Master thesis : Game Intelligent Analyst - Anomaly Detection in Casino Games using Machine Learning Algorithms

Auteur : Merchie, Florian
Promoteur(s) : Wehenkel, Louis; 3950
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2017-2018
URI/URL : http://hdl.handle.net/2268.2/4651

UNIVERSITÉ DE LIÈGE

MASTER THESIS

Game Intelligent Analyst - Anomaly Detection in Casino Games using Machine Learning Algorithms

Author:
Florian MERCHIE

Supervisor:
Prof. Louis WEHENKEL

A thesis submitted in fulfillment of the requirements for the degree of Master in Civil Engineering in Computer Sciences

Faculty of Applied Sciences
GAMING1

June 8, 2018

UNIVERSITÉ DE LIÈGE

Abstract

Faculty of Applied Sciences

Master in Civil Engineering in Computer Sciences

Game Intelligent Analyst - Anomaly Detection in Casino Games using Machine Learning Algorithms

by Florian MERCHIE

Anomaly detection is a very large and complex field. In recent years, several techniques based on data science were designed in order to improve the efficiency of methods developed for this purpose. In particular, density-based anomaly detection allows to estimate how much probability densities differ from each other by solving a supervised learning problem using data drawn from these densities as a dataset. Besides, the casino games company GAMING1 aims at automating the detection of behavior modifications in its games when an update is performed. This master thesis introduces a method based on density-based anomaly detection to reach GAMING1's objective. We explore two main approaches to solve this problem. The first one relies on standard supervised learning algorithms. For this approach, two algorithms were considered: Extremely randomized trees and linear support vector machine. The second approach is based on deep learning and exploits recurrent neural networks architectures. Using experimental cases of anomalies in casino games, it's been shown that extremely randomized trees outperforms both other methods as far as performances and visual interpretation are concerned, but also considering computational resources. In addition to accuracy, we rely on several other metrics related to the comparison of probability densities in order to assess formal performances of the presented algorithm. We show that the tree-based method allows us to distinguish irregular data with an accuracy of at least 0.7 for standard anomalies, while providing relevant visual support thanks to probability densities representation. Moreover, it's also possible to identify more hardly detectable anomalies by using classifier calibration in order to enhance the visual support of the probability densities, despite an associated low accuracy.

UNIVERSITÉ DE LIÈGE

Acknowledgements

Faculty of Applied Sciences

Master in Civil Engineering in Computer Sciences

Game Intelligent Analyst - Anomaly Detection in Casino Games using Machine Learning Algorithms

by Florian MERCHIE

I would like to express my deepest thanks to the people who contributed to this master thesis:

- My thesis director, M. Wehenkel, for its supervision, advice and encouragements.
- GAMING1's CTO, Christophe Boniver, for offering me the opportunity to realize this project.
- M. Louppe and M. Geurts for guiding me in the algorithmic design of the solution presented in this master thesis.
- Aurélie Boniver for supervising my day-to-day work at GAMING1 and providing me regular feedback about the progress of my work.
- GAMING1 company, in particular the Game Engine team, for welcoming me for several months and providing the material without which I couldn't have realized this master thesis.
- My family and friends for their support and patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gambling games find their origin in a time almost as old as writing. While historians acknowledge that the first written language seems to date from the half of the fourth millennium BC, the oldest traces of gambling games have been associated to the end of that same millennium, around 3000 BC. Then, as most of mankind's inventions, these have evolved through times and places. For example, the first card games were used in ancient Chinese Empire, while Romans were the first to bet money on entertainment events, like gladiators fights. Nowadays, these games have taken an incredibly high number of different shapes. From classical casino games, such as roulette or slot machines, to sport betting passing by lottery and scratch tickets, there is always a way to bet money on something. In particular, with the rise of the Internet, most of these games have been dematerialized and are now available on almost every popular enough connected device, such as computers and smartphones. The demand in such games have grown so much in recent years that new companies were born to provide safe and reliable services to the classical gambling games companies. GAMING1 is one of these companies.

Among the services provided by such companies, there is of course the ability to develop and maintain reliable online games. In the past, the reliability of physical machines were only depending on mechanical properties. It's of course not the case anymore considering online casino games, where these can be subjected to bugs. Up to now, before the releasing of a game or when updating one, GAMING1 principally uses one main technique in order to detect bugs. This consists in using simulators incorporating the different game engines. The results of the performed simulations are then collected in the form of different statistical features. GAMING1's game engine team analyzes these results and then make conclusions about the behaviour of the game. However, such a method often takes time to be done and is not always very reliable. Indeed, it sometimes happens that games are released and that undesired things occur once these are played by real people. The consequences of such events can sometimes be very serious for the company, as the early experiences of players on online games often influences a lot the fact that they will continue to play it or not. The concerns in terms of profitability are therefore really important.

The purpose of this master thesis is to develop a method allowing to identify games of which the behaviour has changed when an update has been performed. Of course, GAMING1 wants this method to independent of the game which is analyzed. In order to achieve this goal, data obtained from simulations of the game before and after the update have been compared. This kind of problem is denoted as a density-based anomaly detection problem. The solution to this problem has been designed relying mainly on two principles. The first one was to consider this statement as a likelihood-ratio problem. Then, this likelihood-ratio problem has been solved a binary classification problem using machine learning algorithms. Two main approaches have been compared in this master thesis. On the one hand, features extraction techniques have been applied to the data generated by the simulations, followed by the application of standard machine learning algorithms (Linear Support Vector Machine or Extremely Randomized Trees). On the other hand, deep learning was used by directly passing the simulations results to recurrent neural networks.

This document is divided into six chapters. Following this introduction Chapter, some explanation about the theoretical background required to understand the work that has been done, is provided in Chapter 2. Then, Chapter 3 dives more deeply into the methodology of this master thesis, describing the objectives and motivations as well as the particular methods that have been designed and applied to achieve these objectives. Then, the results of the different experiments that have been performed to validate our methods are detailed in Chapter 4, before briefly concluding in the final Chapter.

Chapter 2

Theoretical Considerations

For years, anomaly detection has been used to identify unexpected or undesired behavior in many fields. In the early days of this discipline, basic statistical methods were the most commonly applied techniques in order to achieve its goals. However, with the advent of the big data era, these simple methods were quickly put in a difficult position. One of the main reasons of this was the increasing of the dimensionality of the data which is often considered nowadays. Machine learning is able to overcome this limitation by relying on algorithms having the capacity to scale easily according to this dimensionality. In particular, supervised learning is able to learn the impact of very high number of factors on a given target characteristic.

This chapter will first address supervised learning in its global aspects. Then, a few examples of algorithms which are widely used in supervised learning will be explained. Afterwards, a focus will be made on particular techniques used for anomaly detection.

2.1 Supervised learning

In the scope of machine learning, the basic entities are called objects, or samples. To each object is associated an ensemble of values, denoted as the features or attributes. For a given machine learning problem, data is typically available as a set of such objects, called the learning set. In the case of supervised learning, one of the feature is defined as the target or output variable. The goal of an algorithm using this approach is to build a model which is able to predict the output value of a newly observed object. The algorithm selects the best model as possible out of a predefined space of functions that can map a particular set of classical features to the output variable. To be able to do so, the algorithm requires two additional tools: a loss function, measuring the accuracy of output prediction, and a strategy to optimize the selection of the model.

To formalize this clearly, an object is denoted as $o = (\mathbf{x}, y), \mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$, where $\mathcal{X}$ (resp. $\mathcal{Y}$) is the input (resp. output) space. The learning set of size N, consisting of a group of objects, can therefore be defined as $LS = \{o_i | i = 1, \ldots, N\}$. Considering only features, the learning set can be seen as a 2-dimensional array where a row (resp. a column) represents an object (resp. a feature). From this learning set, the goal of supervised learning is to find a a function $f : \mathcal{X} \to \mathcal{Y}$ that would approximate at best the relation between the input $\mathbf{x}$ and the output y. The set of all possible functions f that can be proposed by a given algorithm, is called the hypothesis space $\mathcal{H}$. In order to determine which function among this hypothesis space is the best one, some metric is required. Therefore, we also define a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ which compares the value $f(\mathbf{x})$ estimated by the function f and the true output value y. The objective of f is thus to minimize the expectation of this loss function l. In practice, we often dispose of another dataset called the testing set, TS, to evaluate the accuracy of f for some new objects. If TS has a size M which is sufficiently large, the expectation of the loss function can be approximated as the following:

$$E_{\mathbf{x},y}\{l(f(\mathbf{x}),y)\} \approx \frac{1}{M}\sum_{\mathbf{x},y\in TS} l(f(\mathbf{x}),y)$$

This can be noticed that, if $LS = TS$, this value is called the re-substitution error. Otherwise, it's called the generalization error. As the goal of supervised learning is formally to find a function f such that $f = \arg\min_{f\in\mathcal{H}} E_{\mathbf{x},y}\{l(f(\mathbf{x}),y)\}$, a supervised learning problem can also be seen as a optimization problem. Fundamentally, these are divided into two categories: classification and regression. Classification regroups problems of which the output is symbolic. It's also called a label or class. An example of loss function for this type of problems is the classification error defined as:

$$l(y,\hat{y}) = 1_{\neq}(y,\hat{y}) = \begin{cases} 1, & \text{if } y \neq \hat{y}. \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

By contrast, when the output of an algorithm is a purely numerical value, with no label or class interpretation, this algorithm is said to perform a regression. For these problems, the most widely used loss function is square error loss defined as:

$$l(y,\hat{y}) = (y-\hat{y})^2$$

An important consideration in supervised learning is the amount of samples which is available for the problem. In situations in which we dispose of a sufficient quantity of data, this is separated into the two datasets mentioned above, LS and TS. Usually, the data proportions are 70% for LS and 30% for TS. Then, he fitting of function f is performed using LS and the performances are assessed by comparing the predictions $f(\mathbf{x})$ and the output y for the samples of TS. However, in some applications, it's often difficult to gather a sufficient quantity of data. For example, the required time to produce it could be too large, or this could be subjected to some access restrictions. Therefore, some methods had to be developed to deal with such situations. A widely used technique is called k-fold cross-validation. As illustrated in Figure 2.1, the concept is to partition the initial dataset into k equal

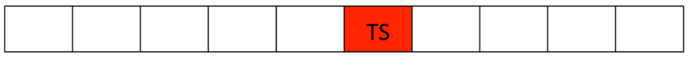


FIGURE 2.1: Concept of k-fold cross validation ($k = 10$) [1]

subsets. Then, for each subset, a supervised learning procedure is performed where the learning set is the set composed of all other $k-1$ subsets and the testing set is the currently considered subset. At the end, the mean error over the performed predictions is reported to evaluate the performances of f. In this way, if the initial dataset has a size N, we can evaluate the average error over N predictions with k learning procedures. Nevertheless, it should be also be considered that successfully using the same data for both learning and testing, even in different procedures, can sometimes lead to some issues. When $k \sim N$, as the size of TS will be close to 1,

[1][1, Geurts 2016]

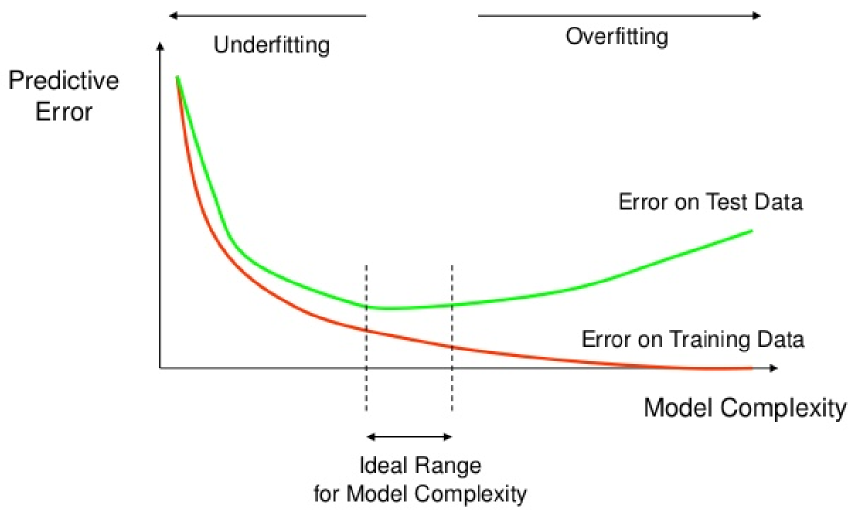


FIGURE 2.2: Controlling complexity in supervised learning [2]

the variance of the error increases a lot, and the problem becomes highly dataset dependent. Moreover, this requires to perform almost N learning procedures, which can be very time-consuming. On the other side, when k is smaller, the procedure becomes faster and the variance is reduced but bias appears due to the larger number of samples that are removed of the initial dataset for each fitting.

Most of the time, the size the of the hypothesis space $\mathcal{H}$ can be very large. Among all the factors that characterize an element $f \in \mathcal{H}$, one of the most relevant is the complexity. Typically, increasing the complexity reduces the re-substitution error. Indeed, the more complex a model will be, the better it will fit the specific characteristics of a given dataset. However, in practice, a model that would perfectly predict the outputs of its learning set is not the desired behaviour. Instead, we want a model which would generalize as best as possible for new samples. That's why using a testing set is relevant at this point. The graph displayed in Figure 2.2 helps to understand this idea. This details the typical behaviour of the re-substitution and generalization errors when the complexity of the model increases. It's observed that, contrary to the re-substitution error, there exists a minimum for generalization error. The complexity range where this error is the smallest defines the optimal complexity to choose for a given problem. Depending on the type of algorithms which is used, the complexity can depend on a various set of factors.

In further sections, some examples of such algorithms will be given and explained, as well as the parameters which influence their complexity.

2.1.1 Linear support vector machine

Support vector machines were originally proposed by [2, Vapnik 1963] in order to classify linearly separable tasks. Let's assume again the learning LS of previous section but let the output space $\mathcal{Y}$ be restricted to $\{-1, 1\}$, i.e. the problem is reduced to a binary classification. LS is said to be linearly separable if it's possible to drawn

[2]General question regarding Over-fitting vs Complexity of Models

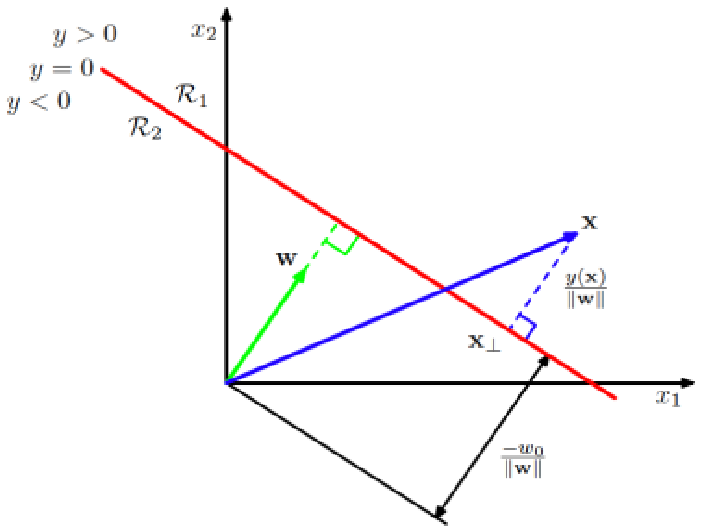


FIGURE 2.3: Example of LSVM [3]

an hyperplane in the input space $\mathcal{X}$ that classifies the different samples in two distinct regions according to their output value y. For a new observed sample $\mathbf{x}_{\text{new}}$, the classifier would thus make the following prediction:

$$f(\mathbf{x}_{\text{new}}) = \text{sgn}(\mathbf{w}^T\mathbf{x}_{\text{new}} + w_0)$$

where $\mathbf{w}$ is the normal vector to the chosen hyperplane, w_0 is linked to the offset $\frac{w_0}{||\mathbf{w}||}$ of the hyperplane from the origin along the normal vector $\mathbf{w}$ and sgn denotes the sign function. In Figure 2.3, an example of SVM hyperplane chosen for an 2 dimensional input space is shown. The regions corresponding to each class label are also annotated with the corresponding prediction of the classifier.

What differentiates linear SVM from classical linear classification problems is its objective function. Indeed, when using SVM, the intrinsic is the maximizes the distance from the hyperplane to the closest point in LS, which can written as:

$$\max_{\mathbf{w},w_0}\{||\mathbf{x} - \mathbf{x_i}|| : \mathbf{w}^T\mathbf{x} + w_0 = 0, i = 1, .., N\}$$

This allows the algorithm to be more robust to changes in attributes values than classical linear classification problems. Figure 2.4 illustrates the choice of the hyperplane according to this objective function. It can also be shown that the solution corresponding to the optimal hyperplane is not unique. Indeed, using some geometry, the optimization problem can be written as:

$$\arg\max_{\mathbf{w},w_0}\{\frac{1}{||\mathbf{w}||}\min_n[y_n \cdot (\mathbf{w}^T\mathbf{x}_n + w_0)]\}$$

. By multiplying $\mathbf{w}$ and w_0 by a constant $c > 0$, the hyperplane remains unchanged. Although, unicity can be obtained by imposing $|\mathbf{w}^T\mathbf{x}_n + w_0| > 1$ for the closest $\mathbf{x}$ to the surface. The problem becomes then equivalent to minimizing $||\mathbf{w}||$ under the

[3]Introduction to Machine learnin: SVM
[4]Introduction to Machine learnin: SVM

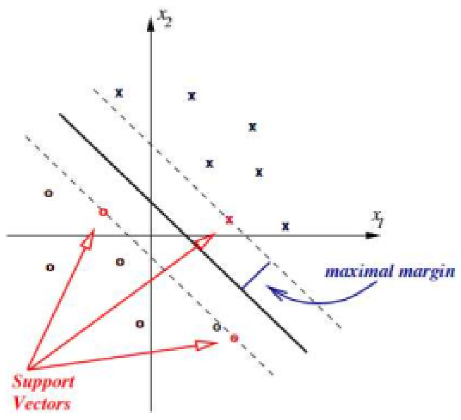


FIGURE 2.4: Objective illustation [4]

constrains $y_i \mathbf{w}^T \mathbf{x}_i + w_0 \geq 1, i = 1, ..., N$.

Up to now, two majors drawbacks can be enhanced considering linear support vector machines. First, they are restricted to linearly separable problems. Secondly, it's mostly limited to binary classification. Several techniques however exist to overcome these issues. For non linearly separable data, [3, Cortes et al. 1995] introduced the concept of soft margin. Along with, the hinge loss function was defined as:

$$\max\{0, 1 - y_i(\mathbf{w}^T \mathbf{x}_n + w_0)\}$$

This allows some samples not to respect the margin by penalizing them if so, proportionally to their distance to the margin. Considering the second limitation, several solutions have been discussed in the literature, such as in [4, Duan et al. 2005]. A common proposed technique is the one vs all classification. In this method, if C classes must be classified, then C classifiers are also trained. For each of them, one class is chosen and gets its label unchanged, while all the other ones get a common label. Then, its classifier is trained as a classic binary classifier.

2.1.2 Extremely randomized trees

Extra trees were introduced in [5, Geurts et al. 2006] and rely on decision trees ([6, Breiman et al. 1984] as base estimators.

First, a decision tree is a binary tree whose nodes can be divided into two categories. On the one hand, interior nodes are associated with a choice regarding one of the features of the samples of LS. On the other hand, each leaf node is labeled with class (resp. numerical value) in the case of classification (resp. regression) problem. A sample is going down the tree by taking the test of each interior node it meets. Once it ends up in a leaf node, it's labeled by its class or value. When a tree is grown, it's required to know which test perform on which feature in order to get the best tree as possible. Therefore, a quality measure must be associated to each that can be chosen at a given node. This measure is called the impurity and quantifies the diversity of a set according to the output variable y. More precisely, the more a test reduces the impurity of its associated set, the better.

Formally, let $\mathcal{N}$ be the set of nodes of a tree, $LS_n \subseteq LS$ with $n \in \mathcal{N}$ be the subset of the learning sample available at node n, $H : 2^{LS} \to C \subseteq \mathbb{R}$ be the impurity measure, where 2^{LS} refers to the power set of the original learning set and $P = \{p_1, p_2, ..., p_k\}$ the set of features of LS. To all features of P is also associated the domains $D_1, D_2, ..., D_k$. Moreover, let $p_i(o)$ the value of the i^{th} feature for sample o. Under these assumptions, the splitting criterion for a node n is the pair:

$$(p^*, d^*) = \underset{p_i \in P, d_j \in D_i}{\arg\max} \Delta H(\{o \in LS_n | p_i(o) = d_j\}),$$

with

$$\begin{aligned}\Delta H(\{o \in LS_n | p_i(o) = d_j\}) =& H(LS_n) \\ &- \frac{|\{o \in LS_n | p_i(o) = d_j\}|}{|LS_n|} H(\{o \in LS_n | p_i(o) = d_j\}) \\ &- \frac{|\{o \in LS_n | p_i(o) \neq d_j\}|}{|LS_n|} H(\{o \in LS_n | p_i(o \neq d_j\})\end{aligned}$$

where $|S|$ denotes the cardinality of set S. Once this splitting criterion is established, the learning set LS is divided into a left and right partition according to the results of the test. This operation is performed recursively until there are only objects left in the node, making it a leaf. Using this process, the resubstitution error becomes null at the end of the growing of the tree, minimzing therefore the loss function. However, if the tree is kept like this, the generalization error can be very large. That's why techniques as pruning have been developed in order to limit the growing of the tree and reduce the generalization error.

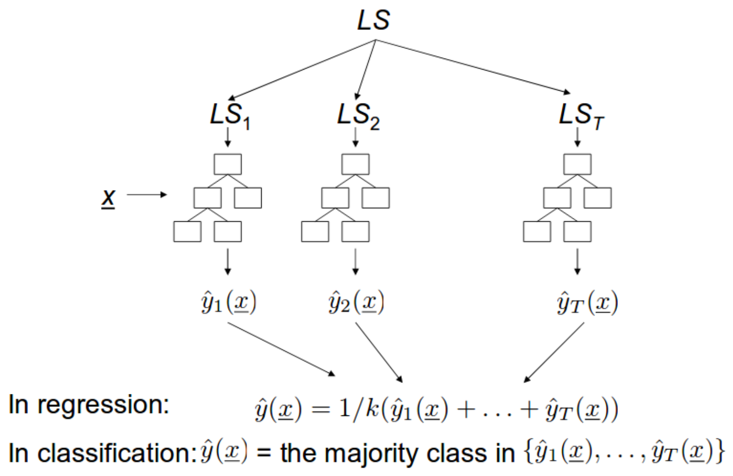

LS
LS_1
LS_2
LS_T
$\underline{x}$
$\hat{y}_1(\underline{x})$
$\hat{y}_2(\underline{x})$
$\hat{y}_T(\underline{x})$
In regression: $\hat{y}(\underline{x}) = 1/k(\hat{y}_1(\underline{x}) + \ldots + \hat{y}_T(\underline{x}))$
In classification: $\hat{y}(\underline{x})$ = the majority class in $\{\hat{y}_1(\underline{x}), \ldots, \hat{y}_T(\underline{x})\}$

FIGURE 2.5: Bagging illustration [5]

Let's now get back to extra trees. They actually consist in building forests, i.e. ensemble of trees, and to make a prediction by computing a sort of average of the prediction of all the trees belong to the forest. This principle is similar to Random Forests ([8, Breiman 2001]). However, they differ in the following way: in random forests, the randomness is due to the introduction of bagging in the learning set. As a reminder, Figure 2.5 illustrates the procedure of bagging. It consists in drawing

[5][7, Geurts 2016]

several learning sets with replacement from the original one and aggregating the predictions made for all these sets to obtain the final prediction. Conversely, in extra trees, the splitting thresholds are determined randomly. This induces several advantages compared to random forests. Firstly, using bagging reduces the learning size of approximately 36% for each tree, which is not the case for extra trees. Secondly, as the spitting threshold is chosen randomly, the the growing of the tree is much faster as it's not required to check all the features to compute the optimal threshold. Lastly, extra trees tend to obtain better results than random forests in most empirical situations (cfr [5, Geurts et al. 2006] for results).

2.1.3 Recurrent neural networks

Recurrent neural networks (RNN's) are particular types of neural networks introduced in [9, Hopfield 1982] as a mathematical model of particular biophysics processes. The first application of RNN's to solve large-scale tasks was done by [10, Schuster et al. 1997] in the context of signal processing. More generally, these kinds of neural networks are particularly adapated to deal with sequential data.

