3 Perceptron Learning; Maximum Margin Classifiers

Perceptron Algorithm (cont'd)

Recall:

- linear decision fn $f(x) = w \cdot x$ (for simplicity, no α)
- decision boundary $\{x : f(x) = 0\}$ (a hyperplane through the origin)
- sample points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$; classifications $y_1, \ldots, y_n = \pm 1$
- goal: find weights w such that $y_i X_i \cdot w \geq 0$
- goal, rewritten: find w that minimizes $R(w) = \sum_{i \in V} -y_i X_i \cdot w$ [risk function]
  where V is the set of indices i for which $y_i X_i \cdot w < 0$.

[Our original problem was to find a separating hyperplane in one space, which I'll call x-space. But we've transformed this into a problem of finding an optimal point in a different space, which I'll call w-space. It's important to understand transformations like this, where a geometric structure in one space becomes a point in another space.]

Objects in x-space transform to objects in w-space:

x-space		w-space	
hyperplane:	$\{z : w \cdot z = 0\}$	point:	w
point:	x	hyperplane:	$\{z : x \cdot z = 0\}$

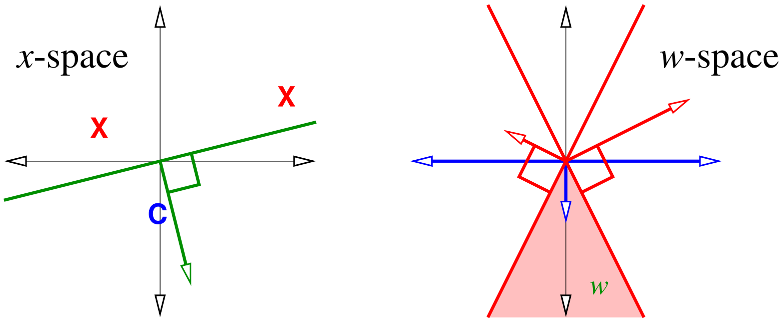
Point x lies on hyperplane $\{z : w \cdot z = 0\} \Leftrightarrow w \cdot x = 0 \Leftrightarrow$ point w lies on hyperplane $\{z : x \cdot z = 0\}$ in w-space.

[So a hyperplane transforms to its normal vector. And a sample point transforms to the hyperplane whose normal vector is the sample point.]

[In this case, the transformations happen to be symmetric: a hyperplane in x-space transforms to a point in w-space the same way that a hyperplane in w-space transforms to a point in x-space. That won't always be true for the weight spaces we use this semester.]

If we want to enforce inequality $x \cdot w \geq 0$, that means

- in x-space, x should be on the same side of $\{z : z \cdot w = 0\}$ as w
- in w-space, w " " " " " " " $\{z : x \cdot z = 0\}$ as x

