SMU
SINGAPORE MANAGEMENT UNIVERSITY

MITB ISSS610
Applied Machine Learning

Instructors:

Dr. Steven HOI

Dr. DAI Bing Tian

School of Information Systems

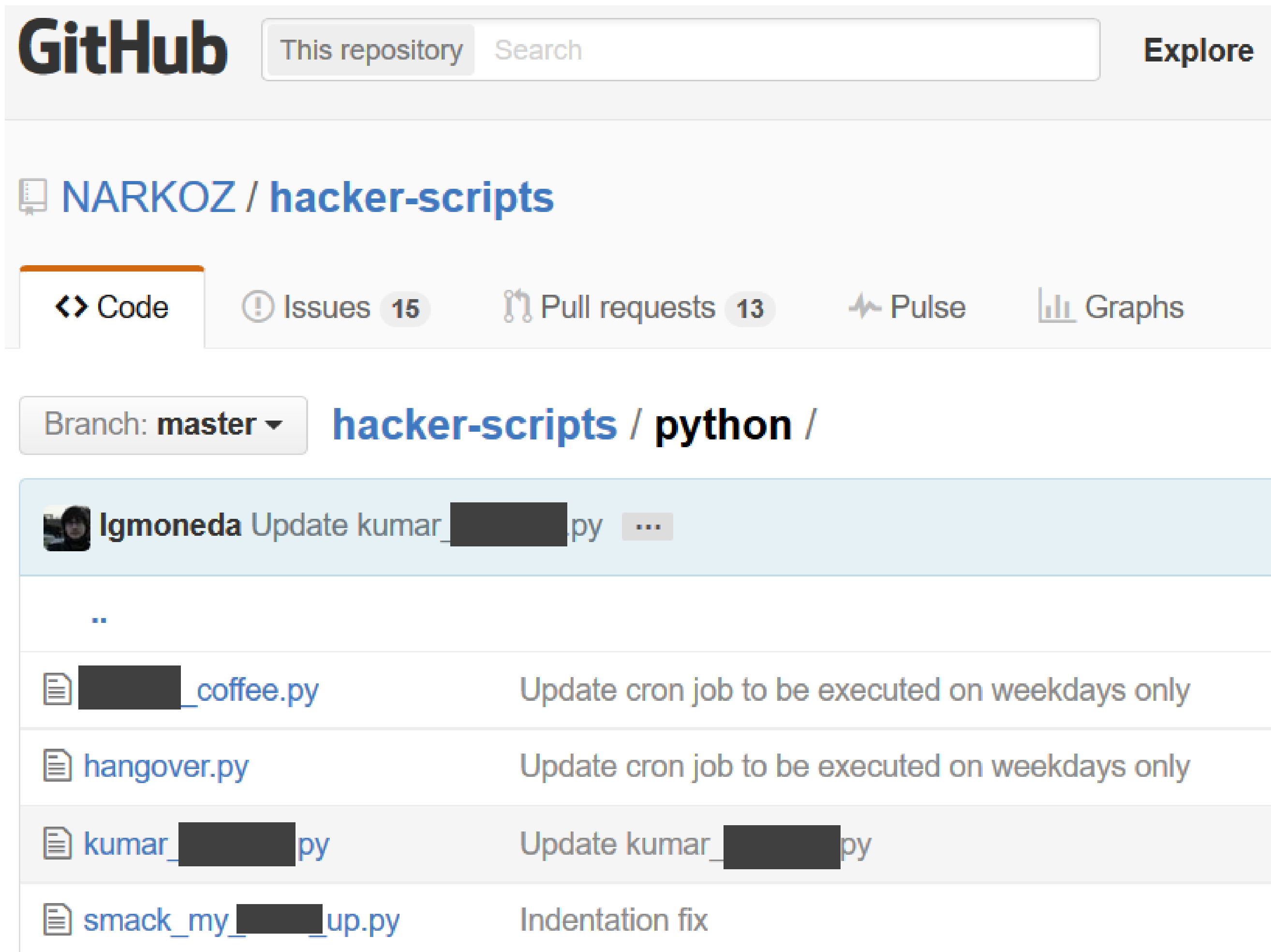
Singapore Management University

Outline

- Introduction to Python
- Linear Regression

Introduction to Python

What can Python do



Python Installation

- Install python
  - https://www.python.org/downloads/
- Python 2 or Python 3
  - Python 2.x is legacy, Python 3.x is the present and future of the language
  - Many libraries are still in Python 2
  - Major syntax differences:
    - print as a function
    - integer division
    - unicode string

Import Python Package

- import numpy

\>>> numpy.log(10)

- import numpy as np

\>>> np.log(10)

- "from x import y" v.s. "import x.y"

```
>>> import os.path
>>> path
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'path' is not defined
>>> os.path
<module 'posixpath' from '/usr/lib64/python3.4/posixpath.py'>
```

```
>>> from os import path
>>> path
<module 'posixpath' from '/usr/lib64/python3.4/posixpath.py'>
```

Print Function

- Print variables
  - Python2: print a
  - Python3: print(a)
- Print formatted strings
  - format function
  - % function:%s, %d, %f

```
>>> a=[1,2,3,4]
>>> b='1234'
>>> print(a)
[1, 2, 3, 4]
>>> print(b)
1234
>>> print(a, b)
[1, 2, 3, 4] 1234
>>> print('a=', a, 'b=', b)
a= [1, 2, 3, 4] b= 1234
```

```
>>> print('a={}'.format(a))
a=[1, 2, 3, 4]
>>> print('a={0} and b={1}'.format(a, b))
a=[1, 2, 3, 4] and b=1234
```

```
>>> c=9
>>> d=1.414214
>>> print('%s %d %f' % (b, c, d))
1234 9 1.414214
>>> print('%s %02d %.4f' % (b, c, d))
1234 09 1.4142
>>> print('%s %-2d %.8f' % (b, c, d))
1234 9  1.41421400
```

Arithmetic Operations

- Division and integer division
  - Python 2
  - Python 3

```
>>> 355 / 113
3.1415929203539825
>>> 355 // 113
3
```

```
>>> 355 / 113
3
>>> float(355) / 113
3.1415929203539825
>>> 355 / float(113)
3.1415929203539825
```

- Mathematical functions
  - import math

```
>>> pow(43, 17)
5874403106360420018879553643
>>> exp(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'exp' is not defined
>>> import math
>>> math.e
2.718281828459045
>>> math.exp(709.782)
1.7964120280206387e+308
>>> from math import exp
>>> exp(709.782)
1.7964120280206387e+308
```

String Operations

- find(substring, begin, end)
- Substrings
  - [bgn:end], [bgn:], [:end]
  - bgn/end can be negative
- split(delimiter)
  - Default: white spaces
  - split(‘,’)

```
>>> str = 'this course is easy.'
>>> str.find('s')
3
>>> str.find('s', 5)
9
>>> str.find('s', 18)
-1
>>> str[5:11]
'course'
>>> str[15:]
'easy.'
>>> str[-5:]
'easy.'
>>> str[:-5]
'this course is '
>>> str.split()
['this', 'course', 'is', 'easy.']
```

Python Containers

- Tuples: (1, 2, 3, 4) -- usually for fixed size
- Lists:
  [1, 2, 3, 4]

```
>>> a=[3,4,1,2]
>>> sorted(a)
[1, 2, 3, 4]
>>> sorted(a, reverse = True)
[4, 3, 2, 1]
```

- Dictionaries:
  {1:2, 3:4}
- Sort

```
>>> from operator import itemgetter
>>> b={3:7, 2:14, 1:1, 4:9}
>>> sorted(b.items(), key = itemgetter(1))
[(1, 1), (3, 7), (4, 9), (2, 14)]
>>> sorted(b.items(), key = itemgetter(1), reverse = True)
[(2, 14), (4, 9), (3, 7), (1, 1)]
```

Loops

- range
  - range(10), range(2, 10), range(2, 10, 2)
- for loops
  - for var in range/tuple/list/dictionary:
- while loops
  - while condition:
- Early termination
  - simulates do-while loop

For Loops on Dictionaries

- For loops on dictionaries
  - On keys? Or on values?
- For loops on dictionary values
- For loops on dictionary items
  - Python 3: dictionary.items()
  - Python 2: dictionary.iteritems()

Functions

- def function_name (parameter1, parameter2 = its_default_value, ...):
  four spaces function body
  four spaces return [something]
- Q1: is there pass-by-reference?
- main function:
  def main():
  four spaces function body
  if __name__ == "__main__":
  four spaces main()
- Q2: Why “main” function?

