Perceptron and Winnow
Cynthia Rudin

1 Perceptron

The Perceptron algorithm (Block, 1962; Novikoff, 1964) is quite an old algorithm. There are three reasons to teach it to you: (1) it works really well, and people still use it (2) it's really fun to study, and (3) you would be an embarrassment to the machine learning world if you didn't know it, and I don't plan to let that happen!

The Perceptron and Winnow algorithms are *online learning* algorithms, which means they receive observations one at a time, and adjust the decision boundary as they see each point, if desired. For now, let us assume that the data are separable (which means there's a separating hyperplane).

The basic idea of the Perceptron and Winnow algorithms is very simple: at each iteration, if the observation given to the algorithm is correctly classified, do nothing. If the observation is incorrectly classified, move the decision boundary towards that point.

Notation recap: Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{+1, -1\}$, be our dataset for binary classification. Since we are doing online learning, we see only one data point at a time. We might cycle around the points more than once. The order in which the points are seen will affect how fast we converge in practice.

Our linear model can be written as $f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, where $b \in \mathbb{R}$ is the intercept term. We can always augment the data by appending 1 at the end of the feature vector: $\widetilde{\mathbf{x}}_i \leftarrow [\mathbf{x}_i, 1]$. If we write $\widetilde{\mathbf{w}} = [\mathbf{w}, b]$, the linear model can be written as $f_{\widetilde{\mathbf{w}}}(\widetilde{\mathbf{x}}) = \widetilde{\mathbf{w}} \cdot \widetilde{\mathbf{x}}$. Thus without loss of generality, we will drop the intercept term b and consider only linear models $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. As a binary classification task, the prediction of the model is $\hat{y} = \text{sign}(f_{\mathbf{w}}(\mathbf{x}))$. Recall that the margins are "signed distances" from data points to the decision hyperplane. Using the linear model $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$, the margin for point $(\mathbf{x}_i, y_i)$ is $y_i \mathbf{w} \cdot \mathbf{x}_i$.

Perceptron Algorithm

The Perceptron algorithm iteratively updates the weight as follows:

- Initialize $\mathbf{w} \leftarrow \mathbf{0}$ (all-zero vector).
- Repeat until no points are wrongly classified:
  - if $\mathbf{x}_i$ is correctly classified, i.e., $y_i \mathbf{w} \cdot \mathbf{x}_i > 0$, then do nothing;
  - if $\mathbf{x}_i$ is wrongly classified, i.e., $y_i \mathbf{w} \cdot \mathbf{x}_i < 0$, then move $\mathbf{w}$ towards $\mathbf{x}_i$: $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$;

Why is this step helpful? Let's look at how the margin of point i changes when we make this adjustment. As you will see, it improves.

