A Binary Classification Approach for Automatic Preference Modeling of Virtual Agents in Civilization IV

Marlos C. Machado, Gisele L. Pappa and Luiz Chaimowicz

Abstract—Player Modeling tries to model players behaviors and characteristics during a game. When these are related to more abstract preferences, the process is normally called Preference Modeling. In this paper we infer Civilization IV’s virtual agents preferences with classifiers based on support vector machines. Our vectors contain score indicators from agents gameplay, allowing us to predict preferences based on the indirect observations of actions. We model this task as a binary classification problem, allowing us to make more precise inference. In this sense, we leveraged previous approaches that also used kernel machines but relied on different preference levels. Using binary classification and parameter optimization, our method is able to predict some agents preferences with an accuracy of 100%. Moreover, it is also capable of generalizing to different agents, being able to predict preferences of agents that were not used in the training process.

I. INTRODUCTION

The main goal of most games is entertainment. Entertainment is a subjective concept and, in order to know how much a game entertains a player, some general metrics are evaluated. One of the most important metrics is immersion, which is generally related to how absorbing and engaging a game is [1], [2], [3]. Two common approaches to achieve immersion are the use of stunning graphics and the development of a good Artificial Intelligence (AI) system. While graphics are responsible for initially “seducing” players, AI is responsible for keeping them interested in the game.

Over the last years, with the increase in computers’ processing power, AI is receiving more attention as a fundamental feature and new techniques are constantly being proposed. One approach that is gaining attention is player modeling. It intends to understand and model players’ characteristics and behaviors during the game. This modeling allows games to customize their AI to specific players, making the game experience more interesting. Among the players characteristics that can be modeled are preferences and knowledge [4].

This paper presents an automatic approach for preference modeling. It is an important topic because, once we are able to identify how a player – virtual or not, likes to play, we are able to adapt the game to improve player’s satisfaction. Some of the features we may adapt in the game are scenario [5], behavior and difficulty [4]. Moreover, Spronck and den Teuling argue that a preference model encompasses preferences and skills, and “has the potential to be much more useful than an action model”, which is the most common approach for player modeling [6]. The authors also suggest that preference models are seldom used due to the fact that modeling this concept is much harder than modeling actions, since preferences are not clearly measurable. One approach we can use to model agent’s preferences is to observe its actions as an indirect evidence of its preferences, as done in [6], [7] and [8], for example.

Machine Learning (ML) is a common approach for preference modeling, since we want to “learn” a model from a set of available examples, and then use this model to classify new examples. There are three main types of learning: supervised, semi-supervised and unsupervised learning [9]. Their main difference is in the data features available for training a model. While in supervised learning a complete set of labeled data is available, in unsupervised learning no labels are known. Taking as an example the task of preference learning, in both supervised and unsupervised learning the data we learn from is a set of matches already played, together with the players preferences. However, in the case of supervised learning, these preferences were previously labeled by an expert, while in unsupervised learning the algorithm learns using distance measures between data examples. Semi-supervised learning, in turn, uses both labeled and unlabeled data during the training process.

Here we model virtual agents’ preferences in the game Civilization IV with a supervised learning technique called Sequential Minimal Optimization (SMO) [10]. Our main contribution is to tackle preference modeling as a binary classification problem, which allows us to better infer agents’ preferences. This results in a method that satisfactorily predicts preferences of unseen agents, i.e., agents that were not included in the training process. This is an essential task for effective player modeling in commercial games.

This paper is organized as follows: the next section discusses some of the related work in the area. In Section III, we present the platform used as testbed (Civilization IV), and discuss the reasons for selecting it and the importance of agents’ preferences in this specific game. Section IV presents preference modeling as a ML problem. Our experimental setup and dataset description are presented in Section V, while results are in Section VI. We conclude this paper and discuss some possible extensions in Section VII.

II. RELATED WORK

Player Modeling is currently receiving a lot of attention in the AI game community. An evidence of that is that two different research groups developed, at the same time and independently, taxonomies for the field [4], [11]. This may
be seen as a general need for a review and organization of the works in the literature. Another work with this goal is [12], where the authors present a survey of the field.

