PLAYER STRATEGY MODELING IN CLASSIC ROLE-PLAYING GAME BATTLE ENVIRONMENTS

Cheuk Man Chan[1] and Robert Haralick[2]

[1]Computer Science Department, CUNY Graduate Center, New York, USA

[2]Distinguished Professor Emeritus, Computer Science Department, CUNY Graduate Center, New York, USA

ABSTRACT

Modern game developers have acknowledged the necessity of a system which adjusts the gameplay experience for gamers. Part of this system of adjustments is called dynamic difficulty adjustment which, as its name suggests, adjusts the difficulty of the game depending on information collected during gameplay. There are many approaches to accomplish this task, amongst which are to change the behavior of the computer-controlled characters according to the player's patterns of behavior detected from a survey of past actions. This paper introduces a method to collect and process information regarding player action selections to produce an estimated model of the player's strategy. The estimated player's model is then used to determine the computer characters' strategy to keep the game not too easy and not too hard.

KEYWORDS

Classification, Naïve Bayes, Data Decay, Feature Reduction

1. INTRODUCTION

People have been deriving enjoyment from games since their invention, and in recent decades the main source of such entertainment lies with video games. The popularity of video games comes from the combination of the interactivity aspect inherent in all games with the audio-visual excitement of television programs or cinematic movies. However, as with most things in life, repeated exposure of the same stimulant usually results in a reduction in its effectiveness in eliciting the anticipated psychological joyous response.

To that end, various attempts to adjust the game play experience have been tried to remedy the situation. One approach, called dynamic difficulty adjustment (DDA), can be used to continuously and subtly alter the behavior of the game to provide a constantly shifting experience for the player. To accomplish this task, the game engine typically must identify the player's pattern of behavior in order to make the appropriate adjustment befitting the situation.

This paper presents a method using the naive Bayes' methodology to estimate the behavior of the player-controlled characters in a role-playing game (RPG) environment derived from records of previous actions by the player-controlled characters along with the state of the game when the actions were performed.

The remainder of this paper will be organized as follows. Section 2 will provide a more detailed description of the problem as well as some of the approaches that have been developed to tackle the issue. Section 3 will describe what information is collected to estimate the player's decision model. Section 4 will present how the collected high-dimension data is compressed and stored. Section 5 will present the full algorithm that is the core of this paper. Section 6 will present potential future expansion on the work in this paper. Section 7 will present the conclusion summarizing the topics presented in this paper.

DOI: 10.5121/ijcsit.2024.16309

2. BACKGROUND

Many papers discussing the idea of DDA revolve around the concept of flow by Csikszentmihalyi presented in various settings like [1, 2, 3]. In its base form, the idea of flow is the ability for the person to derive enjoyment from the task he/she is performing. A key part defining the requirement for such is the coupling of the ability of the person performing the task with the difficulty of the task. If the task is too difficult, the person performing the task will feel frustrated or defeated. If the task is too easy, the person performing the task will feel bored and uninterested.

Under the assumption that as the person repeatedly performs the same task, the person's ability to perform said task will increase. To account for this, video games create static increase in difficulty as the game progresses. However, each person has their own individual starting point in terms of his/her skill level and each person's ability increases at a different rate. To address this issue, game designers seek to dynamically alter the difficulty of the task based on direct and indirect feedback from the player. These personalized dynamic changes are called dynamic difficulty adjustments.

Initially, much of the research surrounding game artificial intelligence revolves around optimizing the performance of the AI against the player. This in part is the consequence of the limitation on the processing power of the machine. However, as hardware development has advanced, so has software development in terms of various approaches toward accomplishing DDA.

The most basic approach is adjusting the difficulty of the game in accordance with the actual physical reaction by the players. Papers like [4], [5], and [6] document methods to establish DDA based on measurements from physical devices such as electroencephalograms (EEG) attached to the player. However, approaches like these are unlikely to be viable in the commercial realm because actual players are likely to be uncomfortable with such an invasion of their privacy.

Chen in [7] proposes a player-controlled difficulty adjustment system that is constructed during the design stages of the game. The idea behind this system is that the game designer will create multiple pathways, each requiring a distinct set of skills, toward a single goal and the player will decide which pathway he/she wishes to challenge. This approach has a hidden pitfall in that the decision is left entirely to the player who does not always have the best understanding of his/her own capabilities. This approach is also dependent on the form of an honor system where the player willingly selects a path that provides a greater challenge instead of taking the easiest path every time.

Most games involve a certain repeating task that is at the heart of the game experience. Papers like [8], [9], [10], and [11] utilize a case-based reasoning approach where the system first gathers information regarding the general play style from a broad population, partitions such styles into various groupings, then develops the appropriate level of response to each play style group. The adjustment phase then seeks to establish the model for the current player and match the model to the established groups. If the player model changes as the game progresses, then the DDA system will attempt to match the new model to the groups and alter the behavior of the game if the new model moves to a different group than the one identified by the system previously.

As stated, with the advent in hardware development, machines are capable of processing more complex systems. Papers like [12], [13], and [14] employ the idea of an evolutionary algorithm where the initial "population" is randomly created and the best performing amongst those "individuals" are preserved for the next generation and the others are altered slightly in a form of "mutation" before the process repeats for the next iteration. The idea behind this approach is that each subsequent generation will at worst remain equally viable due to the retained segment of the population with the possibility of improvement from the mutated portion of the population. Meanwhile, papers like [15] and [16] utilize the method called deep reinforcement learning which can, in its base form, be considered a trial-and-error approach where previously observed actions with satisfactory results are encouraged while previous actions with bad results are suppressed.

Studies involving games utilizing these techniques are primarily focused on the optimization of the algorithm, or the performance of the game AI, without providing much considering into matching the performance of the AI against the opponent, be it a human player or another game AI.

Regarding the use of DDA systems in commercial games, given the proprietary nature of such technology, there are few public reports documenting the structure of these systems. Therefore, mentions of DDA usage in commercial games typically come from magazine articles and press releases. One company, Electronic Arts, did secure a patent detailing the development of their DDA system [17]. This approach is dependent on gathering a large collection of play data from its consumers. The data is then used to construct a profile of the player to be matched to other similar profiles within the database. Based on the profile matching, the system then attempts to adjust the difficulty setting in accordance with the profile cluster. However, according to [18] and [19], Electronic Arts currently has no plans to apply the system to any of their games.

As part of the DDA system, there is a measurement of the players' response toward the various tasks within the game. That measurement will then serve as the basis for adjusting the game mechanics to adapt the difficulty of the game to the players' skill level. One means of such an adjustment is to adjust the environment where the player-controlled character is currently located as is presented in [20]. Another approach is to adjust the behavior of the computer-controlled characters in accordance with the expected behavior of the player-controlled characters. These two forms of DDA are mentioned in various magazine articles like [21], [22], and [23] with regards to commercial games.

Given that all action by any of the active characters (as opposed to an "inactive" character which does not directly impact the course of the game) has an associated gain for either the player or the game AI, if the player decision model can be established, then the expected gain of the player's turn can be computed and DDA can be performed by matching the expected gain of the computer controlled characters full set of available action choices to the expected gain of the player's response.

