Notes Uncertainty in Machine Learning

Andrea Campagner

February 2025

1 Introduction

The aim of these lectures will be to give an introduction to Uncertainty in Machine Learning (ML). This seems a relevant topic, but we first need to understand what we mean when we talk about *uncertainty*.

Looking at the ML literature, there are essentially three notions of uncertainty that have been widely studied [31, 45]:

1. Firstly, in any task that we may want to solve using ML techniques we are typically not able to access all data on the phenomenon of interest. For example, we may never hope to get the population of all possible patients of all possible diseases. Instead, we only typically have access to limited data. This has profound consequences, in that we generally want to develop ML models that work well on all of the data, while observing only a (typically small) portion of it. This, in turn, implies that every estimate of performance (based only on a finite sample, e.g. a test set) will be affected by uncertainty (this is the reason we compute confidence intervals on, e.g., accuracy). This form of uncertainty is called *epistemic uncertainty*, as it derives from a lack of knowledge (about the complete data generating process);

2. Second, not only there is uncertainty due to having access to only limited data, but the task itself may be uncertain. For example, a task may be inherently probabilistic/non-deterministic (e.g., quantum mechanics), or we may want to use probabilities to model our system in a simpler way (e.g., it would be impossible to measure all variables about a patients... but if we restrict to just a subset of them, then two patients with the same description could have different diseases). This is called *aleatoric uncertainty*, as it derives from the randomness (real or not) intrinsic in the problem and (generally, or at least, not easily) ineliminable;

3. Finally, in the two above cases, we have assumed that we have access to a veritable representation of our phenomenon (albeit potentially incomplete). However, in some cases we may not be able to access a truthful and precise representation of our phenomenon: for example, it may be hard to

collect all data (so that we may have missing data, or gaps), to collect it in a sufficiently accurate manner (so that we may have errors in the data, or noise), or to collect it at a desired level of granularity/precision. In this case we speak of *data uncertainty* or *uncertain data*.

The first form of uncertainty described above has been the focus of most of the work in the statistical literature (and, by extension, in ML, especially in its more theoretically-oriented form). The goal, in this case, is to design algorithms that are able to learn well (i.e., do not over- or under-fit) from limited data, as well as studying ways to bound the performance of models.

The second topic has also been studied for a long time in the statistical literature (keywords are parametric models, consistency, identifiability). The aim is to design algorithms and models that are not only "accurate" (whatever that means) but are also able to provide reliable estimates of the aleatoric uncertainty. Ideally, we may also want to provide a decomposition of the uncertainty of a model into aleatoric and epistemic uncertainty: this has only been considered in the ML literature more recently and is the subject of the *Uncertainty Quantification* (UQ) subfield [27].

Finally, while the third topic has a long history (missing data have been studied in the statistical literature since at least the '70s, see the book by Little and Rubin [31]), only more recently it has received increased attention in the ML community, especially with increasing popularity of the *weakly supervised learning* sub-field [50], that naturally overlaps with the phenomenon of data uncertainty (in both cases, we want to learn despite having access only to imperfect information). In recent years, development in this area has been consolidating within the *learning from imprecise data* community [24].

The aim of this course will be to provide an introduction to all these three forms of uncertainty, through a mathematically and algorithmically-oriented perspective. This means that we will study several computational problems and, for each of them, our objective will be to provide an answer to the two following questions:

1. Is the problem solvable? We will be particularly interested in constructive solutions, i.e. explicit solutions in the form of algorithms;

2. Which resources are required to solve the problem? The main resources we will be interested in are *time complexity* (how many computational steps are required to solve the problem) and *sample complexity* (how much data we require to reach a desired level of performance).

2 Mathematical Notation

In this course, we will adopt an abstract, mathematically-oriented approach to studying Machine Learning models and algorithms. This means that we need to adopt a precise, formal mathematical notation that enables us to talk about ML tasks, models, algorithms and their mathematical properties.

Think of a ML model you may have used in your work or research. At a high-level, we conceptualize such a model as a black-box that takes some input (representing the salient characteristics of an instance) and gives some output (a prediction). We will try to formalize this idea in a precise manner.

2.1 Learning Tasks

We will denote with X the input space: each object $x \in X$ is a representation of an instance. X could be finite or infinite, and we generally do not make assumptions about its structure. For some results we may need some specific additional conditions, for example:

- X is a (d-dimensional, real) vector space (e.g., $X = \mathbb{R}^3$);
- X is a metric space (there is some notion of distance among instances) (e.g., $X = \mathbb{R}^3$ with the cosine distance);
- X is a *convex set* (given any two instances $x, y \in X$, all instances on the line connecting x and y are also in X) (e.g., $X = S^2$, the sphere in $\mathbb{R}^3$ with radius 1).

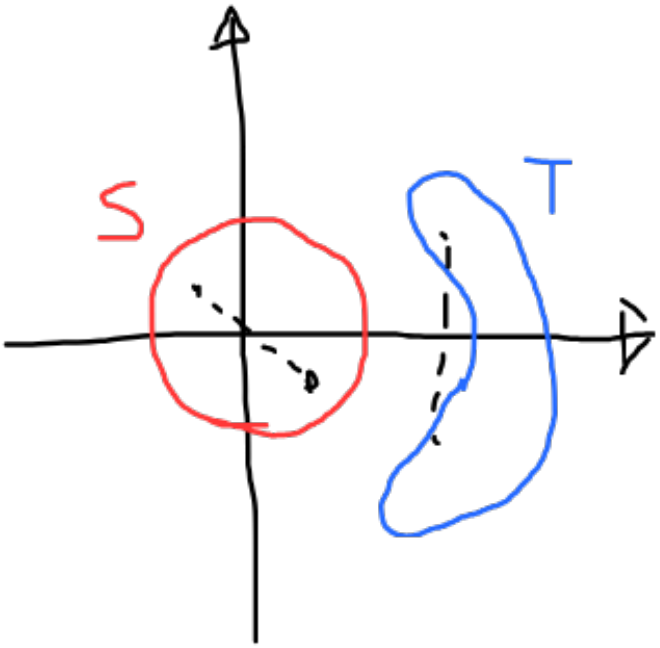

Figure 1: Example of a convex (S) and non-convex (T) sets.

While X denotes the representation of instances, we are typically interested in associating some information with these instances. In particular, we will

focus on *supervised learning*, in which each instance is associated with a special, particularly important piece of information called the *target*. We assume that target values are taken from a set Y, called the *target space*. We will mostly focus on the binary classification case, in which $Y = \{0, 1\}$.

Even though we have talked about instances and targets, so far we have not established any connection between them... how can we say that a specific target value is associated with a specific instance?

The most general approach is to assume the existence of a probability measure $\mathcal{D}$ defined over $X \times Y$: intuitively[1], we may think of the value $\mathcal{D}(x, y)$ as the probability of observing the labeled pair (x, y). $\mathcal{D}$ is called the *data-generating process*, as it describes how the data that we may observe can arise.

To understand this scenario, consider the following example: there is some hospital at which patients (X) are admitted to be treated for one among a set of diseases (Y). Obviously, not all of the population comes to the hospital at the same time (and some of them will never come to the hospital!), so we may think of the process (unknown and very complex) that regulates how patients come to the hospital as the distribution $\mathcal{D}$ defined above.

Note that, in general, we will not assume a deterministic relationship between X and Y, i.e. we will not assume that $\forall x \in X, \exists! y \in Y$ such that $\mathcal{D}(x, y) > 0$.

In the binary setting (more in general, in regression) the random variable $\mathcal{D}(x) = \mathbb{E}[Y|x]$ (that is, the expected value of Y conditioned on the instance being x) takes a fundamental role since, in general, it is the primary learning objective (i.e., the thing we want our models to predict). Indeed, $\mathcal{D}(x)$ encodes all relevant information about the data-generating process, if we only care about predictive performance:

- $\mathcal{D}(x) = P(Y = 1|x)$, so it encodes the probability of observing the two possible labels (because $P(Y = 0|x) = 1 - \mathcal{D}(x)$);
- $\mathcal{D}(x)$ encodes the uncertainty intrinstic in the task: $Var[Y|x]$ represents the variability of the target information, and can obviously computed as $\mathcal{D}(x)(1 - \mathcal{D}(x))$ (variance of a Bernoulli variable). This is one of the most commonly applied ways to quantify (and formalize) *aleatoric uncertainty* in the literature [12, 13, 38].

We will get back to this point in the following lectures.

2.2 Models, Losses and Algorithms

Now, we have all ingredients that describe a learning task, but we still lack some pieces: what is a model? how we evaluate its quality? how we select a model based on data?

[1]The intuition breaks up when $X \times Y$ is uncountable, as commonly happens when we assume that X is (a subspace of) a real vector space. In this case, to be precise, we will need to define the measurable sets and we will also need to require that ML models are *measurable functions*... we will hide these technicalities under the carpet, but keep in mind that they are sometimes important!

For our purpose, a model (or *hypothesis*) is a function $h : X \to Z$, where Z is called the *label space*. Note that Z may differ from Y! For example, neural networks do not generally return a target value (unless some post-processing is applied, e.g. argmax), but rather a softmax distribution: in this case, $Z = [0,1]^Y$ (with the added requirement that $\sum_{y \in Y} z_y = 1$). As mentioned before, in most of the course $Y = \{0,1\}$: in this case, we will usually assume that, either, $Z = Y$, or $Z = [0,1]$. Moreover, we will assume that models are selected from a set $\mathcal{H}$, which is a subset of the set of all function $X \to Z$.

Similarly as for X, we will not assume any general constraints on $\mathcal{H}$. When needed, we may impose the following constraints:

- $\mathcal{H}$ is a (subset of a) vector space (e.g., $\mathcal{H} = \mathbb{R}^3$, which is the set of 3-dimensional linear predictors on $X = \mathbb{R}^3$);
- $\mathcal{H}$ is a convex set (e.g., $\mathcal{H} = \mathbb{S}^2$, which is the set of 3-dimensional linear predictors on $X = \mathbb{R}^3$ with l_2 norm regularization);
- $\mathcal{H}$ is an Hilbert space (it is a vector space with an inner product $\langle \cdot, \cdot \rangle$ that defines a complete metric space... imagine a (possibly infinite-dimensional) Euclidean space) (e.g., $\mathcal{H} = \ell_2$, the set of infinite sequence of real numbers s.t. $\sum_i h_i^2 < \infty$, with the inner product given by $\langle h, g \rangle = \sum_i h_i g_i$... this is the set of infinite-dimensional linear predictors!);
- $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (RKHS) [33, 39]. An RKHS is an Hilbert space, such that it exists a *feature map* function, $\phi : X \to \mathbb{R}^d$, where d can also be infinite, such that $\forall h \in \mathcal{H}, \exists v_h \in \mathbb{R}^d$ with $h(x) = \langle \phi(x), v_h \rangle$... as an example, assume $X = \mathbb{R}$ and $\mathcal{H}$ is the set of polynomials over X with degree up to 3. Then $\phi(x) = (1, x, x^2, x^3)$ and if $h(x) = a + bx + cx^2 + dx^3$ we have $v_h = (a, b, c, d)$.

Given a model h and an instance (x, y), how do we evaluate the prediction of h on x? To this aim, we consider the notion of a *loss function*, that is a map $l : X \times Y \times \mathcal{H} \to \mathbb{R}$: the value $l(x, y, h)$ represents the error that model h makes on instance x when the true target value is y. We will generally assume that lower values of l denote better predictions. You may have seen several loss functions in your research:

- If $Y = Z$, $l_{0-1}(x, y, h) = \mathbb{1}_{h(x) \neq y}$ is the 0-1 loss (also called *error rate*);
- If $Y, Z \subseteq \mathbb{R}$, $l(x, y, h) = (y - h(x))^2$ is the *squared loss* (commonly used in regression tasks... but we will see it is quite important in uncertainty quantification!);
- If $Y, Z \subseteq [0,1]$, $l(x, y, h) = -y \log h(x) - (1 - y) \log(1 - h(x))$ is the *logarithmic loss* (also called *cross-entropy*, commonly used in deep learning).

