MA2823: Foundations of Machine Learning
Chapter 2: Supervised Learning

Lecturer: Chloé-Agathe Azencott Scribe: Chloé-Agathe Azencott

In this chapter we will see:

— how to formulate a supervised machine learning problem;

— some basic elements of learning theory;

— how to define and use the notion of model complexity.

1 Supervised classification

The goal of *supervised classification* is to learn a class of objects from examples. Let us define a p-dimensional representation of the objects of interest.

Definitions and notations. Let us consider that we are given n *training examples*, or *training (data) points*, or *observations*, or *training samples*. These n points form the *training set*, and are represented by n p-dimensional vectors $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\}$: $\boldsymbol{x}^i \in \mathbb{R}^p, i \in \{1, \ldots, n\}$. The p dimensions chosen to represent the data are called *features*, *descriptors*, *variables* or *attributes*. Finally, the *training set* also contains the *labels*, *or targets*, or *outcomes* $\{y^1, \ldots, y^n\}$: $y^i \in \{0, 1\}$. For any $i \in \{1, \ldots, n\}$, $y^i = 1$ if and only if $\boldsymbol{x}^i$ represents an object that belongs to $\mathcal{C}$. In that case, $\boldsymbol{x}^i$ is said to be a *positive example*. If it does not belong to $\mathcal{C}$, $y^i = 0$ and $\boldsymbol{x}^i$ is called a *negative example*.

The training set is denoted by $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \ldots, (\boldsymbol{x}^n, y^n)\}$. The goal of *supervised classification* is to use $\mathcal{D}$ to define a function $f : \mathbb{R}^p \to \{0, 1\}$ that returns 1 if and only if it is given as input an object that belongs to $\mathcal{C}$. We will use $c : \mathbb{R}^p \to \{0, 1\}$ to denote the true function that accurately labels data points and that we want to approximate.

Empirical error. On the training set $\mathcal{D}$, the *empirical error* of f is given by

$$E(f|\mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} 1_{f(\boldsymbol{x}^i) \neq y^i}.$$

It counts the number of mistakes made by the classifier.

2 Version space

Our beliefs about the class $\mathcal{C}$ allows us to define the *hypothesis space* $\mathcal{H}$ from which we believe $\mathcal{C}$ to be drawn. We will be looking for a prediction function (or *discriminant*) $f \in \mathcal{H}$.

One of the central questions in machine learning is to define the hypothesis space $\mathcal{H}$. Another one is to formulate ways to choose $f \in \mathcal{H}$. One of our main concerns will be *generalization*: we want f to work well on data other than the training data, i.e. data that we have not observed yet. With this in mind, we can define the *most speficic hypothesis* as the hypothesis $S \in \mathcal{H}$ that is most tight to the positive training examples. We can also define the *most generic hypothesis* as the hypothesis $G \in \mathcal{H}$ that is most tight to the negative training examples.

Ideally, with enough training samples, the most specific and the most generic hypotheses coincide. This can be illustrated with a card classification example. We are going through a deck of cards, and are being told for each card whether it is a "good card" (positive class) or a "bad card". Before we start, the most specific hypothesis is that there is no good card. The most generic hypothesis is that all cards are good cards. After each card we see, we can update the hypotheses, until they converge.

However, it is generally the case that we do not have enough training points to make the most specific and the most generic hypotheses match. In that setting, all hypotheses that are between S and G are said to be *consistent* with the training data $\mathcal{D}$. This means that these hypotheses do not make any mistake on the training set. Their empirical error on $\mathcal{D}$ is 0. The set of consistent hypotheses is called the *version space*. This concept was defined by Tom Mitchell in the early 1980s.

One possible strategy is to choose f half-way between S and G. This strategy maximizes the *margin*, that is to say the distance between the decision boundary and the nearest point from the training set.

Example. Let us look at objects that are cars and are represented by a 2-dimensional representation: x_1 is the price of car $\boldsymbol{x}$ and x_2 is its engine power. Let $\mathcal{C}$ be the class of "family cars".

