

+

# Machine Learning and Data Mining

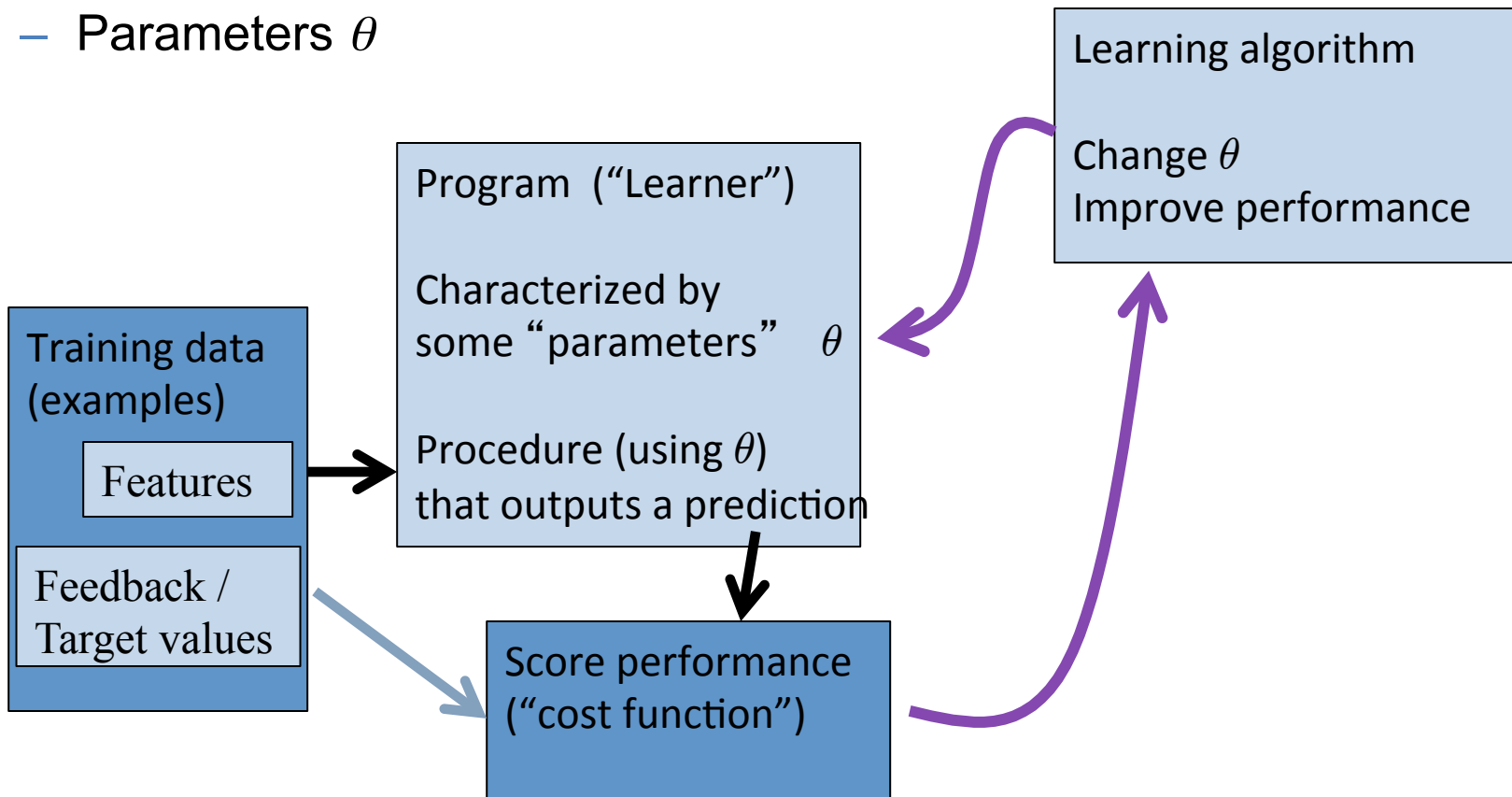
## Linear regression

Prof. Alexander Ihler

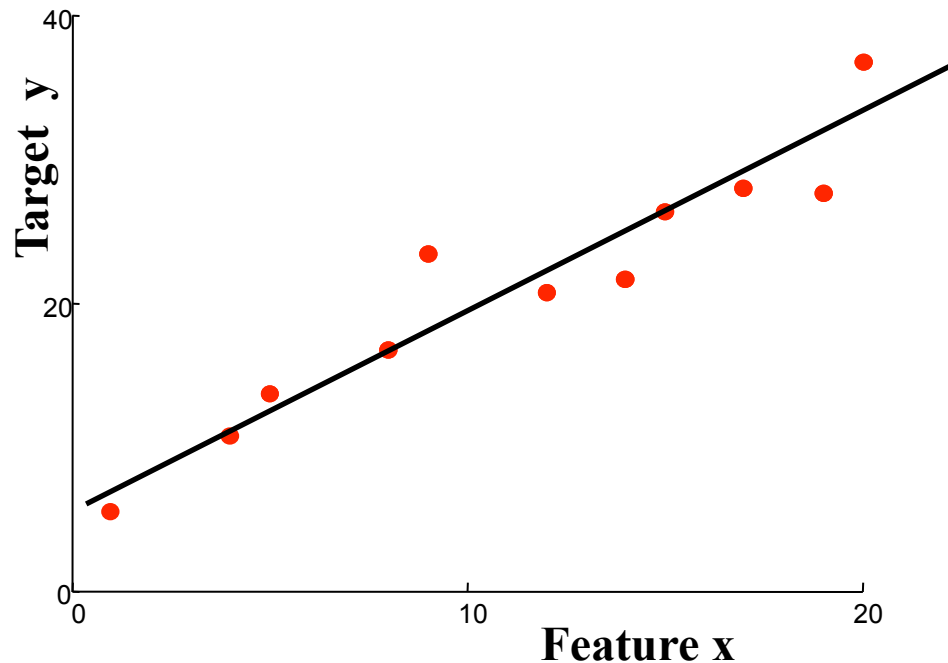


# Supervised learning

- Notation
  - Features  $x$
  - Targets  $y$
  - Predictions  $\hat{y}$
  - Parameters  $\theta$



# Linear regression



**“Predictor”:**

Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

- Define form of function  $f(x)$  explicitly
- Find a good  $f(x)$  within that family

# Notation

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Define “feature”  $x_0 = 1$  (constant)

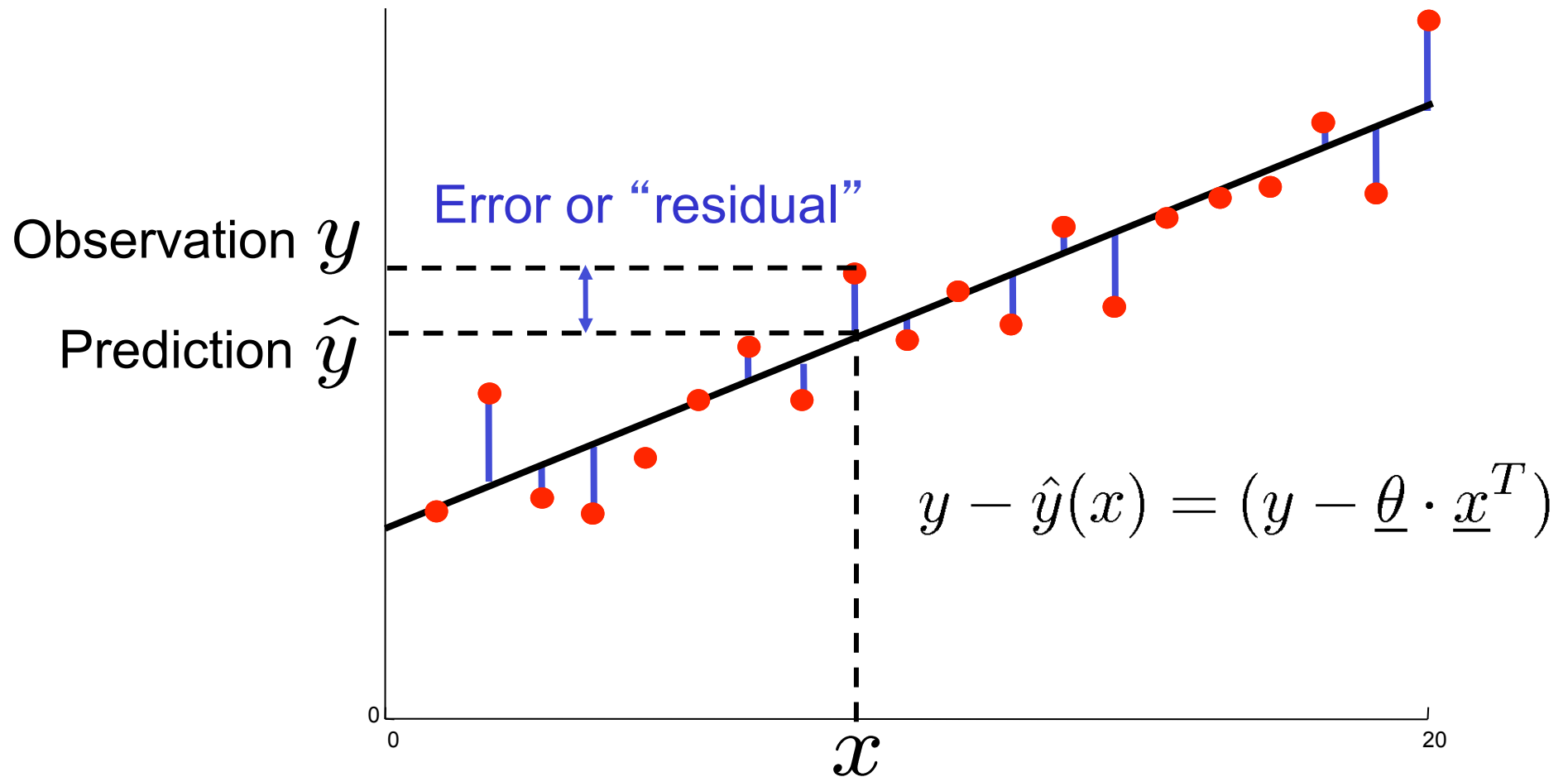
Then

$$\hat{y}(x) = \theta x^T$$

$$\underline{\theta} = [\theta_0, \dots, \theta_n]$$

$$\underline{x} = [1, x_1, \dots, x_n]$$

# Measuring error



# Mean squared error

- How can we quantify the error?

$$\begin{aligned}\text{MSE, } J(\underline{\theta}) &= \frac{1}{m} \sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2 \\ &= \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2\end{aligned}$$

- Could choose something else, of course...
  - Computationally convenient (more later)
  - Measures the variance of the residuals
  - Corresponds to likelihood under Gaussian model of “noise”

$$\mathcal{N}(y ; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{1}{2\sigma^2} (y - \mu)^2 \right\}$$

# MSE cost function

$$\begin{aligned}\text{MSE, } J(\underline{\theta}) &= \frac{1}{m} \sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2 \\ &= \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2\end{aligned}$$

- Rewrite using matrix form

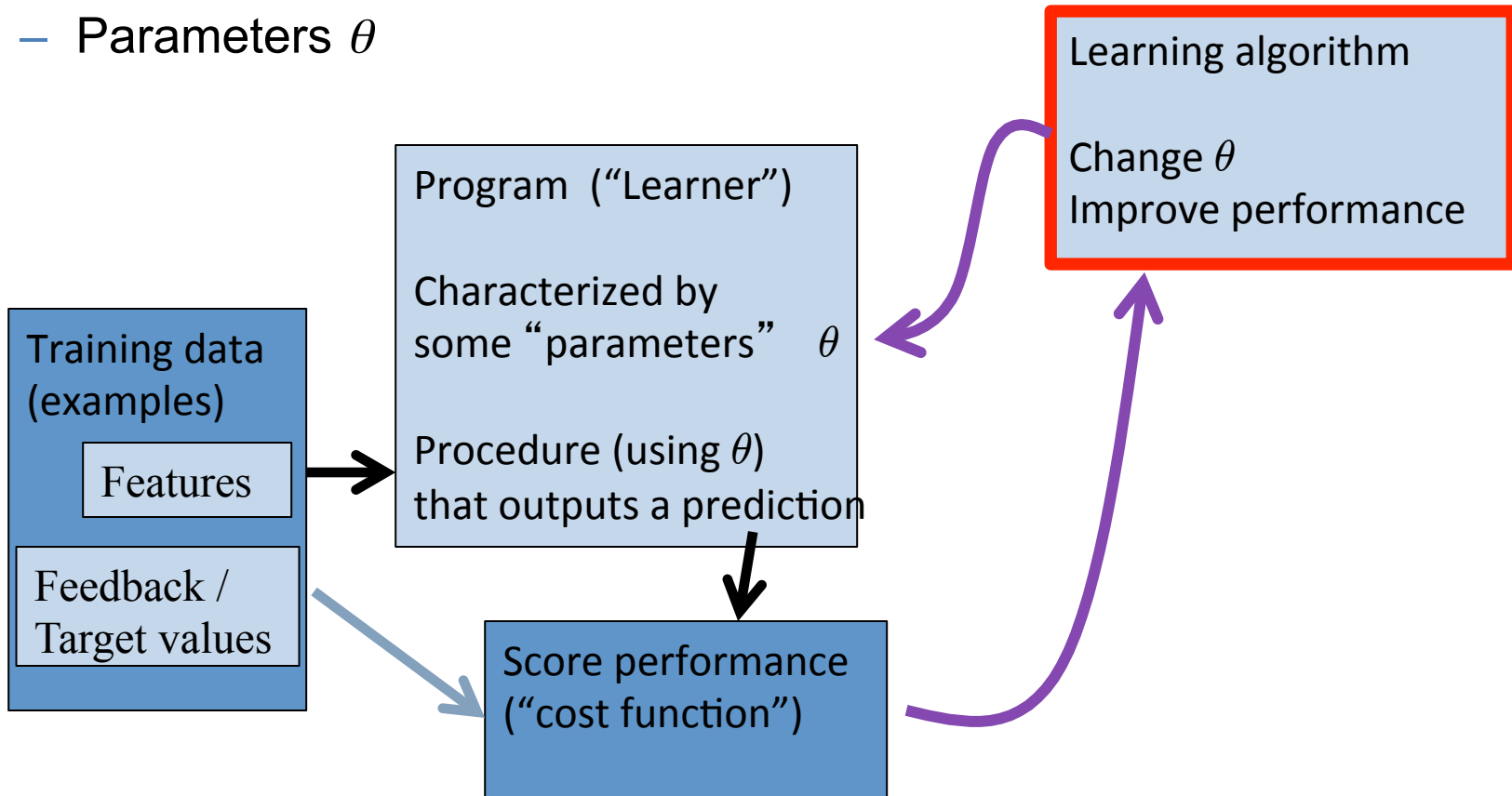
$$\begin{aligned}\underline{\theta} &= [\theta_0, \dots, \theta_n] \\ \underline{y} &= [y^{(1)} \dots, y^{(m)}]^T\end{aligned}\quad \underline{X} = \begin{bmatrix} x_0^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

$$J(\underline{\theta}) = \frac{1}{m} (\underline{y}^T - \underline{\theta} \underline{X}^T) \cdot (\underline{y}^T - \underline{\theta} \underline{X}^T)^T$$

```
# Python / NumPy:  
e = Y - X.dot( theta.T );  
J = e.T.dot( e ) / m # = np.mean( e ** 2 )
```

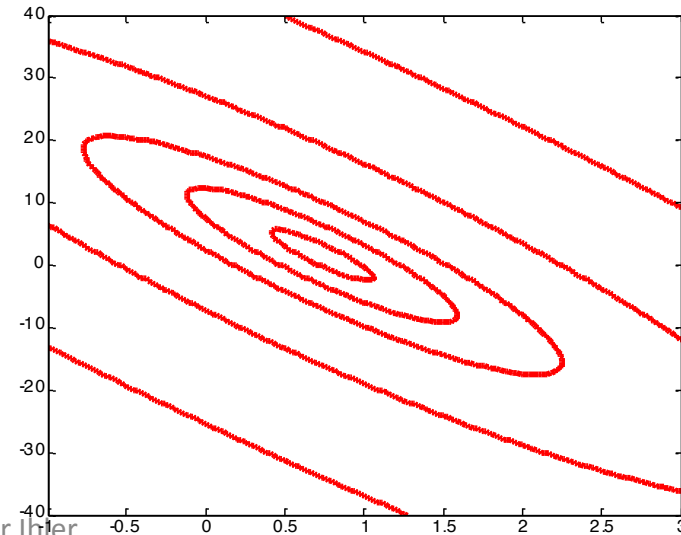
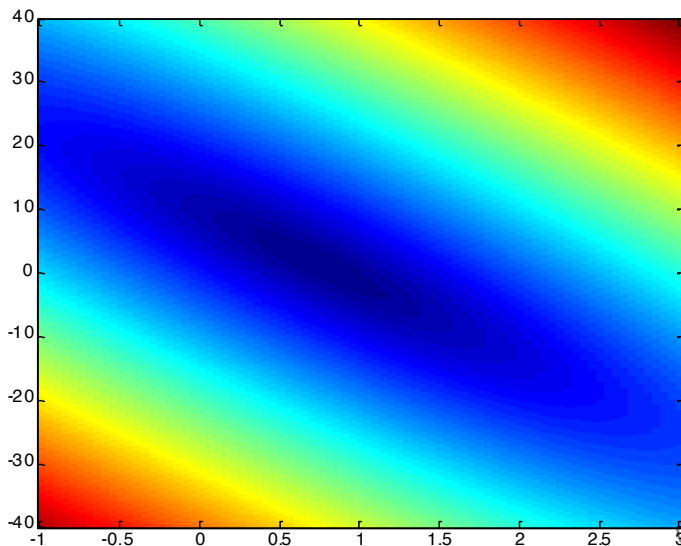
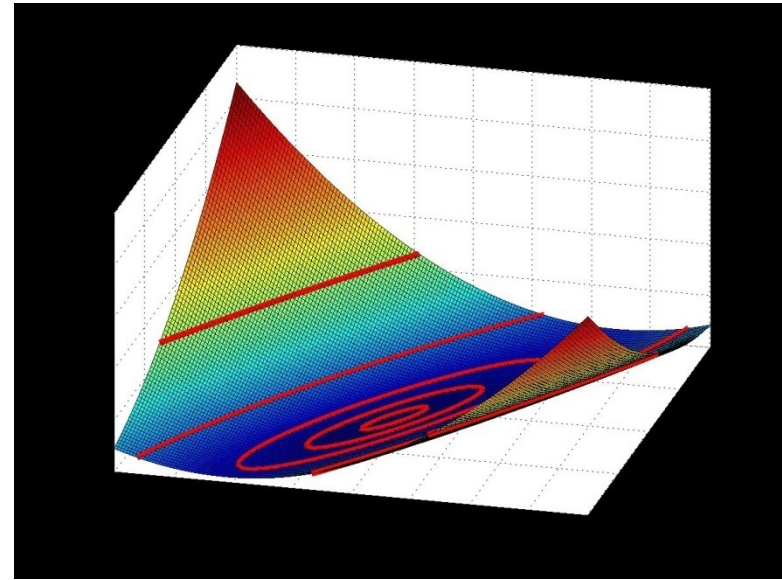
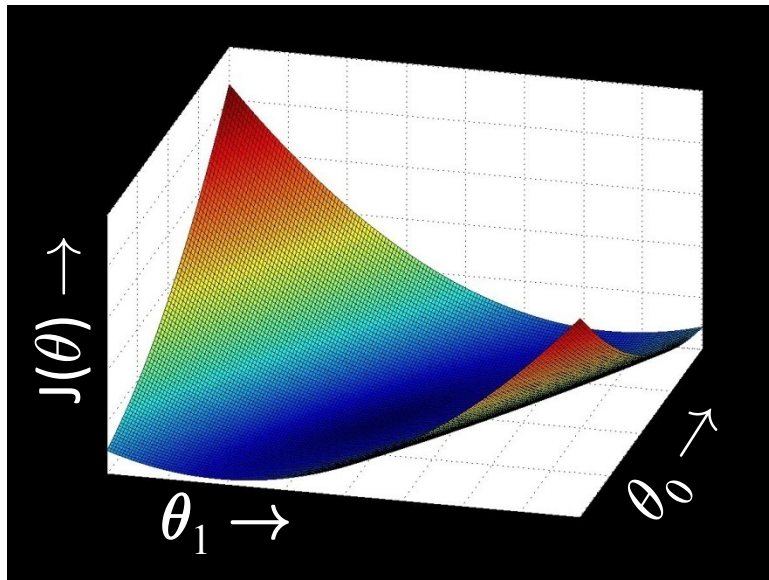
# Supervised learning

