University of California, Irvine

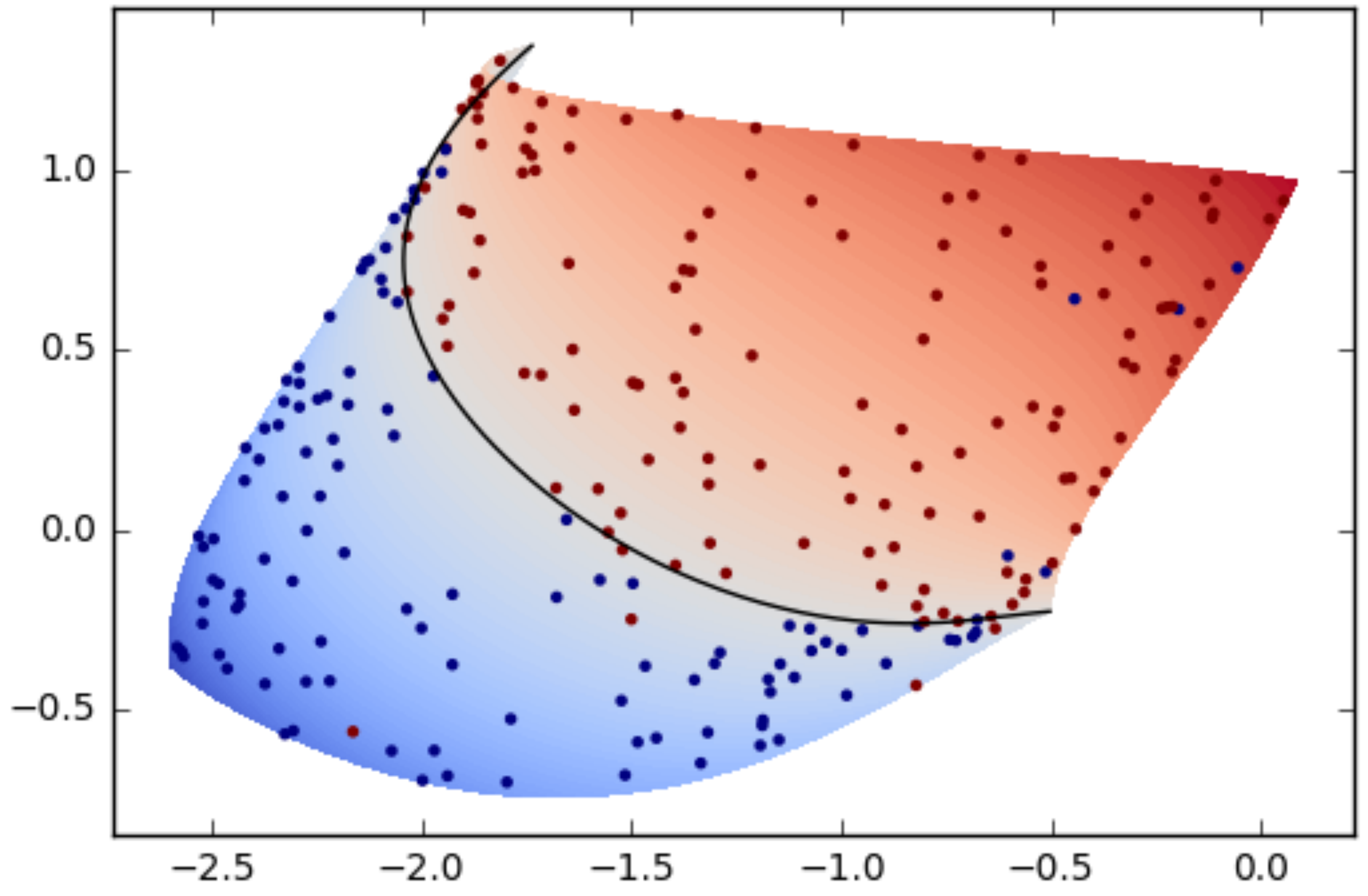
CS 273A: Machine Learning

Fall 2021

Lecture 7: Linear Classifiers

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 2 due next Tuesday, Oct 19

project

- Project guidelines on Canvas
- Team rosters due next Tuesday, Oct 19 on Canvas

Today's lecture

Regularization and cross-validation

Perceptrons

Separability

Learning perceptrons

L_2 regularization

- Modify the loss function by adding a regularization term
- L_2 regularization (ridge regression) for MSE: $\mathscr{L}_\theta = \frac{1}{2}(\|y - \theta^\intercal X\|^2 + \alpha\|\theta\|^2)$
- Optimally: $\theta^\intercal = yX^\intercal(XX^\intercal + \alpha I)^{-1}$
  - αI moves $XX^\intercal$ away from singularity → inverse exists, better "conditioned"
  - Shrinks θ towards 0 (as expected)
    - At the expense of training MSE
- Regularization term $\alpha\|\theta\|^2$ independent of data = prior?

Gaussian distribution vs. quadratic log-prob

$$p(z) = \mathcal{N}(z; \mu, \Sigma) = (2\pi)^{-\frac{d}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z-\mu)^\intercal \Sigma^{-1}(z-\mu)\right)$$

$$\log p(z) = -\frac{1}{2}(z-\mu)^\intercal \Sigma^{-1}(z-\mu) + \text{const}$$

$$\log p(z) = -\frac{1}{2}z^\intercal A z + b^\intercal z + c = -\frac{1}{2}(z - A^{-1}b)^\intercal A(z - A^{-1}b) + \text{const}$$

$$p(z) = \mathcal{N}(A^{-1}b, A^{-1})$$

$$\log p(z, w) = -\frac{1}{2}z^\intercal A(w) z + b(w)^\intercal z + c(w) \implies p(z|w) = \mathcal{N}(A^{-1}(w)b(w), A^{-1}(w))$$

Regularization and Bayesian prediction

- Assume the data was generated using this process:
  - Parameter vector θ was sampled from a Gaussian: $\theta \sim \mathcal{N}(0,\alpha^{-1}I)$
  - Features X were sampled “somehow” (it won't matter)
  - Labels y are linear in X, but with Gaussian noise: $y = \theta^\intercal X + \epsilon \qquad \epsilon \sim \mathcal{N}(0,I)$
- What is the joint distribution $p(\theta, X, y)$?
  - $p(\theta, X, y) = p(\theta)p(X)p(y \mid \theta, X) = \mathcal{N}(\theta; 0,\alpha^{-1}I)p(X)\mathcal{N}(y - \theta^\intercal X; 0,I)$
  - $\log p(\theta, X, y) = \log p(X) - \frac{1}{2}\alpha^2\|\theta\|^2 - \frac{1}{2}\|y - \theta^\intercal X\|^2 + \text{const}$
  - $p(\theta \mid X, y) = \mathcal{N}(\theta; \underbrace{(XX^\intercal + \alpha I)^{-1}Xy}_{\text{MAP } \theta}, (XX^\intercal + \alpha I)^{-1})$

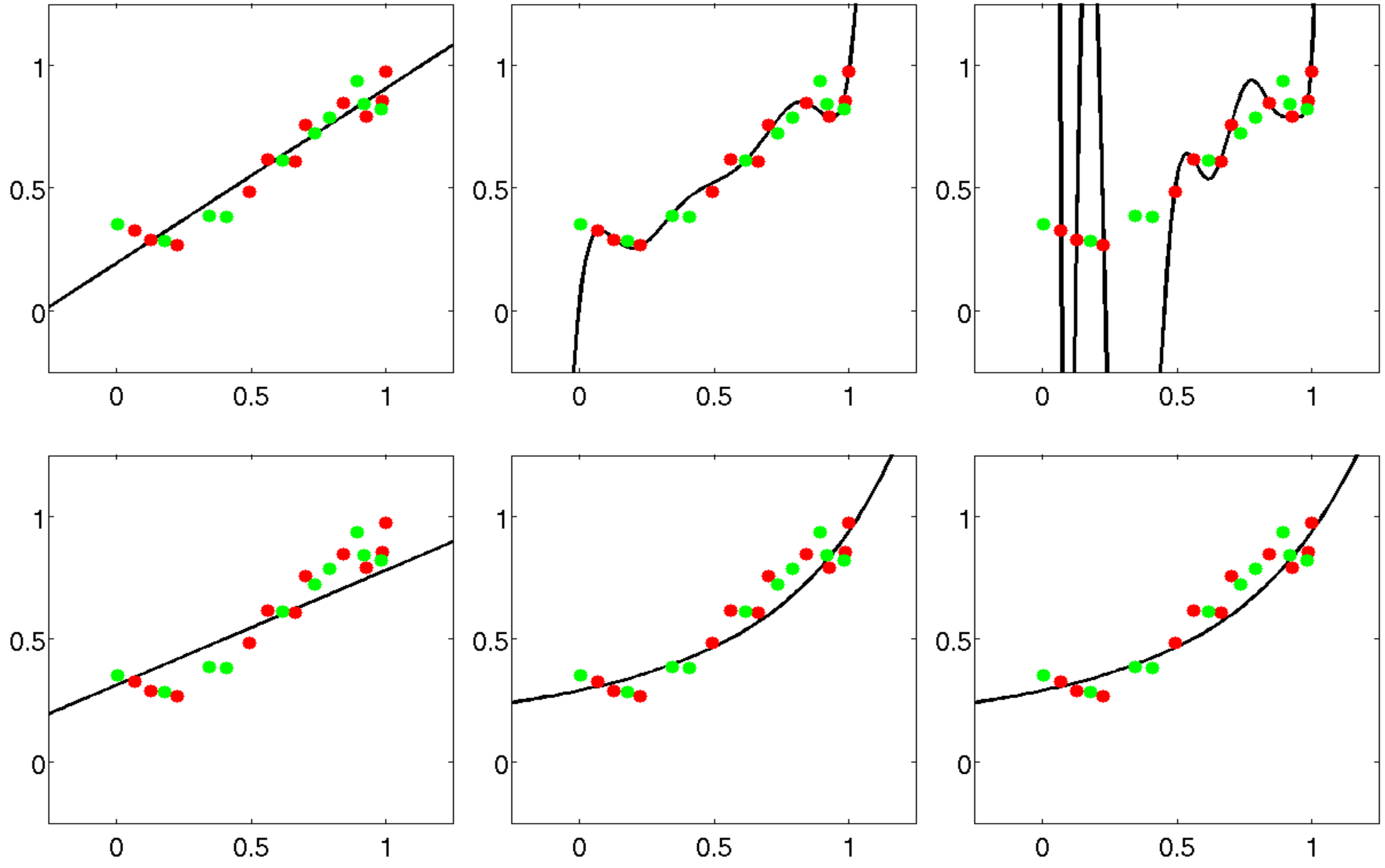
Regularization

- Comparing unregularized and regularized regression:

- $\alpha = 0$

  (Unreg.)

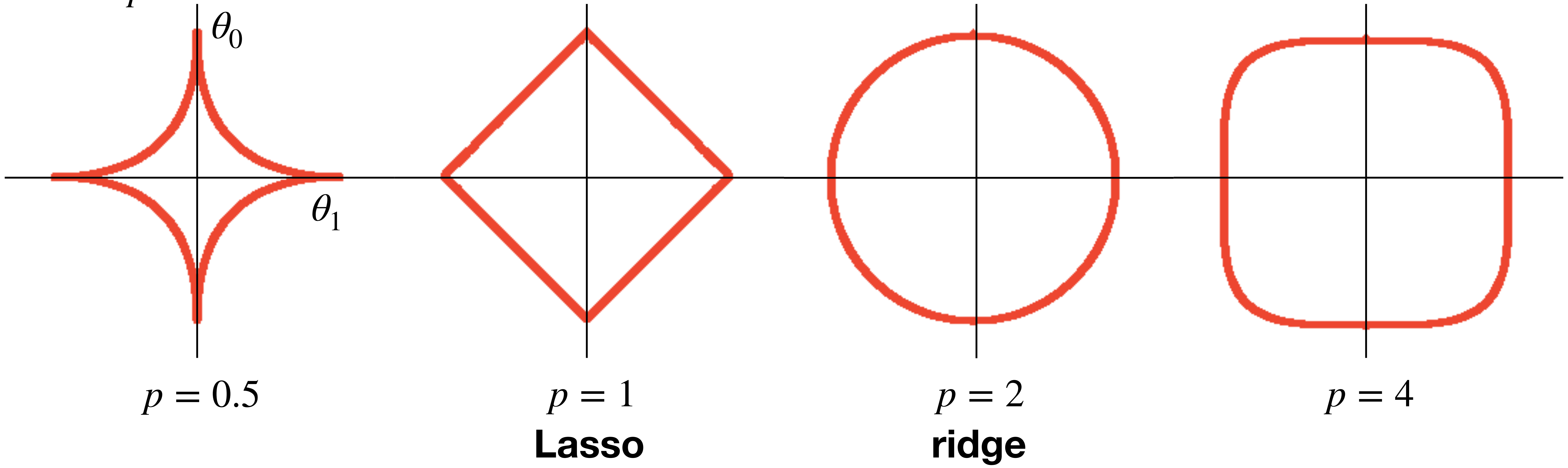
- $\alpha = 1$



L_p regularization

- Other popular regularizers are L_p norm: $\|\theta\|_p = \left(\sum_i |\theta_i|^p\right)^{\frac{1}{p}}$

- Isosurfaces: $(\|\theta\|_p = \text{const})$

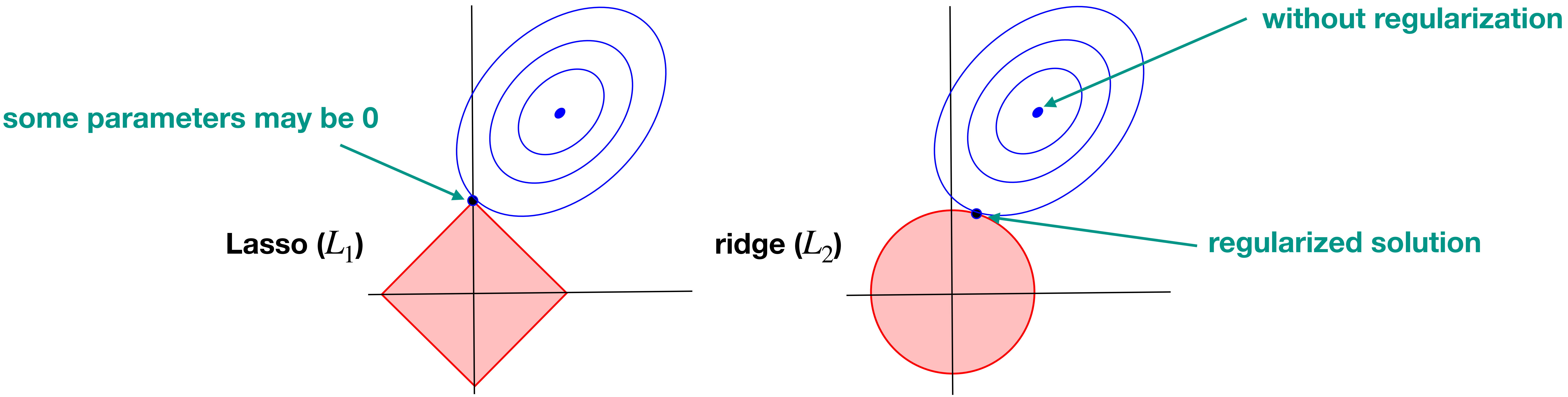


- $L_0 = \lim_{p \to 0} L_p$: number of nonzero parameters, natural notion of model complexity

