MITB B.9
Applied Machine Learning

Instructor:

Asst. Prof. DAI Bing Tian (Education)

btdai@smu.edu.sg SIS bldg. room 0430

Teaching Assistant:

Ms. YANG Cheng

cheng.yang.2016@mitb.smu.edu.sg

Recap

- Machine learning framework
  - ?
  - ?
  - ?

Agenda

- Regression
- Linear Regression
- Polynomial Regression
- Gradient Descent

Wedding Ang Pow Rate

Hotel	Lunch or Dinner	W/day or W/end	Rate
3-star	Dinner	Weekday	80
3-star	Lunch	Weekend	90
4-star	Lunch	Weekend	100
4-star	Dinner	Weekend	120
5-star	Lunch	Weekend	130
5-star	Dinner	Weekend	160
5-star	Dinner	Weekday	140

Supervised Learning

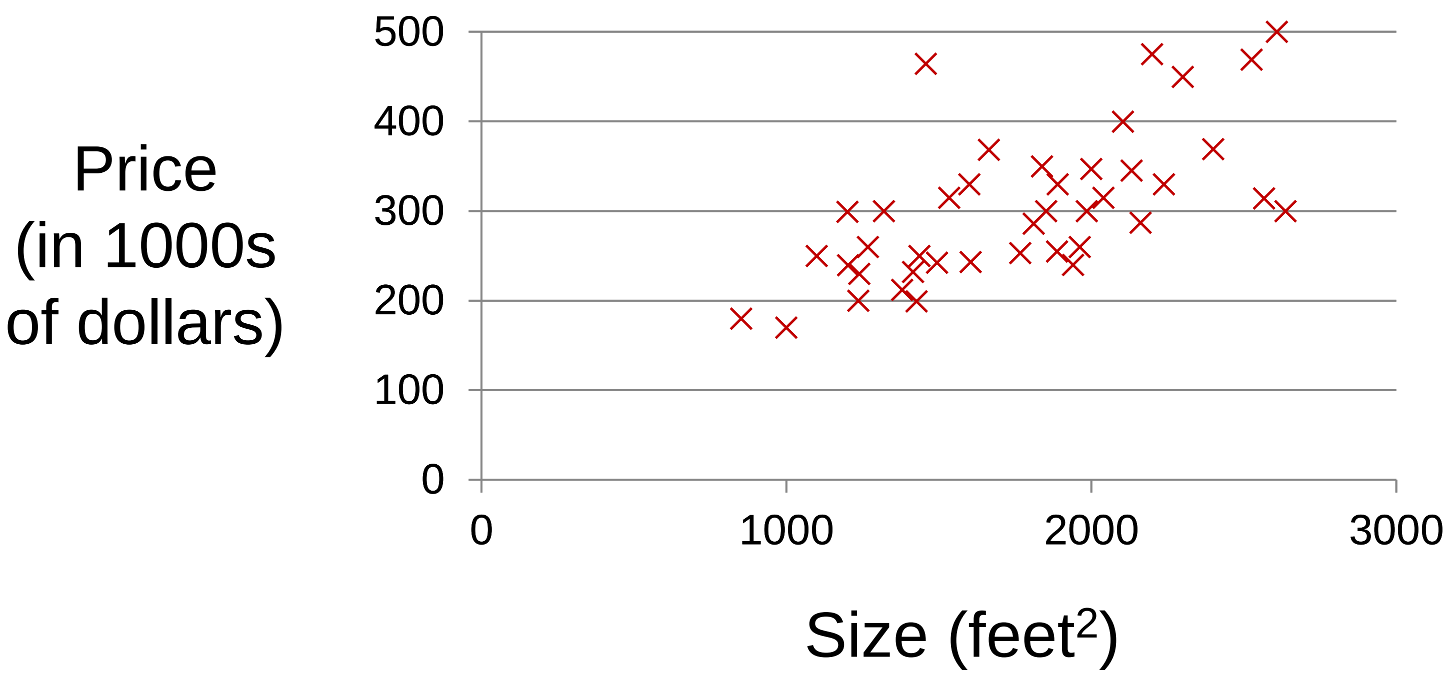
- Formalization
  - Input: $\mathbf{x} \quad \in \mathcal{X} \quad \mathbb{R}^n$
  - Output: $y \quad \in \mathcal{Y} \begin{cases} \mathbb{R} & \text{regression} \\ \{+1, -1\} & \text{binary classification} \\ \{1, 2, \ldots, K\} & \text{multi-class classification} \end{cases}$
  - Target function: $f : \mathcal{X} \rightarrow \mathcal{Y}$ (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 feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

