to the most popular label of the points in that region. To classify a new point, we have to walk recursively on the tree until reach a leaf and assign the leaf class to the point. More details over the *Decision Tree* model can be found in the book published by Zaki and Wagner Meira [12].

In a similar manner to SVM models, the Decision Tree models also have their own regulation parameters to avoid *overfitting*. Some examples of this parameters are the deep of the tree, the number of elements in each leaf, and the maximum number of leaves. For the purpose of this work, we only used the number of elements in each leaf as regulation.

4.1.3 Random Forests

The *Random Forest* model is a combination of multiple tree prediction models. Each one of these tree models are built considering only a random fraction, independently distributed for each model and with same distribution, of the whole vector that represents each element of the dataset.

Random Forest can achieve great results once that the generalization error for forests converges to a limit as the number of trees that ensemble the forest grows. This property leads us to an interesting consequence: *Random Forest*, as others ensemble models, rarely overfit. Besides that, there is still a regularization parameter, the number of trees, that is also used to reduce the time to compute the model. For a detailed explanation on how this model works, see the work of Breiman [15].

4.1.4 Validation Metrics

It is possible to employ a wide range of metrics to evaluate and compare the results of different classifiers. Here we present the *accuracy* and the *class precision*, as they are the metrics we use to evaluate our models. The *Accuracy* of any given classifier is the ratio between the correctly predicted labels over the total of predicted elements. It can be defined as:

$$Accuracy = \frac{1}{n} \sum_{i=1}^{n} I(y_i = \hat{y}_i) \tag{5}$$

The *Accuracy* is an estimative of the probability of a given classifier makes a correct prediction. High values of accuracy indicates a better classifier.

The next metric is known as *class precision*. The *class precision* of a classifier M for a class c_i is the ratio of correct predictions over all predictions to the class c_i. In our work, this metric is important since the *class precision* of the positive label is equivalent to what we defined as *user satisfaction*. The mathematical definition of class precision is as follows:

$$\text{Class Precision for class } i = \frac{n_i i}{m_i} \tag{6}$$

Where m_i is the total number of points predicted as belonging to the class i, and $n_i i$ is the number of points correctly predicted as belonging to the class i.

4.2 Dataset

To evaluate our results, we used a dataset from the game *Mr. Square*, developed by the *Ludic Side Game Studio* company. *Mr. Square* is a puzzle game for mobile devices in which users can, besides playing the default set of levels, create and submit new puzzles using the built-in level editor. While playing the mode with random levels, the player will be challenged to solve the puzzles created by others users. After solving each puzzle in this mode the player can provide a positive or negative rating for the solved level. During the period of collecting the individual ratings, the game delivered each level to the user in a completely random manner.

Our dataset features (1) level rating's logs, composed by likes and dislikes given by real game players, and (2) each *puzzle* details such as its dimensions, initial player position, obstacle positions, puzzle solution, and the number of positive and negative reviews.

Our user rating's *logs* were collected during a period of 5 months. They are made of 646,838 user ratings of 70,323 active players through the period of collection. Our dataset is complemented by a total of 457,823 levels that were created since the game launch and until the end date of the log collection process. From the total of ratings, 453,455 are positive ratings, remaining 193,393 negative ratings. By the ratio of positive and total number of ratings, we can state that the overall user satisfaction over the user-generated content is 70.1%.

4.3 Feature Engineering

When using content-based recommendation systems it is necessary to create a vector of representations for the user and for the content. These two vector representations can be combined as a single vector representation for the user-content pair, bringing a representation for the user rating. The dataset collected provides structured data about all the levels created by users until the collection's end date. Regarding users, the data available relates to a log of their ratings.

To build the representation of each *puzzle*, we choose a set of 11 features, in which all of them belong to the puzzle original data. They are: the height; the width; the number of steps to the solution; and the presence of each one of the eight possible obstacles in that puzzle. Some of them can be seen in Figure 1.