To that end, this paper will present a method of estimating the player decision making process using the naive Bayes' approach. The choice of naive Bayes' will allow the algorithm to simplify and compress the full high dimensional data space into a simple collection of two-dimensional tables. As data is added through time, the algorithm will make use of a data decaying technique to add bias to the most recent information. This step is added to the algorithm to address the potential changes to the player's behavior through both changes to the player-controlled characters and the game's natural environmental changes through progression of the story line. In addition, during the application of the naive Bayes' approach, an additional step to remove variables that do not provide significant amounts of information is introduced to reduce potential errors in the estimated results. The full description of the algorithm will be presented in subsequent sections.

3. DATA INPUT

Given that the algorithm is based on observations of previous player decisions, a mechanism to capture the information is necessary. Normally, the agents controlling the computer characters are embedded within the game program itself and thus have access to information regarding the player's characters. However, the player does not have access to the same information regarding the computer characters. To address the difference, this algorithm only makes use of information regarding the computer characters and the publicly available information displayed on the screen that both the player and the computer would have access to.

Let $PP = (p_1, \dots, p_G)$ be the list of player-controlled characters and $EP = (e_1, \dots, e_H)$ be the list of computer-controlled characters involved in the battle. Let $\frown$ be the concatenation function for

lists and $CP = PP \frown \text{EP} = (c_1, \dots, c_{G+H})$ be the list of all characters involved in the battle. Let $HE(c) = (he_{c1}, \dots, he_{cJ})$ be the list of all health/energy statistics (shortened to stats from this point on) of each of the characters in CP where $he_{cj} \in [0.0,1.0]$ with $j \in \{1, \dots, \text{J}\}$. Note that typically he_{c1} is the health/energy stat that determines if character c is defeated, and this algorithm will make the same assumption (Character c is defeated if $\text{he}_{c1} = 0$). Let $SE(c) = (se_{c1}, \dots, se_{cK})$ be the list of all status effects, or conditions that grant boons or disadvantages, to character c where $se_{ck} \in \{0,1\}$ with $k \in \{1, \dots, \text{K}\}$ and $se_{ck} = 0$ if the status effect denoted by se_{ck} is inactive for character c and $se_{ck} = 1$ if the status effect is active. As stated before, the only information used for this algorithm is those that pertain only to the computer-controlled characters or those that are visible on the display. Health, energy, and status effect information stored in these lists are all contained in the second category and are thus available to both the player and the agents controlling the computer characters.

The observable state is the collection of all health/energy statistics for all characters along with all status effects applied to all characters. To represent the observable state, we first need to have the combined numeric (health/energy) state of each of the characters and the combined binary (status effect) state of each of the characters. Let the combined state of all the characters' health/energy stats be denoted by

$$x = \frown_{g=1}^{G+H} HE(c_g) = \left(he_{c_1 1}, \dots, he_{c_1 J}, he_{c_2 1}, \dots, he_{c_2 J}, \dots, he_{c_{G+H} 1}, \dots, he_{c_{G+H} J}\right) \quad (1)$$

with the combined state of all the characters' status effect conditions be denoted by

$$y = \frown_{g=1}^{G+H} SE(c_g) = \left(se_{c_1 1}, \dots, se_{c_1 K}, se_{c_2 1}, \dots, se_{c_2 K}, \dots, se_{c_{G+H} 1}, \dots, se_{c_{G+H} K}\right) \quad (2)$$

The state of the battle can then be denoted by $s = (x, y)$.

In a typical RPG game, there are tens if not hundreds of different actions available to the characters. It would be impractical to consider each individual action as its own entity. Instead, our algorithm establishes action classes to group similar actions into a single collection and any computations and estimations are based on these action classes rather than each individual actions. Let the set of all action classes in the game be denoted by $C = [Q] = \{1, \dots, \text{Q}\}$, and $d = (s, q)$, $q \in C$ be a single observation of the state-action pair during a player character's turn. Given the varying skill sets of each individual character in an RPG, it would be impossible to derive any form of meaningful information regarding the player's strategy if the observations of all player-controlled characters are placed in a single list. Therefore, each player-controlled character will have its own list of observations. Let the entire record of all observed state-action pairs by player character p from turn $= 1$ to turn $= T$ be denoted by $d_p = (d_{p1}, \dots, d_{pT})$, $T \in \mathbb{Z}^+$.

With the extended game play for typical RPGs, it is impossible both in terms of storage and processing time to consider each action as an independent entity. The information must be stored in a compressed form that is still meaningful and representative of the decision-making process of the player. The data structure and compression method will be presented in Section 4.

4. DATA COMPRESSION AND STORAGE STRUCTURE

In order to compress the list of all observed data D, there are two levels of computationally infinite range that need to be addressed: the health/energy record is a real number and lies in the closed interval $[0.0,1.0]$, thus the number of possible values is uncountably infinite; and the size of the observation record can theoretically extend to a large T beyond the amount of memory available.

Recall from the previous section that the observed state $s = (x, y)$ consists of a list x of continuously valued variables denoting the health and energy level of all the characters, and a list y of discretely values variables containing information regarding the status effect conditions of all the characters. In order to compress the data, the continuous variables must be converted to discrete variables through a quantization process.

Recall from Section 3 that there is a total of G player-controlled characters and H computer-controlled characters. For each of those characters, the number of all observable health/energy statistics is J. The combined size of the numeric portion of the state, x as defined in Equation 1, is $(G + H) * J$. Similarly, the combined size of the symbolic portion of the state, y, is $(G + H) * K$. To simplify the following equation definitions, let the size of the list x be $N = |x| = (G + H) * J$ and the size of the list y be $M = |y| = (G + H) * K$. Let the finite range set $[P] = \{1, \dots, \mathrm{P}\}$ denote the quantization set for the continuous variables in the tuple x. Define quantization function quantize: $[0.0,1.0]^{\mathrm{N}} \rightarrow [\mathrm{P}]^{\mathrm{N}}$, which reduces each continuous variable in each state from the interval $[0.0,1.0]$ to a finite range set $\{1, \dots, \mathrm{P}\}$.

Define tensors $A^{N \times P \times Q}$ and $B^{M \times 2 \times Q}$. Tensor A contains the relationship between the health/energy statistics of the characters involved in the battle (defined over the N rows), the quantized value of each of those statistic (defined over the P columns), and the class of actions chosen under those conditions (defined over the Q planes). Similarly, tensor B will contain the relationship between the status effect conditions of the characters (defined over the M rows), if those status effects are activated (defined over the 2 columns), and the action class chosen under those conditions (defined over the Q planes). Initialize the two matrices as zero matrices such that $A^0_{ijk} = 0, i \in [N], j \in [P], k \in [Q]$ and $B^0_{ijk} = 0, i \in [M], j \in [2], k \in [Q]$. The two matrices will reduce the potentially infinite number of observations to a limited size as defined.

Given observation $d^t = (s^t, q^t)$ where $s^t = (x^t, y^t)$ is the state of the battle at time t and q^t be the class of the action selected by the player at time t, update the two tensors with the following functions:

$$Update_A(quantize(x^t), q^t): A^{t+1}_{ijk} = A^t_{ijk} + 1, i \in 1, \dots, N, j = quantize(x^t)_i, k = q^t \quad (3)$$

$$Update_B(y^t, q^t): B^{t+1}_{ijk} = B^t_{ijk} + 1, i \in 1, \dots, M, j = y^t_i, k = q^t \quad (4)$$

4.1. Data Decay

Unlike most classification problems where the underlying population is static, the player's decision-making process evolves over time. To account for such changes, the older information is gradually rendered obsolete, though not eliminated, by the application of the decay factor.

