Seminar 1

Kirill Korolev

September 14, 2022

1.7 Let $H_1, H_2, ...$ be be a sequence of finite hypothesis classes from proposition 1.4. To each classifier $f \in H_k$ we assign weight w_f from Weighted Halving algorithm.

$$w_f = \frac{1}{k(k+1)|H_k|} \tag{1}$$

We decrease weight of each classifier that is mistaken on input x_t as we did in Weighted Majority algorithm and no classifier is discarded after the mistake.

We define

$$W_t = \sum_{f \in H} w_{ft} \tag{2}$$

$$F_t = \sum_{\substack{f \in H \\ f(x_t) = y_t}} w_{ft} \tag{3}$$

where H represents $H_1 \cup H_2 \cup ...$. Then the same inequalities hold for the upperbound on W_t as in Weighted Majority algorithm. After s mistakes we have

$$W_t \leq \left(\frac{\beta + 1}{2}\right)^s W_1 \tag{4}$$

W_1 is the sum of all weights at the beginning of the algorithm. From the proof of the Weighted Halving algorithm we know that it is equal to 1.

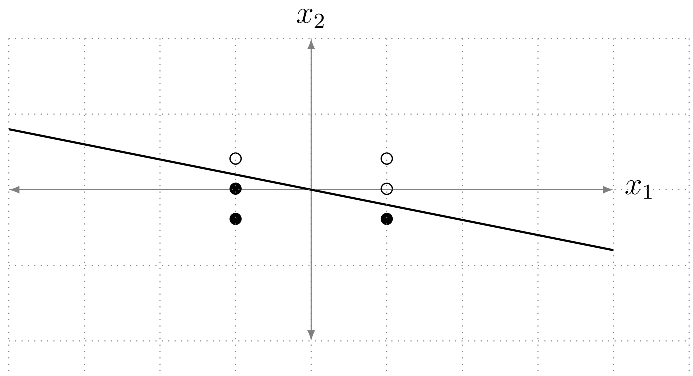
Also, in agnostic setting we have true function f^* that can make at most $m*$ mistakes. Suppose $f^* \in H_{k^*}$. Then its weight has been decreased by β at most $m*$ times.

$$\frac{\beta^{m*}}{k^*(k^*+1)|H_{k^*}|} \leq W_t \tag{5}$$

By taking logarithms from both sides and using bounds from WH and WM algorithms we end up with the bound from the task. Then we assign $H_k = T_{dk}$ for all k and $f^* = T_{dk^*}$.

Basically, we operate as in Weighted Majority algorithm with weights from halving variant and partitioning H_k, so we don't need to know size of a tree in advance.

1.8 Let's draw some points from $RN^{-1}(-1)$ and $RN^{-1}(1)$ in $\mathbb{R}^2$.



We see that line $x_1 + 2x_2 = 0$ perfectly separates two classes. Let's try to generalize this idea.

Consider strings of length d. We want to find a separating hyperplane of a form $w_1x_1 + w_2x_2 + \ldots + w_dx_d = 0$. Let's take $w = (w_1, \ldots, w_d) = (1, 2, \ldots, 2^{d-1})$ and show why it separates points from different classes. Let $x = (x_1, x_2, \ldots, x_{l-1}, 1, 0, \ldots, 0) \in RN^{-1}(1), l = 1, \ldots, d$. Whichever values we set instead of x_j $j = 1, \ldots, l-1$ dot product with w gives us a positive sign which means x lies above the hyperplane. Because in worst case $x_j = -1$ for all j the geometric sum with $q = 2$ equals $2^l - 1 < 2^l$. For other combinations of x_j 2^l would be even bigger. So indeed points from the positive class lie above the hyperplane. Symmetrically for the negative class we apply the same idea and all points lie below the hyperplane.

Now let's proof that the Perceptron algorithm makes at least $\Omega(2^d)$ updates. We know that a vector which is normal to a hyperplane is $w = (1, 2, \ldots, 2^{d-1})$ and we want our algorithm to converge to w. Starting from the zero vector in order to get 2^{d-1} in a last component we must perform at least 2^{d-1} updates, because each coordinate $|x_i| \leq 1$, which is exactly $\Omega(2^d)$.