Chapter 1

Supervised learning

1.1 Introduction to supervised learning

The course focus on *Supervised Learning* in which a learner has access to a dataset $D_n = ((X_1, Y_1), \dots, (X_n, Y_n))$ of feature/label couples. The features can be thought of as what can be observed on the field, the input, and the labels can be thought of as what we want to predict, the output.

Examples:

- The features can be an MRI of the head of a person and the target whether the person has Alzheimer or not.
- The features can be the caracteristics of a field in a farm (size, nitrogen content, type of plant in there, type of pests...) and the label can be the yield at the end of the year.
- The features can be search history and current keyword in a search enging and the label can be a recommandation, the output of the search.

The goal is to learn from a dataset D_n on which we have access to both features and labels, in order to be able to predict the label of a new sample from which we have only access to the feature and not the label.

The features/labels are typically modelled as random variables with X_i in a label space $\mathcal{X}$ (typically $\mathcal{X} = \mathbb{R}^d$ for some $d \geq 1$) and Y_i in a label space $\mathcal{Y}$ (for classification $\mathcal{Y} = \{0, 1\}$, for regression $\mathcal{Y} = \mathbb{R}$). The main goal of supervised learning is to predict a new $y \in \mathcal{Y}$ given a new previously unseen $x \in \mathcal{X}$. The unobserved data are called the testing data and D_n are called the training data.

One way to model this is to suppose that we observe a quantitative response Y and d different predictors, $X \in \mathbb{R}^d$. We assume that there is some relationship between Y and X, which can be written in the very general form

$$Y = f(X) + \varepsilon$$

where ε is a random variable with mean 0, it is the error term and $f(X)$ is a systematic information that X provides on Y. Then, the goal reduces to constructing a decision function f using the data from D_n such that if a new feature X_{n+1} is given, we can predict (i.e. approximate) the associated (unknown) label Y_{n+1} using $f(X_{n+1})$. In short, we seek a function f such that we have the following relation in distribution $f(X) \simeq Y$, where $\simeq$ is kept purposely vague, we will use several criteria of closeness between $f(X)$ and Y, and they will not be all equivalent. We call f a decision function or decision rule because it will allow us to decide on a prediction $f(x)$ we want to make on a new example x. Sometimes, the link between X and Y can be more complicated than $Y = f(X) + \varepsilon$, but we will always suppose that $f(X)$ "approximate" in some way Y.

The goal is to *learn* given past experiences, what to predict on futur samples.

1.1.1 A simple example of classification

Question: man or woman?

Statistics from one person.
- **Size** :
  1m72
- **Age** :
  23 years
- **Alcohol consumption** :
  0
- **Job** :
  C.S. Student.

Mean statistics for each gender.
- **Size** :
  M: 1m78, W: 1m64
- **Age** :
  M: 40.8 ans, W: 42.2 ans
- **Alcohol consumption**:
  Proportion of 0 alcool M: 39%, W: 61%
- **Job** :
  Among C.S. Students, M: 81%, W: 19%

This example shows how a database (here statistics from French government) can help in predicting the laber of an individual given his faetures. Remark that we still can't be sure whether the person is man or woman, but man seems more probable. Remark also that the age may seem not useful as is to predict the gender but itcould have been useful to correlate the age and the size in order to have a more precise prediction. This kind of link between features is done automatically using learning algorithms.

1.1.2 Applications and class of learning algorithms

Machine learning has been applied in various domains in particular in the following applications.

- Text or document classification, e.g., spam detection;
- Speech recognition, speech synthesis;
- Autocompletion;
- Image recognition:
  - Optical character recognition (OCR);
  - Face recognition;
  - Medical application (classification of tumor as cancerous, detection of early onset of Alzheimer...);
  - Detect animals for population estimation for use in ecology research;
- Fraud detection (credit card, telephone);
- Unassisted vehicle control (robots, navigation);
- Games, e.g., chess, backgammon, go, starcraft;
- Recommendation systems, search engines;
- Targetted advertising.

This list is not comprehensive, most of these applications belong to one of the following major class of learning problems.

Classification Assign a category to each item (image recognition, fraud detection).

Regression Predict a real value for each item (predict power consumption of computer, predict burned area of forest fires).

Ranking Order items according to some criterion (search engines).

Clustering Partition items into homogeneous regions (identify a community in Facebook graph).

Reinforcement Take actions in an online, streaming, maneer in reaction to past actions/rewards to optimize the sum of rewards (Games, vehicle control).

