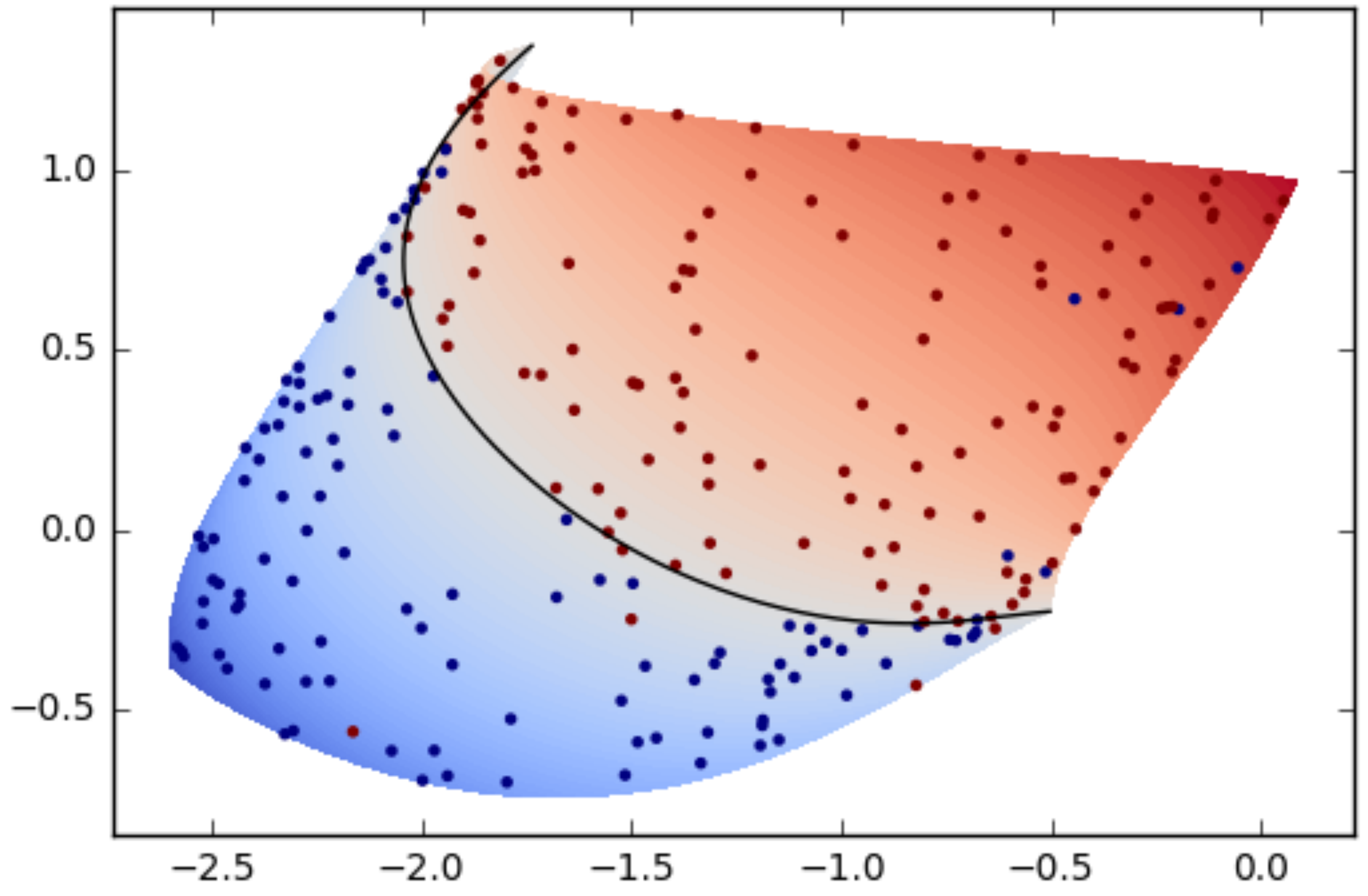
CS 273A: Machine Learning

Fall 2021

Lecture 5: Linear Regression (cont.)

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 1 due today
- Assignment 2 to be published soon

Today's lecture

ROC curves

Linear regression

Least squares

Gradient descent

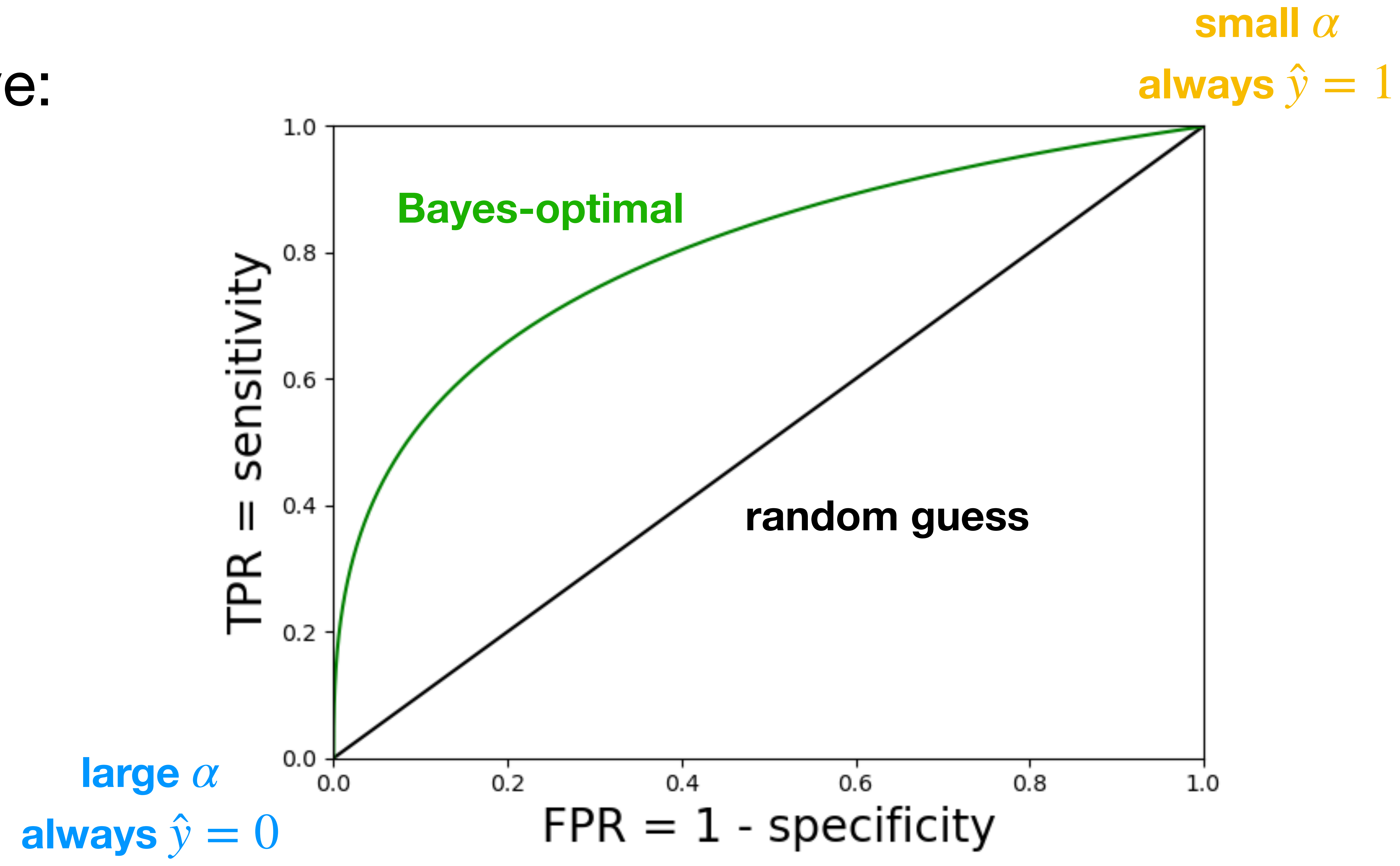
Bayes-optimal decision

- Maximum posterior decision: $\hat{p}(y=0 \mid x) \lessgtr \hat{p}(y=1 \mid x)$
  - 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? $\alpha_{\text{FP}}, \alpha_{\text{FN}}$
  - $\mathscr{L} = \mathbb{E}_{x,y\sim p}[\alpha_{\text{FP}} \cdot \#(y=0, \hat{y}(x)=1) + \alpha_{\text{FN}} \cdot \#(y=1, \hat{y}(x)=0)]$
- Bayes-optimal decision: $\alpha_{\text{FP}} \cdot \hat{p}(y=0 \mid x) \lessgtr \alpha_{\text{FN}} \cdot \hat{p}(y=1 \mid x)$
  - Log probability ratio: $\log \dfrac{\hat{p}(y=1 \mid x)}{\hat{p}(y=0 \mid x)} \lessgtr \log \dfrac{\alpha_{\text{FP}}}{\alpha_{\text{FN}}} = \alpha$

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:

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

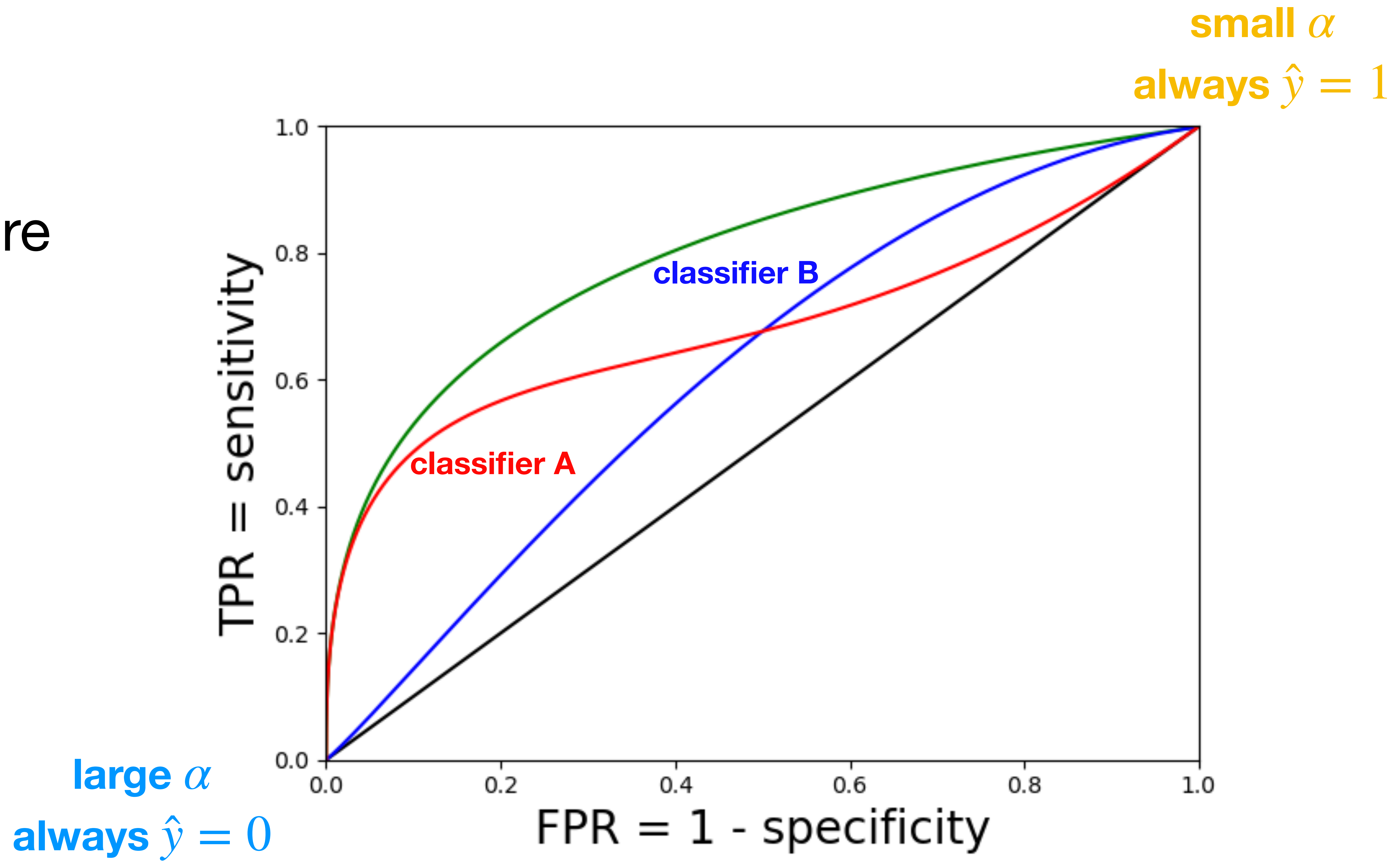


Demonstration

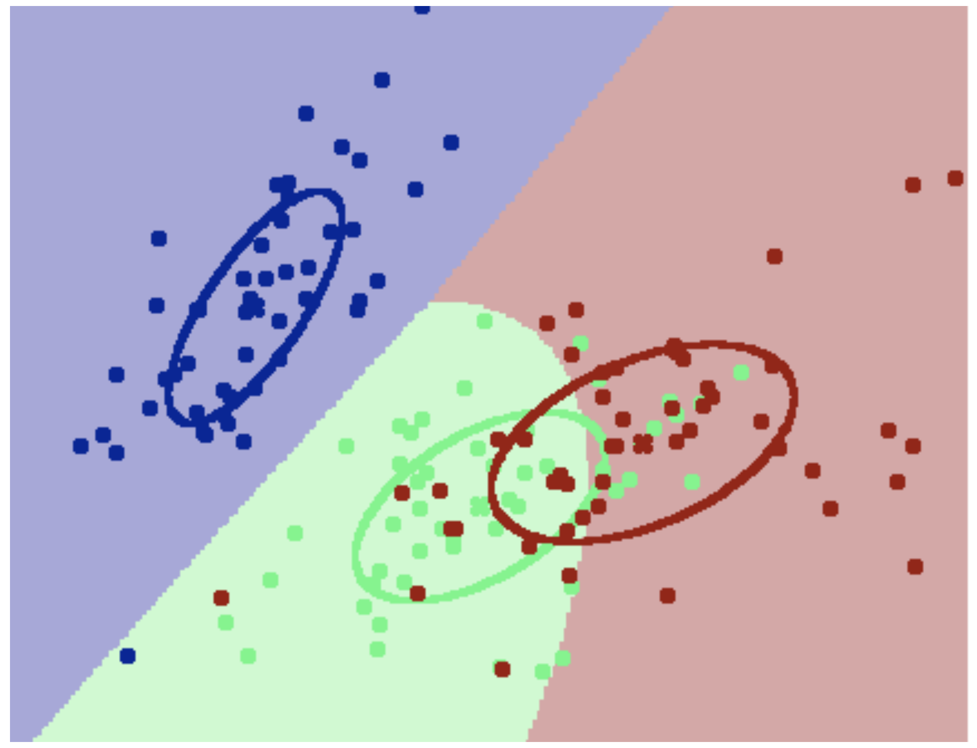
- http://www.navan.name/roc

Comparing classifiers

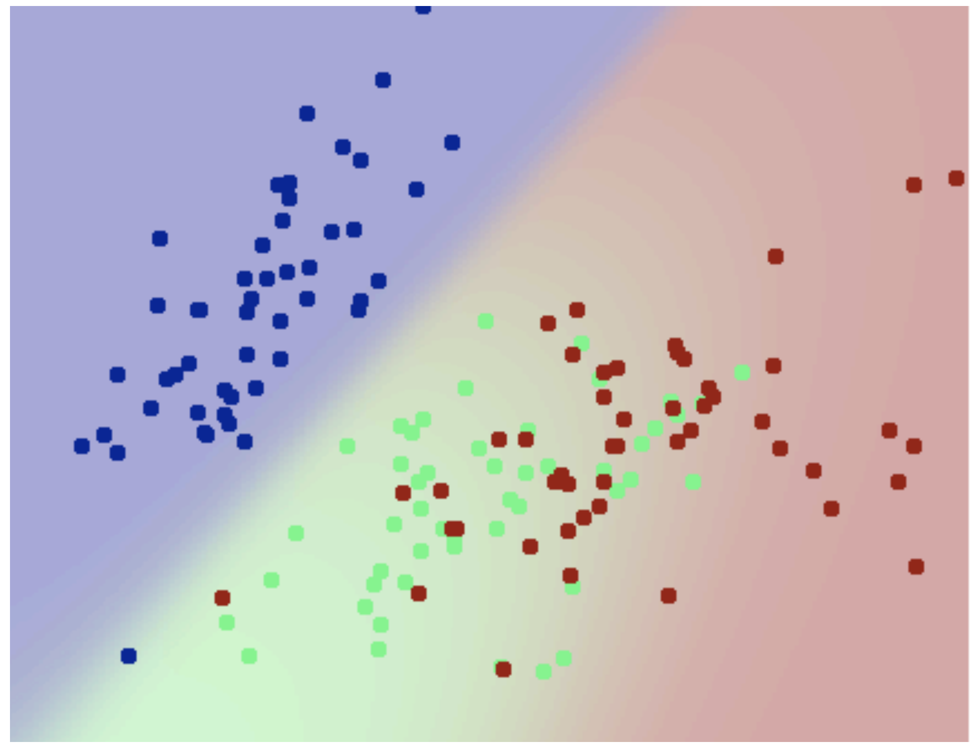
- Which classifier (A or B) performs "better"?
  - A is better for high specificity
  - B is better for high sensitivity
  - Need single performance measure
- Area Under Curve (AUC)
  - 0.5 ≤ AUC ≤ 1
  - AUC = 0.5: random guess
  - AUC = 1: no errors



