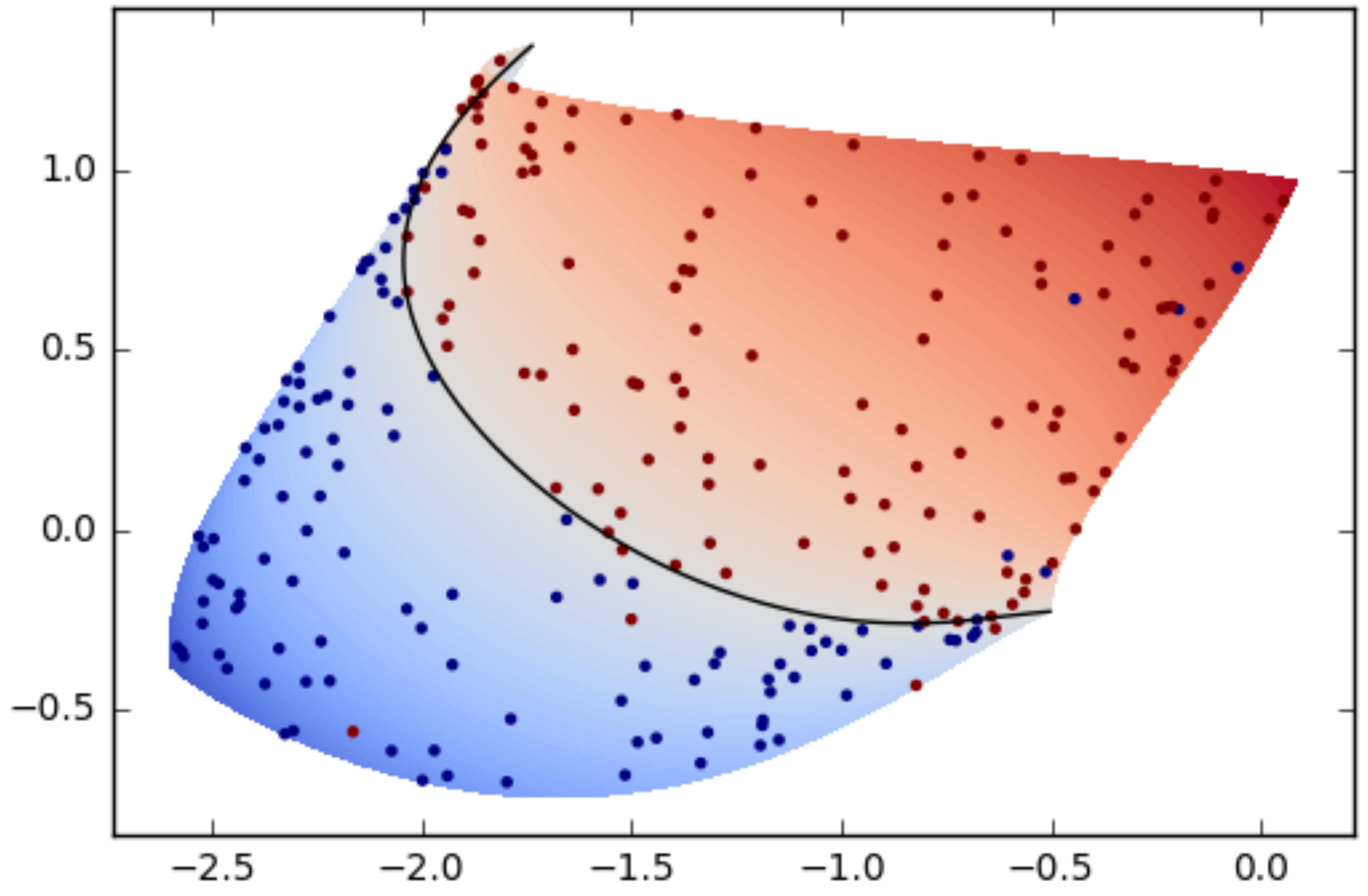
CS 273A: Machine Learning

Fall 2021

Lecture 9: Logistic Regression

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

All slides in this course adapted from Alex Ihler & Sameer Singh



Logistics

assignments

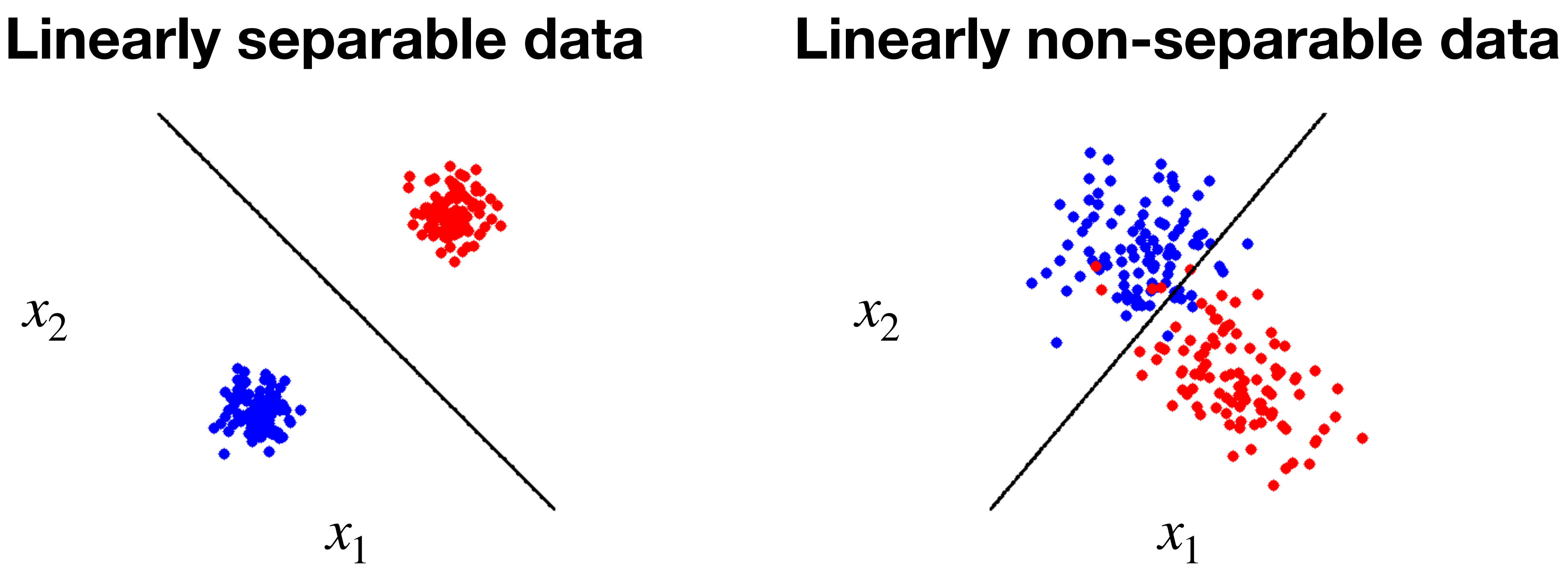
- Assignment 3 due next Tuesday, Oct 26

midterm

- Midterm exam on Nov 4, 11am–12:20 in **SH 128**
- If you're eligible to be remote — let us know by Oct 28
- If you're eligible for more time — let us know by Oct 28
- Review during lecture next Thursday

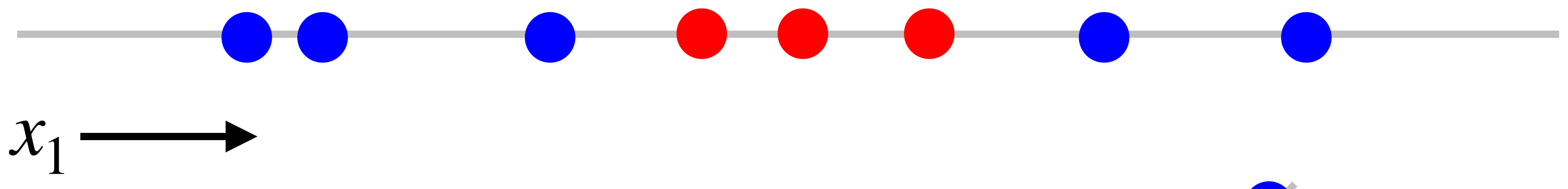
Separability

- Separable dataset = there's a model (in our class) with perfect prediction
- Separable problem = there's a model with 0 test loss $\mathbb{E}_{x,y \sim p}[\ell(y, \hat{y}(x))] = 0$
  - Also called realizable
- Linearly separable = separable by a linear classifiers (hyperplanes)

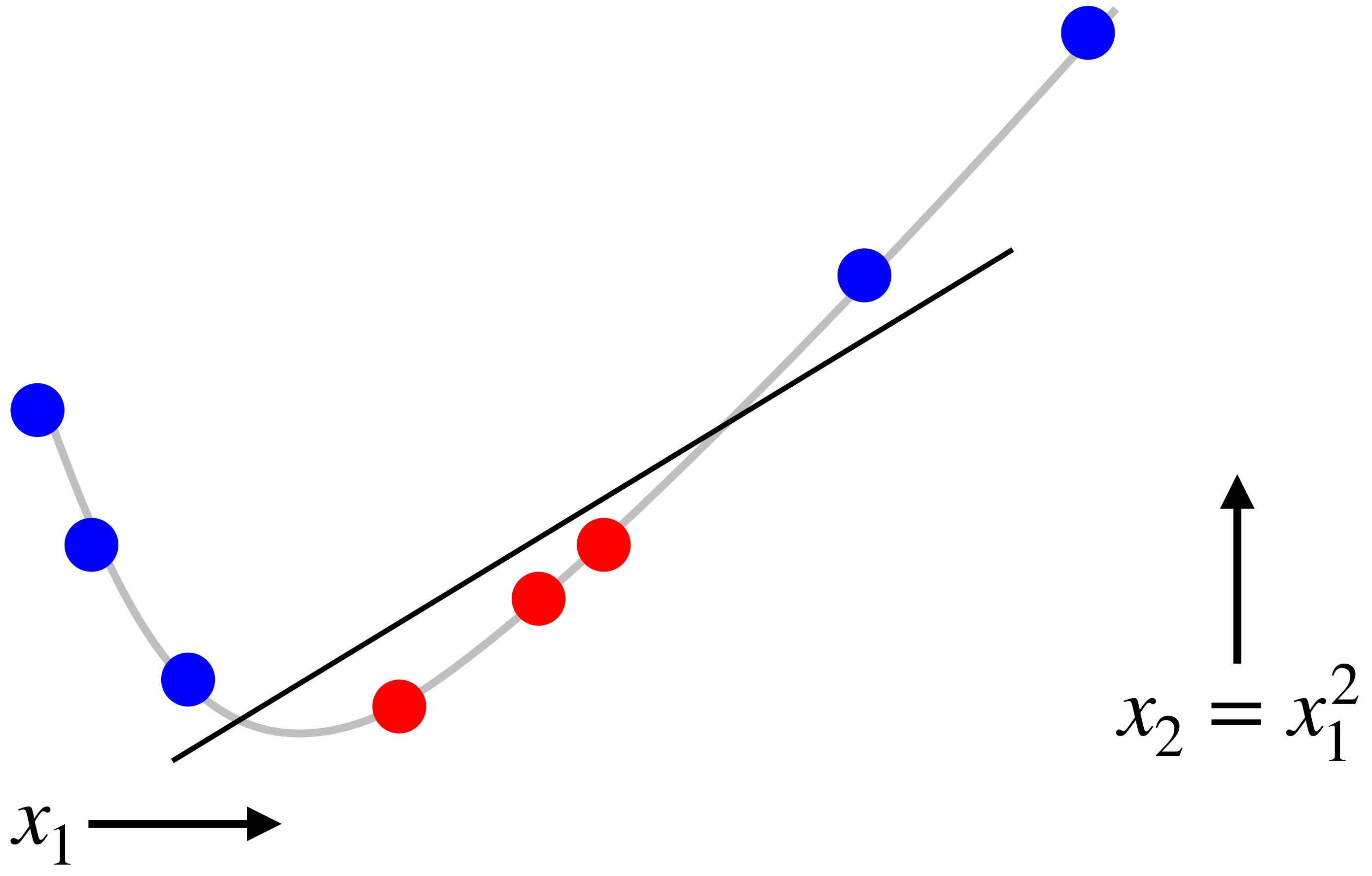


Adding features

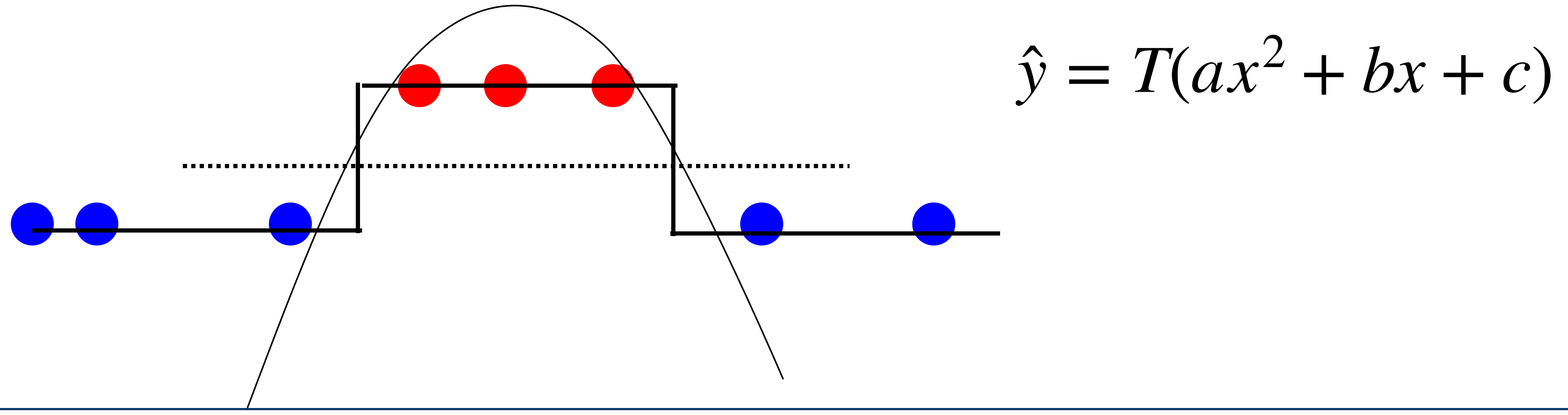
- How to make the perceptron more expressive?
  - Add features — recall linear → polynomial regression
- Linearly non-separable:



