Homework 6
APPM 4490/5490 Theory of Machine Learning, Spring 2024

Due date: Friday, Mar 1 '24, before 11 AM, via paper or via Gradescope

Instructor: Prof. Becker

Revision date: 2/24/2024

Theme: Boosting

Instructions Collaboration with your fellow students is OK and in fact recommended, although direct copying is not allowed. The internet is allowed for basic tasks (e.g., looking up definitions on wikipedia) but it is not permissible to search for proofs or to *post* requests for help on forums such as `http://math.stackexchange.com/` or to look at solution manuals. Please write down the names of the students that you worked with.

An arbitrary subset of these questions will be graded.

Reading You are responsible for reading chapter 10 about "boosting" of Understanding Machine Learning by Shai Shalev-Shwartz and Shai Ben-David (2014, Cambridge University Press), as well as chapter 7 in Foundations of Machine Learning, 2nd edition, by Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.

Comment AdaBoost is the classical boosting algorithm that boosts weak learners into strong learners in the setting of realizable PAC learning, e.g., enables you to reduce the training error arbitrarily low. This homework explores other types of "boosting" ideas; in particular, instead of using boosting to lower the error, these problems will use boosting to lower the failure probability.

Problem 1: Exercise 10.1 in Shalev-Shwartz and Ben-David, slight typo fixes (i.e., the expression is a bound for $m_{\mathcal{H}}(2\epsilon, 2\delta)$ not $m_{\mathcal{H}}(\epsilon, \delta)$). Suppose $\mathcal{H}$ is a fixed hypothesis class, and $\mathtt{A}$ is an algorithm such that learns it with arbitrary accuracy, but only with probability $1-\delta_0$ (not with arbitrarily high probability), i.e., if $m \geq m_{\mathcal{H}}(\epsilon)$ then for every distribution $\mathcal{D}$, with probability at least $1-\delta_0$, $L_{\mathcal{D}}(A(S)) \leq \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$.

Suggest a procedure that relies on $\mathtt{A}$ and learns $\mathcal{H}$ in the usual agnostic PAC learning model and has a sample complexity of

$$m_{\mathcal{H}}(2\epsilon, 2\delta) \leq k \cdot m_{\mathcal{H}}(\epsilon) + \left\lceil \frac{2\log(2k/\delta)}{\epsilon^2} \right\rceil, \quad \text{where} \quad k = \left\lceil \frac{\log(\delta)}{\log(\delta_0)} \right\rceil.$$

Hint: Divide the data into $k+1$ chunks, where each of the first k chunks is of size $m_{\mathcal{H}}(\epsilon)$, and train each chunk using $\mathtt{A}$. Argue that the probability that *all* of these k classifiers, $h_j = \mathtt{A}(S^{(j)})$ for $j \in [k]$, fails to be good is at most $\delta_0^k = \delta$, i.e.,

$$\mathbb{P}\left(\bigwedge_{j=1}^{k} \left(L_{\mathcal{D}}(h_j) > \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon \right) \right) < \delta_0^k.$$

Then use the last chunk (which has a different size), to choose from the k hypotheses h_j, $j \in [k]$, and apply Corollary 4.6.

Problem 2: Exercise 2.2.8 in Vershynin's High-Dimensional Probability (2018, Cambridge University Press). Suppose you have a randomized algorithm that computes a binary output, and the algorithm can either be right or wrong (for example, it could be an algorithm that applies a stochastic

primality test). Suppose your algorithm does only slightly better than chance, so that it is incorrect with probability $\delta_0 = 1/2 - \gamma$ for some margin $\gamma > 0$. Show that you can boost the probability of success arbitrarily high, to $1 - \delta$ for any $\delta \in (0, 1)$, by running the algorithm k times (for different random seeds) and taking the majority vote, provided

$$k \geq \left\lceil \frac{1}{\gamma^2} \log\left(\frac{1}{\delta}\right) \right\rceil.$$

Hint: Let X_j be an indicator variable that indicates whether the j^{th} run of the algorithm was successful or not, and use Hoeffding's inequality. In fact, you can also show that you only need $k \geq \left\lceil \frac{1}{2\gamma^2} \log\left(\frac{1}{\delta}\right) \right\rceil$ but you don't need to show that extra factor of 2 for your homework.

Problem 3: [**for APPM 5490 students only** (APPM 4490 students do not need to do this)] Consider checking if two $n \times n$ matrices A and B are equal, but at least one matrix is given only implicitly (if both matrices are explicitly given, then just compute $\|A - B\|_F$ in $\mathcal{O}(n^2)$ time). For example, your friend claims that they can multiply two matrices with their code, so they compute $C \cdot D = A$, but you're suspicious. You have C and D, and you could compute $C \cdot D = B$ using a code that you trust, but this multiplication is expensive ($\mathcal{O}(n^{2.81})$ using Strassen or $\mathcal{O}(n^3)$ using classical multiplies).

Freivalds' algorithm from the 1970s solves this problem by picking a vector $x \in \mathbb{R}^n$ where each entry $x_i \in \{0, 1\}$ is iid Bernoulli with $p = 1/2$. You then compute Ax and Bx and check if $Ax = Bx$. If $A = B$, then it's always true that $Ax = Bx$. If $A \neq B$, then there's a chance that $Ax = Bx$ still, since, for example, you could have chosen $x = 0$.

For this problem, do not use the internet! The wikipedia page on Freivald's contains too much information (though of course you can check out the wikipedia page after you turn in your homework).

a) Prove that if $A \neq B$, then $\mathbb{P}[Ax = Bx] \leq \frac{1}{2}$.

b) Describe a practical variation of the above that redraws new random x, and stops when it either certifies $A \neq B$ or if it can guarantee $A = B$ with probability at least $1 - \delta$ (more precisely, we mean that if $A \neq B$ then the chance of us failing to observe this is at most $1 - \delta$). In particular, for a given δ, how many random draws of x would it take to have $1 - \delta$ confidence in our result?

An interesting idea would be to use a variant of this to prove whether two functions f and g are equal, where f is known (so we know some smoothness property of it) and we work on a bounded domain. This could be useful for an automatic grading program, where a calculus student enters some expression for an answer (say, $g(x) = \sin^2 x + \cos^2 x$) and you want to check if they got the problem right (the answer should be $f(x) = 1$) without knowing whether they simplified things or not, but you can use a computer to evaluate it at a few sample points x. A further variant of this is to figure out whether two algorithms are essentially the same; look at An automatic system to detect equivalence between iterative algorithms by Zhao, Lessard, and Udell 2021 if you're interested.

Another variant is the Deutsch–Jozsa algorithm which is interesting (though not of practical interest) because it's efficient on a quantum computer but slow on a *deterministic* classical computer.