Linear Regression Extensions

James G. Shanahan [1,2,3]

[1]Church and Duncan Group,

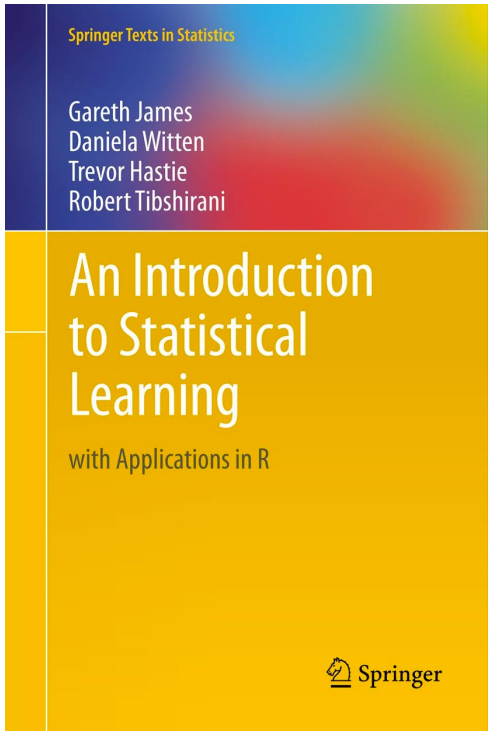
[2]Information School, UC Berkeley

[3]School of Informatics, Computing and Engineering, Indiana University

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Reading material

- **See Free PDF**
  - Please read chapter 4and 6 for a very good intro on linear regression and regularization
  - An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics) 1st ed. 2013, Corr. 6th printing 2016 Edition by Gareth James (Author), Daniela Witten (Author), Trevor Hastie (Author), Robert Tibshirani (Author)
  - http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf
  - http://cs231n.github.io/linear-classify/



Lecture Outline

- **Introduction**
- **Linear regression Gradient Descent flavors**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Lecture Outline

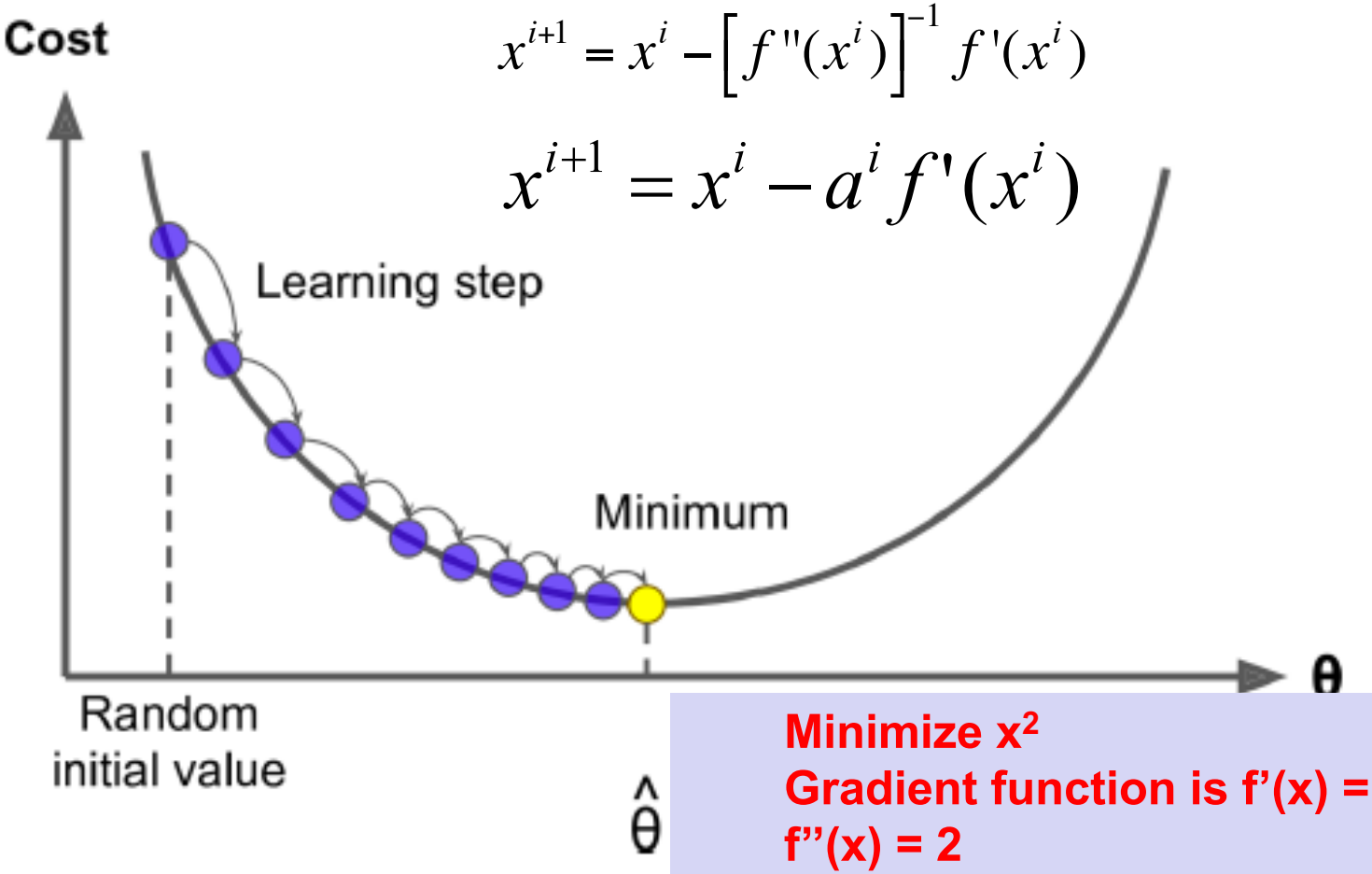
- **Introduction**
- **Linear regression Gradient Descent flavors**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Gradient provides direction and f''(x) the stepsize

TASK: Find the minimum of x^2

$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i)$$

$$x^{i+1} = x^i - a^i f'(x^i)$$



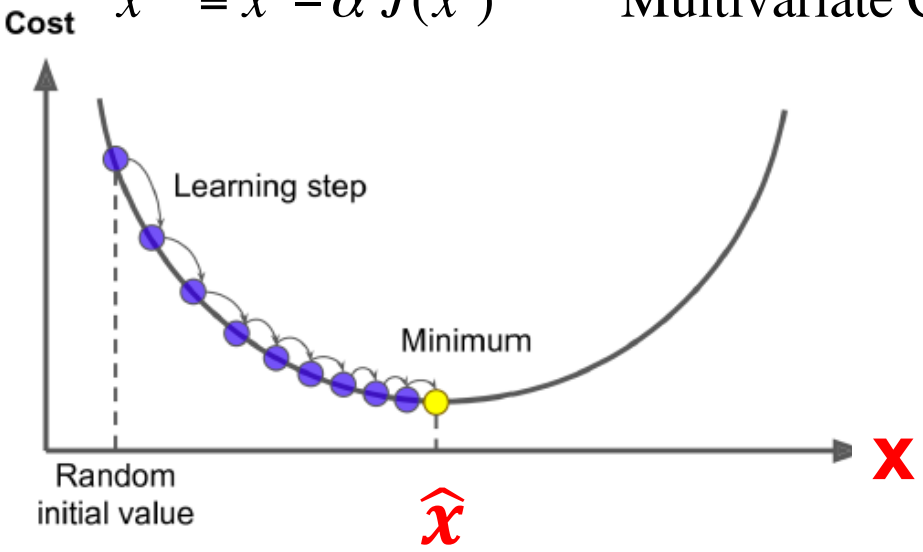
Minimize x^2
Gradient function is f'(x) = 2x
f''(x) = 2

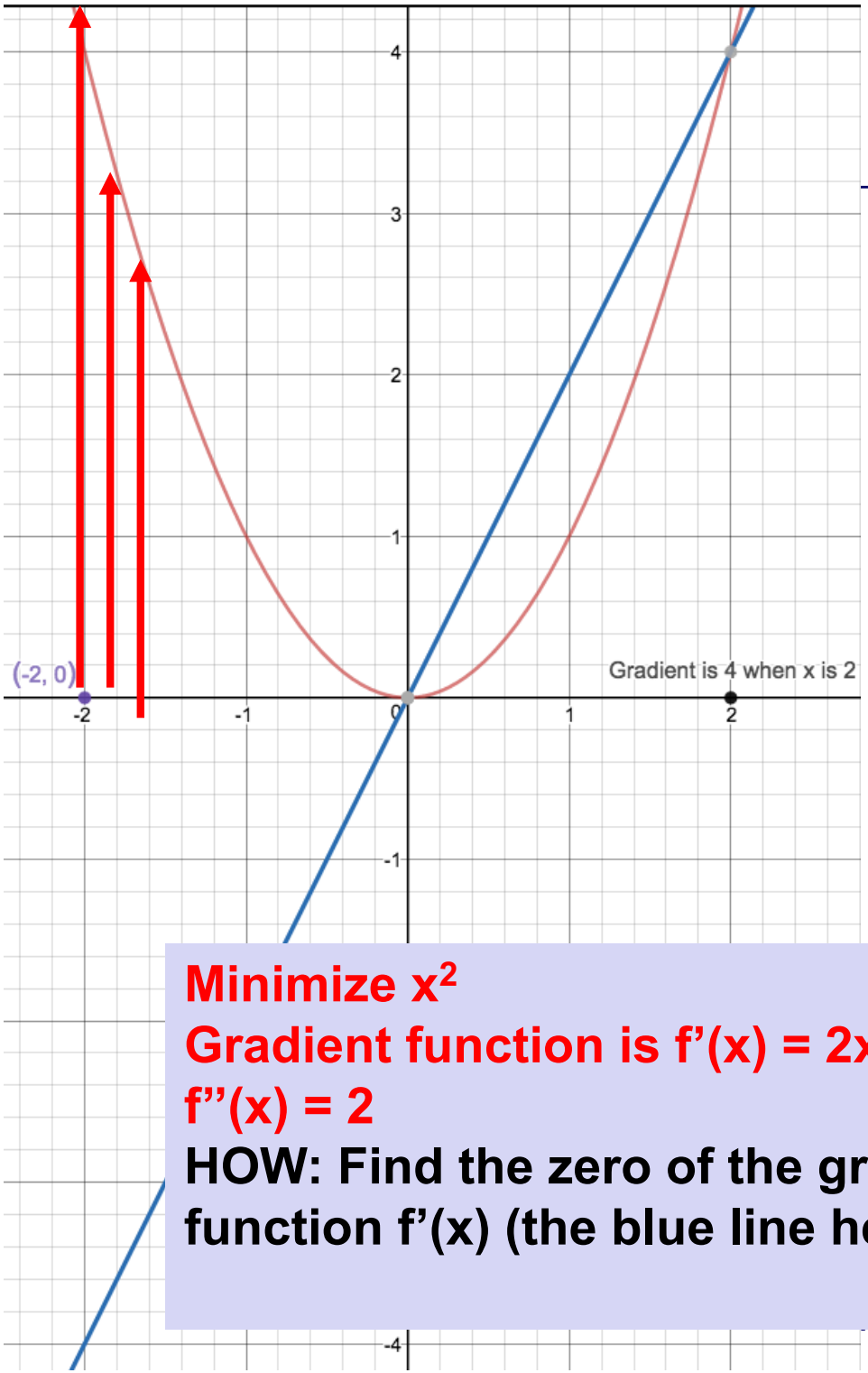
HOW: Find the zero of the gradient function f'(x)

Gradient update at x=-2

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$





Minimize x^2
Gradient function is f'(x) = 2x
f''(x) = 2
HOW: Find the zero of the gradient function f'(x) (the blue line here)

$$x = x - \frac{df(x)}{dx} \qquad (1)$$

$$x = x - \frac{dx^2}{dx} \qquad (2)$$

$$x = x - 2x \qquad (3)$$

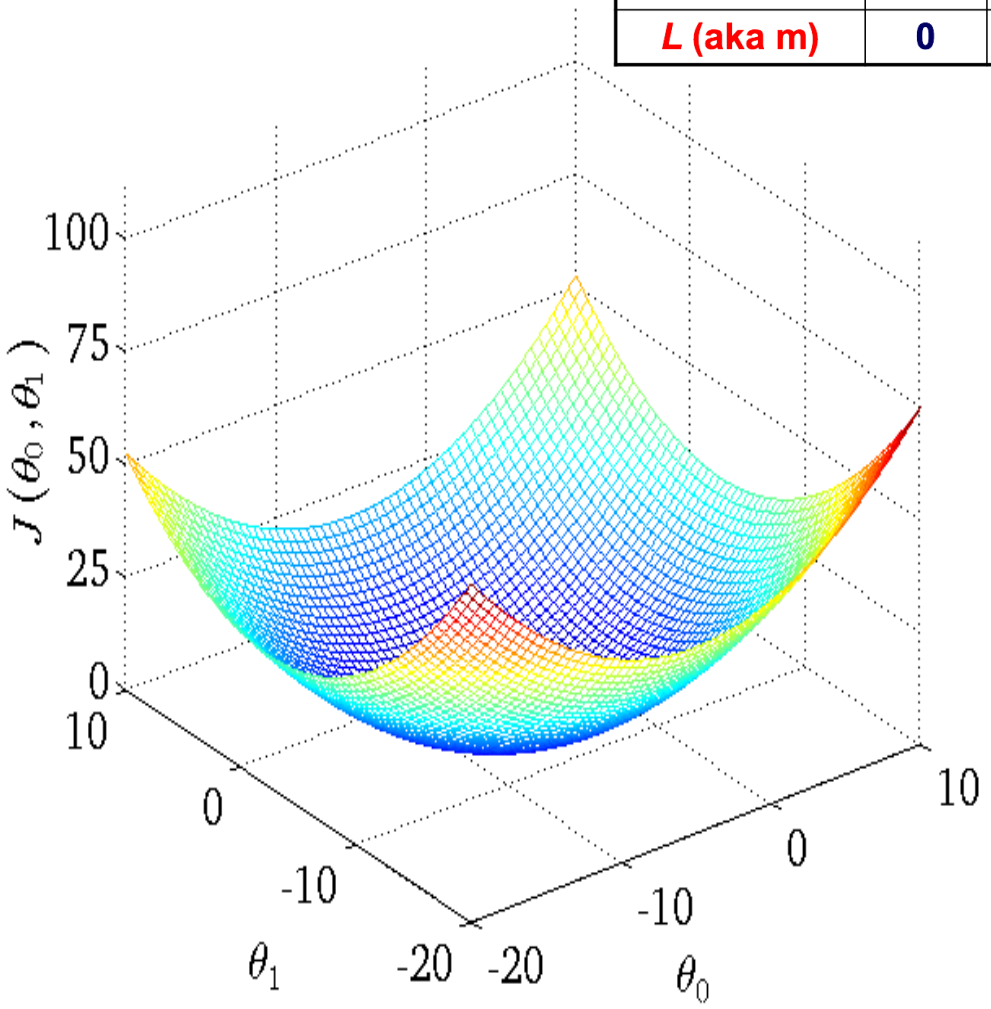
$$x = 2 - 2\times -2 \qquad (4)$$

$$x = 2 + 4 \qquad (5)$$

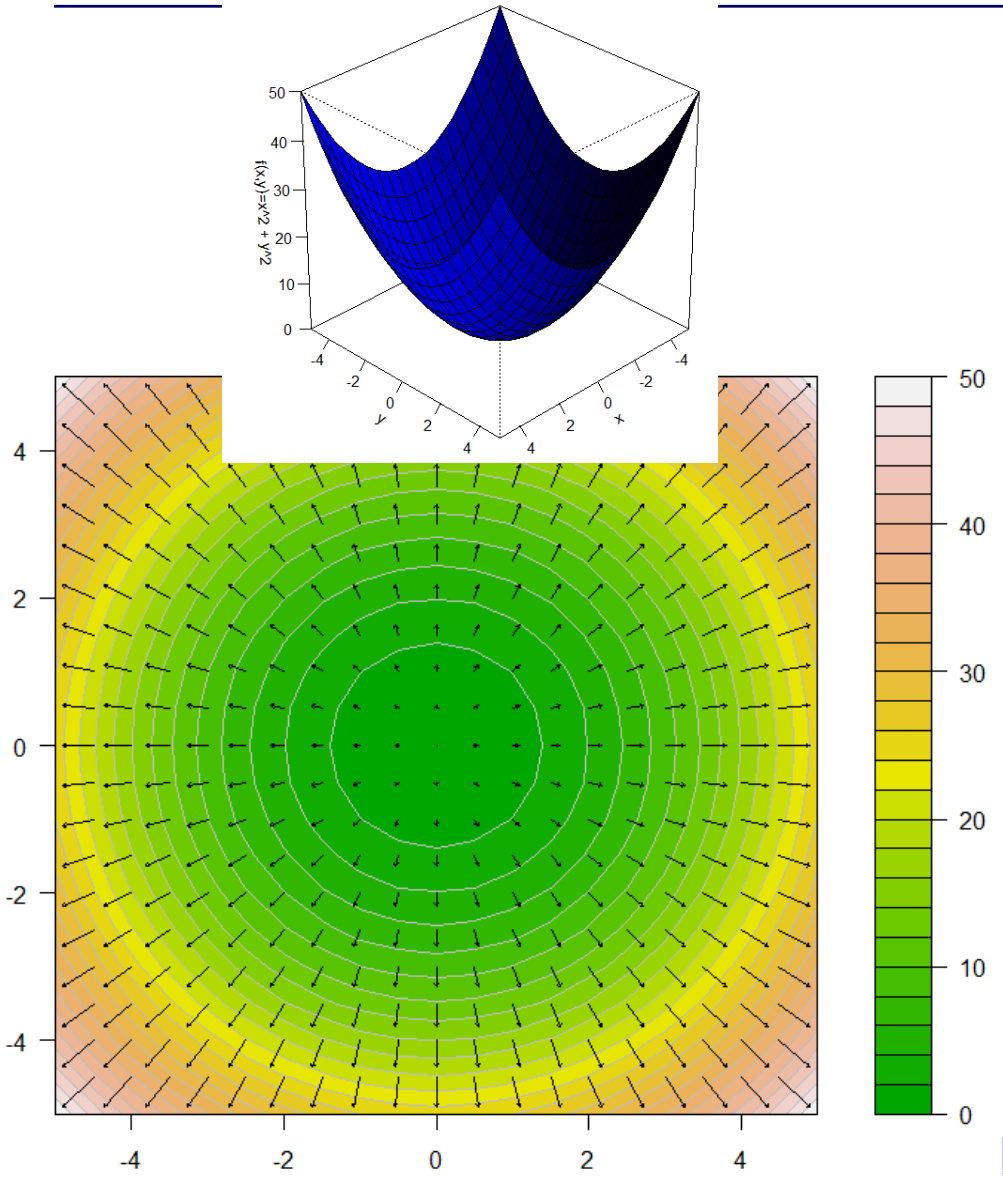
Error Surface for OLS: Price~Size

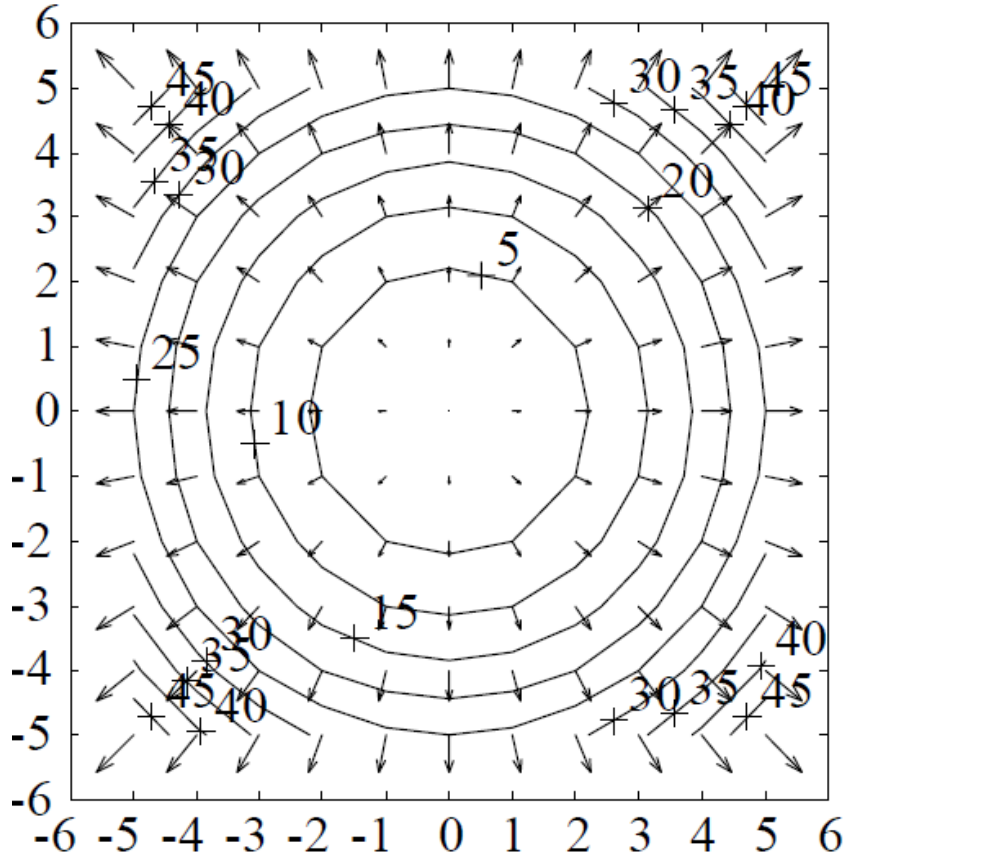
$$J_q(W, X_1^L) = \frac{1}{m}\sum_{i=1}^{m}(W^T X_i - y_i)^2$$

Instance\Attr	x_1	x_2	...	x_n	y
1	3	0	..	7	73
2					76
...	...	...	...	...	...
L (aka m)	0	4	...	8	97



Gradient Vector Plots of $f(x,y)=x^2+y^2$





http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf

Linear Regression

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

- θ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- θ^T is the transpose of θ (a row vector instead of a column vector).
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\theta^T \cdot \mathbf{x}$ is the dot product of θ^T and $\mathbf{x}$.
- h_θ is the hypothesis function, using the model parameters θ.

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Single variable $$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Gradient Descent in vector form

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

MSE Loss Function (Objective)

$$\nabla_\theta \mathrm{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0}\mathrm{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1}\mathrm{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n}\mathrm{MSE}(\theta) \end{pmatrix} = \frac{2}{m}\mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

Gradient of partial derivatives

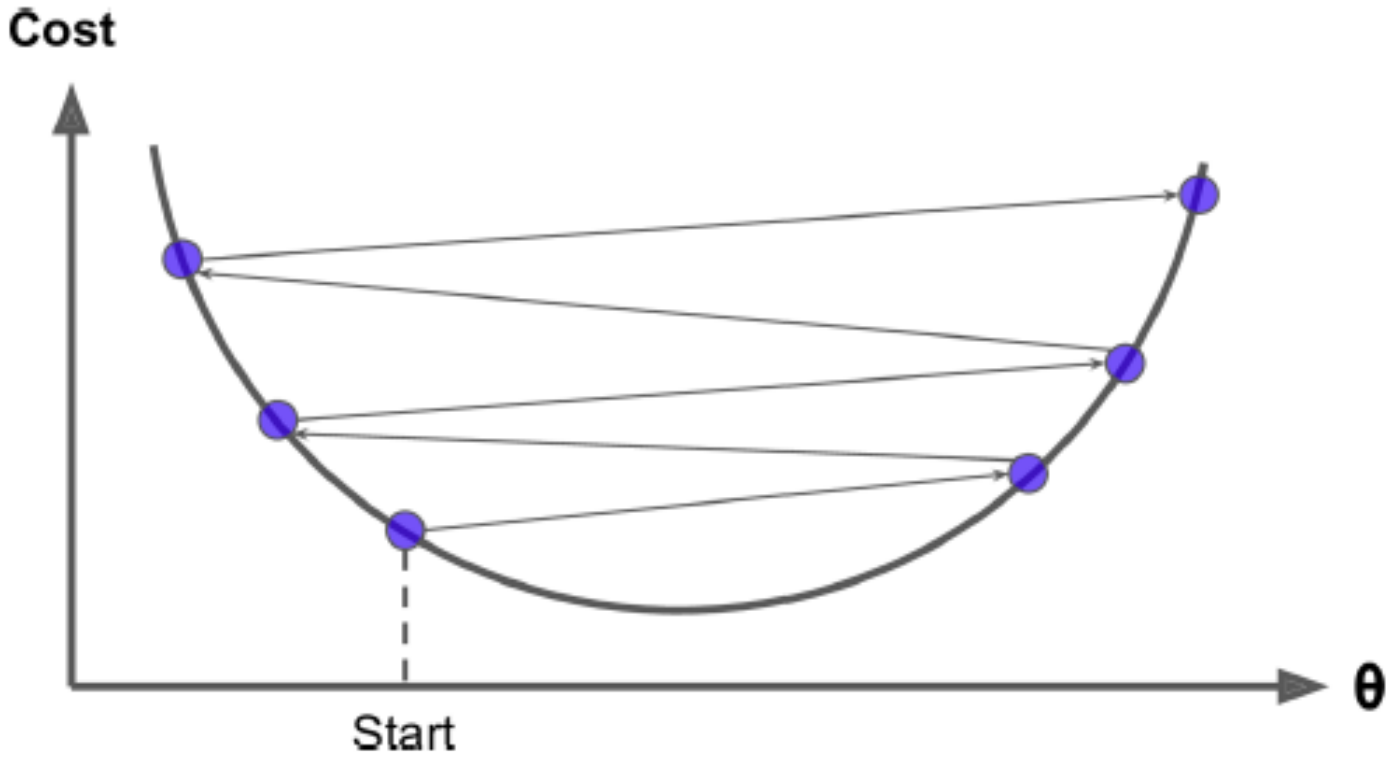
$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \mathrm{MSE}(\theta)$$

