Lecture 2
Linear Models

Last lecture

- You have learned about what is machine learning
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning
- You have seen an example learning problem and the general process that one goes through to design a learning system, which involves determining:
  - Types of training experience
  - Target function
  - Representation of the learned function
  - Learning algorithm

Supervised learning

Let’s look at the problem of spam filtering

Now anyone can learn how to earn $200 - $943 per day or More ! If you can type (hunt and peck is ok to start) and fill in forms, you can score big! So don't delay waiting around for the next opportunity...it is knocking now! Start here: http://redbluecruise.com/t/c/381/polohoo/yz37957.html

Do you Have Poetry that you think should be worth $10,000.00 USD, we do!.. Enter our International Open contest and see if you have what it takes. To see details or to enter your own poem, Click link below. http://e-suscriber.com/imys?e=0sAoo4q9s4zYYUoYQ&m=795314&l=0

View my photos!
I invite you to view the following photo album(s): zak-month27

Hey have you seen my new pics yet???? Me and my girlfreind would love it if you would come chat with us for a bit.. Well join us if you interested. Join live web cam chat here: http://e-commcentral.com/imys?e=0sAoo4q9s4zYYUoYQ&m=825314&l=0

Let's look at the design choices

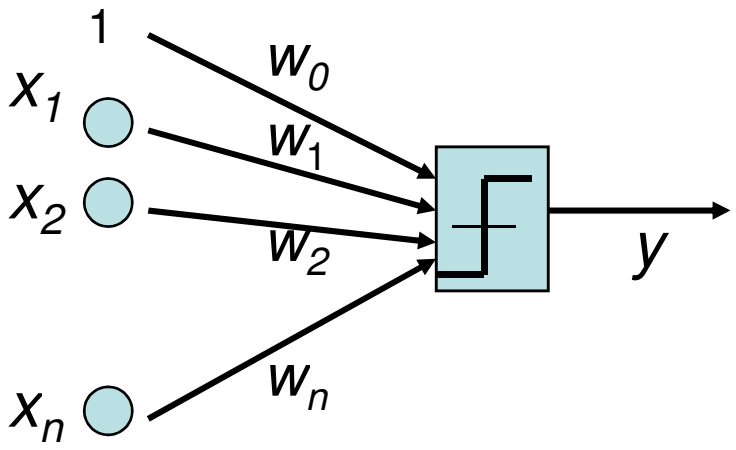
- Learning experience?
  - Past emails and whether they are considered spam or not (you can also choose to use non-spam or spam emails only, but that will require different choices later on)
- Target function?
  - Email -> spam or not
- Representation of the function?
  - ?
- Learning algorithm
  - ?

We will focus mostly on these two aspects in this class. In some cases, you will also need to pay attention to the first two questions.

Continue with the design choices

- Representation of the function (email -> spam or not) ?
- First of all, how to represent an email?
  - Use bag-of-words to represent an email
  - This will turn an email into a collection of features, e.g., where each feature describe whether a particular word is present in the email
- This gives us the standard supervised classification problem typically seen in text books and papers
  - **Training set: a set of examples (instances, objects) with class labels, e.g., positive (spam) and negative (non spam)**
  - **Input representation: an example is described by a set of** attributes (e.g., whether "$" is present, etc.)
  - **Given an unseen email, and its input representation, predict its label**
- Next question: what function forms to use?

