UCI University of California, Irvine

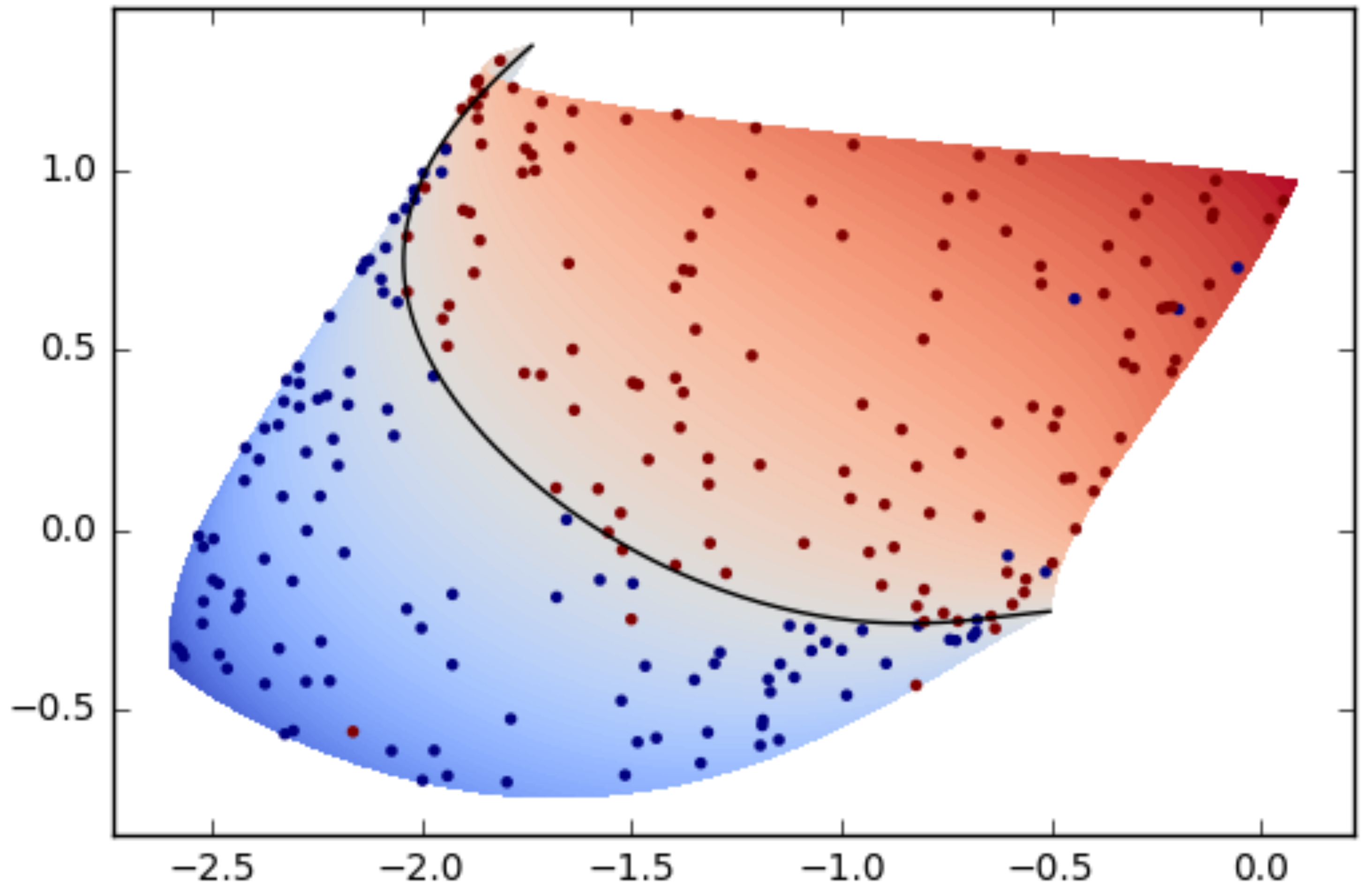
CS 273A: Machine Learning

Winter 2021

Lecture 4: Linear 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

staff

- Emad Naeini is joining the course staff
- Emad's office hours:
  - https://calendly.com/ekasaeya/cs-273a-emad-s-office-hour

assignments

- Assignment 1 due today
- Assignment 2 to be published next week

Today's lecture

ROC curves

Linear regression

Gradient descent

Terminology

- Class prior probabilities: $p(y)$
  - Prior = before seeing any features
- Class-conditional probabilities: $p(x \mid y)$
- Class posterior probabilities: $p(y \mid x)$
- Bayes' rule: $p(y \mid x) = \dfrac{p(y)p(x \mid y)}{p(x)}$
- Law of total probability: $p(x) = \sum_y p(x, y) = \sum_y p(y)p(x \mid y)$

Measuring error

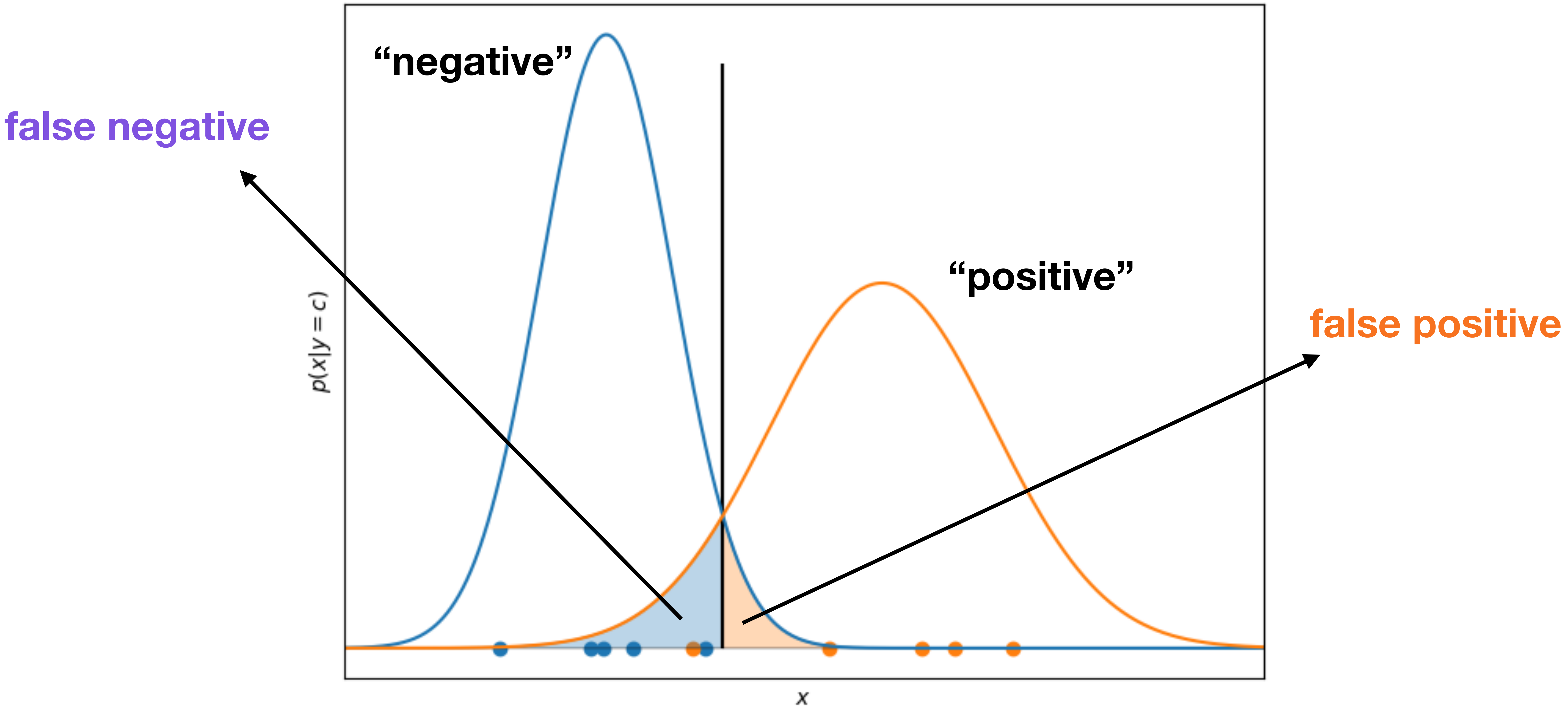
- Confusion matrix: all possible values of $(y, \hat{y})$
- Binary case: **true** / **false** (correct or not) **positive** / **negative** (prediction)

	Predict 0	Predict 1
Y=0	380 **TN**	5 **FP**
Y=1	338 **FN**	3 **TP**

  - Accuracy: $\dfrac{TP + TN}{TP + TN + FP + FN} = 1 -$ error rate
  - True positive rate (TPR): $\hat{p}(\hat{y} = 1 \,|\, y = 1) = \dfrac{\#(y = 1, \hat{y} = 1)}{\#(y = 1)}$ (aka, sensitivity)
  - False negative rate (FNR): $\hat{p}(\hat{y} = 0 \,|\, y = 1) = \dfrac{\#(y = 1, \hat{y} = 0)}{\#(y = 1)}$
  - False positive rate (FPR): $\hat{p}(\hat{y} = 1 \,|\, y = 0) = \dfrac{\#(y = 0, \hat{y} = 1)}{\#(y = 0)}$
  - True negative rate (TNR): $\hat{p}(\hat{y} = 0 \,|\, y = 0) = \dfrac{\#(y = 0, \hat{y} = 0)}{\#(y = 0)}$ (aka, specificity)

Types of error

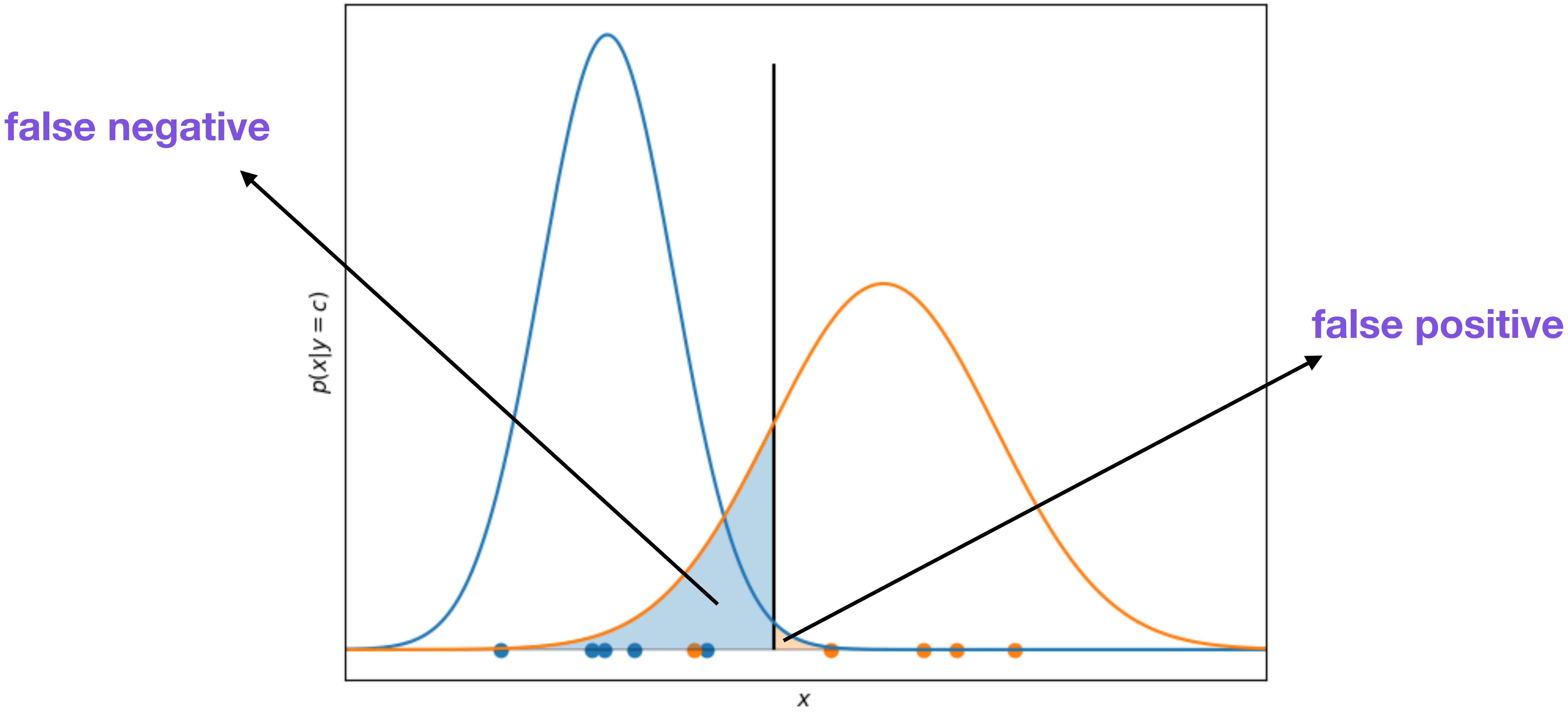
- Not all errors are equally bad
  - Do some cost more? (e.g. red / green light, diseased / healthy)



