Machine Learning — Statistical Methods for Machine Learning

Introduction

Instructor: *Nicolò Cesa-Bianchi* version of January 13, 2026

Machine learning is the core technology driving the recent revolutionary advances in Artificial Intelligence (AI). While much of this extraordinary progress relies on the paradigm of Generative AI (GenAI), building large-scale systems—such as ChatGPT or Gemini—requires the entire toolbox of ML techniques. Therefore, to comprehend modern AI, we must first understand the tools and principles that underpin its foundations. Generally speaking, ML can be used to solve the following basic tasks.

- **Data clustering:** Grouping data points according to their similarity (e.g., segmenting customers based on consumer profiles).
- **Data prediction:** Mapping input data to output labels/targets (e.g., classifying documents by topic or predicting numerical values).
- **Planning and control:** Performing a sequence of actions in an environment to maximize a notion of utility (e.g., navigating a robot through unknown terrain).
- **Data generation:** Creating new data samples that resemble a training distribution (e.g., synthesizing text or images).

Algorithms that solve learning tasks based on annotated historical data (e.g., documents tagged with topics or images labeled with object names) operate in a **supervised learning** mode. In contrast, algorithms that leverage data without explicit semantic annotations operate in an **unsupervised learning** mode. Finally, algorithms that learn by observing the effects of their actions within an environment operate in a **reinforcement learning** mode. It is worth noting that modern GenAI systems utilize a hybrid approach: unsupervised (or self-supervised) learning for pre-training, supervised learning for instruction tuning, and reinforcement learning for alignment. In this course, we focus on **supervised learning** and study the design of systems whose goal is to learn predictors, i.e., functions that map data points $\boldsymbol{x}$ to their correct labels/targets y. Once learned, these functions can be used to categorize documents, predict the future price of a stock, diagnose a disease based on medical records, and much more.

Label sets. We use $\mathcal{Y}$ to denote the set of all possible labels for a data point of a given learning problem. Note that labels can be of two different types: **categorical** labels, like the topics of a document, and **numerical** labels, like the price of a stock or the demand for a product. Categorical labels define **classification** problems, where labels sets $\mathcal{Y}$ are typically finite and small (such as $\mathcal{Y} = \{\text{sport}, \text{politics}, \text{business}\}$ for document topics). Numerical labels define **regression** problems, where label sets $\mathcal{Y} \subseteq \mathbb{R}$ are typically infinite.

As we can always map symbols to numbers, we need to specify more precisely how regression differs from classification. In regression, prediction mistakes are typically a function of the difference $|y-\widehat{y}|$,

where $\widehat{y}$ is the prediction for y. In classification, mistakes are typically binary: either $\widehat{y} = y$ (no mistake) or $\widehat{y} \neq y$ (mistake). This because the label set in a classification task is an abstract set of symbols (irrespective to whether we encode the symbols using numbers) without a natural notion of distance between them. When $|\mathcal{Y}| = 2$, for example $\mathcal{Y} = \{\text{healthy}, \text{sick}\}$, we have a binary classification problem, and conventionally use the numerical encoding $\mathcal{Y} = \{-1, 1\}$.

Loss functions. In order to measure the goodness of a prediction for a prediction task we use a nonnegative loss function ℓ, measuring the discrepancy $\ell(\widehat{y}, y)$ between the predicted label $\widehat{y}$ and the correct label y. In this course we always assume that $\ell(\widehat{y}, y) = 0$ when $y = \widehat{y}$. The simplest loss function for classification is the zero-one loss:

$$\ell(\widehat{y}, y) = \begin{cases} 0 & \text{if } y = \widehat{y}, \\ 1 & \text{otherwise.} \end{cases}$$

In certain cases, we need more complex classification losses. Consider the problem of categorizing spam email using the label set $\mathcal{Y} = \{\text{spam}, \text{nonspam}\}$. We may penalize a **false positive** mistake (i.e., a nonspam email wrongly classified as spam) more than a **false negative** mistake (i.e., a spam email wrongly classified as nonspam). For example,

$$\ell(\widehat{y}, y) = \begin{cases} 2 & \text{if } y = \text{nonspam and } \widehat{y} = \text{spam}, \\ 1 & \text{if } y = \text{spam and } \widehat{y} = \text{nonspam}, \\ 0 & \text{otherwise.} \end{cases}$$

In regression, typical loss functions are the **absolute loss** $\ell(\widehat{y}, y) = |y - \widehat{y}|$ and the **quadratic loss** $\ell(\widehat{y}, y) = (y - \widehat{y})^2$. Note that these losses are only meaningful for numerical labels.