GD Update rule

Implementation

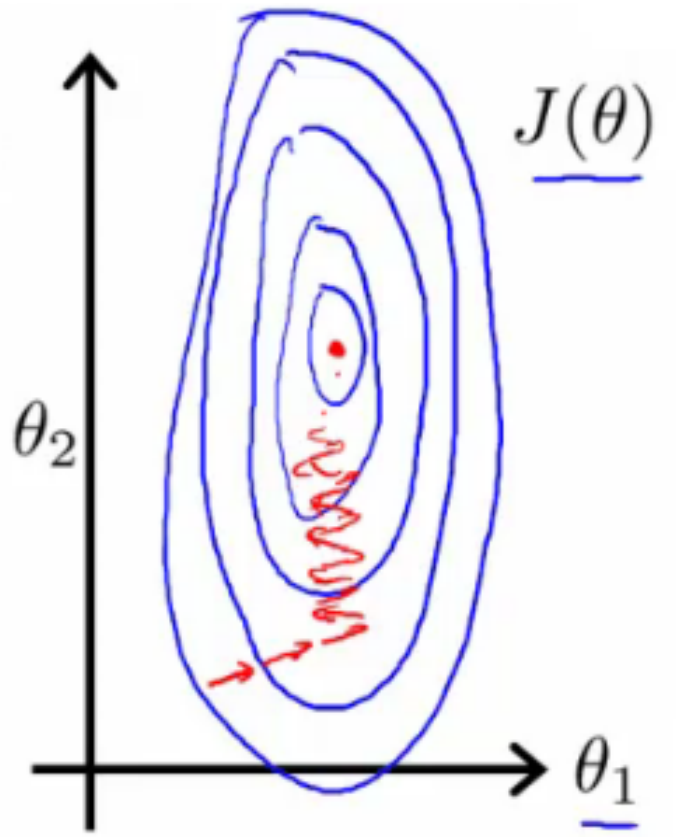
```
eta = 0.1  # learning rate
n_iterations = 1000
m = 100

theta = np.random.randn(2,1)  # random initialization

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

Update rate (aka learning rate)



Sometimes we overshoot

Alpha our learning rate needs to be small enough to allow us to converge. Otherwise we will oscilliate. …can also decrease ALPHA over time… Gradient descent provides no guarantees to find the minimum but….



$$RSS = Variance \text{ of } \varepsilon$$

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^{n} \left(X_j W - y_i \right)^2 \right)}{\partial W}$$

$$\nabla J(W) = \left(\sum_{j=1}^{n} \left(X_j W - y_i \right) X_j \right)$$

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^{n} \left(X_j W^t - y_i \right) X_j \right)$$

Learning Rate: α

- **Fixed learning rate**
  - While it is more common to run stochastic gradient descent as we have described it and with a fixed learning rate
- **Dynamic, decreasing learning rate**
  - by slowly letting the learning rate decrease to zero as the algorithm runs, it is also possible to ensure that the parameters will converge to the global minimum rather then merely oscillate around the minimum

- **Or it can be calculated**

$$\alpha = \frac{\mathbf{h}^{\mathbf{T}}\mathbf{h}}{\mathbf{h}^{\mathbf{T}}\mathbf{H}\mathbf{h}} = \frac{\nabla f(x_0)^{\mathbf{T}} \nabla f(x_0^{\mathbf{T}})}{\nabla f(x_0)^{\mathbf{T}} H \nabla f(x_0^{\mathbf{T}})}$$

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Stochastic Gradient Decent vs. Batch

- **Stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at.**
- **Often, stochastic gradient descent gets W "close" to the minimum much faster than batch gradient descent.**
  - Note however that it may never "converge" to the minimum, and the parameters W will keep oscillating around the minimum of J(W); but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.
- **For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.**

Shuffle the data thru permuting

```
import numpy as np
indx = np.arange(10)
print("before permutation", indx)
indx = np.random.permutation(indx)
print("after permutation ", indx)
```

```
('before permutation', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))
('after permutation ', array([4, 3, 9, 8, 6, 5, 0, 7, 2, 1]))
```

SGD: Use ALL data in each epoch (just change order)

```
: import numpy as np
  x = np.arange(15).reshape(-1, 3)
  x[0:3]  #take first three rows of data

: array([[0, 1, 2],
         [3, 4, 5]])
```

Mini-batch1: MB1 MB2

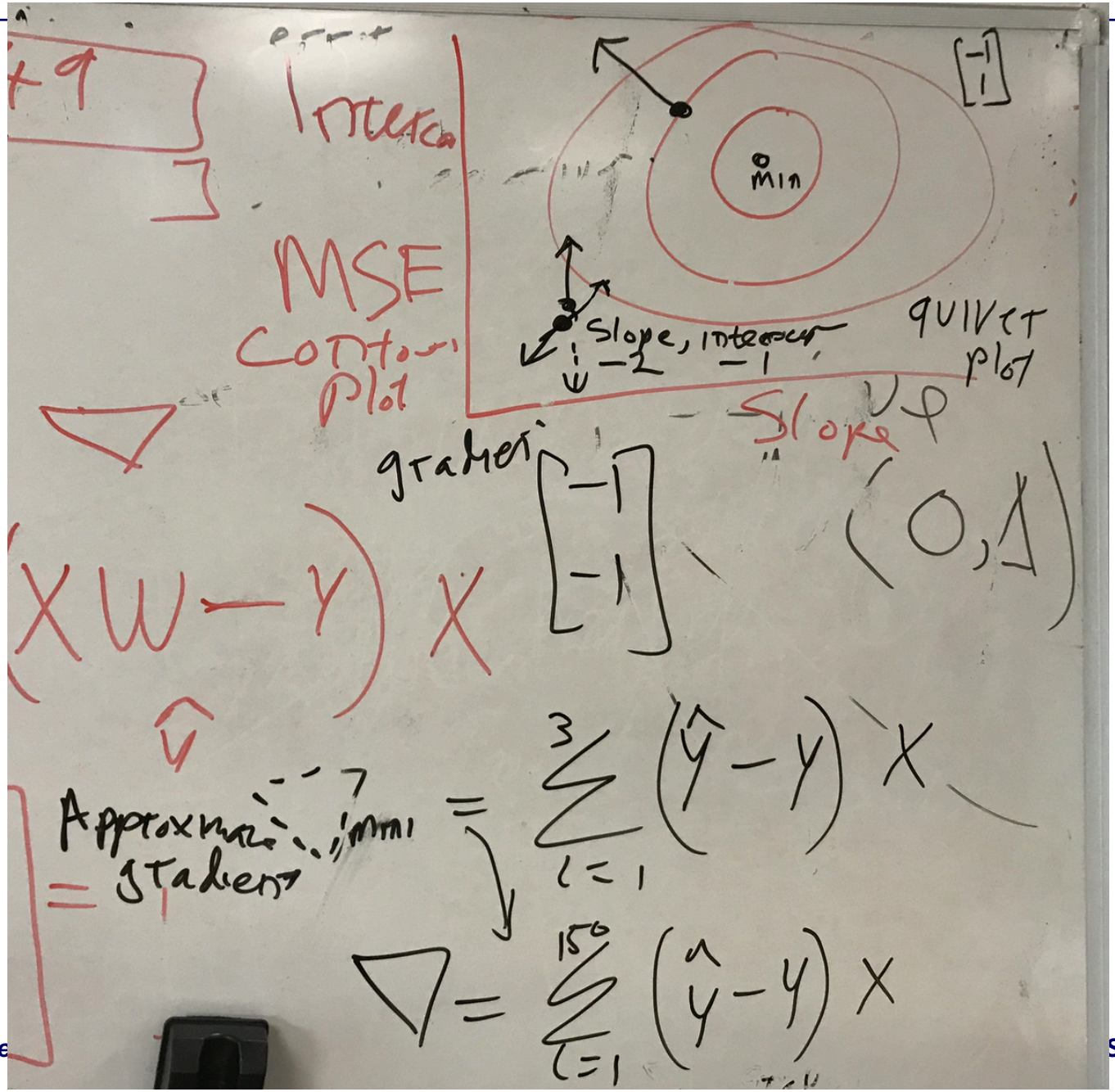
Idxs = ([4, 3, 9, 8, 6, 5, 0, 7, 2, 1]))
with batch is size =3

```
for epoch in range(max_iter):
    idxs = np.random.permutation(X.shape[0])
    X = X[idxs]
    y = y[idxs]
    for i in range(0, len(X), batch_size):

        self.history["coef"].append(self._theta[1:])
        self.history["intercept"].append(self._theta[0])

        rmse = self.score(X, y)
        self.history["cost"].append(rmse)

        # calculate gradient
        grad = self._grad(X[i:i + batch_size], y[i:i + batch_size])
        self.history["grad"].append(grad)

        # numerical gradient
        grad_num = self._grad_num(X[i:i + batch_size], y[i:i + batch_size])
        self.history["grad_num"].append(grad_num)

        # do gradient step
        self._theta -= alpha * grad
```

Shuffle data during each epoch

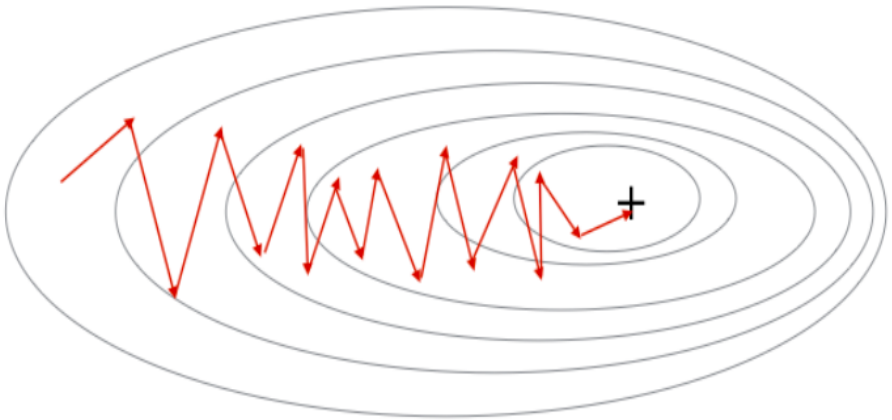
Mini-batch via slicing

MiniBatch approximates the slope

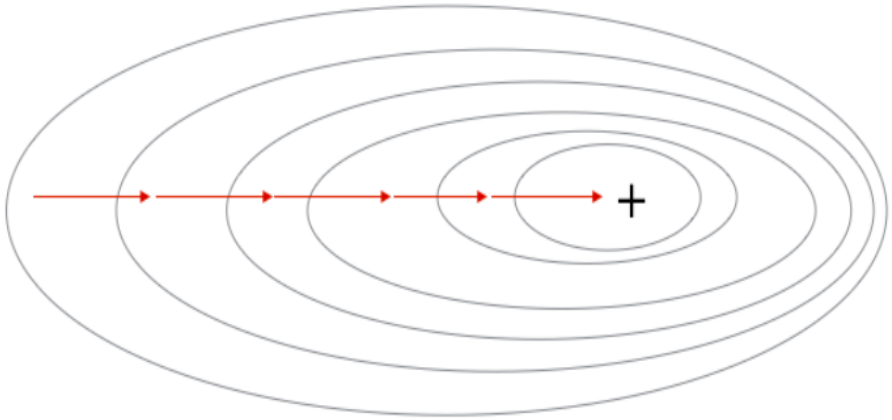


Stochastic GD (one example per gradient descent step)

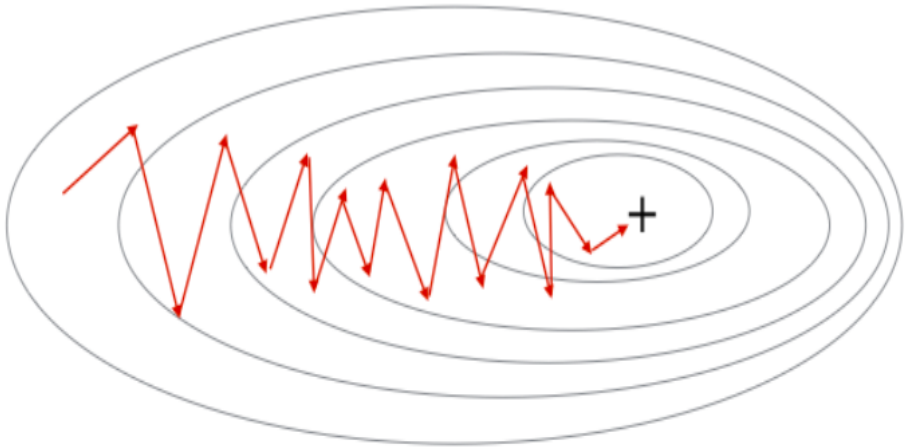
Stochastic Gradient Descent



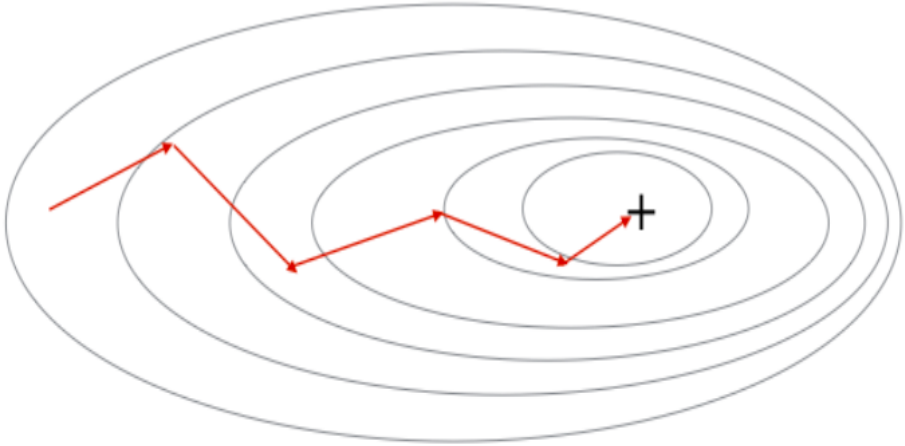
Gradient Descent



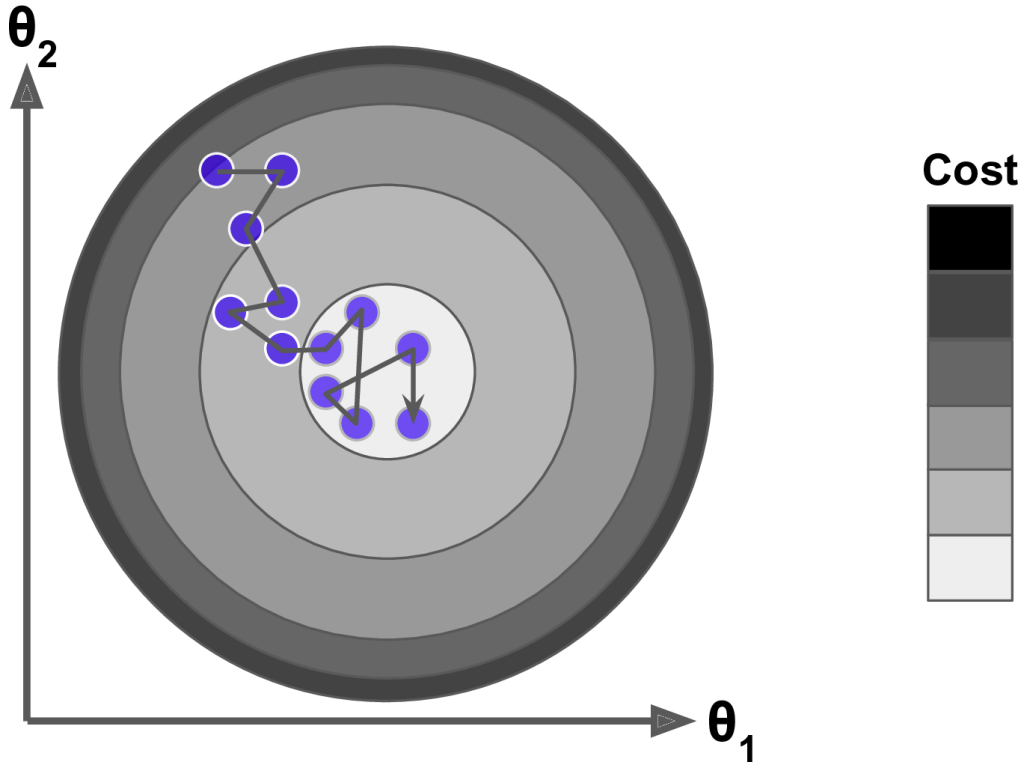
Stochastic Gradient Descent



Mini-Batch Gradient Descent



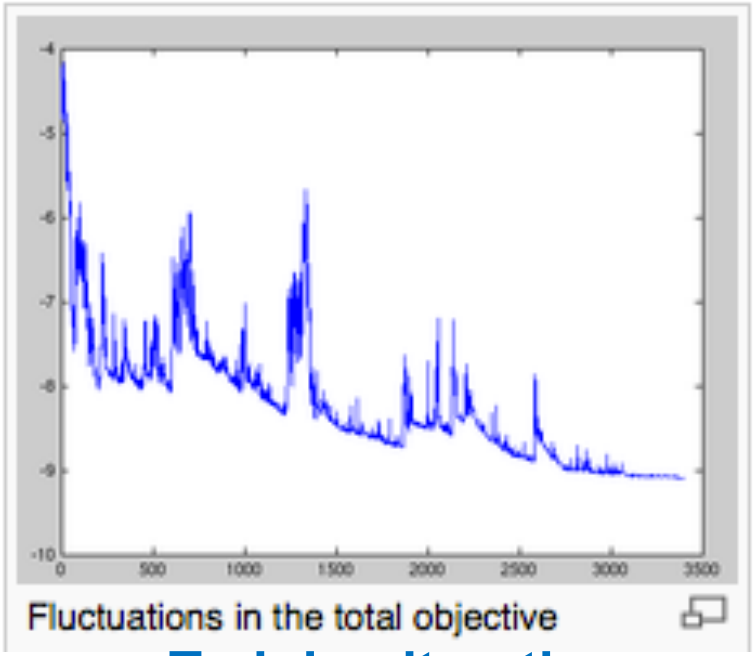
Stochastic Gradient Descent



OLS using Stochastic Gradient Descent

- **Stochastic update (after each example)**

$\nabla J_{wj}(W_t)$
Partial derivate WRT to variable w_j of error function J(W) at point W_t



Let $W = (0,0,....)$

Repeat

For j in 0..n #each variable

For i in 1..m #each example

$$W_{j,t+1} = W_{j,t} + \alpha * (y^i - W_{j,t} X^i) X^i_j$$

until convergence (i.e., no big changes in W or error)

Stochastic Gradient Descent using a simple learning schedule

m training examples
Adopt learning rate

```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)  # random initialization

for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
```

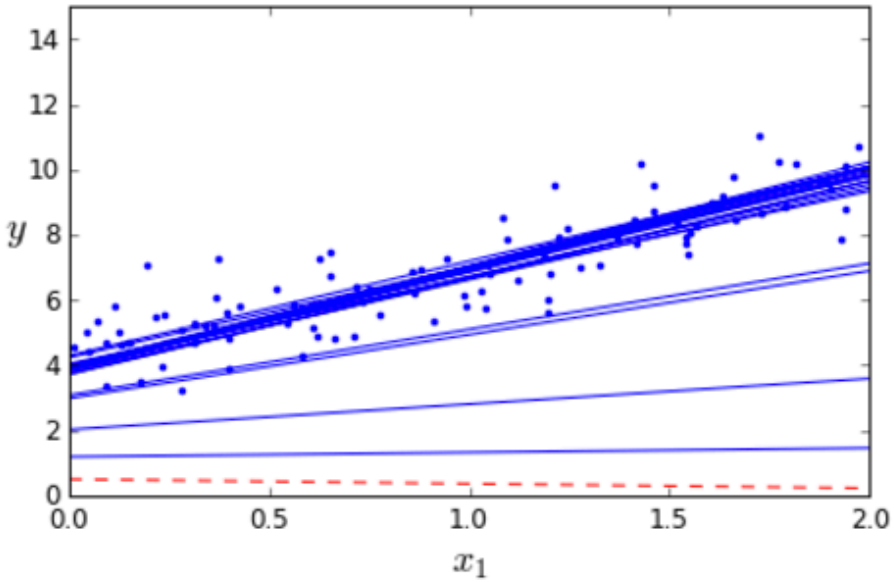
Update model with each example

By convention we iterate by rounds of m iterations; each round is called an epoch

4 Stochastic Gradient Descent

Pick one example at the time

```
theta_path_sgd = []

n_iterations = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

rnd.seed(42)
theta = rnd.randn(2,1)  # random initialization

def learning_schedule(t):
    return t0 / (t + t1)

m = len(X_b)

for epoch in range(n_iterations):
    for i in range(m):
        if epoch == 0 and i < 20:
            y_predict = X_new_b.dot(theta)
            style = "b-" if i > 0 else "r--"
            plt.plot(X_new, y_predict, style)
        random_index = rnd.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
        theta_path_sgd.append(theta)

plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
save_fig("sgd_plot")
plt.show()
```



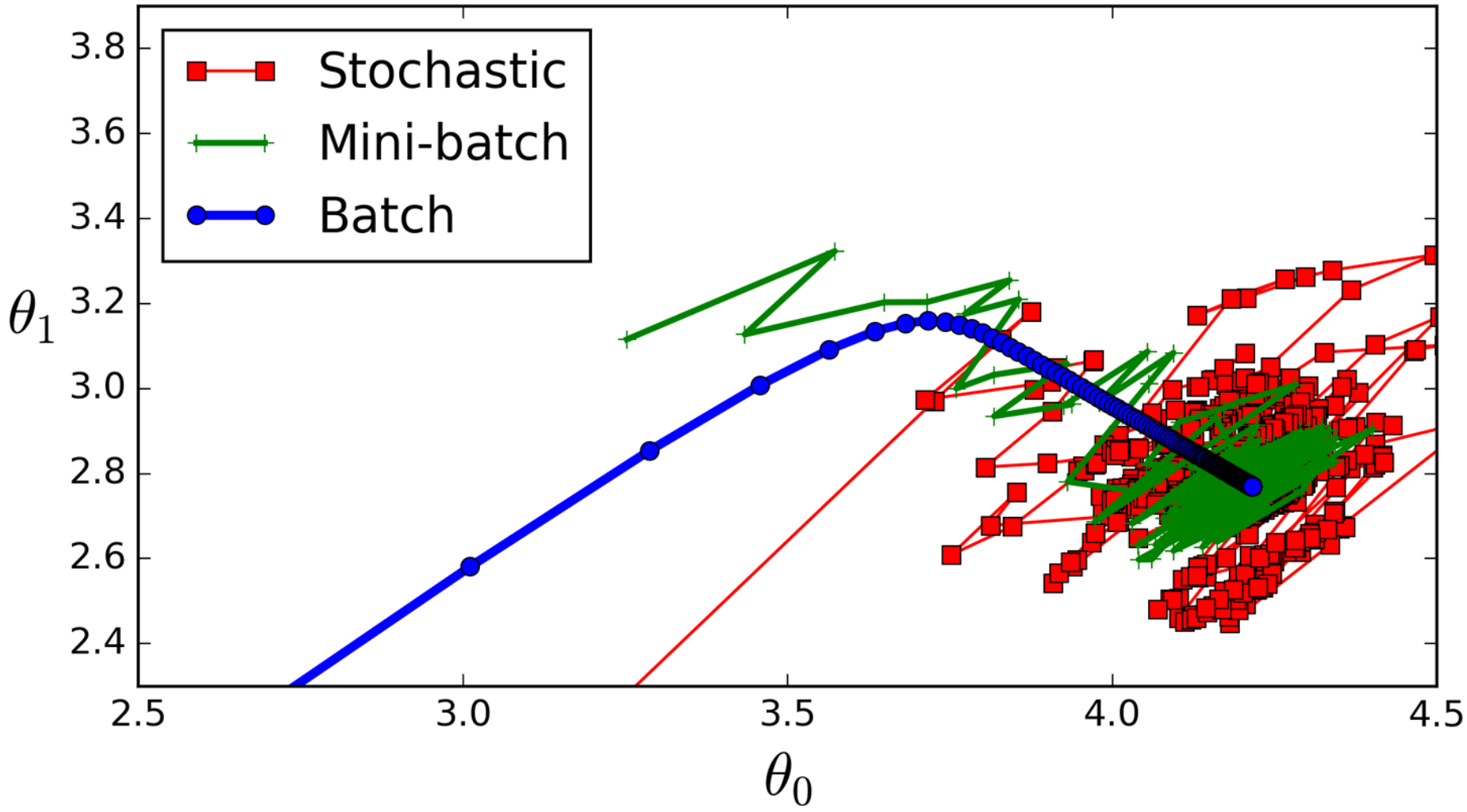
Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Mini-batch GD

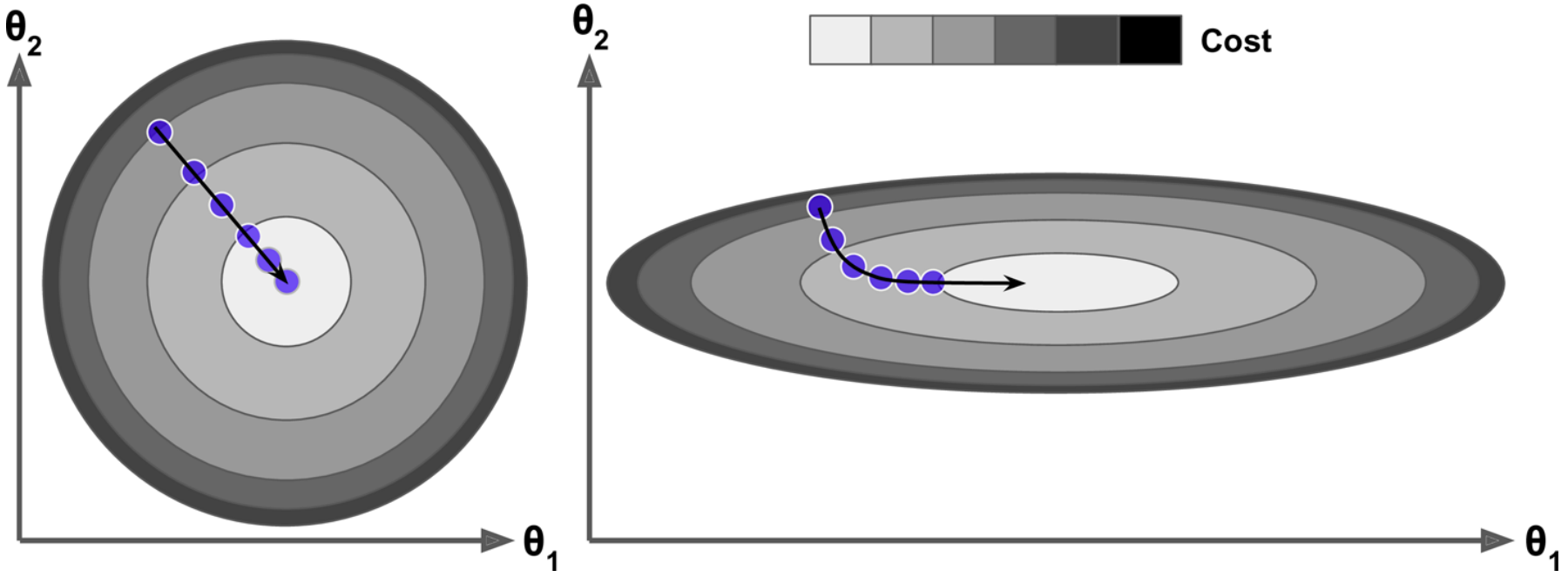
- **The last Gradient Descent algorithm we will look at is called Mini-batch Gradient Descent. It is quite simple to understand once you know Batch and Stochastic Gradient Descent:**
  - At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called minibatches.
- **The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.**

Gradient Descent paths in parameter space



Gradient Descent with and without feature scaling

- In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales. Figure below shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set
- On the RHS graph, the GD algorithm first goes in a direction almost orthogonal to the direction of the global minimum, and it ends with a long march down an almost flat valley



Lecture Outline

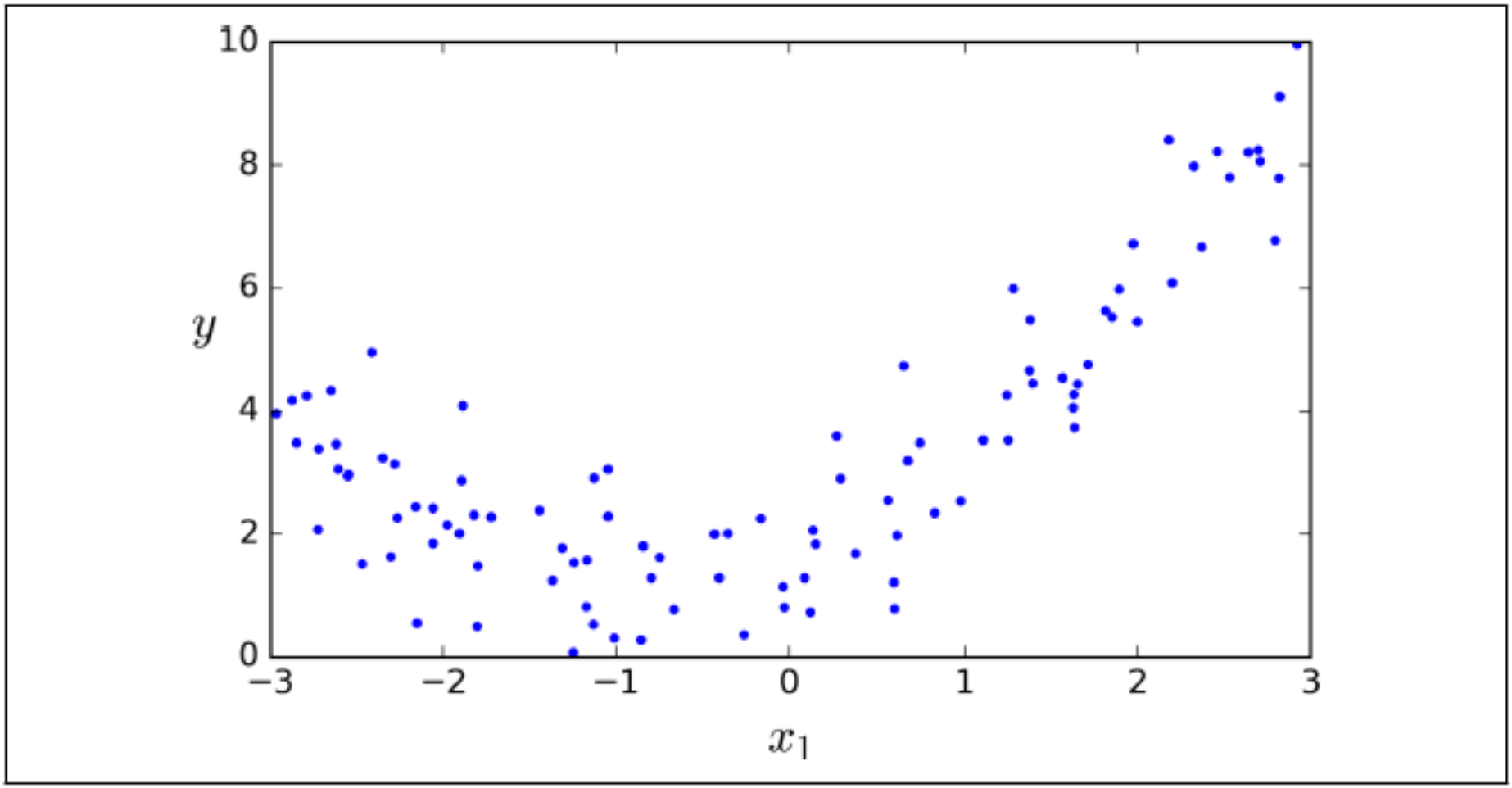
- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Polynomial Regression

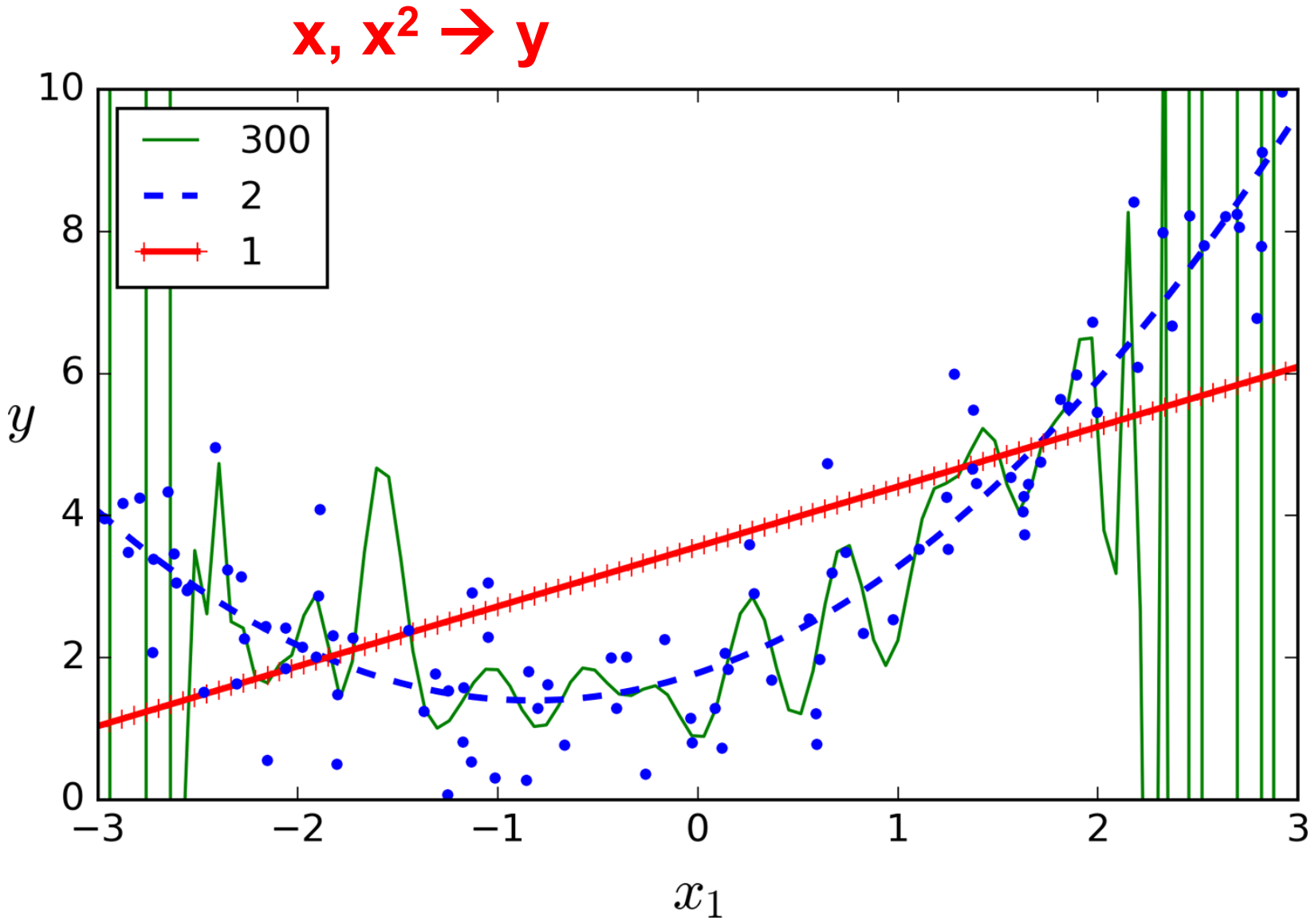
- In statistics, polynomial regression is a form of linear regression in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial.
- Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y | x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments, and the progression of disease epidemics.
- Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

Generating nonlinear and noisy dataset

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



High-degree Polynomial Regression



nbviewer.jupyter.org/urls/dl.dropbox.com/s/xw7yh7lk2y9icqq/ols.ipynb

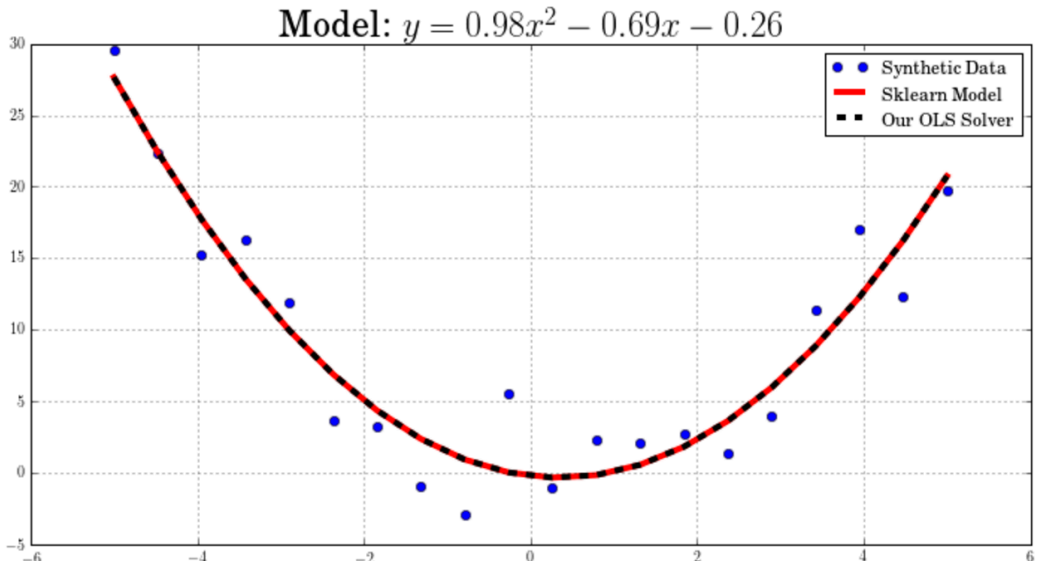
ogle Docs (99+) MIDS-MLS-2... Word2Vec: an intro... nbviewer.ipython.org Bookmarks (2) MIDS-MLS-201... Stanford Machine L... Getting Started Statistical Analysis...

jupyter nbviewer

JUPYTER FAQ

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/xw7yh7lk2y9icqq/ols.ipynb

```
y_plot = model.predict(np.reshape(x, (num

# Synthetic data
plt.figure(figsize=(12, 6))
plt.plot(x, y, 'o', label='Synthetic Data

# again, sklearn needs a nx1 array
plt.plot(x, y_plot, color='red', linewidth=4, label='Sklearn Model')

# our ols solver
c,b,a = polyfit(x, y, 2)
plt.plot(x, a*x**2 + b*x + c, '--k', linewidth=4, label='Our OLS Solver')

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
title = r'Model: $y = %.2f {x^2} %.2fx %.2f$' % (a, b, c)
plt.title(title, fontsize=24)

plt.legend()
plt.grid()
plt.show()
```



In [21]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
X[0]
```

Out[21]:
```
array([-0.75275929])
```

In [22]:
```
X_poly[0]
```

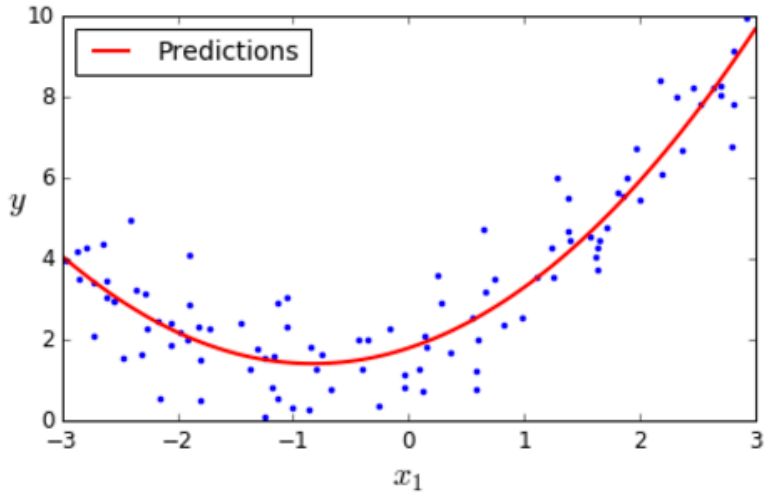
Out[22]:
```
array([-0.75275929,  0.56664654])
```

In [23]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

Out[23]:
```
(array([ 1.78134581]), array([[ 0.93366893,  0.56456263]]))
```

In [24]:
```
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)
plt.plot(X, y, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([-3, 3, 0, 10])
save_fig("quadratic_predictions_plot")
plt.show()
```

```
Saving figure quadratic_predictions_plot
```



PolynomialFeatures with degree=3

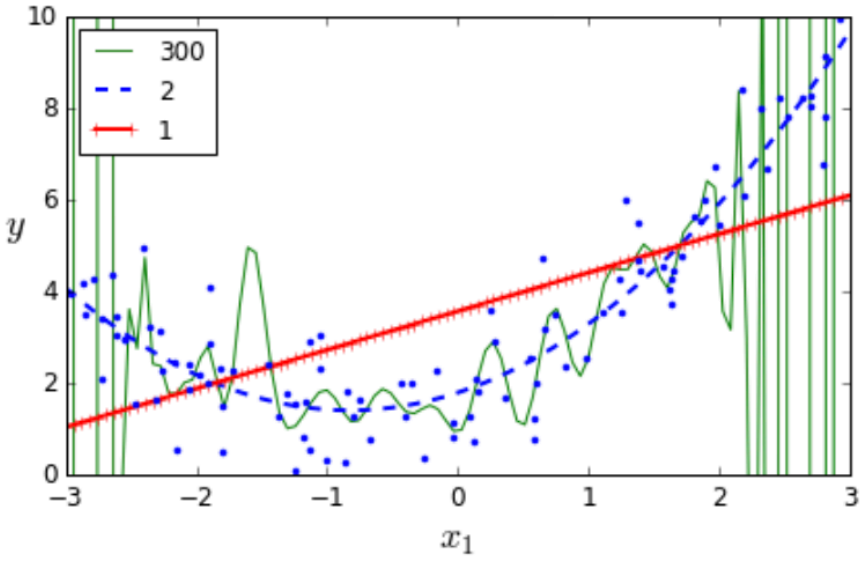
- PolynomialFeatures with degree=3 would not only add the features a^2, a^3, b^2, and b^3, but also the combinations ab, a^2b, and ab^2.

- PolynomialFeatures(degree=d) can get combinatorially explosive very quickly

Pipeline

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

for style, width, degree in (("g-", 1, 300), ("b--", 2, 2), ("r-+", 2, 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline((
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ))
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig, style, label=str(degree), linewidth=width)

plt.plot(X, y, "b.", linewidth=3)
plt.legend(loc="upper left")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([-3, 3, 0, 10])
save_fig("high_degree_polynomials_plot")
plt.show()
```

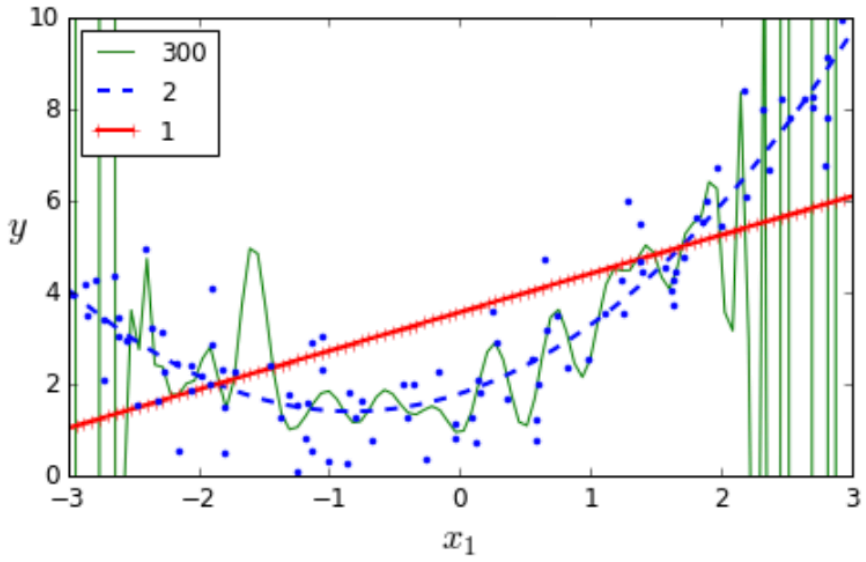
```
Saving figure high_degree_polynomials_plot
```



Pipeline

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

for style, width, degree in (("g-", 1, 300), ("b--", 2, 2), ("r-+", 2, 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline((
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ))
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig, style, label=str(degree), linewidth=width)

save_fig("high_degree_polynomials_plot")
plt.show()
```

```
Saving figure high_degree_polynomials_plot
```



Polynomial feature: x^2

```
:
```

```
1  import statsmodels.formula.api as sm
2  from sklearn.linear_model import LinearRegression
3  import pandas as pd
4  lin_reg = LinearRegression()
5
6  df = pd.DataFrame(np.c_[y, np.array(X_poly)],columns=["Y", "X","XP2"])
7  df.head()
```

```
:
```

	Y	X	XP2
0	1.617611	-0.752759	0.566647
1	8.061859	2.704286	7.313162
2	4.452506	1.391964	1.937563
3	0.779585	0.591951	0.350406
4	1.846257	-2.063888	4.259634

Polynomial model

```
1 lin_reg.fit(X_poly, y)
2 #https://medium.com/@dhwajraj/learning-python-regression-analys
3 lm= sm.ols(formula='Y~X+XP2',data =df).fit()
4 Y_ols_pred=lm.predict(df[["X", "XP2"]]) #make predictions
5 lm.summary()
```

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.853
Model:	OLS	**Adj. R-squared:**	0.849
Method:	Least Squares	**F-statistic:**	280.3
Date:	Sun, 10 Dec 2017	**Prob (F-statistic):**	4.85e-41
Time:	10:57:29	**Log-Likelihood:**	-129.29
No. Observations:	100	**AIC:**	264.6
Df Residuals:	97	**BIC:**	272.4
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.7813	0.139	12.782	0.000	1.505	2.058
X	0.9337	0.051	18.420	0.000	0.833	1.034
XP2	0.5646	0.034	16.748	0.000	0.498	0.631

Omnibus:	1.694	**Durbin-Watson:**	2.295
Prob(Omnibus):	0.429	**Jarque-Bera (JB):**	1.283
Skew:	0.269	**Prob(JB):**	0.526
Kurtosis:	3.135	**Cond. No.**	6.71

R: Linear Regression via lm()

example.lm ()

Output Window

```
> colnames(dataEx1)=c("time", "temperature")

>  lm.temp <- lm(temperature ~ time, data=as.

>  summary(lm.temp)

Call:
lm(formula = temperature ~ time, data = as.data.frame(dataEx1))

Residuals:
         1          2          3          4          5
-1.490e-16 -9.000e-01  1.200e+00  3.000e-01 -6.000e-01

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.100      0.995   0.101  0.92628
time           1.900      0.300   6.333  0.00796 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9487 on 3 degrees of freedom
Multiple R-squared: 0.9304, Adjusted R-squared: 0.9072
F-statistic: 40.11 on 1 and 3 DF,  p-value: 0.00796
```

Pay attention to

1. **Residual standard error**
2. **Or Deviance (SSE),**
3. **And variable significance**

Residuals = ($WX^i - y^i$)

Variable significance

Residual standard error

> deviance(lm.temp)

[1] 2.7

Residual standard error =σ=Sqrt(deviance/m-n-1)

$$\sigma = \sqrt{\left(\frac{1}{m-n-1}\right)\sum_{i=1}^{m}\left(WX^i - y^i\right)^2} = \sqrt{1/3 * 2.7}$$

residualStandardError=sqrt((t(lm.temp$residuals) %*% lm.temp$residuals)/3)

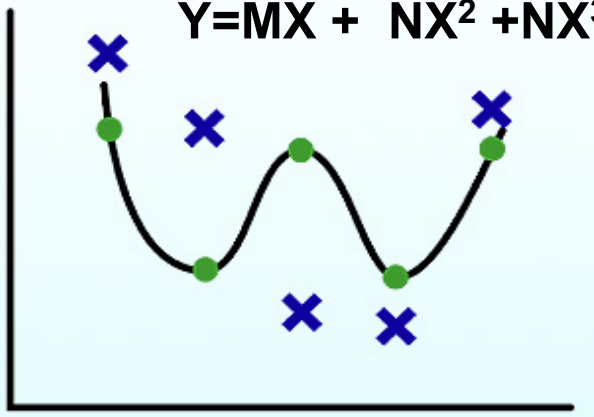
Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

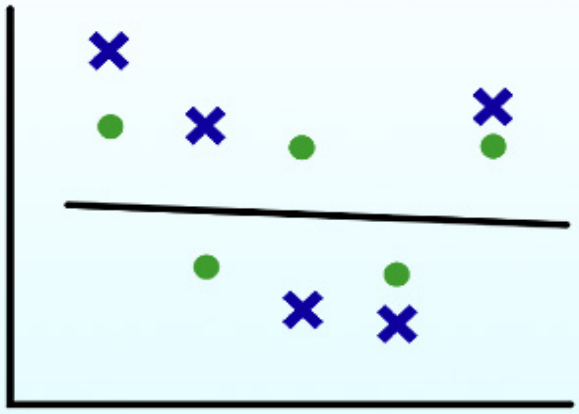
Bias-Variance Tradeoff in Model Selection in Simple Problem

Rote learning (learning by heart)

$Y=MX + NX^2 + NX^3 + C$



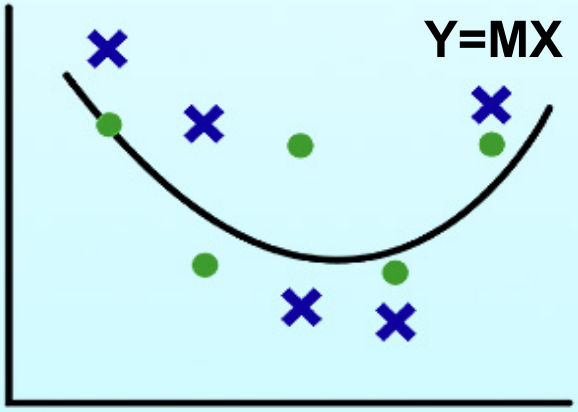
(a) High variance/low bias.
4th-order polynomial ($p = 5$).



$Y=MX + C$

(b) Low variance/high bias.
1st-order polynomial ($p = 2$).

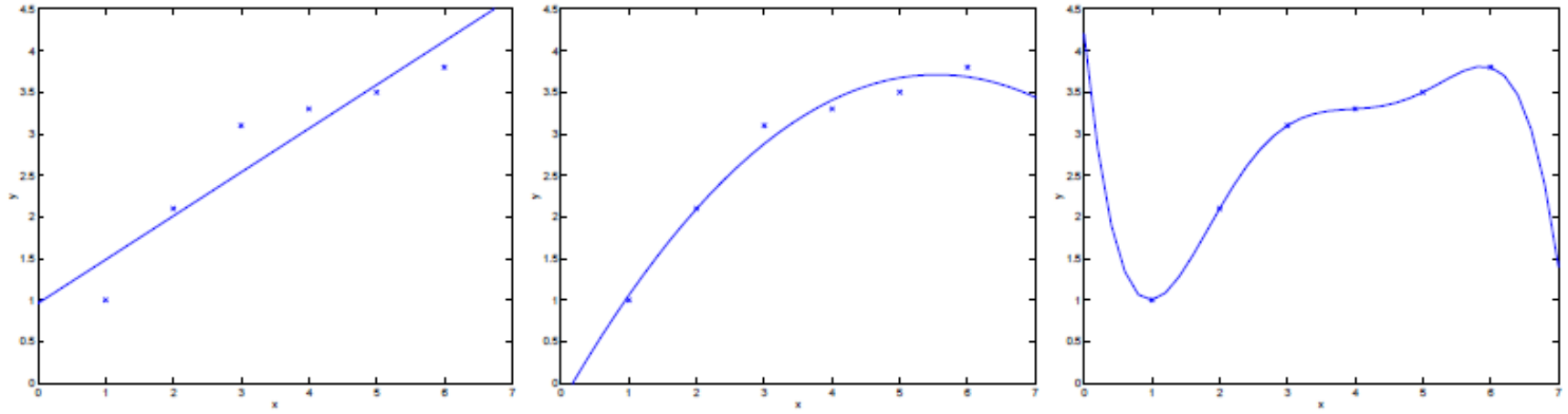
$Y=MX + NX^2 + C$



(c) Balanced variance & bias.
Minimum MSE.
2nd-order polynomial ($p = 3$).

- Data points for fitting
- Typical new data points

Overfitting versus Underfitting



Instead, if we had added an extra feature x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data. (See middle figure) Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5-th order polynomial $y = \sum_{j=0}^{5} \theta_j x^j$. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**.

[Ng, 2008 Stanford]

Bias-Variance Tradeoff

- **In statistics and machine learning, the bias–variance tradeoff (or dilemma) is the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithms from generalizing beyond their training set**
  - The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
  - The variance is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).
- **Applies to all supervised learning algos**

How good is your learnt model?