- False negative rate: $\frac{p(y=1,\hat{y}=0)}{p(y=1)}$; false positive rate: $\frac{p(y=0,\hat{y}=1)}{p(y=0)}$

Cost of error

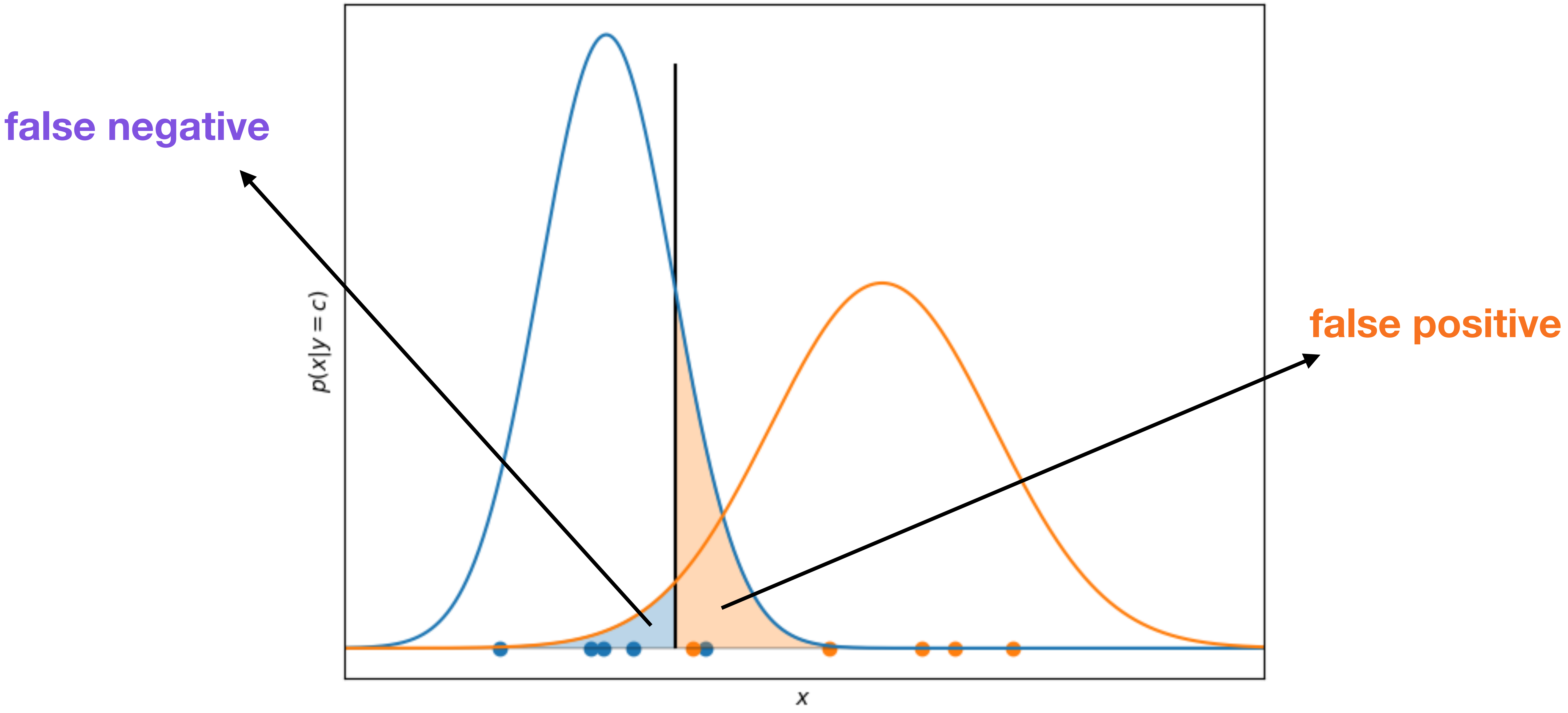
- Weight different costs differently
  - $\alpha \cdot p(y=0)p(x \mid y=0) \lessgtr p(y=1)p(x \mid y=1)$



- Increase α to prefer class 0 — increase FNR, decrease FPR

Cost of error

- Weight different costs differently
  - $\alpha \cdot p(y=0)p(x \mid y=0) \lessgtr p(y=1)p(x \mid y=1)$



- Decrease α to prefer class 1 — decrease FNR, increase FPR

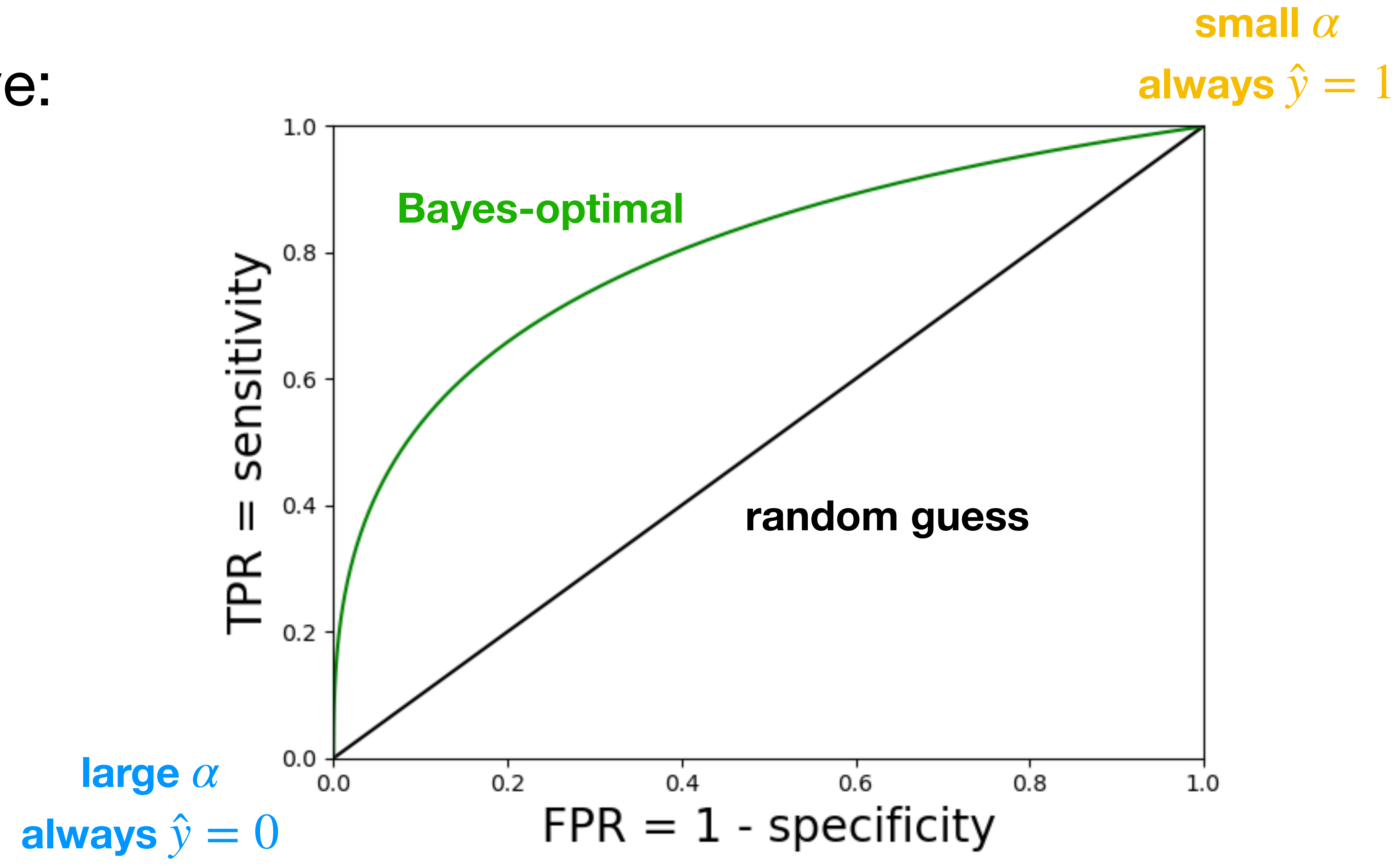
Bayes-optimal decision

- Maximum posterior decision: $\hat{p}(y=0\,|\,x) \lessgtr \hat{p}(y=1\,|\,x)$
  - Optimal for the error-rate (0–1) loss: $\mathbb{E}_{x,y\sim p}[\hat{y}(x) \neq y]$
- What if we have different cost for different errors? α_{FP}, α_{FN}
  - $\mathscr{L} = \mathbb{E}_{x,y\sim p}[\alpha_{\text{FP}} \cdot \#(y=0, \hat{y}(x)=1) + \alpha_{\text{FN}} \cdot \#(y=1, \hat{y}(x)=0)]$
- Bayes-optimal decision: $\alpha_{\text{FP}} \cdot \hat{p}(y=0\,|\,x) \lessgtr \alpha_{\text{FN}} \cdot \hat{p}(y=1\,|\,x)$
  - Log probability ratio: $\log \dfrac{\hat{p}(y=1\,|\,x)}{\hat{p}(y=0\,|\,x)} \lessgtr \log \dfrac{\alpha_{\text{FP}}}{\alpha_{\text{FN}}} = \alpha$

