EVALUATING MODEL FIT

LEARNING OBJECTIVES

- Define regularization, bias, and error metrics for regression problems
- Evaluate model fit using loss functions
- Select regression methods based on fit and complexity

COURSE

PRE-WORK

PRE-WORK REVIEW

- Understand goodness of fit (r-squared)
- Measure statistical significance of features
- Recall what a residual is
- Implement a sklearn estimator to predict a target variable

OPENING

R-SQUARES AND RESIDUALS

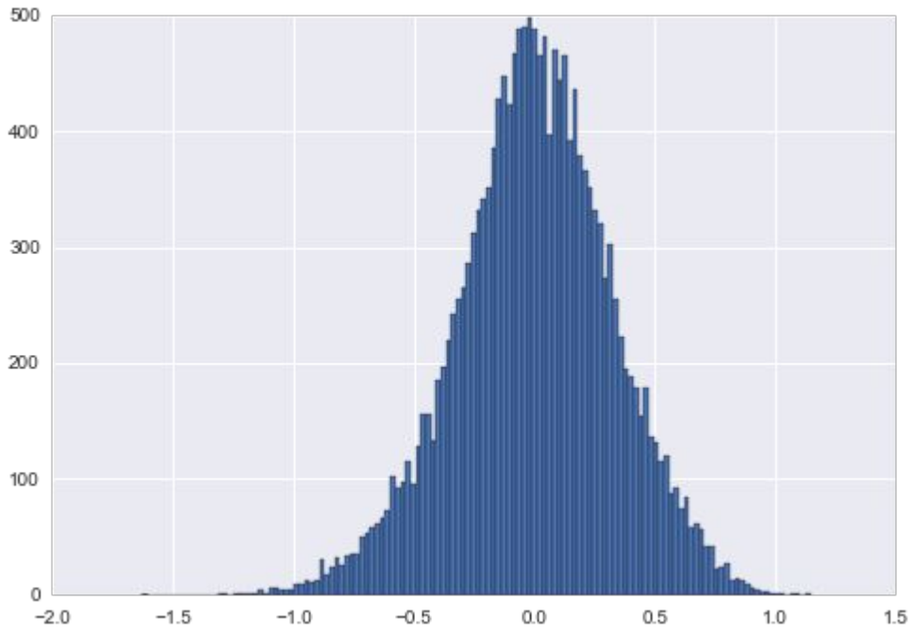
WHAT IS R-SQUARED? WHAT IS A RESIDUAL?

- R-squared, the central metric introduced for linear regression
- Which model performed better, one with an r-squared of 0.79 or 0.81?
- R-squared measures explain variance.
- But does it tell the magnitude or scale of error?
- We'll explore loss functions and find ways to refine our model.

LINEAR MODELS AND ERROR

RECALL: WHAT'S RESIDUAL ERROR?

- In linear models, residual error must be normal with a median close to zero.
- Individual residuals are useful to see the error of specific points, but it doesn't provide an overall picture for optimization.
- We need a metric to summarize the error in our model into one value.
- Mean square error: the mean residual error in our model



MEAN SQUARED ERROR (MSE)

- To calculate MSE:
  - Calculate the difference between each target y and the model's predicted value y-hat (i.e. the residual)
  - Square each residual.
  - Take the mean of the squared residual errors.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

MEAN SQUARED ERROR (MSE)

- sklearn's metrics module includes a mean_squared_error function.

```
from sklearn import metrics
metrics.mean_squared_error(y, model.predict(X))
```

MEAN SQUARED ERROR (MSE)

- For example, two arrays of the same values would have an MSE of 0.

```
from sklearn import metrics
metrics.mean_squared_error([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])
0.0
```

MEAN SQUARED ERROR (MSE)

- Two arrays with different values would have a positive MSE.

```
from sklearn import metrics
metrics.mean_squared_error([1, 2, 3, 4, 5], [5, 4, 3, 2, 1])
# (4^2 + 2^2 + 0^2 + 2^2 + 4^2) / 5
8.0
```

HOW DO WE MINIMIZE ERROR?

- The regression method we've used is called "Ordinary Least Squares".
- This means that given a matrix X, solve for the *least* amount of square error for y.
- However, this assumes that X is unbiased, that it is representative of the population.

LET'S COMPARE TWO RANDOM MODELS

```
import numpy as np
import pandas as pd
from sklearn import linear_model

df = pd.DataFrame({'x': range(100), 'y': range(100)})
biased_df  = df.copy()
biased_df.loc[:20, 'x'] = 1
biased_df.loc[:20, 'y'] = 1

def append_jitter(series):
    jitter = np.random.random_sample(size=100)
    return series + jitter
```

LET'S COMPARE TWO RANDOM MODELS

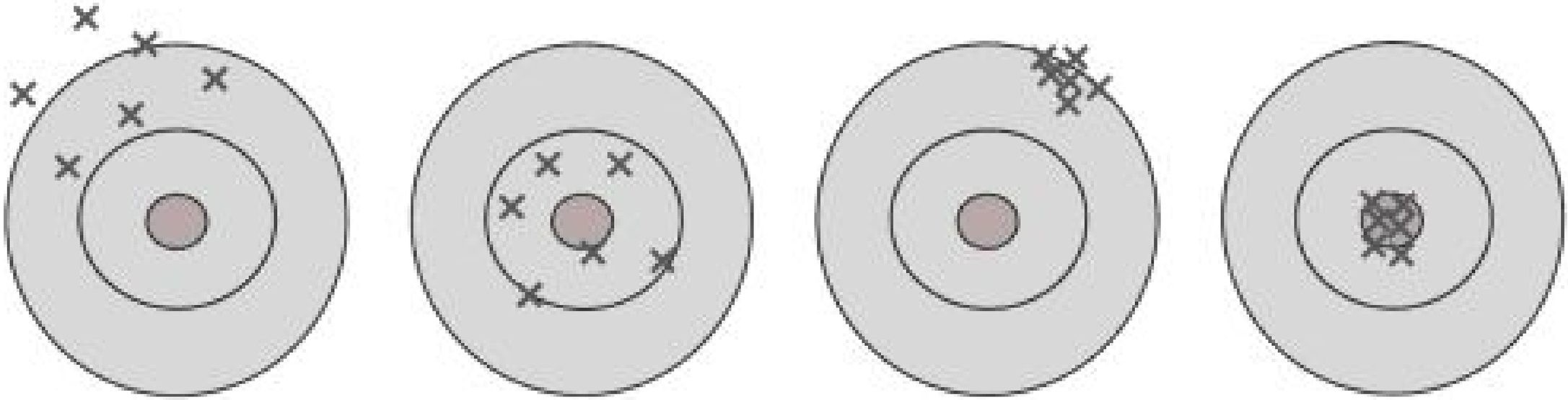
```
df['x'] = append_jitter(df.x)
df['y'] = append_jitter(df.y)

biased_df['x'] = append_jitter(biased_df.x)
biased_df['y'] = append_jitter(biased_df.y)

- Fit:
lm = linear_model.LinearRegression().fit(df[['x']], df['y'])
print metrics.mean_squared_error(df['y'], lm.predict(df[['x']]))

- Biased fit:
lm = linear_model.LinearRegression().fit(biased_df[['x']], biased_df['y'])
print metrics.mean_squared_error(df['y'], lm.predict(df[['x']]))
```