In general, we will assume that the loss functions we consider (with the exception of the 0-1 loss) are convex: that is, it holds that $l(x, y, \alpha h_1 + (1 -$

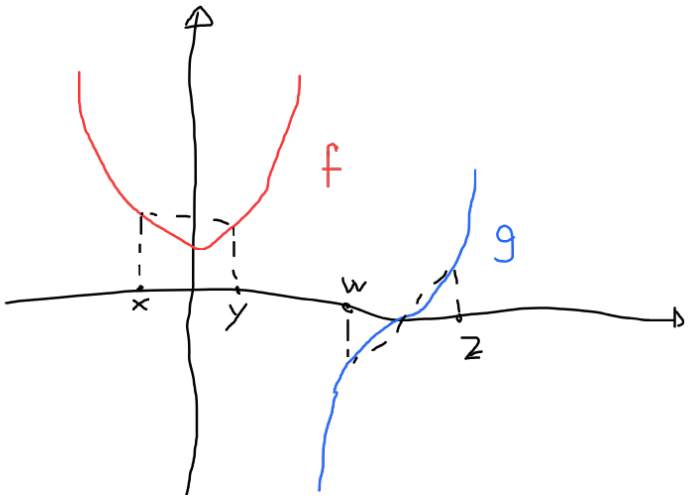

Figure 2: Example of a convex (f) and non-convex (g) functions.

$\alpha)h_2) \leq \alpha l(x, y, h_1) + (1 - \alpha)l(x, y, h_2)$ (notice that this implies that the set of models $\mathcal{H}$ is a subset of a vector space).

While a loss function evaluates the quality of a model on just a single instance, it also gives us the tools to understand whether a model works well for a task (or not). We define the *true risk* of a model h as:

$$R_l(h, \mathcal{D}) = \int_{X \times Y} l(x, y, h) d\mathcal{D}, \tag{1}$$

that is, we evaluate a model in terms of its *expected loss* on the whole learning task, where expectation is computed w.r.t. the data-generating process. When l and $\mathcal{D}$ are clear from context we will write simply $R(h)$. When $l = l_{0-1}$, the true risk takes a particularly convenient form:

$$R_{0-1}(h) = P(h(x) \neq y) = \mathcal{D}(\{(x, y) : h(x) \neq y\}), \tag{2}$$

that is, the probability to sample an instance (and corresponding target value) which is misclassified by the model.

Notice that the definition of the true risk not only provides a way to evaluate an arbitrary model h, but it also gives an absolute comparison scale in the sense that it defines (for every learning task) an optimal model. We define a *Bayes predictor* to be a model h^* s.t.

$$h^* = \arg\min_{f \text{ measurable function}} R(f) \tag{3}$$

Note that, in general, there is no requirement that $h^* \in \mathcal{H}$ (i.e., we may have selected a class of models that does not include the optimal one... more on this

later!). In most cases, it is easy to actually understand what is the form of the Bayes predictor:

- If l is the Brier score, then $h^*(x) = \mathcal{D}(x)$;
- If l is the logarithmic loss, then $h^*(x) = \mathcal{D}(x)$;
- If $l = l_{0-1}$, then $h^*(x) = \begin{cases} 1 & \mathcal{D}(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

Note that both the logarithmic loss and Brier score correctly identify the Bayes predictor as the model that predicts the conditional target probability $\mathcal{D}(x)$: as we mentioned this being the primary objective of learning, this is a desirable property! By contrast, the 0-1 loss does not satisfy this property...

Now that all ingredients are in place, we can proceed with studying the three forms of uncertainty we described beforehand.

3 Epistemic Uncertainty, or Learning from Finite Data

Our discussion identified a clear learning objective: we would like to recover the Bayes predictor... but is it possible to do so?

The problem, in practice, is that we do not have access to the data-generating process (otherwise, learning would be trivial!)... instead, we typically only have access to data sampled from $\mathcal{D}$. Formally, we say that we have a finite sample (also called training set) $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \sim \mathcal{D}^m$, consisting of m instances sampled from $\mathcal{D}$ i.i.d. (independent and identically distributed). This has two deep consequences.

First, since we have no direct access to $\mathcal{D}$, we cannot calculate the true risk $R(h)$ of a model h. Instead, we can approximate it through the *empirical risk*:

$$\hat{R}_S(h) = \frac{1}{m} \sum_{(x_i, y_i) \in S} l(x_i, y_i, h). \tag{4}$$

Given the empirical risk, the most natural thing to select a model is to simply choose (one of the) models that minimize the empirical risk:

$$ERM(S, \mathcal{H}) = h_S \in \arg\min_{h \in \mathcal{H}} \hat{R}_S(h). \tag{5}$$

This approach is called *empirical risk minimization* and it will be the algorithm we focus on in this part of the course. An important question, then, is to which degree the empirical risk is a good approximation to the true risk and, hence, how close the output of ERM will be to being a Bayes predictor.

Second, as long as our learning algorithm relies on the available data S (every meaningful algorithm would do so!), the performance of the model h_S it gives as output will be random, because S is random in the first place. For this

reason, we will be interested in probabilistic guarantees about the performance of learning algorithms (ERM in particular), such as:

$$P(R(h_S) - R(h^*) \geq \epsilon) \leq \delta, \tag{6}$$

where we want to say that δ is as small as possible (as a function of both ϵ, the error we accept as tolerable, and m, the sample size). This is called the *PAC learning* (probably, approximately correct) model in the literature [44]: the reason for the name is that we aim at learning algorithms that are probably correct (i.e., close to a Bayes predictor) with high probability, provided the training data is sufficiently big.

So, the basic question we pose is: given only a finite set S, is it possible to recover the Bayes predictor h^*, having minimal *true risk*? We have two huge problems in facing this question:

1. Since we cannot compute $R(h)$, we cannot directly find h^*... moreover, notice that we do not even know the value of $R(h^*)$!
2. Even if we knew $R(h^*)$, given a model h how do we know whether $R(h) = R(h^*)$ (and so h is a Bayes predictor) if we can only compute $\hat{R}_S(h)$ for an available set of data S?

3.1 Epistemic Uncertainty and Limited Data

The second question is a first form of what we called *epistemic uncertainty* in the introduction: since we only have access to limited knowledge (finite data), we are not able to precisely know the performance of our model... However, in this case, the problem has an easy answer:

Theorem 1 (Hoeffding Inequality [22])**.** *Let h be a model selected independently from S and let l be a loss function whose value is bounded in $[0, b]$. Then, with probability higher than $1 - \delta$, it holds that*

$$R(h) \leq R_S(h) + b\sqrt{\frac{\log(1/\delta)}{2m}} \tag{7}$$

This theorem tells us that, once we know the empirical risk of a model h, we can also know its true risk (up to some small probability of error) and this latter is not much larger then former: furthermore, they become closer and closer as the sample size grows larger. This implies that this form of epistemic uncertainty is, in principle, eliminable: we can simply obtain more data! This makes sense: the more information we have, the more knowledge we are able to extract from it and the less our uncertainty!

Note, hovewer, that the result above requires that h is selected independently of S: this means that, for example, the results would be applicable if we previously trained h on a separate set of data and use S only as test set... but we cannot apply this result if we compare multiple models on S and select the best among them, which is what we commonly do when we perform model

selection! Fortunately, we can recover very similar guarantees also for this latter case:

Theorem 2 (Uniform Convergence (for finite model classes)[42])**.** *Let $\mathcal{H}$ be a* finite *set of models. Let S be a set of data and let l be a loss function whose value is bounded in $[0, b]$. Let h_S be the result of $ERM(S, \mathcal{H})$. Then, with probability higher than $1 - \delta$, it holds that*

$$R(h_S) \leq \hat{R}_S(h_S) + b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2m}} \tag{8}$$

Compared to Hoeffding inequality, we pay only an additional $\log(\mathcal{H})$ penalty: thus, the more models we compare (on S) the larger our epistemic uncertainty. This provides two ways for reducing our uncertainty: either we increase the amount of data, or we consider a *simpler* class of models!

Note, however, that the result requires that $\mathcal{H}$ is finite: this may seem a big limitation (is the set of neural networks with a certain architecture finite?) but in practice it can always be used because all models have to be represented on a computer (with finite memory and finite precision)... yet $\mathcal{H}$ may grow very large: in the literature there are generalizations of this result to infinite $\mathcal{H}$ but we won't cover them in these lectures (see [3, 42, 49] if you're interested)!

The previous theorems provide a way to resolve the second problem above, and do so efficiently: indeed, once we have computed the empirical risk of a model, we can simply compute (an approximation to) its true risk in constant time... this means that, if somehow we are magically told by an oracle the true risk of the Bayes predictor $R(h^*)$, we could tell whether our model is also a Bayes predictor or not.

However, we've not yet addressed our first problem: that is, we do not have an answer yet to the question of whether it is possible to recover h^* by applying ERM (with high probability). The (seemingly) simplest way to do so would be to have the set $\mathcal{H}$ (the hypothesis set) contain all possible models: in this way, we are certain that h^* is contained in $\mathcal{H}$ and we only need to hope that ERM is able to find it. However, this approach is bound to failure:

Theorem 3 (No-Free Lunch Theorem [48])**.** *Let l_{0-1} and $\mathcal{H}$ be any set of models. Then, there exists $\mathcal{D}$ such that $\mathbb{E}_{S\sim\mathcal{D}^m}[R(ERM(S, \mathcal{H})) - R(h^*)] \geq \frac{1}{2}$. This holds, in particular, when $\mathcal{H}$ is the set of all possible models.*

This theorem implies that we need to know something about the learning problem we need to solve: otherwise, if we had complete uncertainty, we have no hope of solving the problem! To better understand this, we can see another result that enables us to describe the gap between the performance of (the model resulting from applying) ERM and that of h^*:

Theorem 4 (Error Decomposition [3, 42])**.** *Let $\mathcal{H}$ be a (finite) set of models, l a loss function bounded in $[0, b]$. For any dataset S, let $h_S = ERM(S, \mathcal{H})$.*

Then, with probability larger than $1-\delta$ over the sampling of S, it holds that:

$$\begin{aligned} R(h_S) - R(h^*) &= R(h_S) - \hat{R}_S(h_S) + \hat{R}_S(h_S) - \hat{R}_S(h^*) + \hat{R}_S(h^*) - R(h^*) \\ &\leq 2b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2m}} + \hat{R}_S(h_S) - \hat{R}_S(h^*) \end{aligned} \tag{9}$$

If $h^* \in \mathcal{H}$ this implies that, with probability larger than $1-\delta$:

$$R(h_S) - R(h^*) \leq 2b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2m}} \tag{10}$$

The theorem says that the gap in performance between $ERM(S, \mathcal{H})$ and the Bayes predictor h^* can be decomposed in two parts:

- $R(h_S) - \hat{R}_S(h_S) + \hat{R}_S(h^*) - R(h^*) \leq 2b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2m}}$ which stems from epistemic uncertainty (problem 2 above: given the empirical risk of a model we're only able to approximately tell its true risk). This is called *estimation error* in the literature, because it is the component of error that stems from us estimating our models based on finite data;
- $\hat{R}_S(h_S) - \hat{R}_S(h^*)$, which is the empirical gap in performance between h_S (that we know, since we found it through ERM) and h^* (that we do not know). This is called the *approximation error* in the literature, because it is the component of error introduced by the fact that we are approximating h^* with models in $\mathcal{H}$.

