What Are The Rules For Tic-Tac-Toe? A Theoretical Machine Learning Perspective

Luis Vitor Zerkowski *

Institute of Mathematics and Statistics of University of São Paulo (IME-USP)

July 2021

1 Abstract

This document is an introduction to theoretical machine learning field. It contains the fundamental concepts of the subject followed by an experimental section to help people better understand machine learning applications, potentials and also its limitations. It is written in a formal mathematical way, but it also has a storytelling essence to properly welcome every reader who ventures to peruse it.

2 Introduction

This article is focused on applications in machine learning using the knowledge framework of PAC-Learning theory. The goal of this paper is to make some experiments with a learning agent given some context, previous knowledge about the task and, most important, a formalized and controlled environment. Since this study is not specially interested in optimizations but in the way one describes its domain, it starts with simple learning algorithms and then try to improve them until it reaches more advanced and modern techniques.

More objectively, this document is about PAC-Learning experiments in the Tic-Tac-Toe game. The main idea is to build an agent capable of differentiate victory states from non-victory states or, put in another way, an agent that learns the rules of the game. In order to achieve this agent, some predicates that analyze the state of a given match will be created and some samples will be assembled to form a special database used by the learning algorithms.

*This work was sponsored by the Brazilian funding agency CNPq.

After preliminary phases, some information will be hidden from the agent to make learning a little harder and the impacts of these changes will be discussed later. This procedure goes on until the sample size required for learning are completely exaggerated. At this point, other learning approaches will be evaluated and some other problems will be brought up for discussion.

3 The Fundamentals

Before entering directly in the experiments approached in this paper, we introduce some essential concepts in computational learnability theory. The first topic that comes to mind whenever we talk about computational learnability is the very definition of learning. Although there exists some controversy about the exact meaning of the word - which is also a whole philosophy study field -, we here focus a little bit more on a formal statistical model for solving a given problem and claim without much concerning that, in general, learning resides somewhere between the simple memorization of data and the random estimates.

For a little bit of immersion, imagine you got a break from work and decided to go on vacation to Brazil, more precisely to Itapicuru, a small town in Bahia. On the journey to clear your mind and get away from the big city, you end up meeting a wise skinny man who invites you to play a board game and gives you the following information:

1. Everything happens on a three by three square board.
2. The game is played only by two players, one of them represented by the white stones and the other represented by the black stones.
3. To play the game you just need to choose an empty square of the board and place your stone on it. From now on, this square belongs to you.
4. I will not tell you how to win the game, but I will always tell you when the game is done and whether you win or lose it.
5. Your single task is not to lose from me five times in a row explaining how you did it.
6. I assure you that you will better understand machine learning and the very concept of learnability when our work here is done.

Although you get a little confused by the unexpected proposal, your scientific soul does not allow you to walk away from such an adventure. You start to systematically think of good approaches to the problem and, for the rest of this article, we will formalize everything you possibly thought from the beginning to the end of this intellectual challenge.

3.1 PAC-Learning

The first thing that comes to mind when you accept the proposal is how not to lose this game that I really don't know nothing about. And the answer is simple: I have to learn the rules of a victory state so I can prevent the wise man from winning. But what is exactly learning and how to do it?

We now formalize these first thoughts of yours as well as our main learning framework and provide you some important intuitions:

> Definition of PAC-Learnability: A hypothesis class H is PAC learnable if there exists a function $m_H : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in (0,1)$, for every distribution D over X , and for every labeling function $f : X \to \{0,1\}$, if the Realizable Assumption holds with respect to H, D, f , then when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d. examples generated by D and labeled by f , the algorithm returns a hypothesis $h \in H$ such that, with probability of at least $1 - \delta$ (over the choice of the examples), $L_{D,f}(h) \leq \epsilon$[1].

Domain $X \to$ The set of all mathematical objects we want to predict something for. In our context, X can be understood as a picture of the entire board but also as a tuple of board properties - the presence of white stone lines or white stone columns for example.

Hypothesis class $H \to$ A set of all hypothesis - functions - from the domain X to $\{0,1\}$ we believe can approximate or even be the labeling function f. In our context, H can be understood as ways to evaluate a given state of the board as a win or a lose - theorize that the presence of a stone in the middle of the board indicates a victory to the stone owner for example. More objectively, it is the set of all rules you are considering good options to describe the actual rules of the game.

Accuracy ϵ and confidence $\delta \to$ The chosen parameters for how much error we tolerate - approximately - and how confident we want to be - probably - that we have achieved at most that error. These two parameters are responsible for the name of the approach: Probably Approximately Correct Learning. In our context, ϵ can be understood as how confident you are that your strategy is really close to the real function that determines whether the game is a win or a lose, and δ can be understood as how certain you are that you would pick this good strategy even if you went back in time and started the challenge all over again.

Distribution $D \to$ A probability distribution over the domain that chooses instances of X according to its rules. In our case, it can be

understood as our games. Every time we play a game until the end it is just like we have picked an example from the distribution.

Labeling function $f \rightarrow$ The function we optimally would like our predictor to be. It is the function truthfully responsible for determining the output, 0 or 1, given an input x from our domain. In our case, it would be the rules of the game we are trying to learn.

Realizable assumption $\rightarrow$ A core concept for the PAC-Learning theory. It is the assumption that the labeling function we are looking for can be found in our hypothesis class H. In other words, $\exists h^* \in H : L_{D,f}(h^*) = L_{D,f}(f) = 0$. This one is a little more tricky. In our case, it would be something like knowing for a fact that you already played this game before and that if you try hard enough you could remember them.

Loss function ($L_{D,f} : T \rightarrow [0,1]$) $\rightarrow$ A function that takes functions over the domain X to $\{0,1\}$, all those functions represented by T, and maps them to a real number in the $[0,1]$ interval according to the distribution D and the labeling function f. This mapping process is done by comparing the labels obtained by a function $t \in T$ with the real labels given by f over the instances from D. Hence the loss function measures the probability of error of some $t \in T$ over D when compared to f. In our case, it can be understood as a function that computes the correctness of a set of rules to the game built by you.

One may ask what is the worth of the PAC-Learning definition. Its richness lies on the ability to formalize what we are trying to achieve as predictor using some algorithm and some context to do so for any distribution over our domain. Intuitively what PAC-Learning definition states is: given any distribution over a domain and some accuracy parameters, if, on a set of predictors that we previously knew contained the optimal one, we search for the best predictor to approximate that distribution, we are probably going to find it and, even better, the maximum number of examples needed to find it is computable.

Improving a little the definitions we have stated so far, we present some important results:

Corollary: Every finite hypothesis class is PAC-Learnable with sample complexity [1] [2]

$$m_H(\epsilon, \delta) \leq \lceil \frac{\log \frac{|H|}{\delta}}{\epsilon} \rceil$$

[1] The proof of this result can be found at [1]
[2] The result itself can also be found at [1]

This corollary is shown here because of its importance in translating the PAC-Learning model into an applicable situation. This very important result tells us that, if we maintain the Realizable Assumption over our hypothesis class, every finite set of these hypothesis classes is PAC-Learnable and, furthermore, we can tell how big our sample will need to be so we can guarantee the success of the learning process relative to the desired accuracy and confidence.

Taking us back to our tale, by understanding PAC-Learnability you just entered a prosperous field that can lead you to a first approach to the problem. You defined learning as choosing a set of rules that comfortably fit the task of describing victories or losses states on the various matches you have taken. Moreover, you realize that you have already played this game once in your lifetime and you are certain that, after playing a big - but finite - amount of matches, you are probably capable of describing the rules of the game, or at least a set of rules so close to the actual rules that you can't even distinguish between them only by analyzing the matches you have already played.

We now encourage the reader to jump ahead a little bit and explore the next section of this paper to take a good look at the PAC-Learning experiment done there and better understand the potential and limitations of this approach.

3.2 Agnostic PAC-Learning

Continuing to the next stage of this section, we relax some of the restrictions imposed for PAC-Learning to achieve a more general and realistic tool: the Agnostic PAC-Learning. For this method to work, we give up on the Realizable Assumption, which is typically difficult to build on a real problem, but we pay an expensive price in terms of sample complexity. Although the size of the samples grow much faster than the previous case, we can still achieve arbitrary accuracy with arbitrary probability.