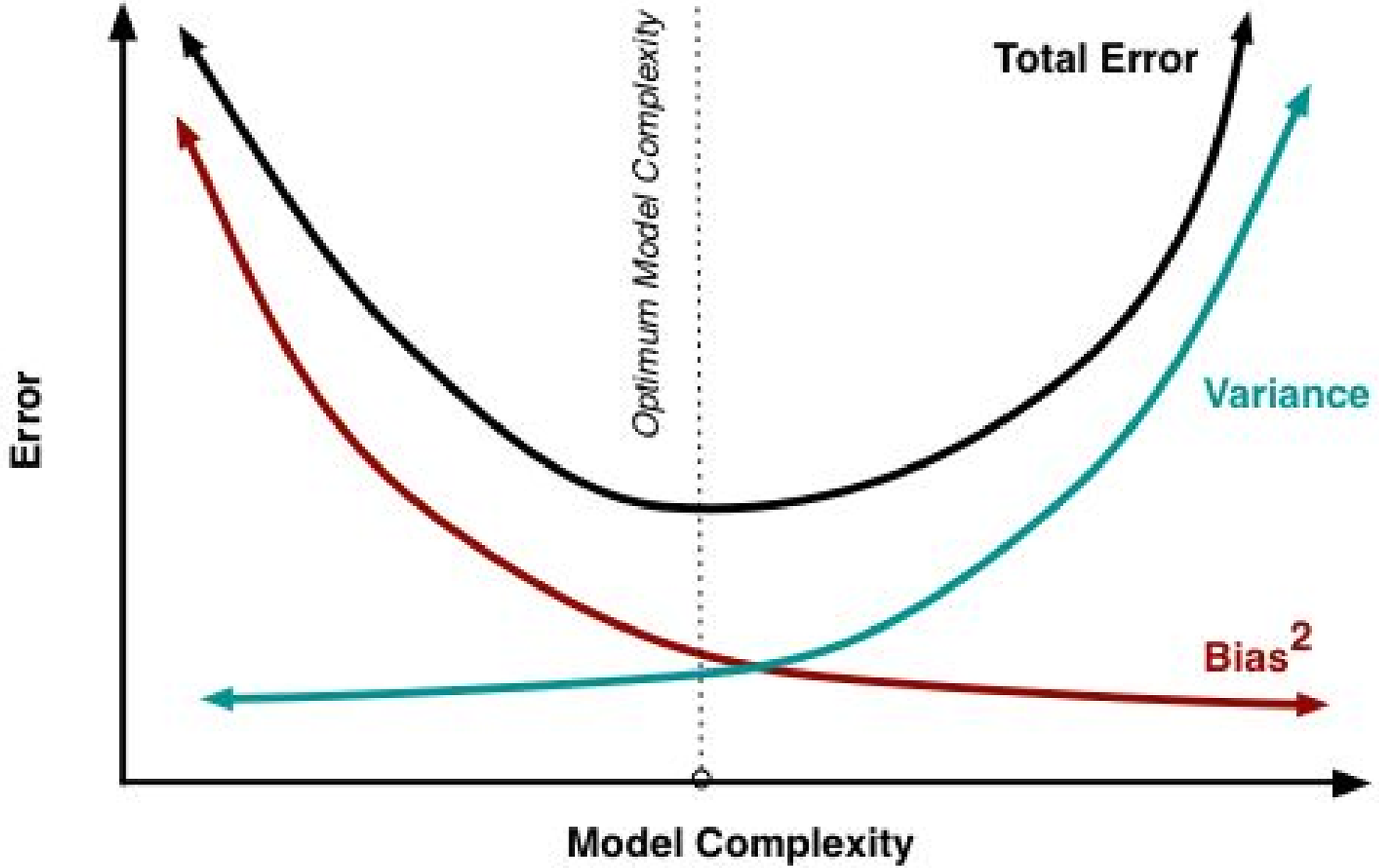
BIAS VS. VARIANCE



BIAS VARIANCE TRADEOFF

- When our error is ***biased***, it means the model's prediction is consistently far away from the actual value.
- This could be a sign of poor sampling and poor data.
- One objective of a biased model is to trade bias error for generalized error. We prefer the error to be more evenly distributed across the model.
- This is called error due to ***variance***.
- We want our model to *generalize* to data it hasn't seen even if doesn't perform as well on data it has already seen.

BIAS VARIANCE TRADEOFF



ACTIVITY: KNOWLEDGE CHECK

ANSWER THE FOLLOWING QUESTIONS (5 minutes)

1. Which of the following scenarios would be better for a weatherman?
   a. Knowing that I can very accurately "predict" the temperature outside from previous days perfectly, but be 20-30 degrees off for future days
   b. Knowing that I can accurately predict the general trend of the temperate outside from previous days, and therefore am at most only 10 degrees off on future days