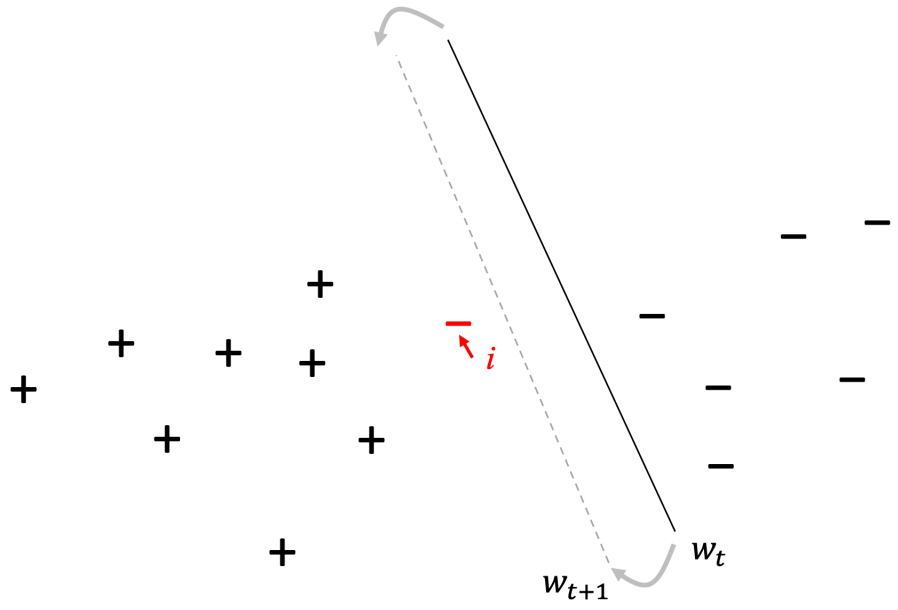

Figure 1: The decision boundary moves closer to the (misclassified) point i at iteration $t+1$. The margin of i improves during this process.

Let us say that at iteration t, the Perceptron finds that the point which it is given (point i) is misclassified. In that case, it will update the weight vector as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_i \mathbf{x}_i. \tag{1}$$

Here is what happens to the margin of point i after this update:

$$\begin{aligned}(\text{margin of } i \text{ at } t+1) = y_i(\mathbf{w}_{t+1}\mathbf{x}_i) &= y_i\left((\mathbf{w}_t + y_i\mathbf{x}_i)\cdot\mathbf{x}_i\right) \\ &= y_i(\mathbf{w}_t\cdot\mathbf{x}_i) + y_i(y_i\mathbf{x}_i\cdot\mathbf{x}_i) \quad (\text{Note: } y_i\cdot y_i = 1) \\ &= y_i(\mathbf{w}_t\cdot\mathbf{x}_i) + \|\mathbf{x}_i\|^2 \\ &= (\text{margin of } i \text{ at } t) + (\text{positive}),\end{aligned}$$

which means that the margin at $t+1$ is a strict improvement over the margin at time t. The margin at t was negative (because point i was misclassified), so it would be good if this improvement made the margin positive. If it did not, then perhaps the margin will become positive at the next time we see the same point.

Perceptron Convergence Bound

We will show that the Perceptron algorithm converges to a separable solution at a fast rate. The convergence rate depends on "how separable" the dataset is, which we now define.

Assume $\{\mathbf{x}_i, y_i\}$ are separable by linear functions, so that there exists a separator $\mathbf{w}$ such that $\|\mathbf{w}\|_2 = 1$ and $y_i(\mathbf{w}\cdot\mathbf{x}_i) \geq \delta$ for all i. (That is, $\mathbf{w}$ is a "good" separator.)

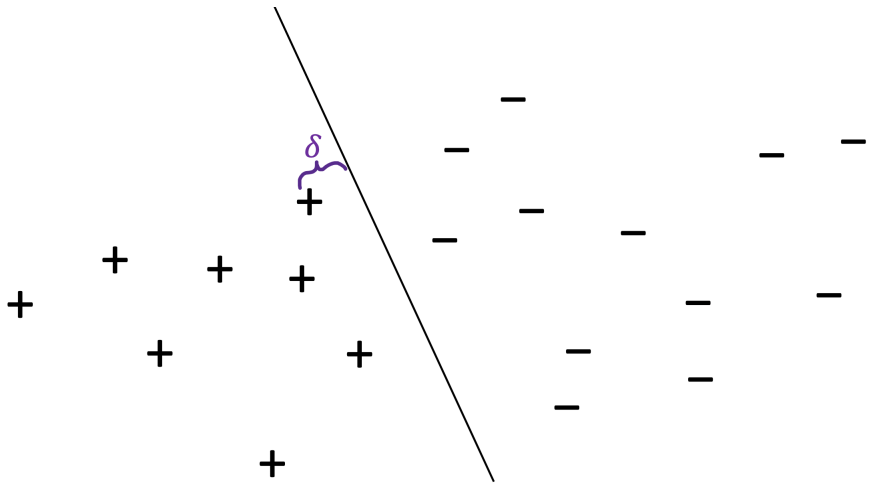

Figure 2: A good separator. We assume such a separator exists for this proof.