ROC curve

- Often models have a "knob" for tuning preference over classes (e.g. α)
  - Changing the decision boundary to include more instances in preferred class
- Characteristic performance curve:

$$\log \frac{\hat{p}(y=1\,|\,x)}{\hat{p}(y=0\,|\,x)} \lessgtr \alpha$$

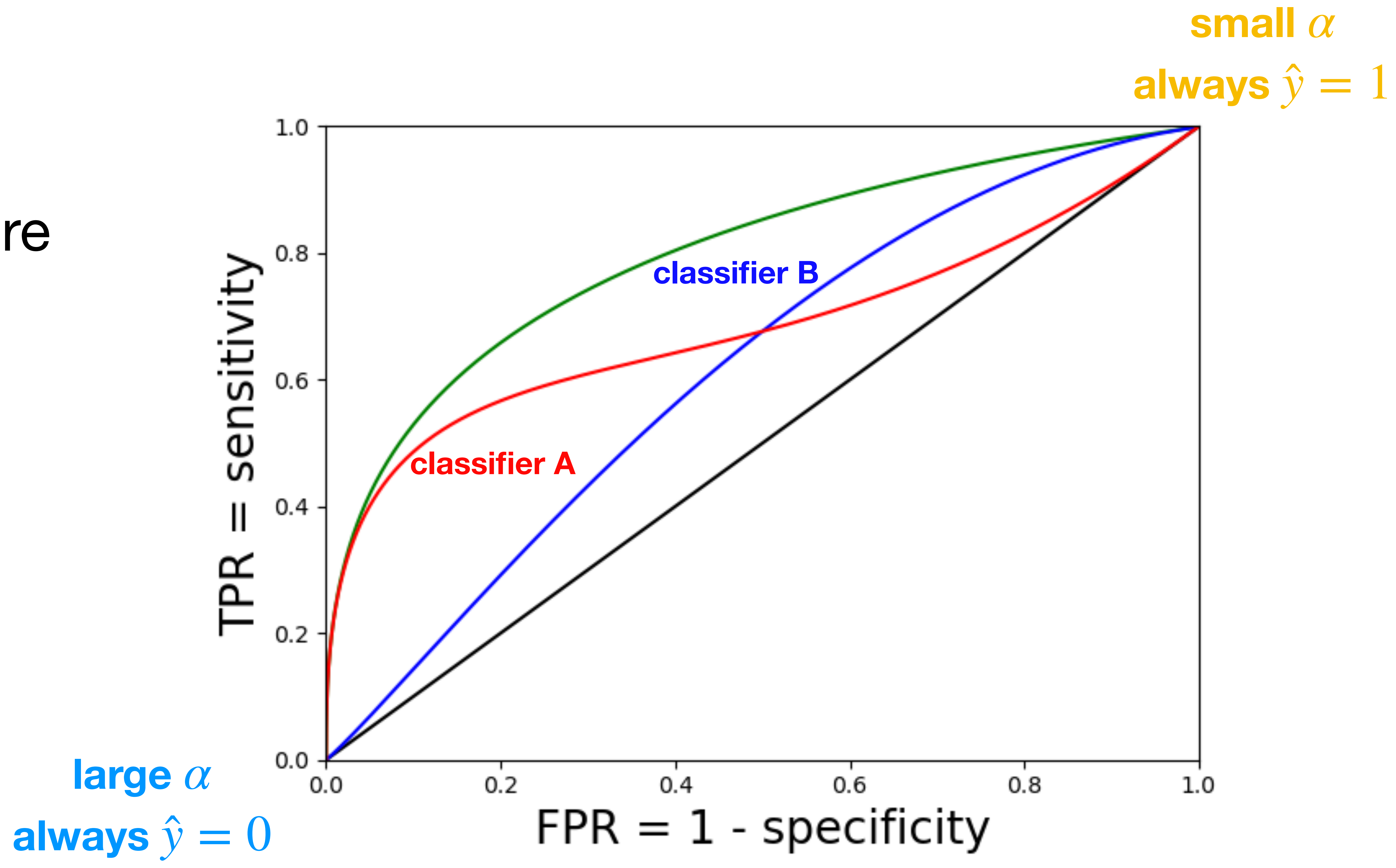


Demonstration

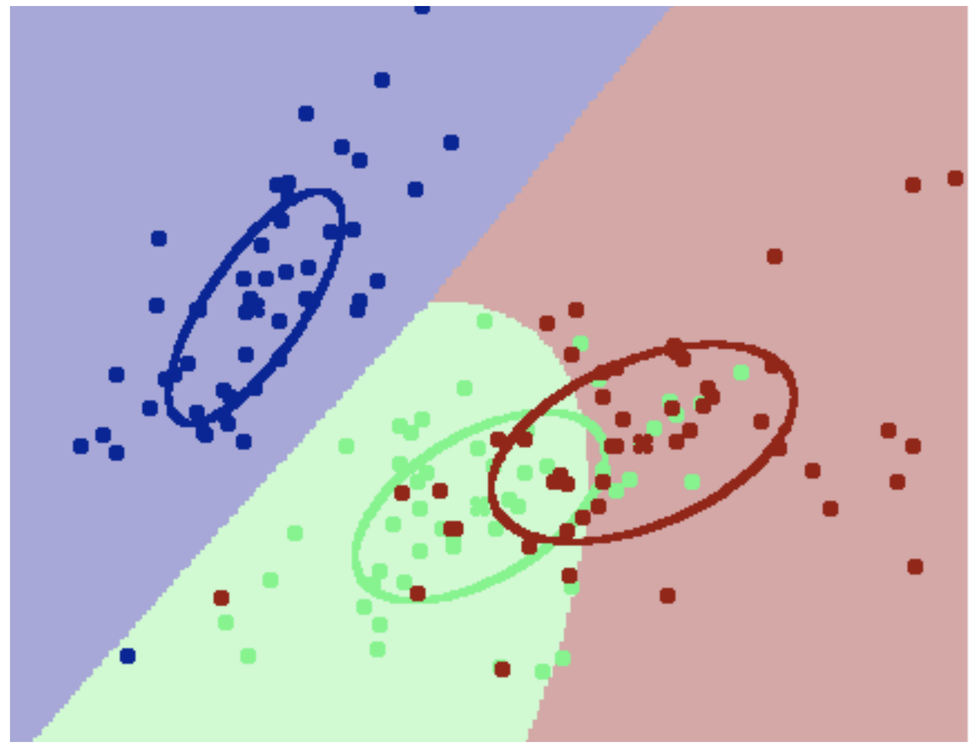
- http://www.navan.name/roc

Comparing classifiers

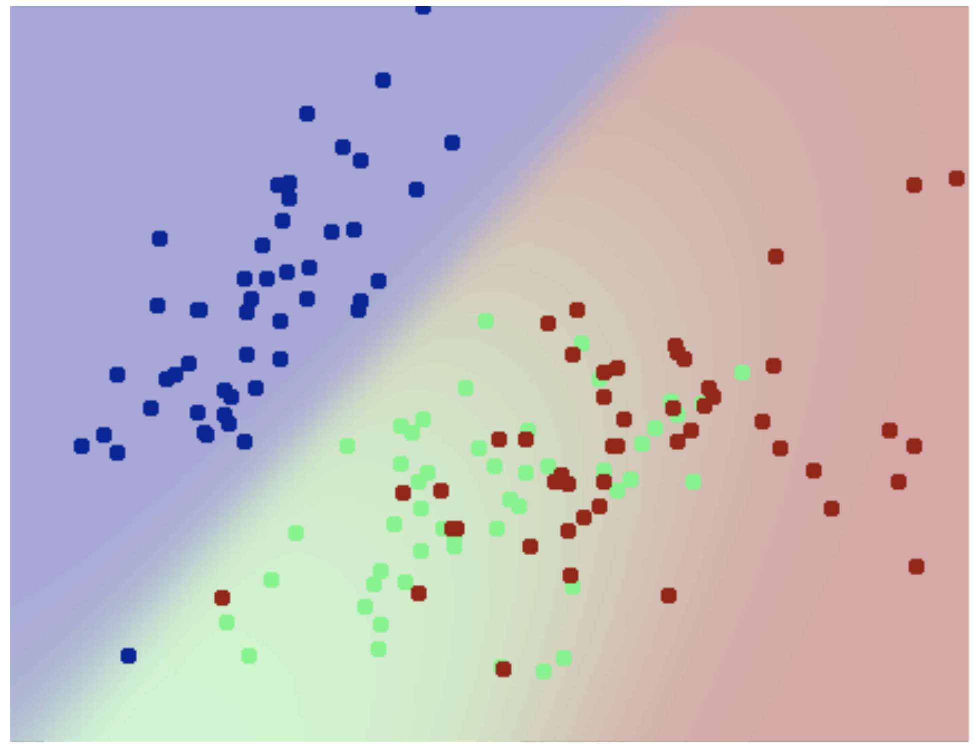
- Which classifier performs "better"?
  - A is better for high specificity
  - B is better for high sensitivity
  - Need single performance measure
- Area Under Curve (AUC)
  - 0.5 ≤ AUC ≤ 1
  - AUC = 0.5: random guess
  - AUC = 1: no errors



Discriminative vs. probabilistic predictions



discriminative predictions $\hat{y}(x)$



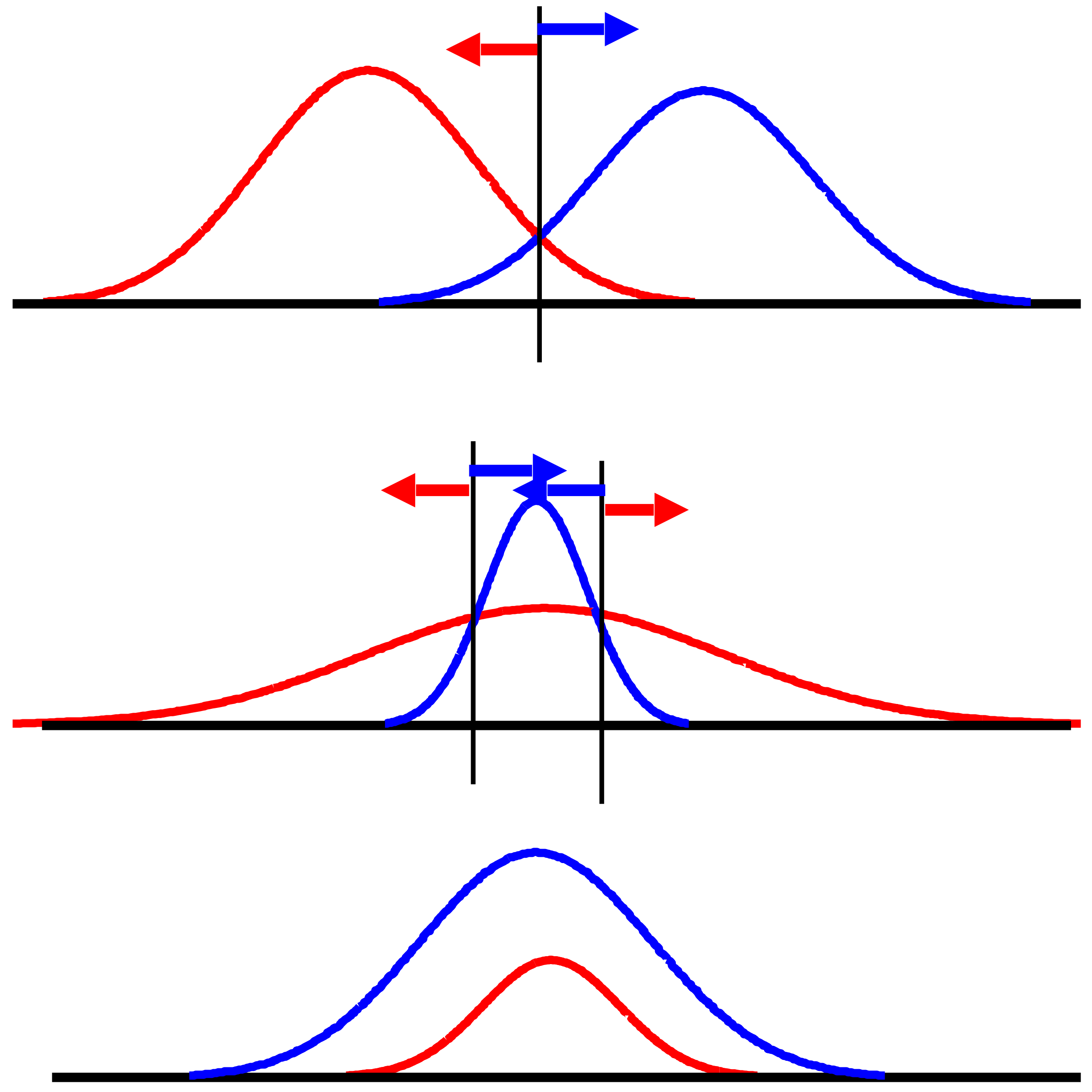
probabilistic predictions $p(y \mid x)$

```
>> learner = gaussianBayesClassify(X,Y)  % build a classifier
>> Ysoft = predictSoft(learner, X)       %  M x C matrix of confidences
>> plotSoftClassify2D(learner,X,Y)       %  shaded confidence plot
```