In terms of our personal story, we are still dealing with the concept of learnability and how to approach it. This time, imagine you asked yourself the same questions as before and thought of the same strategy, but this time you have never heard of the game you are playing and you are completely on your own to create the rules you are going to apply to differentiate a victory state from a non victory state in the game.

Formalizing the approach and giving some intuition, we have:

> Definition of Agnostic PAC-Learnability: A hypothesis class H is Agnostic PAC-Learnable if there exists a funciton $m_H : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in (0,1)$ and for every distribution D over $X \times Y$, when running the learning algorithm on $m \geq m_H(\epsilon, \delta)$ i.i.d. examples generated by D, the algorithm returns a hypothesis h such that, with probability of at least $1-\delta$ (over the choice of the m training examples), $L_D(h) \leq min_{h' \in H} L_D(h') + \epsilon$[1].
>
> Co-domain $Y \to$ The set of labels of X states. In this case, since we are not assuming the existence of an optimal labeling function f anymore, the labels are given according to the distribution D,

therefore include some intrinsic probability. In our case, it can also be understood as the wise man decision about the state of the game - if it is terminal or not and whether it is a victory, a loss or a draw in the case of a terminal state.

Loss function ($L_D : T \to [0,1]$) $\to$ A function that takes functions over the domain X to $\{0,1\}$, all those functions represented by T, and maps them to a real number in the $[0,1]$ interval according to the distribution D. This mapping process is done by comparing the labels obtained by a function $t \in T$ with the real labels given by the distribution, not an optimal function f anymore. Hence the loss function measures the probability of error of some $t \in T$ over D when compared to D pairs of instances and labels. In our case, similarly to the PAC-Learning case, it can be understood as a function that computes the correctness of a set of rules to the game built by you.

Intuitively what Agnostic PAC-Learnability states is: given a certain number of examples of states and labels based on a distribution D over a domain X and its co-domain Y, if this number is large enough we can guarantee with arbitrary probability that we will make predictions arbitrarily close to the best possible predictor on the hypothesis class H.

Despite the simple definition of Agnostic PAC-Learnability, as it is very similar to the PAC-Learnability one, we still have to deal with other elements of our model to make sure it works. Differently from the former case of learning, we do not have any clues about the labeling optimal function. Hence, to choose a good predictor, one with a loss as little as possible, we can not rely on the fact that the labeling function will probably be chosen. To make a good prediction here, we will need some guarantees on the sample S we are choosing. Not only this method requires a lot of sample data, it also requires the sample to be representative to our context. Thus we introduce our next definition:

ϵ-representative Sample: A training set S is called ϵ-representative (with respect to domain X, hypothesis class H, loss function L, and distribution D) if $\forall h \in H, |L_S(h) - L_D(h)| \leq \epsilon$[1].

Loss function ($L_S : T \to [0,1]$) $\to$ This function is our classical loss function, but tested over the sample S, not the entire domain like L_D function.

What this definition brings is a formal characterization of the samples we can really learn and generalize from. Choosing an arbitrary accuracy, if we are working with a ϵ-representative sample we already know that a hypothesis from the hypothesis class will probably perform predictions on the sample as well or as bad as it would do if applied to all the domain over the distribution D. We now proceed to study methods of seeking and finding these kind of samples, but first we state one of the most important lemmas that follows from the last definition.

Lemma: Assume that a training set S is $\frac{\epsilon}{2}$-representative (with respect to domain X, hypothesis class H, loss function L, and distribution D). Then, any output of $ERM_H(S)$ algorithm, namely, any $h_S \in argmin_{h \in H} L_S(h)$, satisfies $L_D(h_S) \leq min_{h \in H} L_D(h) + \epsilon$[1].[3]

It follow from the previous lemma an important result: to achieve Agnostic PAC-Learnability we can simply show that, with arbitrary high probability over the random choice of training sets, the chosen S will be ϵ-representative. At this point, we can already proceed to our next definition.

Definition of Uniform Convergence: We say that a hypothesis class H has the Uniform Convergence Property (with respect to domain X and loss function L) if there exists $m_H^{UC} : (0,1)^2 \to \mathbb{N}$ such that for every $\epsilon, \delta \in (0,1)$ and for every probability distribution D over X, if S is a sample of $m \geq m_H^{UC}(\epsilon, \delta)$ examples drawn i.i.d. according to D, then, with probability of at least $1 - \delta$, S is ϵ-representative.[1]

Intuitively we are trying to guarantee that our models use ϵ-representative samples with high probability and, even more, we are looking for a sample complexity lower bound that ensures we reach these samples. Now we state the last two important corollaries of this sub-section so we can finally work on an Agnostic PAC-Learning model.

If a class H has the uniform convergence property with a function m_H^{UC} then the class is Agnostic PAC-Learnable with the sample complexity $m_H(\epsilon, \delta) \leq m_H^{UC}(\frac{\epsilon}{2}, \delta)$. Furthermore, in that case, the ERM_H paradigm is a successful Agnostic PAC-Learner for H.[1]

Let H be a finite hypothesis class, let X be a domain and let $L : H \times X \to [0,1]$ be a loss function. Then, H enjoys the uniform convergence property with sample complexity

$$m_H^{UC}(\epsilon, \delta) \leq \lceil \frac{\log \frac{2|H|}{\delta}}{2\epsilon^2} \rceil$$

Furthermore, the class is Agnostically PAC-Learnable using the ERM algorithm with sample complexity [4] [5]

$$m_H(\epsilon, \delta) \leq m_H^{UC}(\frac{\epsilon}{2}, \delta) \leq \lceil \frac{2 \log \frac{2|H|}{\delta}}{\epsilon^2} \rceil$$

These corollaries imply two direct conclusions: we can still safely work on our model and guarantee its predictor learnability, but we will suffer from the major impacts on the sample complexity lower bound when taking a lot of previous knowledge away from the context.

[3] The proof of this result can be found at [1]
[4] The proof of this result can be found at [1]
[5] The result itself can be found at Ibid.

Back to our tale, you have just learnt you can try any set of rules you want and, with the right and huge amount of examples, you will probably understand the quality of the rules you choose. What that means is if you try your set of rules long enough, you will probably find it's approximate potential to determine whether a state is a victory one or a non-victory one. Although you may still be a bit lost, you know now that with the right method and with the right amount of games played, you can face the challenge.

We again encourage the reader to jump ahead a little bit and explore the next section of this paper to take a good look at the agnostic PAC-Learning experiment done there and better understand the potential and limitations of the approach.

3.3 VC Dimension

We now proceed to a very important result in theoretical machine learning field. The ramifications discussed in this subsection brought up the possibility of studying infinite hypothesis classes as agnostic PAC-Learning problems and hence opened up the doors to much more realistic approaches to problems in general. By studying some properties of a particular hypothesis class, we can now calculate a special quantity that indicates the learnability of the class and bounds our sample complexity a little more pragmatically.

In terms of our story, this new result will allow you to get more creative when thinking about hypothesis class. Basically you can start using families of functions with low level features to try achieving better results in your task even without necessarily knowing much about the problem itself.

Imagine, for example, you tried your best to find out what were the rules of the game, but your made up rules can only reach up to fifty percent accuracy on determining whether a state is a victory or not. You are already getting tired, but still don't want to disappoint the wise man. Therefore, you decide to dramatically change your approach to the problem. You decide to assume you have not learnt anything about the problem at all and look at the game from a three by three matrix perspective. You also decide not to evaluate the quality of the states based on your own speculations, but to let a linear separator do the job for you. Although you probably won't be able to calculate this hyper-plane yourself, because of the VC dimension and with a little help from the computer, you can guarantee success on finding your desired separator.

