

# CPSC 340: Machine Learning and Data Mining

## Linear Classifiers

Andreas Lehrmann and Mark Schmidt  
University of British Columbia, Fall 2022  
<https://www.students.cs.ubc.ca/~cs-340>

# Admin

- Midterm is **tonight**.
  - Time: 6:30pm.
  - Location: IRC 2.
  - Duration: 80 minutes.
  - See **Piazza** for additional details (what to bring, exam content, etc.).

Good luck!

# Last Time: L1-Regularization

- We discussed **L1-regularization**:

$$f(w) = \frac{1}{2} \|Xw - y\|^2 + \lambda \|w\|_1$$

- Also known as “LASSO” and “basis pursuit denoising”.
  - **Regularizes ‘w’** so we decrease our test error (like L2-regularization).
  - Yields **sparse ‘w’** so it selects features (like L0-regularization).
- **Properties:**
    - It’s **convex and fast** to minimize (with “proximal-gradient” methods).
    - Solution is **not unique** (sometimes people do L2- and L1-regularization)

# L1-loss vs. L1-regularization

- Don't confuse the L1 loss with L1-regularization!
  - L1-loss is robust to outlier data points.
    - You can use this instead of removing outliers.
  - L1-regularization is robust to irrelevant features.
    - You can use this instead of removing features.
- And note that you can be robust to outliers and irrelevant features:

$$f(w) = \underbrace{\|Xw - y\|_1}_{L_1\text{-loss}} + \lambda \underbrace{\|w\|_1}_{L_1\text{-regularizer}}$$

- Can we smooth and use “Huber regularization”?
  - Huber regularizer is still robust to irrelevant features.
  - But it's the non-smoothness that sets weights to exactly 0.

# L\*-Regularization

- **L0-regularization** (AIC, BIC, Mallows's Cp, Adjusted R<sup>2</sup>, ANOVA):
  - Adds **penalty on the number of non-zeros** to select features.

$$f(w) = \|Xw - y\|^2 + \lambda \|w\|_0$$

- **L2-regularization** (ridge regression):
  - Adding **penalty on the L2-norm** of 'w' to decrease overfitting:

$$f(w) = \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2$$

- **L1-regularization** (LASSO):
  - Adding **penalty on the L1-norm** decreases overfitting and selects features:

$$f(w) = \|Xw - y\|^2 + \lambda \|w\|_1$$

# L0- vs. L1- vs. L2-Regularization

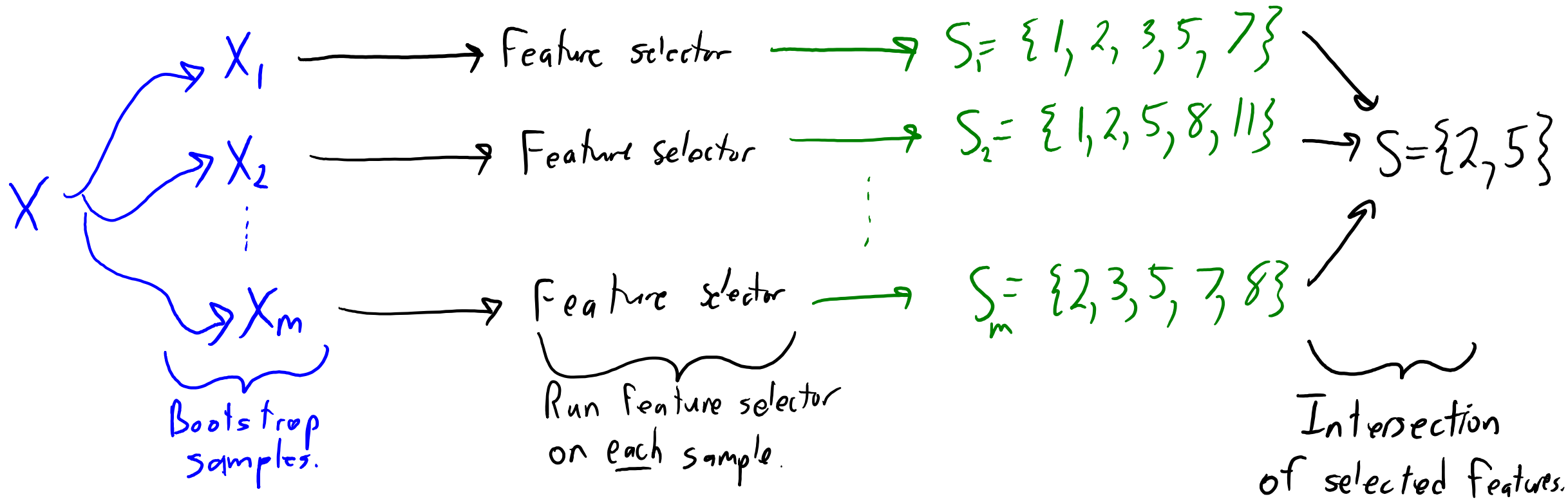
	Sparse 'w' (Selects Features)	Speed	Unique 'w'	Coding Effort	Irrelevant Features
L0-Regularization	Yes	Slow	No	Few lines	Not Sensitive
L1-Regularization	Yes*	Fast*	No	1 line*	Not Sensitive
L2-Regularization	No	Fast	Yes	1 line	A bit sensitive

- L1-Regularization isn't as sparse as L0-regularization.
  - L1-regularization tends to give more false positives (selects too many).
  - And it's only "fast" and "1 line" with specialized solvers.
- Cost of L2-regularized least squares is  $O(nd^2 + d^3)$ .
  - Changes to  $O(ndt)$  for 't' iterations of gradient descent (same for L1).
- "Elastic Net" (L1- and L2-regularization) is sparse, fast, and unique.
- Using L0+L2 does not give a unique solution.

# Ensemble Feature Selection

- We can also use **ensemble methods** for feature selection.
  - Usually designed to **reduce false positives** or **reduce false negatives**.
- In the case of L1-regularization, we **want to reduce false positives**.
  - Unlike L0-regularization, the **non-zero  $w_j$  are still “shrunk”**.
    - “Irrelevant” variables can be included before “relevant”  $w_j$  reach best value.
- A **bootstrap** approach to reducing false positives:
  - Apply the method to bootstrap samples of the training data.
  - Only take the **features selected in all bootstrap samples**.

# Ensemble Feature Selection



- Example: bootstrapping plus L1-regularization (“BoLASSO”).
  - Reduces false positives.
  - It’s possible to show it recovers “correct” variables with weaker conditions.
    - Can replace “intersection” with “selected frequency” if has false negatives too.



Next Topic: Linear Classifiers

# Motivation: Identifying Important E-mails

- How can we automatically identify ‘important’ e-mails?



- A **binary classification** problem (“important” vs. “not important”).
  - Labels are approximated by whether you took an “action” based on mail.
  - High-dimensional feature set (e.g., bag of words).
- Gmail uses **regression for this binary classification** problem.

# Binary Classification Using Regression?

- Can we apply linear models for **binary classification**?
  - Set  $y_i = +1$  for one class (“important”).
  - Set  $y_i = -1$  for the other class (“not important”).
- At training time, **fit a linear regression** model:

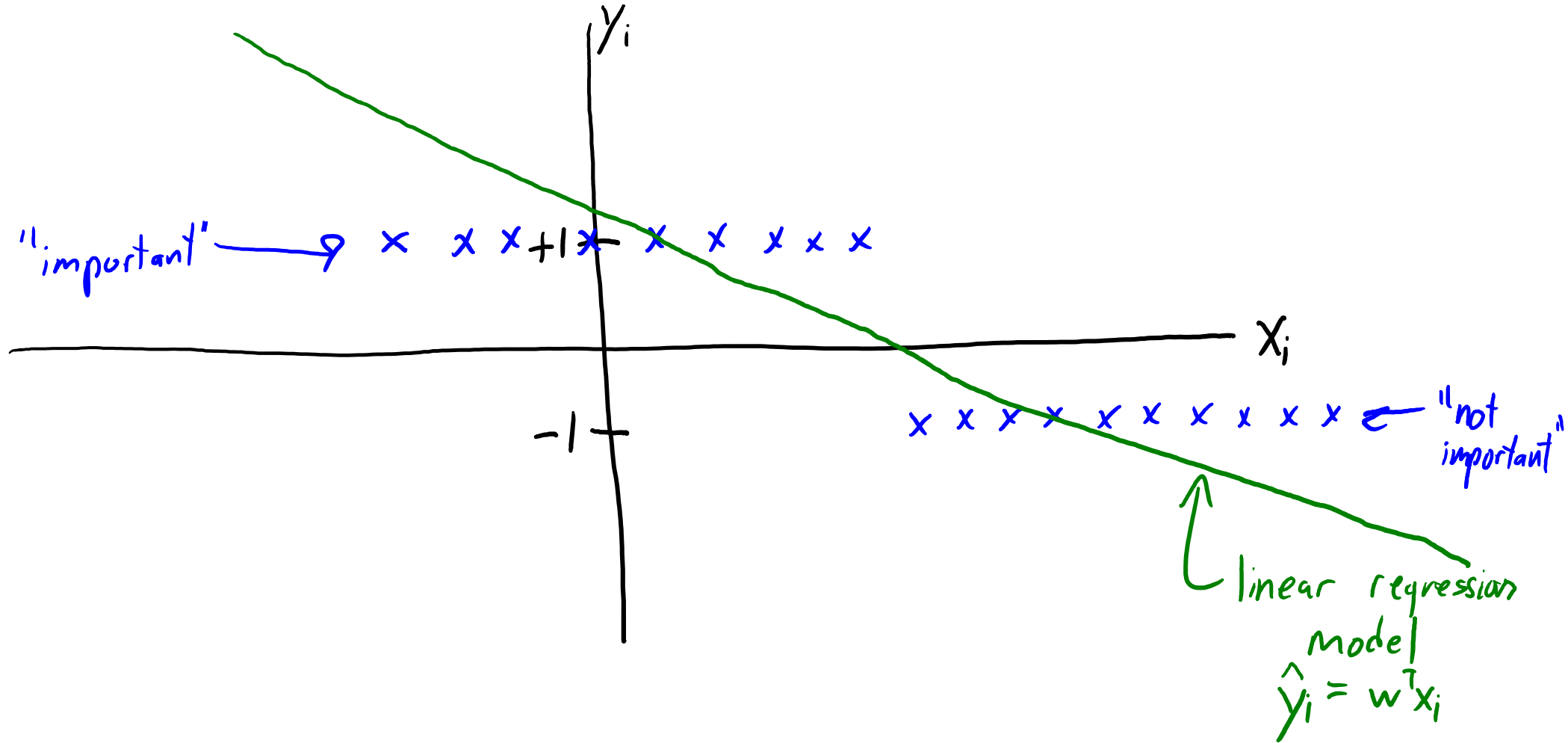
$$\begin{aligned}\hat{y}_i &= w_1 x_{i1} + w_2 x_{i2} + \dots + w_d x_{id} \\ &= \mathbf{w}^T \mathbf{x}_i\end{aligned}$$