- Notation
  - Features  $x$
  - Targets  $y$
  - Predictions  $\hat{y}$
  - Parameters  $\theta$



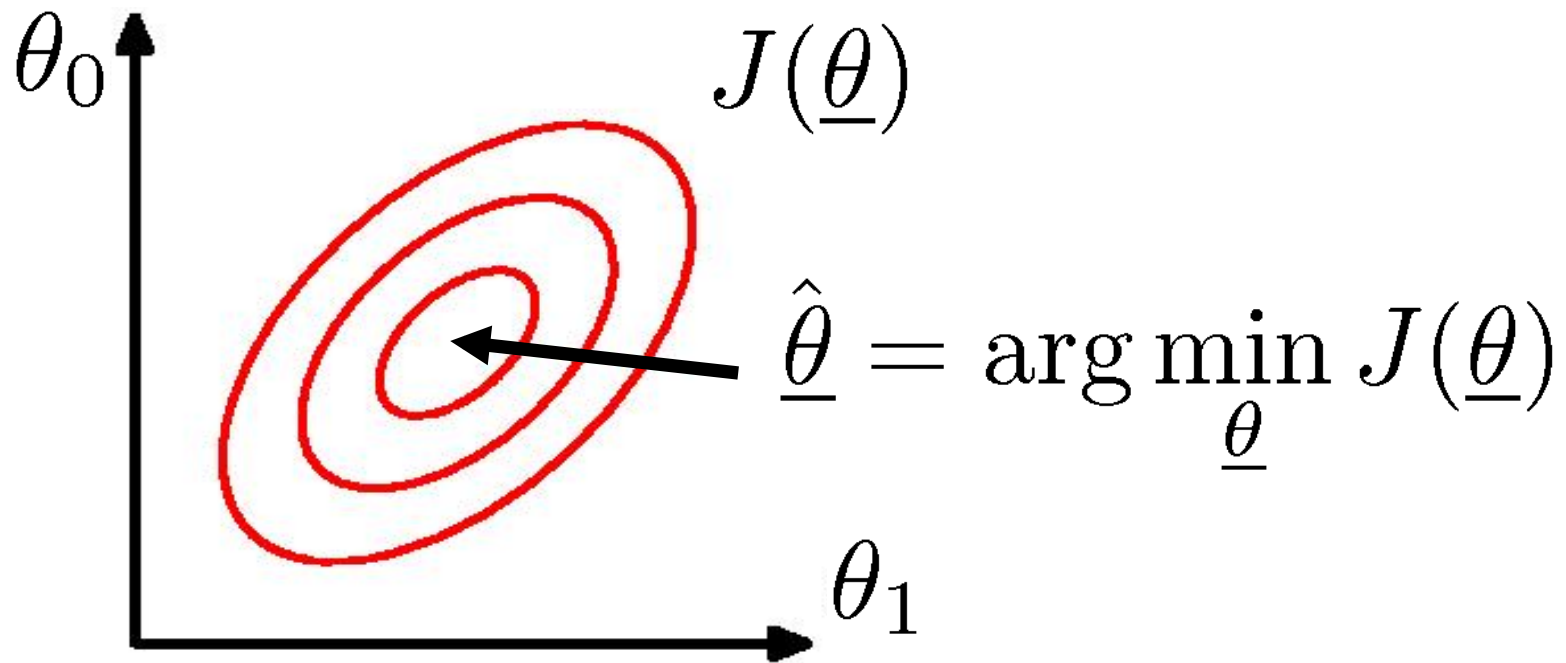


# Visualizing the cost function



# Finding good parameters

- Want to find parameters which minimize our error...
- Think of a cost “surface”: error residual for that  $\theta$ ...



+

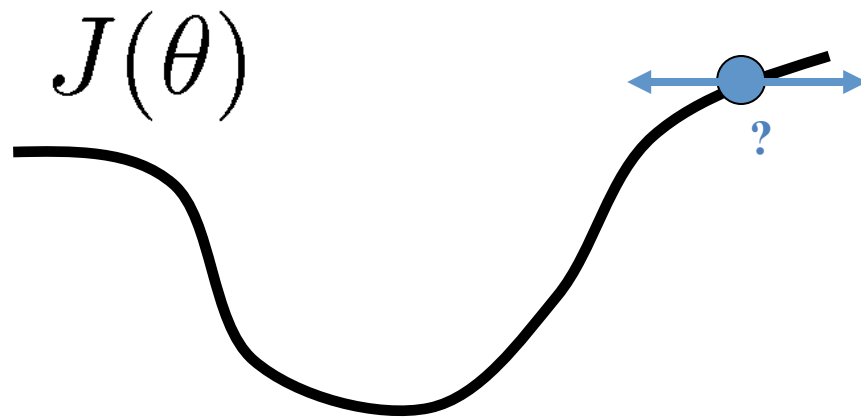
# Machine Learning and Data Mining

Linear regression:  
Gradient descent & stochastic gradient descent

Prof. Alexander Ihler

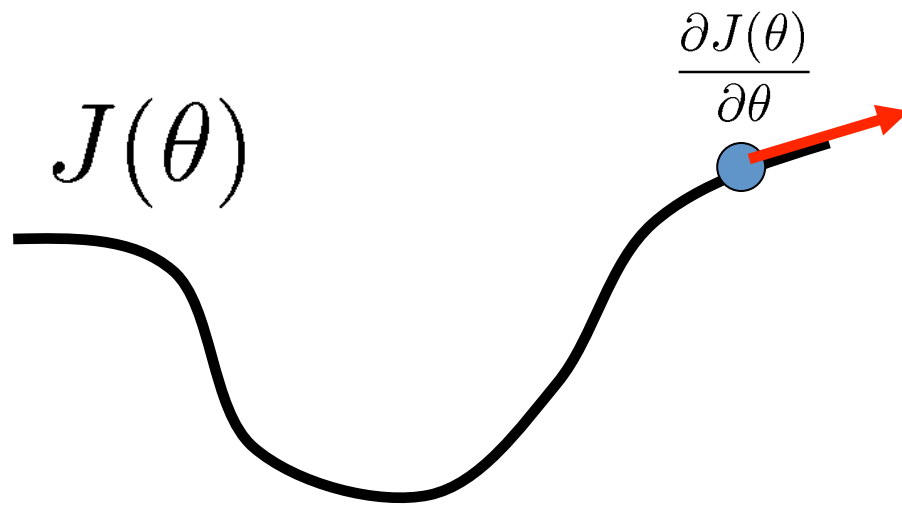


# Gradient descent



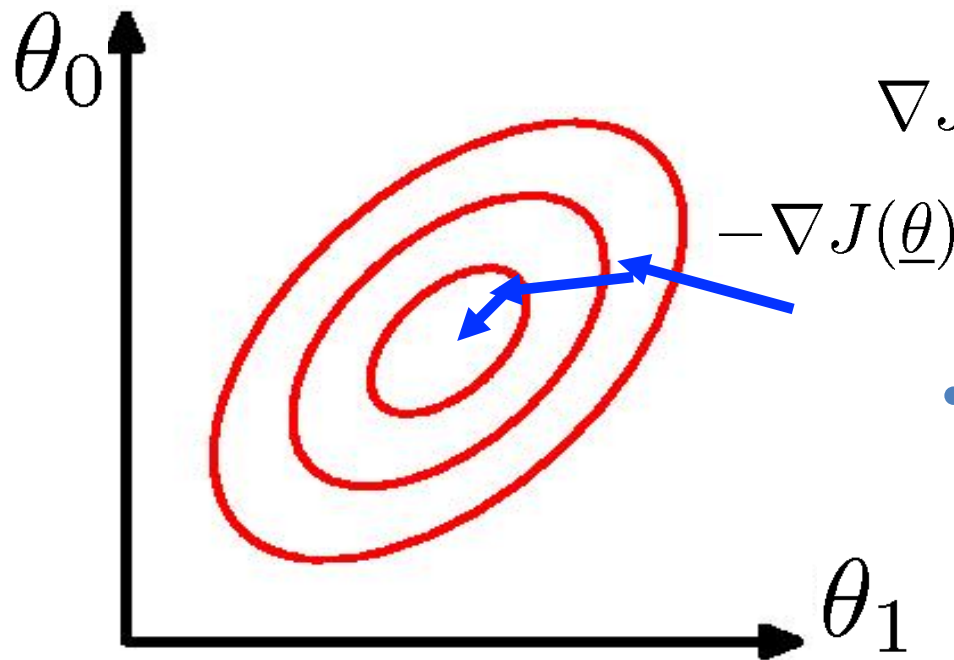
- How to change  $\theta$  to improve  $J(\theta)$ ?
- Choose a direction in which  $J(\theta)$  is decreasing

# Gradient descent



- How to change  $\theta$  to improve  $J(\theta)$ ?
- Choose a direction in which  $J(\theta)$  is decreasing
- Derivative  $\frac{\partial J(\theta)}{\partial \theta}$
- Positive  $\Rightarrow$  increasing
- Negative  $\Rightarrow$  decreasing

# Gradient descent in more dimensions



- Gradient vector

$$\nabla J(\underline{\theta}) = \begin{bmatrix} \frac{\partial J(\underline{\theta})}{\partial \theta_0} & \frac{\partial J(\underline{\theta})}{\partial \theta_1} & \cdots \end{bmatrix}$$

- Indicates direction of steepest ascent  
(negative = steepest descent)

# Gradient descent

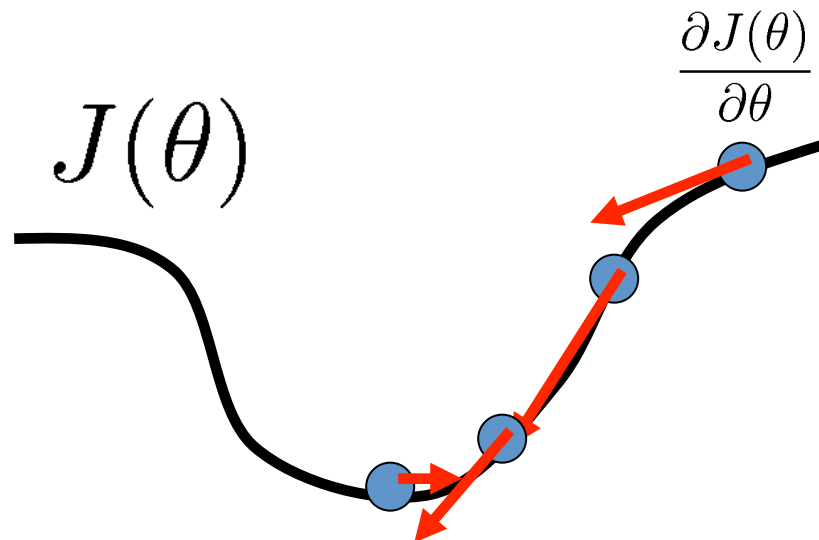
- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stopping condition

Initialize  $\theta$

Do {

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$$

} while (  $\alpha \|\nabla J\| > \epsilon$  )



(c) Alexander Ihler

# Gradient for the MSE

- MSE  $J(\underline{\theta}) = \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$

- $\nabla J = ?$

$$J(\underline{\theta}) = \frac{1}{m} \sum_j \overbrace{(y^{(j)} - \theta_0 x_0^{(j)} - \theta_1 x_1^{(j)} - \dots)}^{e_j(\theta)}^2$$

$$\frac{\partial J}{\partial \theta_0} = \frac{\partial}{\partial \theta_0} \frac{1}{m} \sum_j (e_j(\theta))^2$$

$$= \frac{1}{m} \sum_j \frac{\partial}{\partial \theta_0} (e_j(\theta))^2$$

$$= \frac{1}{m} \sum_j 2e_j(\theta) \frac{\partial}{\partial \theta_0} e_j(\theta)$$

$$\frac{\partial}{\partial \theta_0} e_j(\theta) = \cancel{\frac{\partial}{\partial \theta_0} y^{(j)}} - \frac{\partial}{\partial \theta_0} \theta_0 x_0^{(j)} - \cancel{\frac{\partial}{\partial \theta_0} \theta_1 x_1^{(j)}} - \dots$$

**0**                      **0**

$$= -x_0^{(j)}$$



# Gradient for the MSE

- MSE  $J(\underline{\theta}) = \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$

- $\nabla J = ?$   
$$J(\underline{\theta}) = \frac{1}{m} \sum_j \overbrace{(y^{(j)} - \theta_0 x_0^{(j)} - \theta_1 x_1^{(j)} - \dots)}^{e_j(\theta)}^2$$

$$\begin{aligned} \nabla J(\underline{\theta}) &= \begin{bmatrix} \frac{\partial J}{\partial \theta_0} & \frac{\partial J}{\partial \theta_1} & \dots \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{m} \sum_j -e_j(\theta) x_0^{(j)} & \frac{2}{m} \sum_j -e_j(\theta) x_1^{(j)} & \dots \end{bmatrix} \end{aligned}$$

# Gradient descent

- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stopping condition

Initialize  $\theta$

Do {

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$$

} while (  $\alpha \|\nabla J\| > \epsilon$  )

$$J(\underline{\theta}) = \frac{1}{m} \sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)})^2$$

$$\nabla J(\underline{\theta}) = -\frac{2}{m} \sum_j \underbrace{(y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)})}_{\text{Error magnitude \& direction for datum } j} \cdot \underbrace{[x_0^{(j)} \ x_1^{(j)} \ \dots]}_{\text{Sensitivity to each } \theta_i}$$

**Error magnitude &  
direction for datum  $j$**

**Sensitivity to  
each  $\theta_i$**

# Derivative of MSE

$$\nabla J(\underline{\theta}) = -\frac{2}{m} \sum_j \underbrace{(y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)})^T}_{\text{Error magnitude \& direction for datum } j} \cdot \underbrace{[x_0^{(j)} \ x_1^{(j)} \ \dots]}_{\text{Sensitivity to each } \theta_i}$$

- Rewrite using matrix form

$$\underline{\theta} = [\theta_0, \dots, \theta_n]$$

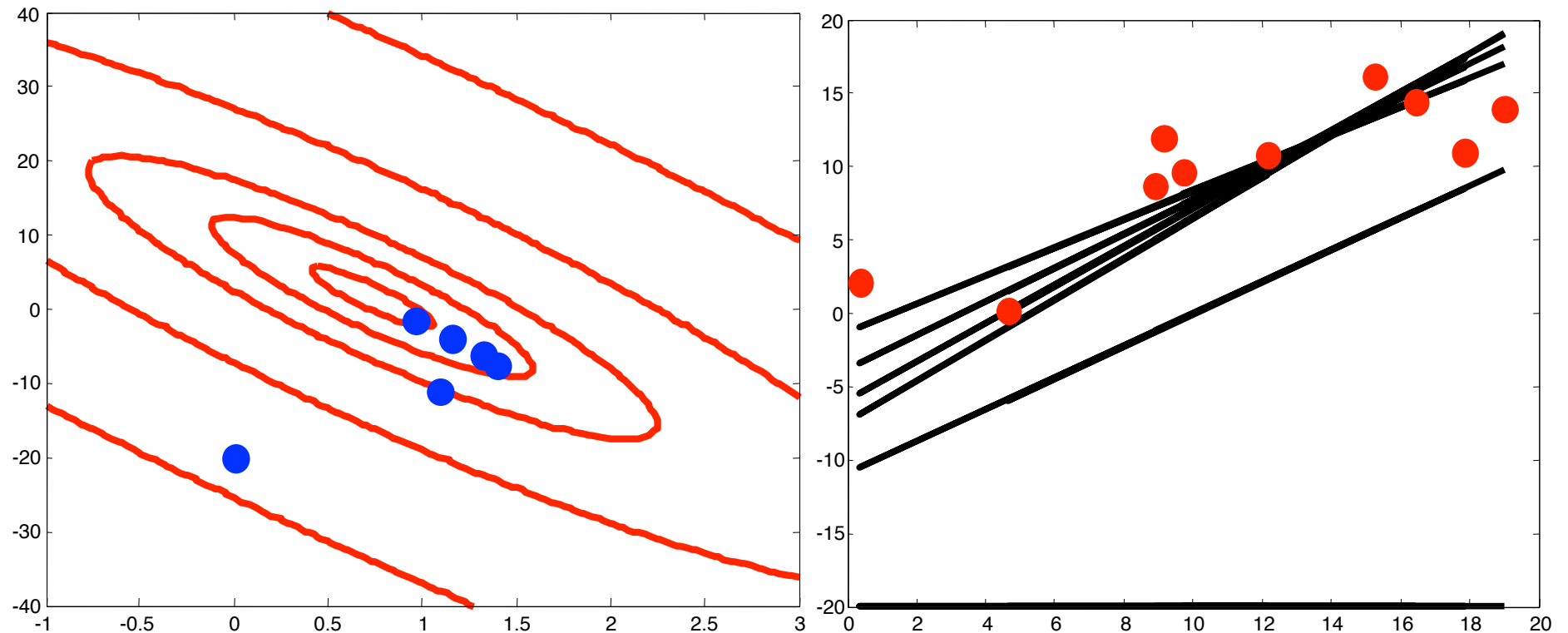
$$\underline{y} = [y^{(1)} \dots, y^{(m)}]^T$$

$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

$$\nabla J(\underline{\theta}) = -\frac{2}{m} (\underline{y}^T - \underline{\theta} \underline{X}^T) \cdot \underline{X}$$

```
e = Y - X.dot( theta.T ); # error residual
DJ = - e.dot(X) * 2.0/m # compute the gradient
theta -= alpha * DJ      # take a step
```

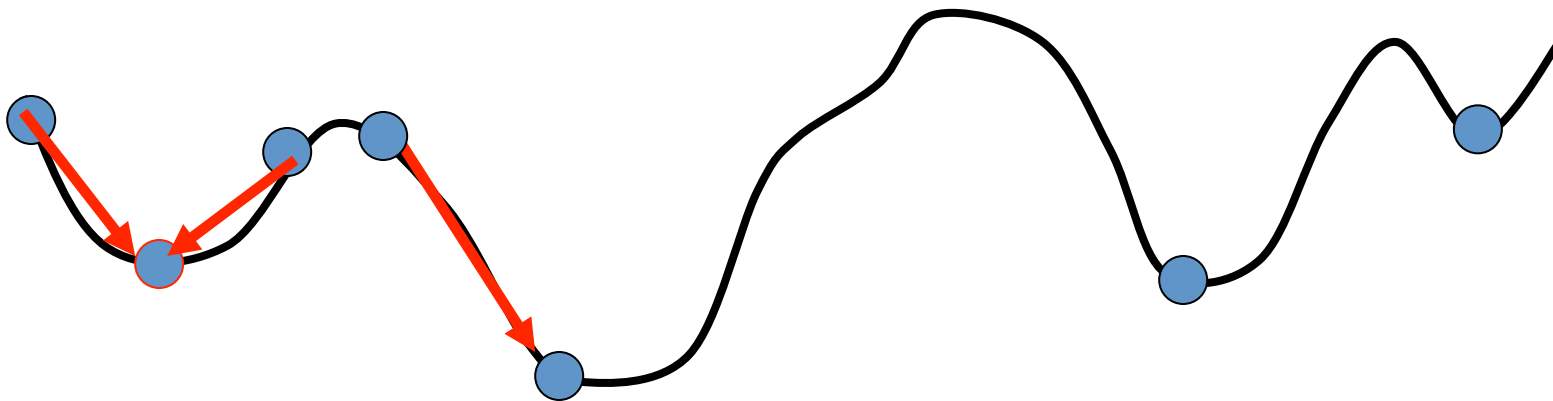
# Gradient descent on cost function



(c) Alexander Ihler

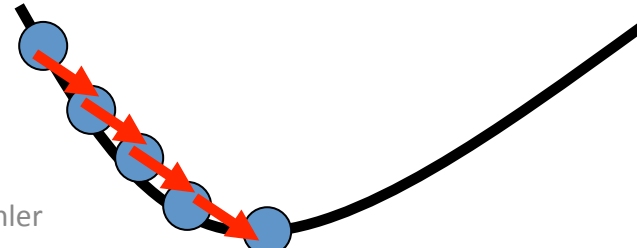
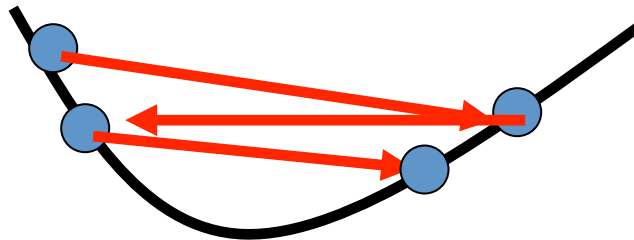
# Comments on gradient descent

