LEARNING OBJECTIVES

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

PRE-WORK

PRE-WORK REVIEW

- Understand goodness of fit (R-Squared)
- Measure statistical significance of features
- Recall what a residual is
- Implement a scikit-learn estimator to predict a target variable

OPENING

R-SQUARED AND RESIDUALS

OPENING

WHAT IS R-SQUARED? WHAT IS A RESIDUAL?

- R-Squared is 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 will explore loss functions and find ways to refine our model

WHAT IS R-SQUARED? WHAT IS A RESIDUAL?

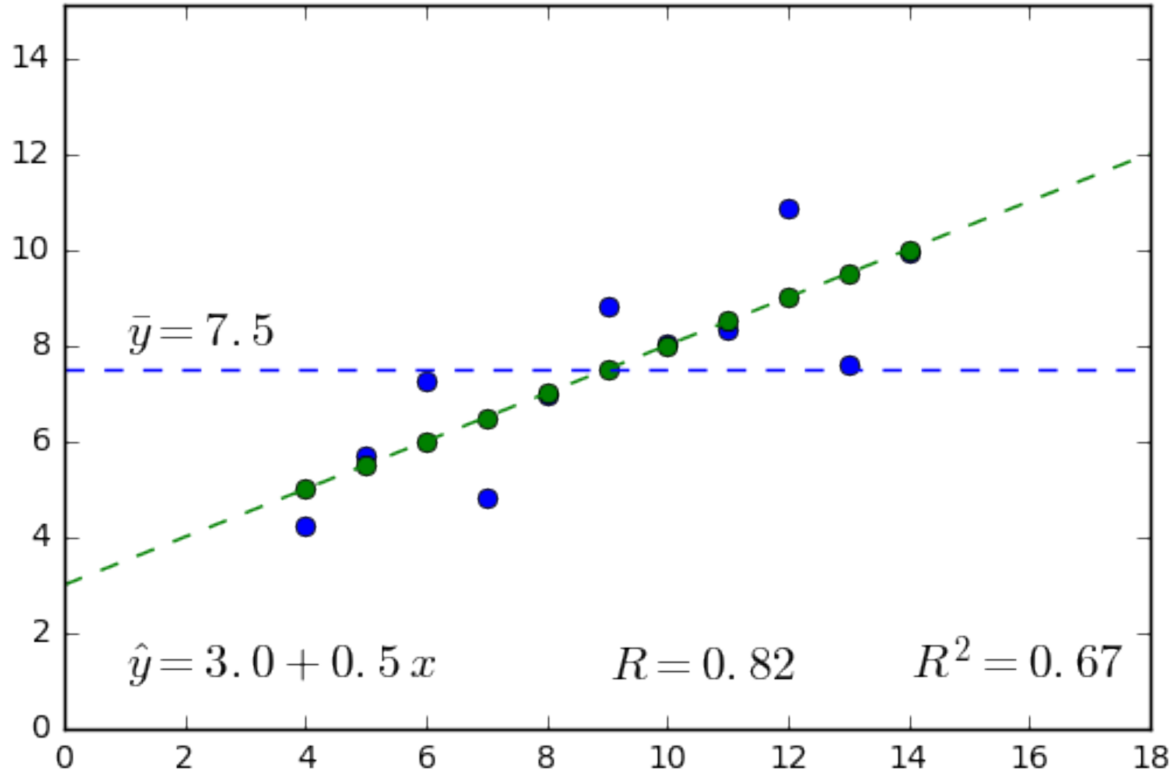
- In a data set with n cases where the outputs are $y_i = y_1, \cdots, y_n$
- And the predicted values are (note the y-**hat**) $\hat{y}_i = \hat{y}_1, \cdots, \hat{y}_n$

- The mean of y is (note the y-**bar**)

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$SS_{tot} = SS_{reg} + SS_{res}$$

$$R^2 = \frac{SS_{reg}}{SS_{tot}}$$



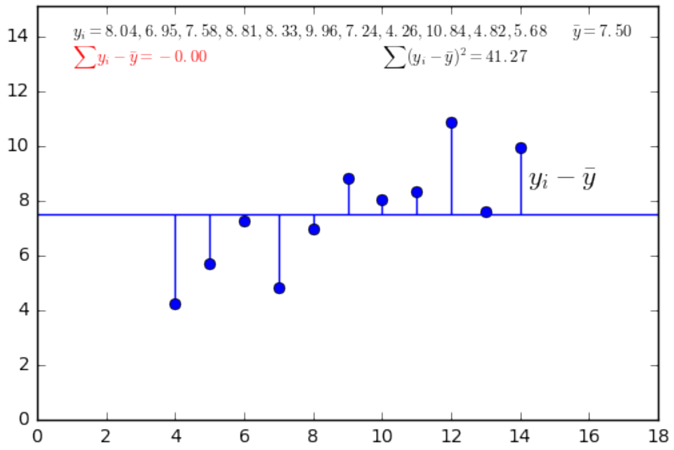
WHAT IS R-SQUARED? WHAT IS A RESIDUAL?