In [4], besides the taxonomy, the authors also discuss the common techniques that have been used in Player Modeling, including machine learning. As mentioned, ML techniques can be divided in three different approaches: supervised, semi-supervised and unsupervised learning. The supervised methods are the most commonly used for player modeling, despite unsupervised learning being applied sometimes. Semi-supervised techniques, in contrast, are not common in this field. We first review two relevant works on unsupervised learning, and then discuss the approach of interest in this paper: supervised learning.

Relevant works involving unsupervised learning are presented in [13] and [14]. Thurau et al. [13] used self-organizing maps to cluster training samples in the state space, i.e., to reduce the dimensionality of input data for a subsequent training with multi layered perceptrons (MLPs). They generated simple behaviors for Quake II bots. More recently, Drachen et al. [14] used emergent self-organizing maps to obtain types of players of the game Tomb Raider: Underworld.

Regarding supervised learning, the works of [5], [6] and [15] are interesting to mention. Pedersen et al. [5] used artificial neural networks, both single-layer and multi-layer perceptrons, to learn the relation between features and the value of the emotional preference of a player in a game. In [15], Lavers et al. used Support Vector Machines (SVM) to recognize a defensive play “as quickly as possible in order to maximize (...) team’s ability to intelligently respond with the best offense” in the game Rush 2008.

Another work that also used supervised learning to model agents was [6]. In this work, the authors used the SMO algorithm, a method that solves the optimization problem, which arises during the training of SVMs, to model player preferences on the game Civilization IV. This work is very related to ours, as the authors tackle the same problem of preference learning. Furthermore, we use a dataset derived from the dataset they generated. However, the present work differs from [6] by the way we model the classification problem, using a binary classification, and by our approach to search for the best classifier parameters. These characteristics allowed us to obtain better results than those presented in [6].

Finally, in a previous work [7], we characterized some Civilization IV agents behaviors with linear regressions. We showed that some agents’ game indicators, when analyzed individually, can be linearly separable between agents with different preferences, and drew conclusions about preferences in relation to the time in the game (passage of turns), other preferences, and the gameplay.

III. Civilization IV and its Agents Preferences

This section discusses the game platform used as testbed and the importance and impact of agents preferences in this virtual environment.

Civilization IV is a turn-based strategy game (TBS) where each player is represented by a leader who controls an empire. Players/Empires compete with each other to reach one of the victory conditions. The game dynamics can be described as follows: you take your turn, move your units, conduct diplomacy, build and manage your cities, encourage scientific and cultural progress, create religions and so on. When your turn is over, the other players take their turns one by one and the cycle is repeated [16].

Differently from other games, Civilization IV offers six different victory possibilities, ranging from peaceful approaches to aggressive ones. This characteristic makes this game very interesting to this research, being one of the main reasons for selecting this platform. The game allows completely different behaviors to succeed, unlike other games in which the unique way to succeed is to defeat your opponents by attacking them.

A second important reason for selecting Civilization IV is the fact that it explicitly represents its agents preferences in models defined in its configuration files. Furthermore, these preferences are responsible for defining agents behavior. We assume, as in [6], that the computer-controlled leaders act based on their preferences.

The game represents its agents by weights on each preference, and different combinations generate different behaviors. The preferences present in the game are: (1) Culture, (2) Gold, (3) Growth, (4) Military, (5) Religion and (6) Science. The assigned weights represent a “weak” or “strong” preference, besides no preference at all (value 0). Each behavior allows the agent to seek, somehow, one of the six victory conditions: (1) Time Victory, (2) Conquest Victory, (3) Domination Victory, (4) Cultural Victory, (5) Space Race and (6) Diplomatic. All these possibilities are further described in [16].

When we try to model agents preferences we do not have a clear measure that defines a preference. Hence, we need to use indirect observations to infer these preferences. We use game score indicators as evidences for different behaviors generated from different preferences. These indicators are constantly available to all players, and we decided to use them instead of directly evaluating actions because they are a generalization of actions. Besides, due to the game structure (turn-based), we were able to define specific moments when to collect these data: at the end of each turn.

These data collected represent the agent’s game state in a given moment. Since a match may have, at most, 460 turns, we can summarize a whole match with 460 observations. In [7], we were able to show that the evolution of some of these indicators is able to represent different preferences when analyzed along several matches.

IV. Preference Modeling as a ML Problem

The approach proposed in this paper uses a ML method to model the preferences listed in Section III. In our case, supervised learning is performed, and the algorithm learns a model by finding relations between a set of features that describe the examples and a class. For the preference learning problem, each example represents a player turn, the features
are the game score indicators, and the class the presence or absence of a given preference.