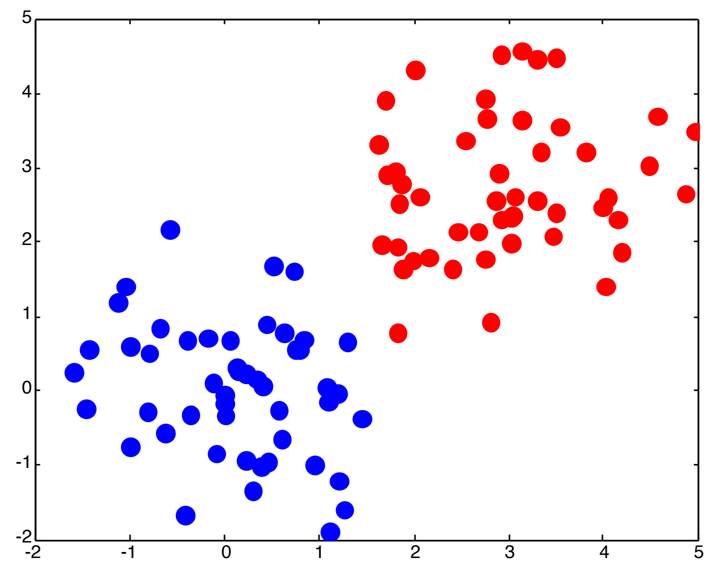
- Probabilistic learning gives more nuanced prediction
  - Can use $p(y \mid x)$ to find $\hat{y}(x) = \arg\max_y p(y \mid x)$ (if argmax is feasible)
  - Express confidence in predicting $\hat{y}$
  - Conditional models: $p(y \mid x)$; vs. generative models: $p(x, y)$
    - Can be used to generate x
    - Bayes classifiers, Naïve Bayes classifiers are generative

Gaussian models

- Bayes-optimal decision:
  - Scale each Gaussian by prior $p(y)$ and relative cost of error
  - Choose the larger scaled probability density
- Decision boundary = where scaled probabilities equal



Gaussian models



- Consider binary classifier with Gaussian conditionals
  - $p(x \mid y = c) = \mathcal{N}(x; \mu_c, \Sigma_c) = (2\pi)^{-\frac{d}{2}} |\Sigma_c|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \mu_c)^\intercal \Sigma_c^{-1}(x - \mu_c)\right)$
  - Assume same covariance $\Sigma_0 = \Sigma_1$
- What is the shape of the decision boundary $p(y = 0 \mid x) = p(y = 1 \mid x)$?

$$\alpha \lessgtr \log \frac{p(y=1)p(x \mid y=1)}{p(y=0)p(x \mid y=0)} = \frac{p(y=1)}{p(y=0)} + \text{const}$$

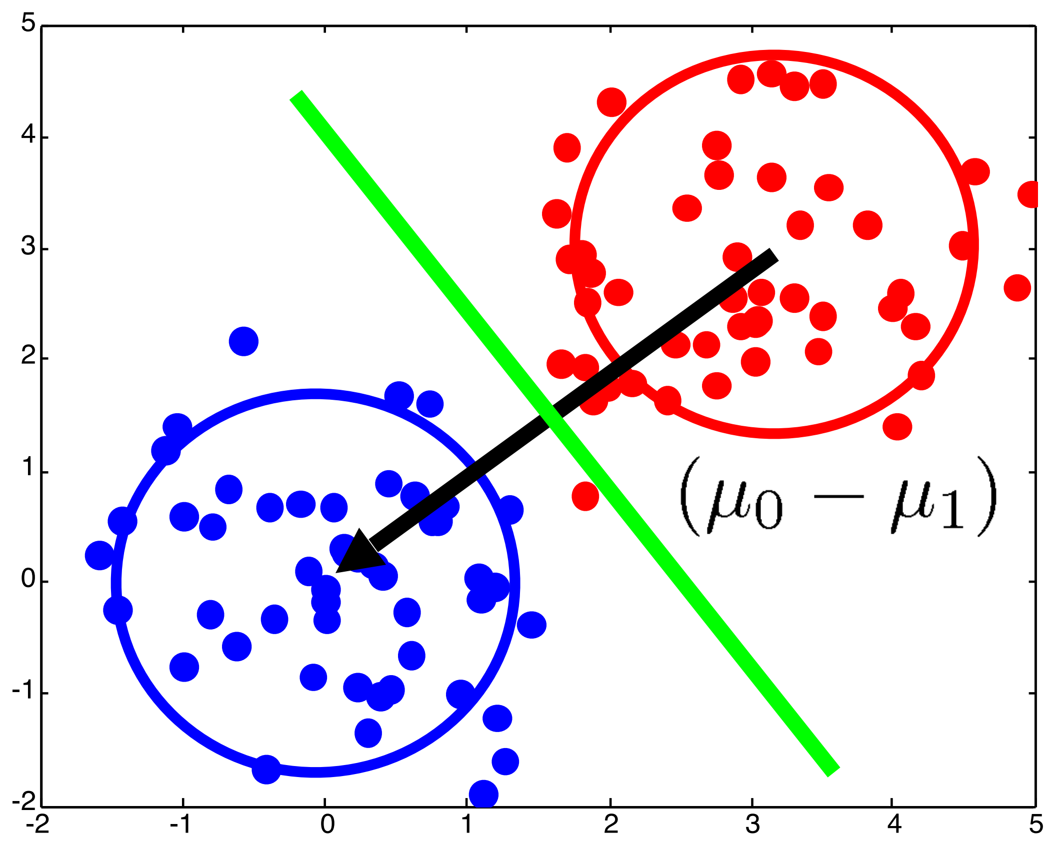
$$+\frac{1}{2}\left(x^\intercal \Sigma^{-1} x - 2\mu_0^\intercal \Sigma^{-1} x + \mu_0^\intercal \Sigma^{-1} \mu_0\right)$$

$$-\frac{1}{2}\left(x^\intercal \Sigma^{-1} x - 2\mu_1^\intercal \Sigma^{-1} x + \mu_1^\intercal \Sigma^{-1} \mu_1\right)$$

$$= \frac{1}{2}(\mu_1 - \mu_0)^\intercal \Sigma^{-1} x + \text{const}$$ ⟵ **linear!**

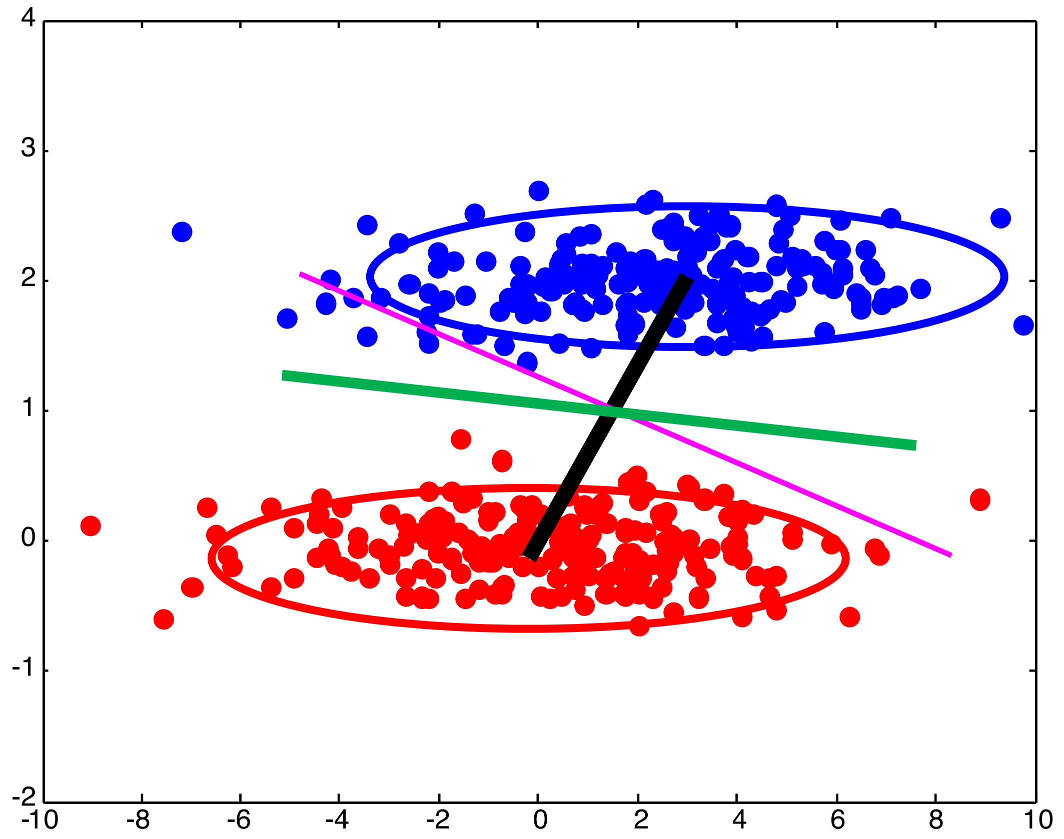
Gaussian models

- Isotropic covariance: $\Sigma = \sigma^2 I_d$
  - Decision: $(\mu_1 - \mu_0)^\intercal x \lessgtr \alpha$
  - Decision boundary perpendicular to segment between means