DELIVERABLE

Answers to the above questions

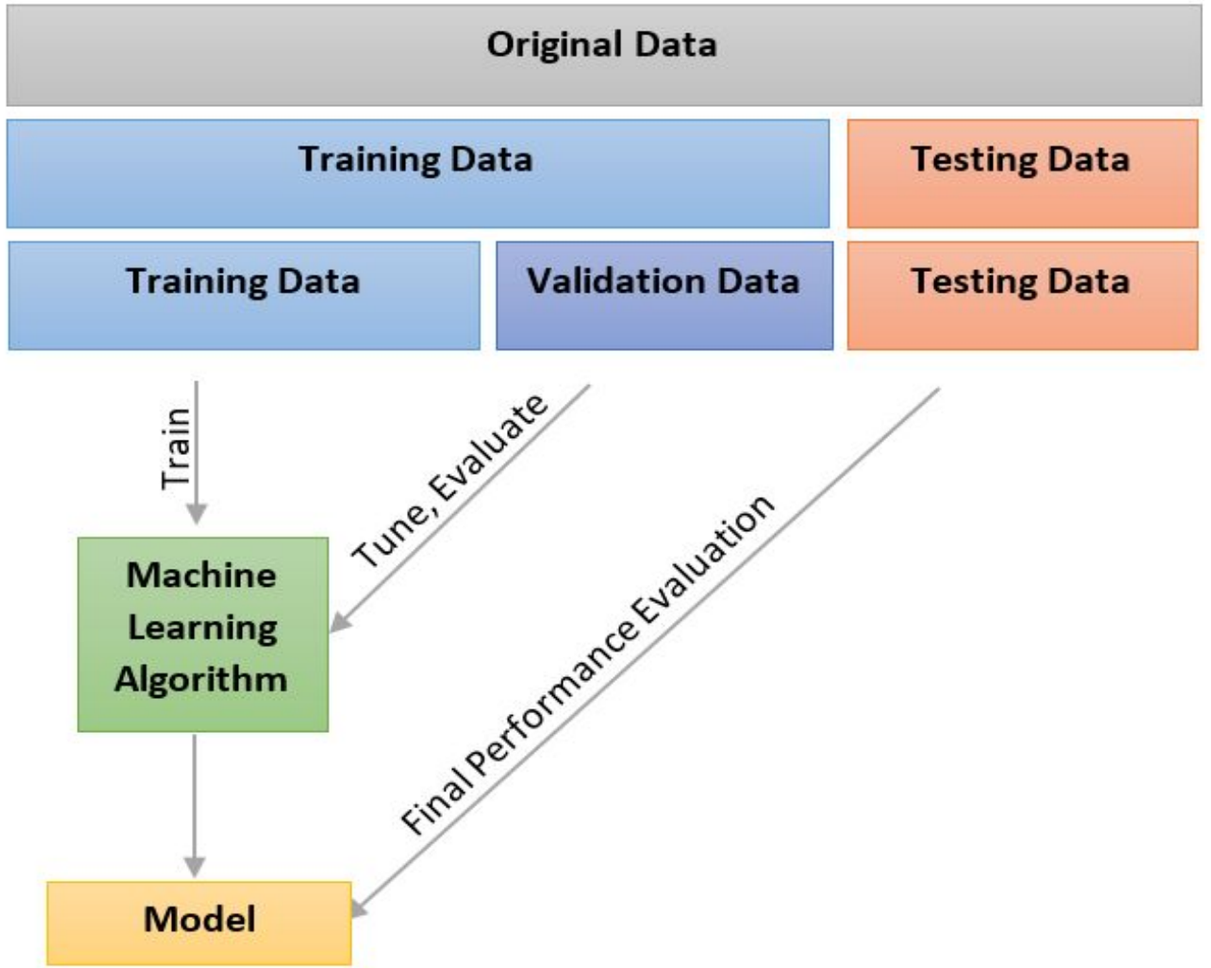
DEMO

CROSS VALIDATION

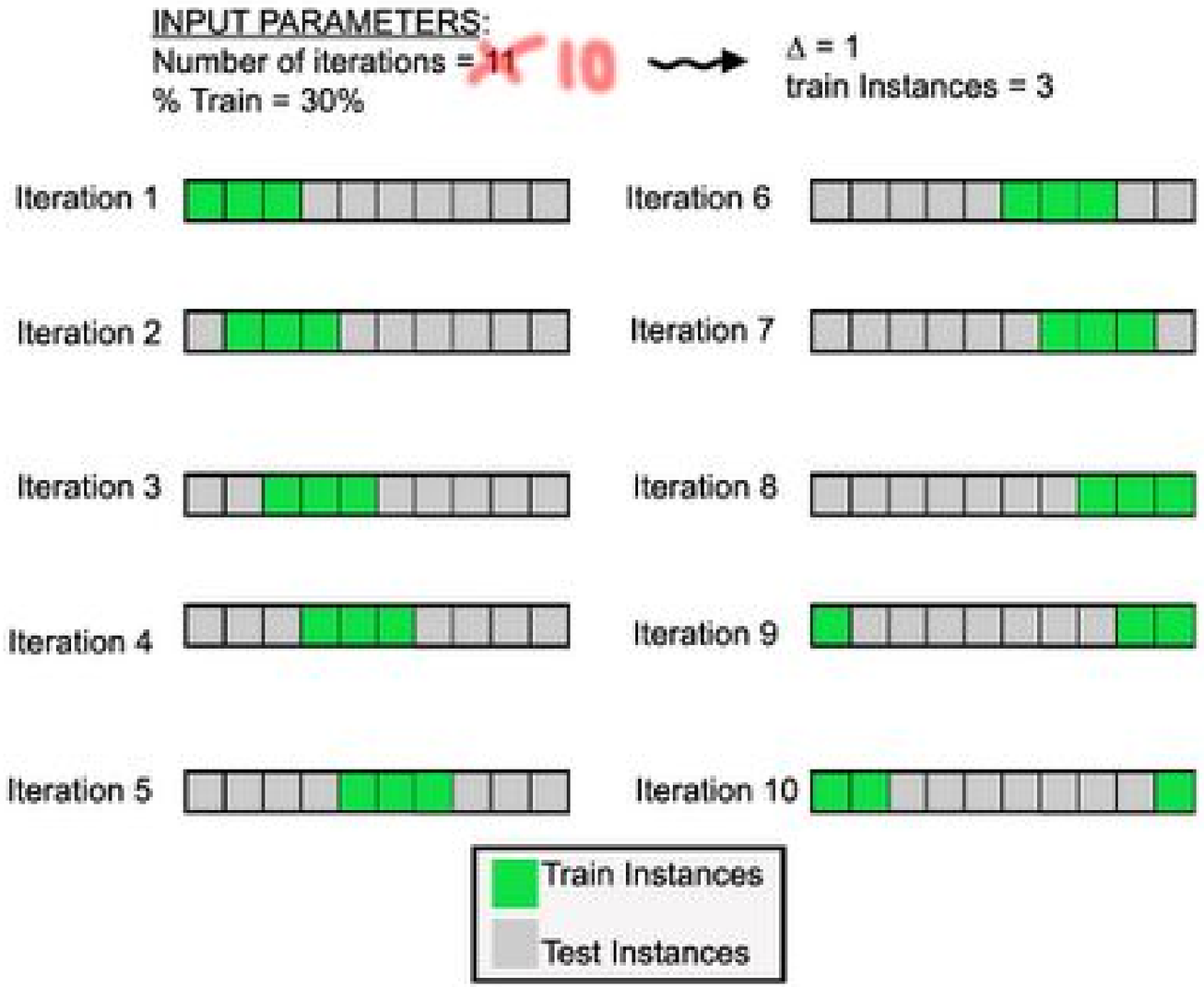
CROSS VALIDATION

- Cross validation can help account for bias.
- The general idea is to
  - Generate several models on different cross sections of the data
  - Measure the performance of each
  - Take the mean performance
- This technique swaps bias error for generalized error, describing previous trends accurately enough to extend to future trends.

CROSS VALIDATION



CROSS VALIDATION



K-FOLD CROSS VALIDATION

- k-fold cross validation
  - Split the data into *k* group
  - Train the model on all segments except one
  - Test model performance on the remaining set
- If k = 5, split the data into five segments and generate five models.

USING K-FOLD CROSS VALIDATION WITH MSE

- Import the appropriate packages and load data.

```
from sklearn import cross_validation
wd = '../../datasets/'
bikeshare = pd.read_csv(wd + 'bikeshare/bikeshare.csv')
weather = pd.get_dummies(bikeshare.weathersit, prefix='weather')
modeldata = bikeshare[['temp', 'hum']].join(weather[['weather_1',
'weather_2', 'weather_3']])
y = bikeshare.casual
```

USING K-FOLD CROSS VALIDATION WITH MSE

- Build models on subsets of the data and calculate the average score.

```
kf = cross_validation.KFold(len(modeldata), n_folds=5, shuffle=True)
scores = []
for train_index, test_index in kf:
    lm =
linear_model.LinearRegression().fit(modeldata.iloc[train_index],
y.iloc[train_index])
    scores.append(metrics.mean_squared_error(y.iloc[test_index],
lm.predict(modeldata.iloc[test_index])))

print np.mean(scores)
```

USING K-FOLD CROSS VALIDATION WITH MSE

- This can be compared to the model built on all of the data.

```
- This score will be lower, but we're trading off bias error for
generalized error:
lm = linear_model.LinearRegression().fit(modeldata, y)
print metrics.mean_squared_error(y, lm.predict(modeldata))
```

- Which approach would predict new data more accurately?