Discriminative vs. probabilistic predictions



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



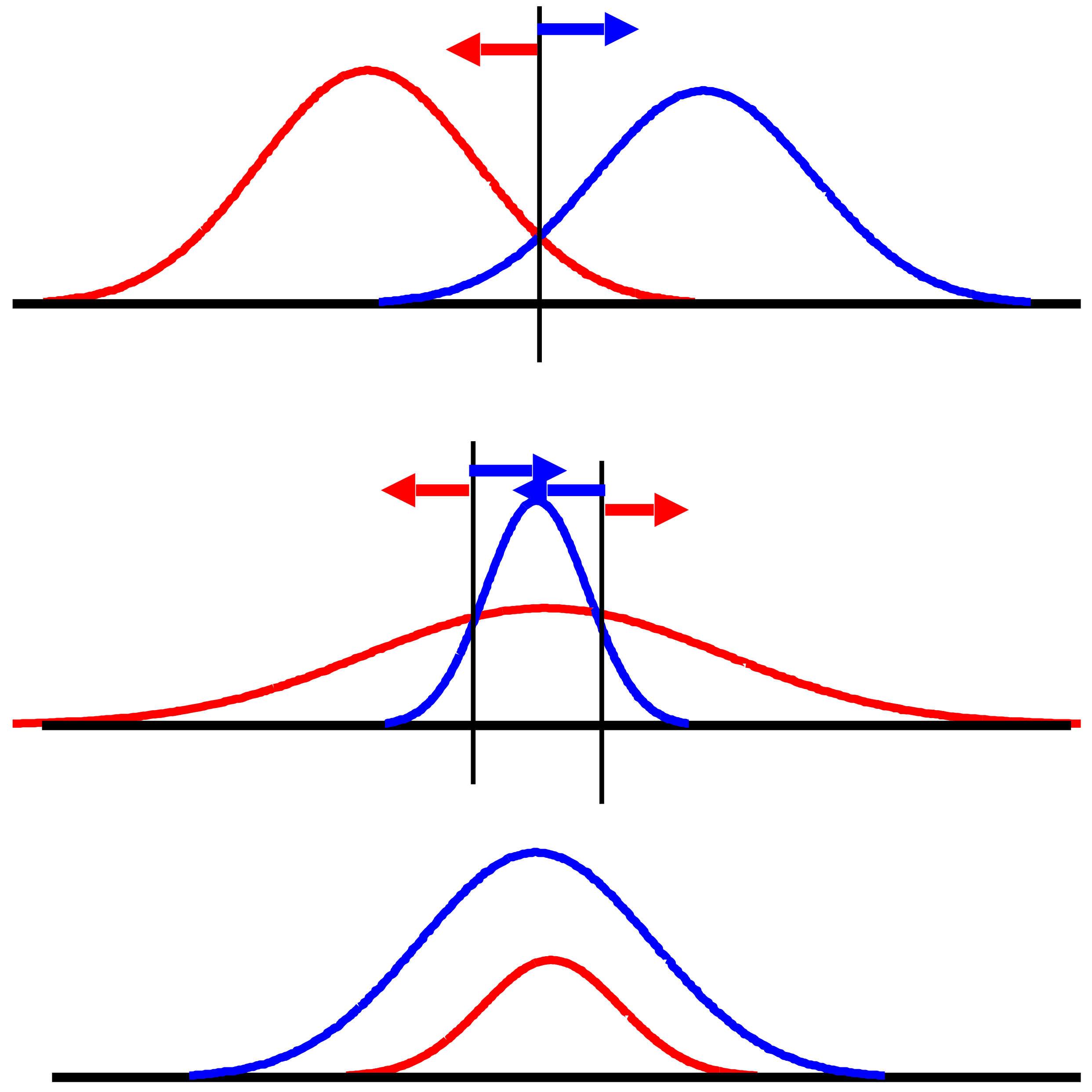
probabilistic predictions $p(y \mid x)$

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

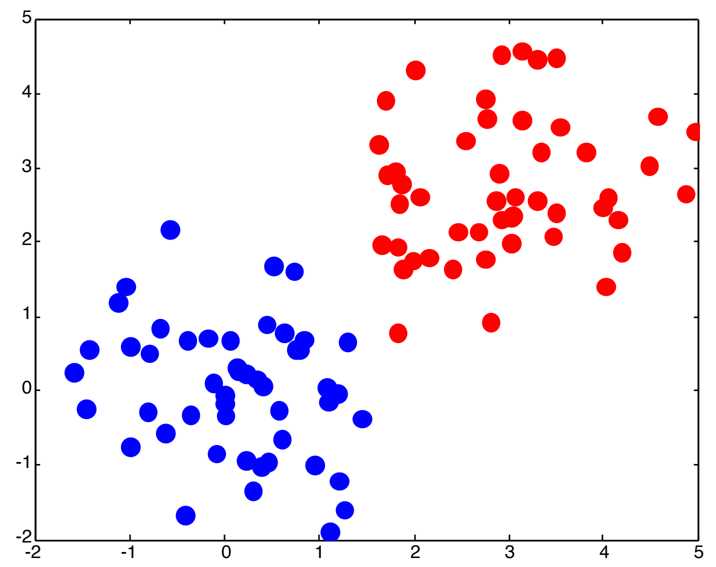
- Probabilistic learning gives more nuanced prediction
  - Can use $p(y \mid x)$ to find $\hat{y}(x) = \arg\max_y p(y \mid x)$ (if argmax is feasible)
  - Express confidence in predicting $\hat{y}$
  - Conditional models: $p(y \mid x)$; vs. generative models: $p(x, y)$
    - Can be used to generate x
    - 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
  - $p(x \,|\, y=c) = \mathcal{N}(x; \mu_c, \Sigma_c) = (2\pi)^{-\frac{d}{2}} |\Sigma_c|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu_c)^\intercal \Sigma_c^{-1} (x-\mu_c)\right)$
  - Assume same covariance $\Sigma_0 = \Sigma_1$
- What is the shape of the decision boundary $p(y=0 \,|\, x) = p(y=1 \,|\, x)$?

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

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

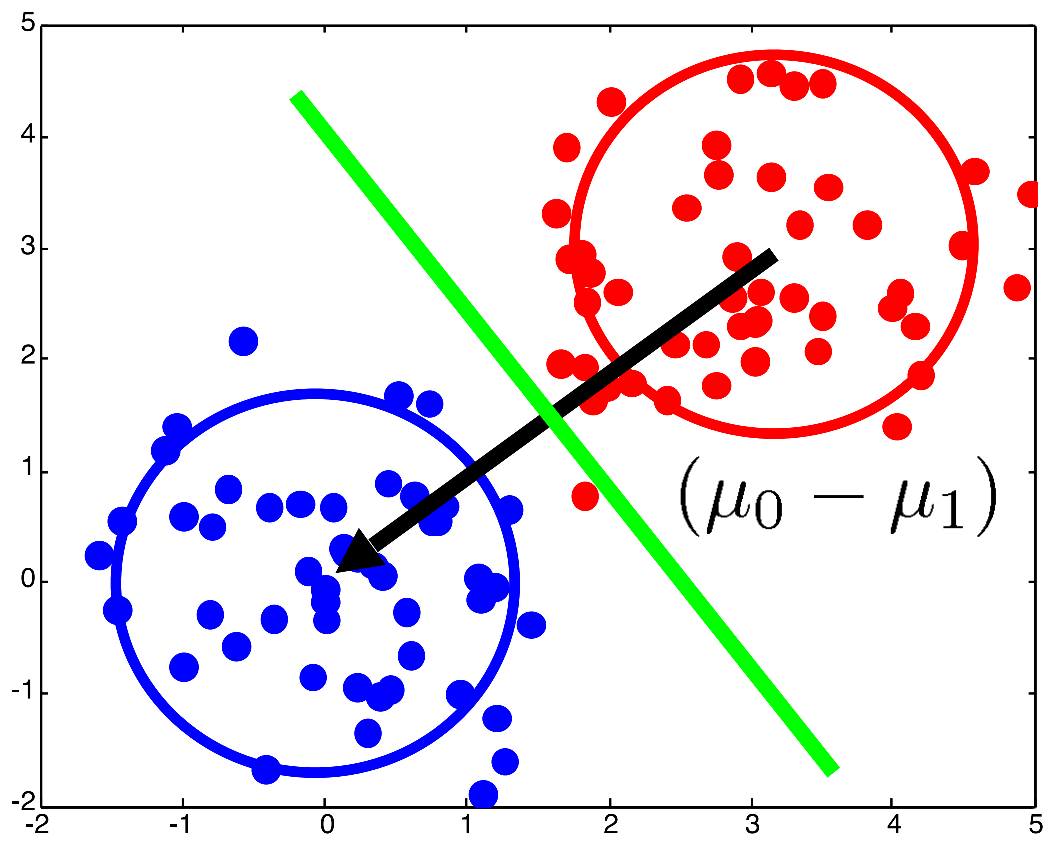
$$-\frac{1}{2}\left(x^\intercal \Sigma^{-1} x - 2\mu_1^\intercal \Sigma^{-1} x + \mu_1^\intercal \Sigma^{-1} \mu_1\right)$$

