UNIVERSITY OF TRIESTE

Department of Mathematics, Informatics and Geosciences

Bachelor's Degree in
Artificial Intelligence & Data Analytics

Anomaly Detection in a virtual environment

July 19, 2024

Candidate	Supervisor
Gabriele Gianuzzo	**Prof. Luca Manzoni**

A.Y. 2023/2024

Abstract

The online video game industry has experienced a significant growth over the last two decades becoming one of the most lucrative and dynamic sectors of all time. With the growing popularity of video games, there has also been an increase in players who cheat to gain an unfair advantage. Unfortunately, traditional anti-cheating systems have difficulty in detecting new exploits. Consequently, new models based on machine learning algorithms, that can analyze large amounts of data, could be more efficient in finding cheaters.

Contents

Introduction

In recent decades, the world of the video game industry has gained increasing popularity and media attention. It has allowed not only moments of entertainment, but also social interaction, especially during the global pandemic when they became a tool for maintaining social connections. This growing popularity has enabled the sector to achieve rapid economic growth, with billions of dollars generated annually. [1]

One of the key concepts that capture players on different platforms is competitiveness. Whether it's against other people or the program itself, players want to face and overcome challenges to gain a sense of accomplishment and achievement. However, not everyone is willing to spend dozens or even hundreds of hours improving in a video game to excel, which is why the phenomena of cheating started. Cheaters are people who use third-party software to gain unfair advantages over other players.

Video game companies, in order to maintain a healthy environment and to avoid spreading a sense of frustration among honest players, develop anti-cheat systems to eliminate the problem. Despite this, especially in online games, identifying cheater can be extremely complicated. Traditional anti-cheat softwares rely on predefined rules and patterns to find malevolent players. On the other hand cheaters's software evolves and find new ways to avoid being detected.

One interesting approach to solve the problem could be the use of machine learning. This is an area that has not yet been fully explored and could greatly simplify the detection of cheaters based on a more statistical and mathematical approach to identify anomalous behaviour. Analyzing in-game data such as player position, points of view and actions perform, machine learning algorithms can learn to distinguish between legit and suspicious players.

This thesis aims to explore the possibilities of using machine learning algorithms as an anticheat system. Specifically, we are going to analyze a multivariate time series from the game Counter-Strike: Global Offensive, a popular first person shooter game with a competitive scene. Thanks to in-game data that explains player actions we try to create a robust and effective model to identify cheaters.

Chapter 1

Dataset Analysis

1.1 Dataset Description

The dataset was taken from Kaggle's site[1], it is described as the collection of viewangles of cheaters and legit players playing Counter Strike: Global Offensive. It consists of engagement data between Counter-Strike players, specifically focusing on interactions that could potentially identify cheating behavior. The data is organized as numpy arrays and captures both legit and cheating players.

Player Categories:

- Legit Players: 10,000 players who are not cheater.
- Cheaters: 2,000 players who are cheater.

Data Format:

- The data is stored in numpy arrays with a shape of (players, 30, 192, 5).
- Each player has data for 30 engagements, each engagement is represented by 192 timesteps and for each of those there are 5 related variables.

[1] `https://www.kaggle.com/datasets/emstatsl/csgo-cheating-dataset`

1.1.1 Timesteps

In Counter Strike: Global Offensive the timesteps is called a tick. This term indicates the frequency at which the server updates the game state. It is a measure of how many time per second the server store new values such as player movements, shots fired and other in game actions. The tick rate is the number of server updates per seconds, it is usually measured in Heartz and common value are 32, 64 and 128. A higher tick rate generally implies better accuracy and responsivness. The idea is: more frequent the updates are, the more accurate are the measurements. Operation of a tick can be describe as follows:

- Player input: Each player send their inputs (Key pressed, movements, actions) to the server
- Server processing: Server collects all the inputs from the players and update the game state accordingly. New positions are calculated as well as other game dynamics, such as if a shot hit the target.
- Update: Server then sends the updated game state to all players

In our dataset 192 ticks are registered, the tick rate is 32 which means that six seconds of actions are observed, 5 seconds before the engagement and 1 second after.

1.1.2 Input Variables

For each player, each action and each timestep, there are 5 feature in the dataset. This variables describe the basic input registered, they are the key features on which our analyses is based. The data are mainly focus on mouse input, especially the movement and the firing action. We are giving a quick description of each variable below.

AttackerDeltaYaw

It refers to the angular difference in degrees between the direction a player is facing and the direction they are being attacked from. It is calculated around the vertical axis, it represent the horizontal direction a player is aiming with respect to the other. We can compute the attacker delta yaw with

$$\Delta\theta = \theta_A - \theta_B$$

This value is then normalized to be within the range $[-180°, 180°]$. where

- θ_A is the yaw angle of the attacking player.
- θ_B is the yaw angle of the attacked player.

A practical example could be if B player is watching straight ahead with an angle of [0°] and A player is aiming at B's back with an angle of [180°] In this case, the attacker delta yaw would be:

$$\Delta\theta = 180^\circ - 0^\circ = 180^\circ$$

AttackerDeltaPitch

It refers to the angular difference in degrees between the vertical direction a player is facing and the vertical direction they are being attacked from. It represent the vertical direction a player is aiming with respect to the other. We can compute the attacker delta pitch with:

$$\Delta\phi = \phi_A - \phi_B$$

where

- ϕ_A is the pitch angle of the attacking player.
- ϕ_B is the pitch angle of the attacked player.

A practical example could be if B player is watching upwards, with an angle of [50°] and A player is aiming slightly downwards, with an angle of [-20°]. In this case, the attacker delta pitch would be:

$$\Delta\phi = -20^\circ - 50^\circ = -70^\circ$$

CrosshairToVictimYaw

It refers to the angular difference between the direction a player's crosshair is pointing and the direction to the target from the attacker's prospective. This determine how much the attacker needs to adjust their aim on the horizontal axis in order to match the target. We can compute the crosshair to victim yaw with:

$$\Delta\theta = \theta_V - \theta_C$$

This value is then normalized to be within the range $[-180^\circ, 180^\circ]$. where

- θ_C is the yaw angle of the player's crosshair.
- θ_V is the yaw angle to the victim.

A practical example could be if the player's crosshair is pointing straight ahead, with an angle of [0°] and the victim is to the player's right, with an angle of [70°] In this case, the CrosshairToVictimYaw would be:

$$\Delta\theta = 70^\circ - 0^\circ = 70^\circ$$

CrosshairToVictimPitch

It refers to the angular difference between the direction a player's crosshair is pointing and the direction to the target from the attacker's prospective. This determine how much the attacker needs to adjust their aim on the vertical axis in order to match the target. We can compute the crosshair to victim pitch with:

$$\Delta\phi = \phi_V - \phi_C$$

where

- ϕ_C is the pitch angle of the player's crosshair.
- ϕ_V is the pitch angle to the victim.