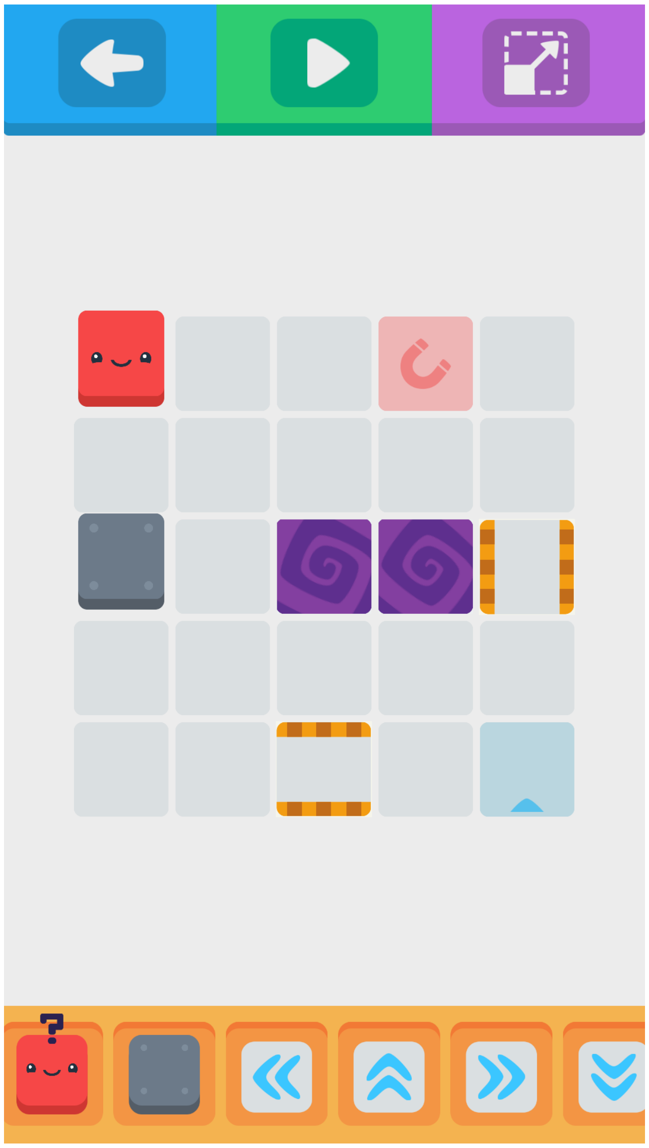

Figure 1: Mr. Square level editor screen.

Our vector representation for each user represents and derives from their history of ratings. Each user has 22 features, divided into features of positive ratings and features of negatives ratings. The first 11 features relate to the average of the features' values over only the positive-rated levels. Similarly, the other 11 features relate only to the negative-rated levels. This way, to create a user's rating representation we just needed to join the 22 features representing the user and the 11 features representing the specific level into a single vector of 33 features as user-level representation, or user rating.

While creating the vector representation for each puzzle it was detected 4,960 levels with invalid configurations, such as size below the minimum or higher than the maximum value possible. These invalid levels were ignored in all remaining tasks and the remaining number of valid levels was 452,863. Due to the removal of invalid levels, it was also needed to remove 8,499 user ratings related to these levels.

4.4 Sampling

The process of building a classification model is a time-consuming operation that can be even worse while evaluating different models on a try to find the best configuration of parameters. One of the factors that largely impact the amount of time spent to construct each model is the size of the dataset used in training tasks. To be able to evaluate a diverse range of models, we used a sampling of our whole dataset in order to tune each of the models. Our sampling size was configured with 50 thousand ratings.

The process of sampling can compromise the overall quality of results case the resulting sample does not represent well the original dataset. A way to measure how well does the sample represent the original dataset is to compare the variance values of each one of the features in the resulting sample with the values in full dataset.

To ensure that the sample technique used cannot lead to a unique bad sample we added to our process a confidence interval for the variance of features. In total, we built 10 different samples, and for each sample, we repeated the following process of calculating the variance of each feature of the dataset. For each feature, we calculate the ratio between the variance of the feature in the original dataset over the value found in the sample. Therefore, we expect this ratio to be as close to 1 as possible. The average between all the features' ratios of all the samples was 1,0006 and the standard deviation found was 0,0039. These values show that our different samples represent our original dataset in a proper manner, making reasonable the use of sampling in our process. Regarding the calculus of *confidence interval*, we found, at a 99% confidence, that our variance ratio is between the interval of 0,997 and 1,005.