Probabilstic classification. In some cases, it may be convenient to choose predictions from a set $\mathcal{Z}$ different from the label set $\mathcal{Y}$. For example, consider the problem of assigning a probability $\widehat{p} \in (0, 1)$ to the event $y =$ "it will rain tomorrow" (and, consequently, assigning **probability** $1 - \widehat{p}$ to the complementary event $y =$ "it will not rain tomorrow"). In this case, $\mathcal{Y} = \{\text{"rain"}, \text{"no rain"}\}$ and $\mathcal{Z} = (0, 1)$. Denoting these two events with 1 (for rain) and 0 (for no rain), we may use a loss function for regression, such as the absolute loss $\ell(\widehat{p}, y) = |y - \widehat{y}| \in (0, 1)$.[1] In order to extend the range of the loss function, so to punish more harshly predictions that depart too much from reality, we may use instead the **logarithmic loss**,

$$\ell(\widehat{y}, y) = \begin{cases} \ln \frac{1}{\widehat{y}} & \text{if } y = 1 \text{ (rain)}, \\ \ln \frac{1}{1-\widehat{y}} & \text{if } y = 0 \text{ (no rain)}. \end{cases}$$

Unlike the absolute loss, the logarithmic loss may take arbitrarily high values, see Figure 1. In practice, this fact prevents the predictor from using predictions $\widehat{y}$ that are too certain, namely too close to zero or one.

Data domain. We use $\mathcal{X}$ to denote the set of all possible data points for a given learning problem. Each data point $\boldsymbol{x}$ is typically stored as a database record. In many interesting cases, data can be encoded as vectors of real numbers. Such a vector representation is natural whenever the data consist of a set of homogeneous quantities, such as pixels in an image or word frequencies in a

[1] Note: While we typically use $\{-1, 1\}$, probabilistic frameworks (like logistic regression) often use $\{0, 1\}$. We will switch between these representations as needed.

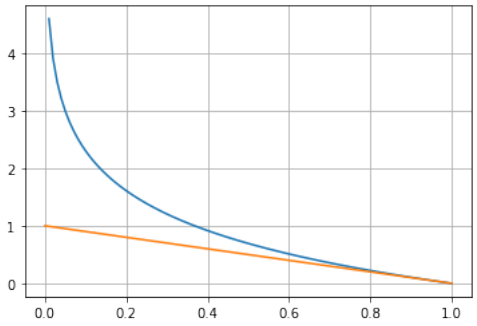

Figure 1: Logarithmic loss function $\ell(\widehat{y}, 1) = \ln \frac{1}{\widehat{y}}$ (blue curve) and absolute loss function $\ell(\widehat{y}, 1) = |1 - \widehat{y}|$ (red line) for the case $y = 1$. In particular, we have $\lim_{\widehat{y} \to 0^+} \ell(\widehat{y}, 1) = \lim_{\widehat{y} \to 1^-} \ell(\widehat{y}, 0) = \infty$.

document. The standard Euclidean geometry then works well because all coordinates use the same unit of measurement. In these cases we may hope that the semantic information carried by the labels be reflected in the geometrical relationships between data points (e.g., in a classification task data points with the same label are close to each other in terms of their Euclidean distance).

In other cases, a vector space representation may not be that natural. For example, the values in the fields of a medical record may use different units, such as age and height, or even be categorical, such as sex. Using rescaling and other techniques (like binary encoding) for dealing with categorical quantities, we can provide a homogenenous vector space representation to most types of data. However, the geometrical properties of these vectors might not be well correlated to their labels.

In this course, we often assume that data can be represented as vectors of numbers, namely $\mathcal{X} \equiv \mathbb{R}^d$. Irrespective of whether $\mathcal{X} \equiv \mathbb{R}^d$ or $\mathcal{X} \equiv \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ for some arbitrary sets $\mathcal{X}_1, \ldots, \mathcal{X}_d$, given a data point $\boldsymbol{x} = (x_1, \ldots, x_d)$, we say that x_i is the value of the i-th **feature** or **attribute**.

A **predictor** is a function $f : \mathcal{X} \to \mathcal{Y}$ mapping data points to labels (or $f : \mathcal{X} \to \mathcal{Z}$ if the predictions belong to a set $\mathcal{Z}$ different from $\mathcal{Y}$). Informally speaking, in a prediction problem the goal is to learn a function f that generates predictions $\widehat{y} = f(\boldsymbol{x})$ such that $\ell(\widehat{y}, y)$ is small for most data points $\boldsymbol{x} \in \mathcal{X}$ observed in practice. In practice, the function f is represented by a certain choice of parameters in a certain model. For examples, a certain setting of the weights of a neural network.

Examples. In supervised learning, an **example** is a pair $(\boldsymbol{x}, y)$ where $\boldsymbol{x}$ is a data point and y is the "true" label associated with $\boldsymbol{x}$. In some cases, there is a unique true label for $\boldsymbol{x}$. This happens when y measures some objective property of the data point; for example, y is the closing price of a stock on a certain day. In some other cases, the label y is subjectively assigned by a human annotator; for example, the genre of a movie. Clearly, different annotators may have different opinions about a movie's genre, implying that the same data point may occur with different "true" labels.

In order to estimate the predictive power of a predictor, which is what we are ultimately interested in, we typically use a **test set**. This is a set[2] of examples $(\boldsymbol{x}'_1, y'_1), \ldots, (\boldsymbol{x}'_n, y'_n)$. Given a loss

[2]Technically, it is a multiset because some examples may occur more than once. However, treating datasets as

function ℓ, the test set is used to compute the **test error** of a predictors f,

$$\frac{1}{n}\sum_{t=1}^{n}\ell\big(f(\boldsymbol{x}'_t), y'_t\big)\ .$$

The test error is meant to estimate the average performance of the predictor on typical real-world data.