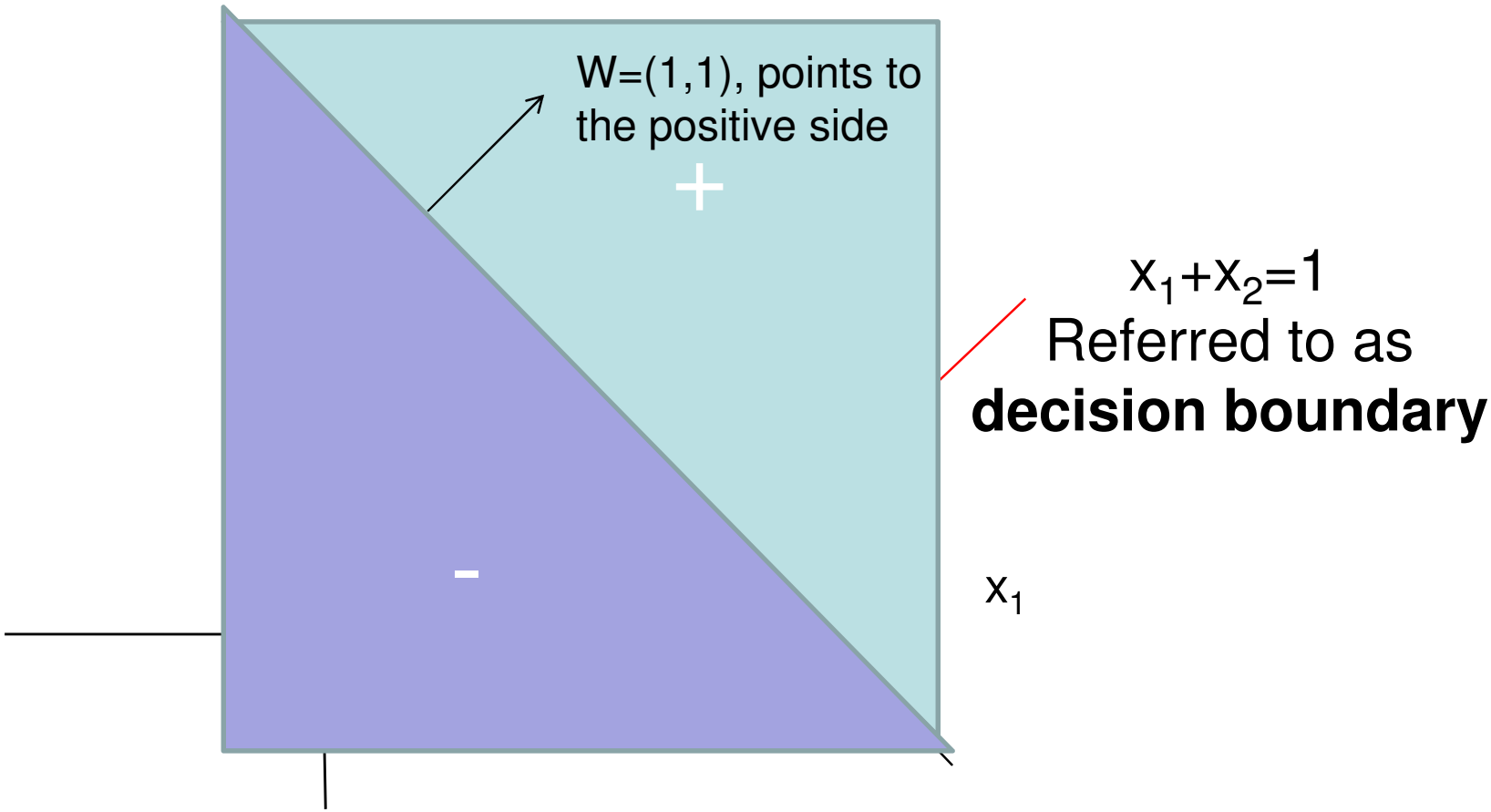
Linear Threshold Units (McCulloch & Pitts 1943)



$$y = \begin{cases} +1 & \text{if } w_0 + \sum_{i=1}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Assume each feature x_j and weight w_j is a real number
- LTU computes $\boldsymbol{w} \bullet \boldsymbol{x}$ and takes threshold it to produce the prediction y
- Why linear model?
  - Simplest model – fewer parameters to learn
  - Visually intuitive - drawing a straight line to separate positive from negative