Our goal is to define a function $f : \mathbb{R}^2 \to \{0, 1\}$ that only returns 0 if (x_1, x_2) represents a family car. This example is illustrated on Fig. 1.

In this case, what we believe about family cars is that they are of medium-range prices and medium-range engine power. Formally, this means we believe $\mathcal{C}$ to be a rectangle aligned with the axes, defined by $p_1 \leq x_1 \leq p_2$ and $e_1 \leq x_2 \leq e_2$. Our hypothesis space is the set of axis-aligned rectangles.

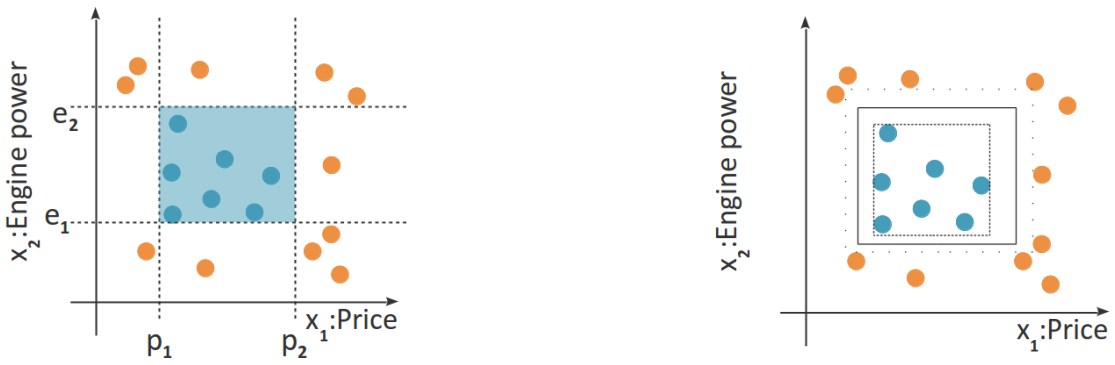


Figure 1: Left: Training data for the class "family cars". Blue points are family cars, orange points are not. We believe the discriminant to be an axis-aligned rectangle and $\mathcal{C}$ to be defined by $p_1 \leq x_1 \leq p_2$ and $e_1 \leq x_2 \leq e_2$. Right: the most specific hypothesis (dashed), the most generic hypothesis (dotted), and the margin-maximizing hypothesis (continuous).

3 Model complexity

3.1 Overfitting

In real-world applications, there are multiple sources of noise in the training data:

— *measurement noise*, i.e. imprecision in recording the features that describe the data;

— *teacher noise*, i.e. errors in data labeling;

— missing features, i.e. hidden (or *latent*) factors that would be necessary to be able to really make the classification.

This means that it might not be possible to make no error on the training set.

In addition, aiming at all cost to fit the training data might result in *overfitting*: the model is too complex and is not likely to generalize well. This happens when $\mathcal{H}$ is more complex than $\mathcal{C}$.

On the contrary, *underfitting* takes place when the hypothesis space is too simple, and no function from $\mathcal{H}$ can fit the training data well enough to have a good chance to generalize.

Both behaviors are illustrated on Fig. 2.

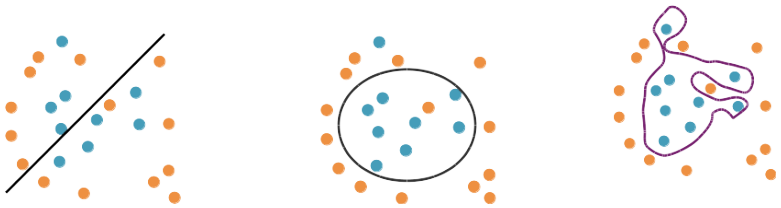


Figure 2: Models of increasing complexity. The simple straight line on the left might underfit. The complex line on the right is likely to overfit.