However, to get to this scenario, a methodology with the following steps was proposed:

- Define relevant features for the ML algorithm according to the game,
- Select which relevant examples should be used,
- Model the preference problem as a classification task,
- Select the appropriate classification algorithm, and
- Find the best parameters configuration for the selected algorithm.

All these topics are generically discussed here, since this approach can be applied to any other game to define agents preferences.

A. Features and Data Definition

ML algorithms require features as input. For the preference modeling problem, for instance, these features should be able to represent different behaviors of agents with different preferences. This is based on the assumption that different behaviors are generated by different preferences.

This approach is completely dependent of the game being used as testbed. Once the game platform that is going to be used is defined, a study of the selected data may be useful to assure that the assumptions of the data relevance are correct. An example of this type of study is [7], where we analyzed the discriminative capacity of some Civilization IV indicators for the preference modeling task.

Despite being dependent of the game used as testbed, some general approaches may be common among games of the same genre. For example, strategy games generally present several game indicators along a match. These indicators are related to resources, military and technological characteristics of the game and they are promising to be used as relevant features for a ML technique.

In a First-Person Shooter (FPS) game, indicators derived from agents behavior observations, such as the amount of time running, the selected weapons and spots, the amount time alive, number of kills, among others, may define an agent preference. In the case of Action-adventure games, such as the Tomb Raider series, agents preferences may be defined by other indicators. Among them, we can list causes of death, total number of deaths, completion time and demand for help. In fact, all these indicators were used by Drachen et al. in [14] to define models of players in the game Tomb Raider: Underworld.

Although this approach is generic for any game genre, it is important to stress that the data availability to researchers may vary among different games. This is mainly because different game developers have different approaches to data extraction. While some allow scripts and even source code modifications, others are extremely reluctant to permit any interaction with the game different from playing it.

Apart from the attributes, for turn-based games, decisions on how much data to use may also be crucial. For instance, [6] suggested that the first turns should be ignored, because the indicators of different players evolve similarly at the beginning and are not useful to distinguish preferences.

B. Problem Modeling

Once we define relevant features for preference modeling, we need to define how to represent different preferences. This representation depends on two different aspects: if we are able to previously identify the agents’ preferences and what algorithm we will select for the task.

All the ML approaches we previously discussed may be applied to preference modeling. If the preferences information is not previously available, unsupervised learning may fit better. In this case, players are clustered according to similar attributes using a distance metric, and the researcher is expected to identify each group by its characteristics. An example of this approach can be found in [14].

On the other hand, if agents preferences are previously defined, we can classify new agents based on the predefined preferences. This is the approach presented in this paper, which uses the preferences of Civilization IV’s agents. In general, classifiers are appropriate algorithms for this task.

There are many ways to model this problem using classifiers. Perhaps the most intuitive approach is to model each preference as a classification problem. In this case, modeling n preferences requires running the classifier n times. For each classifier, we can predict different types of things. One approach is to predict different levels of preference. For example, the player can be considered to have no preference, weak preference, or strong preference for culture. This is the approach followed in [6] and [8], called a multi-class problem, where three classes are considered. We believe that the main problem with this approach, as detailed discussed in Section V, is that even human users have difficulties in defining their preferences in terms of levels, being much easier to say simply if they have a preference or not. Currently, this is not a big problem in our approach, as we are dealing with artificial players. However, a next step is to model human players, it needs to be considered. Furthermore, obtaining datasets detailing these preference levels can be more complicated, as also discussed in the experiments.

Given these drawbacks, an alternative approach, which is the one used in this paper, is to model the problem using a binary classification strategy, where we want to find if the player simply has or has not a preference for a given characteristic. Hence, a binary classifier will predict if a vector (example) belongs to the class “with preference” or not. This modeling represents one of the main difference between our approach and the one presented in [6], [8] and, as showed by experimental results, achieves better results. Another possible way of modeling this problem is to consider it as a multi-label classification problem. In this case, only one classifier would have to be run, regardless of the number of preferences being modeled. For each example, we could have many different classes (that is the reason for the name multi-label), and a single classifier could be able to predict all of them. For the
best of our knowledge, this approach was not tested yet, and will be considered as a future work.