A practical example could be if the player's crosshair is pointing upwards, with an angle of [10°] and the victim is below the player, with an angle of [−10°] In this case, the CrosshairToVictimPitch would be:

$$\Delta\theta = -10^\circ - 10^\circ = -20^\circ$$

Firing

This is a binary variable that refers to the action of shooting a weapon. The firing action is one of the most important because it represent the moment when the player trigger the shooting mechanism, it has value 0 if there is no input and the value 1 if the weapon is firing.

1.1.3 Data Types

- The dataset is stored in a float32 numpy arrays.
- The Firing variable is binary (1 for firing, 0 for not firing).
- The other variables are continuous floats, representing various angles described above.

1.2 Data Exploration

First, let's start by analyzing our data and identifying the mean values and variances for the 5 key features around which our dataset revolves. We will calculate these values for both legit players and cheaters to see if any clear distincions emerge. Additionally, we will also keep track of the maximum and minimum values for each feature.

LEGIT	**Mean**	**Std.Dev**	**Max**	**Min**
AttackerDeltaYaw	-0.00481124	3.4206119	0.916651	-1.0942421
AttackerDeltaPitch	0.01660785	0.60591614	0.14140256	-0.12676828
CrosshairToVictimYaw	0.5582932	38.624165	40.746914	-37.241077
CrosshairToVictimPitch	-2.141741	7.289039	7.6832547	-12.650329
Firing	0.05210425	0.21751763	0.10121528	0.008159722

Table 1.1: Legit Player paramters

CHEATER	**Mean**	**Std.Dev**	**Max**	**Min**
AttackerDeltaYaw	-0.00515089	3.3006325	0.824648	-0.654368
AttackerDeltaPitch	0.01396316	0.7151011	0.1292548	-0.14456432
CrosshairToVictimYaw	0.8609003	39.32024	35.646404	-27.564005
CrosshairToVictimPitch	-2.537668	7.384222	15.190266	-46.289238
Firing	0.05117144	0.21584399	0.1015625	0.008680556

Table 1.2: Cheater Player paramters

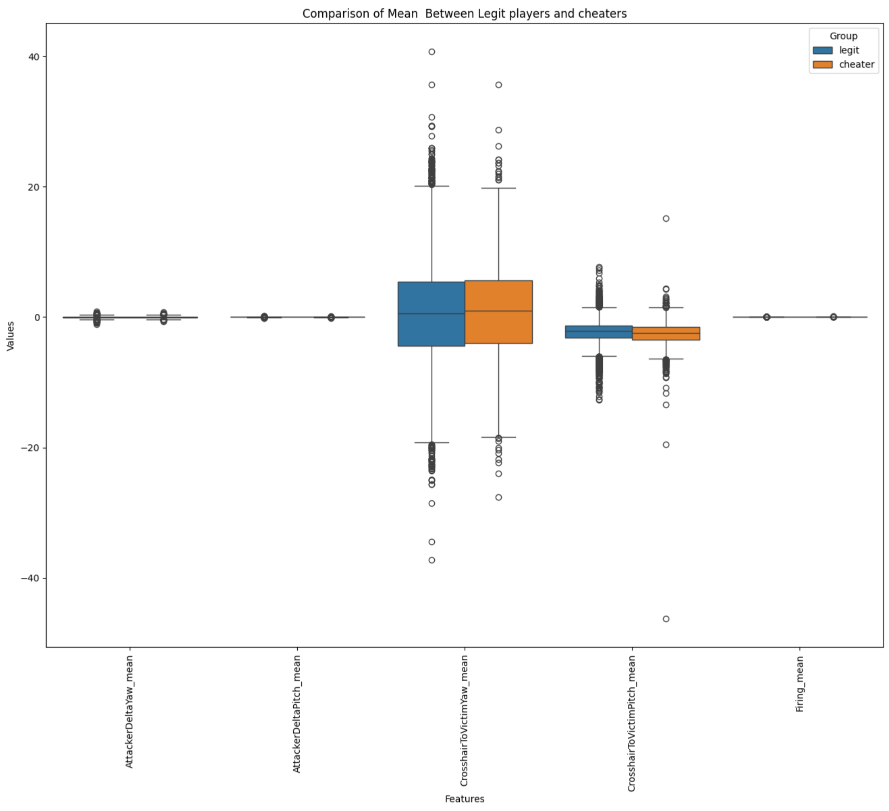
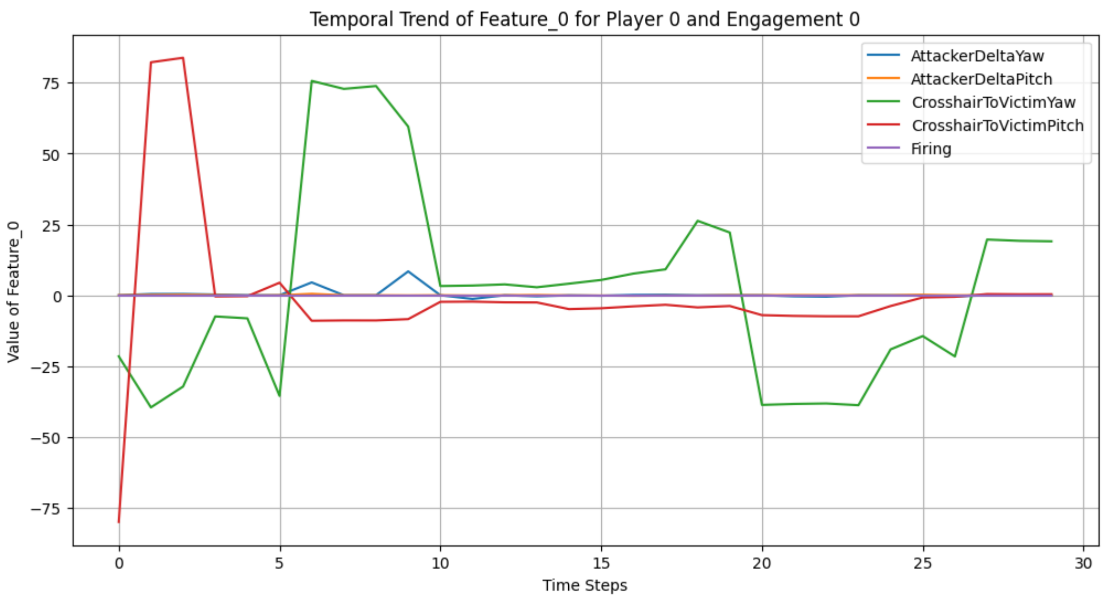