While the two tensors as constructed are sufficient in storing the entire collection of observations in a compressed state, there is no differentiation between newer information and older information. To address this issue an additional term, call the decay factor is required to implicitly introduce the time element into the compressed data collection.

Definition 1 (Decay Factor). The decay factor δ is the percentage of the weight assigned to a particular piece of information during the current time unit compared to the weight assigned to the same piece of information during the immediate previous time unit. A δ of 1.0 is the same as no data aging (decay).

The idea of decay factor or rate of decay has a wide range of applications. The most well-known example is that of half-life in fissure materials. A simple definition of half-life from Encyclopedia Britannica is "the interval of time required for one-half of the atomic nuclei of a radioactive sample to decay (change spontaneously into other nuclear species by emitting particles and energy)."

This decay is exponential. In other words, after one half-life, there should be $1/2$ of the original material remaining and after two half-life there should be $(1/2)^2 = 1/4$ of the original material remaining. Nuclear decay is also continuous and not discrete. In other words, the amount of original material decreases continuously at a constant rate such that after one half-life, only half of the original material remains. This allows for the computation of values in between half-life.

For example, after $1/2$ a half-life has passed, there should be $(1/2)^{1/2}$ of the original material remaining.

Given that a radioactive sample with overwhelming probability did not form exactly x half-life's ago where x is a positive integer, the process of dating a piece of radioactive material, carbon dating for example, depends on the fact that radioactive decay is a continuous process. With advancements in computer storage and processing power, the idea of continuous decay is then applied to other areas of study in data analysis. Two of the more prominent areas are business/finance and medicine with examples in [24], [25], and [26].

4.2. Estimation Process

In the context of this paper, the models for projecting player behavior are created using the Bayes theorem $p(A|B) = p(B|A)p(A)/p(B)$ where $p(A)$ is the prior probability in the data sample and $p(B)$ is the observed probability in the data sample. The method in which the projection is updated with the introduction of new information is called Bayesian inference.

Traditional Bayesian inference depends on a key assumption: that the data being observed is drawn from the same pool as previously observed data. However, by the examples shown later in this subsection, which is not necessarily the case in real world situations.

In traditional role-playing games, the environment in which the data is observed is constantly changing. This change can be due to several factors: a player abruptly changing his/her strategy, perhaps for no apparent reason; a player moving to a new area in the game world which contains a different set of enemy characters; a player character gaining a level which changes the performance of that character in battle; or a player character learning a new skill which is previously unobserved by the computer. Any of these changes can render all prior information, which includes the prior distribution $p(A)$ and the likelihood $p(B|A)$, completely irrelevant to the current situation.

To address this situation, the decision is made to decay the older observed data with each new observation. While this would require a re-computation of all parts of the Bayes theorem with each new entry, this additional computation will provide a more accurate reflection of the current state of the game. If the population from which observations are made did not change, the additional scalar multiplication will not change the resulting likelihood value. If the population did change, then the newest information will be given greater weight than prior information drawn from a different population than the one currently active, which is a more desirable outcome for the accuracy of the projection.

In terms of the selection of the decay factor, the value should be a number within the open interval (0.5, 1). The exclusion of the value 1, or no change to the weight of the data, is already mentioned. All values greater than 1 will result in older information being given greater weight than the newer information, which is counter to the purpose of using the decay factor. The elimination of values 0.5 or smaller is based on Zeno's paradox, or the fact that $\sum_{n=1}^{\infty}(1/2)^n < 1$. Therefore, if such a value is chosen as the decay factor, every single new entry to the table will dominate all the observed previous information.

With the modification through the addition of the decay factor, the counting table will display a bias toward the newest information, but as stated before, the underlying population is dynamic and older information is less reliable than the newest observations. The bias toward the most recent entries should give a better overall representation of the actual current decision-making process of the player as compared to a static, unbiased collection.

The adjusted update function is given as follows:

$$Update_A(quantize(x^t), q^t): \forall_{i\in[N],j\in[P],k\in[Q]} A_{ijk}^{t+1} = \begin{cases} \delta * A_{ijk}^t + 1 & if\ i = a, j = quantize(x^t)_a, k = q^t \\ \delta\ *\ A_{ijk}^t & otherwise \end{cases} \quad (5)$$

$$Update_B(y^t, q^t): \forall_{i\in[M],j\in[2],k\in[Q]} B_{ijk}^{t+1} = \begin{cases} \delta * B_{ijk}^t + 1 & if\ i = a, j = y_a^t, k = q^t \\ \delta\ *\ B_{ijk}^t & otherwise \end{cases} \quad (6)$$

5. CLASSIFICATION

The algorithm presented in this paper seeks to solve one simple question: given the current state, what is the probability distribution of player character actions over all action classes? As stated before, this probability distribution is estimated by the application of the Bayes' Theorem on the two matrices introduced in Section 4.

Recall that a given state s^t at time t is defined as the pair (X^t, Y^t), where $X^t = (x_1^t, \dots, x_N^t)$ with $x_i^t \in \{1, \dots, \mathrm{P}\}$ and $i \in \{1, \dots, \mathrm{N}\}$ as the numerical portion of the state and $Y^t = (y_1^t, \dots, y_M^t)$ with $y_j^t \in \{0,1\}$ and $j \in \{1. \dots, \mathrm{M}\}$ as the Boolean portion of the state. Let c^t be the action class selected by the character at time t with $c^t \in \{1, \dots, \mathrm{Q}\}$.

Bayes' Theorem states that $p(A|B) = p(B|A)p(A)/p(B)$. Given that we are dealing with probabilities using the data structures defined above, we will first define the exact terminologies.

Because we are working with probability, there are two different types of variables: random variables which carry information pertaining to the possible outcomes regarding a certain event, and normal variables which contain a specific event.

For this paper, random variables are denoted with bold-type lower case letters (e.g. $\boldsymbol{x}$), and normal variables are denoted with regular lower-case letters (e.g. x). Specifically, the following notation specifies the probability that a random variable $\boldsymbol{x}$ has a value of x: $p(\boldsymbol{x} = x)$.

Expanding upon this idea, tuples of random variables are denoted with bold-type capital letters (e.g. $\boldsymbol{X} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_N})$) and tuples of normal variables are denoted with regular capital letters (e.g. $X = (x_1, \dots, x_N)$). The following specifies the probability that random variable tuple $\boldsymbol{X}$ has the values defined in normal variable tuple X: $p(\boldsymbol{X} = X) = p(\boldsymbol{x_1} = x_1, \dots, \boldsymbol{x_N} = x_N)$.

To shorten the size of the probability function, we move the random variable outside the parenthesis as the subscript to the probability function: $p_{\boldsymbol{x}}(x) = p(\boldsymbol{x} = x)$. To extend this idea to conditional probability function, the shortened form will appear in the following format: $p_{\boldsymbol{x}|\boldsymbol{y}}(x|y) = p(\boldsymbol{x} = x|\boldsymbol{y} = y)$, and $p_{\boldsymbol{x}|\boldsymbol{Y}}(x|Y) = p(\boldsymbol{x} = x|\boldsymbol{y_1} = y_1, \dots, \boldsymbol{y_M} = y_M)$.