Let's first briefly introduce neural networks in general. The first model of neural network is the perceptron ([11, Rosenblatt 1958]). It consists of network of two layers: an input layer, related to the input value $\mathbf{x}$ of a given sample $o \in LS$ and an activation layer, to which is associated a weight vector $\mathbf{w}$, an offset b and an function

[6]Deeplearning4j: example of MLP representation

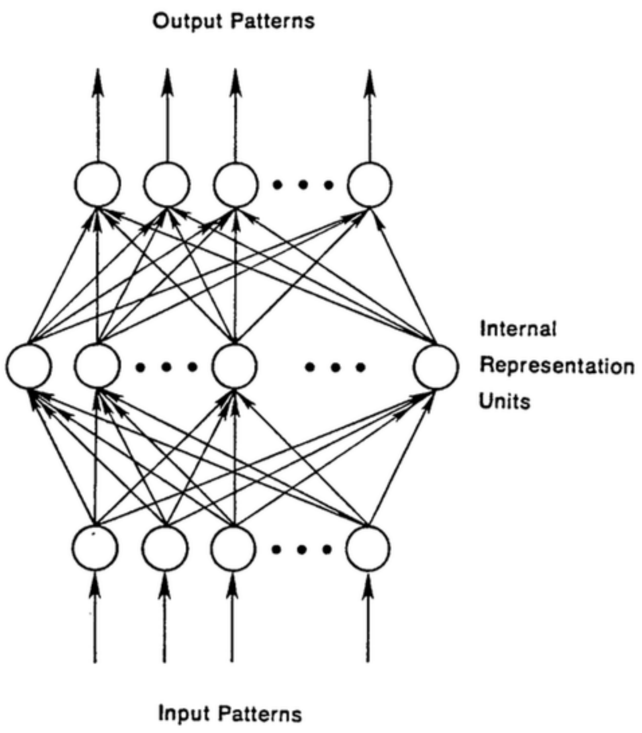


FIGURE 2.6: MLP: network representation [6]

$f(\mathbf{x})$ of the following shape:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_i w_i x_i + b \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

As the output is binary, the intrinsic goal of the perceptron model is to define an hyperplane separating the learning samples in the input space according to their output value, just like Support Vector Machine in section 2.1.1. However, contrary to SVM, the goal of the perceptron is not the maximization of the margin. Instead, the optimization problem is solved by gradient descent (which can be stochastic or not). Furthermore, like for SVM, solutions have been designed in order to deal with non-liear learning data. In the case of the perceptron, the component which is modified is the function $f(\mathbf{x})$ of the activation layer. A function widely used to deal with non-linearity is the sigmoid function.

Later, more complex networks called multi-layer perceptrons could be created when the backpropagation algorithm was proposed. This method is very practical as applying it implies that each neuron can learn its own model with the only constraint of knowing its direct neighbour's information. The algorithm is divided into the following steps. Firstly, the information about the learning samples is propagate forward the network to generate the output values of each neuron. Secondly, the error term is computed. Lastly, the output activations are propagated back through the network in order to compute the difference between the targeted and actual output values of all the hidden neurons and the weights of the different neurons are corrected according to the computed differences. As an example, Figure 2.6 illustrates the graph representation of a multi-layer perceptron. As it can be observed, the connections between two neurons are always made from a layer to another. This is where the RNN's differ from classical MLP's, as it will be explained next.

Now MLP's have been explained, RNN's can be tackled more easily. Indeed, as

[7] Deeplearning4j: example of RNN layer

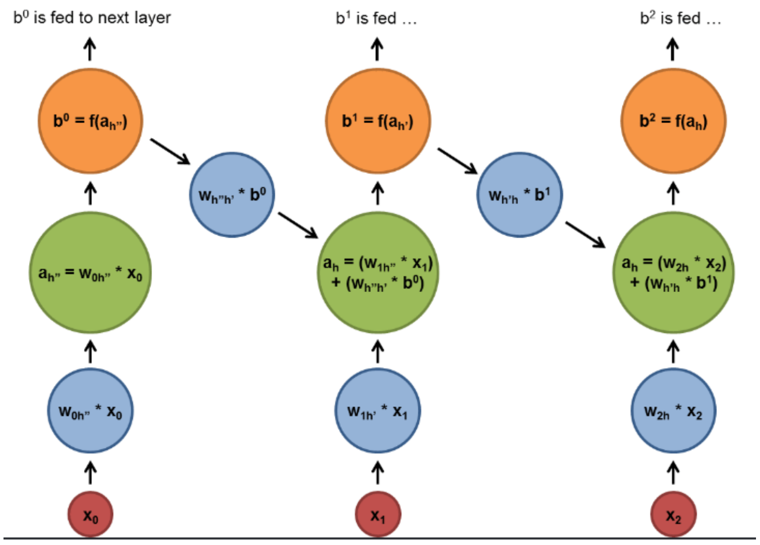


FIGURE 2.7: RNN: layer structure [7]

illustrated in Figure 2.7, a RNN layer includes one more type of connection compared to MLP's. Let assume the input $\mathbf{x}$ of a RNN corresponds to some time-series and can be written as $\mathbf{x} = (x_0, x_1, ..., x_t, ..., x_{T-1}, x_T)$ where T is the number of time steps. Then, the connection links the output neuron of the recurrent layer associated to some input value x_t with the hidden neuron of the same layer associated to the next input value x_{t+1}. Therefore, the value given to the activation function this next neuron becomes a function of the previous one.

The representation given in Figure 2.7 is what is called an unfold representation of the layer. There also exists a folded representation, in which the layer is represented as a single triplet of neurons. Fold and unfold layer representations are compared in Figure 2.8.

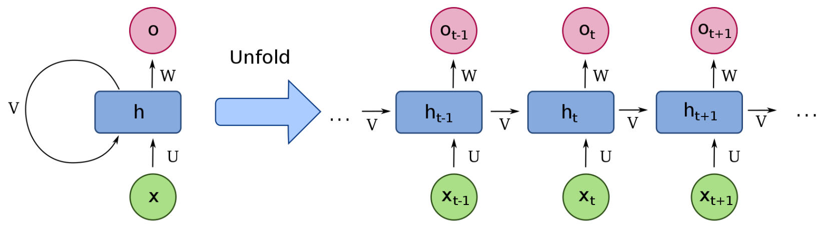


FIGURE 2.8: RNN: layer structure [8]

Nowadays, many various and more complex architectures have been developed. The ones that have been used in the scope of this master thesis will be introduced in the next chapter. Moreover, compared to the previous algorithms, the intrinsic complexity of such deep learning architectures increases outrageously. As an example, [12, Z. Mhammedi et al. 2016] details benchmark experiments conducted with RNN's of which the number of parameters to more than 180000. The running time of such networks can therefore be really important. Combined with time required to optimize all the hyper-parameters, the effective time to correctly train these networks sometimes make them impractical for some applications. However, there remain an incredibly interesting subject of research and their outstanding performances in fields such as natural language processing ([13, Q. Chen et al. 2017] still promise them a very bright future.

2.2 Anomaly detection

This section focuses on the concept of anomaly detection. First, it globally introduces the principle and the main different types of anomalies which are often met nowadays. It also focuses on a specific kind of anomaly detection problem, which is based on the comparison of probability distributions, and proposes a method based on supervised learning to solve this problem.

2.2.1 General considerations

As explained in [14, Chandola et al. 2009], anomaly detection can be defined as the ability of finding patterns which differ from an expected nature. Such distinctions

[8] Wikipedia: fold vs unfold RNN layer

are mostly referred as anomalies or outliers. Lately, this concept has been adapted to many application fields. Therefore, this gave the possibility to formalize it in a very general way. It's been given more and more importance in recent years because, in many cases, an anomaly in data can be associated to serious (sometimes critical) events.

To illustrate these explanations, Figure 2.9 shows a simple example of anomalies that can be found in a 2-dimensional space. Two normal regions can be easily highlighted on this graph: N_1 and N_2. Besides, three entities can be considered to be sufficiently distant from these regions: objects o_1 and o_2, as well as region O_3. It can be noticed that it's important to correctly make the distinction between anomaly and noise. Contrary to anomaly, noise has no interest to be subject to analysis and, most of the time, we want to remove it from the observed data. Of course, it's not the case for anomaly. Indeed, as we want to be able to identify outliers on purpose, the elements that characterize them should be carefully taken into account in the analysis.

Example shown in Figure 2.9 is only one among many types of outliers that can be detected in practice. Formally, anomalies are generally classified into 3 main categories:

- *Point anomalies*: the outliers introduced in the example of Figure 2.9 are part of this class. This is the most classical type of anomalies that are currently studied. It simply consists of single objects or whole regions that can be considered to be outside the boundary of predefined normal regions.
- *Contextual anomalies*: this type of anomaly introduces the concept of context, i.e. a specific situation defined by a set of attributes, called contextual attributes, which are parts of the features of the original data. On the other side, other attributes, called behavioral, are used to determine if an object is anomaly or not, conditionally to its contextual features. That means, given same behavioral attributes, an object could be classified as an anomaly or a normal instance if

[9] [14, Chandola et al. 2009]
[10] [14, Chandola et al. 2009]

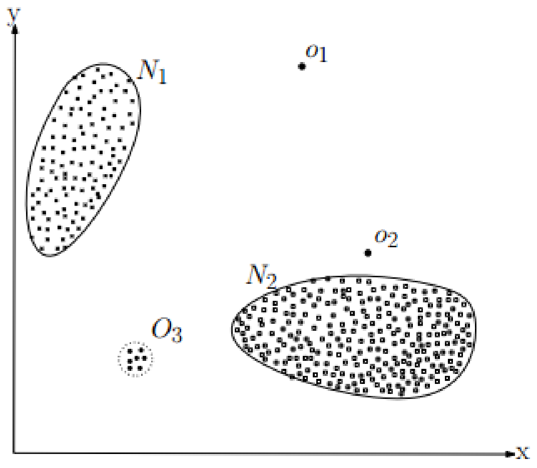


FIGURE 2.9: Example of anomalies in 2-dimensional space [9]

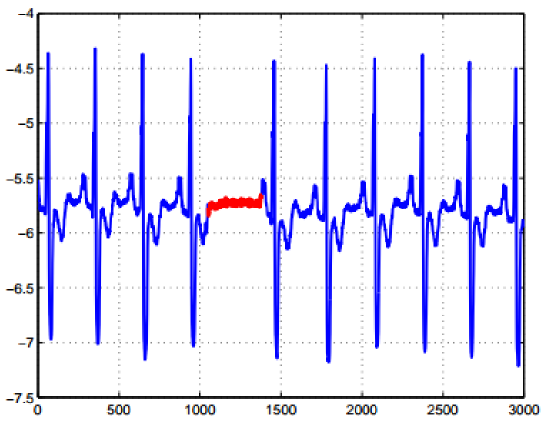


FIGURE 2.10: Example of collective anomaly in a human electrocardiogram output [10]

their contextual features are sufficiently different. Let's also notice that this notion of context must be a consequence of the structure of the initial and has to be specified in the problem statement.

An example of such anomaly can be given in the field of credit card fraud. For this problem, a contextual attribute could be the time of the purchase. Let's consider an individual spending 100€ per week. If this individual decides to spend 1000€ during Christmas week, this would be considered as an anomaly given the context of Christmas. However, if a similar transaction occurs in July for example, this would be detected as an outlier.

- *Collective anomalies*: such outliers consist of a group of data which, together, form a single anomaly. If they were occurring separately, they would not be detected as such. For instance, Figure 2.10 illustrates an example of this kind of anomaly. In this example, a abnormal sequence is highlighted out of a human electrocardiogram output. This sequence is considered as an anomaly because it remains at low values for a too long time. Let's note that an individual low value does not consist of an anomaly itself.

Some remarks can be made about these three categories. First, it's important to know that collective anomalies can be considered only there is exists a dependence between the different instances of the dataset, such as in time-series like in example of Figure 2.10. Then, it's possible to turn a point or collective anomaly into a context anomaly if it's possible to find attributes introducing a context in the problem. Therefore, any point or collective anomaly problem can be translated into a context anomaly problem if a correct contextual information can be expressed.

Further section will zoom on a specific anomaly detection problem based on the comparison of probability densities. The problem which is solved in this master thesis is an application of the generic task that will be described below.

2.2.2 Density-based anomaly detection

In some applications of anomaly detection, the goal is sometimes to identify standard patterns of a particular system and, afterwards, to compare these patterns with new observations of this system in a global way.

In order to make the link with the scope of this master thesis, let's take as an example an online casino game for which it would be required to detect abnormal behaviour. In particular, we want to check if the game still behaves correctly after being updated, assuming it was following its standard patterns before the update.

To do so, simulators which are able to generate data, just as if real people were playing the game, are available. This can be thus seen as a type of collective anomaly detection, as two sets of data are compared in their global aspect. With these simulators, it's possible to acquire data related to the game before and after the update. It can also be assumed that this data is generated according to some probability density depending on the intrinsic parameters of the casino. In this scope, the notation θ will be used to sum up all these parameters. Hence, $p(\mathbf{x}|\theta)$ denotes the associated probability where x represents a sample that can be generated by the simulator, just like it represents a sample in a machine learning dataset. These parameters may also differ when a game is updated, which is not always possible to forecast easily. As the probability density characterizing a version of the game is unknown, the analysis of the data is necessary to achieve the goal.

Let's now assume we want to compare two versions of a same game, each respectively denoted by parameter θ_0 and θ_1. Just like in supervised learning, data generated by the simulators can be viewed as a set of samples, each sample being described by a group of attributes. Therefore, the probability densities $p(\mathbf{x}|\theta_0)$ and $p(\mathbf{x}|\theta_1)$ can be seen as multivariate probability densities following each attribute. If $a_0, ..., a_m$ denote the attributes, we obtain:

$$p(\mathbf{x}|\theta_0) = p(a_0, ..., a_m|\theta_0)$$

$$p(\mathbf{x}|\theta_1) = p(a_0, ..., a_m|\theta_1)$$

At first sight, it could be thought about using some classical metrics providing an idea of the distance between the multivariate probability densities. However, in practice, the high dimensions of both samples and attributes make the computation of such metrics very difficult. This is why, in next section, a method allowing to deal with high dimensional probability density comparison will be introduced.

2.2.3 Likelihood ratio with supervised classification

In this section, a method which is able to perform high dimensional density-based anomaly detection efficiently is going to be explained. To apply this method, the same assumptions as in previous section 2.2.2 are made, i.e. two parametrized probability densities $p_{\mathbf{x}}(\mathbf{x}|\theta_0)$ and $p_{\mathbf{x}}(\mathbf{x}|\theta_1)$, with $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p$ and θ defining the parametrization of the probability density. As a reminder, the objective is to determine how much these probability densities differ from each other. The corresponding procedure has been introduced in [15, Cranmer et al. 2015] and relies on the concept of likelihood-ratio. Under the current assumptions and given a set of i.i.d.

observed data $D = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the likelihood-ratio is defined by:

$$\lambda(D; \theta_0, \theta_1) = \prod_{\mathbf{x} \in D} \frac{p_{\mathbf{x}}(\mathbf{x}|\theta_0)}{p_{\mathbf{x}}(\mathbf{x}|\theta_1)} \tag{2.2}$$

To compute this quantity, it must be possible to evaluate the two probability densities precisely. However, in most of applications, this is not the case. [15, Cranmer et al. 2015] therefore proposes a method allowing to approximate the likelihood ratio λ using supervised learning.

Approximating likelihood ratios with classifiers The main result of the article is the validation of a procedure allowing to transpose the likelihood-ratio expressed in equation 2.2 into another one which is equivalent for the probability densities $p_{\mathbf{x}}(\mathbf{x}|\theta_0)$ and $p_{\mathbf{x}}(\mathbf{x}|\theta_1)$. This has been done by proving that the following theorem is true:

Theorem 1 *Let $\mathbf{X}$ be a random vector with values in $\mathcal{X} \subseteq \mathbb{R}^p$ and a parametrized probability density $p_{\mathbf{x}}(\mathbf{x} = (x_1, ..., x_p)|\theta)$ and let $s : \mathbb{R}^p \to \mathbb{R}$ a function monotonic with the density ratio $r(\mathbf{x}; \theta_0, \theta_1)$, for given parameters θ_0 and θ_1. In these conditions,*

$$r(\mathbf{x}; \theta_0, \theta_1) = \frac{p_{\mathbf{x}}(\mathbf{x}|\theta_0)}{p_{\mathbf{x}}(\mathbf{x}|\theta_1)} = \frac{p_{\mathbf{U}}(u = s(\mathbf{x})|\theta_0)}{p_{\mathbf{U}}(u = s(\mathbf{x})|\theta_1)} \tag{2.3}$$

where $p_{\mathbf{U}}(u = s(\mathbf{x}; \theta_0, \theta_1)|\theta)$ is the induced probability density of $\mathbf{U} = s(\mathbf{X}; \theta_0, \theta_1)$

The validity of this theorem allows the initial problem to be formulated as a probabilistic classification problem, where the decision function

$$s^*(\mathbf{x}) = \frac{p_{\mathbf{x}}(\mathbf{x}|\theta_1)}{p_{\mathbf{x}}(\mathbf{x}|\theta_0) + p_{\mathbf{x}}(\mathbf{x}|\theta_1)}. \tag{2.4}$$

is modeled by a binary classifier used to be able to differentiate the two initial probability densities. Hence, the density ratio of a given sample can be expressed as:

$$r(\mathbf{x}; \theta_0, \theta_1) = \frac{p_{\mathbf{x}}(\mathbf{x}|\theta_0)}{p_{\mathbf{x}}(\mathbf{x}|\theta_1)} = \frac{1 - s^*(\mathbf{x})}{s^*(\mathbf{x})} \tag{2.5}$$

Notice that the monotonicity of $s(\mathbf{x})$ with respect to $r(\mathbf{x})$ is a sufficient but not necessary condition for the transposition to be correct. If the condition is not fulfilled, another solution is to use classifier calibration, which is going to be explained next. Let's notice that, from now, $p(\mathbf{x}|\theta)$ will denote $p_{\mathbf{x}}(\mathbf{x}|\theta)$, $p(s(\mathbf{x})|\theta)$ will denote $p_{\mathbf{U}}(u = s(\mathbf{x})|\theta)$, and $r(\mathbf{x})$ will denote $r(\mathbf{x}; \theta_0, \theta_1)$.

Learning and calibrating classifiers As stated by Theorem 1, the problem of approximating the density $p(\mathbf{x}|\theta)$ can be reduced to the evaluation of $p(s(\mathbf{x})|\theta)$. Again, this must be done from finite samples set $D = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$. There is no direct way from which $p(s(\mathbf{x})|\theta)$ can be computed directly but an approximation $\hat{p}(\hat{s}(\mathbf{x})|\theta)$ can be constructed from the set $D' = \{\hat{s}(\mathbf{x}_1), ..., \hat{s}(\mathbf{x}_n)\}$. Indeed, from density estimation algorithms such as kernel density estimation, it's possible to build a probability density function from a set of sample. The main advantage of this problem transposition is the projection from a p-dimensional space onto a one-dimensional space. This simplifies a lot the representation of the problem and, at the same time, allows

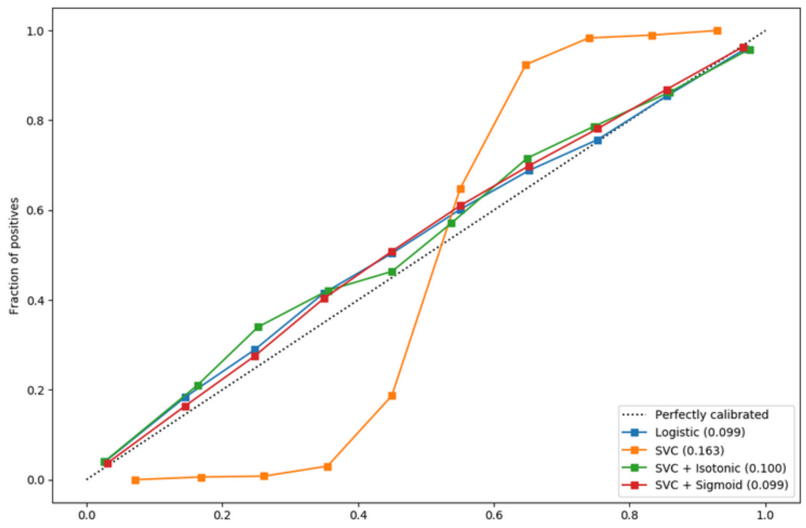


FIGURE 2.11: Example of probability calibration for SVC [11]: probability vs fraction of positive class

to preserve the information required to approximate $r(\mathbf{x})$.

Moreover, as said in previous section, when $s(\mathbf{x})$ is not monotonic with respect to $r(\mathbf{x})$, a solution proposed by the article is to apply probability calibration on the binary classifier. As explained in [16, Niculescu-Mizil et al. 2005], when classification is performed, an output corresponding to the probability is also often required, such as in the likelihood ratio problem described in this section. More precisely, when calibrating a classifier, the probability output by the classifier can directly be interpreted as a confidence level. Therefore, if a calibrated classifier predicts a value close to 0.8 for a group of samples, around 80% of these samples are expected to belong to the positive class. An example of the consequence of calibration is shown in Figure 2.11. In this situation, it's applied to SVC algorithm and, as expected, the curve showing the predicted probability with respect to the fraction of positive class becomes much smoother when calibration is performed.

[11] `http://scikit-learn.org/stable/modules/calibration.html`

Chapter 3

Objectives, Motivations and Methods

In this chapter, the different objectives of this thesis are laid down and formalized in the scope of the different approaches.

3.1 Main objective and motivations

At the moment, Gaming1 uses two principal means in order to detect problems linked to the game engine module of their casino games. On the one hand, released games are monitored by a Risk & Fraud department which reports the identified issues. On the other hand, during the development and updating stages, tests are performed using simulation engines and the generated statistics are manually analyzed by Gaming1's game engine team.

The main objective is to improve this second procedure for two reasons. Firstly, using machine learning techniques would allow to automatize the operation. Gaming1's developers could then focus on other complex tasks. Secondly, the currently used tests are not 100% reliable, as some problems still occur after passing them. Therefore, the machine learning-based method will have the following objective: for a given casino game, provide a procedure that will be able to compare the standard patterns of the game to the patterns of the updated game. To achieve this, two approaches will be tested, both relying on the concept of density ratio explained in section 2.2.3. The first one will be based on standard supervised learning classification algorithms, such as ensemble methods or support vector machine, while the second one will exploit neural networks architectures used in deep learning.

Further sections will now address the formalism and methodology related to the objective described in this section.

3.2 Methodology

This section introduces the whole methodology related to the objective of the master thesis. In particular, it formalizes the problem an introduces the metrics that will be used to evaluate the algorithms.

3.2.1 Problem formalization

This section formalizes the objective in the scope of supervised learning using some of the concepts introduced in chapter 2. Then, it introduces the metrics that will be used in Chapter 4 to assess the performances of the anomaly detection procedure.

To solve this problem, it's been chosen to rely on the concept of likelihood-ratio introduced in section 2.2.3. Indeed, to a given version of a casino game can be associated a probability density $p(\mathbf{x}|\theta)$ where $\mathbf{x}$ corresponds to the input data that will be used in the machine learning algorithm, whatever it is, and θ is associated to the intrinsic game engine parameters influencing the behaviour of the game. Therefore, when a game is updated, 2 versions are available, each one having its own probability density. Let $p(\mathbf{x}|\theta_0)$ (resp. $p(\mathbf{x}|\theta_1)$) be the probability density of the original (resp. updated) version the game. Therefore, as stated in [15, Cranmer et al. 2015], the problem can be solved as a supervised classification problem. If $\hat{s}(\mathbf{x})$ denotes the decision function of the classifier trained by the associated supervised learning algorithm, equations 2.4 and 2.5 hold for this problem and the approximation $\hat{s}(x)$ can be used to model the probability density $p(\hat{s}(\mathbf{x})|\theta)$. This will correspond to a sufficiently good approximation of equation 2.4.

It's important to understand to the problem we want to solve does not correspond to a typical supervised learning problem. Indeed, the goal here is not to train a model that will be re-used later to make some predictions about new data. In the present case, each game update corresponds to an independent supervised learning problem to solve using the technique introduced in section 2.2.3. Once the density ratio of all the samples has been approximated by the trained, this last won't be used anymore.

As said in previous section, two approaches are going to be compared for this problem. The main difference between them, except the types of used algorithms, is the input data that they require. This distinction will be explained in further sections. At this stage, it's assumed to have a dataset $D = \{\mathbf{x}_i, y_i | i = 1, \ldots, N\}, y_i \in \{0, 1\} \ \forall i$. Label 0 (resp. 1) is associated to original (resp. updated) data. It's also assumed that the proportion of both types of data is approximately the same. Up to now, the only assumption made on $\mathbf{x}_i$ is that it contains real values. The preprocessing procedure for each approach will be detailed in corresponding sections. Moreover, N is assumed to be quite large as there is no clear bound on the amount of data that can be generated by simulations.

3.2.2 Evaluation metrics

Besides, to evaluate correctly the performance of the density-based anomaly detection method, suitable metrics are required to provide a relevant analysis of the outputs of the method. Indeed, more than simply using the classic metrics such as accuracy, precision, etc... used in machine learning, the current problem requires to compare probability densities. Let p and q be two probability densities to compare. To do so, the most widely used measures are the following:

- the Jensen-Shannon divergence defined as:

$$JSD(P \,||\, Q) = \frac{1}{2}D(P \,||\, M) + \frac{1}{2}D(Q \,||\, M)$$

  where:

  - $M = \frac{1}{2}(P + Q)$.

- $D(P \,||\, Q)$ is the Kullback–Leibler divergence between P and Q. When P and Q are distributions of a continuous random variable, it is defined as the following:

$$D(P \,||\, Q) = \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} \, dx$$

  where p and q denote the densities of P and Q.

- The Wasserstein-1 distance, as explained in [17, Louppe 2018], defined as:

$$W_1(p, q) = \inf_{\gamma \in \prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||]$$

  where:

  - $\prod(p, q)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively p and q.
  - $\gamma(x, y)$ indicates how much mass must be transported from x to y in order to transform the distribution p into q.
  - $||.||$ is the L1 norm and $||x - y||$ represents the cost of moving a unit of mass from x to y.