- Very general algorithm
  - we'll see it many times
- Local minima
  - Sensitive to starting point



# Comments on gradient descent

- Very general algorithm
  - we'll see it many times
- Local minima
  - Sensitive to starting point
- Step size
  - Too large? Too small? Automatic ways to choose?
  - May want step size to decrease with iteration
  - Common choices:
    - Fixed
    - Linear:  $C/(\text{iteration})$
    - Line search / backoff (Armijo, etc.)
    - Newton's method



(c) Alexander Ihler

# Newton's method

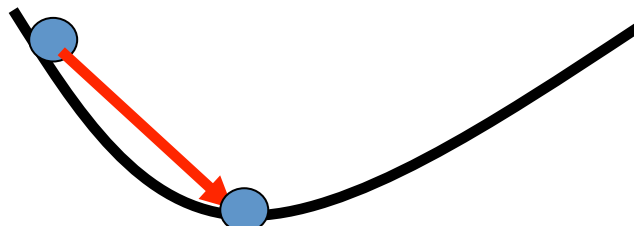
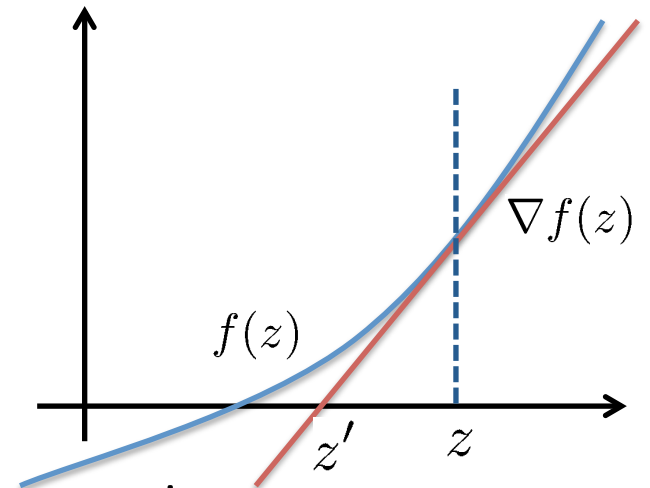
- Want to find the roots of  $f(x)$ 
  - “Root”: value of  $x$  for which  $f(x)=0$
- Initialize to *some* point  $x$
- Compute the tangent at  $x$  & compute where it crosses  $x$ -axis

$$\nabla f(z) = \frac{0 - f(z)}{z' - z} \Rightarrow z' = z - \frac{f(z)}{\nabla f(z)}$$

- Optimization: find roots of  $\nabla J(\theta)$

$$\nabla \nabla J(\theta) = \frac{0 - \nabla J(\theta)}{\theta' - \theta} \Rightarrow \theta' = \theta - \frac{\nabla J(\theta)}{\nabla \nabla J(\theta)} \quad (\text{“Step size” } \lambda = 1/\nabla \nabla J ; \text{ inverse curvature})$$

- Does not always converge; sometimes unstable
- If converges, usually very fast
- Works well for smooth, non-pathological functions, locally quadratic



(Multivariate:

$\nabla J(\theta)$  = gradient vector

$\nabla^2 J(\theta)$  = matrix of 2<sup>nd</sup> derivatives

$a/b = a b^{-1}$ , matrix inverse)

# Stochastic / Online gradient descent

- MSE

$$J(\underline{\theta}) = \frac{1}{m} \sum_j J_j(\underline{\theta}), \quad J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$$

- Gradient

$$\nabla J(\underline{\theta}) = \frac{1}{m} \sum_j \nabla J_j(\underline{\theta}) \quad \nabla J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T}) \cdot [x_0^{(j)} x_1^{(j)} \dots]$$

- Stochastic (or “online”) gradient descent:
  - Use updates based on individual datum  $j$ , chosen at random
  - At optima,  $\mathbb{E}[\nabla J_j(\underline{\theta})] = \nabla J(\underline{\theta}) = 0$   
(average over the data)



# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

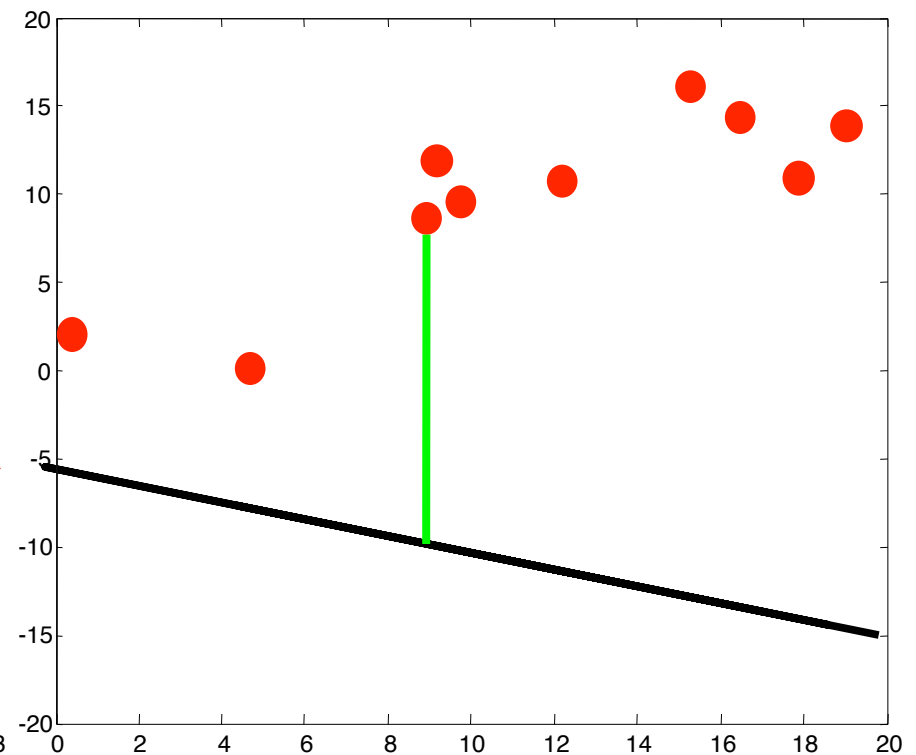
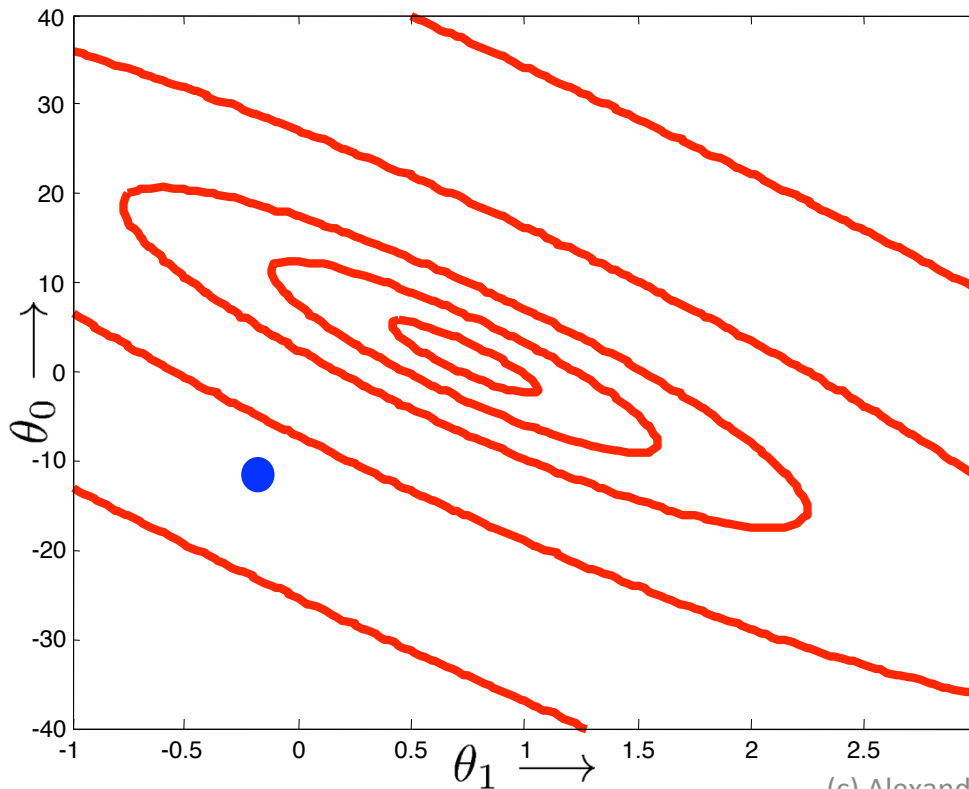
Initialize  $\theta$

Do {

  for  $j=1:m$

$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$

  } while (not done)



(c) Alexander Ihler

# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

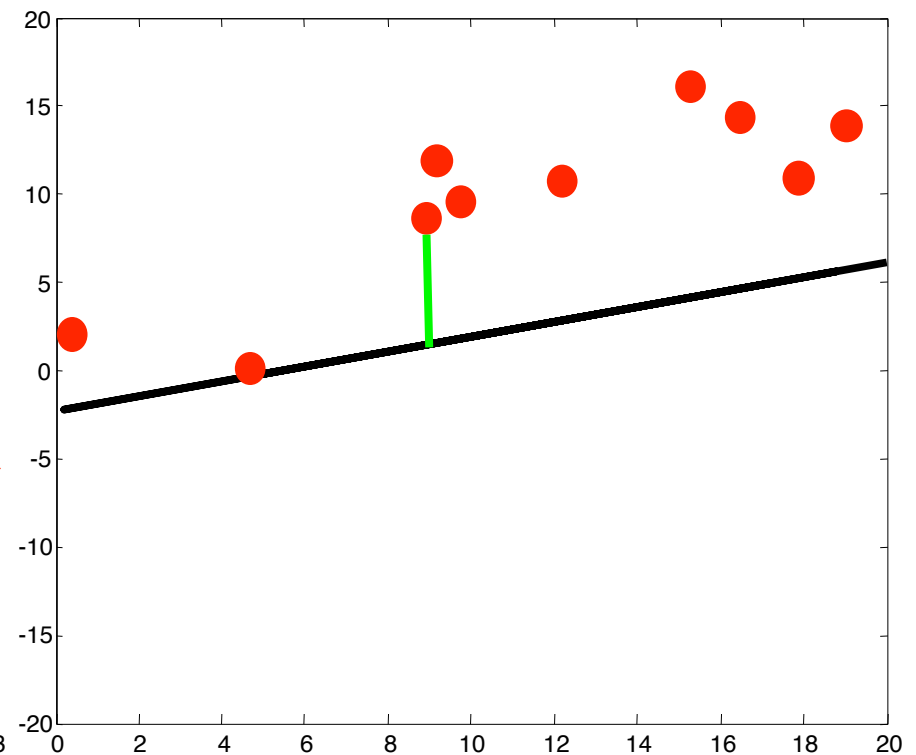
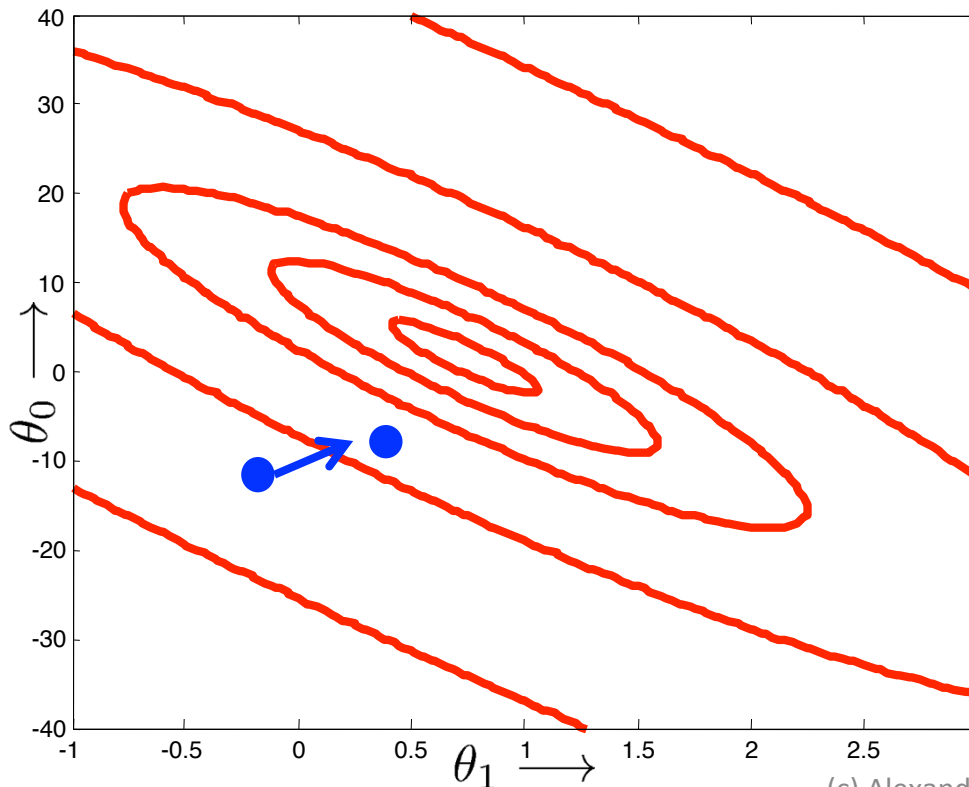
Initialize  $\theta$

Do {

  for  $j=1:m$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$$

} while (not done)



(c) Alexander Ihler

# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

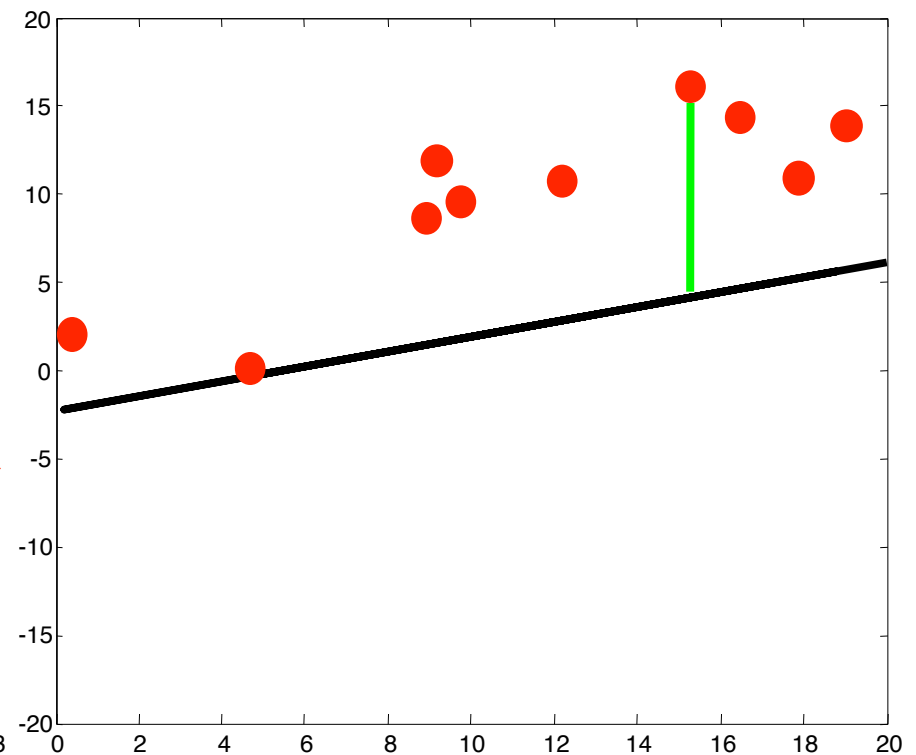
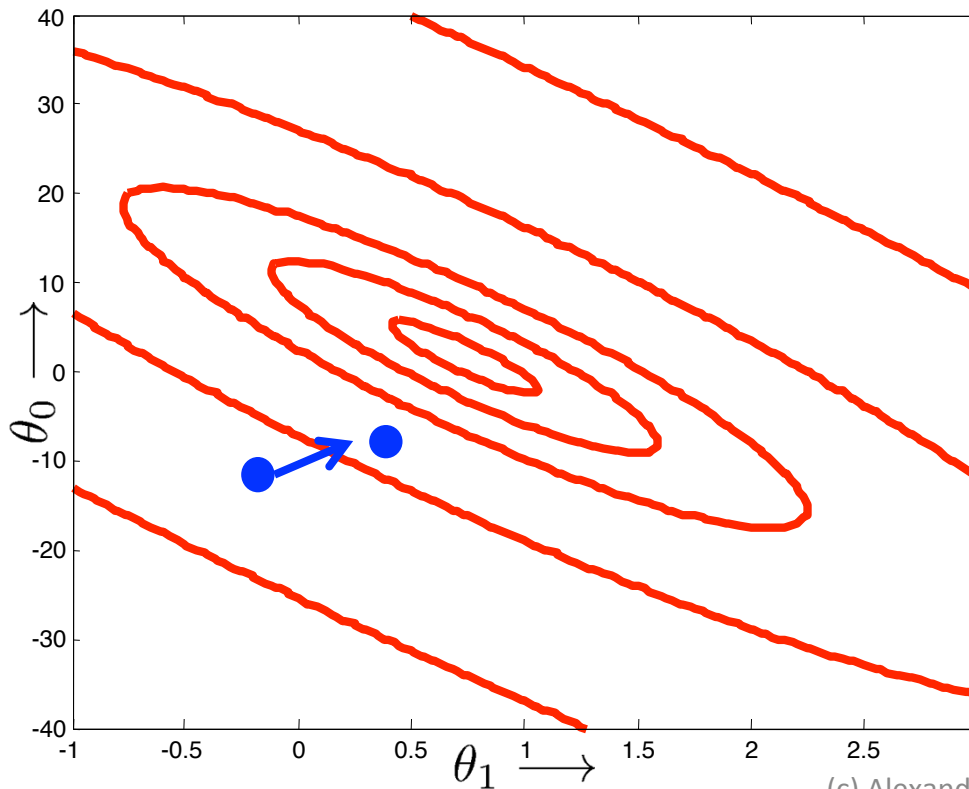
Initialize  $\theta$

Do {

  for  $j=1:m$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$$

} while (not done)



