1 Online Classification

So far we have been considering the batch learning setting, in which the learning algorithm is presented with a sample of training data and must produce a hypothesis that performs well on new data generated from the same distribution. For the next few lectures we shift our attention to the online learning setting. Here the learning algorithm is presented with a sequence of examples over time, and must repeatedly update its hypothesis based on these examples. The online learning setting can be used to model applications, such as spam filtering, in which the algorithm must adapt to feedback.

In the basic online setting, at each round \( t = 1, 2, \ldots \),

1. The learner is presented with a new example \( x_t \).
2. The learner must predict a label \( \hat{y}_t \) for this example.
3. After the prediction is made, the true label \( y_t \) of the example is revealed.
4. The learner updates its prediction rule based on \( x_t, \hat{y}_t, \) and \( y_t \).

Unlike the PAC learning model, no distributional assumptions are made about the sequence of examples \( x_1, x_2, \ldots \). The online learning setting is therefore “adversarial” in that we can imagine the examples and labels are generated by an adversary who would like to force our algorithm to make as many mistakes as possible. Because of this, there are a lot of connections between online learning and game theory, some of which we will discuss in upcoming lectures.

The learning algorithm is said to make a mistake in round \( t \) if \( y_t \neq \hat{y}_t \). Since the adversary can force a mistake in every round we analyse the performance of online learning algorithms in relative terms. We define the regret of the algorithm after receiving a finite sequence of inputs \( x_1, \ldots, x_T \) rounds is

\[
R_T = \text{number of mistakes} - \text{number of mistakes of the best comparator}
\]

for some suitable class of comparators. Notice that regret is a retrospective notion, unlike generalisation error.

To start with we work within the realisable setting: we assume that there is a class of hypotheses \( H \) such that the sequence of inputs and labels \( (x_1, y_1), \ldots, (x_T, y_T) \) is consistent with some \( h \in H \). (In other words, the best comparator is a function from \( H \) that makes no mistakes.) In this setting our goal is to prove an upper bound on the total number of mistakes made by the algorithm.

---

2 Mistake Bounds

We begin by deriving a simple mistake bound in the realisable setting in the case of a finite hypothesis class $H$ using the Halving Algorithm. This algorithm is as follows: in each round $t$ there is a version space $VS_t \subseteq H$ of active hypotheses. Initially, we have $VS_0 = H$. Then, in each round $t = 1, 2, 3, \ldots$,

1. Given $x_t, \hat{y}_t$ is defined to be the label chosen for $x_t$ by the majority of the functions in the current version space $VS_t$.

2. Then $y_t$ is revealed, and $VS_{t+1}$ is assigned the collection of all hypotheses in $VS_t$ that correctly predicted $y_t$.

Notice that every time the Halving Algorithm makes a mistake, the number of hypotheses in the version space drops by at least one half. Thus we have:

**Proposition 1.** Let $H$ be any finite hypothesis class. Under the realisability assumption the number of mistakes made by the Halving Algorithm is no more than $\log_2 |H|$.

This result has the familiar logarithmic dependence on the size of the concept class, reflecting the Occam’s razor principle in that the bigger the class we start with, the harder it is to learn. As an example consider the class of Boolean conjunctions on $n$ variables. In this case $|H| = 3^n$ and so the number of mistakes is bounded by $n \log_2 3$ by Proposition 1. However, the Halving Algorithm is computationally infeasible in this case as it requires tracking a version space of $3^n$ functions and computing a majority label by evaluating up to $3^n$ functions at every step.

It is possible to derive simple lower bounds in this setting too.

**Proposition 2.** Given a class of hypotheses $H$ with VC dimension $d$, for any deterministic learning online learning algorithm there is a sequence of inputs for which the algorithm makes at least $d$ mistakes.

**Proof:** By the definition of VC dimension, there exists some set of $d$ points that is shattered by $H$. The adversary can choose these $d$ points as the first $d$ examples, $x_1, \ldots, x_d$. Since there exist functions in $H$ achieving every labelling of these points, the adversary can ensure that $A$ makes a mistake on the first $d$ rounds, with the labels still being consistent with some hypothesis in $H$.

It is straight-forward to adapt this lower bound for randomized algorithms too and show that the adversary can cause at least $d/2$ mistakes on expectation by choosing the labels of the $d$ shattered points uniformly at random. Notice that the adversarial assumption makes it significantly easier to prove mistake bounds in the online setting compared to the PAC model, as considered in the previous lecture.

3 The Perceptron Algorithm

In this section we introduce Perceptron algorithm, an online algorithm for learning linear threshold functions.

In general a linear threshold function $f : \mathbb{R}^n \rightarrow \{-1, +1\}$ has the form

$$f(x) = \begin{cases} 
+1 & \text{if } w \cdot x \geq b \\
-1 & \text{otherwise}
\end{cases}$$
where $w$ is the normal vector of the separating hyperplane and $b$ is a constant. Without loss of generality we restrict to the case that $b = 0$, i.e., the separating hyperplane passes through the origin. (We can reduce to this case by increasing the dimension by one, mapping $x$ to $\begin{pmatrix} x \\ 1 \end{pmatrix}$ and $w$ to $\begin{pmatrix} w \\ -b \end{pmatrix}$.

