GENERAL ASSEMBLY

EVALUATING MODEL FIT

Chirayu Sariya

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-SQUARED 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 % of variance in y that is explained by variance in x.
- 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

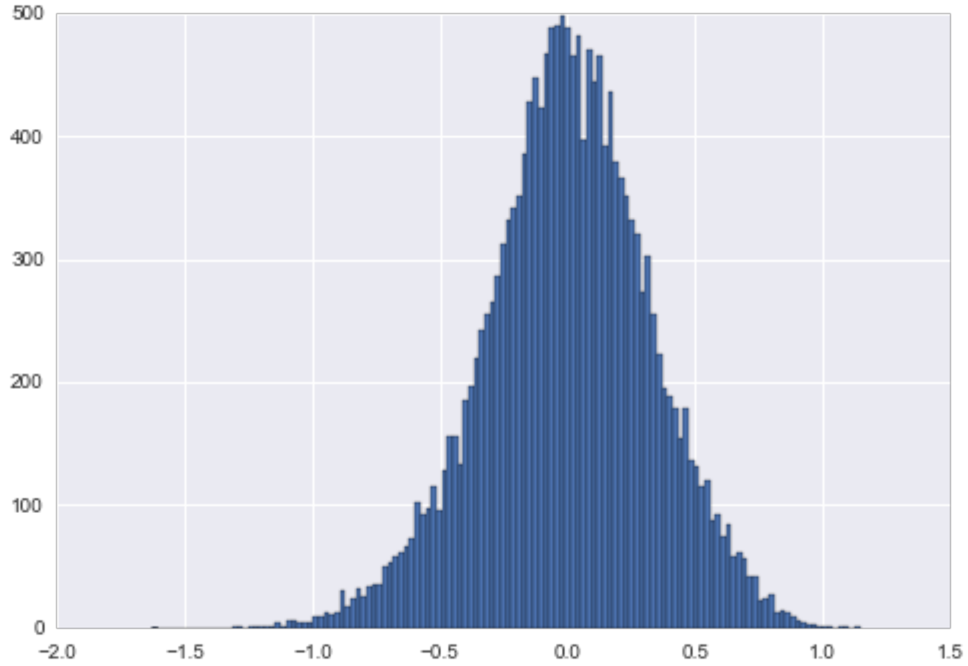
RECALL: WHAT'S RESIDUAL ERROR?

- 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.

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

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.

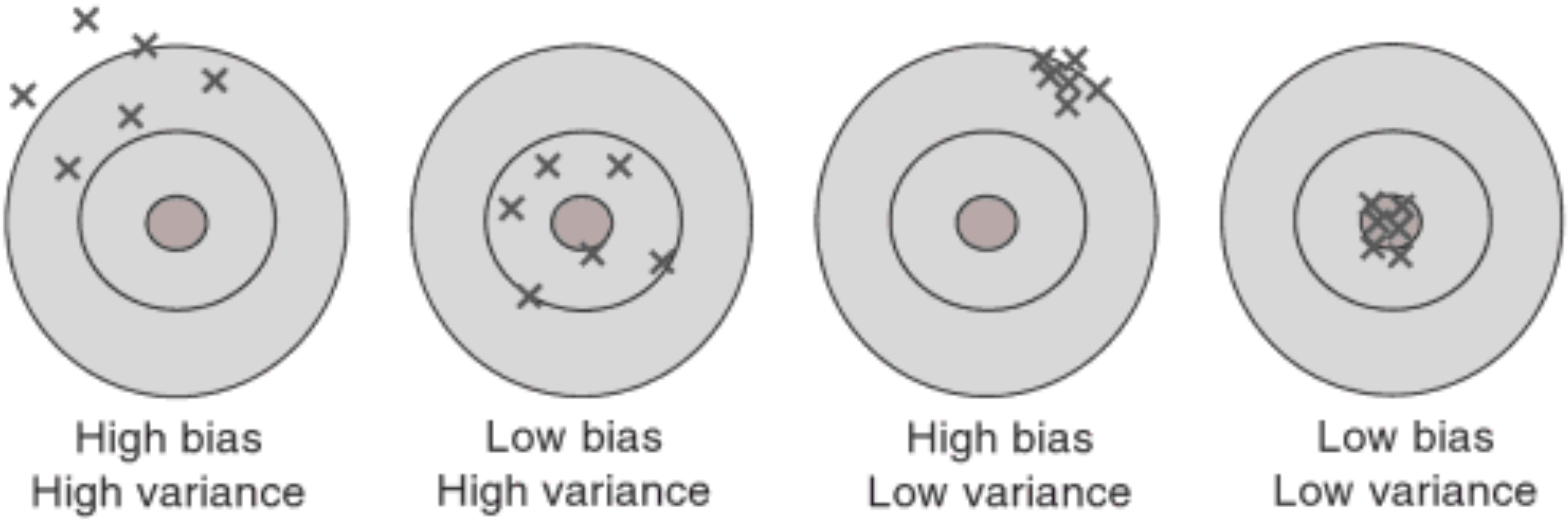
```
from sklearn import metrics
metrics.mean_squared_error([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])
0.0
```