Notice that the estimation error is equivalent to epistemic uncertainty and so can be quantified (as we have shown above that we're able to quantify the epistemic uncertainty) and also reduced, by simply increasing the sample size, irrespective of the learning problem we are aiming to solve. The approximation error, by contrast, cannot in general be quantified (since we do not know $\mathcal{D}$ and h^*) and neither reduced without changing $\mathcal{H}$: in principle, it could be made equal to 0 when $h^* \in \mathcal{H}$ but without making assumptions about the learning task we cannot know whether this holds or not... this is the reason why the No Free Lunch Theorem arises!

So, these two last theorems provide a (partially) positive answer to our question: in general, we cannot guarantee that we are able to find the Bayes predictor h^* using ERM... however we can guarantee that we will be able to do so (with high probability), if h^* is among the models ERM can select! In the following lessons we will see that, in most cases, we are still able to obtain better results and get better guarantees...

3.2 Epistemic Uncertainty and Model Multiplicity

But before moving forward, we note that there is another form of epistemic uncertainty which is less directly linked to estimation error. Assume that we have a finite set of data S on which we apply $ERM(S, \mathcal{H})$: this results in a

model h_S. Even if we could guarantee that $h^* \in \mathcal{H}$ we still do not know whether $h_S = h^*$... because there may other models $h_i \neq h_S \in \mathcal{H}$ s.t. $\hat{R}_S(h_i) = \hat{R}_S(h_S)$: that is, h_S may not be the unique empirical risk minimizer!

This phenomenon, called *model under-specification* [10] or *model multiplicity* [4], is a form of epistemic uncertainty: since we only have access to finite data, we may not be able to unambiguously tell which model is the best one!

However, we can, in principle, quantify also this form of epistemic uncertainty:

$$EU(\mathcal{H}, S)(x) = \sup_{h \in \mathcal{H}: \hat{R}_S(h) = \hat{R}_S(h_S)} h(x) - \inf_{h \in \mathcal{H}: \hat{R}_S(h) = \hat{R}_S(h_S)} h(x). \tag{11}$$

EU quantifies, for each instance x, the amount of uncertainty we have about the correct prediction (i.e., the prediction given by the Bayes predictor h^*) by simply taking the maximum gap in the predictions given by all models that we consider, based on the available data, as plausible candidates for being h^*.

Notice that, as long as we know that $h^* \in \mathcal{H}$, then due to the error decomposition, it holds with high probability that:

$$h^*(x) \in \left[\inf_{h \in \mathcal{H}: \hat{R}_S(h) = \hat{R}_S(h_S)} h(x), \sup_{h \in \mathcal{H}: \hat{R}_S(h) = \hat{R}_S(h_S)} h(x) \right].$$

The interval above, then, can be interpreted as an (instance-wise) confidence interval for the Bayes predictor h^*. This idea is the foundation for models that used imprecise probabilities (of which probability intervals as above are a special case) to model uncertainty in ML [23, 26, 34, 37, 40].

Therefore, the formula for EU tells us that we can quantify the epistemic uncertainty (due to model underspecification) through the following calculation: for each instance x we take all models which are possible candidates for being the Bayes predictor, we take their predictions and compute the maximum possible gap between them[2]. If all models provide predictions that are very close to each other, then our epistemic uncertainty will be small... otherwise, we may have large uncertainty and known almost nothing about the exact form of the Bayes predictor (despite being guaranteed to be close to it)!

3.3 A Note on Computational Complexity

Closing this lecture, we note that we talked about algorithms (ERM) and their guarantees in terms of error... but not about their computational complexity. What can we say in this regard? Unfortunately, in general, learning through ERM is a computationally hard problem (for example, it is known that ERM for neural networks or decision trees is an `NP-HARD` problem). Under some assumptions, however, ERM can be solved (approximately) in polynomial time.

[2]One could also potentially consider other measures of deviation, but the interval length considered has been particularly justified as a good measure in the literature [37].

Theorem 5. *Let $\mathcal{H}$ be a RKHS, and let l be a differentiable, L-Lipschitz, convex loss function, i.e. it holds that:*

1. $l(x, y, \alpha h_1 + (1-\alpha)h_2) \leq \alpha l(x, y, h_1) + (1-\alpha) l(x, y, h_2)$ *(convexity);*
2. $|l(x, y, h_1) - l(x, y, h_2)| \leq L|h_1 - h_2|_{\mathcal{H}}$*, where $|\cdot|_{\mathcal{H}}$ is the norm on $\mathcal{H}$ (Lipschitzness).*

Then, assuming inner products can be computed in polynomial time, there exists a polynomial-time randomized algorithm for ERM: this is obtained by stochastic gradient descent on $\mathcal{H}$ using l as the loss function [35].

The result above applies to a wide class of ML methods, which include all linear models (regularized or not) as well as all approaches that rely on the kernel trick (e.g., support vector machines and Gaussian processes).

Concerning the computation of EU, note that even though solving ERM can be performed efficiently, computing EU may not be easy (because we need to enumerate all optimal solutions to ERM) or even impossible (when $\mathcal{H}$ is infinite or when l is strongly convex, as in this latter case there exist a single empirical risk minimizer... which may not be the Bayes predictor!). If there exists an efficient algorithm for ERM, fortunately, this problem can be approximately solved using a bootstrapping approach (this idea has been proposed many times in the ML literature!):

Algorithm 1 Boostrapping algorithm for computing EU

```
procedure BootstrapEU(S: training set, H: set of models, n_boots: bootstrap iterations)
    H ← ∅
    for i = 1 to n_boots do
        S^(i) ← resample S with replacement
        h ← ERM(S, H)
        Add h to H
    end for
    return H
end procedure
```

Under the same assumptions as above (and some additional technical assumptions), the set H returned by Algorithm 1 can be used to compute EU (furthermore, the convex hull of H contains h^* with high probability) and computing H only requires time proportional to that of applying ERM. Precisely, if T is the time required by ERM, then Algorithm 1 has time complexity $\Theta(T \cdot n\_boots)$: if T is polynomial, then Algorithm 1 also runs in polynomial time. Then, EU can be computed simply from H in constant time $\Theta(n\_boots)$, for each instance x, by applying Eq. 11 to set H.

4 Aleatoric Uncertainty, or Learning under Randomness

Let us recall our assumptions on how data is generated. First, we have a distribution $\mathcal{D}$, the data-generating process, which governs how the data is generated.

Since, in general, we do not assume that $\mathcal{D}(x)$ is deterministic, this implies that there is some ineliminable source of uncertainty, called *aleatoric uncertainty*. This implies that no model (including the Bayes predictor) can have true risk smaller than some fixed amount that depends on the aleatoric uncertainty. Furthermore, we are not able to access $\mathcal{D}$ directly, but rather we can only obtain a finite set of data by sampling from it. This, in turn, implies that our knowledge is limited, leading to *epistemic uncertainty*.

While in the previous lecture we talked extensively about epistemic uncertainty, in this lecture we will instead focus on aleatoric uncertainty. Before getting into technical matters, let us convince ourselves why being able to quantify aleatoric uncertainty, as well as being able to distinguish it from epistemic uncertainty, can be relevant in practical applications of ML.

- Assume we have a task in which the costs of error are different. For example, if our target is to detect some serious disease (e.g., cancer) then false negatives (i.e., predicting 0 when the true target is 1) are more serious than false positives (i.e., predicting 1 when the true target is 0). Given a patient x, and costs $c(FP), c(FN)$, how we decide how to predict? If we knew $\mathcal{D}(x)$, we could simply predict 1 if $(1 - \mathcal{D}(x))c(FN) \leq \mathcal{D}(x)c(FP)$, or 0 otherwise. Aleatoric uncertainty, in this case, gives us an indication of how much we risk incurring in a wrong decision, and its potential impact. Note, also, that though in practice we may use $h(x)$, for h being a model, in place of $\mathcal{D}(x)$, this may incur additional error;

- Assume we trained a model h and measured its performance on a validation set S, obtaining some number $\hat{R}_S(h)$. According to the error decomposition in Eq. (9), we know that the true risk of h can be upper bounded by a quantity that depends on $\hat{R}_S(h)$, the epistemic uncertainty and the risk of the Bayes predictor: this latter, in turn, depends on the aleatoric uncertainty, in the sense that the larger the aleatoric uncertainty the larger the risk of the Bayes predictor. If we are able to quantify the aleatoric uncertainty, then we know what goes wrong with our models and, consequently, we know how we can get better performance: 1) If the epistemic uncertainty is too large, get more data (or consider a simpler class of models); 2) If $\hat{R}_S(h)$ is too large, this may be due to underfitting (then, we need to consider a more complex class of models); 3) if the aleatoric uncertainty is too large, we may need to reformulate our learning task.

In both cases, knowing the aleatoric uncertainty enables us to get some value in practical applications. But how do we compute it?

In the previous lectures, we defined the aleatoric uncertainty to be $\mathcal{D}(x)(1 - \mathcal{D}(x))$, that is the (conditional) variance of the data generating process. Thus, to quantify the aleatoric uncertainty we need to recover the conditional distribution $\mathcal{D}(x)$. This comes, in turn, with two implications:

1. If our aim is to be able to estimate $\mathcal{D}(x)$, we need to design and consider a loss function that guarantees this objective (as an example, we have seen that the 0-1 loss function does not have this property);

2. Estimating the aleatoric uncertainty is intrinsically tied to estimating the epistemic uncertainty: as we have shown in the previous lecture, limited data and model multiplicity may affect our ability to recover the Bayes predictor using ERM.

4.1 Not all Loss Functions are Born Equal: An Excursus

When talking about epistemic uncertainty we were agnostic in the selection of a loss function: as long as it fits the specific learning task we face, every loss function is the same. Indeed, any loss function defines and operationalizes a specific learning problem and defines a target to aspire to: Bayes predictors.

When we talk about aleatoric uncertainty, however, not all loss functions are the same: since our objective is to be able to recover $\mathcal{D}(x)$, which is necessary to be able to quantify the aleatoric uncertainty, we need to guarantee that (at least in the limit of infinite data) our Bayes predictors actually predict this quantity.

In previous lectures we showed that the 0-1 loss satisfies the above mentioned property only in a very weak sense: while the model that predicts, for every instance $x \in X$, $\mathcal{D}(x)$ is a Bayes predictor, it is not the only one. In contrast, for both the Brier loss and the cross-entropy loss, it can be proven that $h^*(x) = \mathcal{D}(x)$ is the unique Bayes predictor. What distinguishes these losses?

The property that sets apart the above mentioned losses is that of being a *strictly proper scoring rule* [17]. A loss function is a strictly proper scoring rule if the following holds:

$$\mathbb{E}_{y\sim\mathcal{D}(x)} l(x, y, \mathcal{D}(x)) < \mathbb{E}_{y\sim\mathcal{D}(x)} l(x, y, h) \tag{12}$$

for every $x \in X$ and every $h : X \to [0, 1]$ such that $\exists x \in X, h(x) \neq \mathcal{D}(x)$. That is, a strictly proper scoring rule guarantees that if we had access to infinite data sampled from $\mathcal{D}$ the unique Bayes predictor will actually predict the true conditional distribution. Both the Brier score and cross-entropy loss are strictly proper scoring rules, while the 0-1 loss is not. In general, the following result provides a sufficient and (almost) necessary condition for a loss function to be a strictly proper scoring rule:

Theorem 6. *Let l be a loss function. l is a strictly proper scoring rule if there exists differentiable and strictly convex (i.e., convex and such that $g(\alpha x + (1-\alpha)y = \alpha g(x) + (1-\alpha)g(y)$ if and only if $\alpha \in \{0, 1\}$) function g such that:*

$$l(x, 1, h) = g(h(x)) - h(x)g'(h(x)) + g'(h(x)) \tag{13}$$
$$l(x, 0, h) = g(1 - h(x)) - (1 - h(x))g'(1 - h(x)) + g'(1 - h(x)) \tag{14}$$

where g' is the derivative of g.

