CSCI699: Machine Learning Theory Fall 2023

Lecture 9: Kernels, Online Learning

Instructor: Vatsal Sharan

These lecture notes are based on scribe notes by Sai Anuroop Kesanapalli, Di Zhang, Sophie Hsu and Grace Zhang.

1 Kernels (continued)

Last time, we defined and motivated kernels.

Definition 1 (Kernel)**.** *A function $k : \mathbb{R}^d \to \mathbb{R}$ is called a kernel function if and only if there exists $\phi : \mathbb{R}^d \to \mathbb{R}^m$, such that for any $x, x' \in \mathbb{R}^d$,*

$$k(x, x') = \phi(x)^T \phi(x')$$

.

We stated the following properties, which can be useful for showing that a function is a kernel function.

Theorem 2. *Properties of kernels:*

1. *For any $f : \mathbb{R}^d \to \mathbb{R}$, $k(x, x') = f(x)f(x')$ is a kernel*
2. *If $k_1(\cdot, \cdot)$ and $k_2(\cdot, \cdot)$ are kernels then the following are also kernels,*
   a) $\alpha k_1(\cdot, \cdot) + \beta k_2(\cdot, \cdot)$ if $\alpha, \beta \geq 0$
   b) $k_1(\cdot, \cdot) \cdot k_2(\cdot, \cdot)$

Let us see two kernels which are quite useful and popular.

1. Polynomial kernel: For any positive integer p, the polynomial kernel can be defined as

   $$k(x, x') = (x^T x' + 1)^p$$

   To verify that this is a kernel, try to show that there is a feature mapping $\phi(x)$ which induces this kernel. For example, last time we saw how we can get the square of the inner product:

   $$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \to \phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ z_2^2 \end{pmatrix}$$

2. Gaussian kernel: For any $\sigma > 0$, the Gaussian kernel is defined as,

   $$k(x, x') = \exp\Big(- \frac{||x - x'||_2^2}{2\sigma^2} \Big).$$

To verify that this is a kernel, we first expand the ℓ_2 norm squared term,

$$\exp\Big(-\frac{||x-x'||_2^2}{2\sigma^2}\Big) = \underbrace{\exp\Big(-\frac{||x||_2^2}{2\sigma^2}\Big)\exp\Big(-\frac{||x'||_2^2}{2\sigma^2}\Big)}_{k_1(x,x')}\underbrace{\exp\Big(\frac{x^Tx'}{\sigma^2}\Big)}_{k_2(x,x')}.$$

Note that $k_1(x,x')$ is a kernel, since it is of the form $k_1(x,x') = f(x)f(x')$. Hence it is sufficient to argue that $k_2(x,x')$ is a kernel, since the product of two kernels is a kernel. To verify that $k_2(x,x')$ is a kernel, we can do the Taylor series expansion.

$$\exp\Big(\frac{x^Tx'}{\sigma^2}\Big) = 1 + x^Tx' + \frac{(x^Tx')^2}{2!} + \frac{(x^Tx')^3}{3!} + \cdots.$$

Each term in this expansion is a power of the inner product and can be verified to be a kernel. Since the sum of kernels is a kernel, this verifies that $k_2(x,x')$ is a kernel.

Visually, the Gaussian kernel looks just like a Gaussian. σ corresponds to the standard deviation or width of the Gaussian.

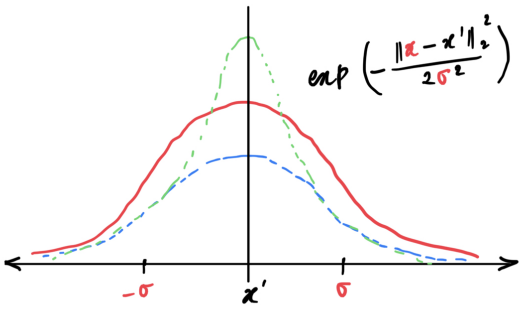


Figure 1: Gaussian kernel, where we have fixed x' and vary x.