  More intuitively, the Wasserstein-1 distance corresponds to the minimum mass displacement required to transform one distribution into the other. Figure 3.1 shows how this distance is calculated for a simple example. We have the following two densities:

  - In blue, $p = \frac{1}{4}\mathbf{1}_{[1,2]} + \frac{1}{4}\mathbf{1}_{[3,4]} + \frac{1}{2}\mathbf{1}_{[9,10]}$
  - In red, $q = \mathbf{1}_{[5,7]}$

  The arrows displayed on the Figure represent the minimum cost for moving each part of p onto q. Therefore, we have:

$$W_1(p, q) = 4 \times \frac{1}{4} + 2 \times \frac{1}{2} + 3 \times \frac{1}{2} = 3$$

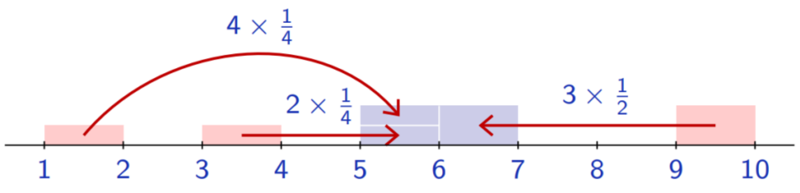


FIGURE 3.1: Wasserstein distance: example [1]

As a reminder, we want to observe if there is a notable difference between the two probability densities $p(\mathbf{x}|\theta_0)$ and $p(\mathbf{x}|\theta_1)$, respectively labeled by $y = 0$ and $y = 1$ in the dataset D. To do so, we dispose of the estimates $\hat{s}(\mathbf{x})$ and $r(\mathbf{x})$, $\forall\, \mathbf{x} \in D$. Therefore, for a given anomaly detection problem, it's been decided to analyze the following outputs:

[1] [17, Louppe 2018]

- Probability densities $p(\hat{s}(\mathbf{x})|y = 0)$ and $p(\hat{s}(\mathbf{x})|y = 1)$ along with their JSD and W_1 measures. This allows a preview of what the classifier has achieved to perform without having to compute the density ratio yet.
- Two graphs displaying each $r(\mathbf{x})$ w.r.t. $\hat{s}(\mathbf{x})$ and the probability density $p(r(\mathbf{x}))$. These are the classical outputs in density ratio problems solved with supervised learning. These allow an accurate display of the approximation of the target value defined by equation 2.2. These graphs can also be translated into the variance of the density ratio $r(\mathbf{x})$ if only a numerical evaluation is required.
- the ROC curve of the model to provide a more classical view of the performances of the classifier. In the scope of the problem, if $p(\mathbf{x}|\theta_0) \sim p(\mathbf{x}|\theta_1)$, the ROC curve should be close to the 45° line; otherwise, it should be clearly distinct from it.

Considering all these metrics, the analyses should be sufficiently complete to distinguish relevant differences between the probability densities associated to two versions of a game. Next sections will focus on describing the specific type data that is going to be exploited, and the preprocessing that should be applied for each approach.

3.3 Raw data collection

This section addresses the procedure allowing to collect the data that will be used to solve the problem. As said in section 3.1, data is generated using game engine simulators. A simulation will perform a given number of turns (called spins) spread into different sessions (called Game Instance Player). Not that the size of a GIP is defined by the simulator as a random variable with a uniform distribution over the interval $[1, 1000]$. The data corresponding to a given GIP is the following: every 10 spins, the difference between the amount of won money and the amount of played money (also called stake) is retrieved. Note that a simulation is performed for some given parameters. Therefore, to make the link with the abstract data defined at section 3.2.1, all the sessions generated by a given simulation will correspond to a fixed value of θ. Hence, the same label will be assigned to all of them. Thus, to dispose of a complete dataset, two distinct simulations must be performed, with their associated values of θ. Considering the amount of data that will be used for our analysis, a simulation is assumed to produce 5 millions of spins. Thus, a complete dataset will contain information related to 10 millions of spins.

Index	Gip_id	Time	Win_ratio
0	id_1	1	$G_{\text{ratio}}(1,1)$
1	id_1	2	$G_{\text{ratio}}(1,2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-2$	id_n	$t-2$	$G_{\text{ratio}}(n,t-2)$
$N-1$	id_n	$t-1$	$G_{\text{ratio}}(n,t-1)$
N	id_n	t	$G_{\text{ratio}}(n,t)$

TABLE 3.1: Raw data format

[2] [18, Merchie 2017]

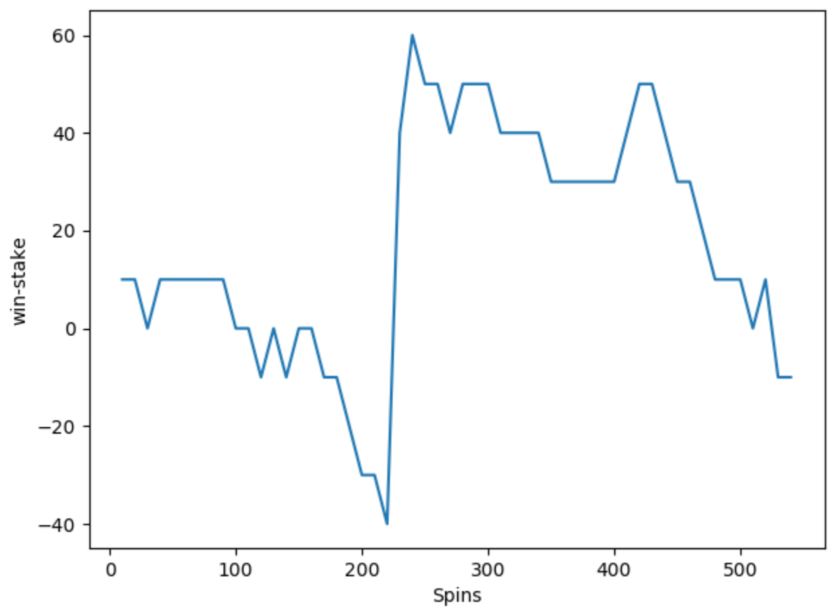


FIGURE 3.2: Example of data related to GIP [2]

An example of such time-series is displayed in Figure 3.2. It can be noticed that, like for spins, the y-axis value is also rounded to the closest multiple of 10. This is a constraint imposed by the simulator itself, reducing the cost for collecting the information because, usually, this kind of simulation are performed for several millions of spins. Of course, there is a non negligible loss of information by proceeding like that but this is the price to pay to be able to analyze a large amount of spins within one simulation. Moreover, in casino games, the main interest of such data analysis mostly resides in the general trends of GIP graphs more than in local behaviours. After the termination of the simulation, the whole data collection is written in a `.csv` file of which the format is described by Table 3.1.

Of course, data collected in such a format is not suitable for any machine learning algorithm. Therefore, it has to be preprocessed according to the approach that is going to be used. Next sections describe, for each approach, the preprocessing procedures that have been designed.

3.4 Data preprocessing

In this section, the issue of data preprocessing is addressed. For each approach, we describe the whole procedure allowing to transform raw data into a dataset that could be directly used by supervised machine learning algorithms.

3.4.1 Standard supervised approach

For this approach, it's required to transform the data described by Table 3.1 into a format corresponding to the standard input format for supervised learning described at Section 2.1. At first, it could be thought to directly use the values of the time-series as features. But, unfortunately, as explained in previous section, the length of a time-series is a defined as a random variable in the simulator. As a consequence, the number of features would not be the same for all samples, which can't happen

Index	Gip_id	Time	Win_ratio
0	id_1	1	$G_{ratio}(1,1)$
1	id_1	2	$G_{ratio}(1,2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-2$	id_n	$t-2$	$G_{ratio}(n,t-2)$
$N-1$	id_n	$t-1$	$G_{ratio}(n,t-1)$
N	id_n	t	$G_{ratio}(n,t)$

$\downarrow$

	ft_1	ft_2	ft_3	...	ft_p
id_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	...	$x_{1,p}$
id_2	$x_{2,1}$	$x_{2,2}$	$x_{3,2}$	...	$x_{2,p}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
id_n	$x_{n,1}$	$x_{n,2}$	$x_{n,3}$	...	$x_{n,p}$

TABLE 3.2: Illustration of the feature extraction procedure

in supervised learning. Therefore, it was required to use a more complex feature extraction method. To do so, a strategy that has been discussed in a previous internship report [18, Merchie 2017] has been applied. This mainly relies on a toolbox ([19, Christ 2016]) allowing to process time-series by automatically extracting a large number of features and build a dataset such as described by Table 3.2.

Nevertheless, the utilization of this method in the scope of the problem lead to some issues as explained in section 6.1.1 of the internship report [18, Merchie 2017]. Indeed, some time-series contain an insufficient number of measures for all the features to be computed correctly. To counter this problem, the following solutions were designed:

- Concatenating some time-series between each other according to a minimal length. Note this makes sense as it's assumed that every time-series of a given simulation is generated according to the same parameters. In this way, longer time-series are artificially created and features can be be computed with more accuracy. However, a trade-off is faced when using this solution. indeed, if time-series are concatenated, that means less samples will be available in the final dataset given a same amount of resources spent to generate the raw data. It's been chosen to analyze the influence of time-series concatenation for different values of the minimal length. It's been decided to set this parameter to 0 (i.e. no concatenation), 500, 1000 or 2000. Note that, for 2000, all features can be computed correctly.

- A more straightforward strategy consisting in using missing values imputation methods. A detailed analysis has been performed during the internship to determine the method providing the best results in the scope of the problem. This enhanced that techniques based on singular value decomposition of matrices gave the best results. Note that, during this step, features containing too many missing values are removed from the dataset. The number of available features will therefore be a function of the minimal concatenation length defined by the first solution.

Combining these two solutions allow therefore to obtain, for a given collection of raw data, a complete dataset of which the samples are labeled according to the parameters of the simulation from which they have been generated. Considering the size of the datasets, this depends of course on the value of minimal concatenation length. Table 3.3 details this relation for the chosen values of the minimal length. It can be observed that, from the point where concatenation is performed, the size of the dataset sharply decreases. As a consequence, cross-validation techniques should be used in this case. In particular, cross-validation preserving the class proportion within the subset, such as stratified *k*-fold cross-validation, will be applied in this scope. Next section will describe the equivalent procedure for the deep learning approach.

3.4.2 Deep learning approach

Concerning deep learning approach, there is one major difference compared to standard supervised approach in terms of data preprocessing. Indeed, recurrent neural networks such as described in section 2.1.3 are able to process time-series directly. Therefore, the feature extraction combined with missing values imputation described in previous section is not required anymore here. However, it could still be interesting to observe the effect of the concatenation of time-series, independently of the fact that it helps increasing the number of features that can be correctly computed for standard supervised approach. Thus, Table 3.3 is still valid as far as datasets' shape is concerned.

Moreover, it's still needed to provide the neural network samples of the same size. To do so, an existing methods consists in truncating and padding the different sequences with 0's. The neural network will then simply learn that such values contain no relevant information. Once it's done, it should also be interesting to standardize the data. This has been shown to improve the accuracy in some cases.

In next section, the algorithm allowing to actually be able to detect anomalies in casino games will be explained.

3.5 Update verification procedure

In this section, the actual procedure allowing to achieve the objective described in section 3.1 is presented. First, the algorithm and its main supervised learning components are described, followed by an explanation of the way this algorithm will be indeed validated.

Minimal concatenation length	Size of the dataset	Number of features
0	$\sim$ 40000	$\sim$ 400
500	$\sim$ 15000	$\sim$ 500
1000	$\sim$ 8000	$\sim$ 600
2000	$\sim$ 4000	$\sim$ 800

TABLE 3.3: Shape of the final dataset according to the minimal concatenation length

3.5.1 Description

Up to now, the problem statement has been formalized and all the preprocessing steps allowing to obtain exploitable datasets have been discussed. In this section, the complete procedure to verify if the behaviour of a casino game has changed after an update will be detailed. As reminder, it globally consists, from simulations generated before and after the update, in providing an estimate of the density ratio between the two theoretical probability densities associated to the versions of the game. Typically, the method reads the contents of the files containing the results of the simulations, and applies the different preprocessing techniques discussed in previous sections, according to the approach. Then, cross-validation is applied to create successive learning and testing sets. For each of these pairs, a learning procedure is performed and the metrics introduced in section 3.2.2 are computed. This can be divided into the following parts:

1. For each simulation, reading the corresponding file into a dataframe.
2. Applying the preprocessing techniques according to the chosen minimum concatenation length and supervised learning approach.
3. Splitting the obtained dataset into k subsets thanks to stratified cross-validation.
4. For each learning/testing set pair, train a density ratio classifier according to a chosen base estimator and associated set of hyper-parameters using the learning set.
5. Predicting the probability associated to equation 2.4 and the log of the density ratio $r(\mathbf{x})$ for each sample of the testing set.
6. Storing all the predictions for all training/testing set pairs.
7. Computing and returning the required metrics.

The pseudo-code of this method is presented by Algorithm 1. It's noticed that several arguments are given to the procedure. Most of them have already been discussed. On the one side, $file$ and $updateFile$ denotes the files containing results of the simulations, which will constitute the input of the preprocessing step. On the other side, $approach$ either denotes the standard or deep learning strategy and $length$ refers to the minimum concatenation length. The last argument that must be discussed is $estimator$. This contains all the information about the algorithm that will be used to learn the data. But the classifier is actually much more complex in our case. It will indeed base composed of several layers, each of them having a set of hyper-parameters that can be tuned. This layer decomposition was necessary because of the different metrics that are required to be computed. Figure 3.3 illustrates this layer decomposition as well as the different components of each of these layers. Starting from the bottom:

- The first layer is made up of the blocks `Base Estimator` and `Parameters`. This corresponds to the classical instance of a supervised learning estimator, associated with its given set of tunable hyper-parameters. In this scope, its role is to provide the predictions of the class probability described by equation 2.4. The following types of estimator will be compared during the analysis part:
  - `Extra Trees` and `Linear SVM` in the case of the standard supervised approach.

Algorithm 1 Update verification procedure

```
1: function PROCEDURE($file, updatedFile, approach, length, estimator$)
2:     $data0 \leftarrow$ READ($file$)
3:     $data1 \leftarrow$ READ($updatedFile$)
4:     $dataset \leftarrow$ PREPROCESS($data0, data1, approach, length$)
5:     $nbCol \leftarrow dataset.shape[1]$
6:     $X \leftarrow dataset[:][: nbCol]$
7:     $y \leftarrow dataset[:][nbCol]$
8:     $skc \leftarrow$ STRATIFIEDKFOLDCV($length$)
9:     $ratios \leftarrow []$
10:    $predictions \leftarrow []$
11:    $trueLabels \leftarrow []$
12:    for $trainIndex, textIndex \in skc$.SPLIT do
13:        $XTrain,\ yTrain \leftarrow X[trainIndex][:],\ y[trainIndex][:]$
14:        $XTest,\ yTest \leftarrow X[testIndex][:],\ y[testIndex][:]$
15:        $clf \leftarrow$ INSTANTIATERATIOCLASSIFIER($estimator$)
16:        $clf$.FIT($XTrain, yTrain$)
17:        $rat,\ pred \leftarrow clf$.PREDICT($XTest$), $clf.baseEstimator$.PREDICTPROBA($XTest$)
18:        $ratios$.APPEND($rat$)
19:        $predictions$.APPEND($pred$)
20:        $trueLabels$.APPEND($yTest$)
21:    $metrics \leftarrow$ COMPUTEMETRICS($ratios, predictions, trueLabels$)
22:    return $metrics$
```

  - `Recurrent Neural Networks` in the case of deep learning approach.

- The second layer corresponds to the calibration of the first one. As explained in Chapter 2, this aims at increasing the robustness of the density ratio estimation in the case where the probability prediction of the estimator is not monotonic with respect to this ratio. Note that this layer is not mandatory. Therefore, the layer itself will be considered as a hyper-parameter for the analysis, as well as the `Calibration Method` block. The `Cross-Validation Method` blocks is assumed to be of type Stratified k-fold cross-validation and is necessary to perform calibration.

- Last layer encapsulates every other as a `Density ratio Classifier`. Its task is to provides the actual estimate of the density ratio using the elements of all the previous layers.

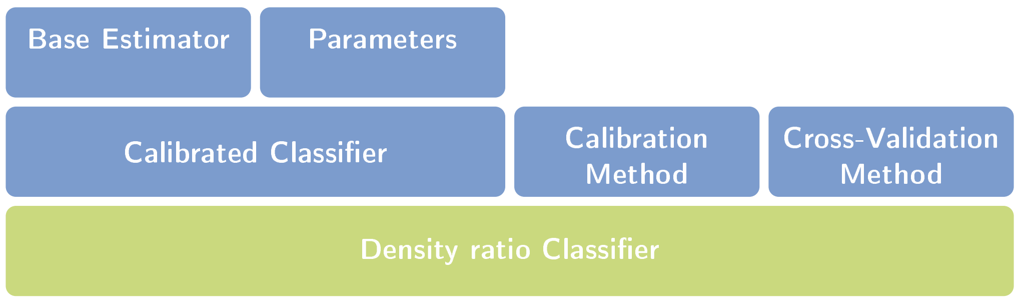


FIGURE 3.3: Layers representation of the classifier of Algorithm 1

These 3 combined layers form a classifier providing the information to compute all the metrics described at section 3.2.2 and thus, to perform a relevant analysis between two probability distributions. However, it can't be assessed yet that Algorithm 1 is correct. Validation cases for which the expected result is already known must be given to the procedure in order to check its correctness. Next section will present and justify these validation situations in the scope of the casino games.

3.5.2 Validation

This section describes the different cases that have been used to assert the validity of the proposed method. To evaluate the performances of the algorithm, analyses will be performed on four types of games: slot, dice, blackjack and roulette. In this section, the different configurations in which these games will be set for the tests are briefly explained.

First, we want of course the algorithm to be able to detect modifications in the behaviour of a given game. These behaviour changes could be voluntary added into the game, such as modifications brought to a game in order to make it more attractive to the player. But also, undesired changes could occur in a game, such as bugs unintentionally incorporated during the development and which are not immediately detected during the test stage. Both types of changes should be identified by the proposed algorithm. It's thus been chosen to insert one anomaly in a game set of each different types of games. The anomalies are the following:

- Slot: increasing of the probability of the bonus symbol of 30%.
- Dice: real bug that occurred in a past update of the game and which has been detected only after the releasing of the game. Briefly, the bug was detected because it was noticed that it was possible to detect phases where the game would pay much more than usual. Therefore, the players being able to do so played very small stakes while the game was not in such a phase, and switched to higher stakes when they detected the phase change.
- Blackjack: Ace cards have been removed from the shoe.
- Roulette: probabilities of 4 symbols have been set to 0.

Secondly, we also want the method not to warn too often about changes occurring after the updating of a game. Therefore, some situations must be defined and tested where the analysis of the output of the algorithm should not conclude to an anomaly in the game. At least, there should be clear differences between the cases where anomalies are expected compared to the situations where nothing abnormal should be observed. Thus, as a comparison with the bugs described previously, another dataset will be analysed. This time, both classes will come from a same game set. Hence, it should not be possible for the classifier to separate the data as accurately as when bugs should be observed.

As a summary, for a given case, the validation proceure can be decomposed into the following steps:

1. Apply Algorithm 1 where $file$ (resp. $updatedFile$) contains data generated by the correct (resp. bugged) version of the corresponding game. Store the metrics obtained from this analysis.

2. Apply Algorithm 1 where both $file$ and. $updatedFile$ contain data generated by the correct version of the corresponding game. Store the metrics obtained from this analysis.

3. Display the obtained metrics. An analysis of the test samples will consists in five different graphs:

   - The probability densities $p(\hat{s}(x)|y)$ for each label in the case where one of the two labels corresponds to a bugged version of the game. From here and in Chapter 4, this is denoted as positive case. In this graph, the two densities should be easily differentiated.
   - The probability densities $p(\hat{s}(x)|y)$ for each label in the case where none of the two labels corresponds to a bugged version of the game. From here and in Chapter 4, this is denoted as negative case. In this case, the two densities should approximately have the same shape. Hence, their JSD and W_1 distance should be negligible compared to these of the two previous densities.
   - The log density ratio $\ln(r(x))$ as a function of the of the prediction $\hat{s}(x)$ given the original case (bug or no bug inserted). For this graph, the samples corresponding to the positive case should spread along a larger range of values than the samples of the negative case. Therefore, their variance should also be higher.
   - The probability density of the log density ratio $\ln(r(x))$ given the original case (bug or no bug inserted). Just as the previous, the density associated to the positive case should be more flattened and spread than the density of the negative case. Hence, their JSD and W_1 distance should not be negligible.
   - The ROC curve associated to each of the two cases. Here, the accuracy corresponding to the area under the curve, should of course be larger in the positive case.

It's important to notice that the probabilities densities detailed in step 3 aren't true original densities. Indeed, there are approximations built from the histograms of the test samples for the corresponding metric. For the experiments conducted in Chapter 4, the approximation method which has been used is the Kernel Density Estimator (KDE). As explained in [20, Gutierrez-Osuna 2016], this estimator approximate the shape of a univariate density p by an i.i.d. sample $D = x_1, ..., x_N$ drawn from this distribution by using the estimator:

$$\hat{p}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K(\frac{x - x_i}{h})$$

where:

- N is the number of samples.
- K is the kernel, a non-negative function integrating to 1.
- $h > 0$ is a smoothing parameter, called the bandwidth.

In practice, the choice of the bandwidth parameter h is crucial. To illustrate this, Figure 3.4 shows the resulting density function estimated from 500 i.i.d. samples

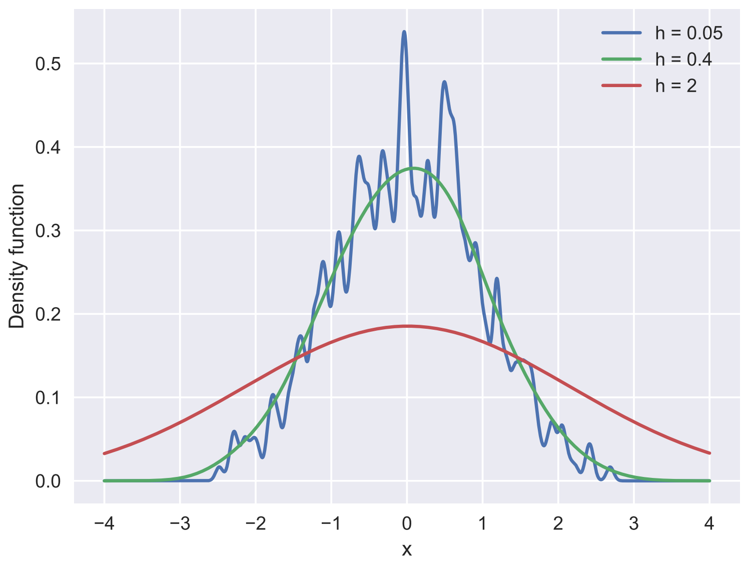


FIGURE 3.4: KDE example for a standard normal distribution: estimation made from 500 i.i.d. samples

drawn from a standard normal distribution for 3 different values of h: 0.05, 0.4 and 2. The curve corresponding to the smallest value of h is clearly undersmoothed and becomes highly noisy, while the curve corresponding to the highest value of h suffers from the exact opposite problem. The effects of the bandwidth parameter are thus clear. One the one hand, high values reduce the variance from one samples set to another but the bias with respect to the true density globally increases. On the other hand, small values reduce the bias with respect to the samples set, but as a consequence, the curve becomes much more dependent on this dataset. To overcome this bias/variance trade-off induced by the choice of h, [20, Gutierrez-Osuna 2016] suggests to select the value that minimizes the mean squared error (MSE) between the estimated density and the true density:

$$MSE_x(\hat{p}(x)) = E[(\hat{p}(x) - p(x))^2] = E[\hat{p}(x) - p(x)]^2 - V[\hat{p}(x)]$$

This bias/variance decomposition appears in the right-hand term. However, in practice, the true density is course not often available. Therefore, some assumptions about this true density must be made in order to be able to solve this optimization problem.

In this section, it's thus been explained how the procedure defined by Algorithm 1 was going to be validated and what kind of information is going to be displayed in Chapter 4. In next section, the implementation of the different parts of the software will be described.

3.6 Implementation and architecture

In this section, some details about the implementation are provided. The goal is to see how the mechanisms introduced in previous sections of this chapter render into code. Also, a part of this section is exclusively dedicated to the architecture of the deep learning approchoach. Indeed, as the structure of a neural network can be very complex, it's important to explain its different components.

3.6.1 Software architecture

Concerning the architecture of the software, it's been decided to divide it into several modules corresponding to the main different steps of the anomaly detection procedure: the extraction of features from the simulation data, the imputation of missing values into the datasets obtained from feature extraction, and the machine learning procedure itself. It's important to be aware that the software has been designed the very specific problem tackle in this master thesis. Transposition to other problems is therefore not the main purpose here. The software has been implemented on a machine featuring a Windows 10 Pro OS, a Intel Xeon CPU of 3.50GHz, 4 cores and 8 logical processors, with 16GB of RAM.