- General (but equal) covariance:
  - Decision boundary linear, but
    - scaled, if Σ has different eigenvalues
    - rotated, if Σ is not diagonal

$$\Sigma = \begin{bmatrix} 3 & 0 \\ 0 & .25 \end{bmatrix}$$



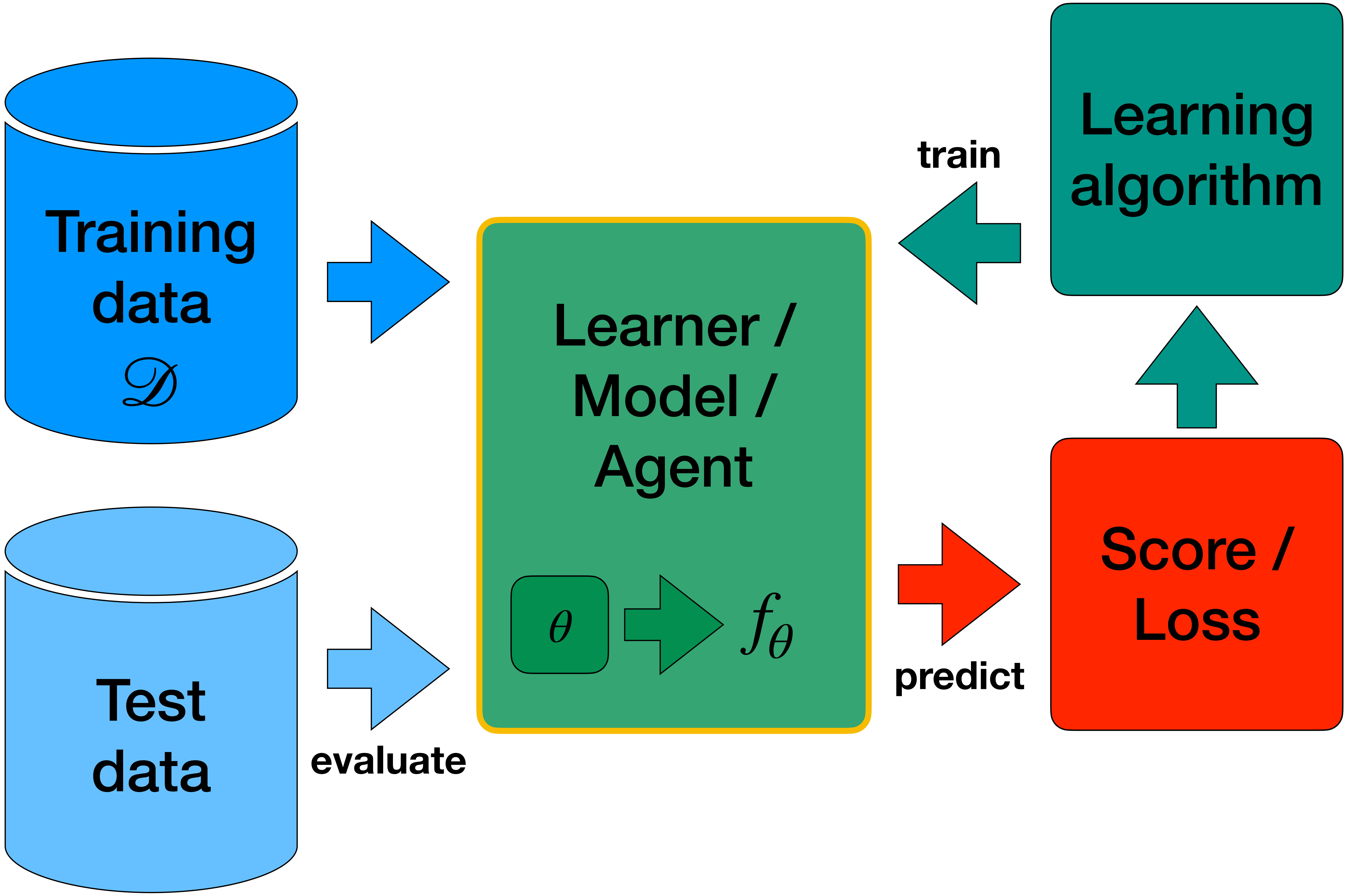
Today's lecture

ROC curves

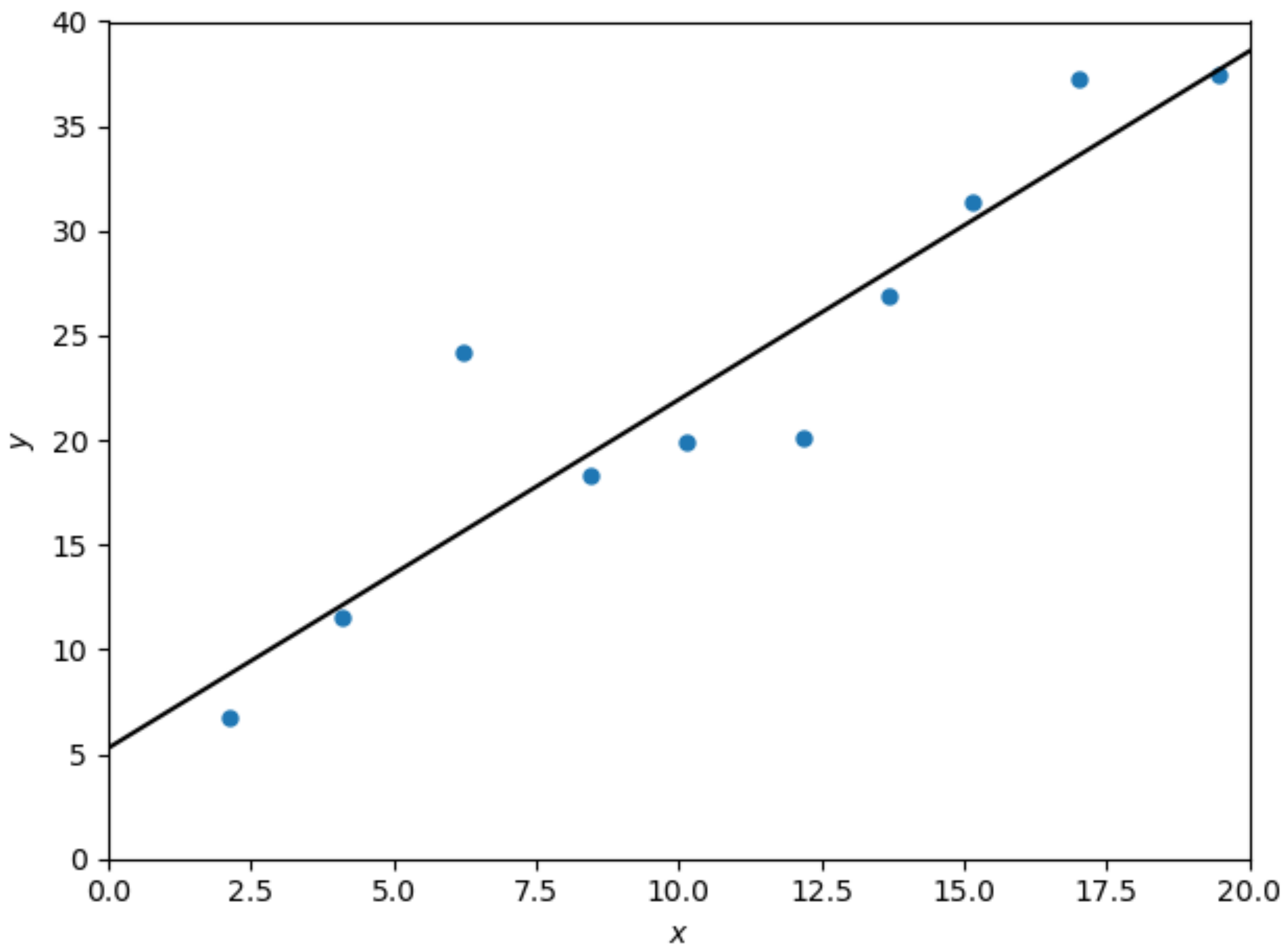
Linear regression

Gradient descent

Machine learning



Linear regression



- Decision function $f : x \mapsto y$ is linear, $f(x) = \theta_0 + \theta_1 x$
- f is stored by its parameters $\theta = \begin{bmatrix} \theta_0 & \theta_1 \end{bmatrix}$

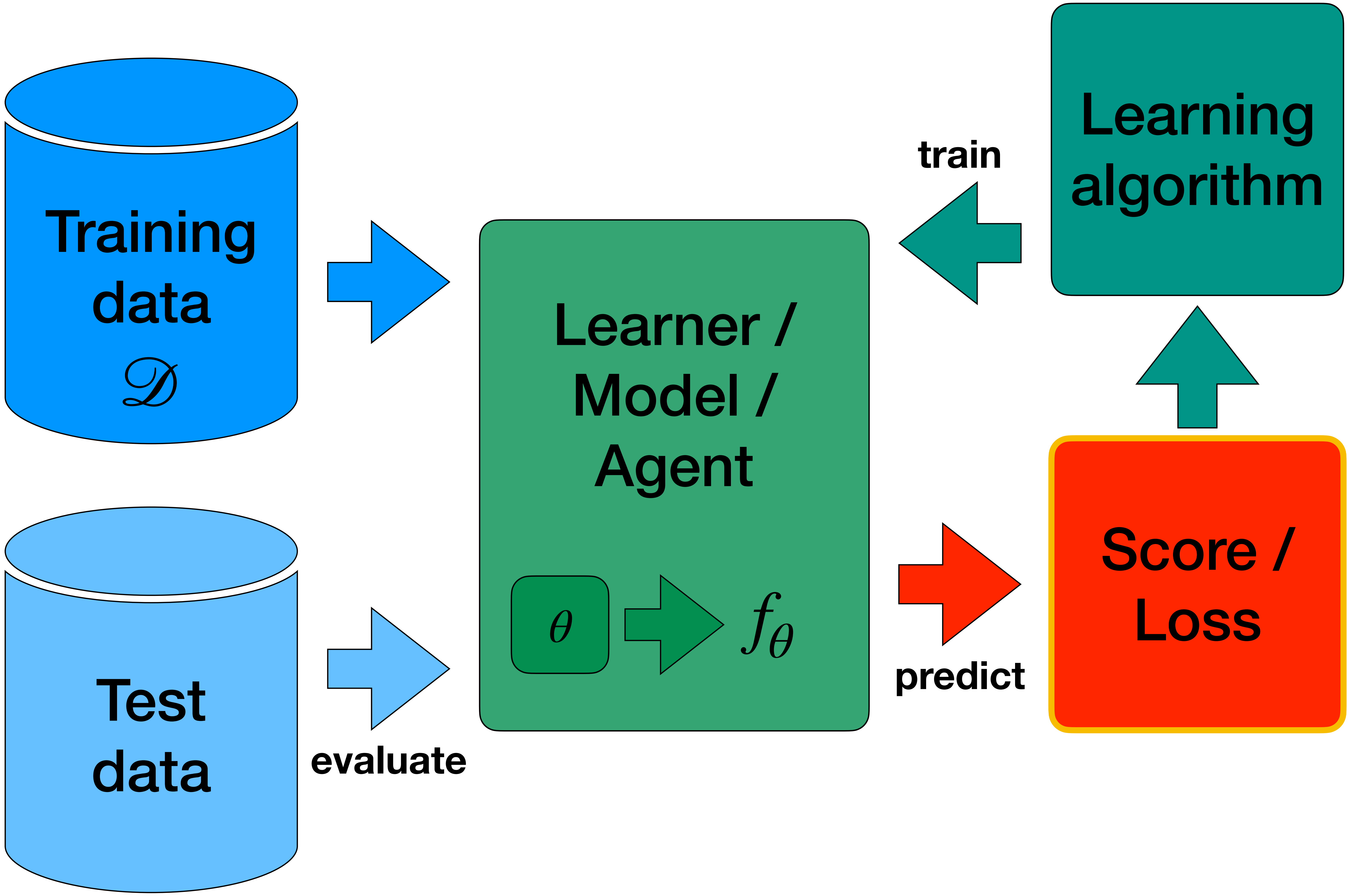
Linear regression

- More generally: $\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_n x_n$

- Define dummy feature $x_0 = 1$ for the shift / bias θ_0

  - $\hat{y}(x) = \theta^\intercal x$; where $\quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