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

← **linear!**

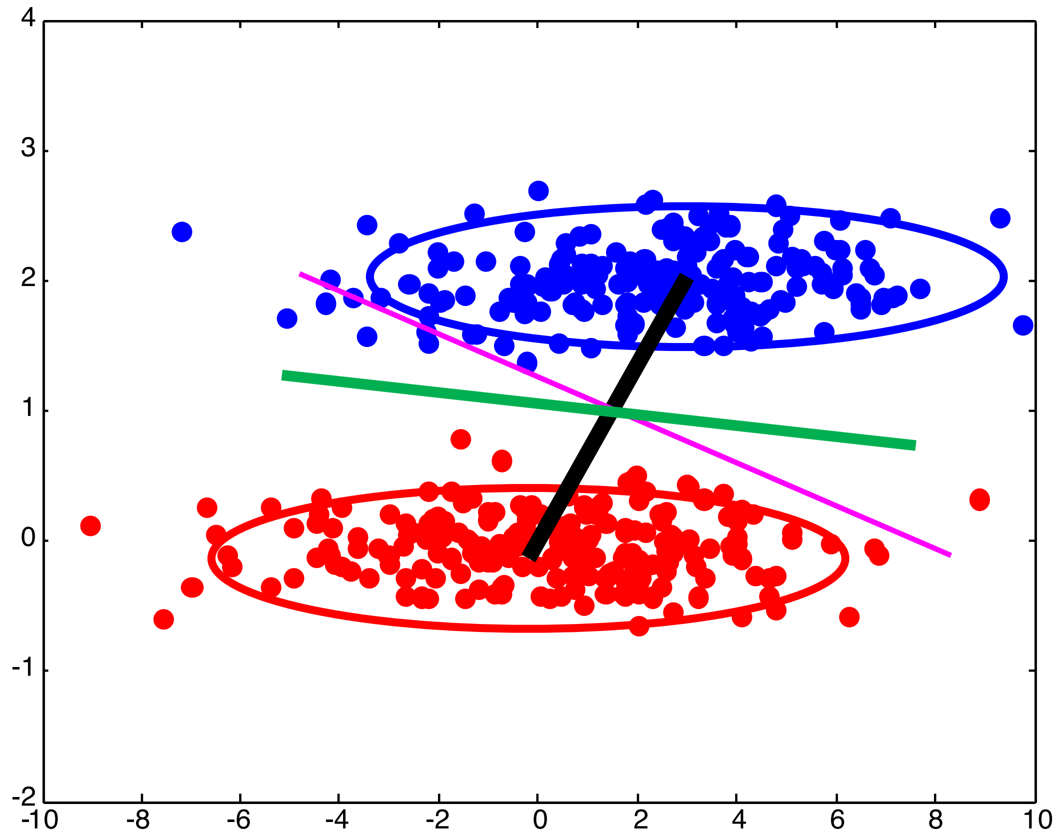
Gaussian models

- 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

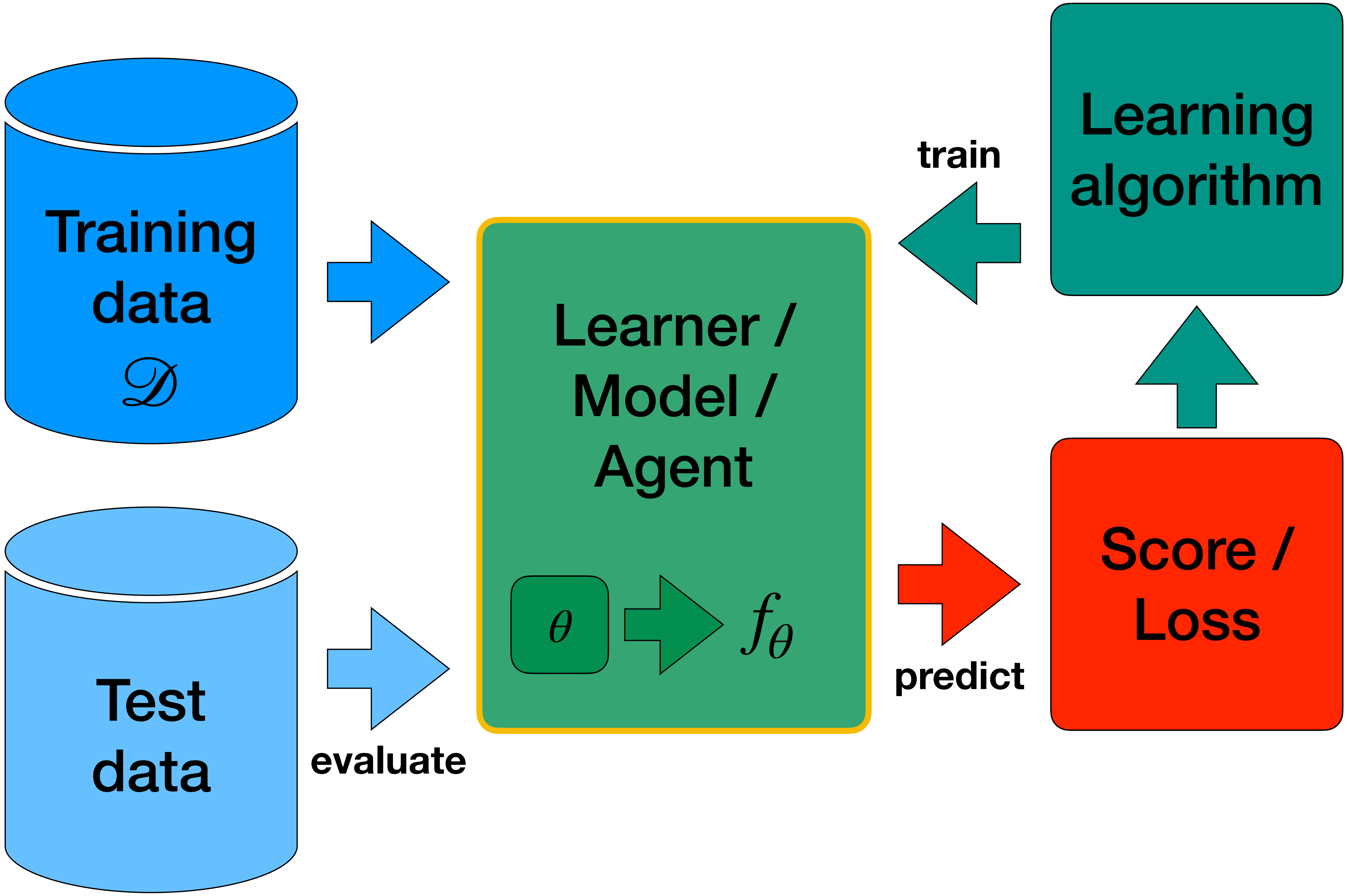
ROC curves

Linear regression

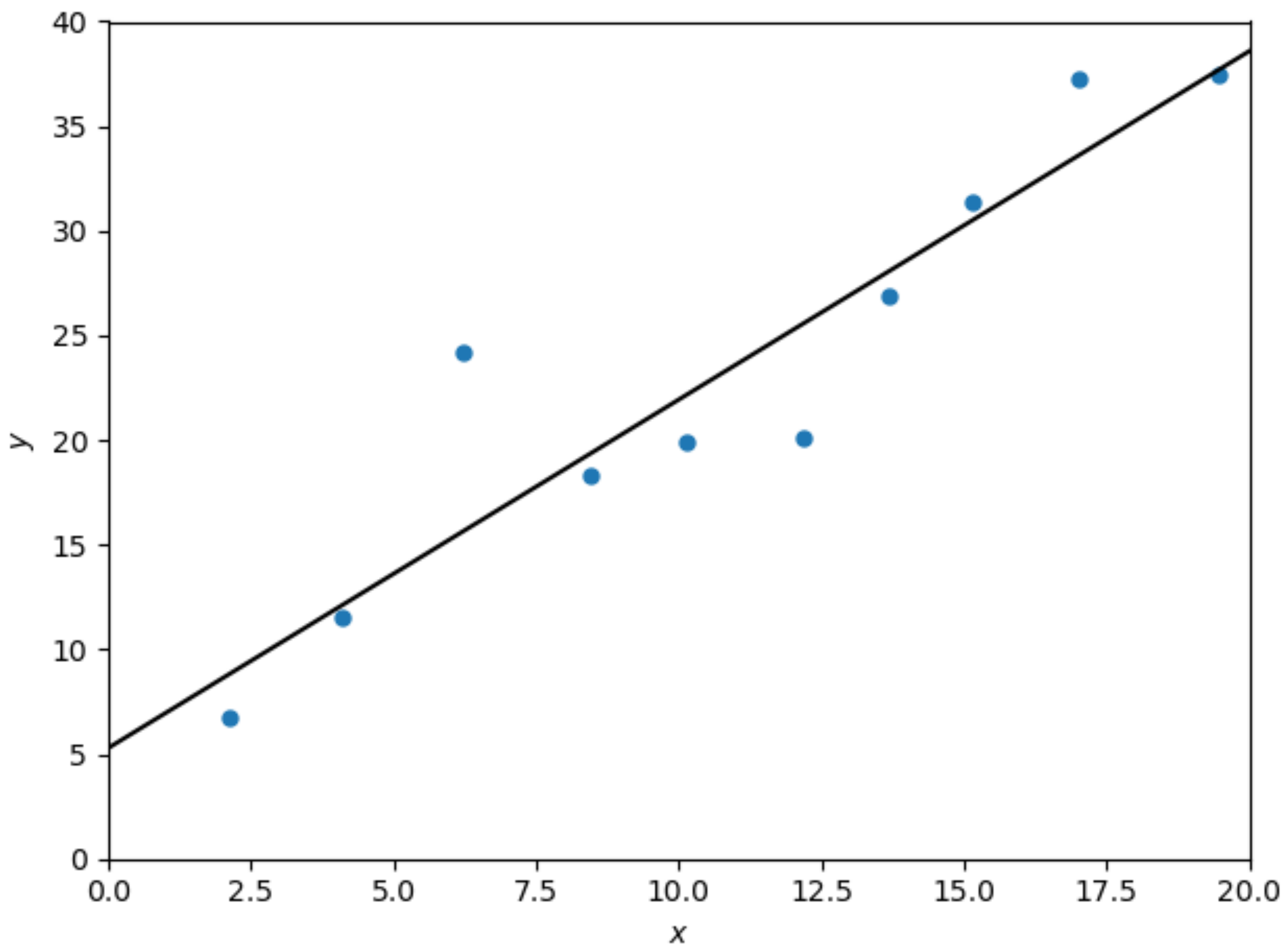
Least squares

Gradient descent

Machine learning



Linear regression

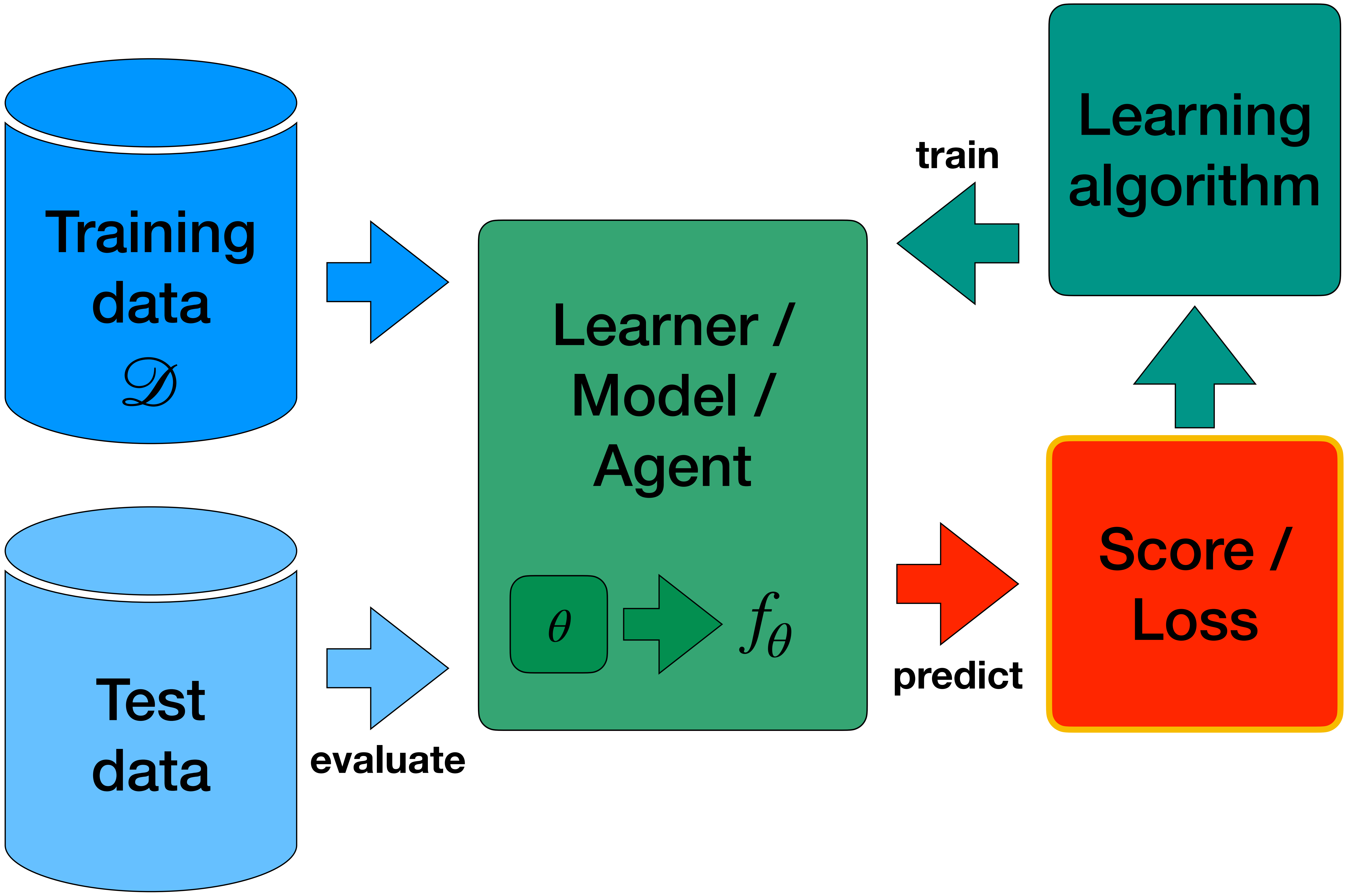


- Decision function $f : x \mapsto y$ is linear, $f(x) = \theta_0 + \theta_1 x$
- f is stored by its parameters $\theta = \begin{bmatrix} \theta_0 & \theta_1 \end{bmatrix}$

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 $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