- Linearly separable in quadratic features:



- Visualized in original feature space:



  - Decision boundary: $ax^2 + bx + c = 0$

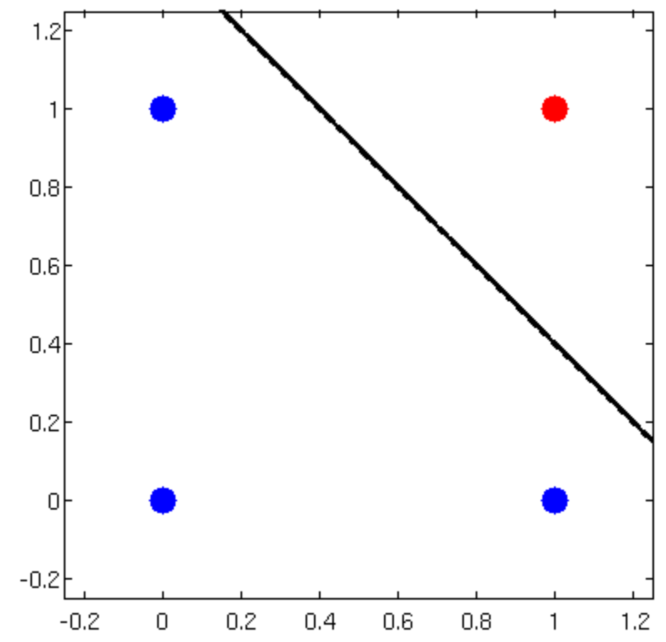
Adding features

- Which functions do we need to represent the decision boundary?
  - When linear functions aren't sufficiently expressive
  - Perhaps quadratic functions are

$$ax_1^2 + bx_1 + cx_2^2 + dx_2 + ex_1x_2 + f = 0$$

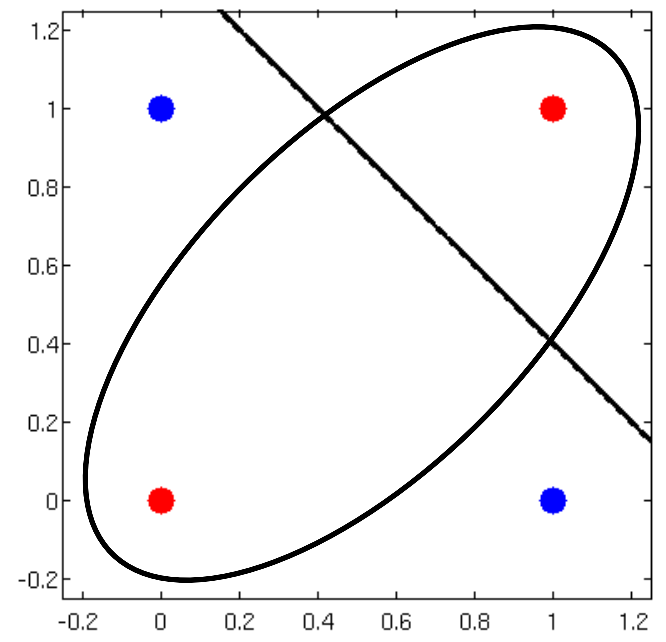
AND

x_1	x_2	y
0	0	-1
0	1	-1
1	0	-1
1	1	1



XOR

x_1	x_2	y
0	0	-1
0	1	1
1	0	1
1	1	-1



Separability in high dimension

- As we add more features → dimensionality of instance x increases:
  - Separability becomes easier: more parameters, more models that could separate
  - Add enough (good) features, and even a linear classifier can separate
    - Given a decision boundary $f(x) = 0$: add $f(x)$ as a feature → linearly separable!
- However:
  - Do these features explain test data or just training data?
  - Increasing model complexity can lead to overfitting

Recap

- Perceptron = linear classifier
  - Linear response → step decision function → discrete class prediction
  - Linear decision boundary
- Separability = existence of a perfect model (in the class)
  - Separable data: 0 loss on this data
  - Separable problem: 0 loss on the data <u>distribution</u>
  - Perceptron: linear separability
- Adding features:
  - Complex features: complex decision boundary, easier separability
  - Can lead to overfitting

Today's lecture

Learning perceptrons

Logistic regression

Multi-class classifiers

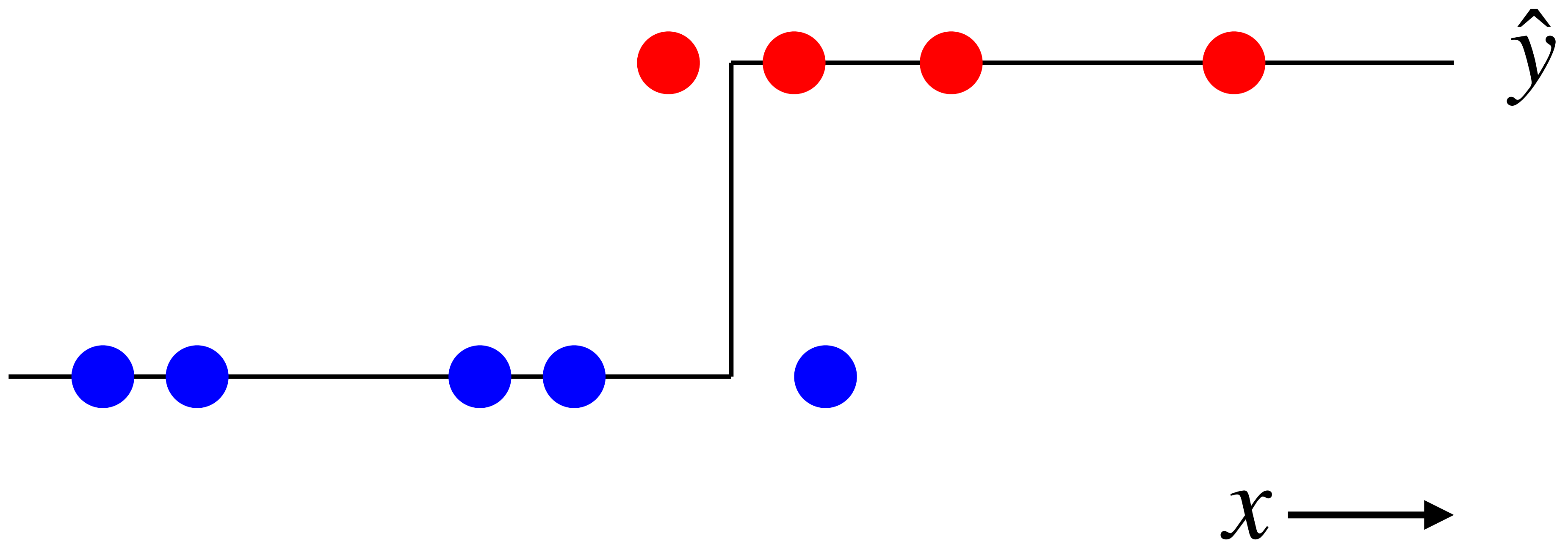
VC dimension

Learning a perceptron

- What do we need to learn the parameters θ of a perceptron?
  - Training data $\mathscr{D}$ = labeled instances
  - Loss function $\mathscr{L}_\theta$ = error rate on labeled data
  - Optimization algorithm = method for minimizing training loss

Error rate

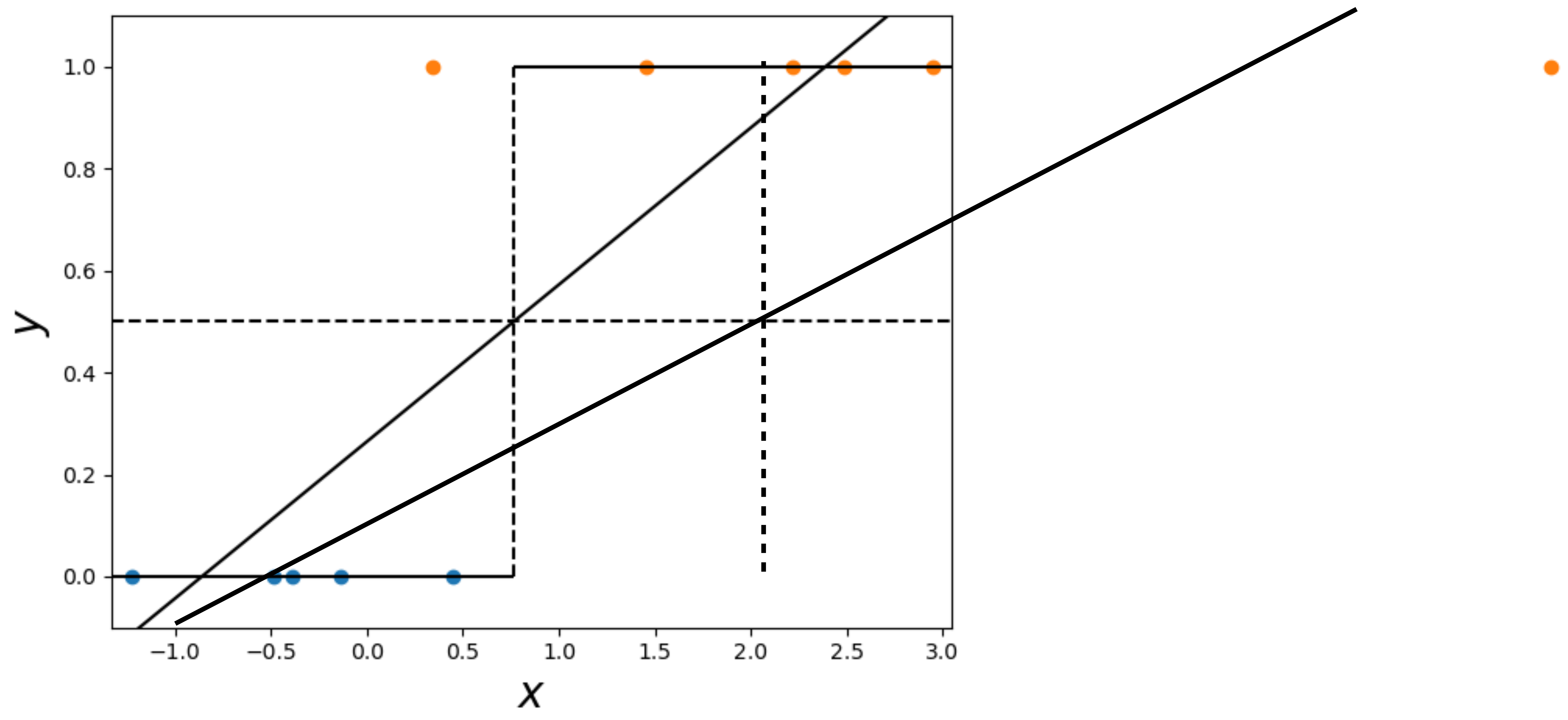
- Error rate: $\mathscr{L}_\theta = \frac{1}{m}\sum_i \delta(y^{(i)} \neq f_\theta(x^{(i)}))$
  - With the indicator $\delta(y \neq \hat{y}) = \begin{cases} 1 & y \neq \hat{y} \\ 0 & else \end{cases}$



$\mathscr{L}_\theta = \frac{2}{9}$

Use linear regression?