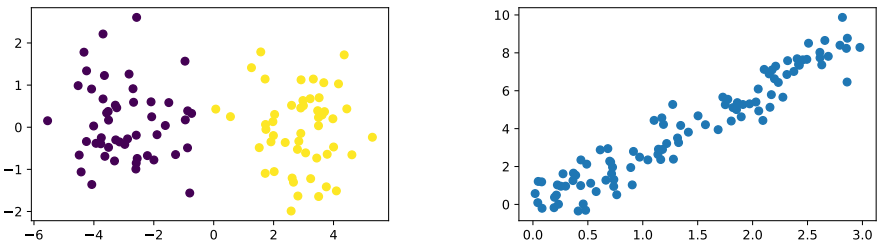
Only the first three classes (classification, regression and ranking) are in the supervised learning framework and among them, we will only look at classification and regression. In contrast, unsupervised learning (clustering in particular) do not have access to Y and tries to guess intrinsic properties of the data, this is in general harder than supervised learning because we try to answer a question without having any example of a correct answer.

Exercise 1 (Class of learning) Which of the examples given above are supervised classification? Which of them are classification problem? Regression problem?

1.2 First steps in Supervised Learning

1.2.1 Visual representation of Supervised Learning in $2D$

As already explained, most of the time the feature space $\mathcal{X}$ is $\mathbb{R}^d$. In the case of $d = 2$ for classification, we can represent the dataset as a point cloud, and we can represent the labels as colors (see Figure 1.1a). On the other hand, in regression, the output variable is a real number, not discrete in general, hence in this case it is easily representable in dimension $d = 1$ with x-axis being the feature and y axis the target we want to predict (see Figure 1.1b).



(a) classification (b) regression

Figure 1.1: Representation of supervised learning tasks in $\mathbb{R}^2$.

1.2.2 First parametric algorithms: linear regressions

Parametric methods suppose a *parametric model* for the decision function, i.e. we make the assumption that the function f can be approximated efficiently by a function of the form

$$\widehat{f}(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \cdots + \widehat{\beta}_d x_d.$$

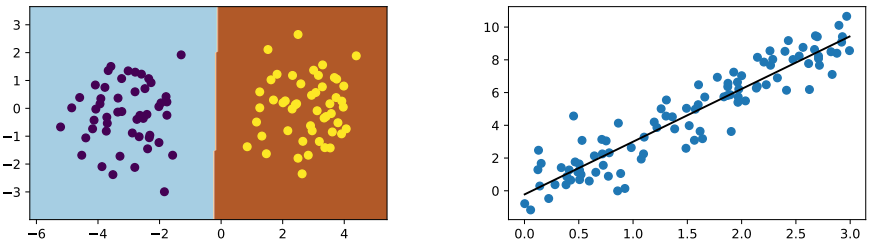
Hence for the estimation, we only need to estimate the coefficients $\widehat{\beta} = (\widehat{\beta}_0, \ldots, \widehat{\beta}_d)^T$. To do that we use a procedure that will *train* the estimator on the data. The most common in regression is the *Ordinary Least Squares* (OLS) regression which find an estimator $\widehat{\beta}$ such that

$$\widehat{\beta} = \arg\min_{\beta \in \mathbb{R}^{d+1}} \sum_{i=1}^{n} (\widehat{f}(X_i) - Y_i)^2.$$

and the most common in classification is the *Logistic Regression* which suppose that the classifier is of the form

$$\widehat{f}(x) = \text{sign}\left(\frac{1}{1+\exp(\widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \cdots + \widehat{\beta}_d x_d)} - \frac{1}{2}\right)$$

We will see more about this in the future.



(a) Logistic Regression (b) OLS

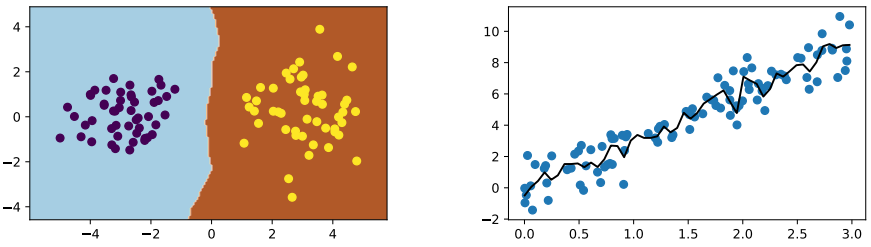
Figure 1.2: Logistic and linear regression $\mathbb{R}^2$.

