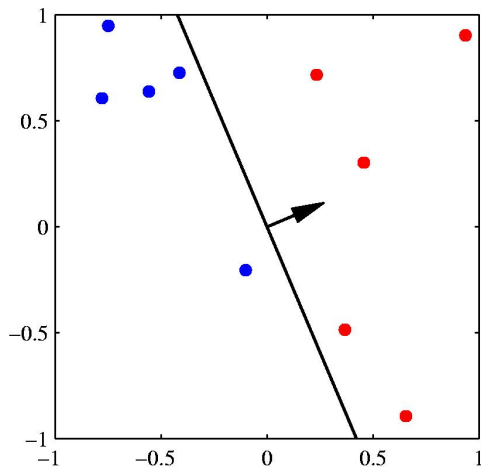


Linear classification

We've talked about regressing on \mathbb{R} ... what about $\{-1, +1\}$?

The whole idea behind linear classifiers is to directly represent the *decision* boundary (in binary classification problems). We begin with the simplest kind of decision boundary: a hyperplane (a flat thing). We put points of each class on either side of the boundary. The mostly-vertical line in the figure below is a possible decision boundary between the blue (left) and red (right) points.



As a *representation* of a decision boundary, we store the vector that is orthogonal (aka “normal”) to the decision boundary. We usually call this \mathbf{w} (for “weights”... this will become clear soon). For instance, if the x-axis is the first coordinate and the y-axis is the second coordinate, the weight vector in the figure above (the arrow) might have value $\langle 0.4, 0.1 \rangle$, indicating that it is pointing right a lot and up a little.

The representation in terms of \mathbf{w} is overkill: if \mathbf{w} represents the hyperplane, then $\alpha\mathbf{w}$ also does, for any $\alpha > 0$ (it just makes the vector longer, but doesn't change its direction). Thus, we'll usually assume that $\|\mathbf{w}\| = 1$ (the 1 is for convenience; any positive constant would be fine).

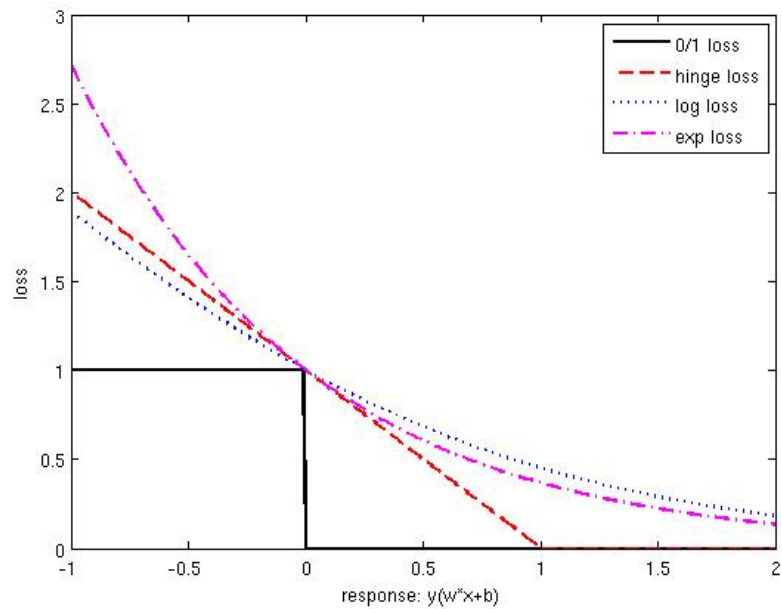
This representation also assumes that the decision boundary passes through the origin. We will shift the hyperplane by adding an additional “bias” term b . In the above figure, if we wished to move the decision boundary in the direction of the arrow, we would use $b < 0$; to move it in the opposite direction, we use $b > 0$.

Given a representation of the decision boundary in terms of \mathbf{w} and b , we make our classification according to $\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$, where \mathbf{x} is an input and \hat{y} is the predicted class (note that we refer to our two classes as $+1$ and -1 instead of 0 and 1).