As an example, for the Brier score we have $g(p) = 1 - p^2$, $g'(p) = -2p$, while for the cross-entropy loss we have $g(p) = p - 1 + (p - 1)\log(1 - p) - p\log p$ and $g'(p) = 1 - \log p + \log(1 - p)$.

Given the above properties, in the following we will only consider loss functions that are strictly proper scoring rules. In particular, we will focus on the

Brier score, as it enjoys some additional properties. First, it is 2-times differentiable, 2-Lipschitz (i.e., $\forall x, |l(x,y,h) - l(x,y,g)| \leq 2|h(x) - g(x)|$), smooth, and strongly convex: these properties are particularly advantageous from a computational point of view. More interestingly, the following result holds:

Theorem 7. *Let $x \in X$, $\mathcal{D}$ be a distribution over $X \times Y$, and l_2 be the Brier score. Define the* aleatoric uncertainty *at x as $AU(x) = \mathcal{D}(x)(1 - \mathcal{D}(x))$. Then, it holds that*

$$\mathbb{E}_{y \sim \mathcal{D}(x)} \left[l_2(x, y, h^*) \right] = AU(x),$$

where $h^*(x) = \mathcal{D}(x)$ is the Bayes predictor with respect to l_2.

Thus, the true risk of the Bayes predictor with respect to the Brier score is the aleatoric uncertainty: this means, that we can reduce the problem of quantifying the aleatoric uncertainty to the problem of computing the Brier score! We can also expect that, if h is sufficiently close to h^*, then, $R(h)$ is a good approximation of the aleatoric uncertainty: thus, our goal will be to find a model that is sufficiently close to a Bayes predictor (for the Brier score).

4.2 Aleatoric Uncertainty, Optimal Models and Testing

In the previous section, we have seen that the Brier score ensures that, in the limit of infinite data and if the Bayes predictor belongs to our set of models $\mathcal{H}$, we would be able to obtain the conditional target distribution, and hence precisely estimate the aleatoric uncertainty, using ERM.

Even in the case of finite data, where we also have epistemic uncertainty, if the Bayes predictor h^* belongs to our set of models $\mathcal{H}$, then, the model obtained through ERM should be close to h^* and hence we should be able to obtain a good approximation to the aleatoric uncertainty. For example, we know that (with probability larger than $1 - \delta$):

$$\hat{R}_S(h_S) - 2b\sqrt{\frac{\log(1/\delta)}{2m}} \leq \mathbb{E}_{x \sim \mathcal{D}} AU(x) \leq \hat{R}_S(h_S) + 2b\sqrt{\frac{\log(1/\delta)}{2m}}.$$

However, this assumes that we know that h^* belongs to the set of model $\mathcal{H}$: if this does not hold then, by error decomposition and the No-Free Lunch Theorem, we know that we may not be able to get close to h^* and hence obtain a good approximation to the aleatoric uncertainty... furthermore, we do not even have a way to check whether $h^* \in \mathcal{H}$ or not!

This is, in some sense, a fundamental flaw of ERM when our goal is to be able to estimate the aleatoric uncertainty (while the epistemic uncertainty comes for free). In the following we will adopt an alternative approach that:

- Does not require the specification of an a-priori set of models $\mathcal{H}$... rather, we search through the space of all possible models;
- Is not based on ERM... rather, it employs a very simple test-and-modify algorithm.

In particular, this approach is based on the following three steps:

1. We identify some property P that the Bayes predictor h^* should satisfy... this property may be satisfied by other models, and in general we need to find a trade-off between specificity (fewer models) and feasibility;

2. We design an efficient algorithm T that, given an already existing model h tests, whether h satisfies the property P *using only finite data*;

3. If, according to the test T, model h does not have property P then it cannot be the Bayes predictor... however, we may apply some other efficient algorithm that modifies h by making it progressively closer to h^*.

Notice that we need to have a model h as a starting point: this can be any model, for example some naive baseline (e.g., always predict 0 for every instance) or a model that has already been trained (using some learning algorithm such as ERM)... in the latter case, however, we need to ensure that training of the model and the above procedure are executed using separate sets of data. The set of data used for steps 2 and 3 is typically called, for reasons that we'll become clear later, the *calibration set*.

As a side note, if you are familiar with cryptography, you may find the above described process to be reminiscent of what you would do to assess if some sequence (of numbers or text) is random: we think of a property that random sequences should have (e.g., values should be approximately uniformly distributed) and then test whether that property holds. This idea has become quite relevant in modern theoretical computer science and goes under the name of *property testing* [18] (though the general focus of this latter field is on obtaining algorithms with sublinear time complexity).

So, the first step would be to identify some relevant property satisfied by the Bayes predictor h^*.

4.2.1 Strong Validity

The first example is *(strong) validity*: we say that a model $h : X \to [0, 1]$ is *valid* if:

$$\forall x \in X, (h(x) - \mathcal{D}(x))^2 = 0.$$

This property seems a good candidate, since obviously h^* is valid and it is, in fact, the unique valid model... however, it is impossible to test validity using only finite data because we would need access to the data-generating process!

Just as an exercise, however, let us see what we could do having access to $\mathcal{D}$. Let $\Delta(x) = \mathcal{D}(x) - h(x)$. Then, define a new model $h'(x) = h(x) + \Delta(x)$. Then, the following holds:

Theorem 8. *h' defined as above is valid.*

Thus, step 2 in our procedure is simply to compute $\Delta(x)$ for each x: if all of them are 0 then the procedure stops, on the other hand, step 3 simply defines a new model by adding to $h(x)$ what it lacks to be the Bayes predictor... while,

in this case, this algorithms seems trivial, we will see that the same approach actually works in practice!

4.2.2 Marginal Mean Consistency

Given that validity is too strong of a property, we need to devise some weaker, but still relevant, properties. Our first tentative in this sense is *marginal mean consistency* [36]. We say that a model h has marginal mean consistency error α if:

$$MMCE(h) := |\mathbb{E}_{x\sim\mathcal{D}_X}[h(x)] - \mathbb{E}_{x,y\sim\mathcal{D}}[y]| = \alpha.$$

Thus, marginal mean consistency measures the distance between the average prediction made by model h and the average target value (across all instances).

Obviously, the Bayes predictor has marginal mean consistency error equal to 0... however, also other models may have this same property (in contrast to validity). For example, the model $h_M(x) = \mathbb{E}_{x,y\sim\mathcal{D}}[y]$ that always predicts the average target value is marginally mean consistent (even though all of its predictions are wrong)!

Thus, marginal mean consistency is a rather weak property, but is nonetheless a good first step: if a model is not even marginally mean consistent (that is, $MMCE > 0$), then surely it is not the Bayes predictor h^*!

Given a model h that is not marginally mean consistent, how can we improve it to satisfy this property? Define $\Delta = \mathbb{E}_{x,y\sim\mathcal{D}}[y] - \mathbb{E}_{x\sim\mathcal{D}_X}[h(x)]$ and $h'(x) = h(x) + \Delta$. Then, the following holds:

Theorem 9. *h' defined as above is marginally mean consistent. Furthermore, it holds that $R(h') = R(h) - \Delta^2 \leq R(h)$.*

Thus, not only making a model marginally mean consistent is easy, but doing so also gets us closer to the Bayes predictor... however, the above approach requires access to the data-generating process!

It is easy, however, to turn the above process into a practical algorithm that works on finite data, by simple replacing expectations with sample averages. This results in Algorithm 2. It is easy to show that the following result holds:

Theorem 10. *Let h' be the model returned by Algorithm 2 on any input model h and calibration set S of size m. Then, with probability larger than $1-\delta$, it holds that*

$$MMCE(h') \leq \sqrt{\frac{\log(1/\delta)}{2m}}.$$

Furthermore, Algorithm 2 runs in $\Theta(m)$ time and then requires $\Theta(1)$ for each prediction on a new instance.

Thus, we have shown that it is easy to guarantee marginal mean consistency... however, this property is relatively weak as shown previously. The main problem with marginal mean consistency is that it is a *marginal* property:

Algorithm 2 Algorithm for Marginal Mean Consistency improvement.

procedure MMC-IMPROVE($S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$: calibration set, h: model)
  $\hat{Delta} \leftarrow \frac{1}{m}\sum_{i=1}^{m} y_i - \frac{1}{m}\sum_{i=1}^{m} h(x_i)$
  Define $h' : X \to [0, 1]$ by $h'(x) = h(x) + \hat{\Delta}$
  return h'
end procedure

this means that it does not consider in any way the instances x, but only averaged population statistics. In contrast, validity is a *fully conditional* property, as it defines a constraint that should hold for every instance: this ensures that it is a strong property (only the Bayes predictor satisfies it) but, in practice, it is impossible to enforce it while having access only to finite data... we need some properties that bridge this gap!

4.2.3 Calibration

Calibration is a property that sits between marginal (e.g., marginal mean consistency) and fully conditional (i.e., validity). Technically, we say that a model h is *calibrated* if:

$$\mathbb{E}[y|h(x) = p] = p, \tag{15}$$

that is, if the average target value, conditioned on the model's prediction being equal to p, is also equal to p. If a model h is not calibrated, we say that its *calibration error* is equal to α if:

$$K_2(h) := \int \mathcal{D}_X(h(x) = p)\left(h(x) - \mathbb{E}[y|h(x) = p]\right)^2 = \alpha.$$

Calibration is an interesting property because it is obviously satisfied by the Bayes prediction and, furthermore, it enables us to interpret the predictions of our models as probabilities. So far we have considered models that predict numbers between $[0, 1]$ and implicitly assumed that these represent probabilities: this is also what you typically do when using neural networks or other models with a softmax... however, these numbers may not represent probabilities (in a frequentist sense[3])!

As an example, consider a model that, for a certain group of patients $C \subseteq X$, predicts that they will develop some disease with $h(x) = 70\%, \forall x \in C$... however, $P(disease|x = C) = 30\%$, that is only 30% of the patients in the group will actually develop the disease. Thus, the model overestimates the actual probability. Similarly, a model could also underestimate the actual probabilities

[3]In some sense, calibration is related to the subjective vs frequentist debate in probability theory. The predictions of ML models, as long as they are in $[0, 1]$ can always be interpreted as subjective probabilities because, in a sense, they represent the confidence of the model. However, in practice, we are interested about frequentist probabilities: if we make some prediction about the probability of an event, this should match the actual frequency of that event! The predictions of a model can be understood as probabilities (in the frequentist sense) if and only if the model is calibrated: for this reason, the model's predictions are usually called *confidence scores* in the Uncertainty in ML literature, rather than probability scores.