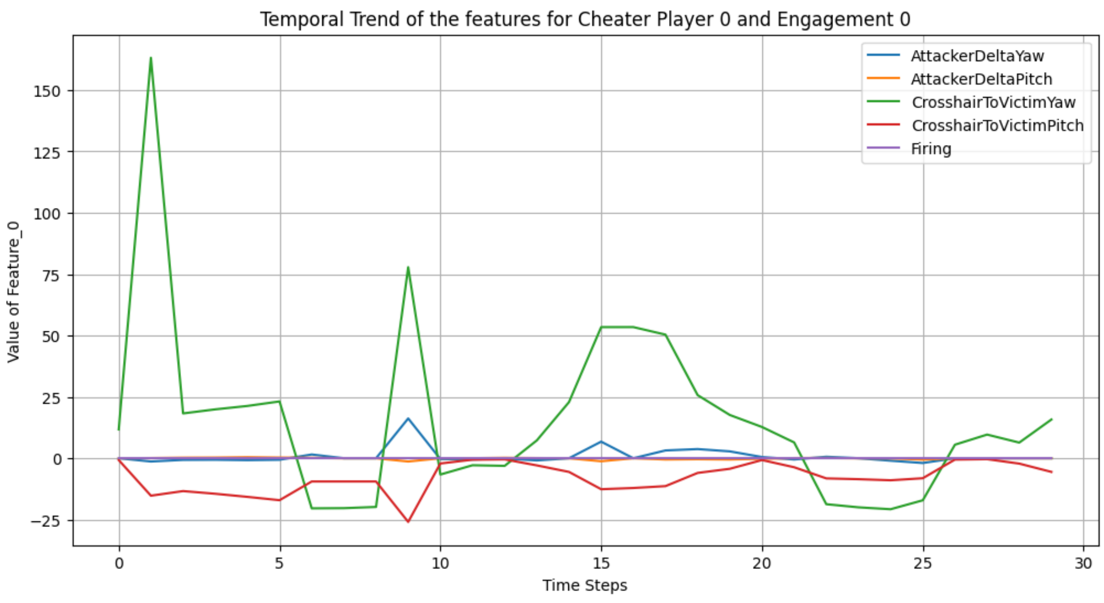
Figure 1.1: Boxplot describing the distribution of values of the input variables

Limiting ourselves to an empirical analysis derived only from the values calculated in the above table, we can notice that the data with the greatest variance and difference in means are CrosshairToVictimYaw and CrosshairToVictimPitch. It will be sensible to take into account this anomaly between the results when we implement our classification models. Intuitively, we could explain this discrepancy between the two distributions by the fact that the movement of a cheater's crosshair will be faster and with greater displacement compared to that of a legit player.

Considering the fact that we have time related data we want to visualize how the features change overtime for the players. The aim is to identify patterns, trends, or anomalies that may exist in the data. In order to do so we are going to select a player from the cheaters and one from the legits and for both of the players we plot the data for the features over the 192 time steps.



(a) Legit line plot for time series data



(b) Cheater line plot for time series data

Figure 1.2: Visual representation of the input features

1.3 Data Pre-Processing

In order to enhance interpretability and performance, after this brief analysis of the dataset we choose to preprocess the data. Initially the original data are standardize using Python's StandardScaler from the scikit-learn library [2]. This algorithm objective is to transform the feature in a way that they are distributed as a Gaussian with mean 0 and standard deviation of 1. In order to do so we take the dataset $\mathbf{X}$ as a matrix $n \times m$, we then let x_{ij} be the element of the i-th row and j-th column $\mathbf{X}$. The algorithm of StandardScaler work as follows:

- Compute the mean :

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

- Compute the standard deviation:

$$\sigma_j = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \mu_j)^2}$$

- Standardize the feature:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Subsequently, we craft an array of labels to properly associate the data of legit players and cheaters with their respective classifications. We then proceed to split the dataset into training set and test set, once again the scikit-learn library was used. It was decided to split the data into a 20 percent test set and an 80 percent training set while having stratify labels. This means that we want to ensure that each split maintains the same proportion of each class of the original dataset. This technique is important because it can improve the model performance. This is due to the fact that the model learns from a representative sample.

1.4 SMOTE method

SMOTE (Synthetic Minority Over-sampling Technique) [3] it's a technique used to face the problem of unbalanced classes in the dataset. The key idea is to create synthetic data from the original ones to give the algorithms a better dataset to learn from. This will increase the number of examples in the minority class, reducing the risk of overfitting and giving the possibility to learn better.

1.4.1 How SMOTE works

- Identifying nearest neighbors: The K-nearest neighbors are identified for each example in the minority class. The Euclidean distance is generally used for this task.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{n}(x_{im} - x_{jm})^2}$$

- Generate examples: For each example a new synthetic data is created by interpolating the values of the features of the original point and the neighbors.
- Creating the new dataset: The generated instances are added to the original dataset in order to balance the nubmer of examples in the classes.

1.4.2 Example

suppose we have a simple dataset with two simple feature (X1, X2) and two classes: 0 which is the minority class and 1 which is the majority class.

Given the fictional dataset with the following coordinates:

- Class 0: (1,1), (2,2)
- Class 1: (5,5), (7,7), (8,8), (9,9)

SMOTE steps

- Select a sample from the minority class: Let's suppose we choose x_i = (1,1)
- Find K-nearest neighbors: With k = 1, the nearest neighbor of x_i is $x_i^{(k)}$ = (2,2)
- Generate new samples: We create a new artificial sample $\tilde{x}$ on the line joining the two points with the formula