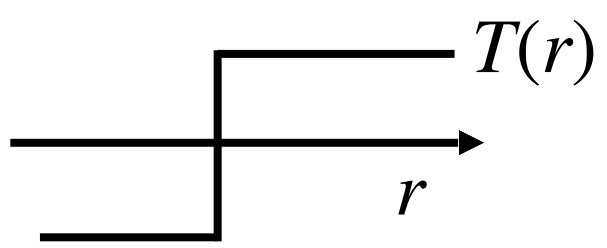
- Idea: find θ using linear regression



- Affected by large regression losses
  - We only care about the <u>classification</u> loss

Perceptron: gradient-based learning

- Problem: loss function not differentiable $\mathscr{L}_\theta(x, y) = \delta(y \neq \text{sign}(\theta^\intercal x))$
  - Write differently: $\mathscr{L}_\theta(x, y) = \frac{1}{4}(y - \text{sign}(\theta^\intercal x))^2$
    - $\nabla_\theta \text{sign}(\theta^\intercal x) = 0$ almost everywhere



  - But we also don't want MSE = $\mathscr{L}_\theta(x, y) = \frac{1}{2}(y - \theta^\intercal x)^2$
  - Compromise: $\mathscr{L}_\theta(x, y) = (y - \text{sign}(\theta^\intercal x))(y - \theta^\intercal x)$
    - $\nabla_\theta \mathscr{L}_\theta = -(y - \text{sign}(\theta^\intercal x))x = -(y - \hat{y})x$

Perceptron training algorithm

while $\neg$ done:

for each data point j:

$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$ **predict output for point j**

$\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}^{(j)})x^{(j)}$ **gradient step on weird loss**

- Similar to linear regression with MSE loss
  - Except that $\hat{y}$ is the class prediction, not the linear response
  - No update for correct predictions $y^{(j)} = \hat{y}^{(j)}$
  - For incorrect predictions: $y^{(j)} - \hat{y}^{(j)} = \pm 2$
    - $\implies$ update towards x (for false negative) or $-x$ (for false positive)

Perceptron training algorithm

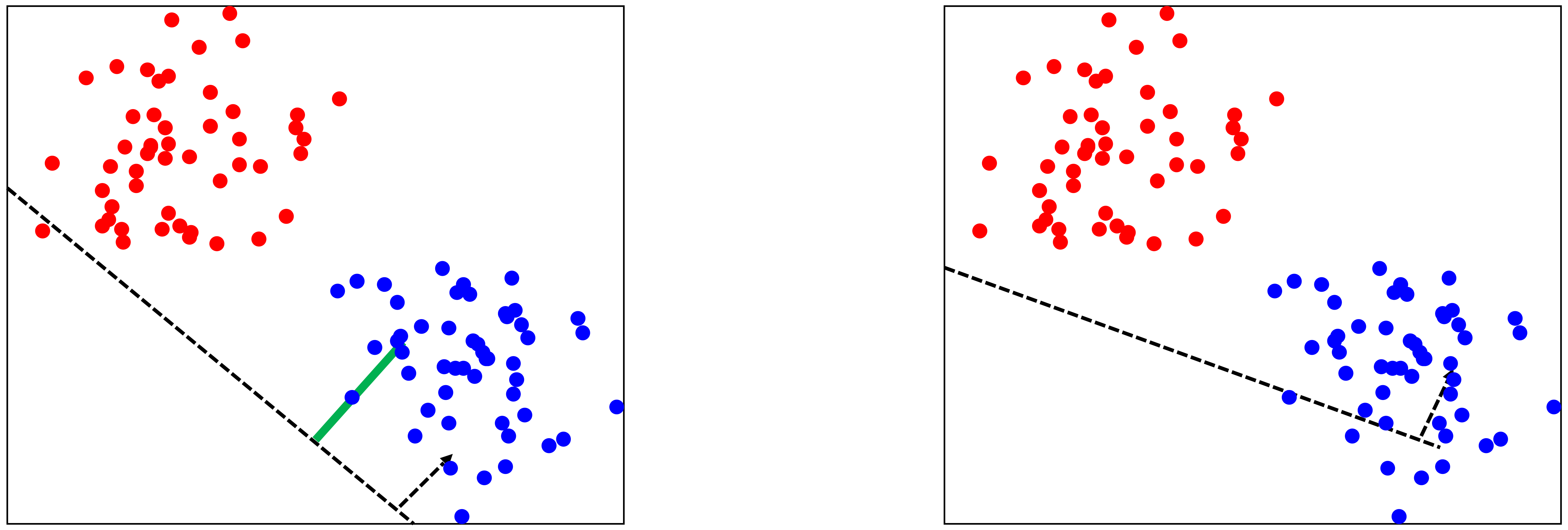
while $\neg$ done:

for each data point j:

$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$ **predict output for point j**

$\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}^{(j)})x^{(j)}$ **gradient step on weird loss**

incorrect prediction: update weights



Perceptron training algorithm

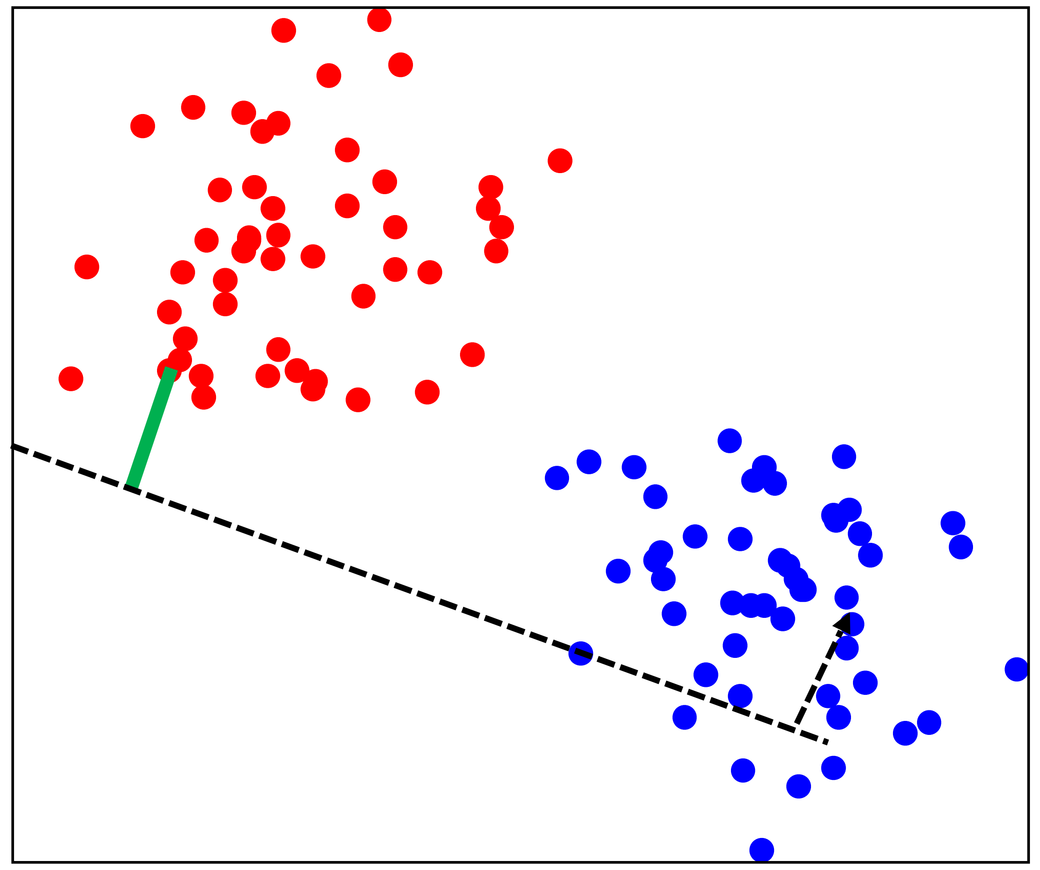
while $\neg$ done:

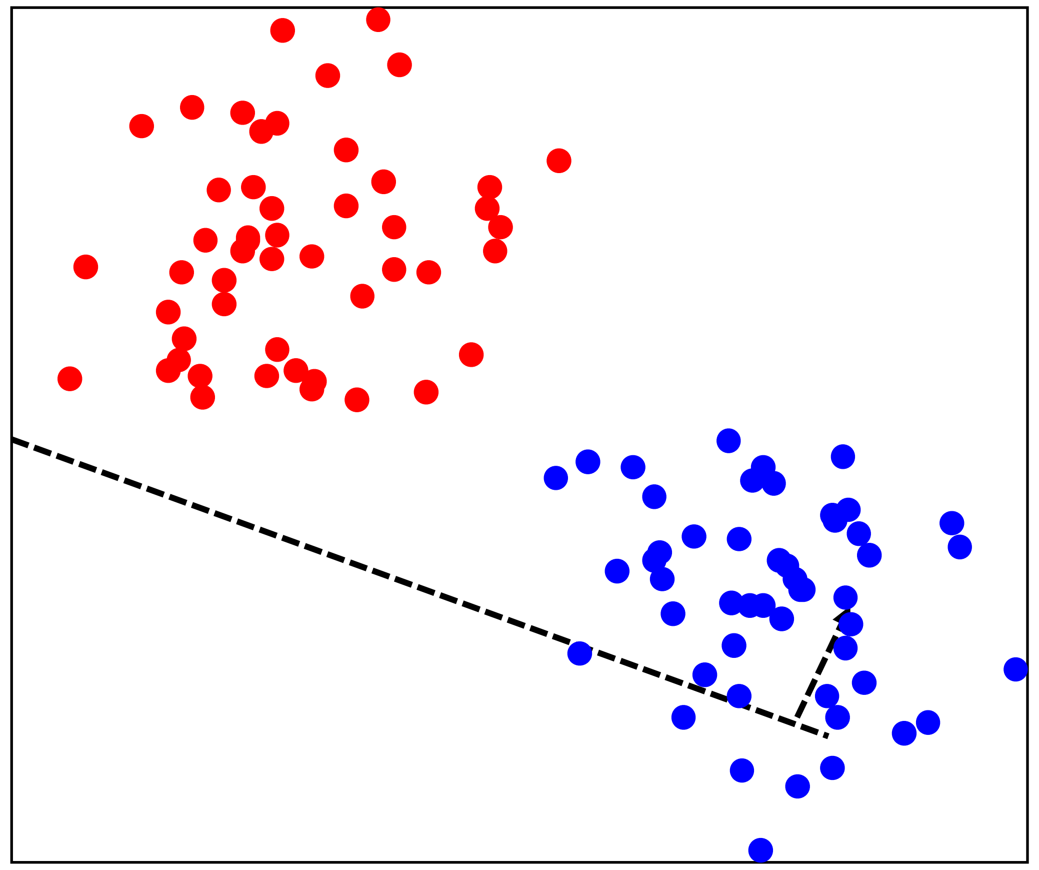
for each data point j:

$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$ **predict output for point j**

$\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}^{(j)})x^{(j)}$ **gradient step on weird loss**

correct prediction: no update





Perceptron training algorithm

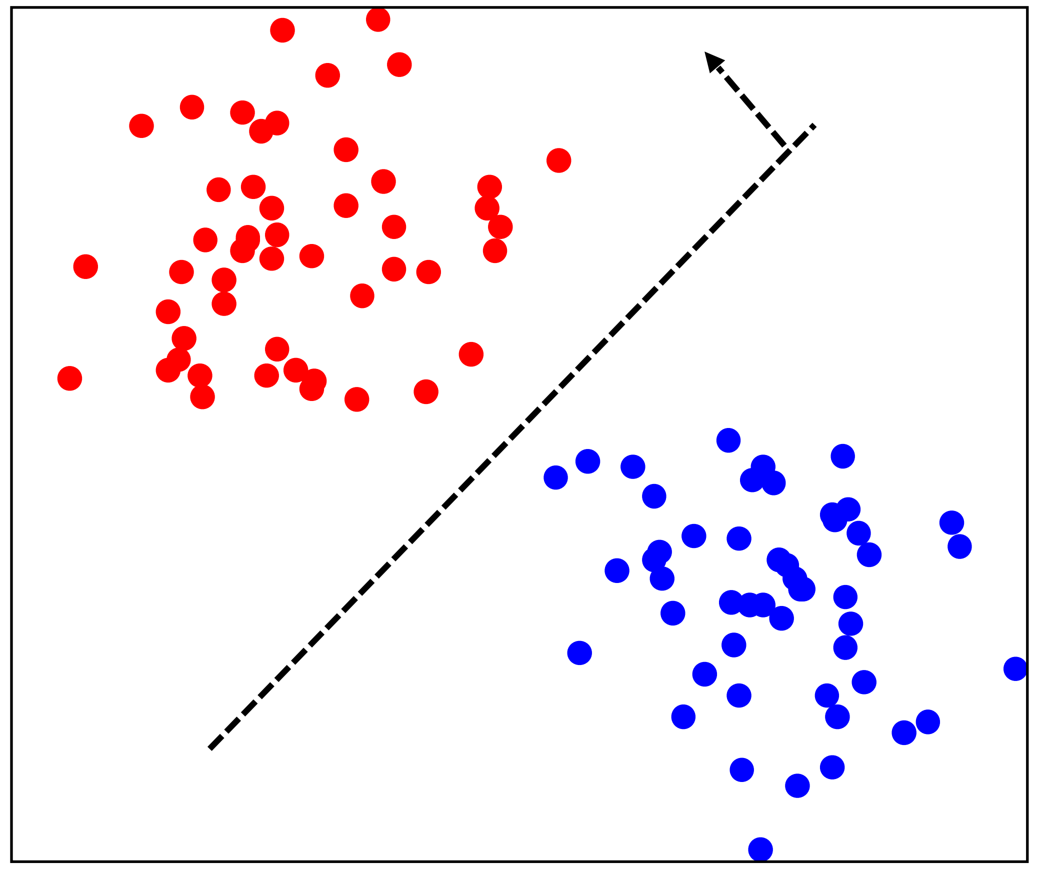
while $\neg$ done:

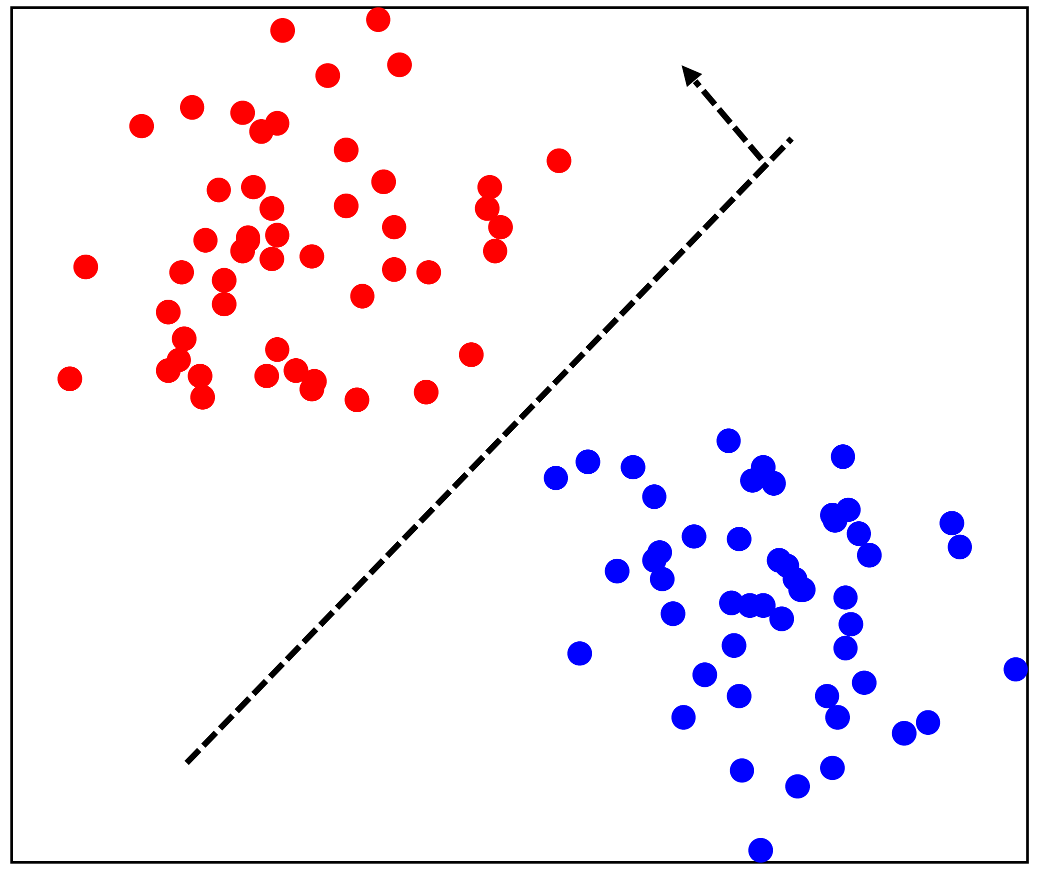
for each data point j:

$$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$$ **predict output for point j**

$$\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}^{(j)})x^{(j)}$$ **gradient step on weird loss**

convergence: no more updates





Today's lecture

Learning perceptrons

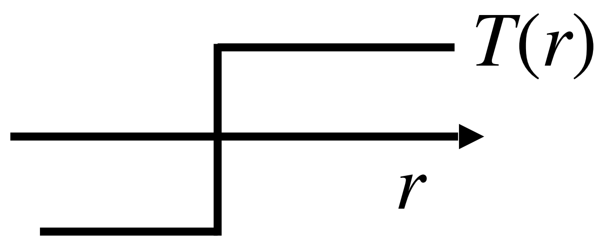
Logistic regression

Multi-class classifiers

VC dimension

Perceptron

- Perceptron = linear classifier
  - Parameters θ = weights (also denoted w)
  - Response = weighted sum of the features $r = \theta^\intercal x$
  - Prediction = thresholded response $\hat{y}(x) = T(r) = T(\theta^\intercal x)$



  - Decision function: $\hat{y}(x) = \begin{cases} +1 & \text{if } \theta^\intercal x > 0 \\ -1 & \text{otherwise} \end{cases}$ (for $T(r) = \text{sign}(r)$)
- Update rule: $\theta \leftarrow \theta - \alpha(\underbrace{y - \hat{y}}_{\text{error}})x$

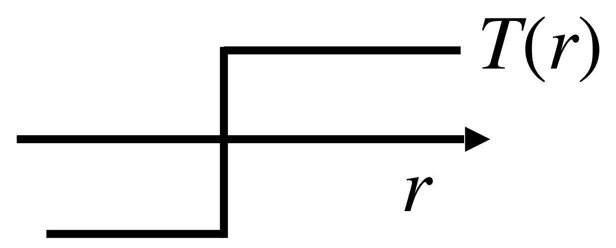
Widening the classification margin

- Which decision boundary is "better"?
  - Both have 0 training loss
  - But one seems more robust, expected to generalize better

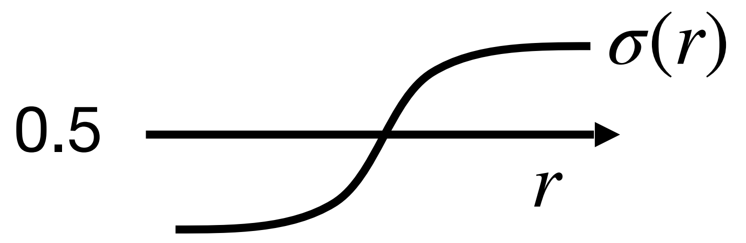
- Benefit of smooth loss function: care about margin
  - Encourage distancing the boundary from data points

Surrogate loss functions

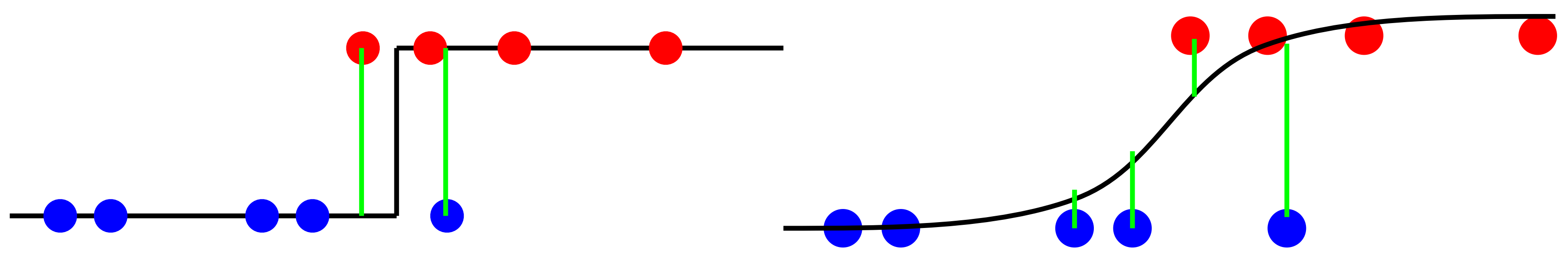
- Alternative: use differentiable loss function
    - E.g., approximate step function with smooth sigmoid function (= ”looks like s”)
    - Popular choice: logistic / sigmoid function $\sigma(x) = \frac{1}{1 + \exp(-x)} \in [0,1]$



    - For this part, let's assume $y \in \{0,1\}$



- MSE loss: $\mathcal{L}_\theta(x, y) = (y - \sigma(r(x)))^2$



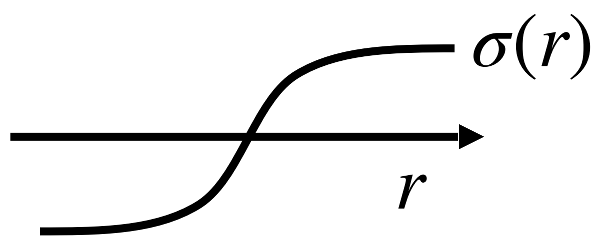
    - Far from the boundary: $\sigma \approx 0$ or 1, loss approximates 0–1 loss
    - Near the boundary: $\sigma \approx \frac{1}{2}$, loss near $\frac{1}{4}$, but clear improvement direction

Learning smooth linear classifiers

- Use gradient-based optimizer on the loss $\mathscr{L}_\theta(x,y) = (y - \sigma(\theta^\intercal x))^2$

$$-\nabla_\theta \mathscr{L}_\theta(x,y) = 2\underbrace{(y - \sigma(\theta^\intercal x))}_{\text{error}}\underbrace{\sigma'(\theta^\intercal x)}_{\text{sensitivity}}\, x$$

- Logistic function: $\sigma(r) = \dfrac{1}{1 + \exp(-r)}$

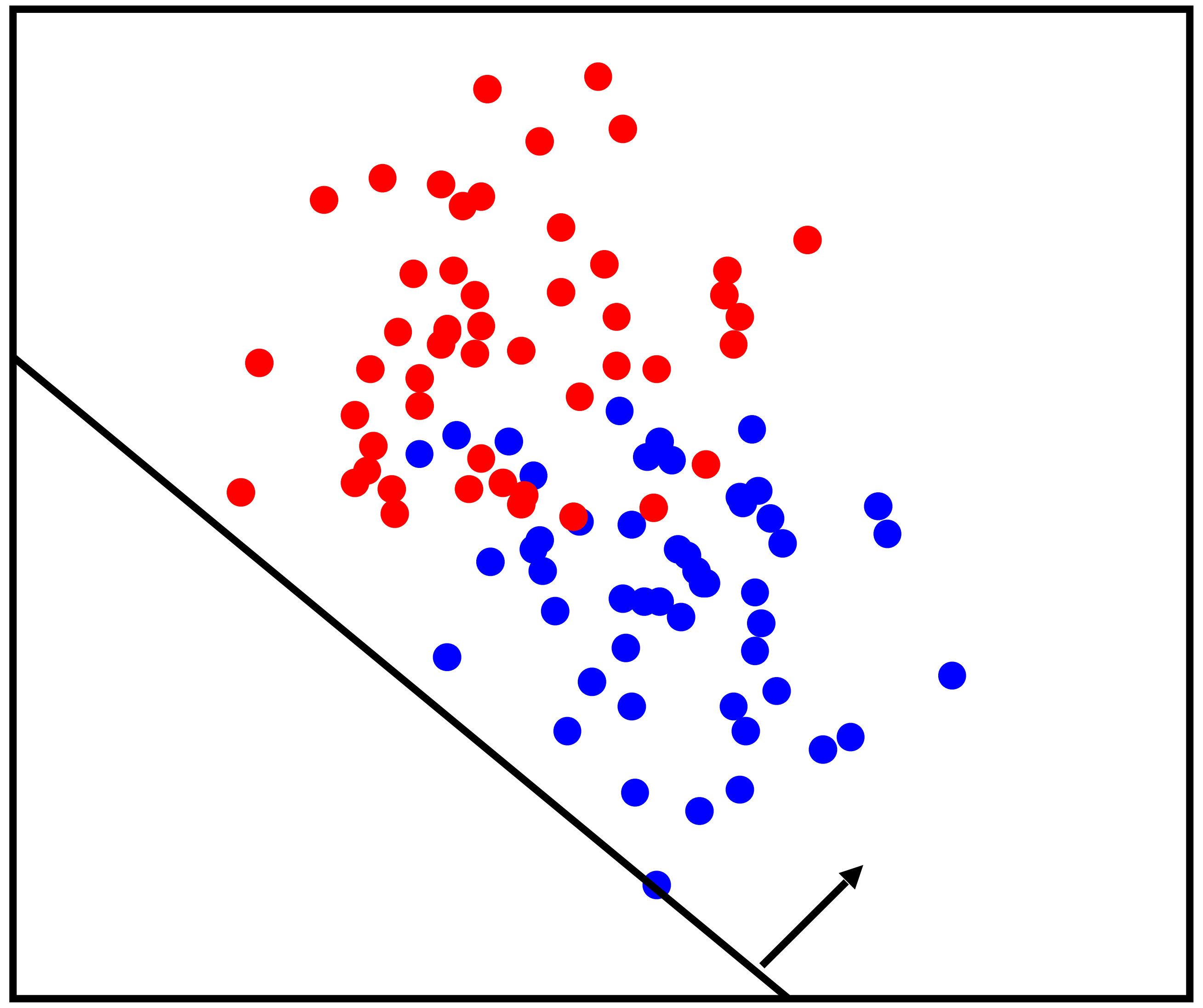


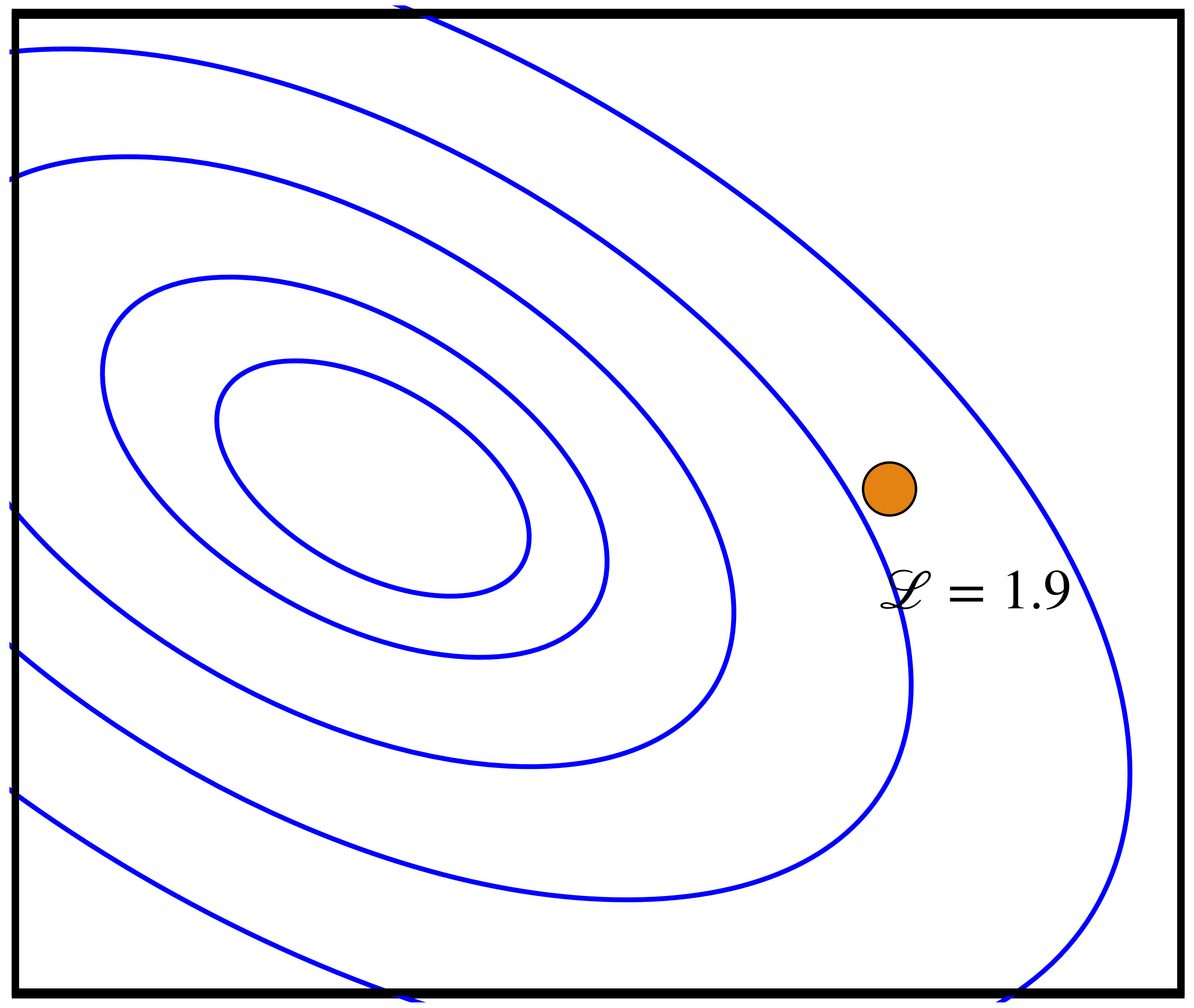
- It's derivative: $\sigma'(r) = \sigma(r)(1 - \sigma(r))$
  - Saturates for both $r \to \infty$, $r \to -\infty$
- Confidently correct prediction: $\sigma(r) \approx y \in \{0,1\} \implies \nabla_\theta \mathscr{L}_\theta \approx 0$ ← good
- Confidently incorrect prediction: $\sigma(r) \approx 1 - y \implies \nabla_\theta \mathscr{L}_\theta \approx 0$ ← bad