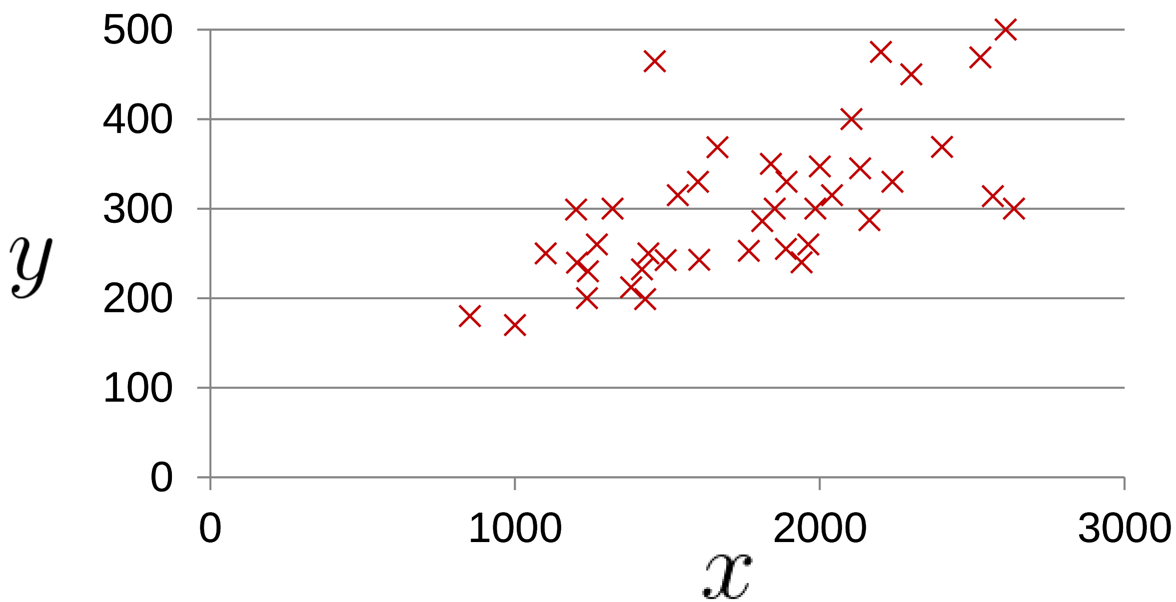
- **x** = input variable / explanatory variable
- **y** = output variable / target variable

Target Functions



A Learning Problem

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



Hypothesis Spaces

- Linear models

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

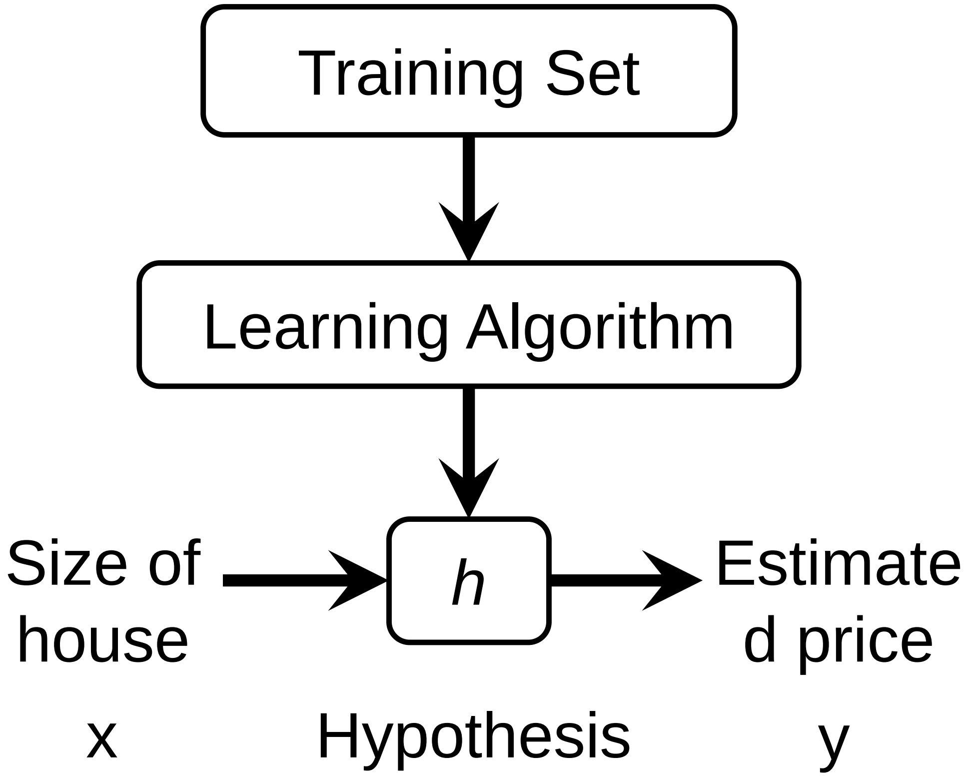
  - **Infinite** possible hypotheses!
  - Any choices of coefficient a and b will result in a possible hypothesis
- Polynomial models