- $L_\infty = \lim_{p \to \infty} L_p$: maximum parameter value

Regularization: L_1 vs. L_2

- θ estimate balances training loss and regularization
- Lasso (L_1) tends to generate sparser solutions than ridge (L_2) regularizer

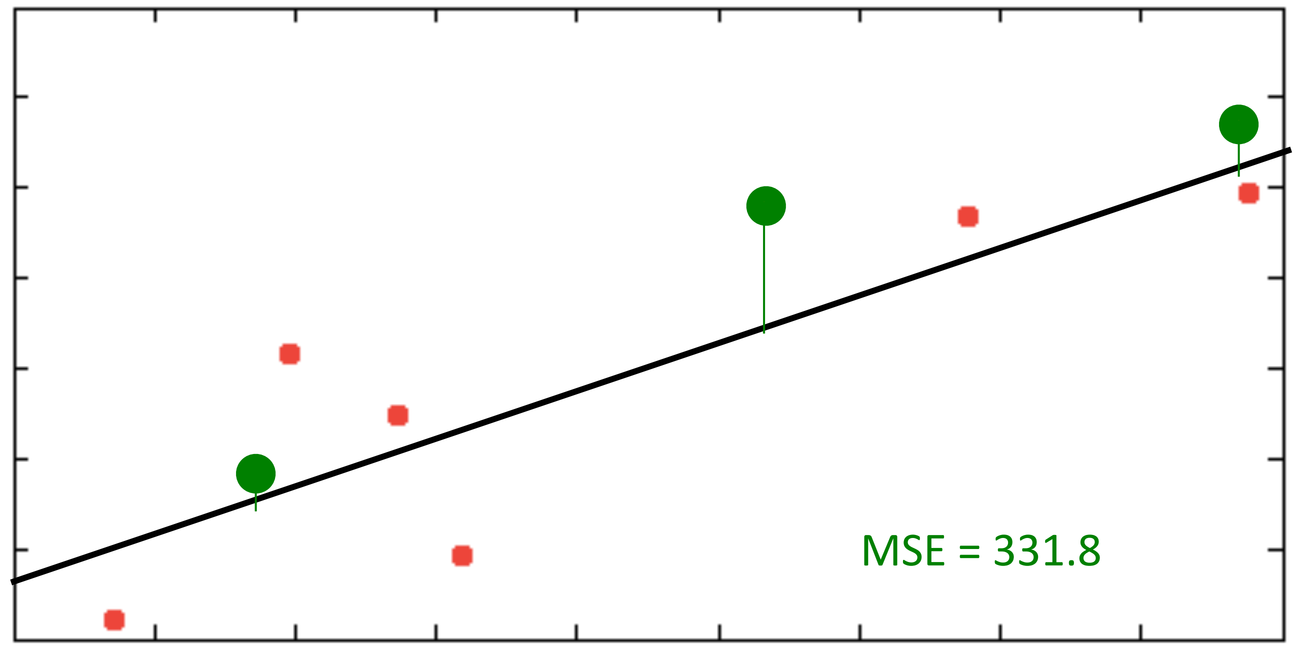


Validation

- To select model class / model hyper-parameters ϕ (e.g. polynomial degree)
  - Train models on training dataset: $\theta = \mathscr{A}_\phi(\mathscr{D}_{\text{training}})$
  - Evaluate models on validation dataset: $\mathscr{L} = \mathbb{E}_{x,y \sim \mathscr{D}_{\text{validation}}}[\ell_\theta(x, y)]$
- What if we don't get a validation set?
  - Split training set into training + validation

Hold-out method

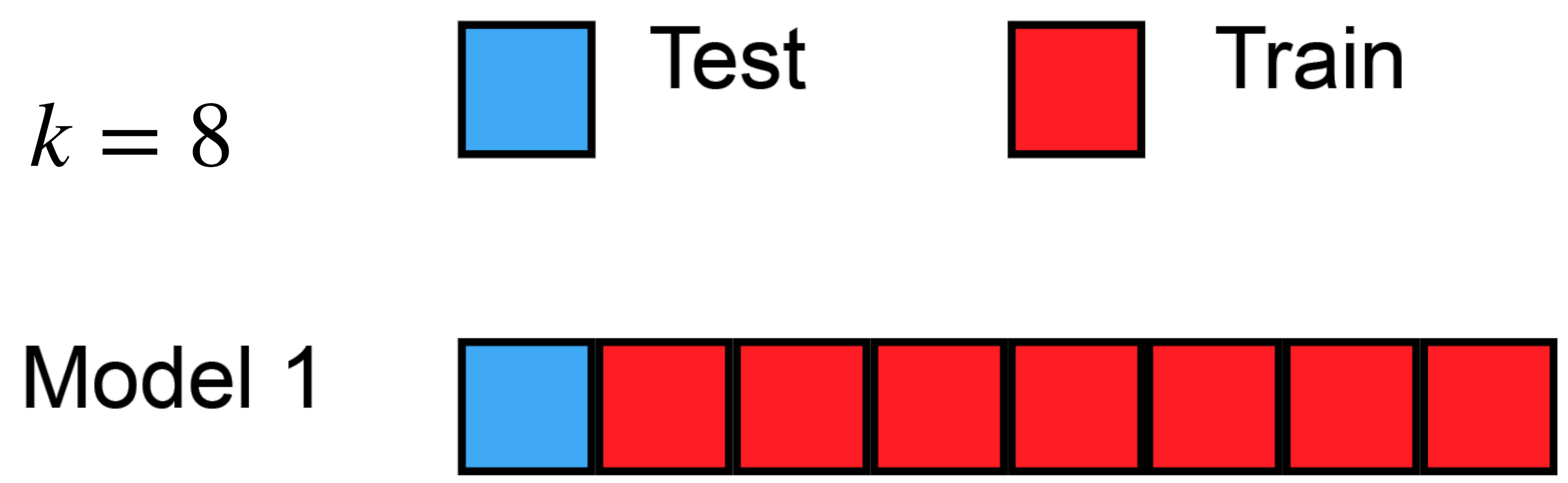
- Hold out some data for validation; e.g., random 30% of the data
  - Don't just sample training + validation with repetitions — they must be disjoint
- How to split?
  - Too few training data points → poor training, bad θ
  - Too few validation data points → poor validation, bad loss estimate
- Can we use more splits?



	$x^{(i)}$	$y^{(i)}$
training data	88	79
	32	-2
	27	30
	68	73
	7	-16
	20	43
validation data	53	77
	17	16
	87	94

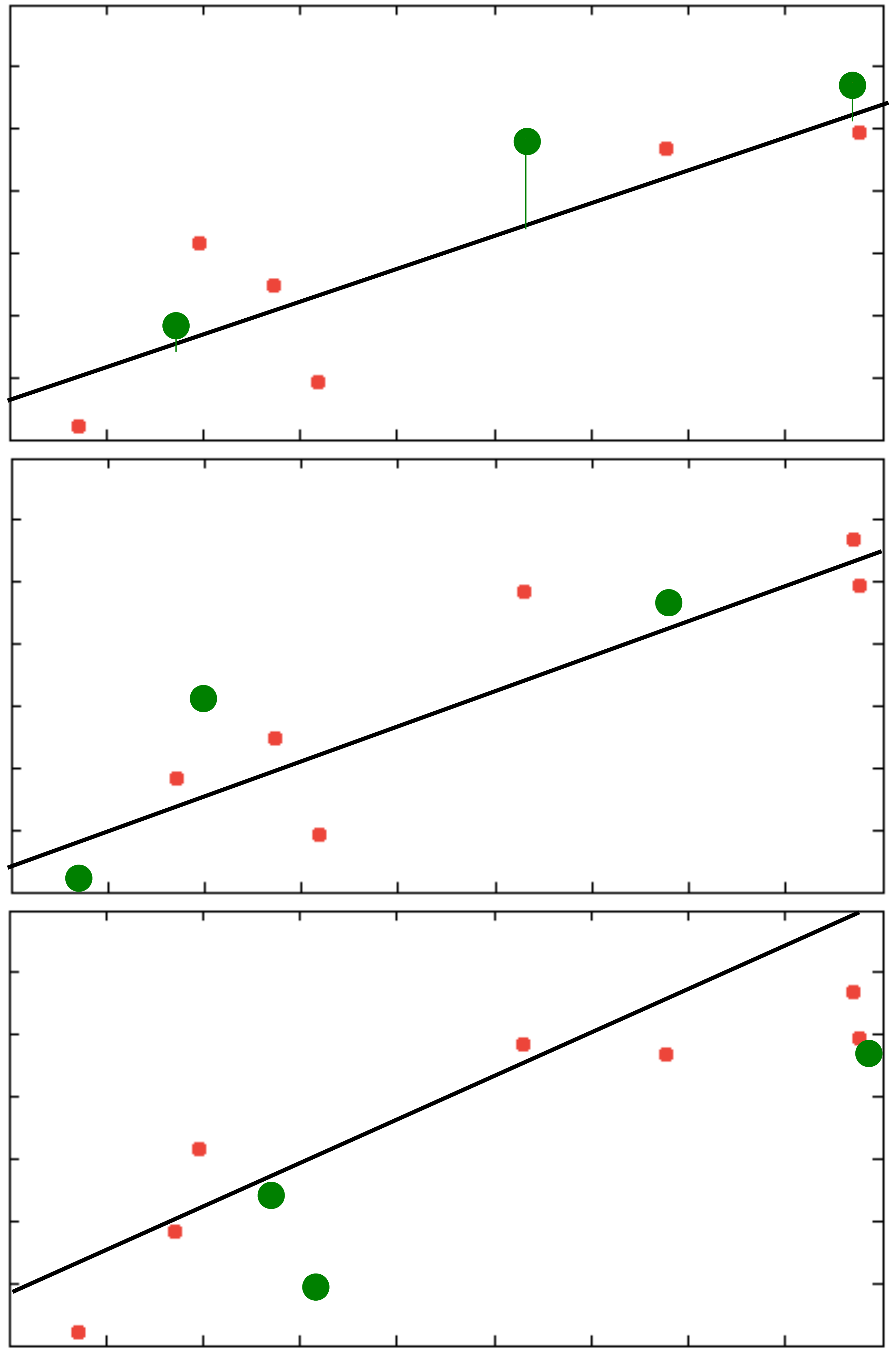
k-fold cross-validation method

- Randomly split the data into k disjoint sets
- For each of the k sets:
  - Hold it, train on the other $k-1$ sets
  - Validate on the held-out set
- Use average validation loss to select model hyper-parameters ϕ
- Train with selected ϕ on full data



k-fold cross-validation method

- Benefits:
  - Use all data for validation
  - Use all data to train final model



Split 1:
MSE = 331.8

Split 2:
MSE = 361.2

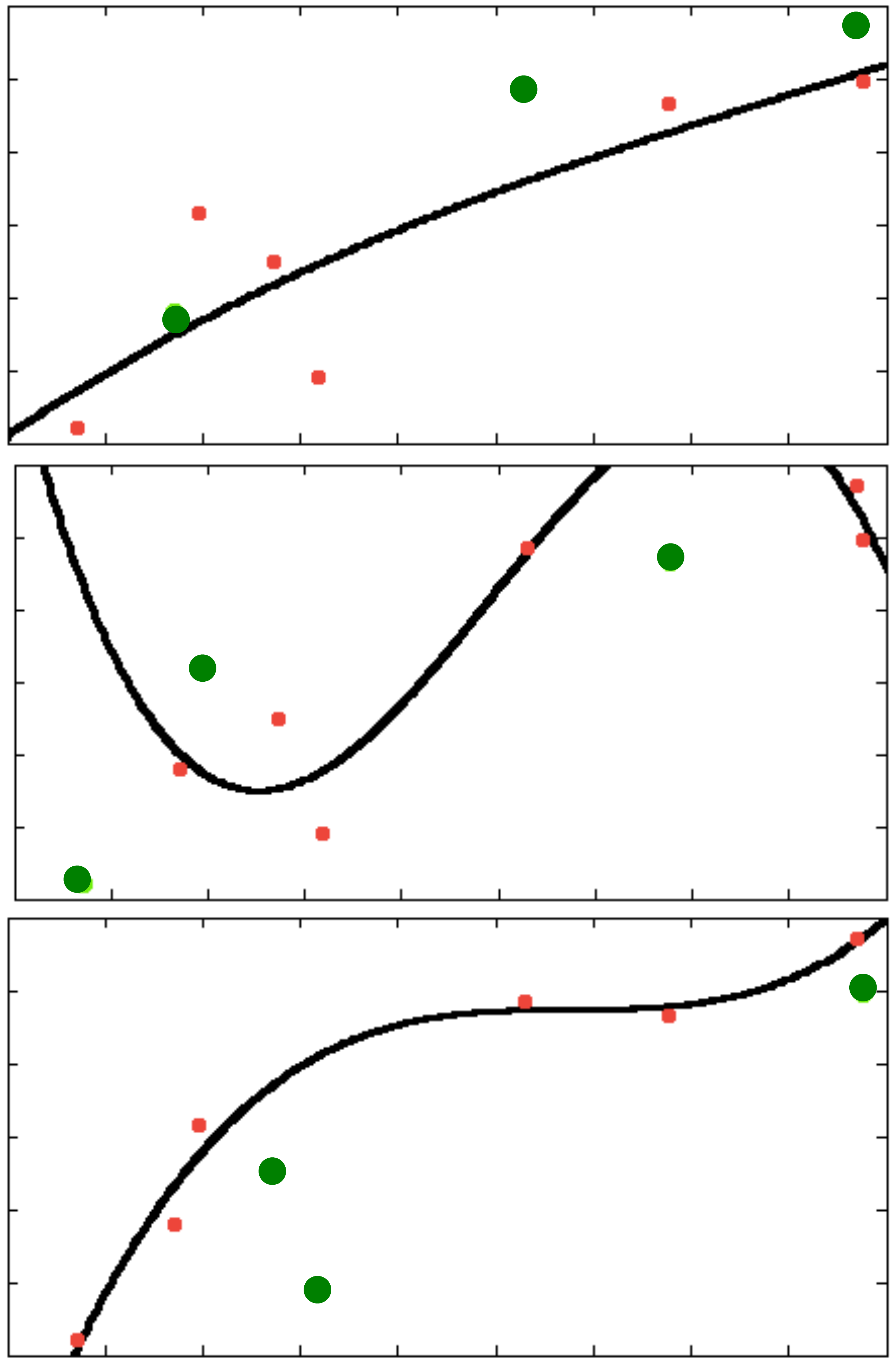
Split 3:
MSE = 669.8

3-Fold X-Val MSE
= 464.1

$x^{(i)}$	$y^{(i)}$
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

k-fold cross-validation method

- Benefits:
  - Use all data for validation
  - Use all data to train final model
- Drawbacks:
  - Trains k (+1) models
  - Each model still gets noisy validation from $\frac{m}{k}$ data points
  - No validation for the final model
- When $k = m$: Leave-One-Out (LOO)