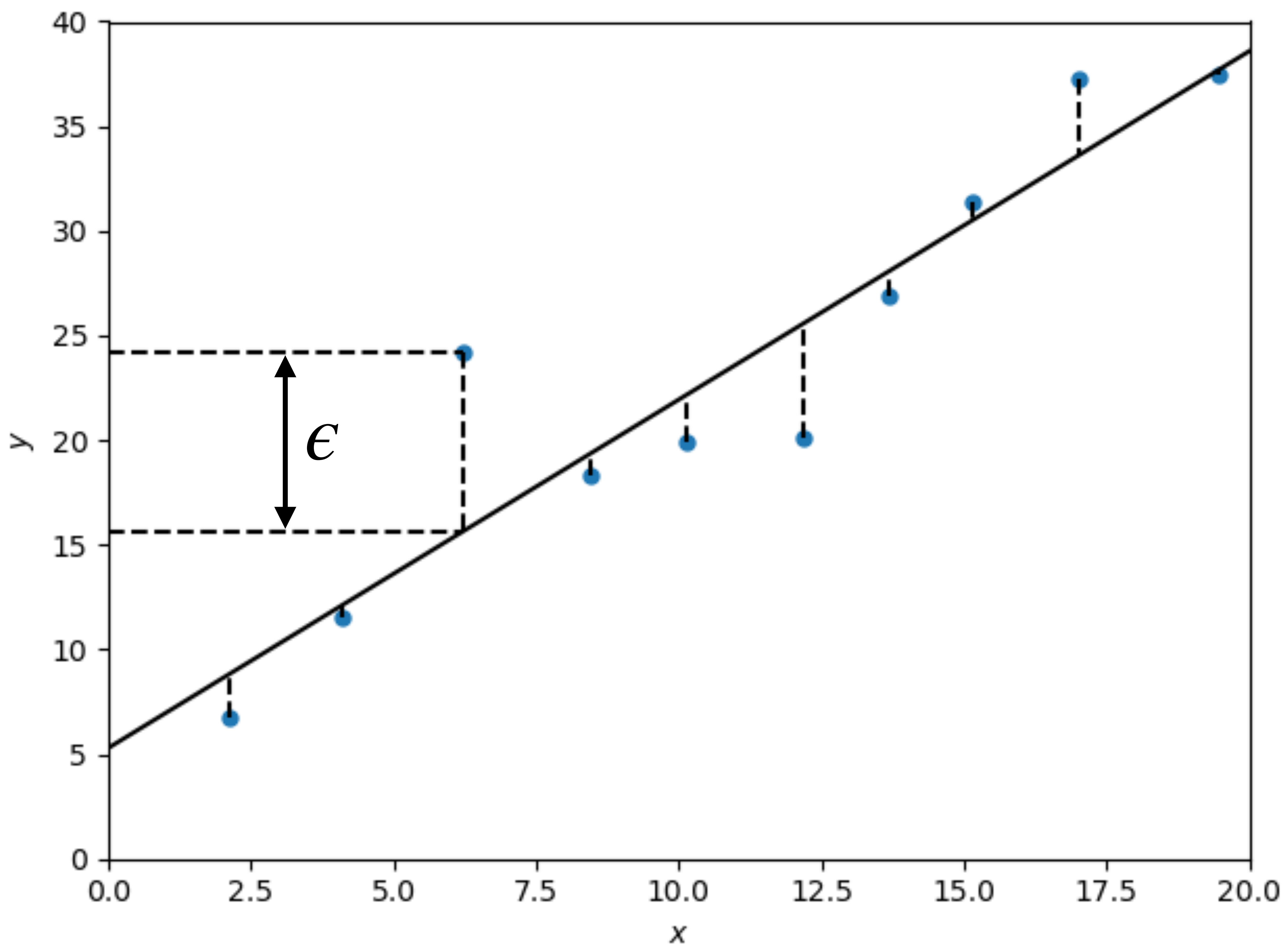
Machine learning



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

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

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

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

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

Today's lecture

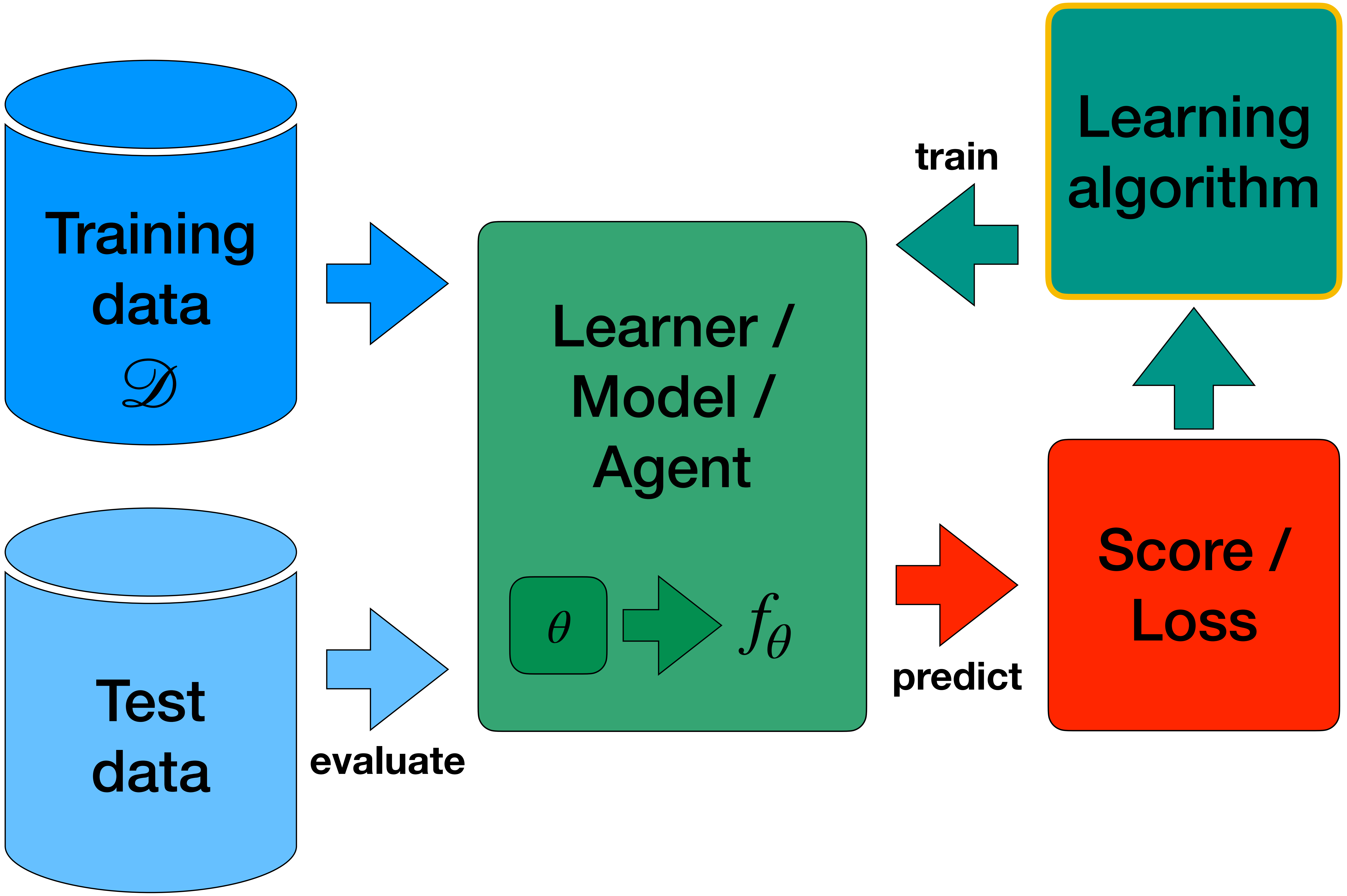
ROC curves

Linear regression

Least squares

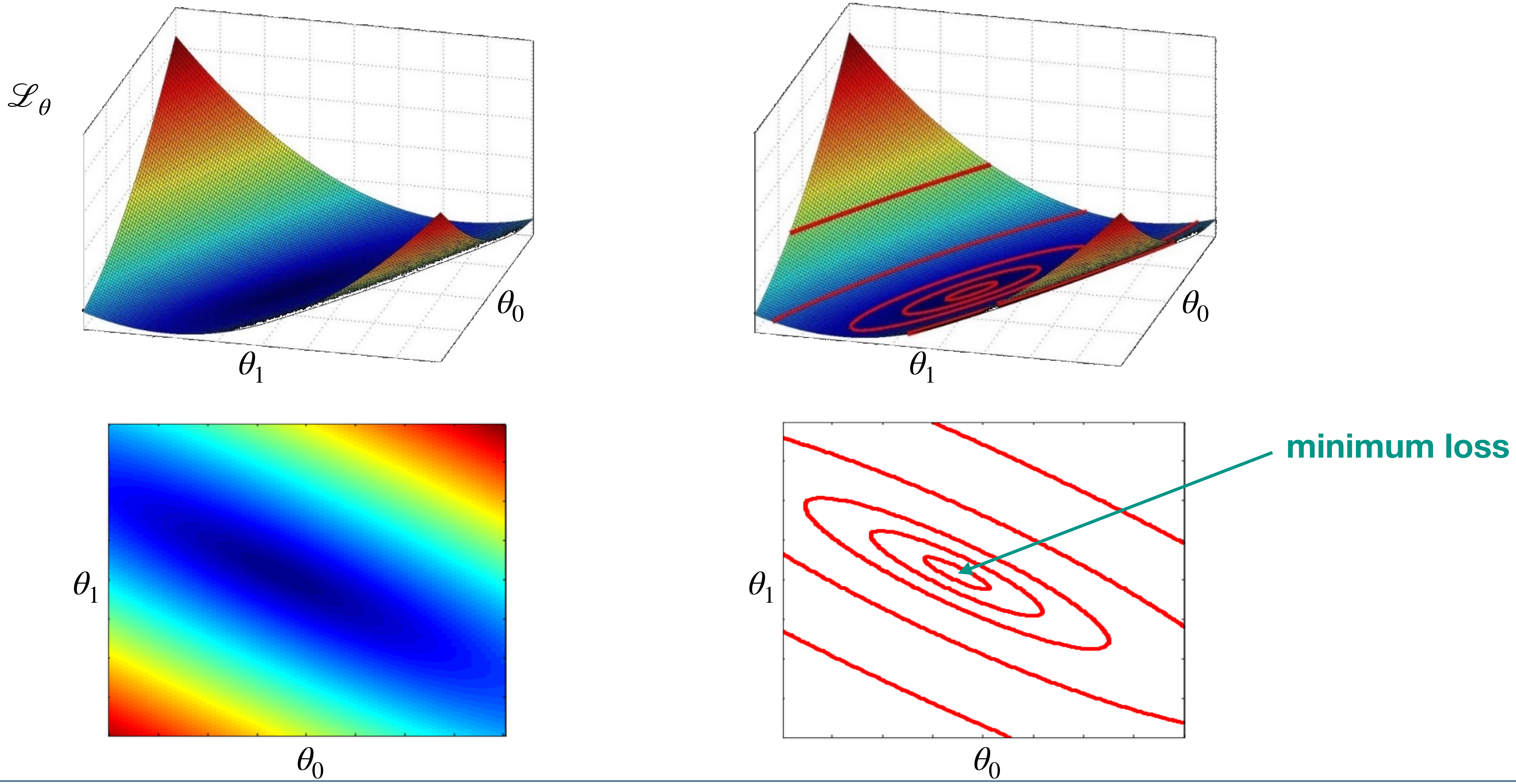
Gradient descent

Machine learning



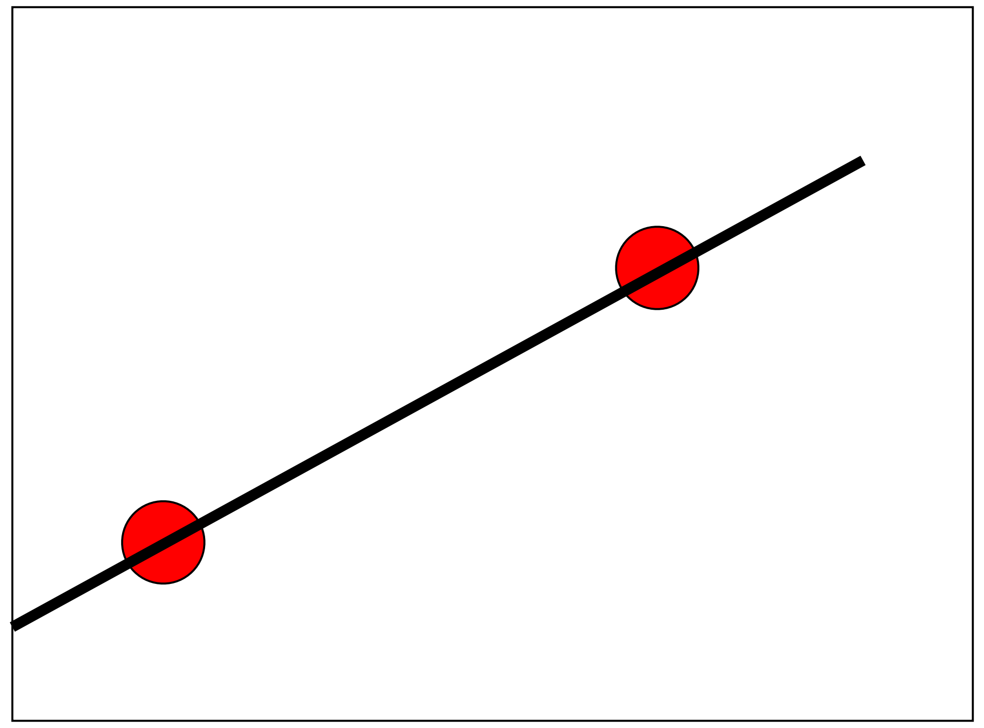
Loss landscape

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



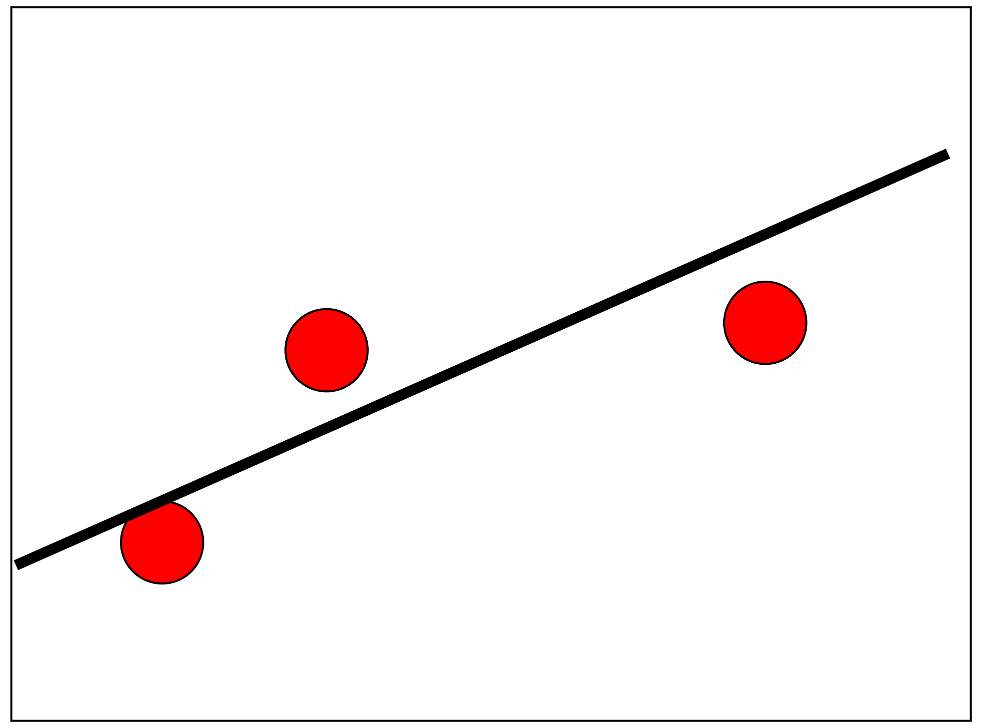
Minimizing MSE

- Consider a simple problem
  - One feature, two data points $x^{(1)}, x^{(2)}$
  - Two unknowns θ_0, θ_1
  - Two equations: $\theta_0 + \theta_1 x^{(1)} = y^{(1)}$ $\qquad \theta_0 + \theta_1 x^{(2)} = y^{(2)}$



- Can solve this system directly: $y = \theta^\intercal X \implies \theta^\intercal = yX^{-1}$
- Generally, X may not have an inverse; e.g., $m > n + 1$
- There may also be training loss, no θ achieves equality of y to $\theta^\intercal X$

Least Squares



- The minimum is achieved when the gradient is 0

$$\nabla_\theta \mathscr{L}_\theta = -\frac{2}{m}(y - \theta^\intercal X)X^\intercal = 0$$

$$\theta^\intercal X X^\intercal = y X^\intercal$$

$$\theta^\intercal = y X^\intercal (X X^\intercal)^{-1}$$

- $XX^\intercal$ is invertible when X has linearly independent rows = features
- $X^\dagger = X^\intercal (XX^\intercal)^{-1}$ is the Moore-Penrose pseudo-inverse of X
  - $X^\dagger = X^{-1}$ when the inverse exists
  - Can define $X^\dagger$ via Singular Value Decomposition (SVD) when $XX^\intercal$ isn't invertible
- $\theta^\intercal = yX^\dagger$ is the Least Squares fit of the data (X, y)

Linear regression in NumPy

- Linear regression with MSE: $\min_\theta \frac{1}{m}\|y - \theta^\intercal X\|^2$

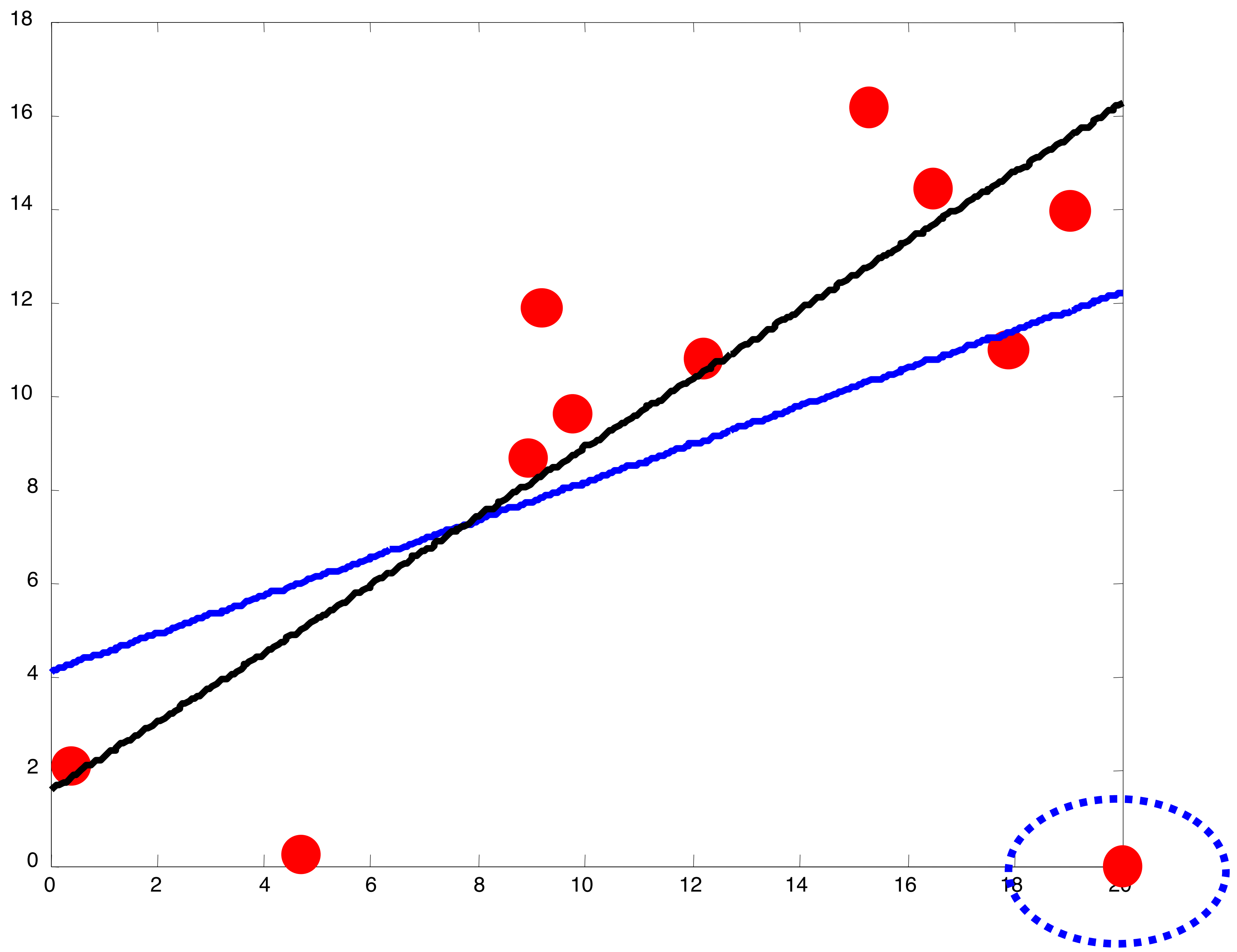
$$\theta^\intercal = yX(XX^\intercal)^{-1} = yX^\dagger$$