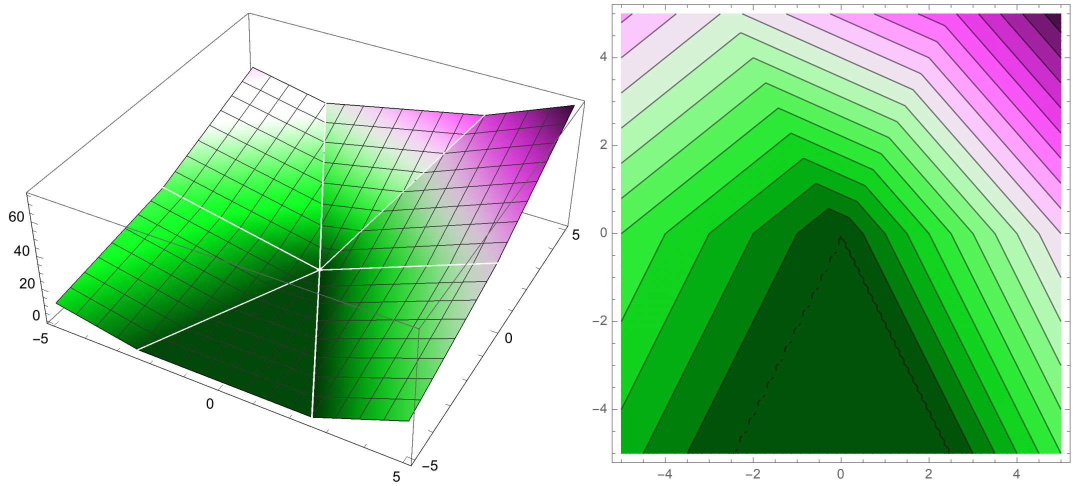


[Draw this by hand. xwspace.pdf]
[Observe that the x-space sample points are the normal vectors for the w-space lines. We can choose w to be anywhere in the shaded region.]

[For a sample point x in class C, w and x must be on the *same* side of the hyperplane that x transforms into. For a point x not in class C (marked by an X), w and x must be on *opposite* sides of the hyperplane that x transforms into. These rules determine the shaded region above, in which w must lie.]

[Again, what have we accomplished? We have switched from the problem of finding a hyperplane in x-space to the problem of finding a point in w-space. That's a much better fit to how we think about optimization algorithms.]

[Let's take a look at the risk function these three sample points create.]



riskplot.pdf, riskiso.pdf [Plot & isocontours of risk $R(w)$. Note how R's creases match the dual chart above.]

[In this plot, we can choose w to be any point in the bottom pizza slice; all those points minimize R.]

[We have an optimization problem; we need an optimization algorithm to solve it.]

An optimization algorithm: gradient descent on R.

Given a starting point w, find gradient of R with respect to w; this is the direction of steepest ascent. Take a step in the opposite direction. Recall [from your vector calculus class]

$$\nabla R(w) = \begin{bmatrix} \frac{\partial R}{\partial w_1} \\ \frac{\partial R}{\partial w_2} \\ \vdots \\ \frac{\partial R}{\partial w_d} \end{bmatrix} \qquad \text{and} \qquad \nabla(z \cdot w) = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix} = z$$

$$\nabla R(w) = \sum_{i \in V} \nabla - y_i X_i \cdot w = -\sum_{i \in V} y_i X_i$$

At any point w, we walk downhill in direction of steepest descent, $-\nabla R(w)$.

```
$w \leftarrow$ arbitrary nonzero starting point (good choice is any $y_i X_i$)
while $R(w) > 0$
    $V \leftarrow$ set of indices $i$ for which $y_i X_i \cdot w < 0$
    $w \leftarrow w + \epsilon \sum_{i \in V} y_i X_i$
return $w$
```

$\epsilon > 0$ is the step size aka learning rate, chosen empirically. [Best choice depends on input problem!]

[Show plot of R again. Draw the typical steps of gradient descent.]

Problem: Slow! Each step takes $O(nd)$ time. [Can we improve this?]

Optimization algorithm 2: stochastic gradient descent

Idea: each step, pick **one** misclassified X_i;
do gradient descent on loss fn $L(X_i \cdot w, y_i)$.

Called the perceptron algorithm. Each step takes $O(d)$ time.
[Not counting the time to search for a misclassified X_i.]

```
while some $y_i X_i \cdot w < 0$
    $w \leftarrow w + \epsilon\, y_i X_i$
return $w$
```

[Stochastic gradient descent is quite popular and we'll see it several times more this semester, especially for neural nets. However, stochastic gradient descent does not work for every problem that gradient descent works for. The perceptron risk function happens to have special properties that guarantee that stochastic gradient descent will always succeed.]

What if separating hyperplane doesn't pass through origin?
Add a fictitious dimension.