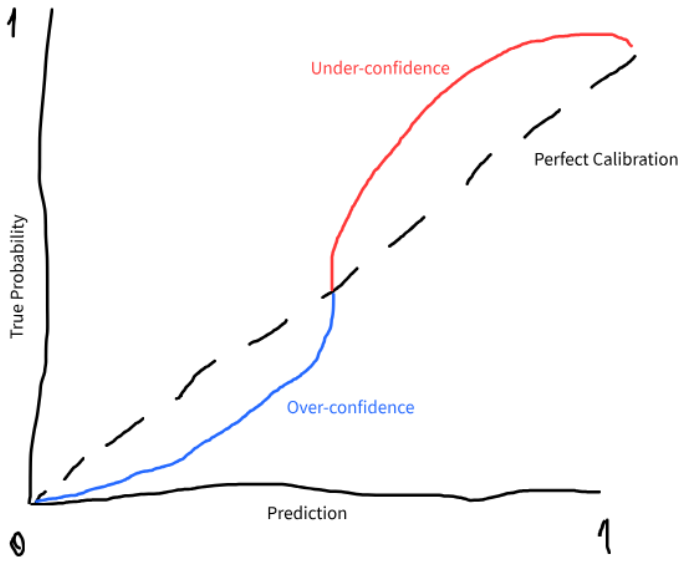

Figure 3: Calibration plot: it illustrates the relationship between a model's predictions and true probabilities

of an event... the important thing to note, however, is that both underestimation and overestimation of probabilities happen only when a model is not calibrated: if a model is calibrated, predicted scores correspond to actual probabilities! This issue is illustrated in Figure 3.

In the following, we will make the assumption that our models have finite range, that is the set

$$C(h) = \{p \in [0,1] | \exists x \in X, h(x) = p\}$$

has finite cardinality. In this case, the expectation in the definition of the calibration error K_2 can be replaced with an average, and we can easily test (in principle, if we had access to the data-generating process) whether a model is calibrated.

How we can modify a non-calibrated model to make it calibrated? The algorithm we will define is based on making isolated modifications to a model, based on an operation called *value patch*:

$$VP(h, p \mapsto p', x) = \begin{cases} p' & h(x) = p \\ h(x) & otherwise \end{cases}.$$

Intuitively, given an arbitrary value $p \in C(h)$, we know that if h was calibrated it should hold that $\mathbb{E}[y|h(x) = p] = p$... if this does not hold, then

h is not calibrated and we could reduce the calibration error of h by simply computing the value patch $VP(h, p \mapsto \mathbb{E}[y|h(x) = p])$: thus, we replace the current prediction of h with the corresponding correct probability. This simple idea can be easily turned into an algorithm by applying value patches iteratively and replacing expectations with sample averages. In particular, if we define the empirical calibration error as:

$$\sum_{p \in C(h_t)} \left(\frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} \right) \left(p - \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} y_i \right)^2, \tag{16}$$

we obtain Algorithm 3 [19, 21, 36].

Algorithm 3 Algorithm for Model Calibration.

procedure CALIBRATE($S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$: calibration set, h: model, ϵ: target calibration error)
  $h_0 \leftarrow h$
  $t \leftarrow 0$
  while $\sum_{p \in C(h_t)} \left(\frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} \right) \left(p - \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} y_i \right)^2 > \epsilon$ **do**
    $v \leftarrow \arg\max_{p \in C(h_t)} \left(\frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} \right) \left(p - \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} y_i \right)^2$
    $p' \leftarrow \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{h_t(x_i)=p} y_i$
    Define $h_{t+1} : X \to [0, 1]$ by $h_{t+1}(x) = VP(h_t, p \mapsto p', x)$
    $t \leftarrow t + 1$
  end while
  return h_t
end procedure

Algorithm 3 satisfies the following guarantees:

Theorem 11. *Let h' be the model given as output by Algorithm 3 on an arbitrary input h and S. Then, after $T \leq \frac{m}{\epsilon}$ iterations and with probability greater than $1 - \delta$ it holds that*

$$K_2(h') \in O\left(\epsilon + \sqrt{\frac{\log(\frac{1}{\epsilon^2 \delta})}{\epsilon^3 m}} + \frac{\log(\frac{1}{\epsilon^2 \delta})}{\epsilon^3 m} \right).$$

Furthermore, with probability higher than $1 - \delta$, it holds that $R(h') \leq R(h)$.

Thus, similarly to marginal mean consistency, we can ensure we obtain a model close to being calibrated by means of a relatively simple algorithm whose time complexity is $O(poly(m, \frac{1}{\epsilon}))$. Note, however, that since calibration is a stronger property than marginal mean consistency, it is harder to enforce it. Indeed, we are not guaranteed to obtain a calibrated model, but only a model that (with high probability) has calibration error lower than a certain threshold: the smaller the calibration error we are willing to tolerate, the harder it is to enforce the property using Algorithm 3.

As a final point, note that the efficiency of Algorithm 3 critically depends on $C(h)$ being finite and possibly small: otherwise, steps 4 and 5 in the Algorithm may be very costly! In practice, there are algorithms that can be used to

calibrate models when $C(h)$ is large or even infinite (see e.g. the recent survey [43])... these algorithms, however, do not come with theoretical guarantees (or do so only under additional assumptions), so there is a trade-off to be made!

4.2.4 Going Beyond Calibration

One natural question is by how much is calibration stronger than marginal mean consistency. The answer, however, is far from being trivial: for example if $|C(h)| = 1$ (i.e., h always predicts the same value) then calibration and marginal mean consistency actually coincide!

In general, however, calibration is a strictly stronger property and its strength depends in a non-trivial way on the size of the range of the models we consider.

In any case, calibration is in general strictly weaker than validity, so it may be of interest to consider other properties that bridge this gap. The most natural generalization in this sense is the notion of *multicalibration* (short for *multi-group calibration*) [21]. While calibration is still a marginal property[4], multicalibration is an attempt to bridge the gap towards full conditionality (i.e., validity) by requiring that calibration holds not just with respect to the whole population, but also with respect to some notable sub-groups.

As an example, assume we want to predict the probability of patients developing some disease. Calibration simply asks that for each prediction p made by the model, exactly $p * 100\%$ of the patients will actually develop the disease. However, if we stratify patients by biological sex (males, females, possibly along with intersex, if this may be clinically relevant) it could still hold that for some groups either fewer (or more) than $p * 100\%$ patients in that group will actually develop the disease. Hence, multicalibration requires that the above mentioned guarantee holds not only for all patients (disregarding biological sex), but also for males, females (and possibly intersex individuals) at the same time.

In general, multicalibration is defined with respect to a set of groups $\mathcal{G}$, where each group $g \in \mathcal{G}$ is $g \subseteq X$ (equivalently, $g : X \to \{0, 1\}$). The multicalibration error (with respect to $\mathcal{G}$) is defined as:

$$K_2(h, \mathcal{G}) := \sum_{g \in \mathcal{G}} \int \mathcal{D}_X(h(x) = p, x \in g) \left(h(x) - \mathbb{E}[y | h(x) = p, x \in g]\right)^2$$

and we say that a model h is multicalibrated when $K_2(h, \mathcal{G}) = 0$.

Multicalibration is actually a spectrum of properties that ranges from calibration (when $\mathcal{G} = X$) to validity (when $\mathcal{G} = \{\{x\} | x \in X\}$), as illustrated in Figure 4, and it can also be extended to more general models (e.g., regression models where $Y = \mathbb{R}$, in this case we talk about *multiaccuracy* [28]).

Interestingly, multicalibration can be tested and enforced using an algorithm which is very similar to Algorithm 3 [21, 36] (the only modification required is to add also the conditioning on the groups in steps 4, 5 and 6 of the algorithm,

[4]Indeed, calibration does not look into the instances, insofar we know the associated predictions made by the model.

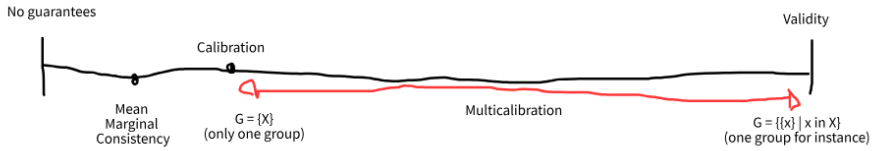

Figure 4: Spectrum of possible guarantees that could be required for a model $h : X \to [0, 1]$. Mean marginal consistency is the most basic property: stricter guarantees can be seen as refinements of marginal mean consistency, in which we ask the same guarantee to hold for finer and finer groups.

as well as slightly revising the definition of the Value Patch operation). For this reason, we won't cover in detail this topic, which has been discussed in several recent articles. Nonetheless, it is important to note that while guarantees similar to Theorem 11 can be given also for multicalibration, they have an additional dependency on $|\mathcal{G}|$: this is reasonable, as the more groups we want to control for, the harder the problem it becomes (because we are asking for guarantees that come progressively closer to validity).

4.3 Conformal Prediction: Uncertainty Quantification Beyond Probabilities

In the previous lectures we focused mostly on probabilistic classifiers: that is models that provide as output (something that can be interpreted as) probability distributions over the target space. This seems reasonable, since our goal is to recover the actual conditional target distribution.

However, in the literature, it has been argued that probability theory is not an adequate way to model and quantify and uncertainty when we have both epistemic and aleatoric uncertainty [27]. Indeed, if you notice, in our discussion of epistemic uncertainty, our estimates of uncertainty were always in the forms of intervals rather than point-wise probabilities.

A similar discourse has also be seen in statistics, where one can compare *point estimators* [30] and *interval estimators* [20] (e.g., confidence intervals). While point estimates are more useful when making decisions, interval estimates are typically better at conveying uncertainty and usually provide better guarantees when one is interested in prediction (and decision is undertaken as a separate step). This is particularly relevant in risk-sensitive applications (e.g., medicine, or legal decision-making), where typically we do not want AI and ML systems to make decisions on their own, and we are instead interested in conveying as best as possible the uncertainty to the human decision-makers.

In this lecture we will study a family of methods, called *conformal prediction*, whose aim is to apply the ideas of interval estimation to the realm of ML [2, 41, 47], and which has become quite popular in modern ML.

The central notion of conformal prediction is that of *non-conformity score*

$$s : (X \times Y)^m \times (X \times Y) \to \mathbb{R}.$$

Intuitively, a non-conformity score measures, on the basis of a *calibration set* $C = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, measures how *strange* (different, dissimilar, non-conforming) a new instance (x, y) looks relative to the previously seen data.

Given these elements, we define the *p-value* of (x, y), given C, to be the fraction of instances in C that are as strange as, or more, than (x, y) as measured by the non-conformity score:

$$p_{C,s}(x, y) = \frac{|\{i \in [1, m] : s(C \setminus \{(x_i, y_i)\} \cup \{(x, y)\}, (x_i, y_i)) \geq s(C, (x, y)\}| + 1}{m + 1} \tag{17}$$

Thus, the p-value of an instance is close to 0 when that instance is considered to be "very strange", while it will be close to 1 when the instance is quite "typical".

Finally, we define a *conformal predictor*, with respect to a non-conformity score s, to be a function $\Gamma : [0, 1] \times (X \times Y)^m \times X \to 2^Y$ defined as:

$$\Gamma^\epsilon_s(C, x) = \{y \in Y : p_{C,s}(x, y) \geq \epsilon\} \tag{18}$$

That is, a conformal predictor takes as input a calibration set C, a *confidence level* ϵ and an *unlabeled* instance x and gives as output the set of target values $\Gamma^\epsilon_s(C, x)$ whose p-value is greater than ϵ.

Intuitively, the set $\Gamma^\epsilon_s(C, x)$ should be interpreted as a *confidence set*: with high confidence, the true target value associated with x should be contained in the set. This interpretation is guaranteed by the following Theorem.