Although the agnostic PAC-Learning itself gives more flexibility to the concept of learning and its possibilities, it is still a very hard constrain to have a finite hypothesis class. In the attempt to generalize learning a little bit more, Vapnik and Chervonenkis define what is called the Vapnik-Chervonenkis (VC) Dimension. It is defined as follows:

> Definition of VC Dimension: The VC-dimension of a hypothesis class H, denoted VC_{dim}, is the maximal size of a set $C \subset X$ that can be shattered by H. if H can shatter sets of arbitrary large size we say that H has infinite VC-dimension.[1]

To better understand what this means, we now proceed to the definitions of restriction and shattering:

> Definition of Restriction of H to C: Let H be a class of functions from X to $0, 1$ and let $C = C_1, ..., c_m \subset X$. The restriction of H to C is the set of functions from C to $0, 1$ that can be derived from H. That is, $H_C = (h(c_1), ..., h(c_m)) : h \in H$, where we represent each function from C to $0, 1$ as a vector in $0, 1^{|C|}$.[1]

> Definition of Shattering: A hypothesis class H shatters a finite set $C \subset X$ if the restriction of H to C is the set of all functions from C to $0, 1$. That is, $|H_C| = 2^{|C|}$.[1]

Now that we have all the definitions in mind, we begin to intuitively explain what VC dimension is. We start by explaining shattering. When we say a hypothesis class H shatters a finite set $C \subset X$, we want to express that, with the correct set of $h \in H$, we can generate any combination of classes for the $c_i \in C, i = 1, ..., m$ points. Since we are defining shattering and working on a binary classification context, this means we can generate all the combinations of classes $0_1...0_m, 0_1...1_m, ..., 1_1...1_m$ for the points in C. From this binary perspective, it is easier to see that it means we can generate up to $2^{|C|}$ combinations.

When we talk about finite VC dimension, hence, we are talking about the maximal number of points that can be perfectly described by a chosen hypothesis class H. Although it may seems that the VC dimension has nothing to do with machine learning, it is one of the most powerful tools to describe the expressiveness of a hypothesis class. When we are able to determine the VC dimension of a hypothesis class, we can better understand its limitations conceptually and we can even calculate its sample complexity in an applied context - using our typical ERM paradigm.

By the fundamental theorem of statistical learning - quantitative version - we state:

> Let H be a hypotheis class of functions from a domain X to $0, 1$ and let the loss function be the 0-1 loss. Assume that $VC_{dim}(H) = d < \infty$. Then there are absolute constants C_1, C_2 such that:
>
> (a) H has the uniform convergence property with sample complexity $C_1 \frac{d+\log \frac{1}{\delta}}{\epsilon^2} \leq m_H^{UC}(\epsilon, \delta) \leq C_2 \frac{d+\log \frac{1}{\delta}}{\epsilon^2}$
>
> (b) H is agnostic PAC learnable with sample complexity $C_1 \frac{d+\log \frac{1}{\delta}}{\epsilon^2} \leq m_H(\epsilon, \delta) \leq C_2 \frac{d+\log \frac{1}{\delta}}{\epsilon^2}$
>
> (c) H is PAC learnable with sample complexity $C_1 \frac{d+\log \frac{1}{\delta}}{\epsilon} \leq m_H(\epsilon, \delta) \leq C_2 \frac{d \log \frac{1}{\epsilon} + \log \frac{1}{\delta}}{\epsilon^2}$
>
> [1]

As can be seen in the theorem above, we can go back to our sample complexity evaluation even on problems with infinite hypothesis class. As much as

this result may seem impressive, it still does not solve our overestimated bounds for sample complexity problem, as the last experimental subsection will show.

Once again back to our story, from this point on, the machine learning paradigm completely changed and your job is not to find the solution anymore, but to think of ways to translate your data to a computer so it can optimize whatever finite VC dimension hypothesis set you choose. You have just understood the very nature of what modern machine learning algorithms are and you are probably ready to overcome the challenge, to thank Itapicuru's wise man and go home, since your vacation is likely to be over by now.

4 The Experiments

We now proceed to the experiments themselves. This section is intended to help the reader to better understand the theoretical concepts previously explained by modeling real agents to solve practical and simple tasks. The main problem approached will be to decide whether a state of the board is a win state or a non-win state on the Tic-Tac-Toe game. Details about the model built, the implementation itself and small variations to the initial problem will be described in each of the subsections.

At the end of the first three experiments, related to PAC-Learning, Agnostic PAC-Learning and VC Dimensions respectively, we added some extra experiments to make use of more modern machine learning techniques without necessarily formalize them, but trying to intuitively link the theory we have studied so far to these approaches.

4.1 PAC-Learning Experiment

For our first experiment, we will model the classic Tic-Tac-Toe game as a PAC-Learnable problem. When modeling any artificial intelligence problem, one needs to initially establish what are the goals - what is the agent trying to predict. In our case, as we are working on a toy problem, we will simplify as much as we can the predictor. The objective here is to decide whether a given state is a victory for player one - namely "X" - or is not a victory state for that same player - notice we are not differentiating losses from draws.

As we have already decided what our agent is learning, we define our domain, which will be all the possible three by three matrices on a Tic-Tac-Toe game. In our version of the board, -1 indicates an empty place, 0 indicates a place controlled by player 2 ("O"), and 1 indicates a spot controlled by player 1 ("X"). We now proceed to build an hypothesis class by using our prior knowledge of the environment - in this case, the game itself and its rules. For this construction we will simply enumerate some predicates and our hypotheses will be all the possible disjunctions of these heuristics. We define:

1. Determine whether the state has a horizontal line of "X"'s;
2. Determine whether the state has a vertical line of "X"'s;

3. Determine whether the state has a diagonal line of "X"'s;

4. Determine whether the state has two "X"'s together somewhere in the map;

5. Determine whether the state has a "X" in the center spot of the board;

More precisely about the hypotheses, assume that the hypothesis class H is the set of all h's of the form $h = a_1a_2a_3a_4a_5$ where each a_i is a binary digit indicating the presence or absence of the respective predicate on the disjunction - $h = 10010$ or $h = P_1 \vee P_3$ indicates the function that maps a state of the board on 1 if it contemplates the first or the third predicates. It's important to notice that the choice of the predicates was not arbitrary. From our prior knowledge, we know that $h = 11100$ is the true labeling function, supporting the Realizable Assumption necessary to use PAC-Learning approach. Moreover, we chose the last two heuristics in a way the hypotheses that contained them wouldn't be that much of a bad choice so the predictor can get confused sometimes. Finally, by construction, we can tell $\mid H \mid = 2^5 = 32$, since all of its functions are represented by five binary digits.

For the distribution D, we could pick anyone we wanted to, since the PAC-Learning approach works for all of them. We chose one given by the code below. it simulates a game been randomly played and it has a small chance of stopping the game before it reaches a terminal state. We made use of a less intuitive distribution only for exemplification purpose. In the last subsection, when experimenting with the VC analyses, we go back to a more natural uniform distribution.

```
int terminal = terminal_state(tab);
while(terminal == -2) {
    int aux_i = rand()%3;
    int aux_j = rand()%3;
    while(tab[aux_i][aux_j] != -1) {
        aux_i = rand()%3;
        aux_j = rand()%3;
    }
    tab[aux_i][aux_j] = jogador;
    jogador = 1 - jogador;
    terminal = terminal_state(tab);
    int r = rand()%100;
    if(r == 0)
        break;
}
```

A distribution that fills the game board. It tries to simulate a game been played randomly, but with a small probability of stopping before the terminal state is reached.

OBS: *tab* is a 3 by 3 matrix containing the state of the board. *terminal_state(tab)* is a function that returns -2 if the game isn't over yet, -1 if it's a draw, 0 if the player 2 wins ("O"), and 1 if the player 1 wins ("X"). *jogador* is a variable indicating the player of the turn.

Now that we have a model, we can finally choose some accuracy and confidence parameters to test the functionality of the learner with ERM algorithm and also study the sample complexity given by the approach. For tradition, we choose $\epsilon = 0.1, \delta = 0.1$, which means we want $P[L_{D,f}(h) \leq 0.1] \geq 0.9$. We now calculate $m_H(\epsilon, \delta)$:

$$m_H(0.1, 0.1) \leq \lceil \frac{\log \frac{|H|}{\delta}}{\epsilon} \rceil = \lceil \frac{\log \frac{2^5}{0.1}}{0.1} \rceil = 26$$

Now that we have an upper bound for $m_H(\epsilon, \delta)$ that guarantees the success of the method, we proceed to the experiment itself. The experiment consist on running the program ten thousand times with a certain sample size m. We then graph the results, showing how many times the algorithm got the labeling function right - or at least with error smaller than ϵ - and judging the bound we calculated before. Notice that the numbers associated with the X axis of the subplots are the decimal representation of an hypothesis h - suppose $h = 00101$, then the number associated with it would be 5, for example.