- The Sum of Squares

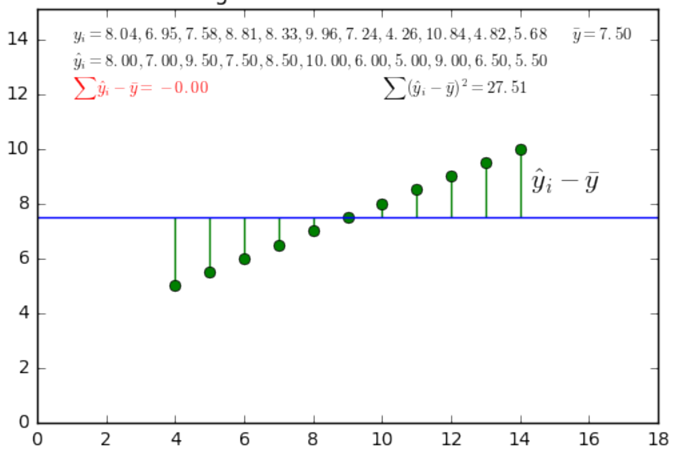
Total SS = **Regression SS** + **Residual SS**

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \qquad SS_{reg} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 \qquad SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

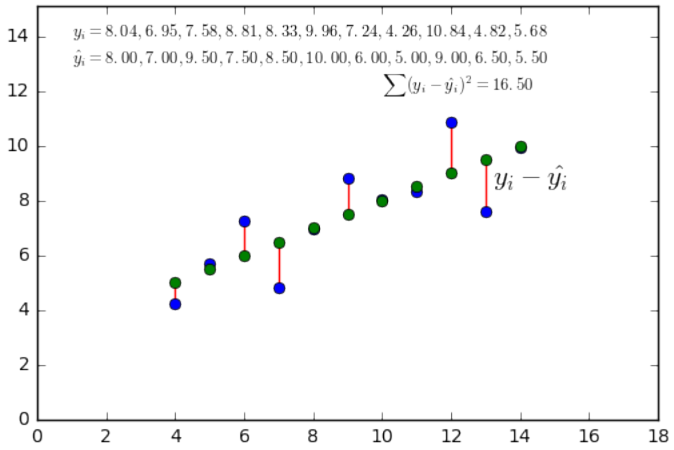
Deviation: Actual - Mean



Regression: Predicted - Mean



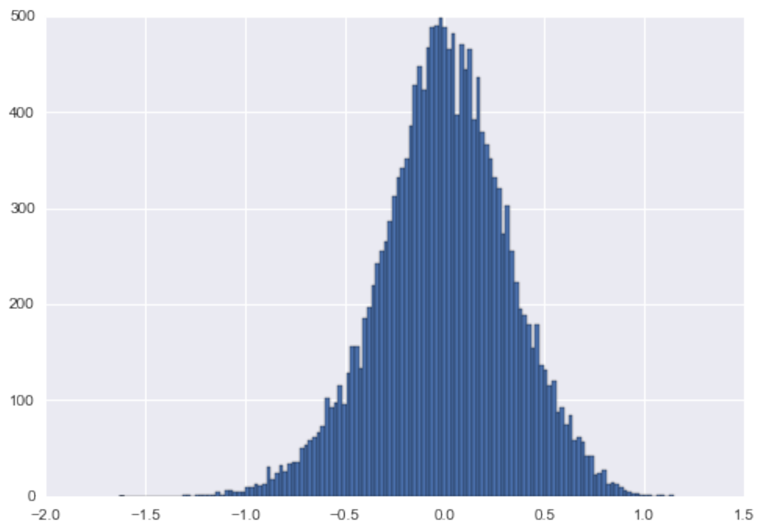
Residual: Actual - Predicted



INTRODUCTION

LINEAR MODELS AND ERROR

RECALL: WHAT IS 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 does not provide an overall picture for optimisation
- We need a metric to summarise the error in our model into one value
  - **Mean Square Error**: the mean residual error in our model

MEAN SQUARE 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

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

MEAN SQUARE ERROR (MSE)

- scikit-learn's metrics module includes a mean_squared_error() function
- For example, two arrays of the same values would have an MSE of 0
- Two arrays with different values would have a positive MSE

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

print(metrics.mean_squared_error([1, 2, 3, 4, 5], [1, 2, 3, 4, 5]))
0.0

print(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 MINIMISE THE ERROR?

- The regression method we have 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