Learning smooth linear classifiers

- With a smooth loss function with can use Stochastic Gradient Descent

$$\mathscr{L}_\theta(x, y) = (y - \sigma(r(x)))^2$$

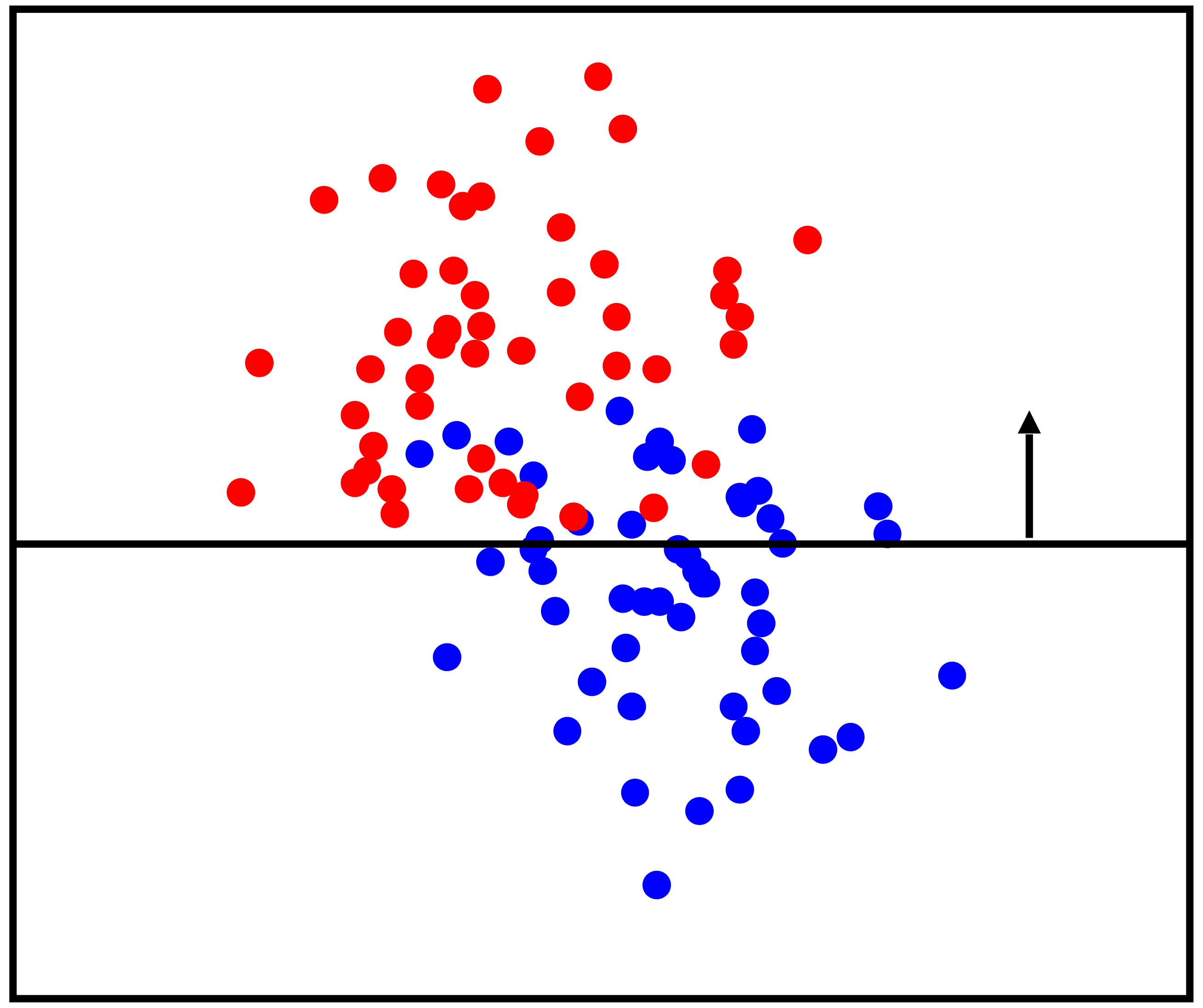


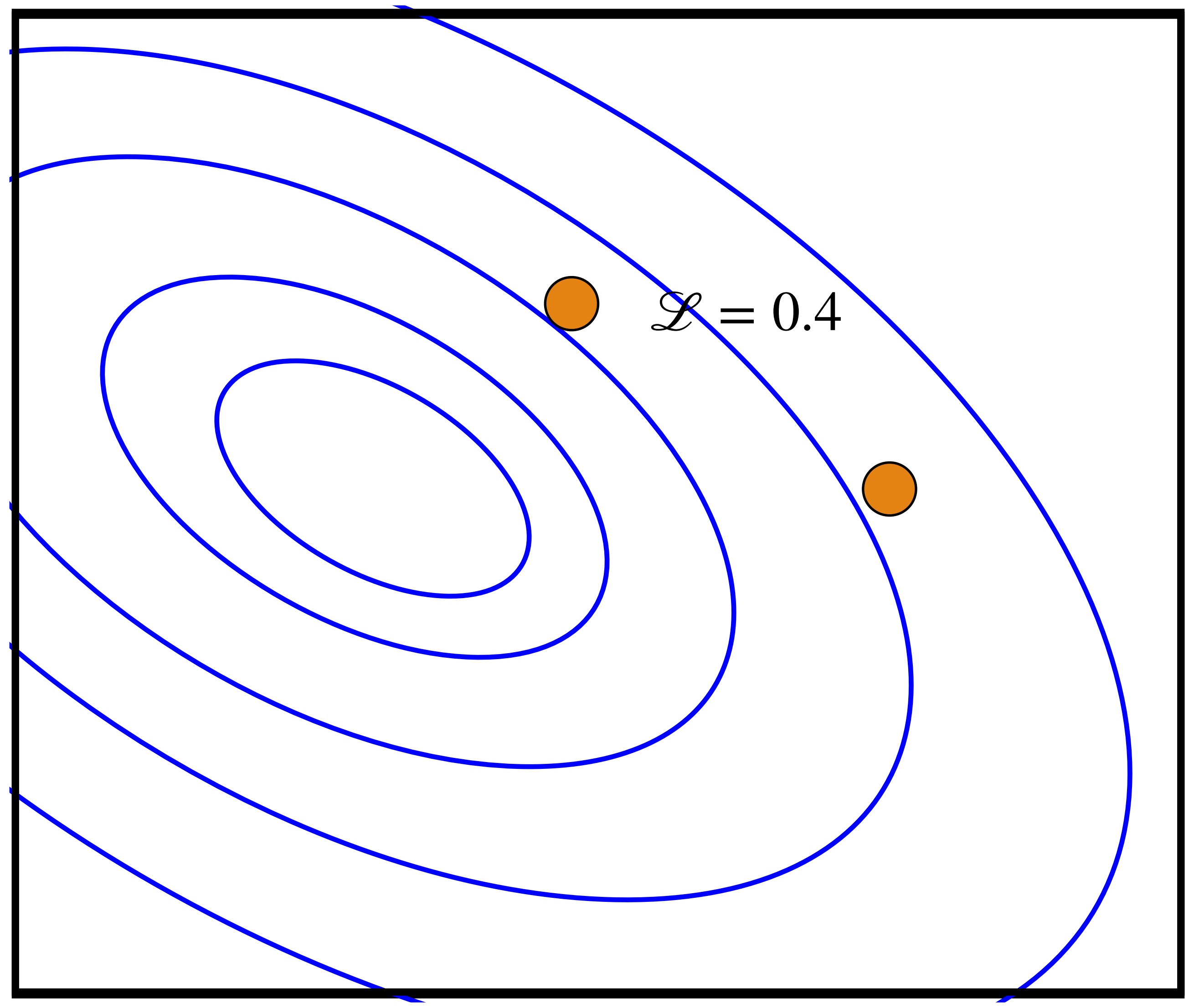


Learning smooth linear classifiers

- With a smooth loss function with can use Stochastic Gradient Descent

$$\mathscr{L}_\theta(x, y) = (y - \sigma(r(x)))^2$$

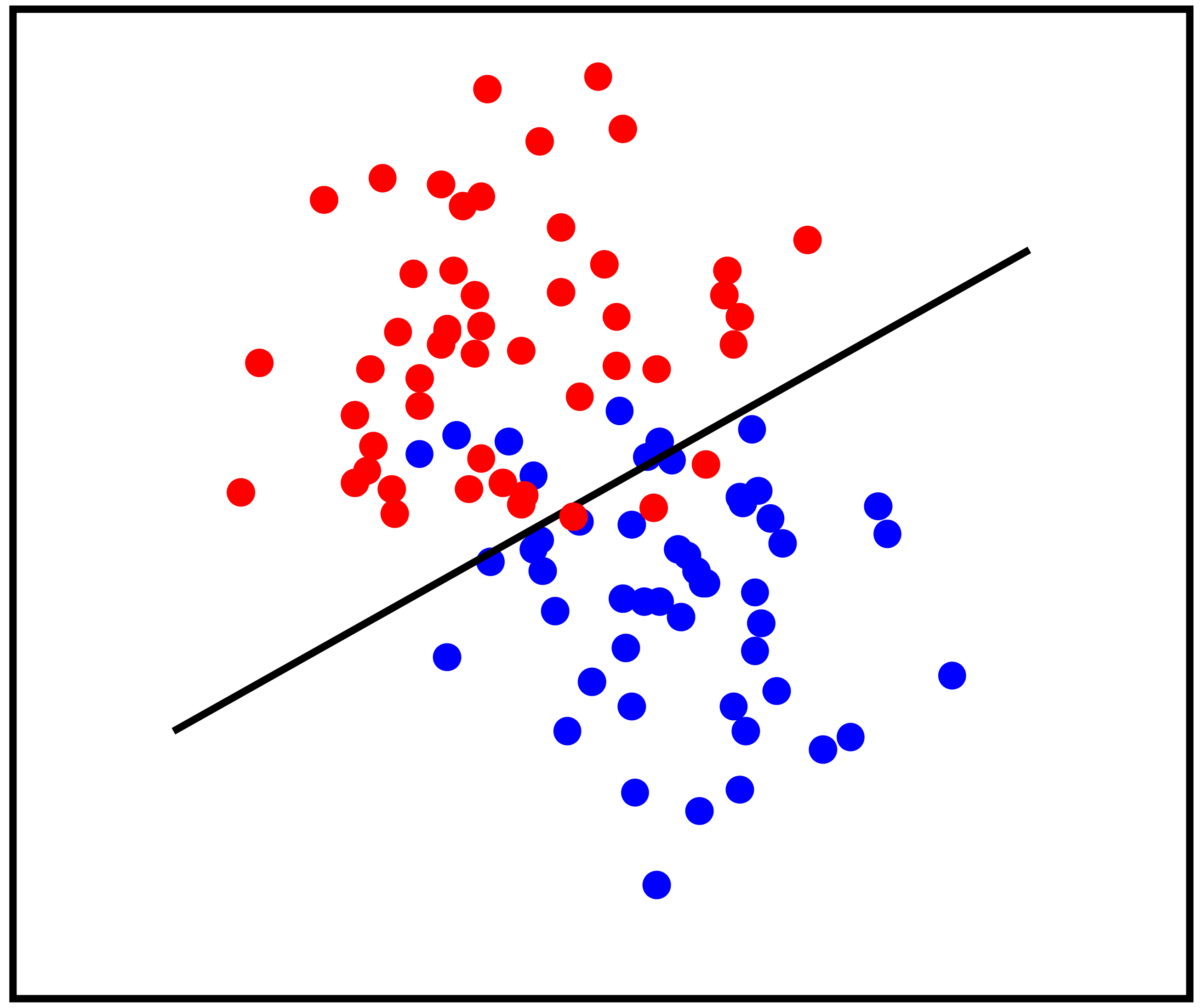


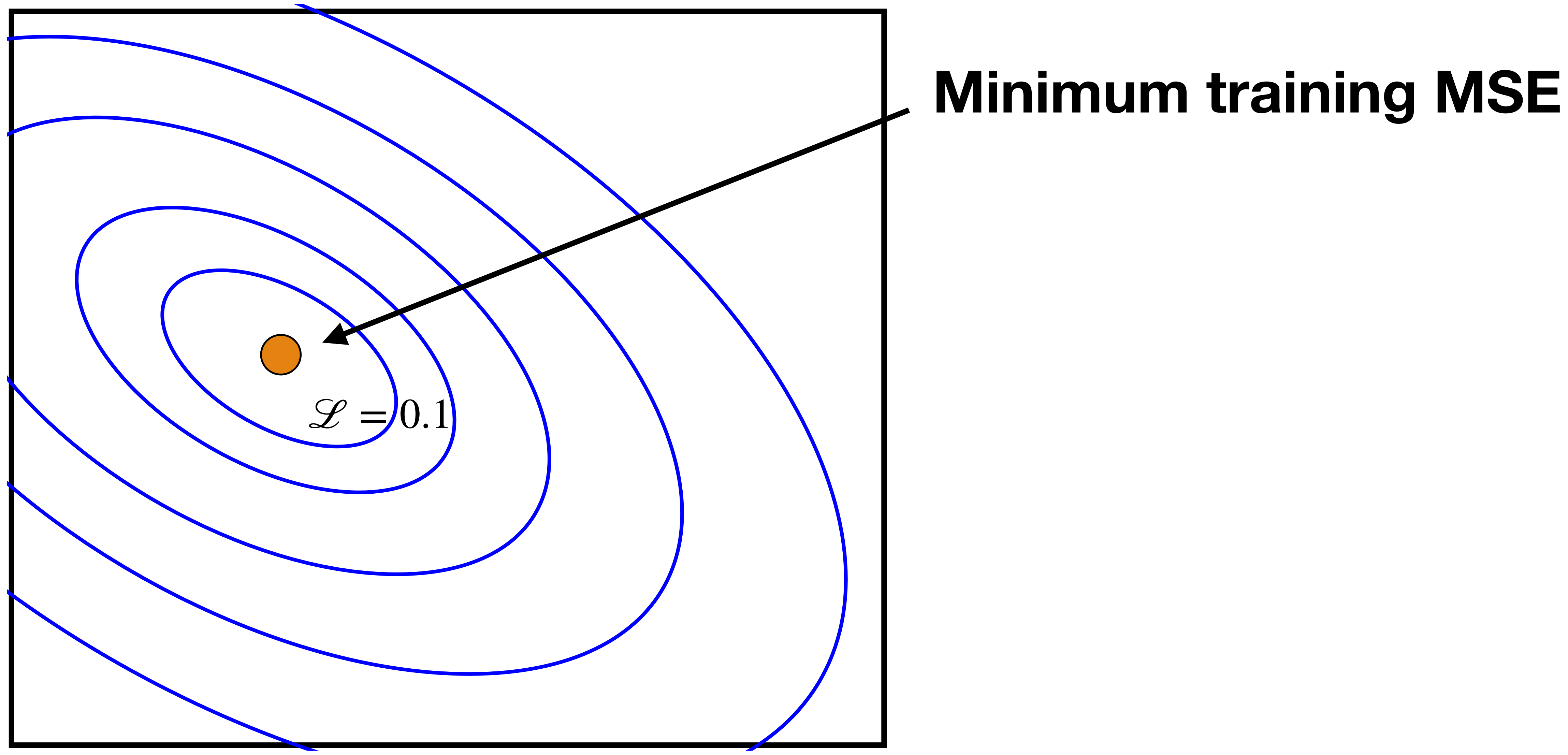


Learning smooth linear classifiers

- With a smooth loss function with can use Stochastic Gradient Descent

$$\mathcal{L}_\theta(x, y) = (y - \sigma(r(x)))^2$$





Maximum likelihood

- What if we had a probabilistic predictor $p_\theta(y|x)$?

- The better the parameter θ, the more likely the training data:

$$p_\theta(y^{(1)}, \ldots, y^{(m)} | x^{(1)}, \ldots, x^{(m)}) = \prod_j p_\theta(y^{(j)} | x^{(j)})$$

- Bayesian interpretation?

except, often there's no uniform distribution over parameter space

  - MAP: $\arg\max_\theta p(\theta|\mathscr{D}) = \arg\max_\theta p(\theta)p(\mathscr{D}_x)p_\theta(\mathscr{D}_y|\mathscr{D}_x) = \arg\max_\theta p_\theta(\mathscr{D}_y|\mathscr{D}_x)$

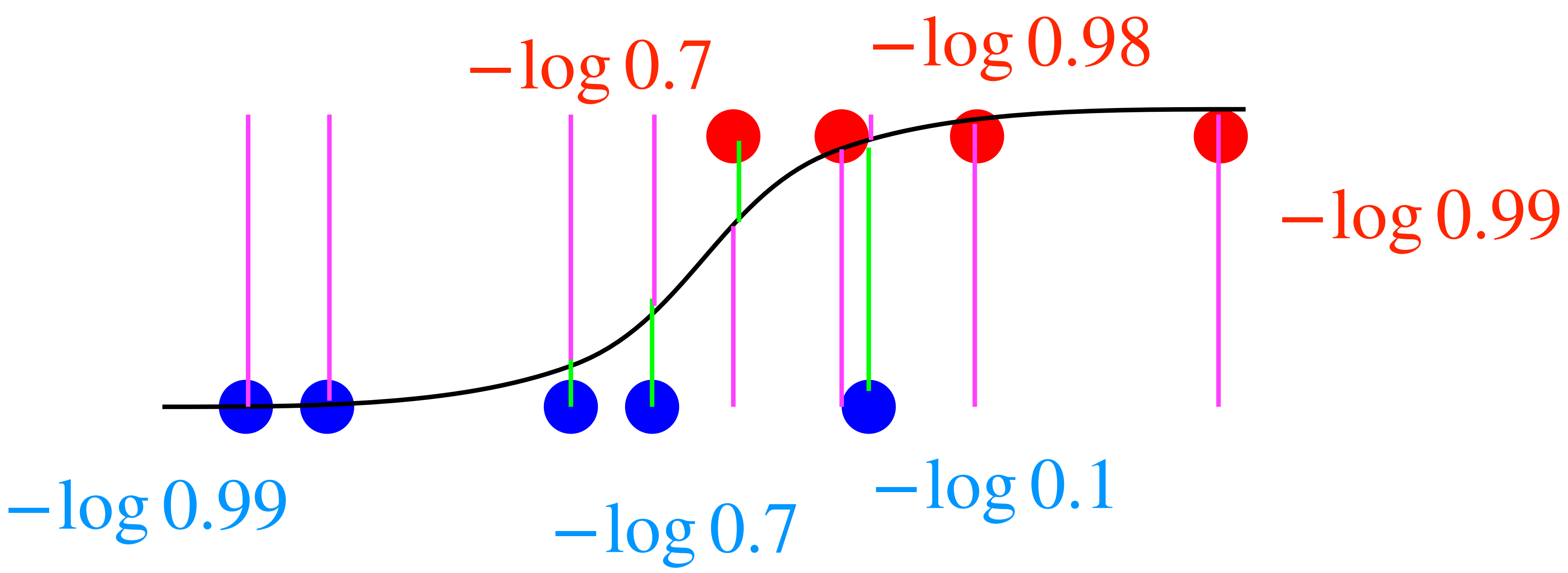
average over training dataset

- Maximum log-likelihood: $\max_\theta \frac{1}{m} \sum_j \log p_\theta(y^{(j)} | x^{(j)})$

Logistic Regression

- Can we turn a linear response into a probability? Sigmoid! $\sigma : \mathbb{R} \to [0,1]$
- Think of $\sigma(\theta^\intercal x) = p_\theta(y = 1 | x)$
- Negative Log-Likelihood (NLL) loss:

$$\mathscr{L}_\theta(x,y) = -\log p_\theta(y|x) = \underbrace{-y\log\sigma(\theta^\intercal x)}_{\text{for } y=1} - \underbrace{(1-y)\log(1-\sigma(\theta^\intercal x))}_{\text{for } y=0}$$