5 Results

The first task while working with classification models is to tune each one of the models. For the SVM we needed to find the optimal value for the regularization parameter C, while in Decision Trees one of the values that were optimized is the number of elements in each *leaf*. For the Random Forest, it is common to choose the best number of components to ensemble the classifier.

In order to evaluate the learned classifiers with different parameter values, we split our dataset into a training set and a validation set. The first one is used only to learn the model and the second one is used only to evaluate our results. This process is relevant since it helps our model's metrics to avoids an *overffiting* interference, meaning that it preserves our model of memorizing the training data instead of learning the generalized pattern regarding the classification problem. All results presented in this section were calculated using only the validation set.

In the Figure 3, Figure 2 and Figure 4 we see the plot of *accuracy* and *user satisfaction* in function of the parameter C for the SVM models trained with kernels, polynomial with degree 2, polynomial with degree 3 and RBF, respectively.

For the polynomial with degree 3, the best model was found when C is equal to 2^2 with *accuracy* of 87.1% and *user satisfaction* of 96,2%. When using the polynomial with degree 2, the optimal value of C was also 2^2, with the values of *accuracy* and *user satisfaction* only differing from the polynomial with degree 3 after the

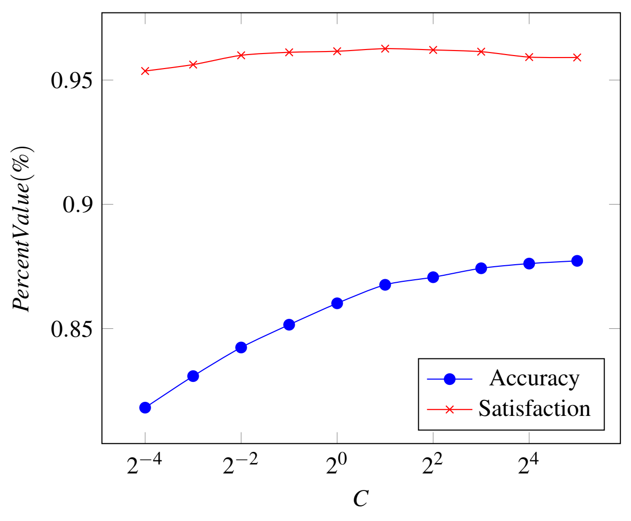

Figure 2: Optimizing C value for SVM-Poly2

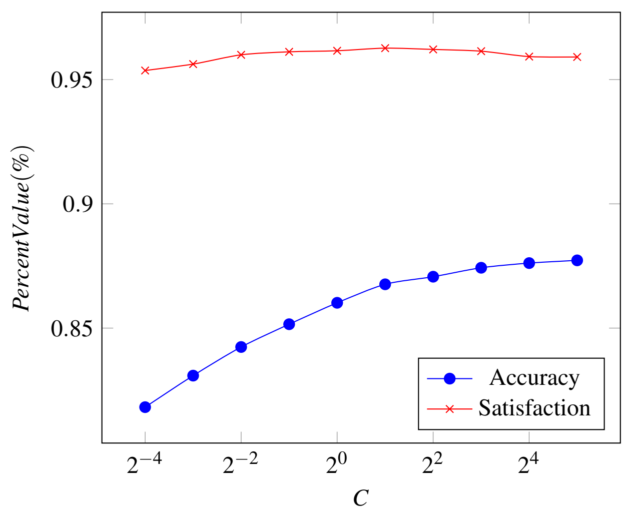

Figure 3: Optimizing C value for SVM-Poly3

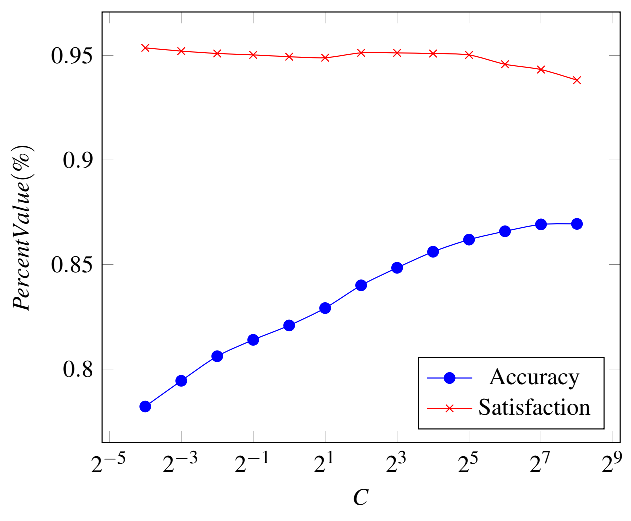

Figure 4: Optimizing C value for SVM-RBF

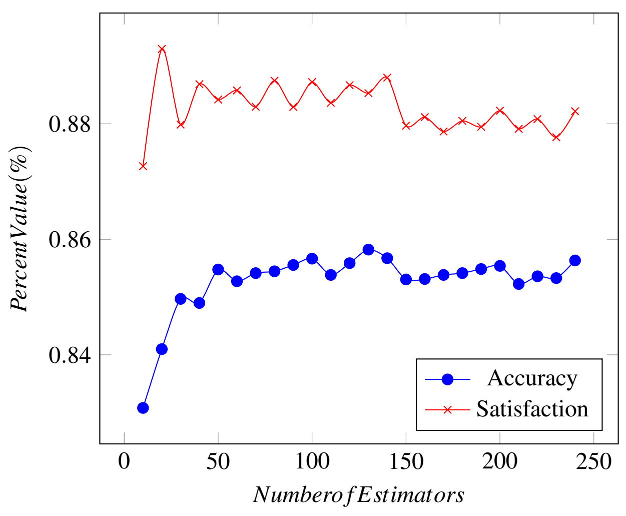

Figure 5: Optimizing N Estimators for Random Forest

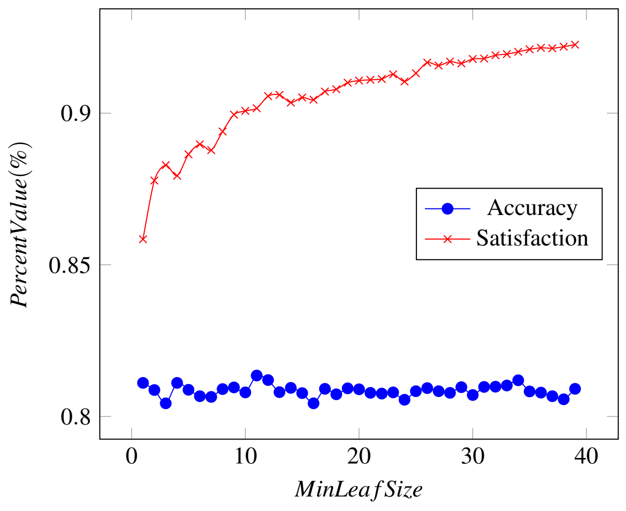

Figure 6: Optimizing Min Leaf Size for Decision Trees

8ª decimal case. Finally, for the RBF kernel the best value found for C was 2^3, while the values for *accuracy* and *user satisfaction* was 84.8% and 95.1%, respectively.

The values for the SVM with polynomial kernels with degree 3 showed as irrelevant for the given task. We can interpret that the hyperspace built with the polynomial kernel with degree 2 was sufficient for splitting the points in this task. This way, for degree 3, the relevant features remained the same as in degree 2.

While evaluating the *Random Forest* classifier, we can see a sequential significant improves in the model *Accuracy* until hit the value of 50 *estimators*. After hit this value the value of accuracy vary a little under and above, but at no significant value and we can not find a clear local maximum. We believe that this behavior is due to the randomness nature of the algorithm.

The highest *User Satisfaction* value, 88,8%, was found when the total of *estimators* was 140, and the value of *Accuracy* for the same parameter value is 85,6%.