GUIDED PRACTICE

CROSS VALIDATION WITH LINEAR REGRESSION

ACTIVITY: CROSS VALIDATION WITH LINEAR REGRESSION

DIRECTIONS (20 minutes)

If we were to continue increasing the number of folds in cross validation, would error increase or decrease?

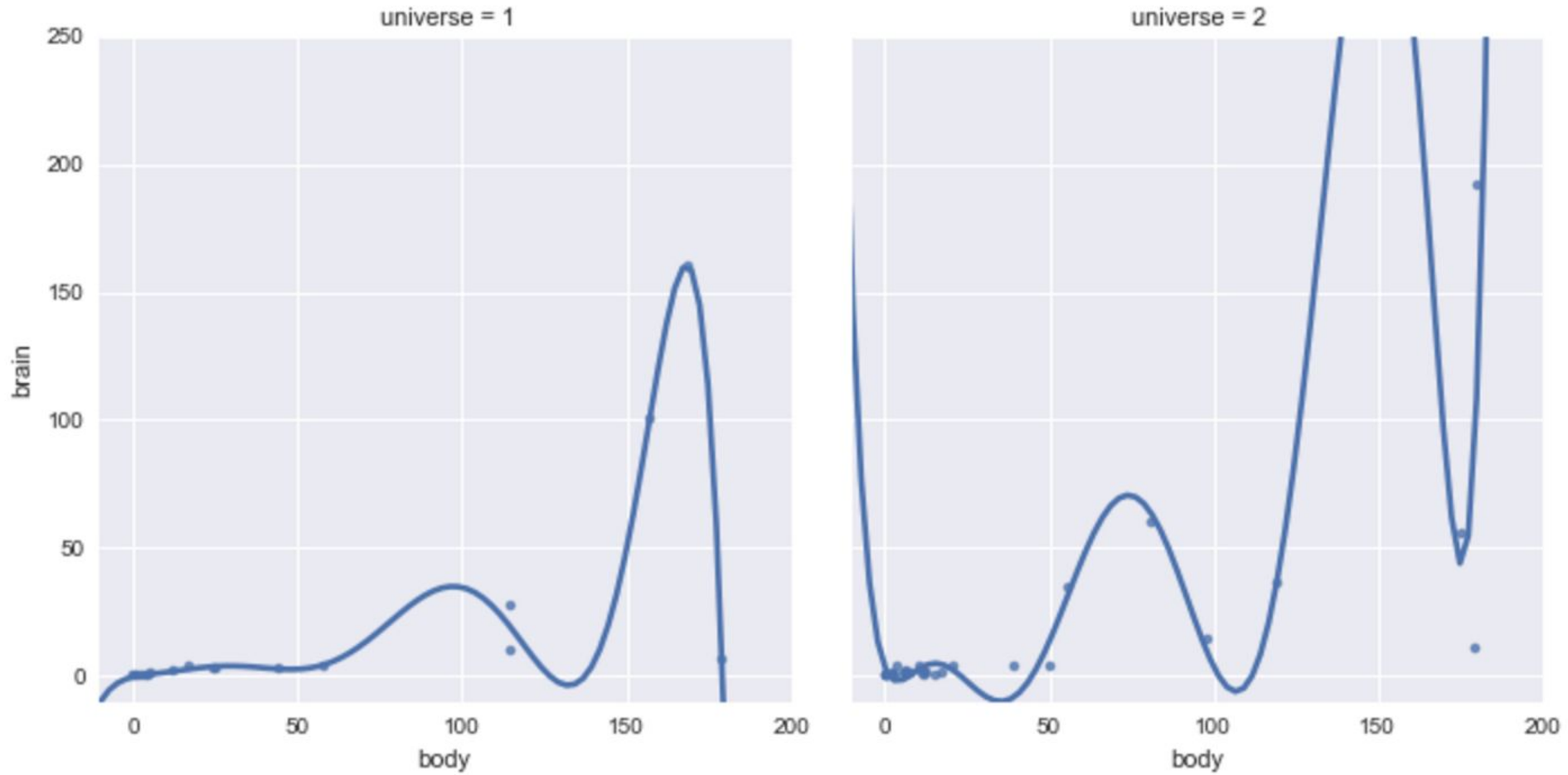
1. Using the previous code example, perform k-fold cross validation for all even numbers between 2 and 50.
2. Answer the following questions:
   a. What does `shuffle=True` do?
   b. At what point does cross validation no longer seem to help the model?
3. Hint: `range(2, 51, 2)` produces a list of even numbers from 2 to 50

DELIVERABLE

Answers to questions

REVIEW: OVERFITTING

Overfitting with polynomial regression



OVERFITTING WITH LINEAR MODELS

What are the general characteristics of linear models?

Low model complexity

High bias, low variance

Does not tend to overfit

Nevertheless, **overfitting can still occur** with linear models if you allow them to have **high variance**. Here are some common causes:

CAUSE 1: IRRELEVANT FEATURES

Linear models can overfit if you include "irrelevant features", meaning features that are unrelated to the response. Why?

Because it will learn a coefficient for every feature you include in the model, regardless of whether that feature has the signal or the noise.

This is especially a problem when **p (number of features) is close to n (number of observations),** because that model will naturally have high variance.

CAUSE 2: CORRELATED FEATURES

Linear models can overfit if the included features are highly correlated with one another. Why?

From the scikit-learn documentation:

"...coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance."

http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

CAUSE 3: LARGE COEFFICIENTS

Linear models can overfit if the coefficients (after feature standardization) are too large. Why?

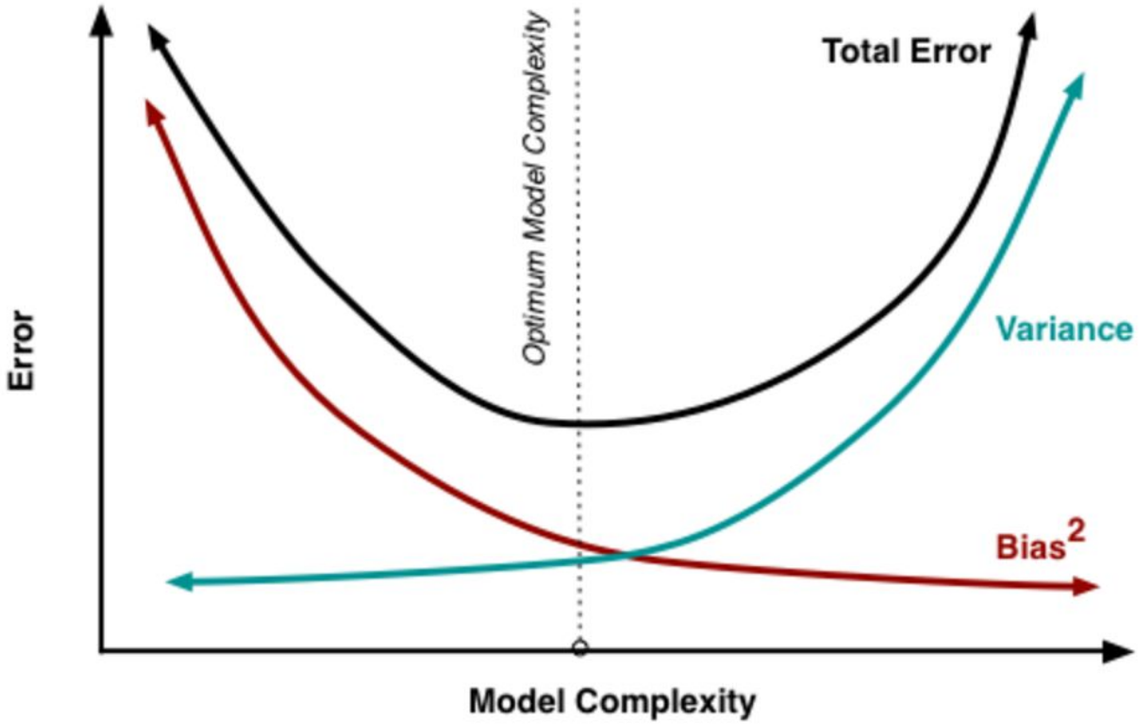
Because the **larger** the absolute value of the coefficient, the more **power** it has to change the predicted response, resulting in a higher variance.