$$\tilde{x} = x_i + \lambda \cdot (x_i^{(k)} - x_i)$$

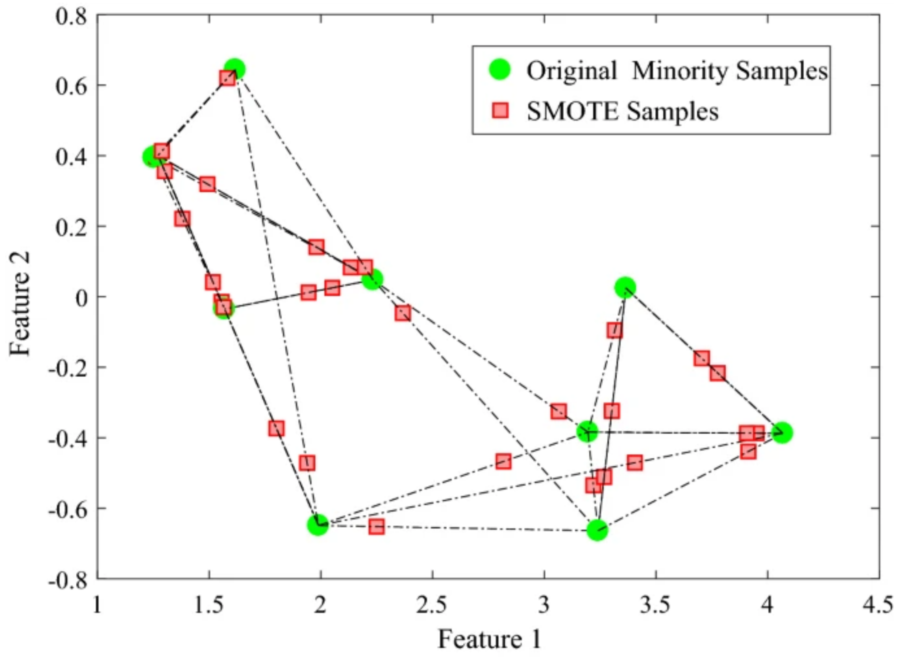


Figure 1.3: SMOTE generation process

Chapter 2

Main Methods for data elaboration

We are about to use some of the most common machine learning algorithms to determine which one, and with which parameters, achieves the best classification on our dataset. Each algorithm will undergo a brief analysis and parameter tuning. For each one, we have decided to use the following approaches: our data will be crafted by collapsing the features. For each player we are gonna compute the mean of the 5 basic features among all timesteps and actions of the player. This is done to reduce the dimensionality of the dataset. Then we will proceed with the implementation of machine learning models such as:

- Logistic regression
- SVM for binary classification
- Random forest
- Isolation forest
- Multi layer perceptron neural network

For each algorithm we will compute a confusion matrix and a classification report to have a better understanding of how well it performs.

2.1 Logistic Regression

2.1.1 Brief analysis

The logistic regression model [4] is used to predict the probability of a binary outcome based on the feature of the dataset. Differently from the linear regression, which predicts value in a continuos space, logistic regression uses a sigmoidal function in order to classify the label based on the probability.

The idea is to estimate how probable is that an observation X belongs to class 1 or 0

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p)}}$$

where:

- $\beta_0, \beta_1, \ldots, \beta_p$ are coefficients of the model.
- $x_1, x_2, \ldots, x_p$ are the predictor variables.
- The logistic function $\frac{1}{1+e^{-z}}$ transforms the linear combination $z = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$ into a probability between 0 and 1.

Once the probability is calculated we take a threshold value(typically 0.5). If the predicted probability is above 0.5 the point is classified as belonging to class 1, otherwise it is classified as belonging to class 0.

2.1.2 Loss function

The loss function for logistic regression which measure the error of the predicted probability is

$$J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

where:

- n Is the number of point in the dataset.
- $y^{(i)}$ Is the true label of the i-th point.
- $\hat{y}^{(i)}$ Is the predicted label for the i-th point.

2.2 SVM for binary classification

2.2.1 Brief analysis

SVM for binary classification [5] is a machine learning algorithm used for anomaly detection. The aim is to learn a decision hyperplane that separates the normal data points from the anomalies. It is important to find the hyperplane which maximize the margin to nearest data point following the formula:

$$\min_{w,\rho,\xi} \frac{1}{2}\|w\|^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho$$

subject to:

$$\langle w, \phi(x_i) \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \sum_{i=1}^{n} \xi_i \leq \nu n$$

- w: Weight vector defining the hyperplane.
- ρ: Offset term.
- ξ_i: Slack variables measuring violation of constraints.
- ν: Parameter controlling fraction of outliers.
- $\phi(x_i)$: Feature map into higher-dimensional space.

2.2.2 Kernel

A function that is used to transform data into another feature space to made it easier to separate. The way it works it calculates the dot product in a higher dimensional space without explicitly calculating the coordinates of the new space, this process is know as the kernel trick.
Different kernel function can be used, a kernel function $K(x, y)$ takes two data points and returns the dot product $\phi(x) \cdot \phi(y)$. The kernel function can be of different type on of the most common one is the Gaussian Kernel:

$$K(x, y) = \exp\left(-\gamma \|x - y\|^2\right)$$

where γ is a parameter that defines how much a single training example influence the result.

2.3 Random forest

2.3.1 Brief analysis

Random forest [6] is a machine learning algorithm that combines the output of multiple decision tree to reach a single result. It is a ensemble method specifically designed for decision tree classifiers. Random forest introduce the concept of bagging and feature randomness to build a strong classifier. The algorithm is divided in steps:

- Ensemble of decision trees: builds multiple decision tree, each one is trained on a random subset of data and feature. This prevents overfitting and improve robustness

- Decision tree: Each tree is trained independently to predict the target value. To do so, data are recursively split based on criterion such as Gini impurity

  $$G(t) = \sum_{i=1}^{C} p_i(1 - p_i)$$

  where p_i is the proportion of sample belonging to class in i in node t.
  Gini impurity range between 0, when all the elements belong to a single class, and 0.5, when the elements of the node are evenly distributed. The idea is that higher impurity indicates a higher chance of misclassifying a new element.

- Final classification: Lastly, the final prediction of the random forest is determined by the majority of classification among the trees.

2.4 Isolation forest

2.4.1 Brief analysis

Isolation forest [7] is an unsupervised learning algorithm used for anomaly detection. The key concept is that anomalies are less frequent and different from normal values, therefore easier to isolate by randomly partitioning the data. The process of isolation tree is based on three steps:

- Ensemble of isolation trees: builds multiple isolation tree, each one is trained on the original dataset and is splitted at random among the feature.

- Iteration of splitting: Data are splitted until every data point is placed in a leaf, or until the maximum number of split is reached.

- Anomaly score: each node of the tree gain an anomaly score based on the path length from the root to the node which is then averaged across all the trees

  $$E(h(x)) = \frac{1}{t}\sum_{i=1}^{t} h_i(x)$$

  where $h_i(x)$ is the path length of x in the i-th isolation tree. If the anomaly score is closer to 1 the data is likely to be an anomaly, if it is less then 0.5 it is likely to be normal

2.5 Multilayer perceptron neural network

2.5.1 Brief analysis

A neural network [8] is a computational model designed to mimic how the human brain works. The power lies in their ability to learn non linear and complex relationship making them extremely versatile.