- **To assess the effectiveness of a machine learnt model, we want to know the expectation of the MSE (assume a regression problem domain) if we test the model on arbitrarily many test points drawn from the unknown function.**

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

In Simulated world we know f(X)

- the data set for training

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary **Linear regression** to approximate the function f by

$$y = g(\mathbf{x}, \mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

Explore the expectation of the MSE

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

Perfect world

- the data set for training

Observed

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary neural network to approximate the function f by

t is the target (or observed) and y is prediction

$$y = g(\mathbf{x}, \mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

To assess the effectiveness of the network, we want to know the expectation of the MSE if we test the network on arbitrarily many test points drawn from the unknown function.

$$E\{MSE\} = E\left\{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2\right\} = \frac{1}{N}\sum_{i=1}^{N}E\left\{(t_i - y_i)^2\right\}$$

Bias-variance decomposition (2)

- Putting everything together, we have:

$$
\begin{aligned}
E_P\left[(y - h(\mathbf{x}))^2\right] &= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \\
&\quad \bar{h}(\mathbf{x})^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) + f(\mathbf{x})^2 + \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \\
&= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \qquad \text{(variance)} \\
&\quad (h(\mathbf{x}) - f(\mathbf{x}))^2 + \qquad \text{(bias)}^2 \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \qquad \text{(noise)} \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + E_P[\varepsilon^2] \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + \sigma^2
\end{aligned}
$$

$h_D(x^*)$ model prediction (assume 20 training datesets)
h(x*) Average model prediction
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

- Expected prediction error = Variance + Bias^2 + Noise^2

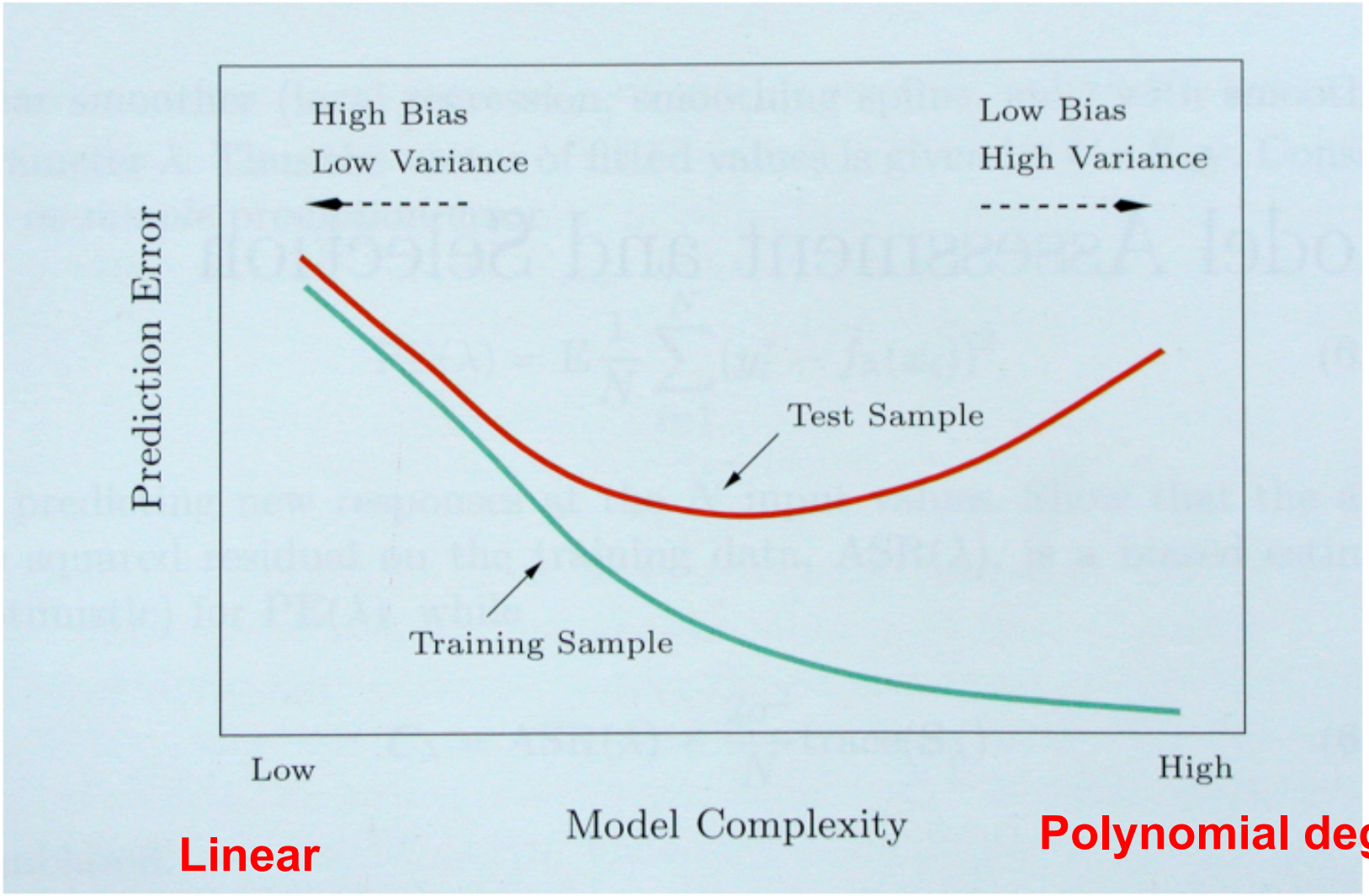
bias–variance decomposition

- **The bias–variance decomposition is a way of analyzing a learning algorithm's expected generalization error with respect to a particular problem as a sum of three terms:**
  - the bias
  - variance,
  - and a quantity called the irreducible error, resulting from noise in the problem itself.
- **This tradeoff applies to all forms of supervised learning: classification, regression (function fitting), and structured output learning.**

bias–variance tradeoff in supervised learning

- **Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data.**
- **Unfortunately, it is typically impossible to do both simultaneously.**
- **High-variance (ROTE Learning)**
  - High-variance learning methods may be able to represent their training set well, but are at risk of overfitting to noisy or unrepresentative training data.
- **High bias**
  - In contrast, algorithms with high bias typically produce simpler models that don't tend to overfit, but may underfit their training data, failing to capture important regularities.

Bias/Variance Tradeoff



Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

Optimum Model: Validation Error

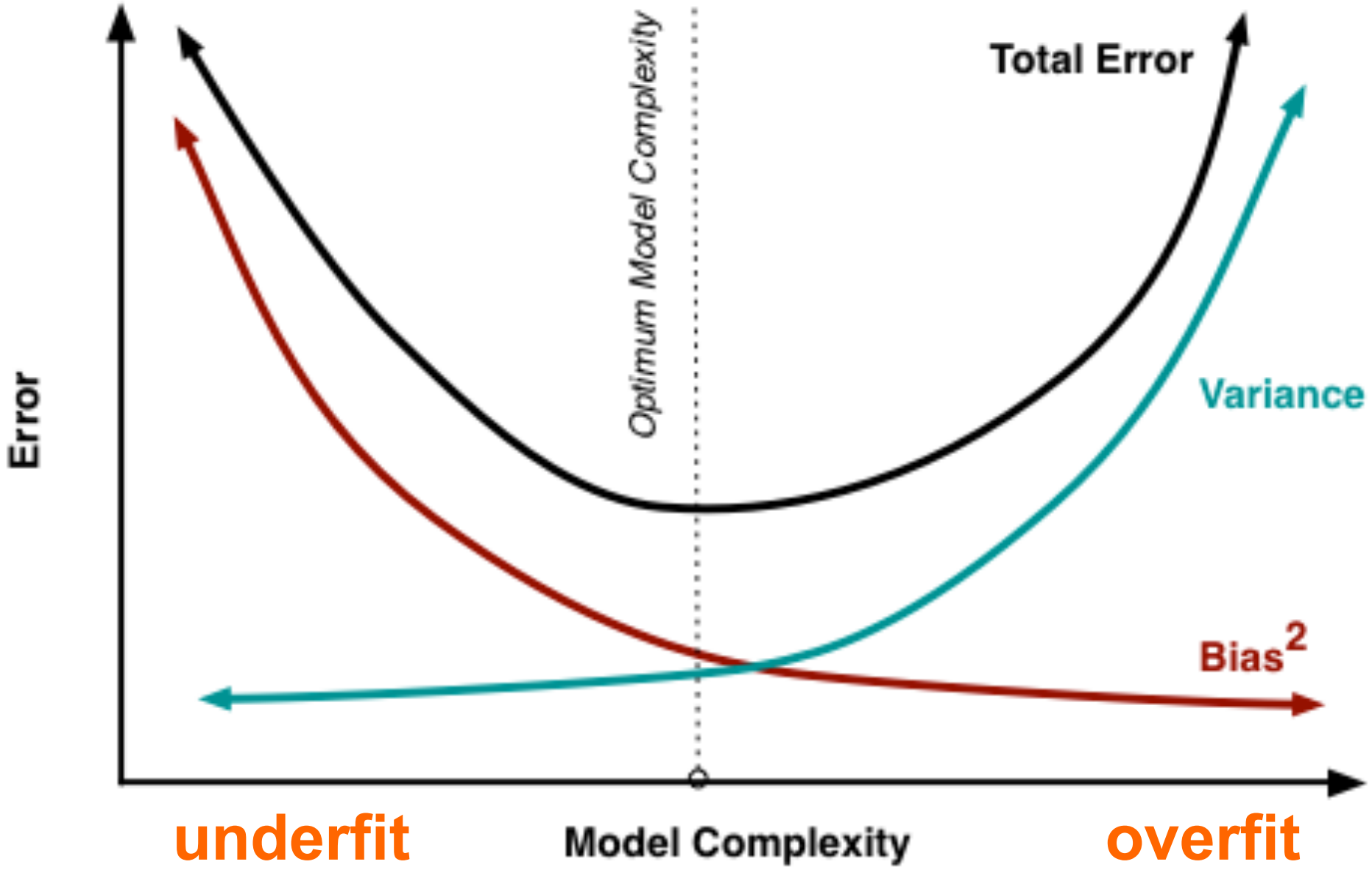


image credit: scott.fortmann-roe.com

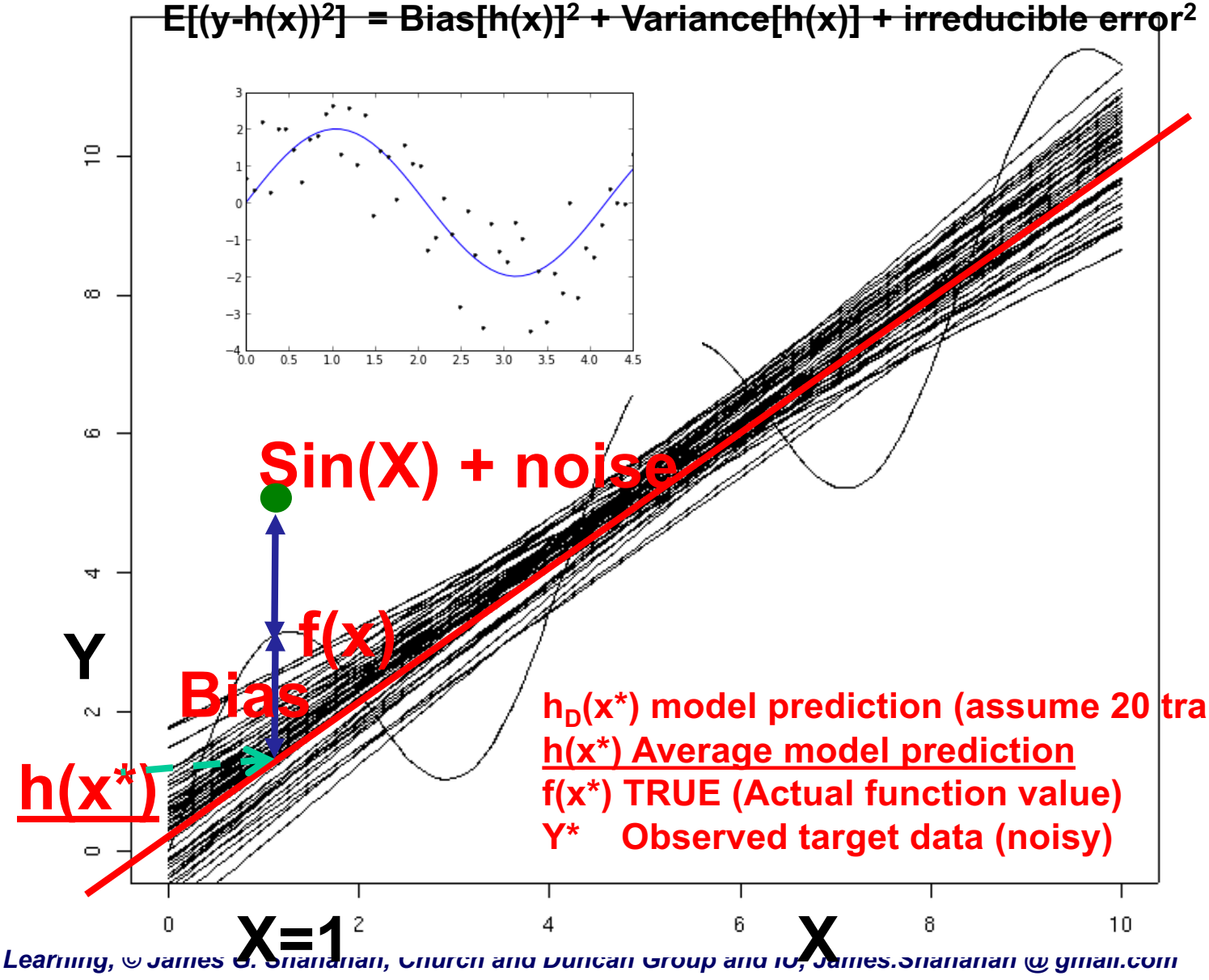
Estimating Bias-Variance: 2 Cases

- **Case 1: Simulated world**
  - Estimating Bias and Variance in a simulated world where we know the target function
- **Case 2: Real world**
  - Estimating Bias and Variance in the real world where we do NOT know the target function

50 linear models (using different samples)

$E(SE) = f(\text{Variance}, \text{bias}^2 + \text{irreducible error}^2)$

$E[(y-h(x))^2] = \text{Bias}[h(x)]^2 + \text{Variance}[h(x)] + \text{irreducible error}^2$



Optimum Model: Validation Error

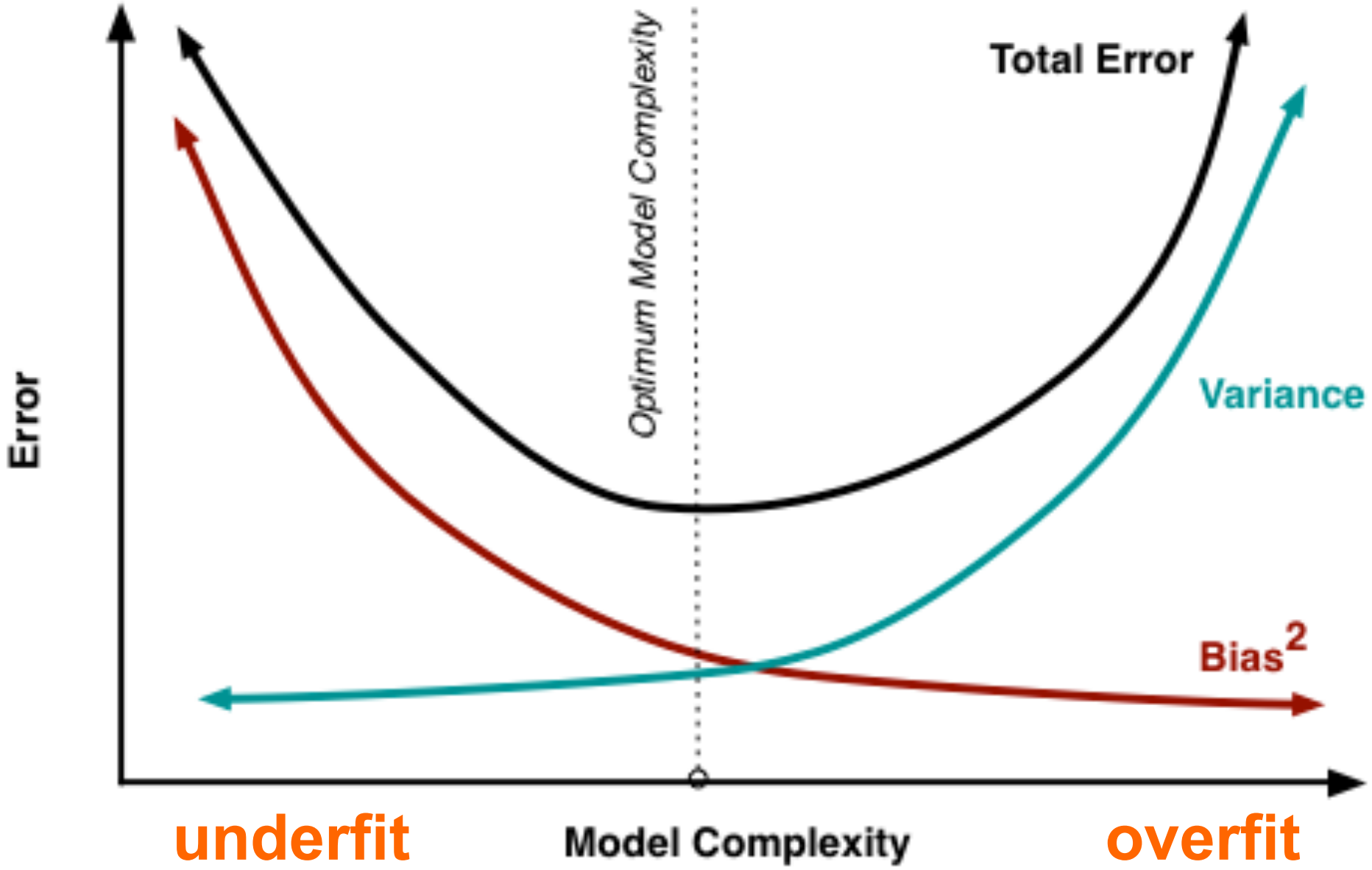


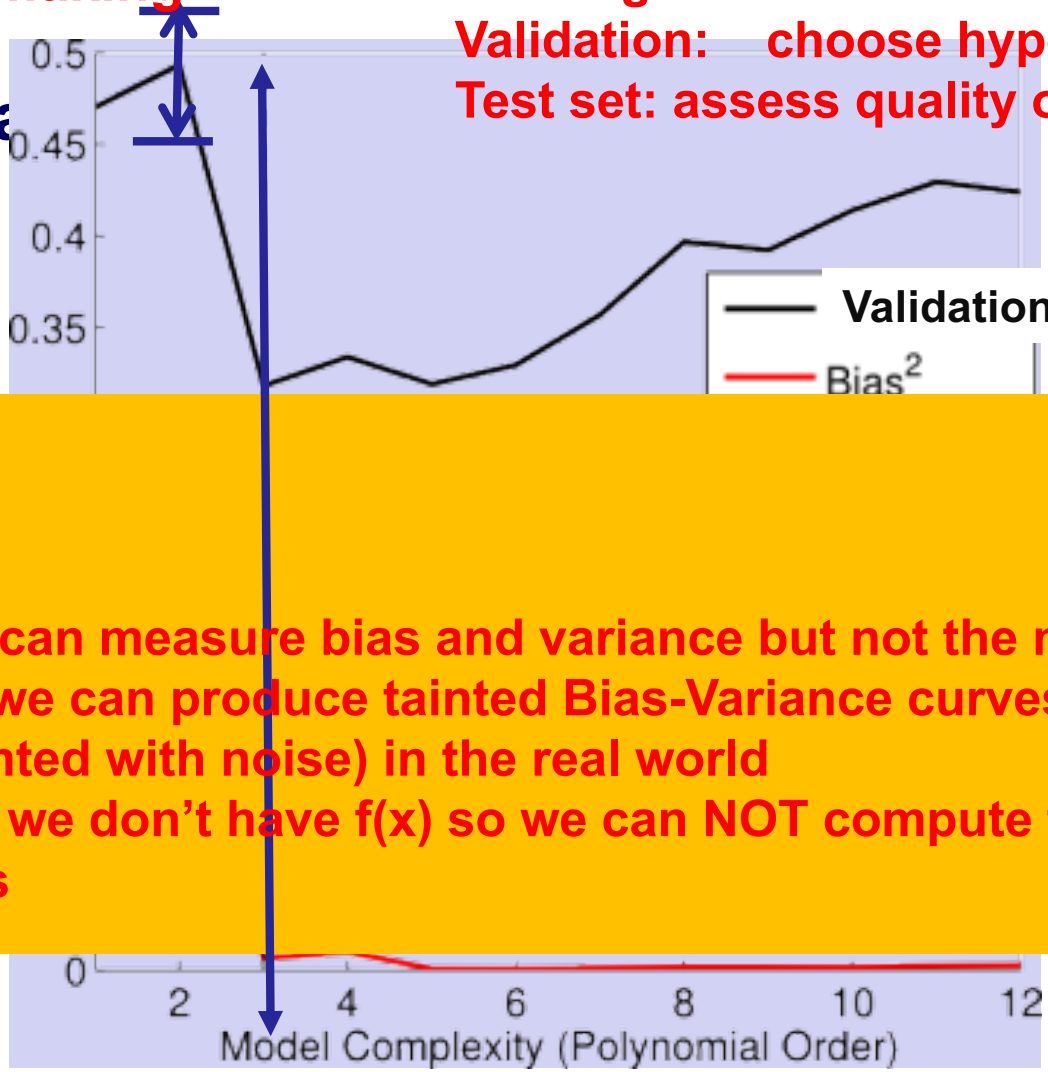
image credit: scott.fortmann-roe.com

Real World

Use Cross fold validation for model setup to get better estimate of the MSE
We NEVER use the TEST set for decision making

Training set: train model
Validation: choose hyperparameters/model
Test set: assess quality of model

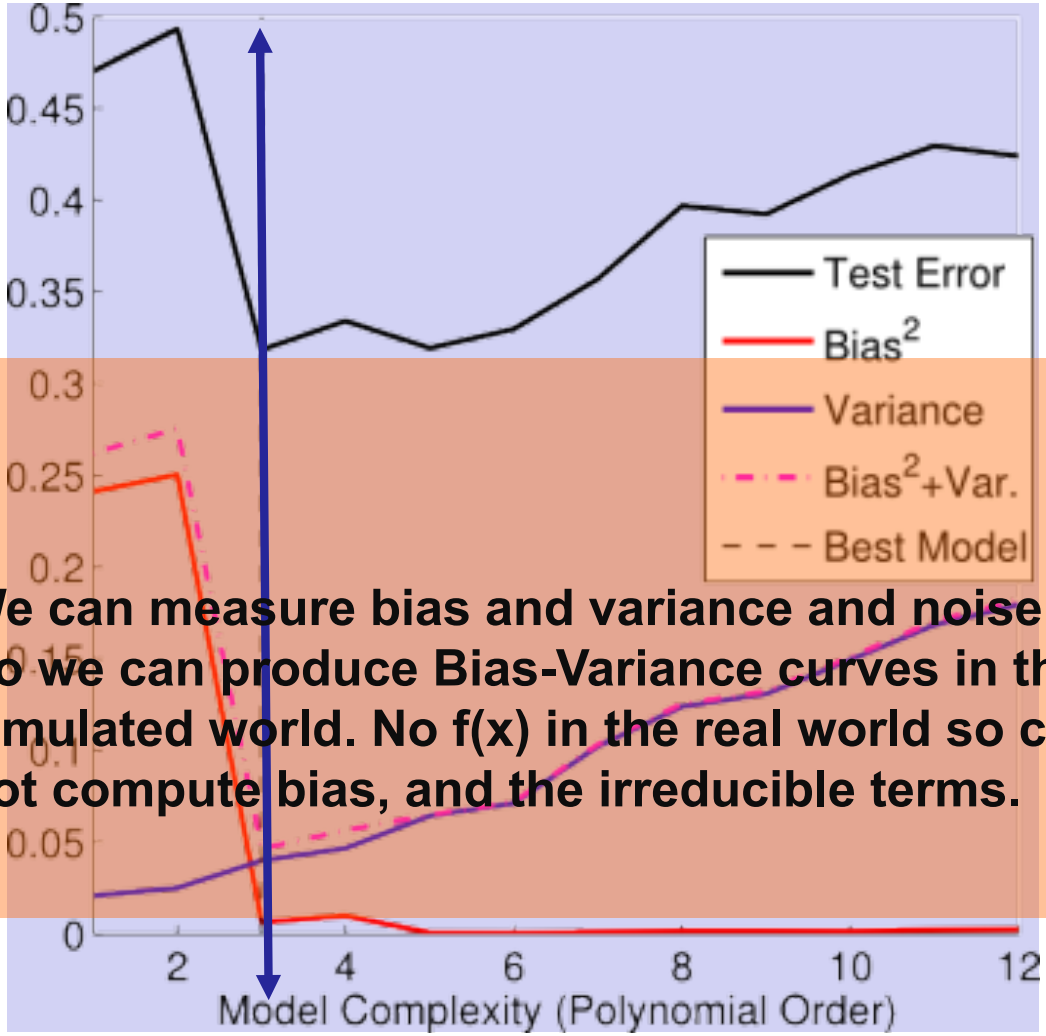
- .. Va



We can measure bias and variance but not the noise
So we can produce tainted Bias-Variance curves (tainted with noise) in the real world
But we don't have f(x) so we can NOT compute the bias

Simulated world: we know f(x)

- ..



We can measure bias and variance and noise So we can produce Bias-Variance curves in the simulated world. No f(x) in the real world so can not compute bias, and the irreducible terms.

Measuring Bias and Variance in Practice

- **Measuring Bias and Variance in Practice**
- **There is no true function available so improvise and estimate bias and variance as follows:**

- **Bootstrapping**
  - Get multiple bootstrap replicates (see next slides for details)
  - Assume 1000 datapoints; sample with replacement 1000 time to generate a bootstrap sample

- **Alternative to boostrapping**
  - If multiple values of y for the same X value
  - Or bucket X values

Measuring Bias and Variance

- In practice (unlike in theory), we have only ONE training set S.
- We can simulate multiple training sets by <u>bootstrap replicates</u>
  - $S' = \{\mathbf{x} \mid \mathbf{x}$ is drawn at random with replacement from $S\}$ and $|S'| = |S|$.