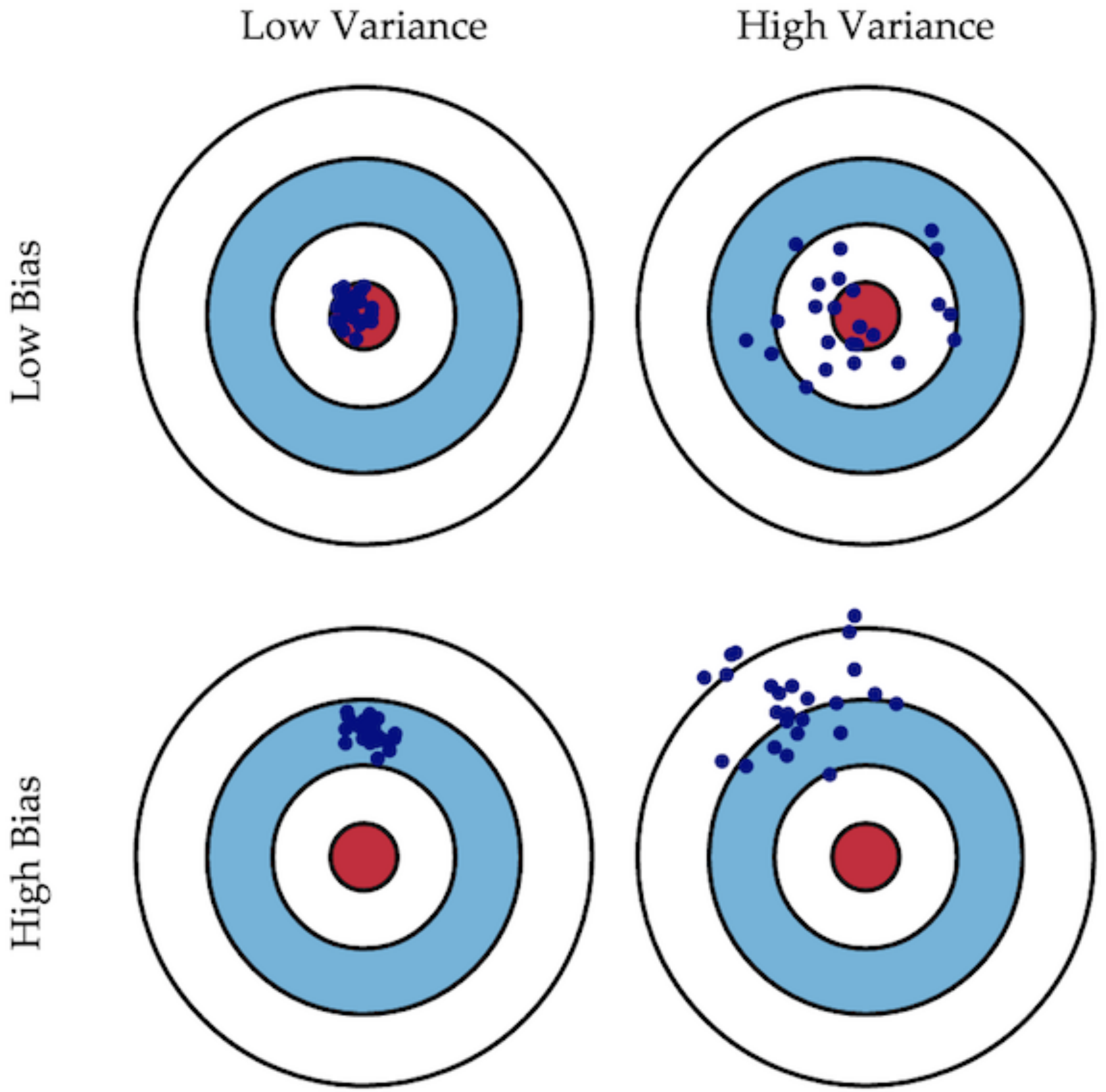
LET'S COMPARE TWO RANDOM MODELS

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

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

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

df["x"], df["y"] = append_jitter(df.x), append_jitter(df.y)
biased_df["x"], biased_df["y"] = append_jitter(biased_df.x), 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"]])))
```

0.175065131142

0.187115987374

BIAS VS VARIANCE



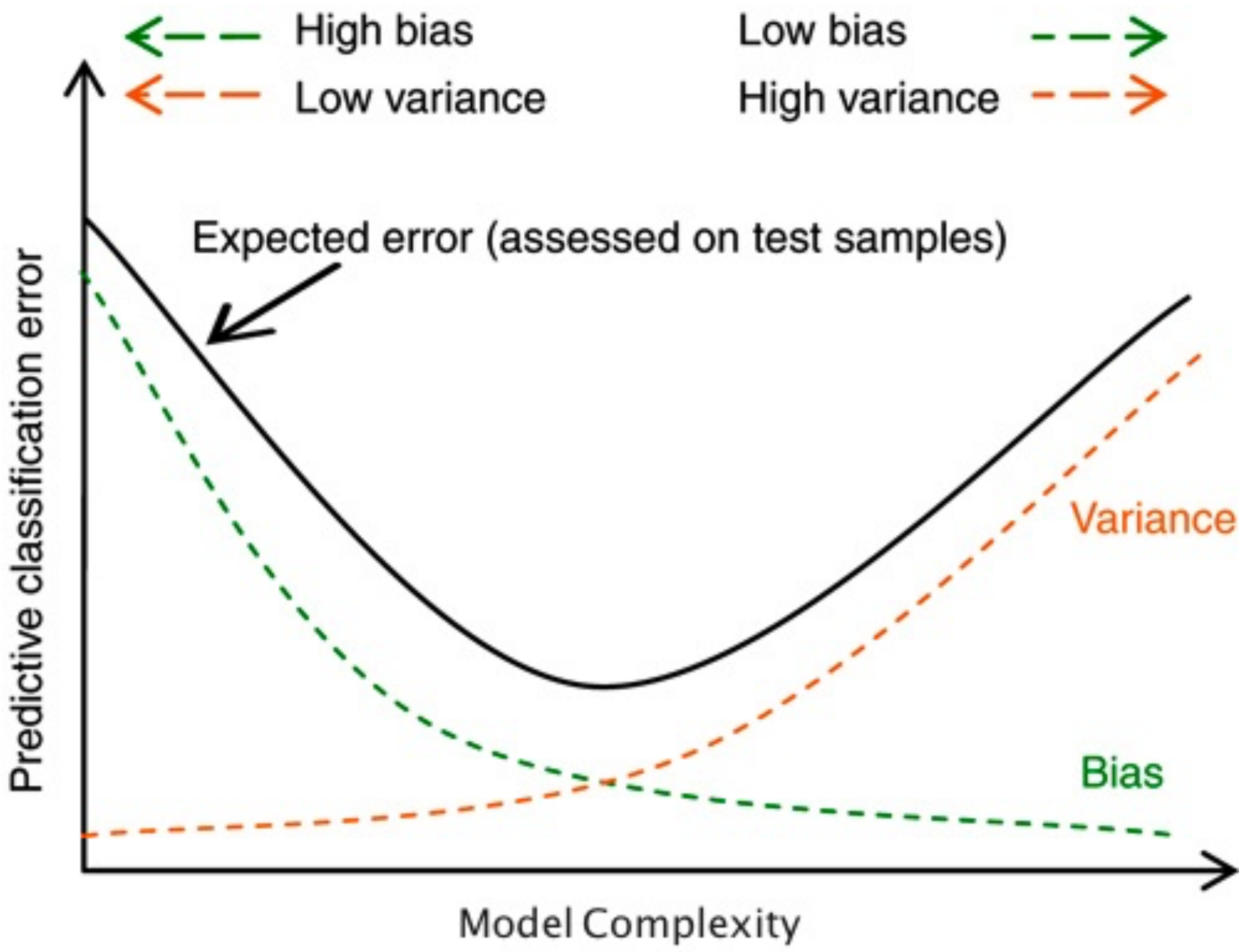
BIAS / VARIANCE TRADEOFF

- 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 generalised 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 generalise to data it has not seen even if does not perform as well on data it has already seen

BIAS / VARIANCE TRADEOFF



ACTIVITY: KNOWLEDGE CHECK