2.5.2 Neural network structure

- Neurons: Also called nodes, this are the basic units of the network. Each neuron is based on the perceptron model which recives input, process it, and produces an output. In order to do so the neuron must be connected to other neurons and have an activation function.
- Layers: Neural networks consist of multiple layers, various structure are possible in the way this layers are connected between each other. A popular one is the fully connected which have each neuron of a layer connected to all the neurons of the previous and next layer. Layers usually can be divided as:
  - Input layer: Receives the input from the raw data and passed it to next layer.
  - Hidden layers: Intermediate layers, they perform the computations to extract patterns from feature correlation.
  - output layer: Final layer that produces the final prediction based on the hidden layers output.
- Weights: Each connection between two neurons have an associated weight. They are updated throughout the training and they determine the importance of the neurons
- Activation function: Used to introduce non linearity in the network, allowing it to learn complex pattern in the data.

Sigmoid	**ReLU**	**Tanh**
$\frac{1}{1+e^{-z}}$	$\max(0, z)$	$\frac{2}{1+e^{-2z}} - 1$

Table 2.1: Common Activation Functions

2.5.3 Neural network training

Training a neural network is an iterative process divided in epochs. During each epoch is fed to the neural network which compute the predictions, evaluate the error and update the weights. In order to minimize the Loss function usually backpropagation and gradient descent are performed.

- Loss function: Used to quantify the difference between the real value and the predicted value. Depending on the problem different loss function can be used, the most common are Mean Square Error for regression and Cross-Entropy for classification Mean Square Error (MSE):

  $$\mathrm{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  Cross Entropy Loss (CE):

  $$\mathrm{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- Backpropagation: Compute the gradient of the loss and propagate the error from output to input in order to update the weights and improve the performance.

- Gradient descent: Updates the weights to minimize the loss following the rule

  $$\theta \leftarrow \theta - \eta \nabla J(\theta)$$

  where η is the learning rate and $\nabla J(\theta)$ denotes the gradient of J with respect to θ which are the weights.

  Different techniques can be used to update the weights with variation of gradient descent. Each one updates the parameters of the models in distinct, impacting efficiency and performance of the model.

  - Full-batch Gradient Descent: computes the gradient of the loss function updating the model parameters.
    It is useful because it converge to minimum of the function which are more stable, this is due to the fact that the entire train set is used. On the other hand it can be slow and computationally quite expensive.

- Stochastic Gradient Descent: it randomly select a feature and compute the gradient based on this feature, on the next step the process repeat itself.
  It is useful because it requires less memory and is faster, thanks to the fact that it needs to calculate only one gradient at a time. On the other hand the noise of the data can affect the gradient resulting in a erratic convergence.
- Mini-Batch Gradient Descent: Is a compromise between full batch and stochastic gradient descent. It splits the training in K batches and computes the gradient and updates the parameters using one batch at a time.
  It is useful because gives a good balance between the speed and stability of the previous algorithms. More stable convergence are easily obtain. On the other hand the choice of the size of the batch it still requires good amount of memories.

2.5.4 MLP architecture

For our problem an MLP neural network was created, the structure of this network has 3 hidden layer with respectively 128, 256 and 128 neurons. We use a fully connected architecture with the following paramters:

- Loss criterion: Binary Cross Entropy
- Optimizer: Adam
- Learning rate: 0.001
- Number of epochs: 40
- Activation function: ReLU and Sigmoid

Chapter 3

Results

Results of the algorithm are shown below in the form of tables and confusion matrix

3.1 Logistic Regression

labels	**precision**	**recall**	**F1 score**	**support**
legit	0.85	0.55	0.67	2000
cheater	0.19	0.52	0.28	400

Table 3.1: results for logistic regression

accuracy: 0.55

The logistic regression model predict 1099 true positive, 901 false negative, 191 false positive and 209 true negative. Which means that:

- Precision: 0.852 and 0.19
  The algorithm it's pretty accurate in classyfing the legit, which means that among all the predicted legit 85 percent are indeed legit. On the other hand it is difficult to classify the cheaters, indeed only 19 percent of the predicted cheaters are cheating.

- Recall: 0.55 and 0.52
  Recall is realtively low, which means that it is difficult both for the legit and the cheaters to be assigned in the right class.

- F1-score: 0.67 and 0.28
  F1-score which is a mean between precision and recall give a better and general understanding of the classification. F1-score show us that logistic regression is not the best tool to analyze this problem.

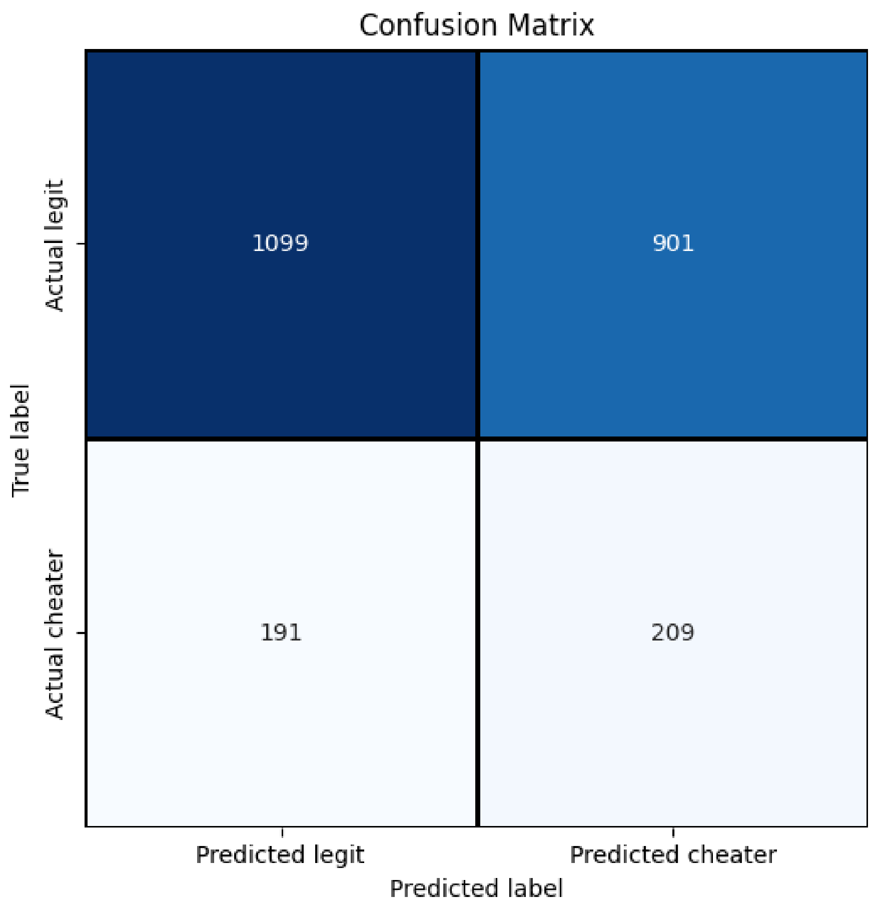


Figure 3.1: Confusion matrix of Logistic regression

3.2 SVM for binary classification

labels	**precision**	**recall**	**F1 score**	**support**
legit	0.83	0.81	0.82	2000
cheater	0.17	0.19	0.18	400

Table 3.2: results for SVM for binary classification

accuracy: 0.71

The SVM for binary classification model predict 1629 true positive, 371 false negative, 324 false positive and 76 true negative. Which means that:

- Precision: 0.83 and 0.17
  The algorithm it's pretty accurate in classyfing the legit, which means that

among all the predicted legit 83 percent are indeed legit. On the other hand it is difficult to classify the cheaters, indeed only 17 percent of the predicted cheaters are cheating.