Expected prediction error* = *estimator variance* + *squared estimator bias* + *noise

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

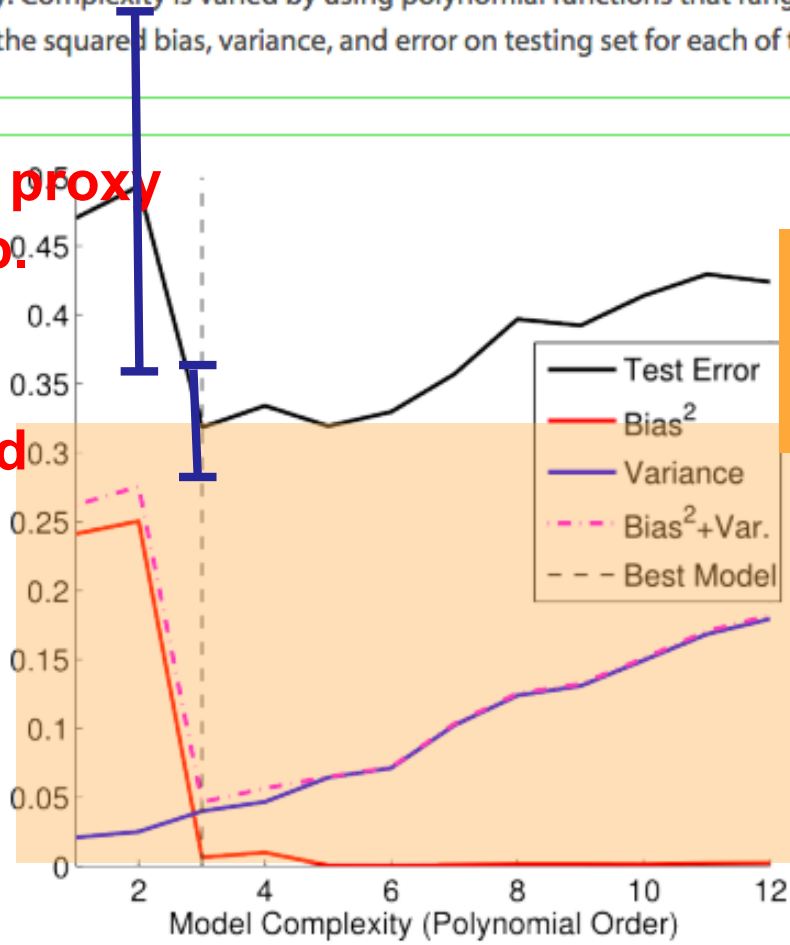
Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Black curve is a good proxy
Bias-variance decomp.

Cross fold validation
Gives us mean and std

Assume TRUE function is available



Black curve MSE is what we will have in practice

Expected prediction error* = *estimator variance* + *squared estimator bias* + *noise

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

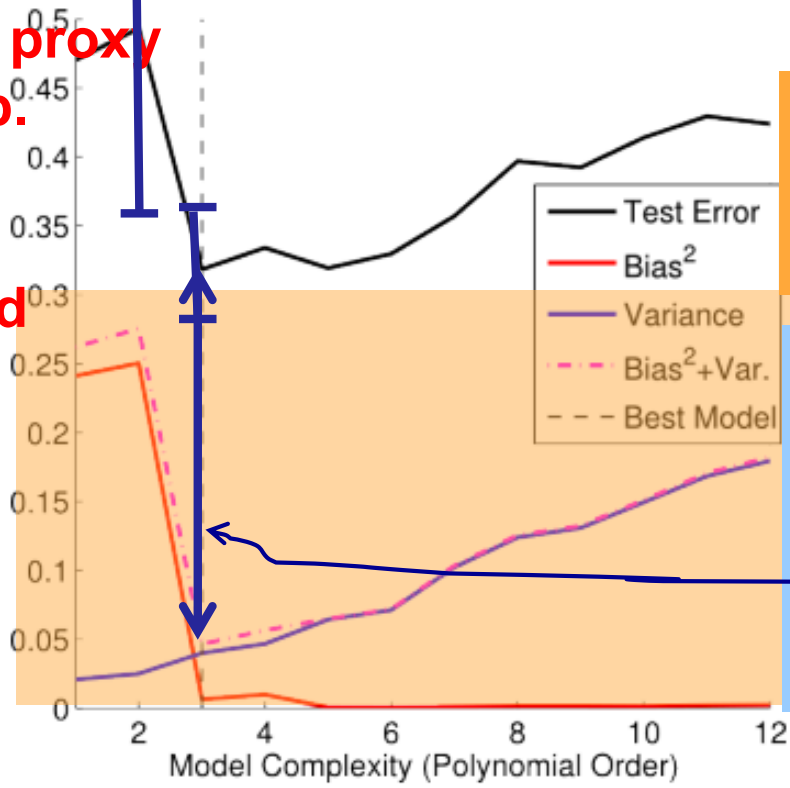
Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Black curve is a good proxy Bias-variance decomp.

Cross fold validation Gives us mean and std

Assume TRUE function is available



Black curve MSE is what we will have in practice

noise term (σ^2) in Bias-Variance equation the vertical displacement between the black curve and dashed magenta curve.

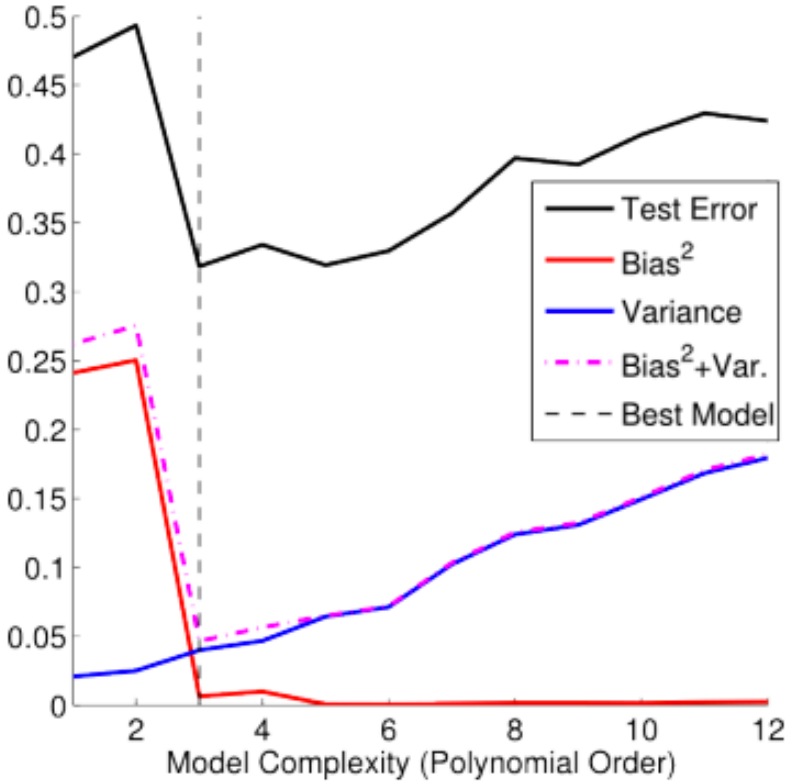
In this example, we highlight the best estimator in terms of prediction error on the testing set (black curve) with a dashed black vertical line. The best estimator corresponds to a polynomial model of order of N=3.

(most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

In Practice It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data), i.e., Black curve MSE is what we will have in practice

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

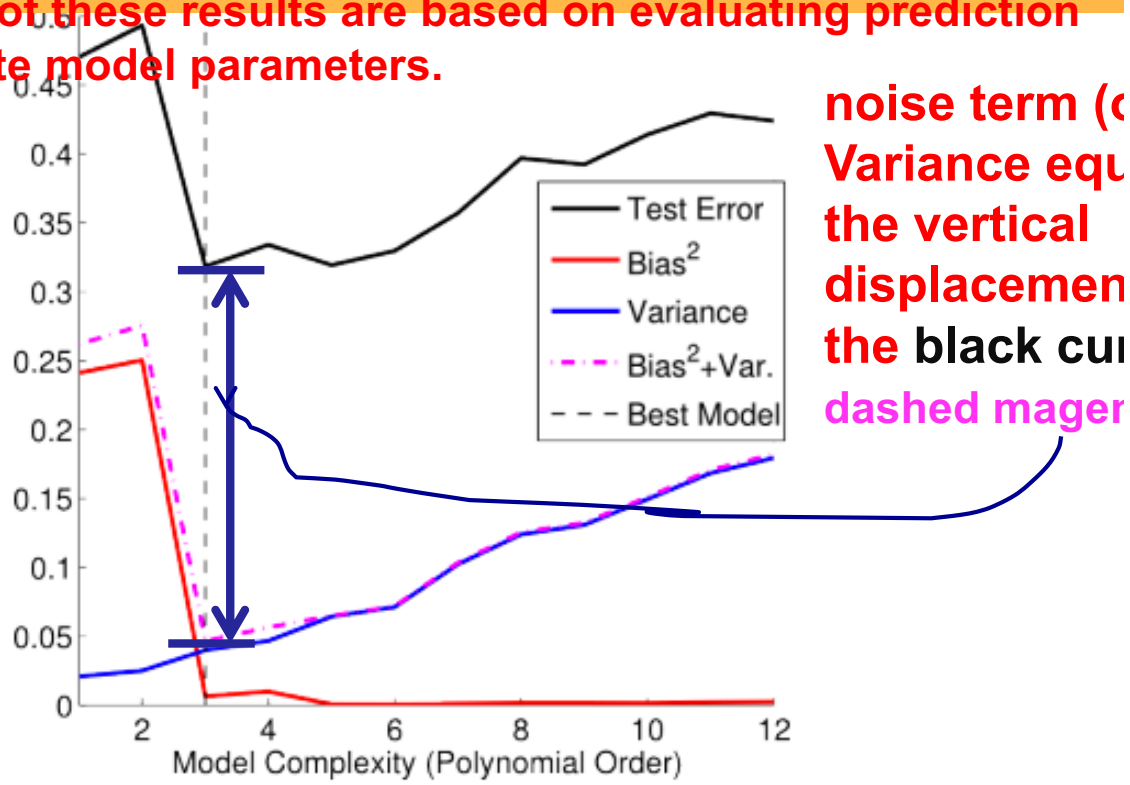


In this example, we highlight the best estimator in terms of prediction error on the testing set (black curve) with a dashed black vertical line. The best estimator corresponds to a polynomial model of order of N=3.
Notice that the vertical black line is located where function defined by the sum of the squared bias and variance (dashed magenta curve) is also at a minimum.
Notice also how the sum of the squared bias and variance also has the same shape as curve defined by the prediction error on the testing set. This exemplifies how the error on novel data can be used as a proxy for determining the best estimator from a candidate set based on squared bias and variance. The noise term in Bias-Variance equation is also represented in the figure by the vertical displacement between the **black curve** and dashed magenta curve.

It is very important to point out that all of these results are based on evaluating prediction error on novel data, not used to estimate model parameters.

In Practice It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data):

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$



noise term (
Variance equ
the vertical
displacemen
the black cu
dashed mage

Approximate expected prediction error E(MSE) with mean squared error over data pairs (novel test data)

Expected prediction error** = **estimator variance** + **squared estimator bias** + **noise

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

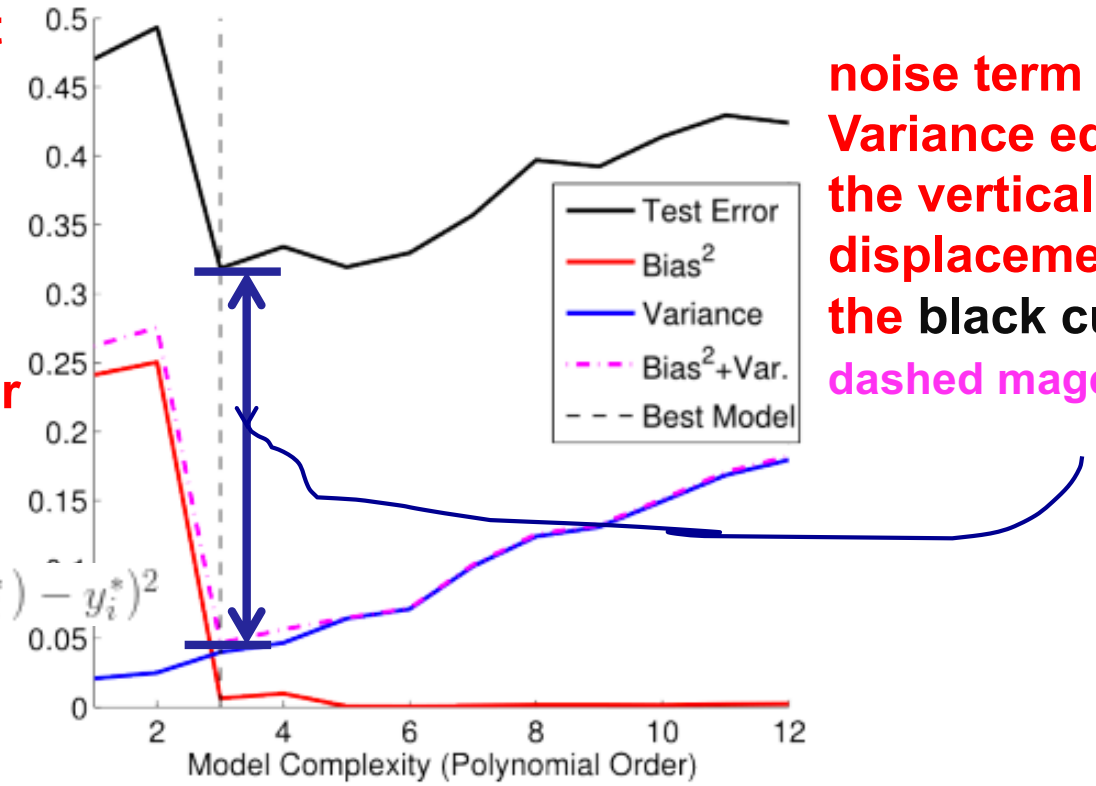
- $\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$

Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

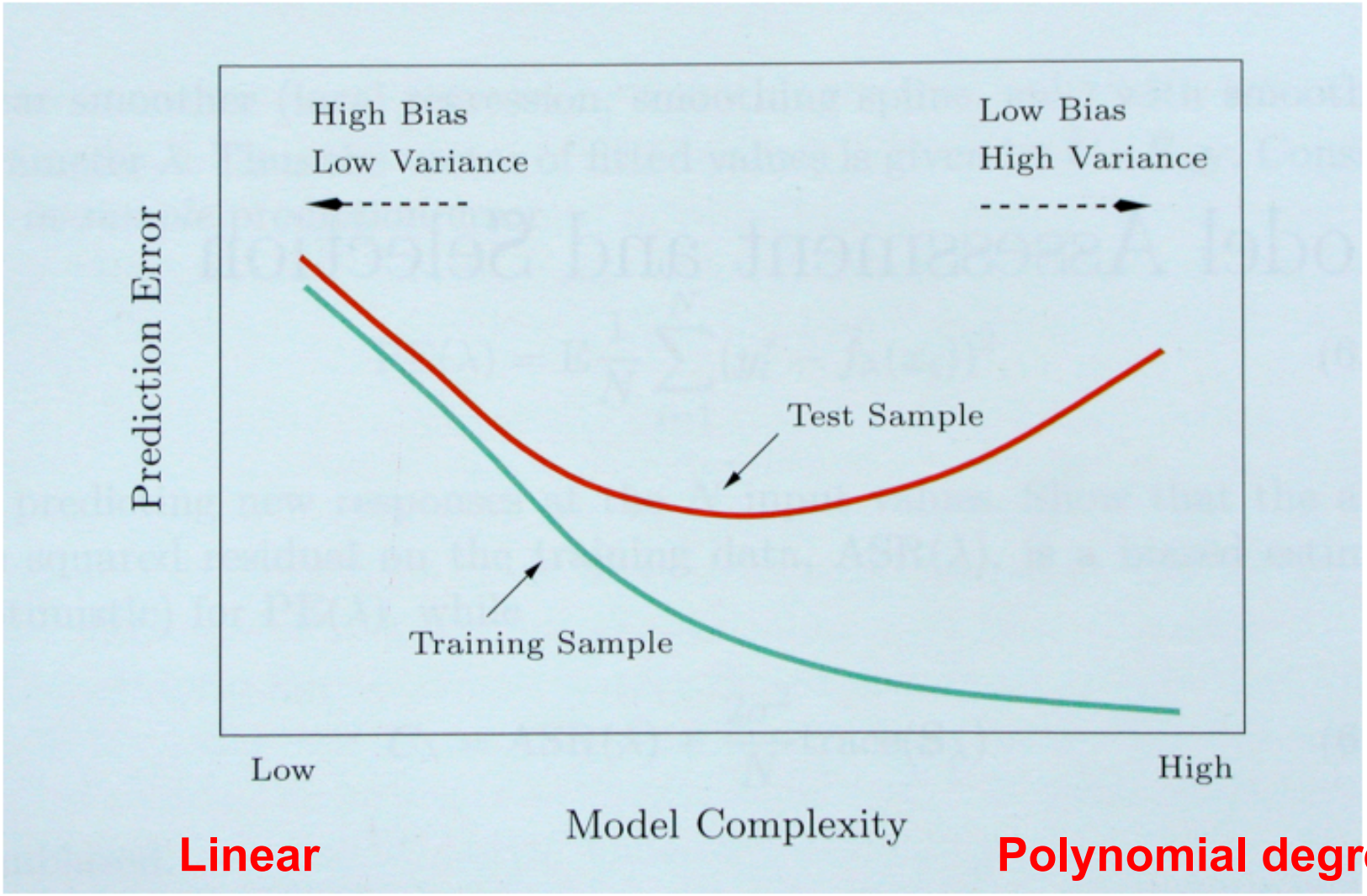
In Practice It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data):

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$



noise term in Bias-Variance equation the vertical displacement between the black curve and dashed magenta curve.

Bias/Variance Tradeoff



Linear **Polynomial degree 12**

Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Linear Model Selection and Regularization

- **For more details**
  - https://www.dropbox.com/s/q96xyy17xy73bkn/model_selection-ISRL.pdf?dl=0

Three classes of methods

- *Subset Selection*. We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.
- *Shrinkage*. We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as *regularization*) has the effect of reducing variance and can also perform variable selection.
- *Dimension Reduction*. We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different *linear combinations*, or *projections*, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

Problem: Non-invertibility in Closed form

Most common causes for the non-invertibility of $X'X$:

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

- **Redundant features** (linearly dependent).
  - E.g. x_1 = size in feet^2

    x_2= size in m^2

Concretely, let's say we want to predict housing prices. If x_1 is the size of the house in square feet and x_2 is the size of the house in square meters, then you will have a linear dependence between these two features, because $1m = 3.28\ feet$, they will always satisfy the constraint:

$$x_1 = (3.28)^2 x_2$$

If we will remove one of these features, the non-invertibility problem should be solved.

- **Too many features** (e.g. $m \leq n$, so if we are trying to run the learning algorithm having more features then observations in the training set).

For example, let's imagine we have m = 10 training examples and n = 100 features, then we are trying to fit a parameter back to θ which is n+1 dimensional. So, we would be trying to fit 101 parameters from just 10 training observations, which obviously is not a good idea.

Problem: Non-invertibility in Closed form

Most common causes for the non-invertibility of $X'X$:

- **Redundant features** (linearly dependent).
  - E.g. x_1 = size in feet^2

    x_2= size in m^2

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

If #Examples < # features then $(X^TX)^{-1}$ is non-invertible or singular (matrix will be degenerate)
If m == n then $(X^TX)^{-1}$ may be non-invertible.
Pinv (pseudo-inverse)

Concretely, let's say we want

of the house in square mete

3.28 feet, they will always satisfy the constraint:

$$x_1 = (3.28)^2 x_2$$

If we will remove one of these features, the non-invertibility problem should be solved.

- **Too many features** (e.g. $m \leq n$, so if we are trying to run the learning algorithm having more features then observations in the training set)

If #Examples < # features then $(X^TX + \lambda[[0, \ldots][0,1, \ldots][0,0, 1, \ldots]])^{-1}$ is invertible or nonsingular
Using Regularization takes care of any non-invertabilty
Pinv (pseudo-inverse)

For example, let's imag

parameter back to θ w

observations, which ol

Pseudo-Inverse (Pinv)

- Once can use a pseudoinverse, as long as ƛ (lambda) > 0, the matrix $(X^TX + ƛ\mathbf{I})$ is always invertible.

- pinv (X' * X) * X' * y
-

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

$$\theta = (X^TX + ƛ\mathbf{I})^{-1} * X' * y$$

$$\theta = \text{pinv}(X' * X) * X' * y$$

Problem: Non-invertibility in Closed form

Suppose $m \leq n$,

(#examples) (#features)

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

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

Problem

If #Examples < # features then $(X^TX)^{-1}$ is non-invertible or singular (matrix will be degenerate)
If m == n then $(X^TX)^{-1}$ may be non-invertible.
Pinv (pseudo-inverse)

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Notice that bias term is zero in the diagonal matrix

Solution

If #Examples < # features then $(X^TX + \lambda[[0, \ldots][0,1, \ldots][0,0, 1, \ldots]])^{-1}$ is invertible or nonsingular
Using Regularization takes care of any non-invertabilty
Pinv (pseudo-inverse)

Regularization and the normal eqn.

Normal Equation

Now let's approach regularization using the alternate method of the non-iterative normal equation.

To add in regularization, the equation is the same as our original, except that we add another term inside the parentheses:

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

where $L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}$

Notice that bias term is zero in the diagonal matrix

L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else. It should have dimension (n+1)×(n+1). Intuitively, this is the identity matrix (though we are not including x_0), multiplied with a single real number λ.

This is known as Ridge Regression closed-form solution

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Ridge Regression closed-form solution

- **Ridge regression closed form**

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- **Can be rewritten as follows:**

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Ridge regression: Pseudo inverse

- **Pseudo inverse**
  - Here is how to perform Ridge Regression with Scikit-Learn using a closed-form solution (using a matrix factorization technique by Andr.-Louis Cholesky):

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[ 1.55071465]])
```

And using Stochastic Gradient Descent:[14]

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([[ 1.13500145]])
```

Ridge regression via closed-form or GD!

- **As with Linear Regression, we can perform Ridge Regression either**
  - by computing a closed-form equation
  - or by performing Gradient Descent

Closed from solution versus SGD

- **Closed from solution using Pseudo-inverse**
  - Always gives the best solution
  - Good for small problems

- **SGD (and variants, High-Dim optimization problems)**
  - Good large high dimensional problems
  - Does not always give the best model

- **Video**
  - https://www.coursera.org/learn/machine-learning/lecture/QrMXd/regularized-linear-regression

Loss functions; a unifying view

- Loss function consists of:
  - loss term ($L(m_i(w)$, expressed in terms of the margin of each training example) and
  - regularization term (R(w) expressed as a function of the model complexity)

Loss term regularization term

$$J(w) = \sum_i L(m_i(w)) + \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

The entire loss is Twice differentiable BUT sometimes either one of these maybe not be twice differentiable everywhere

KISS Models

- **reduce overfitting is to regularize the model (i.e., to constrain it): the fewer degrees of freedom it has, the harder it will be for it to overfit the data. For example, a simple way to regularize a polynomial model is to reduce the number of polynomial degrees.**
- **For a linear model, regularization is typically achieved by constraining the weights of the model.**
- **Three different ways to constrain the weights**
  - Ridge Regression,
  - Lasso Regression,
  - and Elastic Net,

2. Ridge Regression

Via Gradient Descent (see gradient below)

The objective function (also called the cost) to be minimized is the RSS plus the sum of square of the magnitude of weights. This can be depicted mathematically as:

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ squares\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

In this case, the gradient would be:

1

$$\frac{\partial}{\partial w_j} Cost\ (W) = -2 \sum_{i=1}^{N} x_{ij} \left\{ y_i - \sum_{k=0}^{M} w_k\, x_{ik} \right\} + 2\lambda w_j$$

gradient

Again in the regularization part of gradient, only w_j remains and all other would become zero. The corresponding update rule is:

$$w_j^{t+1} = w_j^t - \eta \left[-2 \sum_{i=1}^{N} x_{ij} \left\{ y_i - \sum_{k=0}^{M} w_k * x_{ik} \right\} + 2\lambda w_j \right]$$

GD update

Ridge regression via closed-form equation

- **As with Linear Regression, we can perform Ridge Regression either**
  - by computing a closed-form equation
  - or by performing Gradient Descent

Generalized LR learning algorithms

- **Generalized LR learning algorithms with regularization**
  - Normal equation: Closed from algorithm with regularization
  - Gradient descent with regularization

Minimize f(W)

where

$$f(W) = \frac{1}{2m}\left[\sum_{n=1}^{m}\left(X^{iT}\boldsymbol{W} - y^{i}\right)^{2} + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

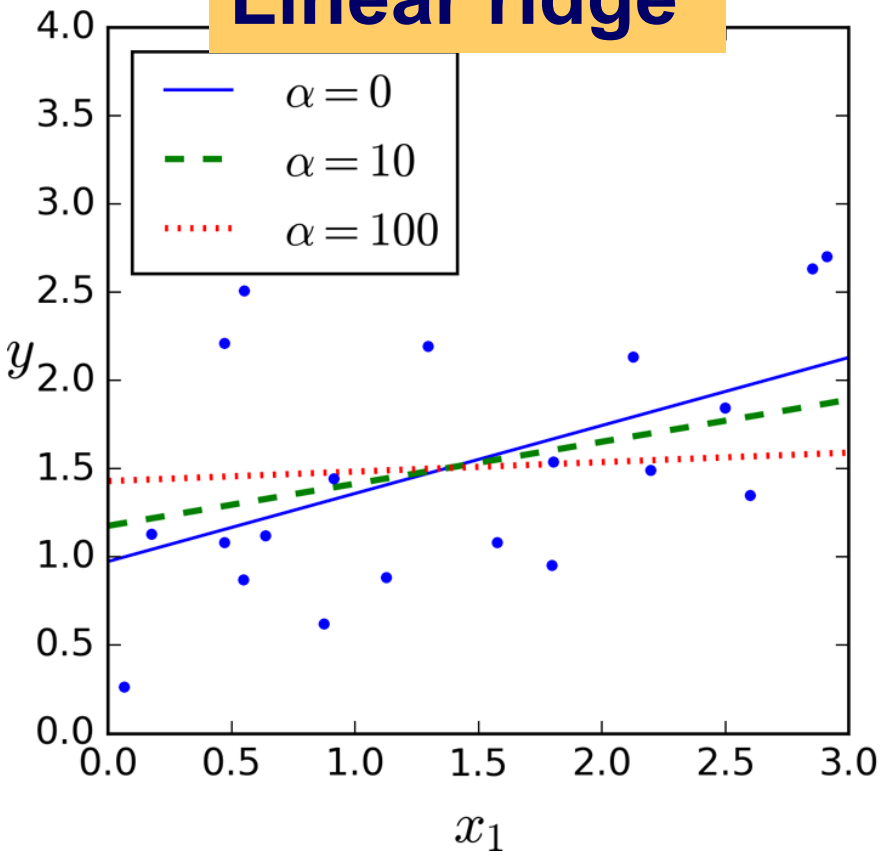
$$f(W) = \frac{1}{2m}\left[\sum_{n=1}^{m}\left(X^{iT}\boldsymbol{W} - y^{i}\right)^{2} + \boldsymbol{\alpha}\sum_{j=1}^{n}\theta_j^2\right]$$

Note ignore bias term in the regularization term (in most cases)

Pay attention to notation λ versus α

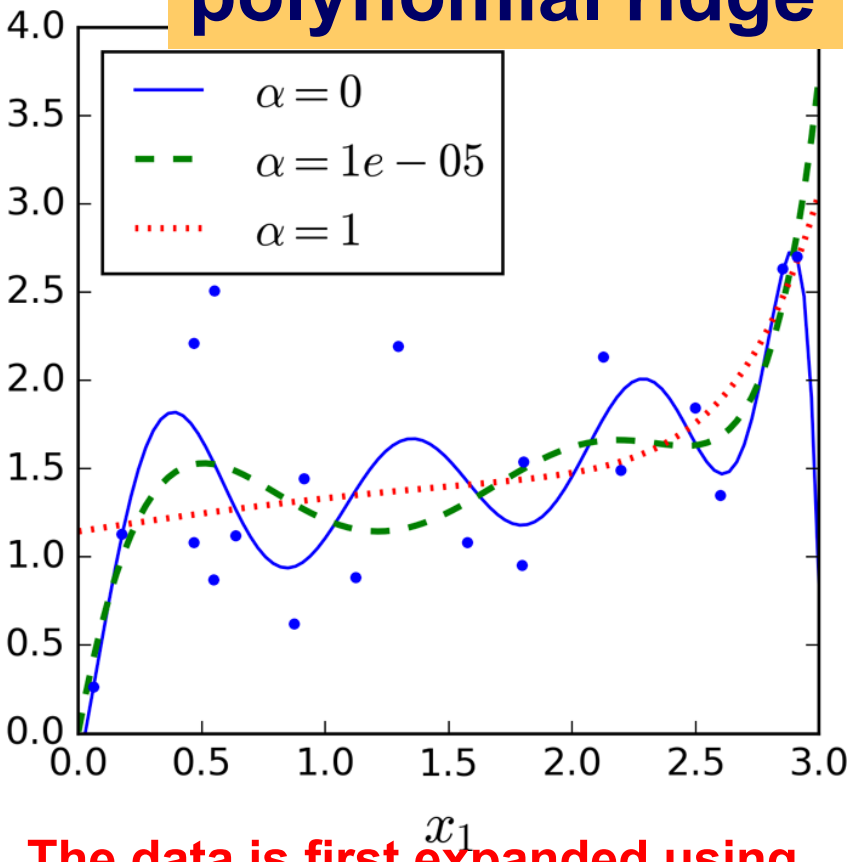
Linear ridge versus polynomial ridge

Linear ridge



Plain Ridge models are used, leading to linear predictions.