EXERCISE

DIRECTIONS: ANSWER THE FOLLOWING QUESTIONS (5 MINUTES)

1. Which of the following scenarios would be better for a weather reporter?
   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

DEMONSTRATION

CROSS VALIDATION

CROSS VALIDATION

- Cross Validation can help account for bias

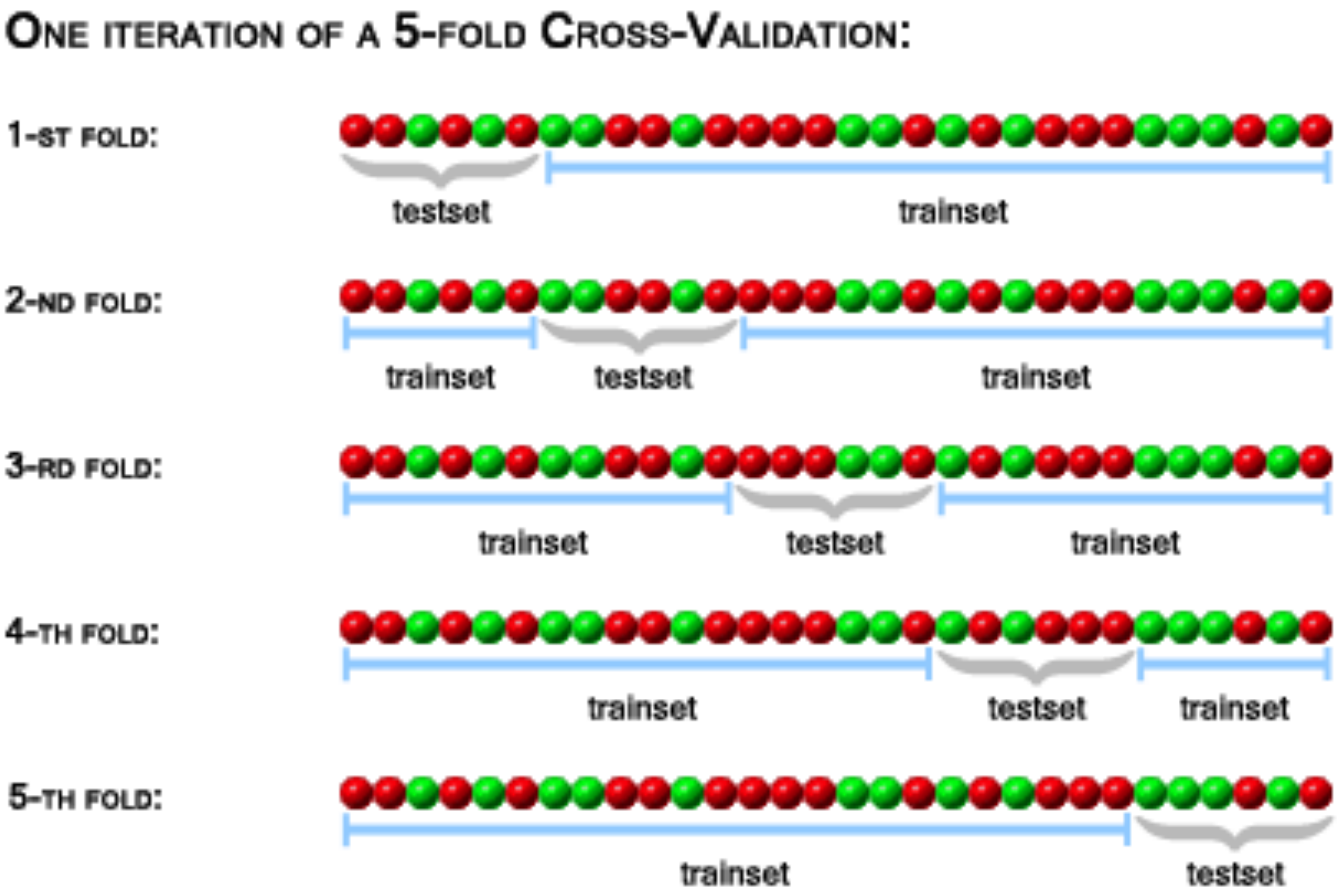
- The general idea is to
  - Generate several models on different cross sections of the data
  - Measure the performance of each
  - Take the mean performance

- This technique swaps bias error for generalised error, describing previous trends accurately enough to extend to future trends

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

CROSS VALIDATION



USING K-FOLD CROSS VALIDATION WITH MSE

- Import the appropriate packages and load data