- Recall: 0.81 and 0.19
  Recall is in this situation unbalanced, which means that among all the legit players 81 percent of them are correctly classified, whereas only 19 percent of predicted cheaters are actually cheater.

- F1-score: 0.82 and 0.18
  F1-score show us that SVM for binary classification is actually a pretty good algorithm to distinguish between legit players and cheaters. The good results of the model were expected, SVM for binary classification is indeed a algorithm to distinguish between two classes thanks to a support vector machine.

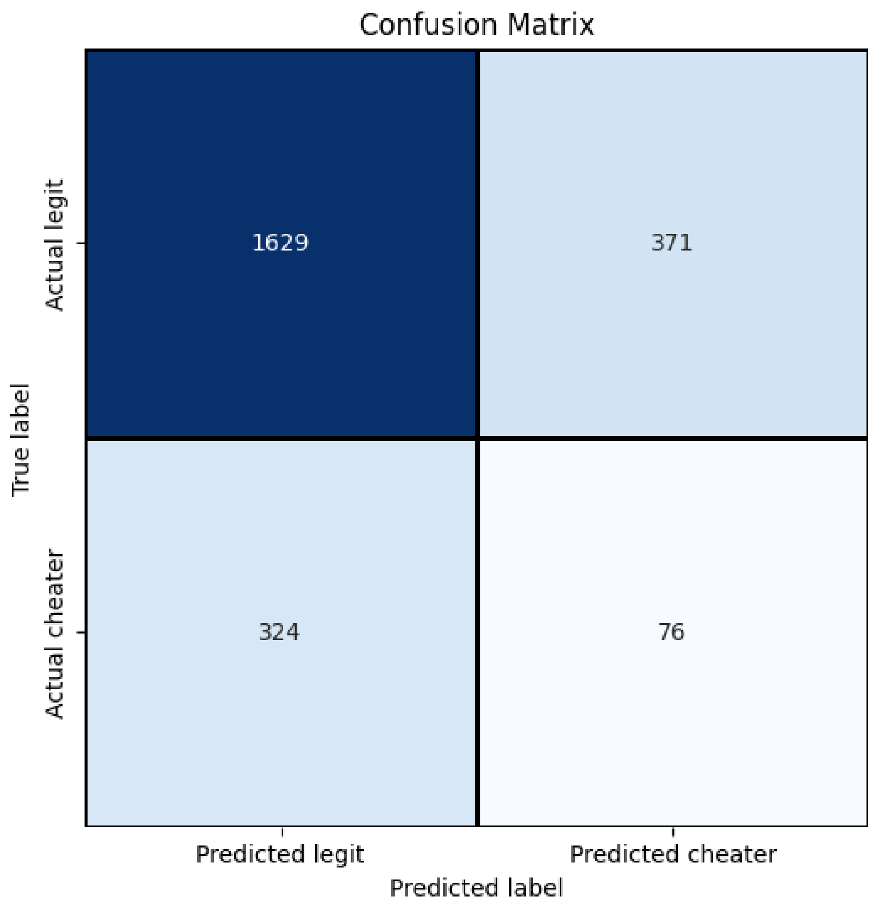


Figure 3.2: Confusion matrix of SVM for binary classification

3.3 Random forest

labels	precision	recall	F1 score	support
legit	0.83	0.99	0.91	2000
cheater	0.27	0.01	0.03	400

Table 3.3: results for Random forest

accuracy: 0.83

The random forest model predict 1984 true positive, 16 false negative, 394 false positive and 6 true negative. Which means that:

- Precision: 0.83 and 0.27
  The algorithm it's pretty accurate in classyfing the legit, which means that among all the predicted legit 83 percent are indeed legit. On the other hand it is difficult to classify the cheaters, only 27 percent of the predicted cheaters are cheating.

- Recall: 0.99 and 0.01
  Recall is really high for legit players, as high as 99 percent, indicating that the model is almost perfect for classifying legit players. On the other hand it performs very poorly in the cheaters classification process, being able to correctly predict only 1 cheater out of 100.

- F1-score: 0.91 and 0.03
  F1-score show us that random forest have a really bad overall performance. Although recall shows a high value for the legit players, if we took a closer look on how the instance are classified we can see that it basically behave like a baseline model, predicting the majority class for all the data.