Theorem 12 ([1])**.** *Let $\mathcal{D}$ a data-generating process. Let s be a non-conformity score. Then, given a calibration set C sampled i.i.d. from $\mathcal{D}$ it holds that:*

$$P_{(x,y)\sim\mathcal{D}}(y \notin \Gamma^\epsilon_s(C, x)) \leq \epsilon + O(\frac{1}{m}).$$

The property above is called *(weak) validity* (or *coverage*) in the conformal prediction literature, but note that it is a considerably weaker property than strong validity, as we defined in the previous lectures (hence the prefix "weak"). Indeed, weak validity is a marginal property: it only guarantees that, on expectation with respect to all the population, the confidence sets produced by conformal prediction will fail to contain the correct target value at most a ϵ fraction of times. However, similarly to the difference between calibration and multicalibration, there exist extensions of conformal prediction that provide also conditional guarantees [16].

Note that Theorem 12 holds irrespective of the non-conformity score, this gives considerable freedom in designing non-conformity score that can optimize different resources. Most relevantly, the non-conformity score influences the *size* of the sets Γ^ϵ_s: obviously, sets that are too large are not particularly useful (consider the trivial conformal predictor that always gives as output the full

set of target values Y), and thus we are typically interested in designing non-conformity scores that minimize the expected confidence set size. Notice that this stands in contrast with the standard approach we take when we design ML algorithms: in this latter case, our focus is on designing methods that can reach good accuracy. With conformal prediction, instead, accuracy comes for free due to Theorem 11, however we want to make our confidence sets to be as "specific" (small) as possible.

A natural question, at this point, is which non-conformity scores one should use. In practice, there is particularly popular and effective strategy. Let $h : X \to [0,1]^Y$ be a ML model that has been previously trained on a training set S separate from C. Then, we set s_h to be the non-conformity score defined as:

$$s_h(C,(x,y)) = 1 - h(x)^y,$$

that is, the non-conformity score for instance (x,y) is simply the complement of the confidence score that h assigns to label y. Thus, intuitively, the higher the confidence score $h(x)^y$ the more typical (with respect to data-generating process) the instance (x,y) is. When using the non-conformity score s_h, the conformal predictor Γ_s is usually called *inductive conformal predictor*. Note that s_h does not consider the calibration set C, so its computation is particularly efficient.

More generally, in regard to computational complexity, conformal prediction is a *lazy approach*, in the sense that there is no proper training of a model, but rather all work (for constructing the confidence sets Γ^ϵ) is performed at inference time. In general, if S is the time complexity required to compute the non-conformity score s, then conformal prediction requires time $\Theta(mS + m|Y| + S|Y|)$. In the case of inductive conformal prediction, if we pre-compute the non-conformity scores of the instances in the calibration set and sort them (this requires total time $O(m + m\log m)$, as $S = O(1)$), then the complexity at inference time can be reduced to $O(|Y|\log m)$.

5 Data Uncertainty, or Learning with Imperfect Data

In the previous lectures we focused on epistemic and aleatoric uncertainty. The crucial assumption, in both cases, is that our data are perfect: they may be limited, but they always are a *complete* and *truthful* representation of the phenomenon of interest.

In practice, however, this may not hold: data can have gaps (missing data), may be wrong or erroneous (noisy data), or may be only partially known (imprecise data). These issue can make learning much more difficult, as conventional ML algorithms may not satisfy their usual guarantees under such conditions (or, in some cases, may not be even used at all!). In this lecture we will try to understand how can we learn effectively when we have such imperfect data, focusing on the case of *missingness*.

5.1 Missing Features

When working with real-world datasets, it often happens that data are not complete: there may be some values that are not observed and hence are *missing*. An example of this issue is illustrated in Table 1.

Table 1: An example of a dataset with missing feature values.

Headache	Fever	High Pressure	Flu?
Y	N	Y	Y
N	N	$\perp$	N
$\perp$	Y	$\perp$	Y

Data may be missing for several different reasons:

- Non-systematic missing: data is missing due to essentially random reasons (e.g., a measurement instrument sometimes not working);
- Systematic missing: data is missing for some specific reason. An example, illustrated in Table 1, would be not measuring the blood pressure to a patient that has no other specific symptoms.

In any case, the possibility of missing data, implies that we can no longer rely on the simplified data generation assumptions we made so far.

Let us assume that $X = \mathbb{R}^d$ and denote with X_i the i-th feature. Then, we assume that, for each feature it exists a distribution $\mathcal{M}_i$ defined over $X_1 \times \ldots X_i \times \ldots X_d \times Y \times X_i \cup \{\perp\}$: the symbol $\perp$ denotes a missing value. We will be particularly interested in the conditional distributions $\mathcal{M}_i(\perp | X_1 = x_1, \ldots, X_i = x_i, \ldots, X_d = x_d, Y = y)$, that is the probabilty of observing a missing value in the i-th feature given that the true instance was $(x_1, \ldots, x_i, \ldots, x_d, y)$.

Given this setup, we assume that our data is generated according to the following two-step mechanism:

1. First, we sample a (complete) instance $x = (x_1, \ldots, x_d, y)$ according to the data-generating process $\mathcal{D}$;
2. Then, for each feature i, the corresponding value x_i is either set to $\perp$ (with probability $\mathcal{M}_i(\perp | X_1 = x_1, \ldots, X_i = x_i, \ldots, X_d = x_d, Y = y)$), or its original value is kept.

Importantly, we can distinguish three different types of missing data, depending on how the conditional distributions $\mathcal{M}_i(\perp | X_1 = x_1, \ldots, X_i = x_i, \ldots, X_d = x_d, Y = y)$ are allowed to depend on the actual values of the underlying instance (see also Figure 5):

- *Missing Completely at Random* (MCAR): if $\mathcal{M}_i(\perp | X_1 = x_1, \ldots, X_i = x_i, \ldots, X_d = x_d, Y = y) = \mathcal{M}_i(\perp)$. That is, the probability of observing a missing value in feature i is completely independent of the underlying instance x and only depends on the feature index;

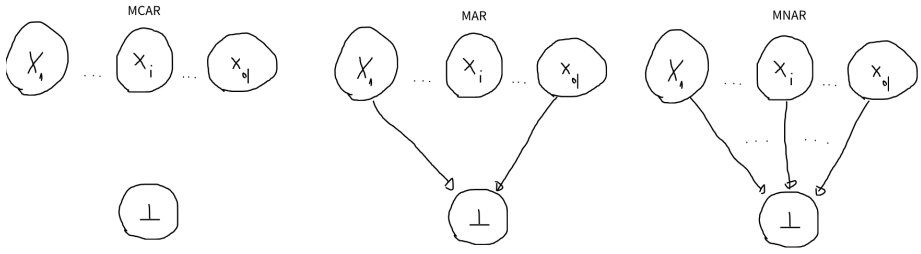

Figure 5: An illustration of the three types of missing data.

- *Missing at Random* (MAR): if $\mathcal{M}_i(\perp|X_1 = x_1, \ldots, X_i = x_i, \ldots, X_d = x_d, Y = y) = \mathcal{M}_i(\perp|X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_d = x_d, Y = y)$. That is, the probability of observing a missing value in feature i can depend on all the other features (or the target value) but not on feature i itself;
- *Missing Not at Random* (MNAR): if the above do not hold. That is, the probability of observing a missing value in feature i could also depend on the unobserved value x_i itself.

These three types of missing data have different implications about the hardness of learning with missing data, as well as about the ways learning can be done in practice. Obviously, we can expect MAR to be harder than MCAR, and MNAR, in turn, to be harder than both MAR and MCAR.

In general, we may ask: how can we learn when some data is missing? Traditional ML algorithms (think of backpropagation in neural networks) cannot be applied when data is not complete! In the following we will examine several popular approaches proposed in the literature, and analyze under which conditions we can expect them to work (and with which guarantees).

5.1.1 Listwise Deletion

The simplest way to address the issue of missing data is *listwise deletion* (also called, *complete data analysis*): we simply discard all instances having at least one missing value, and then apply our learning algorithms as if all data were completely observed. This approach is intuitively wasteful: we are throwing away a lot of potentially useful data. However, at least, we know that under certain assumptions learning after listwise deletion is the same as learning if the data were completely observed from the start:

Theorem 13. *Assume that the missing data is MCAR. Then, with probability larger than $1 - \delta$ (over the sampling of a training set S of size m) it holds that*

$$R(h_S) - R(h^*) \leq 2b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2|S_C|}}, \tag{19}$$

where S_C is the subset of S having only complete data. In particular, assuming $\exists p \in [0,1], \forall i \in [1,d], \mathcal{M}_i(\perp) = p$, then the right-hand expression above is approximately equal to:

$$R(h_S) - R(h^*) \leq 2b\sqrt{\frac{\log(|\mathcal{H}|/\delta)}{2m(1-p)^d}}. \tag{20}$$

Furthermore, learning can performed with an additional time complexity penalty of $\Theta(md)$.

Thus, listwise deletion enables using traditional learning algorithms, at the expense of throwing away a portion of the data.

However, this approach has several issues. First of all, as already observed above, listwise deletion can be extremely wasteful, and Theorem 13 quantifies this waste: in general, penalty we pay (in terms of additional true risk our models can suffer) increases exponentially with the data dimensionality (note, no explicit dependency on the dimensionality exists in the complete data setting)!

Furthermore, listwise deletion is guaranteed to work only when the missing data is MCAR. This assumption may be unrealistic in many cases: measurement instruments having larger probability of failure on *anomalous* values (MNAR), doctors not recording *normal* values (MNAR), values that are not recorded because they are irrelevant given the other values (MAR or MNAR). In all such cases, listwise deletion can lead to erroneous results and we risk under-estimating the true risk of our models.

5.1.2 Imputation

A more general approach, which aims at addressing the limitations of listwise deletion, is *imputation*: in this case, rather than discarding incomplete instances, we attempt to reconstruct the unobserved values. Imputation is by far the most common approach to deal with missing data: the general idea is to first replace the missing data with a plausible guess about the unobserved value, and then training a ML model by means of traditional learning algorithms.

Common approaches in this sense include mean (for continuous data) and mode (for categorical data) imputation, in which all missing values in a given feature are replaced with the corresponding mean (or mode), computed considering only the complete data. While this approach can sometimes work well, it generally fails when the data is not MCAR, and, even in this latter case, its properties are not clear!

In this lectures, we will consider a more general, and currently more popular, approach called *regression imputation*. The general idea of regression imputation is the following:

1. First, train, for each feature X_i, a regression model h_i that predicts the value of X_i based on the other features. Obviously, each h_i is trained only on the set of complete data (similarly to listwise deletion);

2. Second, impute (that is, fill) the missing data in each feature using the trained regression models;

3. Finally, train the desired ML model on the imputed data.

In practice, steps 1 and 2 in the above procedure must be repeated iteratively when multiple features can be missing at the same time: this means that while at the first round only the complete data are used to train the regression models, subsequent rounds will employ all the data (using the temporary imputed values). This approach, for example, is adopted by the MICE algorithm [46] and illustrated in Algorithm 4.

Algorithm 4 Algorithm for Regression Imputation.

procedure IMPUTE($S = \{(x^1, y^1), \ldots, (x^m, y^m)\}$: dataset, $\mathcal{R}$: set of regression models, $\mathcal{H}$: set of models, n: number of imputation steps)
  $S_1 \leftarrow S$
  for $k = 1$ to n **do**
    for $i = 1$ to d **do**
      Create dataset S' from S_k by replacing, for all features $j \neq i$, $\perp$ with $mean(X_j)$
      Set the i-th feature of S' equal to that of S
      Discard from S' all instances for which the i-th feature is equal to $\perp$
      $r_i^k \leftarrow$ train on S' a regression model from $\mathcal{R}$ to predict feature i
      Create dataset S_{k+1} from S by replacing $\perp$ in feature i with the predictions of r_i^k
    end for
  end for
  $h \leftarrow$ train a model from $\mathcal{H}$ on S_k
  return $h, r_1^n, \ldots, r_d^n$
end procedure

Algorithm 4 is conceptually simple: we try to use the information in the complete values to reconstruct the unobserved ones. Therefore, conceptually, we may expect the approach to work when the missing data is MCAR or MAR (when it is MNAR, the missingness can also depend on the unobserved values, and thus only using the information in the other features is not sufficient), as long as the regression models are able to reconstruct a good approximation of the real values. However, under which conditions does the approach really work?