Since the goal of this part of the process is to discover the probability that a particular class of action c will be selected by the player given the current state $s^t = (X^t, Y^t)$, the overall theorem should be written as follows:

$$p_{\boldsymbol{c}|\boldsymbol{s}}(c|s^t) = \frac{p_{s|c}(s^t|c)p_c(c)}{p_s(s^t)} = \frac{p_{\boldsymbol{XY}|c}(X^tY^t|c)p_c(c)}{p_{\boldsymbol{XY}}(X^tY^t)} \quad (7)$$

We make the conditional independence assumption that X^t and Y^t are conditionally independent given c. This results in the formula becoming the following:

$$p_{\boldsymbol{c}|\boldsymbol{s}}(c|s^t) = \frac{p_{X|c}(X^t|c)p_{Y|c}(Y^t|c)p_c(c)}{p_{\boldsymbol{XY}}(X^tY^t)} \quad (8)$$

We make a further conditional independence assumption that each individual term in $X^t = (x_1^t, \dots, x_N^t)$ is conditionally independent given c and that each individual term in $Y^t = (y_1^t, \dots, y_M^t)$ is conditionally independent given c. Also, each individual term in X^t is conditionally

independent of each term in Y^t given c. The three independence assumptions combined states that every term in $s^t = (x_1^t, \dots, x_N^t, y_1^t, \dots, y_M^t)$ are conditionally independent of each other term given c. With this conditional independence assumption, Equation 8 becomes the following:

$$p_{c|s}(c|s^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{p_{XY}(X^tY^t)} \quad (9)$$

Given that conditional independence and marginal independence are incompatible with each other, we cannot partition the denominator in the same manner as the numerator. However, given that the Bayes Theorem computes a probability, the sum of the individual probabilities must add up to 1, Equation 9 can be rewritten as the following:

$$p_{c|s}(c|s^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{\sum_{c' \in C}\left[\prod_{n=1}^{N} p_{x_n|c'}\left(x_n^t|c'\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c'}\left(y_m^t|c'\right)\right]p_{c'}(c')} \quad (10)$$

The proof of this can be seen in the following:

Proposition 1. $p_{c|s}(c|s^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{\sum_{c' \in C}\left[\prod_{n=1}^{N} p_{x_n|c'}\left(x_n^t|c'\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c'}\left(y_m^t|c'\right)\right]p_{c'}(c')}$

Proof.

1. $p_{c|XY}(c|X^tY^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{p_{XY}(X^tY^t)}$
2. $\sum_{c \in C} p_{c|XY}(c|X^tY^t) = \sum_{c \in C} \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{p_{XY}(X^tY^t)}$
3. $1 = \frac{\sum_{c \in C}\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{p_{XY}(X^tY^t)}$
4. $p_{XY}(X^tY^t) = \sum_{c \in C}\left[\prod_{n=1}^{N} p_{x_n|c}(x_n^t|c)\right]\left[\prod_{m=1}^{M} p_{y_m|c}(y_m^t|c)\right]p_c(c)$
5. $p_{c|XY}(c|X^tY^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{\sum_{c' \in C}\left[\prod_{n=1}^{N} p_{x_n|c'}\left(x_n^t|c'\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c'}\left(y_m^t|c'\right)\right]p_{c'}(c')}$
6. $p_{c|s}(c|s^t) = \frac{\left[\prod_{n=1}^{N} p_{x_n|c}\left(x_n^t|c\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c}\left(y_m^t|c\right)\right]p_c(c)}{\sum_{c' \in C}\left[\prod_{n=1}^{N} p_{x_n|c'}\left(x_n^t|c'\right)\right]\left[\prod_{m=1}^{M} p_{y_m|c'}\left(y_m^t|c'\right)\right]p_{c'}(c')}$

Therefore, to compute the conditional probability for the class of action c given the current state s^t, $p_{c|s}(c|s^t)$, one needs to find the two components of the Bayes Theorem equation: $p_{s|c}(s^t|c) = \left[\prod_{n=1}^{N} p_{x_n|c}(x_n^t|c)\right]\left[\prod_{m=1}^{M} p_{y_m|c}(y_m^t|c)\right]$, and $p_c(c)$ for all $c \in C$.

The reason this algorithm makes the above conditional independence assumption is because of the issue of space limitation. If no conditional independence assumptions were made, then the relationship among the variables from the list $(x_1, \dots, x_N, y_1, \dots, y_M, c)$ must be established. To compute these relationships, a structure must be created to store all possible combination of values for all the variables.

Given that every variable in the list $(x_1, \dots, x_N)$ has a range of $\{1, \dots, P\}$ and every variable in the list $(y_1, \dots, y_M)$ has a range of $\{0,1\}$, and variable c has a range of $\{1, \dots, Q\}$, the size of the structure is $P^N \times 2^M \times Q$. For most real-world games, it would be impossible to create such a structure to store all the relationships among all variables and there would not be any real-world games that would produce enough data to estimate all the related probabilities.

Therefore, the conditional independence assumption is made, and the structures created in Section 4 are used to establish the relationship between each individual variables in $(x_1, \dots, x_N, y_1, \dots, y_M)$

with class assignment variable c becomes sufficient to compute the conditional probability based on Bayes theorem with the conditional independence assumption given c.

As a matter of note, the maximum likelihood estimate for the probability of an event is determined as the number of desired events divided by the number of total events. Therefore, to compute any probability values in this algorithm, we must first determine the total number of observations stored in the data structure. Recall that while the two structures A and B are defined as tensors, which are three dimensional matrices, they function as collections of two-dimensional tables. That effect can be seen in the two update functions Equation 5 and Equation 6 where the increments are applied to each table related to a single variable within the observable game state. Since each of the N $P \times Q$ tables in tensor A and each of the M $2 \times Q$ tables in tensor B are incremented by exactly δ^{T-1} with each observation, every size $P \times Q$ table in tensor A and every size $2 \times Q$ table in tensor B have the exact same total sum. Therefore, to further expedite the computation process, all calculation related to the sum of the table is done using the A_0^t table instead of computing the sum of each individual table in tensor A and B.

5.1. Feature Reduction

Recall that the classification algorithm is operating under the conditional independence assumption where the conditional probability of each individual variable in the observation is conditionally independent of each other given the class. However, given the number of variables in the observations, it is entirely possible that there are variables that do not contribute to the classification algorithm. We define such terms as having no influence or being redundant. Mathematically, the definition of a variable v having no influence on the result d is given as the following:

$$p_{\boldsymbol{d}|v}(d|v) = p_{\boldsymbol{d}}(d) \quad (11)$$

Based on the given definition, we can also conclude that if v has no influence on d, then d also has no influence on v: $p_{\boldsymbol{v}|\boldsymbol{d}}(v|d) = p_{\boldsymbol{v}}(v)$. The proof can be seen in the following:

Proposition 2. $p_{\boldsymbol{d}|v}(d|v) = p_{\boldsymbol{d}}(d) \rightarrow p_{\boldsymbol{v}|\boldsymbol{d}}(v|d) = p_{\boldsymbol{v}}(v)$

Proof.

1. $p_{\boldsymbol{d}|v}(d|v) = p_{\boldsymbol{d}}(d)$
2. $\frac{p_{\boldsymbol{d}v}(d,v)}{p_v(v)} = p_{\boldsymbol{d}}(d)$
3. $p_{\boldsymbol{d}v}(d,v) = p_{\boldsymbol{d}}(d)p_{\boldsymbol{v}}(v)$
4. $\frac{p_{\boldsymbol{d}v}(d,v)}{p_{\boldsymbol{d}}(d)} = p_{\boldsymbol{v}}(v)$
5. $p_{\boldsymbol{v}|\boldsymbol{d}}(v|d) = p_{\boldsymbol{v}}(v)$

As part of this proof, it can also be seen that if v has no influence on d, it can be concluded that v is independent of d. This is often written as $v \perp\!\!\!\perp d$.