Logistic Regression: gradient

- Logistic NLL loss: $\mathcal{L}_\theta(x,y) = -y\log\sigma(\theta^\intercal x) - (1-y)\log(1-\sigma(\theta^\intercal x))$

- Gradient:

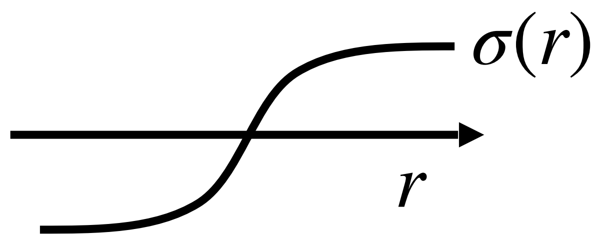
$$-\nabla_\theta \mathcal{L}_\theta(x,y) = \left(y\frac{\sigma'(\theta^\intercal x)}{\sigma(\theta^\intercal x)} - (1-y)\frac{\sigma'(\theta^\intercal x)}{1-\sigma(\theta^\intercal x)} \right) x$$

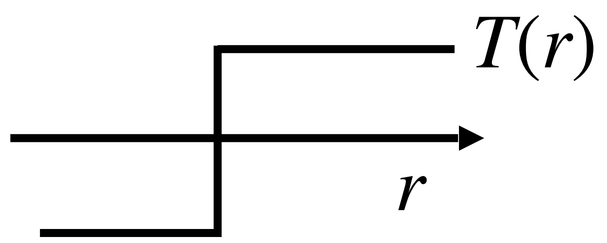
$$= (\underbrace{y\,(1-\sigma(\theta^\intercal x))}_{\text{error for } y=1} - (1-y)\underbrace{\sigma(\theta^\intercal x)}_{\text{error for } y=0})x$$

but update toward $-x$

$$= (y - p_\theta(y=1|x))x$$

- Compare:





  - Perceptron: $(y-\hat{y})x$ ⟵ constant error (± 2), insensitive to margin
  - Logistic MSE: $-\nabla_\theta \mathcal{L}_\theta(x,y) = 2(y-\sigma(\theta^\intercal x))\sigma'(\theta^\intercal x)x$ ⟵ 0 gradient for bad mistakes

Recap

- Linear classifiers:
  - Perceptron
  - Logistic classifier
- Measuring decision quality:
  - Error rate / 0–1 loss
  - MSE loss
  - Negative log-likelihood (Logistic Regression)
- Learning the weights
  - Perceptron algorithm — not quite gradient-based (or gradient of weird loss)
  - Gradient-based optimization of surrogate loss (MSE / NLL)

Today's lecture

Learning perceptrons

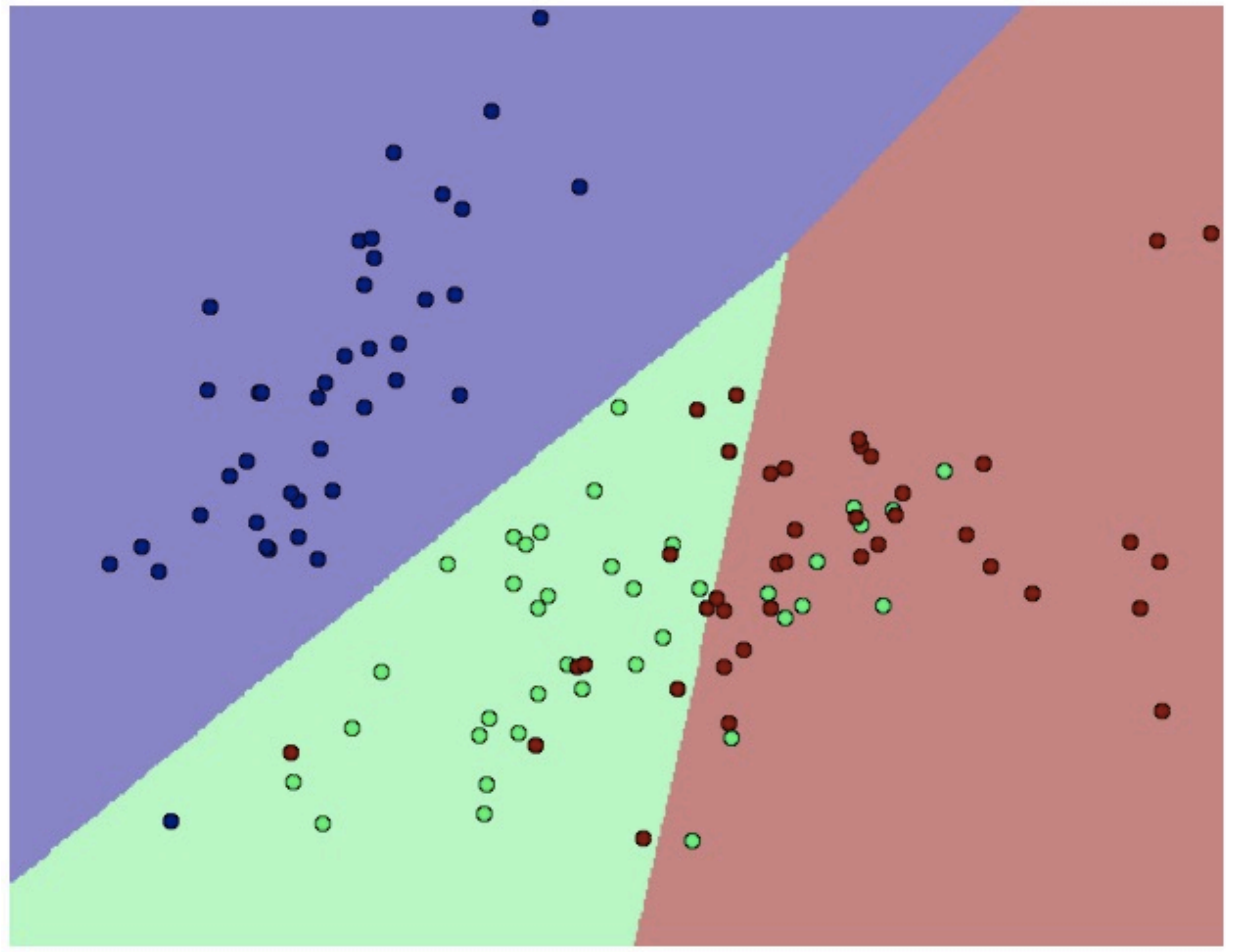
Logistic regression

Multi-class classifiers

VC dimension

Multi-class linear models

- How to predict multiple classes?
- Idea: have a linear response per class $r_c = \theta_c^\intercal x$
  - Choose class with largest response: $f_\theta(x) = \arg\max_c \theta_c^\intercal x$
- Linear boundary between classes c_1, c_2:
  - $\theta_{c_1}^\intercal x \lesseqgtr \theta_{c_2}^\intercal x \iff (\theta_{c_1} - \theta_{c_2})^\intercal x \lesseqgtr 0$