Assume the data are normalized, so $\max_i \|\mathbf{x}_i\|_2 = 1$ (if the data are not normalized, you can normalize them before running Perceptron).

Theorem. *The Perceptron algorithm makes at most* $\frac{1}{\delta^2}$ *mistakes.*

Proof. Define $\mathbf{w}^*$ to be a "good separator," so that $y_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq \delta$ for all i and $\|\mathbf{w}^*\|_2 = 1$.

An important note on indexing in what follows is that we only really want to count iterations for which a mistake is made. So t will be an index just on iterations for which there is a mistake. (If at any point, Perceptron encounters a point that is already correctly classified, nothing happens anyway, so this is in some sense equivalent to assuming that at each iteration a mistake is made.)

After T mistakes are made, we will show that $\mathbf{w}^{(T+1)}$ is "close to" $\mathbf{w}^*$. In particular, we will show that:

$$\delta\sqrt{T} \overset{①}{\leq} \frac{\mathbf{w}^* \cdot \mathbf{w}_{T+1}}{\|\mathbf{w}^*\|_2 \|\mathbf{w}_{T+1}\|_2} \overset{②}{\leq} 1,$$

where we need to show ①, and ② is true because the dot product between two normalized vectors (the cosine) is always ≤ 1. Once we have shown ①, the rest is easy, because we will have $\delta\sqrt{T} \leq 1$, thus $\sqrt{T} \leq 1/\delta$ and thus $T \leq 1/\delta^2$. This means that we can never have an iteration T more than $1/\delta^2$. So, there can be no more than $1/\delta^2$ mistakes, which is what we wanted to prove. So let's do it! We just need to show ①.

Step 1: Let $\mathbf{w}_1 = \mathbf{0}$ be the initialization. Let i be the point on which the t-th mistake of the Perceptron algorithm is made. We have:

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{w}_{t+1} - \mathbf{w}^* \cdot \mathbf{w}_t &= \mathbf{w}^* \cdot (\mathbf{w}_{t+1} - \mathbf{w}_t) \\ &= \mathbf{w}^* \cdot y_i \mathbf{x}_i \quad \text{from the update rule (1)} \\ &\geq \delta \quad \text{from the assumption that } \mathbf{w}^* \text{ is a good separator.} \end{aligned}$$

Now let us consider iteration $T+1$. Then we have

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{w}_{T+1} &= (\mathbf{w}^* \cdot \mathbf{w}_{T+1}) - (\mathbf{w}^* \cdot \mathbf{w}_1) \quad (\text{since } \mathbf{w}_1 = \mathbf{0}) \\ &= \sum_{t=1}^{T} [(\mathbf{w}^* \cdot \mathbf{w}_{t+1}) - (\mathbf{w}^* \cdot \mathbf{w}_t)] \geq T\delta \quad (\text{from the step just above}). \end{aligned}$$

Step 2: For iteration t, let us say that point i_t is chosen. Then,

$$\begin{aligned} \|\mathbf{w}_{t+1}\|_2^2 &= \|\mathbf{w}_t + y_{i_t}\mathbf{x}_{i_t}\|_2^2 \\ &= \|\mathbf{w}_t\|_2^2 + 2y_{i_t}(\mathbf{w}_t \cdot \mathbf{x}_{i_t}) + y_{i_t}^2\|\mathbf{x}_{i_t}\|_2^2 \\ &\leq \|\mathbf{w}_t\|_2^2 + 1, \end{aligned}$$

where the last inequality uses that point i_t was misclassified at iteration t, so margin $y_{i_t}(\mathbf{w}_t \cdot \mathbf{x}_{i_t}) \leq 0$. The last inequality also uses that $y_{i_t}^2 = 1$, and also $\|\mathbf{x}_{i_t}\|_2^2 \leq 1$ because of the normalization we did to the data. Thus, we have shown that

$$\|\mathbf{w}_{t+1}\| \leq \|\mathbf{w}_t\|_2^2 + 1.$$

Summing this from 1 to T gives

$$\|\mathbf{w}_{T+1}\|_2^2 \leq T, \quad \text{or} \quad \|\mathbf{w}_{T+1}\|_2 \leq \sqrt{T}.$$