REGULARIZATION OF LINEAR MODELS

Regularization is a method for "constraining" or "regularizing" the size of the coefficients, thus "shrinking" them towards zero.

It reduces model variance and thus **minimizes overfitting**.

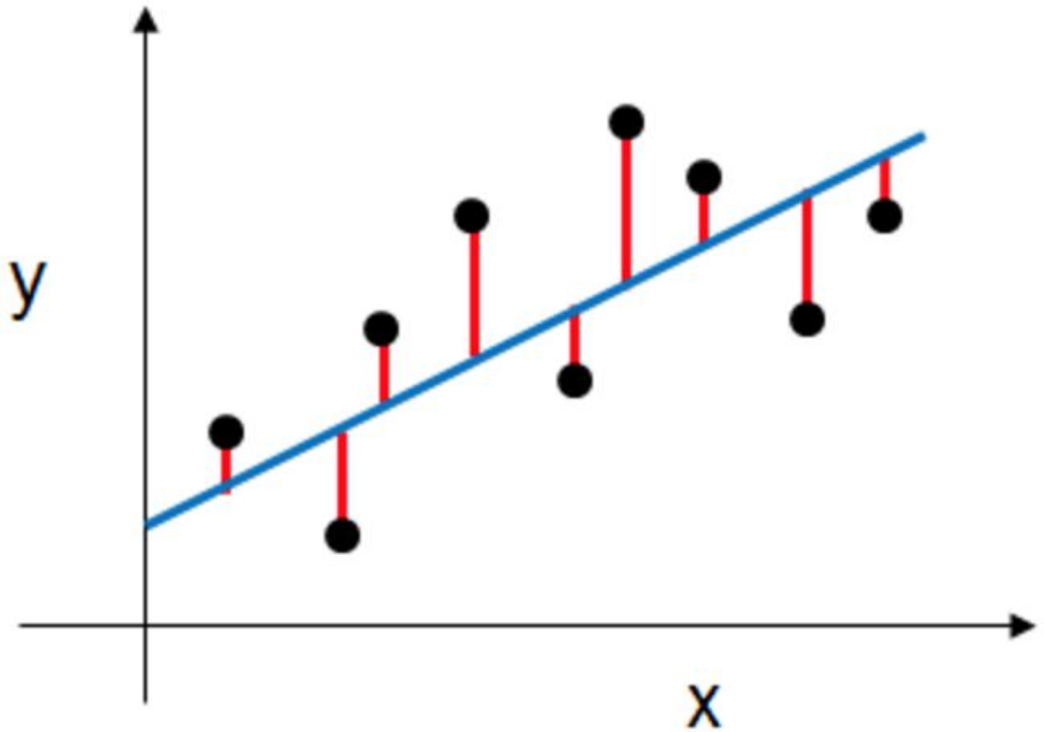
If the model is too complex, it tends to reduce variance more than it increases bias, resulting in a model that is more likely to generalize.



Our goal is to locate the **optimum model complexity**, and thus regularization is useful when we believe our model is too complex.

HOW DOES REGULARIZATION WORK?

For a normal linear regression model, we estimate the coefficients using the least squares criterion, which minimizes the residual sum of squares (RSS):



Model Prediction

$$SS_{residuals} = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Observed Result

RIDGE REGRESSION

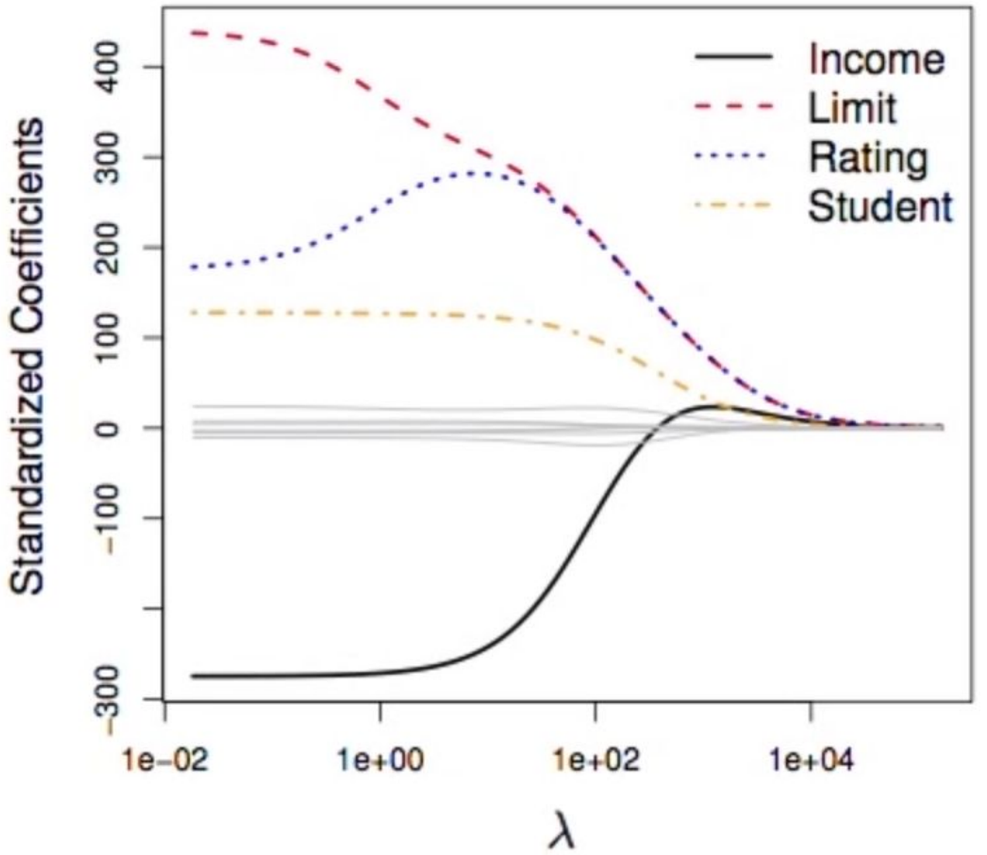
We seek to minimize the squared errors AND some penalty term, whose power is equal to lambda.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately.