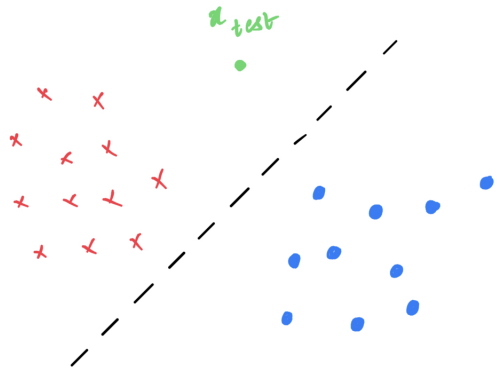
1.1 Parametric vs non-parametric models

To conclude our discussion of kernels, we discuss a characterization of ML models which you may come across commonly in the literature. The differences here are not meant to be rigorous.

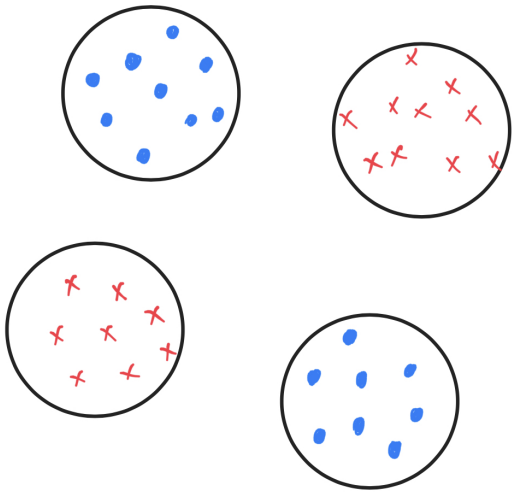
1. Parametric model: The model has a fixed number of parameters which we learn based on some training set.
2. Non-parametric model: The model does not have a fixed set of parameters which are learned from a training set. The complexity of the learned model itself depends on the data.

Example 3 (Kernel nearest-neighbors)**.** *Suppose we are given a training set $S = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}$ and $y_i \in \{\pm 1\}$. Consider a model for which the prediction at any point x_{test} is given by $x_{test} = \text{sign}\left(\sum_{i=1}^n \frac{y_i k(x_{test}, x_i)}{\sum_i k(x_{test}, x_i)}\right)$, where $k(\cdot,\cdot)$ is the Gaussian kernel. This is a simple kernelized version of the nearest neighbor algorithm.*

Suppose the dataset is of the following form.



Then the decision boundary of the model will be close to linear. However, if the dataset is more complicated, such as a mixture of the following four Gaussians.



Then the decision boundary will be more complicated. In particular, we would expect the datapoints within each cluster (circle in the figure) to be classified correctly for some appropriate size of σ in the Gaussian kernel. Therefore, the decision boundary will not be linear in this case. Even though the model itself has not changed, the complexity of the decision boundary has changed because the data has changed. Therefore this model is a non-parametric model.

2 Online Learning

In our usual PAC learning/statistical learning setup we ask the learn to do well under probabilistic assumptions on data (i.e. train/test data are drawn from the same distribution). We developed a theory of generalization to understand how much an algorithm's test accuracy can differ from its training accuracy.

In *online learning*, we make no probabilistic assumptions on the data. The goal is to predict well on datapoints as we see them.

Example: Weather forecasting

- We're interested in predicting rain/no rain.
- Every night, make prediction about next day, based on current conditions.
- Next day, we see whether or not it rained.

Note that there's no train/test split. Every example is both a training example and a test example. Formally, at every time step t,

- Learner receives an input $x_t \in \mathcal{X}$.
- Makes prediction $p_t \in \mathcal{Y}$.
- Sees true label $y_t \in \mathcal{Y}$. Suffers loss $\ell(p_t, y_t)$.

For most of our discussion, we will take $\mathcal{Y} = \{0, 1\}$ and $\ell(p_t, y_t) = \mathbf{1}\{p_t \neq y_t\}$.

2.1 Realizability

As we did in PAC learning/statistical learning, we begin with the realizability assumption on the sequence, which says that there is some hypothesis in the hypothesis classes which correctly labels all datapoints.

Definition 4 (Mistake bound model)**.** *Let $\mathcal{H}$ be a hypothesis class and A be a online learning algorithm. Given any sequence $S = (x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))$ of T labeled datapoints where $h^* \in \mathcal{H}$, let $M_A(S)$ be the number of mistakes A makes on the sequence S. Let $M_A(\mathcal{H})$ be the supremum of $M_A(S)$ over all possible S.*

*If there exists an algorithm A that satisfies a **mistake bound** of the form $M_A(\mathcal{H}) \leq B < \infty$, we say $\mathcal{H}$ is **online learnable** in the mistake bound model.*

Note: B should be independent of length of sequence T. As $T \to \infty$, average number of mistakes $(\leq B/T) \to 0$.