Given that there are variables that have no influence, there are also variables with minor or small influence and variables with major or great influence. No influence is defined by the fact that the conditional probability $p(a|b)$ does not change regardless of the value of variable b. A variable b is said to have minor influence on consequence a if, by some measurement of deviation, the value of $p(a|b)$ shows a very minor deviation as the value of b changes. A variable b is said to have a major influence on consequence a if, by the same measurement of deviation, the value of $p(a|b)$ shows a major deviation as the value of b changes.

Given these definitions, if any of the variables have no influence on the result, in theory, the conditional probability of that variable given each individual class action in the result is the same. In this case, that variable can be safely eliminated from the computation of the conditional probability of the action class given the entirety of the observation. The mathematical proof can be seen in the following:

Let the tuple of random variables $\boldsymbol{V} = (\boldsymbol{v_1}, \dots, \boldsymbol{v_N})$ be the condition, with $\boldsymbol{v_i} = v_i$. Let $\boldsymbol{d}$ takes values from the set $\{1, \dots, \mathrm{K}\}$, and d is the expected value of $\boldsymbol{d}$. The naive Bayes' approach makes the following conditional independence assumption:

$$p_{\boldsymbol{d}|\boldsymbol{V}}(d|v_1, \dots, v_N) = \frac{\left[\prod_{i=1}^{N} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|d)\right] p_{\boldsymbol{d}}(d)}{p_{\boldsymbol{V}}(v_1, \dots, v_N)} \quad (12)$$

The proof in Proposition 1 shows that the naive Bayes' approach can be written as follows:

$$p_{\boldsymbol{d}|\boldsymbol{V}}(d|v_1, \dots, v_N) = \frac{\left[\prod_{i=1}^{N} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|d)\right] p_{\boldsymbol{d}}(d)}{\sum_{k=1}^{K}\left[\prod_{i=1}^{N} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|k)\right] p_{\boldsymbol{d}}(k)} \quad (13)$$

Therefore, if there is some $a \in \{1, \dots, N\}$ such that $\boldsymbol{v_a}, \dots, \boldsymbol{v_N}$ have no influence on d, then for those variables, regardless of the conditional, the probability remains the same, which is the probability of the variables. Therefore, the naive Bayes' approach can be rewritten as the following:

$$p_{\boldsymbol{d}|\boldsymbol{V}}(d|v_1, \dots, v_N) = \frac{\left[\prod_{i=1}^{a} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|d)\right]\left[\prod_{j=a+1}^{N} p_{\boldsymbol{v_j}}(v_j)\right] p_{\boldsymbol{d}}(d)}{\sum_{k=1}^{K}\left[\prod_{i=1}^{a} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|k)\right]\left[\prod_{j=a+1}^{N} p_{\boldsymbol{v_j}}(v_j)\right] p_{\boldsymbol{d}}(k)} \quad (14)$$

Given that $\left[\prod_{j=a+1}^{N} p_{\boldsymbol{v_j}}(\mathrm{v_j})\right]$ is in both the numerator and the denominator of the equation, it can be removed from the equation, so the end-product is the following:

$$p_{\boldsymbol{d}|\boldsymbol{V}}(d|v_1, \dots, v_N) = \frac{\left[\prod_{i=1}^{a} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|d)\right] p_{\boldsymbol{d}}(d)}{\sum_{k=1}^{K}\left[\prod_{i=1}^{a} p_{\boldsymbol{v_i}|\boldsymbol{d}}(v_i|k)\right] p_{\boldsymbol{d}}(k)} \quad (15)$$

If we look at how the algorithm operates, a much less stringent requirement is required. Note that the input into the algorithm is provided in the form of the state s^t. In other words, the specific condition for the conditional probability is given. Therefore, to remove specific variables from the naive Bayes' approach in this algorithm, rather than requiring full independence between a given input variable and the action class, the system only requires that, given a specific value for the input variable (for example x_i) in question, the conditional probability of the input variable given the action class be the same for all action classes $p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_1) = \cdots = p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_Q)$. So long as this condition is satisfied, the same term canceling method performed on the naive Bayes' approach done in the case of full independence between the given variable and action class can be performed considering how the naive Bayes' Theorem is applied.

However, while the elimination of variables with no influence on the result can easily be done in theory based on the definition of no influence that the conditional probability is the same regardless of the value assigned to d, it is much more difficult to do when the conditional probability is a mere estimation of the true probability given the available sample, as sampling variations will, more likely than not, create a false perception that there is a minor influence even if that is not the case in the true probability. These slight variations, which preclude the naive Bayes' theorem from excluding terms that does not contribute to the computation (i.e. $\prod_{j=a+1}^{N} p_{\boldsymbol{v_j}}(\mathrm{v_j})$ in the equations above), are all sources of errors that create a worse estimation than what could have been achieved if those terms were removed instead. In addition, given that

each of the $p_{v_j}(\mathrm{v_j})$ terms in the product are a source of error, the more terms are involved, the greater the potential for error in the computation of the result.

The performance of the algorithm may be improved if such features that have little to no influence on the result were to be eliminated from the computation. In order to eliminate such features, we must first identify properties of those features that separate them from features that have significant influence on the result. A key property is that features with little to no influence produce probabilities that are much closer to each other compared to features that have significant influence. This is due in part to the fact that the gap between these values is the result of sampling variations, which, given sufficient sample size, is relatively minor compared to the gap from the actual underlying probability distribution, which should be more significant. Therefore, the process of eliminating the features with little to no influence will be dependent on how similar the values of the estimated conditional probabilities drawn from the observed data are to each other.

Recall that the state $s^t = (X^t, Y^t)$ where $X^t = (x_1^t, \dots, x_N^t)$ such that $x_n^t \in \{1, \dots, \mathrm{P}\}$ and $n \in \{1, \dots, \mathrm{N}\}$, $Y^t = (y_1^t, \dots, y_M^t)$ such that $y_m^t \in \{0,1\}$ and $m \in \{1, \dots, \mathrm{M}\}$, and t is the current turn index. The $p_{x_i|c}(x_i^t|c_k)$ and $p_{y_j|c}(y_j^t|c_k)$ portion of the naive Bayes' theorem represents the estimated conditional probability of the class assignment given the current state derived from the $N \times P \times Q$ tensor A and the $M \times 2 \times Q$ tensor B where Q is the size of the action class set. However, as stated before, it may not be the case that all N components in $X^t = (x_1^t, \dots, x_N^t)$ and all M components in $Y^t = (y_1^t, \dots, y_M^t)$ are useful features for defining the current strategy.