Step 3: We combine Steps 1 and 2 to get

$$\frac{\mathbf{w}^* \cdot \mathbf{w}_{T+1}}{\|\mathbf{w}^*\|_2\|\mathbf{w}_{T+1}\|_2} \geq \frac{T\delta}{\sqrt{T}} = \sqrt{T}\delta,$$

which is equation ① that we were trying to prove. As we showed at the beginning of the proof, combining this with the fact that the left side is ≤ 1, this means

$$T \leq \frac{1}{\delta^2}.$$

We are done with the proof.

So, we have shown that when the data are seen one at a time, and there is a separable classifier with margin δ, Perceptron cannot make too many mistakes before it finds a separating hyperplane. There's a few things that are important in practice though. The order in which the points are viewed determines how fast the Perceptron can get to a separating hyperplane – if the Perceptron sees the points closest to the final decision boundary first, this may make it converge more quickly. Also, the bound does not consider the number of correctly classified points it will see before it finds that hyperplane. If the Perceptron sees quite a lot of correctly classified points before it sees the few points it needs to finalize its decision boundary, it can take a longer time in practice. The theory above just provides a bound on the total number of mistakes Perceptron will make before it finds a separating hyperplane, so it doesn't consider these extra practical aspects of convergence. It's a very nice guarantee though!

2 Winnow

Winnow (Littlestone and Warmuth, 1994) also learns a linear classifier. The difference is that Winnow uses an exponential weight update. Start with dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Again we will be learning linear weights $\mathbf{w}$ so that we will classify $\mathbf{x}$ according to the sign of $\mathbf{w} \cdot \mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^p$.

The high level idea of Winnow is that we will reward feature j's weight w_j if feature j would correctly classify data point i if it is used on its own as a classifier. In other words, if for point $\mathbf{x}_i$, we find that x_{ij} agrees with the label y_i, then we will increase w_j. The weights are renormalized after each update, so that they are a discrete probability distribution, with $w_j \geq 0$ for all j, and $\sum_i w_j = 1$.

Specifically, Winnow executes the following procedure: Let $\mathbf{w}_t$ be the weight vector at t:

- Input: learning rate η.
- Initialize $\mathbf{w}_1 \leftarrow \left[\frac{1}{p}, \frac{1}{p}, \cdots, \frac{1}{p}\right]$.
- For $t = 1, 2, \cdots$, do
  - Try to locate a misclassified data point $(\mathbf{x}_i, y_i)$, i.e., $y_i \mathbf{w} \cdot \mathbf{x}_i < 0$.
  - If no such data point exists, terminate and output $\mathbf{w}_t$.
  - Otherwise, for all $j = 1, 2, \cdots, p$,

$$w_{t+1,j} = w_{tj} \frac{\exp\left(\eta y_i x_{ij}\right)}{Z_t} \text{ where } Z_t := \sum_{j'=1}^{p} w_{tj'} \exp\left(\eta y_i x_{ij'}\right).$$

Thus, for features j where y_i agrees with x_{ij}, the weight on feature j increases. And otherwise, the weight decreases. That is,
If $\text{sign}(x_{ij}) = y_i$ then $w_{t+1,j} \propto w_{tj} e^{\eta \cdot (+)\text{ve}}$ (reward features that agree with label)
If $\text{sign}(x_{ij}) \neq y_i$ then $w_{t+1,j} \propto w_{tj} e^{\eta \cdot (-)\text{ve}}$ (punish features that disagree with label).