(c) Alexander Ihler

# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

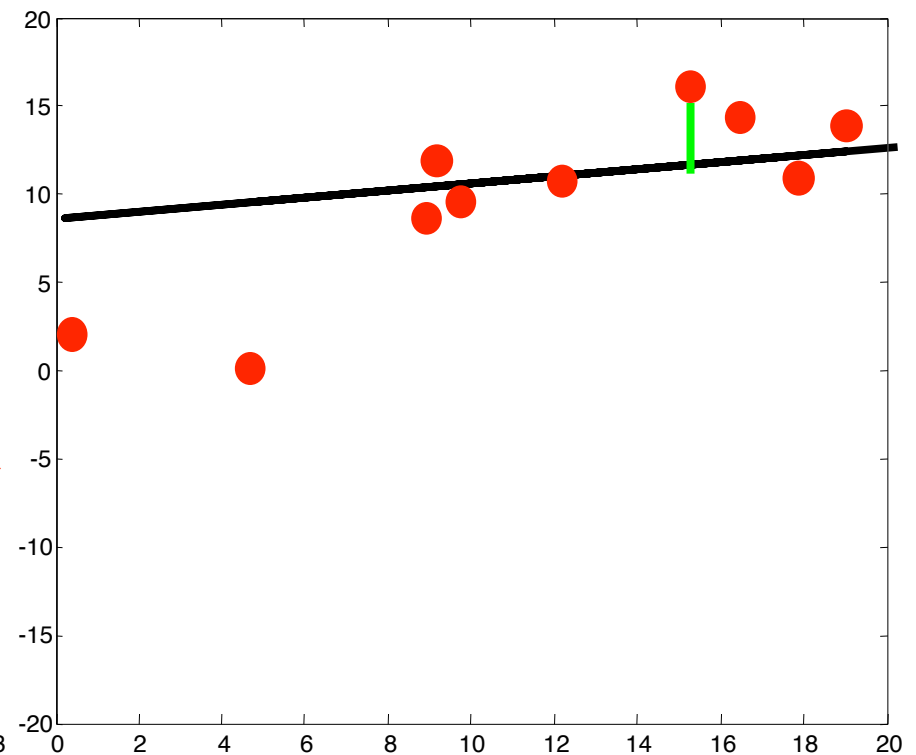
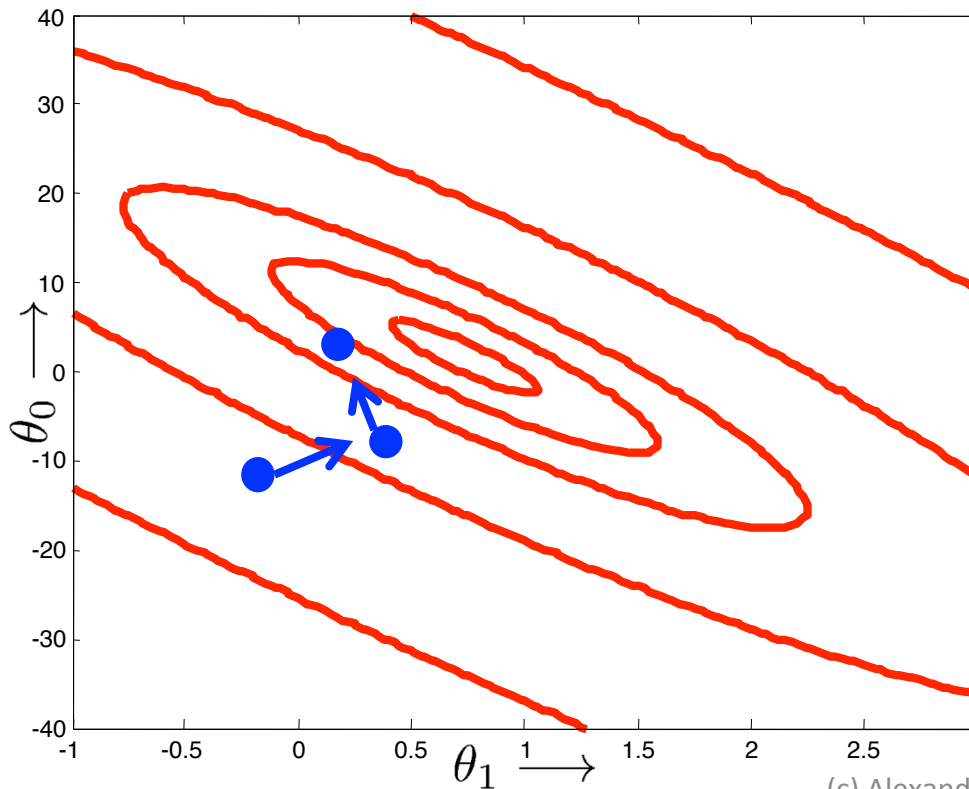
Initialize  $\theta$

Do {

  for  $j=1:m$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$$

} while (not done)



(c) Alexander Ihler

# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

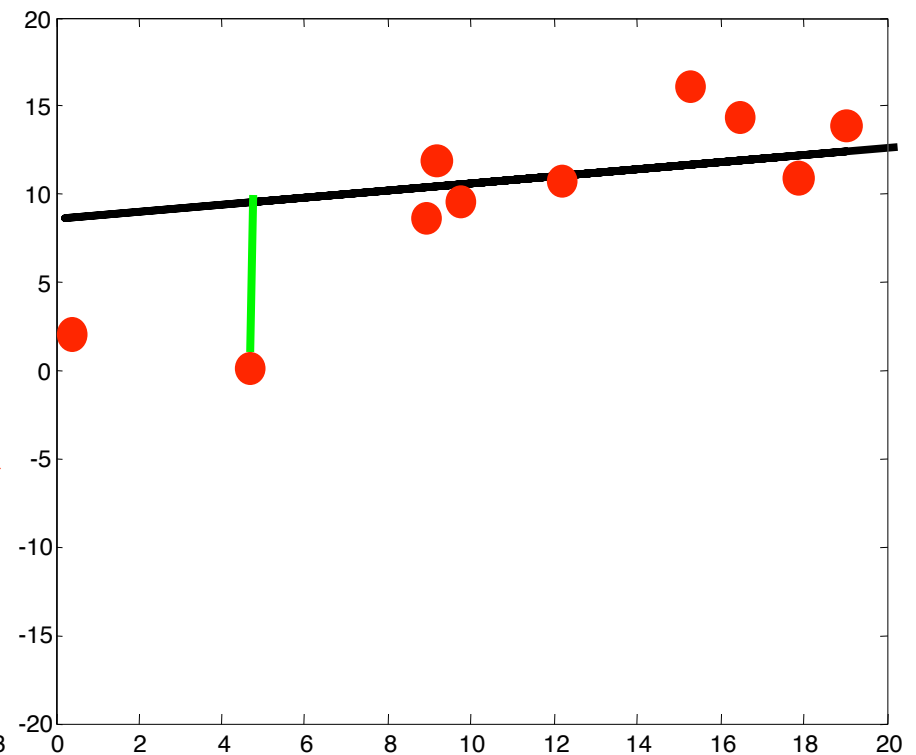
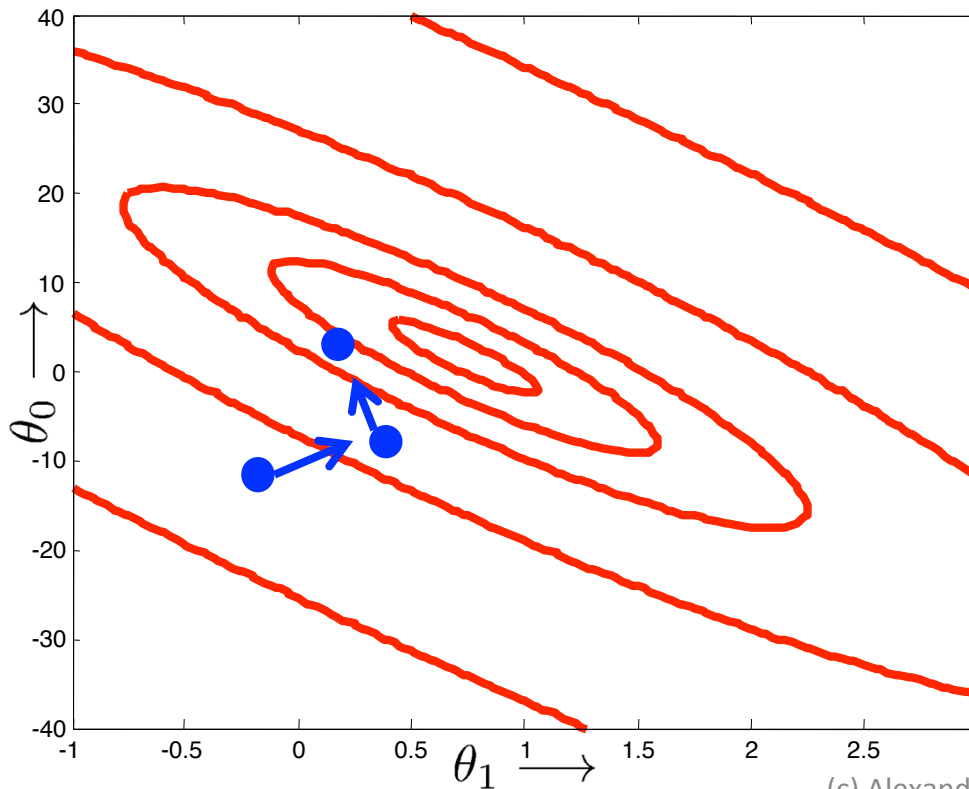
Initialize  $\theta$

Do {

  for  $j=1:m$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$$

} while (not done)



(c) Alexander Ihler

# Online gradient descent

- Update based on each datum at a time
  - Find residual and the gradient of its part of the error & update

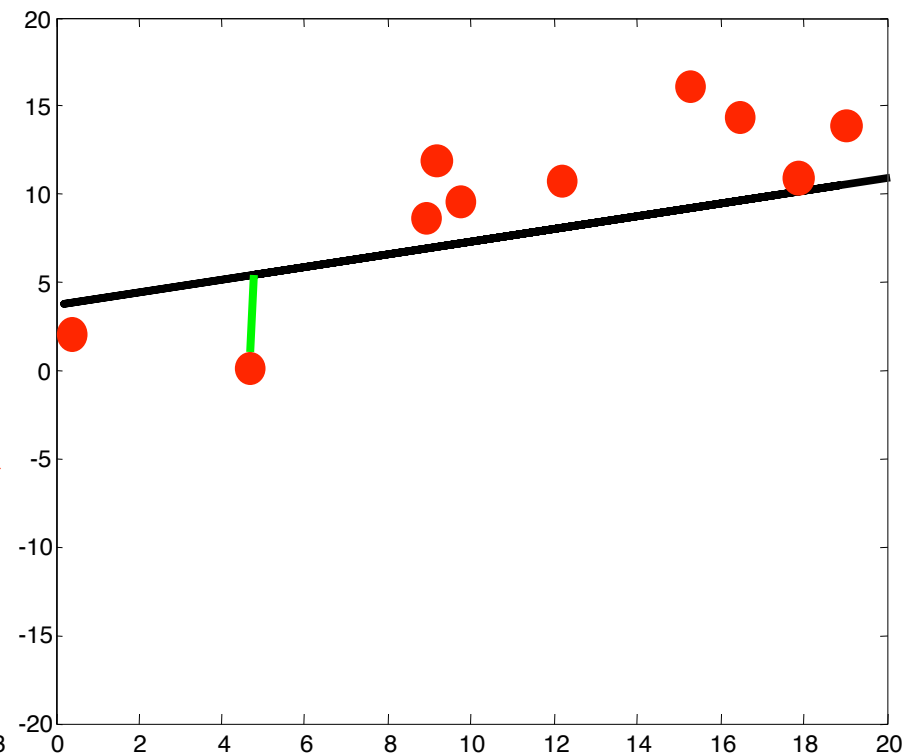
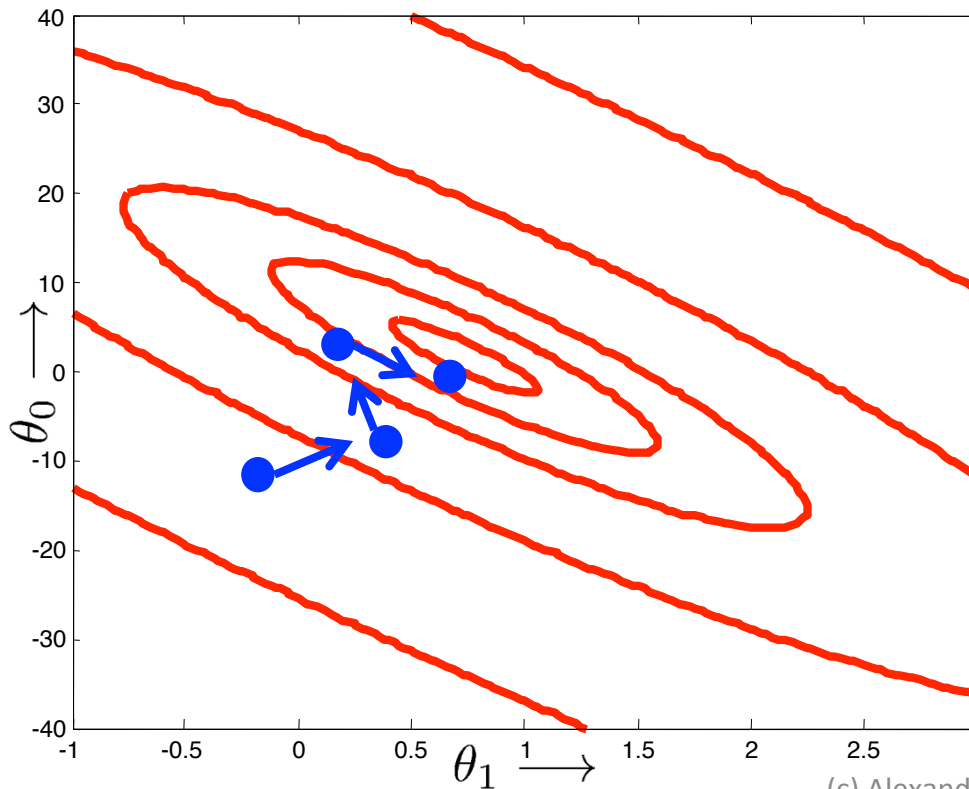
Initialize  $\theta$

Do {

  for  $j=1:m$

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$$

} while (not done)



(c) Alexander Ihler

# Online gradient descent

$$J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$$

$$\nabla J_j(\underline{\theta}) = -2(y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T}) \cdot [x_0^{(j)} x_1^{(j)} \dots]$$

Initialize  $\theta$

Do {

  for  $j=1:m$

$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_j(\theta)$

} while (not converged)

- Benefits

- Lots of data = many more updates per pass
- Computationally faster

- Drawbacks

- No longer strictly “descent”
- Stopping conditions may be harder to evaluate  
(Can use “running estimates” of  $J(\cdot)$ , etc. )

- Related: mini-batch updates, etc.

+

# Machine Learning and Data Mining

## Linear regression: direct minimization

Prof. Alexander Ihler



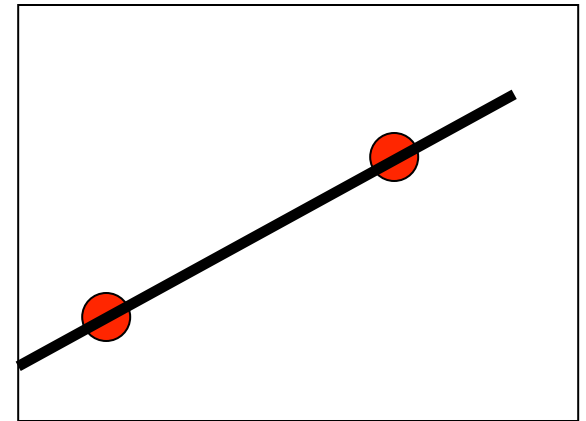


# MSE Minimum

- Consider a simple problem
  - One feature, two data points
  - Two unknowns:  $\theta_0, \theta_1$
  - Two equations:

$$y^{(1)} = \theta_0 + \theta_1 x^{(1)}$$

$$y^{(2)} = \theta_0 + \theta_1 x^{(2)}$$



- Can solve this system directly:

$$\underline{y}^T = \underline{\theta} \underline{X}^T \quad \Rightarrow \quad \hat{\underline{\theta}} = \underline{y}^T (\underline{X}^T)^{-1}$$

- However, most of the time,  $m > n$ 
  - There may be no linear function that hits all the data exactly
  - Instead, solve directly for minimum of MSE function

# MSE Minimum

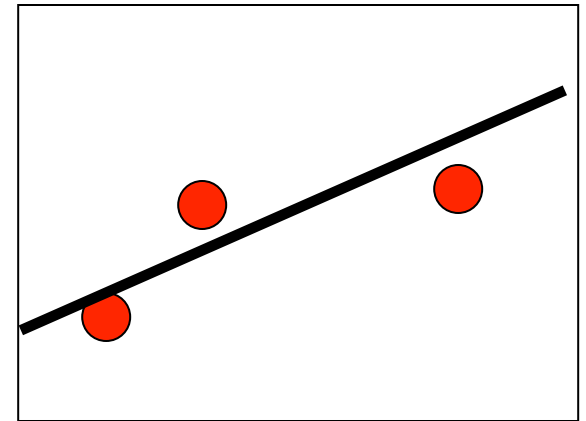
$$\nabla J(\underline{\theta}) = -\frac{2}{m}(\underline{y}^T - \underline{\theta} \underline{X}^T) \cdot \underline{X} = \underline{0}$$

- Reordering, we have

$$\underline{y}^T \underline{X} - \underline{\theta} \underline{X}^T \cdot \underline{X} = \underline{0}$$

$$\underline{y}^T \underline{X} = \underline{\theta} \underline{X}^T \cdot \underline{X}$$

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



- $\underline{X} (\underline{X}^T \underline{X})^{-1}$  is called the “pseudo-inverse”
- If  $\underline{X}^T$  is square and independent, this is the inverse
- If  $m > n$ : overdetermined; gives minimum MSE fit

# Python MSE

- This is easy to solve in Python / NumPy...

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

```
# y = np.matrix( [[y1], ... , [ym]] )
# X = np.matrix( [[x1_0 ... x1_n], [x2_0 ... x2_n], ...] )

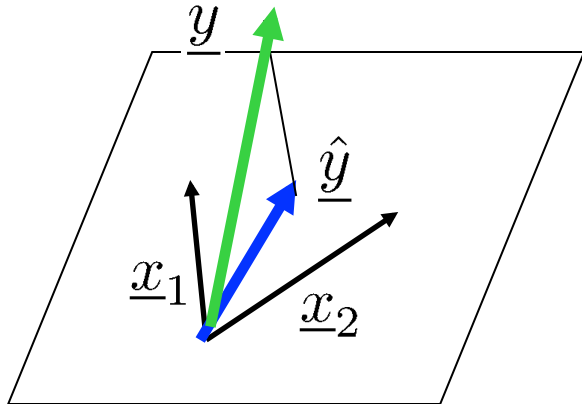
% Solution 1: "manual"
th = y.T * X * np.linalg.inv(X.T * X);

% Solution 2: "least squares solve"
th = np.linalg.lstsq(X, Y);
```

# Normal equations

$$\nabla J(\underline{\theta}) = 0 \quad \Rightarrow \quad (\underline{y}^T - \underline{\theta} \underline{X}^T) \cdot \underline{X} = \underline{0}$$

- Interpretation:
  - $(y - \theta X) = (y - \hat{y})$  is the vector of errors in each example
  - $X$  are the features we have to work with for each example
  - Dot product = 0: orthogonal

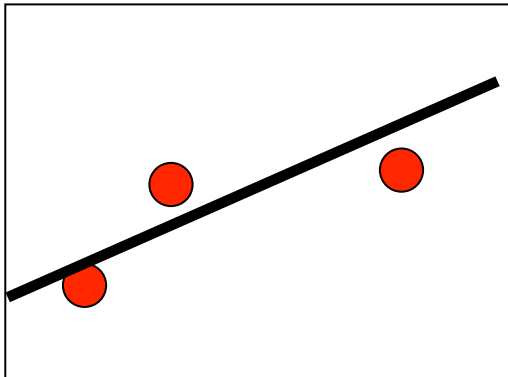


$$\underline{y}^T = [y^{(1)} \dots y^{(m)}]$$
$$\underline{x}_i = [x_i^{(1)} \dots x_i^{(m)}]$$

# Normal equations

$$\nabla J(\underline{\theta}) = 0 \quad \Rightarrow \quad (\underline{y}^T - \underline{\theta} \underline{X}^T) \cdot \underline{X} = 0$$

- Interpretation:
  - $(y - \theta X) = (y - \hat{y})$  is the vector of errors in each example
  - $X$  are the features we have to work with for each example
  - Dot product = 0: orthogonal
- Example:

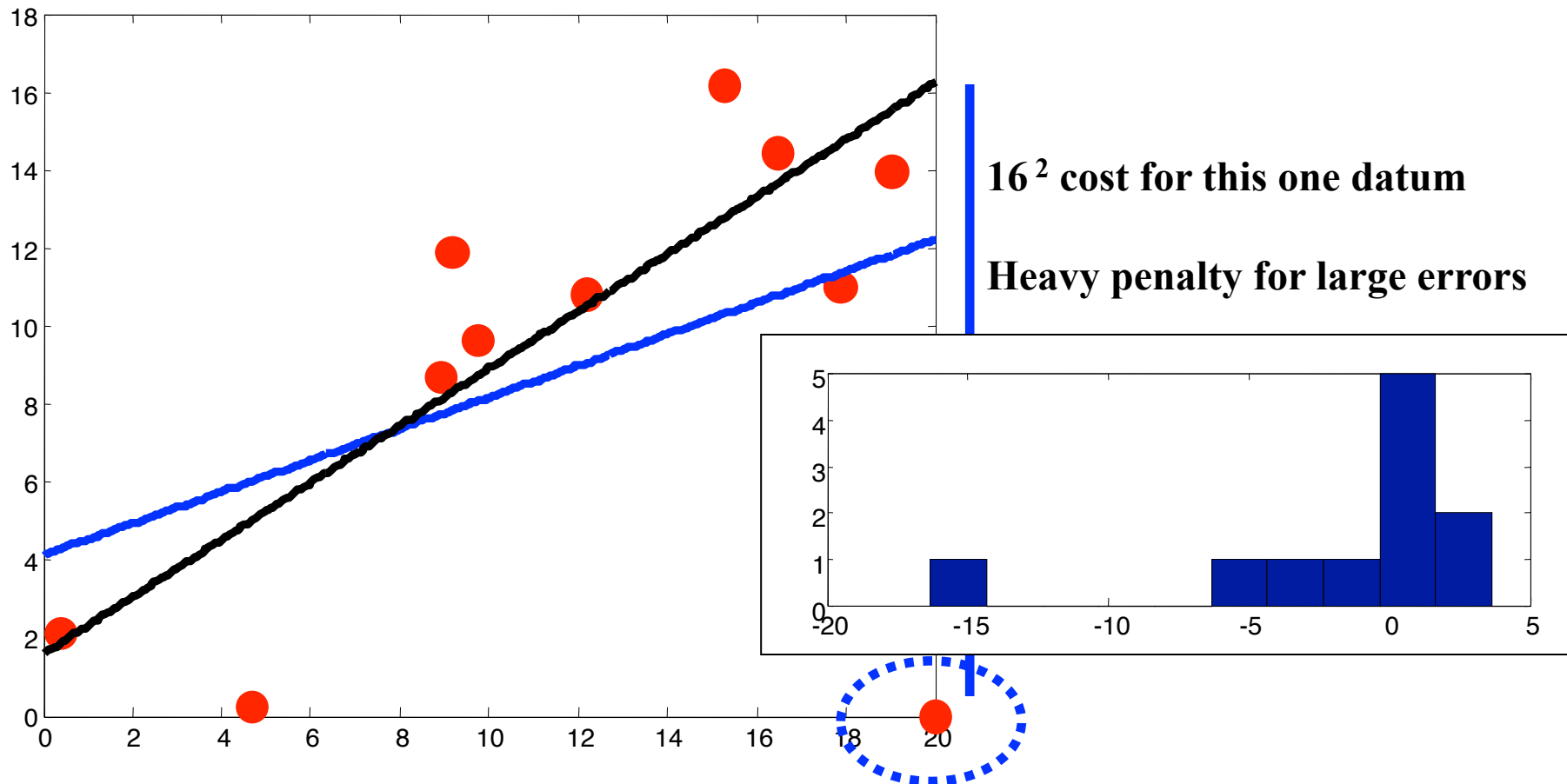


$$\begin{aligned} \underline{y} &= [1 \quad 3 \quad 3]^T \\ \underline{x}_0 &= [1 \quad 1 \quad 1]^T \\ \underline{x}_1 &= [1 \quad 2 \quad 4]^T \end{aligned} \quad \theta = [1.00 \quad 0.57]$$

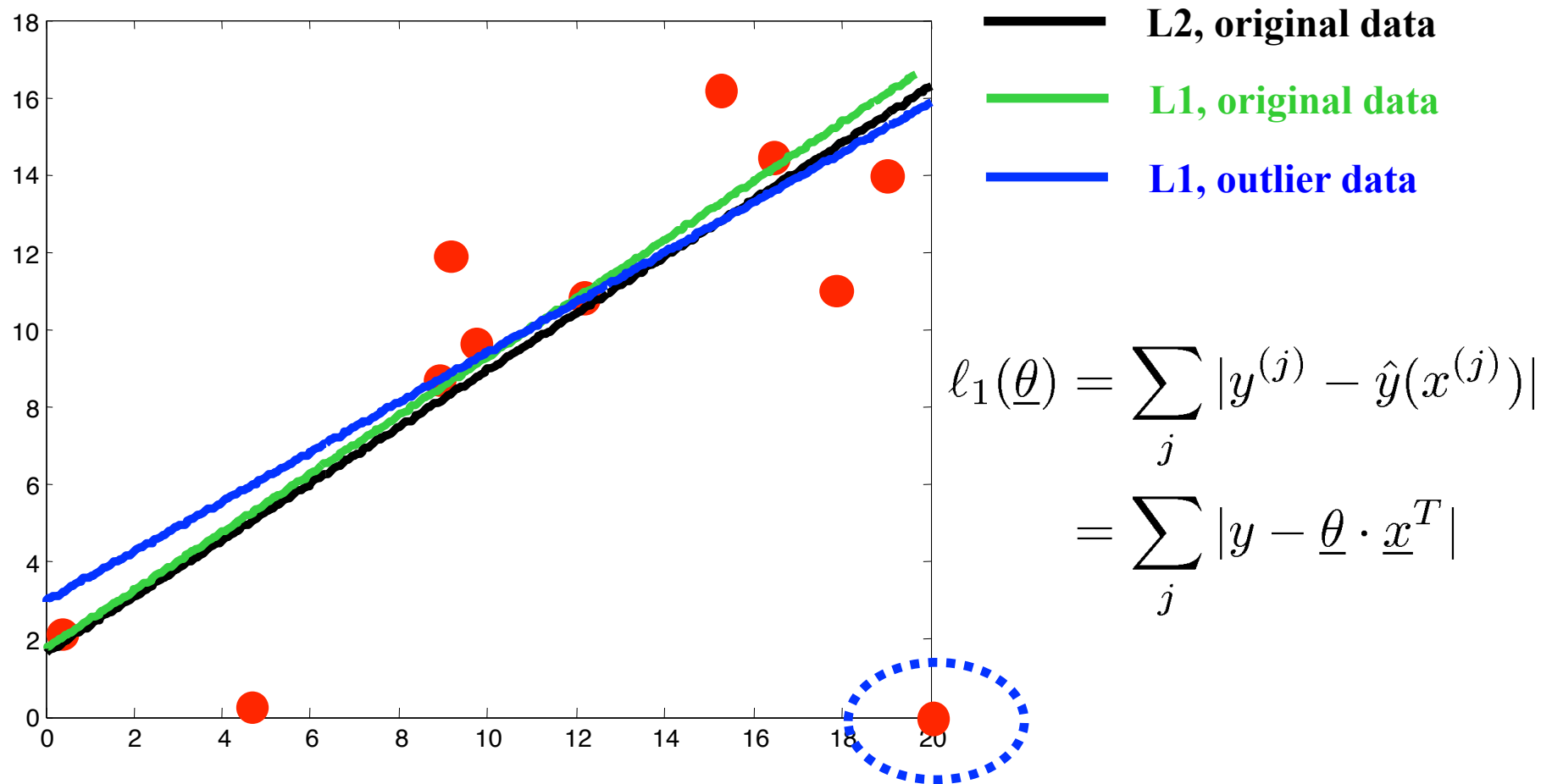
$$\underline{e} = (y - \hat{y}) = [-0.57 \quad 0.85 \quad -0.28]^T$$

# Effects of MSE choice

- Sensitivity to outliers



# L1 error



# Cost functions for regression

$$\ell_2 : (y - \hat{y})^2 \quad \text{(MSE)}$$

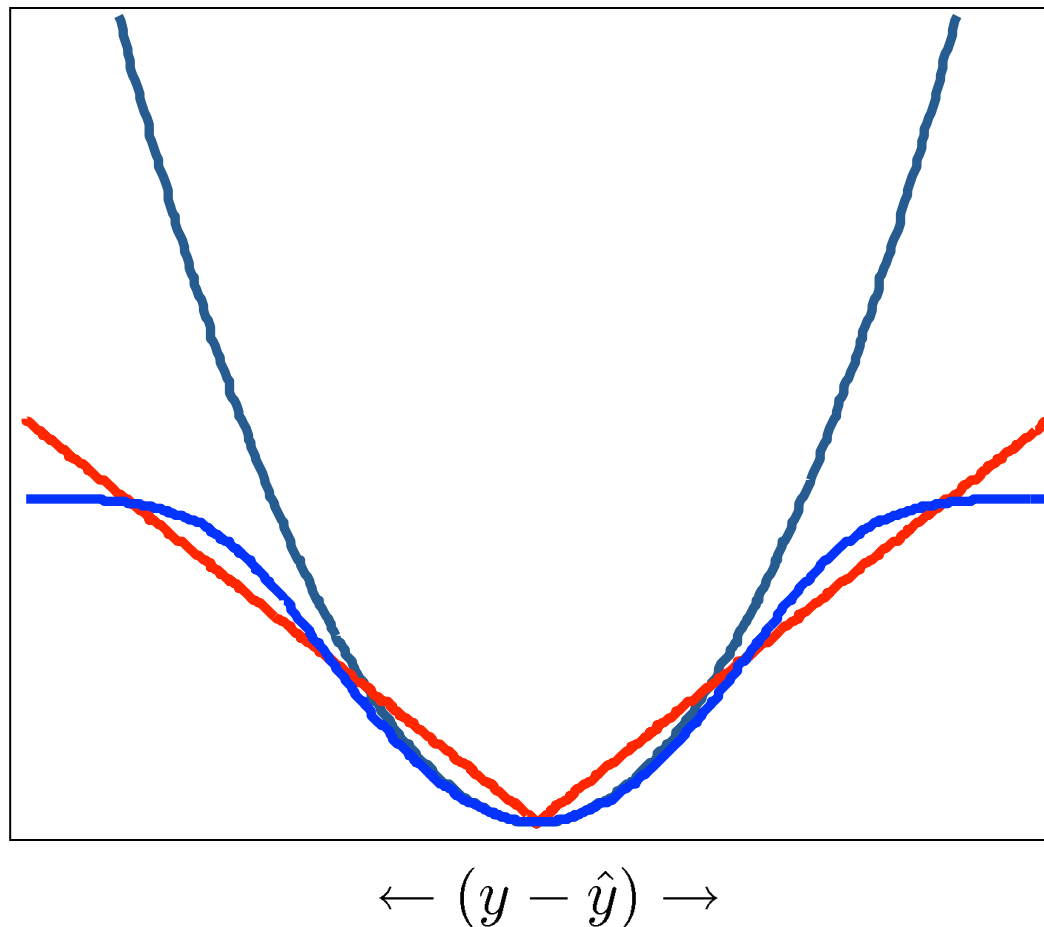
$$\ell_1 : |y - \hat{y}| \quad \text{(MAE)}$$

Something else entirely...

$$c - \log(\exp(-(y - \hat{y})^2) + c) \quad \text{(???)}$$

**“Arbitrary” functions can’t be solved in closed form...**

**- use gradient descent**





+

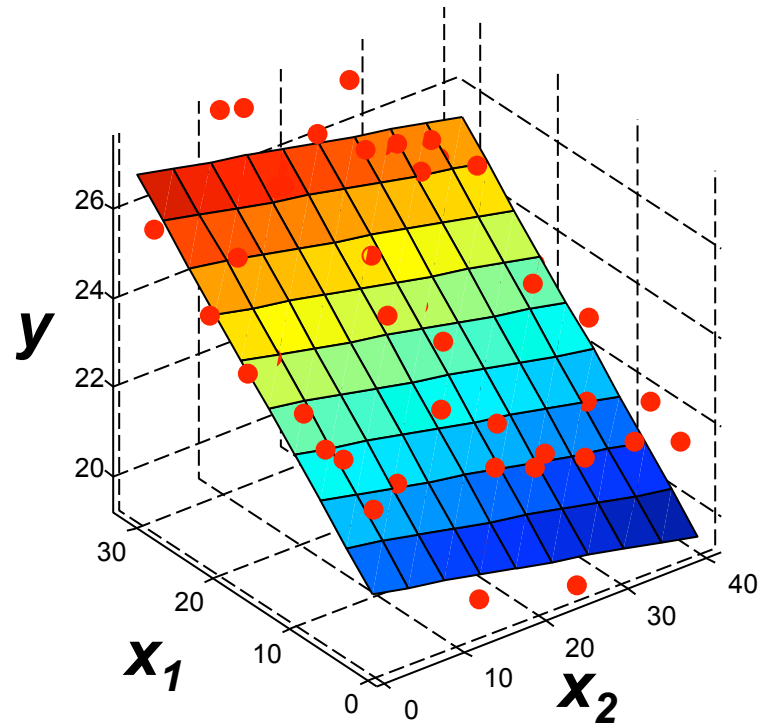
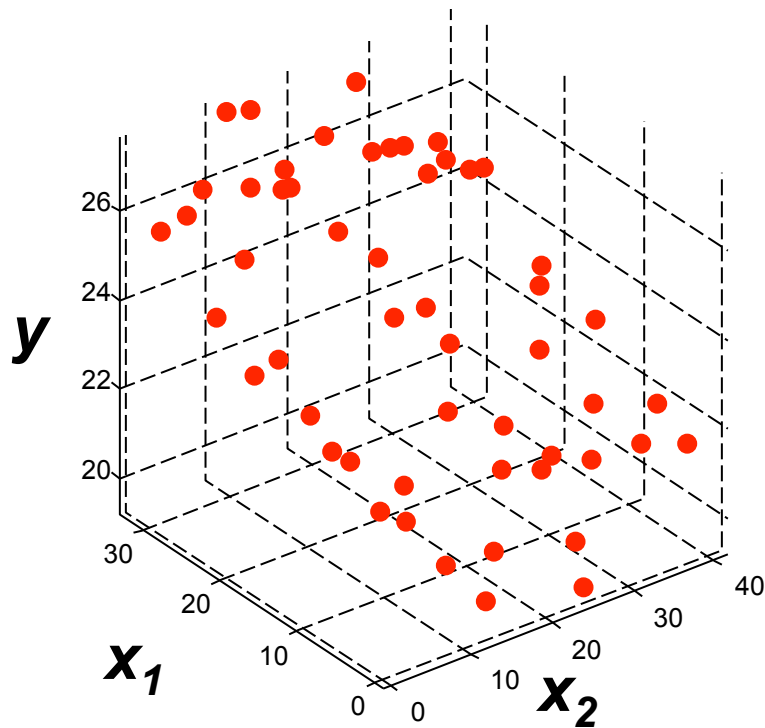
# Machine Learning and Data Mining

## Linear regression: nonlinear features

Prof. Alexander Ihler



# More dimensions?

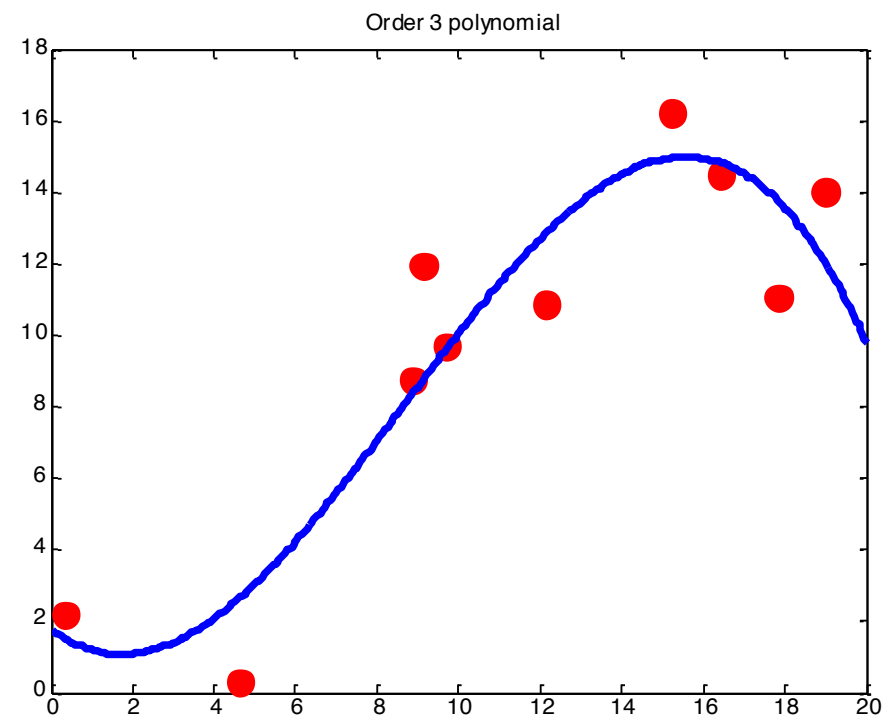
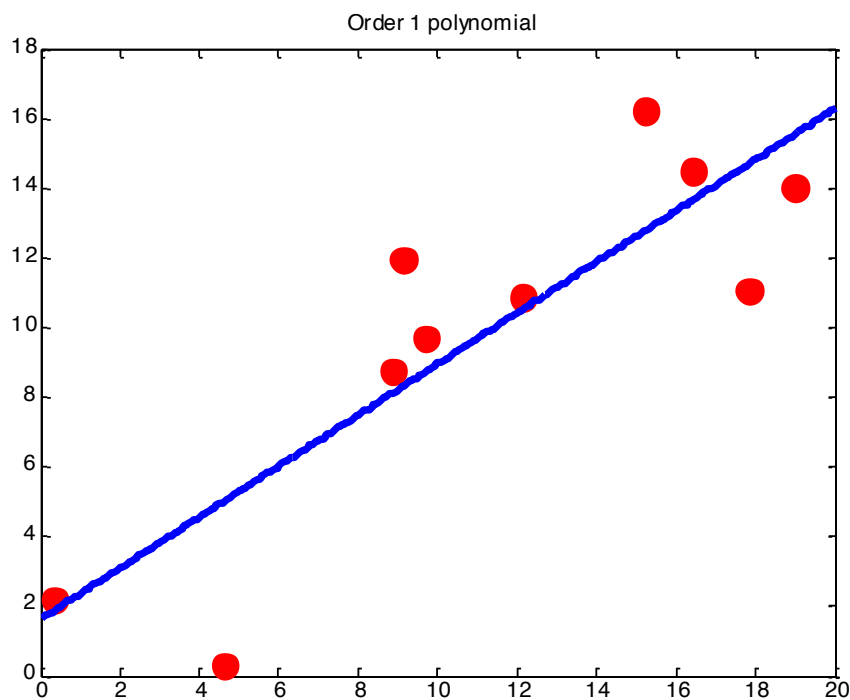


$$\hat{y}(x) = \underline{\theta} \cdot \underline{x}^T$$

$$\underline{\theta} = [\theta_0 \ \theta_1 \ \theta_2]$$
$$\underline{x} = [1 \ x_1 \ x_2]$$

# Nonlinear functions

- What if our hypotheses are not lines?
  - Ex: higher-order polynomials



# Nonlinear functions

- Single feature  $x$ , predict target  $y$ :

$$D = \{(x^{(j)}, y^{(j)})\}$$

$$\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



Add features:

$$D = \{([x^{(j)}, (x^{(j)})^2, (x^{(j)})^3], y^{(j)})\}$$



$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Linear regression in new features