In PAC learning, we saw that the ERM algorithm which chooses any consistent hypothesis over the training set does well, as long as the size of the training set is large enough that the generalization error is small. We start by defining an analogous algorithm for the online learning setup.

Algorithm 1 Consistent

Initialize $V_1 = \mathcal{H}$
for for $t = 1, \ldots, T$ **do**
  receive x_t
  choose any $h \in V_t$
  predict $p_t = h(x_t)$
  receive $y_t = h^*(x_t)$, loss $\mathbb{1}(p_t \neq y_t)$
  update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
end for

The consistent algorithm will remove at least one hypothesis from V_t when there is a mistake prediction. Therefore, after M mistakes, $|V_t| \leq |\mathcal{H}| - M$. Since we have the realizability assumption, V_t should always be nonempty. Therefore we have the following guarantee on the algorithm,

Proposition 5. *Let $\mathcal{H}$ be a finite hypothesis class. The above algorithm gets a mistake bound*

$$M_{consistent}(\mathcal{H}) \leq |\mathcal{H}| - 1.$$

Can we do better? Yes, by quite a lot. If we refine the above algorithm to make its predictions p_t in a smarter way than choosing any consistent hypothesis, then we can improve exponentially on the above mistake bound.

Algorithm 2 Halving

Initialize $V_1 = \mathcal{H}$
for for $t = 1, \ldots, T$ **do**
  receive x_t
  predict $p_t = \text{argmax}_{r \in \{0,1\}} |\{h \in V_t : h(x_t) = r\}|$ (if tie, $p_t = 1$)
  receive $y_t = h^*(x_t)$, loss $\mathbb{1}(p_t \neq y_t)$
  update $V_{t+1} = \{h \in v_t : h(x_t) = y_t\}$.
end for

Proposition 6. *Let $\mathcal{H}$ be a finite hypothesis class. Then halving algorithm satisfies the mistake bound $M_{Halving}(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$.*

Proof. Whenever the algorithm errors, $|V_{t+1}| \leq \dfrac{|V_t|}{2}$. If M is the number of mistakes

$$|V_{T+1}| \leq |\mathcal{H}| 2^{-M}$$

as $|V_{T+1}| \geq 1 \implies M \leq \log_2(|\mathcal{H}|)$. ■

3 Littlestone Dimension

Next, we aim to characterize the complexity online learning. Nick Littlestone suggested a dimension of hypothesis classes that characterizes the best achievable mistake bound, in the same way that VC dimension characterizes the sample complexity of PAC learning.

Littlestone dimension

The key idea behind the notion of Littlestone dimension is to view online learning as a 2-player game between learner and the environment.

At time t :

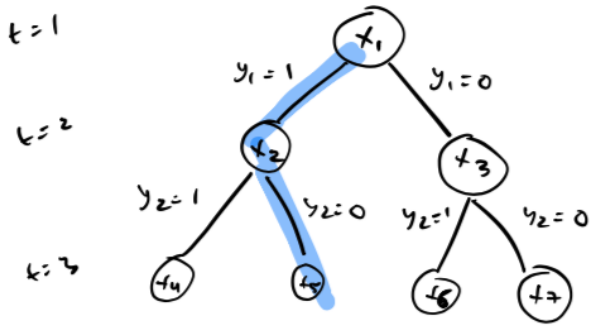
$\rightarrow$ environment picks x_t

$\rightarrow$ learner picks p_t

$\rightarrow$ environment picks $y_t = 1 - p_t$

Q) How to choose x_t to get the learner to make maximum number of mistakes, while ensuring realizability?

Strategy for the environment can be formally described as a binary tree. Each node of the tree is associated with an instance from $\mathcal{X}$. If the learner predicts $p_t = 1$ the environment will declare that this is a wrong prediction (i.e., $y_t = 0$) and will traverse to the right child of the current node. If the learner predicts pt $= 0$ then the environment will set $y_t = 1$ and will traverse to the left child. This process will continue and at each round.