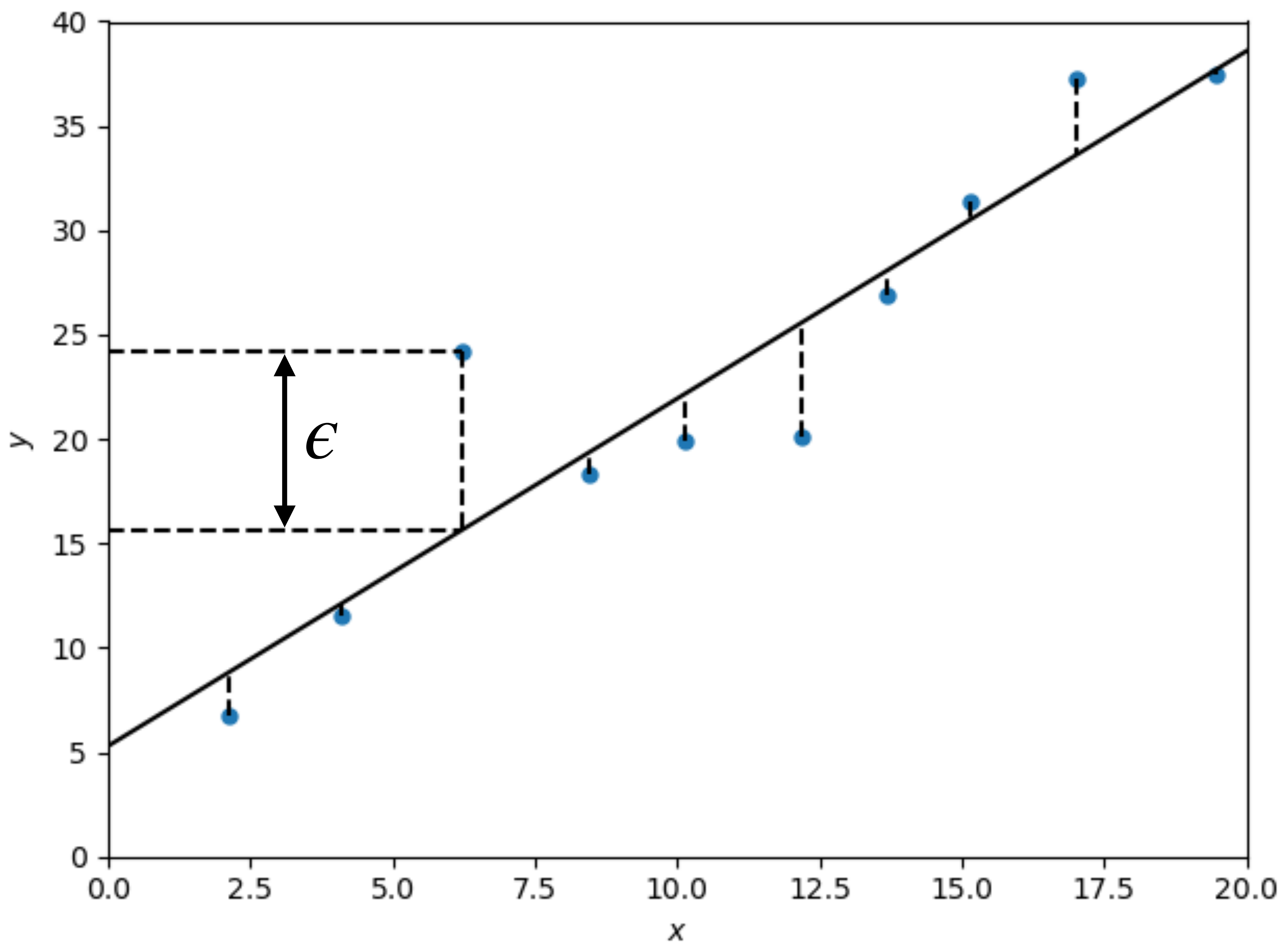
Machine learning



Measuring error

observation y

prediction $\hat{y} = f(x)$



- Error / residual: $\epsilon = y - \hat{y}$
- Mean square error (MSE): $\frac{1}{m} \sum_j (\epsilon^{(j)})^2 = \frac{1}{m} \sum_j (y^{(j)} - \hat{y}^{(j)})^2$

Mean square error

- $\mathcal{L}_\theta = \frac{1}{m} \sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2 = \frac{1}{m} \sum_j (y^{(j)} - \theta^\intercal x^{(j)})^2$

- Why MSE?
  - Mathematically and computationally convenient (we'll see why)
  - Estimates the variance of the residuals
  - Corresponds to log-likelihood under Gaussian noise model

$$\log p(y \mid x) = \log \mathcal{N}(y; \theta^\intercal x, \sigma^2) = -\frac{1}{2\sigma^2}(y - \theta^\intercal x)^2 + \text{const}$$

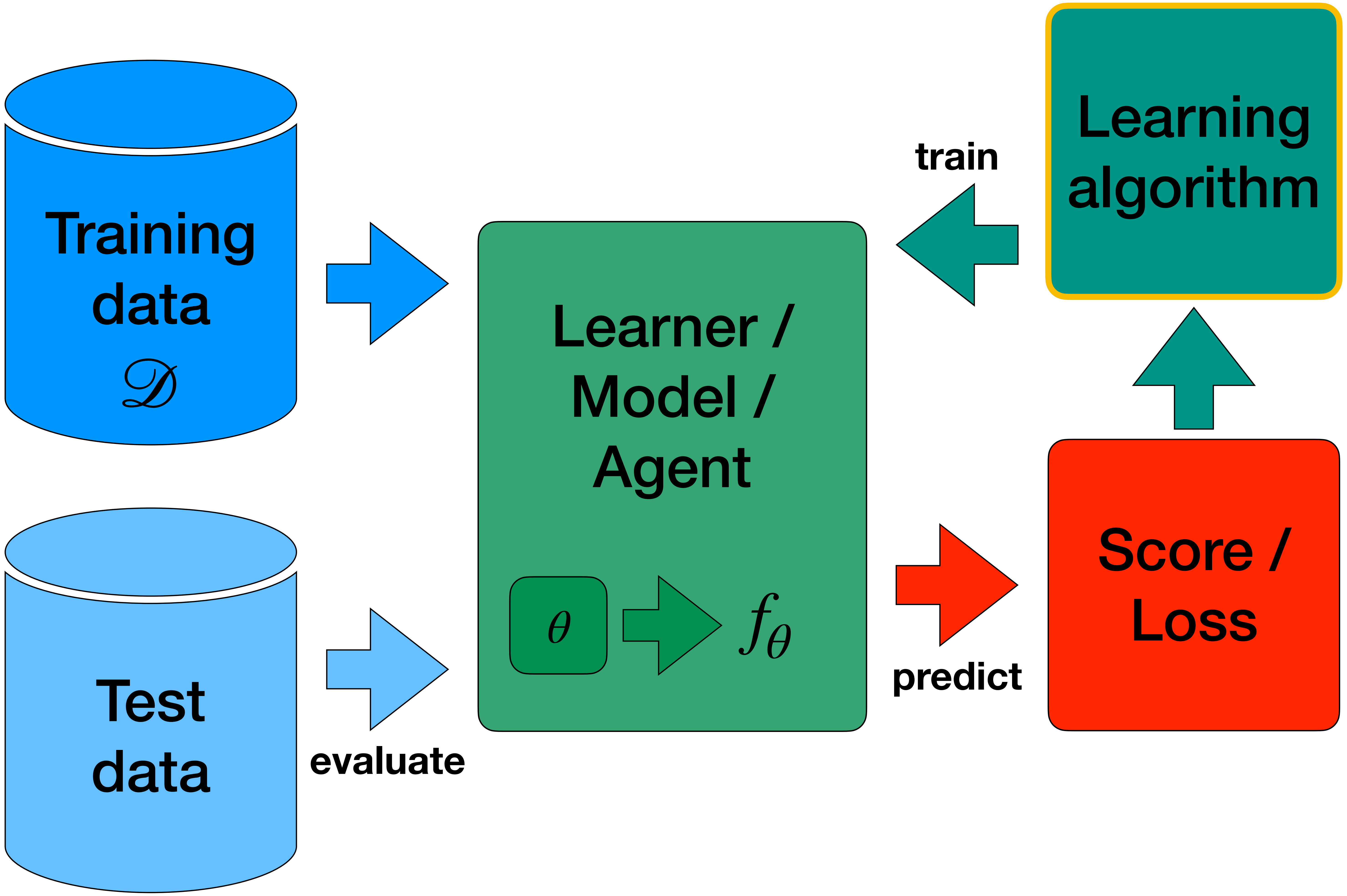
MSE of training data

- Training data matrix: $X = \begin{bmatrix} x_0^{(1)} & \cdots & x_0^{(m)} \\ x_1^{(1)} & \cdots & x_1^{(m)} \\ \vdots & & \vdots \\ x_n^{(1)} & \cdots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{(n+1)\times m}$
- Training labels vector: $y = \begin{bmatrix} y^{(1)} & \cdots & y^{(m)} \end{bmatrix}$
- Prediction: $\hat{y} = \begin{bmatrix} \hat{y}^{(1)} & \cdots & \hat{y}^{(m)} \end{bmatrix} = \theta^\intercal X$

```
# Python / NumPy:
e = y - theta.T @ X
loss = (e @ e.T) / m  # == np.mean( e ** 2 )
```

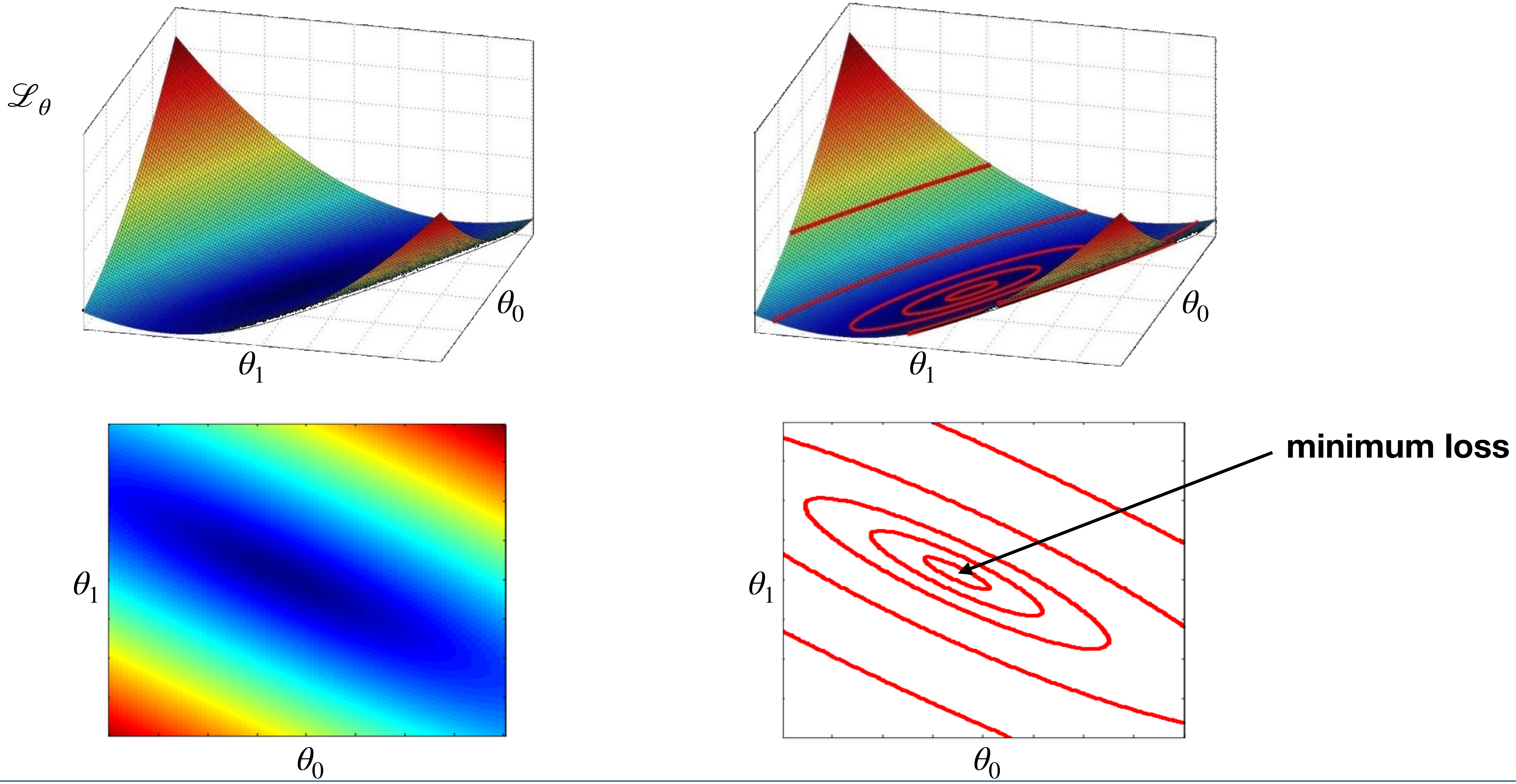
- Training MSE: $\mathcal{L}_\theta(\mathcal{D}) = \frac{1}{m}\sum_j (y^{(j)} - \theta^\intercal x^{(j)})^2 = \frac{1}{m}(y - \theta^\intercal X)(y - \theta^\intercal X)^\intercal$