- The model will try to make  $\mathbf{w}^T \mathbf{x}_i = +1$  for “important” e-mails, and  $\mathbf{w}^T \mathbf{x}_i = -1$  for “not important” e-mails.

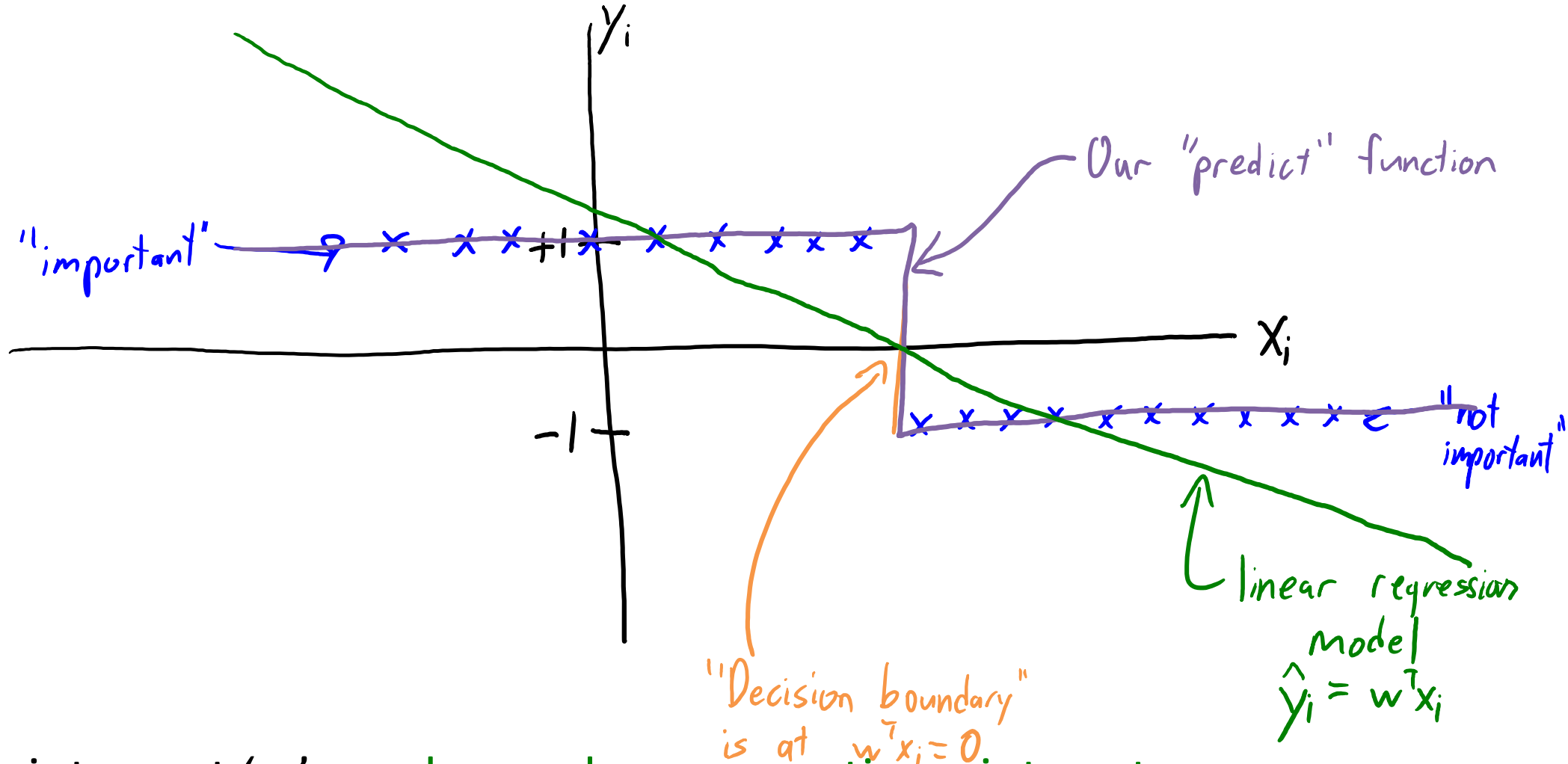
# Binary Classification Using Regression?

- Can we apply linear models for **binary classification**?
  - Set  $y_i = +1$  for one class (“important”).
  - Set  $y_i = -1$  for the other class (“not important”).
- **Linear model gives real numbers** like 0.9, -1.1, and so on.
- So to predict, we **look at whether  $w^T x_i$  is closer to +1 or -1**.
  - If  $w^T x_i = 0.9$ , predict  $\hat{y}_i = +1$ .
  - If  $w^T x_i = -1.1$ , predict  $\hat{y}_i = -1$ .
  - If  $w^T x_i = 0.1$ , predict  $\hat{y}_i = +1$ .
  - If  $w^T x_i = -100$ , predict  $\hat{y}_i = -1$ .
  - We write this operation (rounding to +1 or -1) as  $\hat{y}_i = \text{sign}(w^T x_i)$ .

# Decision Boundary in 1D



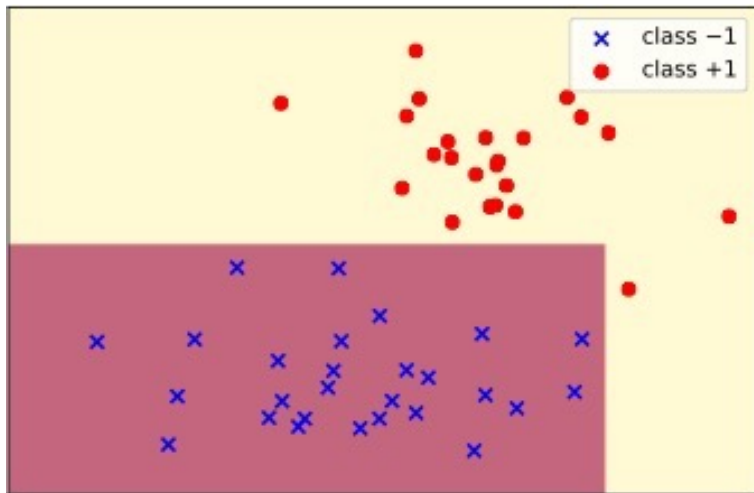
# Decision Boundary in 1D



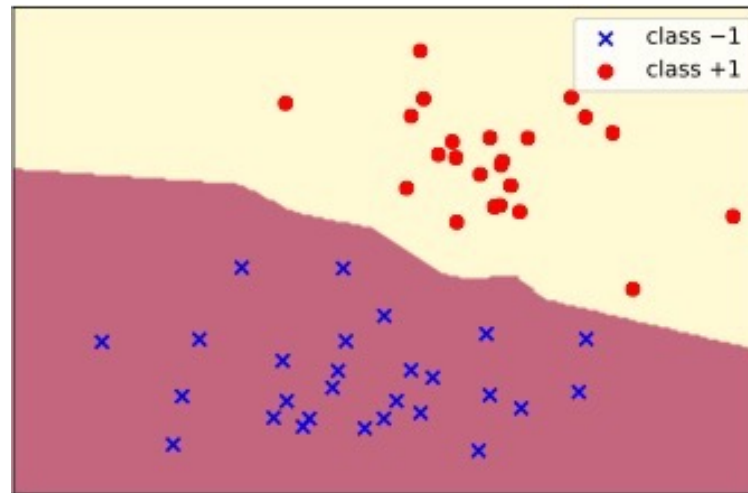
- We can interpret 'w' as a **hyperplane** separating  $x$  into sets:
  - Set where  $w^T x_i > 0$  and set where  $w^T x_i < 0$ .