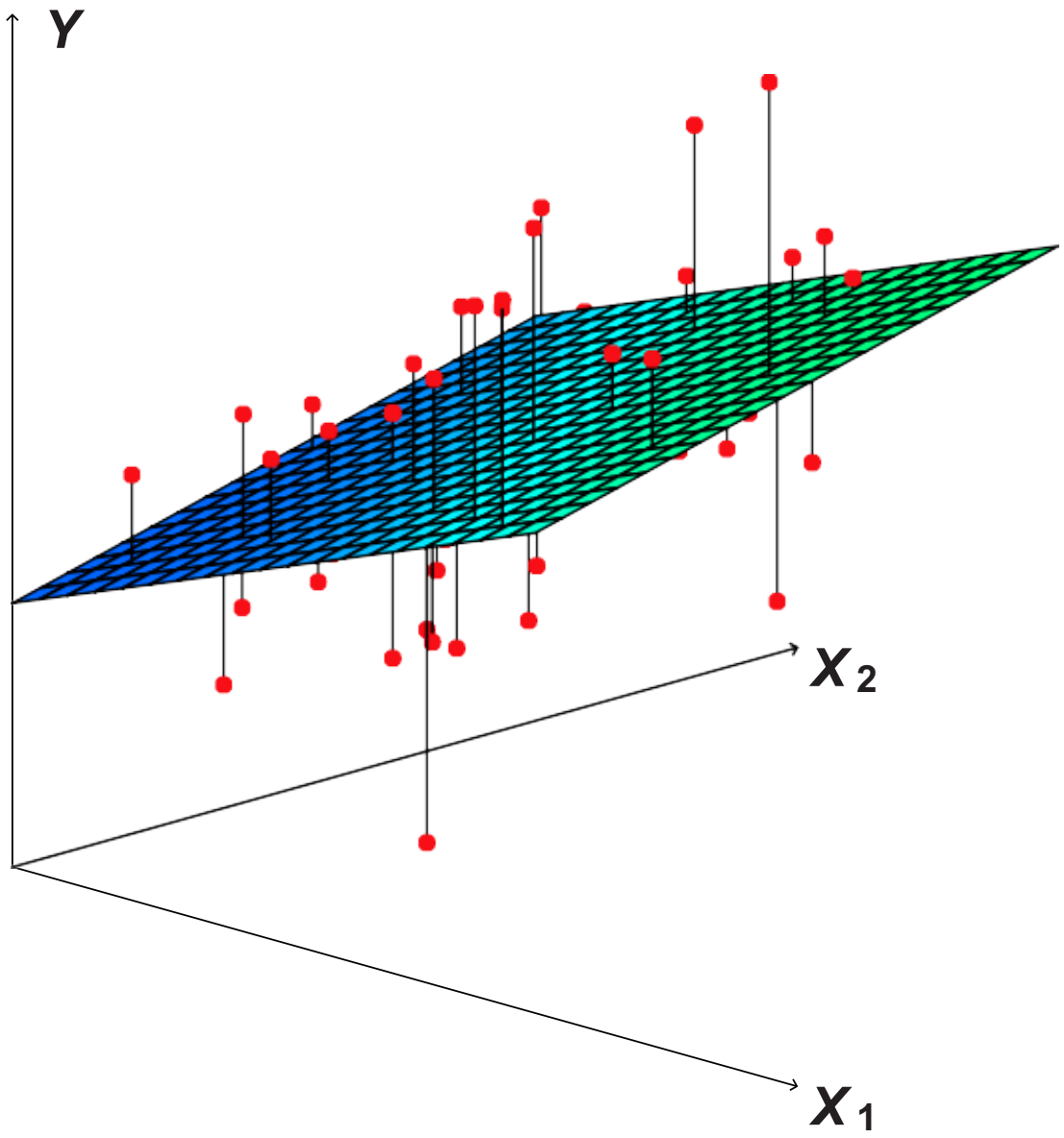
polynomial ridge



The data is first expanded using PolynomialFeatures(degree=10), then it is scaled using a StandardScaler, and finally the Ridge models are applied to the resulting features

StandardizeScaler: for each feature: zero mean and unit variance.

Learn $f(X_1, X_2)$



Small training set

X1	X2	Y
-1	1	-1.5
-0.3	-1	-1.1
1	0.1	-2.05

MSE + L2

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Train a model with no bias terms so we can compare the following heatmaps:

- **MSE**
- **Model complexity loss term**
- **Ridge (i.e., MSE + L2)**
- **LASSO (i.e., MSE + L1)**

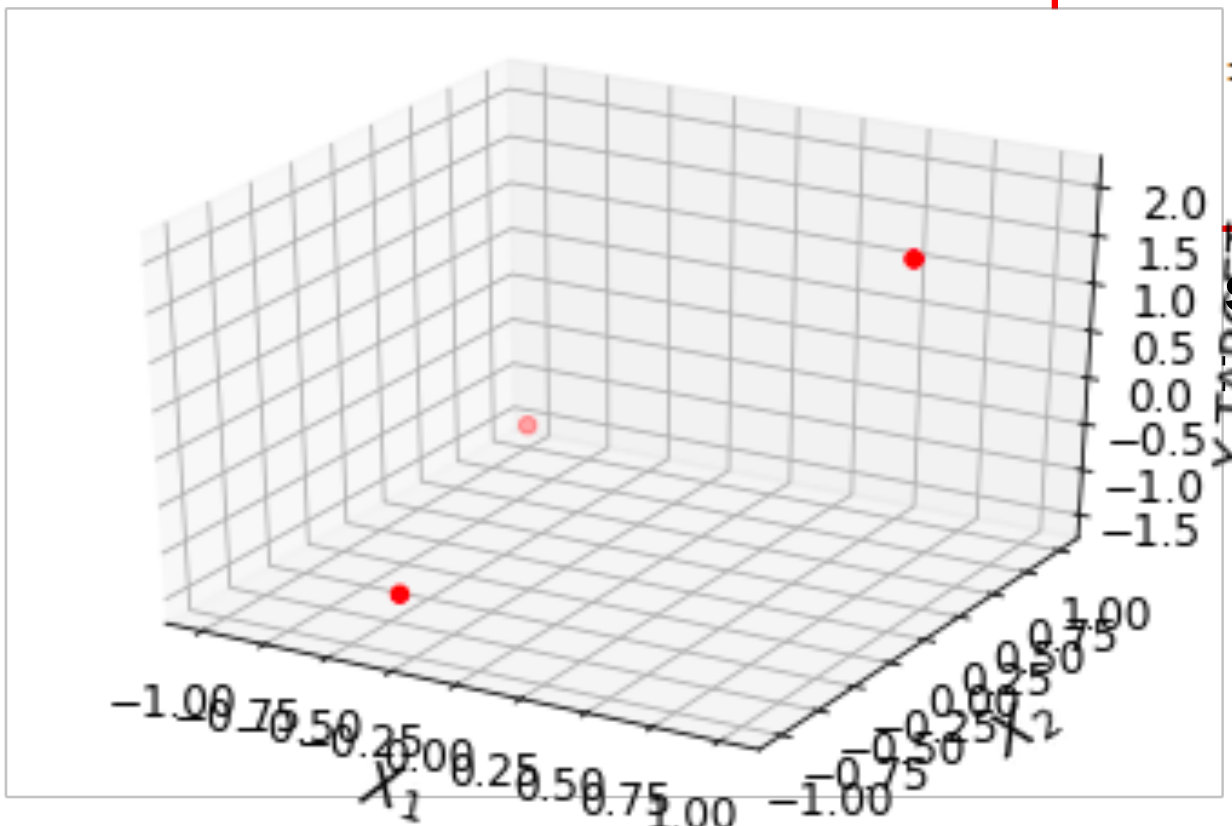
Small training set

X1	X2	Y
-1	1	-1.5
-0.3	-1	-1.1
1	0.1	-2.05

MSE + L2

$$= \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

so we can

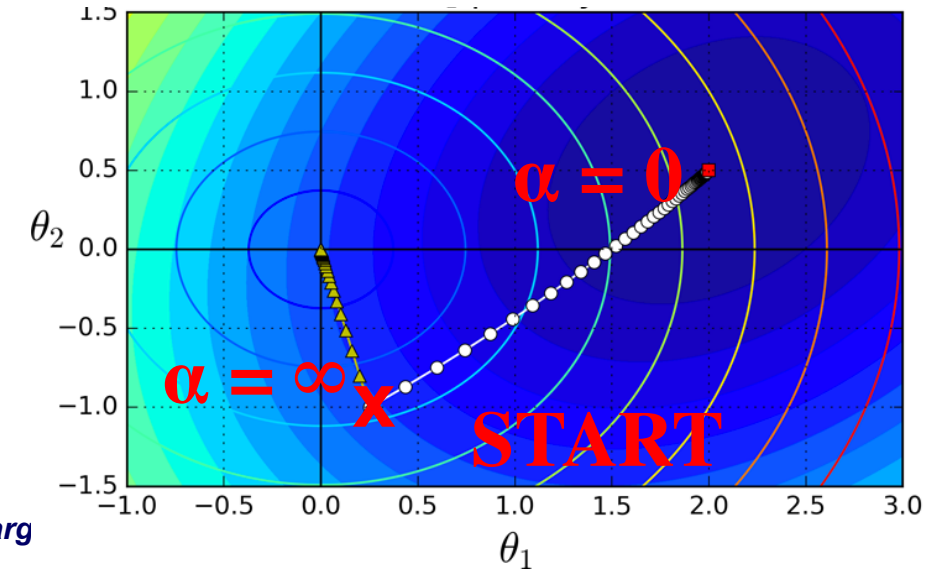


The background contours (ellipses) represent an unregularized MSE cost function (α = 0), and the white circles show the Batch Gradient Descent path with that cost function.
The foreground contours (diamonds) represent the ℓ1 penalty, and the triangles show the BGD path for this penalty only (α → ∞).

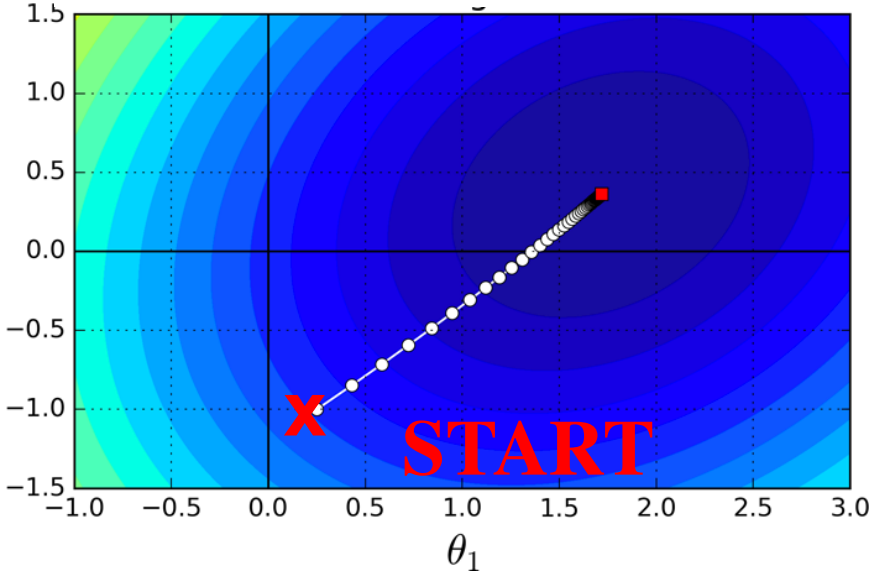
Ridge

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

L2 contours versus MSE heatmap



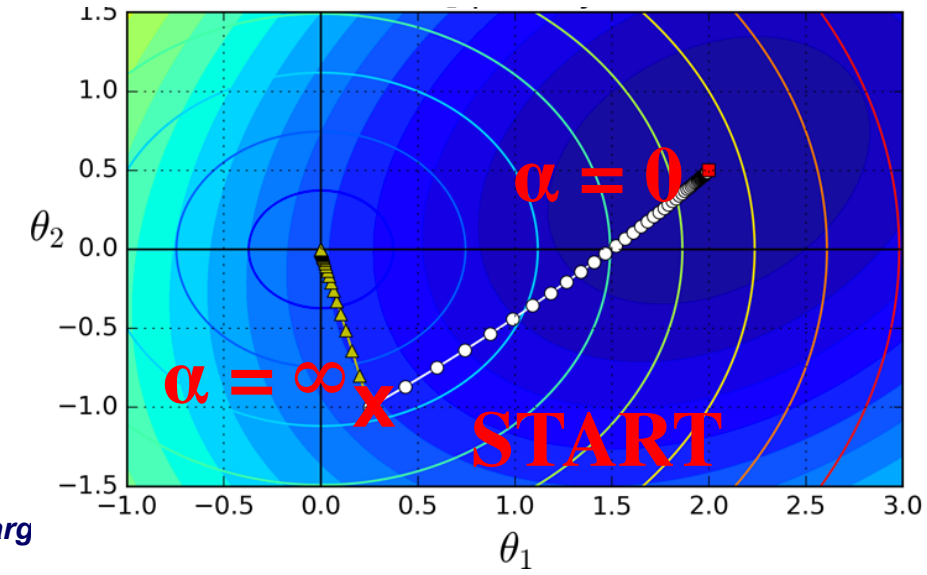
Ridge loss heatmap
Combined Loss terms with α=0.5



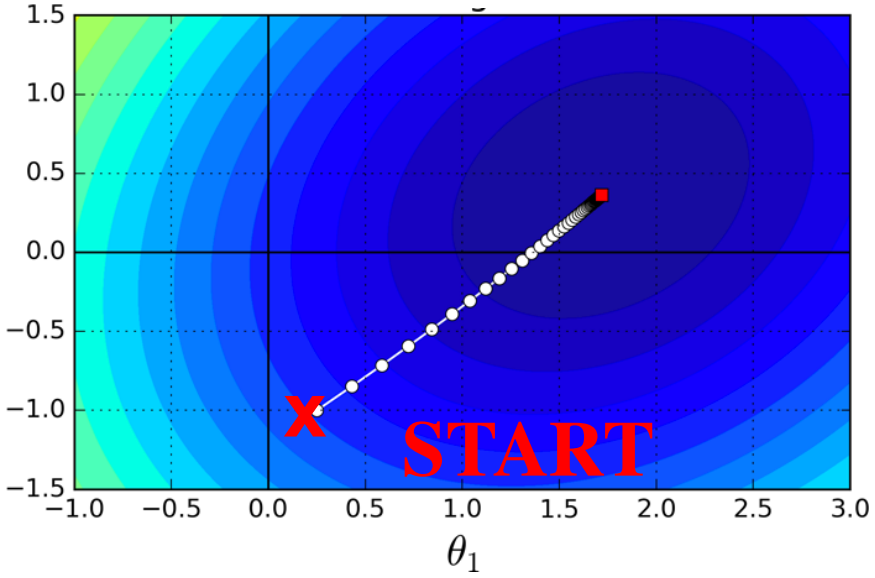
The background contours (ellipses) represent an unregularized MSE cost function (α = 0), and the white circles show the Batch Gradient Descent path with that cost function.
The foreground contours (diamonds) represent the ℓ1 penalty, and the triangles show the BGD path for this penalty only (α → ∞).

Ridge

L2 contours versus MSE heatmap



Ridge loss heatmap
Combined Loss terms



Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

LASSO

- Least Absolute Shrinkage and Selection Operator Regression (simply called Lasso Regression) is another regularized version of Linear Regression:
- just like Ridge Regression, it adds a regularization term to the cost function, but it uses the ℓ1 norm of the weight vector instead of half the square of the ℓ2 norm

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Lasso Regression

- https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

The objective function (also called the cost) to be minimized is the RSS plus the sum of absolute value of the magnitude of weights. This can be depicted mathematically as:

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Note that the bias term θ_0 is not regularized (the sum starts at i = 1, not 0).

Lasso and Ridge Regression

- https://www.analyticsvidhya.com/blog/2016/01/co

3. Lasso Regression

The objective function (also called the cost) to be minimized is the RSS plus the sum of absolute value of the magnitude of weights. This can be depicted mathematically as:

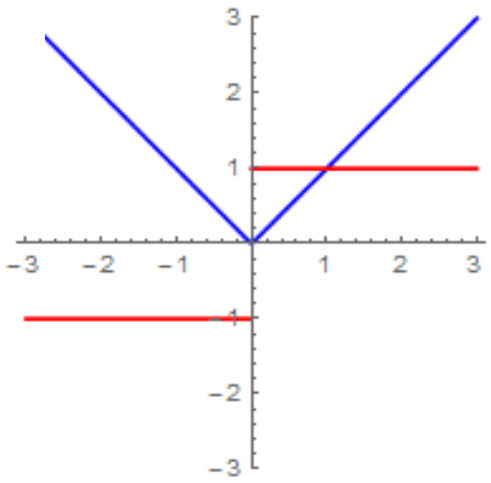
$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

θ > 1: J'(θ) =1

θ < 1: J'(θ) = -1

θ =1 : does not exist



Lasso: SubGradient

- The Lasso cost function is not differentiable at θi = 0 (for i = 1, 2, ⋯, n), but Gradient Descent still works fine if you use a subgradient vector g instead when any θi = 0.
- Equation below shows a subgradient vector equation you can use for Gradient Descent with the Lasso cost function.

$$g(\theta, J) = \nabla_\theta \mathrm{MSE}(\theta) + \alpha \begin{pmatrix} \mathrm{sign}\,(\theta_1) \\ \mathrm{sign}\,(\theta_2) \\ \vdots \\ \mathrm{sign}\,(\theta_n) \end{pmatrix} \quad \text{where} \quad \mathrm{sign}\,(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

Constrained form

It can be helpful to think of our two problems constrained form:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \|y - X\beta\|_2^2 \quad \text{subject to } \|\beta\|_2^2 \leq t$$

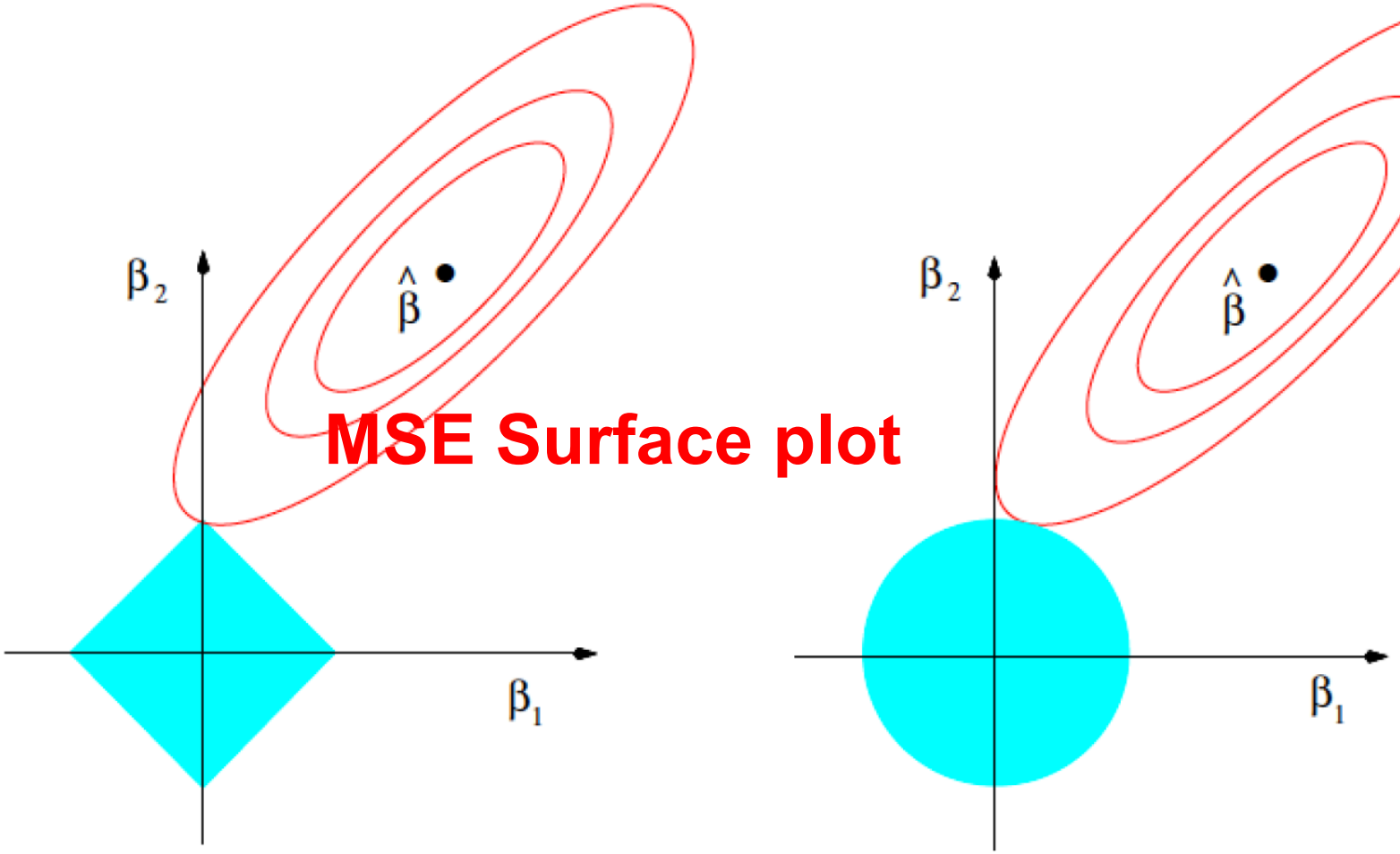
$$\hat{\beta}^{\text{lasso}} = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \|y - X\beta\|_2^2 \quad \text{subject to } \|\beta\|_1 \leq t$$

Now t is the tuning parameter (before it was λ). For any λ and corresponding solution in the previous formulation (sometimes called penalized form), there is a value of t such that the above constrained form has this same solution

In comparison, the usual linear regression estimate solves the unconstrained least squares problem; these estimates constrain the coefficient vector to lie in some geometric shape centered around the origin. This generally reduces the variance because it keeps the estimate close to zero. But which shape we choose really matters!

Why does the lasso give zero coefficients?