1.2.3 First non-parametric algorithms: k-nearest neighbors

A very simple classification algorithm is the k-nearest neighbor algorithm (k-nn) which, given a new example x, predicts as a label the value of y_i which is in majority among its k closest (in the space $\mathcal{X}$) in the training sample. Formally, suppose we wanrt to predict the label of a new point in x, we begin by constructing $d_1 = \|X_1 - x\|, d_2 = \|X_2 - x\|, \ldots, d_n = \|X_n - x\|$ the distances of x to all the data points. We sort them such that $d_{i_1} \leq d_{i_2} \leq \cdots \leq d_{i_n}$. Then, the algorithm predict 1 if $\frac{1}{k}\sum_{j=1}^{k} Y_{i_j} \geq \frac{1}{2}$ and 0 otherwise.

Similarly, the k-nn algorithm can be used in a regression setting with the same reasoning except that the prediction becomes the value of $\frac{1}{k}\sum_{j=1}^{k} Y_{i_j}$.

In Figure 1.3, we represented the output of nearest neighbors algorithms on toy (simulated) datasets for regression and classification. In the case of classification, the background color represents the prediction we make and in the case of regression, the black line is the prediction we make.



(a) classification (b) regression

Figure 1.3: K-nearest neighbors in $\mathbb{R}^2$.

1.2.4 Towards more flexible methods: polynomial bases

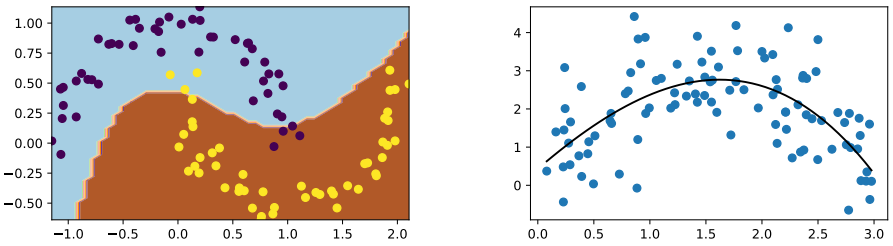
Remark that when the model is linear, we chose a very restrictive class of functions $\mathcal{H}$, which is all the functions that are linear. The principle of the OLS does not depend on being linear in x, we only need to be linear in β in order to have an easy optimization. Hence, we can define a feature function $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ and consider the regressor $\widehat{h}_n(x) = \widehat{\beta}^T \Phi(x) = \sum_{j=1}^{d} \widehat{\beta}_j \Phi(x)_j$. This allows us to consider a non-linear relation between x and y.

Intercept Consider $\Phi(x) = \begin{pmatrix} x \\ 1 \end{pmatrix}$, i.e. we just add a 1 at the end. In this case, $\widehat{h}_n(x)$ becomes $\widehat{h}_n(x) = \sum_{j=1}^{d} \beta_j x_j + \beta_{d+1}$ which is an affine line which does not necessarily go through 0.

Polyomial basis We may consider all the polynomial combinations of x up to a certain order, for example with a polynomial basis of order 2 with $d = 1$ we could have

$$\widehat{h}_n(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{1,2} x_1 x_2 + \beta_{1,1} x_1^2 + \beta_{2,2} x_2^2$$

i.e. a combination of order 1 and order 2 terms.



(a) Logistic Regression (b) OLS

Figure 1.4: Logistic and linear regression, polynomial basis of degree 3.

1.3 Possible problems in learning

Empirical illustration showing that the algorithm works well is **not sufficient** to guarantee that an ML algorithm will work when applied in the wild. There are numerous problems that can arise, and we need **theoretical guarantees** to understand the weakness and strong points of a given algorithm and to be able to assess whether an algorithm will perform well before using it on real applications. One of the main goals of this course will be to explain when and why these problems arise and to be able to identify which problem is causing an algorithm to fail on simple examples.

Non-deterministic label

The label y may not be a deterministic function of x. For simplicity, we will often make the hypothesis that there is an additive noise in the correspondance between y and x: $Y = f(X) + \varepsilon$ where ε is a random variable with $\mathbb{E}[\varepsilon] = 0$, but this is only a simplified model and may not reflect reality and in general the link could be modelled as $Y = f(X, Z)$ where X, Y, Z are all random variables, but we have only access to examples of Y and X.