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
```

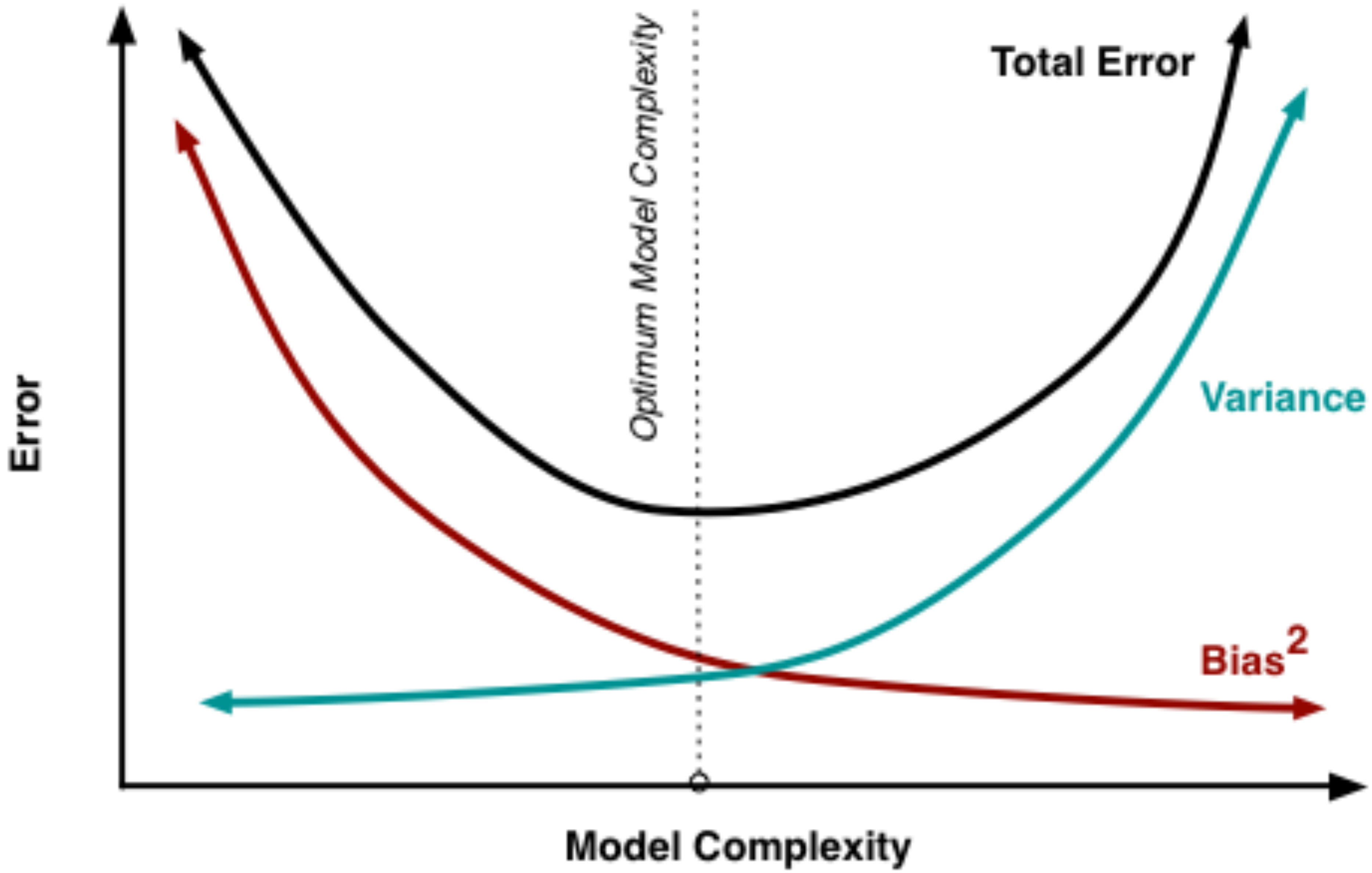
BIAS VS. VARIANCE



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



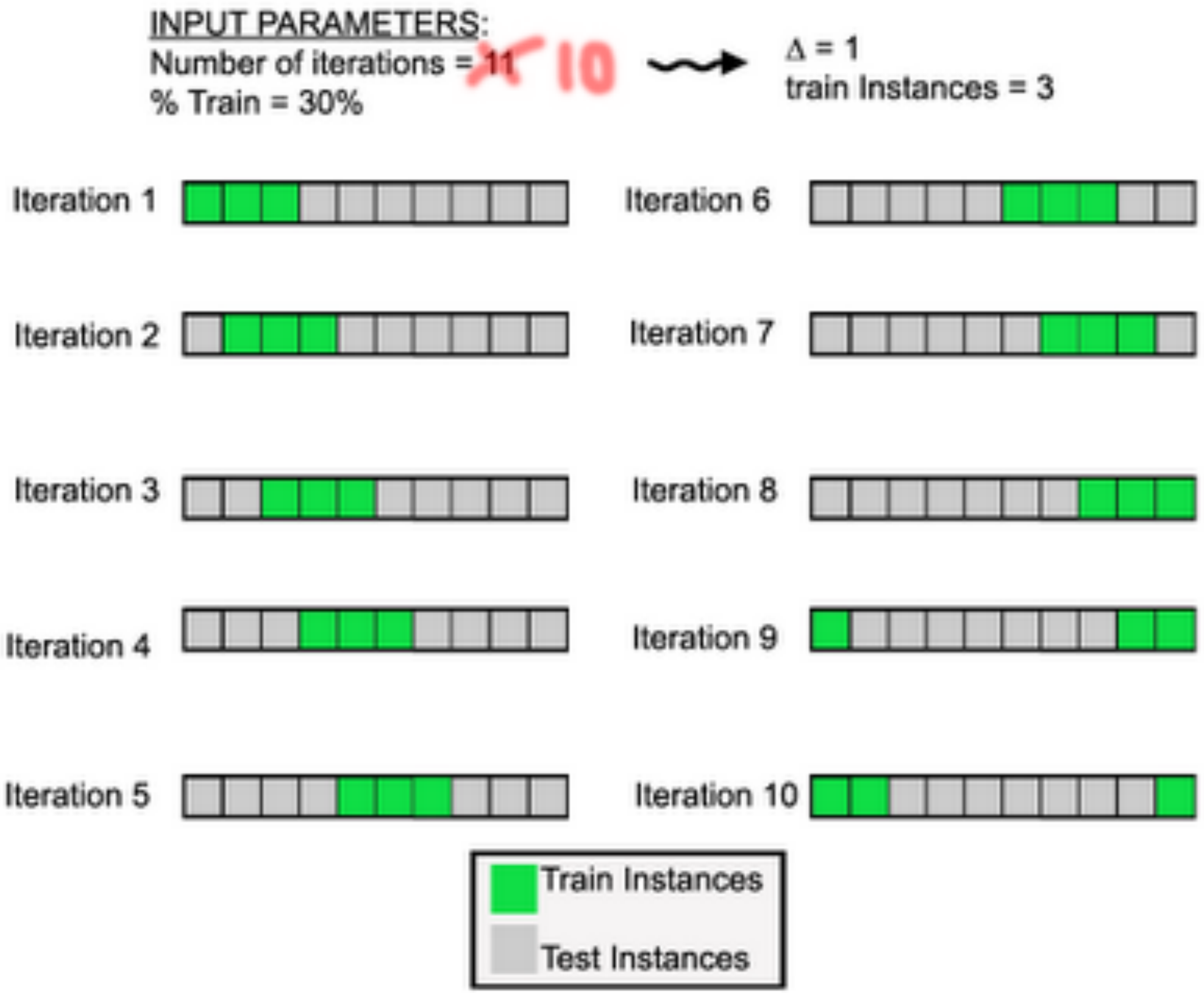
DEMO

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



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

EXERCISE

DIRECTIONS (20 minutes)

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

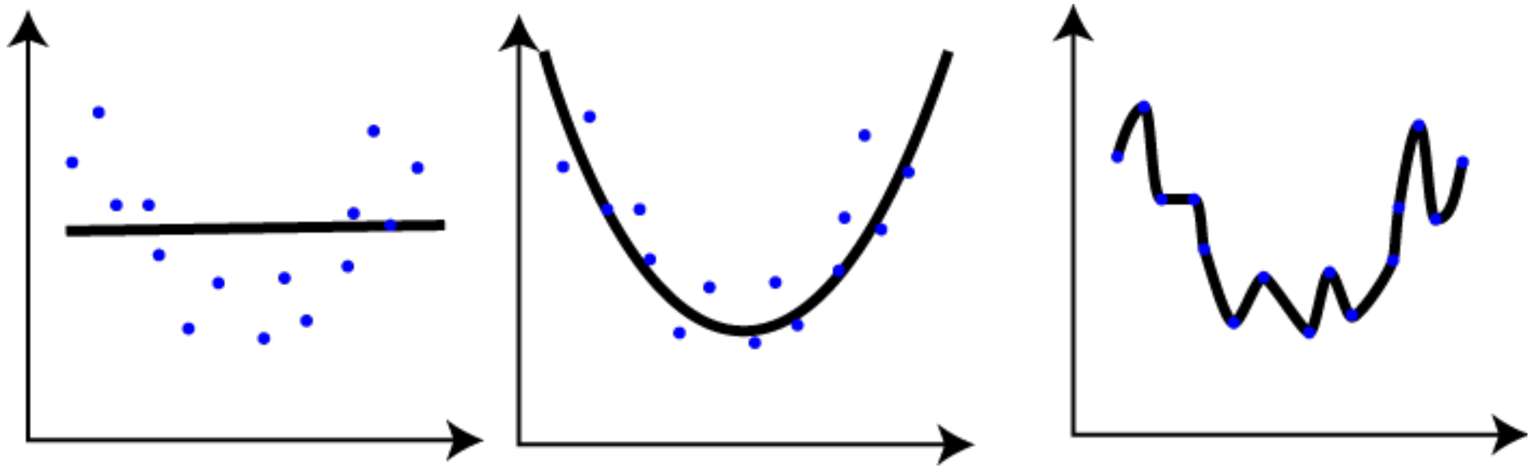
1. Using the previous code example, perform k-fold cross validation for all even numbers between 2 and 50.
2. Answer the following questions:
   a. What does `shuffle=True` do?
   b. At what point does cross validation no longer seem to help the model?
3. Hint: `range(2, 51, 2)` produces a list of even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

DELIVERABLE

Answers to questions

REGULARIZATION AND CROSS VALIDATION

WHAT IS OVERFITTING?



- The first model poorly explains the data.
- The second model explains the general curve of the data.
- The third model drastically overfits the model, bending to every point.
- Regularization helps prevent the third model.

WHAT IS REGULARIZATION? AND WHY DO WE USE IT?

- Regularization is an additive approach to protect models against overfitting (being potentially biased and overconfident, not generalizing well).

- Regularization becomes an additional weight to coefficients, shrinking them closer to zero.

- L1 (Lasso Regression) adds the extra weight to coefficients.

- L2 (Ridge Regression) adds the square of the extra weight to coefficients.

- Use Lasso when we have more features than observations ($k > n$) and Ridge otherwise.

WHERE REGULARIZATION MAKES SENSE

- What happens to MSE if use Lasso or Ridge Regression directly?