To this end, we define the notion of *noise stability*, which encodes the intuition above. Let $\mathcal{D}$ be a data-generating process over $X \times Y$ and assume that X is a d-dimensional vector space. Let $Corrupt : X \times Y \mapsto X \times Y$ be a function, such that

$$\mathbb{E}[|(x, y) - Corrupt(x, y)|] \leq \epsilon. \tag{21}$$

Intuitively, the function *Corrupt* defines, in an abstract way, the action of an imputation function. Therefore, Eq. (21) expresses the condition that the imputation function should not introduce too much noise.

A learning algorithm A for a set of models $\mathcal{H}$ is ϵ-noise stable, if, for any $\delta > 0$, with probability larger than $1-\delta$, given a dataset $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ sampled from $\mathcal{D}$ and $T = \{Corrupt(x_i, y_i) | (x_i, y_i) \in S\}$, and defining $h = A(S), g = A(T)$, it holds that

$$\|h - g\| \leq p_{\mathcal{H}}(\epsilon, m, d, \delta) \in poly(\epsilon, d, \delta) \cdot o(\sqrt{m}). \tag{22}$$

That is, a learning algorithm is noise stable if, given two datasets that (on average) are not too different from each other, the models obtained by training on the datasets will not be too different from each other (with high probability): in particular, this deviation is bounded by a function which depends on (and does not grow too fast with) the problem's parameters.

The notion of noise stability enables us to characterize under which conditions Algorithm 4 is expected to work well.

Theorem 14. *Assume the following:*

- *All models in Algorithm 4 are trained using ERM;*
- *The regression models used for imputation are trained using the a loss function bounded in* $[0, b_1]$*;*
- *The final model is trained using a loss function* l *that is bounded in the range* $[0, b_2]$ *and* L*-Lipschitz;*
- $\mathcal{H}$ *is such that ERM is* ϵ*-noise stable;*
- $h^* \in \mathcal{H}$ *(i.e., the approximation error is 0).*

Additionally, for simplicity, assume that both $\mathcal{H}$ *and* $\mathcal{R}$ *are finite sets of models. Then, with probability larger than* $1 - \delta$ *it holds that:*

$$R(h) - R(h^*) \leq 2b_2\sqrt{\frac{\log(\frac{2|\mathcal{H}|}{\delta})}{2m}} + L \cdot p_{\mathcal{H}}(d \max_{i \in [1,d]} S_C^i(r_i^n) + db_1\sqrt{\frac{\log(\frac{2d|\mathcal{R}|}{\delta})}{2m}}, m, d),$$

where h, r_i^n *are the the ouputs of Algorithm 4, and* S_C^i *is the subset of* S *with no missing values in feature* i*.*

Additionally, if $T(\mathcal{R}, m, d)$ *and* $T(\mathcal{H}, m, d)$ *are the time complexities for training models* $\mathcal{R}$ *and* $\mathcal{H}$*, then the time complexity of Algorithm 4 is*

$$O(kd \cdot T(\mathcal{R}, m, d) + T(\mathcal{H}, m, d)).$$

Thus, if ERM is noise stable for our set of models, then regression imputation works with guarantees similar to those that hold when learning from complete data. Note, however, that compared to the latter case, with regression imputation we pay an additional penalty. Indeed, we pay a penalty $p_{\mathcal{H}}(\epsilon, m, d)$ which derives from the fact that our imputed data will not in general be exactly equal to the real unobserved data, but will be rather contaminated with a noise of size ϵ. This penalty is controlled by the stability of our learning algorithm under noise (i.e., the term $p_{\mathcal{H}}(\epsilon, m, d)$), as well as by the complexity of the set of regression models $\mathcal{R}$ (which governs the term ϵ).

This latter dependency, in particular, is far from trivial: a larger set of models will reduce the component $d \max_{i \in [1,d]} S_C^i(r_i^n)$ while at the same time increasing the component $db_1\sqrt{\frac{\log(\frac{2d|\mathcal{R}|}{\delta})}{2m}}$. Note, however, that simply having the Bayes predictor to be in $\mathcal{R}$ does not suffice to guarantee that we will able

to learn a good model after imputation. In practice, we need to guarantee that each of the features can be reconstructed having only information about the other features: in other words, regression imputation is guaranteed to work well only under the MAR assumption!

A natural question, at this point, is whether there exist learning algorithms that are noise stable. The following results shows that there exists at least one model that satisfies this guarantee and characterizes the stability $p_{\mathcal{H}}$.

Theorem 15. *Let $\mathcal{H}$ be the class of linear models, l be the Brier score, and consider the ERM learning algorithm. Assume that $X \subseteq \mathbb{R}^d$ and all instances are normalized (i.e., $\|x\| = 1$) and all features are centered (i.e., $\mathbb{E}[X_i] = 0$). Assume that the precision matrix (the inverse of the correlation matrix)*

$$(\mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)])^{-1}$$

is sparse (i.e., the number of non-zero entries is $o(m^2)$). Then, ERM is ϵ-noise stable for $\mathcal{H}$ with

$$p_{\mathcal{H}}(\epsilon, m, d) \leq 3d^2\sqrt{\frac{2 * \log(2/v)}{m}} + d\epsilon.$$

So, there exists at least one class of models (namely, OLS linear regression... however, the result above can be extended to every RKHS of models) that is learnable after imputation (provided the imputation is good enough). Note, however, that the complexity of the problem grows with the dimensionality of our data: this is a manifestation of the so-called *curse of dimensionality*! Thus, in practice, applying dimensionality reduction or feature selection can be crucial for obtaining good results: though traditional approaches cannot be applied (as they require complete data), efficient algorithms exists for this problem [7, 11]!

An additional limitation of regression imputation is that the above results are guaranteed to hold only when the missing data is not MNAR: unfortunately, as hinted at above, in many settings this assumption is clearly too strong... so what can we do when our data is MNAR?

In practice, the above question is even more important than it seems, as it is impossible to check whether the missing values in a given dataset are MCAR, MAR or MNAR! We can illustrate this issue with a simple example:

- Let $X = \mathbb{R}^2$;
- Consider the data-generating process $\mathcal{D}_1$ such that $\mathcal{D}(x_1, x_2) > 0$ if and only if $x_2 \geq 0$, and assume that missing values can occur only in feature X_2 with missingness process $\mathcal{M}_2(\perp|x_1, x_2) = 0.5$ (i.e., we have probability 0.5 of not observing the value of x_2, irrespective of anything). Then, clearly the missing data are MCAR;
- Consider, in contrast, the data-generating process $\mathcal{D}_2$ such that $\forall(x_1, x_2)$ it holds $\mathcal{D}(x_1, x_2) > 0$. Assume that missing values can occur only in feature X_2 with missingness process $\mathcal{M}_2(\perp|x_1, x_2 < 0) = 1$ (we never observe negative values in x_2) and $\mathcal{M}_2(\perp|x_1, x_2 \geq 0) = 0.5$. Then, clearly the missing data are MNAR;

- Crucially, given two datasets S and T drawn, respectively, from the two processes described above we have no way to tell which process generated which dataset!

There are basically two ways to deal with MNAR data:

1. The first approach is to gather some information about the process that generates the missingness and then incorporating this information in the way we train models: some popular strategies in this sense are *likelihood-based models* [14, 31] and *multiple imputation* [45]. Multiple imputation (which is used, for example, by MICE) is particularly attractive because it allows us to decouple the management of missing data and model training;
2. The second approach is to perform a worst-case analysis: that is, we try to understand how our models fare under the worst possible imputation of the data.

In the following section we will show how this second approach works, by exemplifying it in the case of missing target values (rather than missing data).

5.2 Missing Targets

While in the previous section we talked of missing data as if this issue could only affect features, in practice missing values can also appear in the target feature: that is, we could miss information about the thing we would like to predict! This problem is called *semi-supervised learning* [9] in the literature.

Table 2: An example of a semi-supervised datasets.

Headache	Fever	High Pressure	Flu?
Y	N	Y	$\perp$
N	N	Y	N
N	Y	N	$\perp$
Y	N	N	N

From a theoretical perspective, there is no distinction between these two cases: everything we said about missing features can equally be applied to missing targets: this includes the distinction between MCAR/MAR/MNAR as well as the approaches we described to address the problem. From the practical point of view, however, the two problems are quite different from each other:

- In semi-supervised learning we have all the information which would be needed to use our models: however, in some cases, we do not know what these models should say;
- In semi-supervised learning we are not typically interested in imputation: that is, we do not need to "fill the gaps" in our data; rather, we want to somehow use the information in all the available data (including the data without an associated target value) to train a model. Thus, we are typically more interested in *induction* rather than *transduction* [15].

So, given these differences, how can we deal with semi-supervised learning? A first approach would be to adapt the notion of regression imputation to this setting: rather than reconstructing missing data, we try to guess the unobserved target values and then use this guesses to train a ML model. This approach is called *pseudo-label learning* [29] and is illustrated in Algorithm 5.

Algorithm 5 Algorithm for Pseudo-Label Learning.

procedure PSEUDO-LABEL LEARNING($S_C = \{(x_1, y_1), \ldots, (x_m, y_m)\}$: dataset with observed target values, $S_I = \{c_1, \ldots, c_m\}$: dataset with unobserved target values, $\mathcal{H}$: set of models, n: number of iterations, P: condition for inclusion)
    $S_1 \leftarrow S_C$
    for $k = 1$ to $n - 1$ **do**
        Train a model h_k on S_k
        $S' = S_k \cup \{(c_i, h(c_i)) | c_i \in S_I\}$
        Create dataset S_{k+1} from S' by discarding all cases $(c_i, h(c_i))$ that do not satisfy P
    end for
    $h \leftarrow$ train a model from $\mathcal{H}$ on S_k
    return h
end procedure

Note that Algorithm 5 requires specifying a condition P: this tells us which of the model's predictions should be considered "sure enough" so that they could be used for training our subsequent models. A common such condition is that the epistemic uncertainty $EU(\mathcal{H}, S)(c_i)$ at c_i is lower than some threshold value. Aside from the difficulty to select a good inclusion condition, one of the main problems of pseudo-label learning is that it is hard to provide theoretical guarantees for it.

We will examine a different approach, called *generalized risk minimization* (GRM) [5, 8, 6, 24, 32], which is the approach that has been more widely studied in the (theoretical) literature. The idea behind GRM is to generalize the ERM algorithm to cases in which we cannot directly evaluate the loss function, since we are not able to observe the true target value. This idea is formalized by means of *aggregation functions* [8]. An aggregation function $A : \mathbb{R}^k \to \mathbb{R}$ is a map that takes a sequence of numbers and returns a single number, such that

$$\min_i v_i \leq A(v_1, \ldots, v_k) \leq \max_i v_k.$$

Common examples of aggregation functions include the minimum, the maximum and the average.