The static diagram of Figure 3.5 represents the structure and the relations between the different modules of the software. Let's notice that the code, written in Python 3.6, is not all brand new. For several modules, existing packages have been used in order to limit the programming time. Let's first describe the features extractor. This module is responsible for the first part of the operations explained in Section 3.4: the concatenation of the different GIP one after each other and, only for the standard approach, the extraction of statistical features from temporal data. The latest has been done using a feature extraction package especially designed to deal with time-series, called tsfresh ([19, Christ 2016], version 0.11.0). After concatenating the GIP given a minimal length and potentially extracting features, the module produces a dataset such as described In Table 3.2

Then, there is the Missing Values Imputer, dealing with second part of the procedure described in Section 3.4. As a reminder, this is divided into 3 main steps. Firstly, the features having too many missing values are removed. Secondly, the dataset is prepared for the imputation with a standardization procedure. Finally, the missing

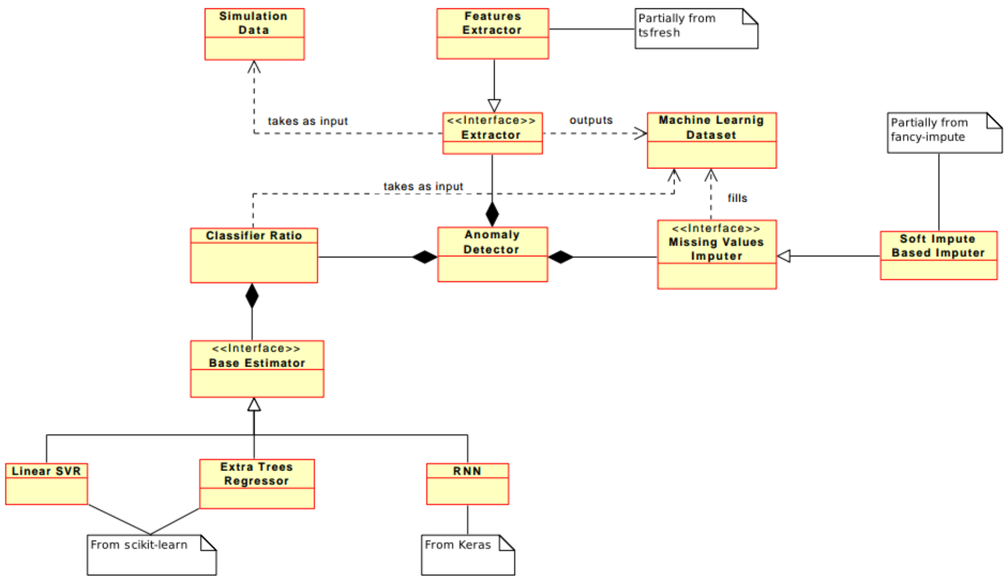


FIGURE 3.5: Static diagram of the software

values are actually imputed using the package fancyimpute ([21, Rubinsteyn 2016], version 0.2.0). This package proposes several techniques in order to perform this task. According to the analysis performed in [18, Merchie 2017], the Soft Impute method was chosen. After being processed by this module, the dataset should be fully ready to be used for the actual anomaly detection procedure.

Moreover, The machine learning procedure is performed by the module Classifier Ratio. This module corresponds to the layers-based classifier described in Figure 3.3. Let's note that the calibration layer is here directly included in the Classifier Ratio module. Several packages have been used in order to implement this part of the software. First, the Classifier Ratio itself finds it implementation in the carl package ([22, Louppe et al. 2016], version 0.2). This package has been especially made for solving likelihood-free inference problems. In particular, the density ratio-based anomaly detection presented in this master thesis is one of these sub-problems. Furthermore, in order to perform this task, the module needs a base estimator, i.e. an actual machine learning algorithm to which the supervised learning task will be dedicated. The implementation of these algorithms is found in the scikit-learn package (version 0.19.1).

Finally, the module allowing to use all the previous ones together in order to complete the required task is the Anomaly Detector. For this master thesis, it groups all the experiments that have been performed to assess the performances of Algorithm 1, which will be analyzed in Chapter 4. Considering the final application which be used later by GAMING1, this will consists of a class receiving the simulation data as input and calling the appropriate functions of the other modules.

Some other more frequently used packages are also part of the implementation. Particularly, there are numpy (version 1.14.2), seaborn (version 0.8.1) for the design of the graphs, and scipy (version 1.1) for the computation of statistical data. In the next section, the architecture related to the deep learning approach is tackled. This corresponds to the RNN base estimator of the diagram of Figure 3.5.

3.6.2 Deep learning architecture

This section deals with the particular architecture of the module RNN of the class diagram of previous section. Indeed, as the complexity of the architecture of neural networks tends to increase nowadays, it's important to introduce the different components of a neural network and to show how they're gathered together.

First, let's first describe the basic unit of the architecture of the network that has been used in this master thesis. It's called LSTM (Long Short-Term Memory) and was proposed by [23, Hochreiter et al. 1997] in order to solve long lag problems. Figure 3.6 illustrates the general structure of such a cell. As observed, the input of a LSTM cell consists of the concatenation of the output of the previous cell (in temporal order) and the current input. The cell is generally made up of three main parts: an input gate, a forget gate and an output gate. The input gate produces a term consisting of the product of the input passed through a layer of sigmoid functions and the input which has been squashed by a hyperbolic tangent function. The forget gate is slightly more complicated. First, it also processes the input with a series of

[3]LSTM tutorial

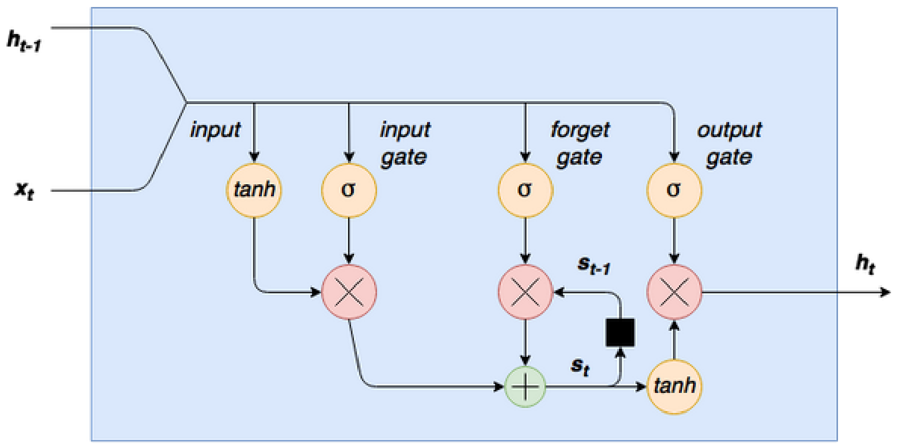


FIGURE 3.6: LSTM cell description [3]

sigmoid functions. But, then, it's multiplied by a variable called the internal state of the cell. This internal state corresponds to the sum of the result of the input and forget gates. This introduces the concept of memory into the cell. In the last stage of the cell, the internal state in squashed with another hyperbolic tangent, then multiplied by the result of the output gate. This gate once again passes the input through another layer of sigmoid functions. The result of this last product corresponds to the output of the cell. Let's notice that the activation functions corresponding to a hyperbolic tangent can be replaced by other types of activation functions.

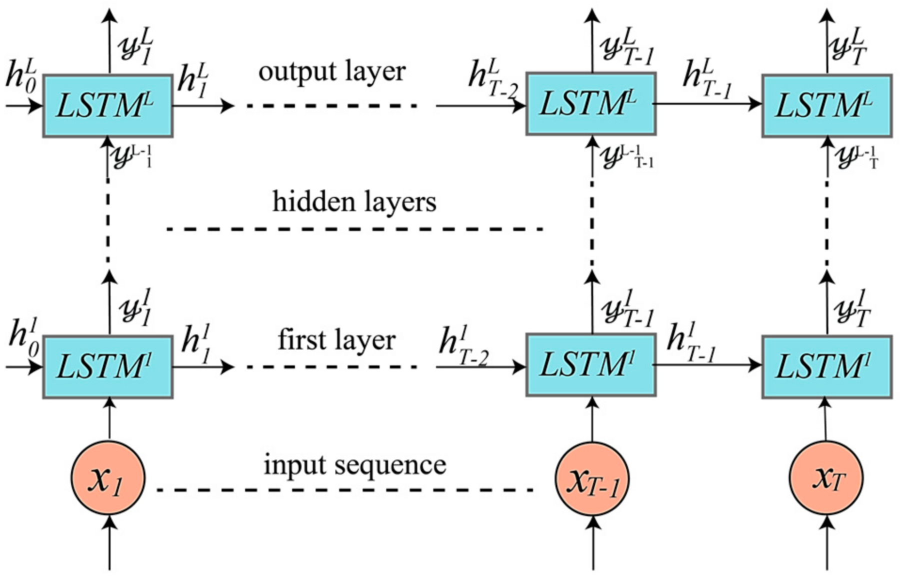


FIGURE 3.7: RNN architecture used in the master thesis [4]

After describing the main component of the network, let's switch to its global structure. Figure 3.7 represents the architecture of the recurrent neural network chosen as base estimator for the deep learning approach of this master thesis. It simply consists of adding LSTM layers sequentially, where an LSTM layer is composed of a given number of LSTM cells such as described by Figure 3.6. Afterwards, in order to produce only one input, the outputs of the last LSTM layer are sent through a Dense

[4][24, Abdulmajid et al. 2017]

layer of one node. The number of LSTM cells by layer will be a hyper-parameter of the network.

Before diving into the results of the experiments performed in Chapter 4, next section summarizes the hyper-parameters of which the influence will be analyzed.

3.7 Hyper-parameters

Before plunging into the analysis of the results, this section rapidly summaries all the hyper-parameters involved in Algorithm 1.

- Data preprocessing
  - Concatenation length l.
  - Type of supervised approach.
- First layer of the classifier
  - Type of base estimator.
- Second layer of the classifier
  - Type of calibration method.
- Extra trees
  - Number of trees in the forest (default: 10).
  - Fraction of the total number of the features to consider when looking for the best split k (default: square root number of features / number of features).
  - Maximum depth of a tree (default: no max depth).
  - Minimum number of samples to split n_{min} (default: 2).
  - Minimum number of samples per leaf n_{leaf} (default: 1).

hyper-parameter	Default value
l	500
Base estimator	Extra-trees
Calibration method	None
Number of trees	200
k	Square root of the number of features / number of features
Maximum depth	None
n_{min}	2
n_{leaf}	1
Bootstrap	False
c	1
f	Epsilon insensitive
l_{lstm}	2
n_{lstm}	100

TABLE 3.4: Default values of hyper-parameters in the scope of Algorithm 1

  - Whether to use bootstrap or not (default: no bootstrap).

- Linear SVM
  - Regularization parameter c (default: 1).
  - Loss function f.

- Recurrent Neural Network
  - Number of LSTM layers l_{lstm} (default: 2).
  - Number of LSTM units per layer n_{lstm} (default: 100).

Out of these hyper-parameters, several of them influence the supervised algorithm directly. These correspond to the hyper-parameters of Extra trees and linear SVM. Besides, the other hyper-parameters influence more the classification method itself than the algorithm used for classification. Such hyper-parameters, called structural parameters, have thus a deeper impact than classical ones.

Considering Extra trees, some of its parameters can be highlighted due to their particular role. Indeed, the maximum depth, n_{min} and n_{leaf} are three different ways of controlling the pruning of the trees. Let's simply notice that n_{min} is associated to internal nodes of the trees, which means it prevents an internal from splitting if the number of samples is below its value. By contrast, n_{leaf} does not prevent from splitting as it only influences the leaf nodes. However, it can cancel a split if the number of samples of one the produced leaf is under its value. As far as linear SVM is concerned, one can highlight the c parameter. This consists of a penalty term added to the error in order to perform regularization.

Table 3.4 show the default values of the hyper-parameters considered for the analysis. If not stated otherwise, these values have to be assumed for the experiments performed in the next chapter.

Chapter 4

Results

This chapter depicts the experiments conducted with Algorithm 1 and data configuration described in section 3.5.2. It's divided into two section, each corresponding to one of the approaches defined in Chapter 3. As a reminder, the first approach aims at using standard supervised learning datasets, which have been built by feature extraction. By contrast, the second approach directly uses time-series as input for recurrent neural networks. Let's also recall that, when no information about a parameter is given, its value from Table 3.4 has to be assumed. It's also good to remember that both cases when anomaly should be identified and when it should not, are going be tested, as explained in section 3.5.2. From now, they will respectively be denoted as positive and negative cases (with respect to the presence of an anomaly).

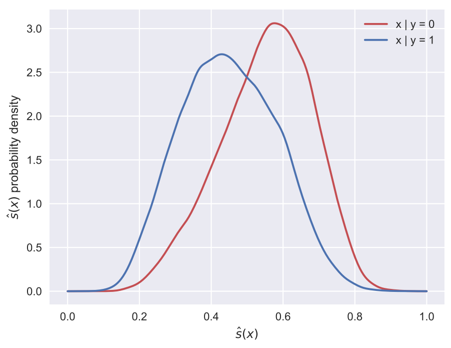
4.1 Standard supervised approach - Extra trees

This section details all the analyses that have been performed using the standard supervised approach and extremely randomized trees as base estimator.

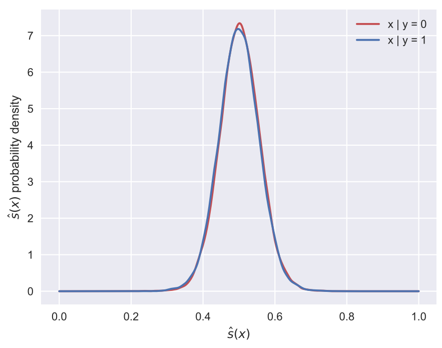
4.1.1 Experiment with default parameters

The first experiment simply consists in choosing one data configuration and performing a density ratio analysis with the default parameters. This aims as providing a first graphical visualization of the different outputs of the algorithm. Indeed, some of the metrics that are going to be displayed in this analysis are not usually used for solving machine learning problems. For this experiment, it's thus been chosen to analyze the data corresponding to slot game. As a reminder, the bug introduced in the corrupted part of the data is an increasing of the probability of the bonus symbol.

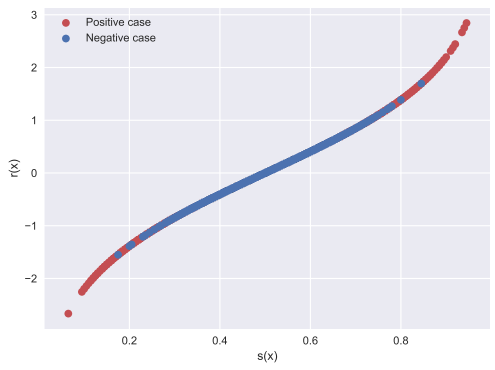
The metrics introduced at section 3.2.2 have thus been computed for this configuration. Their graphical representation is displayed in Figure 4.1. Let's depict them one by one. Figures 4.1a and 4.1b both show the density probability of the output of the classifier $\hat{s}(x)$, which is here the probability of belonging the class $y = 0$; They respectively correspond to situations when one of the class corresponds to data generated from the bugged version of the game and when both classes have been generated by the same version of the game. It's clear that densities of Figure 4.1a are much less similar than the ones of Figure 4.1b. While label curves are clearly distinguishable on the left graph, they almost overlay on each other on the right graph. Therefore, this provides first clues on the ability of the method to detect the anomaly on the one hand, and not to identify a false positive on other hand.



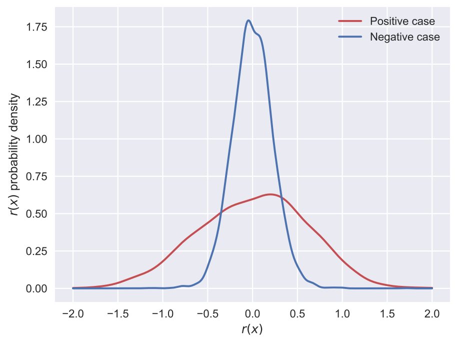
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



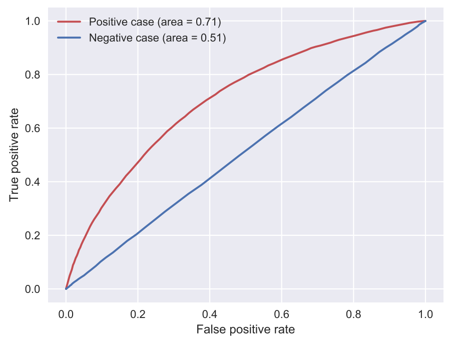
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio



(E) ROC curve

FIGURE 4.1: Density ratio analysis of slot game: experiment with default parameters

Then, let's analyze the density ratio metric itself for both correct and bugged games. Figure 4.1c plots it as a function of the classifier output $\hat{s}(x)$, while its probability density is displayed in Figure 4.1d. In both figures, the red curve (resp. blue curve) is related to probability densities of Figure 4.1a (resp. 4.1b). As expected, greater values of the ratio are more likely to occur when an anomaly should be detected. This is expressed by the fact that larger values are effectively obtained for test samples of Figure 4.1c and that the probability density is flattened in Figure 4.1d. Here, the density starts to be negligible from $|r(x)| > 1.5$. However, concerning the negative case, it's observed that the range of values for the density ratio is not as small as Figure 4.1b could suggest it. Indeed, its density starts to be negligible only from $|r(x)| > 0.75$. It should thus be interesting to see if some hyper-parameters are able to reduce this range, without reducing the one of the positive casen in order to cause even bigger differences between them.

Finally, by looking at the ROC curve in Figure 4.1e, it clearly appears that the classifier is not able to perform the binary classification correctly for the negative case. Indeed, the curve fits the 45° line very well, with an accuracy of 0.5. Meanwhile, a distinct increasing of the accuracy is observed for the positive case, as it goes up to 0.71. Intuitively, maybe an even higher rise could be expected but it's important to keep in mind that, even if there is an anomaly in the positive case, both data classes come from a same base game. From this first experiment, it can be assessed that the standard approach seems to already give good results. In particular, differences between the positive and negative case are already visually perceptible. For further experiments, it will be important to observe how the parameters influence the different metrics. In particular, identifying the ones reducing the range of the density ratio values for the negative case should be considered.

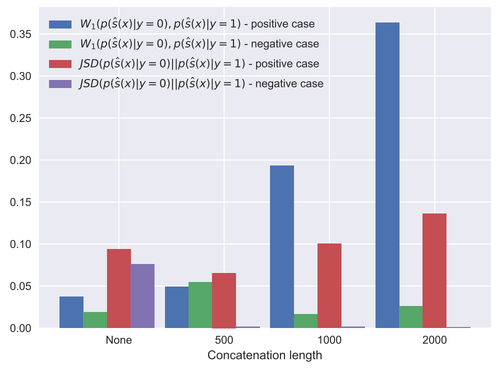
4.1.2 Influence of the concatenation length

In this section, the influence of the minimum size to concatenate the GIP data during the preprocessing stage is analyzed. The values that have been chosen for the experiments are the following: None (no concatenation), 500, 1000 and 2000. It's also been explained thanks to Table 3.3 that using concatenation reduced the size of the dataset. Therefore, it's been decided to use cross-validation techniques as detailed in the pseudo-code of algorithm 1. Table 4.1 details the chosen values for the number folds as a function of the concatenation size. The choice seems logical, as the greater the size, the smaller dataset, and thus, the safer it will be to use a larger number of folds. Note that, from this point, it's not possible anymore to display the data in a similar way as on Figure 4.1. Therefore, the numerical values associated to the metrics (described in section 3.2.2 will be plotted as a function of the varying parameter:

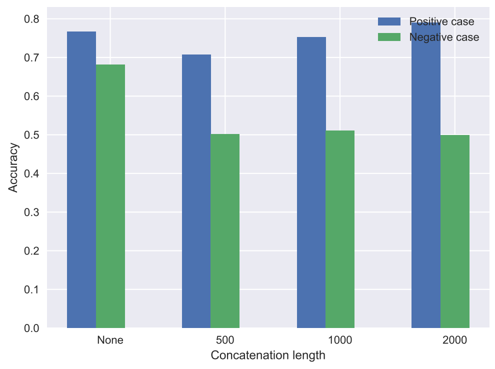
- Probability densities of $\hat{s}(x) \rightarrow$ Wasserstein distance W_1 and Jensen-Shannon divergence JSD.
- ROC curve $\rightarrow$ accuracy.
- $\hat{s}(x)$ vs $r(x) \rightarrow$ Variance of $r(x)$.
- Probability densities of $r(x) \rightarrow$ Wasserstein distance W_1 and Jensen-Shannon divergence JSD.

Concatenation length	Number of folds
None	1
500	10
1000	15
2000	20

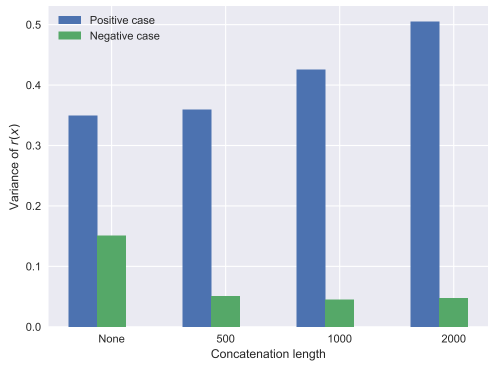
TABLE 4.1: Number of folds for stratified cross-validation according to the concatenation length



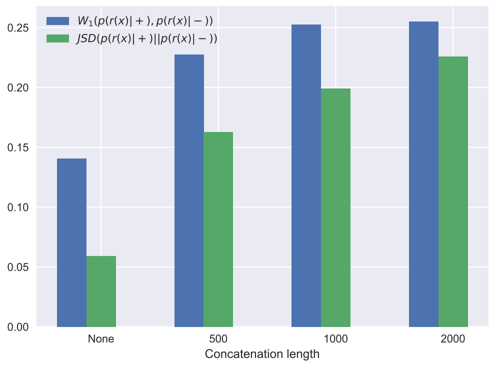
(A) W_1 and JSD for $p(\hat{s}(x))$



(B) Accuracy



(C) Variance of $r(x)$



(D) W_1 and JSD for $p(r(x))$

FIGURE 4.2: Metrics as a function of the concatenation length for slot game

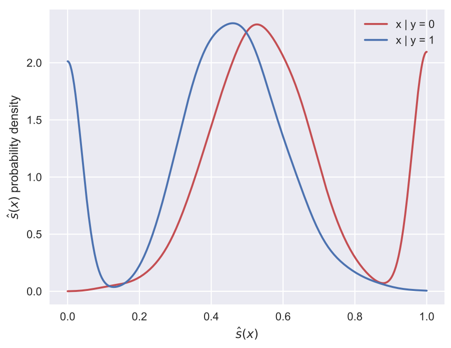
Slot game Figure 4.2 displays the corresponding for the slot game configuration. By looking at all figures, it's clearly noticed that something particular happens when no concatenation is performed. Indeed, the following elements can be highlighted:

- In Figure 4.2a, the JSD of the negative case is relatively greater than in the three other cases (where $JSD \simeq 0$) , when concatenation is performed. This is of course unexpected as the probability densities of which the JSD is computed are expected to be very similar is the negative case. Therefore, something particular must occur in this situation.

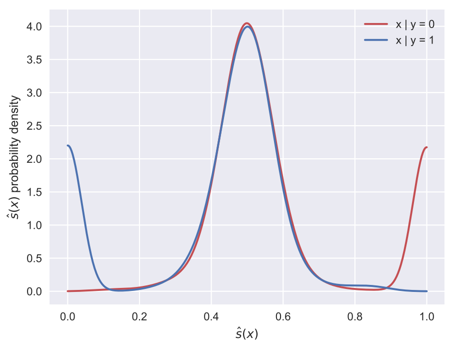
- The accuracy of the negative case in Figure 4.2b is quite high (almost 70%) while it stays around 50% in the other cases. Once again, this is not what should occur as, in the negative case, data of both classes come from the same game set, i.e. the same original probability density.

- In Figure 4.2c, again the negative case, the variance of the density ratio clearly distinguishes from the three other ones. It's around trice as great as in concatenation situation.

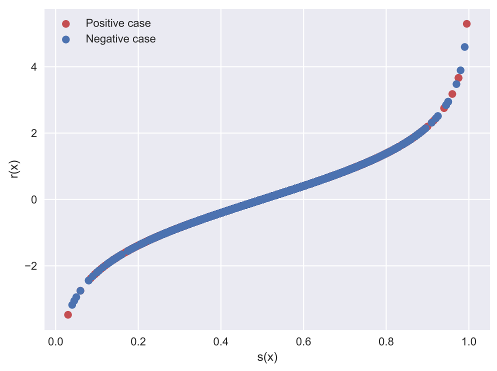
In addition, considering now only concatenation, there are some interesting remarks which can be done. Indeed, in the positive case, it seems that all metrics are monotonically growing functions of the concatenation length. That can be easily interpreted. As we know, in the positive case, each class contains time series generated from a distinct game set, i.e. probability density. Therefore, if these time series are



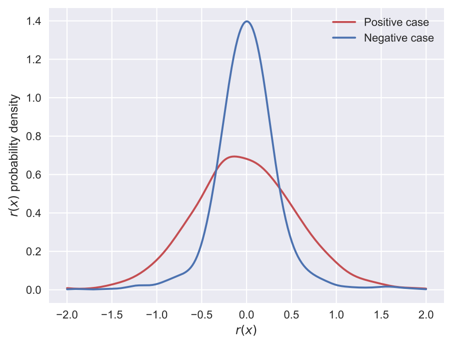
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



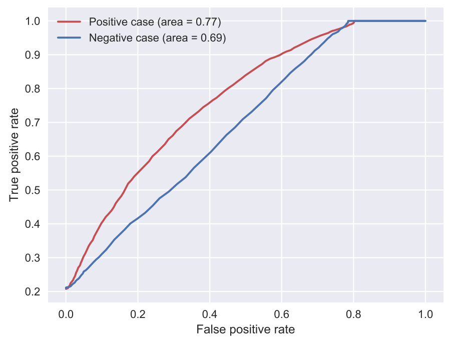
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio



(E) ROC curve

FIGURE 4.3: Density ratio analysis of slot game: l =None

combined with each other, the differences between the two probability densities will be enhanced and the classes will be distinguished more easily by the classifier. Another consequence of this is Figure 4.2d. Indeed, it's clear that increasing the length also causes the W_1 and JSD between the two probability distributions of $r(x)$ to rise.

