Lecture 10: Online Learning

Instructor: Aditya Bhaskara   *Scribe: Yiliang Shi*

CS 5966/6966: Theory of Machine Learning

February 13th, 2017

Abstract

This lectured first reviewed optimization from the past lectures, then introduced new framework of online learning.

1 Review

Last week, we discussed gradient descent. Let f be a convex function on a convex domain S. Suppose that f is ρ-Lipschiz. We showed that after O($1/\epsilon$) iterations, we get ϵ close in f value to the optimum.

Gradient descent is pretty flexible. If the examples are from another domain, it is possible to project from the other domain. If it is not ρ-Lipschiz, it is possible to take the sub gradient.

A key inequality involved in gradient descent is convexity. $f(x^{(t)}) - f(x^*) \leq \langle x^{(t)} - x^*, \nabla f(x^{(t)}) \rangle$, which implies $\frac{1}{T}\sum_{t=1}^{T}(f(x^{(t)}) - f(x^*)) \leq \frac{B^2}{2\eta} + \frac{T\rho^2\eta}{2}$, where B is the difference between the actual x and x^*

The more assumptions we make on gradient descent, the better our gurantees. If f is β-smooth, O($1/\epsilon$) iterations suffice. If f is β-smooth and α convex, $O(\frac{\beta}{\alpha}\dot{\log}(1/\epsilon))$ iterations suffice. β is basically a tangent above the curve while α is a tangent below the curve. $\frac{\beta}{\alpha}$ is basically the condition number used in linear regression.

When the gradient is too expensive to compute, an alternative is stochastic gradient-descent. We can decompose f into an average, $f = \frac{1}{m}[f_1 + f_2 + \cdots + f_m]$ where ∇f_i is easier to compute. Instead of computing ∇f each time, we replace it with a random ∇f_i. The expectation is correct.

1.1 THEOREM. *Suppose that f_i's are all ρ-Lipschitz, and we run SDG for T iterations with T= $\frac{B^2\rho^2}{\epsilon^2}$ and$\eta = \frac{\epsilon}{\rho^2}$. Then $\mathbb{E}[f(\bar{w}) - f(w^*)] \leq \epsilon$*

Proof: $\|x^{(t+1)} - x^*\|^2 - \|x^{(t)} - x^*\|^2 = 2\eta\langle x^{(t)} - x^*, g^{(t)}\rangle + \eta^2\|g_t\|^2$

2 A New Framework - Online learning

Thus far, we assume that $x \sim D$.

Motivation: What happens then if we dropped the distribution assumption in learning?

We want to do well in the future, given the examples we've seen in the past. Given a hypothesis class H, we want to compete with the best in the class instead of the best in hindsight.

With online algorithms, the future might be adversarial. An analogy is the secretary problem - when interviewing candidates to hires, you don't know anything about the next candidate.

Online Learning Setup

Given a hypothesis class H, "data points" arrive one by one. We don't know the label of each "data point" until we make a prediction. After making a prediction, we can see the loss for the prediction.

	$x^{(1)}$	$x^{(2)}$	$\cdots$	$x^{(T)}$
h_1	0	0		1
h_2	1	0		
h_3				
$\vdots$				$\vdots$
h_N				
Alg.	y_1		$\cdots$	y_T

We have a collection of hypothesis $h \in H$ that makes a prediction on the examples seen so far. At every t, we make a 'prediction' y_t according to a function of $x^{(t)}$. Next, we get to see the true value of $f(x^{(t)}$. We then know the loss for the prediction, loss = $\mathbb{1}[y_t \neq f(x^{(T)}]$.

Out goal in online learning is to compete with the best hypothesis in H in hindsight.