# Decision Boundary in 2D

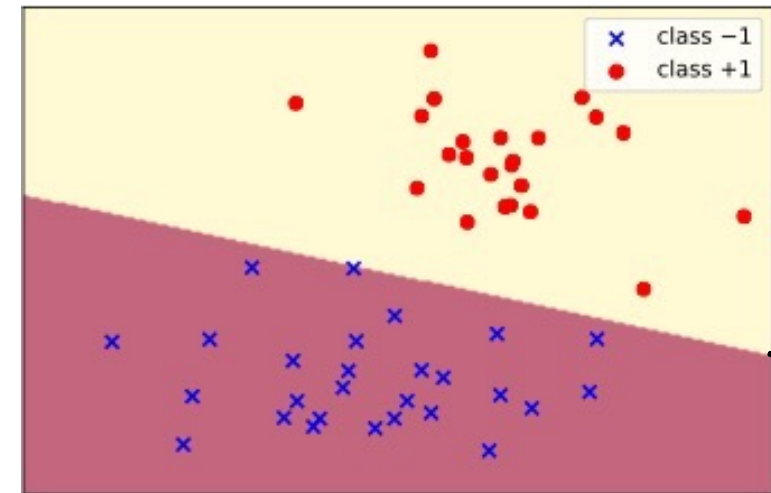
decision tree



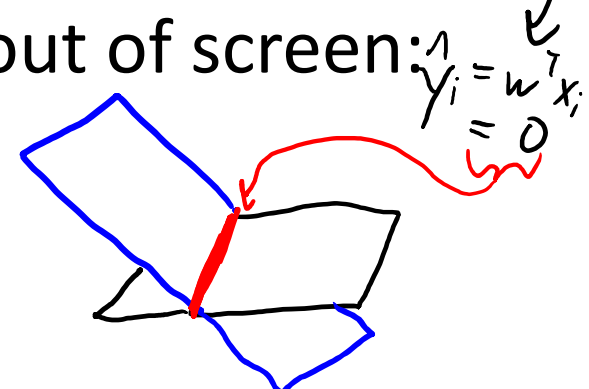
KNN



linear classifier

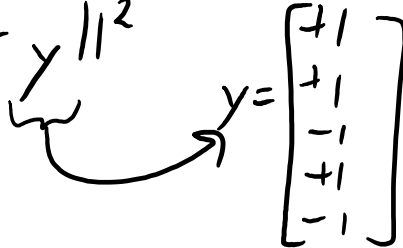


- Linear classifier would be a  $\hat{y}_i = w^T x_i$  function coming out of screen:
  - The boundary is at  $\hat{y}_i = 0$ .



# Should we use least squares for classification?

- Consider training by minimizing squared error with  $y_i$  that are +1 or -1:

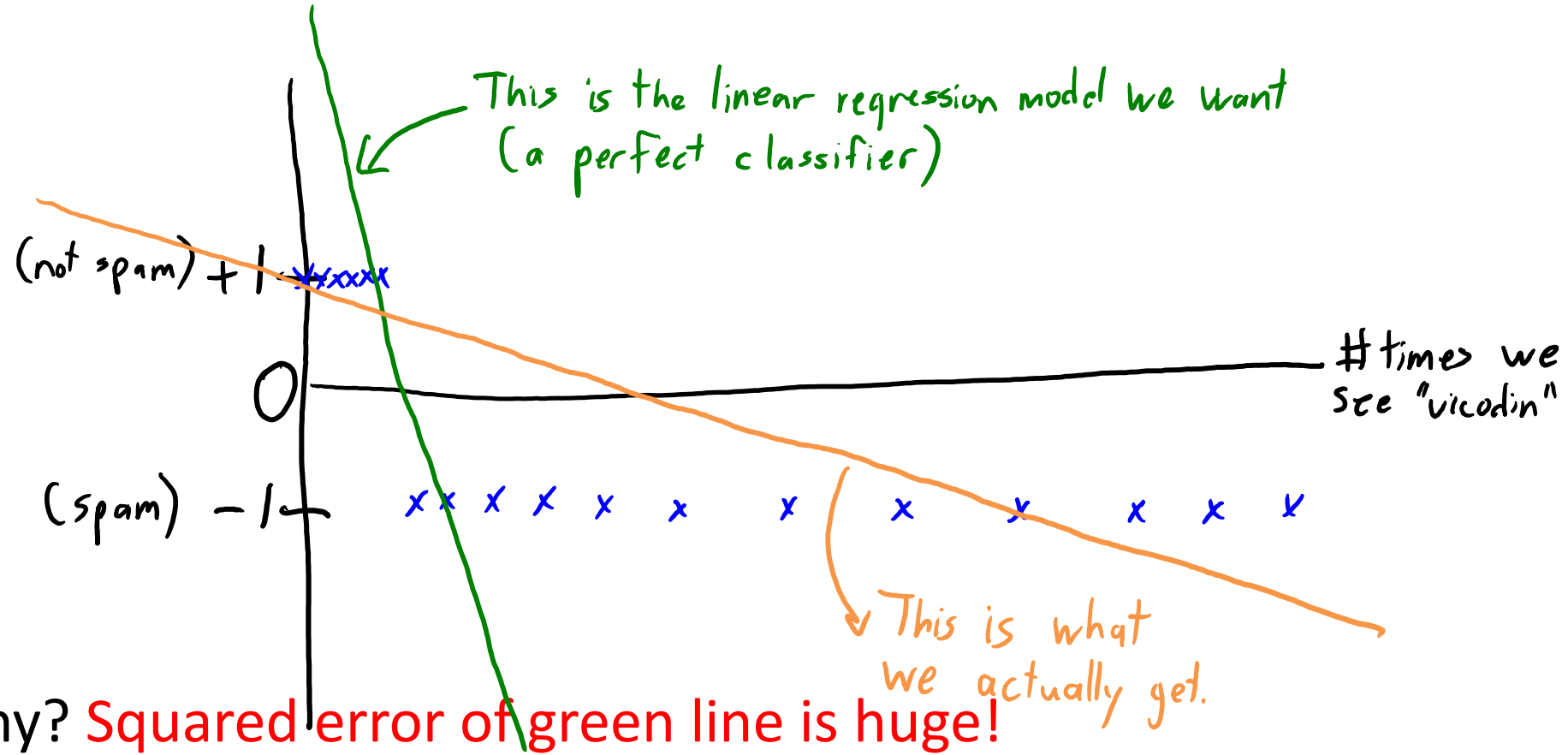
$$f(w) = \frac{1}{2} \|Xw - y\|^2$$


- If we predict  $w^T x_i = +0.9$  and  $y_i = +1$ , error is small:  $(0.9 - 1)^2 = 0.01$ .
- If we predict  $w^T x_i = -0.8$  and  $y_i = +1$ , error is bigger:  $(-0.8 - 1)^2 = 3.24$ .
- If we predict  $w^T x_i = +100$  and  $y_i = +1$ , error is huge:  $(100 - 1)^2 = 9801$ .
  - But it shouldn't be, the prediction is correct.
- Least squares penalized for being “too right”.
  - +100 has the right sign, so the error should not be large.



# Should we use least squares for classification?

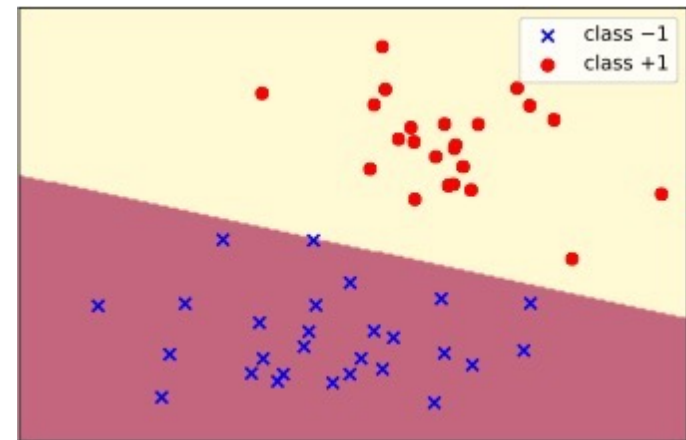
- Least squares can behave weirdly when applied to classification:



- Why? **Squared error of green line is huge!**
  - Make sure you understand why the green line achieves 0 training error.