Let's now try to discover why the case where no concatenation is performed seems to lead to some issues. To do so, another experiment is performed and the information is displayed in the same way as in Figure 4.1. As seen in Figure 4.3, the reason of issues appears immediately. Something very particular is observed in both Figures 4.3a and 4.3b: several samples are classified with a very high probability. Even though this seems strange at first sight, the reason of such a behaviour is obvious when knowing the data generation and preprocessing procedures. Indeed, remember that when a GIP such as in Figure 3.2 is produced during a simulation, its size is drawn from a uniform distribution $\mathcal{U}[1, 1000]$. Therefore, when there is no concatenation, it's possible that the procedure would extract features from very small time-series. Hence, features of such small time-series would be distorted and corresponding samples should easily identifiable by the classifier. The dataset is thus corrupted by such samples. Indirect consequences of this can also be observed in Figures 4.3c and 4.3e where the positive and negative cases are much more difficult to distinguish than in Figure 4.1.

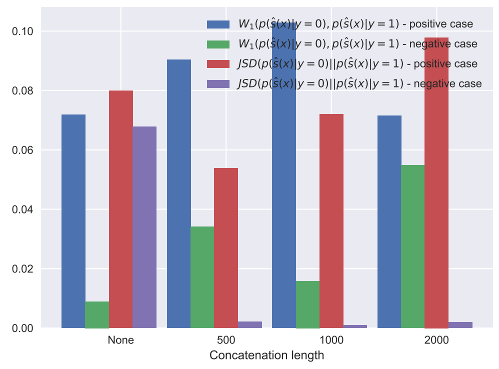
For this game configuration, the results clearly speak in favour of concatenation. However, this has to be considered carefully at the moment for two reasons:

- We don't know if this analysis is valid for other game configurations yet. It could indeed be possible that concatenation could give inappropriate in other cases as, as it's been already, every case is a new machine learning problem.
- The more time-series are concatenated, the less samples are available in the final dataset. Therefore, even though it was concluded that concatenation is always efficient, it should still be performed with parsimony, in order to have enough samples for the analysis to be relevant. This is why no size greater than 2000 was considered to do concatenation, as this size already drastically reduces the number of samples in the dataset.

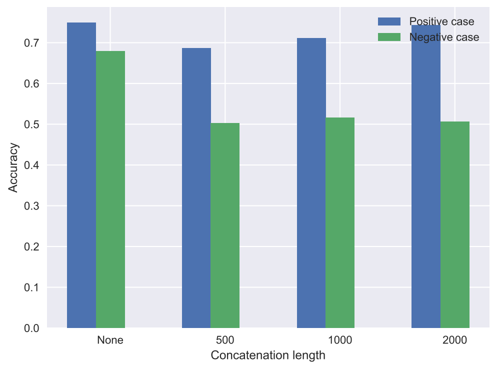
For the next experiments of this section, other game configurations are going to be considered.

Dice game Similar experiment has been conducted to evaluate the influence of the concatenation size on the algorithm for the dice game configuration. Figure 4.4 displays the corresponding results. First thing that can be noticed is, as for slot games, there is clear behaviour modification brought by concatenation and the metrics are still increasing functions of the length. On the other hand, when looking at Figure 4.4a, something new can be noticed. The behaviour of the W_1 metric seems very anarchic in both positive and negative cases, while the JSD remains consistent with expected results. This may point out that W_1 metric might not be suited very well to the densities that are measured in this Figure, compared to the JSD, or that the analysis of the divergence of these densities could not be that relevant. We should then be careful with this quantity in further analyses. Lastly, in terms of pure numerical results, the values obtained for the slot game configuration seem to be slightly greater than for dice games. The main reason for this could be simply the smaller influence of the bug inserted in the dice game on the probability density of the game set than for slot game. Intuitively, it could be thought than increasing the probability of the bonus in a slot game would have stronger and more regular effects than a bug exploited only during specific periods of the game.

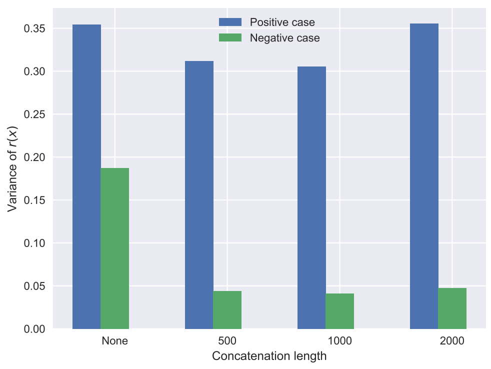
Roulette game Concerning now roulette game, the results of the corresponding experiment are shown at Figure 4.5. In this situation, the deviant behaviour of the algorithm without concatenation is really enhanced. Particularly, in Figure 4.5a, there is a huge relative gap between the JSD values of the first case compare to the 3 other ones. Moreover, the same remarks can be made about the monotony of the metrics



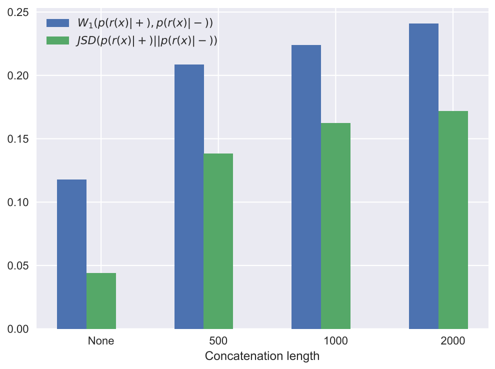
(A) W_1 and JSD for $p(\hat{s}(x))$



(B) Accuracy



(C) Variance of $r(x)$



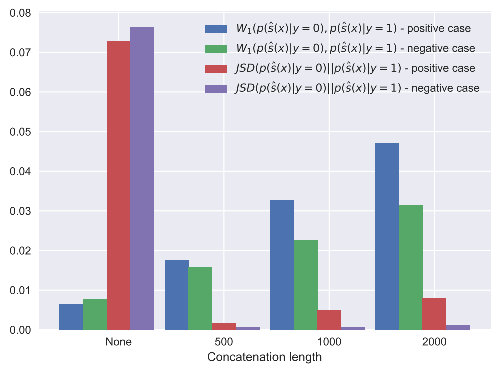
(D) W_1 and JSD for $p(r(x))$

FIGURE 4.4: Metrics as a function of the concatenation length for dice game

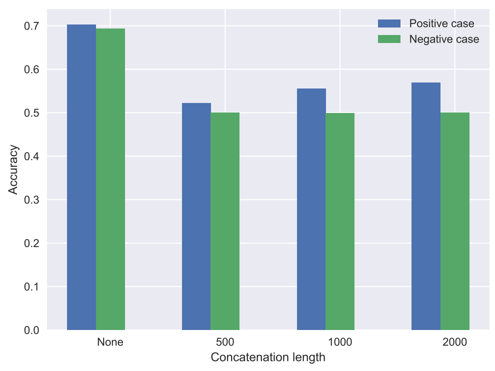
with respect to the concatenation length as for previous games. It also seems that the W_1 metric has this same trend for the negative case as well, although there is no reason for this. Once again, this tends to prove that the analysis of the metrics such as in Figure 4.5a has to be considered carefully. Finally, maybe the most interesting thing can be first noticed thanks to Figure 4.5b. On this one, it's observed that in the positive case, the accuracy is only slightly higher than 50% and doesn't even reach 60% when the concatenation length equals 2000. And indeed, when the numerical values of the other metrics are analyzed for other Figures, it's observed that they are also much smaller than for previous configurations.

This is a typical example of situation where the classifier is not able to detect a bug a very well. Although, not each parameter has been analyzed yet and it may possible to find parameters combinations allowing to improve the performances. However, let's also remember that the bug considered for roulette game was to set the probability of 4 symbols to 0. It may be possible that, due to the large amount of symbols and possible bets for roulette, the effects of the bug are less easy to detect. In any way, this will be something to improve in further analysis.

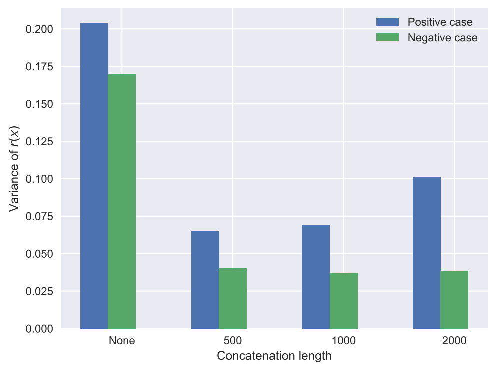
Blackjack game The last game configuration to analyze concerns blackjack. Figure 4.6 displays the corresponding results. This time, contrary to roulette, the effects of the bug are clearly visible. In Figure 4.6b and 4.6c, the numerical differences between the positive and negative cases are non negligible when concatenation is performed.



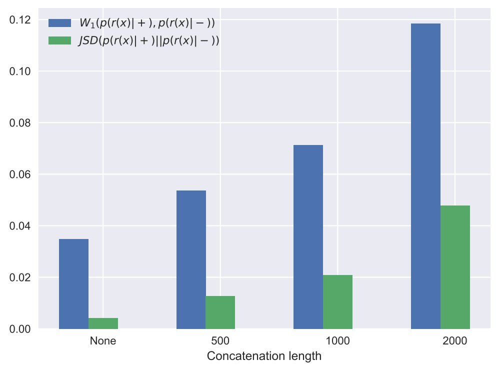
(A) W_1 and JSD for $p(\hat{s}(x))$



(B) Accuracy



(C) Variance of $r(x)$



(D) W_1 and JSD for $p(r(x))$

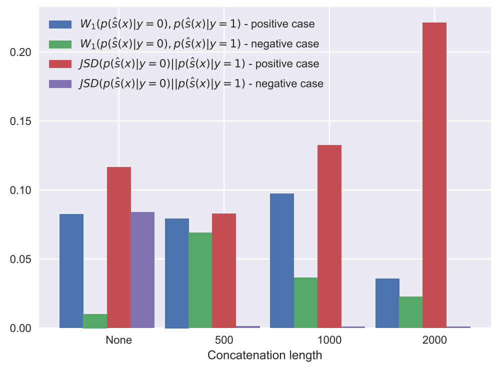
FIGURE 4.5: Metrics as a function of the concatenation length for roulette game

While the accuracy rises from 75 to more than 80%, the variance of the density ratio goes up from almost 1 to 3 in the positive case, even though it's not higher than 0.1 in the negative case. Moreover, in Figure 4.6d, it's notice that really high values are quickly achieved compared to roulette game as soon as concatenation is performed.

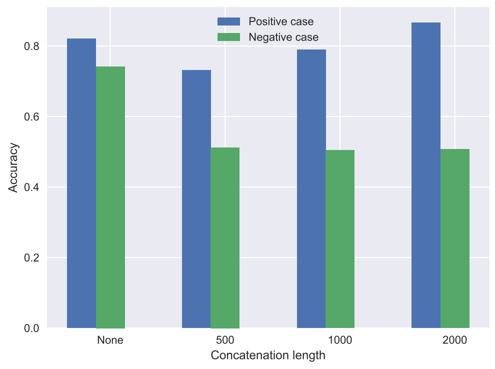
However, undesired trends still occur for the W_1 metric of Figure 4.6a, just like with dice game as we observe that this decreases from size 1000 to 2000. That means, for 2 situations out of 4, metrics of the $p(\hat{s}(x))$ densities are not consistent with other observations.

Summary The experiments conducted to analyze the influence of the concatenation length have been very useful and allowed to understand the following things:

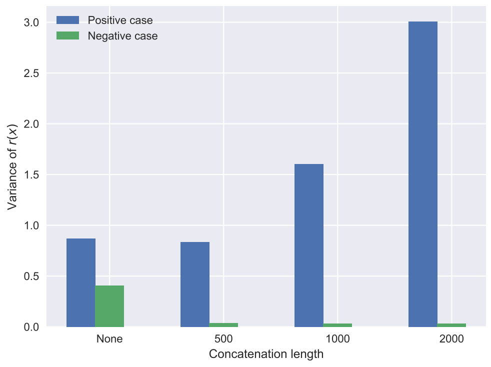
- Concatenation seems to be essential for the classification problem to be relevant. Indeed, it's been observed in all situations short GIP's with very few measures distorted the datasets. As illustrated by the example in Figures 4.3a and 4.3b, almost no uncertainty was associated to the samples corresponding to the short GIP's, no matter the case (positive or negative).
- Increasing the concatenation length improves the ability of the algorithm to detect anomalies (in the positive case). In the meantime, it seems that it has no undesired effect on the negative case, as the densities metrics remain really



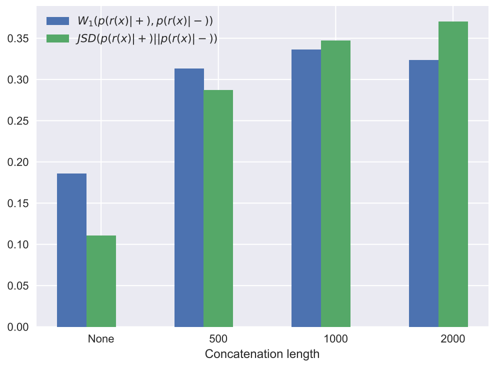
(A) W_1 and JSD for $p(\hat{s}(x))$



(B) Accuracy



(C) Variance of $r(x)$



(D) W_1 and JSD for $p(r(x))$

FIGURE 4.6: Metrics as a function of the concatenation length for blackjack game

low and the accuracy stays around 50%. However, it's important not to use a too large size , as it reduces a lot the number of samples in the dataset.

- In half of the game configurations, inconsistent behaviour is observed for the metrics related to the probability densities of $\hat{s}(x)$. Therefore, for the analysis of the influence of other parameters, these metrics won't be used anymore.

- For the roulette game configuration, the algorithm has more difficulties to dissociate the positive and negative cases. One the main concerns of the further sections is to see if it's possible to improve the performances of the algorithm for this particular case.

Next sections are going to cover the parameters related to the Extra trees classifier used in the algorithm, i.e. the bottom layer of the classifier.

4.1.3 Influence of the number of trees

This section covers the analysis of the influence of the number of trees in the forest on the chosen metrics. In most applications, increasing the number of trees often result in higher accuracy. Indeed, this tends to reduce the variability of the model due to overfitting, by taking into account a higher number of features in many different ways. Also, aggregating the predictions of several unique trees provide as much information as the number of trees grows. However, in our situation, there's a small

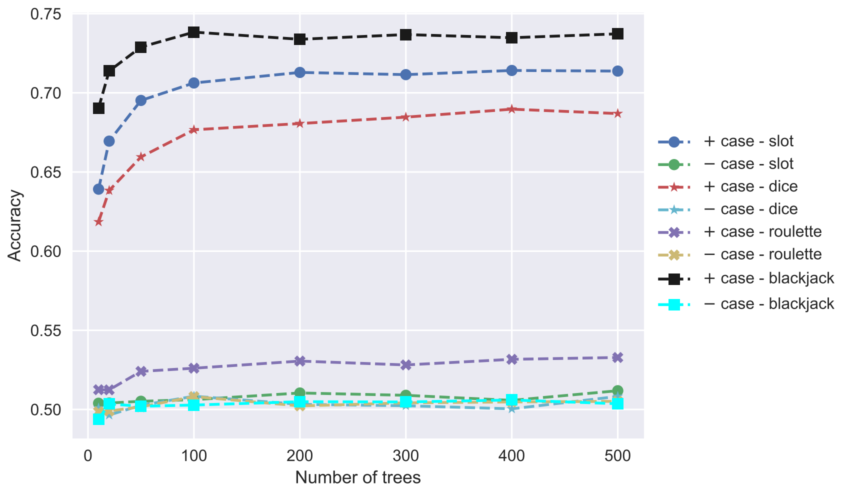


FIGURE 4.7: Accuracy as a function of the number of trees

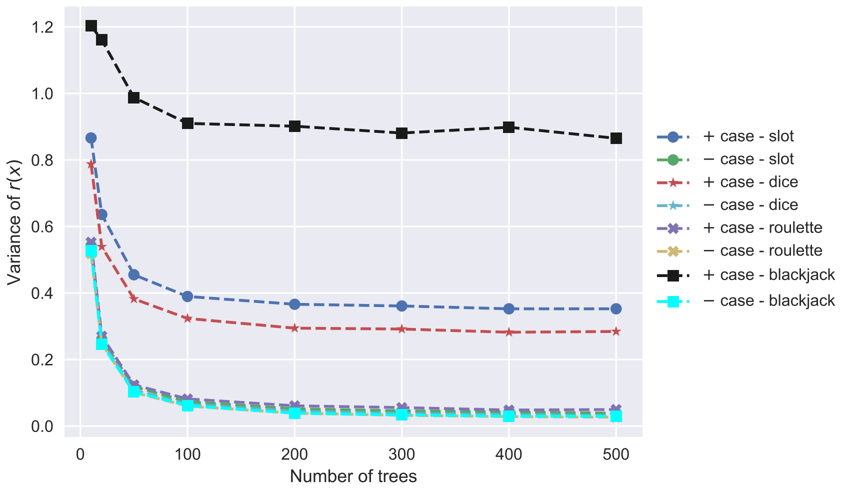


FIGURE 4.8: Variance of $r(x)$ as a function of the number of trees

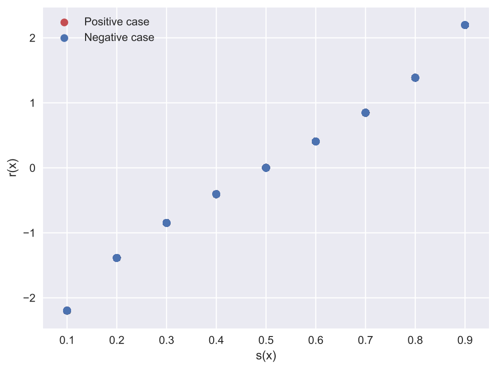
difference compared to classical classification problems. Although the accuracy is expected to be an increasing function of the number of trees in the negative case, it's also expected that this parameter won't affect this metric of in the negative case. Indeed, when two classes in a dataset are assumed to be non separable, parameter tuning should allow to differentiate them better.

For this experiment, it's been chosen to set the varying parameter to the following values: 10, 20, 50, 100, 200, 300, 400 and 500. Considering other parameters, the values of Table 3.4 will be used once again. The accuracy as a function of the number of trees is shown in Figure 4.7. First, as expected, the accuracy is globally increasing in the positive when the size of the forest grows too. For slot, dice and blackjack games, the gain is non negligible as the difference between the lowest and highest values of the accuracy is approximately 10%. However, for roulette game, even with the highest value of the parameter, the accuracy does not go over 55%, which is insufficient to be differentiated from the negative case. As far the negative case is concerned for the other types of games, even though small oscillations are observed, the accuracy remains at sufficiently low values, below 55%.

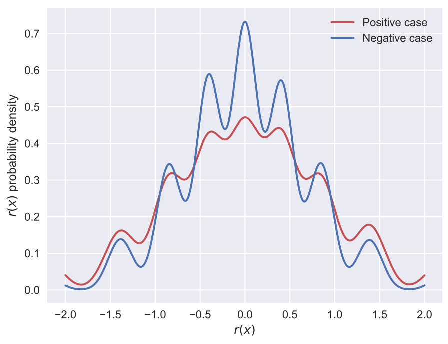
Let's now observe the influence on the metrics related to the density ratio. Figure 4.8 displays the evolution of the variance of $r(x)$ according to the number of trees. Here, several particular things occur and can be noticed:

- In every situation, even the positive case, the variance decreases. Intuitively, this is unexpected as, if the accuracy increases, the log density ratio between the probabilities of the two classes should take higher values in the positive case.
- Considering the negative case only, unexpectedly high values of the variance are observed for small numbers of the trees. This should be linked with the decreasing of the variance in the positive case. The reason why such high values are not expected is exactly the same as for the first remark but in the opposite situation: if the accuracy is close to 50%, small absolute values should be obtained for the log density ratio as the probability of the two classes are close to each other. However, this is not the case for small forests.

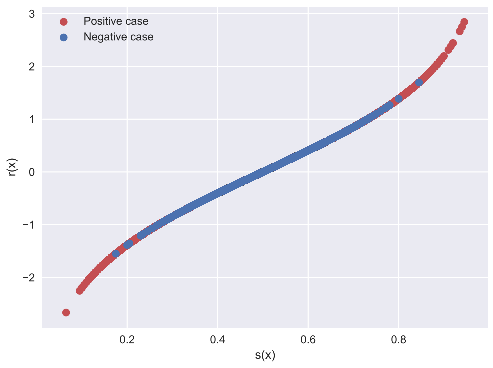
In order to try the explain the trends observed in Figure 4.8, the graphical metrics of the log density ratio will be compared for the lowest and highest values of the number of trees thanks to a new experiment. The results of this new experiment are



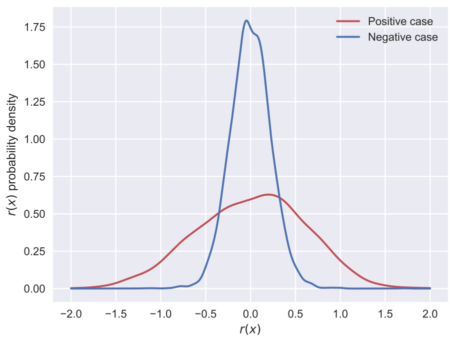
(A) Density ratio as a function of $\hat{s}(x)$ given the true label - Number of trees = 10



(B) Probability density function of the density ratio - Number of trees = 10



(C) Density ratio as a function of $\hat{s}(x)$ given the true label - Number of trees = 500



(D) Probability density function of the density ratio - Number of trees = 500

FIGURE 4.9: Comparison of the density ratio metrics for low and high number of trees

shown in Figure 4.9. This immediately allows to understand the values obtained for the variance of $r(x)$. Indeed, it's observed in Figure 4.9a that having a small forests induces kind of discrete values for the density ratio, for both positive and negative cases. This causes the floor-shaped probability densities in Figure 4.9b. The contrast with Figures 4.9c and 4.9d, where the larger forest allows much smoother curves, is therefore very important. This means that having a too small number of trees does not provide a fully relevant analysis of the density ratio, even though the accuracy seems to have consistent values. Therefore, it's really important to understand that the decreasing of the variance is due to an increasing of the range of possible values of the ratio, which is directly caused by the growing size of the forest. Also, the distinction between the positive and negative cases can still be done by looking at the relative decreasing of the variance. In the negative case, it reaches around 90%, while only 50-60% at most for the positive one (except for roulette game).

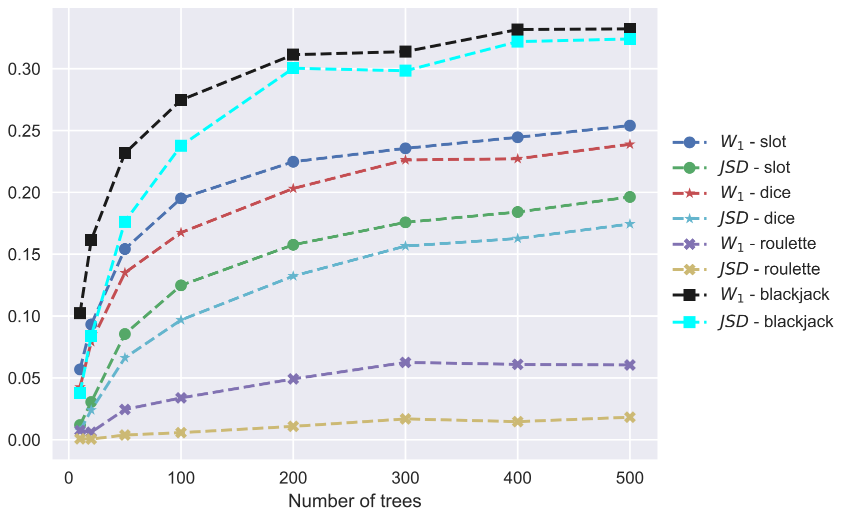


FIGURE 4.10: W_1 and JSD for $p(r(x))$ as a function of the number of trees

The last metrics to analyze is the JSD and W_1 distance between the probability densities of the log density ratio. These are displayed in Figure 4.10 as a function of the number of trees. This time, nothing unexpected happens as each metric is an increasing function of the number of trees and has more or less the same trend as for the accuracy in Figure 4.7. With this Figure, it's also really easy to classify the bugs according to the scale of their effects. The blackjack is clearly the game which has been affected the most by its bug, while the slot is slightly more affected than the dice game. As forecast by previous Figures, the bug of the roulette game stays very hard to identify as shown by the very small differences between the density ratio densities of the positive and negative cases expressed by low values of the metrics.

This possibility of classification also allows us to enhance a particular feature of this class of problems. The values observed for the metrics from a case to another can be very different. Therefore, it seems really difficult to establish standard values from which it could be said that the behaviour has indeed been modified. Hence, this confirms that the validation procedure defined in section 3.5.2 will be necessary even in further applications of this method. Indeed, in realistic situations, we don't know whether a game has been modified. Therefore, the comparison of a case for which the result is already known but for which the metrics will take values that are specific to this game, is indispensable. In next section, the influence of the size of the subset of features will be analyzed.