To eliminate potential noisy features, the standard deviation (σ) and the one minus entropy (H') of each class probability distribution is computed. Given a numeric list z, the function definition for the mean (μ), standard deviation (σ), entropy (H) and one minus entropy (H') are given in the following:

$$\mu(z) = \frac{\sum_{i \in z} i}{|z|} \quad (16)$$

$$\sigma(z) = \sqrt{\frac{\sum_{i \in z} i - \mu(z)}{|z|}} \quad (17)$$

$$H(z) = -\sum_{i \in z} i \log_2 i \quad (18)$$

$$H'(z) = 1 - \frac{H(z)}{H(1/|z|, \dots, 1/|z|)} \quad (19)$$

Given state $s^t = (X^t, Y^t)$, compute N-tuple $\sigma_A = (\sigma_{A,1}, \dots, \sigma_{A,N})$ with $\sigma_{A,i} = \sigma\left(p_{x_i|c}(x_i|c_1), \dots, p_{x_i|c}(x_i|c_Q)\right)$, or standard deviation of the conditional probability of x_i^t derived from tensor A and partial state X^t, and M-tuple $\sigma_B = (\sigma_{B,1}, \dots, \sigma_{B,M})$ with $\sigma_{B,j} = \sigma\left(p_{y_j|c}(y_j|c_1), \dots, p_{y_j|c}(y_j|c_Q)\right)$, or the standard deviation of the conditional probability of y_j^t derived from tensor B and partial state Y^t. Let $\sigma = (\sigma_{A,1}, \dots, \sigma_{A,N}, \sigma_{B,1}, \dots, \sigma_{B,M})$.

Note that in standard deviation, the larger the difference in the given list of probabilities, the larger the result of the two metrics is. However, entropy's behavior is completely opposite. In other words, the more similar the given list of probabilities (or the more similar the list of probability is to a uniform distribution), the higher the entropy value. To maintain a consistent ordering with the other two metrics, instead of using the standard entropy formula in Equation 18, the one-minus entropy in Equation 19 is used instead.

In addition, given that $\log 0 = \infty$, the direct computation of $x \log x$ is impossible if $x = 0$. By L'Hôpital's Rule, $\lim_{x \to 0} x \log x = 0$. However, during the actual implementation, the computer

program cannot resolve this computation on its own. Therefore, a special case must be constructed for Equation 18 in the following manner:

$$H_{lh}(x) = \begin{cases} x\ log_2\ x & x \neq 0 \\ 0 & x = 0 \end{cases} \quad (20)$$

$$H(z) = -\sum_{i \in z} H_{lh}(i) \quad (21)$$

Note that this definition creates an additional issue during implementation of floating-point comparison. To handle that issue, we use the relative difference function in Equation 22 is constructed to conduct floating point comparisons between a and b.

$$rd(a,b) = \frac{|a-b|}{0.5*(a+b)} \quad (22)$$

The algorithm will consider two values a and b equal if $rd(a,b) < 0.000000001$.

Beyond the given issue, most theorems and laws related to entropy are dependent on the fact that the input forms a probability distribution. In other words, using the z term used in Equation 21, $\forall_{i \in z} 0 \leq z \leq 1$ and $\sum_{i \in z} i = 1$. However, given that the input in this case are the lists $\sigma_{A,i} = \left(p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_1), \dots, p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_Q)\right)$ and $\sigma_{B,j} = \left(p_{\boldsymbol{y_j}|\boldsymbol{c}}(y_j|c_1), \dots, p_{\boldsymbol{y_j}|\boldsymbol{c}}(y_j|c_Q)\right)$, the input does not add up to 1 and thus does not form a probability distribution.

The main conflict that this creates is the fact that the one-minus entropy equation defined in Equation 19 uses the law that the highest entropy is drawn from the uniformed distribution. However, that is only true if what is compared to that value is also derived from a probability distribution. If the input into the entropy function is not a probability distribution, it can result in an entropy value larger than the entropy drawn from the uniform distribution resulting in a negative one-minus entropy value.

To address this issue, we compute the estimated one-minus entropy value by first normalizing the input tuple to force the input into a probability distribution, which will guarantee that the denominator in Equation 19 is at minimum as large as the computed entropy. The edited functions can be seen in the following:

$$Norm(i,z) = \begin{cases} 0 & \Sigma_{j \in z} j = 0 \\ i / \Sigma_{j \in z} j & otherwise \end{cases} \quad (23)$$

$$Norm(z) = \left(Norm(i,z)\right)_{i \in z} \quad (24)$$

$$H(z) = -\sum_{i \in Norm(z)} H_{lh}(i) \quad (25)$$

Given state $s^t = (X^t, Y^t)$ as the previous metric, compute N-tuple $H'_A = \left(H'_{A,1}, \dots, H'_{A,N}\right)$ with $H'_{A,i} = H'\left(p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_1), \dots, p_{\boldsymbol{x_i}|\boldsymbol{c}}(x_i|c_Q)\right)$, and M-tuple $H'_B = \left(H'_{B,1}, \dots, H'_{B,M}\right)$ with $H'_{B,j} = H'\left(p_{\boldsymbol{y_j}|\boldsymbol{c}}(y_j|c_1), \dots, p_{\boldsymbol{y_j}|\boldsymbol{c}}(y_j|c_Q)\right)$. Let $H' = \left(H'_{A,1}, \dots, H'_{A,N}, H'_{B,1}, \dots, H'_{B,M}\right)$.

There are two different paths toward accomplishing the task of feature reduction. The first path makes use of a simple scalar value to control which terms in the defined variance values list (σ or H′) are to be included in the computation of the estimated probability distribution. The second path makes use of order statistics in order to determine which terms in the defined variance values list are to be included. Both methods are presented in the following subsections.

5.1.1. Feature Reduction with Scalar Value

Let λ be the feature reduction factor, define the (**F**)eature (**R**)eduction vector for tensor A, $FR_A = (FR_{A,1}, \dots, FR_{A,N})$ and feature reduction vector for tensor $\boldsymbol{B}$, $FR_B = (FR_{B,1}, \dots, FR_{B,M})$ by the following:

$$FR_{A,i} = \begin{cases} 1 & \sigma_{A,i} \geq \mu(\sigma) * \lambda \\ 0 & otherwise \end{cases} \quad (26)$$

$$FR_{B,i} = \begin{cases} 1 & \sigma_{B,i} \geq \mu(\sigma) * \lambda \\ 0 & otherwise \end{cases} \quad (27)$$

If standard deviation is used as the feature reduction function and

$$FR_{A,i} = \begin{cases} 1 & H'_{A,i} \geq \mu(H') * \lambda \\ 0 & otherwise \end{cases} \quad (28)$$

$$FR_{B,i} = \begin{cases} 1 & H'_{B,i} \geq \mu(H') * \lambda \\ 0 & otherwise \end{cases} \quad (29)$$

If one-minus entropy is used as the feature reduction function.

Equation 26 and Equation 28 assign the value 1, or true in Boolean terms, if the standard deviation/one-minus entropy of a given observation x_i^t is greater than the mean of the standard deviations/mean of the one-minus entropy of all x_j^t such that $j \in \{1, \dots, N\}$ times the feature reduction factor, and has the value 0, or false in boolean terms, otherwise. Similarly, Equation 27 and Equation 29 assigns the value 1, or true in Boolean terms, if the standard deviation/one-minus entropy of a given observation y_i^t is greater than the mean of the standard deviations/mean of the one-minus entropy of all y_j^t such that $j \in \{1, \dots, \mathrm{M}\}$ times the feature reduction factor, and has the value 0, or false in Boolean terms, otherwise.

5.1.2. Feature Reduction with Order Statistic

Recall that the full list of standard deviation values σ and one-minus entropy values H' has already been computed when the algorithm has reached this point. Instead of finding the mean as would be done if a scalar value is used as the feature reduction factor, order statistic requires that the maximum value and the minimum value of the respective list be found. Once the maximum and minimum are found, the range between the two values are partitioned into equal interval bins and each value in the given list is placed in the respective bins.