Definition 7 ($\mathcal{H}$ shattered tree)**.** *A shattered tree of depth d is a sequence of instances $x_1, x_2, \ldots, x_{2^d-1} \in \mathcal{X}$ such that for every path from the root to a leaf, $\exists\ h \in \mathcal{H}$ which realizes all the labels along this path.*

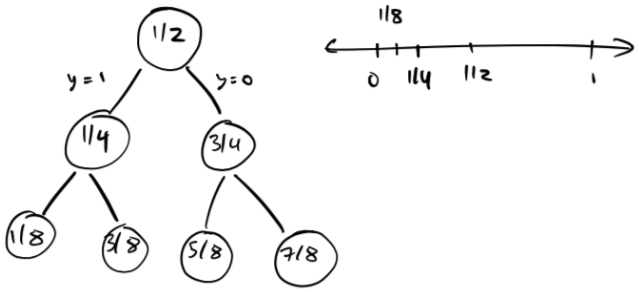
Definition 8 (Littestone dimension)**.** *The Littlestone dimension of a hypothesis class $\mathcal{H}$, denoted by $Ldim(\mathcal{H})$, is the maximal T such that $\exists$ a tree of depth T shattered by $\mathcal{H}$.*

The following result is immediate from the definition of Littlestone dimension.

Proposition 9. *For any online learning algorithm A, $M_A(\mathcal{H}) \geq Ldim(\mathcal{H})$.*

Examples

1. Let $\mathcal{H}$ be a finite hypothesis class. Then $Ldim(\mathcal{H}) \leq \log_2(\mathcal{H})$.
   $\rightarrow$ Any tree which is shattered by $\mathcal{H}$ must have $|\mathcal{H}| \geq \#$ leaves in the tree.
2. $x = [0, 1]$, $\mathcal{H} = \{x \mapsto \mathbb{1}(x < a), a \in [0, 1]\}$ (thresholds on $[0, 1]$). Then, $Ldim(\mathcal{H}) = \infty$



As a direct corollary of the above bound, we have that thresholds are not online learnable.

Corollary 10. *Cannot learn thresholds in the online learning model.*

This follows because algorithm can have a mistake bound smaller than $Ldim(H)$. In fact, it turns out that there is an intimate relation between thresholds and Littlestone dimension. Based on some deep results in model theory, it can be shown that a class has finite Littlestone dimension if and only if it does not embed threshold as a sub-class (see [2] for more details).

Next we see that there is in fact an algorithm which matches the Littlestone dimension bound.

Lemma 11. *There exists an algorithm A with $M_A(\mathcal{H}) \leq Ldim(\mathcal{H})$.*

Instead of predicting according to the larger class as done in Halving, we present an algorithm that predicts according to the class with larger Littlestone dimension.

Algorithm 3 Standard optimal algorithm (SOA)

Initialize $V_1 = M$
for for $t = 1, \ldots, T$ **do**
  receive x_t
  for $r \in \{0, 1\}$, let $V_t^{(r)} = \{h \in V_t : h(x_t) = r\}$
  predict $p_t = \text{argmax}_{r \in \{0,1\}} Ldim(v_t^{(r)})$
  receive $y_t = h^*(x_t)$, loss $\mathbb{1}(p_t \neq y_t)$
  update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$.
end for

Claim 12. *Whenever algorithm makes a mistake $Ldim(V_{t+1}) = Ldim(V_t) - 1$*

Proof. We prove by contradiction. For the sake of contradiction, assume $Ldim(V_{t+1}) = Ldim(V_t)$ though the algorithm has made a mistake. Without loss of generality, assume that the algorithm predicts 0. Then $y_t = 1$, since the algorithm made a mistake. Therefore,

$$Ldim(V_{t+1}) = L_{dim}(V_t^{(1)}) = Ldim(V_t)$$

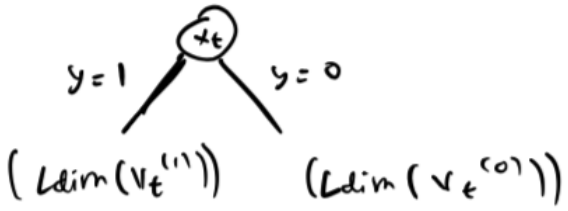
By the definition of the algorithm,

$$Ldim(V_t^{(0)}) \geq Ldim(V_t^{(1)}) = Ldim(V_t)$$

But since $V_t^{(0)} \subseteq V_t$, we also have that

$$\begin{aligned} & Ldim(V_t^{(0)}) \leq Ldim(V_t) \\ \implies & Ldim(V_t^{(0)}) = Ldim(V_t). \end{aligned}$$