Modules

- modules: a collection of Python functions and statements
- import my_module
- from my_module import my_function
- from my_module import *

Classes

- Python classes

```
>>> class complex:
...     def __init__(self, real, imaginary):
...         self.r = real
...         self.i = imaginary
...
>>> x=complex(1.789, -2.345)
>>> x
<__main__.complex object at 0x7fa6048f76d8>
>>> x.r, x.i
(1.789, -2.345)
```

- Class Inheritance
  - class DerivedClassName(BaseClassName):
- Private variables
  - There is no “private” variable by definition
  - Convention: name begin with a single _
  - Name mangling

Write your first Python Program

- print ‘hello world’
- print (‘hello world’)

C++ "Hello World"

```
#include <iostream.h>
main()
{
cout << "Hello World! ";
}
return 0
```

Java "Hello World"

```
class HelloWorldApp
  {
    public static void main(String[] args)
        {
         System.out.println("Hello World!");
         }
    }
```

Python

```
print "Hello world"
```

from http://getadsensetip.blogspot.sg/2011/11/brace-yourselves-programming-jokes-are.html

Python Classes

```
class triangle:
    def __init__ (self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
    def perimeter (self):
        return self.a + self.b + self.c
    def area (self):
        return math.sqrt((self.a + self.b + self.c) * (- self.a + self.b + self.
c) * (self.a - self.b + self.c) * (self.a + self.b - self.c)) / 4

t = triangle (3, 4, 5)
print(t.perimeter(), t.area())
```

Python Classes

```
class parallelogram:
    def __init__ (self, a, b):
        self.a = a
        self.b = b
    def perimeter (self):
        return 2 * (self.a + self.b)
    def area (self):
        return 'the area of a parallelogram is not definite.'

p = parallelogram (3, 4)
print (p.perimeter(), p.area())
```

Python Classes

```
class rectangle (parallelogram):
    def __init__ (self, a, b):
        self.a = a
        self.b = b
    def area (self):
        return self.a * self.b

r = rectangle (3, 4)
print(r.perimeter(), r.area())
```

- What if the base class – parallelogram – is not specified?

Python Classes

```
class square (rectangle, parallelogram):
    def __init__ (self, a):
        self.a = a
        self.b = a

s = square (4)
print(s.perimeter(), s.area())
```

- What if the second base class – parallelogram – is removed?
- What if the order of the two base classes is changed?

Fibonacci

```python
def fib1 (n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib1 (n - 2) + fib1 (n - 1)

result = []
for i in range(10):
    result.append (fib1 (i))
print(result)
```

- Try fib1(30), fib1(40), fib1(50)

Fibonacci

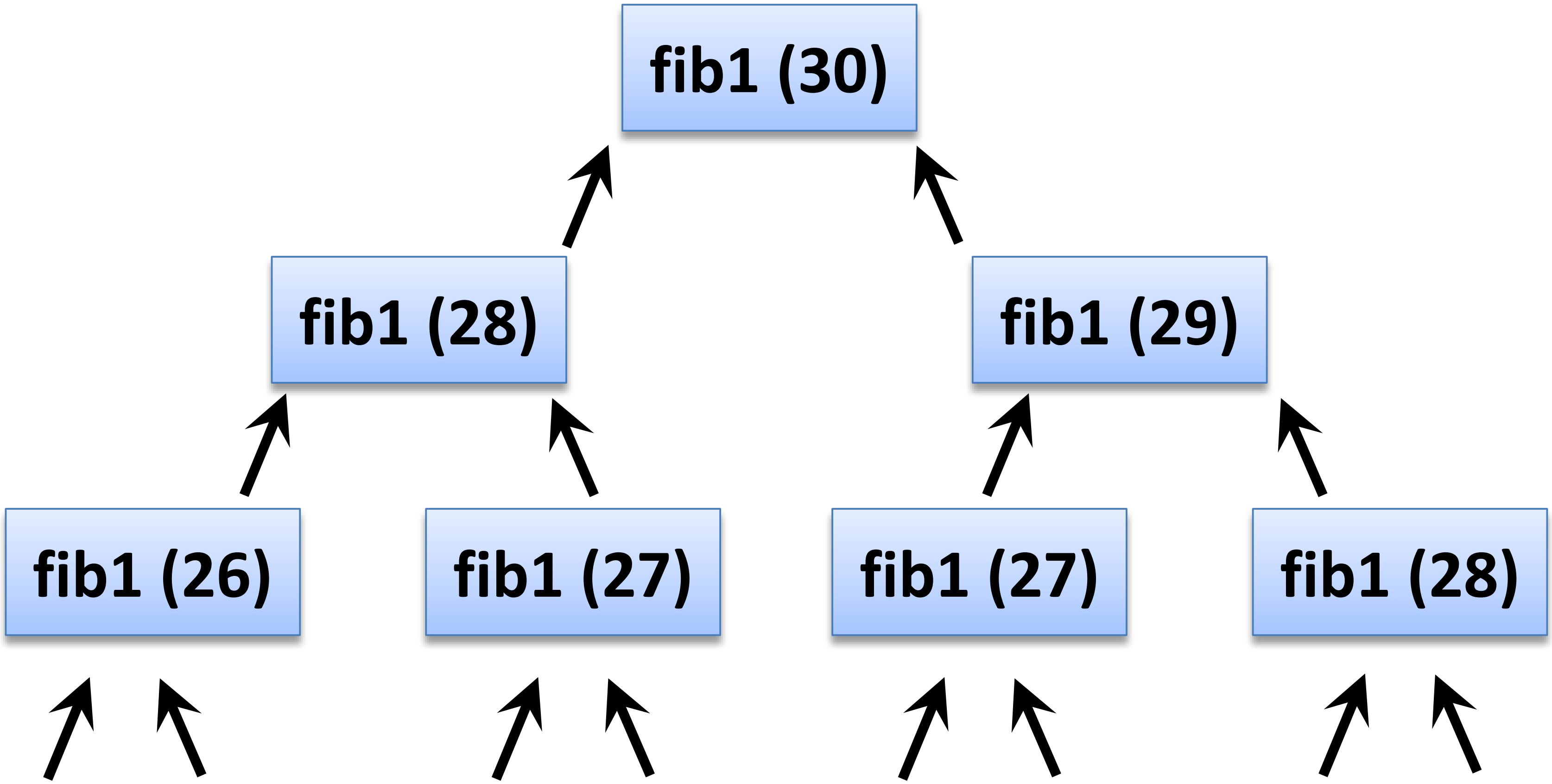
```python
def fib2 (n, a, b):
    if n == 0:
        return b
    else:
        return fib2 (n - 1, b, a + b)

result = []
for i in range(10):
    result.append (fib2 (i, 0, 1))
print(result)
```

- Try fib2(30), fib2(40), fib2(50)

Why faster?

- Recursive and Iterative
- Why is it so?

Recursive and Iterative



Recursive and Iterative

fib2 (30, 0, 1)

fib2 (29, 1, 1)

fib2 (28, 1, 2)

fib2 (27, 2, 3)

fib2 (26, 3, 5)

Install scikit-learn

- scikit-learn requires:
  - Python (>= 2.6 or >= 3.3),
  - NumPy (>= 1.6.1),
  - SciPy (>= 0.9).
- If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using pip
- pip – python package management system
  - https://pip.pypa.io/en/stable/installing/
  - python get-pip.py
- pip install -U scikit-learn

Linear Regression

Wedding AngPow Rate

Hotel	Lunch or Dinner	W/day or W/end	Rate
3-star	Dinner	Weekday	40
3-star	Lunch	Weekend	40
4-star	Lunch	Weekend	50
4-star	Dinner	Weekend	60
5-star	Lunch	Weekend	60
5-star	Dinner	Weekend	70
5-star	Dinner	Weekday	60

Outline

- Linear Regression: Analytical Solutions
- Polynomial Models
- Gradient Descent
- Regression Visualization