Split 1:
MSE = 280.5

Split 2:
MSE = 3081.3

Split 3:
MSE = 1640.1

3-Fold X-Val MSE
= 1667.3

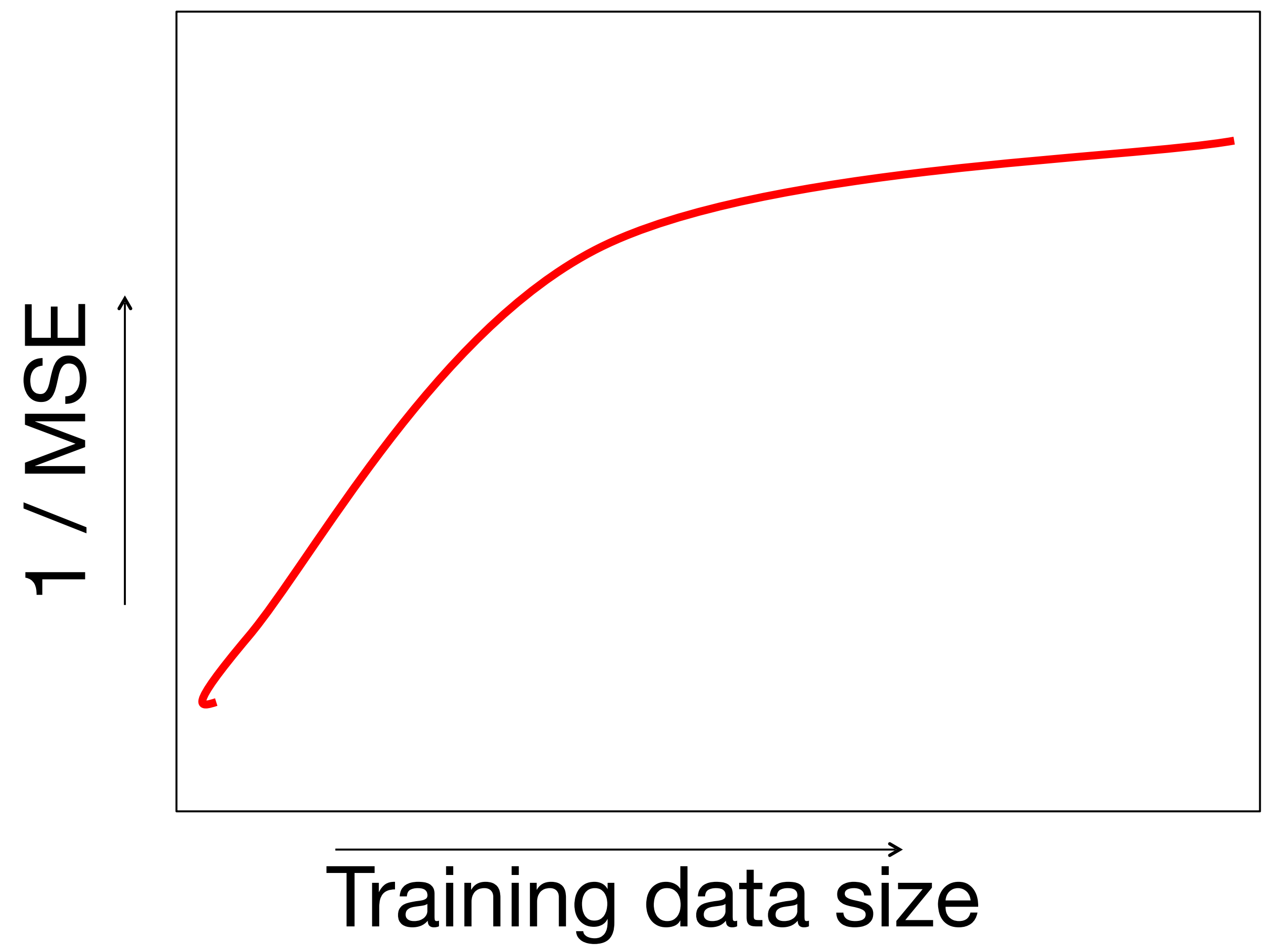
$x^{(i)}$	$y^{(i)}$
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

Cross-validation: considerations

- Trade off model training time with loss estimation accuracy
- Single held-out set: train on $m' < m$ data points, estimate loss on the rest
  - m must be large enough for both training and validation
  - We have an estimate of the final model performance
- k-fold XVal: split data into k disjoint sets, train on all but one used for validation
  - Computationally more expensive: training k models
  - Each validated model may be worse: trained on $m - \frac{m}{k}$ data points
  - But: estimate loss on more data, output model trained on all data
- LOO XVal: train on all but one data point, validate it, average this over all data points

Learning curves

- Plot performance (higher = better) as a function of training size
  - Assess impact of fewer data on performance
  - E.g., MSE0 – MSE for regression, or 1 – error rate for classification
- Performance (properly measured) should increase with training size
  - Should improve quickly when data is scarce, saturate when there's "enough"
  - May need to average over multiple experiments / trials / runs



Today's lecture

Regularization and cross-validation

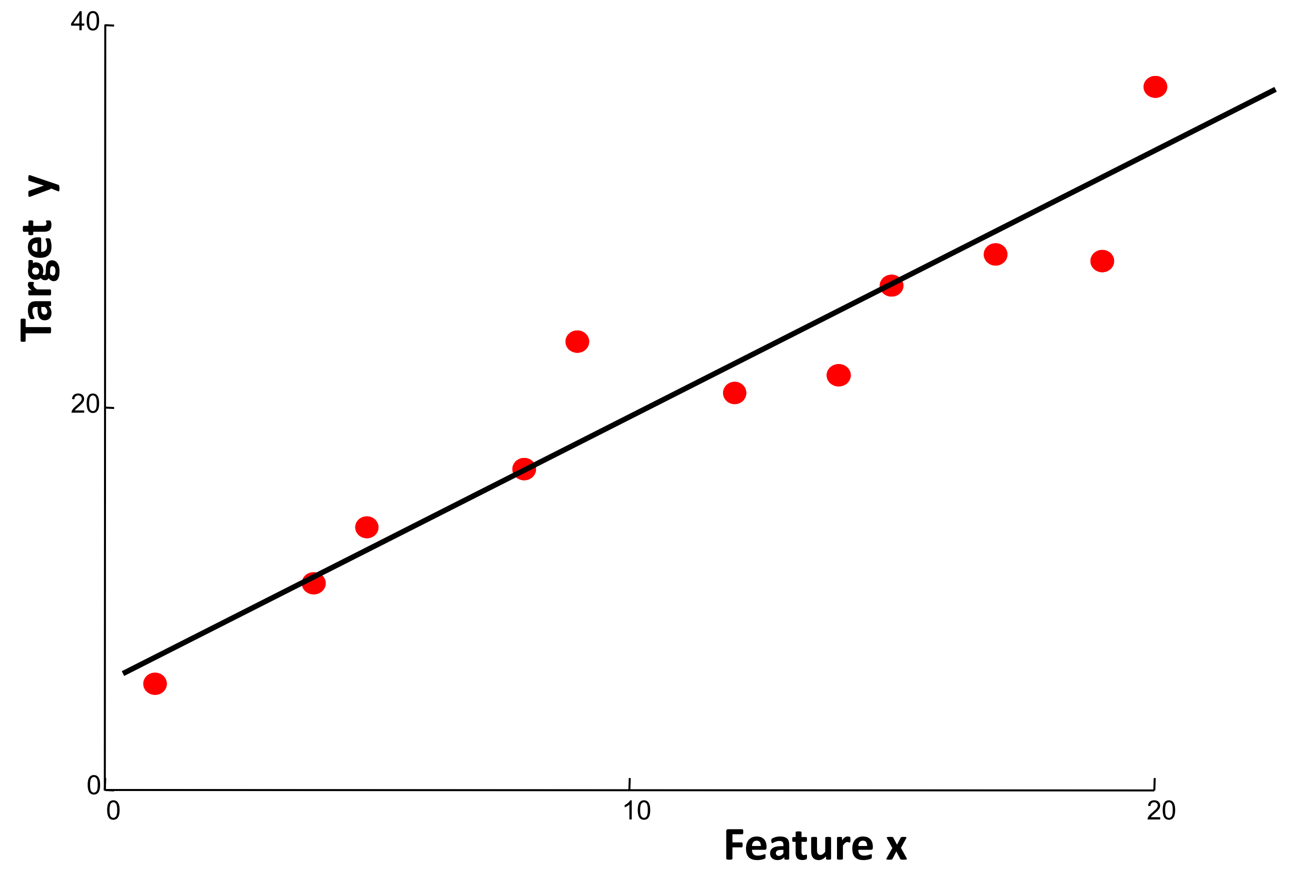
Perceptrons

Separability

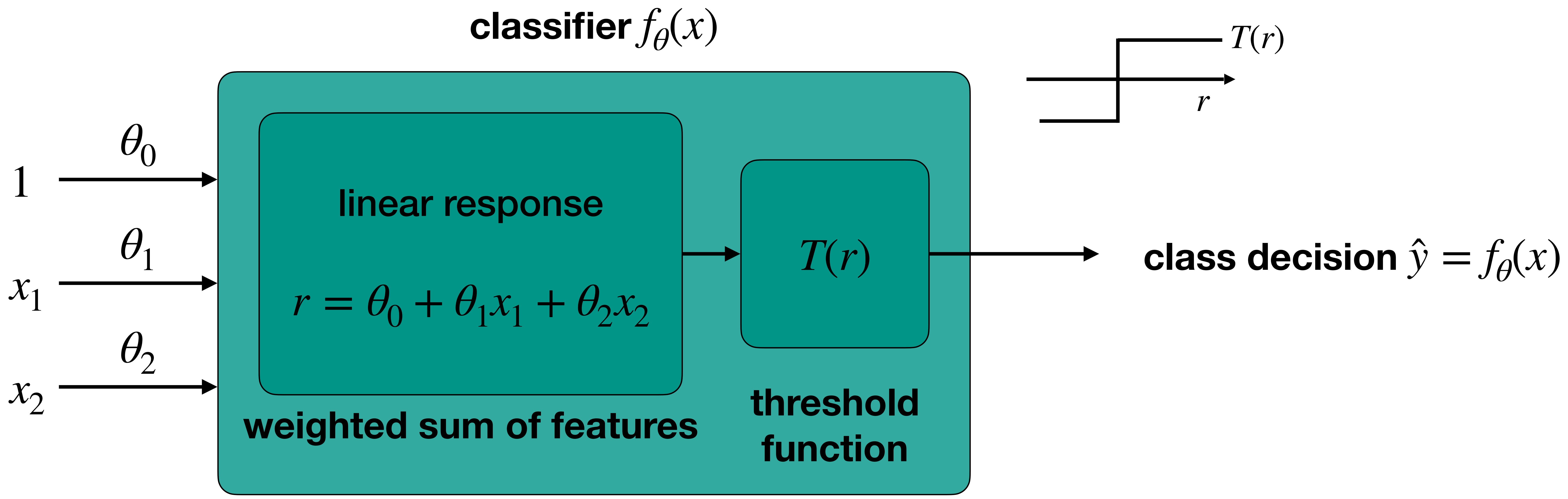
Learning perceptrons

Linear regression vs. classification

- Regression:
  - Continuous target y
  - Regressor $\hat{y} = \theta^\intercal x$
- Classification:
  - Discrete label y
  - Classifier $\hat{y} = ?$

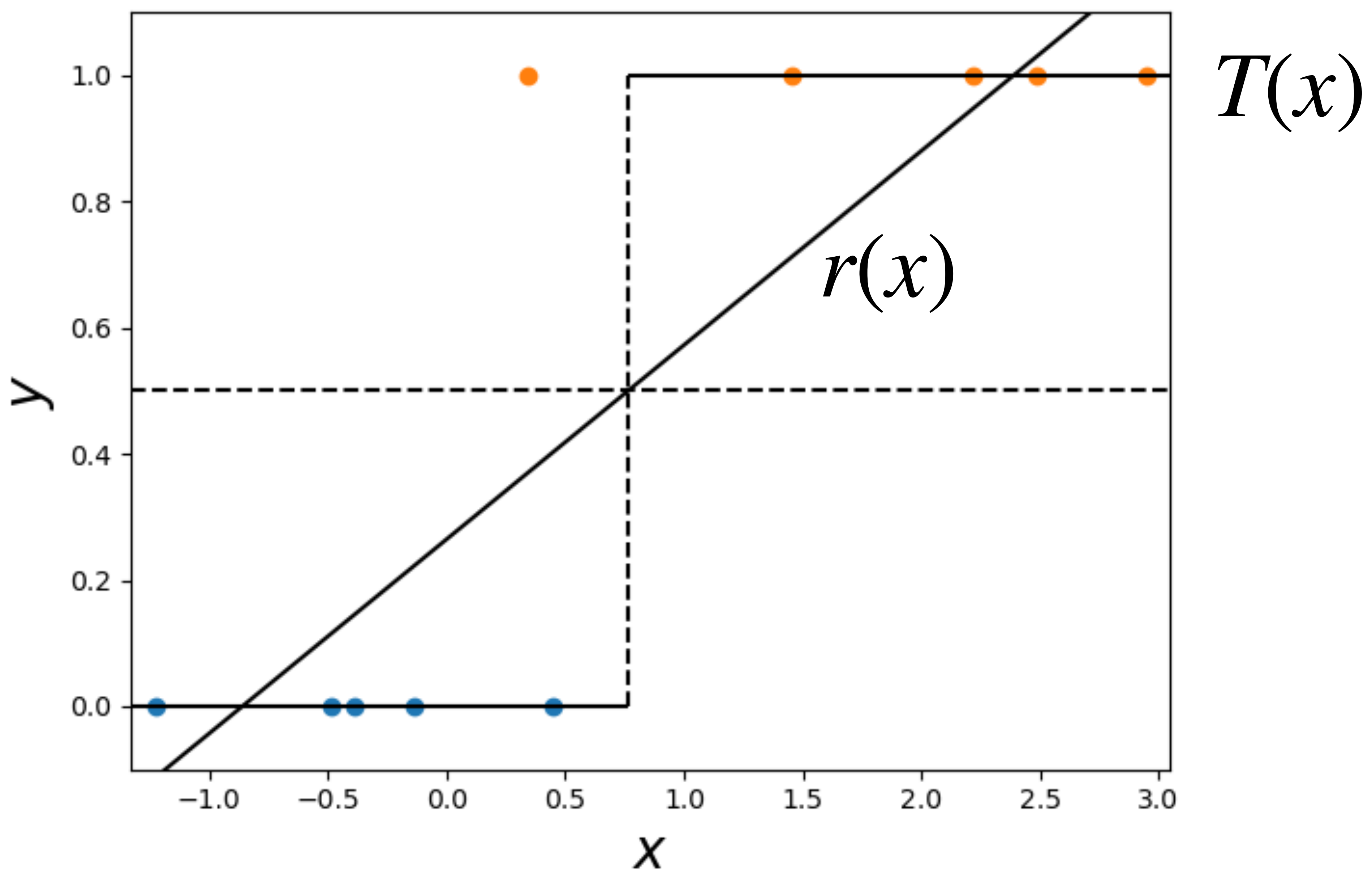


Perceptron



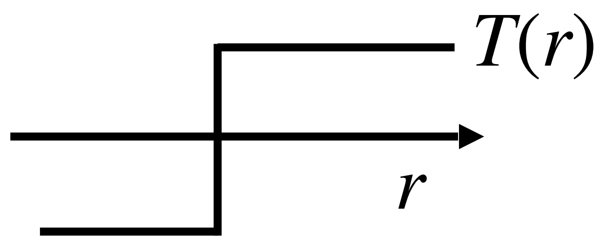

```
r = theta.T @ X           # compute linear response
y_hat = (r > 0)           # predict class 1 vs. 0
y_hat = 2*(r > 0) - 1     # predict class 1 vs. -1
```