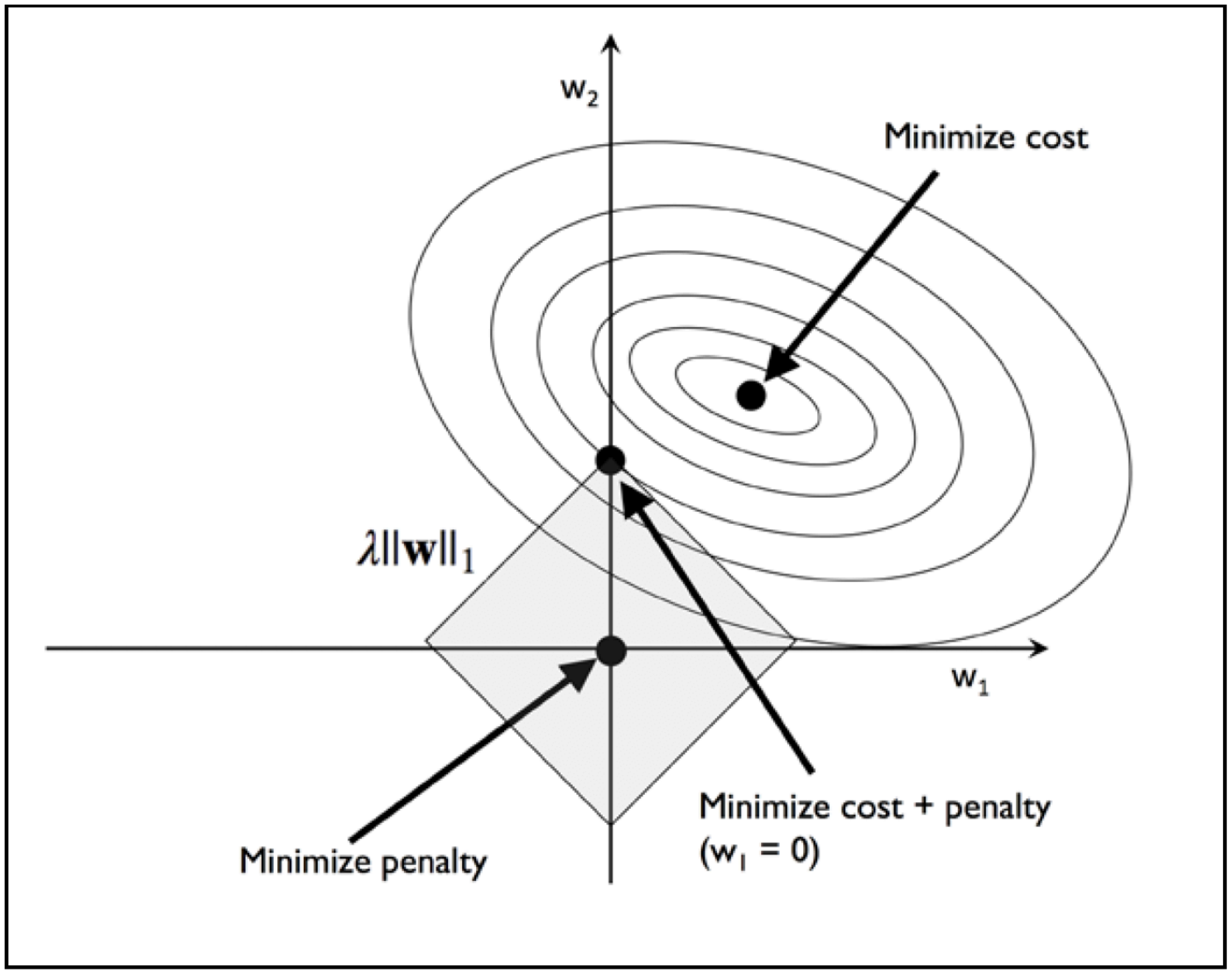
L1 contours versus MSE Surface **L2 contours versus MSE Surface**



MSE Surface plot

(From page 71 of ESL)

Lasso Regression



Linear

Poly, deg=10

Ridge

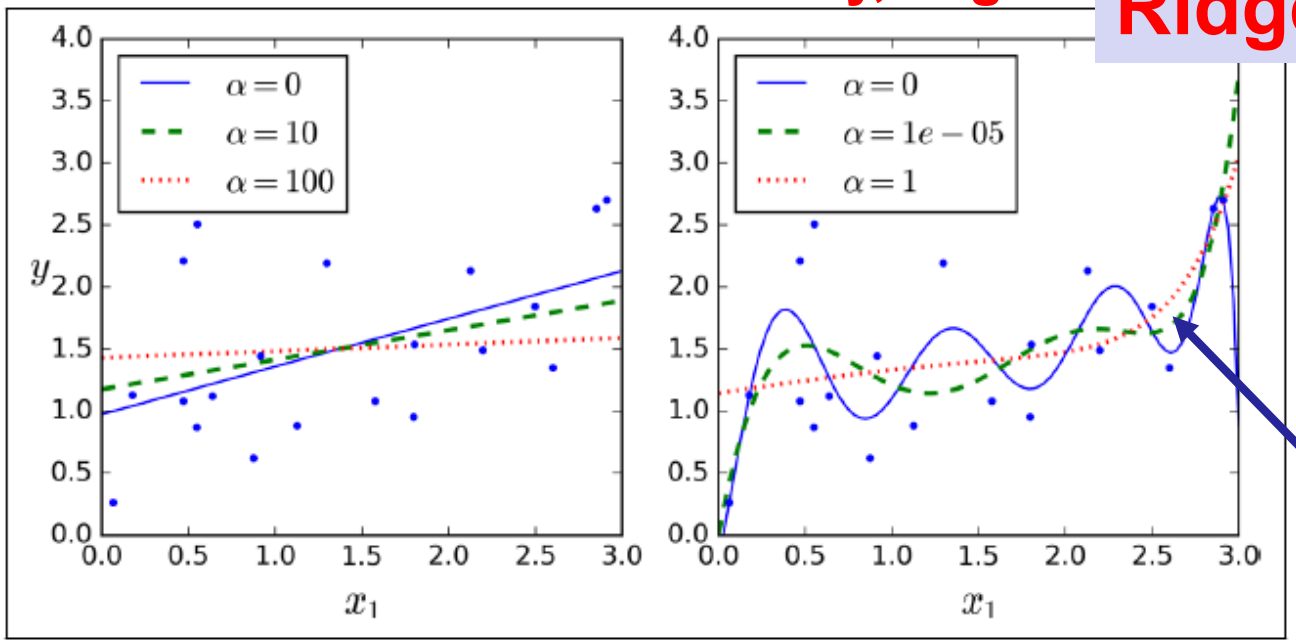


Figure 4-17. Ridge Regression

Feature selection

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Lasso

Figure 4-18 shows the same thing as Figure 4-17 but replaces Ridge models with Lasso models and uses smaller α values.

Linear

Poly, deg=10

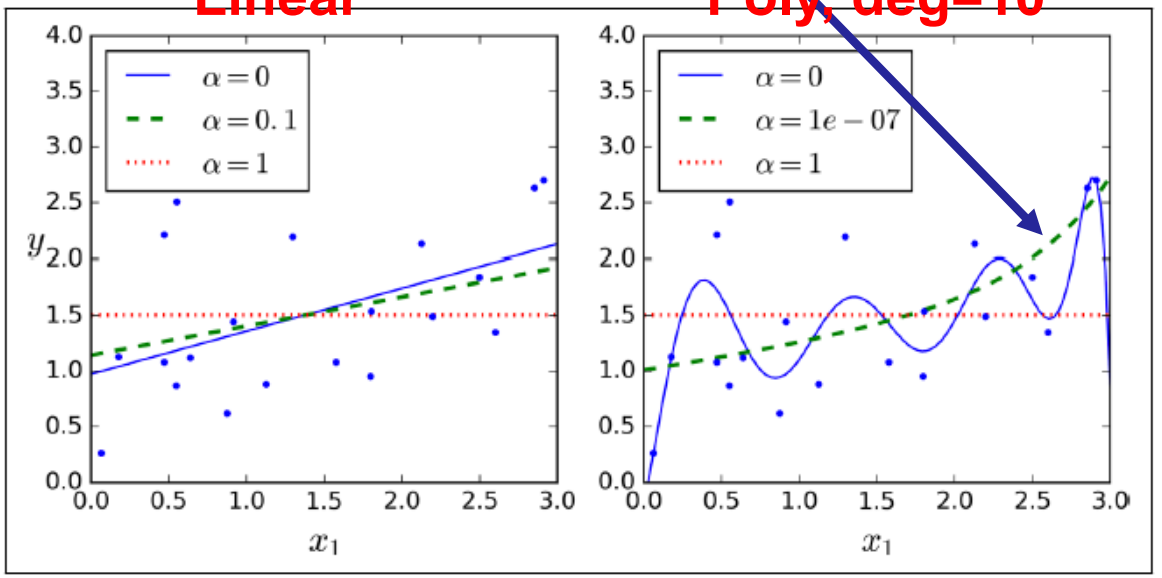
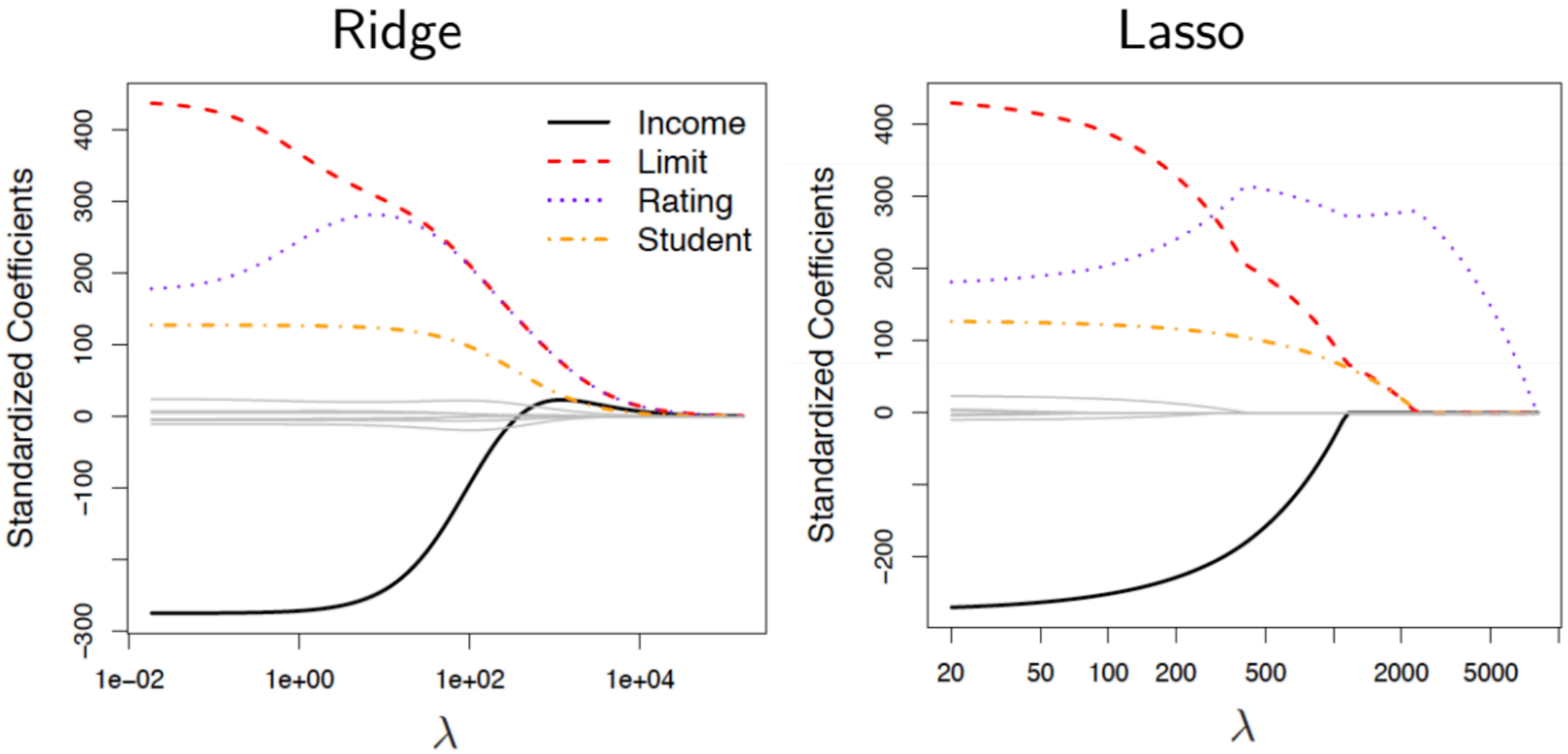


Figure 4-18. Lasso Regression

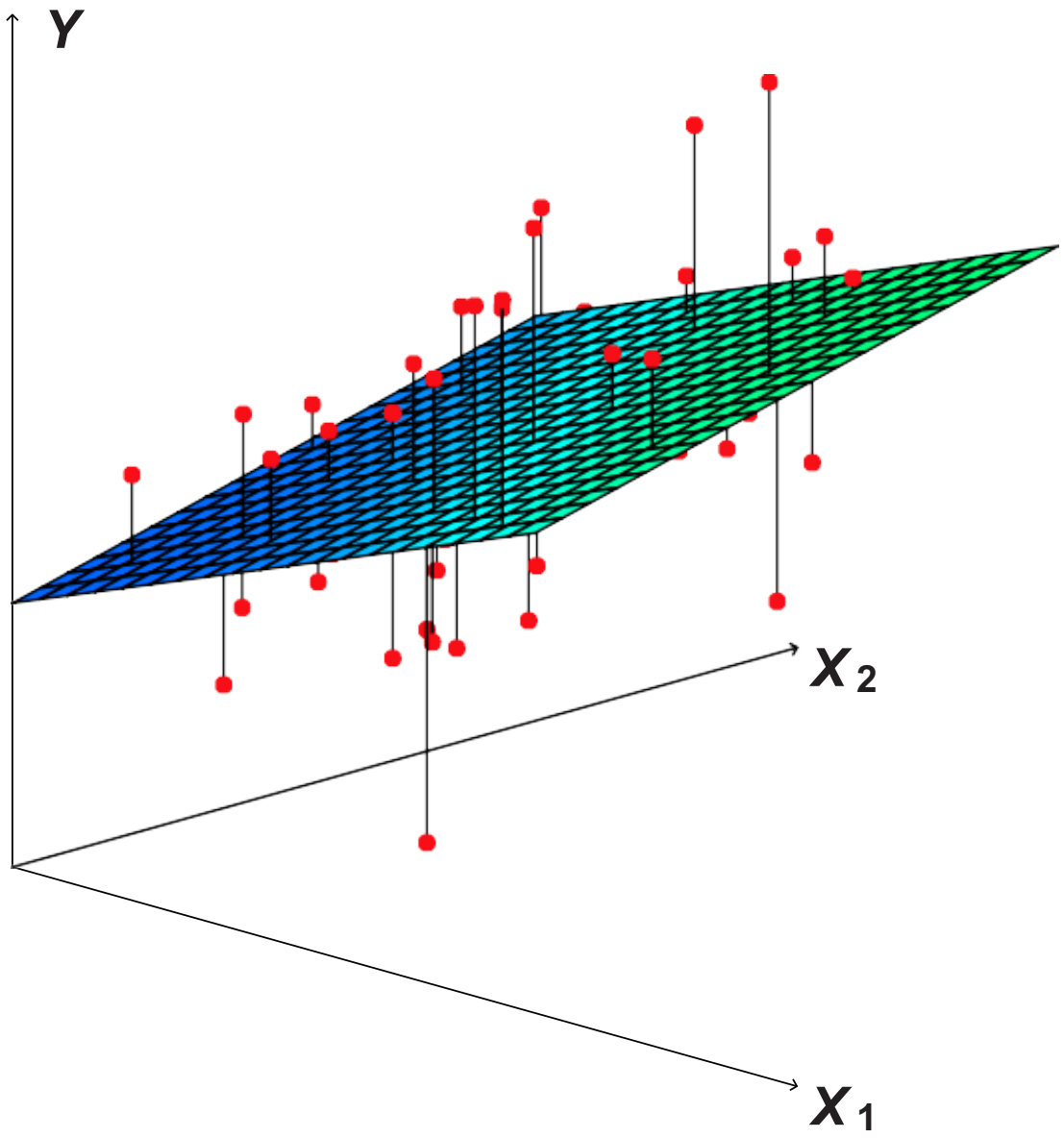
For example, the dashed line in the right plot on Figure 4-18 (with α = 10-7) looks quadratic, almost linear: all the weights for the high-degree polynomial features are equal to zero. In other words, Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights)

Example: credit data

Example from ISL sections 6.6.1 and 6.6.2: response is average credit debt, predictors are income, limit (credit limit), rating (credit rating), student (indicator), and others



Learn $f(X_1, X_2)$



Small training set

X1	X2	Y
-1	1	-1.5
-0.3	-1	-1.1
1	0.1	-2.05

MSE + L2

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Train a model with no bias terms so we can compare the following heatmaps:

- **MSE**
- **Model complexity loss term**
- **Ridge (i.e., MSE + L2)**
- **LASSO (i.e., MSE + L1)**

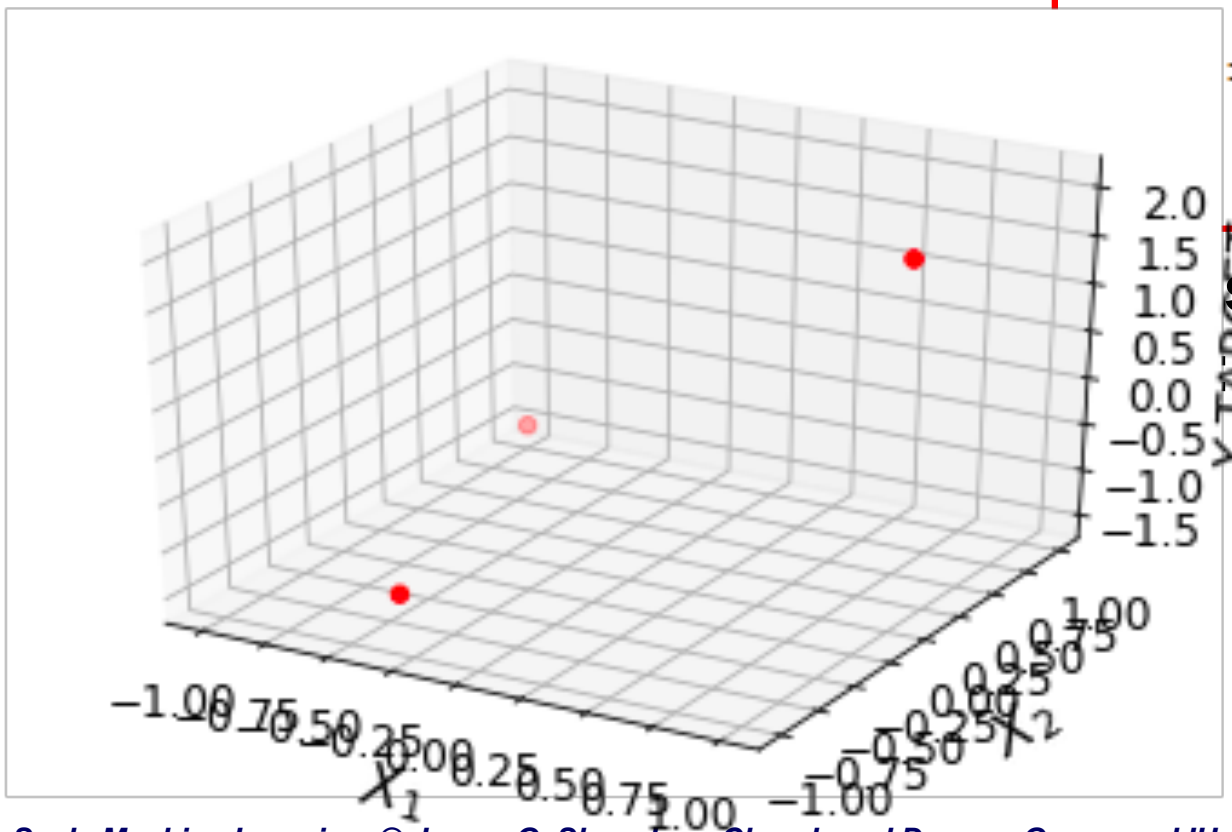
Small training set

X1	X2	Y
-1	1	-1.5
-0.3	-1	-1.1
1	0.1	-2.05

MSE + L2

$$= \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

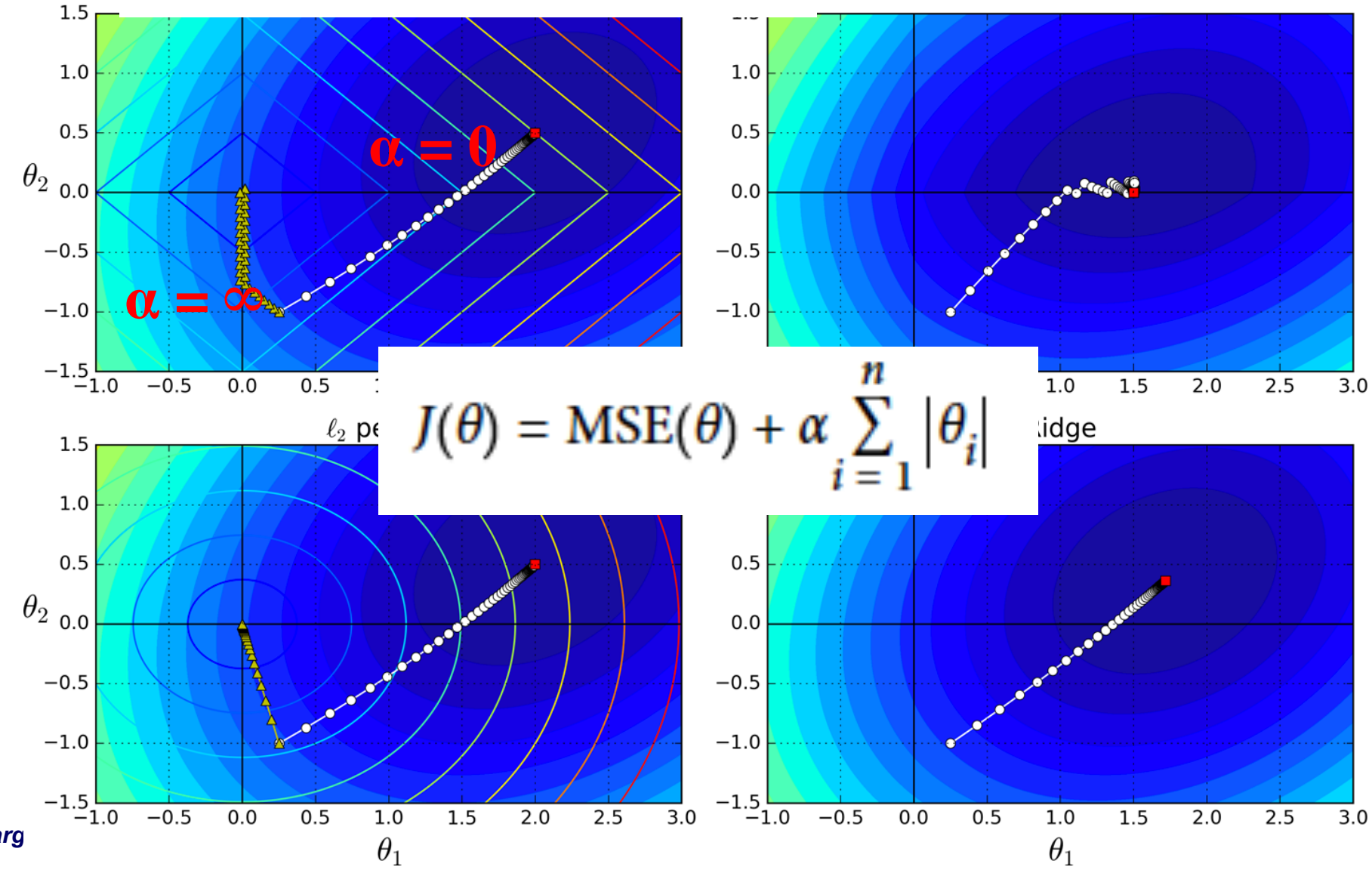
so we can



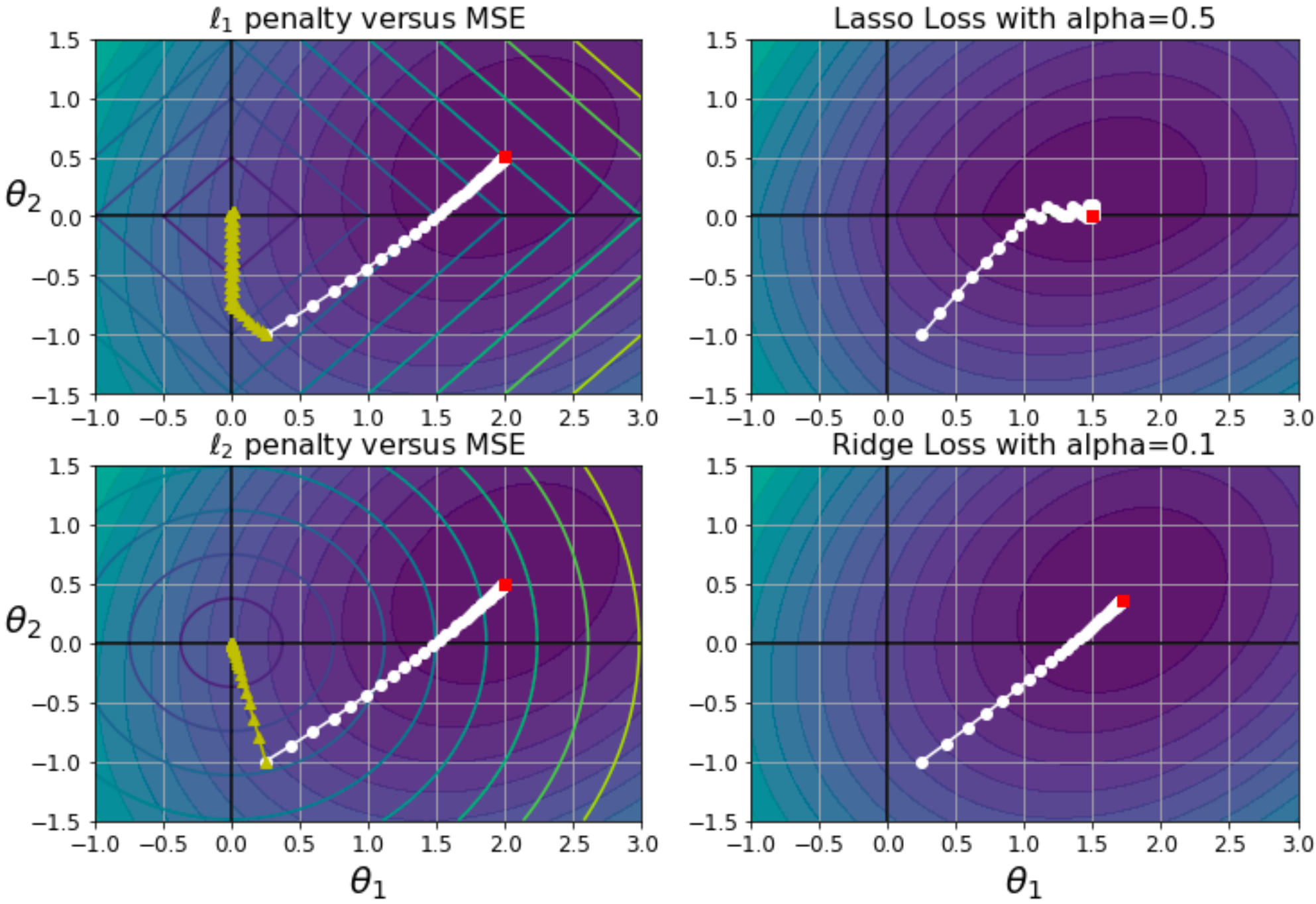
The background contours (ellipses) represent an unregularized MSE cost function (α = 0), and the white circles show the Batch Gradient Descent path with that cost function.
The foreground contours (diamonds) represent the ℓ1 penalty, and the triangles show the BGD path for this penalty only (α → ∞).