Example 1

Assume we have 2 hypothesis h_1, h_2 in our hypothesis class. Our algorithm predicts label of the next example using the hypothesis that made the least mistakes so far.

	x_1	x_2	$\cdots$
h_1	1 x	0 $\surd$	$\cdots$
h_2	0 $\surd$	1 x	$\cdots$

At t=1, h_1 predicts a 1 while h_2 predicts a 0. Say our algorithm goes with h_1 and predicts 1. Then the true label $f(x_1)$ is revealed to be 0. At t=2, h_1 predicts a 0 while h_2 predicts a 1. Since the mistakes so far made by h_1, m_1, is 1 while the mistakes made by h_2, m_2, is 0, the algorithm selects h_2 as the predicted true hypothesis and predicts 1. However, the true label $f(x_2)$ in this case is 0. The algorithm would have made 2 mistakes, while m_1 and m_2 both equal 1.

The goal of online learning is to ensure that the number of mistakes the algorithm makes is less or equal to that of the best hypothesis in hindsight, a.k.a. the hypothesis that made the least mistakes.

$$\#mistakes(Alg) \leq min\{m_1, m_2\} + \text{"}small\text{"}$$

3 Realizable case

First, we look at the realizable case, where $\exists$ a hypothesis h^* that makes no mistakes.

$$h^*(x_i) = f(x_i) \forall i = 1 \cdots T$$

A simple algorithm to find $h*$ is to start with all hypothesis in H, then eliminate everyone that makes a mistake with each update. For instance, if $f(x_1) = 1$, then every hypothesis that predicted 0 is eliminated.

More generally, at each time t, we maintain "candidate" hypothesis $H_t \subseteq H$. At time $t+1$, the algorithm predicts what the majority of H_t predicts. Then we update H_t, removing all the incorrect hypothesis.

Mistake Bound

The next question is if we can bound the mistakes made by the algorithm. If the algorithm did not make a mistake at time t, we do not have control over its update. It is possible that all $h \in H_t$ is correct, or there might be some that are wrong.

What happens then if the algorithm made a mistake at time t? If the algorithm makes a mistake, at least half of the hypothesis must be wrong. In this case, we know that we reduced the size of $|H|$ by half or more.

$$|H_{t+1}| \leq |H_t|/2$$

If we start off with $|H_0| = N$, than we can halve at most $[\log N]$ times, as $\exists$ a h^* that is always right, which implies that $|H_t| \geq 1 \ \forall t$.

3.1 theorem. *Suppose $\exists$ a $h \in H$ that makes no mistakes, the the number of mistakes made by the algorithm is $\leq [\log N]$.*

Littlestone vs VC dimension

The example and proof above assumes a finite hypothesis class. Is there a way to handle $N = \infty$? For PAC learning, having small VC dimensions was equivalent.

It turns out that there is a equivalent concept called Littlestone dimension, which "characterizes" learnability in the sense of mistake bounds. We will not go into the exact definition of Littlestone dimension as it is involved.

3.2 claim. *We can say that if we have a concept class H with VC dim = d, no online algorithm can make fewer than d mistakes.*

This is easy to prove. The general idea is to shatter H into $s_1, s_2, \cdots, s_d$. For each s_i, set the true value $f(s_i)! = Alg(s_i)$. The algorithm must be wrong at least d times.

Next, suppose that VC-dim(H) is small. Can we then ensure that $\exists$ an algorithm that makes a small number of mistakes?

The answer is no, as shown from the example of linear thresholds. With linear thresholds, $\exists$ a τ that divides H into positive and negative labels and has a VC dimension of 1. If the above claim is true, then there must exist an algorithm that can make O(1) constant mistakes

However, it is possible for an adversary to generate a sequence of examples where the algorithm always end up guessing incorrectly. Every time the algorithm says the example is positive, the adversary and reveal the true label to be negative and imply that τ is to its right. If the algorithm says the sample is negative, the adversary can reveal the true label to be positive and imply that τ is to the left.

The process can continue on forever. Unlike with PAC learning, it is not possible to stop learning once we are within a certain accuracy of τ, since there is no concept of distribution.