The curve values for the Decision Trees show us as an interesting pattern and highlight one problem that could occur if we only evaluate our models using the *user satisfaction*. In this last model, we realized an always growing value for the satisfaction while the *accu-*

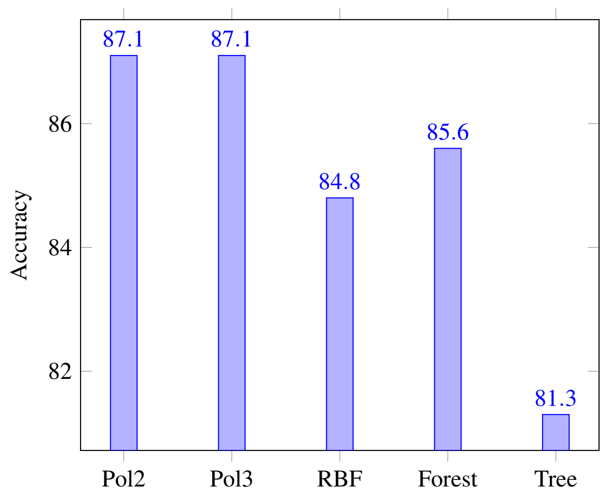

Figure 7: Accuracy comparison between Models

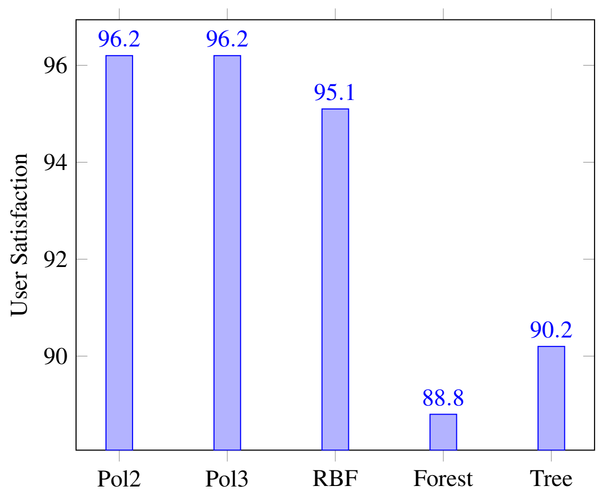

Figure 8: User Satisfaction comparison between Models

racy slowly degrades. If we continued plotting this chart we would be able to see that the disparity between the population of classes would make the decision tree model to lose *accuracy* while growing the *class precision* in favor of the most popular class. While this is not seen as a problem at a first glance, it could lead to not recommending many true positive puzzles.

The best value for the model's accuracy was found when the *minimum leaf size* was equal to 11, with *accuracy* of 81.3% and *user satisfaction* of 90.2%.

The Figure 7 shows a comparison between the *accuracy* values in the best configuration of each model. We can see clearly that the SVM with polynomial kernels have the best accuracies but by a tiny margin. The third best option was the *Random Forest* and in last place, we see the *Decision Tree*.

Comparing the same models but using the metric of *class precision* for the positive class, in this work often called *user satisfaction*, the Figure 8 shows us that the polynomial models are still the best model option. However, here the SVM with RBF kernel shows better results than the next option by a significant margin. Also, it is interesting to notice that the Random Forest shows poor results on this metric.

To better evaluate our learned classifier and estimate a confidence interval, we submitted our SVM classifier with Kernel Polynomial with degree 2 and Log C value equals to 2 to a *K-Fold* validation process, employing K as 10. In this process, we split our database into K partitions with the same size, and we train K classifiers with the same configuration.

Each one of the models were trained with K-1 partitions and their metrics were validated with the remaining partition. Considering the results of the different K classifiers, we estimated the average value of model's *accuracy* and the *standard deviation* of these values. Along with the number of samples created, we estimated the confidence interval of our model [12, Chapter 12]. In the Table 1, we present the values calculated with K equals to 10 and confidence interval of 99%. The values presented in the table show that our results are stable and reliable.

Table 1: K-Fold Validation

Metric	Average	Standard Deviation	Confidence Interval
Accuracy	86.14%	0.436597959%	85.69% to 86.59%
Satisfaction	96.28%	0.276532735%	95.99% to 96.56%

6 Applicability

The study presented in this paper occurred in a *offline* environment of the game *Mr. Square*, with a static number of levels, and using a history of each player ratings. Here we present a discussion about the use of these models in a dynamic environment.

To ensure the accuracy of the recommendation system, it is necessary to update the user representation for each new level rated, while the level representation can be built when it is submitted to the servers once that the level representation is static. A second challenge that will arrive in an *online* environment is the need to update and construct a new classification dynamically. This may be needed (1) when new elements are added to the game, what may reflect on new features to the user and level representations; (2) when the user's playing preferences change significantly while experiencing more the game. Both examples of needs can be detected by the quick calculus of the variance of the dataset and by setting the desired threshold as a trigger value to training new models. This task can be addressed through the use of a daily job in the game server, or manually checked while an update occurs in the game. Keeping a record of the *accuracy* and the *user satisfaction* for the recommendation system may be a good approach too. A drop in these values indicates a strong need to training a new model.