```
lm = linear_model.LinearRegression().fit(modeldata, y)
print metrics.mean_squared_error(y, lm.predict(modeldata))
lm = linear_model.Lasso().fit(modeldata, y)
print metrics.mean_squared_error(y, lm.predict(modeldata))
lm = linear_model.Ridge().fit(modeldata, y)
print metrics.mean_squared_error(y, lm.predict(modeldata))
```

```
1672.58110765 # OLS
1725.41581608 # L1
1672.60490113 # L2
```

WHERE REGULARIZATION MAKES SENSE

- It doesn’t seem to help. Why is that?
- We need to optimize the regularization weight parameter (called alpha) through cross validation.

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS (5 minutes)

1. Why is regularization important?
2. What does it protect against and how?

DELIVERABLE

Answers to the above questions

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

EXERCISE

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

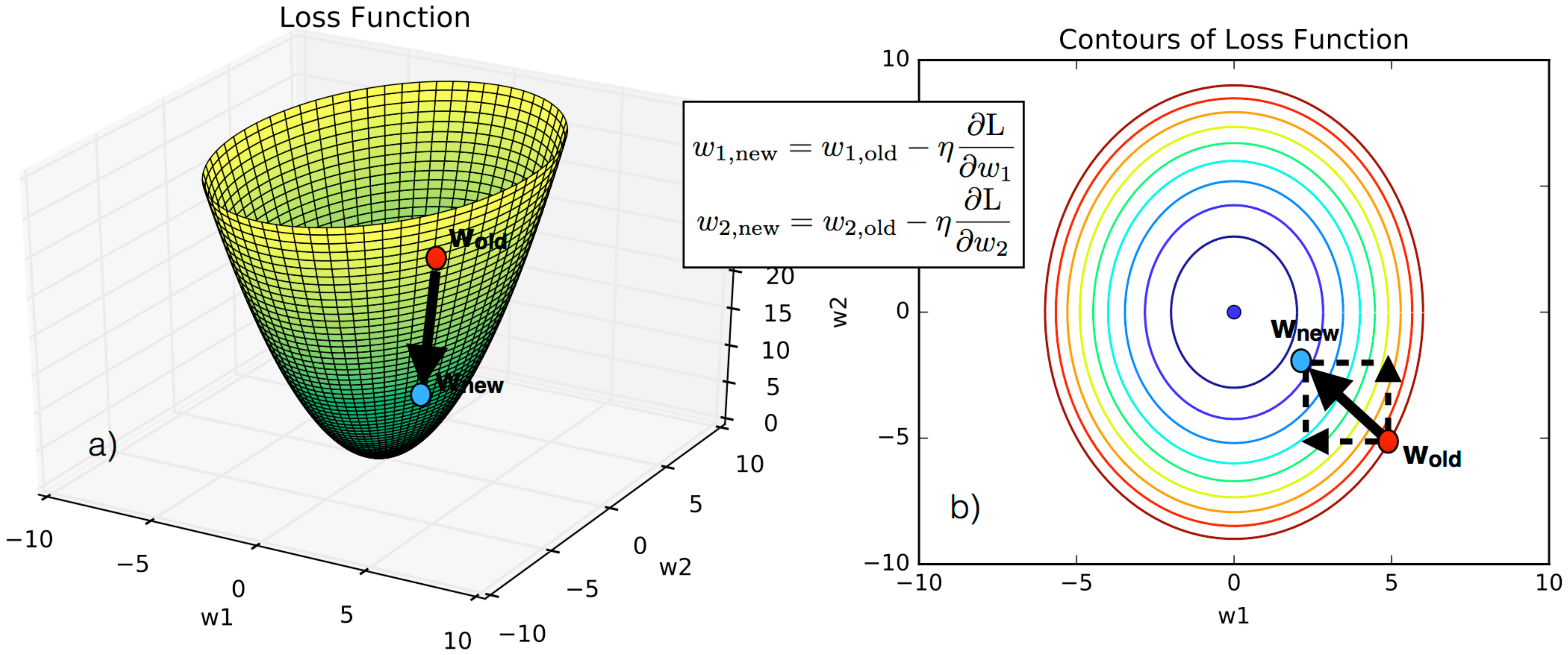
New code and output that meets above requirements