# “0-1 Loss” Function: Minimizing Classification Errors

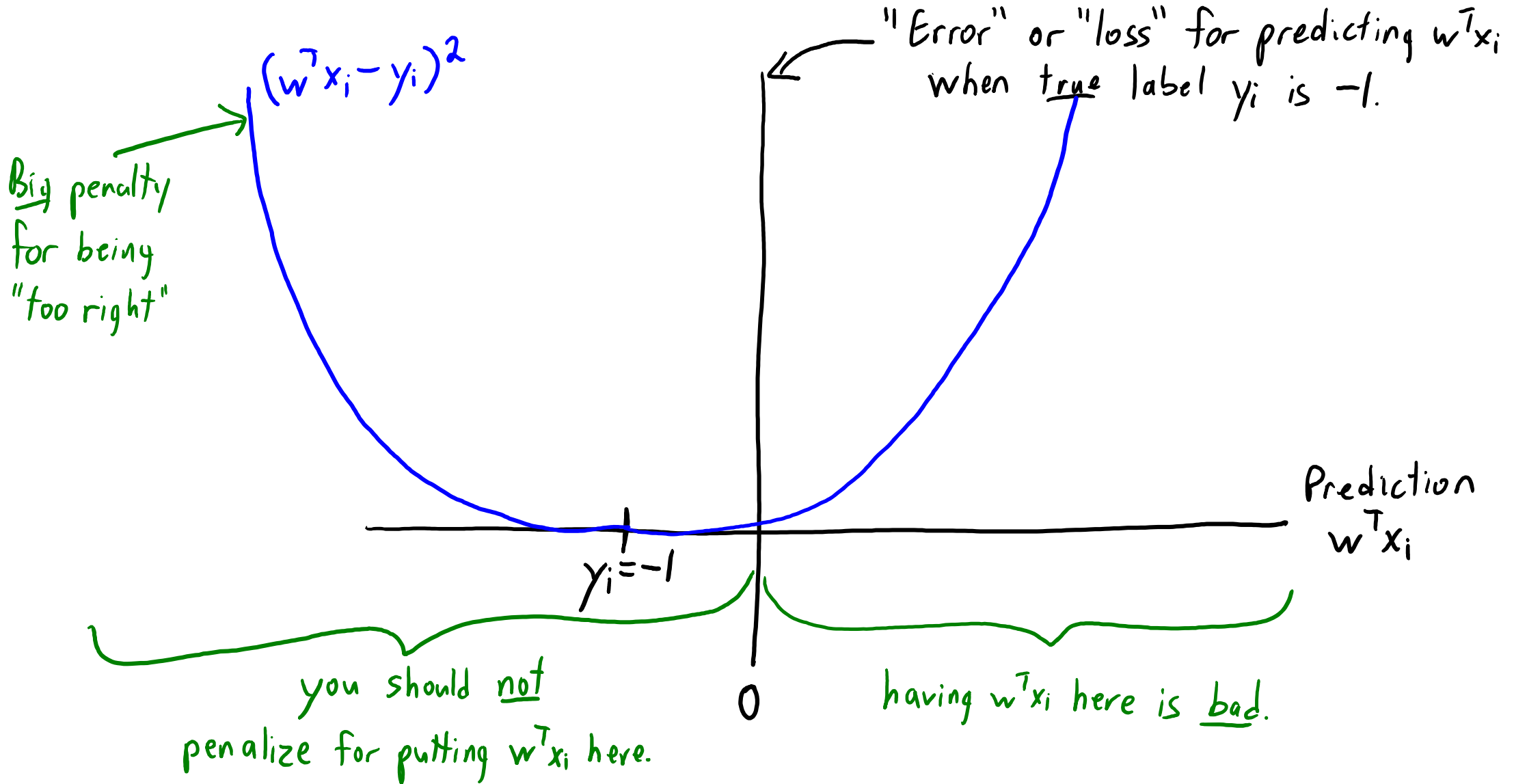
- Could we instead minimize **number of classification errors**?
  - This is called the **0-1 loss** function:
    - You either get the classification wrong (1) or right (0).
  - We can write using the L0-norm as  $||\hat{y} - y||_0$ .
    - Unlike regression, in classification it's reasonable that  $\hat{y}_i = y_i$  (it's either +1 or -1).
- Important special case: “**linearly separable**” data.
  - Classes can be “**separated**” by a hyper-plane.
  - So a perfect linear classifier exists.



# Perceptron Algorithm for Linearly-Separable Data

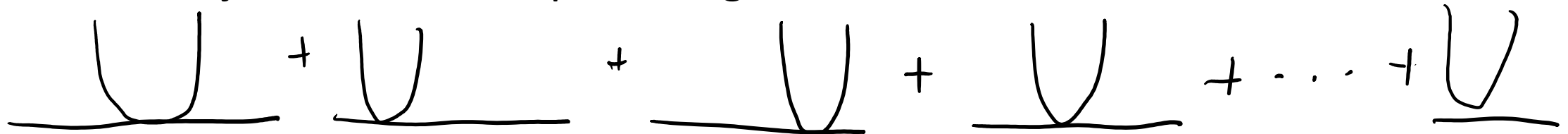
- One of the first “learning” algorithms was the “perceptron” (1957).
  - Searches for a ‘w’ such that  $\text{sign}(w^T x_i) = y_i$  for all  $i$ .
- **Perceptron** algorithm:
  - Start with  $w^0 = 0$ .
  - Go through examples in any order until you **make a mistake** predicting  $y_i$ .
    - Set  $w^{t+1} = w^t + y_i x_i$ .
  - Keep going through examples until you make no errors on training data.
- **If a perfect classifier exists**, this algorithm finds one in finite number of steps.
- Intuition:
  - Consider a case where  $w^T x_i < 0$  but  $y_i = +1$ .
  - In this case the update “**adds more of  $x_i$  to  $w$** ” so that  $w^T x_i$  is larger.
$$(w^{t+1})^T x_i = (w^t + x_i)^T x_i = (w^t)^T x_i + x_i^T x_i = (\text{old prediction}) + \|x_i\|^2$$
  - If  $y_i = -1$ , you would be subtracting the squared norm.

# Geometry of why we want the 0-1 loss



# Thoughts on the previous (and next) slide

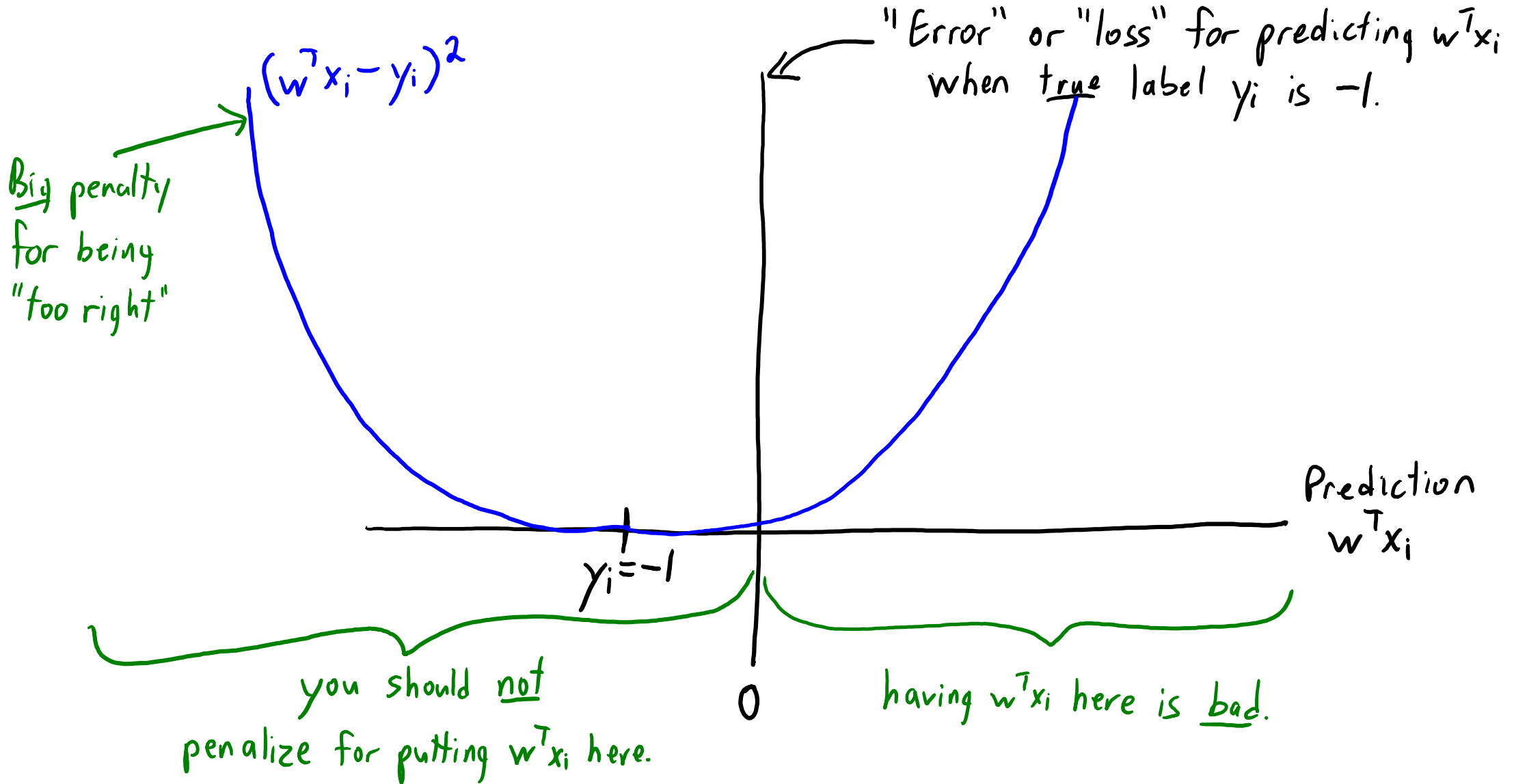
- We are now plotting the **loss vs. the predicted  $w^T x_i$** .
  - “Loss space”, which is different than parameter space or data space.
- We're plotting the individual loss **for a particular training example**.
  - In the figure the **label is  $y_i = -1$  (so loss is centered at -1)**.
    - It will be centered at +1 when  $y_i = +1$ .
  - The objective in least squares regression is a sum of ‘n’ of these losses:



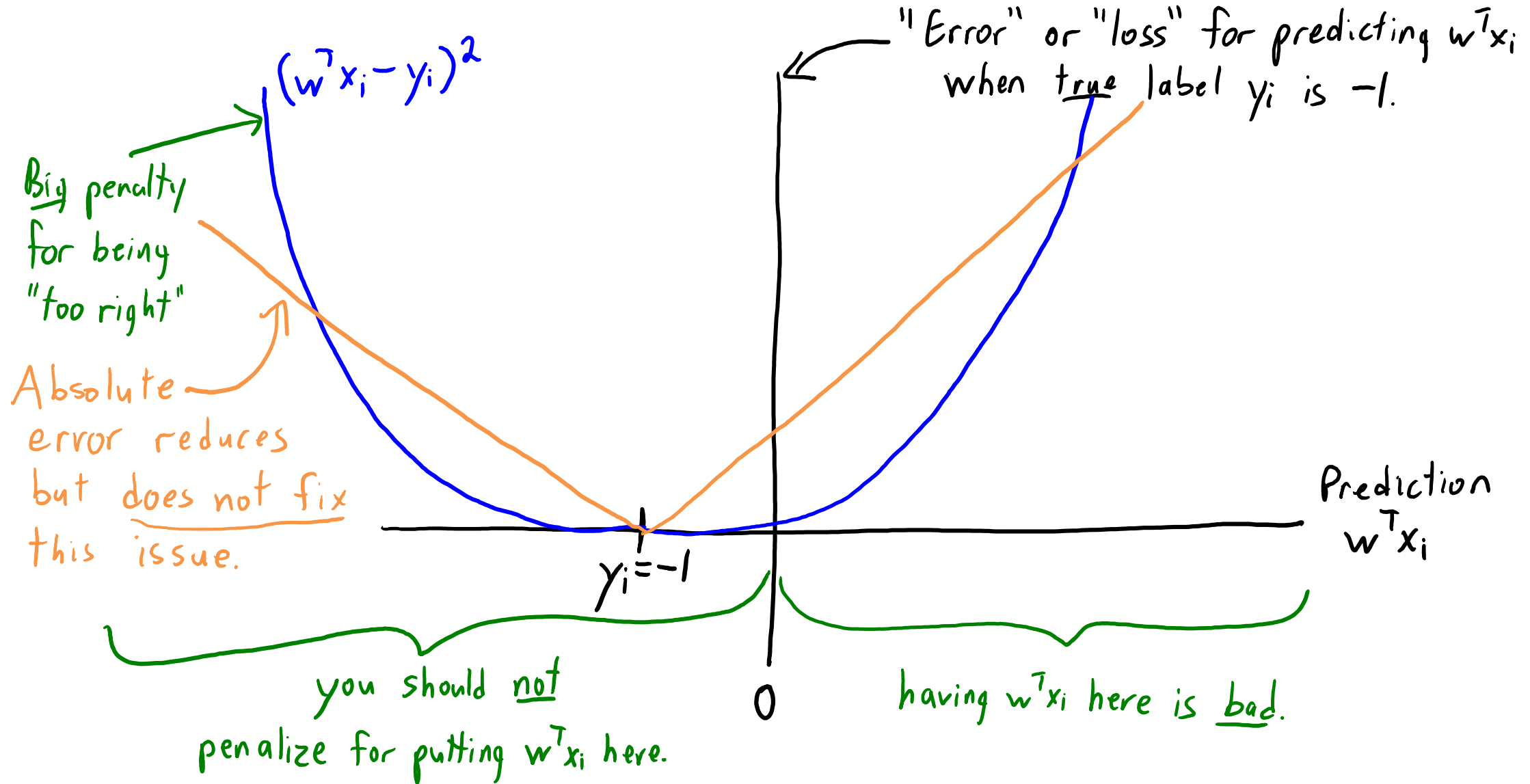
A hand-drawn diagram illustrating the sum of individual losses. It consists of five parabolic curves, each drawn on a horizontal baseline. The curves are connected by plus signs and an ellipsis, representing a summation. The first curve is centered on the left, the second is centered slightly to the right of the first, the third is centered further to the right, the fourth is centered even further to the right, and the fifth is centered on the far right. This visualizes how the total loss is a sum of individual losses for different training examples.

- (The next slide is the same as the previous one)

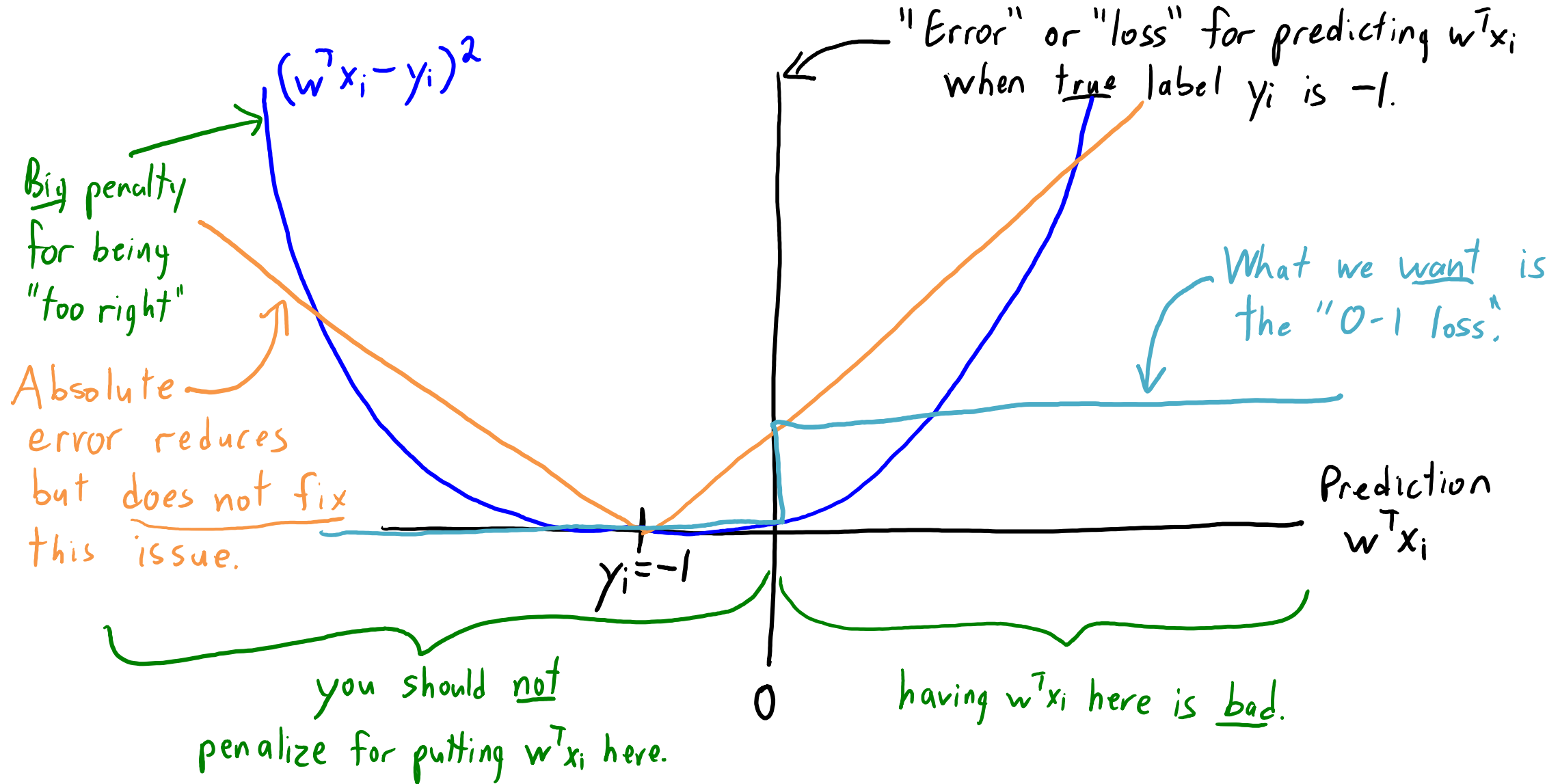
# Geometry of why we want the 0-1 loss



# Geometry of why we want the 0-1 loss



# Geometry of why we want the 0-1 loss





# 0-1 Loss Function

- Unfortunately the **0-1 loss is non-convex** in 'w'.
  - It's easy to minimize if a perfect classifier exists (perceptron).
  - Otherwise, finding the 'w' **minimizing 0-1 loss is a hard problem**.
  - Gradient is zero everywhere: don't even know "which way to go".
  - NOT the same type of problem we had with using the squared loss.
    - We can minimize the squared error, but it might give a bad model for classification.
- Motivates **convex approximations to 0-1 loss...**

# Degenerate Convex Approximation to 0-1 Loss

- If  $y_i = +1$ , we get the label right if  $w^T x_i > 0$ .
- If  $y_i = -1$ , we get the label right if  $w^T x_i < 0$ , or equivalently  $-w^T x_i > 0$ .
- So “classifying ‘i’ correctly” is equivalent to having  $y_i w^T x_i > 0$ .
- One possible convex approximation to 0-1 loss:
  - Minimize how much this constraint is violated.