Regarding the classification algorithms used, our first idea was to use the SMO to make our results directly comparable with those reported in [6], [8]. However, any other classifier could be used. In fact, we also ran experiments with artificial neural networks but the results achieved were worse than those obtained by the SMO. Note that the problem of choosing the best classifier for a learning problem is an open issue in the machine learning literature, tackled by the sub-area of meta-learning [17].

C. Parameters Configuration

ML algorithms are very sensible to parameters, specially those based on kernel machines, as is the case of SMO. Hence, it is very important for researchers to spend some time tuning the algorithm parameters. A common and simple approach for this task is to perform a grid search on the relevant parameters. We further discuss this topic in the next section.

If the researcher does not know the most relevant parameters to be studied, it may be useful to apply some technique that may assist him/her to select the most relevant parameters. A common approach is called $2^k$ Factorial Design [18].

An important point when talking about parameter tuning is computational time. The higher the number of examples used for training, the worse the computational time. A simple solution to this problem is to sample the dataset preserving its original features, such as class distribution. The proposed methodology encourages data sampling for parameter tuning. The process we used samples data by games and not randomly, as explained in Section V.

V. PREFERENCE MODELING IN CIVILIZATION IV

Given the methodology described in the previous section, here we apply it to model player preferences in Civilization IV. The dataset chosen for the experiments, generated from observations of Civilization IV’s AI agents gameplay, is described in Section V-A. The steps of the methodology followed to obtain the preference models for this particular game are described in Section V-B.

A. Dataset definition

The dataset used in the experiments is a modification of the dataset created by den Teuling and Spronck [6], since it already had many of the characteristics we wanted to work with. We first describe how den Teuling and Spronck obtained it, and then discuss our modifications.

Since the game Civilization IV allows developers to create code and attach to it, the data was gathered with a modification of a script easily found on the Web called AiAutoPlay [6]. This script allows the game to be played by two virtual agents, removing the requirement of human players. This is an important feature because it allows the dataset to have hundreds of matches, what would be impossible if human players had to generate these matches. Spronck and den Teuling [6] modified this script to collect, for each turn, a set of game indicators, which will be discussed next.

Spronck and den Teuling generated the dataset by initially selecting randomly six leaders in the game, and making them play against each other eight times, in a total of 40 games per leader. For each turn, information of each agent was collected. At the end, the shorter game had 240 turns while the longer took 460 turns (the maximum allowed).

We kept the same 21 features used in [6], which are game indicators available to every player during the game. These indicators are scores and counters, modified by players actions, and here we refer to them as features. They all describe some aspect of the game. Some of them are presented in Table I. A complete list, also including features’ range, is presented in [8].

<table>
<thead>
<tr>
<th>Feature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn</td>
<td>Turn number</td>
</tr>
<tr>
<td>War</td>
<td>0 = not in war; 1 = in war</td>
</tr>
<tr>
<td>Cities</td>
<td>Number of cities</td>
</tr>
<tr>
<td>Units</td>
<td>Number of units</td>
</tr>
<tr>
<td>Economy</td>
<td>Overall economic score</td>
</tr>
<tr>
<td>Industry</td>
<td>Overall industrial score</td>
</tr>
<tr>
<td>Culture</td>
<td>Overall cultural score</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Gold needed for maintenance per turn</td>
</tr>
<tr>
<td>ResearchRate</td>
<td>Amount of research gained per turn</td>
</tr>
<tr>
<td>CultureRate</td>
<td>Amount of culture gained per turn</td>
</tr>
</tbody>
</table>

Notice that the decision to use the same features used in [6] and [8] was based on the work in [7], where we showed that some indicators are able to distinguish different behaviors. We made regressions on these indicators showing that the curves obtained are statistically different for different preferences when analyzed individually.

Here we modeled each turn as a vector (example), and since evolution along the match is an important factor, we extended the set of basic features to add this notion of time. All the extensions were proposed in [8], where the authors name these new features as composite features. The composite features are presented in Table II.

Furthermore, the first 100 turns were removed from each match, following the observations made in [6], where the authors state that these first turns generally reduce the classifier accuracy since the initial turns are very similar even among agents with different preferences. At the end, the dataset generated contained 72,653 turns with each turn being a vector represented by 130 features, containing the original and composite features.