Multi-class linear models

- More generally: add features — can even depend on y!

$$f_\theta(x) = \arg\max_y \theta^\intercal \Phi(x, y)$$

- Example: $y = \pm 1$
  - $\Phi(x, y) = xy$

$$\implies f_\theta(x) = \arg\max_y y\theta^\intercal x = \begin{cases} +1 & +\theta^\intercal x > -\theta^\intercal x \\ -1 & +\theta^\intercal x < -\theta^\intercal x \end{cases}$$

$$= \text{sign}(\theta^\intercal x) \longleftarrow$$ **perceptron!**

Multi-class linear models

- More generally: add features — can even depend on y!

$$f_\theta(x) = \arg\max_y \theta^\intercal \Phi(x, y)$$

- Example: $y \in \{1,2,\ldots,C\}$
  - $\Phi(x, y) = [0 \;\; 0 \;\; \cdots \;\; x \;\; \cdots \;\; 0] = \text{one-hot}(y) \otimes x$
  - $\theta = [\theta_1 \;\; \cdots \;\; \theta_C]$

$$\implies f_\theta(x) = \arg\max_c \theta_c^\intercal x$$ ⟵ **largest linear response**

Multi-class perceptron algorithm

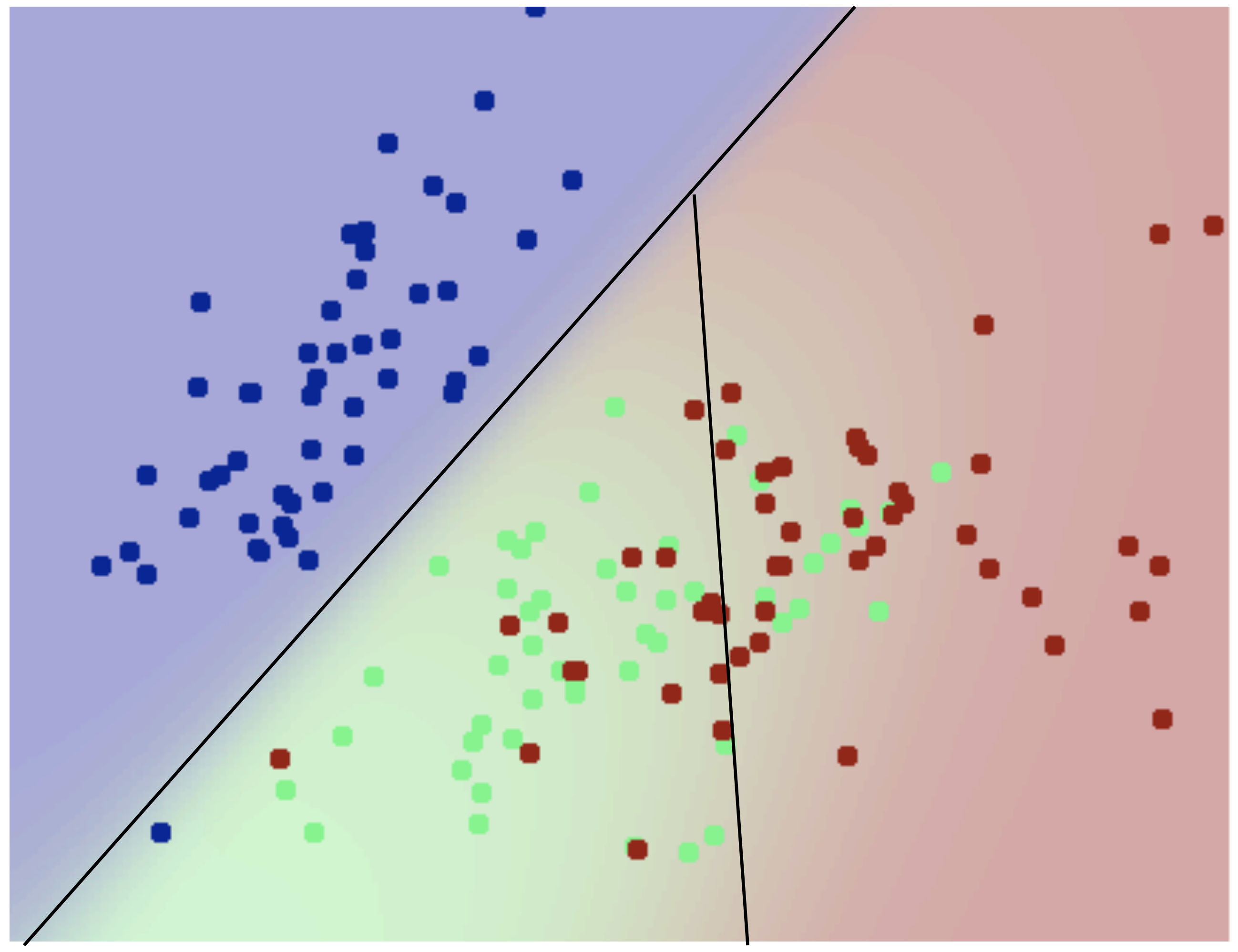
- While not done:
  - For each data point $(x, y) \in \mathscr{D}$:
    - Predict: $\hat{y} = \arg\max_c \theta_c^\intercal x$
    - Increase response for true class: $\theta_y \leftarrow \theta_y + \alpha x$
    - Decrease response for predicted class: $\theta_{\hat{y}} \leftarrow \theta_{\hat{y}} - \alpha x$
- More generally:
  - Predict: $\hat{y} = \arg\max_y \theta^\intercal \Phi(x, y)$
  - Update: $\theta \leftarrow \theta + \alpha(\Phi(x, y) - \Phi(x, \hat{y}))$

Multilogit Regression

- Define multi-class probabilities: $p_\theta(y|x) = \dfrac{\exp(\theta_y^\intercal x)}{\sum_c \exp(\theta_c^\intercal x)} = \operatorname{soft\,max}_c \theta_c^\intercal x \Big|_y$
  - For binary y: $$p_\theta(y=1|x) = \frac{\exp(\theta_1^\intercal x)}{\exp(\theta_1^\intercal x) + \exp(\theta_2^\intercal x)}$$ $$= \frac{1}{1+\exp((\theta_2-\theta_1)^\intercal x)} = \sigma((\theta_1-\theta_2)^\intercal x)$$

Logistic Regression with $\theta = \theta_1 - \theta_2$

- Benefits:
  - Probabilistic predictions: knows its confidence
  - Linear decision boundary: $\arg\max_y \exp(\theta_y^\intercal x) = \arg\max_y \theta_y^\intercal x$
  - NLL is convex



Multilogit Regression: gradient

- NLL loss: $\mathscr{L}_\theta(x, y) = -\log p_\theta(y \,|\, x) = -\theta_y^\intercal x + \log \sum_c \exp(\theta_c^\intercal x)$

- Gradient:

$$-\nabla_{\theta_c} \mathscr{L}_\theta(x, y) = \delta(y = c)x - \frac{\nabla_{\theta_c} \sum_{c'} \exp(\theta_{c'}^\intercal x)}{\sum_{c'} \exp(\theta_{c'}^\intercal x)}$$

$$= \left(\delta(y = c) - \frac{\exp(\theta_c^\intercal x)}{\sum_{c'} \exp(\theta_{c'}^\intercal x)} \right) x$$

$$= (\delta(y = c) - p_\theta(c \,|\, x))x$$

make true class more likely

make all other classes less likely

- Compare to multi-class perceptron: $(\delta(y = c) - \delta(\hat{y} = c))x$

Today's lecture

Learning perceptrons

Logistic regression

Multi-class classifiers

VC dimension

Complexity measures

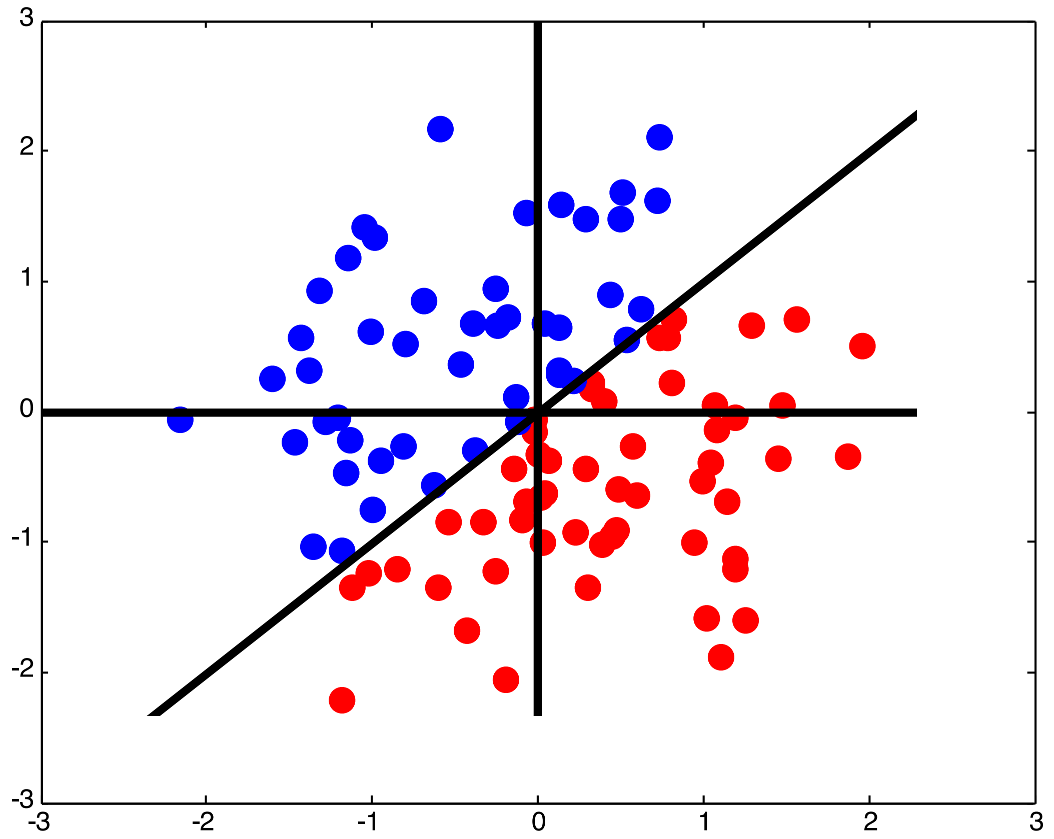
- What are we looking for in a measure of model class complexity?
  - Tell us something about generalization error $\mathscr{L}_{\text{test}} - \mathscr{L}_{\text{training}}$
  - Tell us how it depends on amount of data m

    also called: risk – empirical risk

  - Be easy to find for a given model class — haha jk not gonna happen (more later)
- Ideally: a way to select model complexity (other than validation)
  - Akaike Information Criterion (AIC) — roughly: loss + #parameters
  - Bayesian Information Criterion (BIC) — roughly: loss + #parameters $\cdot \log m$
    - But what's the #parameters, effectively? Should $f_{\theta_1,\theta_2} = g_{\theta=h(\theta_1,\theta_2)}$ change the complexity?

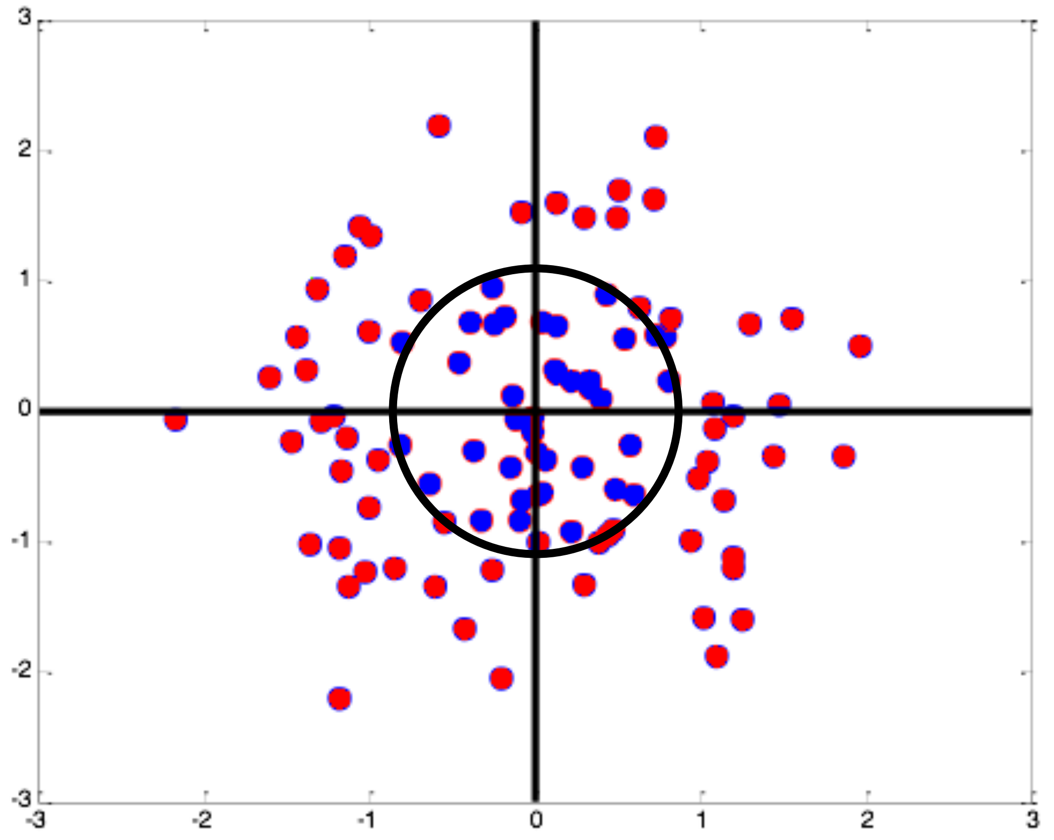
Model expressiveness

- Model complexity also measures expressiveness / representational power
- Tradeoff:
  - More expressive $\implies$ can reduce error, but may also overfit to training data
  - Less expressive $\implies$ may not be able to represent true pattern / trend
- Example: $\text{sign}(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$