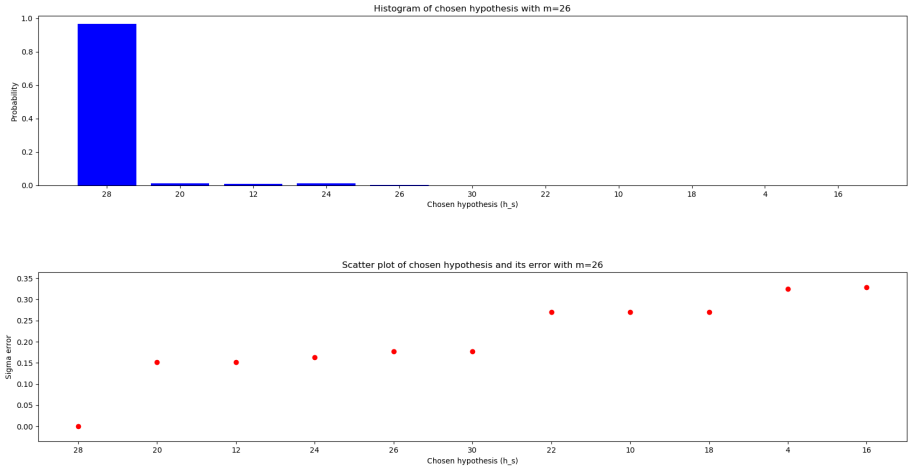


Figure 1: PAC-Learning experiment based on our built hypothesis class H with $m = 26$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.965$, which satisfies our imposed (ϵ, δ) pair.

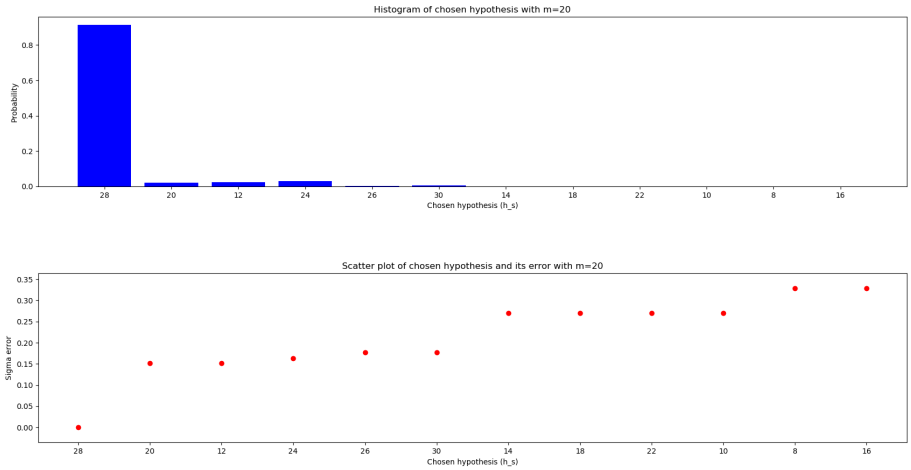


Figure 2: PAC-Learning experiment based on our built hypothesis class H with $m = 20$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.913$, which still satisfies (ϵ, δ) pair.

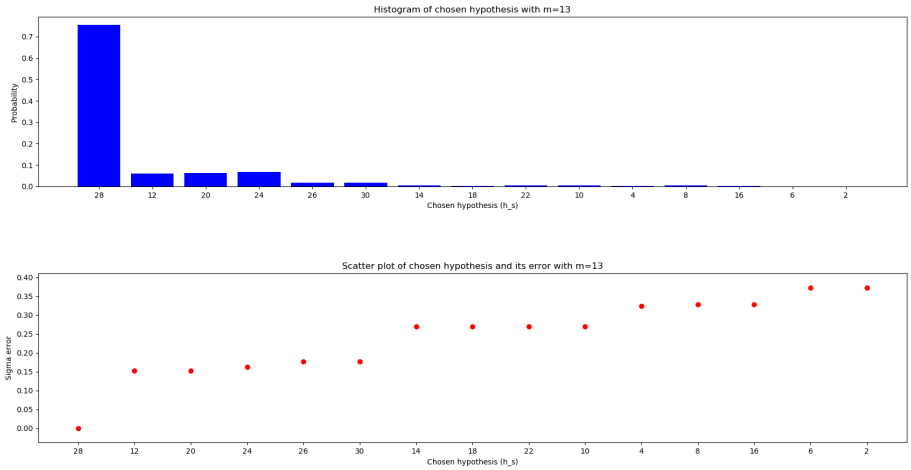


Figure 3: PAC-Learning experiment based on our built hypothesis class H with $m = 13$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.75$, which does not satisfy (ϵ, δ) pair anymore.

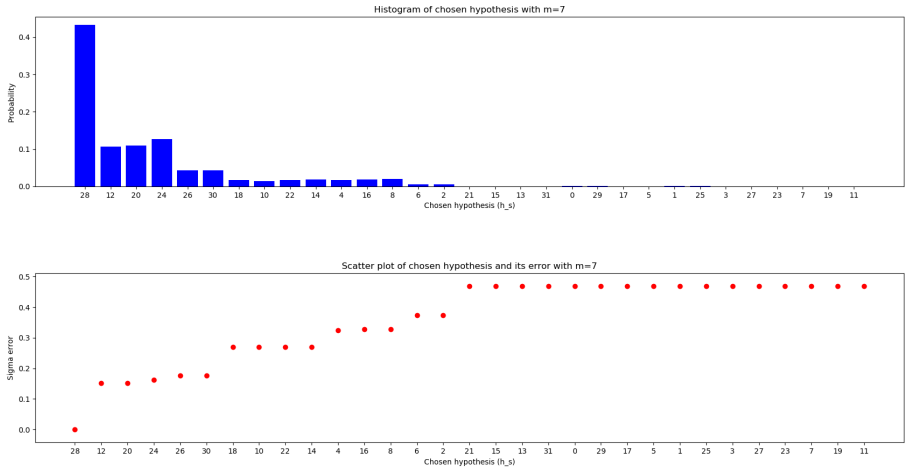


Figure 4: PAC-Learning experiment based on our built hypothesis class H with $m = 7$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.46$, which is not even close to satisfying our (ϵ, δ) pair anymore.

From this first experiment, therefore, we can observe the strengths and the weaknesses of the PAC-Learning model combined with ERM algorithm. For the strengths, one can intuitively notice that they mostly reside in the guarantees given by the approach. Now for the weaknesses, one can directly observe that they mainly reside on the absurd upper bound given by the same approach. Through our study case, particularly from the second figure, we notice that the desired accuracy was achieved with probability as high as we wanted with a sample size about 24% lower than the calculated sample size suggested by the PAC-Learning theory. Moreover, we almost got results as good as we wanted with sample size 50% lower than the calculated $m_H(0.1, 0.1)$. When talking about real world problems with much bigger hypothesis classes, using PAC-Learning model to get some guarantees could mean choosing sample sizes orders of magnitude bigger than what is really needed for a good predictor to learn.

4.2 Agnostic PAC-Learning Experiment

We now proceed to our second experiment, the agnostic PAC-Learning one. For our experiment to be agnostic, not having full information about the quality of our hypothesis class would be enough, since this kind of model only guarantees results as good as we can get from any biased or non biased H. But as we are working on a toy problem, for which is easy to see that our current H has the Realizable Assumption fulfilled, we change a little our hypothesis class to be similar to our former one, but with one less predicate:

1. Determine whether the state has a horizontal line of "X"'s;

2. Determine whether the state has a vertical line of "X"'s;

3. (Removed) ~~Determine whether the state has a diagonal line of "X"'s~~;

4. Determine whether the state has two "X"'s together somewhere in the map;

5. Determine whether the state has a "X" in the center spot of the board;

Removing the diagonal line predicate is abdicating one of the rules of Tic-Tac-Toe. As simple as it is, we have just made our problem agnostic.

Furthermore, we will use the same distribution D as before to draw our examples and the same pair $(\epsilon = 0.1, \delta = 0.1)$ as are desired parameters. Finally for our sample upper bound, considering the uniform convergence property, we have:

$$m_H^{UC}(\epsilon, \delta) \leq \lceil \frac{\log \frac{2|H|}{\delta}}{2\epsilon^2} \rceil = \lceil \frac{\log \frac{2*16}{0.1}}{2 * 0.1^2} \rceil = 1154$$

Now that we have an upper bound for $m_H^{UC}(\epsilon, \delta)$ that guarantees the success of the method, we proceed to the experiment itself. The experiment, as the former one, consist on running the program ten thousand times with a certain sample size m. We then graph the results, showing how many times the algorithm got the labeling function as good as it can be considering our new hypothesis class H and judging the bound we calculated before. Notice that the number associated with each bar on the first subplot of a figure or with each dot on the second subplot of a figure is the decimal representation of an hypothesis h, just like in the other experiment.