Geometric view



A Canonical Representation

- Given a training example: ($<x_1, x_2, \ldots, x_m>$, y)
- transform it to ($<1, x_1, x_2, \ldots, x_m>$, y)
- The parameter vector will then be

  $\mathbf{w} = <w_0, w_1, w_2, \ldots, w_m>$

- Given a training set, we need to learn

  $g(\mathbf{x},\mathbf{w}) = w_0 1 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \boldsymbol{w} \cdot \boldsymbol{x}$

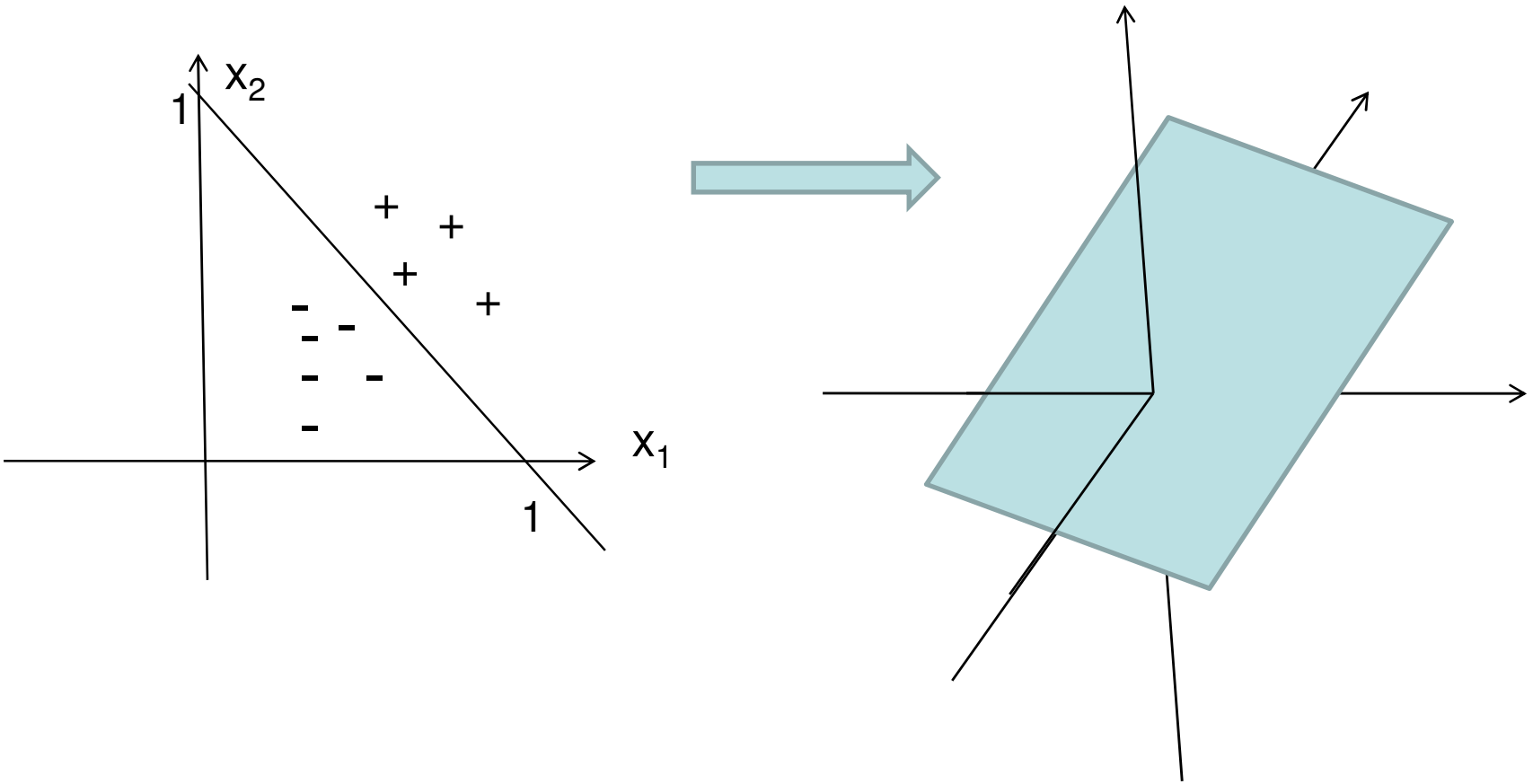
  Dot (or inner) product: takes two equal-length vectors, and returns the sum of their component-wise product