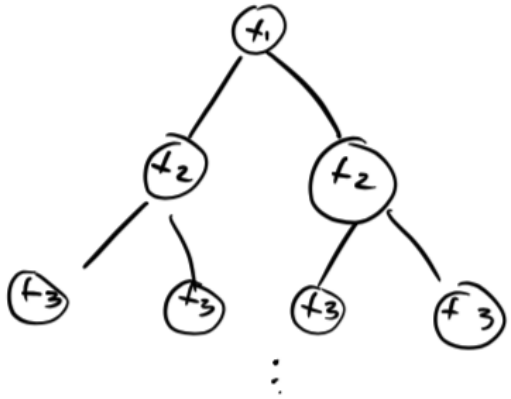
However, this implies that $Ldim(V_t) = Ldim(V_t^{(0)})+1$ by constructing the following tree. Therefore, we have a contradiction.



■

Theorem 13 (Comparison to VCdim($\mathcal{H}$))**.** *For any class $\mathcal{H}$, $VCdim(\mathcal{H}) \leq Ldim(\mathcal{H})$. Further, the gap can be arbitrarily large.*

Proof. Suppose VCdim($\mathcal{H}$) $= d$. Let $x_1, \ldots, x_d$ be a shattered set. Then the following tree can be shattered.



The gap can be arbitrarily large just because of thresholds. ■

Connection to differential privacy. We conclude this section with a somewhat surprising connection between online learning and differential privacy. Let us recall the definition of differential privacy.

Definition 14. *Consider some randomized algorithm A which takes some dataset S as input, and computes some output $A(S)$ (this could be some hypothesis built from the data). Consider some datasets S and S' which differ in just one example. We say that A satisfies ϵ-differential privacy if for any set T in the range of $A(\cdot)$ and for any such datasets S and S', we have that*

$$\frac{P(A(S) \in T)}{P(A(S') \in T)} \leq e^{\epsilon}.$$

It turns out that a hypothesis class is learnable with differential privacy *if and only if* the Littlestone dimension is finite [2, 1]. Since Littlestone dimension characterizes online learning as well, a hypothesis class can be learned with differential privacy if and only if it can be learned in the online learning model.

This also implies that thresholds cannot be learned privately. In fact, even earlier work had shown that thresholds cannot be learned privately via proper learner [3], which was later extended by [2] to improper learners. The reason behind why thresholds cannot be learned privately is that, in general, revealing the threshold can reveal information about where the datapoints are located. In particular, if the datapoints are supported on an infinite set, then revealing threshold function can potentially reveal at least one of the training datapoints.

4 Online learning in the unrealizable case

So far, we've talked about online learning in the realizable case, and defined a complexity measure (Littlestone dimension) which characterizes when online learning is possible. We also showed that finite hypothesis class are online learnable. Following a similar outline as with PAC learning, the next step is to remove the realizability assumption. This brings us to the notion of *regret*.

Definition 15 (Regret)**.** *The regret of an algorithm A relative to a hypothesis h when run on a sequence of T examples is:*

$$Regret_A(h, T) = \sup_{(x_1,y_1),\cdots,(x_T,y_T)} \left[\sum_{t=1}^{T} |p_t - y_t| - \sum_{t=1}^{T} |h(x_t) - y_t| \right].$$

The regret of A relative to a hypothesis class $\mathcal{H}$ is:

$$Regret_A(\mathcal{H}, T) = \sup_{h \in \mathcal{H}} Regret_A(h, T).$$

Note that if the sequence is realizable, this is the same as the mistake bound.

The first question to ask is if we can we get sublinear regret, ($o(T)$)? Unfortunately the answer is no.

Consider $\mathcal{H} = \{h_0, h_1\}$, h_0 always predicts 0 and h_1 always predicts 1. An adversary can force the number of mistakes made by the algorithm to T (by always having the label of the datapoint to be the opposite of the algorithm's prediction).

However, the best predictor in hindsight is the majority of $y_1 \dots y_T$ which makes $\leq T/2$ mistakes. This implies that the regret is $\geq T/2$. We have no made assumptions on the algorithm here, and even our hypothesis class is rather simple. Therefore, sublinear regret is impossible in general.

To get around this, we allow randomized algorithms. Importantly, the environment decides y_t before the random coins are flipped.