Overfitting

The penomenon of overfitting arises when an algorithm learns to match too closely to the training data D_n and as a consequence the algorithm will not *generalize* well, i.e. it will perform poorly when used for prediction on test data. This phenomenon is very important to keep in mind and any ML practitioner should be aware of it so as not to oversel an algorithm that will, in the end, not work when applied on real data. We will explain more about this phenomenon in futur chapters of the course.

Underfitting – Model Misspecification

In an algorithm, one is not able to chose f from the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ as this set is too large. Instead, we pick f in $\mathcal{F} \subset \{f : \mathcal{X} \to \mathcal{Y}\}$, a restricted class of function that is easy to deal with. Typically, $\mathcal{F}$ can be the set of linear functions, or functions coming from a probabilistic model of the data (e.g. logistic regression), or neural networks. If the set of function is too small and if there is no function in $\mathcal{F}$ that can efficiently approximate the relation between X and Y then the model is misspecified, in Machine Learning language this is called underfitting.

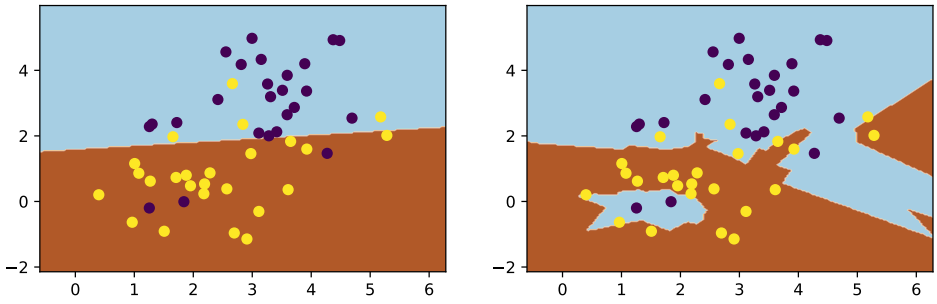


Figure 1.5: Example of overfitting: we represent the training data of a dataset in $\mathbb{R}^2$ by points whose color correspond to the class $y \in \{0, 1\}$. The background color represents the decision of a learning algorithm. The spiky separation line on the right is better at classifying the training data than the figure on the left, but the separation line is so complex that it is likely it will not generalize to unseen data. In contrast, the linear classifier on the left is smoother and should intuitively generalize better.

There exists several theorem showing that any continuous function can be approximated by a polynomial function (for instance using Bernstein polynomials) uniformly well (meaning that the approximation will have a small error at all the points of the dataset) although the degree of the polynomial is not controlled and in practice one may need to use large degree for the polynomial in order to have a sufficiently small error and this may not be practical. With the same idea, one can show that functions such as neural network can also approximate any continuous function uniformly well.

Choosing a model is a tricky business. There is no free lunch in statistics: no one method dominates all others over all possible data sets. George Box, a stastician from the 20th century, stated that "all models are wrong, but some are useful". This is especially true in ML, even if we have $Y = f(X)$, which is rarely the case, it almost never happen that the practitioner is able to use a set of function $\mathcal{F}$ that contains f, because using large sets $\mathcal{F}$ is computationally expensive. Instead, most algorithms use a family $\mathcal{F}$ that is easy to deal with and one of the main tasks will be to choose $\mathcal{F}$ as a trade-off between complexity of the class $\mathcal{F}$ and approximation error $\min_{g \in \mathcal{F}} \|f - g\|$. Such restriction to a restricted class $\mathcal{F}$ is sometimes called *inductive bias* or *inconsistent* model.

Curse of dimensionality

As the dimension of the space $\mathcal{X}$ increases, the learning problem becomes harder. For image recognition, the space $\mathcal{X}$ is $\mathbb{R}^d$ when d is the number of pixels in the image: it can be of the order of a million. We should be careful that in very large dimension, counter-intuitive phenomena can occure. In particular, in high dimension, most of the volume of the unit ball is located in a band of radius r around the unit sphere and this explain that when sampling at random in the ball, the locations of these samples are concentrated at the edge of this ball.

Exercise 2 (Ball in Cube in High Dimension) Let $e_1, \dots, e_{2^d}$ be the points with coordinates either $+1$ or -1 (i.e. in dimension 2, $e_1 = (1, 1), e_2 = (1, -1), e_3 = (-1, 1), e_4 = (1, 1)$). Consider the spheres S_{e_i} of center e_i and radius 1. Each sphere is contained in $[-2, 2]^d$. Now consider $S(d)$ the largest sphere with center 0 that do not overlap with any of the spheres S_{e_i}. Is the central sphere $S(d)$ contained in the cube $[-2, 2]^d$ for all $d \geq 2$?