RIDGE REGRESSION

Ridge coefficients as a function of our regularization penalty:



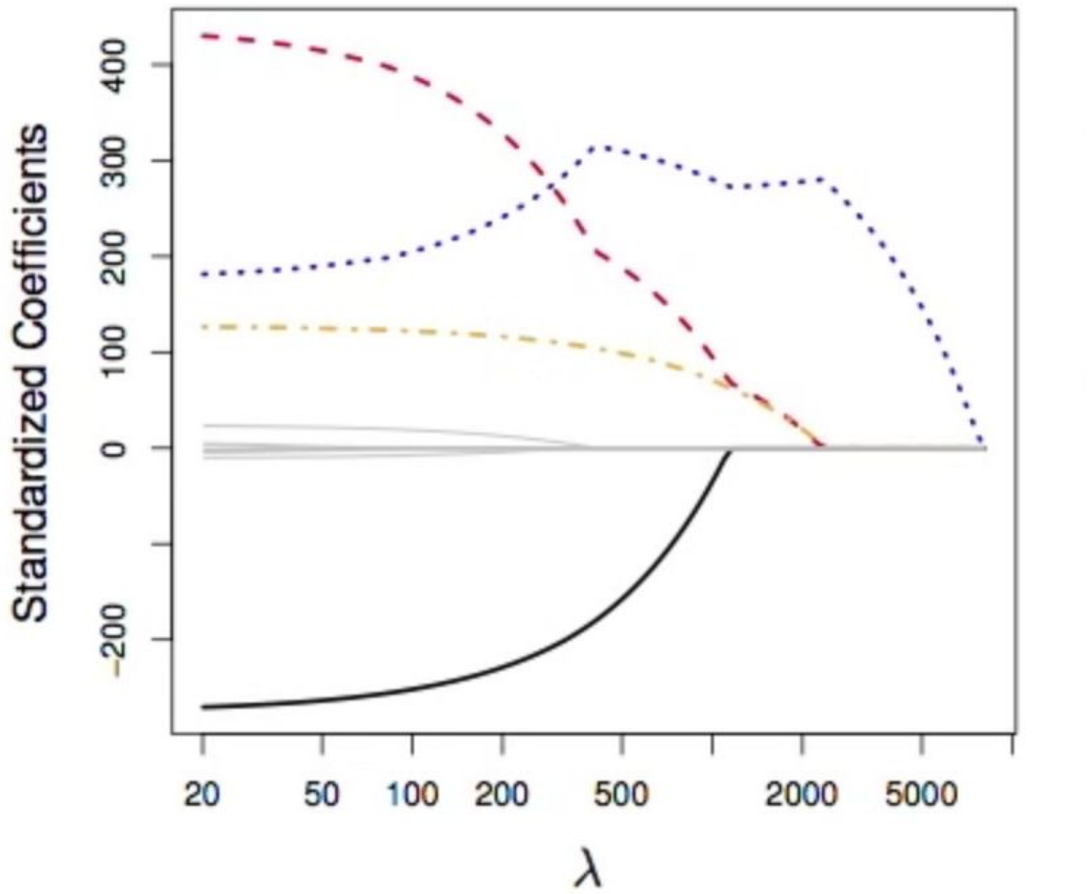
LASSO REGRESSION REGRESSION

We seek to minimize the squared errors AND some penalty term, whose power is equal to lambda.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| \;=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

LASSO REGRESSION REGRESSION

Lasso coefficients as a function of our regularization penalty:



RIDGE VS LASSO REGRESSION

Lasso Regression (L1 norm): shrink towards 0 using the sum of the absolute value of our coefficients as a constraint

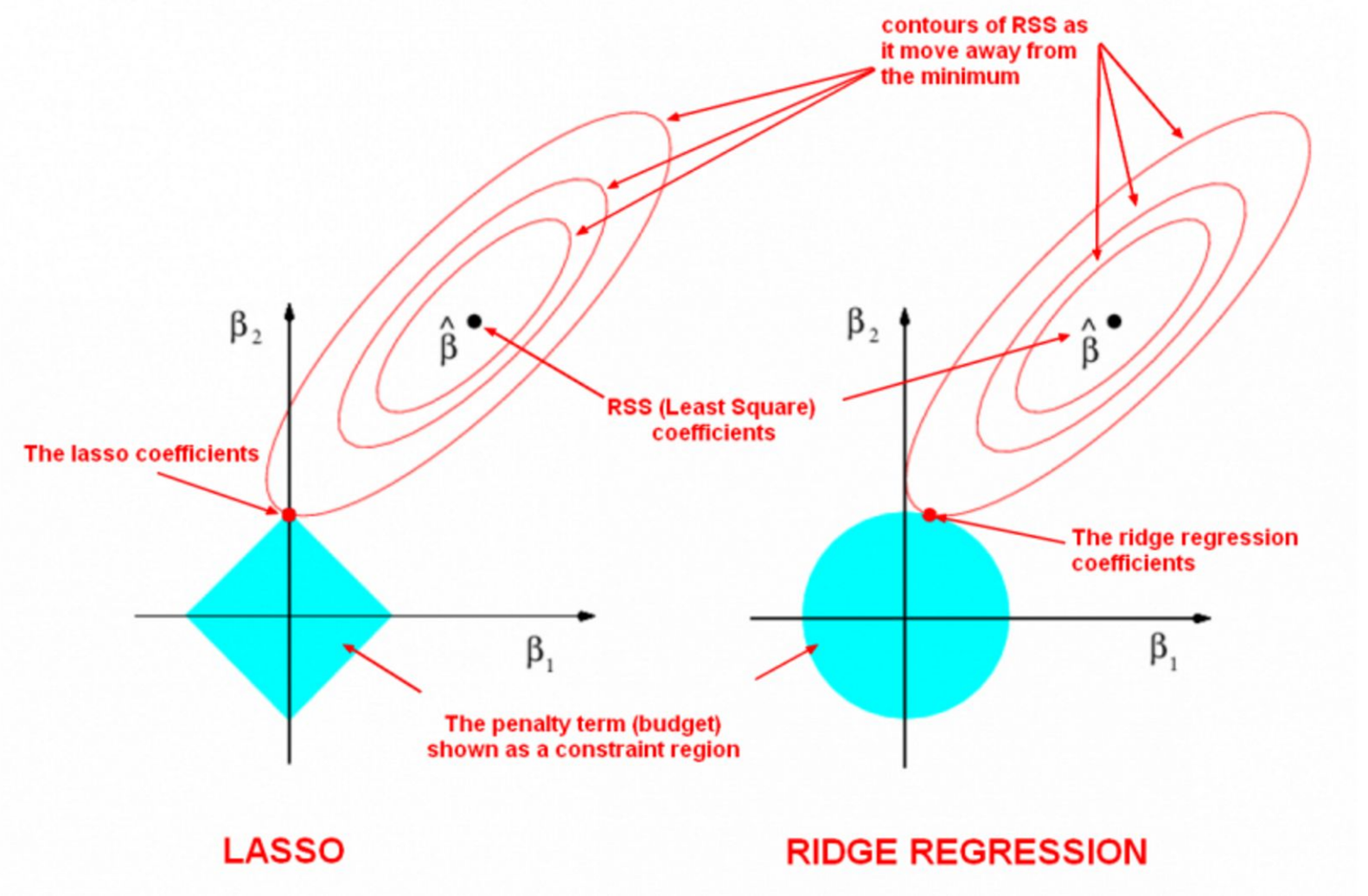
Ridge Regression (L2 norm): shrink the squares of our our coefficients

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq s$$

and

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq s,$$

RIDGE VS LASSO REGRESSION



RIDGE VS LASSO REGRESSION

Previous Slide:

We are fitting a linear regression model with two features, x1 and x2

β represents the set of two coefficients β1 and β2, which minimize the RSS for an **unregularized** model

Regularization restricts the allowed positions of the β to the blue constraint region:

For lasso, this region is a diamond because it constrains the absolute value of coefficients

For ridge, this region is a circle because it constrains the square of the coefficients

The size of the blue region is determined by α (our budget!), with a smaller α resulting in a larger region:

When α is zero, the blue region is infinitely large, and thus the coefficient size are not constrained.