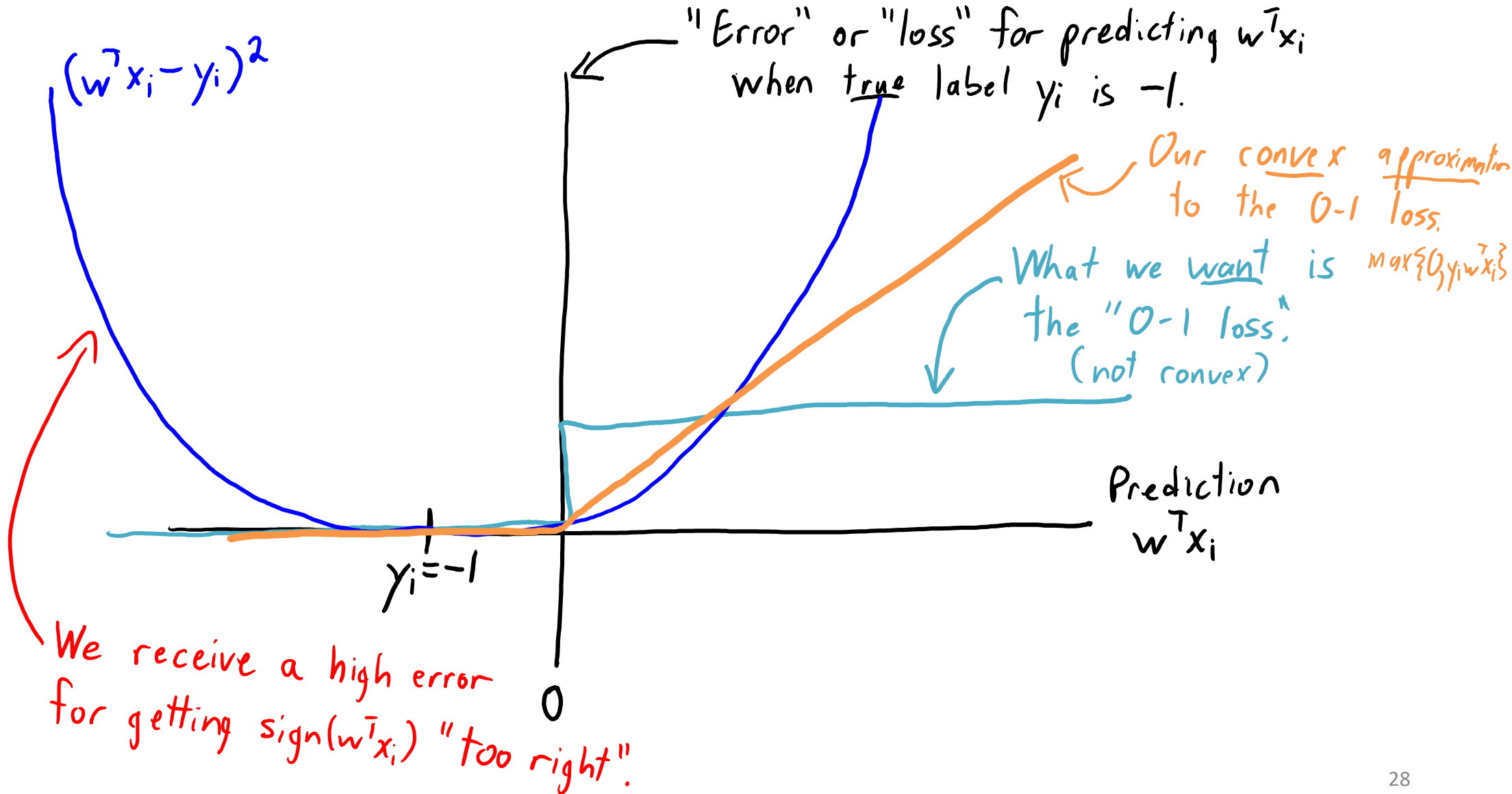
If  $y_i w^T x_i > 0$  then you get an “error” of 0.

If  $y_i w^T x_i < 0$  then you get an “error” of  $-y_i w^T x_i$ .

→ So the “error” is given by  $\max\{0, -y_i w^T x_i\}$

$\max\{\text{constant}, \text{linear}\} \Rightarrow \text{convex}$

# Convex Approximation to 0-1 Loss



# Degenerate Convex Approximation to 0-1 Loss

- Our convex approximation of the error for **one example** is:

$$\max\{0, -y_i w^T x_i\}$$

- We could train by minimizing **sum over all examples**:

$$f(w) = \sum_{i=1}^n \max\{0, -y_i w^T x_i\}$$

- But this has a **degenerate solution**:
  - We have  $f(0) = 0$ , and this is the lowest possible value of 'f'.
- There are two standard fixes: **hinge loss** and **logistic loss**.

# Hinge Loss

- We saw that we **classify examples 'i' correctly** if  $y_i w^T x_i > 0$ .
  - Our convex approximation is the amount this inequality is violated.
- Consider replacing  $y_i w^T x_i > 0$  with  $y_i w^T x_i \geq 1$ .
  - (the "1" is arbitrary: we could make  $\|w\|$  bigger/smaller to use any positive constant)

- The **violation of this constraint** is now given by:

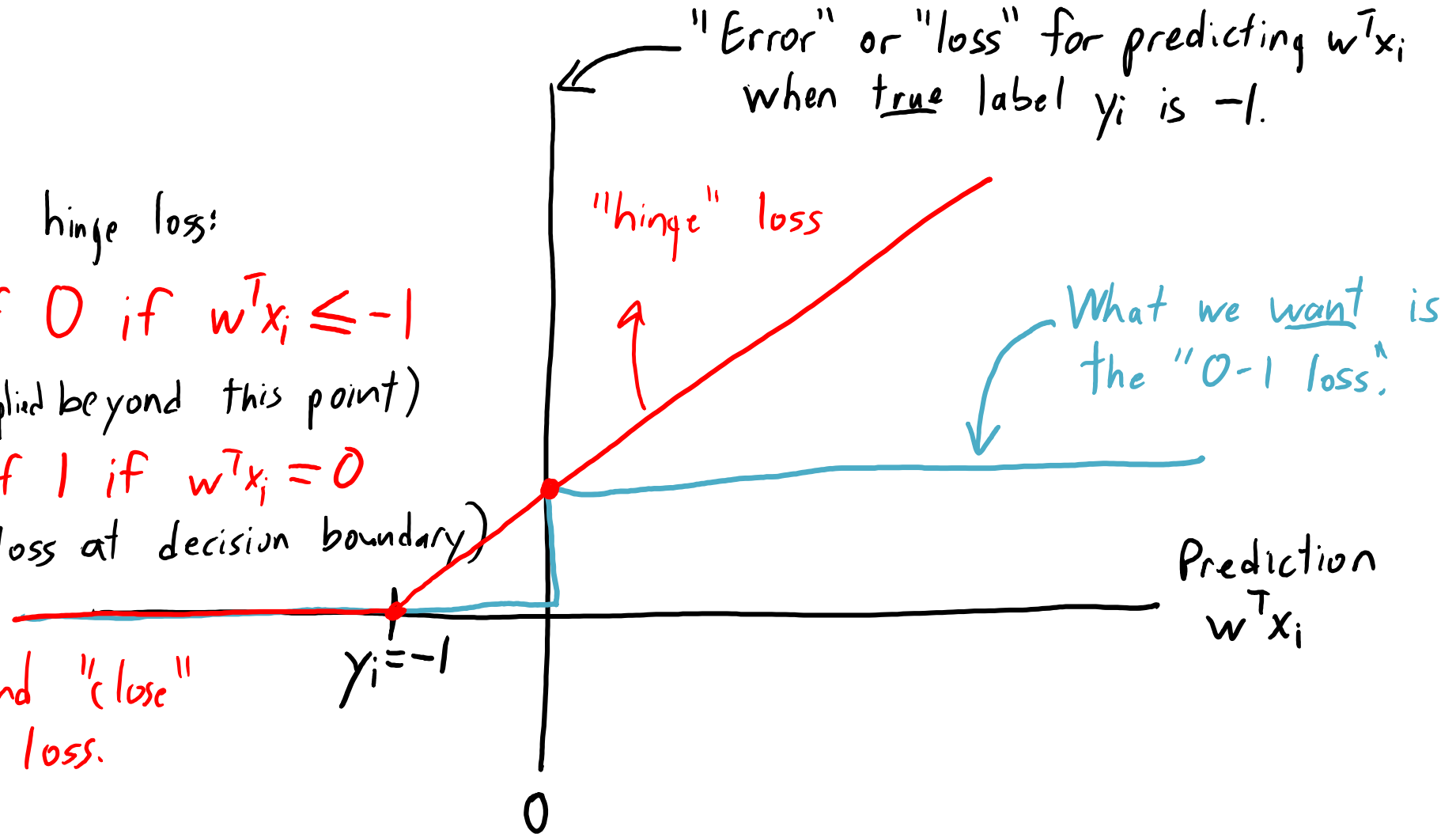
$$\max \{0, 1 - y_i w^T x_i\}$$

- This is the called **hinge loss**.
  - It's **convex**:  $\max(\text{constant}, \text{linear})$ .
  - It's **not degenerate**:  $w=0$  now gives an error of 1 instead of 0.