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

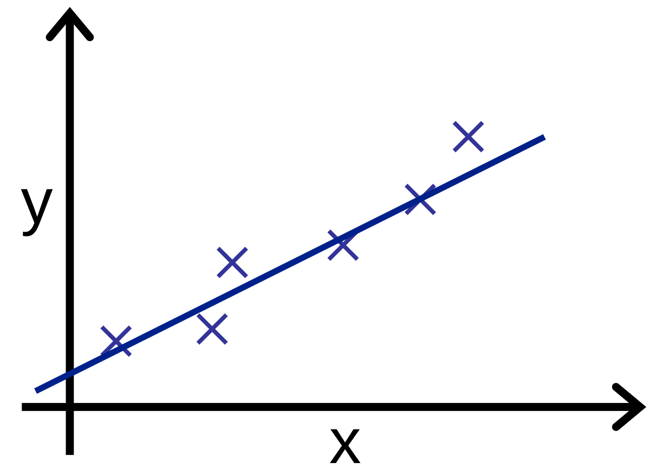
- Any nonlinear models

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

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:

$$w_0, w_1$$

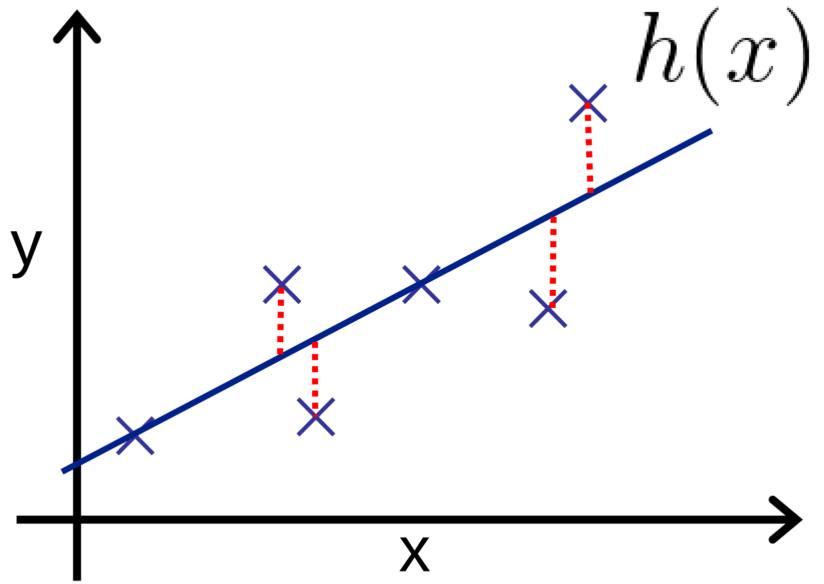
Cost Function:

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

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

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

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

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

Univariate Linear Regression

- Identify explanatory variable as x
- Identify target variable as y

```
import pandas as pd
import numpy as np

data_house = pd.read_csv('house_price.tsv', sep='\t')

x = data_house.as_matrix(['size'])
y = data_house.as_matrix(['price'])

from sklearn import linear_model

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)
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

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

$$\begin{aligned}\nabla J(\mathbf{w}) &= \nabla_{\mathbf{w}}\left(\frac{1}{2m}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y})\right) \\ &= X^T X\mathbf{w} - X^T\mathbf{y} = \mathbf{0}\end{aligned}$$

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

- Analytical solution

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

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

Model Parameters

- Print model parameters:
  - Coefficients
  - intercept

```
x = data_house.as_matrix(['size', 'Taxes'])
y = data_house.as_matrix(['price'])