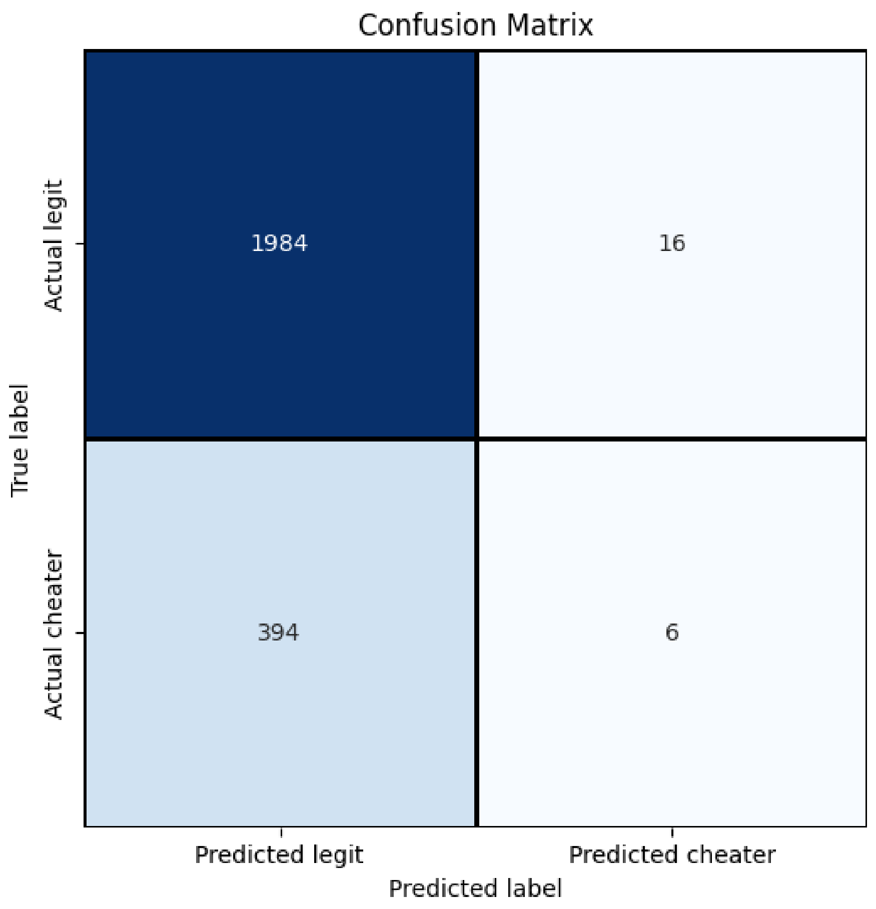


Figure 3.3: Confusion matrix of Random forest

3.4 Isolation forest

labels	**precision**	**recall**	**F1 score**	**support**
legit	0.84	0.85	0.84	2000
cheater	0.19	0.17	0.18	400

Table 3.4: results for Isolation forest

accuracy: 0.83

The isolation forest model predict 1700 true positive, 300 false negative, 330 false positive and 70 true negative. Which means that:

- Precision: 0.84 and 0.19
  The algorithm it's pretty accurate in classyfing the legit, which means that

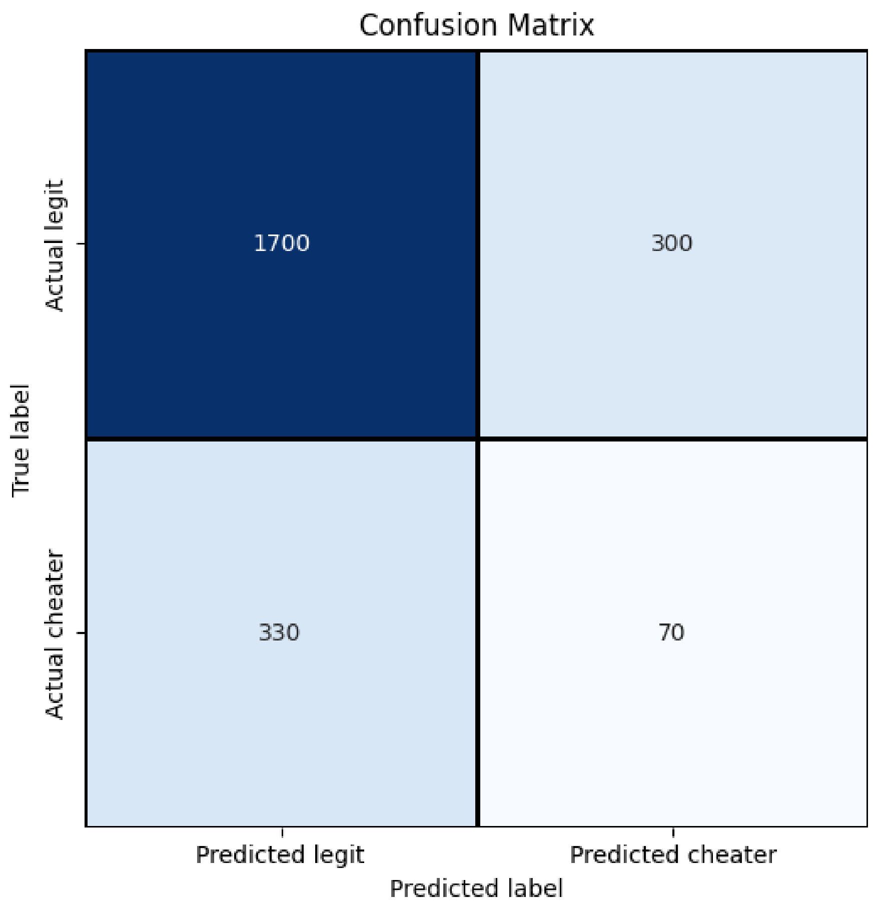


Figure 3.4: Confusion matrix of Isolation forest

among all the predicted legit 84 percent are indeed legit. On the other hand it is difficult to classify the cheaters, indeed only 19 percent of the predicted cheaters are cheating.

- Recall: 0.85 and 0.17
  Recall is in this situation unbalanced, which means that among all the legit players 85 percent of them are correctly classified, whereas only 17 percent of predicted cheaters are actually cheater.

- F1-score: 0.84 and 0.18
  F1-score show us that isolation forest is quite a good algorithm to discern between legit player and cheaters. It is able to keep a good ratio between the correctly predicted positive and correctly predicted negative. This result were expected because isolation forest is a algorithm specifically built for

anomaly detection.

3.5 Multilayer perceptron neural network

Epoch	**Loss**
5	0.03997
10	0.02521
15	0.01419
20	0.01143
25	0.00714
30	0.00134
35	0.00087
40	0.00086