Supervised Learning

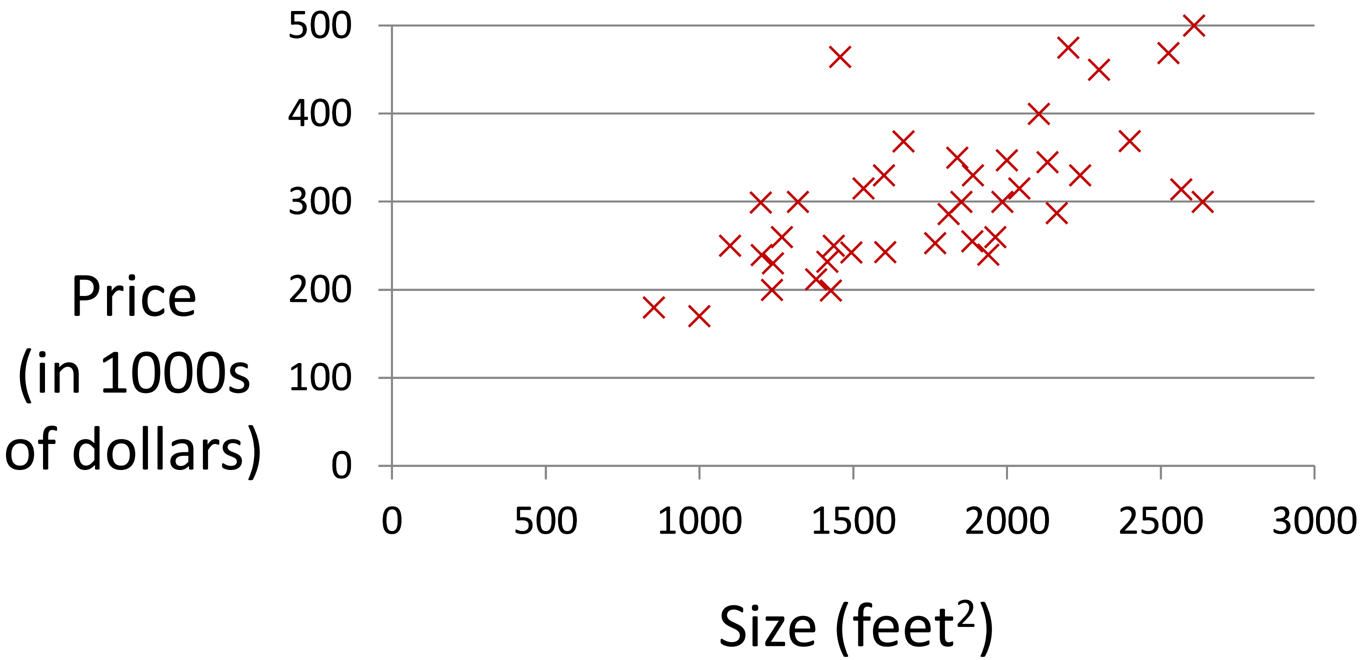
- Formalization
  - Input: $\mathbf{x} \quad \in \mathcal{X} \quad \mathbb{R}^n$
  - Output: $y \quad \in \mathcal{Y} \begin{cases} \mathbb{R} & \text{regression} \\ \{+1, -1\} & \text{binary classification} \\ \{1, 2, \ldots, K\} & \text{multi-class classification} \end{cases}$
  - Target function: $f : \mathcal{X} \rightarrow \mathcal{Y}$ (unknown)
  - Training Data: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$
  - Hypothesis: $h : \mathcal{X} \rightarrow \mathcal{Y} \qquad h \approx f$
  - Hypothesis space: $h \in \mathcal{H}$

Explanatory and Target Variables

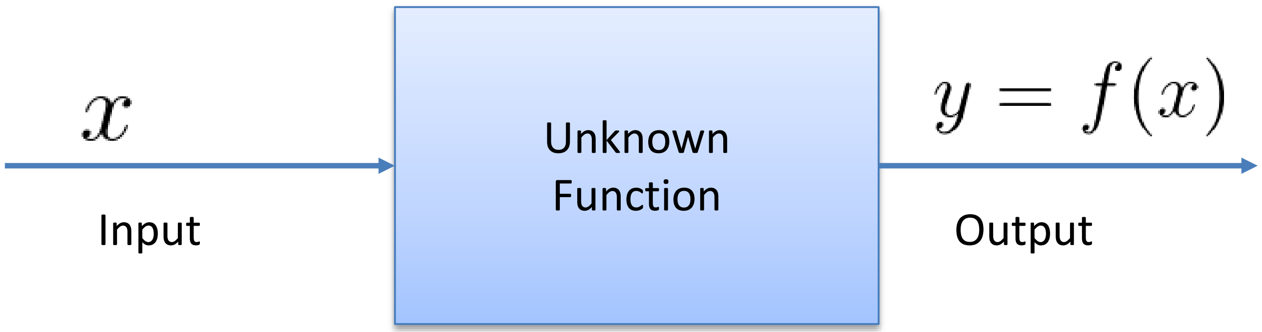
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

- **x** = input variable / explanatory variable
- **y** = output variable / target variable

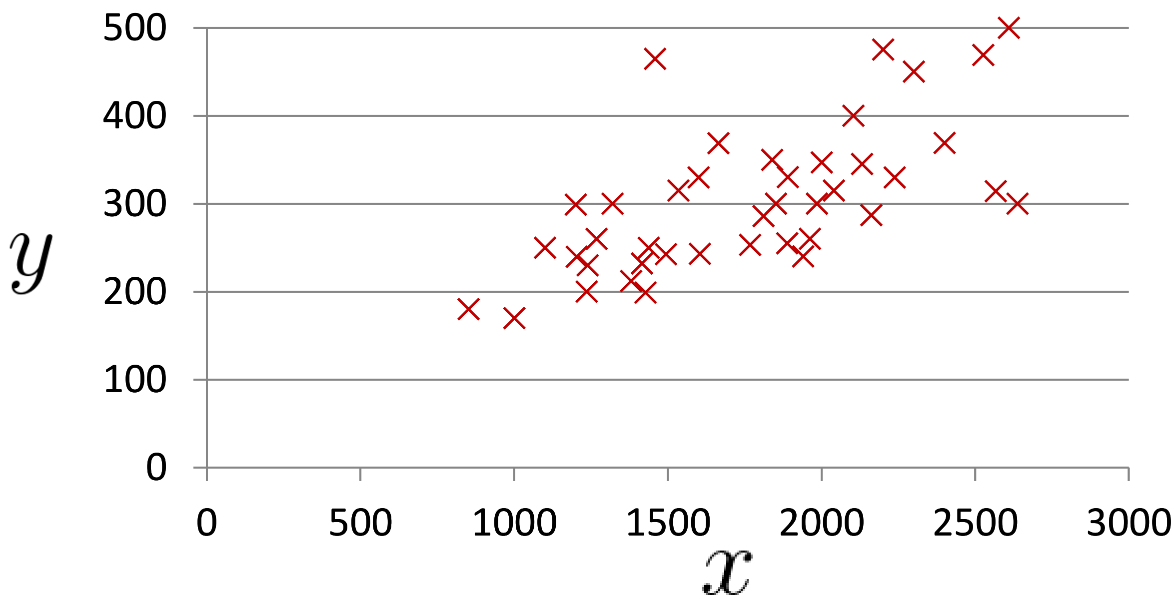
Target Function



A Learning Problem



$$h(x) \approx f(x)$$



Hypothesis Spaces

- Linear models

$$h(x) = ax + b \approx f(x)$$

  - **Infinite** possible hypotheses!
  - Any choices of coefficient a and b will result in a possible hypothesis