4.1.4 Influence of the number of features to split

This section addresses now the now one of the most important hyper-parameters of Extra trees algorithm, the size of the features subspace used for each split. As said in section 2.1.2, the main particularities of Extra trees come from the way splitting criteria are chosen. Concerning the range of values, it's been set to the following: 0.05, 0.1, 0.2, 0.3, 0.5, 0.75, 1.

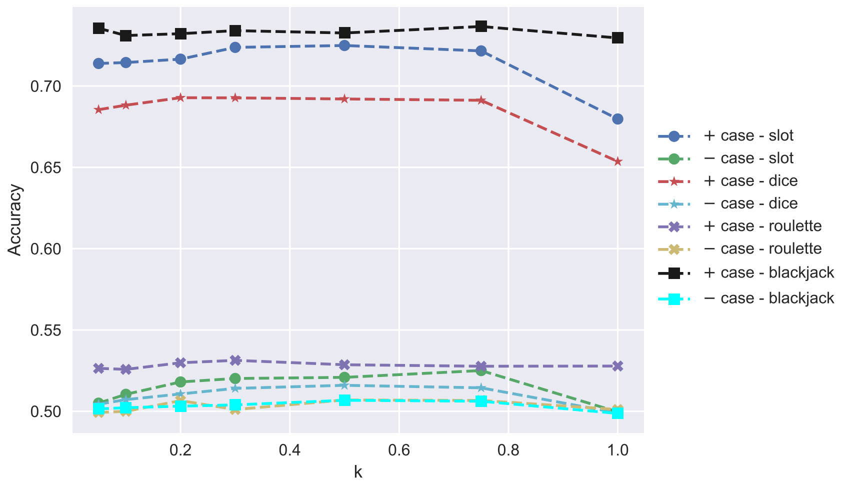


FIGURE 4.11: Accuracy as a function of the subspace of features

First, Figure 4.11 displays the how the accuracy is affected by the number of features. For $k < 1$, it seems that the influence of the parameter is relatively small. However, when the whole features space is used to determine the splitting criterion during the growing of the trees, a drop of the accuracy is observed in many situations. Particularly, the positive cases of the slot and dice games seem to be the most affected. Nonetheless, this graph shows once again that every problem is different and that the influence of every parameter cannot be assumed to be the same in every situation. Nevertheless, it clearly seems that the whole features space should generally not be used with Extra Trees, which is logical. Indeed, the diversity of the forest induced by k parameter depends also on the number of feature that are considered to choose a split.

Then, let's now see how k influences the variance of the density ratio $r(x)$. This information is shown in Figure 4.12. The correlation between this metric and the accuracy is still enhanced here as the same drop is observed for the maximum value of k. However, this time, the drop is even sharper. Additionally to this drop, another trend, which wasn't observed in previous Figure, can be highlighted in this graph. Indeed, especially for most of positive cases, the variance of the density ratio seems to be an increasing function of the k for all the selected values of k which were smaller than 1. For most situations, the best value of k among the range of tested values is thus 0.75.

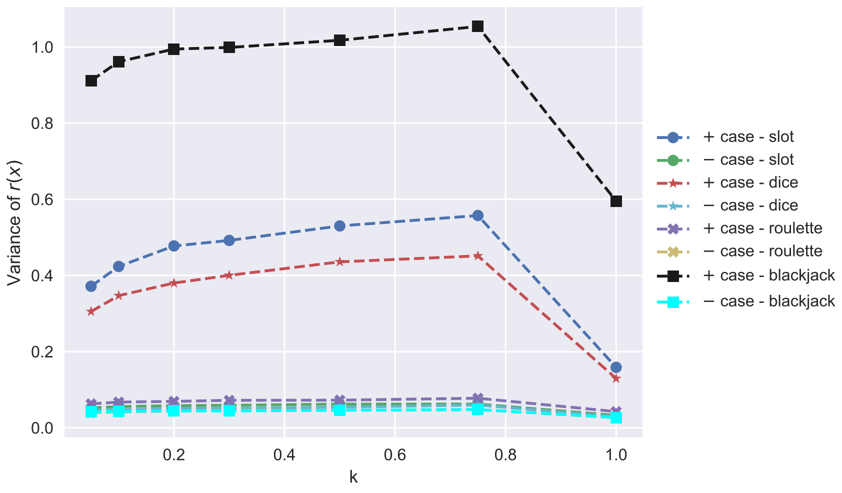


FIGURE 4.12: Variance of $r(x)$ as a function of the subspace of features

Finally, Figures 4.13 illustrates the evolution of the distance metrics between the probability densities of the ratio density for the positive and negative cases. For these metrics, the influence looks to be a bit harder to discern. Indeed, the gap between the $k = 1$ and other values of the subspace size is still visible. However, surprisingly, for the blackjack game, the distances between the densities increase, which doesn't follow at all the trend of the variance of the density ratio. However, this could be partially explained by what we have seen in Figure 4.11. Indeed, in this graph, the accuracy gap between the positive and negative cases for the blackjack game actually increases, contrary to other configurations. Nevertheless, it still seems that 0.75 is globally the best value for k.

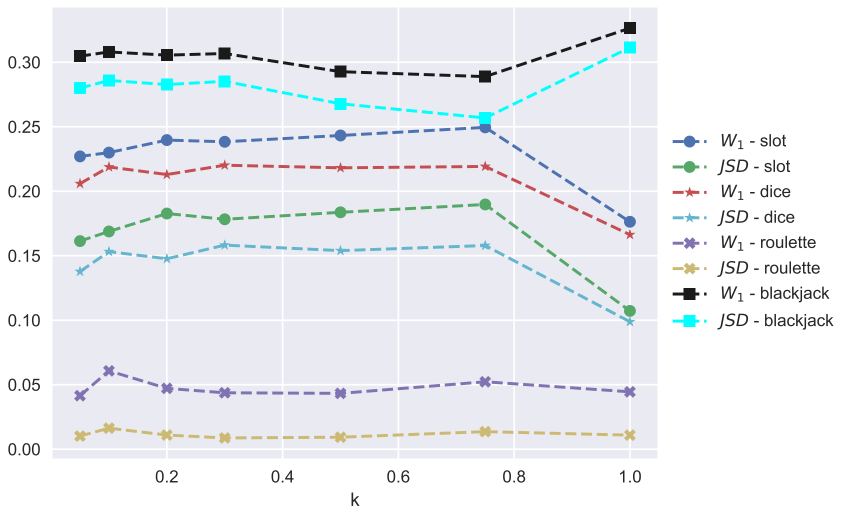


FIGURE 4.13: W_1 and JSD for $p(r(x))$ as a function of the subspace of features

4.1.5 Influence of the maximum depth

In this section, the influence of the maximum depth hyper-parameter is addressed. Considering the chosen values for this parameter, the range has been set to the following: 10, 20, 40, 60, 80, 100, 120, 140. Once again, other hyper-parameters values have been set according to Table 3.4 First, Figure 4.14 displays the evolution of the accuracy according to the maximum depth of the trees. It's observed that this metric remains mostly uninfluenced. Nonetheless, when $max\_depth = 10$, the accuracy seems to be slightly perturbed. In particular, the most remarkable influence concern the positive case curve of the dice game, which drops a little. The analysis of other metrics should help to confirm or not this trend.

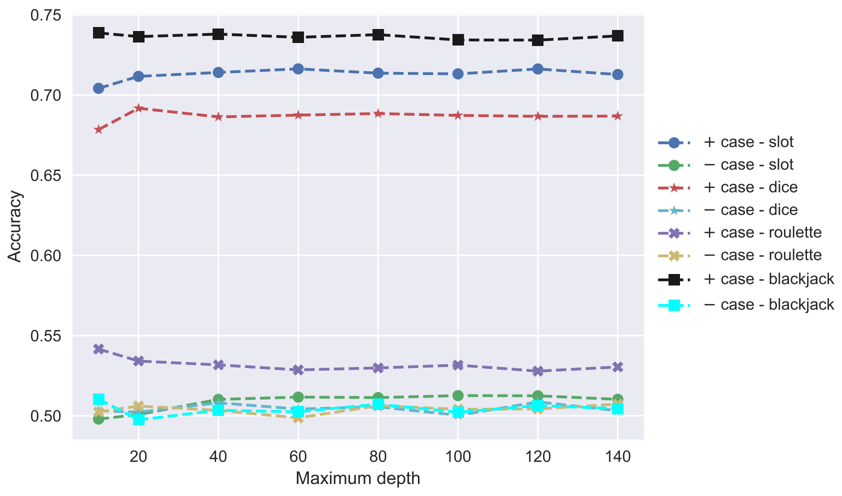


FIGURE 4.14: Accuracy as a function of the maximum depth

Let's now see how the variance of the density ratio behaves as a function of the maximum depth. The results associated to this metric are shown in Figure 4.15. The signs observed for the accuracy are clearly confirmed in this Figure. Particularly, we notice a sharp drop of the variance for the positive case of the slot and dice games. Considering other cases, the decreasing is a bit less pronounced but still visible. Therefore, it can be asserted that smaller maximum depths values lead to a more reduced range of density ratio values. Now, just as for the number of trees, an extra experiment will be performed comparing the graphs associated to the density ratio between a low and a high value of the parameter.

The additional experiment has been conducted comparing $max\_depth = 10$ to $max\_depth =$ None, which theoretically corresponds to an infinite maximum depth. Figure 4.16 displays the associated results. The expectations are confirmed, as in the case of a lower maximum depth, there is significant reduction of the range of values for the variance of the density ratio in both positive and negative cases. This influence is relatively simple to explain. Indeed, setting the maximum depth of a tree to a small value reduces the possibilities of splitting. Therefore, one the one hand, the leaf nodes will tend to differ less from a tree to another. And on the other hand, the impurity of these nodes will remain higher, preventing therefore the samples of the two classes to be more distinctively separated. The reduction of the range of values for the density ratio is thus a logical consequence of smaller depths of the trees.

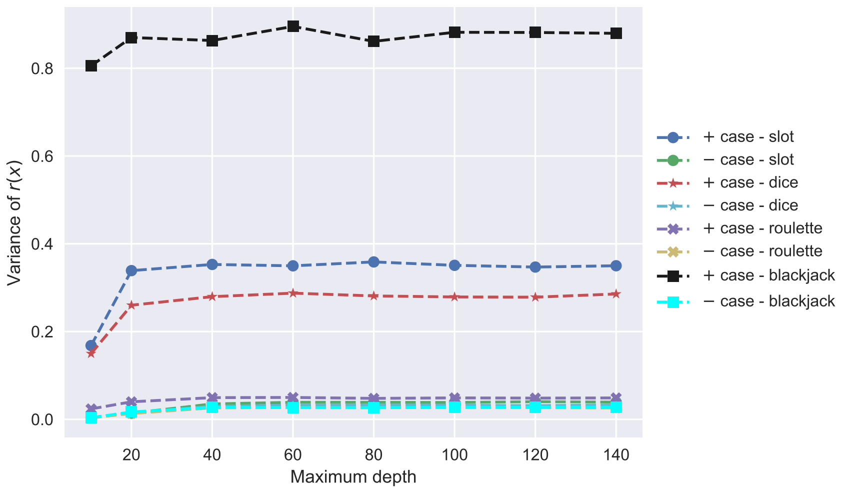
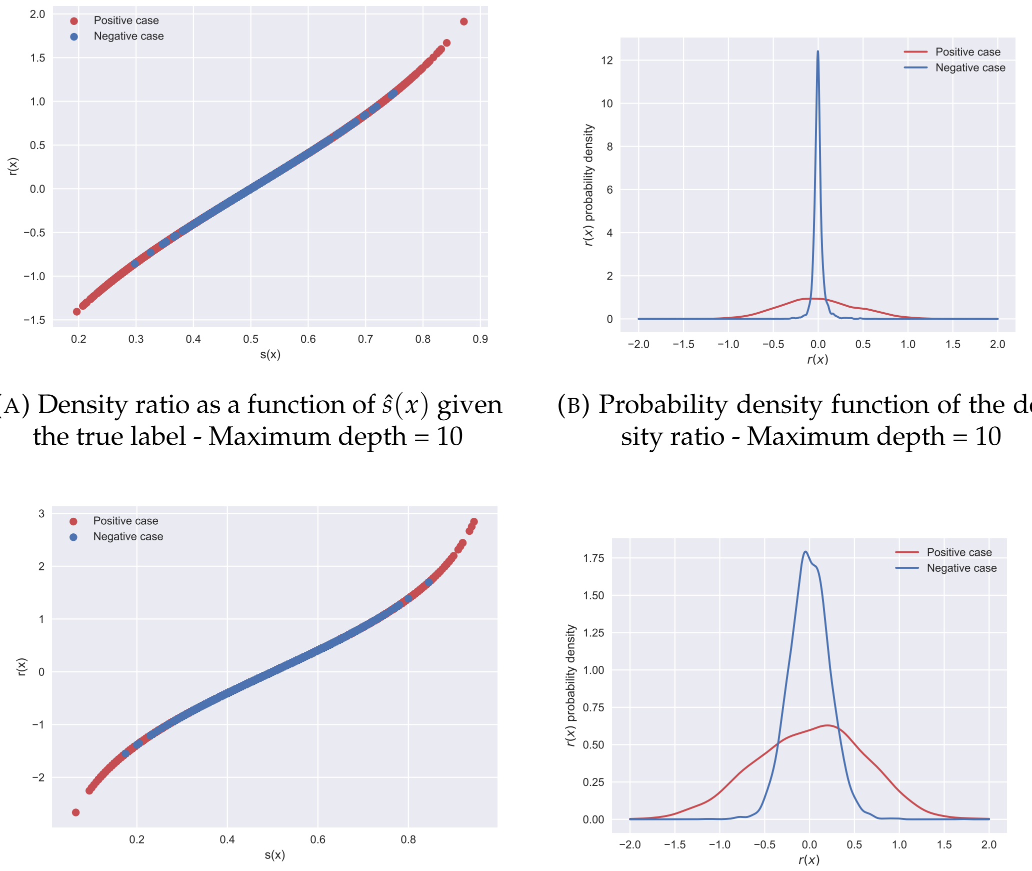


FIGURE 4.15: Variance of $r(x)$ as a function of the maximum depth



(A) Density ratio as a function of $\hat{s}(x)$ given the true label - Maximum depth = 10

(B) Probability density function of the density ratio - Maximum depth = 10

(C) Density ratio as a function of $\hat{s}(x)$ given the true label - Maximum depth = None

(D) Probability density function of the density ratio - Maximum depth = None

FIGURE 4.16: Comparison of the density ratio metrics with and without maximum depth

One last graph has to be depicted as far as the analysis of the maximum depth is concerned. Figure 4.17 illustrates its influence on the distances between the probability densities of the density ratio. Once again, smaller values of the parameter seem to influence the most the metrics. However, this time, it causes an increasing of these metrics rather than a decreasing like in the case of the variance. However, this can be easily explained by observing the experiment performed in Figure 4.16. Indeed, it's obvious that the fact the probability density associated to the negative case becomes so tight, is the main reason of the increasing of the distances metrics between the probability densities.

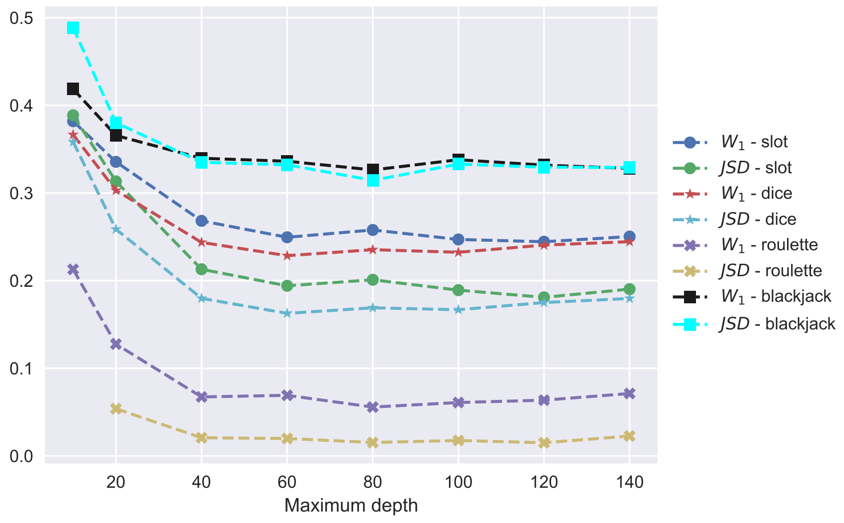


FIGURE 4.17: W_1 and JSD for $p(r(x))$ as a function of the maximum depth

To summarize the analysis of the influence of this parameter, a clear effect has been observed for its smaller values. However, it's been concluded to this influence did not help to enhance the differences between the positive and the negative cases of the different configurations. Therefore, the value of this hyper-parameter should be kept to its default value indicated in Table 3.4.

4.1.6 Influence of bootstrapping

This section tells about the influence of using bootstrapping during the growing of trees. As a reminder, this technique and its effects are summarized in section 2.1.2. All the results associated to the experiments are available in Tables 4.2, 4.3, 4.5 and 4.4. It's noticed that the trends are relatively similar for any game configuration. On the one hand, the accuracy and the distance metrics of the densities are not really influenced by the value of this hyper-parameter. The default value of bootstrapping allows here to get higher variances of the density ratio for the positive case. The metric also increases for the negative case, but much more slightly. Therefore, the default value of bootstrapping performs better and enhances the differences between the two classes for the classification. Changing the value of this parameter is thus not a solution to improve the results of the method.

Bootstrapping	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
True	+	0.7071	0.3156	0.1769	0.2359
	-	0.5074	0.0410		
False	+	0.7105	0.3698	0.1673	0.2320
	-	0.5083	0.0526		

TABLE 4.2: Influence of bootstrapping: slot game

Bootstrapping	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
True	+	0.6807	0.2565	0.1441	0.2096
	-	0.5031	0.0354		
False	+	0.6799	0.3030	0.1366	0.2088
	-	0.4997	0.0441		

TABLE 4.3: Influence of bootstrapping: dice game

Bootstrapping	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
True	+	0.5276	0.0510	0.0150	0.0588
	-	0.4985	0.0320		
False	+	0.5255	0.0623	0.0102	0.0442
	-	0.5019	0.0388		

TABLE 4.4: Influence of bootstrapping: roulette game

Bootstrapping	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
True	+	0.7312	0.8443	0.2967	0.3177
	-	0.5003	0.0345		
False	+	0.7329	0.8896	0.2967	0.3016
	-	0.5010	0.0396		

TABLE 4.5: Influence of bootstrapping: blackjack game

4.1.7 Influence of the calibration method

This section addresses now the influence of calibration. As a reminder, calibration has been explained in section 2.2.3. In Figure 3.3 of Chapter 3, this also corresponds to the second layer of the classifier. For the next experiments, the following calibration methods were compared to the default method without calibration: *Histogram*, *Isotonic*, *Interpolated-isotonic* and *Sigmoid*. Just like for the analysis of bootstrapping, the different results are stored in Tables according to the type of game. These are Tables 4.6, 4.7, 4.8 and 4.9.

Concerning first the accuracy, we note a moderated impact of the parameter. Nonetheless, it's possible to distinguish a method that works the best according to this metric. For instance, the sigmoid gives the most satisfying results for the slot and dice games, respectively corresponding to Tables 4.6 and 4.7. Indeed, it gets the highest accuracy in the positive while keeping it the closest to 0.5 in the negative case.

Although, accuracy is not the metric that is the most influenced by calibration. Indeed, by looking at the second metric, the variance of the density, we observe a much more important difference between the default and specific results. To illustrate this, let's for instance analyze the values of obtained for the blackjack, in Table 4.8. In particular, let's have a look at the ratio of the variance of the positive case over the variance of the negative case. Without calibration, we have $\frac{Var[\hat{r}(x)|+]}{Var[\hat{r}(x)|-]} \approx 10$. By contrast, for all calibration methods, except interpolation, we observe $\frac{Var[\hat{r}(x)|+]}{Var[\hat{r}(x)|-]} > 10^2$. This is a major improvement of the distinction between the positive and the negative case. Same remarks can be made about the first two games, slot and dice. However, roulette is still a problem. Indeed, as long as accuracy or variance are concerned, there is no way to observe a large enough distinction between both cases.

The solution to the problem of roulette might come thanks to the last metrics. Let's have a look at the last columns of Table 4.9. Particularly, it seems that the JSD metric has some issues with this particular configuration, as it's value tends to infinity for some calibrations methods. There is thus a reliability problem with this measure in this case. Nevertheless, the W_1 metric still gives correct results. Moreover, something very interesting can be noticed for sigmoid calibration. The W_1 distance associated to this parameter value is much higher than for other calibration methods, despite the relatively small differences between positive and negative cases for the first metrics. Of course, it remains smaller than the values obtained for other games using similar calibration, but this might be the solution required to detect bugs with slighter impacts like the one inserted in the roulette game.

To sum up this section, calibration indubitably reinforces Algorithm 1. This may be done by intensifying already discovered problems such as for the slot, dice and blackjack configurations thanks the variance metrics. But this also allows to enhance bugs which are more difficult to detect such as for the roulette game, thanks the W_1 between the two density ratio probability densities.

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.7152	0.3650	0.1590	0.2249
	-	0.5116	0.0516		
Histogram	+	0.7171	0.5445	0.5082	0.4314
	-	0.5040	0.0024		
Isotonic	+	0.7154	0.5575	0.5771	0.4636
	-	0.5004	0.0010		
Interpolated	+	0.7178	0.7426	0.1207	0.1960
	-	0.5075	0.1440		
Sigmoid	+	0.7185	0.5871	0.5861	0.4623
	-	0.5020	0.0003		

TABLE 4.6: Influence of the calibration method: slot game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.6816	0.3020	0.1427	0.2122
	-	0.5064	0.0430		
Histogram	+	0.6894	0.4411	0.4791	0.4212
	-	0.5004	0.0021		
Isotonic	+	0.6854	0.4698	0.5145	0.4396
	-	0.5015	0.0022		
Interpolated	+	0.6859	0.5935	0.0826	0.1577
	-	0.5076	0.1218		
Sigmoid	+	0.6907	0.4271	0.5904	0.4663
	-	0.5008	0.0002		

TABLE 4.7: Influence of the calibration method: dice game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.7303	0.9054	0.2864	0.3091
	-	0.4980	0.0402		
Histogram	+	0.7411	0.8689	0.5356	0.4360
	-	0.4970	0.0021		
Isotonic	+	0.7324	0.8960	0.5767	0.4574
	-	0.5016	0.0012		
Interpolated	+	0.7399	1.0734	0.1663	0.2316
	-	0.5092	0.1222		
Sigmoid	+	0.7371	0.8305	0.6083	0.4700
	-	0.5013	0.0003		

TABLE 4.8: Influence of the calibration method: blackjack game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.5276	0.0618	0.0139	0.0590
	-	0.5009	0.0385		
Histogram	+	0.5263	0.0076	∞	0.1540
	-	0.5034	0.0018		
Isotonic	+	0.5283	0.0040	∞	0.0761
	-	0.5002	0.0013		
Interpolated	+	0.5304	0.0814	0.0318	0.1076
	-	0.5025	0.1405		
Sigmoid	+	0.5352	0.0062	∞	0.3312
	-	0.4970	0.0001		

TABLE 4.9: Influence of the calibration method: roulette game

4.1.8 Metrics as a function of the dataset size

After analyzing the influence of the different hyper-parameters of the Extra trees algorithm, this section will address the size of dataset. Indeed, it's important to see how the metrics behave according to this variable. On the one hand, it allows to obtain a lower bound on the amount of required to get satisfying results. On the other hand, this can also provide an upper limit to the metrics we're analyzing and therefore to how much these can enhance anomalies.

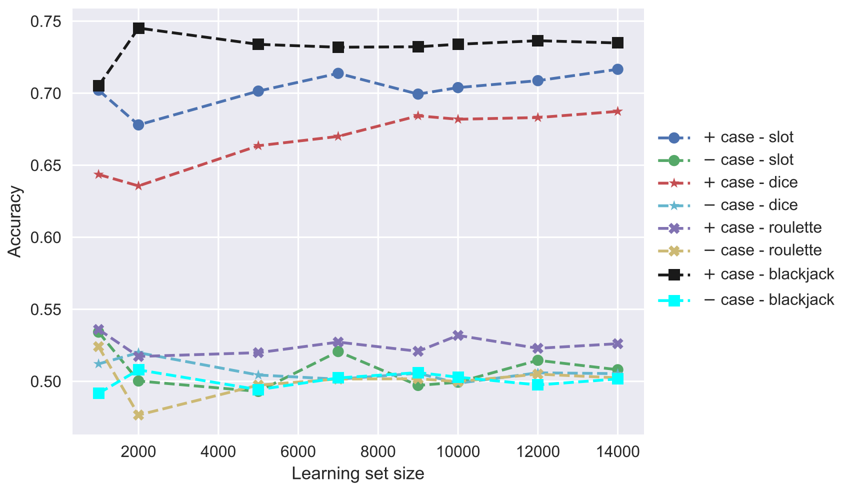


FIGURE 4.18: Accuracy as a function of the learning set size

Let's start with the accuracy graph. This is available in Figure 4.18. Firstly, considering most of positive cases, it's observed that, even though the curves are indeed increasing with respect to the dataset size, it reaches acceptable values relatively quickly. For instance, in dice game, no more than a few thousands samples are required to get performances which are equivalent to the ones of the whole dataset. Secondly, as far as negative cases are concerned, we observe a progressive stabilization around 0.5, which is the desired behavior. This time, for all negatives cases, the whole dataset is required to get a strong stabilization.