We can think about the hyperplane as being defined as the set $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 0\}$... in other words, it's the set of *all* points for which our classifier is totally uncertain.

In general, if there exists a hyperplane that perfectly separates our data onto two sides, we refer to the data as *linearly separable*. Except in very peculiar circumstances, if data is linearly separable, then there are infinitely many separating hyperplanes (just jiggle the one in the figure a little bit any direction).

It's hard to optimize zero/one loss (aka classification error), so we minimize a convex upper bound:



The upper bounds are:

- hinge loss: $[1 - v]_+ = \max\{0, 1 - v\}$
- log loss: $\log(1 + \exp(-v))$ (rescaled in the image by $1/\log 2$)
- exponential loss: $\exp(-v)$

Log loss is what is optimized in logistic regression. We start out with a probabilistic model fit by a sigmoid:

Sigmoid function:

$$\begin{aligned}\sigma(x) &= \frac{\exp[x/2]}{\exp[x/2] + \exp[-x/2]} \\ &= \frac{1}{1 + \exp[-x]}\end{aligned}$$

Inverse is the logistic:

$$\sigma^{-1}(y) = \log\left(\frac{y}{1-y}\right)$$

Specify a probability distribution on y conditioned on x :

$$p_{\theta}(y | x) = \sigma(y\theta^{\top} x) = \frac{1}{1 + \exp[-y\theta^{\top} x]}$$

This is the *logistic regression* model.

No closed-form solution exists; compute log likelihood:

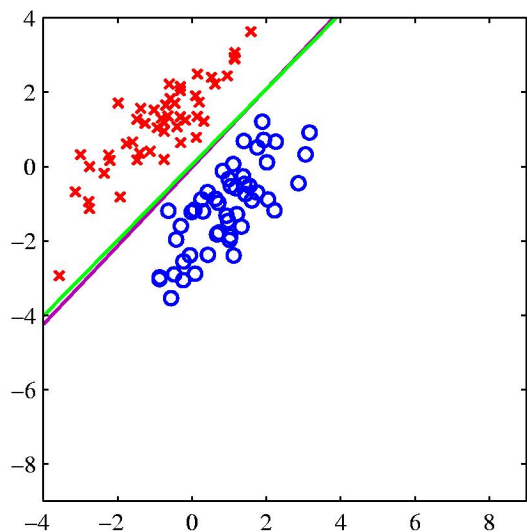
$$\begin{aligned}\ell(\theta) &= \log \prod_n p_\theta(y_n | x_n) \\ &= \sum_n \log \frac{1}{1 + \exp[-y_n \theta^\top x_n]} \\ &= - \sum_n \log (1 + \exp[-y_n \theta^\top x_n])\end{aligned}$$

Now, compute the gradient.

$$\begin{aligned}\nabla_\theta \ell &= - \sum_n \frac{1}{1 + \exp[-y_n \theta^\top x_n]} \exp[-y_n \theta^\top x_n] (-y_n x_n) \\ &= \sum_n \frac{1}{1 + \exp[y_n \theta^\top x_n]} y_n x_n\end{aligned}$$

And we can do gradient descent.

This is a *linear model* because the output (± 1) is a (thresholded) linear function of the input. For linear functions, we can draw the decision boundary as a linear function.



Find set of x for which $\theta^\top x = 0$ and this is the decision boundary, which will be linear.

Similar approaches can be followed to optimize exponential loss. Hinge loss is a bit tricky because it's not differentiable at zero. To solve this problem, we use the method of *subgradients*¹. The idea is that whenever we don't have a "real" gradient, we can choose anything we want that still "kind of" looks like a gradient. And it still works (i.e., will find a minimum). We will just choose a gradient of zero at the point of non-differentiability.

¹See http://en.wikipedia.org/wiki/Subgradient_method for details.