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

print('Coefficients:', regr.coef_)
print('Intercept:', regr.intercept_)
```

Sum of Squares

- In statistical linear models, the total sum of squares (SS_{tot}) can be partitioned into explained sum of squares (a.k.a. SS_{reg} = Sum of Squares due to Regression) and residual sum of squares (a.k.a. SS_{res} = Sum of Squares of Errors)

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

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

$$SS_{reg} = \sum_i (\hat{y}_i - \bar{y})^2$$

$$SS_{res} = \sum_i (\hat{y}_i - y_i)^2$$

$\bar{y}$: mean of y_i

$\hat{y}_i$: prediction of y_i

Mean Squared Error

- How to evaluate the regression model?
- Cost Function:

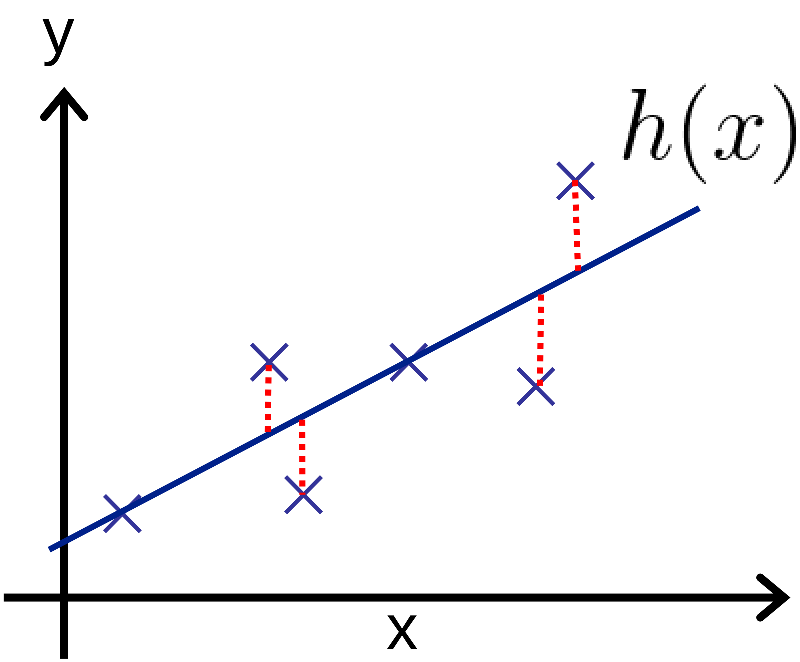
Mean Squared Error (MSE)

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

SS_{res}

Goal:

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



Coefficient of Determination

- Calculate R^2 score

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

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

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

Calculation of R^2 Score

- Calculate from $\text{score}()$ function
- Calculate from SS_{reg} and SS_{tot}

```
sst = sum((y - np.mean(y)) ** 2)
ssr = sum((regr.predict(x) - np.mean(y)) ** 2)
sse = sum((regr.predict(x) - y) ** 2)

print('Total sum of squares:', sst)
print('Explained sum of squares:', ssr)
print('Residual sum of squares:', sse)
print('R^2 score computed from score function:', regr.score(x, y))
print('R^2 score computed from ssr / sst:', ssr / sst)
```

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

```
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)
print('R^2 score: %.2f' % regr.score(x_test, y_test))
```

```
>>> train[:20]
array([False,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True], dtype=bool)
```

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_test)
metrics.explained_variance_score(y_test, y_pred)
metrics.mean_absolute_error(y_test, y_pred)
metrics.mean_squared_error(y_test, y_pred)
```

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

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

- 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

$$h(x) = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$$

```
from sklearn import preprocessing

poly2 = preprocessing.PolynomialFeatures(2)
poly3 = preprocessing.PolynomialFeatures(3)

x2 = poly2.fit_transform(x)
x3 = poly3.fit_transform(x)

x_train = x2[train,:]
x_test = x2[~train,:]
regr.fit(x_train, y_train)
print('R^2 score: %.2f' % regr.score(x_test, y_test))

x_train = x3[train,:]
x_test = x3[~train,:]
regr.fit(x_train, y_train)
print('R^2 score: %.2f' % regr.score(x_test, y_test))
```

Regression without Intercept

- If there exists constant term, the coefficient of the constant term is actually the intercept
- No intercept model can be achieved by polynomial models

```
regr_no_intercept = linear_model.LinearRegression(fit_intercept=False)

x_train = x2[train,:]
x_test = x2[~train,:]
regr.fit(x_train, y_train)
regr_no_intercept.fit(x_train, y_train)

print('Coefficients:', regr.coef_)
print('Intercept:', regr.intercept_)

print('Coefficients:', regr_no_intercept.coef_)
print('Intercept:', regr_no_intercept.intercept_)
```

Polynomial Models

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

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

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

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

Customized Mapping Function

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

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

Customized Mapping Function

- Only quadratic terms
- No linear terms

```
terms = [{0:2}, {1:2}, {0:1,1:1}]
x_mapped = map_to_higher_dim(x, terms)
x_train = x_mapped[train,:]
x_test = x_mapped[~train,:]
regr.fit(x_train, y_train)
print('R^2 score: %.2f' % regr.score(x_test, y_test))
```