Winnow Convergence Bound

Assume we normalized the data, this time so that $\max_i \|\mathbf{x}_i\|_\infty = 1$.

Similar to the case for Perceptron, we assume that there is a separating hyperplane $\mathbf{w}^*$, such that $y_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq \delta \ \ \forall i$, and $w_j^* \geq 0 \ \ \forall j$, and $\|\mathbf{w}^*\|_1 = 1$.

Theorem. *The Winnow algorithm makes at most*

$$T \leq \frac{\log p}{\eta\delta + \log \frac{2}{e^\eta + e^{-\eta}}} \ \textit{mistakes.}$$

In particular, if we choose

$$\eta = \frac{1}{2} \log \frac{1+\delta}{1-\delta},$$

(which is the choice that minimizes the bound), then Winnow converges in $\frac{2\log p}{\delta^2}$ steps.

Proof. The proof idea again is to show that $\mathbf{w}_t$ gets closer to $\mathbf{w}^*$ at each iteration in terms of KL divergence. In case you are not familiar with KL divergence, it is a distance measure between probability distributions. Here, since the $\mathbf{w}_t$ are normalized and nonnegative, they act as a discrete probability distribution. KL divergence between distributions $\mathbf{a}$ and $\mathbf{b}$ is:

$$\mathrm{KL}(\mathbf{a}\|\mathbf{b}) = \sum_j a_j \log\left(\frac{a_j}{b_j}\right).$$

Why is it a "distance"? If $\mathbf{a} = \mathbf{b}$, then $\log\left(\frac{a_j}{b_j}\right) = 0$ so $\mathrm{KL}(\mathbf{a}\|\mathbf{b}) = 0$. It turns out that $\mathrm{KL}(\mathbf{a}\|\mathbf{b}) \geq 0$. (However, it is not symmetric, which is interesting.)

Step 1: Let $\mathbf{w}^*$ be our separating hyperplane where $\|\mathbf{w}^*\|_1 = 1$ and $w_j^* \geq 0$. Define

$$\Phi_t := \mathrm{KL}\left(\mathbf{w}^*\|\mathbf{w}_t\right) = \sum_j w_j^* \log\left(\frac{w_j^*}{w_{tj}}\right).$$

We have

$$\begin{aligned}
\Phi_t - \Phi_{t+1} &= \sum_{j=1}^{p} w_j^* \log \frac{w_j^*}{w_{tj}} - \sum_{j=1}^{p} w_j^* \log \frac{w_j^*}{w_{t+1,j}} \\
&= \sum_{j=1}^{p} w_j^* \log \frac{w_{t+1,j}}{w_{tj}} \\
&= \sum_{j=1}^{p} w_j^* \log \frac{\exp\left(\eta y_i x_{ij}\right)}{Z_t} \quad \text{(Winnow update rule)} \\
&= \sum_{j=1}^{p} w_j^* \eta y_i x_{ij} - \sum_{j=1}^{p} w_j^* \log Z_t \\
&= \sum_{j=1}^{p} w_j^* \eta y_i x_{ij} - \log Z_t \quad \text{(because } \sum_j w_j^* = 1\text{)} \\
&= \eta y_i \mathbf{w}^* \cdot \mathbf{x}_i - \log Z_t \\
&\geq \eta\delta - \log Z_t \quad \text{(margins of } \mathbf{w}^* \text{ are at least } \delta\text{)}.
\end{aligned}$$