One of the well-known effect of this is that local methods like nearest neighbor algorithm fail miserably in high dimension, because in a sense *all the points are far from one another* and the nearest neighbors of a given point may not be close to it and may not be good predictors. On the other hand, in recent years, neural networks algorithm have been successful in doing prediction successfully in very high dimension (i.e. images) and it may be one of the explanation of their popularity.

Non interpretability

Linked to the previous problem, there is a tradeoff to be considered between flexibility of the algorithm (i.e. capacity to approximate arbitrary complex function) and the interpretability/robustness of the algorithm. Simple algorithms like nearest neighbors, linear regression, decision trees are highly interpretable and better suited for applications in which one need to be able to explain why the algorithm took such or such decision (e.g. in medecine, agriculture). In contrast, methods such as neural networks are more complicated and less interpretable which pose a problem for interpretability. Coincidently, there are a lot more theoretical guarantees available for simple algorithms like linear regression, we are much more aware of their limitations than for neural networks. Very often, when a neural network work (or when it does not work) we do not know in detail, mathematically, why. We witness it in practice, but it can be that in a case not uncountered yet, a working neural network would fail miserably.

Other qualities of a learning algorithm

Recently, there have been a lot of interest in other qualities of learning algorithms that are needed in real life applications. Among these, are

- Fairness of an algorithm: when predicting a job matching in LinkedIn, the algorithm should not learn from the dataset that in the past men have had more access to high-qualification jobs,
- Privacy of an algorithm: when learning the next word you say on whatts'app, the algorithm should not also learn information that can be reverse-engeneered to read all your conversations,
- Robustness of an algorithm: in some tasks, when learning on a dataset which contains extremes (e.g. footballer salaries in a regression task) these extremes must be ignored so as not to reduce the efficiency of the algorithm on the non-extreme cases.

1.4 Theoretical guarantees in ML

1.4.1 Why the Need for Theoretical Guarantees?

As already explained, the goal of supervised learning is to construct, from already-seen data D_n, a decision function f that is good for predicting the label of a futur example. In short, we want the distribution approximation $f(X) \simeq Y$. However, in general we cannot say with confidence that because we learned on $(X_1, Y_1), \dots, (X_n, Y_n)$ then necessarily on a futur example X_{n+1} we will have $f(X_{n+1}) \simeq Y_{n+1}$. In order to be able to say with confidence that the decision rule f that we learned is efficient to predict futur examples, we will need to know that (X_{n+1}, Y_{n+1}) follows the same law as $(X_1, Y_1), \dots, (X_n, Y_n)$, maybe in addition to other more technical hypotheses (linear separability in the feature space for instance), and then we will typically aim at proving a theorem of the form: for some $t > 0$,

$$\mathbb{P}\left(\widehat{f}_n(X_{n+1}) \neq Y_{n+1} | D_n\right) \leq t$$

or maybe

$$\mathbb{E}[|\widehat{f}_n(X_{n+1}) - Y_{n+1}||D_n] \leq t$$

in a regression setting. The goal will be to show that this is true with *high probability* on D_n for a reasonable value of t. This means that with high probability, we will have a sample D_n that will allow us to do an error less than t.

Be careful that the previously stated results depend on the dataset we saw, hence they are random which explain the need for high probability results.

We will see later in the course that t will depend mainly on two things: the difficulty of the problem and in particular, how well it can be approximated by a simple function f and the complexity of the class of function $\mathcal{F}$ to which f belongs to.

In the next section, we develop theoretical guarantees for maybe the simplest problem in ML: we want to predict the futur value of a real random variable X and, we want to do this very simply with a constant. In this context, the difficulty of the problem will be the variance of X and the class of function are the constant functions (low complexity).

1.4.2 Concentration inequalities for sum of i.i.d. random variables

Concentration inequalities give probability bounds for a random variable to be concentrated around its mean. In their basic form, they quantify how fast the empirical mean converges to the true mean. The main use of this tool is to *control the random error encountered in the dataset*, this is a mathematical tool needed to show that our algorithm will not be influenced by the error ε.

We know that the empirical mean $\frac{1}{n}\sum_{i=1}^n X_i$ converges, when n goes to infinity, to the true mean $\mathbb{E}[X]$. But this does not give us any information on what happens for a given n, and in particular how can I choose n such that I know that with high probability the empirical mean is within 0.1 of the value of $\mathbb{E}[X]$?

Markov's inequality

The simplest probability inequality is maybe Markov's inequality.