- Or equivalently $h(\mathbf{x},\boldsymbol{w}) = \text{sign}(g(\mathbf{x},\mathbf{w}))$
- To differentiate the learned function and the true underlying function, it is common to refer to the learned function as a hypothesis (each unique set of parameter values is one hypothesis)
- A prediction is correct if $y \cdot g(\boldsymbol{x},\boldsymbol{w}) > 0$ (or $y \cdot h(\boldsymbol{x},\boldsymbol{w}) > 0$)

Geometrically, using the canonical representation translates to two things:

1. *It will increase the input space dimension by 1, and*
2. *the decision boundary now always passes through the origin.*

Geometric view



How to learn: the perceptron algorithm

The equation $w_0 + w_1 x_1 + ... + w_m x_m = 0$ defines a linear decision boundary that separates input space into different decision regions

The goal of learning is to find a weight vector w such that its decision boundary correctly separate positive examples from negative examples.

How can we achieve this?

Perceptron is one approach. It starts with some vector ***w*** and incrementally update ***w*** when it makes a mistake.

Let $\boldsymbol{w}_t$ be current weight vector, and suppose it makes a mistake on example $\langle \boldsymbol{x}, y \rangle$, that is to say $y \cdot \boldsymbol{w}_t \cdot \boldsymbol{x} < 0$. The perceptron update rule is: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y \cdot \boldsymbol{x}$

Perceptron Algorithm

Let $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$

Repeat

Accept training example i **:** $(\mathbf{x}^i, y^i)$

$u^i \leftarrow \mathbf{w} \cdot \mathbf{x}^i$

if $y^i \cdot u^i <= 0$

$\mathbf{w} \leftarrow \mathbf{w} + y^i \mathbf{x}^i$

Effect of Perceptron Updating Rule

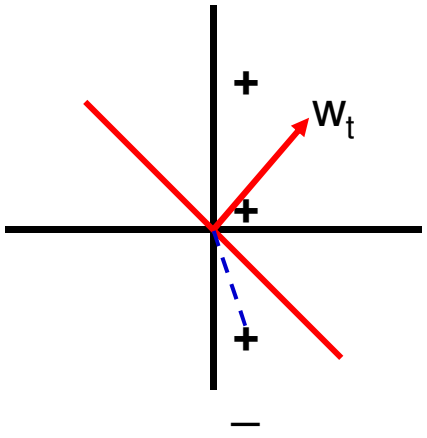
- **Mathematically speaking**

  $y \cdot \boldsymbol{w}_{t+1} \cdot \boldsymbol{x} = y \cdot (\boldsymbol{w}_t + y \cdot \boldsymbol{x}) \cdot \boldsymbol{x} = y \cdot \boldsymbol{w}_t \cdot \boldsymbol{x} + y^2 \|\boldsymbol{x}\|^2$

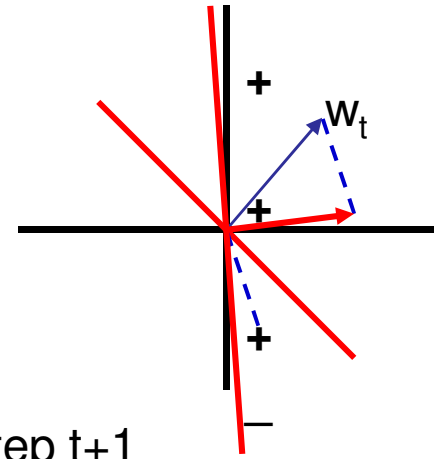
  $> y \cdot \boldsymbol{w}_t \cdot \boldsymbol{x}$

  The updating rule makes $y \cdot \boldsymbol{w}_t \cdot \boldsymbol{x}$ more positive, thus can potentially correct the mistake

- **Geometrically**



Step t



Step t+1

Online vs Batch

- We call the above perceptron algorithm an ***online algorithm***
- Online algorithms perform learning each time it receives an training example
- In contrast, ***batch learning*** algorithms collect a batch of training examples and learn from them all at once.

