A variant of the Perceptron Algorithm:

proof of lower bound

Sophia Kolak - sdk2147

Problem: One is often interested in learning a disjunction model. That is, given binary observations from a d feature space (i.e. each datapoint $x \in \{0,1\}^d$, the output is an 'OR' of few of these features (e.g. $y = x_1 \vee x_{20} \vee x_{33}$). It turns out that the following variant of the perceptron algorithm can learn such disjunctions well.

Perceptron OR variant

learning:

- *Initialize w := 1*
- *for each datapoint (x, y) in the training dataset*
    - $\hat{y} := 1[w \cdot x > d]$
    - *if* $y \neq \hat{y}$ *and* $y = 1$
        - $w_i \leftarrow 2w_i \quad (\forall i : x_i = 1)$
    - *if* $y \neq \hat{y}$ *and* $y = 0$
        - $w_i \leftarrow w_i/2 \quad (\forall i : x_i = 1)$

Classification:

$f(x) := 1[w \cdot x > d]$

Prove that the Perceptron OR variant makes at most $2+3r(1+\log d)$ mistakes when the target concept is an OR of r variables

Solution:

Let $i \in r$ represent the indices in our d dimensional vector that we take the disjunction of to determine the true label of our samples. Let $TW(t)$ denote the total weight of w at iteration t, let M_+ denote the total number of positive mistakes, and let M_- denote the total number of negative mistakes

(Claim 1:) $w_i : i \in r$ is strictly increasing:

The only way for any dimension of the weight vector to decrease is in the case where we mis-classify a true negative sample, and execute the statement $w_i \leftarrow w_i/2$ $(for\ \forall i : x_i = 1)$. But for x to be a true negative, it must be the case that all $i \in r$ are equal to 0 in that sample. Thus, the weight at the indices $i \in r$ can never decrease.

(Claim 2:) Once all $w_i : i \in r$ are $> d$, it is impossible for any positive sample to be mis-classified:

By definition, positive samples have some $x_i = 1$ such that $i \in r$, thus if all $w_i : i \in r > d$, then the dot product $w \cdot x$ must be $> d$, and the sample will be correctly classified.

The question now is simply how many positive mistakes it will it take for each $i \in r$ to become $> d$.

(Claim 3:) We double the value of some $w_i : i \in r$ on each positive mistake:

On each positive mistake, there must be some x_i such that $i \in r$ and x_i has value 1, therefore, we are doubling the value of some $w_i : i \in r$ on each positive mistake. If w_i starts at 1 and is strictly increasing, it will take $log_2(d)$ positive mistakes for a given $i \in r$ to equal d, and $1 + log_2(d)$ for the dot product $w \cdot x$ to be greater than d.

We have already shown that we can no longer make mistakes when $i \in r$ is $> d$ for each $i \in r$, which means that after $r \times (1 + log_2(d))$ positive mistakes, each value $w_i : i \in r$ will be $> d$. Thus, we have proved the upper bound on positive mistakes of

$$r(1 + log_2(d))$$

A mistake is made on a negative sample when $\hat{y} = 1$, but $y = 0$. That is to say, for every mistake made on a negative sample, the algorithm will execute the statement

$$w_i \leftarrow w_i/2 \ (for \ \forall i : x_i = 1)$$

Let $S_0 = \sum_i w_i$ before we update the weight vector on the current iteration. On each negative mis-classification, S_0 will decrease by $w_i/2$ when $x_i = 1$, and stay the same otherwise. But since x is a binary vector, this is equivalent to subtracting:

$$\frac{1}{2}(w \cdot x)$$

From S_0. Thus the sum after re-weighting can be written as:

$$S_1 = \sum_i w_i - \frac{1}{2}(w \cdot x)$$

Since this is a negative mistake, the the algorithm must have classified the sample as 1, which means that $(w \cdot x) > d$, and

$$\frac{1}{2}(w \cdot x) > \frac{1}{2}(d)$$

But $\frac{1}{2}(w \cdot x)$ is precisely the quantity we are subtracting from S_0, or the total weight sum. Thus, each mistake made on a negative sample must decrease the total weight $\sum_i w_i$ by at least $\frac{1}{2}(d)$

(1) $0 < TW(t)$

We initialize each i in our weight vector to 1. Suppose for any $i \in d$, we re-weight on positive mistakes x times, and on negative mistakes y times. Then w_i could only equal zero when the following equation is true:

$$\frac{1 \cdot 2^x}{2^y} = 0$$

This equation clearly has no solutions. Since each w_i is always greater than 0, we can safely conclude that the total weight of our vector is always greater than zero, $0 < TW(t)$

(2) $TW(t) \leq TW(0) + d(M_+) - (d/2)M_-$

The only time TW(t) changes after its initialization is on either a positive or negative mistake. We already showed that each M_- will decrease TW(t) by at

least $d/2$. Now we must show that each M_+ increases the weight of the vector by at most d.
Claim: each M_+ increases TW(t) by at most d
Adopting the terminology used in the previous problem, we can write:

$$S_1 = \sum_i w_i + (w \cdot x)$$

Since this is a positive mistake, it must be the case that $w \cdot x \leq d$, thus our added term is never more than d.
We have now proved that

$$0 < TW(t) \leq TW(0) + d(M_+) - (d/2)M_-$$

(3) Observe that $TW(0) = d$, since we initialize the weight vector to all ones. This allows us to write:

$$\begin{aligned} 0 < TW(t) &\leq d(1 + M_+ - 1/2M_-) \\ 0 &< 1 + M_+ - 1/2M_- \\ M_- &< 2 + 2M_+ \end{aligned}$$

Now substituting, we can write our final expression $M_- + M_+$ as:

$$\begin{aligned} M_- + M_+ &< 2 + 2r(1 + \log d) + r(1 + \log d) \\ M_- + M_+ &< 2 + 3r(1 + \log d) \end{aligned}$$

Hence, we have proven that the total number of mistakes is at most
$2 + 3r(1 + \log d)$