The second dataset, used on the experiment with new matches, was also presented in [6] and [8]. It contains 16,330 turns generated by six agents that are different from those six who generated the first set. Each of its turns is also represented by the same 130 features discussed above.
Tables III and IV present the number of examples belonging to each class in the test set of the original dataset (Table III) and the new dataset (Table IV). Note that we have performed a non-stratified 10-fold cross-validation procedure to generate the results for the dataset of known agents.

### Table III
**Number of samples in each class for the test sets of the original data for both experiments. Extracted from [8].**

<table>
<thead>
<tr>
<th>Preferences</th>
<th>Known Agents</th>
<th>Unknown Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Culture</td>
<td>11,828</td>
<td>-</td>
</tr>
<tr>
<td>Gold</td>
<td>11,937</td>
<td>3,023</td>
</tr>
<tr>
<td>Growth</td>
<td>14,756</td>
<td>3,183</td>
</tr>
<tr>
<td>Military</td>
<td>6,337</td>
<td>5,396</td>
</tr>
<tr>
<td>Religion</td>
<td>11,602</td>
<td>3,023</td>
</tr>
<tr>
<td>Science</td>
<td>15,296</td>
<td>2,643</td>
</tr>
</tbody>
</table>

### Table IV
**Number of samples in each class of the test set for the new datasets for both experiments. Standard deviation showed between parenthesis.**

<table>
<thead>
<tr>
<th>Preferences</th>
<th>Avg. of Known Agents</th>
<th>Unknown Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Culture</td>
<td>4.9147 (158.2)</td>
<td>16,330</td>
</tr>
<tr>
<td>Gold</td>
<td>5.0377 (311.5)</td>
<td>11,412</td>
</tr>
<tr>
<td>Growth</td>
<td>5.8219 (359.3)</td>
<td>4,981</td>
</tr>
<tr>
<td>Military</td>
<td>2.4728 (129.5)</td>
<td>5,454</td>
</tr>
<tr>
<td>Religion</td>
<td>4.7146 (241.5)</td>
<td>13,951</td>
</tr>
<tr>
<td>Science</td>
<td>5.7885 (601.3)</td>
<td>13,552</td>
</tr>
</tbody>
</table>

A cross-validation is a method traditionally used in the machine learning literature to give experiments a statistical confidence. It divides the dataset in k different folds (in our case, k = 10), where one fold is separated for test, and the other nine are used during the training process. This approach generates 10 different training and test sets, each test set being different from the other nine. The final results reported correspond to the average error over the 10 folds, in order to guarantee that the results were not found by chance according to the characteristics of the learning data. When this process is non-stratified, the folds do not preserve the class distribution of the original dataset. Hence, the figures in Table IV represent the average number of samples in each fold followed by its standard deviation. As the original dataset was not used together with a cross-validation procedure, no standard deviation is reported.

The data in these tables corroborate our discussion about modeling agents preferences as binary classes since, in the original dataset, 2 out of 6 preferences did not even have three classes.

### B. Task Modeling and Classifier

As previously said, we selected the SMO algorithm to perform the experiments using a binary classification approach, where we simply want to predict if the user has or has not a predefined preference. We chose to model it using a binary classifier based on the data showed in Table III. Hence, for each preference, a different classification model is created.

The SMO algorithm is based on the concept of Support Vector Machines (SVM), and the implementation used was the `libSVM` [19], which corresponds to the method SMO [10]. Besides, the SVM is considered the state-of-art classification algorithm. The version we worked used an RBF kernel, the `libSVM` default option.

### C. Parameter Tuning

Having decided to use the SMO algorithm, we went to parameter tuning. The best parameters configuration for each data fold were obtained using a tool called `easy`, also present in `libSVM`. In practice, we optimized two different parameters: `cost (c)` and `gamma (g)`. The `cost` parameter is responsible for evaluating the cost of a misclassification in the training examples. A low cost may imply in a simpler surface, which may misclassify some training examples but avoid overfitting, while a high value for this parameter may generate very specific surfaces, able to correctly classify all training examples and with limited generalization capability. The `gamma` parameter defines the distance influence of a single training sample. A low `gamma` means a higher influence.

Algorithm 1 presents the grid search performed by `easy` and its maximum and minimum values (defined in the tool). This algorithm is applied to each fold of the cross-validation. We are looking for `best_c` and `best_g`. The step values used are also those defined in the tool `easy`.