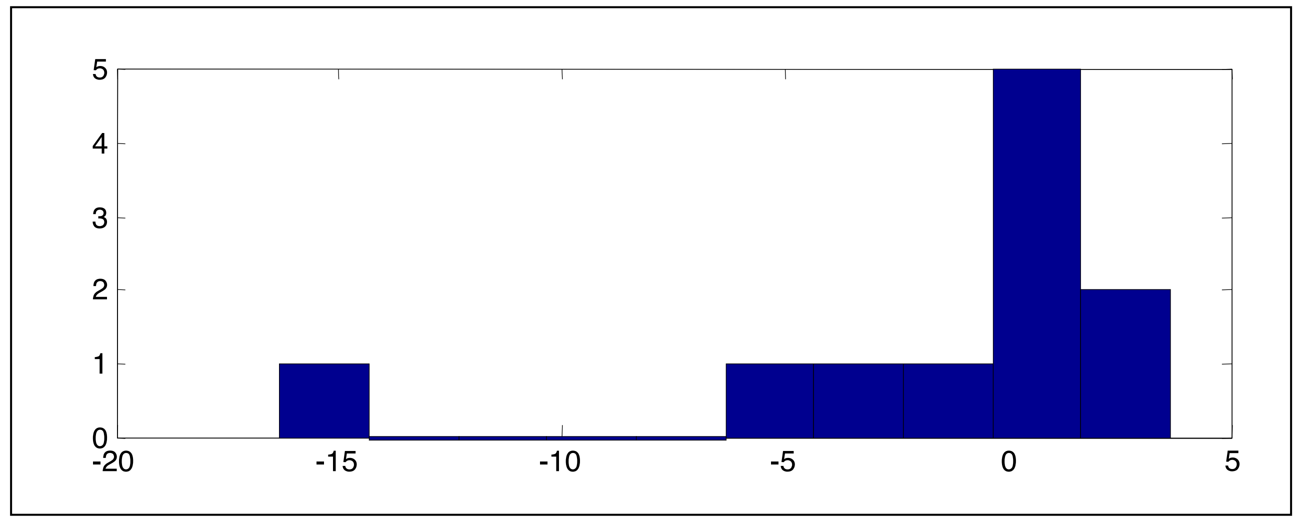
```
# Solution 1: the long way
theta = (y @ X @ np.linalg.inv(X @ X.T)).T

# Solution 2: pseudo-inverse
theta = (y @ np.linalg.pinv(X)).T

# Solution 3: Least Squares solver
theta = np.linalg.lstsq(a=X.T, b=y.T)
```

- Least Squares: approximate $Az = b$ by $\min_z \|Az - b\|^2$

MSE and outliers

- MSE is sensitive to outliers





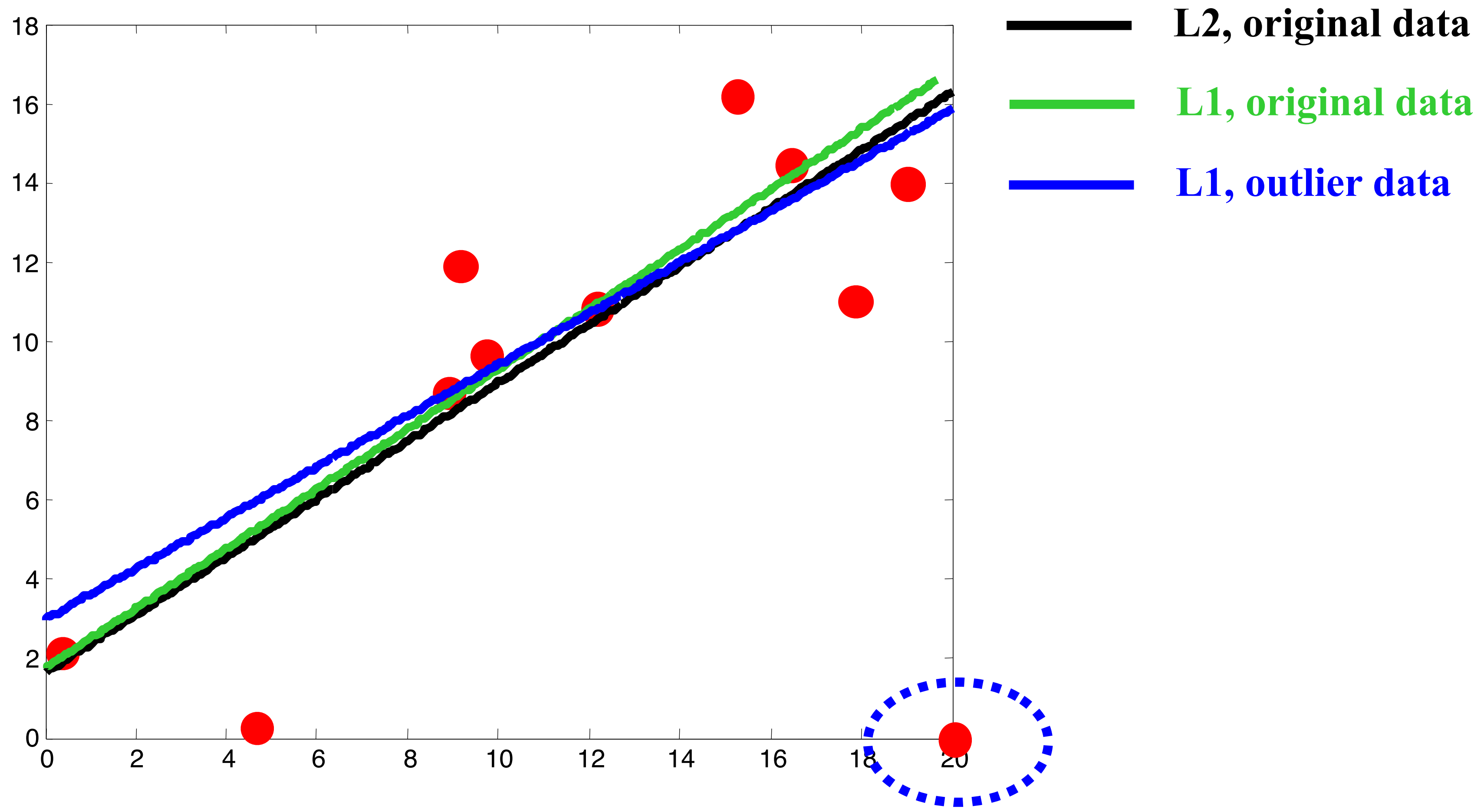
- Square error $\approx 16^2$ throws off entire optimization

Mean Absolute Error (MAE)

- MSE uses the L_2 norm of the error $\|y - \theta^\intercal X\|_2^2 = \sum_j (y - \theta^\intercal X)^2$

- What if we use the L_1 norm $\|y - \theta^\intercal X\|_1 = \sum_j |y - \theta^\intercal X|$?

  - Mean Absolute Error (MAE): $\frac{1}{m} \sum_j |y - \theta^\intercal X|$



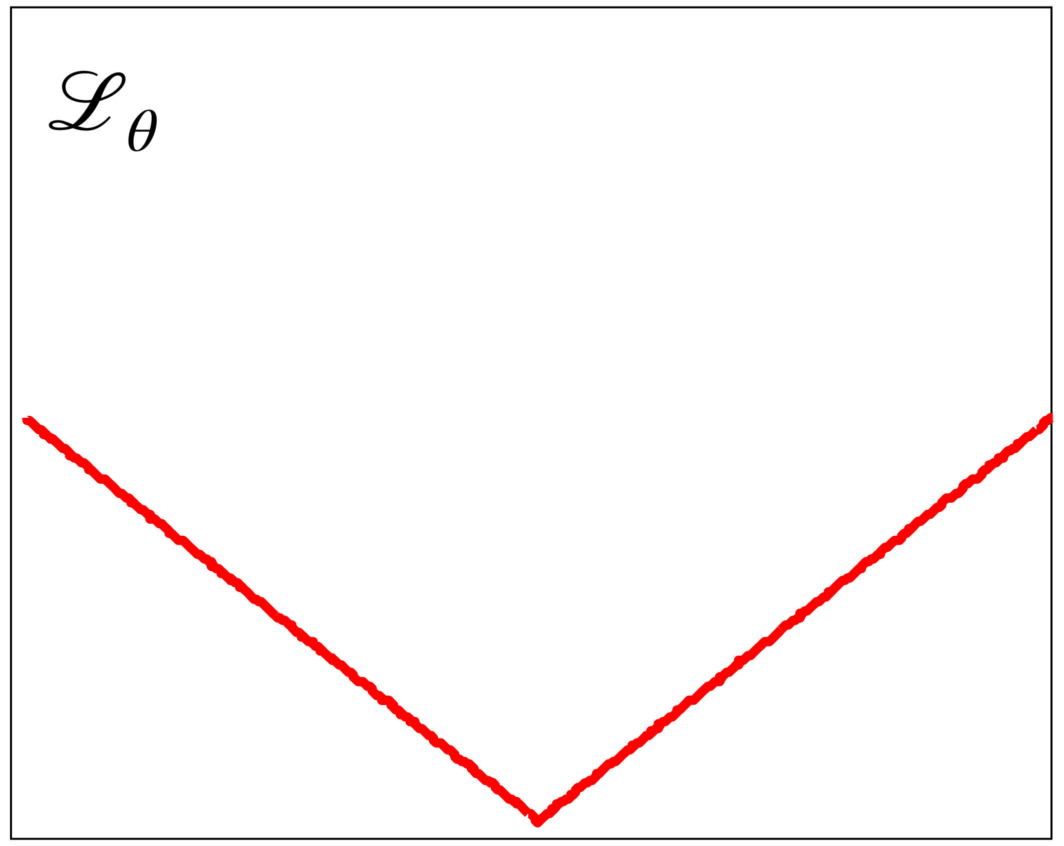
Minimizing MAE

- The absolute operator isn't differentiable
  - But assume no data point has 0 error

$$\nabla_\theta \frac{1}{m} \sum_j |y - \theta^\intercal X| = \frac{1}{m} \left(\sum_{j:\ y^{(j)} < \theta^\intercal x^{(j)}} x^{(j)} - \sum_{j:\ y^{(j)} > \theta^\intercal x^{(j)}} x^{(j)} \right) = 0$$

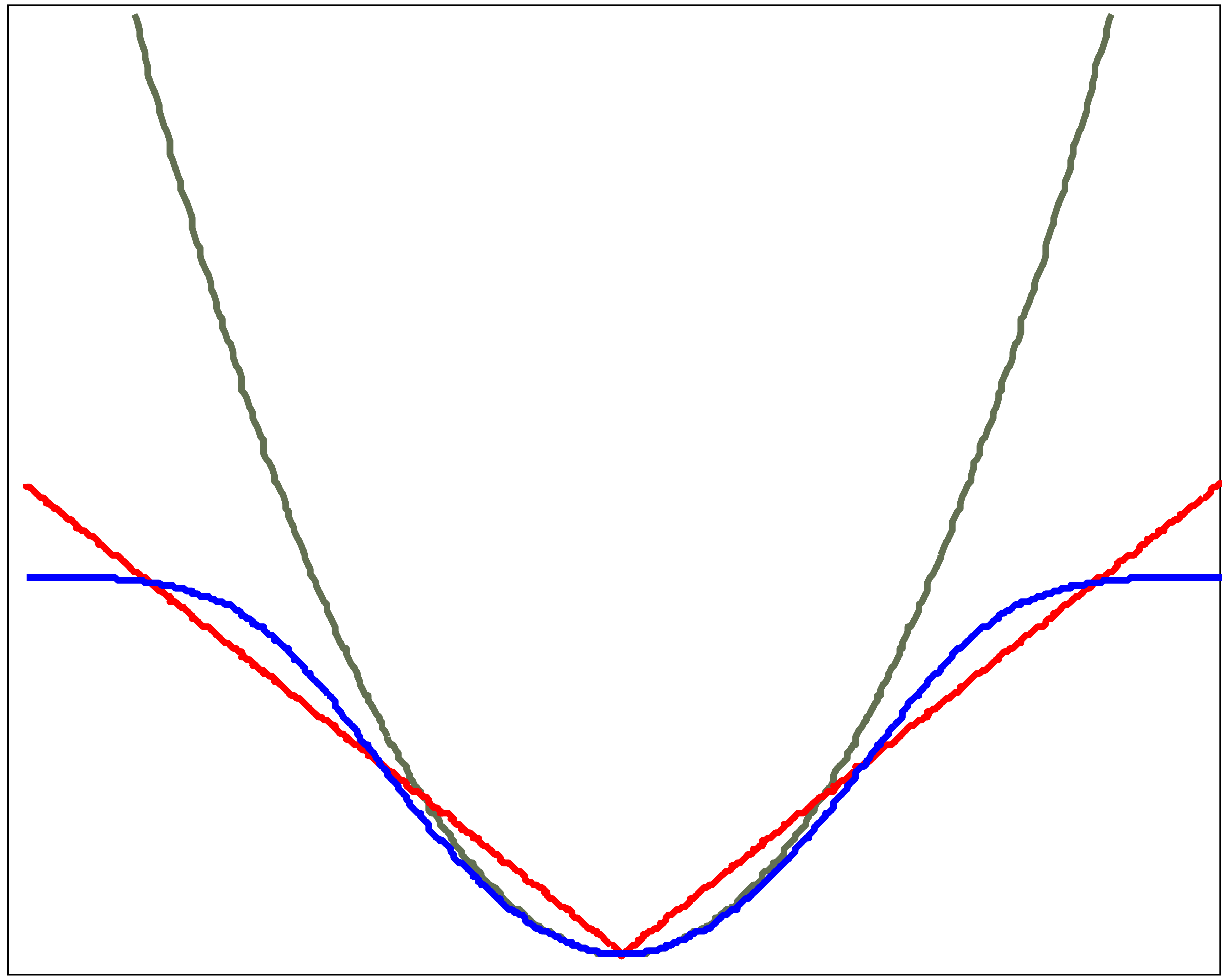
$$\sum_{j:\ y^{(j)} < \theta^\intercal x^{(j)}} x^{(j)} = \sum_{j:\ y^{(j)} > \theta^\intercal x^{(j)}} x^{(j)}$$

- Can be solved with Linear Programming
- Without features (best constant fit for y): median
  - With MSE: mean — more sensitive to outliers