This approach will only select the conditional probability correlating to the top λ portion of the sorted σ or H' list. Rather than sorting the full list, given that the values are placed in the corresponding equal interval bins, the process will begin by accepting values from the bin with the highest values. If the number of terms accepted plus the number of terms in the entire next bin is less than the targeted number, the process will add all the terms in the bin to the accepted list and continue to the next bin. If the number of terms accepted plus the number of terms in the entire next bin is equal to the targeted number, the process will add all the terms in the bin to the accepted list and return the result. If the number of terms accepted plus the number of terms in the entire next bin is greater than the targeted number, only the term in the next bin is sorted and the terms will be accepted in order from largest to smallest until the targeted number is reach and the result is returned. Note that this process works best if the number of variables and thus the size of either σ or H' is large. If the size of σ and H' are sufficiently small, it may be more efficient to sort the full list outright and take the top targeted number of terms.

For this approach, the feature reduction factor λ must be in the range (0.0, 1.0], and it represents the percentage of the highest values in the list. The open lower end of the range is representative of the fact that a λ of 0.0 in this case means the elimination of every term in σ or H' depending on the method used. The upper bound of 1.0 is representative of the fact that 100% of σ or H' are included and a value greater than 100% does not make any sense in this situation.

The feature reduction function FR is then defined as the following:

$$FR_{A,i} = \begin{cases} 1 & \left|\{s|\sigma_{A,i} \leq s, s \in \sigma\}\right| \leq |\sigma| * \lambda \\ 0 & otherwise \end{cases} \quad (30)$$

$$FR_{B,i} = \begin{cases} 1 & \left|\{s|\sigma_{B,i} \leq s, s \in \sigma\}\right| \leq |\sigma| * \lambda \\ 0 & otherwise \end{cases} \quad (31)$$

If standard deviation is used as the feature reduction function and

$$FR_{A,i} = \begin{cases} 1 & \left|\{s|H'_{A,i} \leq s, s \in H'\}\right| \leq |H'| * \lambda \\ 0 & otherwise \end{cases} \quad (32)$$

$$FR_{B,i} = \begin{cases} 1 & \left|\{s|H'_{B,i} \leq s, s \in H'\}\right| \leq |H'| * \lambda \\ 0 & otherwise \end{cases} \quad (33)$$

If one-minus entropy is used as the feature reduction function.

In the each of the equations above, there is a counting function where the left side of the divider | indicates the number of terms to be counted and the right side indicates the conditions that those terms need to satisfy. For example, in Equation 30, the function $\left|\{s|\sigma_{A,i} \leq s, s \in \sigma\}\right|$ means the number of s such that $\sigma_{A,i}$ is less than or equal to s, and s is in the list σ. In other words, count the number of terms in σ that are greater than or equal to $\sigma_{A,i}$. When combined with the right side of the inequality $\leq |\sigma| * \lambda$, the entire condition translates to if the number of terms in σ that are greater than or equal to $\sigma_{A,i}$ is less than or equal to the size of σ times λ. In other words, the condition is true if $\sigma_{A,i}$ is smaller than no more than λ portion of the entire σ list, or $\sigma_{A,i}$ is in the top λ portion of σ.

The same definition can be applied to Equation 32, Equation 31, and Equation 33 with the only difference being the tensor in which the value is generated from (A or B) and the method used for feature reduction (standard deviation or one-minus entropy).

5.1.3. Estimating Probability Distribution Using Feature Reduction

While the normal instinct is to apply the feature reduction vector as a scalar to the conditional probability, it is important to note that in the naive Bayes' approach, the conditional probabilities are combined with a product ($\prod$) and not a summation ($\sum$). If any term within the product sequence is 0, then the result of the product will be 0. Therefore, to allow only the conditional properties that have a sufficiently high level of deviation to be part of the product sequence, an additional function, using $FR_{A,i}$ and $FR_{B,i}$ having boolean values instead of scalar values, is necessary.

The purpose of feature reduction is to extract a probability distribution that has a significant deviation. To that end, the algorithm defines function $CPS_{A,i}$ (for (**C**)onditional (**P**)robability (**S**)elector) as the modified conditional probabilities drawn from tensor A for variable x_i^t and $CSP_{B,i}$ as the modified conditional probabilities drawn from tensor B for variable y_i^t. For cases such that the deviation within the probability distribution for a given variable x_i^t or y_i^t is significantly high ($FR_{A,i} = 1$ or $FR_{B,i} = 1$), the function will return the probability distribution as is for its application within the Bayes Theorem. However, for those distribution where the deviation is not high enough ($FR_{A,i} = 0$ or $FR_{B,i} = 0$), the function returns a size Q vector of 1s to remove the influence of the probability distribution on the result of the application of the Bayes Theorem.

$$CPS_{A,i} = \begin{cases} \left(p_{x_i|c}(x_i|c_1), \dots, p_{x_i|c}(x_i|c_Q)\right) & FR_{A,i} = 1 \\ \underbrace{(1, \dots, 1)}_{Q\ times} & FR_{A,i} = 0 \end{cases} \quad (34)$$

$$CPS_{B,i} = \begin{cases} \left(p_{y_j|c}(y_j|c_1), \dots, p_{y_j|c}(y_j|c_Q)\right) & FR_{B,i} = 1 \\ \underbrace{(1, \dots, 1)}_{Q\ times} & FR_{B,i} = 0 \end{cases} \quad (35)$$

Definition 2 (Feature Reduction Factor). The feature reduction factor λ is the percentage of either the mean of the list of standard deviation $\mu(\sigma)$ or the mean of the list of one-minus entropy $\mu(H')$ of the estimated probability distribution generated by each individual variable of the observed data. If a standard deviation of the estimated probability distribution σ is less than the product of the feature reduction factor and the mean standard deviation of the estimated probability distribution $\mu(\sigma)$ or the one-minus entropy of the estimated probability distribution H' is less than the production of the feature reduction factor and the mean one-minus entropy of the estimated probability distribution $\mu(H')$, then the variable that produced the estimated probability distribution is considered noise data. Given that standard deviations are always greater than or equal to 0.0, a λ of 0.0 is the same as no feature reduction.

The original probability distribution equation, which is the Bayes' theorem with conditional independence of the observed state given the observed action class applied, is given as the following:

$$distribution^t(s^t) = \left(\frac{\left[\prod_{i=1}^{N} p_{x_i|c}\left(x_i^t|c_k^t\right)\right]\left[\prod_{j=1}^{M} p_{y_j|c}\left(y_j^t|c_k^t\right)\right]p_c(c_k^t)}{\sum_{q=1}^{Q}\left[\prod_{i=1}^{N} p_{x_i|c}\left(x_i^t|c_q^t\right)\right]\left[\prod_{j=1}^{M} p_{y_j|c}\left(y_j^t|c_q^t\right)\right]p_c(c_q^t)}\right)_{k=1}^{Q} \quad (36)$$

The final classification function with the feature reduction factor is the following:

$$distribution^t(s^t) = \left(\frac{\left[\prod_{i=1}^{N} CPS_{(A,i)_k}\right]\left[\prod_{j=1}^{M} CPS_{(B,j)_k}\right]p_c(c_k^t)}{\sum_{q=1}^{Q}\left[\prod_{i=1}^{N} CPS_{(A,i)_k}\right]\left[\prod_{j=1}^{M} CPS_{(B,j)_k}\right]p_c(c_k^t)}\right)_{k=1}^{Q} \quad (37)$$

6. FUTURE WORKS

In reviewing how the modified Naive Bayes' approach shown in Equation 37 is formed with the addition of feature reduction process, the "removed" conditional probabilities are replaced by the value of 1 as can be seen in Equation 34 and Equation 35. By this method, if every variable is removed from the computation, then every conditional probability will be replaced by the value of one and the result of Equation 37 will be reduced to the class probability $p(c)$. This raises the question if such a result is appropriate or if the result of such cases be reduced to the default uniform distribution. Both the logical reasoning behind the two potential approaches and the approaches' impact on the overall performance of the algorithm require further examination.