Table 3.5: Loss of the neural network trough the epochs

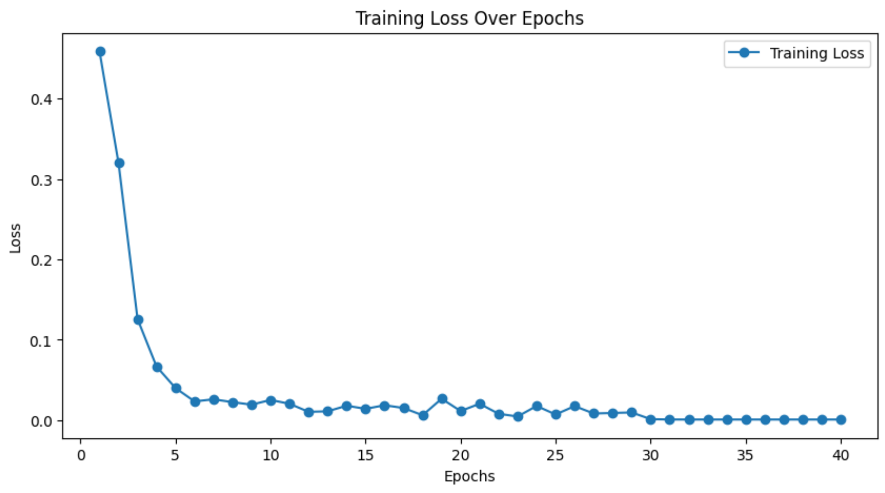


Figure 3.5: Training loss plotted with respect to the epoches

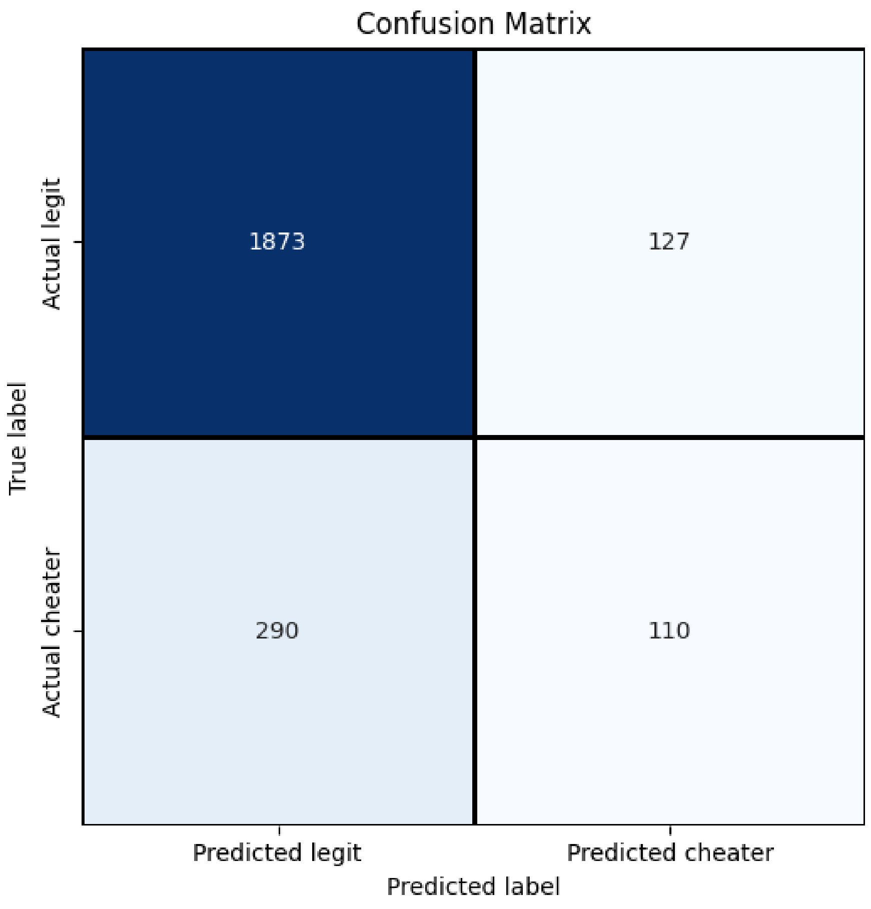


Figure 3.6: Confusion matrix of MLP

labels	precision	recall	F1 score	support
legit	0.88	0.92	0.90	2000
cheater	0.46	0.35	0.40	400

Table 3.6: results for Neural Network

Neural network model predict 1873 true positive, 127 false negative, 290 false positive and 110 true negative. Which means that:

- Precision: 0.88 and 0.46
  The algorithm it's highly accurate in classyfing the legit, which means that among all the predicted legit, 88 percent are indeed legit. Moreover, cheater classification is quite good, 46 percent of the predicted cheaters are cheating. Which means that when we are predicting a cheater we will be right one time out of two.

- Recall: 0.92 and 0.35
  Recall is especially good for legit players and quite good for cheaters. The model is able to correctly predict a legit player 92 percent of the time, while a cheater is detected only with a probability of 35 percent.

- F1-score: 0.90 and 0.40
  F1-score show us that neural network is the best model used, thanks to the complex structure of the network, it is able to grasp more complex data correlation. This makes the neural network able to correctly classify most of the legit players, while maintaining a good accuracy on the cheaters.

- Training loss: It rapidly converge, fitting the train data, the curve then gradually reduce the slope, suggesting that the model is refining the predictions and giving small tweak to the neuron's weight. The process continue until a plateau is reached, meaning that the algorithm converges to the optimal solution for the neural network.

3.6 Comparison between results

After gathering all the results of the different algorithm, we can easily state that neural network is the best performing model of all. This can be expected, because it is the most complex model that can grasp the connection between the feature and the temporal correlation in the data. More simple model have various type of performance, algorithms like isolation forest and SVM for binary classification, which are designed for anomaly detection, are having a better performance. On the contrary, random forest model have a performance close to a baseline model that almost always predict the majority class. At last, logistic regression model is one of the worst performing, behaving like a coin flip for prediction.

Conclusion

3.6.1 Summary

In this thesis we explored the application of different machine learning algorithms to our dataset in order to separate cheaters from legitimate players. We started with a brief pre process of the data to make it easier to compute our predictions. We normalize our values and use specific technique, such as SMOTE, to handle unbalanced difference between the number of cheaters and legit players. We then proceed to create and evaluate different models, such as Support Vector Models and Random forest, as well as more advance technique like Neural Network, that try to enhance correlation in the temporal evolution of the data.

3.6.2 Future works

The analyses conducted in this thesis show that even if there are no groundbreaking result, a machine learning approach to contain the problem of cheaters could be a step in the right direction.
Only classical machine learning technique were explored and many others could be used to get results with grater significance. Crucial aspect of future work could be to improve the explainability of the model, ensuring transparency in the detection process. Furthermore, it would be important to investigate if the results obtained can be generalized to create versatile anti-cheating system across different games. Another important point is that we didn't work with raw data but summary statistic. By using models design with temporal figures we can potentially improve detection abilities. This however will require more computational power to train them.

3.6.3 Conclusion

This thesis demonstrate the effectiveness of using a machine learning algorithm to identify cheating behavior in online games. By utilizing various models with targeted hyperparameters, it was possible to improve the accuracy and reliability of the methods. This contribution can serve as a starting point for future development in the detection of cheaters. Through the use of more advanced techniques and the expansion of the available data, machine learning models can help make online experiences more fair and enjoyable for everyone.

Bibliography

[1] Savelii Pashkov. “Video Game Industry Market Analysis: Approaches that resulted in industry success and high demand”. In: (2021).

[2] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Dina Elreedy, Amir F Atiya, and Firuz Kamalov. “A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning”. In: *Machine Learning* (2023), pp. 1–21.

[4] Michael P LaValley. “Logistic regression”. In: *Circulation* 117.18 (2008), pp. 2395–2399.

[5] Hyun Joon Shin, Dong-Hwan Eom, and Sung-Shick Kim. “One-class support vector machines—an application in machine fault detection and classification”. In: *Computers & Industrial Engineering* 48.2 (2005), pp. 395–408.

[6] Gérard Biau and Erwan Scornet. “A random forest guided tour”. In: *Test* 25 (2016), pp. 197–227.

[7] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[8] Yu-chen Wu and Jun-wen Feng. “Development and application of artificial neural network”. In: *Wireless Personal Communications* 102 (2018), pp. 1645–1656.