CS 391L: Machine Learning **Spring 2021**

Homework 2 - Theory - Solutions

Lecture: Prof. Adam Klivans

Keywords: Perceptron, SGD, Boosting

1. Yes, order matters, even when the dataset is linearly separable. Here is a simple example with points lying in $\mathbb{R}^2$ that demonstrates this. Let $S = \{((-1,1),+1),((1,-2),+1)\}$, and let $S' = \{((1,-2),+1),((-1,1),+1)\}$. Observe that this dataset is linearly separable since for example $w^* = (-3,-2)$ classifies it correctly.

   Suppose we start in both cases with weights $w = (1,0)$. In the case of S we misclassify the first point $(-1,1)$ and update the weights to $(0,1)$, and then misclassify the second point $(1,-2)$ as well and finally end up with weights $(1,-1)$. But in the case of S' we get the first point $(1,-2)$ correct, but misclassify the second point $(-1,1)$ and finally end up with weights $(0,1)$.

2. In Perceptron, we update only when we make a mistake, and the update rule is

$$w_{\text{new}} = w_{\text{old}} + \text{sign}\left(w^T x^i\right) x^i = w_{\text{old}} + y^i x^i$$

   For SGD, the update rule is

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla \phi\left(y^i w^T x^i\right) = w_{\text{old}} - \eta \begin{cases} 0 & \text{if } y^i w^T x^i \geq 0, \text{ i.e. no mistake} \\ -y^i x^i & \text{if } y^i w^T x^i < 0, \text{ i.e. there is a mistake} \end{cases}.$$

   So when step size $\eta = 1$, SGD's update rule is exactly the same as Perceptron, and hence SGD behaves essentially exactly the same as Perceptron. (**Note to peer graders:** the emphasis in this problem is not on the order but rather on the update rule. One way to phrase things would be to say that Perceptron is essentially SGD when the order of training points is randomized.)

3. (a) Suppose that the true labeling function is $c = h_{\theta_1,\theta_2,b}$. Since our distribution is only supported on $[-B,B]$, the behavior of c is determined by the intervals $[-B,\theta_1)$, $[\theta_1,\theta_2]$, and $(\theta_2,B]$. Since the total probability mass of any distribution is 1 and these are three disjoint intervals, one of them must have probability mass at most 1/3. Whichever one it is, we can pick a decision stump $h_{\theta,b}$ that agrees with $h_{\theta_1,\theta_2,b}$ on the other two, and hence has error at most 1/3 (since the error is just the total probability mass of the regions where we err). (For example, suppose the middle interval $[\theta_1,\theta_2]$ has mass at most 1/3. Then we can put our threshold at B and classify everything to the left as $-b$, i.e. we can consider $h_{B,-b}$. Similarly in the other cases we can put our threshold at θ_1 or θ_2 for a similar effect.)

   (b) Since we are only concerned our performance on the m training points, we can assume without loss of generality that we pick one of them as our threshold. We can now look for the best threshold by simply trying all of the m points and calculating the associated error. For any fixed threshold calculating the error just takes a linear scan, so this takes $O(m^2)$ time overall and is efficient enough for our purposes. As an optimization, it is possible to sort the points ahead of time and then proceed in sorted order while carefully keeping track of the error (instead of recalculating it every time). We omit the fine details here, but this takes $O(m \log m)$ time. (**Note to peer graders:** an optimized solution is not required, and $O(m^2)$ time is fine so long as the approach is correct.)

(c) The simple explanation is just that $\mathcal{H}$ has less *model complexity* than $\mathcal{C}$. It is possible to quantify this in many ways, such as minimum description length or VC dimension (the latter being outside the scope of this class), but intuitively the point is very clear. Since generalization is easy to achieve for classes of low complexity, not many training examples are required to achieve a very close approximation of the true error. (**Note to peer graders:** this question was only an intuitive and not a technical question, so no more technical detail than this is required.)

Further comment. Part (a) shows that using $\mathcal{H}$ indeed yields weak learners for $\mathcal{C}$, and parts (b) and (c) show that we can do this efficiently. Altogether we have a simple concrete example of a weak learner for a class. It is worth stressing again that this whole example is just for illustrative purposes; the classes in this problem happen to be simple enough that bothering with weak learning is not really even necessary.

4. Let W^+, W^- denote the sum of the weights of points correctly and incorrectly classified by h_t with respect to D_t, respectively. Then we can write the error $E = \frac{W^-}{W^+ + W^-}$ and $\beta = \frac{E}{1-E} = \frac{W^-}{W^+}$. For D_{t+1}, we update the weights of the points we got correct to βW^+ and leave the incorrect ones the same. Thus we can compute the error E' of h_t with respect to D_{t+1} as

$$E' = \frac{W^-}{\beta W^+ + W^-} = \frac{W^-}{\frac{W^-}{W^+} W^+ + W^-} = \frac{W^-}{2W^-} = \frac{1}{2}$$