Setup At every time step f,

$\rightarrow$ learner receives $x_t \in X$

$\rightarrow$ learner decides $p_t \in [0,1]$, probability of label being 1

$\rightarrow$ environment "decides" true label $y_t \in \{0,1\}$

$\rightarrow$ learner outputs $\hat{y}_t = \begin{cases} 1, & \text{w.p.} \quad p_t \\ 0, & \text{w.p.} \quad 1-p_t \end{cases}$

$\rightarrow$ Expected loss at time t,

$$\begin{aligned} \mathbb{P}\left(\hat{y}_t \neq y_t\right) &= \begin{cases} p_t, & \text{if} \quad y_t = 0 \\ 1-p_t & \text{, if} \quad y_t = 1 \end{cases} \\ &= |p_t - y_t|. \end{aligned}$$

The definition of regret remains the same as it was before, and is the expected number of errors compared to the best possible predictor in hindsight in the class $\mathcal{H}$.

$$Regret_A(\mathcal{H}, T) = \sup_{h \in \mathcal{H}} \sup_{(x_1,y_1),\cdots,(x_T,y_T)} \left[\sum_{t=1}^{T} |p_t - y_t| - \sum_{t=1}^{T} |h(x_t) - y_t| \right]$$

Q) Can we get sublinear regret in this new setting? Yes, using the *Weighted-Majority* algorithm.

To discuss this algorithm, we need to first define the setting of prediction with "expert advice". This is a powerful setup, and comes up frequently in online learning problems.

4.1 Learning with expert advice

The setting is that there are d experts, each of which makes a prediction for the outcome at any time step. At every time t, the learner has to choose one among the d experts to base its prediction on. The learner then sees the true label, and the loss each expert incurs on that time step. The learner can use this information to guide future predictions. How well can the learner do compared to the best expert in hindsight? The Weighted-Majority algorithm provides an approach for the problem.

Algorithm 4 Weighted Majority (also known as Multiplicative Weights/Hedge)

initialize $w^{(1)} = (1, \ldots, 1)$ (d dimensional)
for for $t = 1 \ldots$ T **do**
  set $\tilde{w}^{(t)} = w^{(t)}/Z_t$ where $Z_t = \sum_i w_i^{(t)}$
  choose expert i at random according to $P[i] = \tilde{w}_i^{(t)}$
  receive costs of all expects $v_t \in [0,1]^d$
  pay expected cost: $\langle \tilde{w}^{(t)}, v_t \rangle$
  update: $\forall i : w_i^{(t+1)} = w_i^{(t)} e^{-\eta v_{t,i}}$
end for

Theorem 16. *Assuming $T > 2\log(d)$, the Weighted-Majority algorithm enjoys the bound*

$$\sum_{t=1}^{T} \langle \tilde{w}^{(t)}, v_t \rangle - \min_{i \in [d]} \sum_{t=1}^{T} v_{t,i} \leq \sqrt{2\log(d)T}$$

Proof. We have:

$$\log\left(\frac{Z_{t+1}}{Z_t}\right) = \log\left(\frac{\sum_i \tilde{w}_i^{(t)} e^{-\eta v_{t,i}}}{Z_t}\right) = \log\left(\sum_i \hat{w}_i^{(t)} e^{-\eta v_{t,i}}\right).$$

Using

1. $e^{-a} \leq 1 - a + \frac{a^2}{2}, \forall a \in (0,1)$
2. $\sum_i \tilde{w}_i^t = 1,$
3. $\log(1-b) \leq -b, (b \leq 1),$

we get

$$\begin{aligned}
\log\left(\frac{Z_{t+1}}{Z_t}\right) &\leq \log\left(\sum_i \tilde{w}^{(t)}(1 - \eta v_{t,1} + \frac{\eta^2}{2} v_{t,i}^2)\right) \\
&= \log\left(1 - \sum_i \tilde{w}_i^{(t)} \left(\eta v_{t,i} - \frac{\eta^2}{2} v_{t,i}^2\right)\right) \\
&\leq -\sum_i \tilde{w}_i^{(t)} \left(\eta v_{t,i} - \frac{\eta^2}{2} v_{t,i}^2\right) \\
&= -\eta \langle \tilde{w}^{(t)}, v_t \rangle + \frac{\eta^2}{2} \sum_i \tilde{w}_i^{(t)} v_{t,i}^2/2. \\
&\leq -\eta \langle \tilde{w}^{(t)}, v_t \rangle + \frac{\eta^2}{2}.
\end{aligned}$$