Other loss functions

- MSE: $\ell(y, \hat{y}) = (y - \hat{y})^2$
- MAE: $\ell(y, \hat{y}) = |y - \hat{y}|$
- Should loss of large errors saturate?
  - $\ell(y, \hat{y}) = c - \log(\exp(-(y - \hat{y})^2) + c)$
- Most loss functions cannot be optimized in close form
  - Gradient descent is a general algorithm for differentiable parametrization and loss



Today's lecture

ROC curves

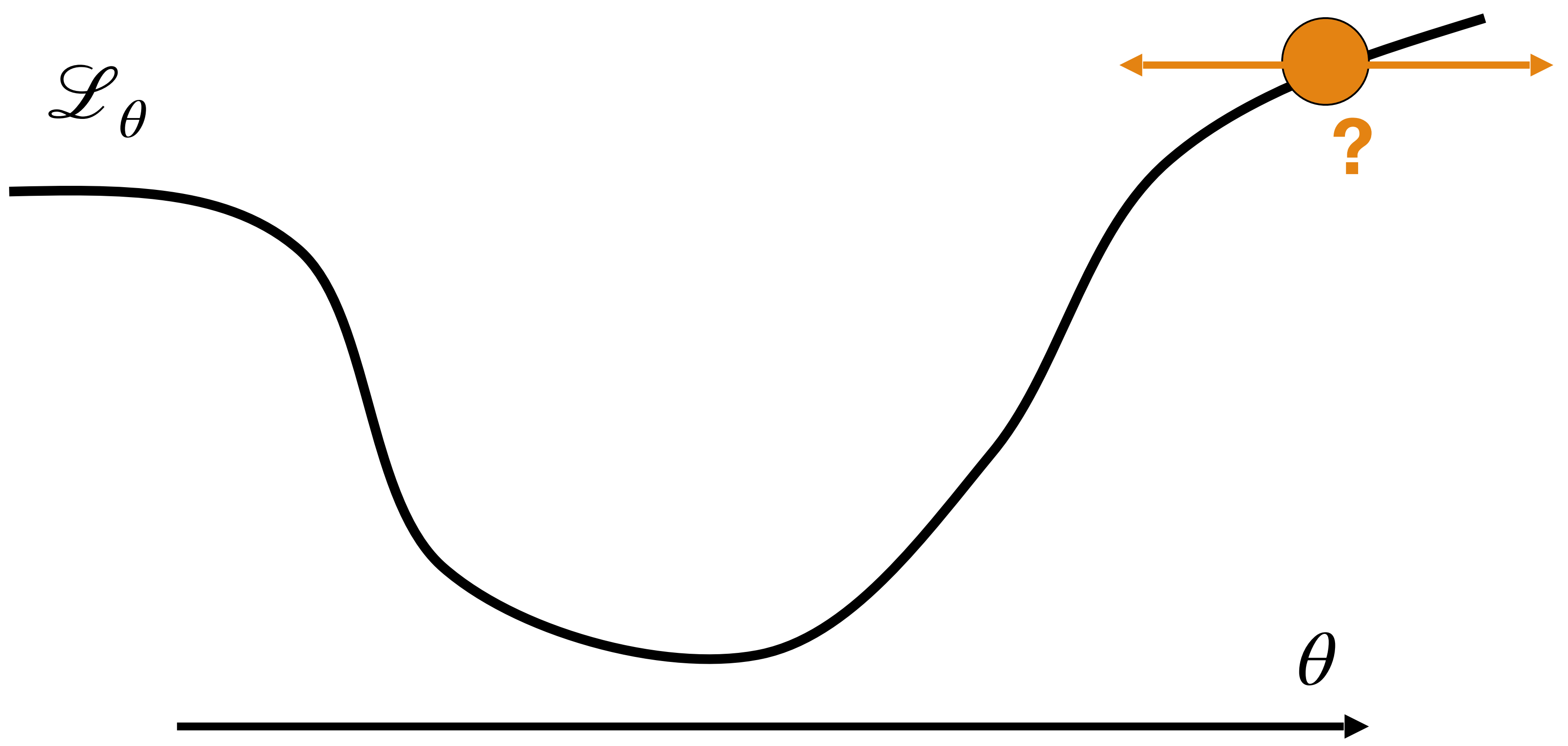
Linear regression

Least squares

Gradient descent

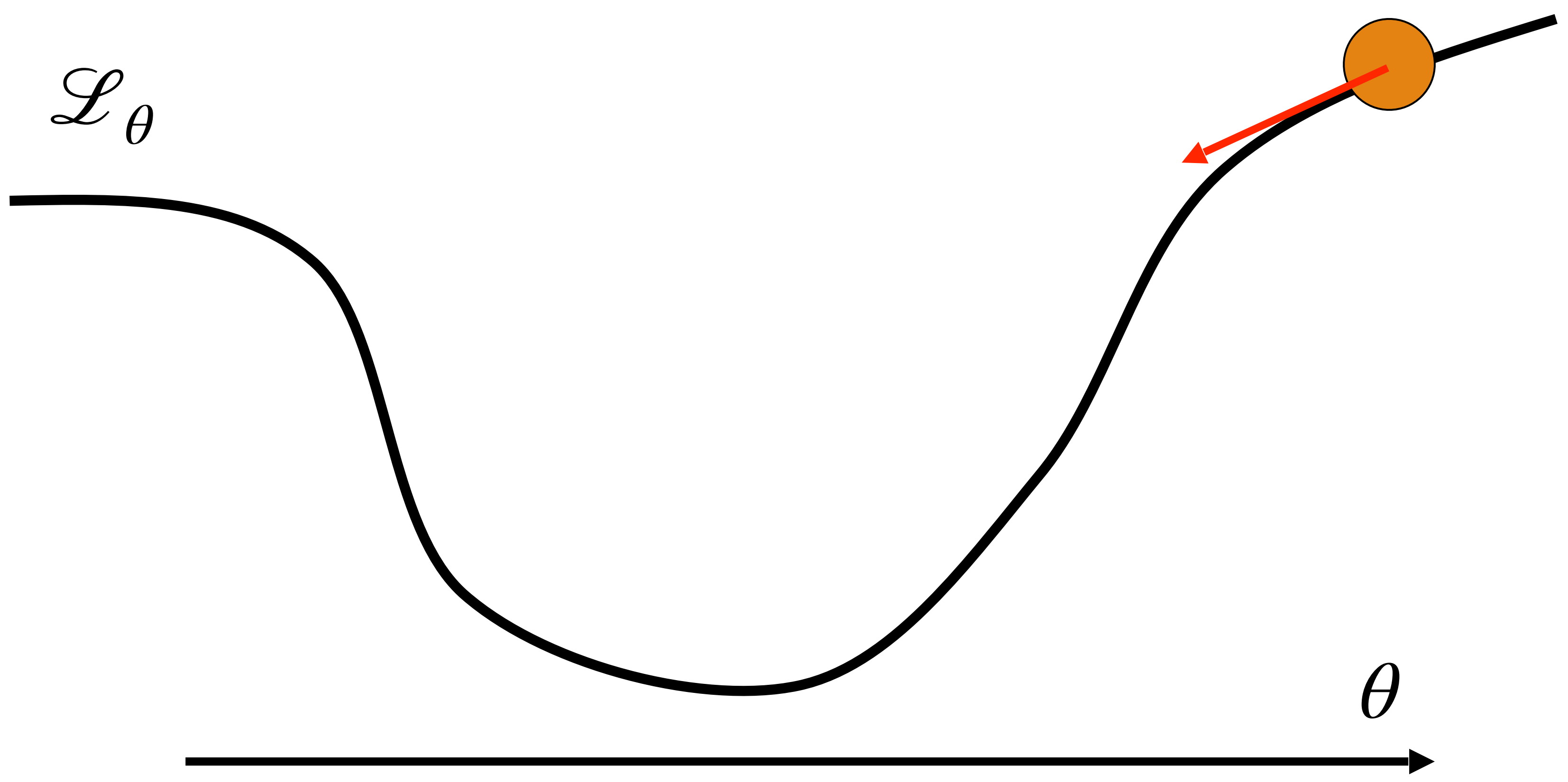
Gradient descent

- 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



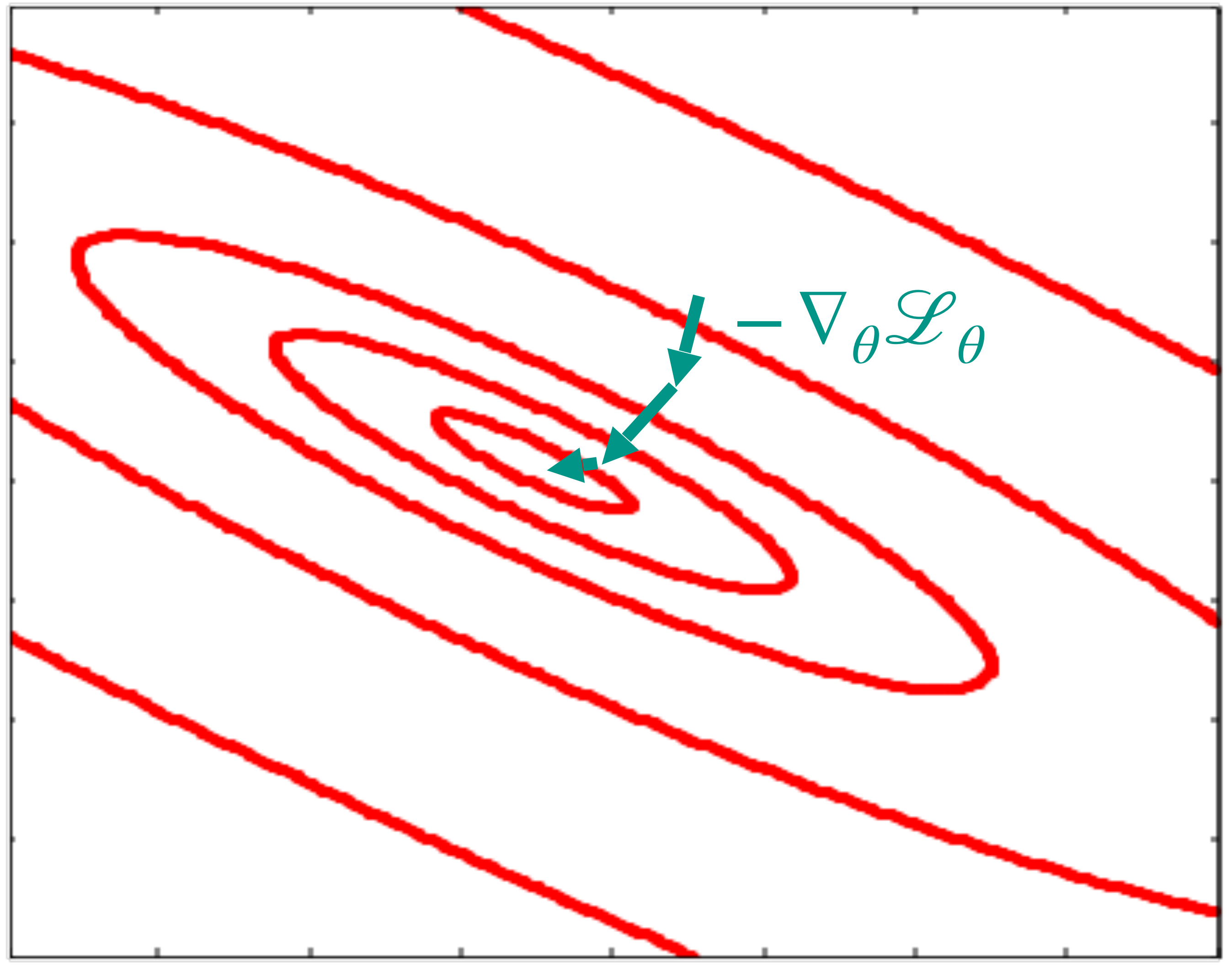
Gradient descent

- 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



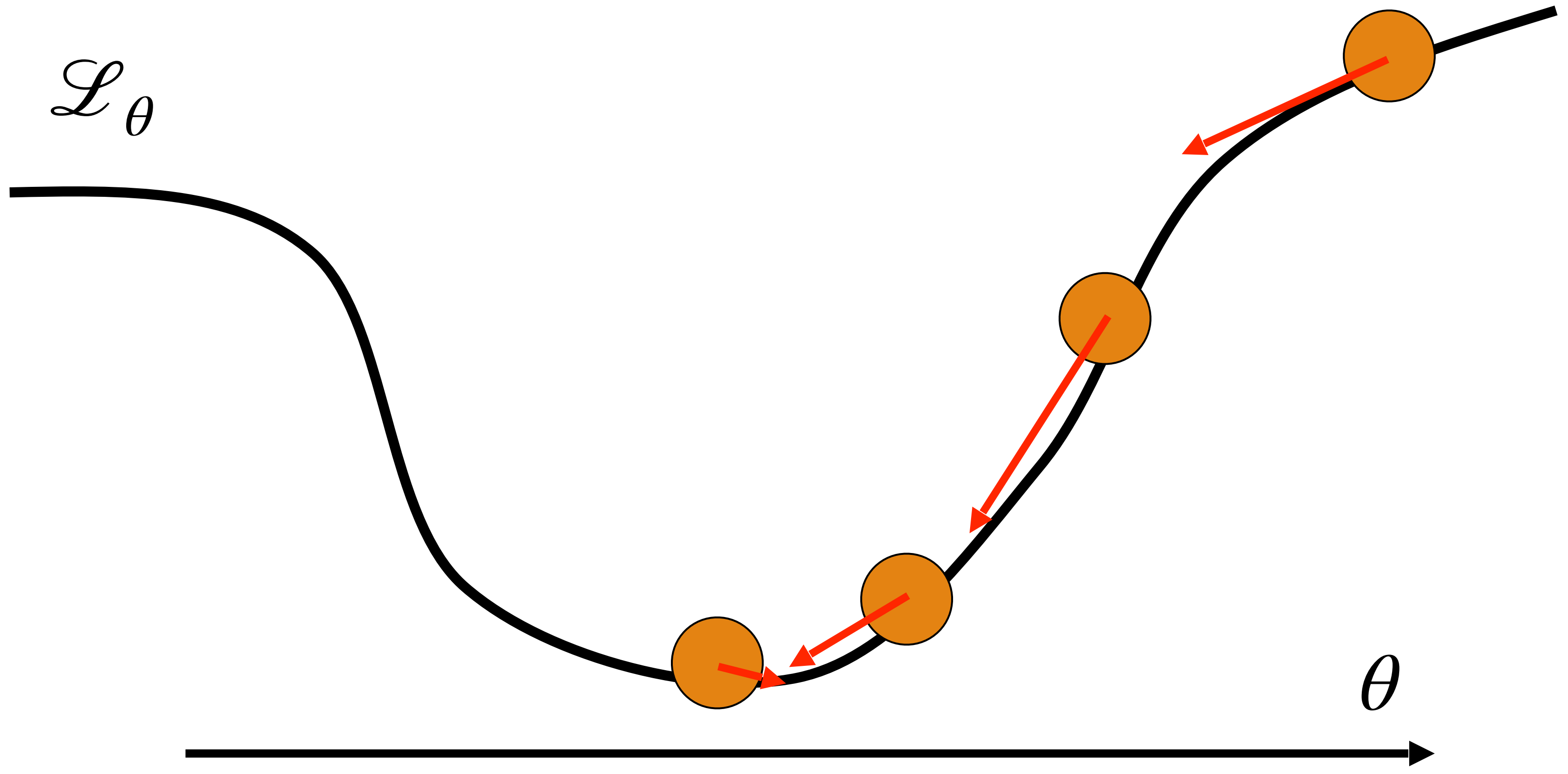
Gradient descent in higher dimension

- 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 $= -x_i^{(j)}$
  - $\partial_{\theta_i}\mathscr{L}_\theta = -\frac{2}{m}\sum_j \epsilon^{(j)} x_i^{(j)} = -\frac{2}{m}(y - \theta^\intercal X)X_i^\intercal$
- $\nabla_\theta \mathscr{L}_\theta = -\frac{2}{m}(y - \theta^\intercal X)X^\intercal$