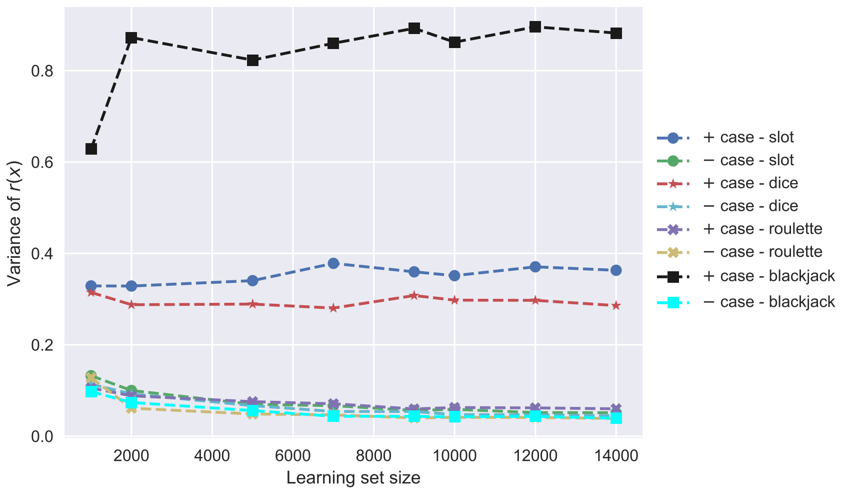


FIGURE 4.19: Variance of $r(x)$ as a function of the learning set size

Figure 4.19 then shows the influence of the size of the dataset on the variance of the density ratio. Here, the analysis can be divided into three points. First, the cases corresponding to low accuracy in Figure 4.18 seem to have their variance decreasing slowly when the size of the dataset increases. Then, there is the positive case of the blackjack for which there seems to be a huge drop when only a thousand samples are used. For all other sizes, the variance does not seem to vary that much. Finally, the positive case of the dice and slot games for which the size looks to have absolutely no impact on the metric. This might be a confirmation that Extra trees are able to achieve quite good performances without requiring that much samples for this particular type of problems.

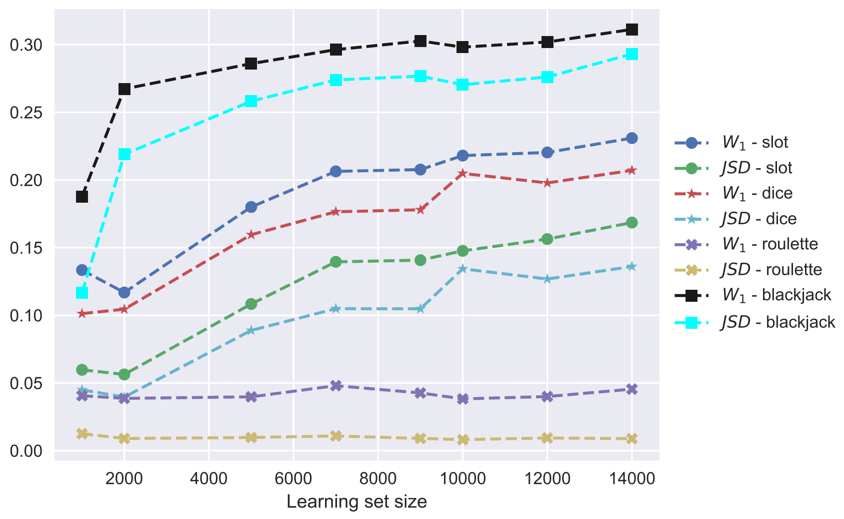


FIGURE 4.20: W_1 and JSD for $p(r(x))$ as a function of the learning set size

Lastly, Figure 4.20 shows how the variation of the probability densities metrics according to the size. Here, in most of the situations, the metrics are increasing slower and slower, but do not seem to reach a limit yet. This would mean that, contrary to other metrics, it would be here possible to highlight anomalies much more by using more samples. However, it should also be considered that, for complex games, generating this data by simulation can take much time. Conversely, for roulette, the size of the dataset seems to have influence on the distance metrics. This is a problem that we have already observed for the roulette game, as the anomaly is quite hard to detect in this situation. The only way that's been found to counter this issue was to use calibration.

4.1.9 Time Performances

In this section, we address the time performances of the supervised learning procedure quickly when Extra trees are used as base estimator. We insist on the fact that the execution times in Table 4.10 represent only the supervised learning task, and not all the data collection and preprocessing in order to obtain a dataset. The time performances of these tasks are assessed in the internship report [18, Merchie 2017]. To be able to interpret these results correctly, it's s required that cross-validation was used for both base estimator learning and classifier calibration according to Table 4.1. That's why the execution time grows so much when calibration is performed. However, without calibration, it's observed that the execution time first increased then decreased. It goes up first from no concatenation to 500 because no cross-validation is performed without concatenation. Then, it slowly drops because Extra trees is more sensitive to a decreasing in the number of samples than an increasing of the number of features and cross-validation folds.

	No concatenation	500	1000	2000
No calibration	37s	91s	82s	64s
Calibration	30s	455s	757s	960s

TABLE 4.10: Execution time of the supervised learning procedure with extra trees according to concatenation length and calibration

4.2 Standard supervised approach - Linear SVM

In the section, we focus on the second base estimator of the standard supervised approach: linear SVM. Just like for Extra trees, the influence of its different hyper-parameters will be depicted. Also, a more general analysis will be performed by analyzing the performances according to the size of the learning set as well as the required time to achieve these performances.

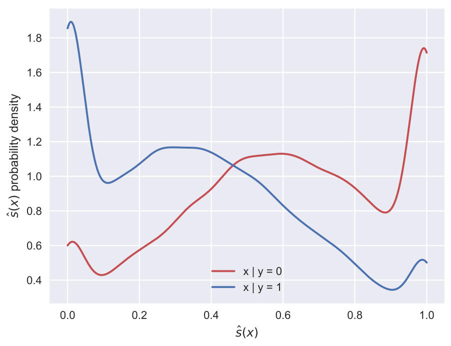
4.2.1 Experiment with default parameters

Proceeding the same way as for Extra trees, a first experiment has been performed using the default parameters of the linear SVM implementation of scikit-learn. As a reminder, these are available in Table 3.4. The main purpose of this section is to compare the interpretability of the results with the ones obtained for the Extra trees. The results of this experiment are displayed in Figure 4.21.

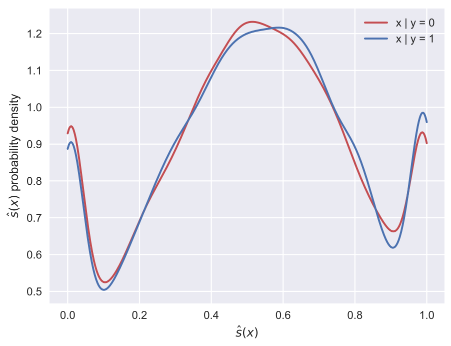
First, some remarks can be made about the probability densities of Figures 4.21a and 4.21b. It's observed that they differ much more in the positive case than when Extra trees were used. In the opposite, the densities tend to mostly overlay each other in the negative case, which is a similar behavior as for the previous estimator. However, the shape of the densities themselves has nothing to do with the ones obtained for the Extra trees. Actually, we observe the same pathological behavior noticed for Extra trees when no concatenation was performed.

The issue of getting probabilities very close to 0 or 1 for some samples can lead to interpretation problems. Indeed, as seen on Figures 4.21c and 4.21d, to get high values for the density ratio even in the negative case. In particular, the values obtained for the variance of the density ratio tend to be much higher in the case where linear SVM is used as base estimator. Therefore, even though the probability densities are very similar in Figure 4.21b, this feature is not really enhanced by other Figures. Particularly, the probability densities in Figure 4.21d look very similar, which should to relatively small JSD and W_1 distances compared to what Extra trees had given. This is a weakness compared this other base estimator where the interpretabilty was graphically very clear. However, Figure 4.21e representing the ROC curve does not seem to be affected by previous issue, which is quite logical.

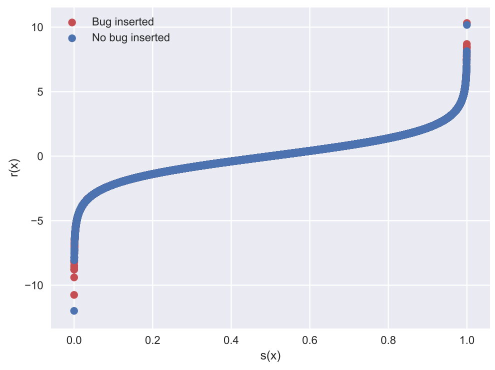
The main goal of hyper-parameters analyses of further sections will be therefore to see whether it's possible to have easier interpretation values for the variance and the probability densities of the density ratio.



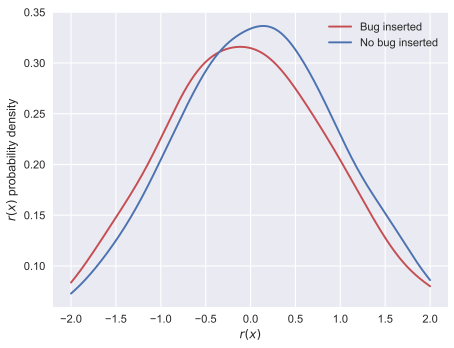
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



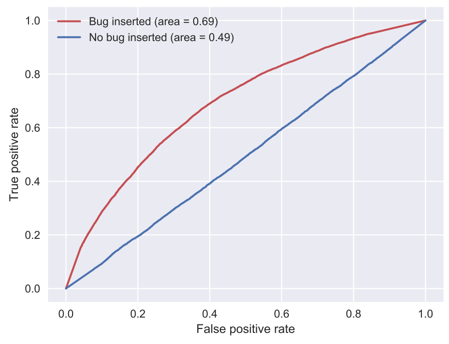
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio



(E) ROC curve

FIGURE 4.21: Density ratio analysis of slot game: experiment with default parameters

4.2.2 Influence of the penalty parameter

The first hyper-parameter for which the influence is analyzed in the case of is the penalty parameter C. Let's remind that this parameter is linked with how much misclassification wants to be avoided for each training sample. Therefore, for higher values, smaller-margin hyperplanes will be preferred by the optimization if they do a better job at classification. For the corresponding experiments, the following range of values for C has been chosen: 0.1, 0.2, 0.5, 0.7, 1, 2, 5.

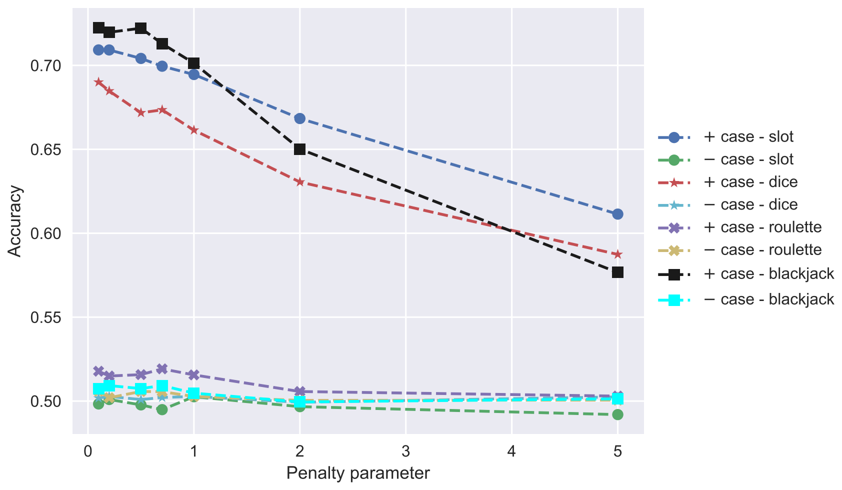


FIGURE 4.22: Accuracy as a function of C

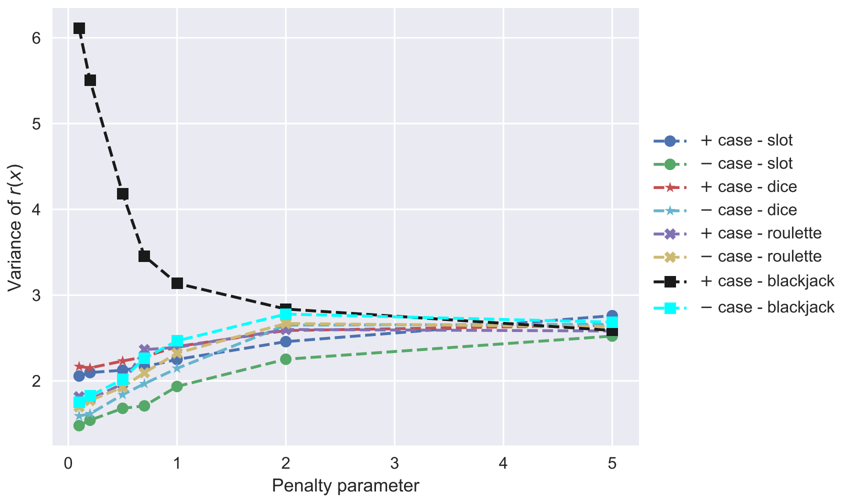


FIGURE 4.23: Variance of $r(x)$ as a function of C

First, Figure 4.22 shows how the accuracy is impacted by the penalty parameter. It's obvious that more distinct results are obtained for smaller values C. Therefore, it can be asserted that high-margin hyperplanes tend to classify the datasets better. Note that remark is only relevant for this positive case of the configurations, as in the negative case, the dataset should not be separable in any way. By contrast, it's observed that increasing the value of C causes the results of the positive and the negative cases to be brought closer to each other, which is the exact opposite of the basically sought goal. This is particularly discernible when having a look at Figures

4.23 and 4.24. In these, the metrics seem to converge to an actual same value for all the configurations. Furthermore, the expectations that we had about the distance metrics of the probability densities of the density ratio are confirmed in this section. Indeed, when looking at the y-axis of Figure 4.24, it's observed that we obtain smaller values for the metrics than we had obtained for Extra trees. This could be problematic, as we have seen, when analyzing Extra trees, that these metrics could be useful to enhance anomalies that were not easily detected thanks to accuracy or variance of the density ratio.

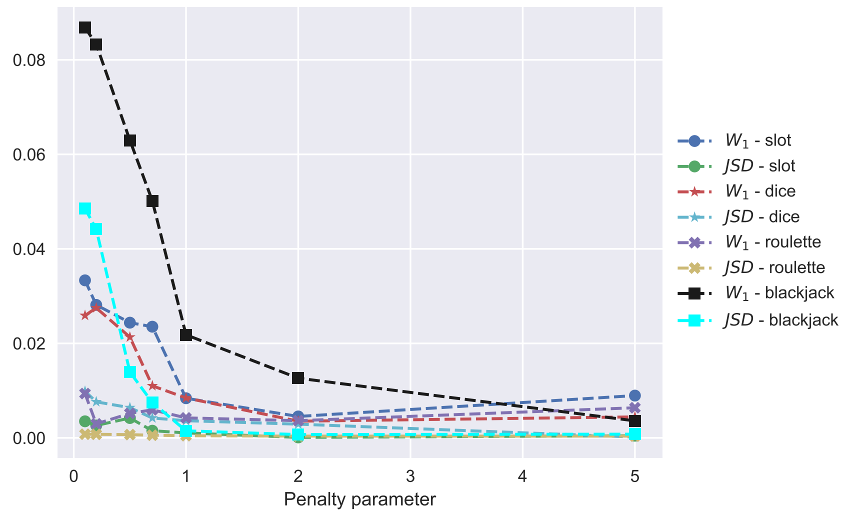


FIGURE 4.24: W_1 and JSD for $p(r(x))$ as a function of C

As a conclusion for this hyper-parameter, it can be said that increasing too much the value of C is equivalent to shadowing anomalies in the data, and prevent therefore the possibility to detect them. This is course totally undesired, and that's why smaller penalty parameter values will be preferred in our situation. Next section will focus on the influence of the loss function.

4.2.3 Influence of the loss function

In this section, we observe how the choice of the loss function can impact the different metrics. There are two possible specifications for this hyper-parameter in the implementation of scikit-learn: ϵ-insensitive or squared ϵ-insensitive, respectively corresponding to a L1 and L2 loss. The associated results are available in Tables 4.11, 4.12, 4.13 and 4.14.

The analysis of the results is quite straightforward. Indeed, concerning first the accuracy, there is only one situation when the parameter has a real impact on the metric, which the dice game. In this case, the L2 loss gives a rise of about 4% of the accuracy compared to the L1 loss in the positive. In every other situations, we observe that the accuracy remains unchanged from one loss specification to another. Concerning the variance of the density ratio, the effects are here slightly more significant. As it can be noticed, choosing a L2 loss function tends to reduce the metric values in both positive and negative cases of all situations but, more importantly, it shrinks the gap between the two cases, which is not was we want at all. In anyway, it can be observed several times in these Tables, the variance gets higher in the negative case. Of course, this is due to the behaviour of the probability densities observed in section 4.2.1, but that doesn't change the fact that the metric becomes irrelevant in the current situation.

Loss function	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
ϵ-insensitive	+	0.6184	2.7314	0.0007	0.0086
	-	0.4973	2.503		
squared ϵ-insensitive	+	0.6200	1.9159	0.0003	0.0044
	-	0.4946	1.8554		

TABLE 4.11: Influence of the loss function: slot game

Loss function	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
ϵ-insensitive	+	0.5775	2.6739	0.0005	0.0056
	-	0.5005	2.6083		
squared ϵ-insensitive	+	0.6111	2.2216	0.0006	0..0100
	-	0.5064	2.0224		

TABLE 4.12: Influence of the loss function: dice game

Loss function	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
ϵ-insensitive	+	0.4989	2.6066	0.0001	0.0051
	-	0.4976	2.6276		
squared ϵ-insensitive	+	0.5056	2.2011	0.0009	0.0050
	-	0.5016	2.2233		

TABLE 4.13: Influence of the loss function: roulette game

Loss function	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
ϵ-insensitive	+	0.5769	2.6816	0.0001	0.0044
	-	0.4968	2.6922		
squared ϵ-insensitive	+	0.5752	2.3807	0.0002	0.0040
	-	0.5060	2.3556		

TABLE 4.14: Influence of the loss function: blackjack game

Finally, let's have a few words about the distances between the probability densities of the density ratio. As noticed in the last two columns of the Tables, they tend to stay very small, just as in previous section. It's therefore very difficult to draw any interesting interpretation about these metrics in the current situation.

4.2.4 Influence of the calibration method

Just as it's been done with Extra trees, this section addresses now the introduction of calibration in the procedure of Algorithm 1. Previously, we could already observed the benefits on using calibration. In particular, it enlarges the gap of density ratio variance between the positive and negative cases of all games by reducing the variance of the negative case a lot. Moreover, it provides a solution to the problem of detecting smaller anomalies such as in roulette game, thanks an increasing of the distances between the probability densities of the density ratio. This section will allow us to know if same conclusions can be drawn about LSVM. The associated results are shown in Tables 4.15, 4.16, 4.17 and 4.18.

Concerning first the accuracy, it's immediately obvious that calibration has again a positive impact on the procedure. This time, contrary to Extra trees, there's already a non negligible influence on the accuracy. The increasing goes from 3-4% for the sigmoid calibration in roulette to almost 10% for several calibration methods in blackjack. It still remains under the best performances obtained with Extra trees but this clearly shows the power of calibration in this kind of problem. Let's now analyze the effects on the variance of the density ratio. It's observed that, globally, the same consequences are observed as for Extra trees. However, the gap between the variance of the positive and negative cases are not as big as in Extra trees. Considering now the last two metrics, there is no new trend to raise out the obtained results. These are globally similar to the ones got for Extra trees and don't give better performances.

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.6314	2.0364	0.0004	0.0089
	-	0.4965	1.7801		
Histogram	+	0.7012	0.1213	0.3639	0.3606
	-	0.4996	0.0029		
Isotonic	+	0.6977	0.2527	0.2089	0.2626
	-	0.4946	0.0203		
Interpolated	+	0.6944	0.3280	0.1256	0.2084
	-	0.4977	0.0507		
Sigmoid	+	0.7020	0.1058	0.5678	0.4595
	-	0.5018	0.0001		

TABLE 4.15: Influence of the calibration method: slot game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.6025	2.2629	0.0005	0.0072
	-	0.5013	2.1536		
Histogram	+	0.6737	0.0612	0.3002	0.3105
	-	0.4960	0.0027		
Isotonic	+	0.6701	0.1522	0.1208	0.1894
	-	0.5052	0.0283		
Interpolated	+	0.6652	0.2501	0.0803	0.1468
	-	0.5011	0.0627		
Sigmoid	+	0.6743	0.0478	0.5322	0.4429
	-	0.4962	0.0001		

TABLE 4.16: Influence of the calibration method: dice game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.5011	2.2428	0.0002	0.0068
	-	0.5033	2.2052		
Histogram	+	0.5143	0.0036	∞	0.0389
	-	0.5051	0.0029		
Isotonic	+	0.5096	0.0298	0.0018	0.0064
	-	0.5046	0.0300		
Interpolated	+	0.5105	0.0632	0.0018	0.0061
	-	0.5022	0.0663		
Sigmoid	+	0.5319	0.0008	∞	0.1956
	-	0.4954	0.0001		

TABLE 4.17: Influence of the calibration method: roulette game

Method	Case	Accuracy	$V[\hat{r}(x)]$	$JSD(+\|\|-)$	$W_1(+,-)$
None	+	0.6154	2.4056	0.0001	0.0058
	-	0.5010	2.2516		
Histogram	+	0.7047	0.0858	0.3543	0.3548
	-	0.5052	0.0026		
Isotonic	+	0.7030	0.1971	0.1609	0.2256
	-	0.5026	0.0248		
Interpolated	+	0.7123	0.3655	0.1199	0.1931
	-	0.5022	0.0646		
Sigmoid	+	0.7049	0.0786	0.5684	0.4606
	-	0.5002	0.0001		

TABLE 4.18: Influence of the calibration method: blackjack game

4.2.5 Metrics as a function of the dataset size

Now the analysis of the influence of the different parameters is done, let's do what has already been performed for Extra trees, i.e. analyzing the evolution of the different metrics according to the size of the dataset. Once again, as calibration had a very significant impact on the results, it's also going to be used in this section.

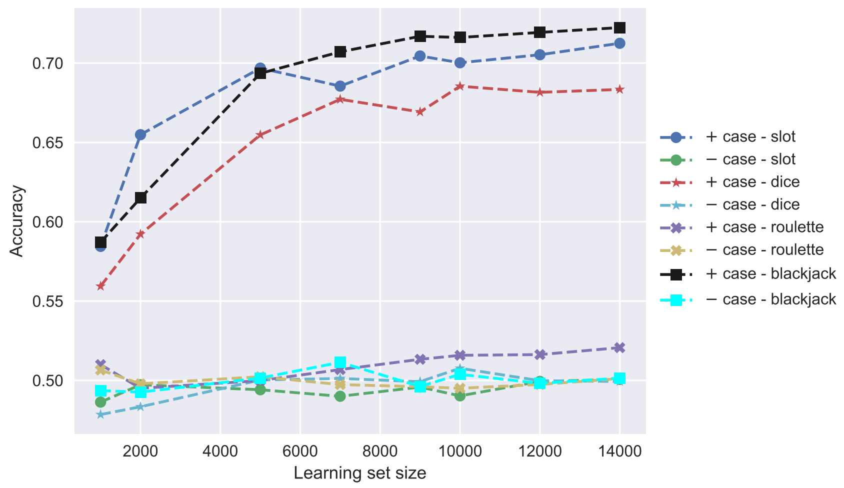


FIGURE 4.25: Accuracy as a function of the learning set size

Let's begin with the accuracy metric. The corresponding Figure 4.25 shows how fast this converges according the size of dataset. First, it should be considered that such a analysis is relevant only in the positive cases of the different configurations. Indeed, for negative cases, the accuracy should only slightly oscillate around 0.5. So, looking at the positive cases in Figure 4.25, it's indeed observed that the accuracy curves progress following a logarithmic increasing, which is the typical shape for such curves. Also, it's observed that from 10000 samples, the 3 top curves start to increase slower, but still gain a few percents for some games, such as slot or roulette. Also, for roulette game, even if the accuracy value itself remains low, the behavior of the positive curve can still give us a clue about the presence of an anomaly, as it doesn't actually oscillate.

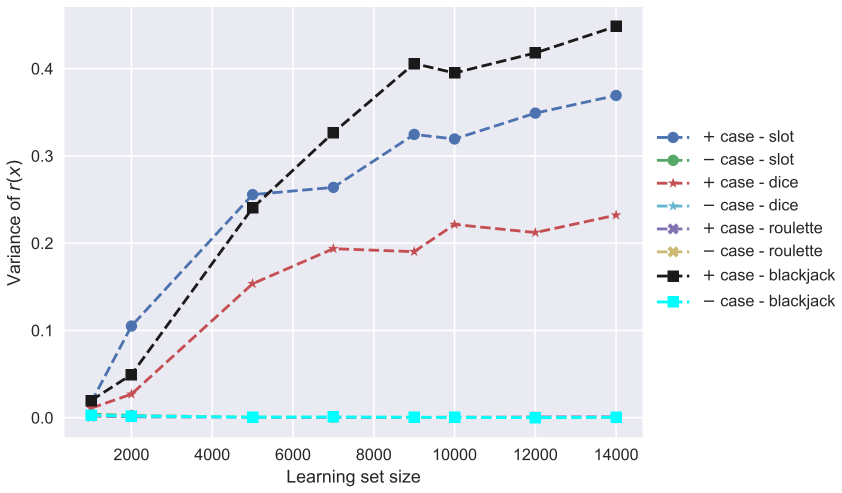


FIGURE 4.26: Variance of $r(x)$ as a function of the learning set size

Let's now consider the variance of the density ratio. Here, we can immediately see that the metric is much more sensitive to the size of the dataset. Indeed, the increasing of the variance seems to be much sharper for the 3 top curves. Here, contrary to accuracy, it's much more difficult to distinguish the positive case of the roulette from all other negative cases. In such situations, it should therefore be interesting not to check the variance value alone, but instead, once an anomaly is suspected by the the accuracy metric, to emphasize the impact of more serious anomalies.