Perceptron



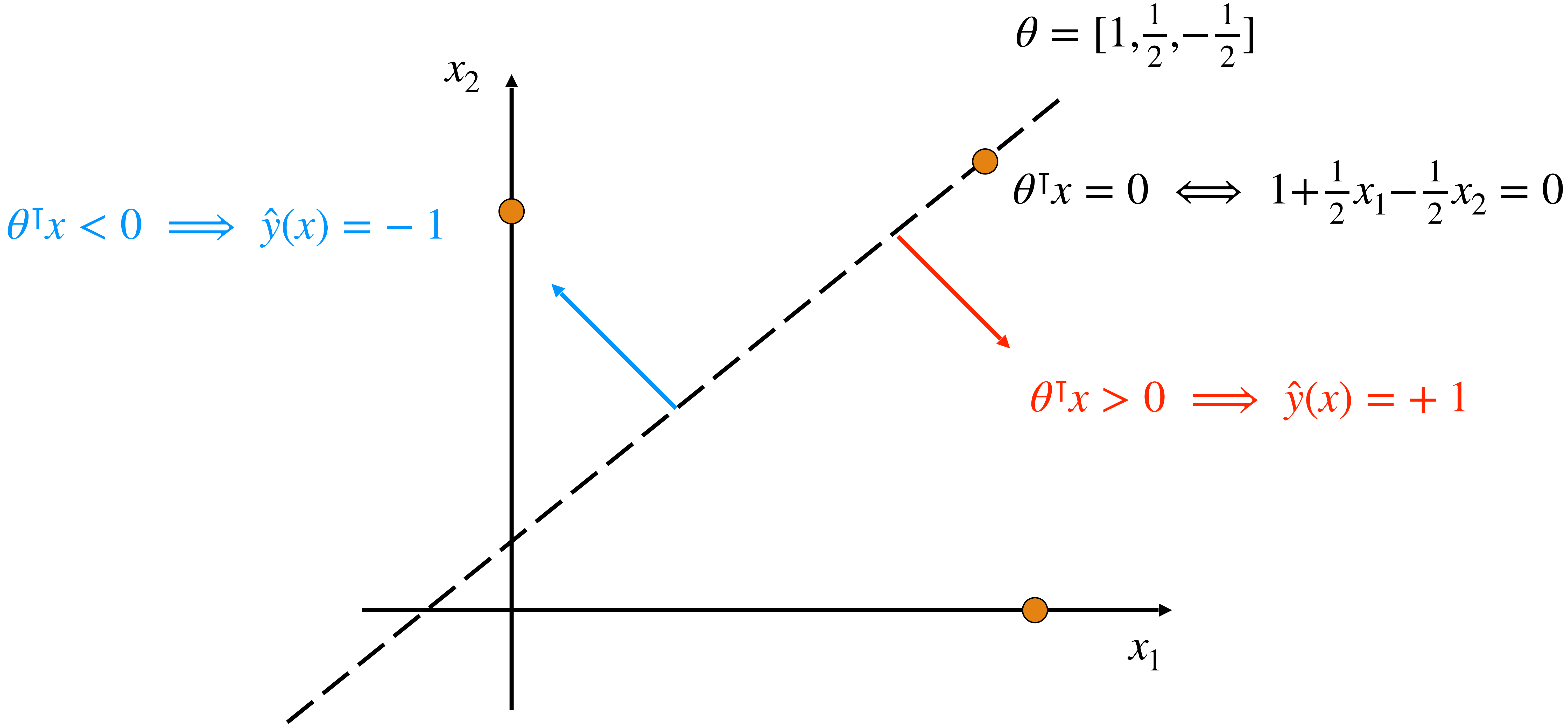
Perceptron

- 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 $T(r) = \text{sign}(r)$)



- Perceptron: a simple (vastly inaccurate) model of human neurons
  - Weights = “synapses”
  - Prediction = “neural firing”

Example



Adapted from Padhraic Smyth

Today's lecture

Regularization and cross-validation

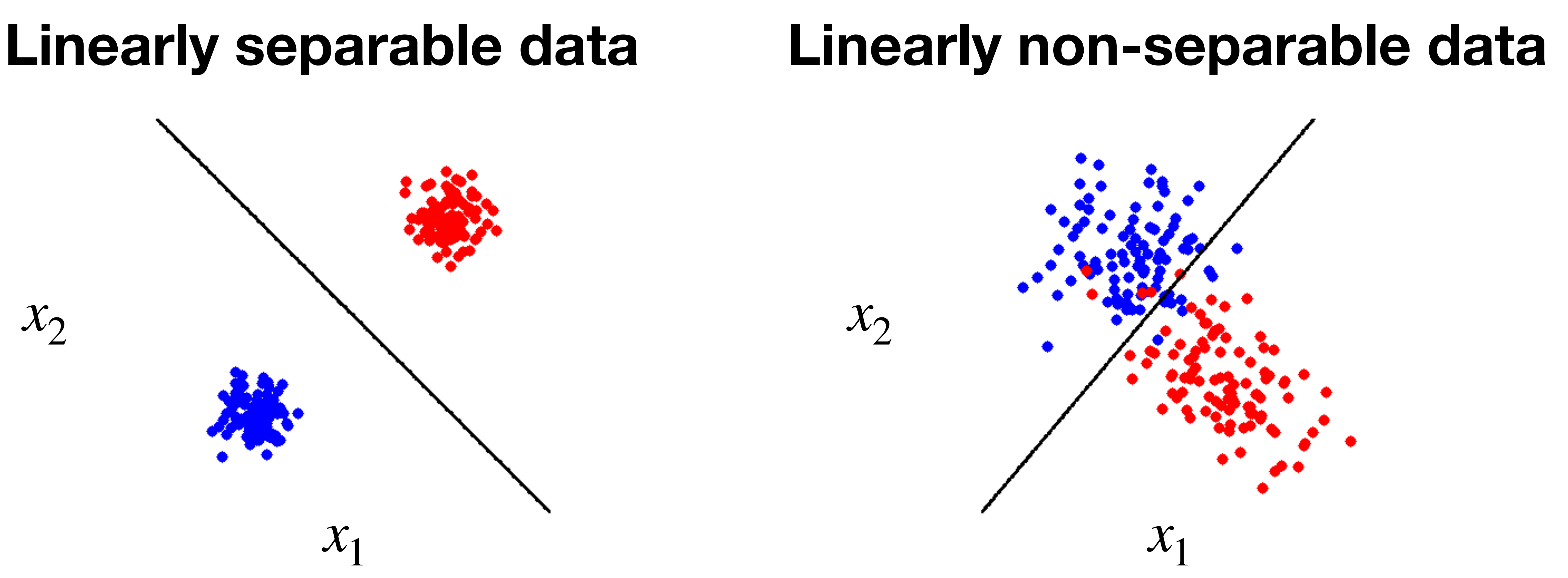
Perceptrons

Separability

Learning perceptrons

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)

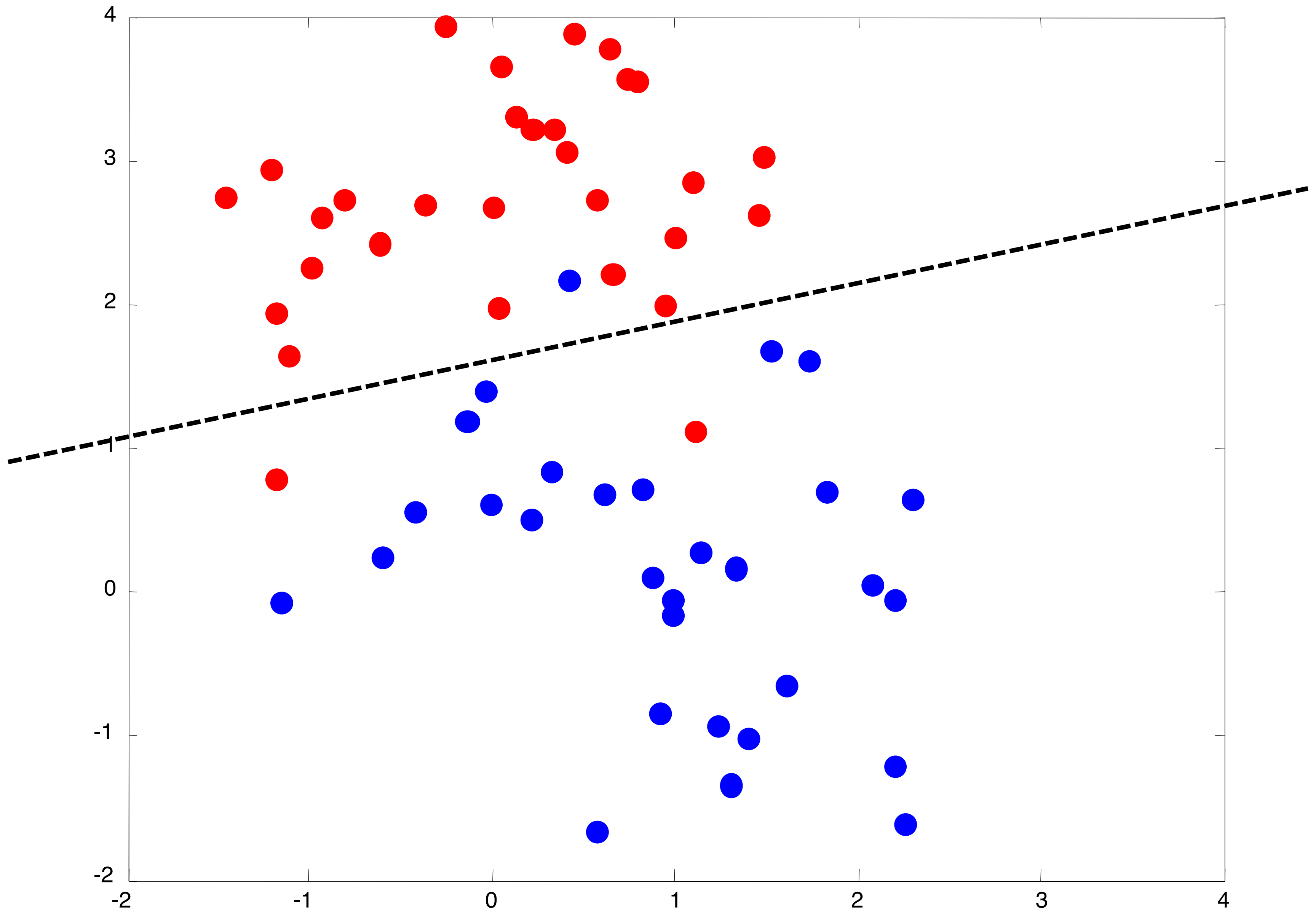


Why do classes overlap?

- Non-separable data means no model can perfectly predicted it
  - Feature ranges for different classes overlap
  - Given an instance in the overlap range — we have uncertainty
- How to improve separation / reduce loss?
  - More complex model class may include a separating model
  - May need more features for that
- Realistically, we must live with some uncertainty / loss
  - But sometimes we can get less of it...