For the first value of m, $m = 1154$ as indicated by our previous calculation, we do not graph the results since we got $h_S = 12$, in binary $h_S = 1100$, for all the algorithm runs, which means best case scenario for every test. It is important to notice that this is probably a sign of a overestimated sample size, as we discuss later in this section.

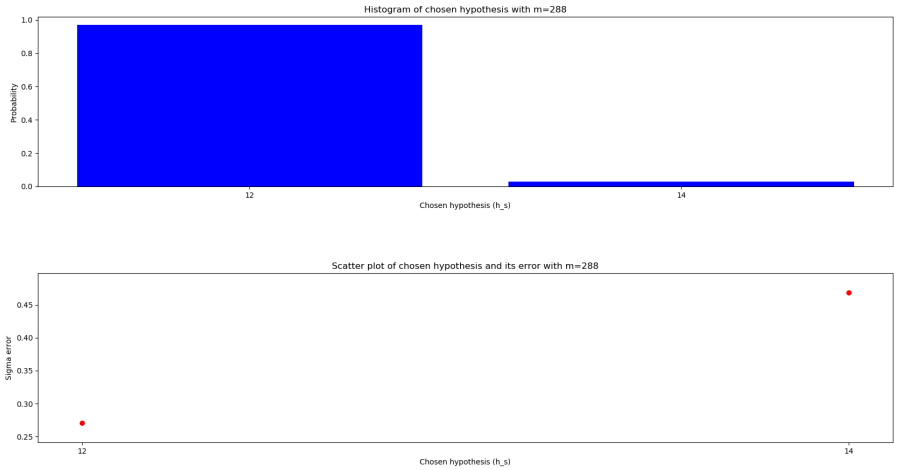


Figure 5: Agnostic PAC-Learning experiment based on our built hypothesis class H with $m = 288$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.032$, which is still in our imposed (ϵ, δ) pair.

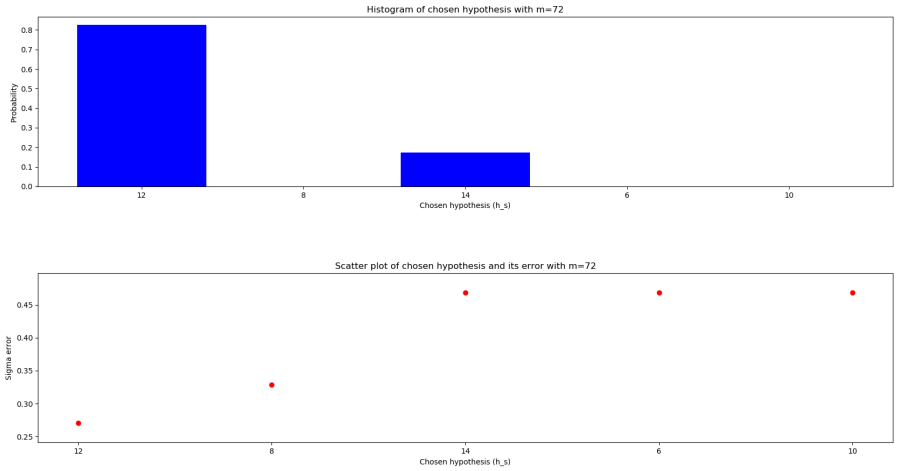


Figure 6: Agnostic PAC-Learning experiment based on our built hypothesis class H with $m = 72$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.18$, which is a little outside our imposed (ϵ, δ) pair.

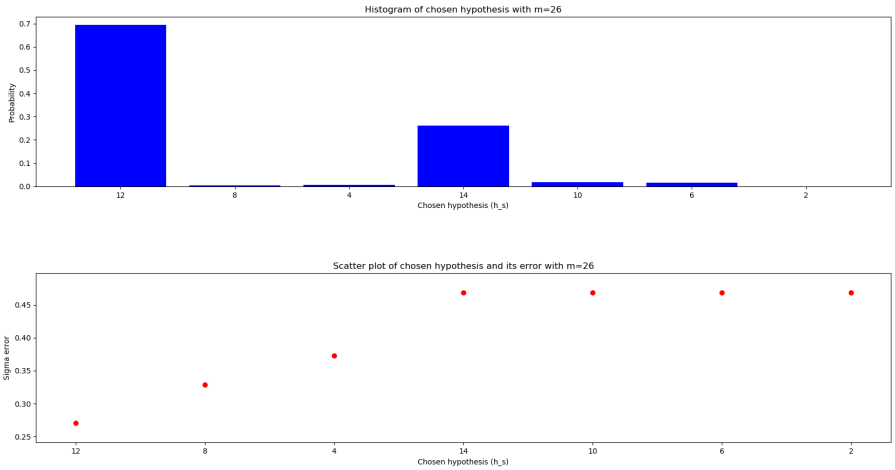


Figure 7: Agnostic PAC-Learning experiment based on our built hypothesis class H with $m = 26$. For this experiment, we got $P[L_{D,f}(h_S) \leq 0.1] = 0.31$, which is already much more outside our imposed (ϵ, δ) pair.

From the second experiment we only ratify the strengths and weaknesses of the PAC-Learning model combined with ERM algorithm already seen from the former investigation. Here we can observe an completely absurd estimate to the sample size needed for learning the better hypothesis in H. We could still learn within the imposed parameters (ϵ, δ) with $m < m_H^{UC}/4$, and we almost achieved the same thing with $m < m_H^{UC}/16$. On the other hand, we could still learn a good hypothesis from H - or at least the best it can give us - without the Realizable Assumption and independently from the distribution.

Now that we know for a fact we can learn from hypothesis classes regardless the Realizable Assumption and, therefore, without explicitly knowing the perfect and desired predictor, we can get a little more creative and try more complex hypothesis classes. From now on, we enter a completely different world called Bias-Complexity Trade Off in which, as the name already explains, we can explore different kinds of hypothesis spaces and trade off the sample complexity for a simpler and probably very biased H or much more complex and subtle hypothesis for a bigger sample complexity.

4.3 Linear Halfspaces Experiment

Although we can get much more creative than that, we start from the basics and try a linear halfspaces hypothesis class with a perceptron algorithm as our learner.

For this experiment we first go back our Realizable Assumption. In terms of halfspaces, it is understood in a more geometric form, though. We say the

Realizable Assumption is fulfilled if our sample points are linearly separable in terms of the features taken into consideration. Hence we model our next agent in a way one can understand it as a linear halfspaces classifier for linearly separable points in space.

We begin by completely rebuilding our Hypothesis Class H. Here we consider every hyperplane in the feature space as a hypothesis of our hypothesis class. We now proceed to the feature space. For this first realizable example, we define our feature space as:

1. f_1 is determined by the presence or absence of a horizontal line of "X"'s;
2. f_2 is determined by the presence or absence of a vertical line of "X"'s;
3. f_3 is determined by the presence or absence of a diagonal line of "X"'s;

The three features used here are binary - 0 in case of absence and 1 in case of presence. This gives us a total of eight possible points in space, although we are working in $\mathbb{R}^3$.