Note that

$$\begin{aligned}
\sum_{t=1}^{T} \log\left(\frac{Z_{t+1}}{Z_t}\right) &= \log\left(Z_{T+1}\right) - \log\left(Z_1\right) \\
&\leq -\eta \sum_{t=1}^{T} \langle \tilde{w}^{(t)}, v_t \rangle + \frac{T\eta^2}{2}
\end{aligned}$$

By summing the inequality over t,

$$\eta \sum_{t=1}^{T} \langle \tilde{w}^{(t)}, v_t \rangle \leq \log\left(Z_1\right) - \log\left(Z_{t+1}\right) + \frac{T\eta^2}{2}.$$

Note that $\log(Z_1) = \log(d)$. We now lower bound Z_{T+1}. Note that $w_i^{(T+1)} = e^{-\eta \sum_t v_{t,i}}$,

$$\begin{aligned}
\log\left(Z_{T+1}\right) &= \log\left(\sum_i e^{-\eta \sum_t v_{t,i}}\right) \\
&\geq \log\left(\max_i e^{-\eta \sum_t v_{t,i}}\right) \\
&= -\eta \min_i \sum_t v_{t,1}.
\end{aligned}$$

Using $\log(Z_1) = \log(d)$,

$$\eta \sum_{t=1}^{i} \langle \tilde{w}^{(t)}, v_t \rangle \leq \log(d) + \eta \min_i \sum_t v_{t,i} + \frac{T\eta^2}{2}$$

$$\implies \sum_{t=1}^{T} \langle \tilde{w}^{(t)}, v_t \rangle - \min_i \sum_t v_{t,i} \leq \frac{\log(d)}{\eta} + \frac{T\eta}{2}.$$

Choosing $\eta = \sqrt{\frac{2\log(d)}{T}}$ upper bounds the RHS by $\sqrt{2\log(d)T}$, proving the result. ■

4.2 Regret bound for online leaning

We can use the Weighted Majority algorithm to get a regret bound for finite hypothesis classes.

Theorem 17. *Let $\mathcal{H} = \{h_1, \cdots, h_d\}$ be a finite hypothesis class. Then Weighted-Majority achieves*

$$\sum_{t=1}^{T} |p_t - y_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |h(x_t) - y_t| \leq \sqrt{2\log(|\mathcal{H}|)T}$$

Proof. Each experts h_i predicts $h_i(x_t)$ on example x_t. The loss $v_{t,i} = |h_i(x_t) - y_t|$.
The predictions of Weighted-Majority are

$$p_t = \sum_i \tilde{w}_i^{(t)} h_i(x_t).$$

The expected loss is

$$\begin{aligned} |p_t - y_t| &= \left| \sum_{i=1}^{d} \tilde{w}_i^{(t)} h_i(x_t) - y_t \right| \\ &\leq \sum_{i=1}^{d} \left| \tilde{w}_i^{(t)} \left(h_i(x_t) - y_t \right) \right| \\ &= \langle \tilde{w}^{(t)}, v_t \rangle \end{aligned}$$

Then we use the regret bound for Weighted-Majority to bound $\langle \tilde{w}^{(t)}, v_t \rangle$. ■

For infinite sized hypothesis classes, we can prove a regret bound using Littlestone dimension.

Theorem 18. *For every hypothesis class $\mathcal{H}$, there exists an algorithm for online learning with a regret bound:*

$$\sum_{t=1}^{T} |p_t - y_t| - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} |h(x_t) - y_t| \leq \sqrt{2\text{Ldim}(\mathcal{H})\log(eT)T}$$

The proof of this result can be found in Section 21.2.1 of [4].

5 Online Convex Optimization

Just as we saw with the PAC learning settings, convex functions can be learned efficiently in the online setting as well.

In online convex optimization, at each timestep, the learner chooses $w_t \in S$ (for some domain S). We have convex loss $f_t : S \to \mathbb{R}$ at every time t. The learner suffers $f_t(w_t)$ at time t. We can write regret as:

$$\text{Regret}(T) = \sum_{t=1}^{T} f_t(w_t) - \min_{w \in S} \sum_{t=1}^{T} f_t(w)$$

Example 19 (Linear regression)**.** *This is just the online version of our prototypical linear regression problem. Let us assume, $S = \mathbb{R}^d$. The loss at any time step t is*

$$f_t(w) = (\langle w, x_t \rangle - y_t)^2 \textit{(this is convex as a function of } w\textit{).}$$

Note that the data (x_t, y_t) are baked into f_t.