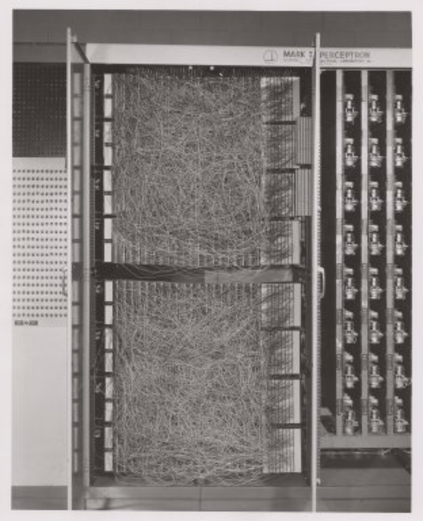
Hyperplane: $w \cdot x + \alpha = 0$

$$[w_1 \;\; w_2 \;\; \alpha] \cdot \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = 0$$

Now we have sample points in $\mathbb{R}^{d+1}$, all lying on plane $x_{d+1} = 1$.

Run perceptron algorithm in $(d + 1)$-dimensional space.

[The perceptron algorithm was invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory. It was originally designed not to be a program, but to be implemented in hardware for image recognition on a 20 × 20 pixel image. Rosenblatt built a Mark I Perceptron Machine that ran the algorithm, complete with electric motors to do weight updates.]



Mark_I_perceptron.jpg (from Wikipedia, "Perceptron") [The Mark I Perceptron Machine. This is what it took to process a 20 × 20 image in 1957.]

[Then he held a press conference where he predicted that perceptrons would be “the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.” We’re still waiting on that.]

[One interesting aspect of the perceptron algorithm is that it’s an “online algorithm,” which means that if new data points come in while the algorithm is already running, you can just throw them into the mix and keep looping.]

Perceptron Convergence Theorem: If data is linearly separable, perceptron algorithm will find a linear classifier that classifies all data correctly in at most $O(R^2/\gamma^2)$ iterations, where $R = \max |X_i|$ is “radius of data” and γ is the “maximum margin.”
[I’ll define “maximum margin” shortly.]

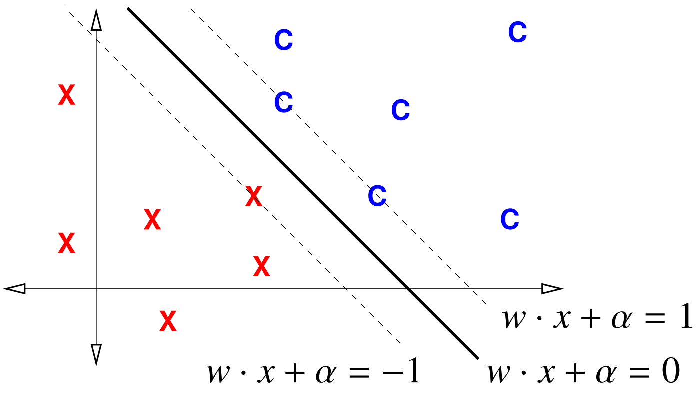
[We’re not going to prove this, because perceptrons are obsolete.]

[Although the step size/learning rate doesn’t appear in that big-O expression, it does have an effect on the running time, but the effect is hard to characterize. The algorithm gets slower if ϵ is too small because it has to take lots of steps to get down the hill. But it also gets slower if ϵ is too big for a different reason: it jumps right over the region with zero risk and oscillates back and forth for a long time.]

[Although stochastic gradient descent is faster for this problem than gradient descent, the perceptron algorithm is still slow. There’s no reliable way to choose a good step size ϵ. Fortunately, optimization algorithms have improved a lot since 1957. You can get rid of the step size by using any decent modern “line search” algorithm. Better yet, you can find a better decision boundary much more quickly by quadratic programming, which is what we’ll talk about next.]

MAXIMUM MARGIN CLASSIFIERS

The margin of a linear classifier is the distance from the decision boundary to the nearest sample point. What if we make the margin as wide as possible?



[Draw this by hand. maxmargin.pdf]

We enforce the constraints

$$y_i\,(w \cdot X_i + \alpha) \geq 1 \qquad \text{for } i \in [1, n]$$

[Notice that the right-hand side is a 1, rather than a 0 as it was for the perceptron algorithm. It’s not obvious, but this a better way to formulate the problem, partly because it makes it impossible for the weight vector w to get set to zero.]