To be a little more explicit about what we are going to do in this experiment, we will evaluate every given terminal state of the game by these three features and classify them into victory for player X - the positive examples - and non victory for player X - the negative examples. Before showing the obtained agents though, we talk about boundaries as usual. For linear halfspaces like ours we use a lower bound of $\Omega(\frac{d+\log(\frac{1}{\delta})}{\epsilon})$ as shown in [2]. However, for this example, we have an upper bound to state as well and this will be $O(\frac{d\log(\frac{1}{\delta})}{\epsilon})$ as shown in [3]. Hence we have:

$$\Omega(\frac{d+\log(\frac{1}{\delta})}{\epsilon}) \leq m_H(\epsilon,\delta) \leq O(\frac{d\log(\frac{1}{\delta})}{\epsilon})$$

$$K_1(\frac{3+\log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1,0.1) \leq K_2(\frac{3\log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 53 \leq m_H(0.1,0.1) \leq K_2 * 69$$

Now that we already have our lower and upper bound in hands, we proceed to algorithmic specification. In this experiment we will make use of the Perceptron algorithm. Although it is a simple algorithm, we will use the Scikit Learn implementation, primarily because of its optimizations. We now show the agents obtained:

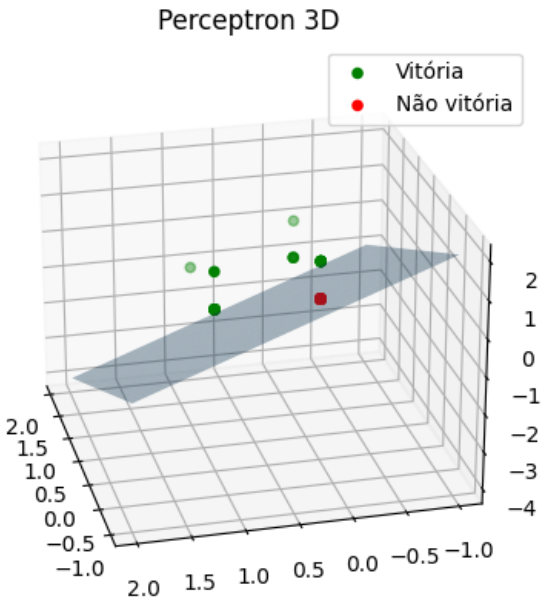
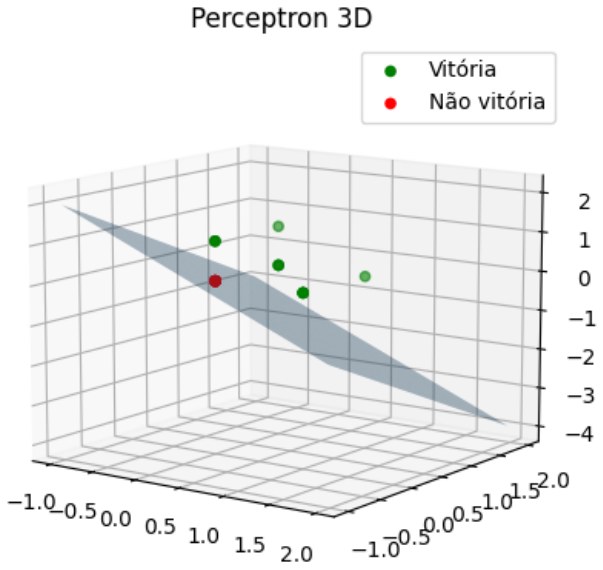


Figure 8: Perceptron experiment based on our built hypothesis class H with $m = 53$. We show here an example of an obtained agent. In this case we got $P[L_{D,f}(h_S) \geq 0.1] = 0$ for every test, which is in our imposed (ϵ, δ) pair and a perfect score.



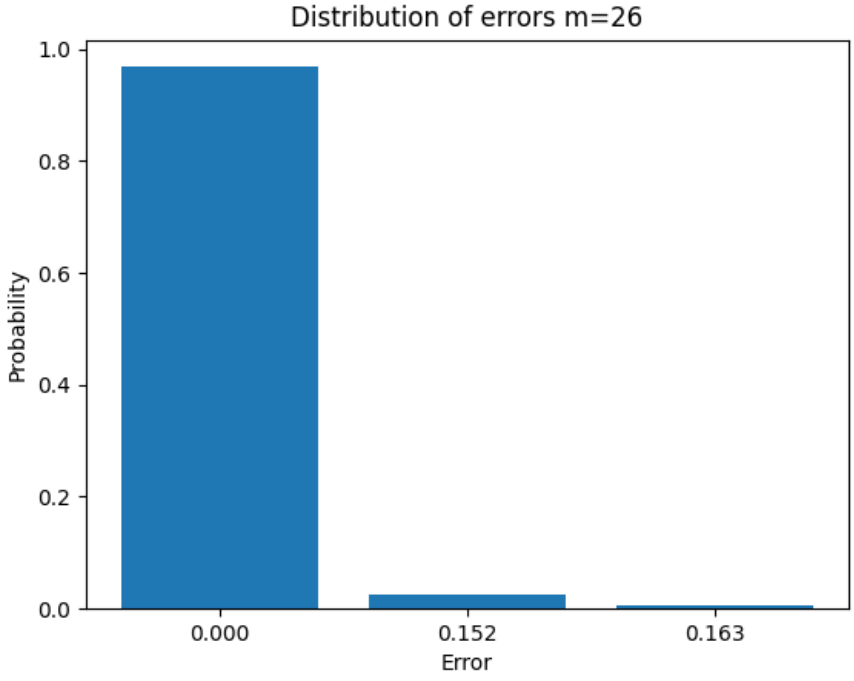
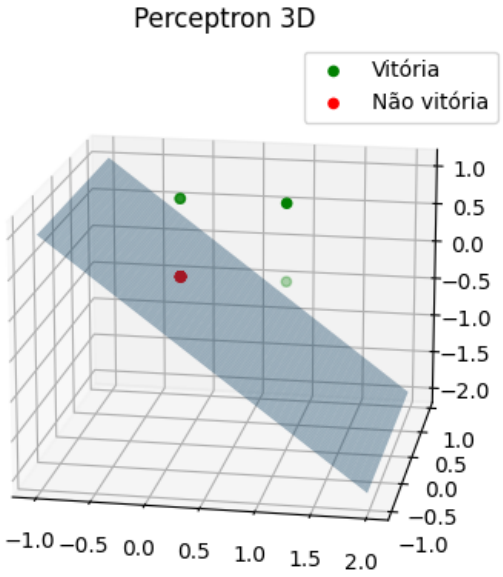


Figure 9: Perceptron experiment based on our built hypothesis class H with $m = 12$. We show here an example of an obtained agent. In this case we got $P[L_{D,f}(h_S) \geq 0.1] = 0.03$ which still is in our imposed (ϵ, δ) pair. We also show the distribution of error for the obtained agents within one thousand tests.



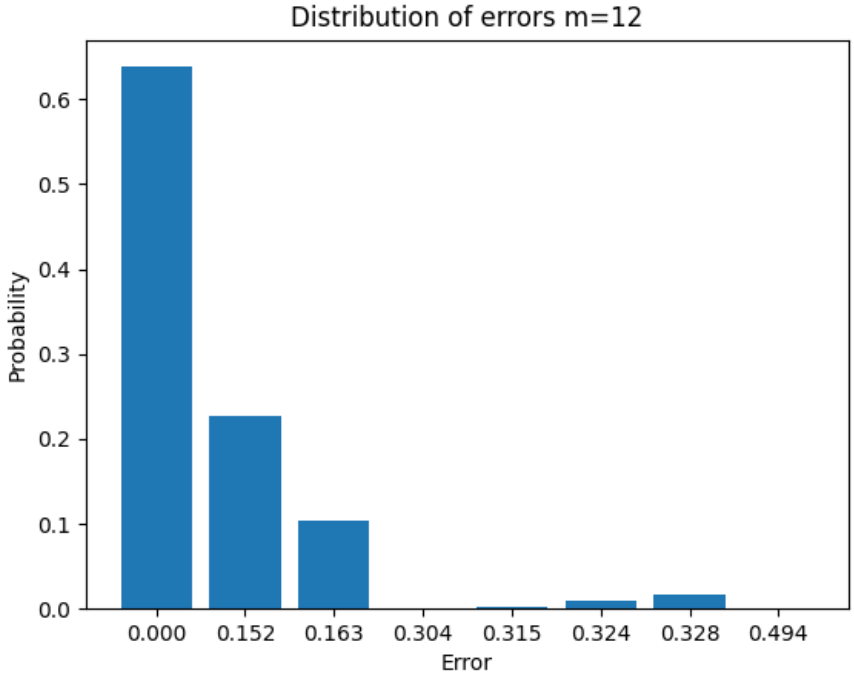


Figure 10: Perceptron experiment based on our built hypothesis class H with $m = 12$. We show here an example of an obtained agent. In this case we got $P[L_{D,f}(h_S) \geq 0.1] = 0.36$ which is not in our imposed (ϵ, δ) pair anymore. We also show the distribution of error for the obtained agents within one thousand tests.

As we can observe through the obtained agent, once again this is a simple task that requires a lot less than the minimum 53 sample points given by the lower bound to train a good agent. On the other hand, we used a completely biased feature space that made our task too much of an easy one. Now as we advance our techniques we will also make the agent life a little harder by taking away the amount of previous knowledge put into the problem formulation.