Batch Perceptron Algorithm

Given : training examples $(\mathbf{x}^i, y^i),\ i = 1,\ldots,N$

Let $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$

do

$delta \leftarrow (0,0,0,\ldots,0)$

for $i = 1$ **to** N **do**

$u^i \leftarrow \mathbf{w} \cdot \mathbf{x}^i$

if $y^i \cdot u^i <= 0$

$delta \leftarrow delta + y^i \cdot \mathbf{x}^i$

$delta \leftarrow delta / N$

$\mathbf{w} \leftarrow \mathbf{w} + \eta\, delta$

until $|delta| < \varepsilon$

Good news

- If there is a linear decision boundary that correctly classify all training examples, this algorithm will find it
- Formally speaking, this is the **convergence Property:**

  For linearly separable data (i.e., there exists an linear decision boundary that perfectly separates positive and negative training examples), the perceptron algorithm converges in a **finite number of steps.**
- *Why?* If you are mathematically curious, read the following slide, you will find the answer.
- *And how many steps?* If you are practically curious, read the following slide, answer is in there too.
- **The further good news** is that you are not required to master this material, they are just for the curious ones

To show convergence, we just need to show that each update moves the weight vector closer to a solution vector by a lower bounded amount
Let w* be a solution vector, and w_t be our w at *t*th step,

Proof

$$\cos ine(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \cdot \|w_t\|}$$

$$w^* \cdot w_t = w^* \cdot (w_{t-1} + y^t x^t) = w^* \cdot w_{t-1} + w^* y^t x^t$$

Assume that w^* classify all examples with a margin γ, i.e., $w^* yx > \gamma$ for all examples

$$w^* \cdot w_t = w^* \cdot w_{t-1} + w^* y^t x^t > w^* \cdot w_{t-1} + \gamma > w^* \cdot w_{t-2} + 2\gamma > \ldots > w^* w_0 + t\gamma = t\gamma$$

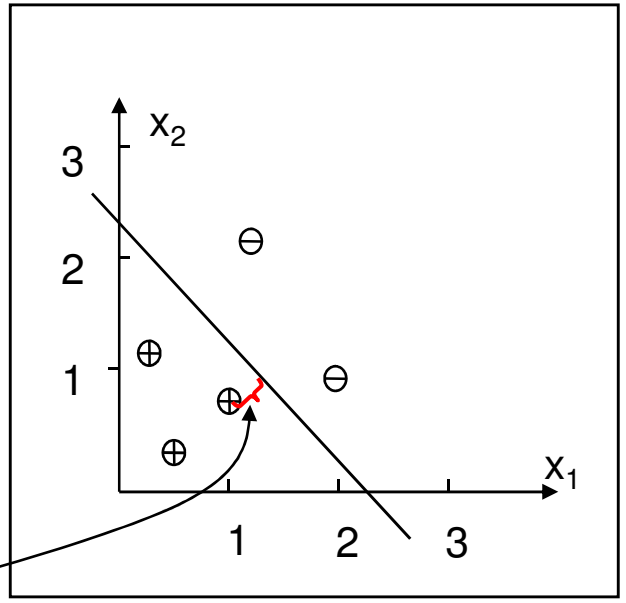
$$\|w_t\|^2 = \|w_{t-1} + y^t x^t\|^2 = \|w_{t-1}\|^2 + y^{t2}\|x^t\|^2 + 2w_{t-1} y^t x^t < \|w_{t-1}\|^2 + \|x^t\|^2$$