Summing from $t = 1$ to T gives

$$\Phi_1 - \Phi_{T+1} \geq T\eta\delta - T \log Z_t,$$

which, since $\Phi_{T+1} \geq 0$ because it is a KL divergence, gives

$$\begin{aligned}
T\eta\delta - T \log Z_t \leq \Phi_1 - \Phi_{T+1} &\leq \Phi_1 \\
&= \mathrm{KL}\left(\mathbf{w}^* \| \mathbf{w}_1\right) = \sum_{j=1}^{p} w_j^* \log \frac{w_j^*}{w_{1j}} \\
&= \sum_{j=1}^{p} w_j^* \log(p w_j^*) \quad \text{(because } w_{1j} = 1/p\text{)} \\
&\leq \sum_{j=1}^{p} w_j^* \log p \quad \text{(because } w_j^* \leq 1\text{)} \\
&\leq \log p \quad \text{(because } \sum_j w_j^* = 1\text{)}.
\end{aligned}$$

Rearranging terms gives

$$T \leq \frac{\log p}{\eta\delta - \log Z_t}.$$

Step 2: Next we bound Z_t.

By convexity of the function e^{-x}, and the fact that $\frac{1+A}{2} + \frac{1-A}{2} = 1$ so the length-2 vector $[\frac{1+A}{2}, \frac{1-A}{2}]$ is a probability distribution for $A \in [-1, 1]$, we can use Jensen's inequality in the form $e^{\mathbb{E}z} \leq \mathbb{E}(e^z)$. (This is the same thing as using an upper bound for the exponential by a line between the points -1 and 1.) We have, for any $A \in [-1, 1]$:

$$\begin{aligned}\exp(\eta A) &= \exp\left(\eta\left(\frac{1+A}{2}\right) + (-\eta)\left(\frac{1-A}{2}\right)\right) \\ &\leq \left(\frac{1+A}{2}\right)e^{\eta} + \left(\frac{1-A}{2}\right)e^{-\eta}.\end{aligned}$$

We will use this bound by letting $A = y_i x_{ij}$. (Its value is between -1 and 1 because $\|\mathbf{x}_i\|_\infty \leq 1$.)

Using the definition of Z_t,

$$\begin{aligned}Z_t &= \sum_{j=1}^{p} w_{tj} \exp(\eta y_i x_{ij}) \\ &\leq \sum_{j=1}^{p} w_{tj}\left(\left(\frac{1+y_i x_{ij}}{2}\right)e^{\eta} + \left(\frac{1-y_i x_{ij}}{2}\right)e^{-\eta}\right) \\ &= \sum_{j=1}^{p} w_{tj}\left(\frac{1}{2}e^{\eta} + \frac{y_i x_{ij}}{2}e^{\eta} + \frac{1}{2}e^{-\eta} - \frac{y_i x_{ij}}{2}e^{-\eta}\right) \\ &= \frac{e^{\eta}+e^{-\eta}}{2}\sum_{j=1}^{p} w_{tj} + \frac{e^{\eta}-e^{-\eta}}{2}\sum_{j=1}^{p} w_{tj} y_i x_{ij} \\ &= \frac{e^{\eta}+e^{-\eta}}{2} + \frac{e^{\eta}-e^{-\eta}}{2} y_i \mathbf{w}_t \cdot \mathbf{x}_i \\ &\leq \frac{e^{\eta}+e^{-\eta}}{2},\end{aligned}$$

where the last line uses that $(\mathbf{x}_i, y_i)$ is misclassified at time t: $y_i \mathbf{w}_t \cdot \mathbf{x}_i \leq 0$, and that $\frac{e^{\eta}-e^{-\eta}}{2}$ is positive for positive η.

Step 3: Plugging the above bound on Z_t from Step 2 back into Step 1 gives:

$$T \leq \frac{\log p}{\eta\delta + \log \frac{2}{e^{\eta}+e^{-\eta}}}, \tag{2}$$

which is the statement of the theorem.