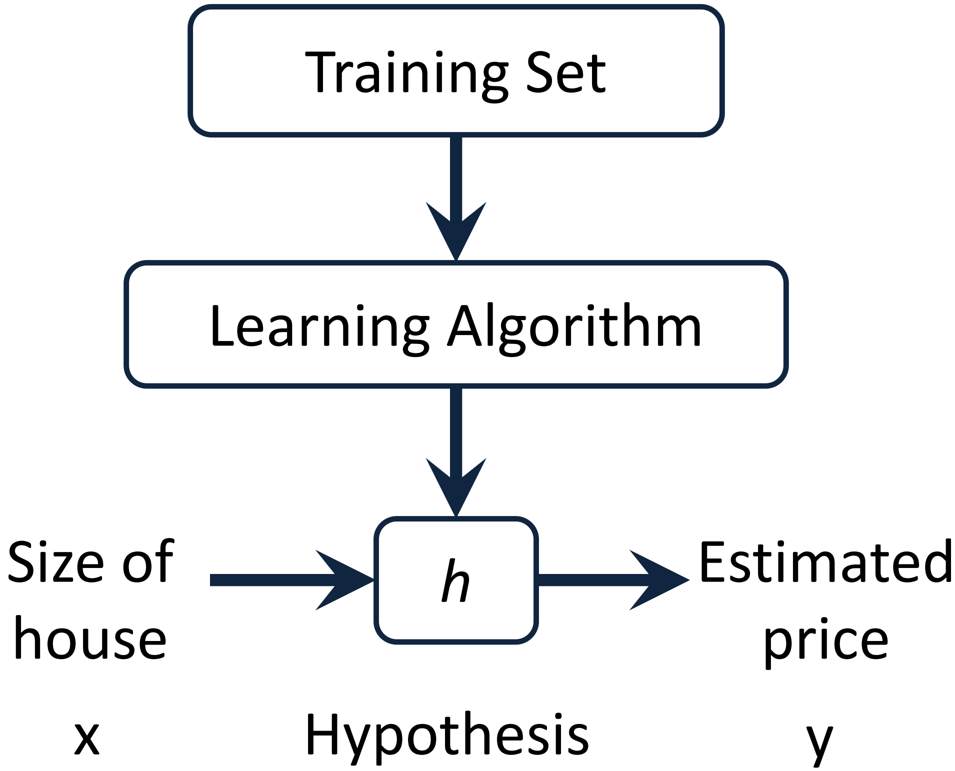
- Polynomial models

$$h(x) = ax^2 + bx + c \approx f(x)$$

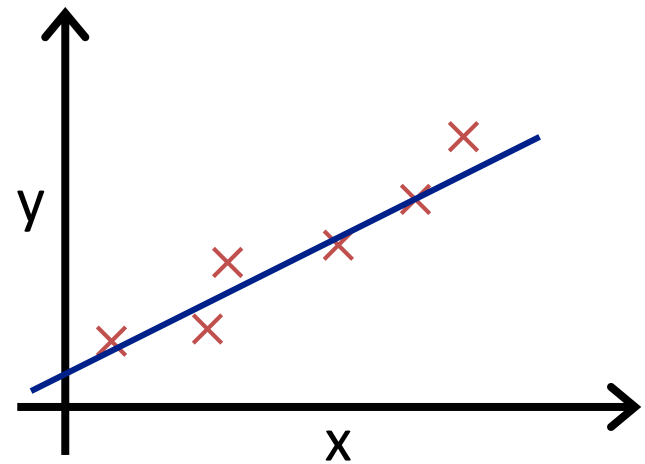
- Any nonlinear models

$$h(x) = g(x) \approx f(x)$$

Model Representation



How do we represent *h* ?



$$h(x) = w_0 + w_1 x$$

Linear regression with one variable.
"**Univariate Linear Regression**"

How to choose parameters w_0, w_1 ?

Formulation: Cost Function

Hypothesis:

$$h(x) = w_0 + w_1 x$$

Parameters:

$$w_0, w_1$$

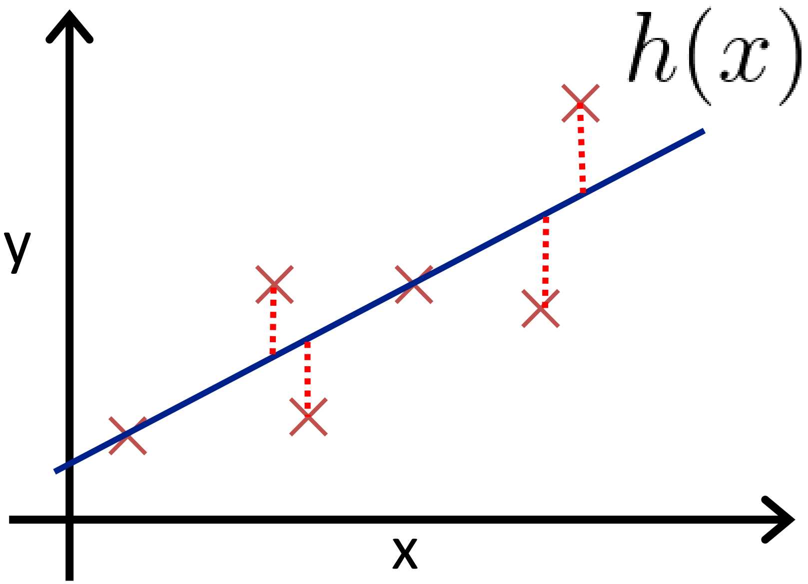
Cost Function:

mean squared error (MSE)

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

Goal:

$$\min_{w_0, w_1} J(w_0, w_1)$$



Normal Equation

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{1}{m} \sum_{i=1}^{m} (w_0 + w_1 x_i - y_i) = 0$$

$$\frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} x_i (w_0 + w_1 x_i - y_i) = 0$$

$$w_0 = \frac{\sum_{i=1}^{m} x_i^2 \sum_{i=1}^{m} y_i - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} x_i y_i)}{m \sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2}$$

$$w_1 = \frac{m \sum_{i=1}^{m} x_i y_i - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} y_i)}{m \sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2}$$

Linear Regression Example

- Identify and load explanatory variable as X
- Identify and load target variable as y

```
from sklearn import datasets, linear_model

boston = datasets.load_boston()
x = boston.data[:,12].reshape((506,1))
y = boston.target

regr = linear_model.LinearRegression()
regr.fit(x, y)
```

Multivariate Linear Regression

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$\mathbf{x}_i$ = input (features) of i^{th} training example.

x_{ij} = value of feature j in i^{th} training example.

Multivariate Linear Regression

Hypothesis:

Previously: $h(x) = w_0 + w_1 x$

$\mathbf{x} \in \mathbb{R}^n$ $\quad h(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

For convenience of notation, define $x_0 = 1$.

$$h(\mathbf{x}) = \sum_{j=0}^{n} w_j x_j = \mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$$

$\mathbf{x} \in \mathbb{R}^{n+1}$ $\quad \mathbf{w} \in \mathbb{R}^{n+1}$

Normal Equation

$$J(\mathbf{w}) = \frac{1}{2m}\sum_{i=1}^{m}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 = \frac{1}{2m}(X\mathbf{w}-\mathbf{y})^T(X\mathbf{w}-\mathbf{y})$$

$$(\mathbf{w}^T\mathbf{x}_1 - y_1)$$

$$X\mathbf{w}-\mathbf{y} = \begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1n} \\ x_{20} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m0} & x_{n1} & \cdots & x_{mn} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

$$m \times (n+1) \qquad (n+1)\times 1 \qquad m \times 1$$

Normal Equation

- Matrix-vector formulation

$$J(\mathbf{w}) = \frac{1}{2m}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y})$$

$$\nabla J(\mathbf{w}) = \nabla_{\mathbf{w}}\left(\frac{1}{2m}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y})\right)$$

$$= X^T X\mathbf{w} - X^T\mathbf{y} = \mathbf{0}$$

$$X^T X\mathbf{w} = X^T\mathbf{y}$$

- Analytical solution

$$\mathbf{w} = \left((X^T X)^{-1} X^T\right)\mathbf{y} = X^\dagger \mathbf{y}$$

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

Boston Example

```
from sklearn import datasets, linear_model

boston = datasets.load_boston()
x = boston.data
y = boston.target

regr = linear_model.LinearRegression()
regr.fit(x, y)
```

Coefficient of Determination

- Calculate R^2 score

$$\forall i,\ \hat{y}_i = h(x_i),\ SS_{res} = \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

$$SS_{tot} = \sum_{i=1}^{m} (y_i - \bar{y})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Model Parameters

- Print model parameters: coefficients and intercept
- Calculate R^2 score

```
>>> print('Coefficients: \n', regr.coef_)
Coefficients:
 [ -1.07170557e-01   4.63952195e-02   2.08602395e-02   2.68856140e+00
  -1.77957587e+01   3.80475246e+00   7.51061703e-04  -1.47575880e+00
   3.05655038e-01  -1.23293463e-02  -9.53463555e-01   9.39251272e-03
  -5.25466633e-01]
>>> print('Intercept: \n', regr.intercept_)
Intercept:
 36.4911032804
>>> import numpy as np
>>> print('Residue sum of squares: %.2f' % np.mean((regr.predict(x) - y) ** 2))
Residue sum of squares: 21.90
>>> print('R^2 score: %.2f' % regr.score(x, y))
R^2 score: 0.74
```

Training Data and Test Data

- It is not recommended to train and test a model with the set of data
- Split data into training data and test data
- Train the model using training data
- Test the model with test data

```
>>> print(train[list(range(20))])
[False False  True  True  True False  True  True  True  True  True  True
 False  True  True  True  True  True  True  True]
```

```
>>> train = np.random.choice([True, False], len(x), replace=True, p=[0.9,0.1])
>>> x_train = x[train,:]
>>> y_train = y[train]
>>> x_test = x[~train,:]
>>> y_test = y[~train]
>>> regr.fit(x_train, y_train)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> print('R^2 score: %.2f' % regr.score(x_test, y_test))
R^2 score: 0.50
```

Multiple Target Example

```
>>> from sklearn import datasets, linear_model
>>> linnerud = datasets.load_linnerud()
>>> X = linnerud.data
>>> Y = linnerud.target
>>> X.shape
(20, 3)
>>> Y.shape
(20, 3)
>>> regr = linear_model.LinearRegression()
>>> regr.fit (X, Y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> regr.coef_
array([[-0.47502636, -0.21771647,  0.09308837],
       [-0.13687023, -0.04033662,  0.0279736 ],
       [ 0.00107079,  0.04202941, -0.02946117]])
>>> regr.intercept_
array([ 208.23351881,   40.59787542,   52.04362105])
>>> regr.score(X, Y)
0.25725245750743864
```