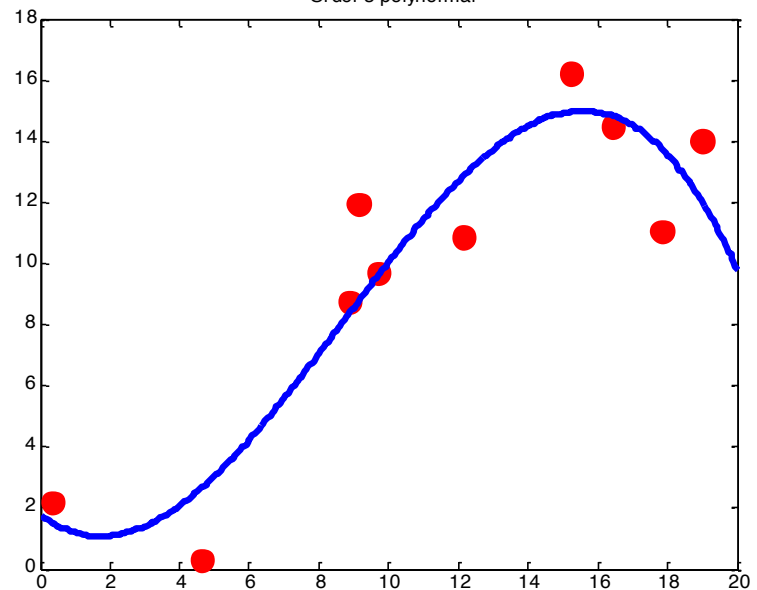
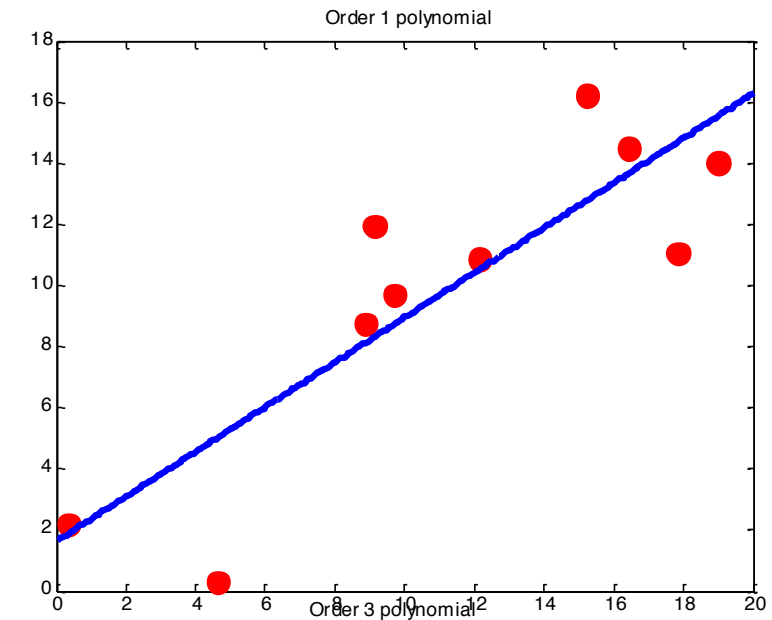
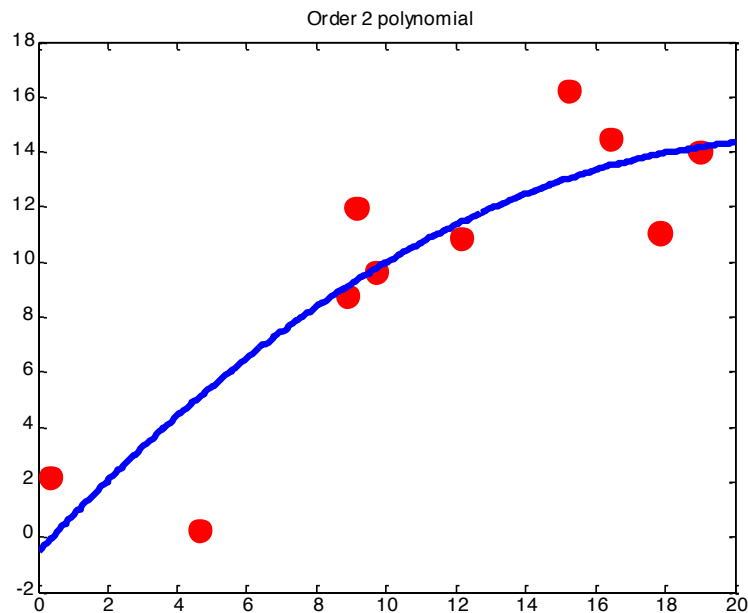
- Sometimes useful to think of “feature transform”

$$\Phi(x) = [1, x, x^2, x^3, \dots]$$

$$\hat{y}(x) = \underline{\theta} \cdot \Phi(x)$$

# Higher-order polynomials

- Fit in the same way
- More “features”

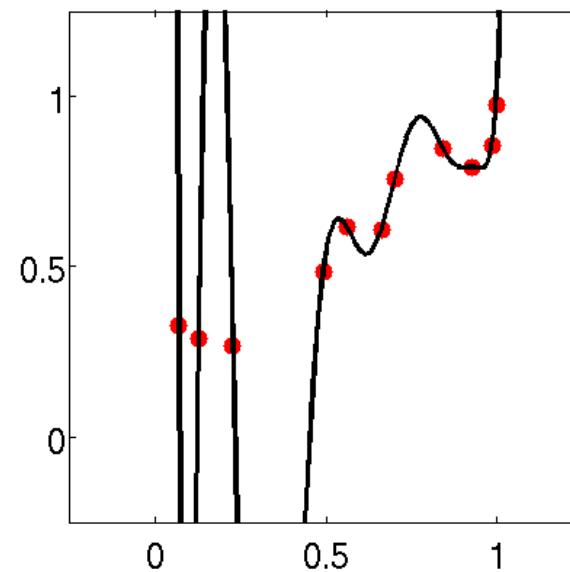
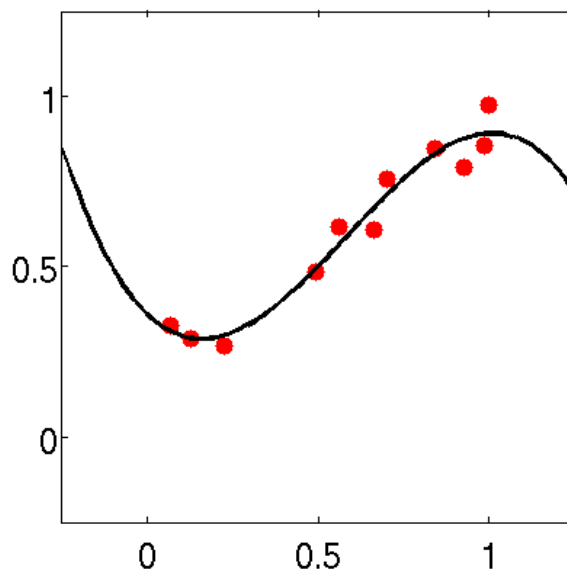
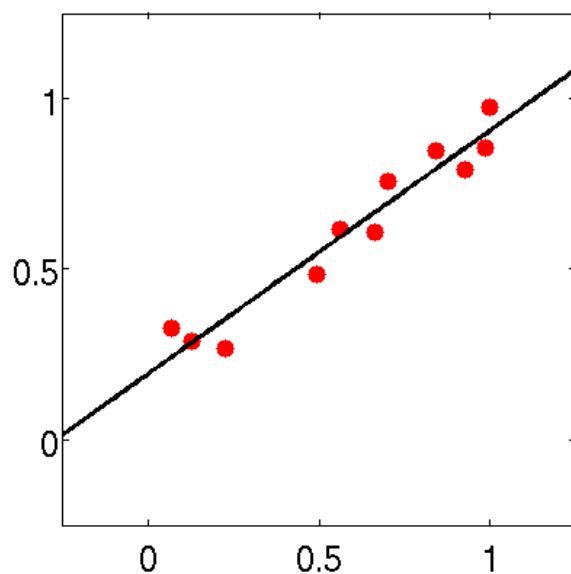
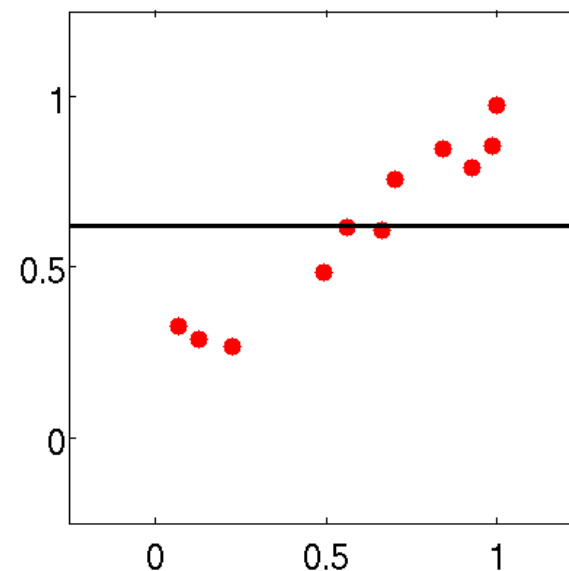


# Features

- In general, can use any features we think are useful
- Other information about the problem
  - Sq. footage, location, age, ...
- Polynomial functions
  - Features  $[1, x, x^2, x^3, \dots]$
- Other functions
  - $1/x$ ,  $\text{sqrt}(x)$ ,  $x_1 * x_2$ , ...
- “Linear regression” = linear in the parameters
  - Features we can make as complex as we want!

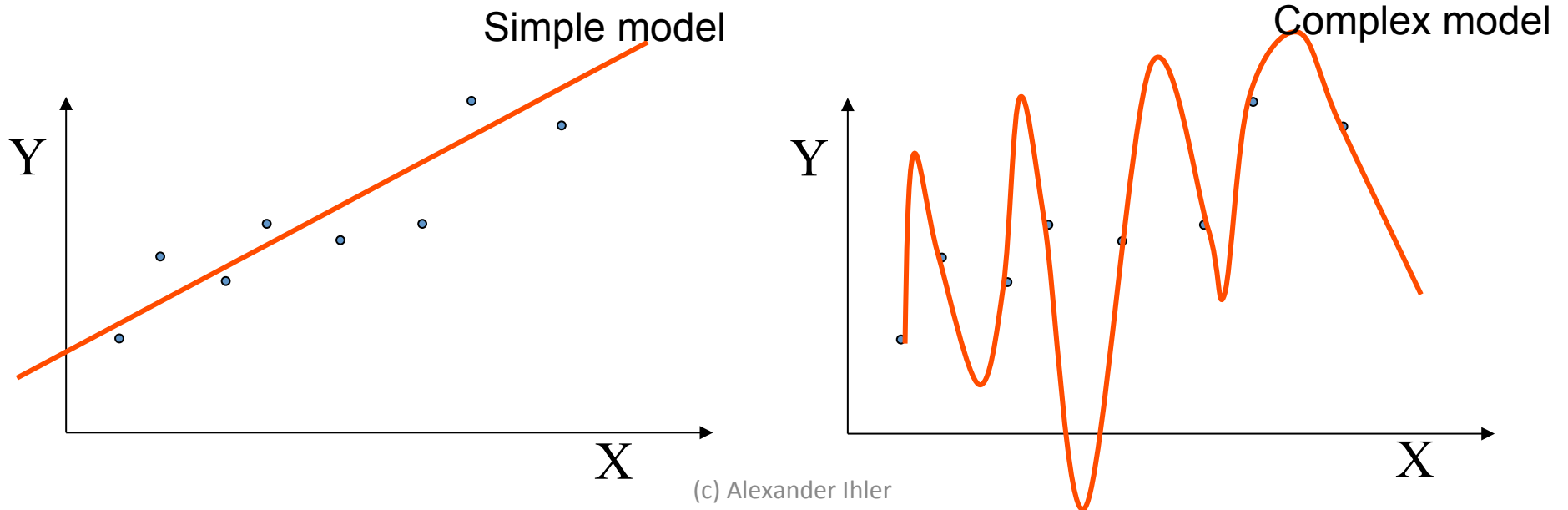
# Higher-order polynomials

- Are more features better?
- “Nested” hypotheses
  - 2<sup>nd</sup> order more general than 1<sup>st</sup>,
  - 3<sup>rd</sup> order “ “ than 2<sup>nd</sup>, ...
- Fits the observed data better



# Overfitting and complexity

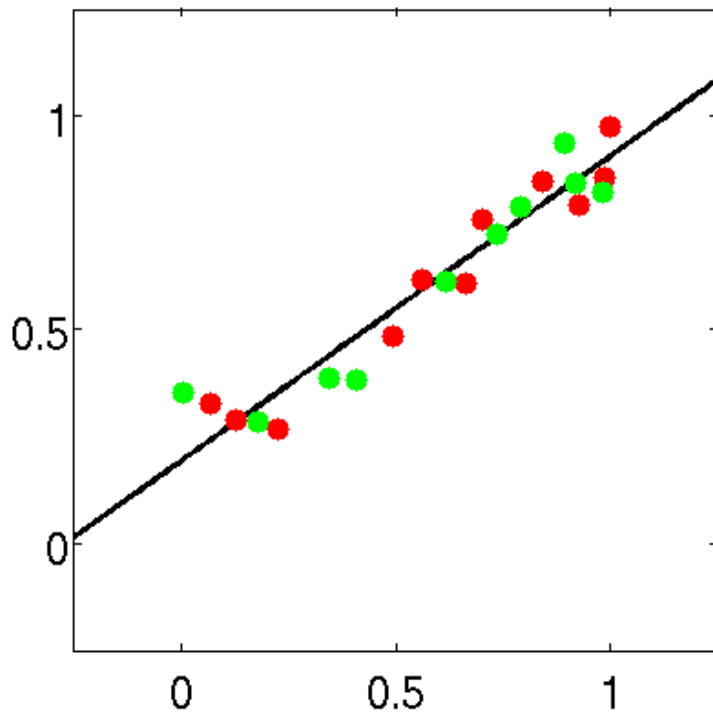
- More complex models will always fit the training data better
- But they may “overfit” the training data, learning complex relationships that are not really present



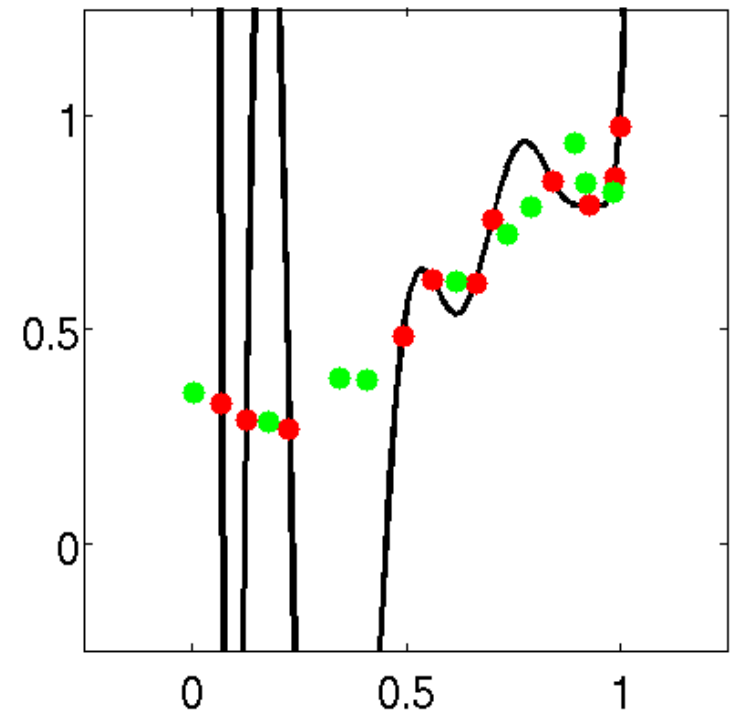


# Test data

- After training the model
- Go out and get more data from the world
  - New observations (x,y)
- How well does our model perform?

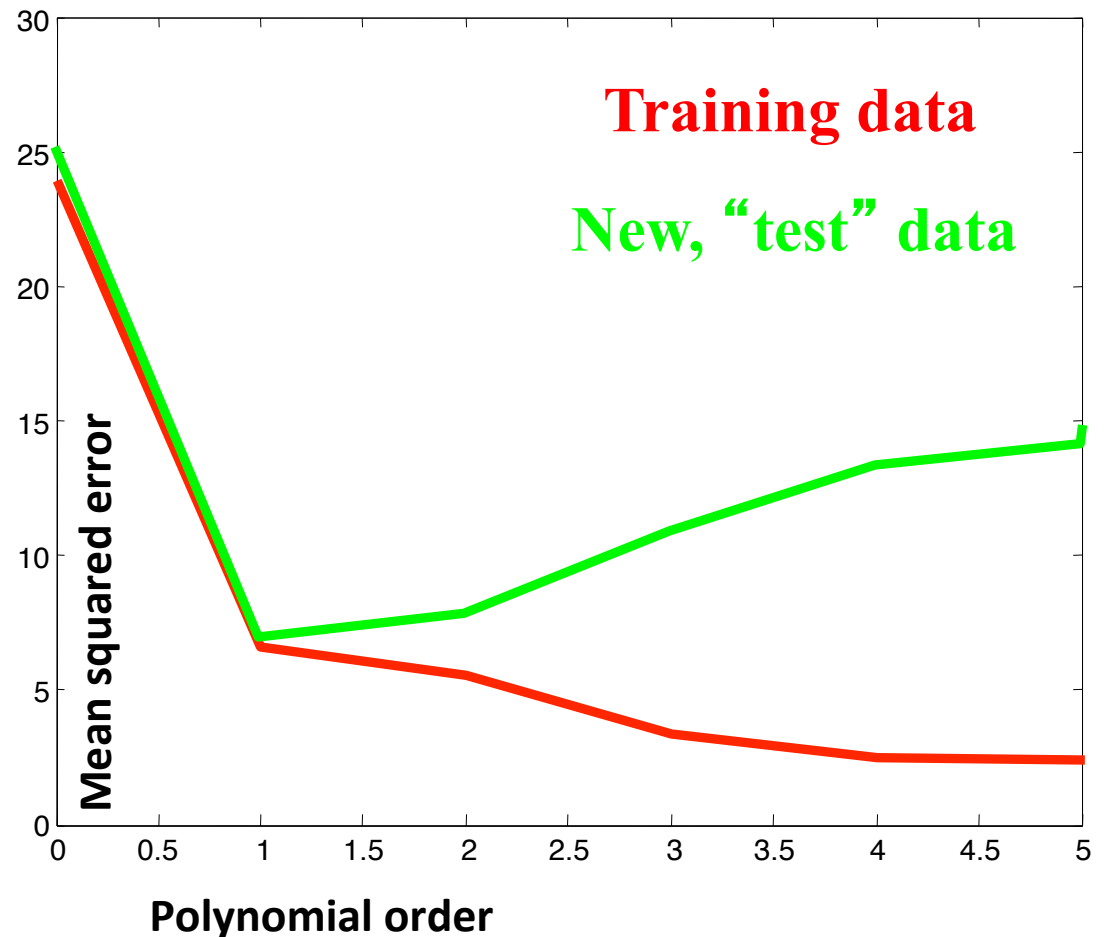


(c) Alexander Ihler



# Training versus test error

- Plot MSE as a function of model complexity
  - Polynomial order
- Decreases
  - More complex function fits training data better
- What about new data?
  - 0<sup>th</sup> to 1<sup>st</sup> order
    - Error decreases
    - Underfitting
  - Higher order
    - Error increases
    - Overfitting