```
from sklearn import cross_validation, metrics
np.random.seed(seed = 100)

wd         = "../assets/dataset/"
bikeshare = pd.read_csv(wd + "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])))
```

USING K-FOLD CROSS VALIDATION WITH MSE

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

```
print(scores)
print(np.mean(scores))
# This score will be lower
#   but we are trading off bias error for generalised error:
lm = linear_model.LinearRegression().fit(modeldata, y)
print(metrics.mean_squared_error(y, lm.predict(modeldata)))
```

```
[1432.81283012, 1758.18152140, 1706.87054437, 1808.52000170, 1662.48998234]
1673.77497599
1672.58110765
```

- Which approach would predict new data more accurately?

GUIDED PRACTICE

CROSS VALIDATION WITH LINEAR REGRESSION

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

DIRECTIONS: (20 MINUTE)

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

INTRODUCTION

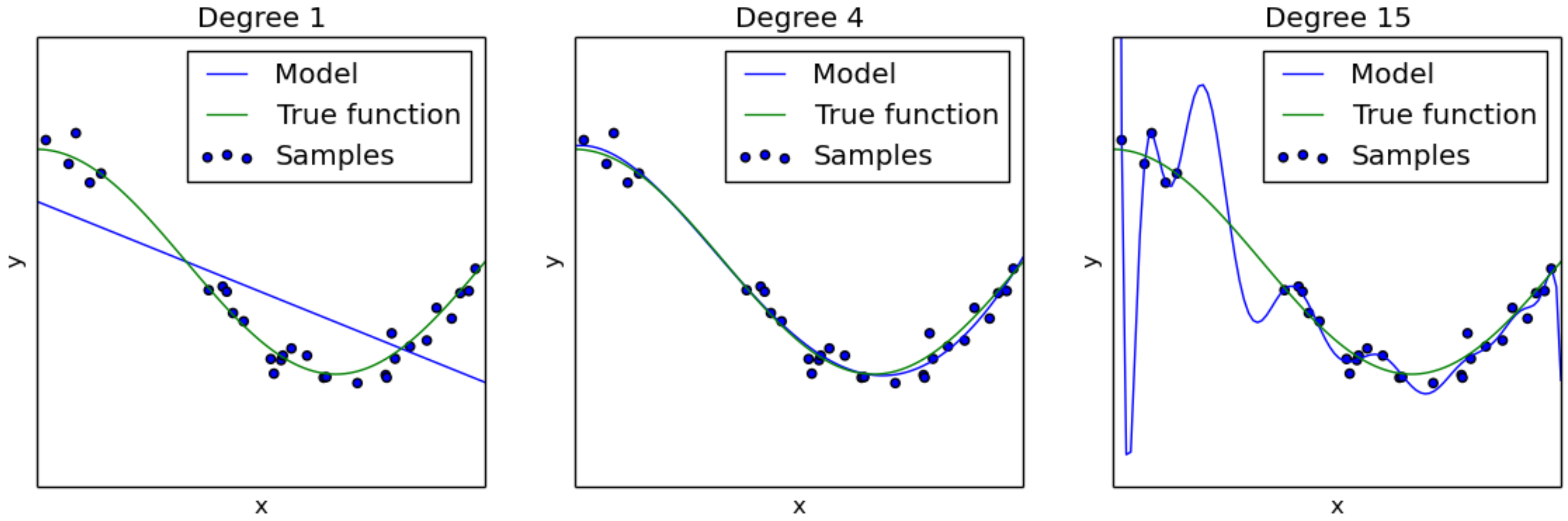
REGULARISATION AND CROSS VALIDATION

WHAT IS REGULARISATION AND WHY USE IT?

- Regularisation is an additive approach to protect models against overfitting (being potentially biased and overconfident, not generalising well)
- Regularisation becomes an additional weight to coefficients, shrinking them closer to zero
- L1 (Lasso Regression) adds the extra weight to coefficients
  - address primarily reducing features by making coefficients zero
- L2 (Ridge Regression) adds the square of the extra weight to coefficients
  - address primarily reducing outliers' influence

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



- Regularisation helps prevent the third model

WHERE REGULARISATION 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 REGULARISATION MAKES SENSE

- It does not seem to help
  - Why is that?

- We need to optimise the regularisation weight parameter (called alpha) through cross validation

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

DIRECTIONS: ANSWER THE FOLLOWING QUESTIONS (5 MINUTES)

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

DEMONSTRATION

UNDERSTANDING REGULARISATION 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 REGULARISATION 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?

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]
}
```

EASIER WITH GRID SEARCH

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

intercept	alpha
True	1
True	2
True	3
False	1
False	2
False	3

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")
```

EASIER WITH GRID SEARCH

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

```
gs.fit(modeldata, y)