Theorem 1 (Markov's inequality)**.** *Let X be a nonnegative real random variable, for any $t > 0$*

$$\mathbb{P}\left(X \geq t\right) \leq \frac{\mathbb{E}[X]}{t}$$

Proof. Let us first prove the vanilla Markov's inequality $\mathbb{P}(X > t) \leq \mathbb{E}[X]/t$ for some X a nonnegative real random variable. Let p be the density of the law of X. We have

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}[X\mathbb{1}\{X \geq 0\}] && (X \geq 0)\\ &\geq \mathbb{E}[X\mathbb{1}\{X \geq t\}]\\ &\geq t\mathbb{E}[\mathbb{1}\{X \geq t\}] = t\mathbb{P}(X \geq t).\end{aligned}$$

from which it follows that $\mathbb{P}(X \geq t) \leq \mathbb{E}[X]/t$. □

In particular, if Φ is a nonnegative, nondecreasing function, we can apply Markov's inequality to $\Phi(X)$ to get

$$\mathbb{P}\left(X \geq t\right) \leq \mathbb{P}\left(\Phi(X) \geq \Phi(t)\right) \leq \frac{\mathbb{E}[\Phi(X)]}{\Phi(t)}$$

where X is not necessarily nonnegative. This remark has a lot of applications in concentration inequalities.

Chebychev inequality

The first concentration inequality that we consider is Chebychev inequality, which is just an application of Markov's inequality to $\Phi(X) = (X - \mathbb{E}[X])^2$: let X be such that $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] < \infty$. Then,

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq t) \leq \frac{\mathrm{Var}(X)}{t^2}.$$

In particular, if applied to the empirical mean, we get

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^n X_i - \mathbb{E}[X]\right| \geq t\right) \leq \frac{\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^n X_i\right)}{t^2} = \frac{\mathrm{Var}(X)}{nt^2}.$$

By change of variable this is equivalent to for any $\delta \in (0, 1)$,

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^n X_i - \mathbb{E}[X]\right| \geq \sqrt{\frac{\mathrm{Var}(X)}{\delta n}}\right) \leq \delta.$$

This is a concentration inequality: it shows that indeed, the empirical mean converges to the true mean and at which speed. This is to be compared to Markov inequality for which, if $X_1, dots, X_n$ are positive, we have

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X] \geq t\right) \leq \frac{\mathbb{E}[X]}{\mathbb{E}[X]+t},$$

Hence,

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X] \geq \mathbb{E}[X]\left(\frac{1}{\delta}-1\right)\right) \leq \frac{\mathbb{E}[X]}{\mathbb{E}[X]+t},$$

the probability does not go to 0 as n goes to infinity.

Exercise 3 (Perudo game) In the game of Perudo (also called dudo), each player starts having five six-faced dices and a cup, which is used for shaking the dice and concealing the dice from the other players. The players shake their dice in their cups, and then each player looks at their own dice, keeping their dice concealed from other players. Then, the first player makes a bid about how many dice of a certain value are showing among all players, at a minimum. For example, a bid of "five threes" is a claim that between all players, there are at least five dice showing a three. The player challenges the next player (moving clockwise) to raise the bid or call dudo ("I doubt") to end the round. When dudo is called, everybody looks at their dices, if there were enough dices for the bet, the player that doubted looses otherwise the doubted player (the one which made the final bid) looses.

1. Imagine a game with 5 players, this is the first round. The player before you just did the bet of "5 sixes". A good first strategy is to call dudo only if the bet is above the expected number of dices with a 6 on it. Is it the case here?

2. You lost at this round. If we ignore the fact that we have partial information thanks to our cup, give an upper bound on the probability that there are more than 6 dices with six in the game. Would you have called "dudo" knowing this probability?

3. For this use case, we can use an inequality that is in fact more powerful than Chebychev inequality. Let us accept that the following inequality (called Cantelli's inequality) is true: for any $t > 0$,

   $$\mathbb{P}(X - \mathbb{E}[X] \geq t) \leq \frac{\mathrm{Var}(X)}{\mathrm{Var}(X)+\lambda^2}$$

   Remark that this is a unilateral version of Chebychev's inequality. With this refined inequality, was your bet in question 1 a good bet?

4. More generally, give a formula that can be expressed as a function of the number of dices (and maybe the dices in your hand) and that can be computed easily in-game in order to have a prediction of how many dices people other than yourselves have in their hands. For instance, we could search for the number of dices above which there is less than 50% chance that the bet is lost.