Assume that $\|\mathbf{x}\|$ are bounded by D

$$\|w_t\|^2 < \|w_{t-1}\|^2 + \|x^t\|^2 < \|w_{t-1}\|^2 + D^2 < \|w_{t-2}\|^2 + 2D^2 < \ldots < tD^2$$

$$\cos ine(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \cdot \|w_t\|} > \frac{t\gamma}{\|w^*\| \cdot \|w_t\|} > \frac{t\gamma}{\|w^*\| \cdot \sqrt{tD^2}}$$

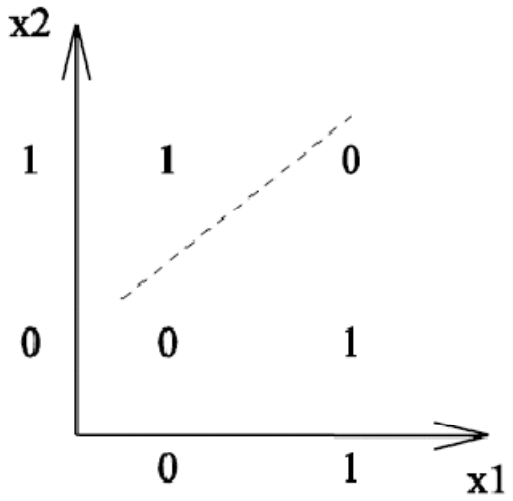
$$\frac{t\gamma}{\|w^*\| \cdot \sqrt{tD^2}} < 1 \Rightarrow \sqrt{t} < \frac{D\|w^*\|}{\gamma} \Rightarrow t < D^2 \frac{\gamma^2}{\|w^*\|^2}$$



Margin

- $\frac{\gamma^2}{\|w^*\|^2}$ is referred to as the margin
  - The bigger the margin, the easier the classification problem is, the perceptron algorithm will likely find the solution faster!
  - Side story: the bigger the margin, the more confident we are about our prediction, which makes it desirable to find the one that gives the maximum margin
  - Later in the course this concept will be core to one of the recent most exciting developments in the ML field – support vector machine

Bad news



What about non-linearly separable cases!

In such cases the algorithm will never stop! How to fix?

One possible solution: look for decision boundary that make as few mistakes as possible – NP-hard (refresh your 325 memory!)

Fixing the Perceptron

Idea one: only go through the data once, or a fixed number of times

Let $\mathbf{w} \leftarrow (0,0,0,\ldots,0)$

for $\mathbf{i} = 1,\ldots,N$

  Take training example i **:** $(\mathbf{x}^i, y^i)$

  $u^i \leftarrow \mathbf{w} \cdot \mathbf{x}^i$

  if $y^i \cdot u^i < 0$

    $\mathbf{w} \leftarrow \mathbf{w} + y^i \mathbf{x}^i$

At least this stops!

Problem: the final w might not be good

e.g., right before terminating, the algorithm might perform an update on a total outlier…

Voted-Perceptron

Idea two: keep around intermediate hypotheses, and have them "vote" [Freund and Schapire, 1998]

Let $w_0 = (0,0,0, ...,0)$

$c_0 = 0$

repeat

  Take example $i : (x^i, y^i)$

  $u^i \leftarrow \mathbf{w}_n \cdot \mathbf{x}^i$

  if $y^i \cdot u^i <= 0$

    $\mathbf{w}_{n+1} \leftarrow \mathbf{w}_n + y^i \mathbf{x}^i$

    $c_{n+1} = 0$

    $n = n + 1$

  else

    $c_n = c_n + 1$

Store a collection of linear separators w_0, w_1,…, along with their survival time c_0, c_1, …

The c's can be good measures of reliability of the w's.

For classification, take a weighted vote among all separators:

$$\text{sgn}\left\{\sum_{n=0}^{N} c_n \, \text{sgn}(w_n \cdot x)\right\}$$