Machine learning



Loss landscape

- $\mathscr{L}_\theta(\mathscr{D}) = \frac{1}{m}(y - \theta^\intercal X)(y - \theta^\intercal X)^\intercal = \frac{1}{m}(\theta^\intercal X X^\intercal \theta - 2yX^\intercal\theta + yy^\intercal)$ ← **quadratic!**



Today's lecture

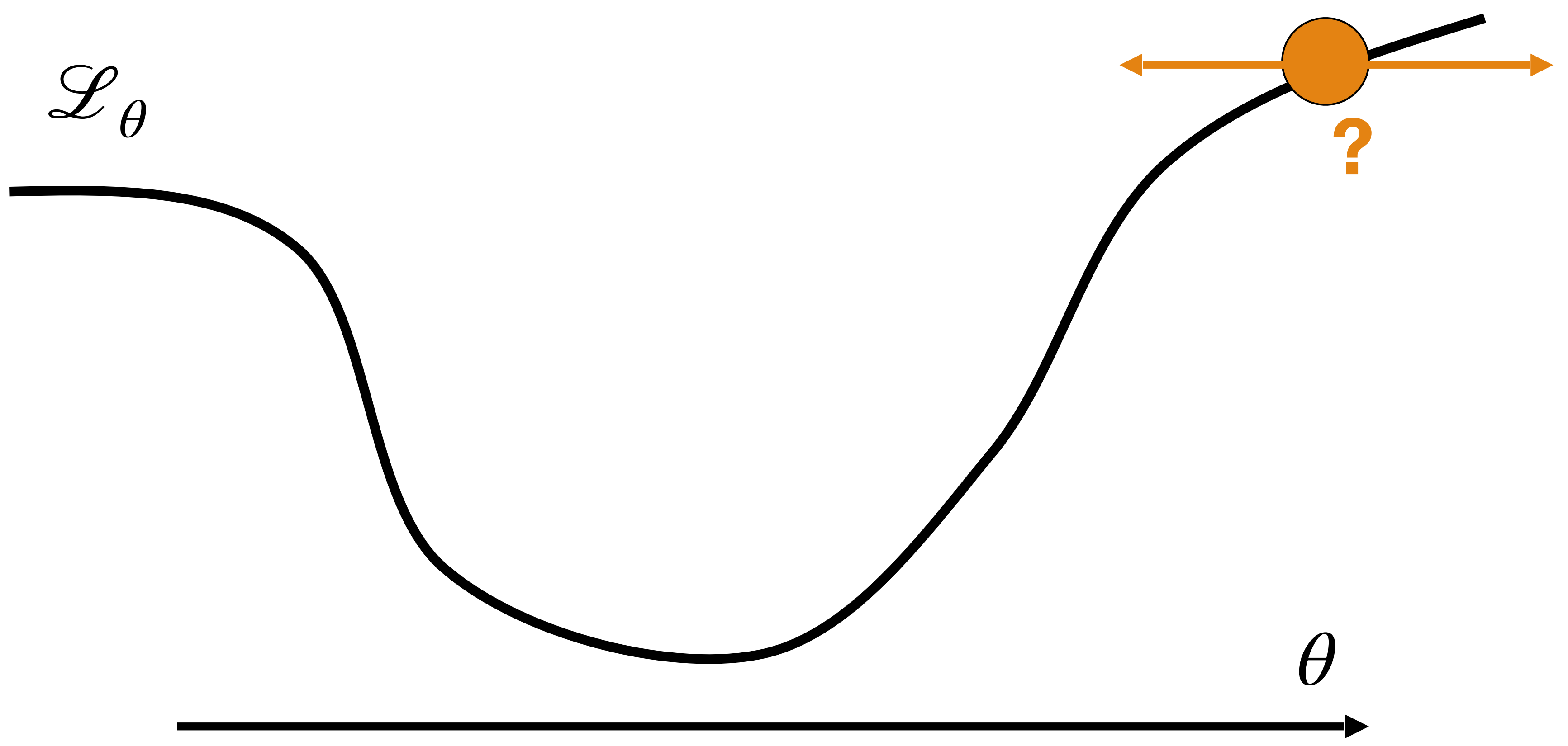
ROC curves

Linear regression

Gradient descent

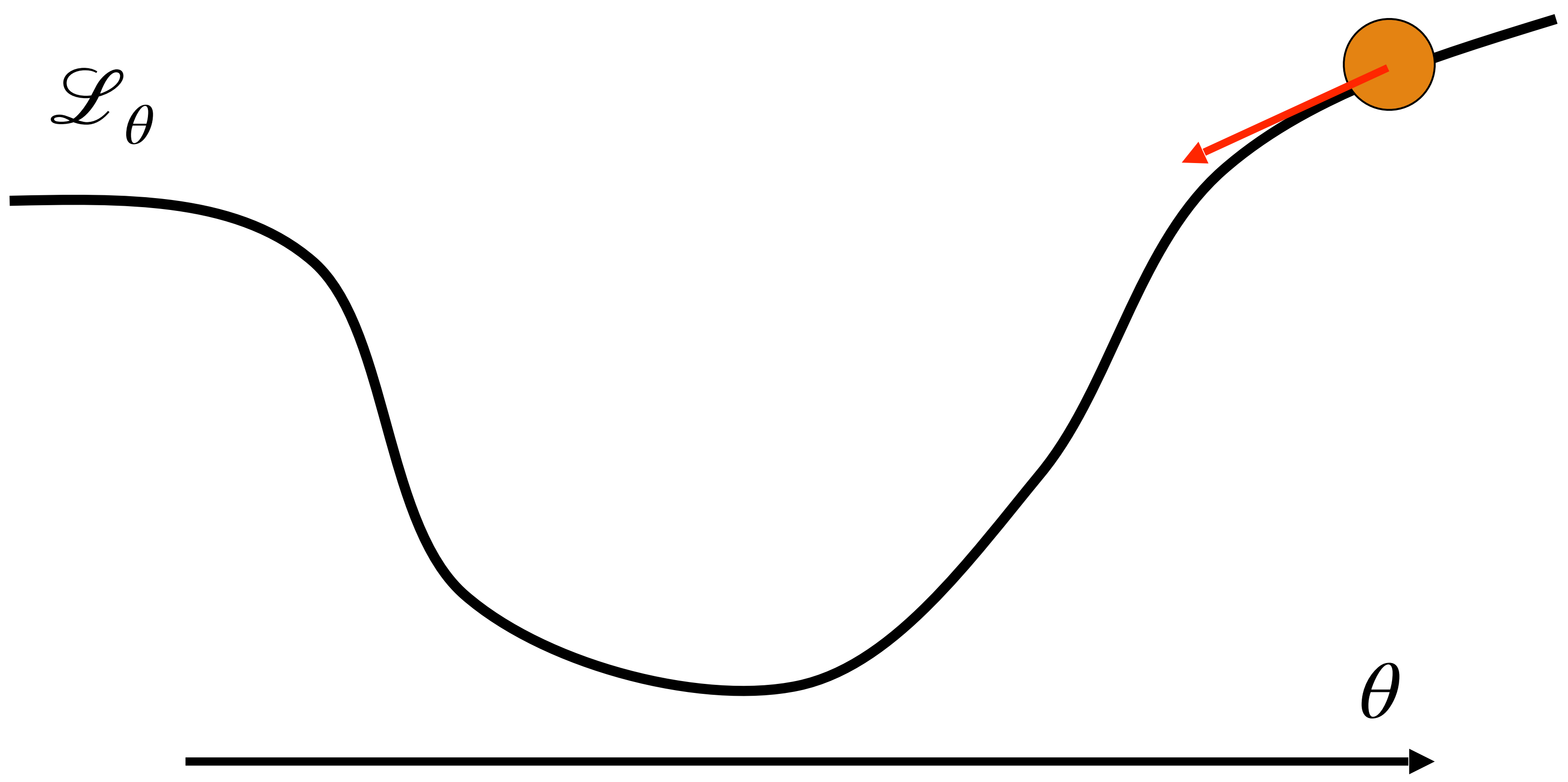
Gradient descent

- How to vary $\theta \in \mathbb{R}^{n+1}$ to improve the loss $\mathscr{L}_\theta$?
  - Find a direction in parameter space in which $\mathscr{L}_\theta$ is decreasing