As observed in the algorithm, this search is computationally expensive, requiring 6,600\textsuperscript{1} different experiments with SMO. For this reason, a sampling method was used to reduce the

---

\textsuperscript{1}11 from c search × 10 from g search × 10 from 10-fold × 6 from number of preferences
Algorithm 1 Grid Search

Ensure: Best parameters $c$ and $g$

$c ← 0$, $best_c ← 0$
$g ← 0$, $best_g ← 0$

$best_accuracy ← 0$

for $c = −5$ to 15 with step 2 do
  for $g = 3$ to -2 with step -2 do
    {Evaluate accuracy with these parameters}
    if $Evaluate(2^c, 2^g) > best_accuracy$ then
      $best_c ← c$
      $best_g ← g$
    end if
  end for
end for

return $\{best_c, best_g\}$

Algorithm 2 Sample Dataset

Input: Sample percentage $perc \{0 \leq perc \leq 1\}$
Array matches containing all matches \{Each match contains all its turns\}

Output: Array sampledMatches with sampled matches

sampledMatches ← ∅
matchesWithPref ← ∅
matchesWithoutPref ← ∅

for $i = 0$ to matches.size do
  {Check if the agent of that match has the preference}
  if matches[i].preference = true then
    matchesWithPref.add(matches[i])
  else
    matchesWithoutPref.add(matches[i])
  end if
end for

shuffle(matchesWithPref)
shuffle(matchesWithoutPref)

for $i = 0$ to matchesWithPref.size × perc do
  sampledMatches.add(matchesWithPref[i])
end for

for $i = 0$ to matchesWithoutPref.size × perc do
  sampledMatches.add(matchesWithoutPref[i])
end for

return sampledMatches

72,653 vectors available for training, speeding up the learning process.

Since our dataset contained data from complete matches and their evolution, we sampled the data considering not its vectors, but its matches. In other words, we added or removed complete sets of vectors, with each set representing a whole match. It is important to stress that we sample just the training set, not the test set. In other words, we first generate a set containing 1/10 of the dataset (due to the 10-fold cross validation). This is the test set. After this step we sample the remaining matches to create the training set. We originally had 240 matches and sampled 25% of its original size, obtaining a test set with 24 matches and a training set with 54² matches. Table IV presents the size of the sampled sets. The sampling algorithm is showed in Algorithm 2.

All results reported in this paper were obtained by the best parameter configuration chosen by easy using the sampling process described above. Note that the sampling process is done with data different from those used for testing the models.

VI. EXPERIMENTAL RESULTS

The experimental phase was divided in two steps. We first predicted preferences of virtual agents who were used to create our dataset, using a standard 10-fold cross-validation procedure. Then, in a second experiment, we used the models generated in the first phase to classify a new set of agents. This second experiment was designed to evaluate the generalization capabilities of the models when predicting preferences of unknown agents, since generalization is the most important characteristic to enable the model to be used in real time during the games.

The results generated are reported in Table V, where the improvement is computed as the difference between the accuracy and the baseline divided by the baseline.

We compare the proposed method (Binary-Class SMO) with the Majority Class and the results reported by den Teuling and Spronck in [6] and [8], which we named Multi-Class SMO Baseline. The Majority Class corresponds to the percentage of the most frequent class of the dataset. For example, for the Culture preference, 67.0% of the turns were generated by agents with no preference for Culture. This means that if the classifier learned nothing and generated a model classifying every turn as “without preference”, it would obtain the reported accuracy.

The column Multi-Class SMO Baseline presents the results reported in [6] and [8]. They modeled the problem as a multi-class classification, i.e., instead of modeling preferences as existent or not, they modeled levels of preference. This result is shown just for a high-level comparison, since the modeling and number of classes were different. Thus, it is not fair to evaluate the improvement as we did for the Majority Class.

Table V shows us some interesting results. First of all, our method got better results than the one presented in [6] and [8] for two different preferences: Religion and Military, and was similar to three others, namely Culture, Growth and Science (considering the variability implicitly reported with the RMSE). We had the worse comparative result in the preference Gold, where we were near the Majority Class, but below Multi-Class SMO Baseline.