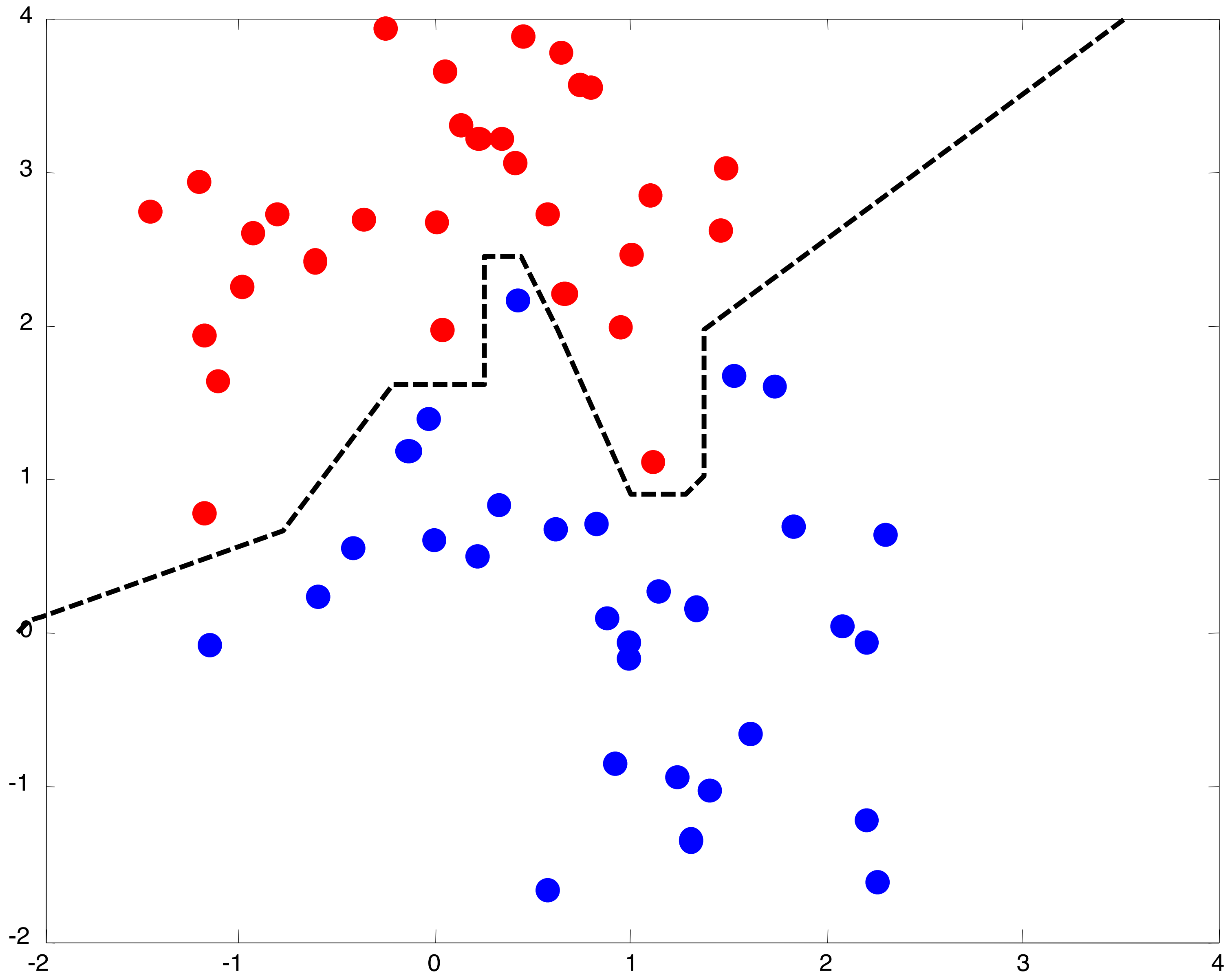
Example: linearly non-separable data

- Data is non-separable with linear classifier



Example: linearly non-separable data

- Data is non-separable with linear classifier
  - ...but separable with non-linear classifier



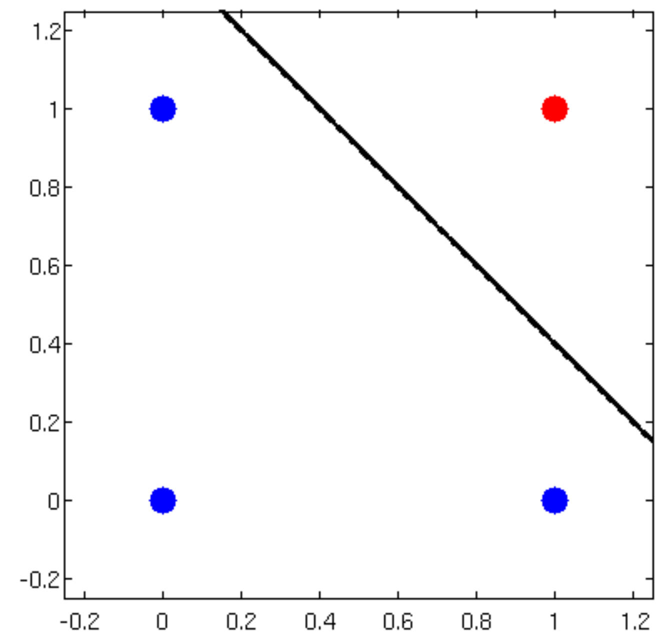
- Is this good? Probably high test loss (overfitting)
  - Problem may be separable by complex model, but no hope of finding a good one

Perceptron: representational power

- A perceptron can represent linearly separable data
- Which functions can a perceptron represent?
  - Those that are linearly separable over all x
- A family of functions that are easy to analyze: boolean functions
  - A perceptron can represent AND but not XOR

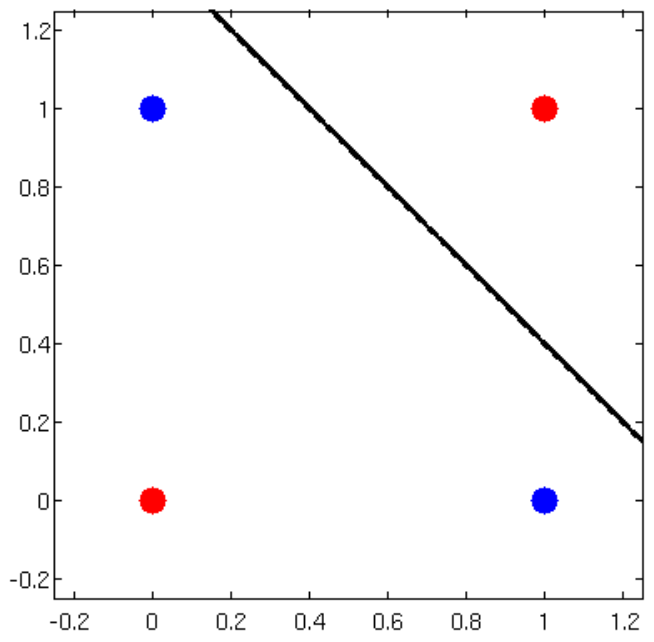
AND

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



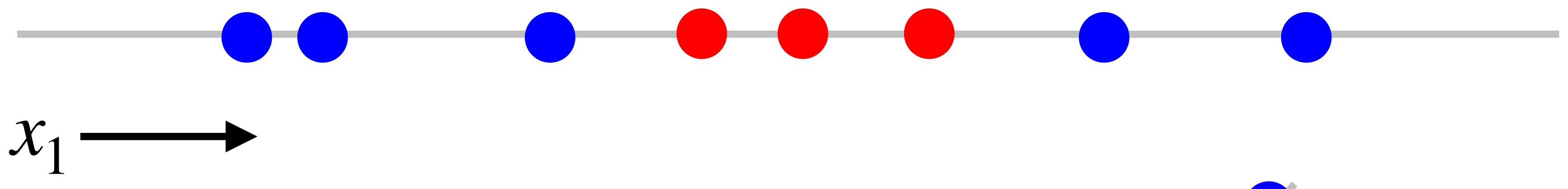
XOR

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

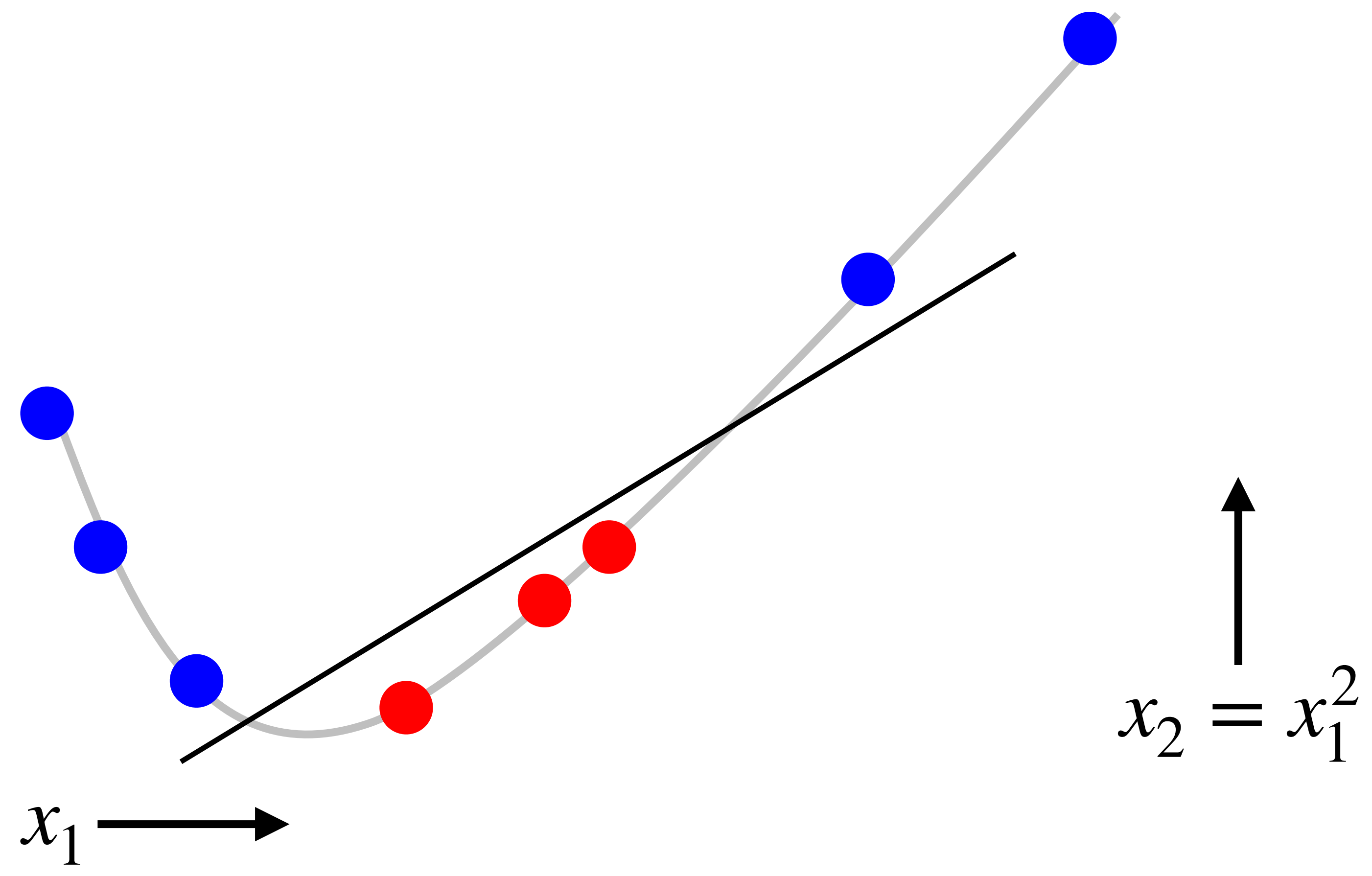


Adding features

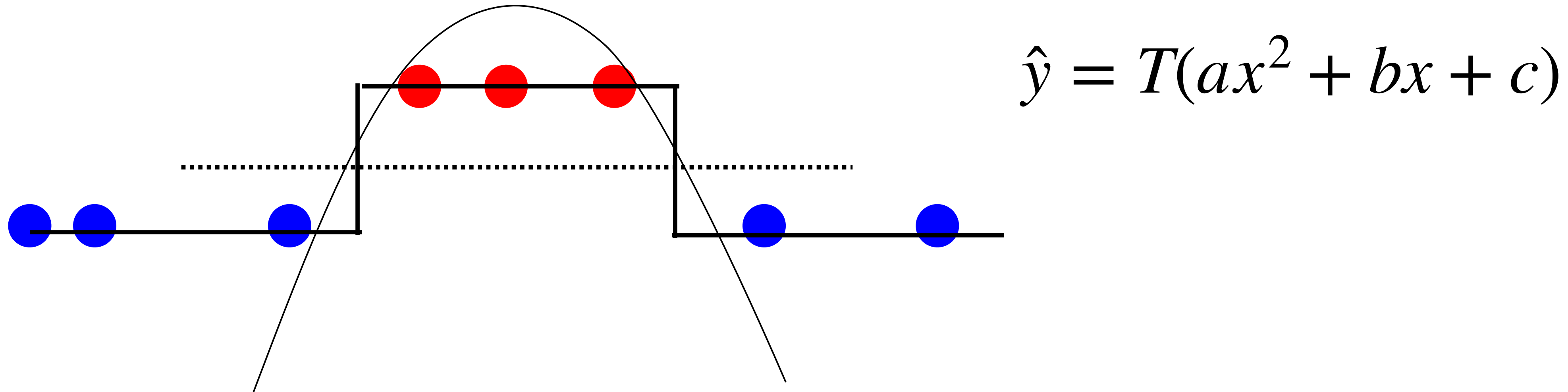
- How to make the perceptron more expressive?
  - Add features — recall linear → polynomial regression
- Linearly non-separable:



- Linearly separable in quadratic features:



- Visualized in original feature space:
  - Decision boundary: $ax^2 + bx + c = 0$



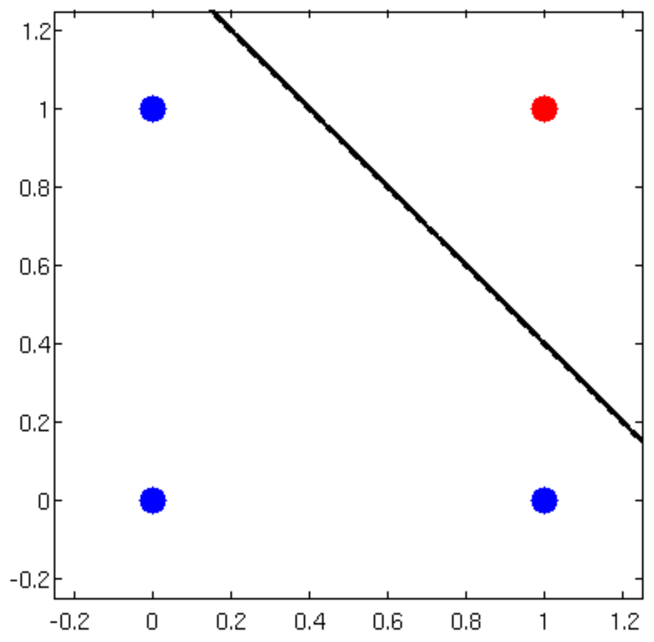
Adding features

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

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

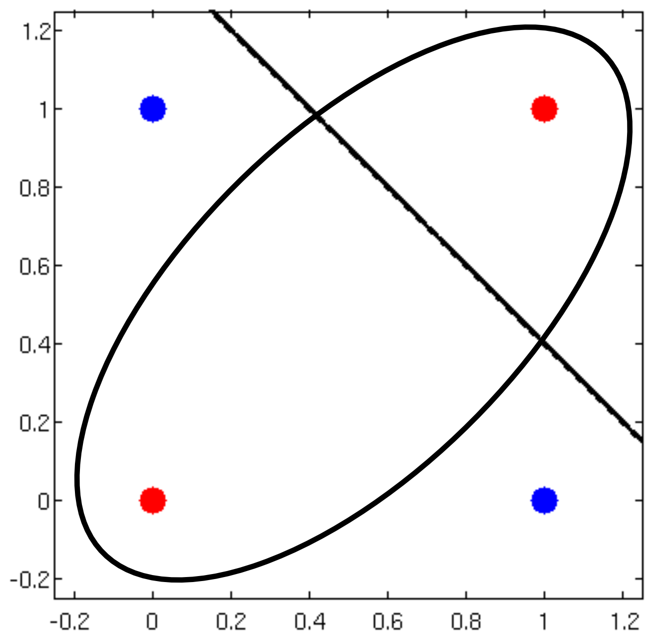
AND

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



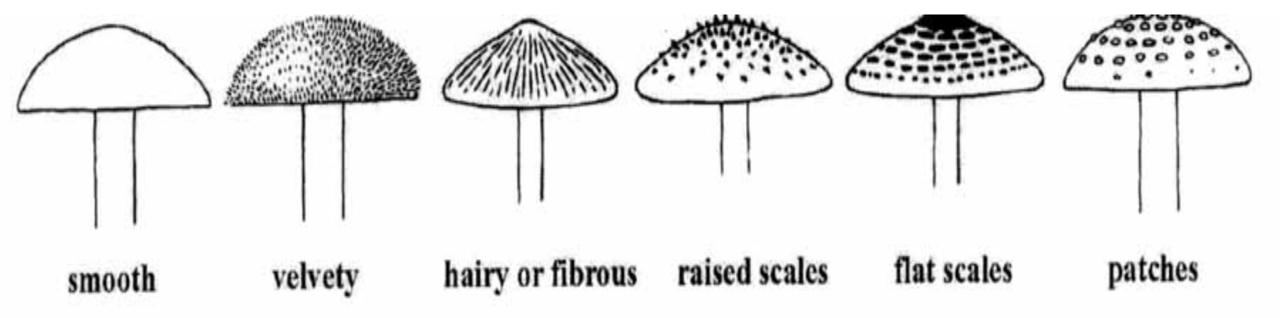
XOR

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



Representing discrete features

- To define "linear" functions of discrete features: represent as real numbers
- Example: classify poisonous mushroom