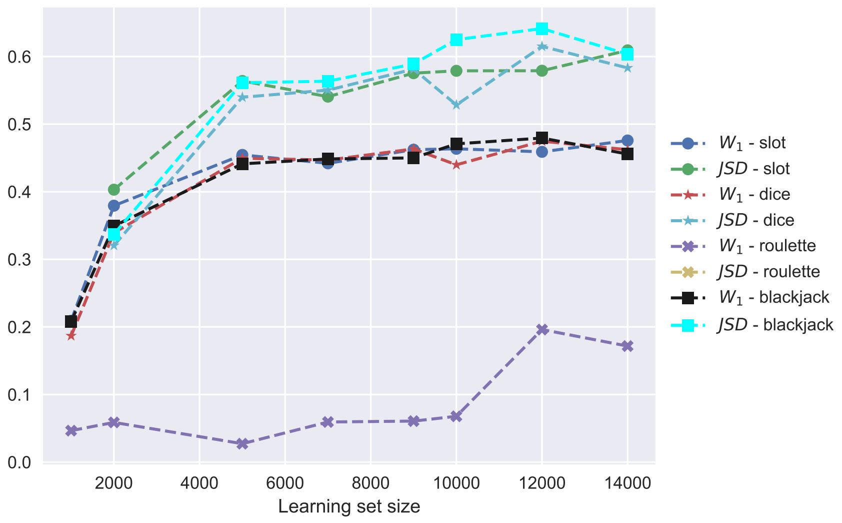


FIGURE 4.27: W_1 and JSD for $p(r(x))$ as a function of the learning set size

Finally, Figure 4.27 provides information about the JSD and W_1 metrics of density ratio probability densities. In this case, considering every situation except roulette game, the metrics seem to stabilize relatively quickly compared to other metrics. However, as it's been observed from other previous Figures, it does not seem that more samples cannot allow to highlight more an anomaly in a game. Indeed, for the variance metrics, no clear stop in the increasing of the metric in the positive cases has been observed, even for highest values of the dataset size. This would mean it could be possible to observe even more differences between positive and negative cases by collecting more samples. However, the time required to generate this data from simulation has also to be taken into account, and it's not negligible (can go up to 5 hours for more complex games).

4.2.6 Time Performances

With this section, we have a quick look at time performances when using LSVM as base estimator. These are available in Table 4.19. It's first noticed that the algorithm is globally slower than Extra trees, even though it obtains poorer results. The reason is simple as, in Extra Trees, the growing of the different trees can be shared in multiprocessing, which is not the case for LSVM. Moreover, the increasing in the number of features affect LSVM more than Extra trees, as the latest selects its features randomly to grow its forest. Otherwise, the influence of calibration and cross-validation remains the same as for Extra trees.

	No concatenation	500	1000	2000
No calibration	60s	216s	223s	200s
Calibration	47s	1362s	2136s	2620s

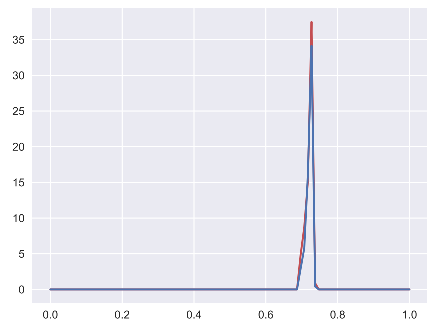
TABLE 4.19: Execution time of the supervised learning procedure with LSVM according to concatenation length and calibration

4.3 Deep learning approach

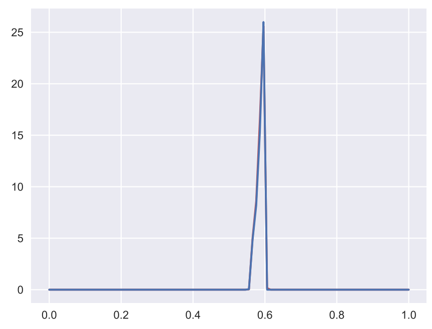
This last major section addresses the third approach that was chosen to achieve the objective stated in Chapter 3. Contrary to the two previous approaches, this one relies on deep learning. The architecture of the corresponding neural network was presented in section 3.6. This section will not be developed as much as previous ones as Extra trees already gave very satisfying results. Moreover, as it will be observed in further subsections, the approach won't give acceptable enough results to allow ourselves to accord more time to this approach. Considering the experiments themselves, it's been decided to test the architecture on the 4 different problem configurations and, for each of them, highlight the different observed issues. The parameters values used for these experiments are still the ones of Table 3.4.

4.3.1 Slot experiment

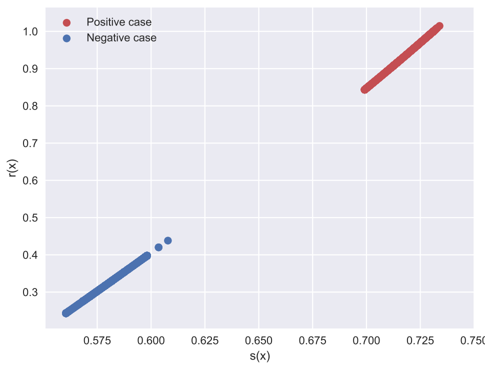
Concerning first slot game, the corresponding experiment is depicted in Figure 4.28. Immediately, something strange appears when looking at the probability densities in Figures 4.28a and 4.28b. Indeed, contrary to Extra trees or LSVM, the densities are here not balanced between the two classes. Instead, they the neural network tend to always give a higher probability in the positive class, in both positive and negative cases, resulting in values of the density ratio being always positive. Moreover, the densities are really shrunk, which means the neural network cannot differentiate the different samples very well according to their temporal features. Of course, this gives dissimilar probability density plots for the density ratio in Figure 4.28d but it's not really supported by the ROC curve displayed in Figure 4.28e. In this experiment, there is thus only one metric that might help us to differentiate positive and negative case. Nevertheless, the predictions of the network about the samples of the negative case are not really consistent with what was expected, i.e. values centered around 0.5.



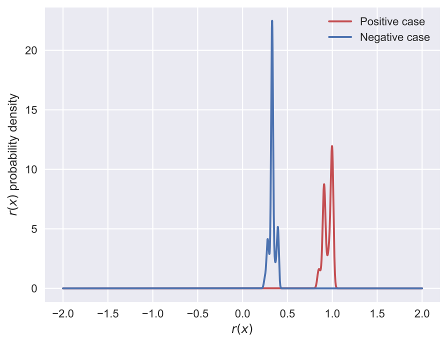
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



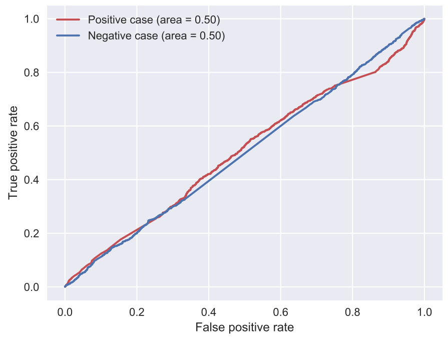
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio



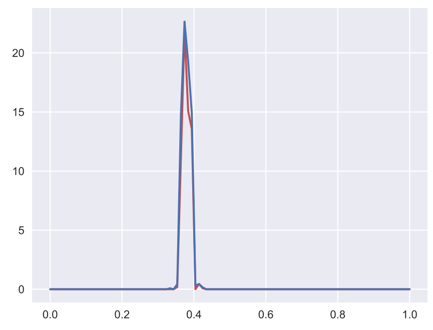
(E) ROC curve

FIGURE 4.28: Density ratio analysis of slot game: deep learning approach

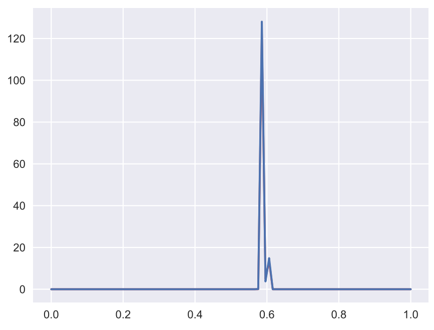
4.3.2 Dice experiment

Let's now analyze the results of the experiment associated to dice game. Results are shown in Figure 4.29. Considering the probability density of the ratio density and the ROC curves, the problems are approximately the same as for slot games.

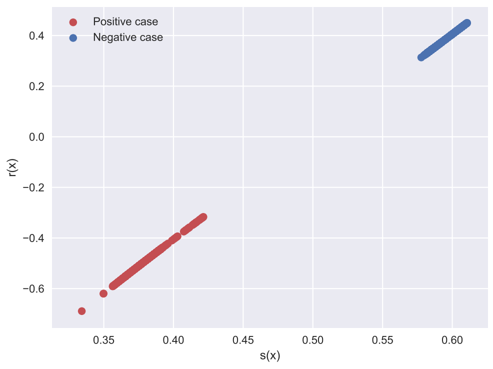
Considering the other metrics, the problem is slightly different. On the one hand, in the negative case, the samples are still all considered to be more likely to belong to the positive class. But on the other hand, in the positive case, the samples are all more likely to belong to the null class, no matter the true original. Once again, this is inconsistent with the assumed dataset. Obtaining such results about the probability densities of the prediction which are the same no matter the true original label, gives us absolutely no relevant information about the presence of an anomaly in the game. Therefore, it wouldn't make any sense to assert an anomaly from Figure 4.29d just because the densities have different shapes.



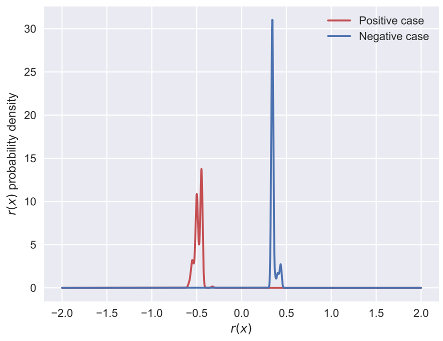
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



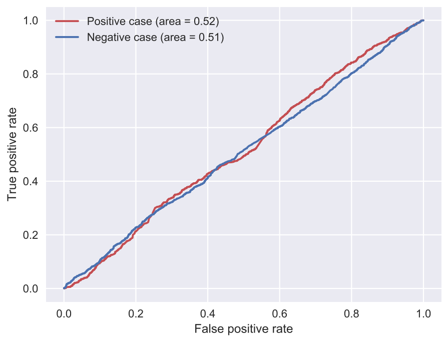
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label

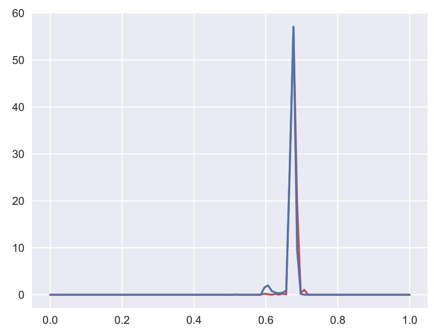


(D) Probability density function of the density ratio

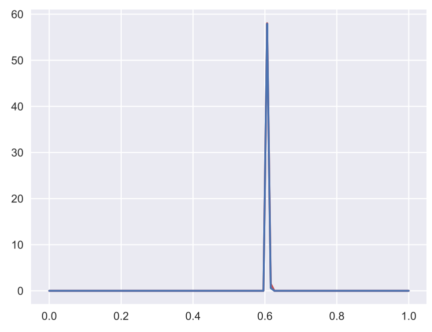


(E) ROC curve

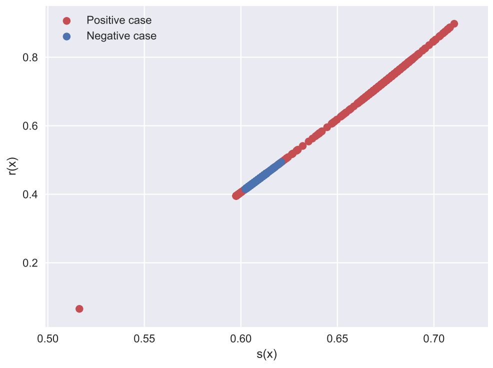
FIGURE 4.29: Density ratio analysis of dice game: deep learning approach



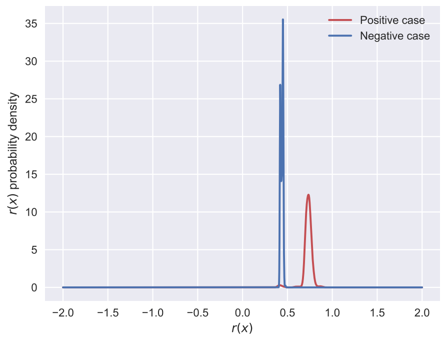
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



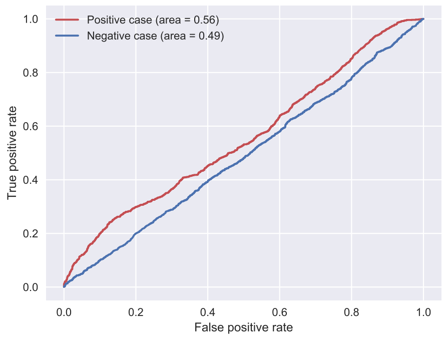
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio

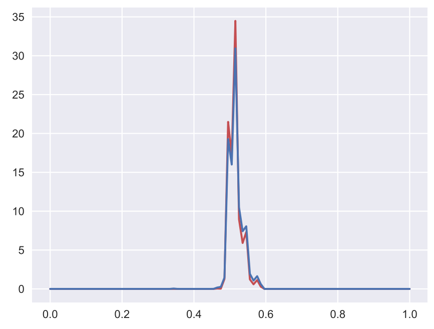


(E) ROC curve

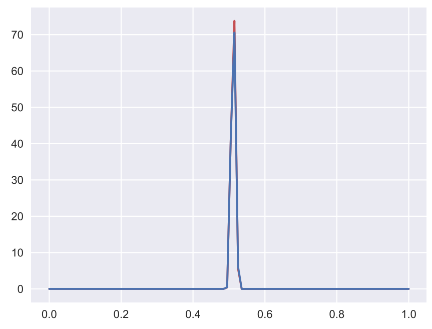
FIGURE 4.30: Density ratio analysis of blackjack game: deep learning approach

4.3.3 Blackjack experiment

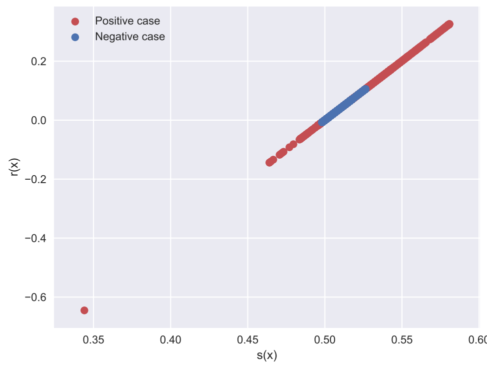
Let's now switch to the blackjack configuration, whose results are displayed in Figure 4.30. In this situation, as observed in Figure 4.30a, the probability densities are centered around approximately the same value as densities of Figure 4.30b, but are slightly more spread. But in the end, this leads to the same kind of problem as in previous experiments, which is the impossibility to distinguish both classes in the positive class according to the probability densities of the prediction. The only auspicious detail concerns the ROC curve of the positive in Figure 4.30e which is a little bit more distinct of the ROC curve of the negative case. However, it's still insufficient to validate the results.



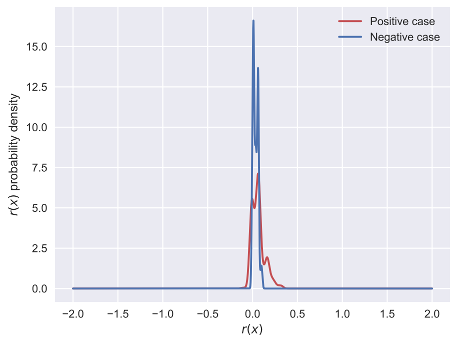
(A) Probability density function of $\hat{s}(x)$ given the true label (positive case)



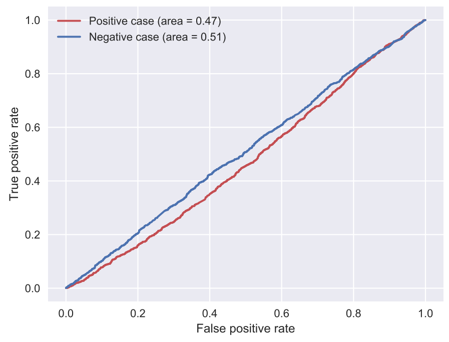
(B) Probability density function of $\hat{s}(x)$ given the true label (negative case)



(C) Density ratio as a function of $\hat{s}(x)$ given the true label



(D) Probability density function of the density ratio



(E) ROC curve

FIGURE 4.31: Density ratio analysis of roulette game: deep learning approach

4.3.4 Roulette experiment

Roulette's experiment may be the most particular experiment among the four. Indeed, probability densities of the prediction are more spread in the positive case (4.31a) than in the negative one (4.31b), leading to a larger range of values for the density ratio in Figure 4.31c. At first sight, it could thus be expected that the network is able to distinguish the classes a bit more in the positive case. But then, when looking at the ROC curve in Figure 4.31e, it's then noticed that the accuracy in the positive gets lower than in the negative case. This is actually due to the fact that, even though the densities are more spread in Figure 4.31a, they're not more distinct, as they almost overlay each other perfectly. Therefore, it can't lead to an improvement of the accuracy.

By way of conclusion for the deep learning approach, we have thus than it did not give satisfying results at all. In particular, they don't compete with the visual interpretability of the standard supervised approach, especially for Extra trees. The main reason that could have caused such of a fail of this approach could be the input data. Indeed, as said in section 3, the collected data is aggregated along both x and y-axis, which leads to a non negligible loss of information. Certainly, this lack of information has been counterbalanced by the preprocessing procedure (feature extraction + imputation) of the standard supervised approach.

Chapter 5

Conclusion

In this chapter, a brief reminder of the main objective of the master thesis is done and the results associated to each approach presented in Chapter 3 are synthesized according to this objective. Our goal was to be able to detect modifications in the behavior of a casino game after an update. This problem was tackled as a density-based anomaly detection problem and solved with supervised learning algorithms. Datasets corresponding to a binary classification problem were built, where one class corresponds to data generated from the game before its update, and the other class after its update. The method should be able to classify with a sufficiently high accuracy datasets for which the second class contained an anomaly, but also shouldn't be able to differentiate classes of clean datasets. Moreover, the density ratio of the different samples were computed and its derived probability density should be explicitly linked to the presence of an anomaly or not in the game.

Extra trees. Concerning the standard supervised approach with Extra trees as base estimator, the results were globally very satisfying. For 3 out of 4 validation cases, the method was able, with its default parameters, to provide a very clear distinction between the case where an anomaly has to be detected in a game and the case where nothing was done. The reached accuracy in these situations was 0.69 at worst for dice game and 0.74 at best for blackjack games. Moreover, for the corresponding cases without anomaly, the accuracy stays remarkably close to 0.5 and the probability densities of the density ratio of an anomaly case were very easy to distinguish visually from a clean case. The introduction of calibration has allowed to enhance even more this last feature by drastically reducing the variance of the density ratio in clean cases. In particular, sigmoid calibration did a really good job about this. At best, this allowed to set the ratio of the variances of the density ratio (of anomaly and clean case) in the order of 10^3. Calibration also permitted to solve the configuration that was an issue with the default parameters. An interesting point that has also been highlighted during the analysis concerns the concatenation of the data obtained from simulations. When no concatenation was done, we observed a major modification of the shape of the probability densities of the prediction of the classifier, according to the true label. Actually, too small time-series tended to break the dataset by being too easy to idendify for the classifier. We have also seen that this method has relatively reasonable execution time, most of the resources being spent during the preprocessing of the data.

Linear SVM. In the second case of standard supervised approach, with Linear SVM as base estimator, the results were still good, but couldn't compete against

Extra trees. There are several reasons to this. The main one is that, with the default parameters, only the accuracy metric became relevant to assert the presence of anomaly in a game. Indeed, with Linear SVM, the same kind of pathological influence on the prediction of the classifier due to absence of concatenation for Extra trees was observed here even with the default parameters, which included data concatenation. Another reason concerns the execution time. Indeed we observed that, despite achieving poorer performances, Linear SVM took more time to solve the given problem than Extra trees. Only the introduction of calibration allowed this method to get closer of the results obtained with Extra trees. For the 3 easiest configurations to solve, we get at worst 0.67 for dice game and at best 0.71 for blackjack game.

Deep learning with RNN. Concerning the deep learning approach, the results were very disappointing. We didn't even go through the whole analysis of the hyper-parameters, as it's immediately been noticed that it wouldn't be possible to reach the performances of the standard supervised approach. Here, none of the validation cases could be solved using the method. The most suspect cause of this fail is certainly the input data. Indeed, in the case of standard supervised approach, the time-series went trough a big preprocessing procedure of which the main feature consisted in extracting statistical features from temporal data, which seemed to counterbalance the loss of information of data aggregation. Of course, this not performed in this case as RNN are usually directly fed with time-series data.

In a nutshell, the global problem to solve in this master thesis allowed to use varying approaches, relying on different types of machine learning algorithms and preprocessing techniques. At the end, we concluded that Extra trees could achieve the requirements of the objectives and obtained the best experimental results.

Bibliography

[1] P. Geurts, *Introduction to machine learning: Bias/variance tradeoff, model assessment and selection*, Slides of the course *Introduction to machine learning*. URL: `http://www.montefiore.ulg.ac.be/~lwh/AIA/model-evaluation-29_10_2012.pdf`. Last visited on 2018/05/11, 2016.

[2] V. Vapnik, “Pattern recognition using generalized portrait method”, *Automation and remote control*, vol. 24, pp. 774–780, 1963.

[3] C. Cortes and V. Vapnik, “Support-vector networks”, *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[4] K.-B. Duan and S. S. Keerthi, “Which is the best multiclass svm method? an empirical study”, in *International workshop on multiple classifier systems*, Springer, 2005, pp. 278–285.

[5] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees”, *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[6] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, “Classification and regression trees”, 1984.

[7] P. Geurts, *Introduction to machine learning: Ensemble methods*, Slides of the course *Introduction to machine learning*. URL: `http://www.montefiore.ulg.ac.be/~lwh/AIA/ensembles-21-11-2017.pdf`. Last visited on 2018/05/11, 2016.

[8] L. Breiman, “Random forests”, *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities”, *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[10] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks”, *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[11] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.”, *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[12] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey, “Efficient orthogonal parametrisation of recurrent neural networks using householder reflections”, *ArXiv preprint arXiv:1612.00188*, 2016.

[13] Q. Chen, X. Zhu, Z.-H. Ling, S. Wei, H. Jiang, and D. Inkpen, “Recurrent neural network-based sentence encoder with gated attention for natural language inference”, *ArXiv preprint arXiv:1708.01353*, 2017.

[14] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey”, *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[15] K. Cranmer, J. Pavez, and G. Louppe, “Approximating likelihood ratios with calibrated discriminative classifiers”, *ArXiv preprint arXiv:1506.02169*, 2015.

[16] A. Niculescu-Mizil and R. Caruana, “Predicting good probabilities with supervised learning”, in *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005, pp. 625–632.

[17] G. Louppe, *Advanced machine learning: Generative adversarial networks*, Slides of the course *Advanced machine learning*. URL: http://www.montefiore.ulg.ac.be/~geurts/Cours/AML/Slides/AML-2018_Lecture6a.pdf. Last visited on 2018/05/11, 2018.

[18] F. Merchie, "Data preprocessing in machine learning for anomaly detection in casino games", 2017.

[19] M. Christ, *Tsfresh: A time-series feature extraction toolbox*, 2016. [Online]. Available: http://tsfresh.readthedocs.io/en/latest/index.html.

[20] R. Gutierrez-Osuna, *Introduction to pattern analysis: Kernel density estimation*, slides of the course *Introduction to pattern analysis. URL: http://csyue.nccu.edu.tw/ch/Kernel%20Estimation(Ref).pdf. Last visited on 2018/05/11*, 2016.

[21] A. Rubinsteyn, *Fancyimpute: A missing values imputation toolbox*, 2016. [Online]. Available: https://github.com/iskandr/fancyimpute.

[22] G. Louppe, K. Cranmer, and J. Pavez, *Carl: A likelihood-free inference toolbox*, 2016. DOI: 10.5281/zenodo.47798. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47798.

[23] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems", in *Advances in neural information processing systems*, 1997, pp. 473–479.

[24] A. Murad and J.-Y. Pyun, "Deep recurrent neural networks for human activity recognition", *Sensors*, vol. 17, no. 11, 2017, ISSN: 1424-8220. [Online]. Available: http://www.mdpi.com/1424-8220/17/11/2556.