

## Online learning: Perceptron and Winnow

In the online learning setting we are presented with a (potentially infinite) *sequence* of examples  $x_1, x_2, x_3 \dots$ . At each time step  $n$ , we have to make a prediction  $\hat{y}_n$  for example  $x_n$ . After making a prediction, the world tells us if we were right or wrong. Formally, we suffer *loss*  $\ell_n$ . We then continue and observe  $x_{n+1}$ .

The goal in online learning is to continually learn so as to minimize  $\sum_n \ell_n$ .

The two algorithms we discuss are *error-correction* based algorithms. Namely, they maintain a current classifier  $h$  and use this to make predictions. At each step, if  $h$  is right, they do nothing. If  $h$  is wrong, the update it so that it is “less wrong.”

Both perceptron and winnow are also linear classifiers. Therefore, they can be represented as having a weight vector  $\mathbf{w}$  and a bias  $b$  and make predictions on  $\mathbf{x}$  of the form  $\text{sign}[\mathbf{w}^\top \mathbf{x} + b]$ .

The perceptron uses simple additive updates:

1. Initialize  $\mathbf{w} = 0, b = 0$
2. For  $n = 1 \dots$ 
  - (a) Obtain  $\mathbf{x}_n$
  - (b) Predict  $\hat{y}_n = \text{sign}[\mathbf{w}^\top \mathbf{x}_n + b]$
  - (c) Suffer loss  $\ell_n$  (zero if  $\hat{y}_n = y_n$ , one otherwise)
  - (d) If  $\ell_n = 0$ , continue on as before. Otherwise, update:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \hat{y}_n \mathbf{x}_n \\ b &\leftarrow b - \hat{y}_n \end{aligned} \tag{1}$$

The idea with the algorithm is that with binary classification, if  $\hat{y}_n$  was wrong, then  $-\hat{y}_n$  is *right*. What we do in update (1) is to move  $(\mathbf{w}, b)$  closer to making the right prediction. Why? Let  $(\mathbf{w}', b')$  denote the updated classifier. Let’s look at what we *would* predict, if we received the same  $\mathbf{x}$  again. Suppose that the truth is  $y$  (so  $y = -\hat{y}$ ); we know we want  $y[\mathbf{w}'^\top \mathbf{x} + b'] > 0$ .

$$\begin{aligned} y[\mathbf{w}'^\top \mathbf{x} + b'] &= y(\mathbf{w} - \hat{y}\mathbf{x})^\top \mathbf{x} + y(b - \hat{y}) \\ &= y[\mathbf{w}^\top \mathbf{x} + b] - y(\hat{y}\mathbf{x}^\top \mathbf{x} + \hat{y}) \\ &= y[\text{old prediction}] - y\hat{y} \|\mathbf{x}\|^2 - y\hat{y} \\ &= y[\text{old prediction}] + \|\mathbf{x}\|^2 + 1 \end{aligned}$$

In the last step, we used the fact that  $y\hat{y} = -1$  if we made an error. So what’s happening is we have our old prediction times  $y$  (which we know was negative—since we erred) plus  $\|\mathbf{x}\|^2 + 1$ , which is always positive. Which means that the whole thing is getting more positive. Which is exactly what we want.

This leads directly to the following theorem for *batch* applications:

**Theorem (Block & Novikoff):** if the training data is linearly separable with margin  $\gamma$  with weights  $\mathbf{u}$ , then Perceptron will terminate in  $\|\mathbf{u}\|^2 \leq R^2/\gamma^2$  iterations (assuming  $\|\mathbf{x}\| \leq R$  for all  $\mathbf{x}$ ).