  - Surface $\in \{\text{fibrous, grooves, scaly, smooth}\}$
  - Represent as $\{1,2,3,4\}$? Is $\text{smooth} - \text{fibrous} = 3(\text{scaly} - \text{grooves})$?
  - Better: one-hot representation: $\{[1000], [0100], [0010], [0001]\}$
    - Requires 4 binary features instead of 1 integer
    - Preserves the original lack of "topology"

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 distribution
  - Perceptron: linear separability
- Adding features:
  - Complex features: complex decision boundary, easier separability
  - Can lead to overfitting

Today's lecture

Regularization and cross-validation

Perceptrons

Separability

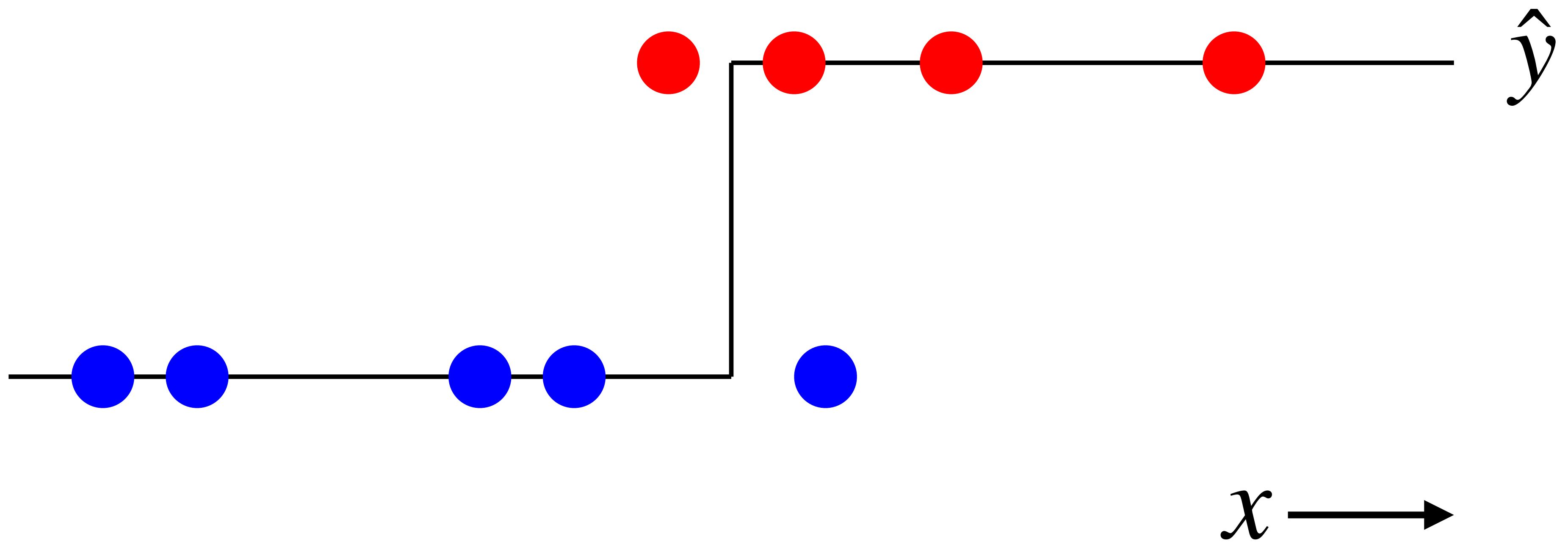
Learning perceptrons

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 <u>training loss</u>

Error rate

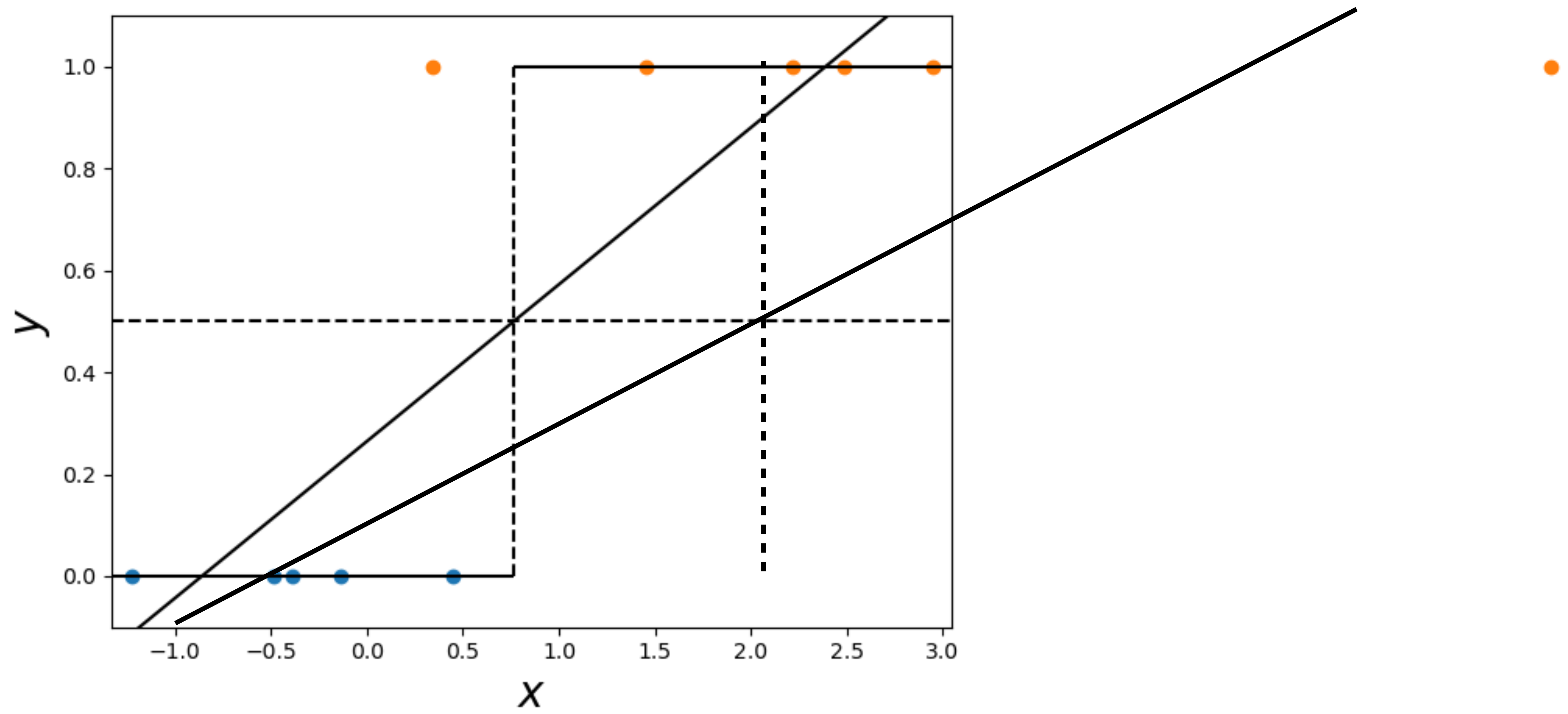
- Error rate: $\mathscr{L}_\theta = \frac{1}{m}\sum_i \delta(y^{(i)} \neq f_\theta(x^{(i)}))$
  - With the indicator $\delta(y \neq \hat{y}) = \begin{cases} 1 & y \neq \hat{y} \\ 0 & else \end{cases}$



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

Use linear regression?

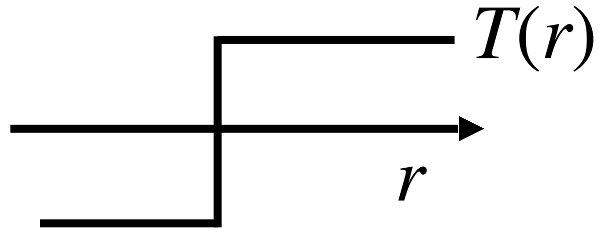
- Idea: find θ using linear regression



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

Perceptron: gradient-based learning

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



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

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

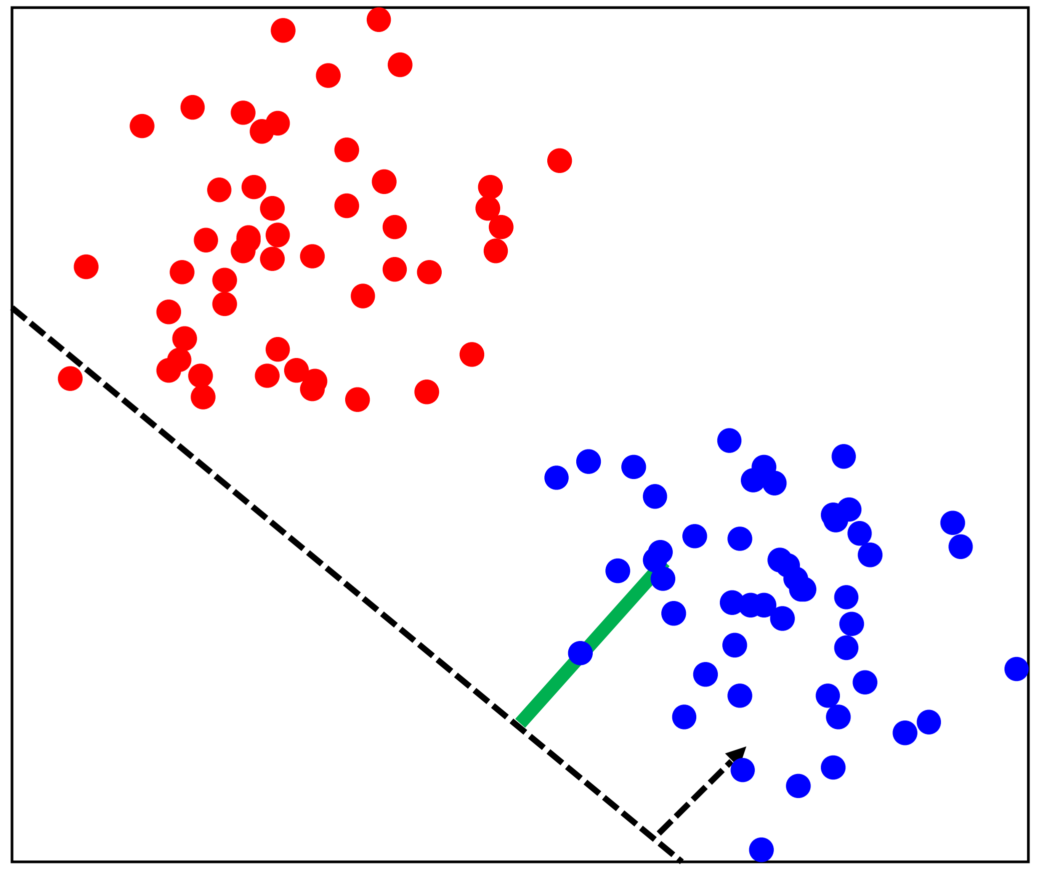
$$\text{while} \neg \text{done:}$$

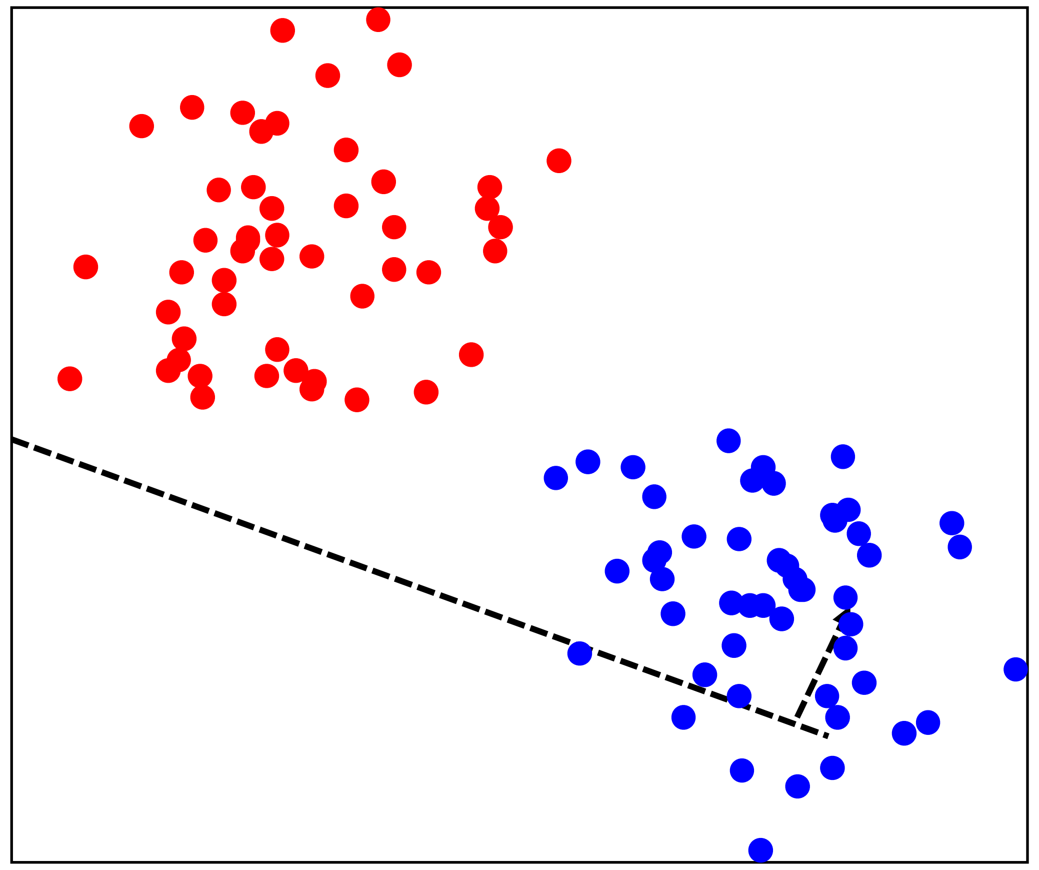
$$\quad \text{for each data point } j\text{:}$$