Recall: if $|w| = 1$, signed distance from hyperplane to X_i is $w \cdot X_i + \alpha$.
Otherwise, it's $\frac{w}{|w|} \cdot X_i + \frac{\alpha}{|w|}$. [We've normalized the expression to get a unit weight vector.]

Hence the margin is $\min_i \frac{1}{|w|} |w \cdot X_i + \alpha| \geq \frac{1}{|w|}$. [We get the inequality by substituting the constraints.]

There is a slab of width $\frac{2}{|w|}$ containing no sample points [with the hyperplane running along its middle].

To maximize the margin, minimize $|w|$. Optimization problem:

> Find w and α that minimize $|w|^2$
> subject to $y_i(X_i \cdot w + \alpha) \geq 1$ for all $i \in [1, n]$

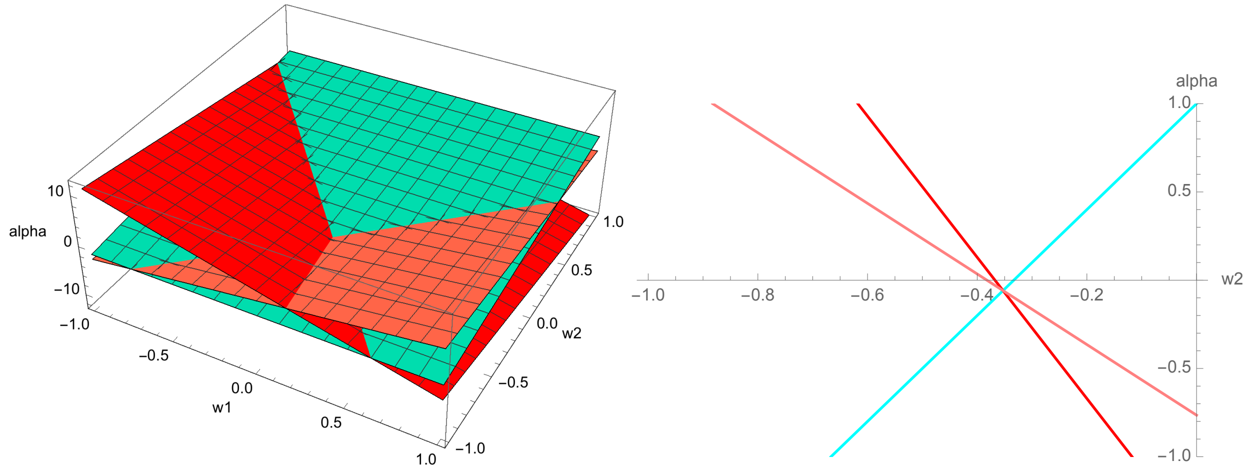
Called a quadratic program in $d + 1$ dimensions and n constraints.
It has one unique solution! [If the points are linearly separable; otherwise, it has no solution.]

[A reason we use $|w|^2$ as an objective function, instead of $|w|$, is that the length function $|w|$ is not smooth at zero, whereas $|w|^2$ is smooth everywhere. This makes optimization easier.]

The solution gives us a maximum margin classifier, aka a hard margin support vector machine (SVM).

[Technically, this isn't really a support vector machine yet; it doesn't fully deserve that name until we add features and kernels, which we'll do in later lectures.]

[Let's see what these constraints look like in weight space.]



weight3d.pdf, weightcross.pdf [This is an example of what the linear constraints look like in the 3D weight space (w_1, w_2, α) for an SVM with three training points. The SVM is looking for the point nearest the origin that lies above the blue plane (representing an in-class training point) but below the red and pink planes (representing out-of-class training points). In this example, that optimal point lies where the three planes intersect. At right we see a 2D cross-section $w_1 = 1/17$ of the 3D space, because the optimal solution lies in this cross-section. The constraints say that the solution must lie in the leftmost pizza slice, while being as close to the origin as possible, so the optimal solution is where the three lines meet.]