Additionally, we made a quick experiment measuring how many recommendations we can produce in a small time interval. In the *Mr. Square* case, the user consumes about 5 levels per minute, that is a case far different from the context of others recommendations systems, while a single recommended content can take hours of the user. In order to validate the viability of producing recommendations for each active user we estimated how much time a single machine take to predict the rating of each user to a small set of levels.

We run this experiment using Spark [16] and his library MLlib [17] on a setup using 8 cores and a remote single MongoDB database. We loaded all users and 0.01% of level representations already stored in the database and evaluated all combinations between this subset. All positive predictions have been written to the MongoDB, so our server quickly consumed then. This process experimented a total of 3,164,535 user-level pairs and classified 2,505,703 of them as a positive rating. All this process took about 4.9 minutes, resulting in a rate of 7 recommendations for each user per minute.

The number is just a bit higher than the 5 levels consumed per minute and was calculated using all active users during the period of collecting the rating *logs*. The peak number of active users is certainly lower than the amount considered in this experiment. Therefore, we believe that the recommendation system can be used without the fear of creating a bottleneck in the whole game server.

7 Conclusion

In this work, we used a classification model to accomplish the task of creating user-generated content *recommendations*. Our results showed that we were able to increase the players' satisfaction from 70.1% to 96.2%. Lately, we discussed the applicability of these models and demonstrated their viability in a real world environment.

Our method does not directly solve the low ratio of high-quality user-generated content problem. Instead, we were able to handle the problem by individually listing only the puzzles that would please the user. By not removing the low-quality levels from the game, we were able to keep a diverse content capable to be consumed by any kind of gamer. This is the biggest advantage of our work against related works that aim to solve the same problem.

In further work, we still need to validate our approach toward different datasets, proving that our strategy can be generalized to others games or environments. In this paper, we have not evaluated the impact of user preferences changing over the time. One approach should be degrading the weight of old ratings and favoring most recent ratings when constructing the user representation.

Besides our good results for user satisfaction, we still need to improve our model *accuracy*. Further work should focus on minimizing the number of positive user-level's relations predicted as negative, what is called *false-negatives* and apply this model on a live environment, more susceptible to failures. This is important since the user can lose the opportunity to play some high-level content.

References

[1] P. Jasek, "User generated content for video games," Department of Computer Science Aalborg University, Tech. Rep., January 2014, 9th semester project report of the Software Development program. [Online]. Available: http://vbn.aau.dk/ws/files/176764496/Report_swd903e13_.pdf

[2] J. Krumm, N. Davies, and C. Narayanaswami, "User-generated content," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 10–11, Oct 2008.

[3] K. Graft, "User-generated content: When game players become developers," Oct. 2012. [Online]. Available: http://gamasutra.com/view/news/179493

[4] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," pp. 293–296, 2010.

[5] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, Jan. 2003.

[6] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, pp. 13:1–13:19, Dec. 2015.

[7] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," pp. 325–341, 2007.

[8] R. Sifa, C. Bauckhage, and A. Drachen, "Archetypal game recommender systems," pp. 45–56, 2014.

[9] P. Skocir, L. Marusic, M. Marusic, and A. Petric, "The mars - a multi-agent recommendation system for games on mobile phones," pp. 104–113, 2012.

[10] A. Hicks, V. Cateté, and T. Barnes, "Part of the game: Changing level creation to identify and filter low quality user-generated levels," 2014.

[11] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*. Springer, 2008, vol. 207.

[12] M. J. Zaki and J. Wagner Meira, *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, May 2014.

[13] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," pp. 144–152, 1992.

[14] M. Musavi, W. Ahmed, K. Chan, K. Faris, and D. Hummels, "On the training of radial basis function classifiers," *Neural Networks*, vol. 5, no. 4, pp. 595 – 603, 1992.

[15] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," pp. 10–10, 2010.

[17] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, Jan. 2016.