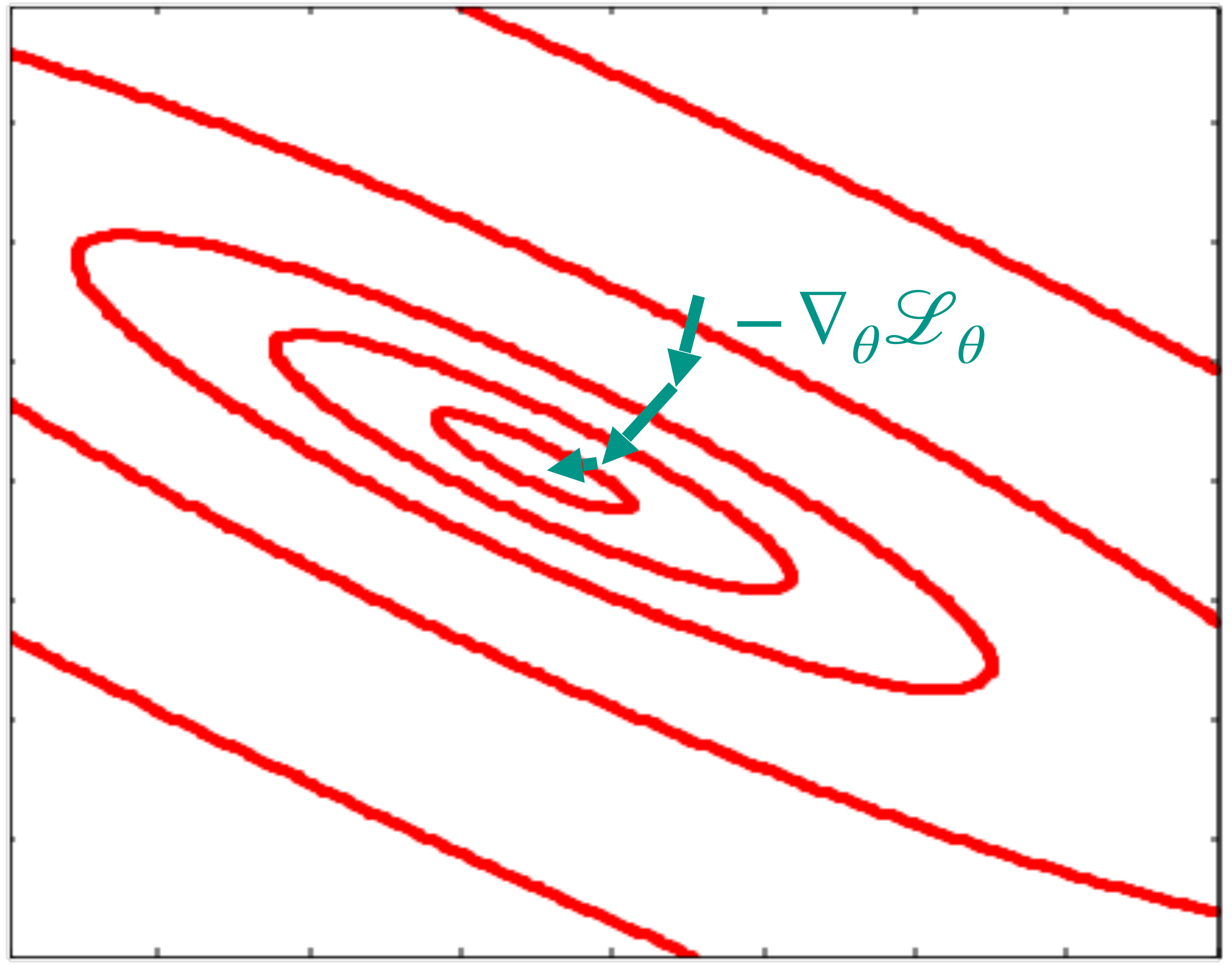
Gradient descent

- How to vary $\theta \in \mathbb{R}^{n+1}$ to improve the loss $\mathscr{L}_\theta$?
  - Find a direction in parameter space in which $\mathscr{L}_\theta$ is decreasing
- Derivative $\partial_\theta \mathscr{L}_\theta = \lim_{\delta\theta \to 0} \frac{\mathscr{L}_{\theta+\delta\theta} - \mathscr{L}_\theta}{\delta\theta}$
  - Positive = loss increases with θ
  - Negative = loss decreases with θ



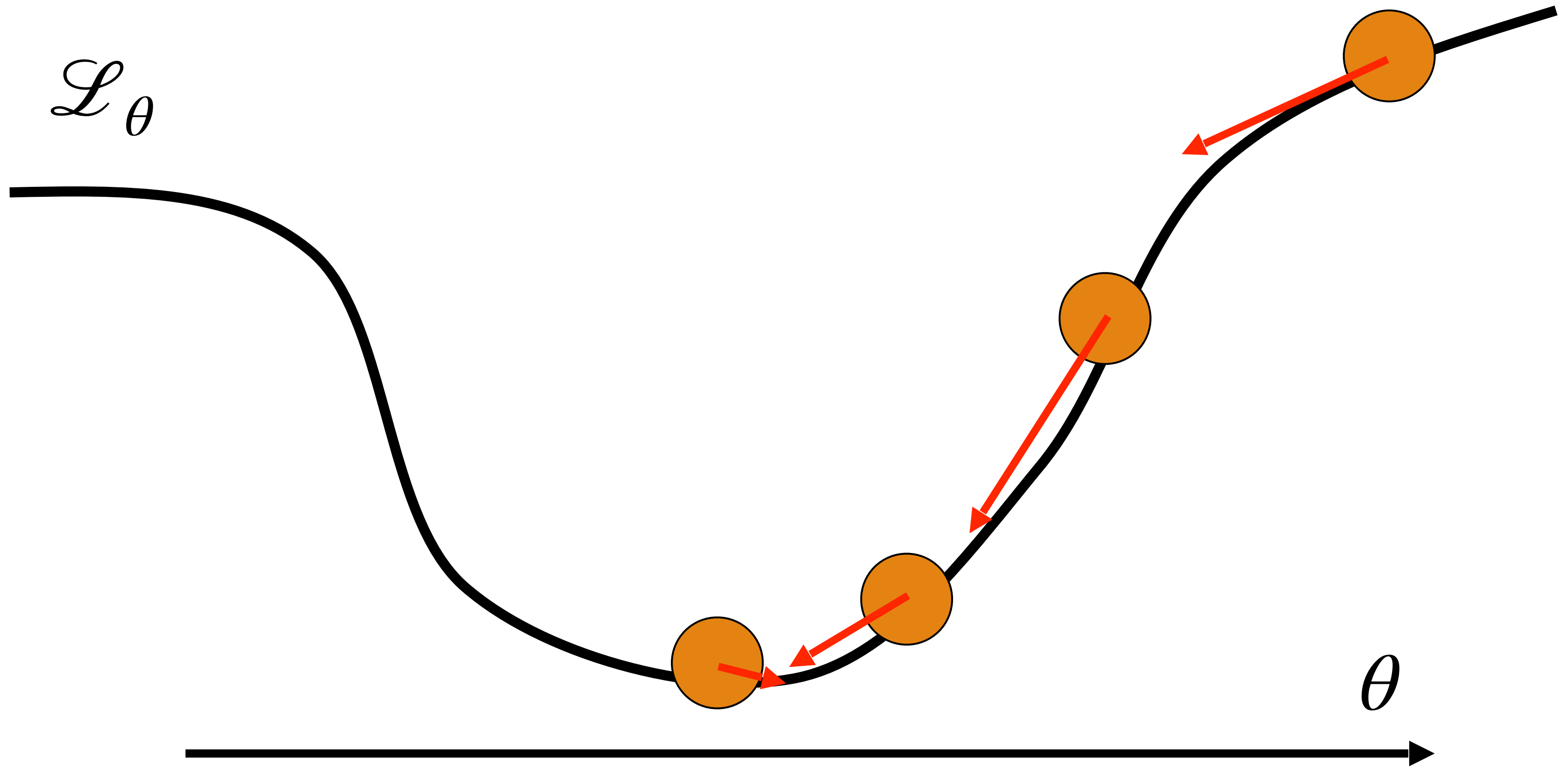
Gradient descent in higher dimension

- Gradient vector: $\nabla_\theta \mathscr{L}_\theta = \begin{bmatrix} \partial_{\theta_0} \mathscr{L}_\theta & \cdots & \partial_{\theta_n} \mathscr{L}_\theta \end{bmatrix}$
- Taylor expansion: $\mathscr{L}(\theta + \delta\theta) = \mathscr{L}(\theta) + (\delta\theta)^\intercal \nabla_\theta \mathscr{L}_\theta + o(\|\delta\theta\|^2)$
  - If we take a small step $\delta\theta$, the best one is in direction $\nabla_\theta \mathscr{L}_\theta$
  - Gradient = direction of steepest ascent (negative = steepest descent)



Gradient Descent

- Initialize θ
- Do
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta$
- While $||\alpha \nabla_\theta \mathscr{L}_\theta|| \leq \epsilon$
- Learning rate: α
  - Can change in each iteration



Gradient for the MSE loss

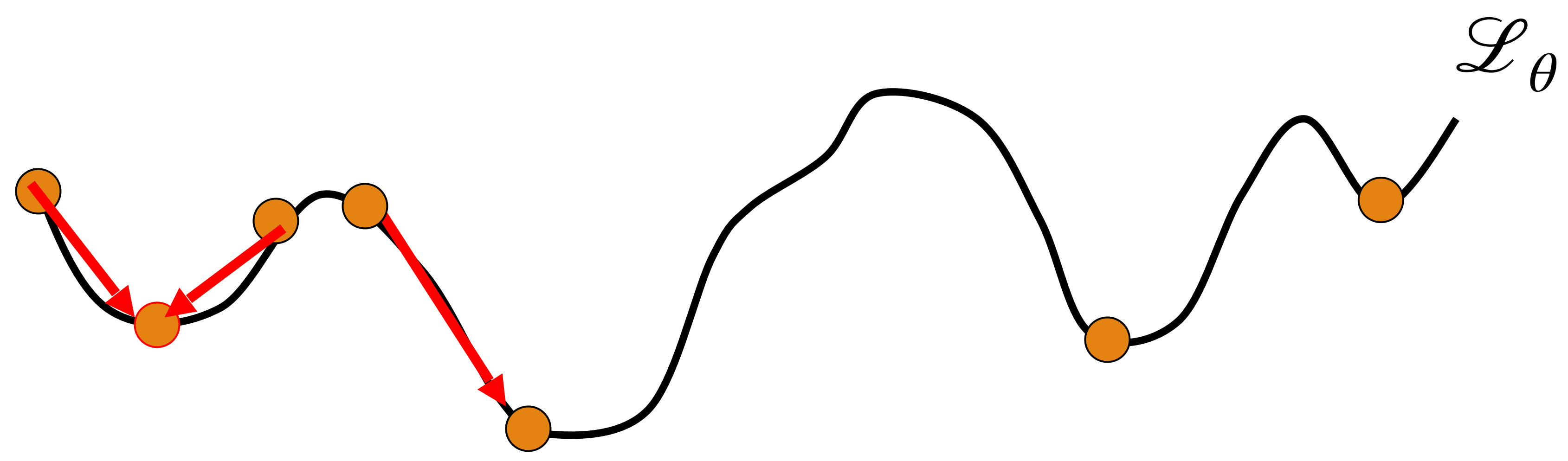
- MSE: $\mathscr{L}_\theta = \frac{1}{m}\sum_j (\epsilon^{(j)})^2 = \frac{1}{m}\sum_j (y^{(j)} - \theta^\intercal x^{(j)})^2$
- $\partial_{\theta_i}\mathscr{L}_\theta = \frac{1}{m}\sum_j \partial_{\theta_i}(\epsilon^{(j)})^2 = \frac{1}{m}\sum_j 2\epsilon^{(j)}\partial_{\theta_i}\epsilon^{(j)}$
  - $\partial_{\theta_i}(y^{(j)} - \theta^\intercal x^{(j)}) = -\partial_{\theta_i}\theta_i x_i^{(j)} + 0$ in the other terms $= x_i^{(j)}$
  - $\partial_{\theta_i}\mathscr{L}_\theta = -\frac{2}{m}\sum_j \epsilon^{(j)}x_i^{(j)} = -\frac{2}{m}(y - \theta^\intercal X)X_i^\intercal$
- $\nabla_\theta\mathscr{L}_\theta = -\frac{2}{m}(y - \theta^\intercal X)X^\intercal$

- Can also be seen directly from

$$\mathscr{L}_\theta = \frac{1}{m}(y - \theta^\intercal X)(y - \theta^\intercal X)^\intercal = \frac{1}{m}(\theta^\intercal X X^\intercal\theta - 2yX^\intercal\theta + yy^\intercal)$$

Gradient Descent — further considerations

- GD is a very general algorithm
  - We'll use it often
  - Much of the engine for recent advances in ML
- Issues:



  - Can get stuck in local minima
    - Worse — can get stuck in saddle points, $\nabla_\theta \mathscr{L}_\theta = 0$ with improvement direction
  - Can be slow to converge, sensitive to initialization
  - How to choose step size / learning rate?
    - Constant? 1/iteration? Line search? Newton's method?

Logistics

staff

- Emad Naeini is joining the course staff
- Emad's office hours:
  - https://calendly.com/ekasaeya/cs-273a-emad-s-office-hour

assignments

- Assignment 1 due today
- Assignment 2 to be published next week