Example 20 (Learning with experts)**.** *Recall our previous learning with experts setting. The set of experts are discrete which is not a convex set. However, we can create a convex set by drawing from a distribution over experts.*

$$S = \Delta_d = \{w \in \mathbb{R}^d, w_i \geq 0 \ \forall i \in [d], \sum w_i = 1\} (\textit{simplex over } \mathbb{R}^d),$$
$$f_t(w) = \langle w, v_t \rangle,$$
$$\textit{where } v_t = (l(h_1(x_t), y_t), l(h_2(x_t), y_t), s, l(h_d(x_t), y_t)), (l(h_i(x), y) \in [0, 1]) \ \forall \ i.$$

As before, note that f_t is merging the loss function and data.

6 Algorithmic Frameworks

We will now discuss some algorithmic frameworks for online convex optimization. As with convex optimization previously, the algorithms themselves are useful even if the functions are not convex.

6.1 Algorithm: Follow-the-Leader (FTL)

At every timestep t, we play

$$w_t \in \arg\min_{w \in S} \sum_{i=1}^{t-1} f_i(w) \tag{1}$$

The following Lemma shows that FTL has bounded regret if it does well as compared to a one-step lookahead oracle.

Lemma 21 (FTL vs. one-step lookahead oracle)**.** *Let $w_1, \cdots, w_T$ be produced by the FTL algorithm according to Equation 1. For any $u \in S$, define*

$$\text{Regret}(u, T) = \sum_{t=1}^{T} \big[f_t(w_t) - f_t(u) \big]$$

Then $\text{Regret}(u, T) \leq \sum_{t=1}^{T} \big[f_t(w_t) - f_t(w_{t+1}) \big]$*. This also gives a bound for regret with respect to any $u \in S$,*

$$\text{Regret}(T) = \max_{u \in S} \text{Regret}(u, T) \leq \sum_{t=1}^{T} \big[f_t(w_t) - f_t(w_{t+1}) \big].$$

Proof. It suffices to show that $\forall u \in S$,

$$\sum_{t=1}^{T} f_t(w_{t+1}) \leq \sum_{t=1}^{T} f_t(u)$$

We prove this by induction. The base case can be shown easily. By the inductive hypothesis on $\tau - 1$,

$$\begin{aligned} &\sum_{i=1}^{\tau-1} f_t(w_{t+1}) \leq \sum_{i=1}^{\tau-1} f_t(u) \ \forall u \in S \\ \implies &\sum_{t=1}^{\tau} f_t(w_{t+1}) \leq \sum_{i=1}^{\tau-1} f_t(u) + f_\tau(w_{\tau+1}) \ \forall u \in S \end{aligned}$$

Applying this for $u = w_{\tau+1}$,

$$\sum_{t=1}^{\tau} f_t(w_{t+1}) \leq \sum_{t=1}^{\tau} f_t(w_{\tau+1})$$

Since $w_{\tau+1} \in \arg\min_{w \in S} \sum_{t=1}^{\tau} f_t(w)$,

$$\sum_{t=1}^{\tau} f_t(w_{t+1}) \leq \sum_{t=1}^{\tau} f_t(u) \ \forall u \in S$$

This completes the inductive step and the result follows. ∎

The regret bound depends on the stability of the FTL iterates,

$$\text{Regret}(T) \leq \sum_{t=1}^{T} \big[\underbrace{f_t(w_t) - f_t(w_{t+1})}_{\text{stability of FTL}} \big].$$

If

7 Further reading

You can refer to Percy Liang's lecture notes for more on kernels. You can read Chapter 21 of [4] for more on online learning.

References

[1] Mark Bun, Roi Livni, and Shay Moran. An equivalence between private classification and online prediction. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 389–402. IEEE, 2020.

[2] Noga Alon, Roi Livni, Maryanthe Malliaris, and Shay Moran. Private pac learning implies finite littlestone dimension. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 852–860, 2019.

[3] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil Vadhan. Differentially private release and learning of threshold functions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 634–649. IEEE, 2015.

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