The Perceptron algorithm is as follows.

**PERCEPTRON**

$w_1 \leftarrow 0$

for $t \leftarrow 1$ to $T$ do

RECEIVE($x_t$)
  if $w_t \cdot x_t \geq 0$ then $\hat{y}_t \leftarrow +1$ else $\hat{y}_t \leftarrow -1$

RECEIVE($y_t$)
  if $y_t \neq \hat{y}_t$ then $w_{t+1} \leftarrow w_t + y_t x_t$ else $w_{t+1} \leftarrow w_t$

return $w_{T+1}$

The intuition behind the algorithm is that every time it makes a mistake on a positive example, it shifts the weight vector toward the input point, whereas every time it makes a mistake on a negative example, it shifts the weight vector away from that point.

We give a mistake bound for the Perceptron algorithm in terms of the notion of margin. Consider a linear threshold function $f : \mathbb{R}^n \rightarrow \{-1, +1\}$ determined by a hyperplane passing through the origin with unit normal $u$. The margin of $f$ at a labelled point $(x, y)$ is defined to be $y(x \cdot u)$. If the margin is positive then $f$ classifies $(x, y)$ correctly and if the margin is negative then $f$ classifies $(x, y)$ incorrectly. Furthermore, the magnitude of the margin gives the distance of $x$ to the decision boundary. The margin of $f$ on a finite set of labelled points is the minimum of its margin at each point. Intuitively the margin represents how robustly $f$ classifies the data. Ideally the margin should be large and positive.

**Theorem 1.** Suppose that $(x_1, y_1), (x_2, y_2), \ldots, (x_T, y_T)$ are such that $||x_t|| \leq D$ some diameter $D > 0$. Suppose also that for some unit vector $u$ and $\gamma > 0$ the margin bound $y_t(x_t \cdot u) \geq \gamma$ holds for $t = 1, \ldots, T$. Then the Perceptron algorithm makes at most $(D/\gamma)^2$ mistakes.

**Proof:** Let $m_t$ denote the number of mistakes before round $t$ (so that $m_1 = 0$). We break the proof of Theorem 1 into two lemmas. The first lemma shows that the projection of $w_t$ on $u$ gets longer with every mistake.

**Lemma 1.** $w_t \cdot u \geq m_t \gamma$ for all $t$.

**Proof:** The proof is by induction on $t$. The case $t = 1$ follows from the fact that $w_1 = 0$ and $m_1 = 0$.

The induction step is as follows. Suppose there is a mistake in round $t$. Then

$$w_{t+1} \cdot u = (w_t + y_t x_t) \cdot u = w_t \cdot u + y_t (x_t \cdot u) \geq m_t \gamma + \gamma \quad \text{(by induction hypothesis and margin assumption)} = m_{t+1} \gamma.$$ 

The induction step in the case that there is no mistake in round $t$ is trivial.

The next lemma bounds the growth of $||w_t||^2$ in terms of the number of mistakes.
Lemma 2. $||w_t||^2 \leq m_tD^2$.

Proof: The proof is by induction on $t$. The only interesting case is the induction step under the assumption that there is a mistake in round $t$. In this case we have:

$$||w_{t+1}||^2 = ||w_t + y_t x_t||^2 = ||w_t||^2 + 2y_t(w_t \cdot x_t) + ||x_t||^2 \leq ||w_t||^2 + ||x_t||^2 \leq m_tD^2 + D^2 = m_{t+1}D^2.$$

We complete the proof of Theorem 1 as follows. From

$$D\sqrt{m_t} \geq ||w_t|| \quad \text{(Lemma 2)}$$

$$= ||w_t|| ||u||$$

$$\geq w_t \cdot u \quad \text{(Cauchy-Schwarz inequality)}$$

$$\geq m_t \gamma \quad \text{(Lemma 1)},$$

it follows that $m_t \leq (D/\gamma)^2$. 

Let’s consider an application of the above mistake bound. Suppose there are $n$ financial analysts who predict every day whether the market will go up or down. We represent the prediction at time step $t$ as a vector $x_t \in \{-1, +1\}^n$ and the actual outcome of the market as $y_t \in \{-1, +1\}$. We would like to use the Perceptron algorithm to figure out which analysts to follow. (The normal vector of a linear threshold function learned by the Perceptron algorithm can be seen as assigning a weight to each analyst.)

Suppose that there is a subset of $k$ “experts” within the group of $n$ analysts, such that a majority vote of the $k$ experts always gives the right prediction about the movement of the market. Define a unit vector

$$u = \frac{1}{\sqrt{k}}(0, \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0),$$

where the 0-1 vector on the right is the characteristic vector of the $k$ experts.

Then $u$ defines a linear separator with margin at least $\frac{1}{\sqrt{k}}$ since $y_t(x_t \cdot u) \geq \frac{1}{\sqrt{k}}$ for all $t$. Also $||x_t|| \leq \sqrt{n}$ for all $t$. Thus Theorem 1 gives a mistake bound of $nk$ in this case.