+

# Machine Learning and Data Mining

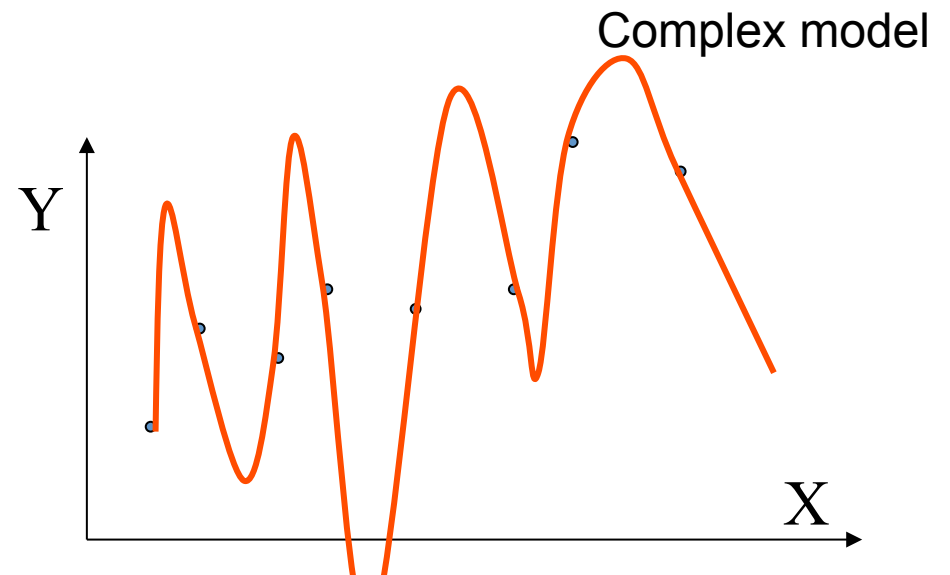
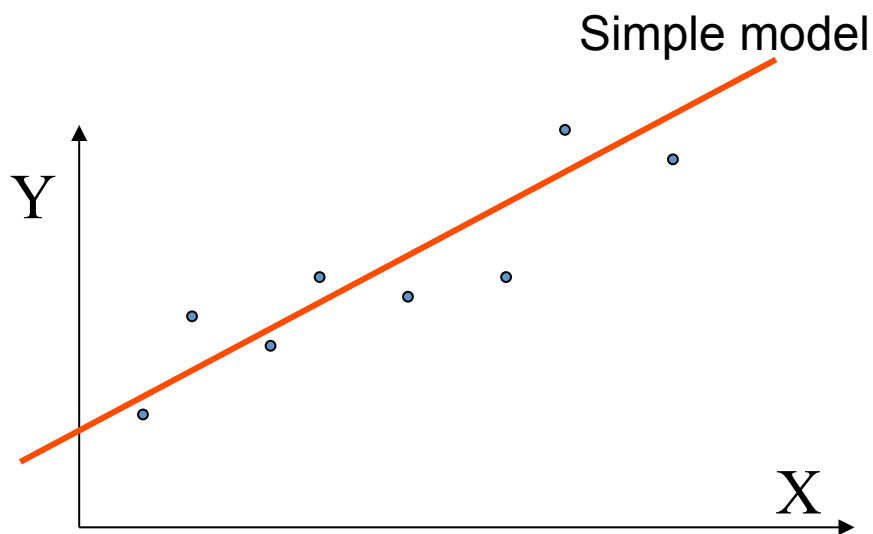
## Linear regression: bias and variance

Prof. Alexander Ihler

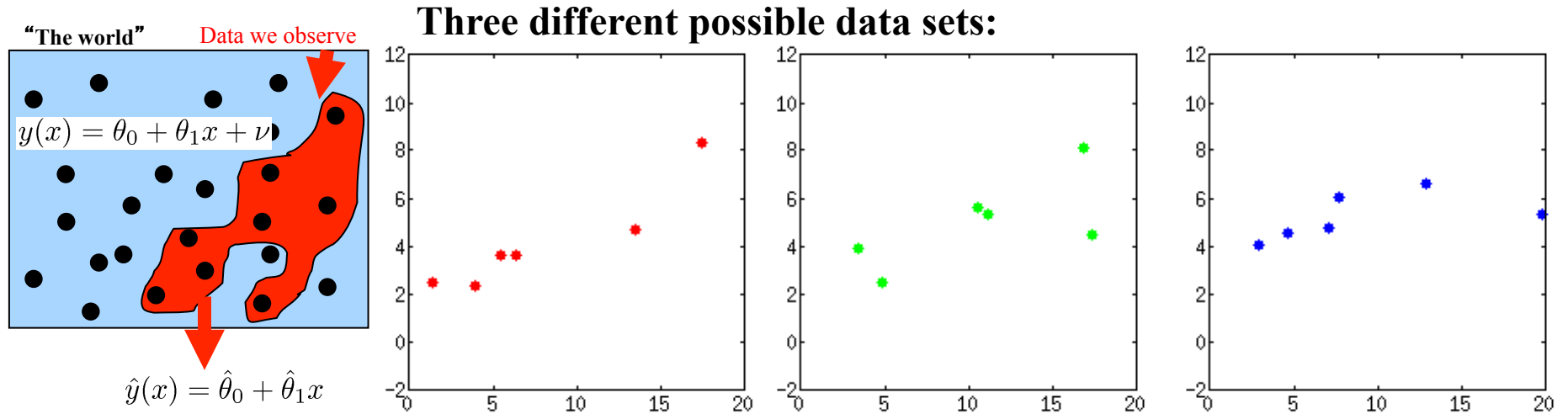


# Inductive bias

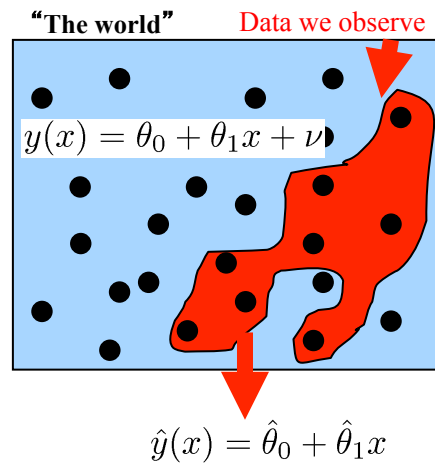
- The assumptions needed to predict examples we haven't seen
- Makes us “prefer” one model over another
- Polynomial functions; smooth functions; etc
- Some bias is necessary for learning!



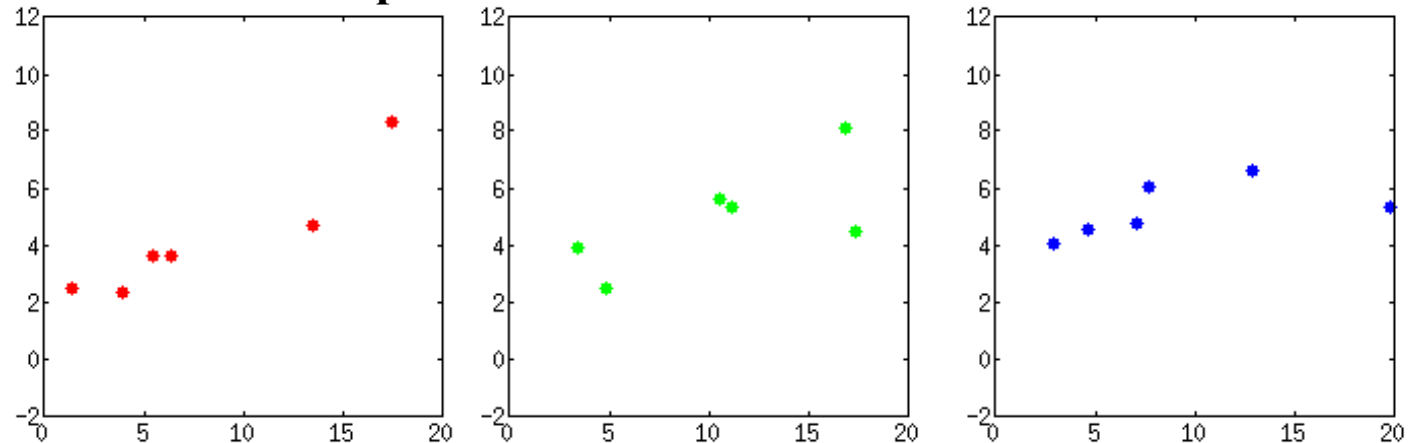
# Bias & variance



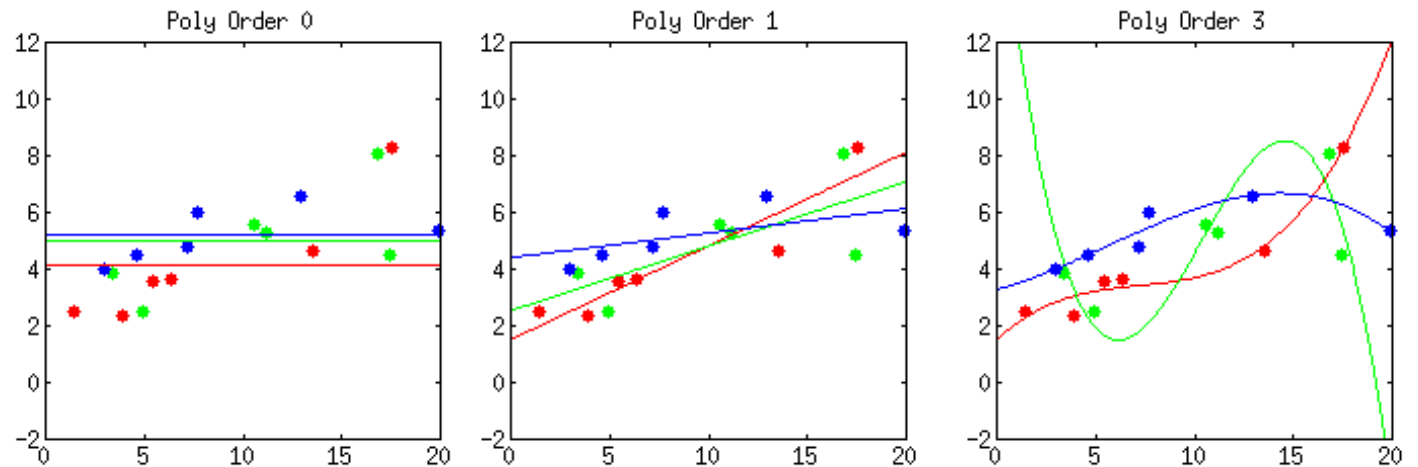
# Bias & variance



Three different possible data sets:



Each would give different predictors for any polynomial degree:

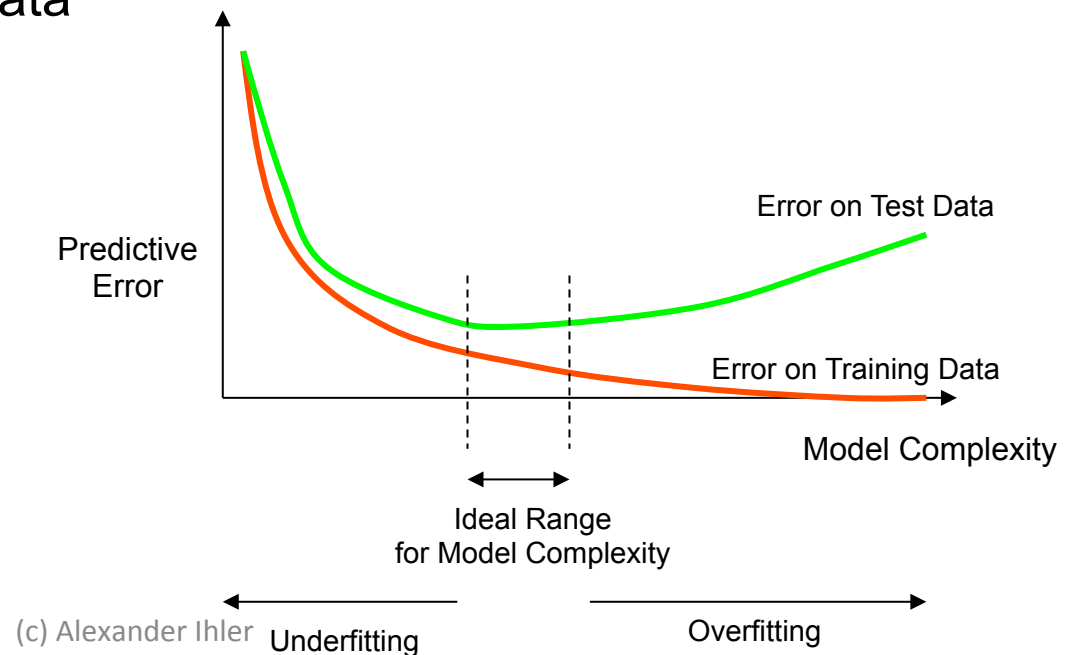


# Detecting overfitting

- Overfitting effect
  - Do better on training data than on future data
  - Need to choose the “right” complexity
- One solution: “Hold-out” data
- Separate our data into two sets
  - Training
  - Test
- Learn only on training data
- Use test data to estimate generalization quality
  - Model selection
- All good competitions use this formulation
  - Often multiple splits: one by judges, then another by you

# What to do about under/overfitting?

- Ways to increase complexity?
  - Add features, parameters
  - We'll see more...
- Ways to decrease complexity?
  - Remove features (“feature selection”)
  - “Fail to fully memorize data”
    - Partial training
    - Regularization





+

# Machine Learning and Data Mining

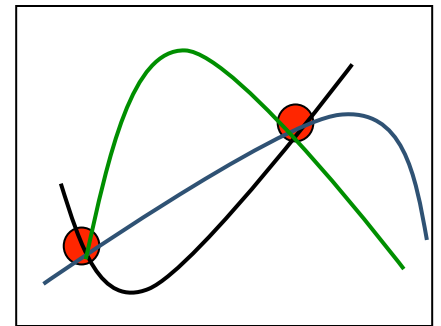
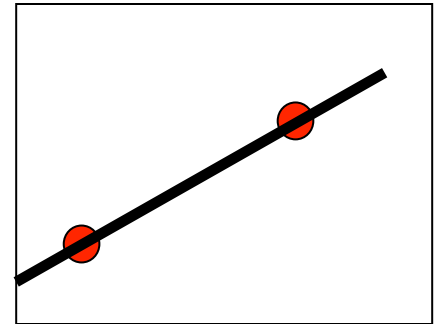
## Linear regression: regularization

Prof. Alexander Ihler



# Linear regression

- Linear model, two data
- Quadratic model, two data?
  - Infinitely many settings with zero error
  - How to choose among them?
- Higher order coefficients = 0?
  - Uses knowledge of where features came from...
- Could choose e.g. minimum magnitude:
$$\min \underline{\theta} \underline{\theta}^T \quad s.t. \quad J(\underline{\theta}) = 0$$
- A type of *bias*: tells us which models to prefer



# Regularization

- Can modify our cost function  $J$  to add “preference” for certain parameter values

$$J(\underline{\theta}) = \frac{1}{2}(\underline{y} - \underline{\theta} \underline{X}^T) \cdot (\underline{y} - \underline{\theta} \underline{X}^T)^T + \alpha \underline{\theta} \underline{\theta}^T$$

$L_2$  penalty:  
“Ridge regression”

- New solution (derive the same way)

$$\underline{\theta} = \underline{y} \underline{X} (\underline{X}^T \underline{X} + \alpha \underline{I})^{-1}$$

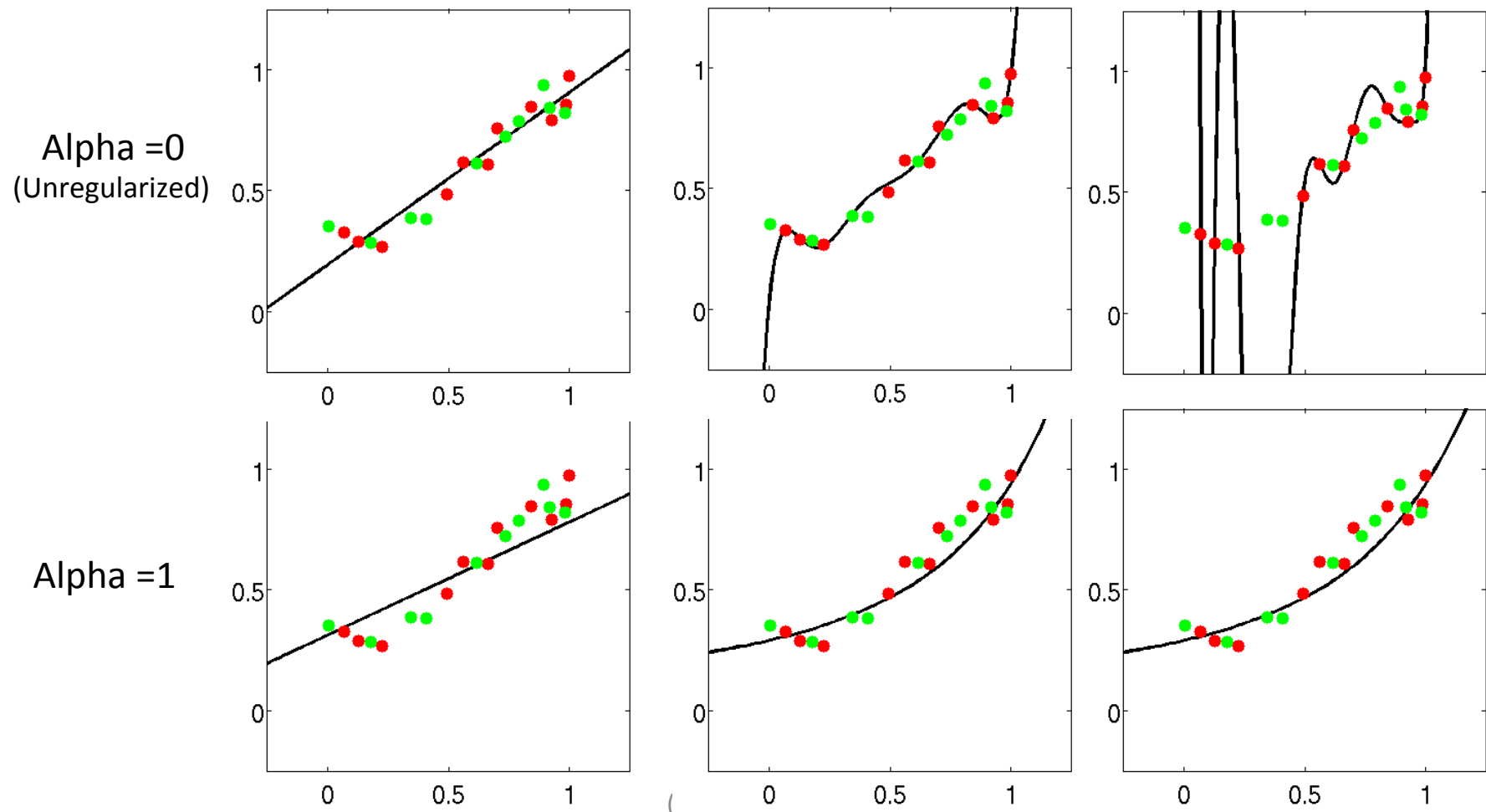
- Problem is now well-posed for any degree

- Notes:

- “Shrinks” the parameters toward zero
- Alpha large: we prefer small theta to small MSE
- Regularization term is independent of the data: paying more attention reduces our model variance

# Regularization

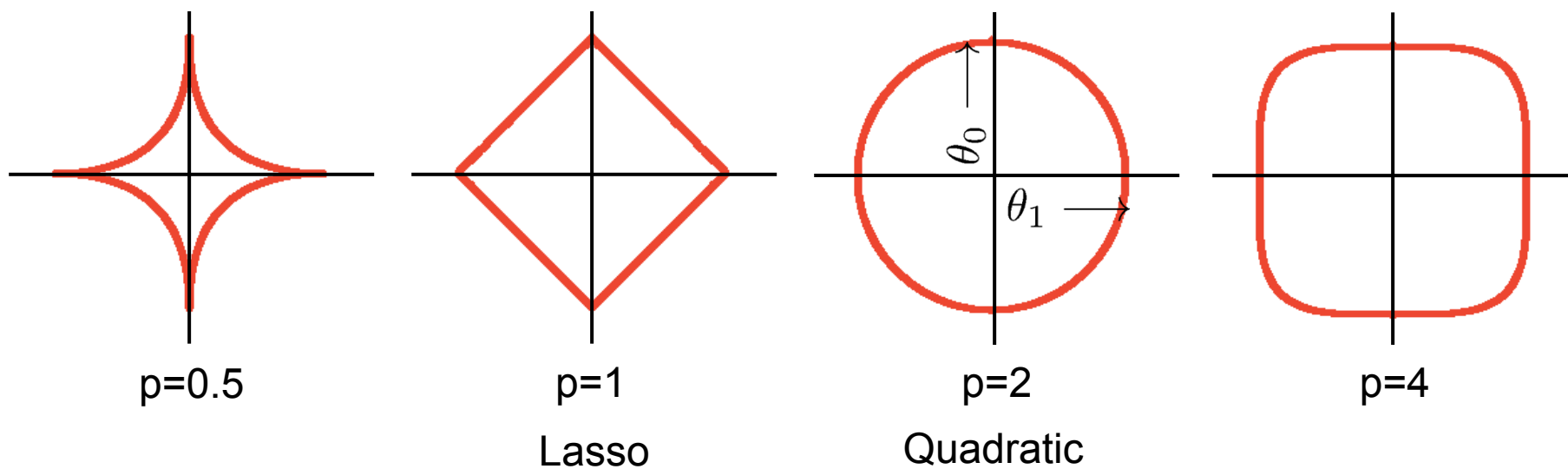
- Compare between unreg. & reg. results



# Different regularization functions

- More generally, for the  $L_p$  regularizer:  $\left( \sum_i |\theta_i|^p \right)^{\frac{1}{p}}$