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

$$\boxed{h(x) = g(x) \approx f(x)}$$

Gradient Descent

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

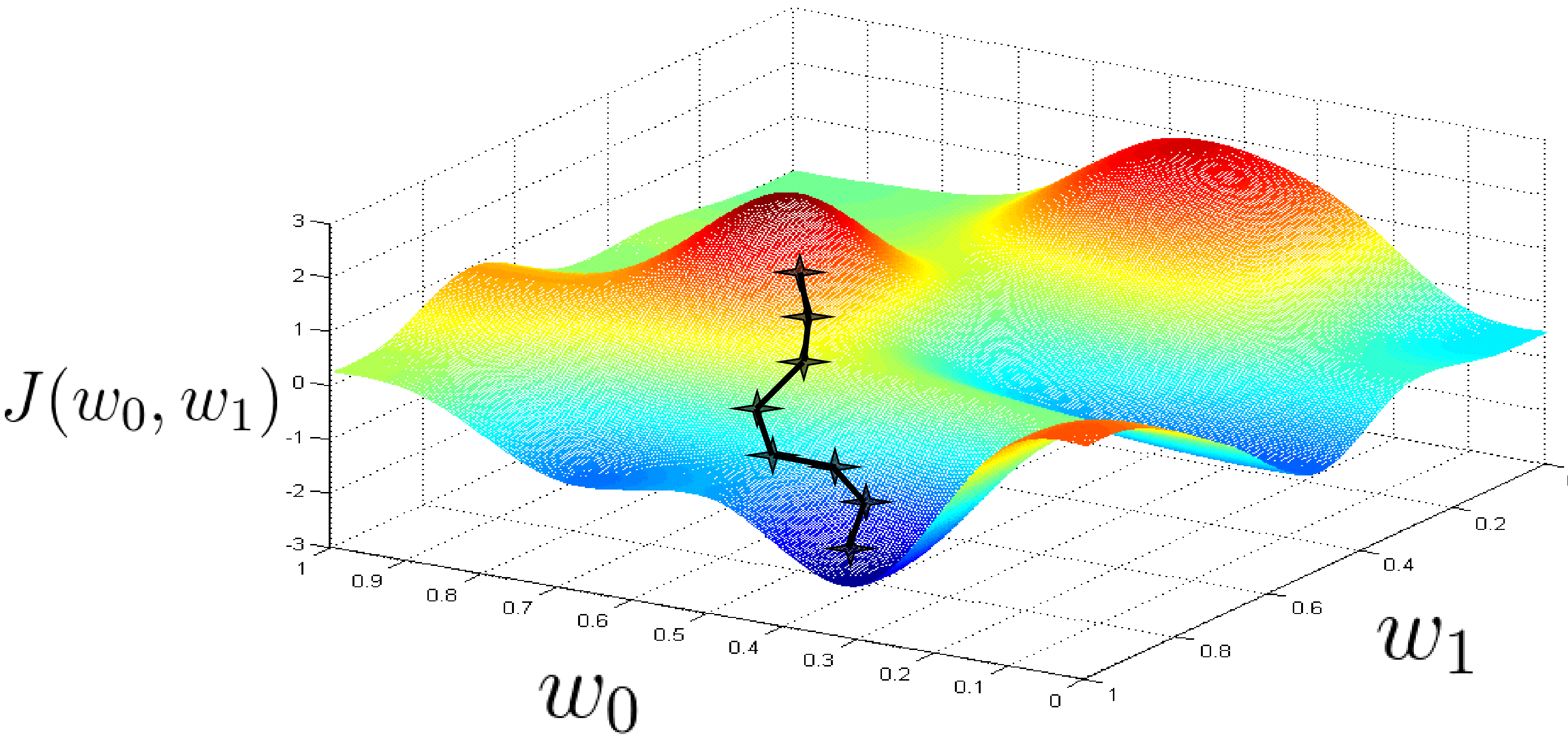
Want to optimize $\min\limits_{w_0, w_1} J(w_0, w_1)$

Outline:

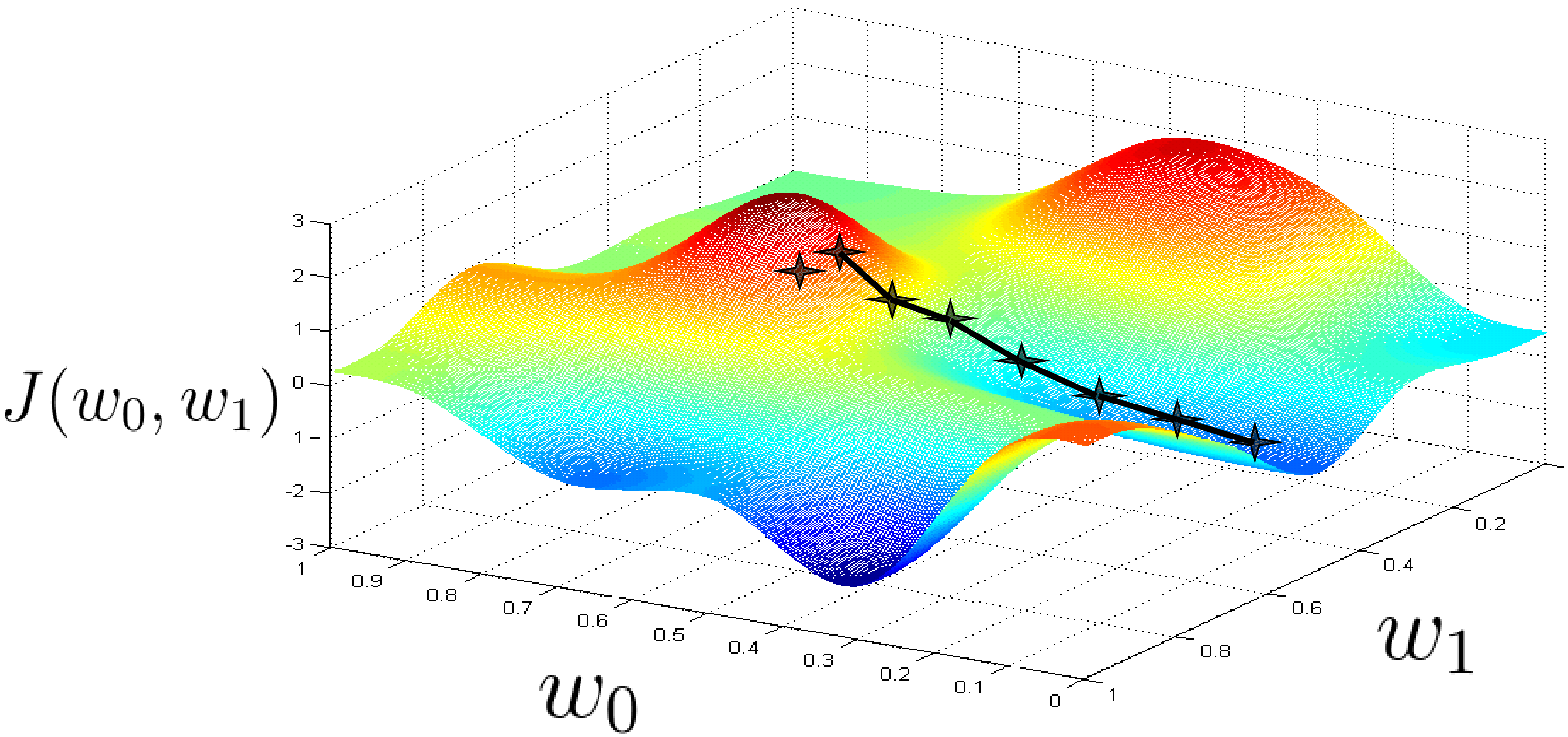
- Start with some w_0, w_1
- Keep changing w_0, w_1 to reduce $J(w_0, w_1)$

  until we hopefully end up at a minimum

Gradient Descent



Gradient Descent



Gradient Descent Algorithm

Gradient descent algorithm

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

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

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

$$\}$$

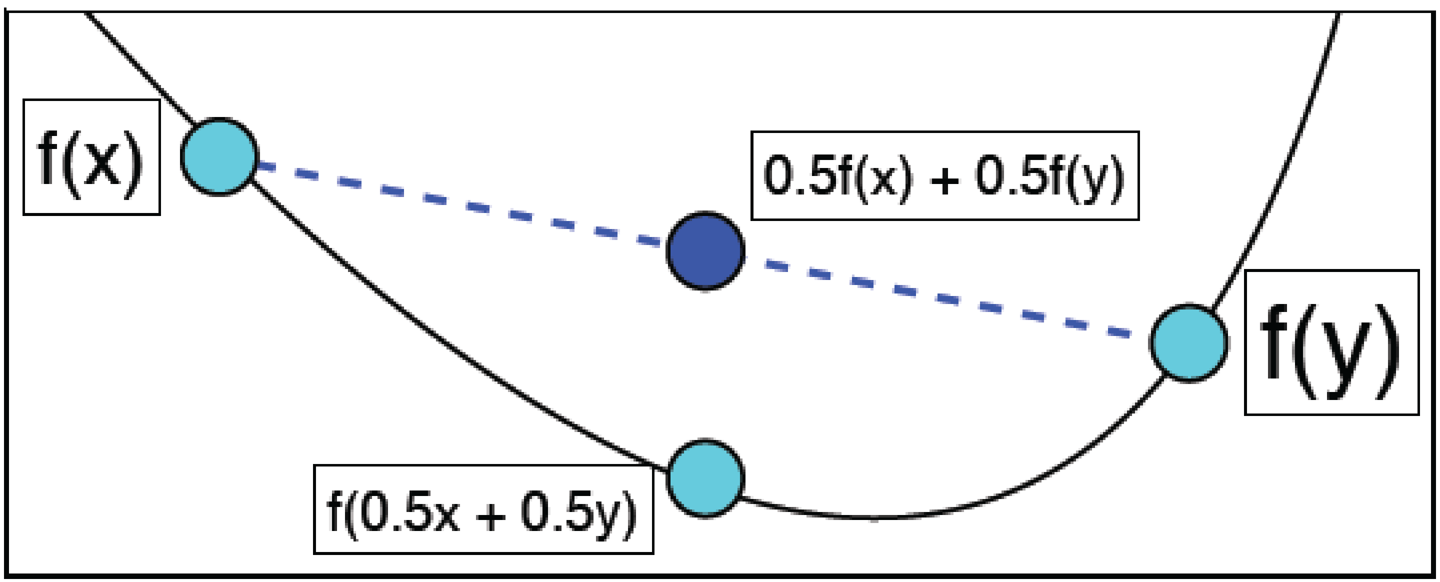
learning rate parameter
(rule of thumb: 0.1)