4.4 Extra Experiments

For this last experiment, we make use of more modern techniques to both improve our agent performance with even less previous knowledge of the environment and approximate cutting-edge technology to the theoretical machine learning frameworks. We begin by analyzing Support Vector Machine(SVM) agents and studying their VC dimensions. We than proceed to analyze a simple neural network approach and also to study its VC dimension.

4.4.1 Support Vector Machines

Before using the SVM, we explain how it works on a high level. In the linear separable case, the SVM approach tries to linear separate data with the biggest margin possible according to the nearest points of different classes, the support vectors. For the non separable cases, the SVM uses a loss that tries to minimize the mislabeled points, but also tries to maximize the margin between the chosen support vectors, maintaining the high probability of correct classification even if not perfect. Although the SVM approach seems to be a slightly optimized version of the Perceptron algorithm, it is much more than that when combined with the kernel trick, a set of non linear transformations to the feature space used to increase its dimension and make data more linearly separable. Since we are talking about linear separability in the feature space, it is already natural for us to think about VC dimension. Therefore, for the SVM experiment, we calculate the VC bound based on the feature space dimensions for a few different transformations, evaluate these bounds and compare them to another VC bound based on the number of support vectors found.

Using the previous linear halfspaces bound, we calculate:

1. Linear Kernel: 10 dimensions and 784 support vector on average. For this kernel, the SVM classifier can reach up to 0.67 accuracy, so 0.33 error.

$$\Omega(\frac{d+\log(\frac{1}{\delta})}{\epsilon}) \leq m_H(\epsilon,\delta) \leq O(\frac{d\log(\frac{1}{\delta})}{\epsilon})$$

$$K_1(\frac{10+\log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1,0.1) \leq K_2(\frac{10\log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 124 \leq m_H(0.1, 0.1) \leq K_2 * 231$$

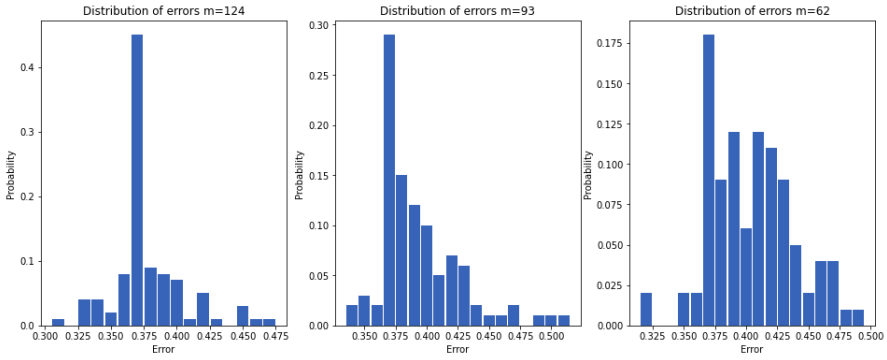


Figure 11: SVM experiments with $m = 124, m = 93, m = 62$. In these cases we got $P[L_{D,f}(h_S) \geq 0.1] = 0.95$ which is in our imposed (ϵ, δ) pair, $P[L_{D,f}(h_S) \geq 0.1] = 0.91$ which is still in our imposed (ϵ, δ) pair, and $P[L_{D,f}(h_S) \geq 0.1] = 0.83$ which is slightly out of our imposed (ϵ, δ) pair. We show the distributions of error for the obtained agents within one hundred tests each.

2. Polynomial Kernel (degree 2): 55 dimensions and 447 support vector on average. For this kernel, the SVM classifier can reach up to 0.89 accuracy, so 0.11 error.

$$K_1(\frac{55 + \log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1, 0.1) \leq K_2(\frac{55 \log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 574 \leq m_H(0.1, 0.1) \leq K_2 * 1267$$

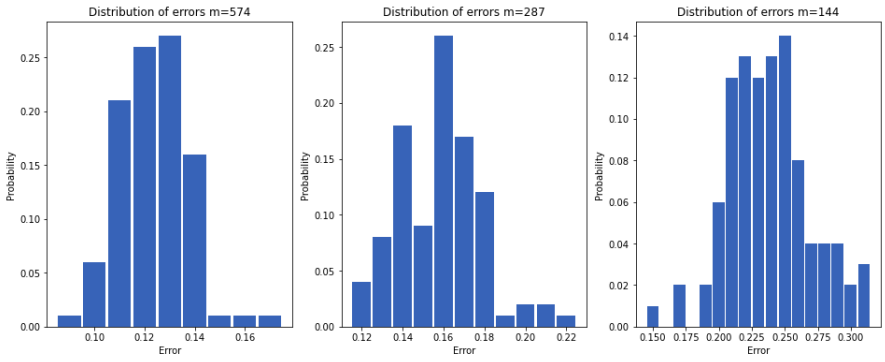


Figure 12: SVM experiments with $m = 574, m = 278, m = 144$. In these cases we got $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is in our imposed (ϵ, δ) pair, $P[L_{D,f}(h_S) \geq 0.1] = 0.97$ which is still in our imposed (ϵ, δ) pair, and $P[L_{D,f}(h_S) \geq 0.1] = 0.11$ which is completely out of our imposed (ϵ, δ) pair. We show the distributions of error for the obtained agents within one hundred tests each.

3. Polynomial Kernel (degree 3): 220 dimensions and 188 support vector on average. For this kernel, the SVM classifier can already reach up to 1.00 accuracy, no error.

$$K_1(\frac{220 + \log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1, 0.1) \leq K_2(\frac{220 \log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 2224 \leq m_H(0.1, 0.1) \leq K_2 * 5606$$

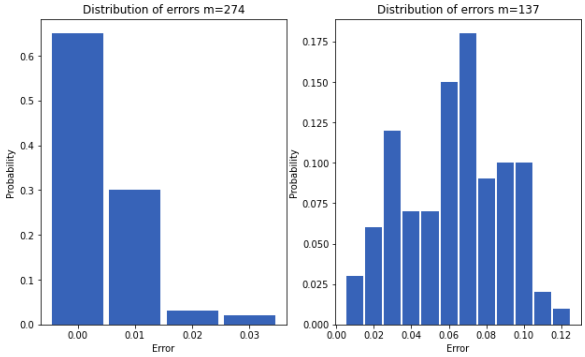


Figure 13: SVM experiments with $m = 2224, m = 274, m = 137$. In these cases we got $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is in our imposed (ϵ, δ) pair, $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is still in our imposed (ϵ, δ) pair, and $P[L_{D,f}(h_S) \geq 0.1] = 0.87$ which is slightly out of our imposed (ϵ, δ) pair. We show the distributions of error for the second and third obtained agents within one hundred tests each. The first one distribution is not shown because it got zero error for every attempt.

4. Polynomial Kernel (degree 4): 715 dimensions and 179 support vector on average. For this kernel, the SVM classifier can already reach up to 1.00 accuracy, no error.

$$K_1(\frac{715 + \log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1, 0.1) \leq K_2(\frac{715 \log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 7174 \leq m_H(0.1, 0.1) \leq K_2 * 16464$$

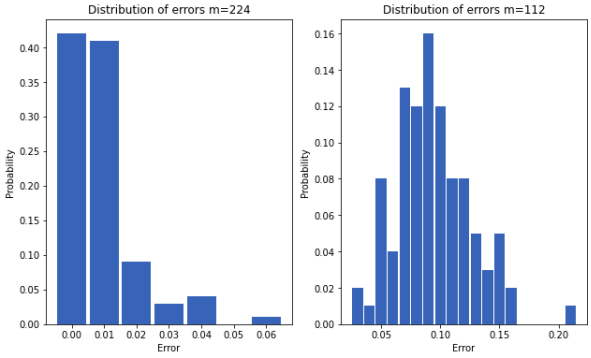


Figure 14: SVM experiments with $m = 7174, m = 224, m = 112$. In these cases we got $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is in our imposed (ϵ, δ) pair, $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is still in our imposed (ϵ, δ) pair, and $P[L_{D,f}(h_S) \geq 0.1] = 0.56$ which is completely out of our imposed (ϵ, δ) pair. We show the distributions of error for the second and third obtained agents within one hundred tests each. The first one distribution is not shown because it got zero error for every attempt.

5. RBF Kernel: infinite dimensions (no VC bound) and 366 support vector on average. For this kernel, the SVM classifier ccontinues to reach up to 1.00 accuracy, no error.