In order to get the best possible bound (the lowest value of the bound), we would choose η to minimize the bound. To do this, we would set the derivative of the bound to 0 with respect to η and set it to 0. We would get the η from the theorem, namely: $\eta = \frac{1}{2}\log\frac{1+\delta}{1-\delta}$. Plugging this into the theorem's bound, we would get

$$T \leq \frac{2\log p}{\delta^2}.$$

Showing this requires some work simplifying. In particular, we need to simplify $\eta\delta + \log\frac{2}{e^{\eta}+e^{-\eta}}$. First, simplifying the denominator in the log, we get $e^{\eta} + e^{-\eta} = \left(\frac{1+\delta}{1-\delta}\right)^{1/2} + \left(\frac{1-\delta}{1+\delta}\right)^{1/2}$ and putting both fractions over a common denominator, this term becomes $\frac{2}{[(1+\delta)(1-\delta)]^{1/2}}$. With this, $\eta\delta + \log\frac{2}{e^{\eta}+e^{-\eta}}$ becomes $\delta\frac{1}{2}\log\frac{1+\delta}{1-\delta} + \log\frac{2}{2/[(1+\delta)(1-\delta)]^{1/2}}$ which simplifies to $\log\left(\frac{1+\delta}{1-\delta}\right)^{\delta/2} + \log[(1+\delta)(1-\delta)]^{1/2}$, which simplifies to $\frac{1}{2}\log[(1+\delta)^{(1+\delta)}(1-\delta)^{(1-\delta)}]$. As it turns out, this quantity is $\geq \frac{1}{2}\delta^2$. Thus, the right side of (2) $\leq \frac{\log p}{\frac{1}{2}\delta^2}$.
We are done with the proof.

After this number of iterations, we will not be able to find any points that are misclassified, and there will be no more updates, as we have proven.

Moment of Truth

In fact, Perceptron and Winnow represent two different types of machine learning algorithmic paradigms. Let us compare them:

	Perceptron	Winnow	
Bound	$T \leq 1/\delta^2$	$T \leq \frac{2 \log p}{\delta^2}$	($\log p$ isn't very important)
updates	additive	multiplicative	
	$\mathbf{w}_{t+1} = \mathbf{w}_t + ...$	$\mathbf{w}_{t+1} = \mathbf{w}_t \times ...$	
normalization	$\|\mathbf{x}_i\|_2 \leq 1 \ \forall i$	$\|\mathbf{x}_i\|_\infty \leq 1 \ \ \forall i$	
reference normaliz.	$\|\mathbf{w}^*\|_2 = 1$	$\|\mathbf{w}^*\|_1 \leq 1$	
Similar algorithms	SVM	AdaBoost	

The $\log p$ term isn't very important because log grows very slowly in p, whereas $1/\delta^2$ grows quickly as δ gets smaller.

The last line in the table is really a revelation: while the online learning algorithms Perceptron and Winnow are so similar as algorithms – and even their convergence proofs and rates are so similar – they actually represent two different mainstreams of machine learning thought: the SVM-like thought-process and the boosting thought-process. Everything involving SVM is ℓ_2. This is why SVM analysis can use kernels and Hilbert spaces. For the exponential weights algorithms (Winnow and AdaBoost), they use multiplicative weights and renormalization at each iteration. Their normalization uses the ℓ_1 norm for the $\mathbf{w}$ vectors and the dual norm, ℓ_∞, for the normalization of the $\mathbf{x}_i$'s. The margins of SVM are ℓ_2 margins, while margins in boosting are ℓ_1 margins. Perceptron and Winnow, as we have given them, stop once the data are separable, so they do not consider margins, but there are variations of them that do consider margins. So, in many ways, we can think of additive algorithms as in the same family as SVM, and multiplicative weights algorithms as in the same family as AdaBoost.

Acknowledgements

Thank you to scribe Tianyu Wang.

References

Block, H.-D. (1962). The perceptron: A model for brain functioning. i. Reviews of Modern Physics, 34(1):123.

Littlestone, N. and Warmuth, M. K. (1994). The weighted majority algorithm. Information and computation, 108(2):212–261.

Novikoff, A. B. (1964). On convergence proofs for perceptrons. Technical report.