Convex Function

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

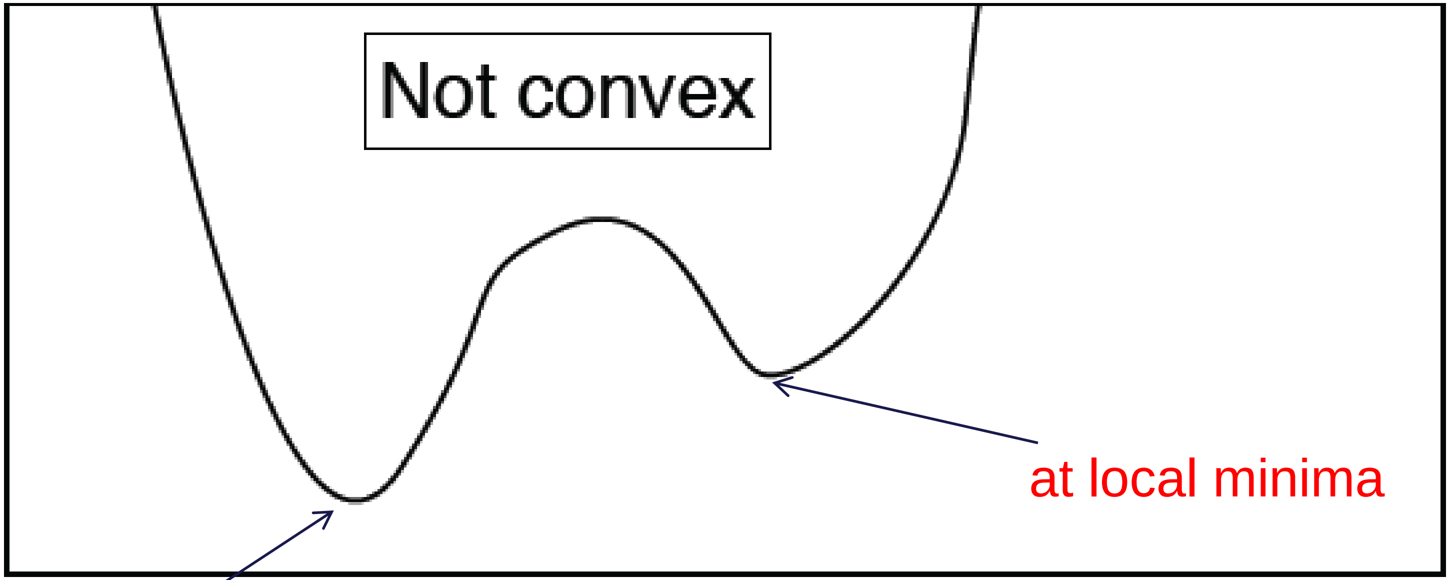
$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y) \quad \forall 0 \leq \theta \leq 1$$