More Evaluation Metrics

- from sklearn import metrics
  - explained_variance_score
  - mean_absolute_error
  - …

$$1 - \frac{cov(y, \hat{y})}{var(y)}$$

$$\frac{1}{m} \sum_{i=1}^{m} |\hat{y}_i - y_i|$$

```
>>> from sklearn import metrics
>>> Y_pred = regr.predict(X)
>>> metrics.explained_variance_score(Y, Y_pred)
0.296877912088l459
>>> metrics.mean_absolute_error(Y, Y_pred)
7.456710474001059
>>> metrics.mean_squared_error(Y, Y_pred)
158.0244913155757
```

Hypothesis Spaces

- Linear models

$$h(x) = ax + b \approx f(x)$$

  - **Infinite** possible hypotheses!
  - Any choices of coefficient a and b will result in a possible hypothesis

- Polynomial models

$$h(x) = ax^2 + bx + c \approx f(x)$$

- Any nonlinear models

$$h(x) = g(x) \approx f(x)$$

Polynomial Models

$$h(x) = ax^2 + bx + c = (a, b)(x^2, x)^T + c$$

- Map each explanatory variable to a higher order space
- Fit a linear model in the higher order space

Polynomial Models

$$h(x) = ax_0^2 + bx_5^2 + cx_0x_5 + dx_0 + ex_5 + f$$

```
boston = datasets.load_boston()
x = boston.data
y = boston.target

mapped = []
for i in x:
    mapped.append([i[0]*i[0], i[5]*i[5], i[0]*i[5], i[0], i[5]])

mapped = np.asarray(mapped)
```

- x[0]: CRIM: per capita crime rate by town
- X[5]: RM: average number of rooms per dwelling

Polynomial Models

$$h(x) = ax_0^2 + bx_5^2 + cx_0x_5 + dx_0 + ex_5 + f$$

```
boston = datasets.load_boston()
x = boston.data
y = boston.target

terms = [{0:2}, {5:2}, {0:1, 5:1}, {0:1}, {5:1}]

x_mapped = map(x, terms)
```

- x[0]: CRIM: per capita crime rate by town
- X[5]: RM: average number of rooms per dwelling

Polynomial Models

```
def map(orig_data, terms):
    mapped = []
    for x in orig_data:
        xx = []
        for d in terms:
            v = 1.0
            for pos, exponent in d.items():
                v *= math.pow(x[pos], exponent)
            xx.append(v)
        mapped.append(xx)
    return np.asarray(mapped)
```

- Map each tuple x by terms into a higher dimensional space

Polynomial Models

```
>>> regr = linear_model.LinearRegression()
>>> regr.fit(x_mapped, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> regr.score(x_mapped, y)
0.6672742746453682
```

$$h(x) = ax^2 + bx + c = (a, b)(x^2, x)^T + c$$

$$h(x) = ax^2 + bx + c = (a, b, c)(x^2, x, x^0)^T$$

- No intercept regression can be achieved by polynomial models

Polynomial Models

```
v = 1.0
for pos, exponent in d.items():
    v *= math.pow(x[pos], exponent)
xx.append(v)
```

```
>>> regr = linear_model.LinearRegression()
>>> regr.fit(x_mapped, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> regr.coef_
array([  7.71277455e-03,   2.42884269e+00,  -2.39957735e-01,
         7.27673940e-01,  -2.24951593e+01])
>>> regr.intercept_
68.59670568444092
>>> regr = linear_model.LinearRegression(fit_intercept=False)
>>> regr.fit(x_mapped, y)
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
>>> regr.coef_
array([ 0.00778588,  0.89357905, -0.3598296 ,  1.47574173, -1.83852107])
>>> regr.intercept_
0.0
>>> terms = [{0:2}, {5:2}, {0:1, 5:1}, {0:1}, {5:1}, {}]
>>> x_mapped = map(x, terms)
>>> regr.fit(x_mapped, y)
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
>>> regr.coef_
array([  7.71277455e-03,   2.42884269e+00,  -2.39957735e-01,
         7.27673940e-01,  -2.24951593e+01,   6.85967057e+01])
```

Polynomial Models

$$X\mathbf{w} - \mathbf{y} = \begin{pmatrix} x_{10} & x_{11} & \dots & x_{1n} \\ x_{20} & x_{21} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m0} & x_{n1} & \dots & x_{mn} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

$m \times (n+1)$ $\quad (n+1) \times 1$ $\quad m \times 1$

$$\mathbf{w} = \left((X^T X)^{-1} X^T\right) \mathbf{y} = X^{\dagger}\mathbf{y}$$

- $x_{10}, x_{20}, \cdots, x_{m0}$ are 1, the intercept is represented by w_0

Outline

- Linear Regression: Analytical Solutions
- Polynomial Models
- Gradient Descent
- Regression Visualization

Hypothesis Spaces

- Linear models

$$h(x) = ax + b \approx f(x)$$

  - **Infinite** possible hypotheses!
  - Any choices of coefficient a and b will result in a possible hypothesis

- Polynomial models

$$h(x) = ax^2 + bx + c \approx f(x)$$

- Any nonlinear models

$$h(x) = g(x) \approx f(x)$$

Gradient Descent

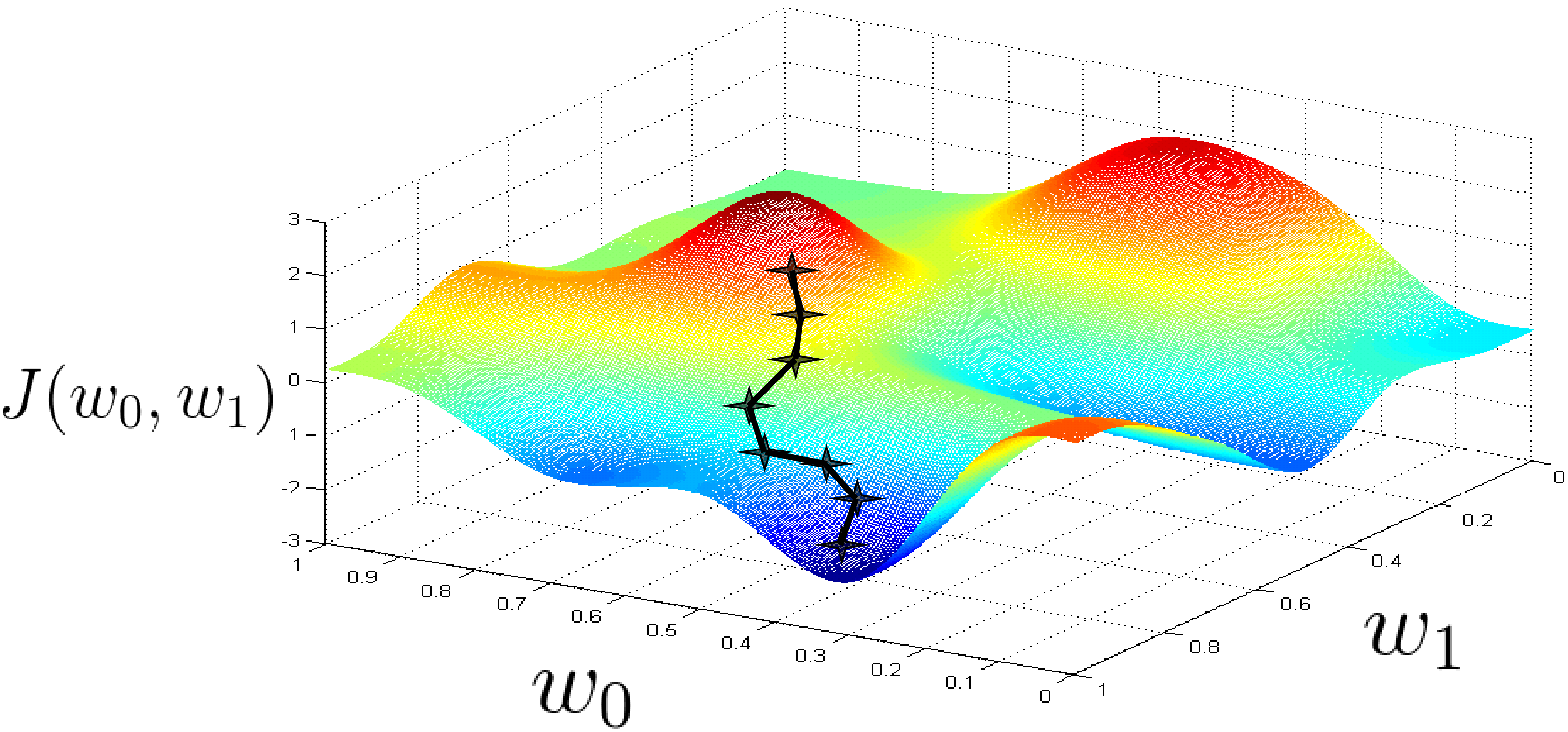
Given some objective function $J(w_0, w_1)$