# mean squared error here comes in negative, so let's make it positive.
print(-gs.best_score_)

# explains which grid_search setup worked best
print(gs.best_estimator_)

# shows all the grid pairings and their performances
print(gs.grid_scores_)
```

GUIDED PRACTICE

GRID SEARCH CROSS VALIDATION, SOLVING FOR ALPHA

ACTIVITY: GRID SEARCH CROSS VALIDATION, SOLVING FOR ALPHA

EXERCISE

DIRECTIONS: ANSWER THE FOLLOWING QUESTIONS (25 MINUTES)

1. Modify the previous code to do the following:
    a. Introduce cross validation into the grid search
        i. 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

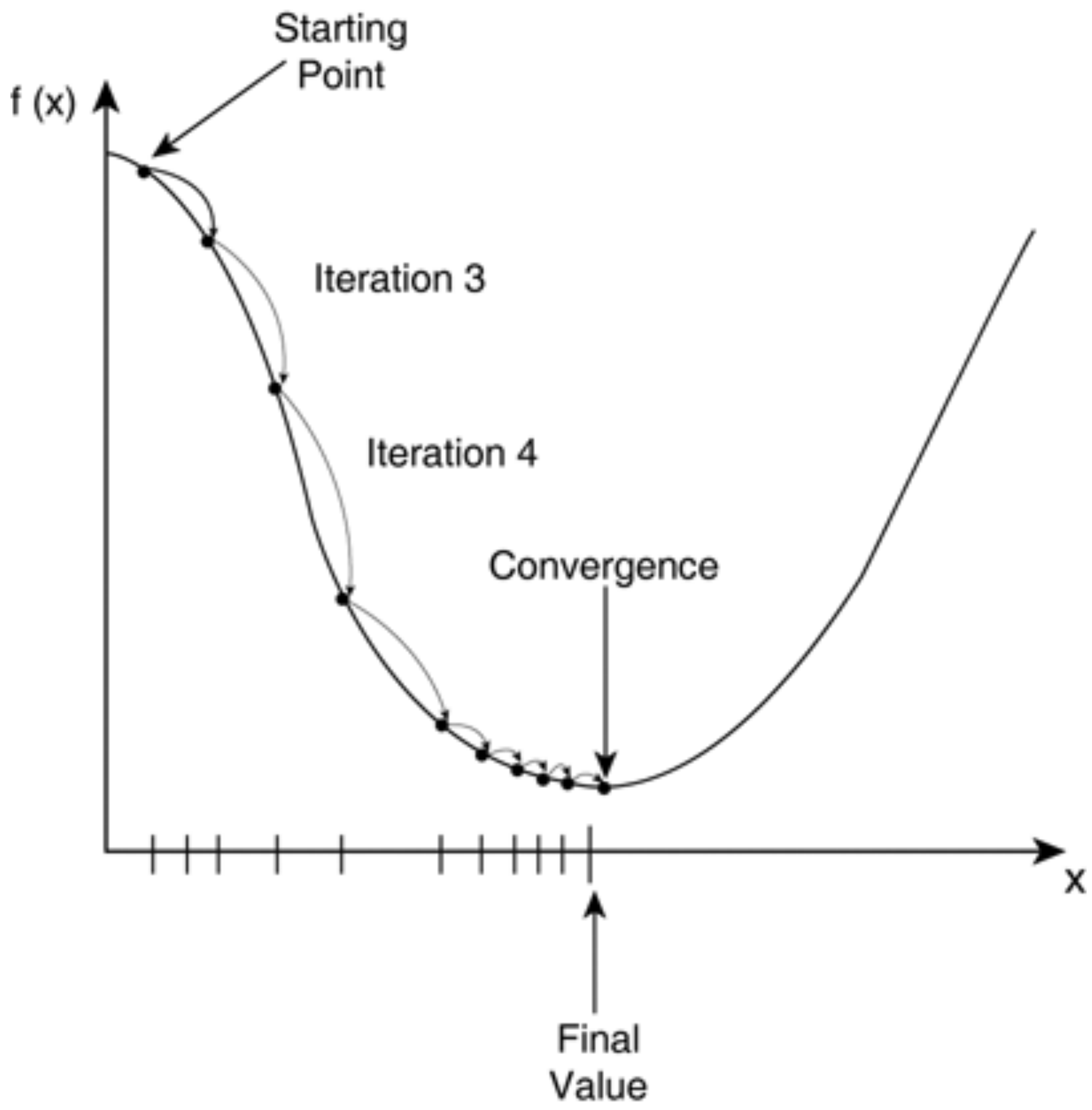
MINIMISING LOSS THROUGH GRADIENT DESCENT

GRADIENT DESCENT

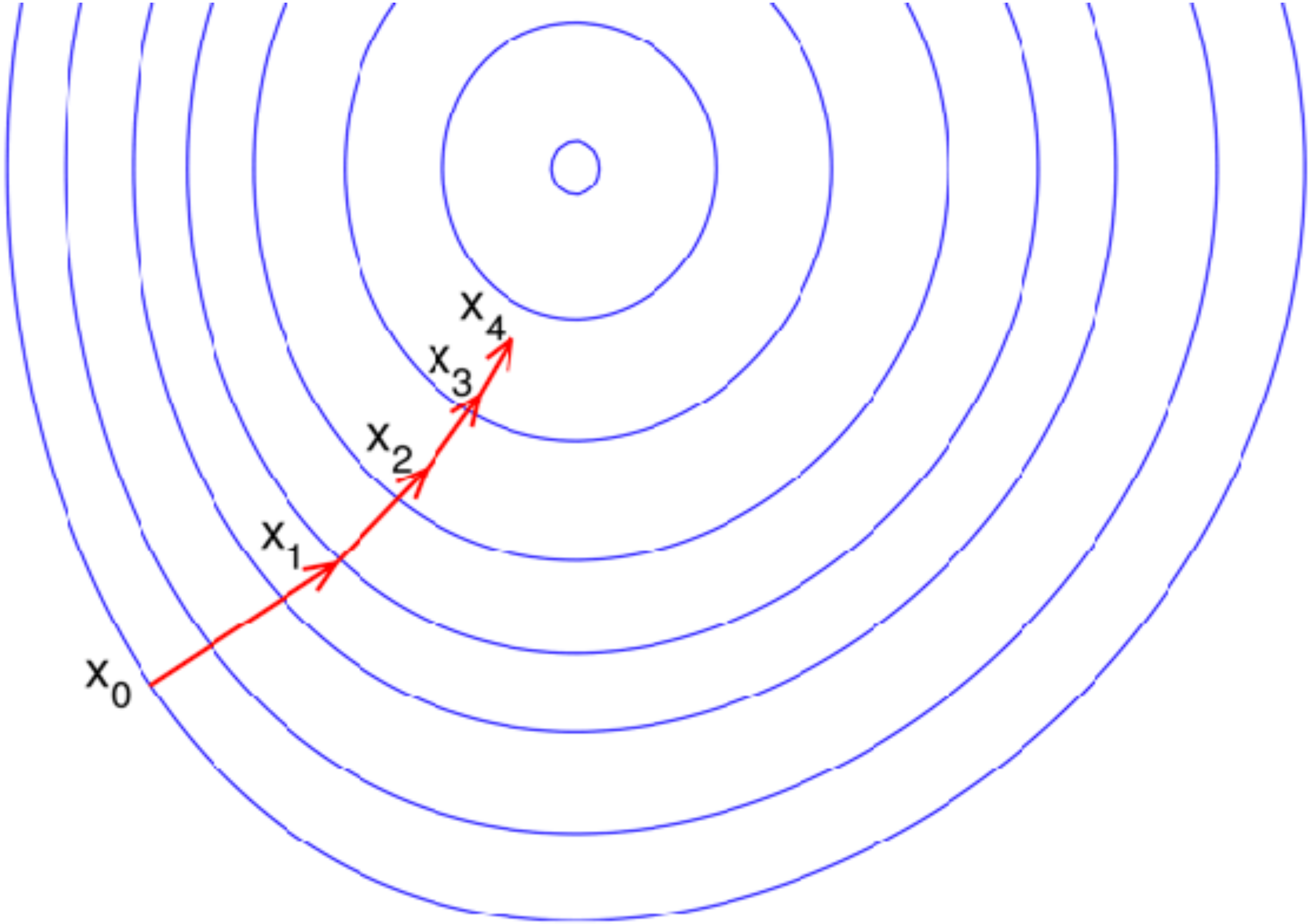
- Gradient Descent can also help us minimise error

- 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 optimised and no "next steps" perform better
    - The size of steps will shrink over time

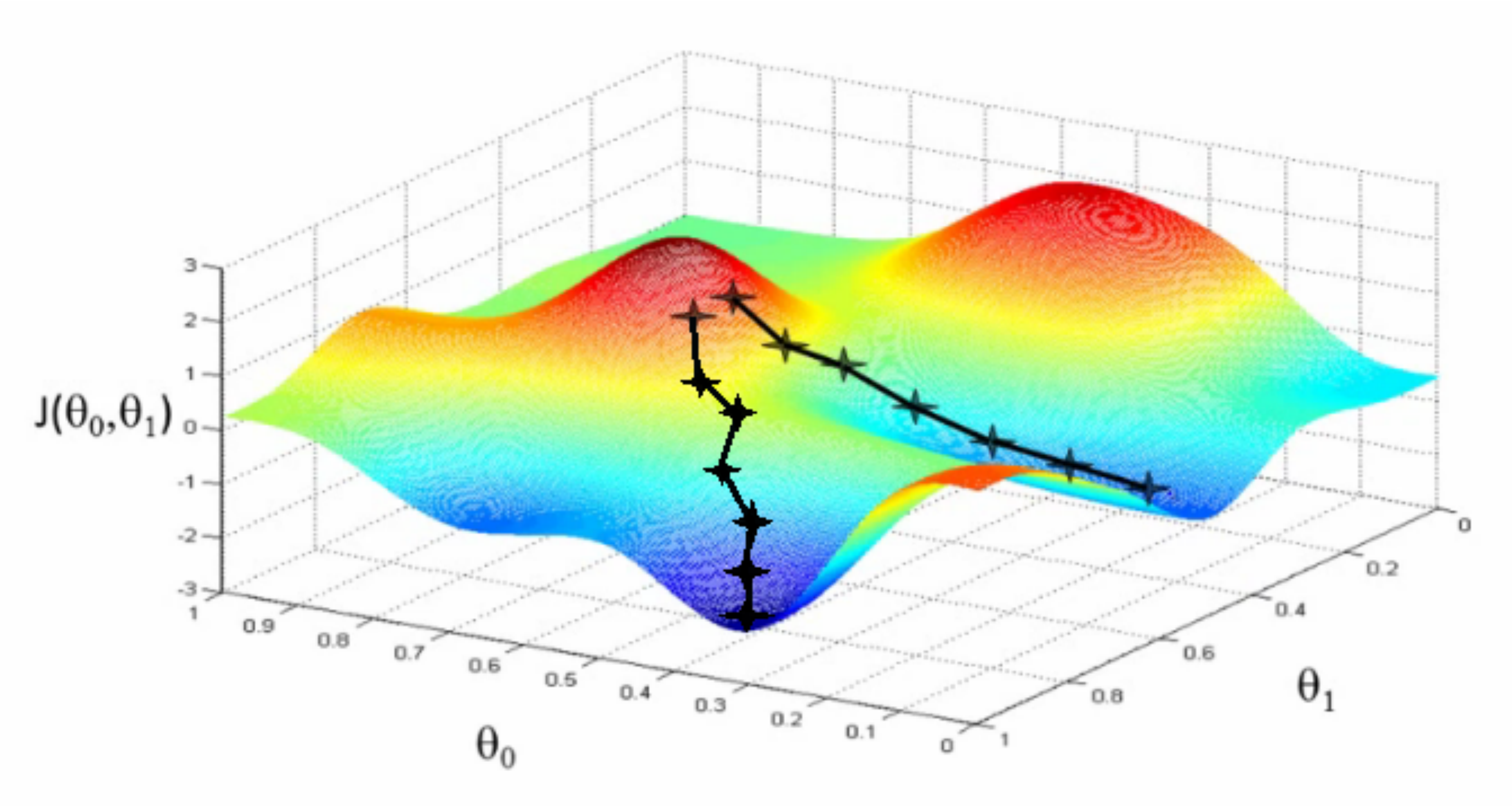
GRADIENT DESCENT



GRADIENT DESCENT



GRADIENT DESCENT



A CODE EXAMPLE OF GRADIENT DESCENT