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
  - Takes steps proportional to the negative of the gradient of the cost/loss function at the current point until it converges to a minimum.

GRADIENT DESCENT



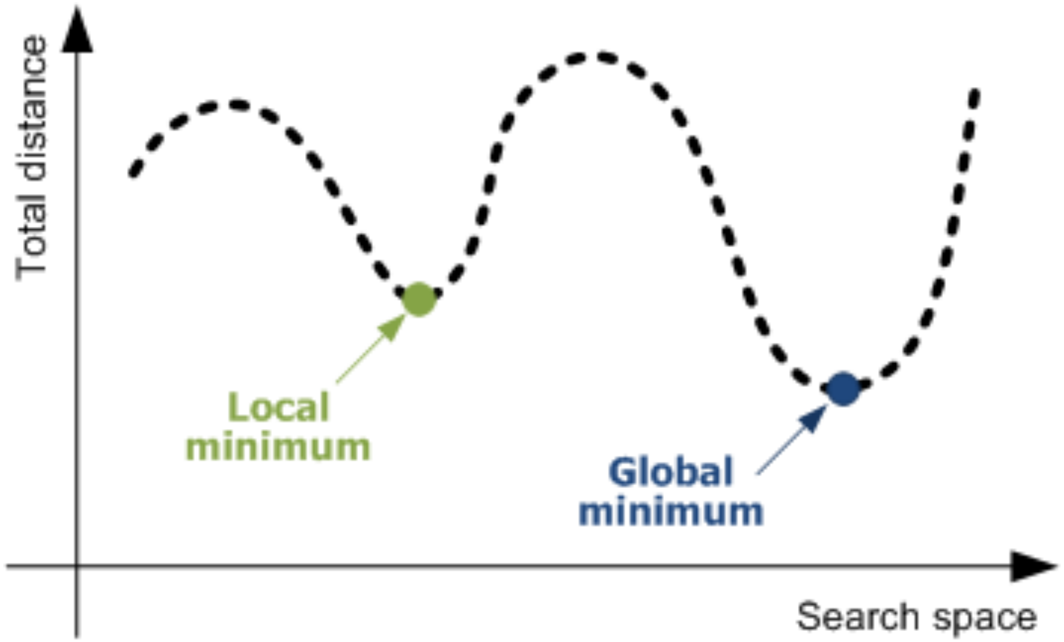
$$w_{1,\text{new}} = w_{1,\text{old}} - \eta \frac{\partial \text{L}}{\partial w_1}$$

$$w_{2,\text{new}} = w_{2,\text{old}} - \eta \frac{\partial \text{L}}{\partial w_2}$$

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 for large number of features. 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

ON YOUR OWN

ACTIVITY: ON YOUR OWN

EXERCISE

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

EXERCISE

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?
- 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?

COURSE

BEFORE NEXT CLASS

DUE DATE

- Homework:
- Project: Final Project, Deliverable 1

LESSON

Q & A

EXIT TICKET

DON'T FORGET TO FILL OUT YOUR EXIT TICKET