Model expressiveness

- Model complexity also measures expressiveness / representational power
- Tradeoff:
  - More expressive $\implies$ can reduce error, but may also overfit to training data
  - Less expressive $\implies$ may not be able to represent true pattern / trend
- Example: $\text{sign}(x_1^2 + x_2^2 - \theta)$



Shattering

- Separability / realizability: there's a model that classifies all points correctly
- Shattering: the points are separable regardless of their labels
  - Our model class can shatter points $x^{(1)}, \ldots, x^{(h)}$

    if for any labeling $y^{(1)}, \ldots, y^{(h)}$

    there exists a model that classifies all of them correctly

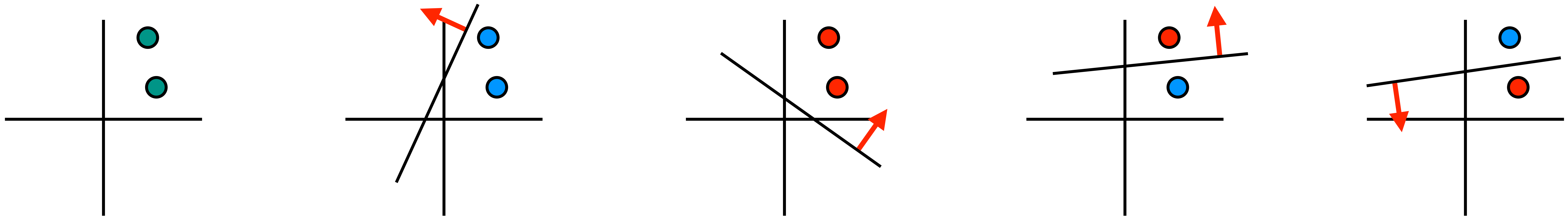
    - The model class must have at least as many models as labelings C^h

Shattering

- Separability / realizability: there's a model that classifies all points correctly
- Shattering: the points are separable regardless of their labels
  - Our model class can shatter points $x^{(1)}, \ldots, x^{(h)}$

    if for any labeling $y^{(1)}, \ldots, y^{(h)}$

    there exists a model that classifies all of them correctly
- Example: can $f_\theta(x) = \mathrm{sign}(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ shatter these points?

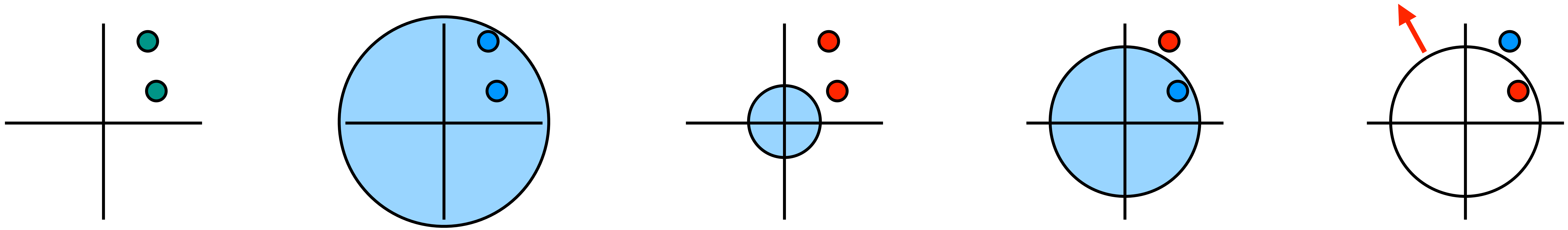


Shattering

- Separability / realizability: there's a model that classifies all points correctly
- Shattering: the points are separable regardless of their labels
  - Our model class can shatter points $x^{(1)}, \ldots, x^{(h)}$

    if for any labeling $y^{(1)}, \ldots, y^{(h)}$

    there exists a model that classifies all of them correctly
- Example: can $f_\theta(x) = \text{sign}(x_1^2 + x_2^2 - \theta)$ shatter these points?

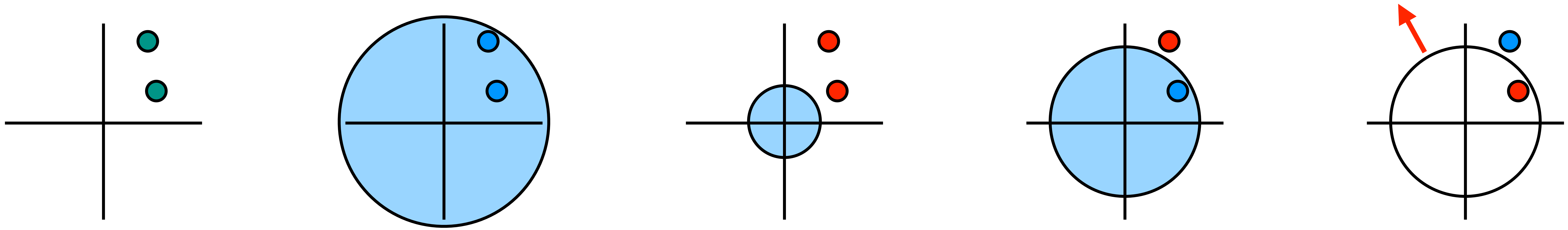


Vapnik–Chervonenkis (VC) dimension

- VC dimension: maximum number H of points that can be shattered by a class
- A game:
  - Fix a model class $f_\theta : x \to y \quad \theta \in \Theta$
  - Player 1: choose h points $x^{(1)}, \ldots, x^{(h)}$
  - Player 2: choose labels $y^{(1)}, \ldots, y^{(h)}$
  - Player 1: choose model θ
  - Are all $y^{(j)} = f_\theta(x^{(j)})$? $\implies$ Player 1 wins $\quad \exists x^{(1)}, \ldots, x^{(h)} : \ \forall y^{(1)}, \ldots, y^{(h)} : \ \exists \theta : \ \forall j : \ y^{(j)} = f_\theta(x^{(j)})$
- $h \leq H \implies$ Player 1 can win, otherwise cannot win

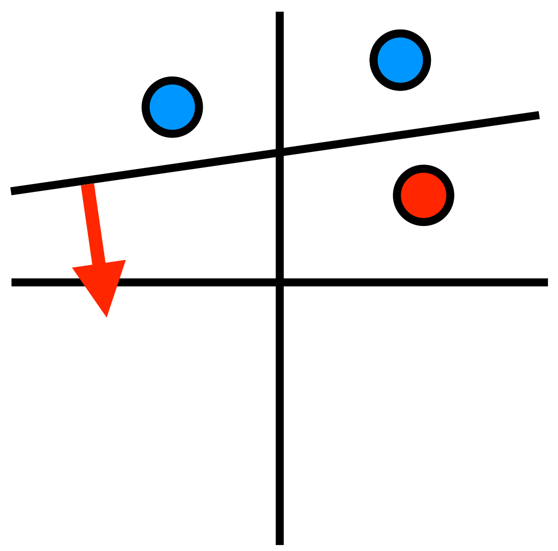
VC dimension: example (1)

- VC dimension: maximum number H of points that can be shattered by a class
- To find H, think like the winning player: 1 for $h \leq H$, 2 for $h > H$
- Example: $f_\theta(x) = \text{sign}(x_1^2 + x_2^2 - \theta)$
  - We can place one point and ”shatter” it
  - We can prevent shattering any two points: make the distant one blue
  - $H = 1$

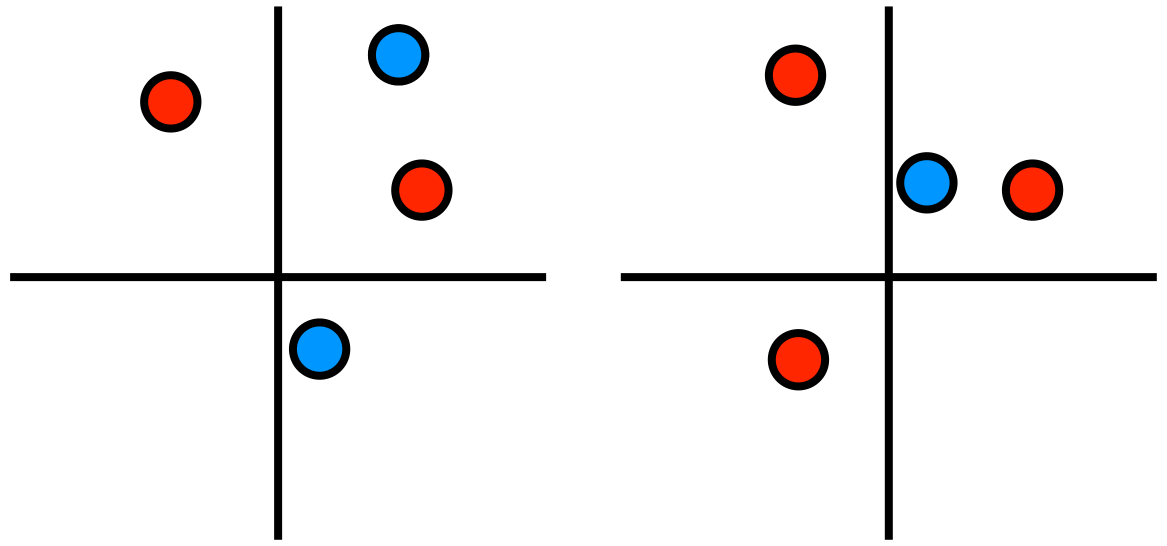


VC dimension: example (2)

- Example: $f_\theta(x) = \text{sign}(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
  - We can place 3 points and shatter them



  - We can prevent shattering any 4 points:
    - If they form a convex shape, alternate labels
    - Otherwise, label differently the point in the triangle



  - $H = 3$
- Linear classifiers (perceptrons) of d features have VC-dim $d + 1$
  - But VC-dim is generally not #parameters

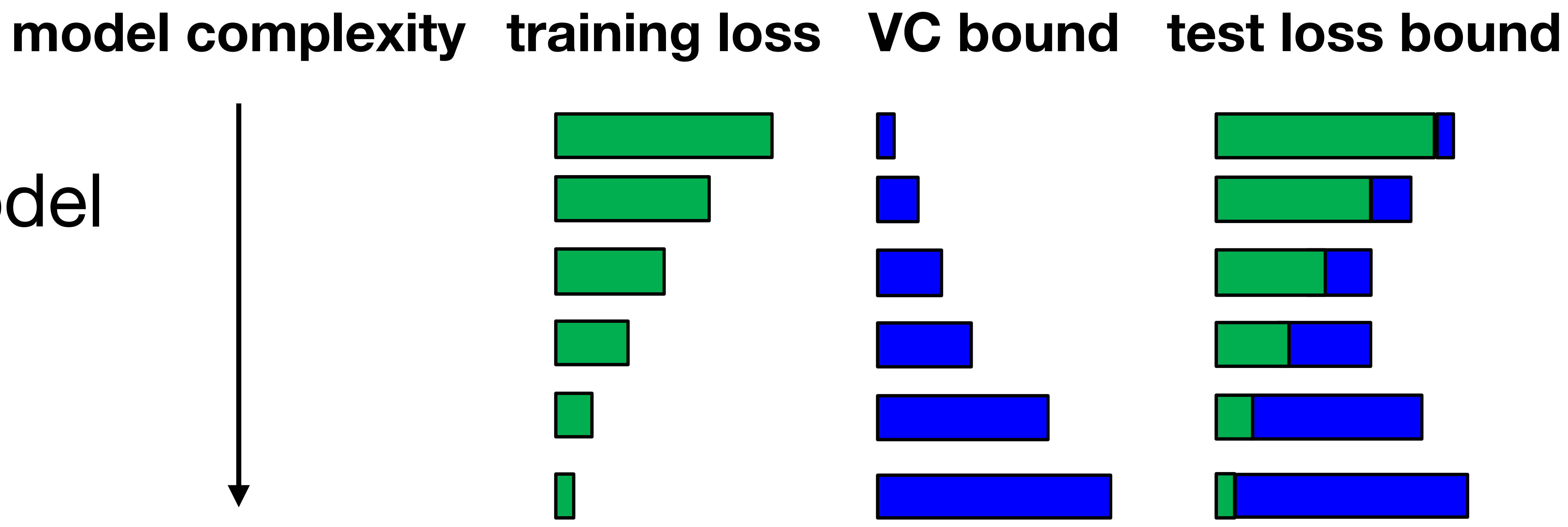
VC Generalization bound

- VC-dim of a model class can be used to bound generalization loss:
  - With probability at least $1-\eta$, we will get a "good" dataset, for which
- $$\underbrace{\text{test loss} \; - \; \text{training loss}}_{\text{generalization loss}} \leq \sqrt{\frac{H\log(2m/H) + H - \log(\eta/4)}{m}}$$
- We need larger training size m:
  - The better generalization we need
  - The more complex (higher VC-dim) our model class
  - The more likely we want to get a good training sample

Model selection with VC-dim

- Using validation / cross-validation:
  - Estimate loss on held out set
  - Use validation loss to select model

- Using VC dimension:
  - Use generalization bound to select model
  - Structural Risk Minimization (SRM)
  - Bound not tight, must too conservative



Logistics

assignments

- Assignment 3 due next Tuesday, Oct 26

midterm

- Midterm exam on Nov 4, 11am–12:20 in **SH 128**
- If you're eligible to be remote — let us know by Oct 28
- If you're eligible for more time — let us know by Oct 28
- Review during lecture next Thursday