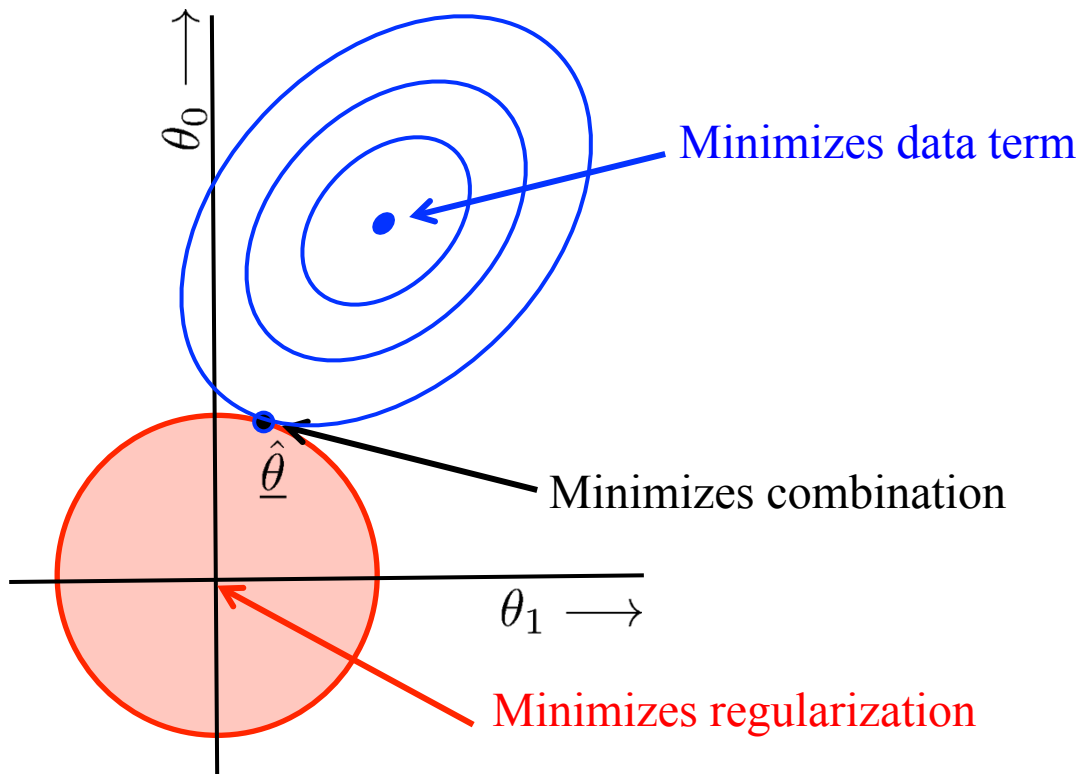
Isosurfaces:  $\|\theta\|_p = \text{constant}$



$L_0$  = limit as  $p \rightarrow 0$  : “number of nonzero weights”, a natural notion of complexity

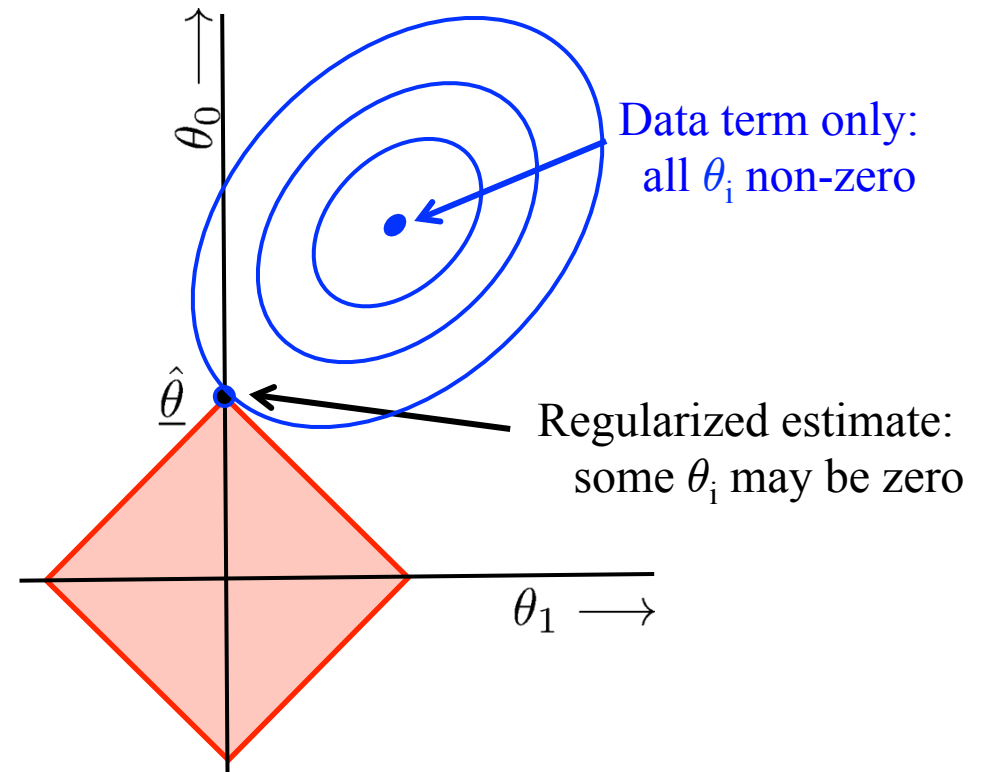
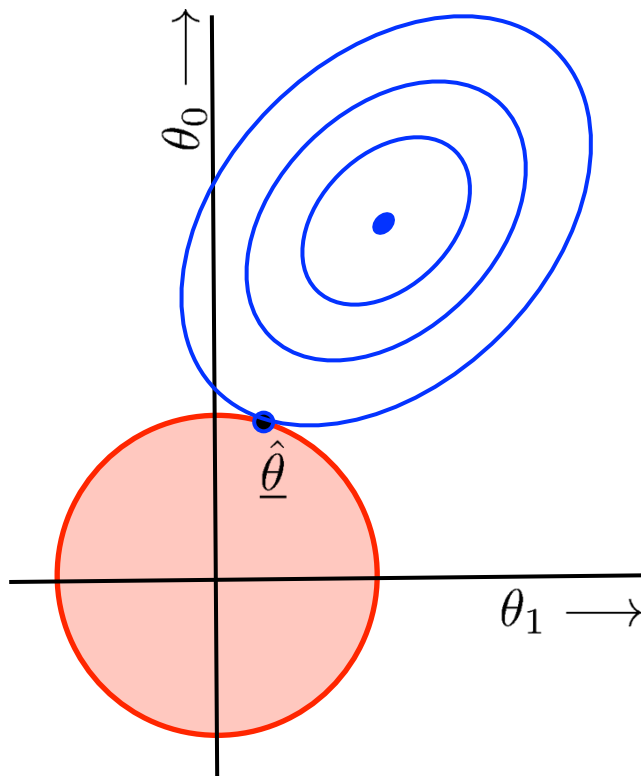
# Regularization: L1 vs L2

- Estimate balances data term & regularization term



# Regularization: L1 vs L2

- Estimate balances data term & regularization term
- Lasso tends to generate sparser solutions than a quadratic regularizer.



(c) Alexander Ihler

+

# Machine Learning and Data Mining

## Linear regression: hold-out, cross-validation

Prof. Alexander Ihler

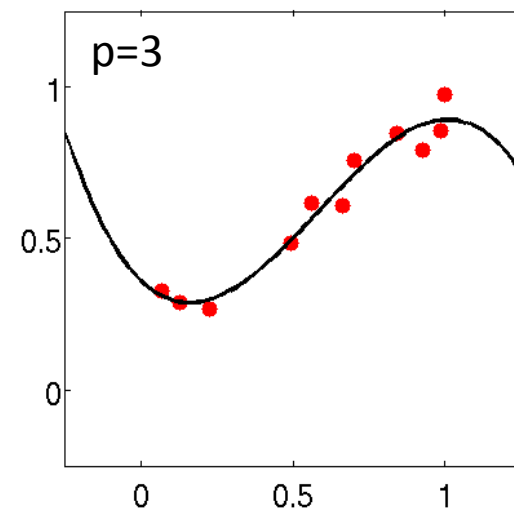
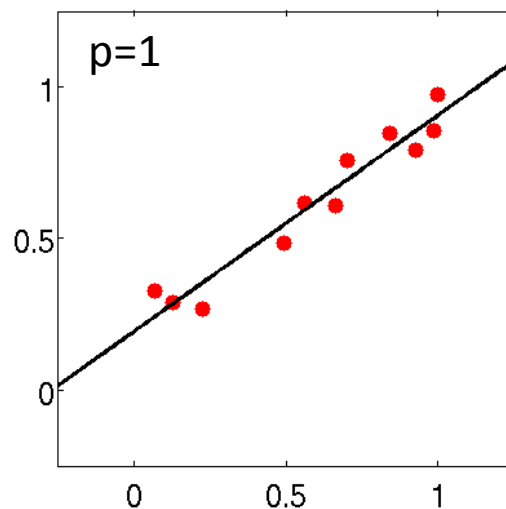
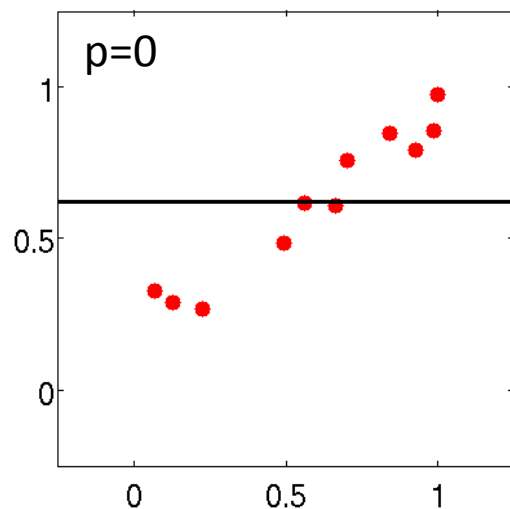




# Model selection

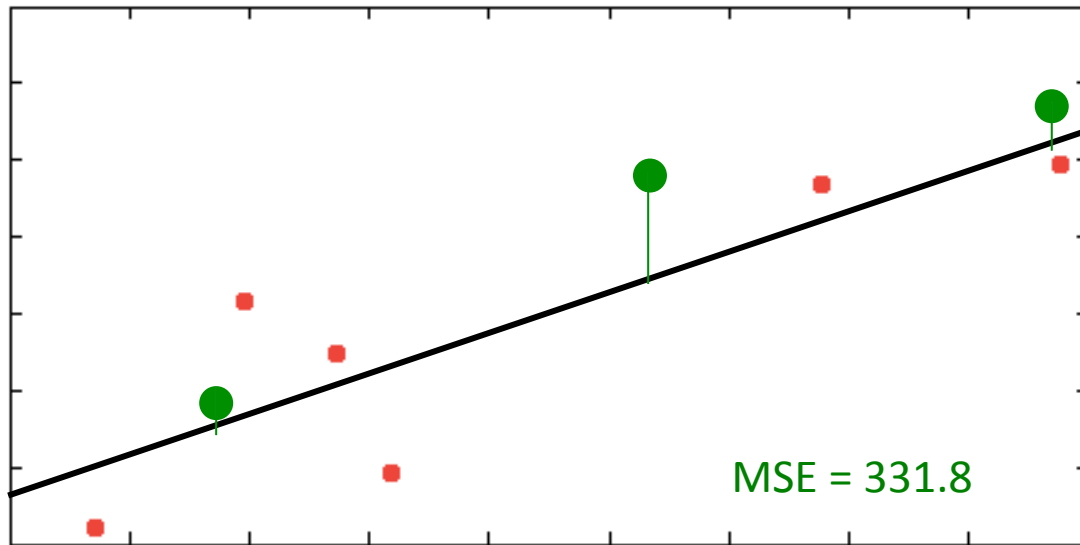
- Which of these models fits the data best?
  - $p=0$  (constant);  $p=1$  (linear);  $p=3$  (cubic); ...
- Or, should we use KNN? Other methods?
- Model selection problem
  - Can't use training data to decide (esp. if models are nested!)
- Want to estimate  $\mathbb{E}_{(x,y)}[J(y, \hat{y}(x; D))]$

$J$  = loss function (MSE)  
 $D$  = training data set



# Hold-out method

- Validation data
  - “Hold out” some data for evaluation (e.g., 70/30 split)
  - Train only on the remainder
- Some problems, if we have few data:
  - Few data in hold-out: noisy estimate of the error
  - More hold-out data leaves less for training!



(c) Alexander Ihler

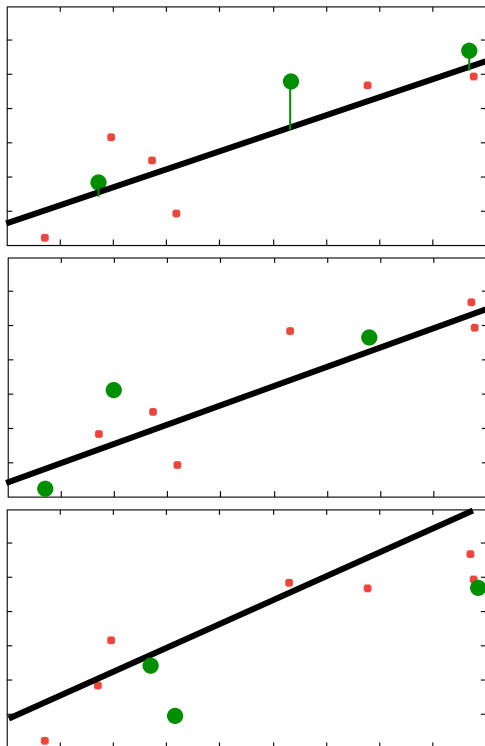
Training  
data

Validation  
data

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

# Cross-validation method

- K-fold cross-validation
  - Divide data into K disjoint sets
  - Hold out one set ( $= M / K$  data) for evaluation
  - Train on the others ( $= M * (K-1) / K$  data)



Split 1:  
MSE = 331.8

Split 2:  
MSE = 361.2

Split 3:  
MSE = 669.8



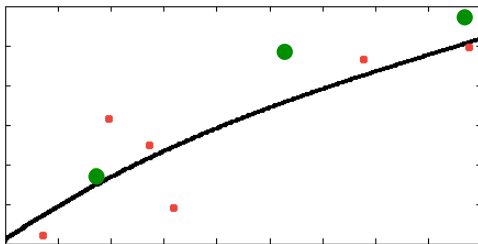
3-Fold X-Val MSE  
= 464.1

Training  
data  
Validation  
data

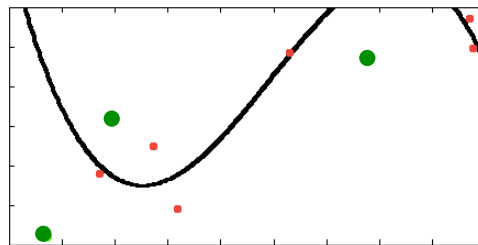
$x^{(i)}$	$y^{(i)}$
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

# Cross-validation method

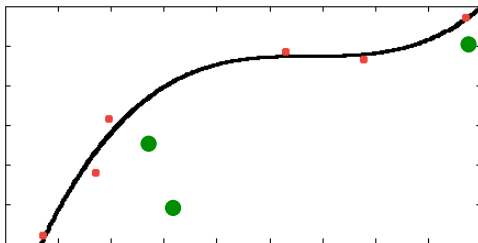
- K-fold cross-validation
  - Divide data into K disjoint sets
  - Hold out one set ( $= M / K$  data) for evaluation
  - Train on the others ( $= M * (K-1) / K$  data)



Split 1:  
MSE = 280.5



Split 2:  
MSE = 3081.3



Split 3:  
MSE = 1640.1



3-Fold X-Val MSE  
= 1667.3

Training  
data  
Validation  
data

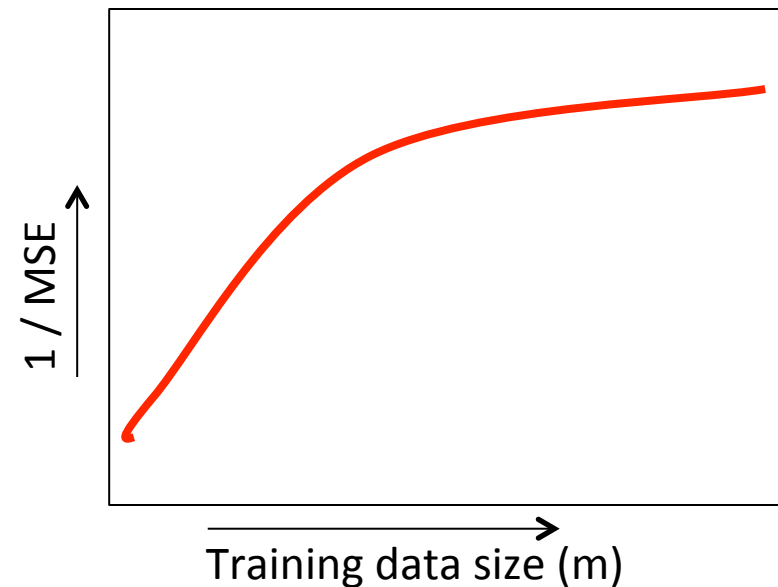
$x^{(i)}$	$y^{(i)}$
88	79
32	-2
27	30
68	73
7	-16
20	43
53	77
17	16
87	94

# Cross-validation

- Advantages:
  - Lets us use more (M) validation data  
(= less noisy estimate of test performance)
- Disadvantages:
  - More work
    - Trains K models instead of just one
  - Doesn't evaluate any *particular* predictor
    - Evaluates K different models & averages
    - Scores *hyperparameters / procedure*, not an actual, specific predictor!
- Also: still estimating error for  $M' < M$  data...

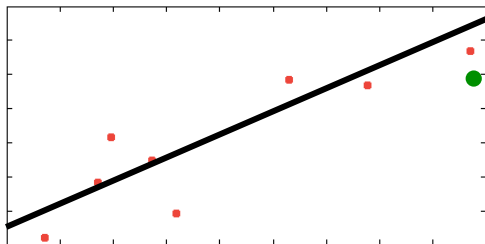
# Learning curves

- Plot performance as a function of training size
    - Assess impact of fewer data on performance  
Ex:  $MSE_0 - MSE$  (regression)  
or  $1 - \text{Err}$  (classification)
  - Few data
    - More data significantly improve performance
  - “Enough” data
    - Performance saturates
- If slope is high, decreasing  $m$  (for validation / cross-validation) might have a big impact...

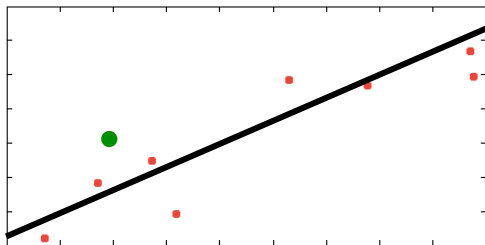


# Leave-one-out cross-validation

- When  $K=M$  (# of data), we get
  - Train on all data except one
  - Evaluate on the left-out data
  - Repeat  $M$  times (each data point held out once) and average



MSE = ...



MSE = ...

⋮



LOO X-Val MSE  
= ...

Training  
data

Validation  
data

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

# Cross-validation Issues

- Need to balance:
  - Computational burden (multiple trainings)
  - Accuracy of estimated performance / error
- Single hold-out set:
  - Estimates performance with  $M' < M$  data (important? learning curve?)
  - Need enough data to trust performance estimate
  - Estimates performance of a particular, trained learner
- K-fold XVal
  - K times as much work, computationally
  - Better estimates, still of performance with  $M' < M$  data
- LOO XVal
  - M times as much work, computationally
  - $M' \approx M$ , but overall error estimate may have high variance