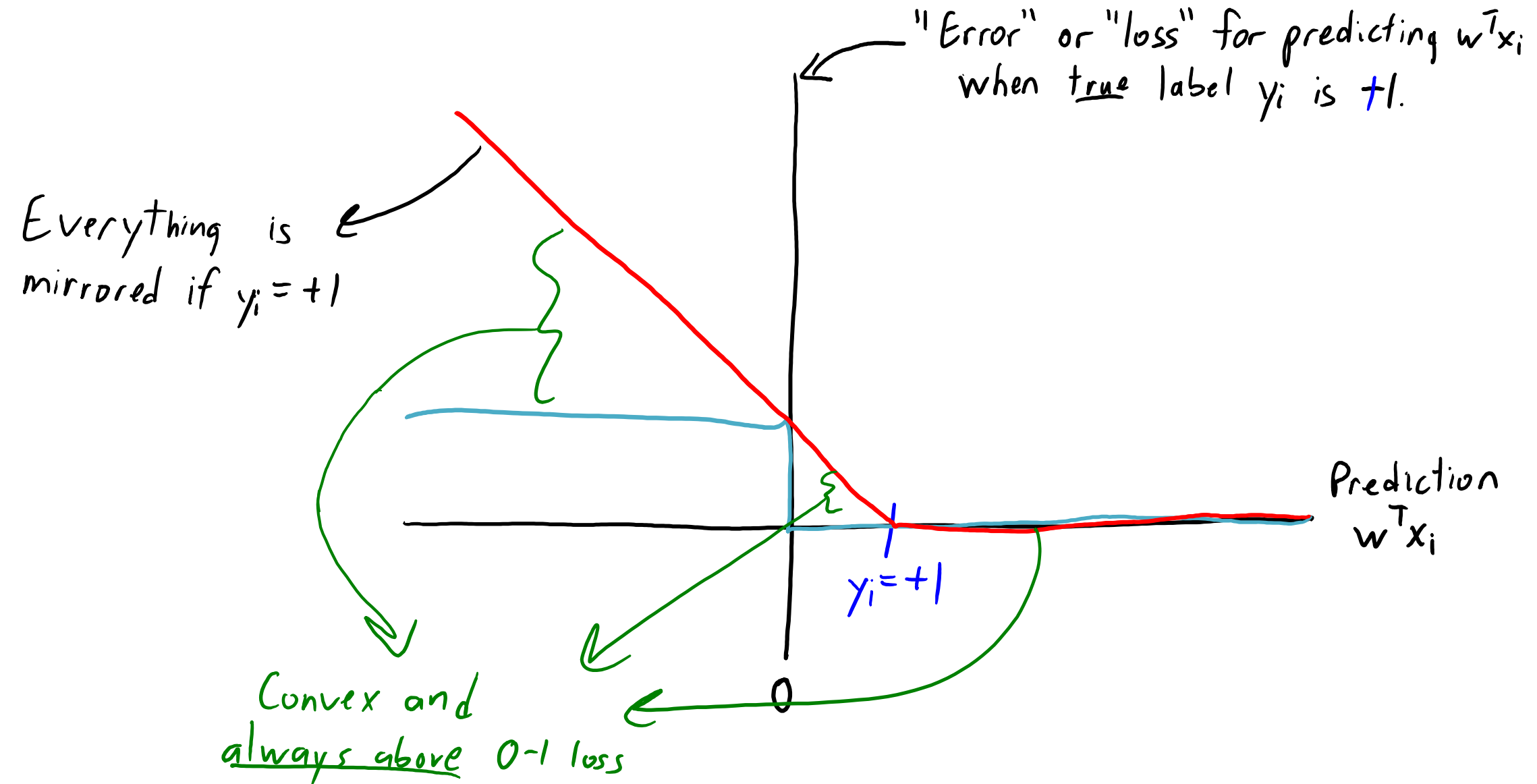
# Hinge Loss: Convex Approximation to 0-1 Loss

Properties of the hinge loss:

1. Has error of 0 if  $w^T x_i \leq -1$   
(no penalty applied beyond this point)
2. Has a loss of 1 if  $w^T x_i = 0$   
(matches 0-1 loss at decision boundary)
3. Is convex and "close"  
to 0-1 loss.



# Hinge Loss: Convex Approximation to 0-1 Loss



# Hinge Loss

- Hinge loss for all 'n' training examples is given by:

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\}$$

- Convex upper bound on 0-1 loss.
  - If the hinge loss is 18.3, then number of training errors is at most 18.
  - So minimizing hinge loss indirectly tries to minimize training error.
  - Like perceptron, finds a perfect linear classifier if one exists.
- Support vector machine (SVM) is hinge loss with L2-regularization.

$$f(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2$$

- There exist specialized optimization algorithm for this problems.
- SVMs can also be viewed as “maximizing the margin” (later).



# Summary

- Ensemble feature selection reduces false positives or negatives.
- Binary classification using regression:
  - Encode using  $y_i$  in  $\{-1,1\}$ .
  - Use  $\text{sign}(w^T x_i)$  as prediction.
  - “Linear classifier” (a hyperplane splitting the space in half).
- Least squares is a weird error for classification.
- Perceptron algorithm: finds a perfect classifier (if one exists).
- 0-1 loss is the ideal loss, but is non-smooth and non-convex.
- Hinge loss is a convex upper bound on 0-1 loss.
  - SVMs add L2-regularization.
- Next time: one of the best “out of the box” classifiers.

# L1-Regularization as a Feature Selection Method

- Advantages:
  - Deals with conditional independence (if linear).
  - Sort of **deals with collinearity**:
    - Picks at least one of “mom” and “mom2”.
  - Very fast with specialized algorithms.
- Disadvantages:
  - Tends to give **false positives** (selects too many variables).
- Neither good nor bad:
  - Does not take small effects.
  - Says “gender” is relevant if we know “baby”.
  - **Good for prediction if we want fast training and don't care about having some irrelevant variables included.**

# “Elastic Net”: L2- and L1-Regularization

- To address **non-uniqueness**, some authors **use L2- and L1-**:

$$f(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda_2}{2} \|w\|^2 + \lambda_1 \|w\|_1$$

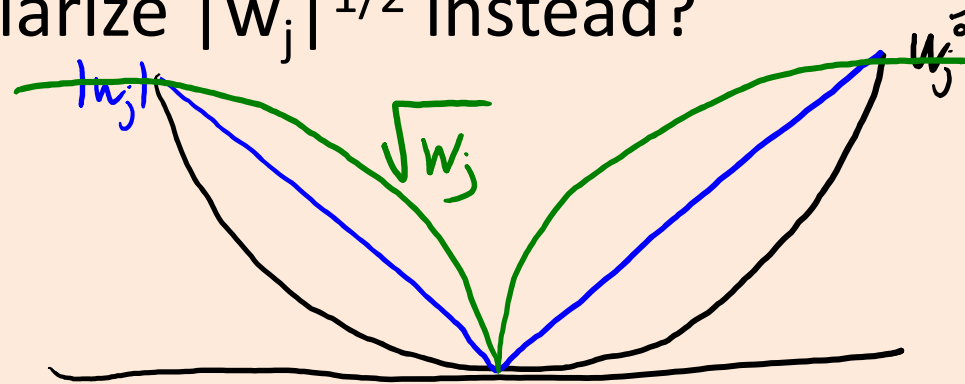
- Called “**elastic net**” regularization.
  - Solution is **sparse and unique**.
  - Slightly better with feature dependence:
    - Selects both “mom” and “mom2”.
- Optimization is easier though still non-differentiable.

# L1-Regularization Debiasing and Filtering

- To remove **false positives**, some authors add a **debiasing step**:
  - Fit 'w' using L1-regularization.
  - Grab the non-zero values of 'w' as the “relevant” variables.
  - Re-fit relevant 'w' using least squares or L2-regularized least squares.
- A related use of L1-regularization is as a **filtering method**:
  - Fit 'w' using L1-regularization.
  - Grab the non-zero values of 'w' as the “relevant” variables.
  - Run standard (slow) variable selection restricted to relevant variables.
    - Forward selection, exhaustive search, stochastic local search, etc.

# Non-Convex Regularizers

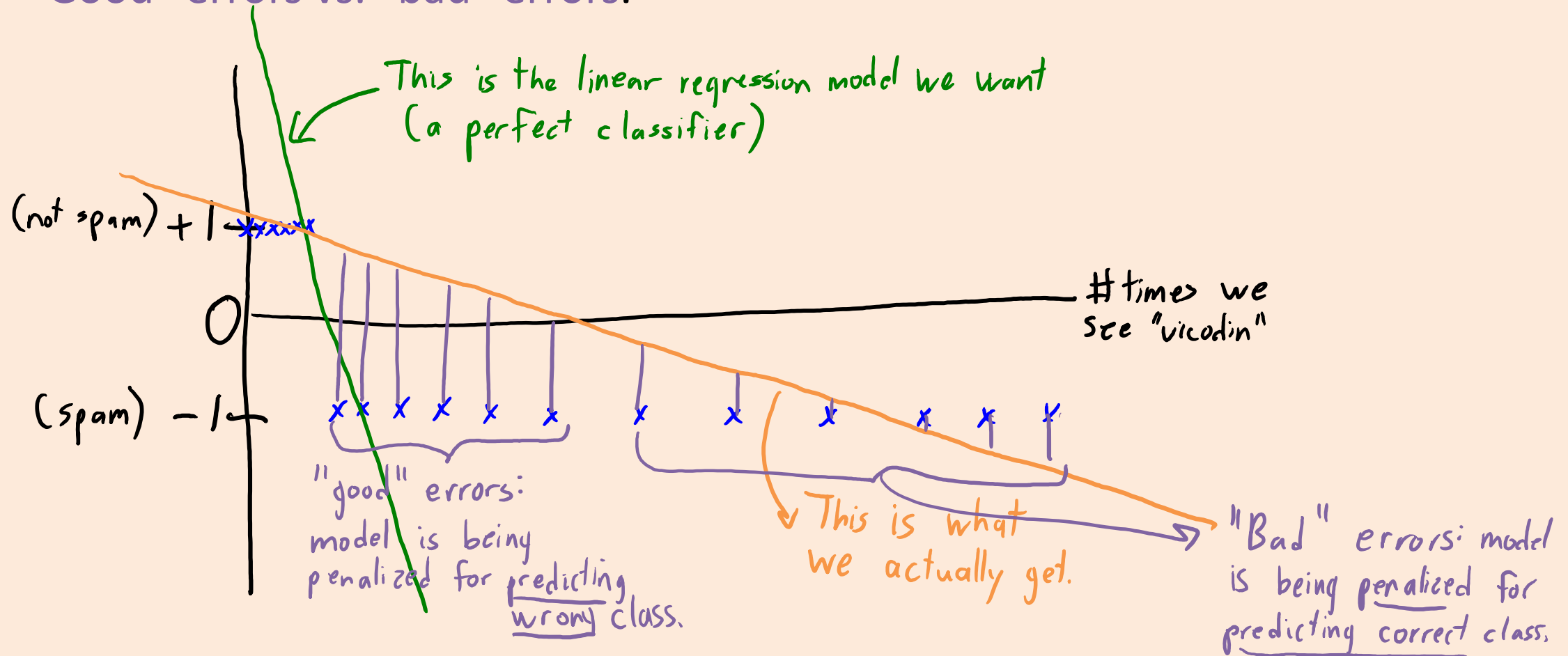
- Regularizing  $|w_j|^2$  selects **all features**.
- Regularizing  $|w_j|$  selects fewer, but still has many **false positives**.
- What if we regularize  $|w_j|^{1/2}$  instead?