Want to optimize $\min_{w_0, w_1} J(w_0, w_1)$

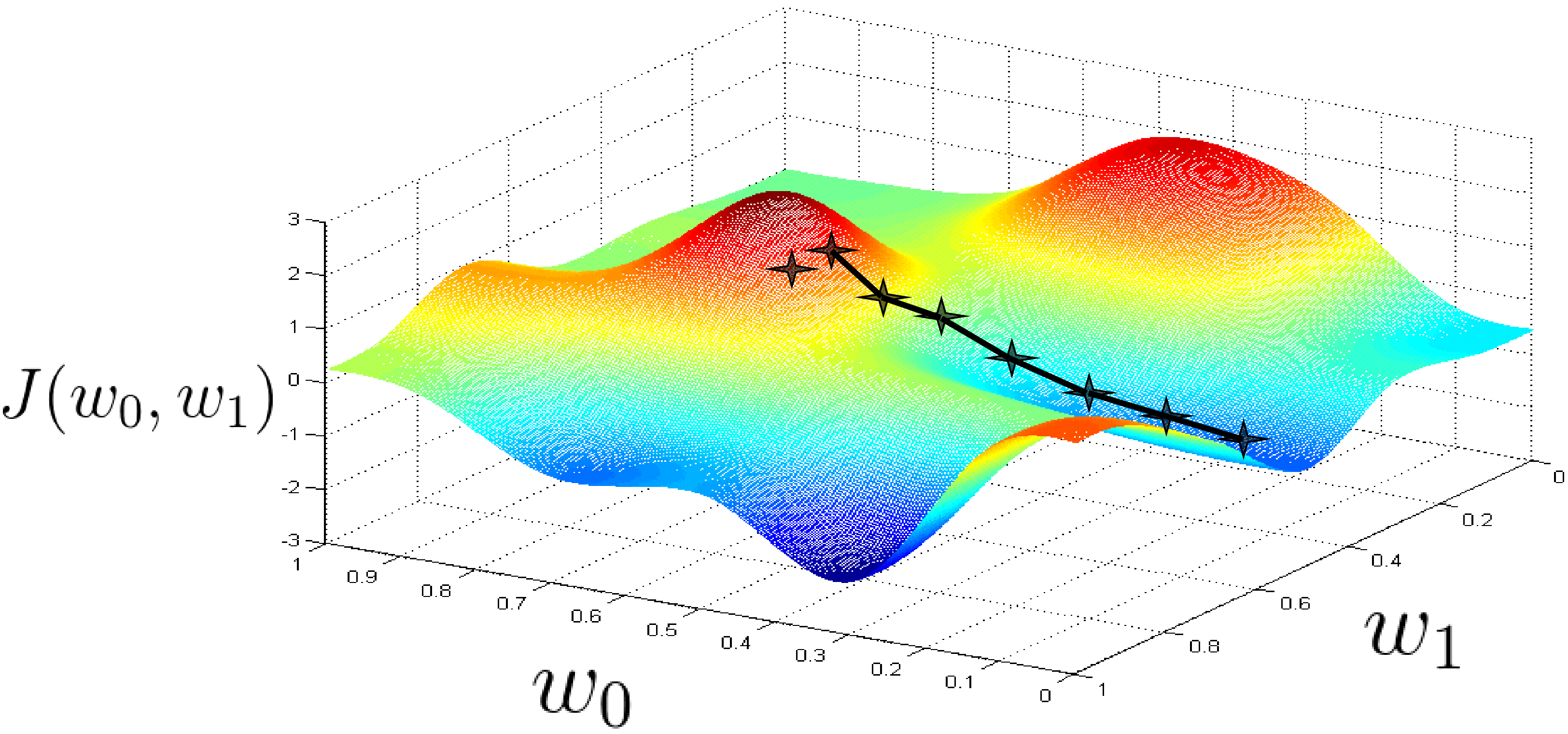
Outline:

- Start with some w_0, w_1
- Keep changing w_0, w_1 to reduce $J(w_0, w_1)$ until we hopefully end up at a minimum

Gradient Descent



Gradient Descent



Gradient Descent Algorithm

Gradient descent algorithm

$$\text{initialize} \quad w_j \quad j = 0, 1$$

$$\text{repeat until convergence } \{$$

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1) \quad \begin{array}{l}(\text{simultaneously update} \\ \quad j = 0 \text{ and } j = 1)\end{array}$$

$$\}$$

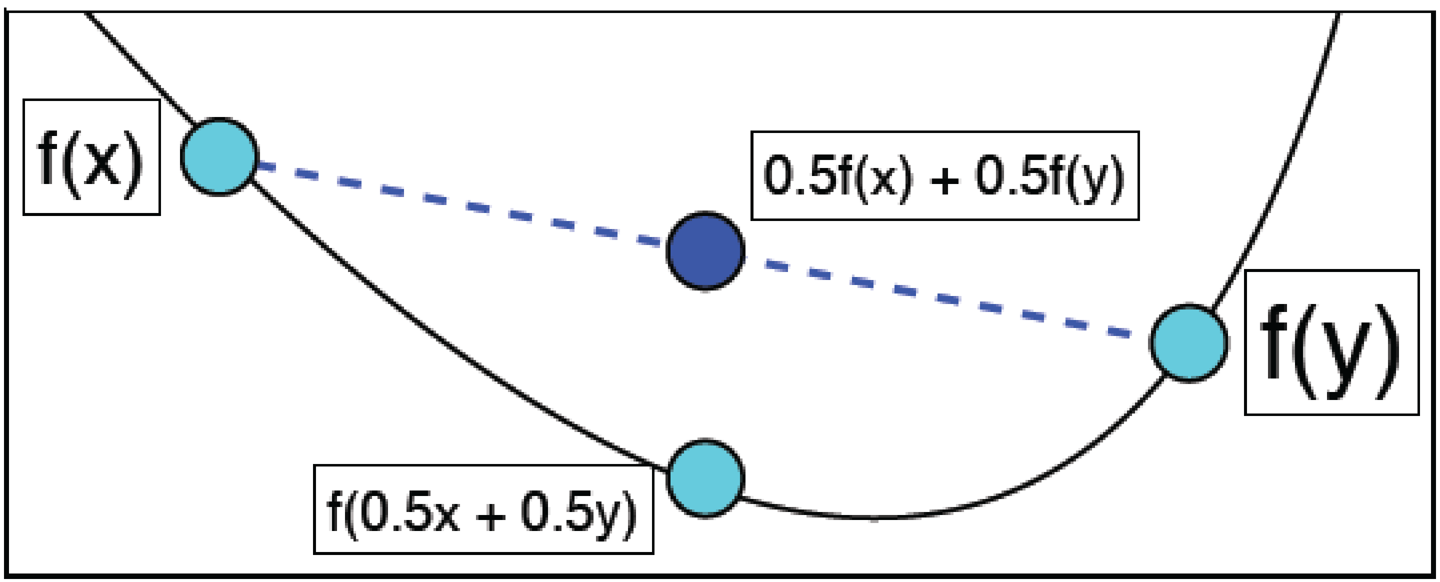
learning rate parameter
(rule of thumb: 0.1)

Convex Function

- A real-valued function f is **convex** if

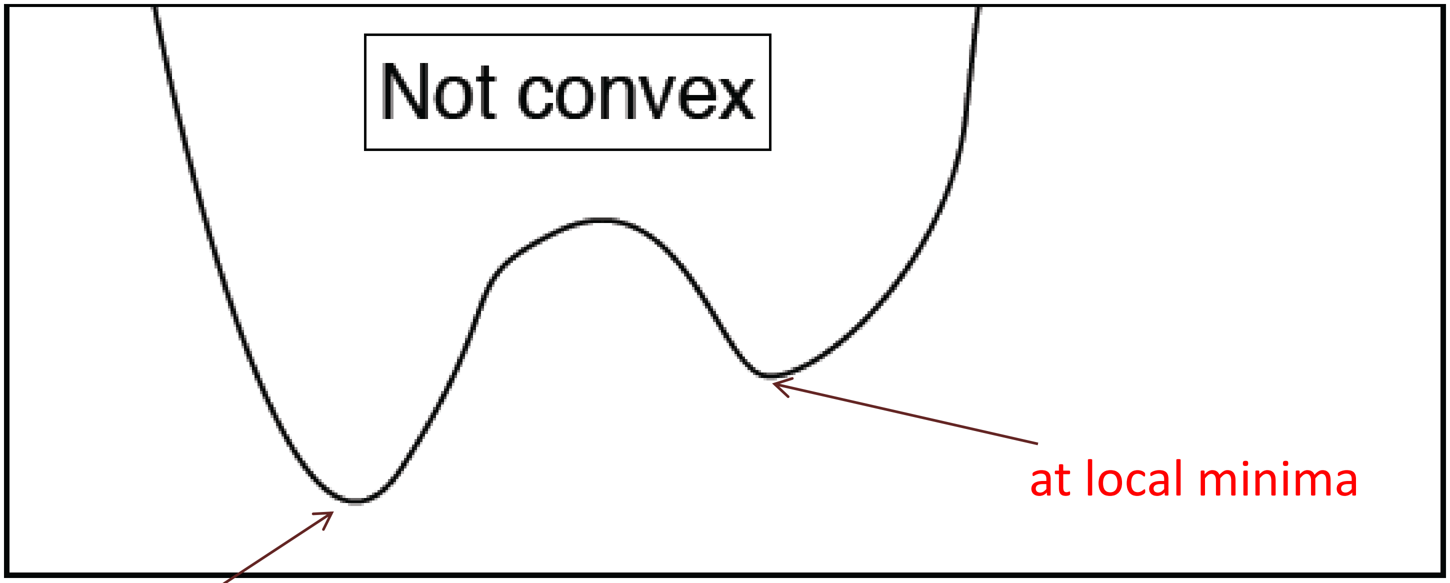
$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y) \quad \forall 0 \leq \theta \leq 1$$