L1 contours versus MSE Surface

LASSO loss surface Combined Loss terms

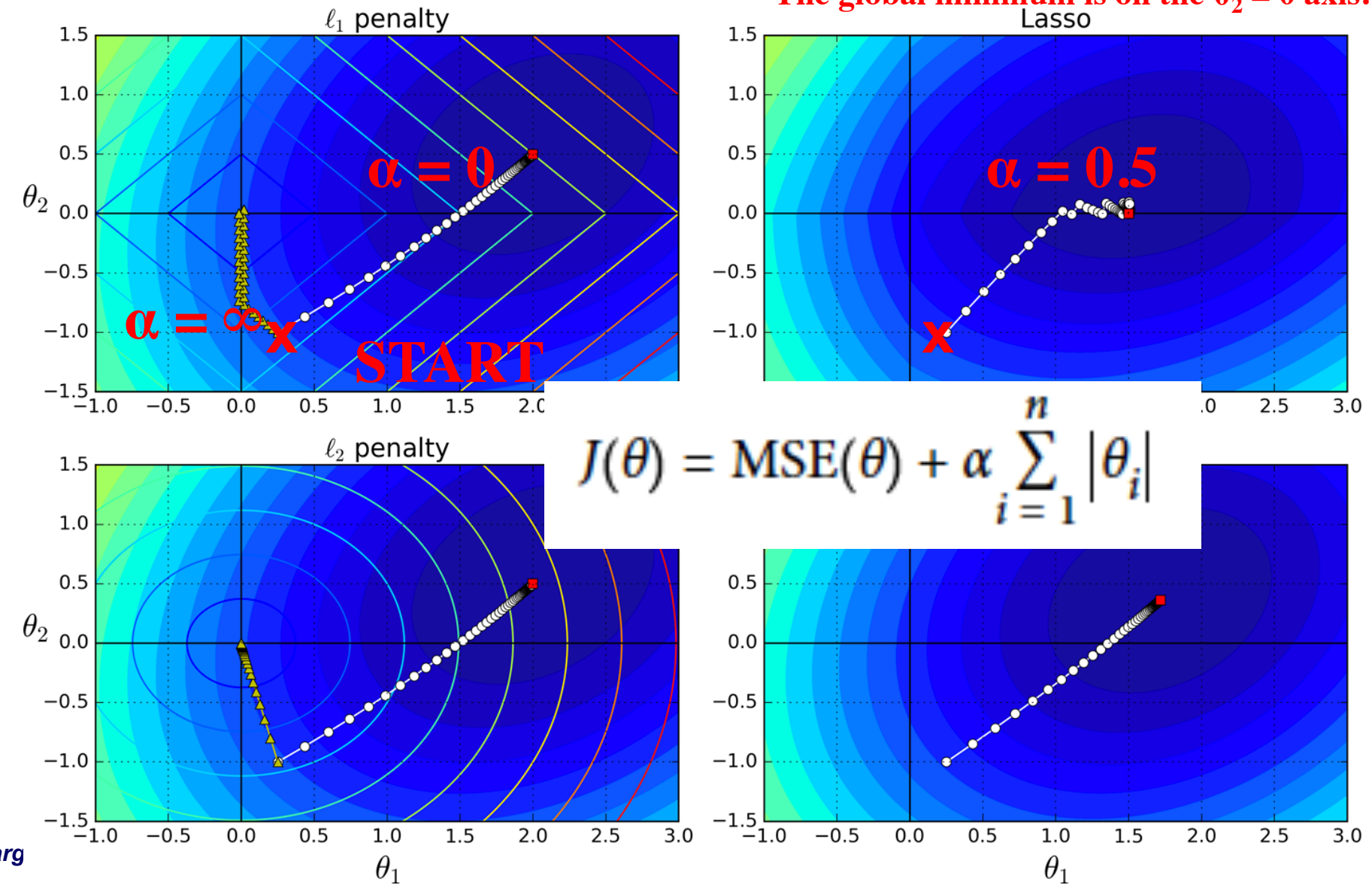


$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$


ℓ_1 penalty versus MSE
Lasso Loss with alpha=0.5
ℓ_2 penalty versus MSE
Ridge Loss with alpha=0.1
θ_1
θ_2

The background contours (ellipses) represent an unregularized MSE cost function (α = 0), and the white circles show the Batch Gradient Descent path with that cost function.
The foreground contours (diamonds) represent the ℓ1 penalty, and the triangles show the BGD path for this penalty only (α → ∞).

The global minimum is on the $\theta_2 = 0$ axis.

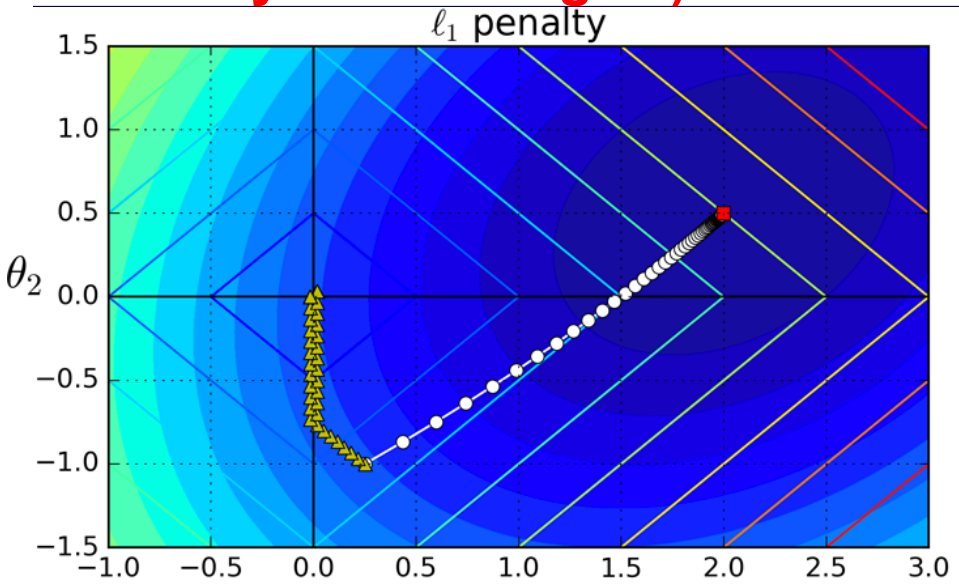


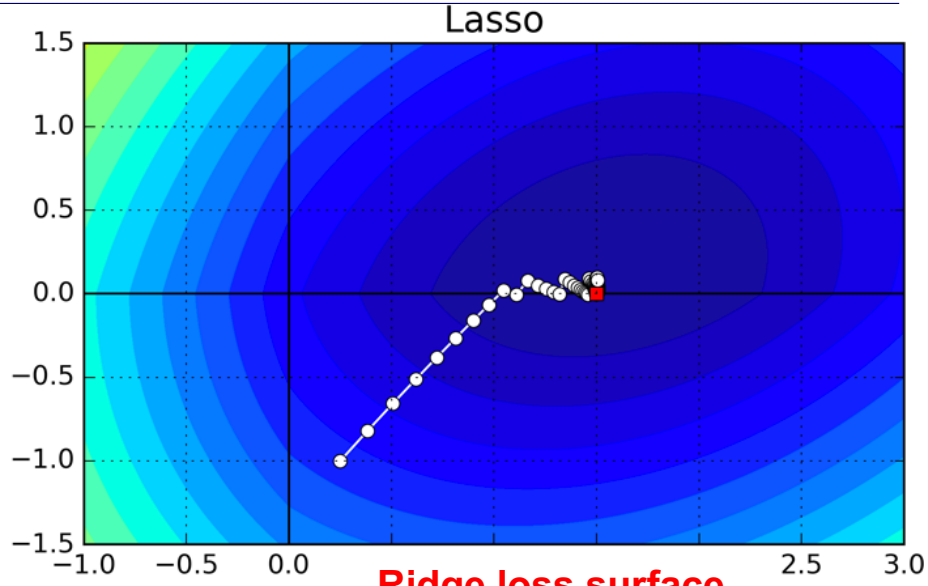
$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Lasso: L1 with $\alpha = 0$ (BGD)

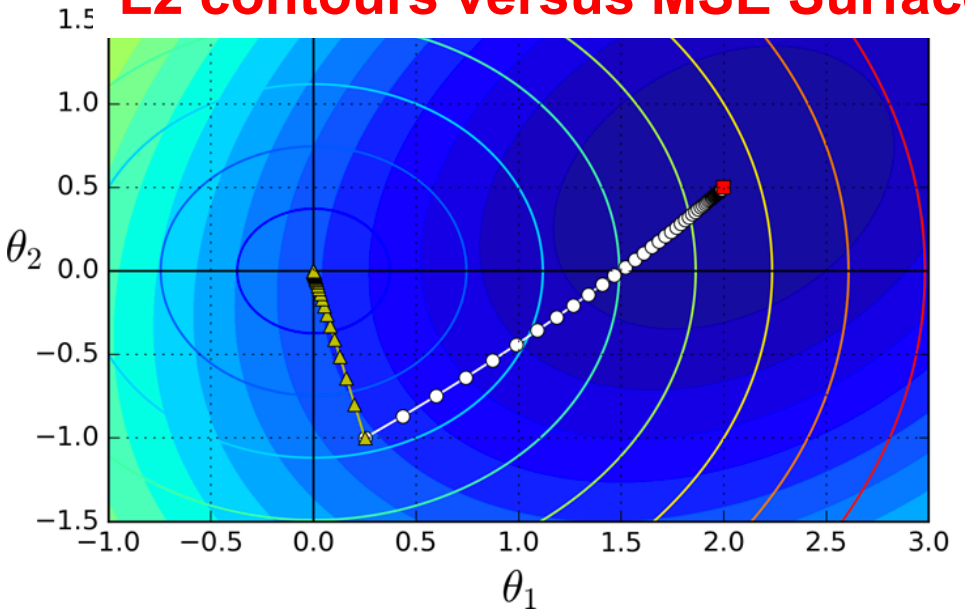
L1 with $\alpha = \infty$ (Data loss is zero Path yellow triangles)

Lasso: L1 with $\alpha = 0.5$

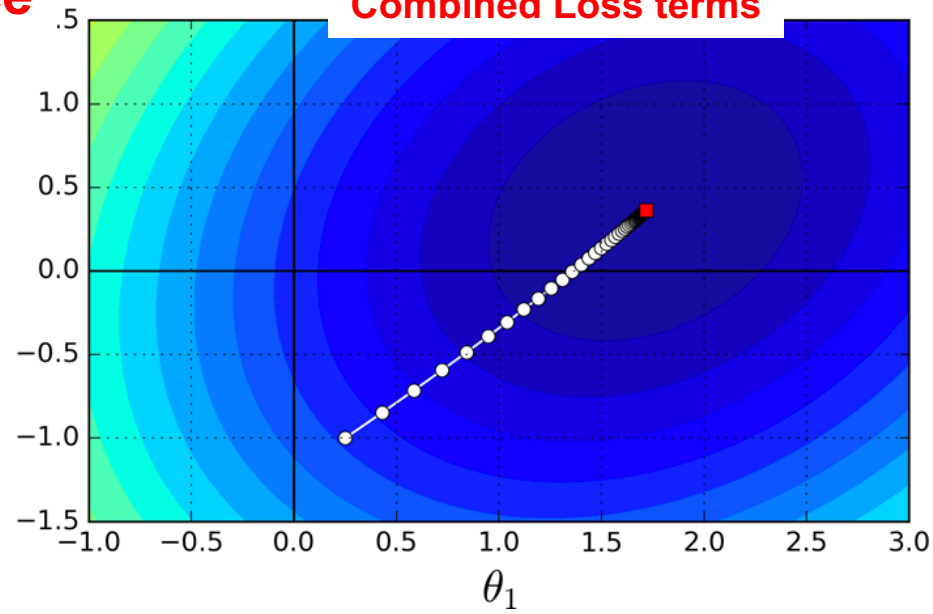




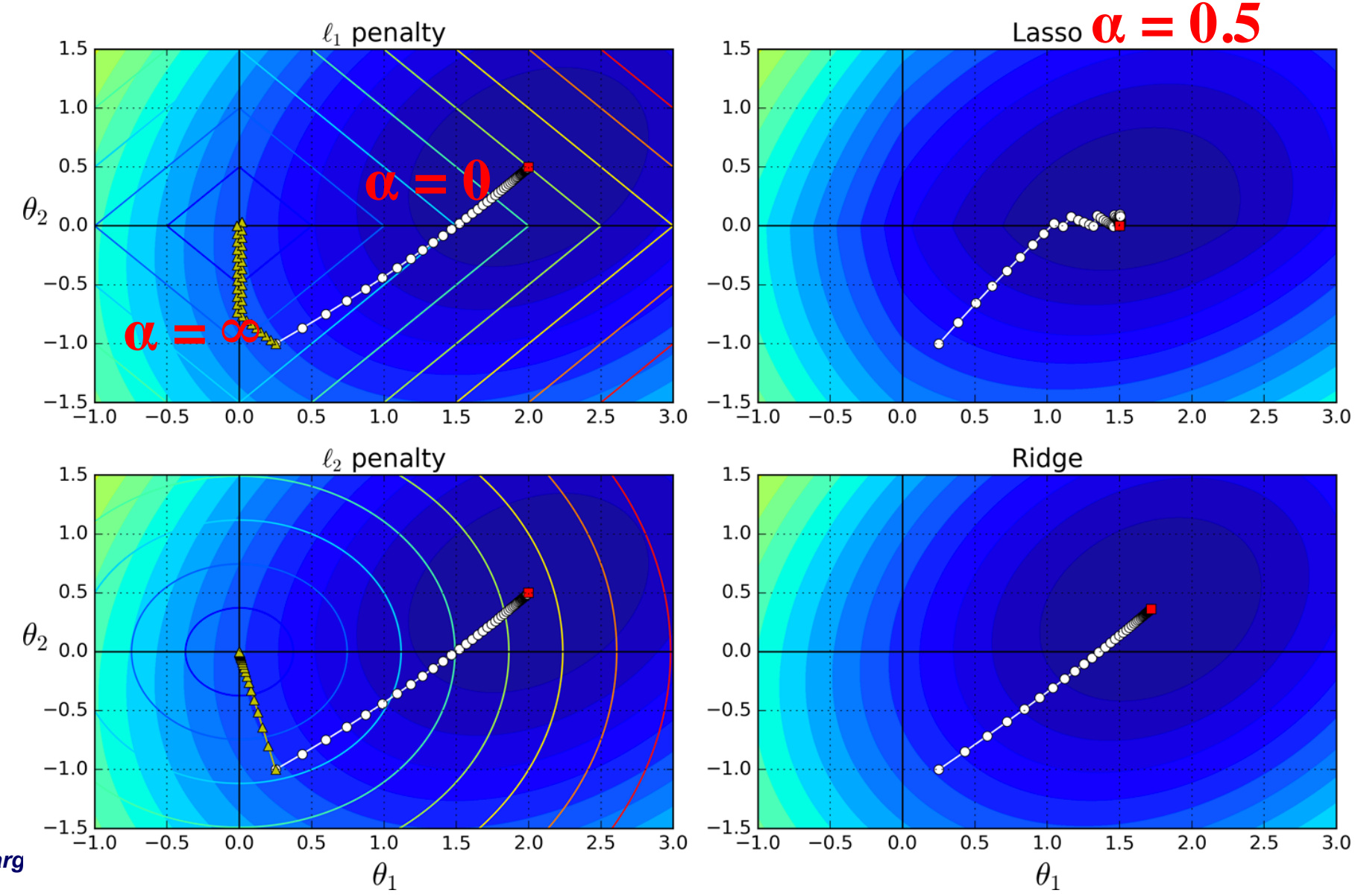
L2 contours versus MSE Surface



Ridge loss surface Combined Loss terms



Lasso vs. Ridge for different α: results in Diff Loss surfaces



localhost:8888/notebooks/shared/Dropbox/Projects/Courses/DataScienceAtScale/Labs/Labs-05-Regression-plus-ext

Apps | UberConference | Personal | http://www.tmtfinan... | Publications | Teaching | Admin | Projects | Deep Learning

jupyter Labs-05-LinRegr-Plus-extensions Last Checkpoint: 3 hours ago (unsaved changes) ?

File Edit View Insert Cell Kernel Navigate Widgets Help

Markdown

Contents [-]

otebook contains sample code for linear regression

Jnit 05 Linear Regression Extensions

Learning objectives

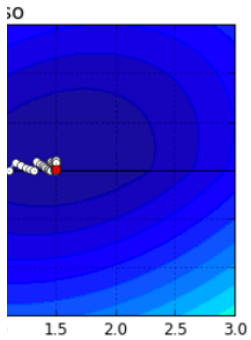
eview Simple Linear regression learning algorithms
- via gradient descent (and brute force)
- Closed form (via Normal Equation)

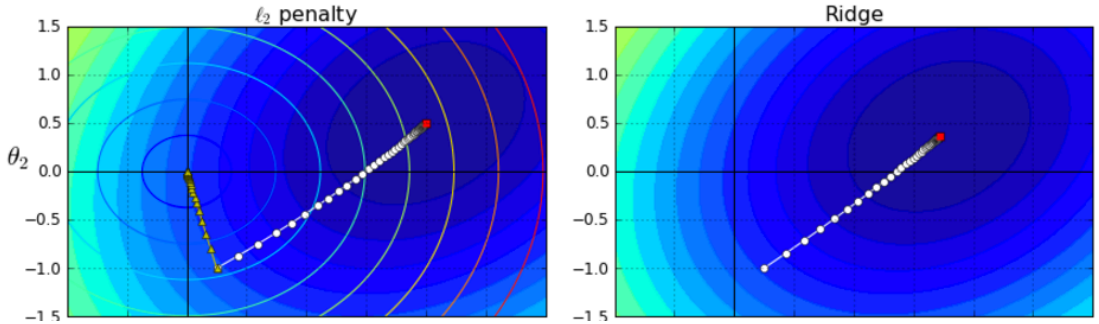
earn how batchsize matters in gradient descent
- Batch gradient descent
- Stochastic Gradient descent (SGD)
- Minibatch Gradient descent

nderstand how polynomial regression extends linear regression

ee how Bias-Variance can be used to analyze underfitting and overfitting a model
- E.g., see how simple regression can be extended to the level that it overfits

eature selection as a means to address Bias-Variance
- Feature selection via regularization (LASSO vs Ridge Regression)
- Sequential feature selection
- Feature importance via ensembles of decision trees





Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Linear regression Elastic Net Regularization

- **Elastic Net is a middle ground between Ridge Regression and Lasso Regression.**
- **The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r.**
- **When r = 0, Elastic Net is equivalent to Ridge Regression, and when r = 1, it is equivalent to Lasso Regression**

$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2$$

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

Linear Model Selection and Regularization

- **For more details**
  - https://www.dropbox.com/s/q96xyy17xy73bkn/model_selection-ISRL.pdf?dl=0

Three classes of methods

- *Subset Selection*. We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.
- *Shrinkage*. We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as *regularization*) has the effect of reducing variance and can also perform variable selection.
- *Dimension Reduction*. We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different *linear combinations*, or *projections*, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

Best Subset Selection

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.

2. For $k = 1, 2, \ldots p$:

   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.

   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.

3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Feature subset selection: Credit data set 10 input variables

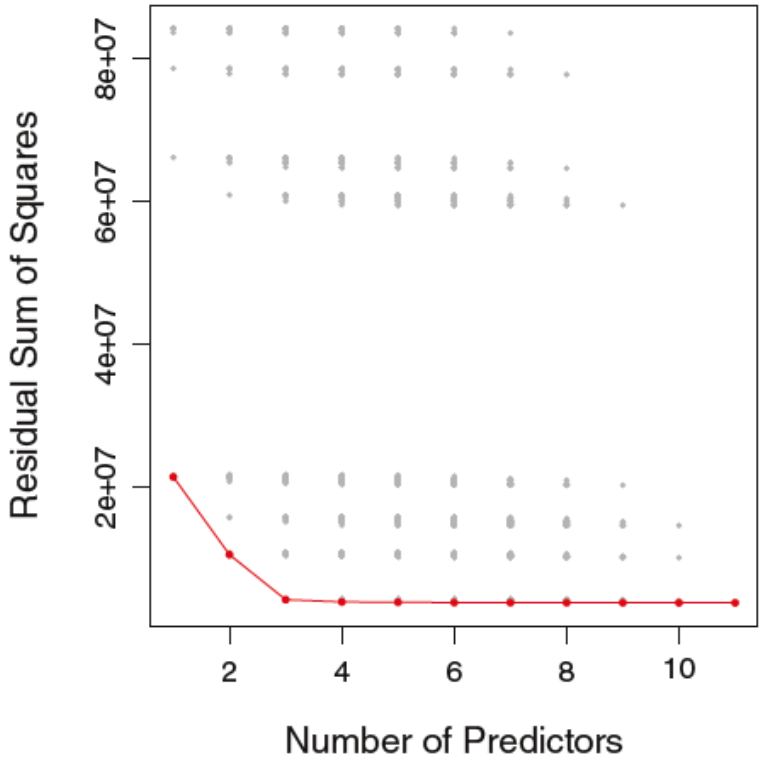


FIGURE 6.1. *For each possible model containing a subset of the ten predictors in the* `Credit` *data set, the RSS and* R^2 *are displayed. The red frontier tracks the best model for a given number of predictors, according to RSS and* R^2*. Though the data set contains only ten predictors, the* x*-axis ranges from* 1 *to* 11*, since one of the variables is categorical and takes on three values, leading to the creation of two dummy variables.*

Best subset selection → Stepwise methods

- **Best subset selection cannot be applied with very large p .**
  - Best subset selection may also suffer from statistical problems when p is large.
  - The larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data.
  - Thus an enormous search space can lead to overfitting and high variance of the coefficient estimates.
- **Stepwise methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.**

Forward Stepwise Selection

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   - 2.1 Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   - 2.2 Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

best subset selection VERSUS forward stepwise selection on the Credit data set.

# Variables	Best subset	Forward stepwise
One	rating	rating
Two	rating, income	rating, income
Three	rating, income, student	rating, income, student
Four	cards, income, student, limit	rating, income, student, limit

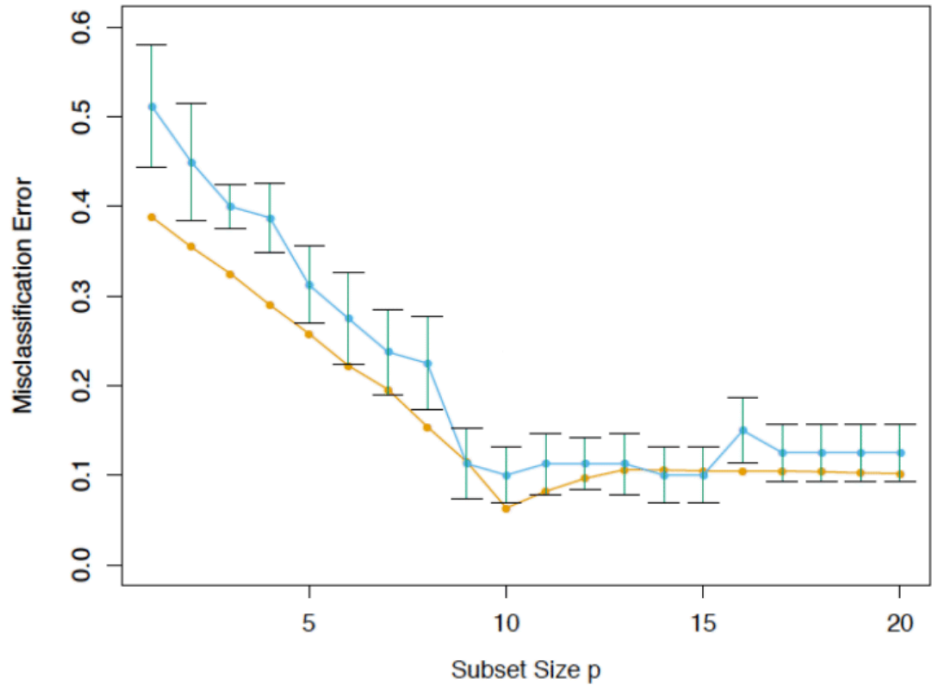
TABLE 6.1. *The first four selected models for best subset selection and forward stepwise selection on the* Credit *data set. The first three models are identical but the fourth models differ.*

The first three models are identical but the fourth models differ.

Forward Selection

model selection and validation

Cross-validation can be used to estimate the prediction error curve



(From ESL page 244)

Backward stepsize selection

Algorithm 6.3 *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Case Studies: Regularized linear regression Lasso and Ridge

- **Ridge Regression Introduction**
  - https://www.dropbox.com/s/3b2epf2vfwcypcg/RidgeRegression-From-Scratch.pdf?dl=0
- **Ridge regression (more details)**
  - https://www.dropbox.com/s/h4wx8avnne47wdp/16-modr1-RidgeRegression.pdf?dl=0
- **Lasso regression**
  - https://www.dropbox.com/s/awqgl3nj6uinlmq/17-modr2-LassoRegression.pdf?dl=0

Linear Model Selection and Regularization

- **For more details**
  - https://www.dropbox.com/s/q96xyy17xy73bkn/model_selection-ISRL.pdf?dl=0

Three classes of methods

- *Subset Selection*. We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.
- *Shrinkage*. We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as *regularization*) has the effect of reducing variance and can also perform variable selection.
- *Dimension Reduction*. We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different *linear combinations*, or *projections*, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

Cover in a Later lecture

Lecture Outline

- **Introduction**
- **Linear regression**
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent
- **Polynomial Regression**
- **Bias-Variance Tradeoff**
- **Feature Selection**
  - Regularization
    - Ridge Regression
    - LASSO Regression
    - Hybrid: Elastic Net
  - Subset selection, forward/backward stepsize selection
- **Summary**

End of lecture