- Minimizing this objective would lead to **fewer false positives**.
  - Less need for debiasing, but it's not convex and **hard to minimize**.
- There are many non-convex regularizers with similar properties.
  - L1-regularization is (basically) the “most sparse” convex regularizer.

# Can we just use least squares??

- What went wrong?
  - “Good” errors vs. “bad” errors.

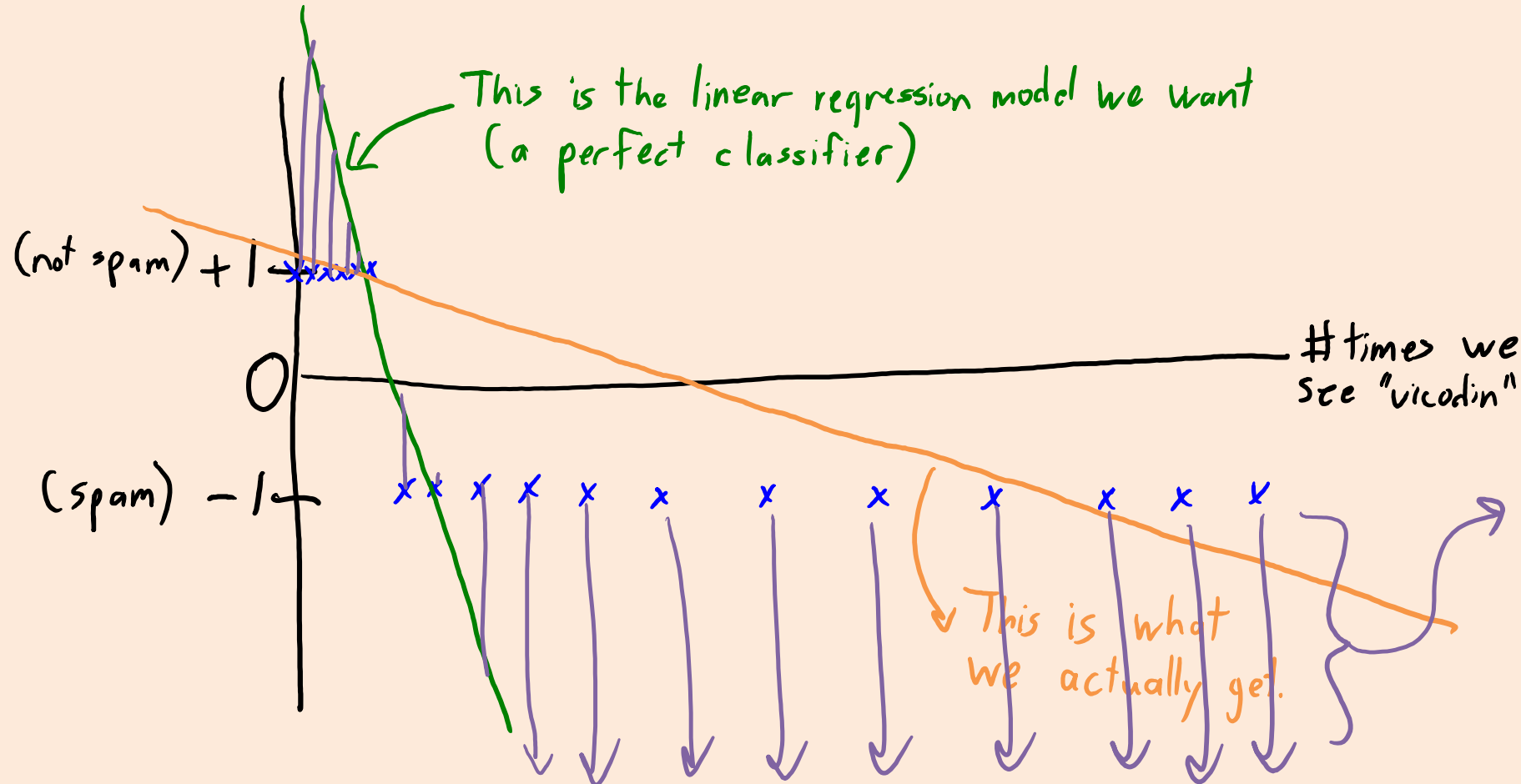


# Can we just use least squares??

- What went wrong?
  - “Good” errors vs. “bad” errors.

$$f(w) = \sum_{i=1}^n (w^T x_i - y_i)^2$$

What happens if  $y_i = -1$  and  $w^T x_i = -1000$ ?



This is the linear regression model we want (a perfect classifier)

This is what we actually get.

“Bad” errors of the perfect linear classifier are HUGE.

# Online Classification with Perceptron

- **Perceptron for online linear binary classification** [Rosenblatt, 1957]
  - Start with  $w_0 = 0$ .
  - At time 't' we receive features  $x_t$ .
  - We **predict**  $\hat{y}_t = \text{sign}(w_t^T x_t)$ .
  - If  $\hat{y}_t \neq y_t$ , then set  $w_{t+1} = w_t + y_t x_t$ .
    - Otherwise, set  $w_{t+1} = w_t$ .

(Slides are old so above I'm using subscripts of 't' instead of superscripts.)

- **Perceptron mistake bound** [Novikoff, 1962]:
  - Assume data is **linearly-separable** with a "margin":
    - There exists  $w^*$  with  $\|w^*\| = 1$  such that  $\text{sign}(x_t^T w^*) = \text{sign}(y_t)$  for all 't' and  $|x_t^T w^*| \geq \gamma$ . > 0
  - Then the **number of total mistakes is bounded**.
    - No requirement that data is IID.



# Perceptron Mistake Bound

- Let's **normalize each  $x_t$**  so that  $\|x_t\| = 1$ .
  - Length doesn't change label.
- Whenever we make a mistake, we have  $\text{sign}(y_t) \neq \text{sign}(w_t^T x_t)$  and

$$\begin{aligned}\|w_{t+1}\|^2 &= \|w_t + yx_t\|^2 \\ &= \|w_t\|^2 + 2 \underbrace{y_t w_t^T x_t}_{<0} + 1 \\ &\leq \|w_t\|^2 + 1 \\ &\leq \|w_{t-1}\|^2 + 2 \\ &\leq \|w_{t-2}\|^2 + 3.\end{aligned}$$

- So **after 'k' errors we have  $\|w_t\|^2 \leq k$** .

# Perceptron Mistake Bound

- Let's consider a solution  $w^*$ , so  $\text{sign}(y_t) = \text{sign}(x_t^T w^*)$ .
  - And let's choose a  $w^*$  with  $\|w^*\| = 1$ ,
- Whenever we make a mistake, we have:

$$\begin{aligned}\|w_{t+1}\| &= \|w_t + y_t x_t\| \\ &\geq w_t^T w_* + y_t x_t^T w_* \\ &= w_t^T w_* + |x_t^T w_*| \\ &\geq w_t^T w_* + \gamma.\end{aligned}$$

- Note:  $w_t^T w_* \geq 0$  by induction (starts at 0, then at least as big as old value plus  $\gamma$ ).
- So after 'k' mistakes we have  $\|w_t\| \geq \gamma k$ .

# Perceptron Mistake Bound

- So our two bounds are  $\|w_t\| \leq \sqrt{k}$  and  $\|w_t\| \geq \gamma k$ .
- This gives  $\gamma k \leq \sqrt{k}$ , or a **maximum of  $1/\gamma^2$  mistakes**.
  - Note that  $\gamma > 0$  by assumption and is upper-bounded by one by  $\|x\| \leq 1$ .
  - After this 'k', under our assumptions we're guaranteed to have a perfect classifier.

# Hinge-Loss Perceptron

- A perceptron-like algorithm for minimizing the hinge loss:
  - Start with any  $w^0$ .
  - Go through examples until you find an example with  $y_i w^T x_i > 1$ .
    - Set  $w^{t+1} = w^t + \frac{1 - y_i (w^t)^T x_i}{x_i^T x_i} y_i x_i$  (minimum change to  $w_t$  that satisfies constraint).
- If a classifier with hinge loss of 0 exists, this converges to one.
  - Looks like perceptron, but with a step size added to update (green term).
    - Get perceptron algorithm if you replace green term with '1'.
  - A special case of the “projection onto convex sets” (POCS) algorithm.