We tend to favor using simpler models when possible (i.e. when they are not likely to underfit):

— they are easier to use (lower computational complexity);

— they are easier to train (lower space complexity);

— they are easier to interpret;

— they tend to generalize better;

— Occam's razor suggests that simple explanations are more plausible.

3.2 Bias-Variance tradeoff

In statistics, *bias* is the difference between the expected value of an estimator and the true value being estimated:

$$\text{Bias}(\hat{y}) = \mathbb{E}\left(\hat{y} - y\right).$$

A simpler model has a higher bias (it makes mistakes on the training data): high bias can cause underfitting.

The *variance* of an estimator is the deviation of the estimates from the expected values:

$$\text{Variance}(\hat{y}) = \mathbb{E}\left(\hat{y} - \mathbb{E}(\hat{y})\right).$$

A more complex model has higher variance. High variance can cause overfitting.

In other words, increasing model complexity towards overfitting decreases the bias and increases the variance. Conversely, decreasing model complexity towards underfitting increases the bias and decreases the variance. This behavior is illustrated on Fig. 3. We must be wary of wanting to minimize bias at all costs, as it might result in a worse model.

The mean squared error of the prediction, $\text{MSE} = \mathbb{E}\left(\hat{y} - y\right)^2)$, can be rewritten as $\text{MSE} = \text{Var}(\hat{y}) + \text{Bias}^2(\hat{y})$. This is refered to as the *bias-variance decomposition*, and illustrate that there is a tradeoff between bias and variance.

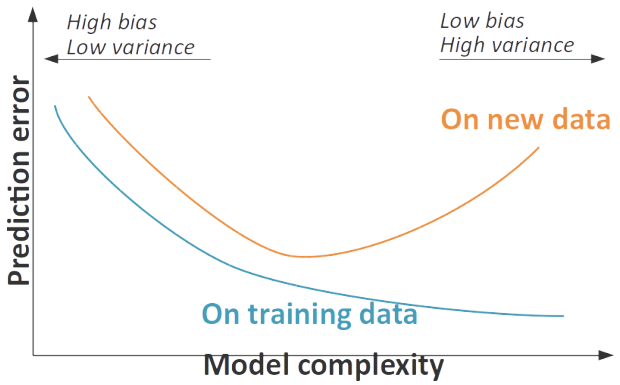


Figure 3: Bias-variance trade-off.

4 Complexity of the hypothesis space: Vapnik-Chervonenkis dimension

Let us consider N points. There are 2^N ways in which these points can be labeled positive and negative. The hypothesis space $\mathcal{H}$ is said to *shatter* N if, for any of these 2^N labelings, there exists a hypothesis $h \in \mathcal{H}$ that is consistent. The *Vapnik-Chervonenkis dimension* (or *VC-dimension*) of $\mathcal{H}$ is the maximum number of points that can be shattered by N. The VC-dimension was defined by Vapnik and Chervonenkis in 1970.

Example: VC-dimension of a line (in $\mathbb{R}^2$). There exist configurations of 3 points such that, whatever their labels, one can find a line that separates the positive point(s) from the negative point(s). However, there is no configuration of 4 points such that, whatever their labels, one can find a line that separates the positive point(s) from the negative point(s). The VC dimension of a line is 3.

Example: VC-dimension of an axis-aligned rectangle (in $\mathbb{R}^2$). There exist configurations of 4 points (for example on the vertices of a parallelogram that is not aligned with the angles) such that, whatever their labels, one can find a rectangle that separates the positive point(s) from the negative point(s). However, there is no configuration of 5 points such that, whatever their labels, one can find a rectangle that separates the positive point(s) from the negative point(s). Hence the VC dimension of a rectangle is 4.

This suggests that, using an axis-aligned rectangle, we can only guarantee learning datasets with 4 data points: if there are more, the rectangle might not be consistent with the data.

However, the VC-dimension is independent of the probability distribution of the data we are interested in. The world changes smoothly, and, especially for well-designed representations, nearby instances tend to have the same label. Hence we can still learn specific classes with an hypothesis space of low VC-dimension.