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

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

incorrect prediction: update weights





Perceptron training algorithm

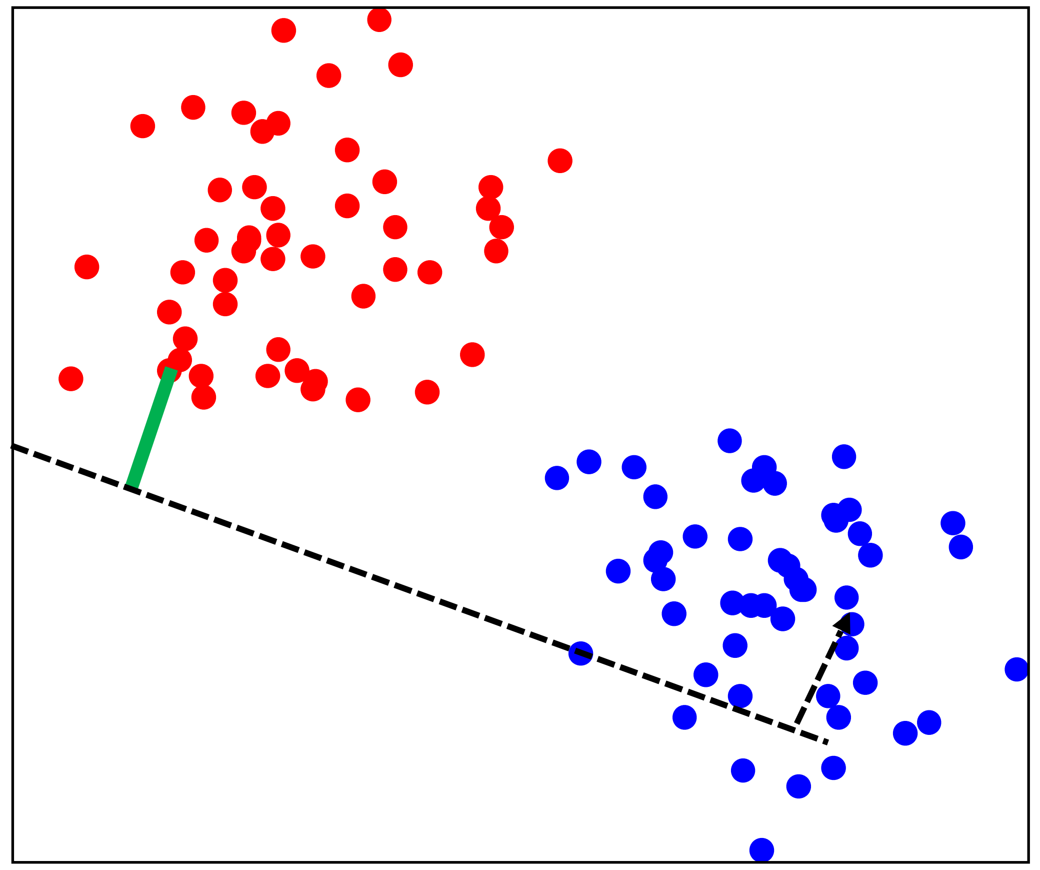
while $\neg$ done:

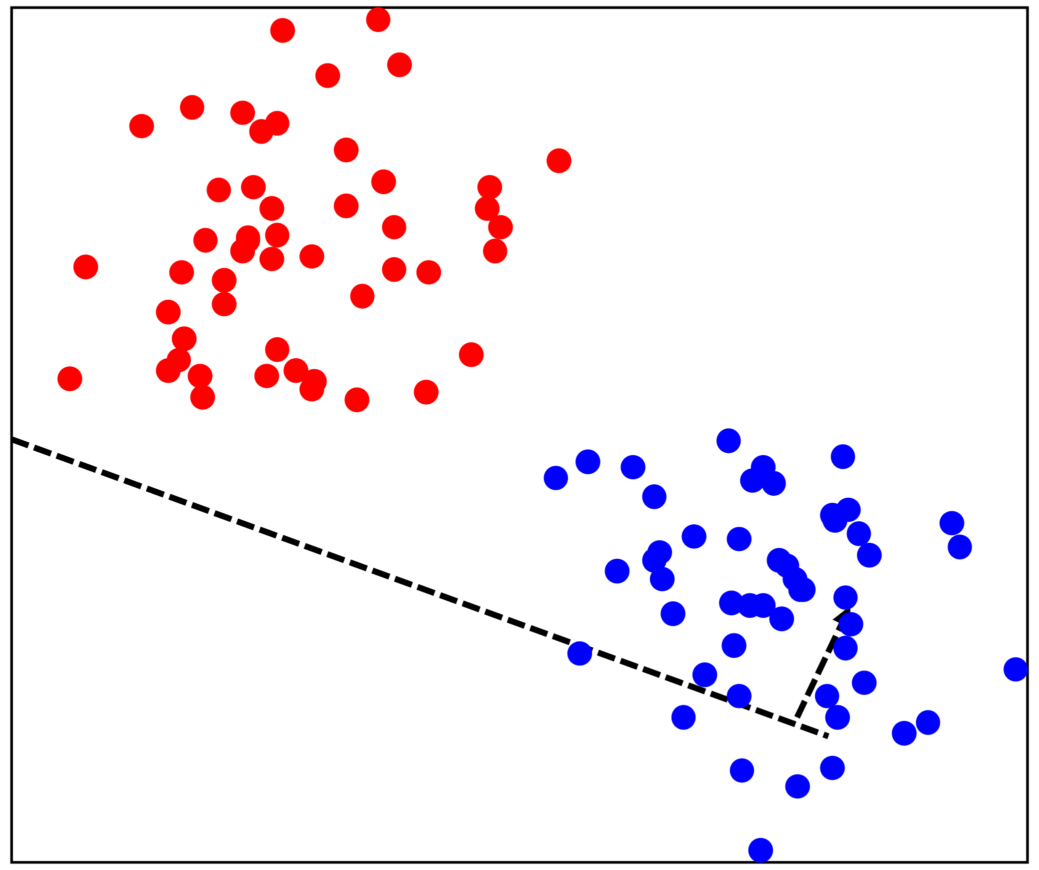
for each data point j:

$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$ **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:

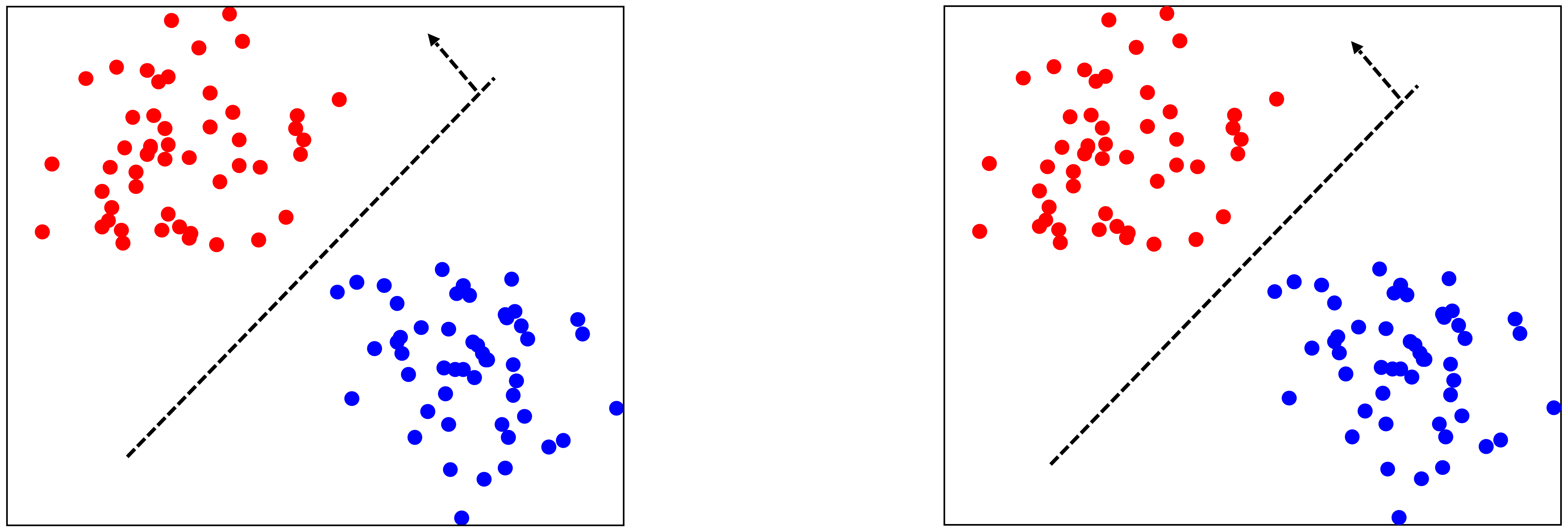
$$\hat{y}^{(j)} = \text{sign}(\theta \cdot x^{(j)})$$

predict output for point *j*

$$\theta \leftarrow \theta + \alpha(y^{(j)} - \hat{y}^{(j)})x^{(j)}$$

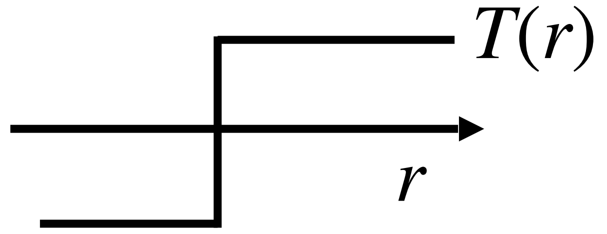
gradient step on weird loss

convergence: no more updates



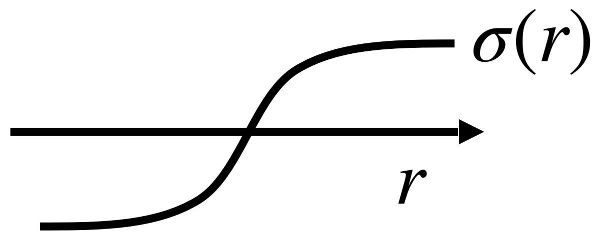
Surrogate loss functions

- Alternative: use differentiable loss function
  - E.g., approximate the step function with a smooth function

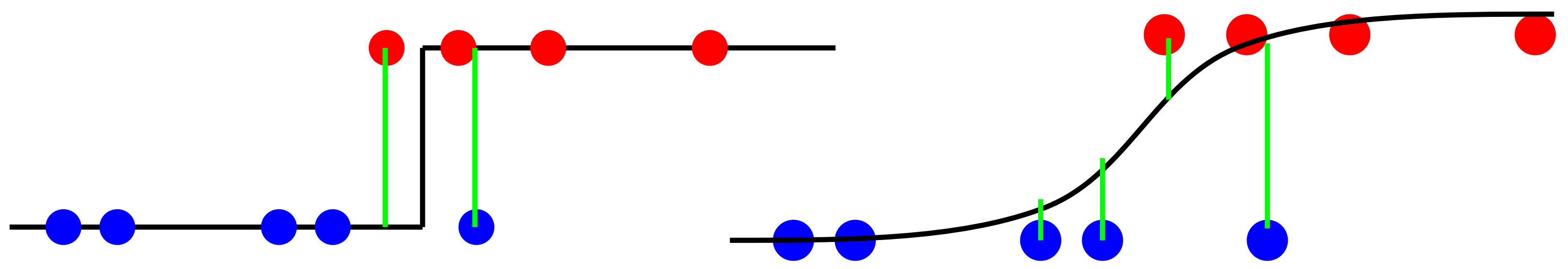


  - Popular choice: logistic / sigmoid function (sigmoid = ”looks like s”)

$$\sigma(r) = \frac{1}{1 + \exp(-r)}$$



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



  - 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

Widening the classification margin

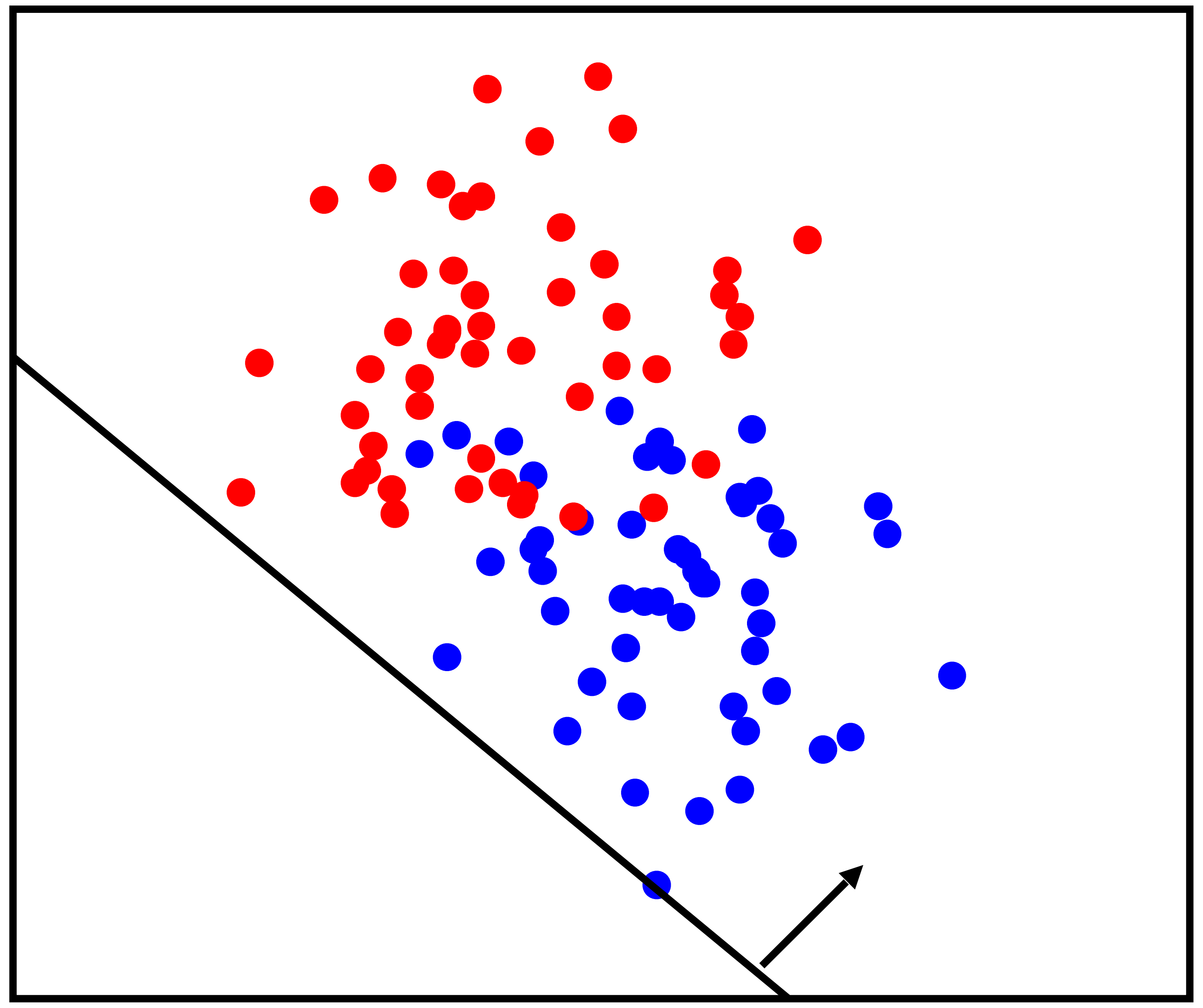
- Which decision boundary is "better"?
  - Both have 0 training loss
  - But one seems more robust, expected to generalize better