Proof sketch: (Suppose  $R = 1$ .) Let  $\mathbf{w}_k$  be the weights before the  $k$ th update, so  $\mathbf{w}_1 = 0$  and if the  $k$ th error is on example  $n$ , then  $y_n(\mathbf{w}_k^\top \mathbf{x}_n) \leq 0$ . Then,  $\mathbf{w}_{k+1}^\top \mathbf{u} = \mathbf{w}_k^\top \mathbf{u} + y_n(\mathbf{u}^\top \mathbf{x}_n) \geq \mathbf{w}_k^\top \mathbf{u} + \gamma$ , so

$\mathbf{w}_{k+1}^\top \mathbf{u} \geq k\gamma$ . Also,  $\|\mathbf{w}_{k+1}\|^2 = \|\mathbf{w}_k\|^2 + 2y_n(\mathbf{w}_k^\top \mathbf{x}_n) + \|\mathbf{x}_n\|^2 \leq \|\mathbf{w}_k\|^2 + 1$  so  $\|\mathbf{w}_{k+1}\|^2 \leq k$ . Thus  $\sqrt{k} \geq \|\mathbf{w}_{k+1}\| \geq \mathbf{w}_{k+1}^\top \mathbf{u} \geq k\gamma$ ; then algebra.

For the inseparable case, just force it to be separable (add a feature).

*Voting*: keep a copy of each value of  $\mathbf{w}$  and make predictions by voting ( $\text{sign}[\sum_i \text{sign}[\mathbf{w}_i^\top \mathbf{x}]]$ ).

*Averaging*: use averaged weights to make prediction ( $\text{sign}[(\sum_i \mathbf{w}_i)^\top \mathbf{x}]$ ).

Connection to SVM: SVMs are *directly* optimizing the perceptron bound!

Winnow works similarly, but always maintains positive weights and updates them *multiplicatively*. In fact, the entire algorithm is the same, except the initialization and update:

1. Initialize  $\mathbf{w} = \mathbf{w}^{(0)}$ ,  $b = b^{(0)}$
2. For  $n = 1 \dots$ 
  - (a) Obtain  $\mathbf{x}_n$
  - (b) Predict  $\hat{y}_n = \text{sign}[\mathbf{w}^\top \mathbf{x}_n + b]$
  - (c) Suffer loss  $\ell_n$  (zero if  $\hat{y}_n = y_n$ , one otherwise)
  - (d) If  $\ell_n = 0$ , continue on as before. Otherwise, update:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} \otimes \exp[-\eta \hat{y}_n \mathbf{x}_n] \\ b &\leftarrow b \exp[-\eta \hat{y}_n] \end{aligned} \tag{2}$$

In Eq (2),  $\otimes$  denotes coordinate-wise multiplication, and the exponentiation is coordinate-wise (i.e., it is not matrix exponentiation).  $\eta > 0$  is a “step size” parameter.

Also note that the initialization changed. We now initialize to some “prior” hyperplane  $(\mathbf{w}^{(0)}, b^{(0)})$ . Choosing both to be just ones is usually a reasonable starting point.

One problem with winnow is that it doesn’t allow negative weights. To get around this, we usually replace our inputs  $\mathbf{x}$  with  $[\mathbf{x}; -\mathbf{x}]$ , effectively doubling the dimensionality.

Like the perceptron, we can prove the following in a batch setting:

**Theorem (Zhang)**: if the training data is linearly separable with margin  $\gamma$  with weights  $\mathbf{u}$ , then Winnow will terminate in  $\leq M$  iterations, where  $0 < \delta < \gamma$ ,  $W \geq \|\mathbf{u}\|_1$  and  $\eta = \delta/(WR^2)$  (assuming  $\|x\|_\infty \leq R$  for all  $x$ ). Here,  $M$  is:

$$M = 2W \left[ \sum_d w_d \log \frac{u_d \|\mathbf{w}^{(0)}\|_1}{w_d^{(0)} \|\mathbf{u}\|_1} \right] \frac{R^2}{\delta^2}$$

Note that everything is now in terms of the  $\ell_1$  norm (sum of absolute values) and the  $\ell_\infty$  norm (maximum absolute value). Importantly, the form is still roughly  $R^2/\gamma^2$ , but now  $R$  is measured by  $\ell_\infty$ .

The big advantage to winnow over perceptron is that it gets rid of useless features *much* better. The disadvantage is the requirement of the learning rate and that the bound depends on this.

You can formulate a “SVM” version of Winnow that looks just like an SVM except the regularizer has the form  $\sum_d w_j \log[w_d/(e w_d^{(0)})]$  instead of  $\|\mathbf{w}\|^2$  (here,  $e$  is the base of the natural logarithm).