5 Probably Approximately Correct learning

PAC-learning is a theoretical framework for the mathematical analysis of machine learning that was proposed by Leslie Vaillant in 1984.

PAC-learnability rests on the two following concepts:

— f must be *approximately correct*, that is to say, the probability that f makes a mistake must be bounded by $\epsilon > 0$: $P_{\mathcal{D}}(f(x) \neq c(x)) \leq \epsilon$.

— f must be *probably correct*, that is to say, f is correct most of the time, i.e. with probability at least $1-\delta$, where $\delta \leq \frac{1}{2}$.

A probably approximately correct hypothesis f verifies

$$P_{\mathcal{D}}\left(P_{\mathcal{D}}(f(\boldsymbol{x}) \neq c(\boldsymbol{x})) \leq \epsilon\right) \geq (1-\delta).$$

$P_{\mathcal{D}}$ refers to the probability over which training examples you happen to get when building $\mathcal{D}$.

A hypothesis class $\mathcal{H}$ is said to be *PAC-learnable* if, for any class $\mathcal{H} \subset \mathcal{C}$, for any training dataset $\mathcal{D}$, there exists an algorithm that produces a probably approximately correct hypothesis in polynomial time in $1/\epsilon$ and $1/\delta$.

In that case, its *sample complexity* is the number n of training examples that it needs in order to learn a probably approximately correct hypothesis.

This means that, given ϵ, δ, and a class $\mathcal{C}$ from which the training data $\mathcal{D}$ is drawn, we want to find n such that

Example: Learning axis-aligned rectangles. Let us consider $f = S$ (the most specific hypothesis, that is to say, the rectangle constructed tightly around the training examples, represented by a dashed line on Figure 1). The true solution is given by $c \in \mathcal{H}$.

Theorem. *For $\epsilon > 0$, $\delta \leq \frac{1}{2}$, choosing the most specific hypothesis is probably approximately correct if the number of training samples n is bounded by*

$$n \geq \frac{4}{\epsilon} \log\left(\frac{4}{\delta}\right).$$

Proof. Because f is tight, f is included in $\mathcal{C}$. This means we can consider the 4 strips (rectangles) between f and $\mathcal{C}$ (see Figure 4a). If we can guarantee that the probability of a positive point to fall in one of these strips is lower than $\frac{\epsilon}{4}$, then the probability of making an error is lower than ϵ:

$$P_{\mathcal{D}}(\boldsymbol{x} \in c \triangle f) \leq P_{\mathcal{D}}(\boldsymbol{x} \in \text{top strip}) + P_{\mathcal{D}}(\boldsymbol{x} \in \text{left strip}) + P_{\mathcal{D}}(\boldsymbol{x} \in \text{right strip}) + P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}).$$

We will hence try to find n such that $P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}) \leq \frac{\epsilon}{4}$. These 4 strips overlap slightly, so the bound is not tight.

Let us now define T as the lower strip of c such that $P_{\mathcal{D}}(\boldsymbol{x} \in T) = \frac{\epsilon}{4}$. There are two possible scenarios:

— Either T is larger than our bottom strip (see Figure 4b). In this case, $P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}) \leq \frac{\epsilon}{4}$, which is what we want.

— Or T is narrower than our bottom strip (see Figure 4c). In this case $P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}) > \frac{\epsilon}{4}$. This is the scenario we want to avoid. This scenario happens when no training example falls in T. Otherwise, because f is tight, T must overlap with f (see Figure 4d).

The probability for the bad scenario to happen is the probability that all n training examples fall outside of T, i.e. $\left(1-\frac{\epsilon}{4}\right)^n$. Here we have used the fact that the probability of falling inside T is exactly $\frac{\epsilon}{4}$ (by definition), and that the n examples are drawn independently from each other, which means that their probability of falling outside of T are independent from each other.

The same analysis holds for all three other strips. Hence the overall probability of the scenario we want to avoid (making a mistake on the training data) is upper bounded by $4\left(1-\frac{\epsilon}{4}\right)^n$.