- Function is **below** a linear interpolation from x to y.
- The negative of a convex function is a **concave** function
- **Convex**: Implies that all local minima are global minima.



Non-Convex Function

$$w_1 := w_1 - \alpha \frac{\partial}{\partial w_1} J(w_1)$$



Global minima

The final solution is sensitive to initialization

Gradient Descent for Linear Models

Gradient descent algorithm

repeat until convergence {

$$w_0 := w_0 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h(x_i) - y_i\right)} \rightarrow \frac{\partial}{\partial w_0} J(w_0, w_1)$$

$$w_1 := w_1 - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h(x_i) - y_i\right) \cdot x_i} \rightarrow \frac{\partial}{\partial w_1} J(w_0, w_1)$$

}

update w_0 and w_1 simultaneously

Stochastic Gradient Descent

- Evaluating the sum of gradient may be expensive
- To save the cost at each iteration, stochastic gradient descent samples a subset of summand gradient at each iteration

$$
\begin{aligned}
\mathbf{w} &:= \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) \\
&:= \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} \sum_{i=1}^{m} g(x_i, y_i, \mathbf{w}) \\
&:= \mathbf{w} - \alpha \sum_{i=1}^{m} \frac{\partial g(x_i, y_i, \mathbf{w})}{\partial \mathbf{w}}
\end{aligned}
$$

Gradient Descent for Linear Regression

Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

Linear Regression Model

$$h(x) = w_0 + w_1 x$$

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

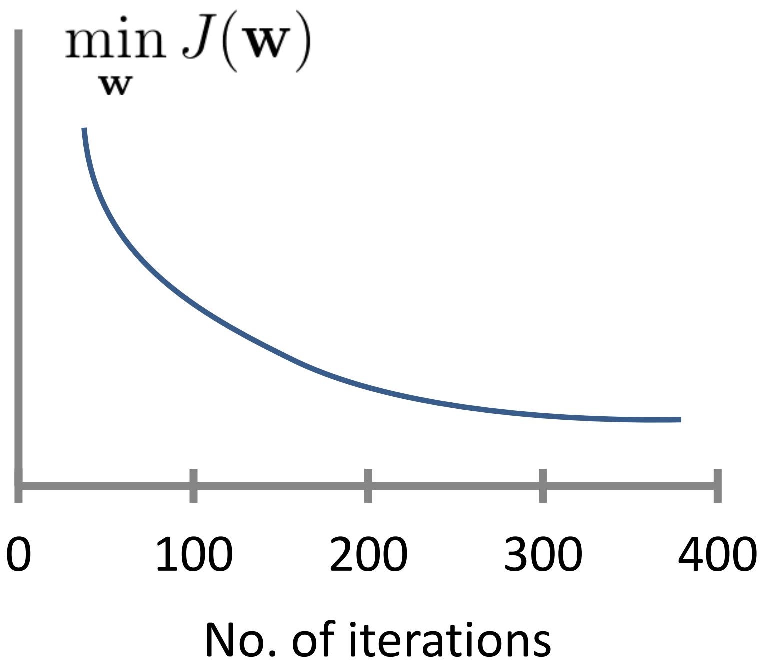
$$\frac{\partial}{\partial w_j} J(w_0, w_1) = \frac{\partial}{\partial w_j} \left(\frac{1}{2m} \sum_{i=1}^{m} ((w_0 + w_1 x_i) - y_i)^2 \right)$$

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)$$

$$\frac{\partial}{\partial w_1} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i) \cdot x_i$$

Convergence and Learning Rate

$$\mathbf{w} := \mathbf{w} - \alpha \nabla J(\mathbf{w})$$

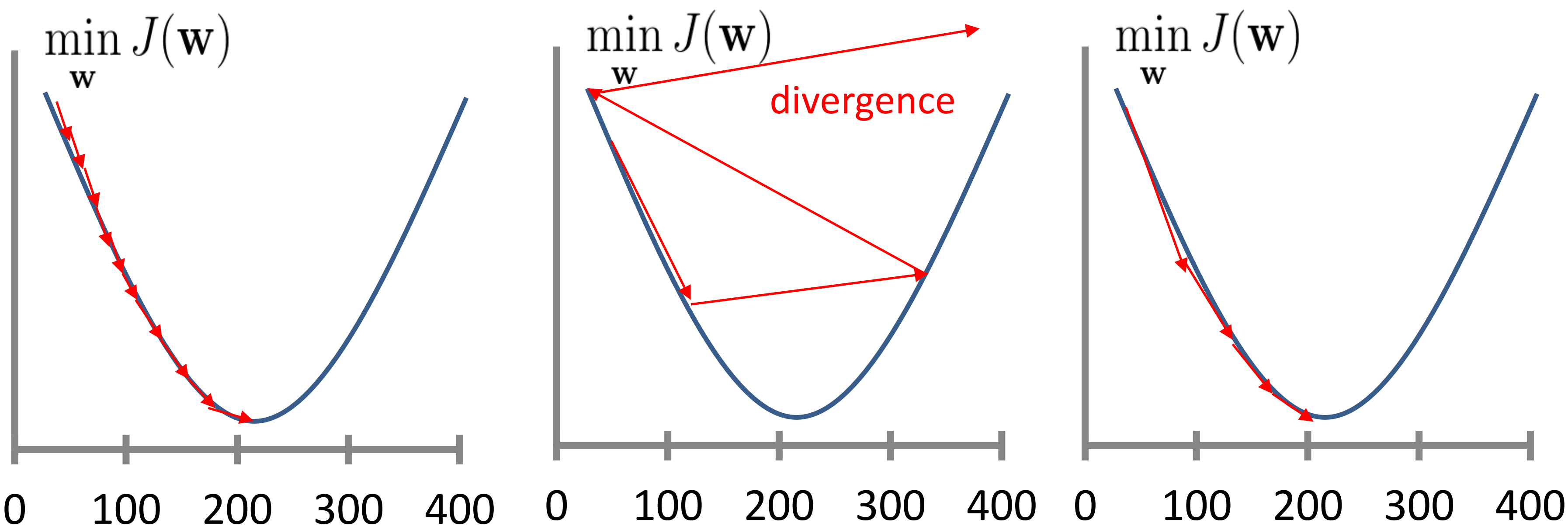


Example automatic convergence test:

Declare convergence if $J(\mathbf{w})$ decreases by less than 10^{-3} in one iteration.

For sufficiently small α , $J(\mathbf{w})$ should decrease on every iteration.
But if α is too small, gradient descent can be slow to converge.
If α is too large: $J(\mathbf{w})$ may not decrease on every iteration; may not converge.