In addition, the fact that the estimation approach requires the computation of the product of all the conditional probability of the various variables from the observed state creates the potential where the estimated probability resolves to 0.0 through the simple means of having a single variable with a conditional probability of 0.0 for the given class. By extension, if, given all the variables within the observed state, the resulting estimated probability of every class given the current state resolves to 0.0, the approach will return an estimation of the default uniform distribution. Much like the first point, it should be studied if such a result is logically reasonable or if a different estimation be produced. Furthermore, if a different estimation is to be produced, what or how the estimation is formed should be further examined.

7. CONCLUSION

This paper introduced a method to estimate a model of the player's strategy using the naive Bayes' approach. As part of this algorithm, the system collects player action selection information to serve as the basis for producing the estimated models. To handle the issues of shifting player

behavior as well as directing the focus of the algorithm only toward pertinent information, the paper also introduces two control variables: decay factor and feature reduction factor.

The first control variable, decay factor, is applied during the information collection phase of the algorithm. The decay factor's effect is most prominent in cases where there is an alteration to the player's behavior. However, given that the decay factor places the emphasis on the most recent information, if there is no change in the player's behavior, it may result in a less accurate estimate.

The second control variable, feature reduction factor, is used in conjunction with the naive Bayes' approach to remove features that do not have a major impact on the computation of the conditional probability. This method takes advantage of the simplicity of the naive Bayes' approach to, through a method akin to factoring, remove random variables that do not have a significant contribution to the overall computation of the conditional probability.

The next step of the work on this algorithm is to test the system in a game environment. Since this approach is used in conjunction with the battle system toward identifying the player's likely next move given a game state, the most reasonable testing ground for this algorithm would be a simulation of the battle system of a turn-based role-playing game.

REFERENCES

[1] Csikszentmihalyi, M. (1990). Flow: The psychology of optimal experience, Volume 1990. Harper & Row New York.

[2] Csikszentmihalyi, M. (1997). Finding flow: The psychology of engagement with everyday life.

[3] Csikszentmihalyi, M. (2004). Mihaly Csikszentmihalyi: Flow, the secret to happiness [video file]. TED Talks in Monterey, California.

[4] Park, S., H. Sim, and W. Lee (2014). Dynamic game difficulty control by using eeg-based emotion recognition. International Journal of Control and Automation 7(3), 267–272.

[5] Stein, A., Y. Yotam, R. Puzis, G. Shani, and M. Taieb-Maimon (2018). Eeg-triggered dynamic difficulty adjustment for multiplayer games. Entertainment computing 25, 14–25.

[6] Alves, T., S. Gama, and F. S. Melo (2018). Flow adaptation in serious games for health. In 2018 IEEE 6th International Conference on Serious Games and Applications for Health (SeGAH), pp. 1–8. IEEE.

[7] Chen, J. (2006). Flow in games. a jenova chen mfa thesis. University of Southern California.

[8] Bakkes, S., P. Spronck, and J. Van Den Herik (2008). Rapid adaptation of video game ai. In 2008 IEEE Symposium On Computational Intelligence and Games, pp. 79–86. IEEE.

[9] Bakkes, S., P. Spronck, and J. Van den Herik (2009). Rapid and reliable adaptation of video game ai. IEEE Transactions on Computational Intelligence and AI in Games 1(2), 93–104.

[10] Bakkes, S., P. Spronck, and J. Van Den Herik (2011). A cbr-inspired approach to rapid and reliable adaption of video game ai. In Case-Based Reasoning for Computer Games Workshop at the International Conference on Case-Based Reasoning (ICCBR), pp. 17–26. Citeseer.

[11] Lora-Ariza, D. S., A. A. Sánchez-Ruiz, P. A. González-Calero, and I. Camps-Ortueta (2022). Measuring control to dynamically induce flow in tetris. IEEE Transactions on Games 14(4), 579–588.

[12] Tang, Z., Y. Zhu, D. Zhao, and S. M. Lucas (2020). Enhanced rolling horizon evolution algorithm with opponent model learning: Results for the fighting game ai competition. arXiv preprint arXiv:2003.13949.

[13] Dockhorn, A., M. Kirst, S. Mostaghim, M. Wieczorek, and H. Zille (2022). Evolutionary algorithm for parameter optimization of context steering agents. IEEE Transactions on Games.

[14] Gaina, R. D., S. Devlin, S. M. Lucas, and D. Perez-Liebana (2021). Rolling horizon evolutionary algorithms for general video game playing. IEEE Transactions on Games 14(2), 232–242.

[15] Nam, S.-G., C.-H. Hsueh, and K. Ikeda (2021). Generation of game stages with quality and diversity by reinforcement learning in turn-based rpg. IEEE Transactions on Games 14(3), 488–501.

[16] Oh, I., S. Rho, S. Moon, S. Son, H. Lee, and J. Chung (2021). Creating pro-level ai for a real-time fighting game using deep reinforcement learning. IEEE Transactions on Games 14(2), 212–220.

[17] Aghdaie, N., J. Kolen, M. M. Mattar, M. Sardari, S. Xue, K. A.-U. Zaman, and K. A. Moss (2018, March 20). Dynamic difficulty adjustment. US Patent 9,919,217.

[18] Peppiatt, D. (2021, 04). Ea's 'dynamic difficulty' system wants to predict your behaviour, keep you playing for longer. https://www.vg247.com/ea-dynamic-difficulty-system. Access: 2023-05-31.

[19] ElectronicArts (2021). Fair play & dynamic difficulty adjustment. https://www.ea.com/en-gb/news/fair-play-and-dynamic-difficulty-adjustment. Accessed: 2023-05-31.

[20] Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp. 429–433.

[21] Nunneley-Jackson, S. (2009b, 06). Left 4 dead 2 set in the southern us, has new characters, loads more. https://www.vg247.com/left-4-dead-2-set-in-the-southern-us-has-new-characters-loads-more. Access: 2023-05-31.

[22] Nunneley-Jackson, S. (2009a, 04). The darkside chronicles features dynamic difficulty system. https://www.vg247.com/the-darkside-chronicles-features-dynamic-difficulty-system. Access: 2023-05-31.

[23] Baranowski, J. (2021, 12). Games you didn't know featured dynamic difficulty. https://www.svg.com/138490/games-you-didnt-know-featured-dynamic-difficulty/. Access: 2023-05-31.

[24] Li Sheng, L. (2015). What is data decay and how does it affect your business? Tech in Asia.

[25] Sullivan, K. (2020). How data decay can spoil your small business database. Enigma.

[26] Chen, J. H., M. Alagappan, M. K. Goldstein, S. M. Asch, and R. B. Altman (2017). Decaying relevance of clinical data towards future decisions in data-driven inpatient clinical order sets. International journal of medical informatics 102, 71–79.