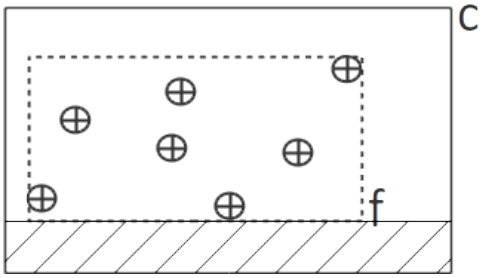
In order to lower bound the probability of being correct by $1-\delta$, we want to upper bound this probability by δ:

$$4\left(1-\frac{\epsilon}{4}\right)^n \leq \delta.$$

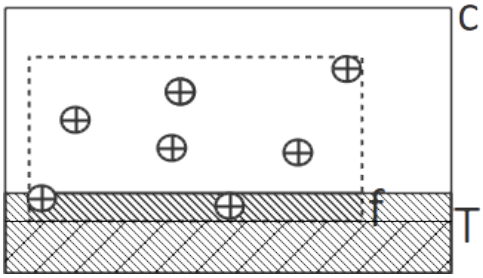
Using the inequality $(1-u) \leq e^{-u}$, we obtain that $4\left(1-\frac{\epsilon}{4}\right)^n \leq \left(4e^{-\frac{\epsilon}{4}}\right)^n$. This means that if n is such that $4e^{-\frac{\epsilon}{4n}} \leq n$, then we can guarantee being probably approximately correct. This is equivalent to

$$n \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right).$$

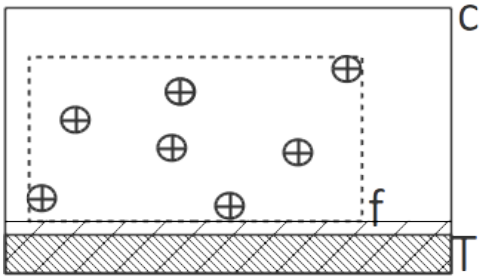
□



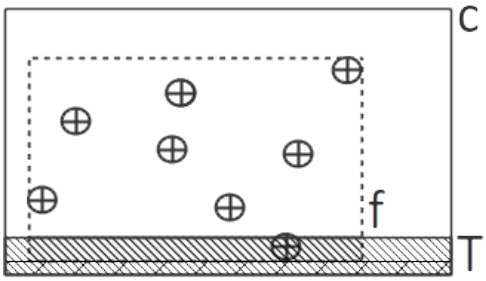
(a) We want to guarantee that $P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}) \leq \frac{\epsilon}{4}$.



(b) Favorable scenario: T is larger than our bottom strip.



(c) Scenario to avoid: T is narrower than our bottom strip and we cannot guarantee that $P_{\mathcal{D}}(\boldsymbol{x} \in \text{bottom strip}) \leq \frac{\epsilon}{4}$.



(d) If a positive example was to fall in T, f, being the most specific hypothesis, would stretch to include it, and we would be in the favorable scenario.

Figure 4: Learning an axis-aligned rectangle c. f is the most specific hypothesis. Only the positive training examples are represented; the negative training examples all fall outside of f.

6 Beyond binary classification

6.1 Multi-class classification

Multi-class classification is the setting in which we want to learn $K > 1$ classes $\mathcal{C}_1, \ldots, \mathcal{C}_K$. Ffor example, instead of just learning "family cars", we now want to learn "family cars" but also "luxury sedans" and "sports cars".

The labels $\{\boldsymbol{y}^i, \ldots, \boldsymbol{y}^K\}$ are now k-dimensional: $y_k^i = 1$ if and only if $\boldsymbol{x}^i \in \mathcal{C}_k$. Multi-class classification can be formulated as learning k hypotheses $f_1, \ldots, f_k$. This is often refered to as *one-vs-all classification*.

6.2 Regression

In regression, the labels are real-valued: $y^i \in \mathbb{R}$.

The empirical error is given as:

$$E(f|\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}^i) \neq y^i)^2.$$