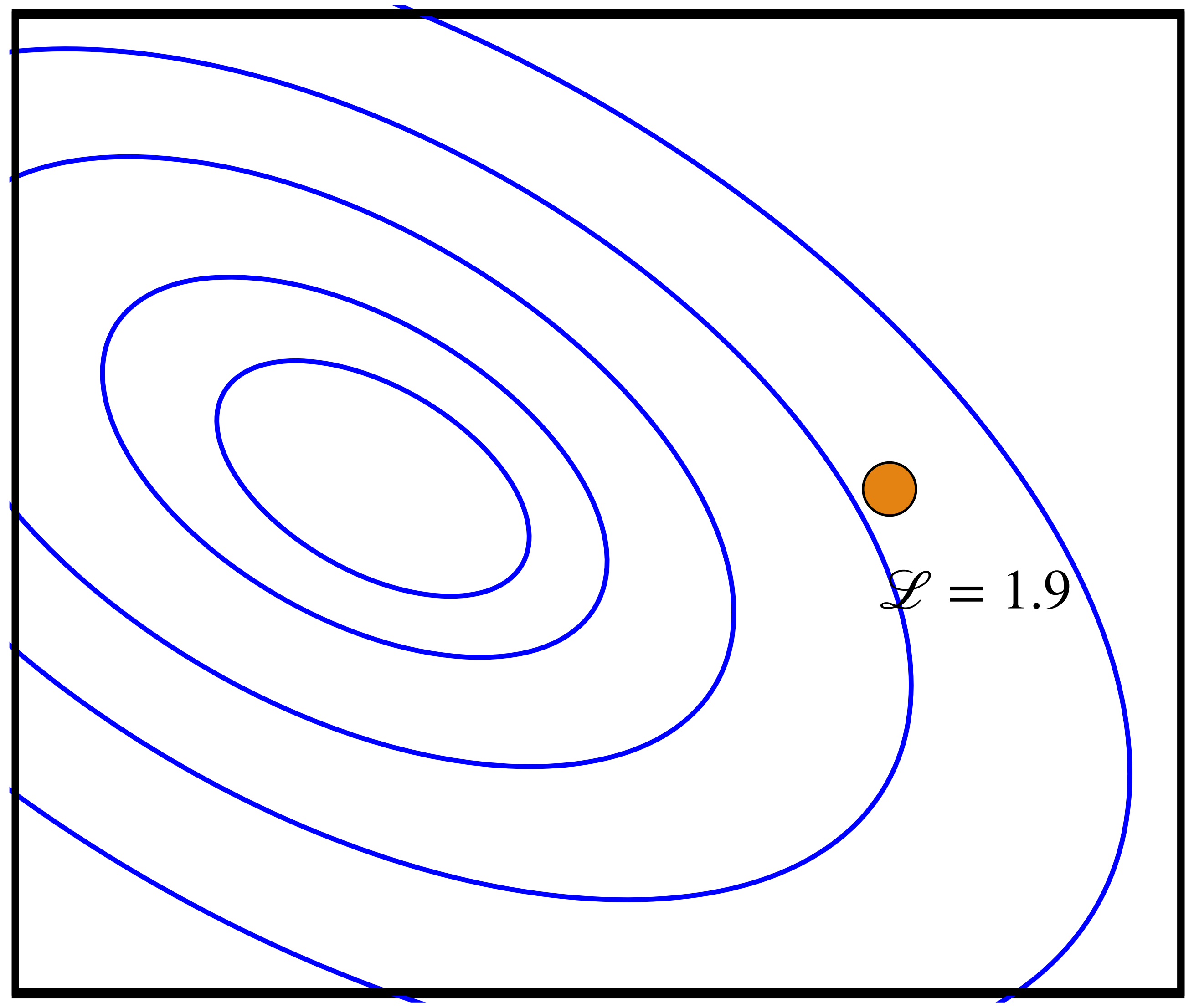
- Benefit of smooth loss function: care about margin
  - Encourage distancing the boundary from data points

Learning smooth linear classifiers

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

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

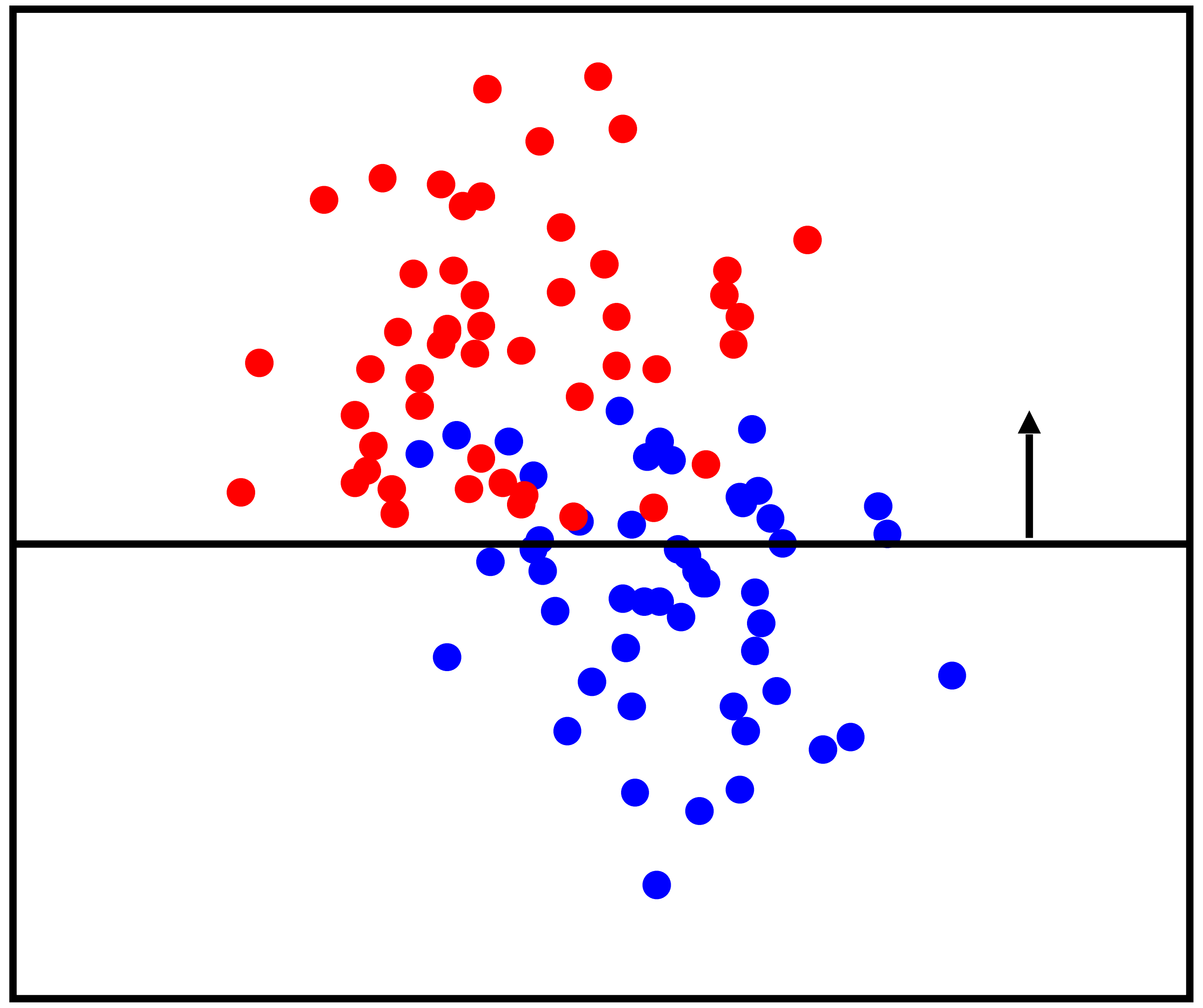


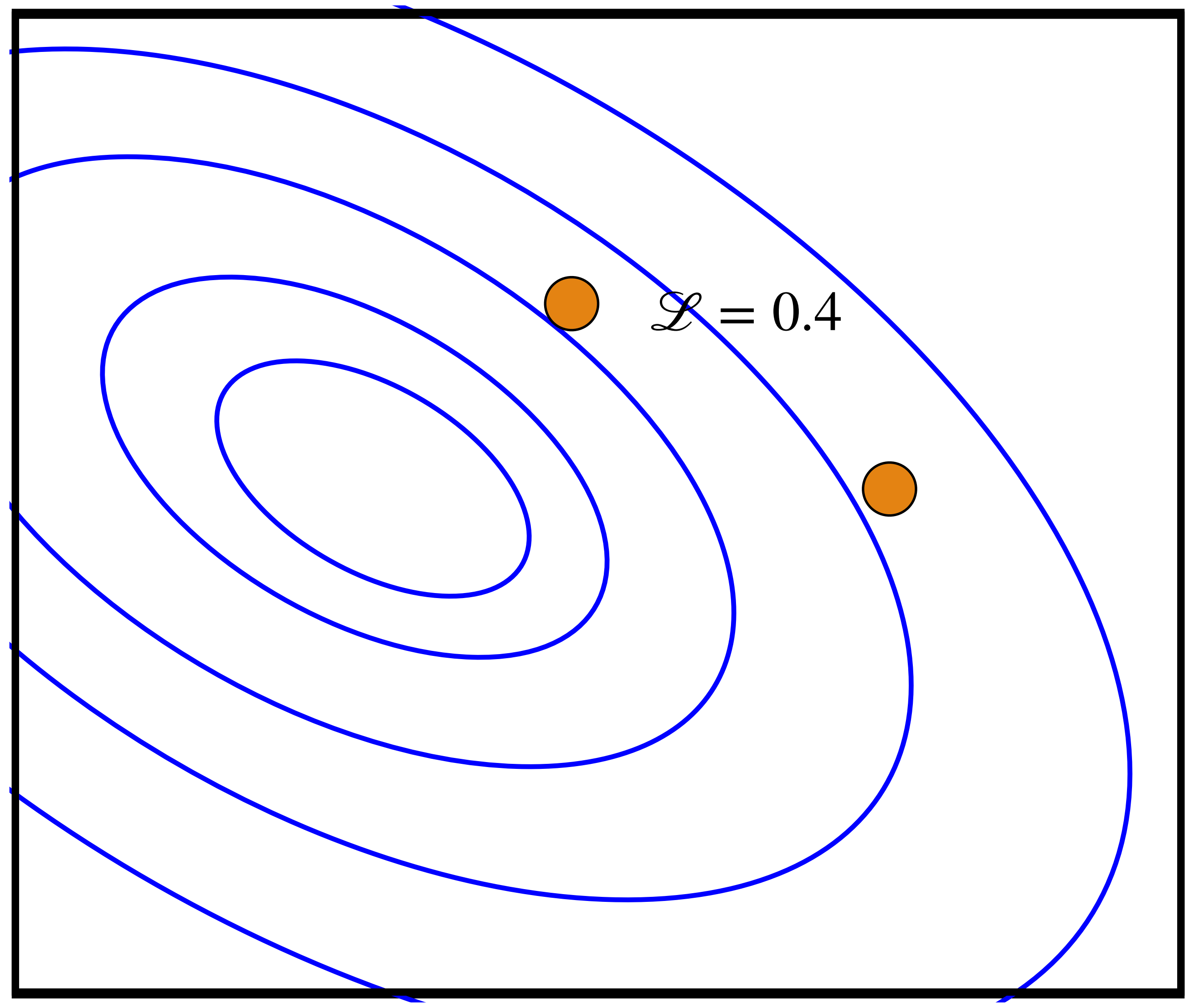


Learning smooth linear classifiers

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

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

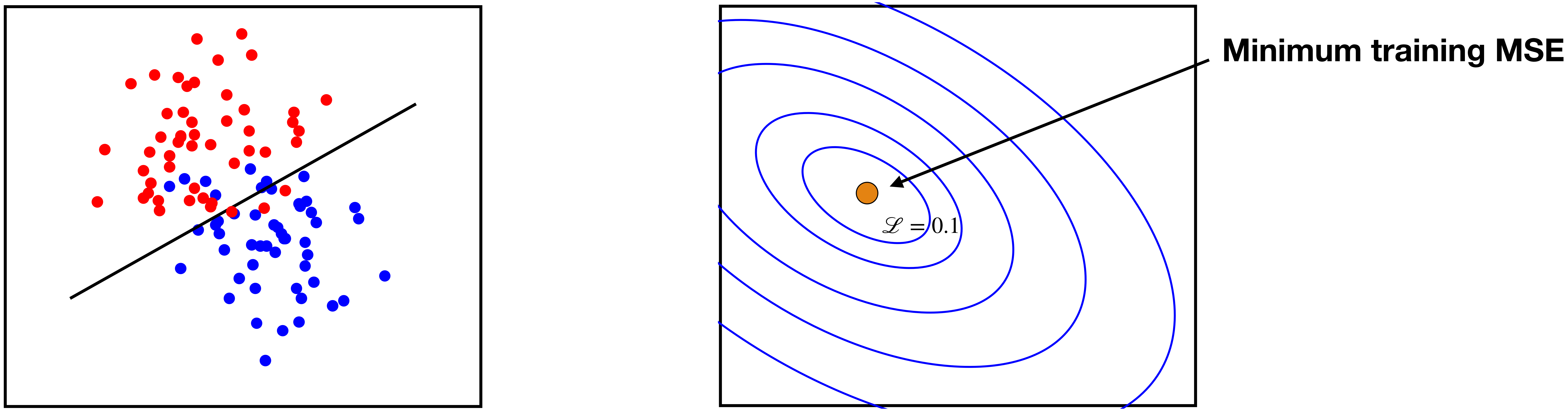




Learning smooth linear classifiers

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

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



Logistics

assignments

- Assignment 2 due next Tuesday, Oct 19

project

- Project guidelines on Canvas
- Team rosters due next Tuesday, Oct 19 on Canvas