When α increases, the blue region gets smaller and smaller.

Ridge regression (L2 norm): shrinks the squares. Lasso (L1 norm) shrinks the absolute value.

RIDGE VS LASSO REGRESSION

But one more thing!

Should features be standardized?

Yes, because otherwise, features would be penalized simply because of their scale.

Also, standardizing avoids penalizing the intercept, which wouldn't make intuitive sense.

How should you choose between Lasso regression and Ridge regression?

Lasso regression is preferred if we believe many features are irrelevant or if we prefer a sparse model.

If model performance is your primary concern, it is best to try both.

ElasticNet regression is a combination of lasso regression and ridge Regression.

THE METHODS ARE NEW AND IMPROVING

In the following informative video:

Derek Zoolander is traditional regression methods.

Hansel is regularized methods.

https://www.youtube.com/watch?v=CV_hDyfmEw4

Tibshirani:

https://www.youtube.com/watch?v=cSKzqb0EKS0&list=PL5-da3qGB5IB-Xdpj_uXJpLGiRfv9UVXI&index=6&t=1m3s

DEMO

UNDERSTANDING REGULARIZATION EFFECTS

QUICK CHECK

- We are working with the bikeshare data to predict riders over hours/days with a few features.
- Does it make sense to use a ridge regression or a lasso regression?
- Why?

UNDERSTANDING REGULARIZATION EFFECTS

- Let's test a variety of alpha weights for Ridge Regression on the bikeshare data.

```
alphas = np.logspace(-10, 10, 21)
for a in alphas:
    print 'Alpha:', a
    lm = linear_model.Ridge(alpha=a)
    lm.fit(modeldata, y)
    print lm.coef_
    print metrics.mean_squared_error(y, lm.predict(modeldata))
```

- What happens to the weights of the coefficients as alpha increases? What happens to the error as alpha increases?

WE CAN MAKE THIS EASIER WITH GRID SEARCH!

- Grid search exhaustively searches through all given options to find the best solution. Grid search will try all combos given in `param_grid`.

```
param_ grid = {
    'intercept': [True, False],
    'alpha': [1, 2, 3],
}
```

WE CAN MAKE THIS EASIER WITH GRID SEARCH!

- This param grid has six different options:
  - intercept True, alpha 1
  - intercept True, alpha 2
  - intercept True, alpha 3
  - intercept False, alpha 1
  - intercept False, alpha 2
  - intercept False, alpha 3

```
param_ grid = {
    'intercept': [True, False],
    'alpha': [1, 2, 3],
}
```

WE CAN MAKE THIS EASIER WITH GRID SEARCH!

- This is an incredibly powerful, automated machine learning tool!

```
from sklearn import grid_search

alphas = np.logspace(-10, 10, 21)
gs = grid_search.GridSearchCV(
    estimator=linear_model.Ridge(),
    param_grid={'alpha': alphas},
    scoring='mean_squared_error')
```

WE CAN MAKE THIS EASIER WITH GRID SEARCH!

```
gs.fit(modeldata, y)

print -gs.best_score_ # mean squared error here comes in negative, so
let's make it positive.
print gs.best_estimator_ # explains which grid_search setup worked
best
print gs.grid_scores_ # shows all the grid pairings and their
performances.
```

GUIDED PRACTICE

GRID SEARCH CV, SOLVING FOR ALPHA

ACTIVITY: GRID SEARCH CV, SOLVING FOR ALPHA

DIRECTIONS (25 minutes)

1. Modify the previous code to do the following:
    a. Introduce cross validation into the grid search. This is accessible from the cv argument.
    b. Add fit_intercept = True and False to the param_grid dictionary.
    c. Re-investigate the best score, best estimator, and grid score attributes as a result of the grid search.

DELIVERABLE

New code and output that meets above requirements

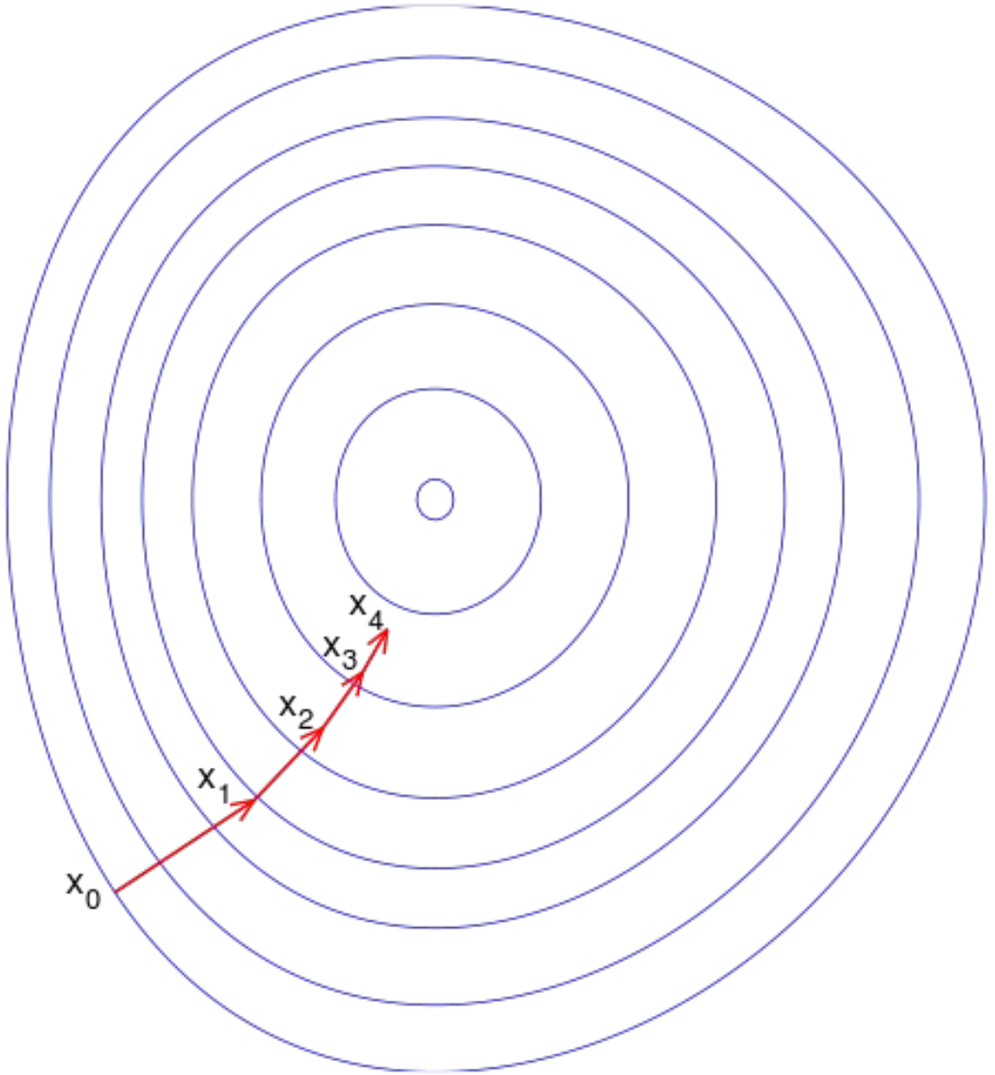
MINIMIZING LOSS THROUGH GRADIENT DESCENT