Learning Rate



too small constant

too large

gradually decreased

$$\alpha_t = \frac{\alpha}{t}$$

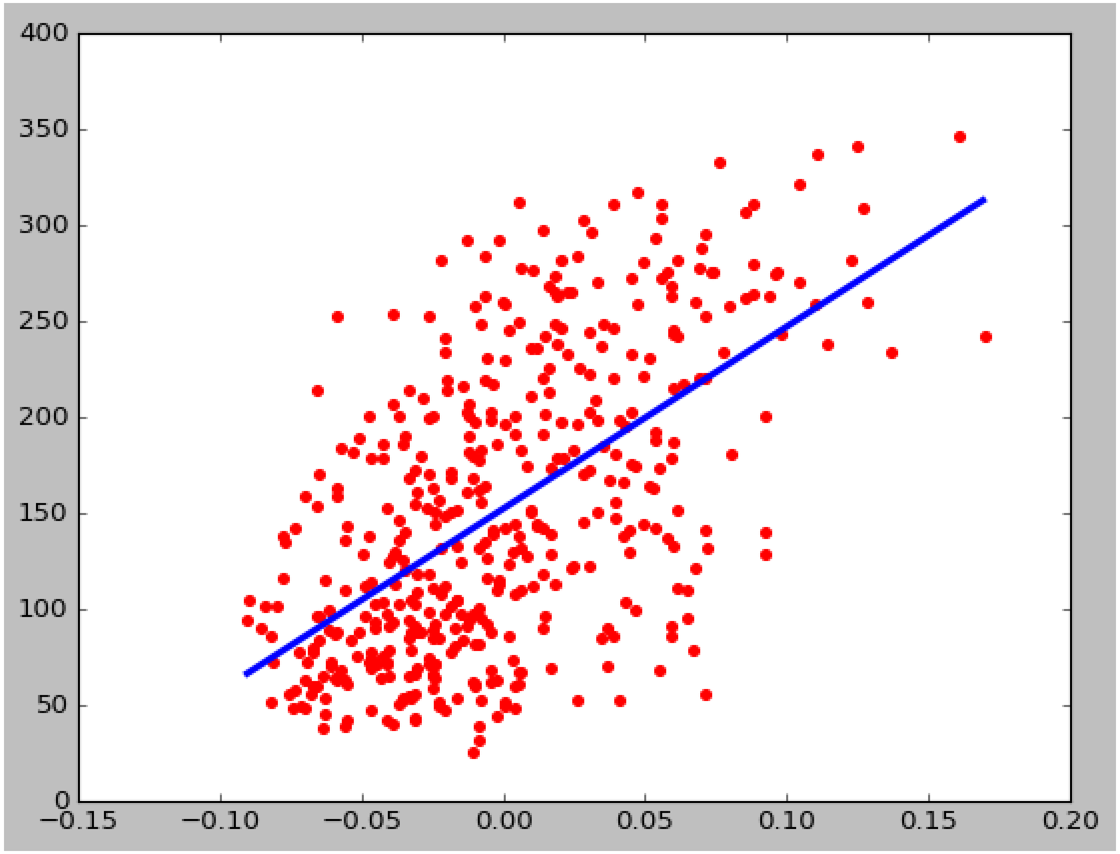
Gradient Descent Regression

```
>>> from sklearn import datasets, linear_model
>>> diabetes = datasets.load_diabetes()
>>> x = diabetes.data[:,list(range(4))]
>>> y = diabetes.target
>>> regr = linear_model.LinearRegression()
>>> regr.fit(x, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
om_state=2016)ar_model.SGDRegressor(eta0=0.1, n_iter=10000, penalty='none', rand
>>> sgd.fit(x, y)
SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.1,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', n_iter=10000, penalty='none', power_t=0.25,
       random_state=2016, shuffle=True, verbose=0, warm_start=False)
>>> regr.score(x, y)
0.40026108237713975
>>> sgd.score(x, y)
0.40025440911212712
>>> print (regr.coef_, regr.intercept_)
[  37.24121082 -106.57751991  787.17931333  416.67377167] 152.133484163
>>> print (sgd.coef_, sgd.intercept_)
[  37.2552641  -106.59457356  787.22821275  416.66054521] [ 151.93457232]
```

Linear Regression Visualization

```
diabetes = datasets.load_diabetes()
x = diabetes.data[:,np.newaxis,2]
y = diabetes.target
regr.fit(x, y)

import matplotlib.pyplot as plt
plt.scatter(x, y, color='red')
lx = np.arange(min(x), max(x), (max(x) - min(x)) / 200).reshape(200,1)
plt.plot(lx, regr.predict(lx), color='blue', linewidth=3)
plt.show()
```

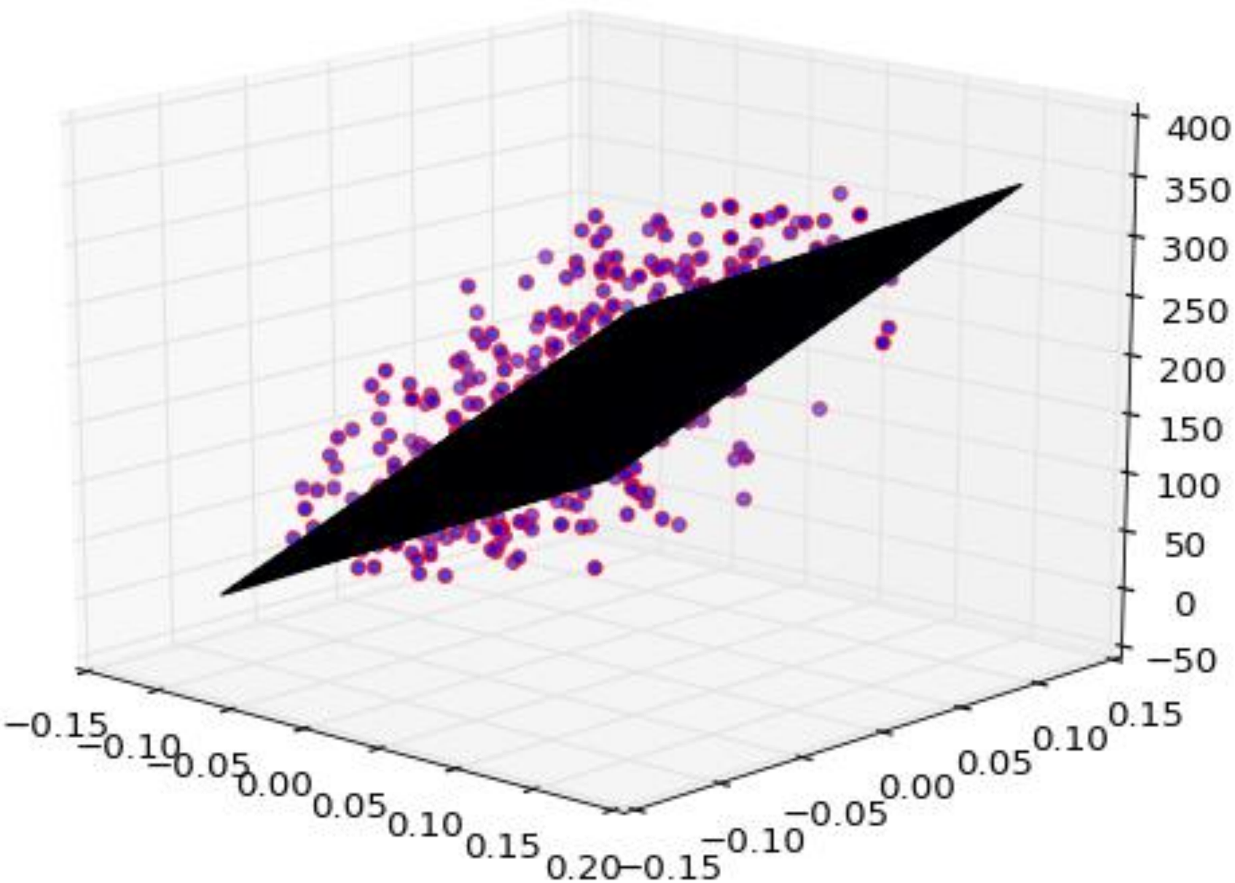
Linear Regression Visualization



Linear Regression Visualization

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

diabetes = datasets.load_diabetes()
x = diabetes.data[:,[2,8]]
y = diabetes.target
regr = linear_model.LinearRegression()
regr.fit(x, y)
steps = 40
lx0 = np.arange(min(x[:,0]), max(x[:,0]), (max(x[:,0]) - min(x[:,0])) / steps).r
eshape(steps,1)
lx1 = np.arange(min(x[:,1]), max(x[:,1]), (max(x[:,1]) - min(x[:,1])) / steps).r
eshape(steps,1)
xx0, xx1 = np.meshgrid(lx0, lx1)
xx = np.zeros(shape = (steps,steps,2))
xx[:,:,0] = xx0
xx[:,:,1] = xx1
x_stack = xx.reshape(steps ** 2, 2)
y_stack = regr.predict(x_stack)
yy = y_stack.reshape(steps, steps)

fig = plt.figure()
ax = fig.gca(projection = '3d')
ax.scatter(x[:,0], x[:,1], y, color = 'red')
ax.plot_surface(xx0, xx1, yy, rstride=1, cstride=1)
plt.show()
```

Linear Regression Visualization



Takeaways

- Basic Python Syntax
- Python functions, modules, and classes
- Linear Regression
- Polynomial Models
- Gradient Descent
- Visualization of Linear Regression