Let $Z = Y \cup \{\bot\}$. Aggregation functions provide a way to extend a loss function $l : X \times Y \times \mathcal{H}$ to a generalized loss function $l^A : X \times Z \times \mathcal{H}$ as follows:

$$l^A(x, z, h) = \begin{cases} l(x, y, h) & z = y \in Y \\ A((x, y_1, h), \ldots, (x, y_{|Y|}, h)) & z = \bot. \end{cases} \tag{23}$$

Thus, the generalized loss function l^A coincides with l on supervised instances, and, on the other hand, is equal to the application of the aggregation function A to all possible loss values when the instance is not supervised. Intuitively, to evaluate a model on a non-supervised instance, we simply consider all

possible target values, evaluate the model on each of these possible alternatives and then aggregate these values to obtain a single one.

We define the generalized risk minimization algorithm as any algorithm satisfying the following property:

$$GRM(\mathcal{H}, S, A) \in \arg\min_{h \in \mathcal{H}} R_S^A(h) = \arg\min_{h \in \mathcal{H}} \frac{1}{m} \sum_i l^A(x_i, z_i, h).$$

Notice that while we evaluate and select models based on the generalized risk $l^A(x_i, z_i, h)$ we still would like to provide guarantees on the true risk $R(h)$, that is the risk computed with respect to the true unobserved target values. Similarly to what we have seen in the previous lectures, we are thus interested in bounding the quantity:

$$R(h) - R(h^*).$$

The following theorem provides such a bound for two commonly adopted aggregation functions:

Theorem 16 ([8, 32])**.** *Let S be a semi-supervised dataset and assume that we select a model $h \in \mathcal{H}$ using $GRM(\mathcal{H}, S, A)$. Assume that $\mathcal{H}$ is finite, $h^* = \arg\min R(f) \in \mathcal{H}$ and $R(h^*) = 0$*[5]*. Assume that l is bounded in $[0, b]$. For every $(x, z) \in S$, let y_x be the unique $y \in Y$ s.t. $\mathcal{D}(x, y) > 0$. Then, with probability larger than $1 - \delta$ the following hold:*

- *If $A = \max$, $R(h) \leq R_S^{\max}(h) + b\sqrt{\frac{\log(\mathcal{H}/\delta)}{2m}}$;*
- *Assume, further, that $l = l_{0-1}$. Then, if $A = \min$, it holds that*

  $$R(h) \leq +R_S^{\min}(h) + \frac{4}{\theta m} * (\log(|\mathcal{H}|) * (\log(4 \log |\mathcal{H}|) + 2 \log |Y|) + \log(1/\delta) + 1),$$

  where $\theta = \log(\frac{2}{1+\gamma})$ and $\gamma = \sup_{(x,y) s.t. \mathcal{D}(x,y)>0} P((x, \perp))$ (that is, γ is the maximum probability, among all instances (x, y), with which y is not observed).

The theorem above shows conditions under which we can learn through GRM, with minimum and maximum aggregation operators: these versions of GRM are called *optimistic risk minimization* and *pessimistic risk minimization* in the literature [25]. The two cases provide quite different guarantees:

- The guarantee for pessimistic risk minimization holds with no (explicit) assumption about the data-generating process. Furthermore, it provides a worst-case guarantee: since the true risk (for any instance) is always lower than the maximum risk (across all possible labels), we can upper bound the true risk by the empirical generalized risk. This means that, in

[5] Note that this implies that $\forall x, \exists! y$ s.t. $\mathcal{D}(x, y) > 0$: that is, the target assignment is deterministic.

general, learning through pessimistic risk minimization provides models that are *robust*, in the sense that they are worst-case optimal. However, in practice the term $R_S^{\max}(h)$ can be quite large!

- By contrast, the guarantee for optimistic risk minimization holds only conditional on an additional assumption about the data-generating process: namely, the probability with which we are not able to observe the true targets is not too large. This is intuitively reasonable: if the supervision signal is sparse, then learning is hard. At the same time, when the above assumption is met, optimistic risk minimization provides a much more favorable guarantee: indeed, the is expected to decrease at a rate $\frac{1}{m}$ (rather than $\frac{1}{\sqrt{m}}$) and generally $R_S^{\min}(h) < R_S^{\max}(h)$.

In practice, one should understand whether for the problem at hand it is better to have a worse guarantee (which, however, is robust, worst-case optimal and holds unconditionally) or a better guarantee that, however, holds only conditionally on some assumptions: in the first case, pessimistic risk minimization is clearly preferable; in the second, one should favor optimistic risk minimization.

Note, crucially, that optimistic risk minimization suffers from two additional limitations (as compared with pessimistic risk minimization). First, since we do not know $\mathcal{D}$, we have no way to say whether the assumption required for the bound in Theorem 16 to work actually holds: this means that, in practice, optimistic risk minimization can greatly underestimate the true risk. By contrast, pessimistic risk minimization always provides a conservative estimate of the true risk. The second limitation concerns the computational complexity, as shown by the following Theorem.

Theorem 17 ([6, 8])**.** *Let l be a L-Lipschitz convex loss function, $\mathcal{H}$ be set of models. Then, the following hold:*

- *Let $A = \max$. Then l^A is $|Y|$L-Lipschitz convex loss function. In particular, if $ERM(\mathcal{H}, \cdot)$ for $\mathcal{H}$ admits a polynomial-time algorithm, then also $GRM(\mathcal{H}, \cdot, \max)$ does;*
- *Let $A = \min$. Then, in general l^A is not convex (this holds already for the case of linear regression with the Brier score [6]). In general, irrespective of the time complexity of $ERM(\mathcal{H}, \cdot)$, the problem of computing $GRM(\mathcal{H}, \cdot, \min)$ is `NP-HARD`.*

Thus, while pessimistic risk minimization generally admits efficient algorithms, this is not true for optimistic risk minimization (this is even more true in reference to Theorem 16 which, for optimistic risk minimization, holds only for the l_{0-1} loss).

References

[1] A. N. Angelopoulos, R. F. Barber, and S. Bates. Theoretical foundations of conformal prediction. *arXiv preprint arXiv:2411.11824*, 2024.

[2] A. N. Angelopoulos, S. Bates, et al. Conformal prediction: A gentle introduction. *Foundations and Trends® in Machine Learning*, 16(4):494–591, 2023.

[3] F. Bach. *Learning theory from first principles*. MIT press, 2024.

[4] E. Black, M. Raghavan, and S. Barocas. Model multiplicity: Opportunities, concerns, and solutions. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 850–863, 2022.

[5] A. Campagner. Learnability in "learning from fuzzy labels". In *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2021.

[6] A. Campagner. Learning from fuzzy labels: Theoretical issues and algorithmic solutions. *International Journal of Approximate Reasoning*, 171:108969, 2024.

[7] A. Campagner, D. Ciucci, and E. Hüllermeier. Rough set-based feature selection for weakly labeled data. *International Journal of Approximate Reasoning*, 136:150–167, 2021.

[8] A. Campagner et al. Credal learning: Weakly supervised learning from credal sets. *FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS*, 372:327–334, 2023.

[9] O. Chapelle, B. Scholkopf, and A. Zien, editors. *Semi-supervised learning*. MIT Press, 2006.

[10] A. D'Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 23(226):1–61, 2022.

[11] C. De Bodt, D. Mulders, M. Verleysen, and J. A. Lee. Nonlinear dimensionality reduction with missing data using parametric multiple imputations. *IEEE Transactions on Neural Networks and Learning Systems*, 30(4):1166–1179, 2018.

[12] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft. Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning. In *International conference on machine learning*, pages 1184–1193. PMLR, 2018.

[13] R. Duan, B. Caffo, H. X. Bai, H. I. Sair, and C. Jones. Evidential uncertainty quantification: A variance-based perspective. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2132–2141, 2024.

[14] C. K. Enders. A primer on maximum likelihood algorithms available for use with missing data. *Structural Equation Modeling*, 8(1):128–141, 2001.

[15] A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. *arXiv preprint arXiv:1301.7375*, 2013.

[16] I. Gibbs, J. J. Cherian, and E. J. Candès. Conformal prediction with conditional guarantees. *arXiv preprint arXiv:2305.12616*, 2023.

[17] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

[18] O. Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[19] P. Gopalan, A. T. Kalai, O. Reingold, V. Sharan, and U. Wieder. Omnipredictors. *arXiv preprint arXiv:2109.05389*, 2021.

[20] G. J. Hahn and W. Q. Meeker. *Statistical intervals: a guide for practitioners*, volume 92. John Wiley & Sons, 2011.

[21] U. Hébert-Johnson, M. Kim, O. Reingold, and G. Rothblum. Multicalibration: Calibration for the (computationally-identifiable) masses. In *International Conference on Machine Learning*, pages 1939–1948. PMLR, 2018.

[22] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[23] P. Hofman, Y. Sale, and E. Hüllermeier. Quantifying aleatoric and epistemic uncertainty: A credal approach. In *ICML 2024 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2024.

[24] E. Hüllermeier. Learning from imprecise and fuzzy observations: Data disambiguation through generalized loss minimization. *International Journal of Approximate Reasoning*, 55(7):1519–1534, 2014.

[25] E. Hüllermeier, S. Destercke, and I. Couso. Learning from imprecise data: adjustments of optimistic and pessimistic variants. In *Scalable Uncertainty Management: 13th International Conference, SUM 2019, Compiègne, France, December 16–18, 2019, Proceedings 13*, pages 266–279. Springer, 2019.

[26] E. Hüllermeier, S. Destercke, and M. H. Shaker. Quantification of credal uncertainty in machine learning: A critical analysis and empirical comparison. In *Uncertainty in Artificial Intelligence*, pages 548–557. PMLR, 2022.

[27] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

[28] M. P. Kim, A. Ghorbani, and J. Zou. Multiaccuracy: Black-box post-processing for fairness in classification. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 247–254, 2019.

[29] D.-H. Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896. Atlanta, 2013.

[30] E. L. Lehmann and G. Casella. *Theory of point estimation*. Springer Science & Business Media, 2006.

[31] R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[32] L. Liu and T. Dietterich. Learnability of the superset label learning problem. In *International conference on machine learning*, pages 1629–1637. PMLR, 2014.

[33] J. H. Manton, P.-O. Amblard, et al. A primer on reproducing kernel hilbert spaces. *Foundations and Trends® in Signal Processing*, 8(1–2):1–126, 2015.

[34] T. Mortier, V. Bengs, E. Hüllermeier, S. Luca, and W. Waegeman. On the calibration of probabilistic classifier sets. In *International Conference on Artificial Intelligence and Statistics*, pages 8857–8870. PMLR, 2023.

[35] A. Rakhlin, O. Shamir, and K. Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*, 2011.

[36] A. Roth. Uncertain: Modern topics in uncertainty estimation. *Unpublished Lecture Notes*, page 2, 2022.

[37] Y. Sale, M. Caprio, and E. Höllermeier. Is the volume of a credal set a good measure for epistemic uncertainty? In *Uncertainty in Artificial Intelligence*, pages 1795–1804. PMLR, 2023.

[38] Y. Sale, P. Hofman, L. Wimmer, E. Hüllermeier, and T. Nagler. Second-order uncertainty quantification: Variance-based measures. *arXiv preprint arXiv:2401.00276*, 2023.

[39] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2018.

[40] M. Sensoy, L. Kaplan, and M. Kandemir. Evidential deep learning to quantify classification uncertainty. *Advances in neural information processing systems*, 31, 2018.

[41] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

[42] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[43] T. Silva Filho, H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull, and P. Flach. Classifier calibration: a survey on how to assess and improve predicted class probabilities. *Machine Learning*, 112(9):3211–3260, 2023.

[44] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[45] S. Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

[46] S. Van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011.

[47] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.

[48] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[49] T. Zhang. *Mathematical analysis of machine learning algorithms*. Cambridge University Press, 2023.

[50] Z.-H. Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018.