```
num_to_approach, start, steps, optimised = 6.2, 0., [-1, 1], False
while not optimised:
    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
    if got_better:
      print("found better solution! using ", current_distance)
      a += 1
    else:
      optimised = True
      print(start, " is closest to ", num_to_approach)
```

A CODE EXAMPLE OF GRADIENT DESCENT

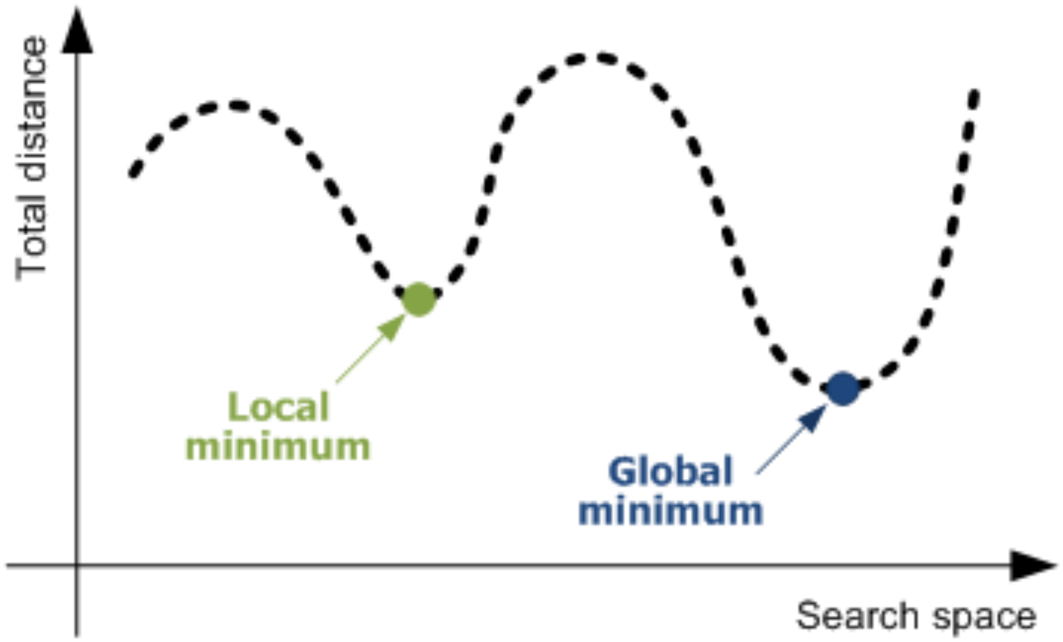
- What is the code doing?
- What could go wrong?

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 is not always true



DEMONSTRATION

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 normalised
- 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 optimisation 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

HANDS ON

ACTIVITY: HANDS ON

EXERCISE

DIRECTIONS (20 MINUTES)

There are tons of ways to approach a regression problem.

1. Implement the Gradient Descent approach to our bikeshare modelling problem
2. Show how Gradient Descent solves and optimises the solution
3. Demonstrate the grid_search module
4. Use a model you evaluated last class or the simpler one from today.
5. 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 regularisation. 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?

ACTIVITY: HANDS ON

EXERCISE

```
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 ESTIMATORS")
print(-gs.best_score_)
print(gs.best_estimator_)
print("ALL ESTIMATORS")
print(gs.grid_scores_)
```

TOPIC REVIEW

TOPIC REVIEW

- What is the (typical) range of R-Squared?
- What is the range of Mean Squared Error (MSE)?
- How would changing the scale or interpretation of y (your target variable) effect the Mean Squared Error?
- What is 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 minimising error?

BEFORE NEXT CLASS

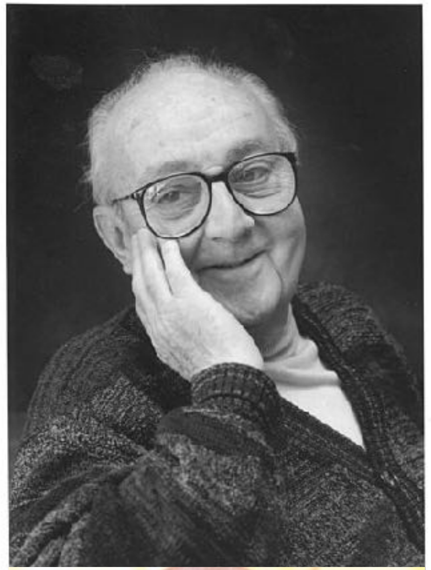
DUE DATE

- Project
  - Final Project, part 1

Q & A

CREDITS AND REFERENCES

EVALUATING MODEL FIT



- “[In Science] All models are wrong; some are useful!”
  - George Box, Wikipedia
- All models are wrong: limits of science
  - Martin Hilbert, University of California, Davis
    - youTube, video