GRADIENT DESCENT

- Gradient Descent can also help us minimize error.

- How Gradient Descent works:
  - A random linear solution is provided as a starting point
  - The solver attempts to find a next “step”: take a step in any direction and measure the performance.
  - If the solver finds a better solution (i.e. lower MSE), this is the new starting point.
  - Repeat these steps until the performance is optimized and no “next steps” perform better. The size of steps will shrink over time.

GRADIENT DESCENT



A CODE EXAMPLE OF GRADIENT DESCENT

```
num_to_approach, start, steps, optimized = 6.2, 0., [-1, 1], False
while not optimized:
    current_distance = num_to_approach - start
    got_better = False
    next_steps = [start + i for i in steps]
    for n in next_steps:
        distance = np.abs(num_to_approach - n)
        if distance < current_distance:
            got_better = True
            print distance, 'is better than', current_distance
            current_distance = distance
            start = n
```

A CODE EXAMPLE OF GRADIENT DESCENT

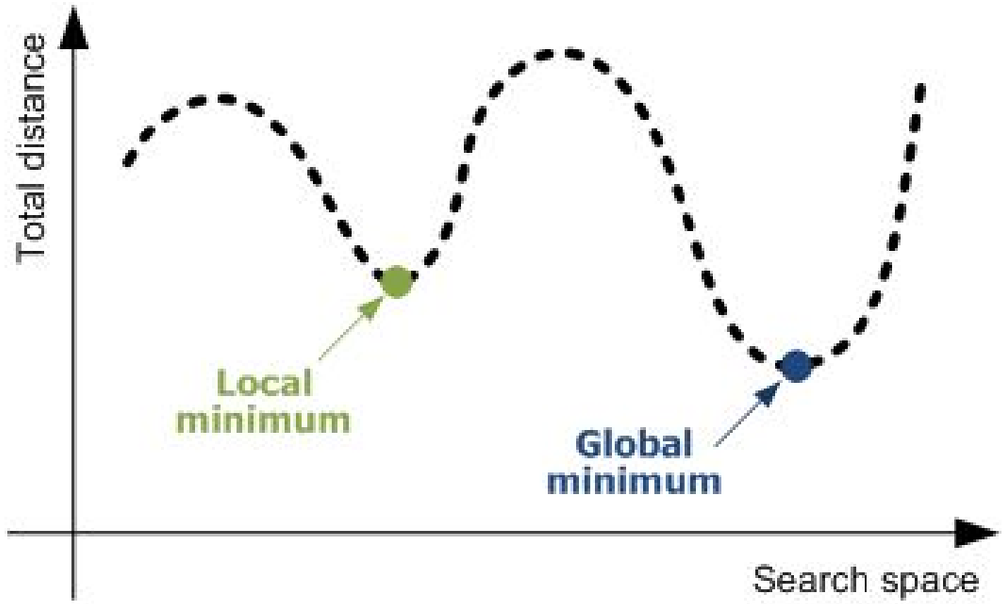
```
    if got_better:
        print 'found better solution! using', current_distance
        a += 1
    else:
        optimized = True
        print start, 'is closest to', num_to_approach
```

- What is the code doing? What could go wrong?

GLOBAL VS LOCAL MINIMUMS

- Gradient Descent could solve for a *local* minimum instead of a *global* minimum.

- A *local* minimum is confined to a very specific subset of solutions. The *global* minimum considers all solutions. These could be equal, but that's not always true.



DEMO

APPLICATION OF GRADIENT DESCENT

APPLICATION OF GRADIENT DESCENT

- Gradient Descent works best when:
  - We are working with a large dataset. Smaller datasets are more prone to error.
  - Data is cleaned up and normalized.
- Gradient Descent is significantly faster than OLS. This becomes important as data gets bigger.

APPLICATION OF GRADIENT DESCENT

- We can easily run a Gradient Descent regression.

- Note: The verbose argument can be set to 1 to see the optimization steps.

```
lm = linear_model.SGDRegressor()
lm.fit(modeldata, y)
print lm.score(modeldata, y)
print metrics.mean_squared_error(y, lm.predict(modeldata))
```

- Untuned, how well did gradient descent perform compared to OLS?

APPLICATION OF GRADIENT DESCENT

- Gradient Descent can be tuned with
  - the learning rate: how aggressively we solve the problem
  - epsilon: at what point do we say the error margin is acceptable
  - iterations: when should be we stop no matter what

INDEPENDENT PRACTICE

ON YOUR OWN

ACTIVITY: ON YOUR OWN

DIRECTIONS (30 minutes)

There are tons of ways to approach a regression problem.

1. Implement the Gradient Descent approach to our bikeshare modeling problem.
2. Show how Gradient Descent solves and optimizes the solution.
3. Demonstrate the grid_search module.
4. Use a model you evaluated last class or the simpler one from today. Implement param_grid in grid search to answer the following questions:
    a. With a set of values between 10^-10 and 10^-1, how does MSE change?
    b. Our data suggests we use L1 regularization. Using a grid search with l1_ratios between 0 and 1, increasing every 0.05, does this statement hold true? If not, did gradient descent have enough iterations to work properly?
    c. How do these results change when you alter the learning rate?

DELIVERABLE

Gradient Descent approach and answered questions

ACTIVITY: ON YOUR OWN

Starter Code

```
params = {} # put your gradient descent parameters here
gs = grid_search.GridSearchCV(
    estimator=linear_model.SGDRegressor(),
    cv=cross_validation.KFold(len(modeldata), n_folds=5, shuffle=True),
    param_grid=params,
    scoring='mean_squared_error',
    )

gs.fit(modeldata, y)

print 'BEST ESTIMATOR'
print -gs.best_score_
print gs.best_estimator_
print 'ALL ESTIMATORS'
print gs.grid_scores_
```

TOPIC REVIEW

LESSON REVIEW

- What's the (typical) range of r-squared?
- What's the range of mean squared error?
- How would changing the scale or interpretation of y (your target variable) effect mean squared error?
- What's cross validation, and why do we use it in machine learning?
- What is error due to bias? What is error due to variance? Which is better for a model to have, if it had to have one?
- How does gradient descent try a different approach to minimizing error?