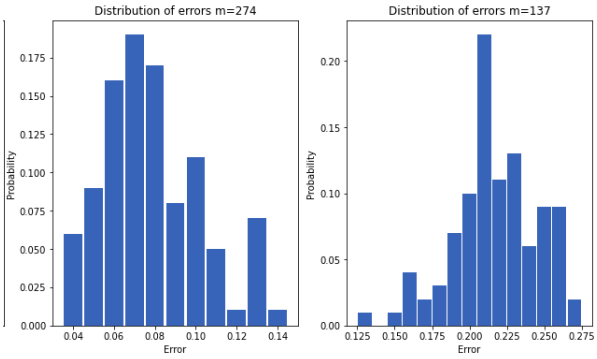


Figure 15: SVM experiments with $m = 2224, m = 274, m = 137$. In these cases we got $P[L_{D,f}(h_S) \geq 0.1] = 1.00$ which is in our imposed (ϵ, δ) pair, $P[L_{D,f}(h_S) \geq 0.1] = 0.75$ which is slightly out of our imposed (ϵ, δ) pair, and $P[L_{D,f}(h_S) \geq 0.1] = 0.00$ which is completely out of our imposed (ϵ, δ) pair. We show the distributions of error for the second and third obtained agents within one hundred tests each. The first one is distribution is not shown because it got zero error for every attempt.

As always, one can observe that the VC dimension sample complexity is over estimated by a lot. The more we grow the degree of the kernel polynomial

of our classifier, the more over estimated the sample complexity get. Another interesting fact to notice is the number of support vectors. The more we grow the degree of the kernel polynomial, the less we need support vectors to linear separate our data. When reaching the linear separable point, though, the number of support vectors stops dramatically decreasing. This phenomenon can be understood as the redundancy of information when the number of dimensions grow much higher than what it need to be to linear separate the data.

Still thinking about sample complexity, it is important to observe that, once the linear separable dimension is reached, the sample complexity bound given by the number of support vectors - the number of parameters really used to separate data - starts to better approximate the sample size needed for learning within our imposed (ϵ, δ) pair. This information can be very useful when dealing with large degree polynomial kernels, once simply changing the VC dimension by the number of support vectors when calculating the sample complexity bound can decrease its value by a lot.

Furthermore, the experiment elucidates the price to pay when using RBF kernel. Although it is a more general approach than the finite polynomial kernels, it can also be less appropriate to use with a small amount of data. The third degree polynomial, the first one that linear separates our data, for example, was able to perform much better than the RBF with $m = 137$.

4.4.2 Neural Networks

We now proceed to analyze the neural network approach. The idea behind the famous neural networks is to apply non-linear transformations to linear combinations of the previous layers, resulting in a series of non-linear separators that evaluate different aspects of the input. These final separators are then weighted to indicate their importance to the classes discrimination. Although it seems like an easy, plug and play approach, a lot can go wrong in the process of optimizing the linear combination weights, which lead to catastrophic classifiers regardless the robustness of the method.

As neural networks already make use of non-linear transformations, and a huge amount of them stacked up, studying their VC dimension can be a tough task. Therefore, for this experiment, we will use a much more intuitive approach, approximating the VC dimension by the number of parameters to optimize. Consider a simple network given by:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	640
dense_1 (Dense)	(None, 128)	8320
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 2)	130

Total params: 33,858
Trainable params: 33,858

Figure 16: Neural network model summary.

This is a small network containing only dense fully connected layers and a dropout layer to help with regularization. As much as our network may be tiny, when we calculate its sample complexity bounds given by:

$$K_1(\frac{33858 + \log(\frac{1}{0.1})}{0.1}) \leq m_H(0.1, 0.1) \leq K_2(\frac{33858 \log(\frac{1}{0.1})}{0.1})$$

$$K_1 * 338604 \leq m_H(0.1, 0.1) \leq K_2 * 779610$$

We immediately notice that the sample complexity is so overestimated that our problem doesn't even have this number of different states. When testing the network, it is possible to observe that agents with a thousand tests or more - much less states than what its VC bound asks for - are already able to perfectly distinguish a victory state from a loss state.

5 Conclusions

Once the fundamental concepts of PAC-Learning and Agnostic PAC-Learning are properly introduced and formalized, we can finally start the discussion about the positive and negative aspects of these techniques while differentiating both. By remembering their weaknesses and extolling their richness, we explain the reason of this article itself.

There are many reasons to talk about machine learning in general. The theoretical machine learning field is only one aspect of a big range of possibilities in the area. Its importance resides on the subtle and deep potential of better understanding what is going on behind the algorithms. The PAC-Learning approach, as one of the most important machine learning theories in the nineties, was the starting point to modern theoretical machine learning.

Beyond the intuitive premise of the framework, the specially strong points of the PAC-Learning theory are the simple formalization and mainly the possibility to bound sample complexity needed for learning independently from the distribution we are working with. This description exhibits the powerful generalization of the technique, which implies a wide range of applicability.

Although it seems at first that PAC-Learning could solve all of our problems in machine learning, modeling a PAC-Learnable hypothesis class and all other specifications needed for this tool to work can be a tough task. And even if we were able to build our model properly, we would still get a completely overdone sample complexity bound, which makes it hard to develop a scalable model.

When talking about Agnostic PAC-Learning, we benefit from the fact that we are not counting on the Realizable Assumption anymore, one of the most difficult requirements to achieve when modeling a problem. On the other hand, we lose a lot in terms of sample complexity and in terms of results as well. The already overdone sample complexity bound of the PAC-Learning model escalates much faster in this case, which only aggravates a problem that was already huge. Moreover, when giving up on the Realizable Assumption, we can guarantee the quality of the results only with respect to the chosen hypothesis class, which can be a disaster if we do not make use of very rich and precise prior knowledge. Briefly, by transitioning a model from PAC-Learning to Agnostic PAC-Learning we make our system as comprehensive as it is computationally difficult.

The VC dimension theory, although it doesn't solve the sample complexity issue, changes the machine learning paradigm a bit. Prior knowledge becomes a feature to our models, but not a necessity anymore. Beyond that, being capable of including infinite hypothesis classes to our models is what really makes practical scenarios learnable and, therefore, what links theoretical machine learning to cutting edge machine learning technologies. The extra experiments themselves show how these fields can be used together to enrich machine learning development and research.

The appeal and value of this paper, therefore, rely on better understanding the fundamentals of machine learning and trying to approximate modern machine learning techniques to the PAC-Learning field. Since we have achieved a new level of computational potential, these once obsolete methods emerge, not with the ambition to overcome all the contemporary machine learning approaches, but with the promise of better and deeply understand what is happening in the core of the most modern statistical frameworks.

Lastly, we really hope you enjoyed your trip to Itapicuru and learnt a bit with Itapicuru's wise man experiment, after all knowledge is a gift worth sharing. We are looking forward to your next vacation.

6 References

1. BEN-DAVID, S., SHALEV-SHWARTZ, S. **Understanding Machine Learning from Theory to Algorithms**. Cambridge University Press,

2014.

2. LONG, M. P. **On the sample complexity of PAC learning halfspaces against the uniform distribution**. Research Triangle Institute, November 1995

3. HAUSSLER, D., LITTLESTONE, N., WARMUTH, M. K., **Predicting {0, 1}-functions on randomly drawn points**, Information and Computation 115, 1994

7 Complementary Bibliography

1. VALIANT, L. G. **Probably approximately correct**. Basic Books. 2013.

2. VALIANT, L. G. **A Theory of the Learnable**. Cambridge, Massachusetts. Communications of the ACM. November 1984.

3. VALIANT, L. G. **Knowledge Infusion**. Cambridge, Massachusetts. AAAI. July 2006.

4. MICHAEL, L.; VALIANT, L. G. **A First Experimental Demonstration of Massive Knowledge Infusion**. Cambridge, Massachusetts. AAAI. September 2008.

5. KHARDON, R.; ROTH, D. **Learning to Reason**. Cambridge, Massachusetts. AAAI. 1994.

6. SHOHAM, Y.; LEYTON-BROWN, K. **Multiagent Systems**. Cambridge University Press. 1a edição. December 2008.

7. MIAO, X.; LIAO, L. **VC-Dimension and its Applications in Machine Learning**. 2003.

8. LONG, P. M. **An upper bound on the sample complexity of PAC-learning halfspaces with respect to uniform distribution**. Singapore. Information Processing Letters 87. April 2003.

9. GÉRON, A. **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow**. O'REILLY. Junho de 2019.