A **training set** is a set of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$. A **learning algorithm** receives a training set as input and outputs a predictor. Training and test set are often prepared together, through a single round of data collection and annotation. Partitioning the examples in training and test data is done afterward, typically using a random split. Our objective is to develop a theory to guide us in the design of learning algorithms that generate predictors with low test error. Given a learning algorithm A and a training set S, we use $A(S)$ to denote the predictor output by A on input S.

Since the only input to a learning algorithm is the training set $S \equiv \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\big\}$, a natural approach to the design of learning algorithms is to assume that the **training error**

$$\ell_S(f) = \frac{1}{m}\sum_{t=1}^{m}\ell\big(f(\boldsymbol{x}_t), y'_t\big)$$

of a predictor be correlated to its test error.

Empirical risk minimization. Let $\mathcal{F}$ be a given set of predictors and ℓ a loss function. The empirical risk minimizer (ERM) is the learning algorithm A that outputs some predictor in $\mathcal{F}$ minimizing the training error

$$A(S) \in \operatorname*{argmin}_{f\in\mathcal{F}} \ell_S(f)\ .$$

The $\in$ notation takes into account the fact that there could be multiple $f \in \mathcal{F}$ minimizing the training error.

Overfitting and underfitting. ERM clearly fails when $\mathcal{F}$ is too small so that all predictors in $\mathcal{F}$ have large training error. If the training error is large, then the test error will be also large. To find a good predictor, we should then run ERM with a large $\mathcal{F}$.

On the other hand, choosing $\mathcal{F}$ too large can also lead ERM to fail. To see this, assume $\mathcal{Y} \equiv \{-1, 1\}$ and consider a toy problem with only five data points, $\mathcal{X} \equiv \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_5\}$. Now, take some $\mathcal{F}$ containing a classifier $f : \big\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_5\} \to \{-1, 1\}$ for each of the $2^5 = 32$ possible binary labelings of the five data points, and suppose the training set contains any three points and the test set contains the two remaining ones. Now assume data labels $y_1, \ldots, y_5$ are all assigned using some $f^* \in \mathcal{F}$, $y_t = f^*(\boldsymbol{x}_t)$ for $t = 1, \ldots, 5$. Clearly f^* has zero training error and zero test error. However, with a training set of three points, ERM always finds four classifiers in $\mathcal{F}$ that have zero training error. Of these four classifiers with zero training error, only one (i.e., f^*) has also zero test error. But the training set does not contain enough information to help ERM select this classifier.

multisets complicates the notation. For this reason, and without much loss of generality, we will mostly view datasets as sets in the standard mathematical acception.

The problem in the previous example is that $\mathcal{F}$ is too large with respect to the training set. Information theory tells us that we need $\log_2 |\mathcal{F}| = 5$ bits of information to identify $f^* \in \mathcal{F}$. Indeed, f^* is determined by the five labels $f^*(\boldsymbol{x}_1), \dots, f^*(\boldsymbol{x}_5)$. This is telling us that the training set should contain at least $\log_2 |\mathcal{F}|$ distinct data points. Equivalently, $|\mathcal{F}|$ should be smaller than 2^m, where m is the training set size.

We may give specific names to these two ways of failing (i.e., returning a predictor with high test error) for a generic learning algorithm A:

- if A fails and returns a predictor with high training error, then we say that A is **underfitting**,
- if A fails and returns a predictor with low training error, then we say that A is **overfitting**.

When A is ERM and the training set size m is fixed, our information-theoretic argument says that we should expect overfitting when $\log_2 |\mathcal{F}| \gg m$. Vice versa, when $|\mathcal{F}|$ is too small, we should expect underfitting as soon as the training set satisfies $m \gg \log_2 |\mathcal{F}|$.

Noisy labels. In practice labels are noisy, meaning that they are not deterministically associated with data points $\boldsymbol{x}$ (like in our previous example where $y_t = f^*(\boldsymbol{x}_t)$). Noise may occur for at least three (not mutually exclusive) reasons.

1. **Human uncertainty:** The labels are assigned by a human annotator who decides the "true" label for each data point. In this case, different annotators may have different opinions.
2. **Epistemic uncertainty:** Each data point is represented by a feature vector $\boldsymbol{x}$ that does not contain enough information to uniquely determine the label. For example, suppose $\boldsymbol{x}$ encodes measurements such as today's temperature, pressure, humidity, whereas $y \in \{-1, 1\}$ denotes whether tomorrow rains or not. It is quite possible that the same observed values lead to rain in some cases and to sun in other cases.
3. **Aleatoric uncertainty:** The feature vector $\boldsymbol{x}$ representing a data point is obtained through noisy measurements. The label associated with a given $\boldsymbol{x}$ is then stochastic because the same $\boldsymbol{x}$ could have been generated by different data points.

Noisy labels can cause overfitting because they may mislead the algorithm with regard to what is the "true" label for a given data point.