error

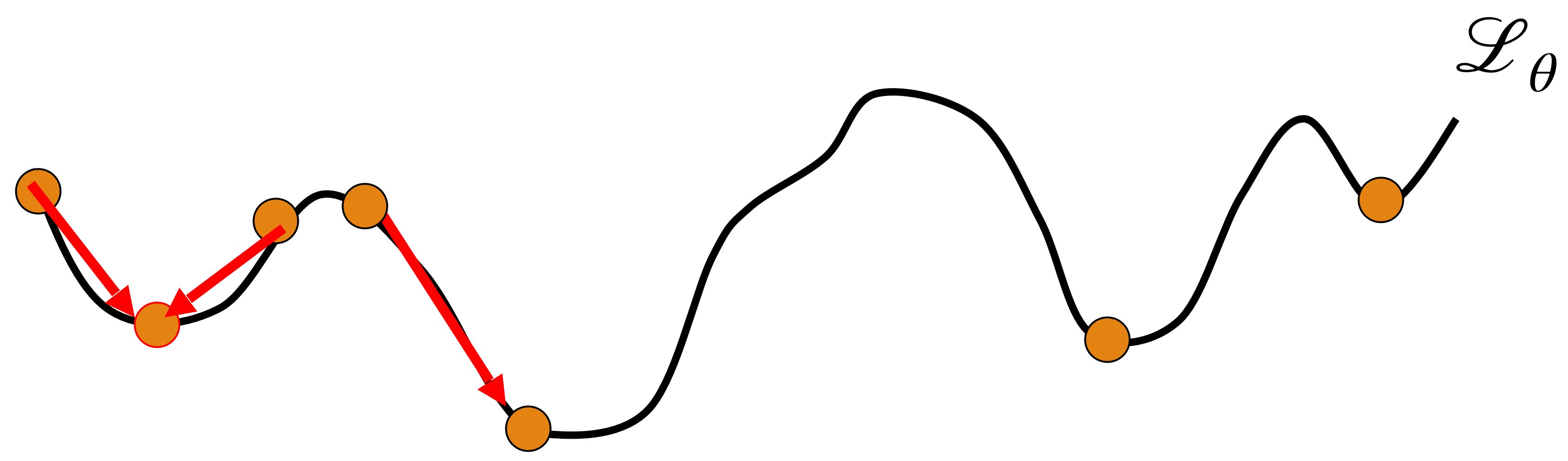
sensitivity to θ

- 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?

Newton's method

- Given black-box $f(z)$, how to find a root $f(z) = 0$?
- Initialize some z
- Repeat:
  - Evaluate $f(z)$ and $\partial_z f(z)$ to find tangent to f at z: $f'(z') = (z' - z)\partial_z f(z) + f(z)$
  - Update z to the root of f': $z \leftarrow z - \dfrac{f(z)}{\partial_z f(z)}$
- Considerations:
  - May not converge, sometimes unstable
  - Usually converges quickly for nice, smooth, locally quadratic functions

Newton's method for gradient descent

- We want to find a (local) minimum $f(\theta) = \nabla_\theta \mathscr{L}_\theta = 0$
- Initialize some θ
- Repeat:
  - Evaluate gradient $g = \nabla_\theta \mathscr{L}_\theta$ and Hessian $H = \nabla_\theta^2 \mathscr{L}_\theta$
  - Update $\theta \leftarrow \theta - H^{-1} g$
- Considerations:
  - Update step may be too large for highly non-convex losses
  - Computational complexity to invert H: $O(n^3)$

Gradient Descent: complexity

- Assume $\mathscr{L}_\theta(\mathscr{D}) = \frac{1}{m}\sum_j \ell_\theta(x^{(j)}, y^{(j)})$
    - MSE: $\ell_\theta(x, y) = (y - \theta^\intercal x)^2$
- Computing $\nabla_\theta \mathscr{L}_\theta = \frac{1}{m}\sum_j \nabla_\theta \ell_\theta^{(j)}$: usually $O(mn)$
    - What if we use really large datasets? ("big data")
    - What if we learn from data streams? (more data keeps coming in...)

Stochastic / Online Gradient Descent

- Estimate $\nabla_\theta \mathscr{L}_\theta$ fast on a sample of data points
- For each data point:

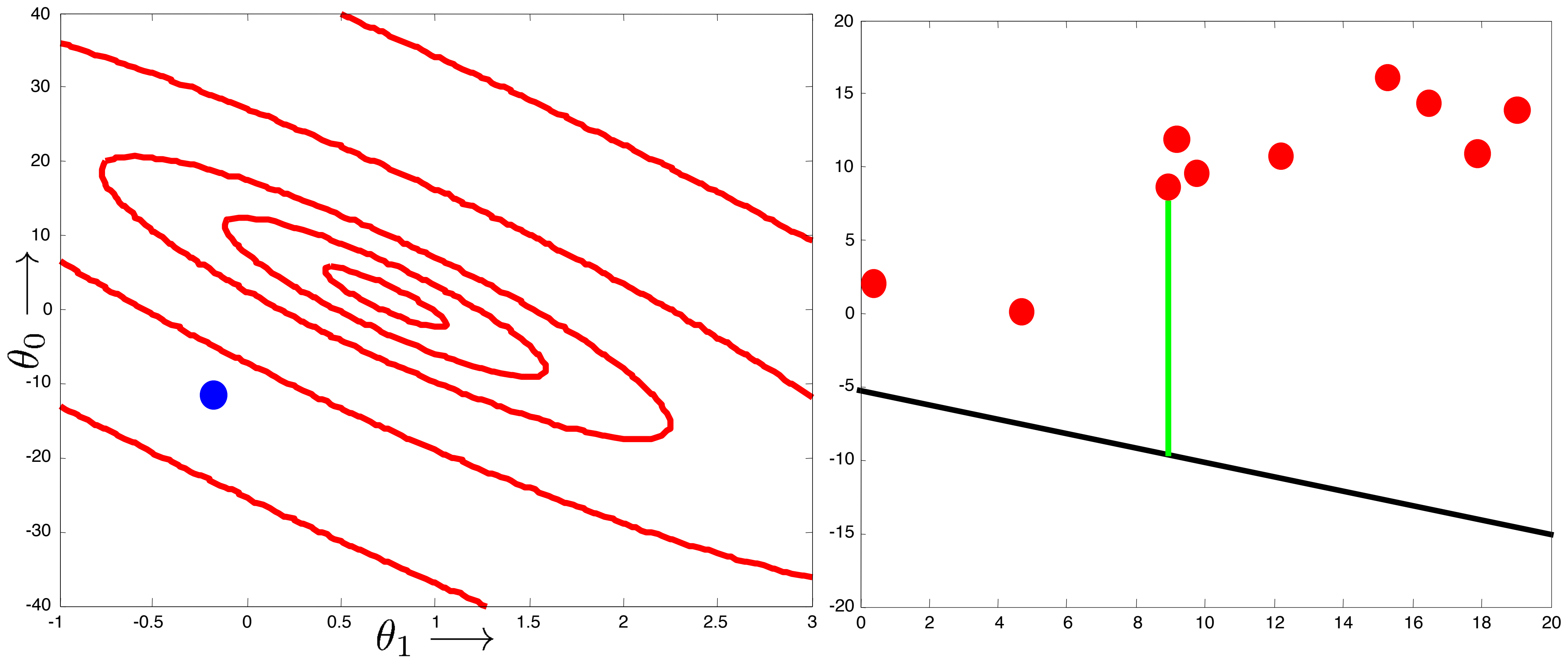
$$\nabla_\theta \mathscr{L}_\theta(x^{(j)}, y^{(j)}) = \nabla_\theta (y^{(j)} - \theta^\intercal x^{(j)})^2 = -2(y^{(j)} - \theta^\intercal x^{(j)})(x^{(j)})^\intercal$$

- This is an unbiased estimator of the gradient, i.e. in expectation

$$\mathbb{E}_{j \sim \text{Uniform}(1,\ldots,m)}[\nabla_\theta \mathscr{L}_\theta^{(j)}] = \frac{1}{m} \sum_j \nabla_\theta \mathscr{L}_\theta^{(j)} = \nabla_\theta \mathscr{L}_\theta(\mathscr{D})$$

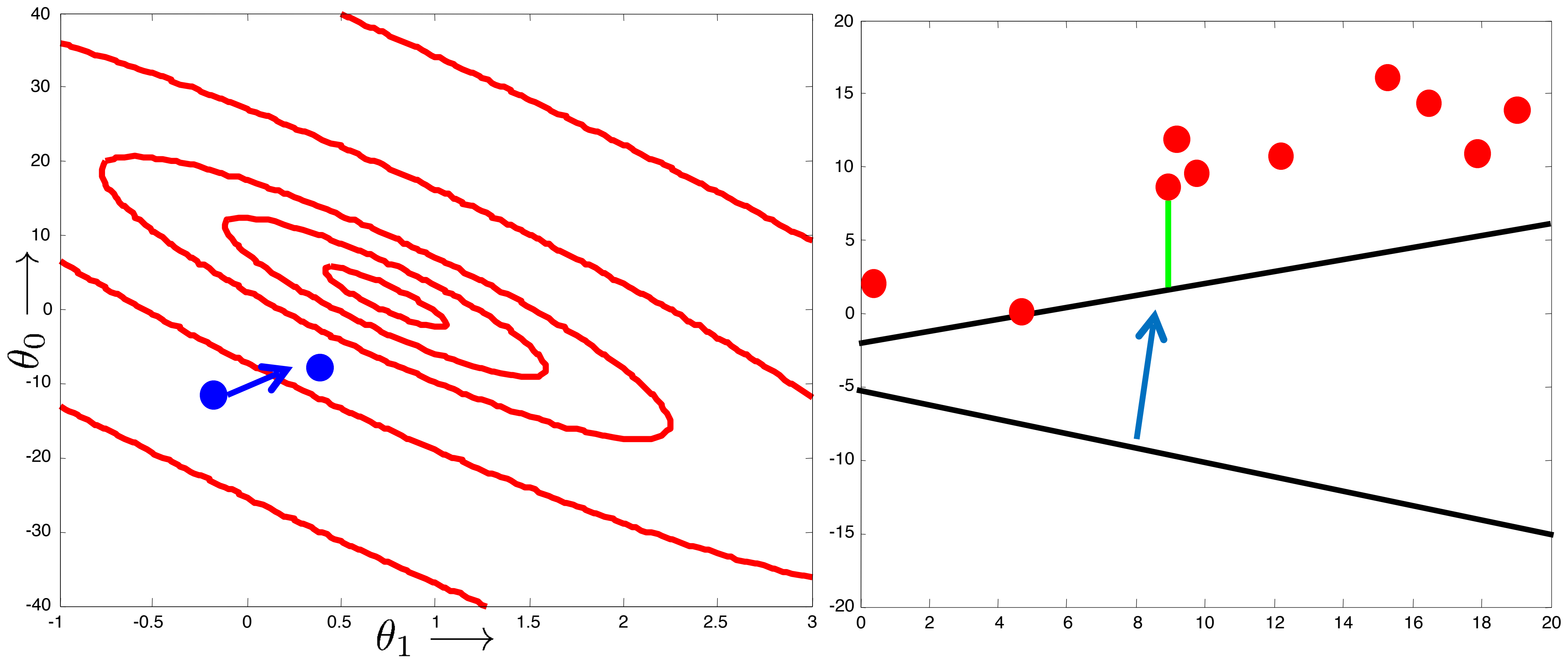
- $\nabla_\theta \mathscr{L}_\theta(\mathscr{D})$ is already a noisy unbiased estimator of true gradient $\mathbb{E}_{x,y \sim p}[\nabla_\theta \mathscr{L}_\theta(x, y)]$
  - SGD is even more noisy

Stochastic Gradient Descent



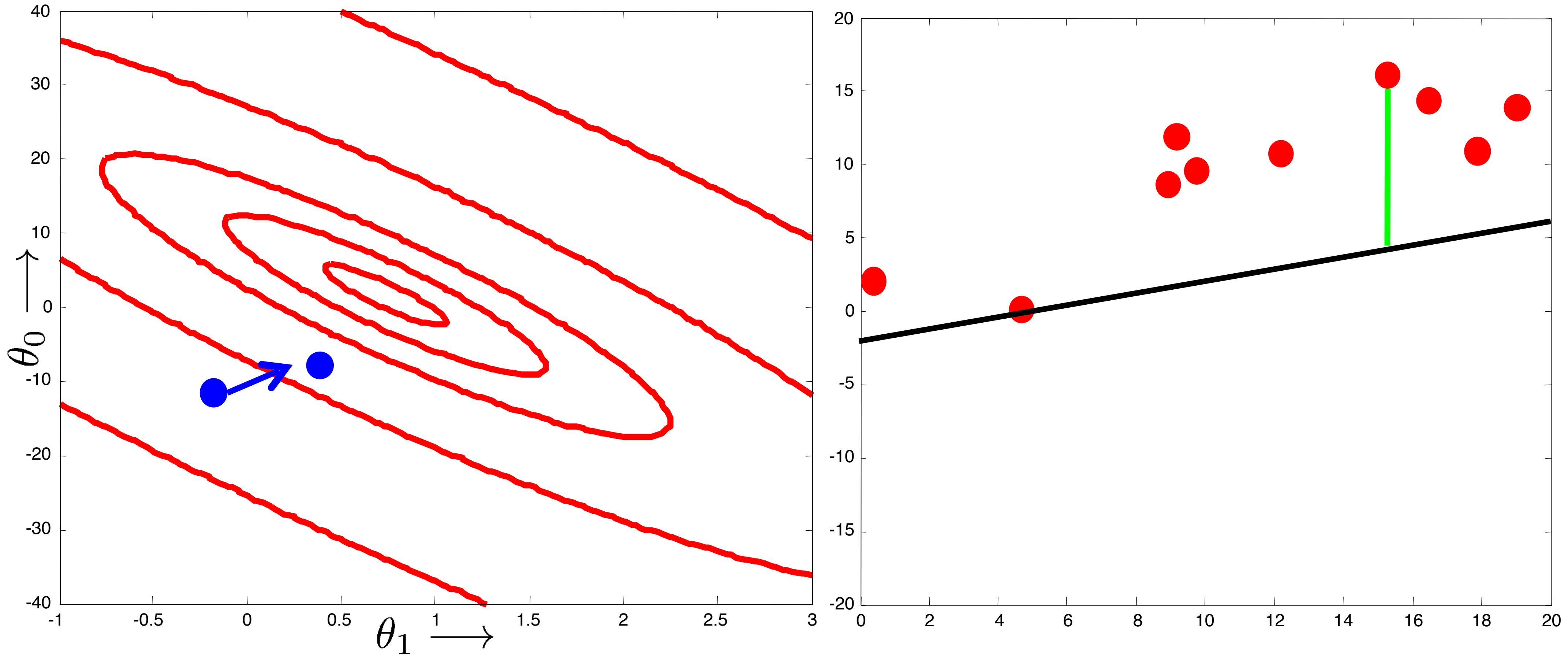
- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