Recall that this comparison is difficult, since the problems were modeled in different ways and tested with different procedures: while binary SMO uses a cross-validation approach, the results of the multi-class SMO are based on a single
Another evidence of this sampling problem is that the results related to preferences with a high class unbalance (Growth and Science) presented better results than Culture and Gold. This result may be seen as an evidence that the sampling was not capable of selecting representative data for the whole dataset when we have a better distribution of data, thus making the task harder. These two preferences were classified with accuracies close to the Baseline in the best scenarios. The worst result, related to Gold, may be explained due to the importance of this game resource for all players. Hence, it is difficult to obtain indicators that differ players preference for Gold. Moreover, the sampling probably was not able to generalize the whole dataset, containing different behaviors of all players.

A second experiment was performed to evaluate if we can generalize the preference classification to other agents. Hence, the models generated for the first group of agents were applied to a second group, where the agents were completely different from the ones in the first group. This is an essential result to preference modeling, since it shows evidences that it is possible to learn generic models that can identify unknown players by their behavior.

Table VI presents the Majority Class, Multi-Class SMO Baseline and Binary-Class SMO (as previously explained), and two other methods that are derived from our approach: Majority Voting and Best Model.

In our first set of experiments, the binary-class SMO generated ten different models from different training sets. Table VI reports the average accuracy obtained by these 10 different models in the new set of agents. It also shows the performance of a voting system called Majority Voting, where each of the models had a vote, and we assigned as the final class the one that received more votes. Finally, the Best Model column reports the results obtained using only the model that generated the best result during the training process.

We did not compute the improvement of our method over the majority class because, in practice, our algorithm does not know the class distribution of the new data. We prefer to analyze them qualitatively, discussing how good our results would be if this method was applied to real applications.

The models with the worst generalization performance were related to the prediction of the preferences Growth and Culture. We believe that the poor accuracy achieved for Growth is due to a specific characteristic of this preference, discussed in [7]. We showed that the initial turns of the game are the most important to differ the Growth preference between agents. However, these turns were removed while preprocessing our dataset in order to make our results directly comparable with

---

**Table V**

Accuracy of our approaches (Binary-Class SMO, Majority Voting and Best Model) contrasted with the most frequent class (Majority Class) and with den Teuling’s approach [6] (Multi-Class SMO Baseline) for unknown agents, not seen in the training process. Binary-Class SMO is the average of 10-folds while Majority Voting and Best Model generate one prediction. Root Mean Squared Error (RMSE) is shown in parenthesis for Binary-Class Model, the only result with more than one answer. Since the agents who generated these data were not present on training, we do not know the class distribution. Thus, it is meaningless to evaluate improvement over Majority Class.

<table>
<thead>
<tr>
<th>Preference</th>
<th>Majority Class</th>
<th>Multi-Class SMO Baseline</th>
<th>Binary-Class SMO</th>
<th>Improvement over Majority Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Culture</td>
<td>67.0%</td>
<td>78.9% (0.46)</td>
<td>73.0% (1.19)</td>
<td>9.0%</td>
</tr>
<tr>
<td>Gold</td>
<td>67.2%</td>
<td>74.6% (0.38)</td>
<td>63.9% (7.41)</td>
<td>-4.9%</td>
</tr>
<tr>
<td>Growth</td>
<td>81.7%</td>
<td>83.5% (0.41)</td>
<td>78.0% (4.27)</td>
<td>-4.5%</td>
</tr>
<tr>
<td>Military</td>
<td>66.1%</td>
<td>61.0% (0.43)</td>
<td>100.0% (0.00)</td>
<td>51.3%</td>
</tr>
<tr>
<td>Religion</td>
<td>64.8%</td>
<td>79.0% (0.46)</td>
<td>100.0% (0.00)</td>
<td>54.3%</td>
</tr>
<tr>
<td>Science</td>
<td>82.0%</td>
<td>88.4% (0.34)</td>
<td>81.0% (7.24)</td>
<td>-1.2%</td>
</tr>
</tbody>
</table>

---

**Table VI**

Accuracy of our approach (Binary-SMO) contrasted with the most frequent class (Majority Class) and with den Teuling’s approach [6] (Multi-Class SMO Baseline). The results are the average accuracy of 10-folds. The Root Mean Squared Error (RMSE) is shown in parenthesis. Since our approach has a different number of classes, it is not fair to evaluate our improvement over den Teuling’s approach. This is the reason we present only the improvement over the Majority Class.