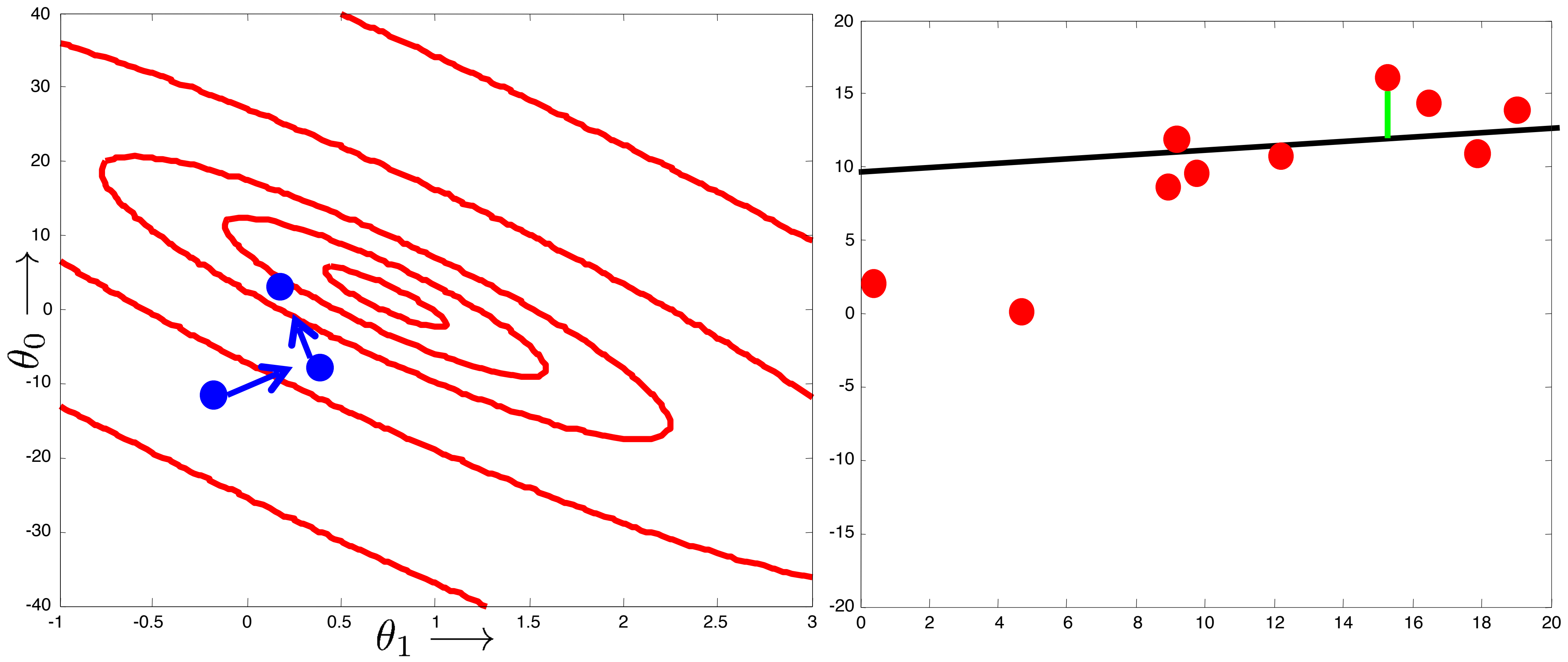
- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no average improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



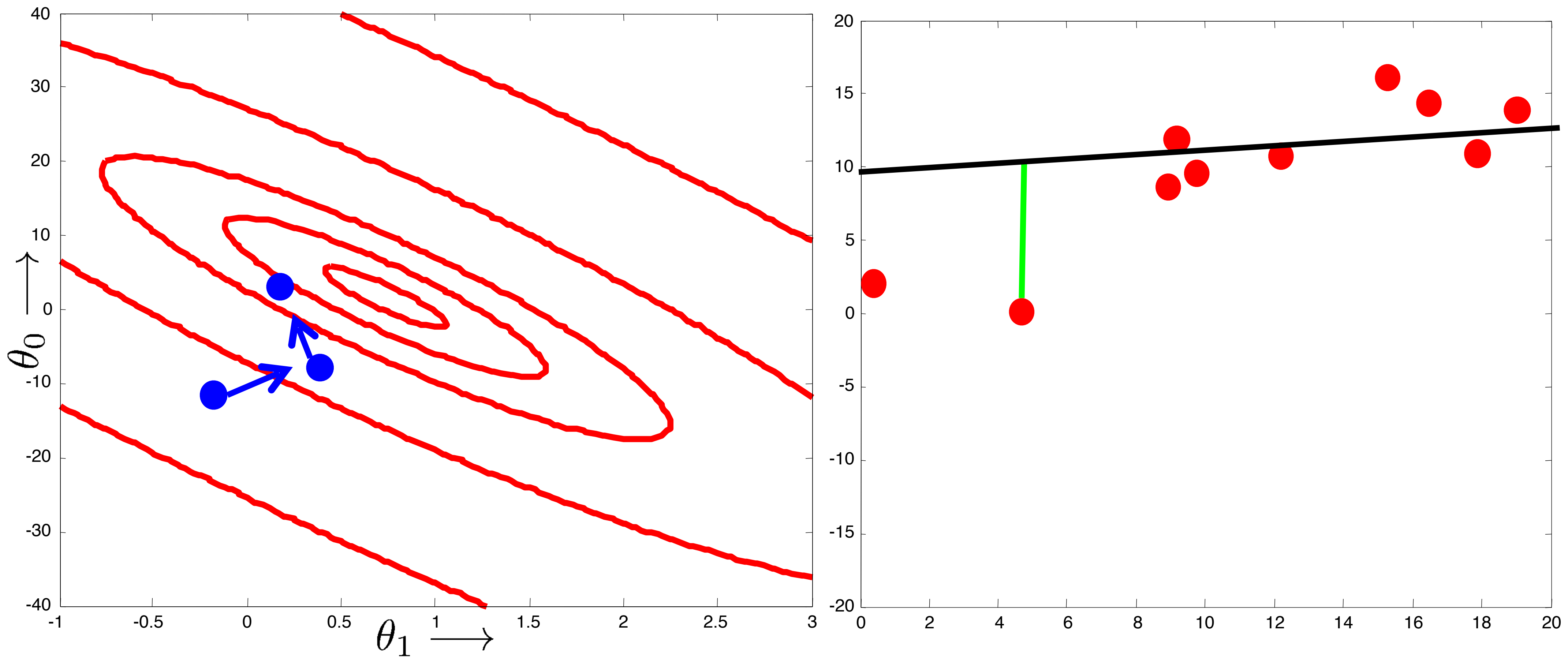
- Initialize θ
- Repeat:
  - Sample $j \sim \mathrm{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no average improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



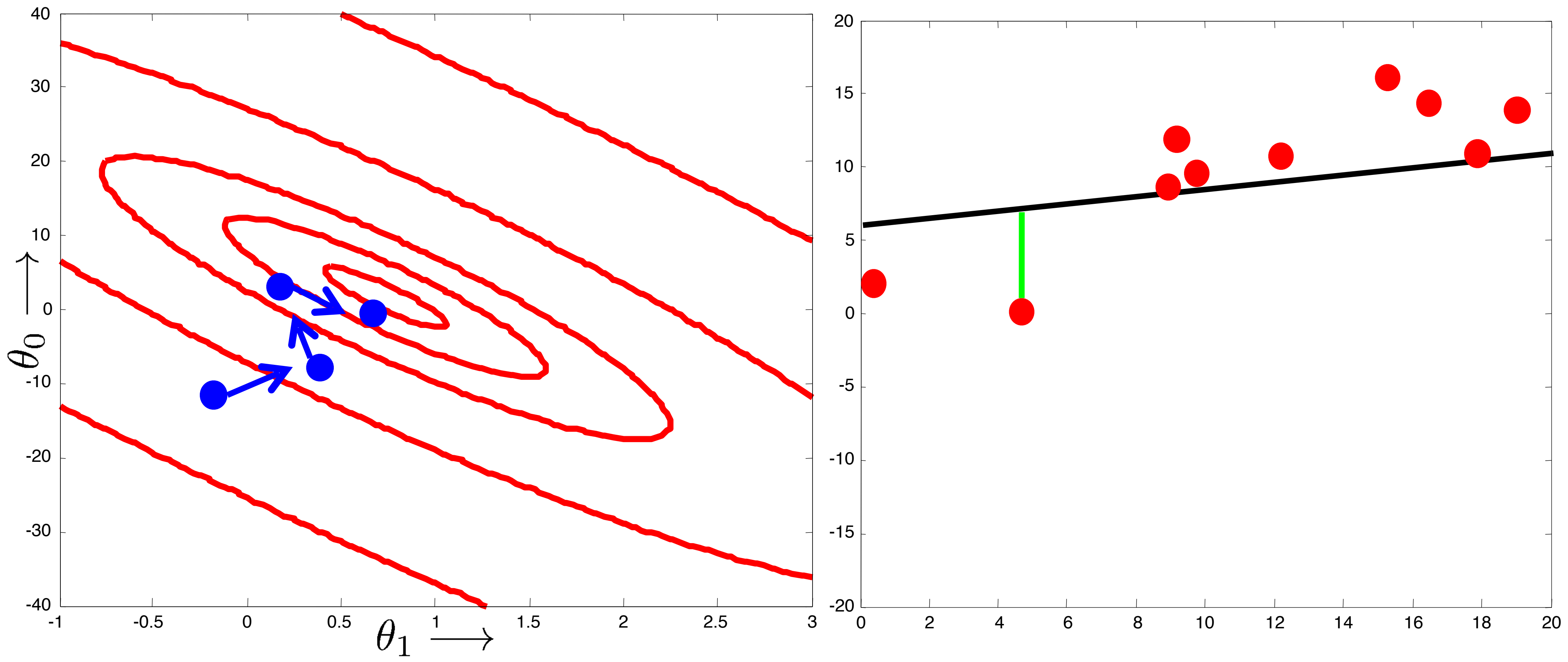
- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathcal{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no average improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent



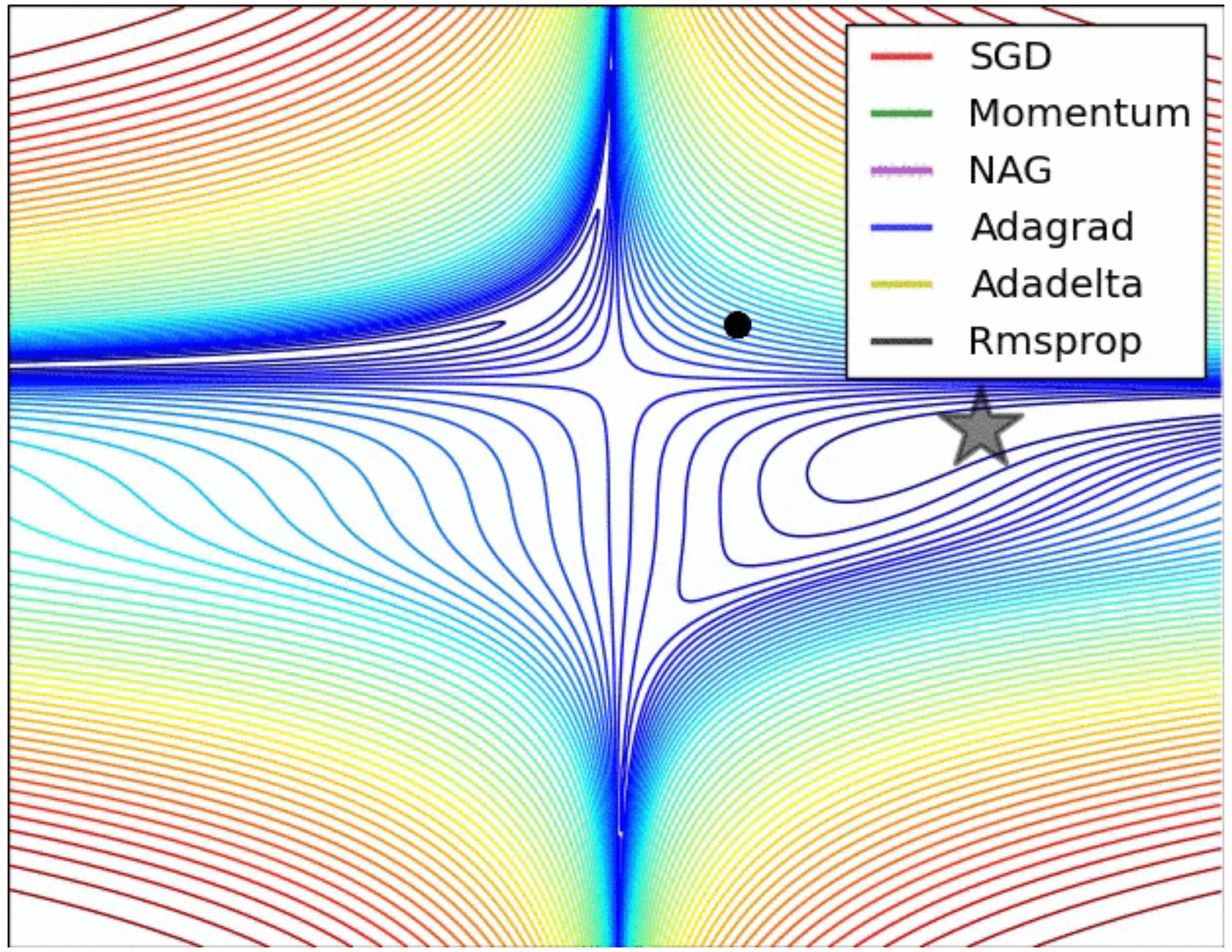
- Initialize θ
- Repeat:
  - Sample $j \sim \text{Uniform}(1,\ldots,m)$
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathscr{L}_\theta^{(j)}$
- Until some stop criterion; e.g., no <u>average</u> improvement in $\mathscr{L}_\theta^{(j)}$ for a while

Stochastic Gradient Descent: considerations

- Benefits:
  - Each gradient step is faster
  - Don't wait for all data with same θ, improve θ “early and often”
  - Arguably the most important optimization algorithm nowadays
- Drawbacks:
  - May not actually descend on training loss
  - Stopping conditions may be harder to evaluate
- Mini-batch updates: draw $b \ll m$ data points
  - $$\operatorname{var} \nabla_\theta \mathscr{L}_\theta(\text{batch}) = \operatorname{var} \frac{1}{b} \sum_{j \in \text{batch}} \nabla_\theta \mathscr{L}_\theta^{(j)} = \frac{1}{b} \operatorname{var} \nabla_\theta \mathscr{L}_\theta(\text{point})$$
  - Variance increases the smaller the batch size
    - Generally bad, but can help overcome local minima / saddle points

Advanced gradient-based methods

- Momentum
  - Gradient is like velocity in parameter space
    - Previous gradients still carry momentum
  - Smoothens SGD path
  - Effectively averages gradients over steps, reduces variance



- Preconditioning
  - Scale and rotate loss landscape to make it nicer
  - E.g., multiply by inverse Hessian (as in Newton's method)

Logistics

assignments

- Assignment 1 due today
- Assignment 2 to be published soon