<table>
<thead>
<tr>
<th>Preference</th>
<th>Majority Class</th>
<th>Multi-Class SMO Baseline</th>
<th>Binary-Class SMO</th>
<th>Majority Voting</th>
<th>Best Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Culture</td>
<td>100.0%</td>
<td>88.2% (0.34)</td>
<td>30.7% (13.54)</td>
<td>29.7%</td>
<td>11.5%</td>
</tr>
<tr>
<td>Gold</td>
<td>69.9%</td>
<td>38.6% (0.50)</td>
<td>61.6% (5.29)</td>
<td>65.8%</td>
<td>67.9%</td>
</tr>
<tr>
<td>Growth</td>
<td>69.5%</td>
<td>30.8% (0.83)</td>
<td>30.5% (0.04)</td>
<td>30.5%</td>
<td>30.5%</td>
</tr>
<tr>
<td>Military</td>
<td>66.6%</td>
<td>34.6% (0.56)</td>
<td>65.3% (0.65)</td>
<td>65.5%</td>
<td>66.1%</td>
</tr>
<tr>
<td>Religion</td>
<td>83.2%</td>
<td>59.0% (0.64)</td>
<td>82.9% (0.32)</td>
<td>83.1%</td>
<td>82.6%</td>
</tr>
<tr>
<td>Science</td>
<td>83.0%</td>
<td>71.0% (0.54)</td>
<td>41.7% (11.67)</td>
<td>43.1%</td>
<td>27.0%</td>
</tr>
</tbody>
</table>
those reported by [6] and [8]. This decision is based on the assumption that the first turns are similar between all agents and they do not offer information that assist the classifier in the task of differing preferences. As will be discussed in Section VII, it may be interesting to evaluate the classifier with all turns in the future.

Poor results were also obtained for the preferences Culture and Science. Analyzing the data, we observe that the Best Model is the one that presents the worst performance. This may be an indicative of overfitting for these two preferences, which might be generated by a small training set, probably due to sampling.

The accuracy obtained for the preferences Gold, Military and Religion were the best among all (accuracies of 67.9%, 66.1% and 83.1%, respectively), an indication that the learned models are very generic. We believe these results are promising, since we were able to correctly predict two thirds of the dataset. In this case, when we compare our results to those presented by den Teuling and Spronck [6], [8] we observe that our accuracies are much higher in 3 out of 6 preferences (for instance, 67.9% over 38.6% for Gold, and 83.1% over 59% for Religion). We also had similar results for the Growth preference. We can see these results as an indicator of the quality of the binary classification modeling, which allowed us to better predict unseen agents.

VII. CONCLUSIONS AND FUTURE WORK

This paper presented a novel approach for preference modeling using machine learning techniques. Using binary classification and parameter optimization, we were able to infer agents preferences with a higher accuracy, improving some of the best known results for this problem. Moreover, as far as we know, our approach was one of the first to satisfactorily predict preferences of unseen agents, i.e., agents that were not observed nor included in the training process.

Several experiments were performed using a dataset generated from observations of Civilization IV agents gameplay. This game is specially interesting since it has six different victory conditions, allowing us to observe different agent preferences. A problem faced during experimentation was the difficulty to train large datasets and to optimize the parameters in a feasible time, what required us to sample our dataset. This sampling may have impaired our results as discussed along this paper. We also did not deal with the unbalance of some preferences in the used dataset, in which a unique class was responsible for 80% of the dataset evidences.

There are several directions for future work. Firstly, we want to perform different analyses with the data. Some interesting approaches are: to evaluate the impact of using the first 100 turns in each preference classification and to apply some feature selection method to the dataset, maybe trying to consider indicators’ semantics. We could also try to apply different kernels in the training process, looking to solve possible different structures of the dataset. Finally, the application of over/undersampling techniques in our dataset could probably improve the results for unbalanced classes.

Another important extension of this work is the application of the proposed method to human players, trying to predict their preferences using the learned models. This is one of our main research goals and we are planning to do that in the near future.

ACKNOWLEDGMENTS

We would like to thank Dr. Pieter Spronck who kindly shared with us the original dataset discussed in this paper and was constantly available to further discuss his work.

This work is partially supported by CAPES, CNPq and Fapemig.

REFERENCES