- 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

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

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

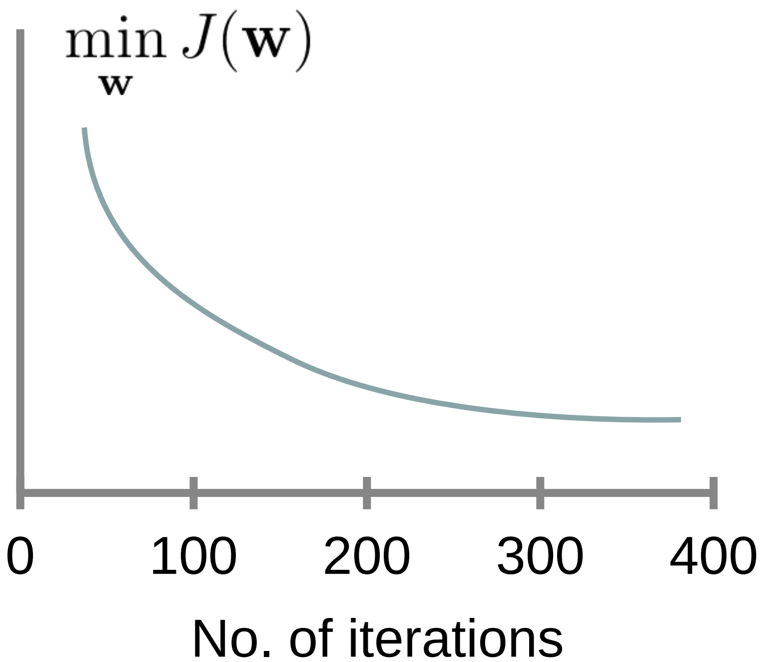
$$\frac{\partial}{\partial w_j} J(w_0, w_1) = \frac{\partial}{\partial w_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left((w_0 + w_1 x_i) - y_i\right)^2 \right)$$

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

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

Convergence and Learning Rate

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

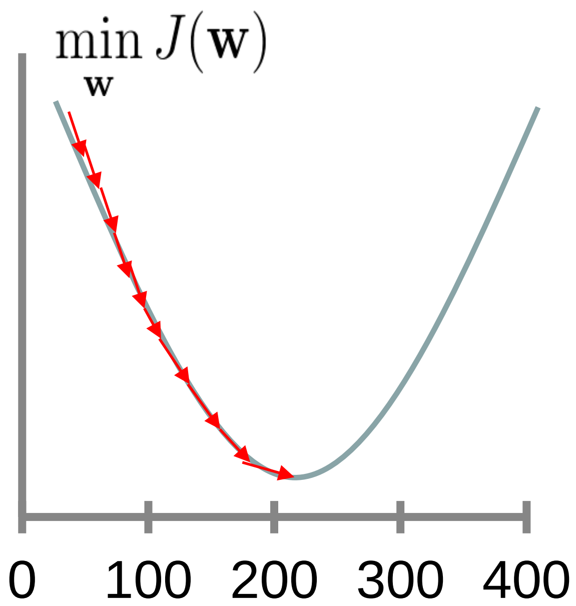


Example automatic convergence test:

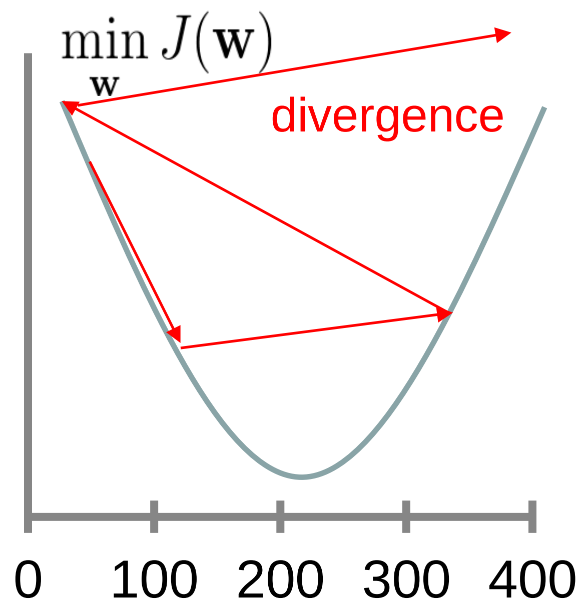
Declare convergence if $J(\mathbf{w})$ decreases by less than 10^{-3} in one iteration.

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.

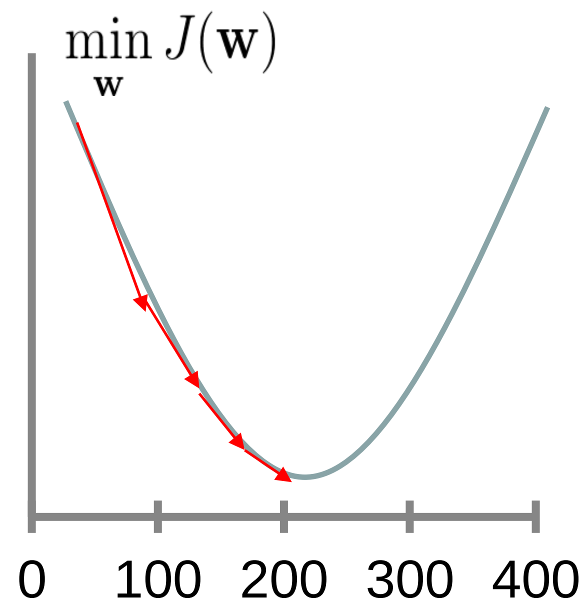
Learning Rate



too small constant



too large



gradually decreased

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

Gradient Descent Regression

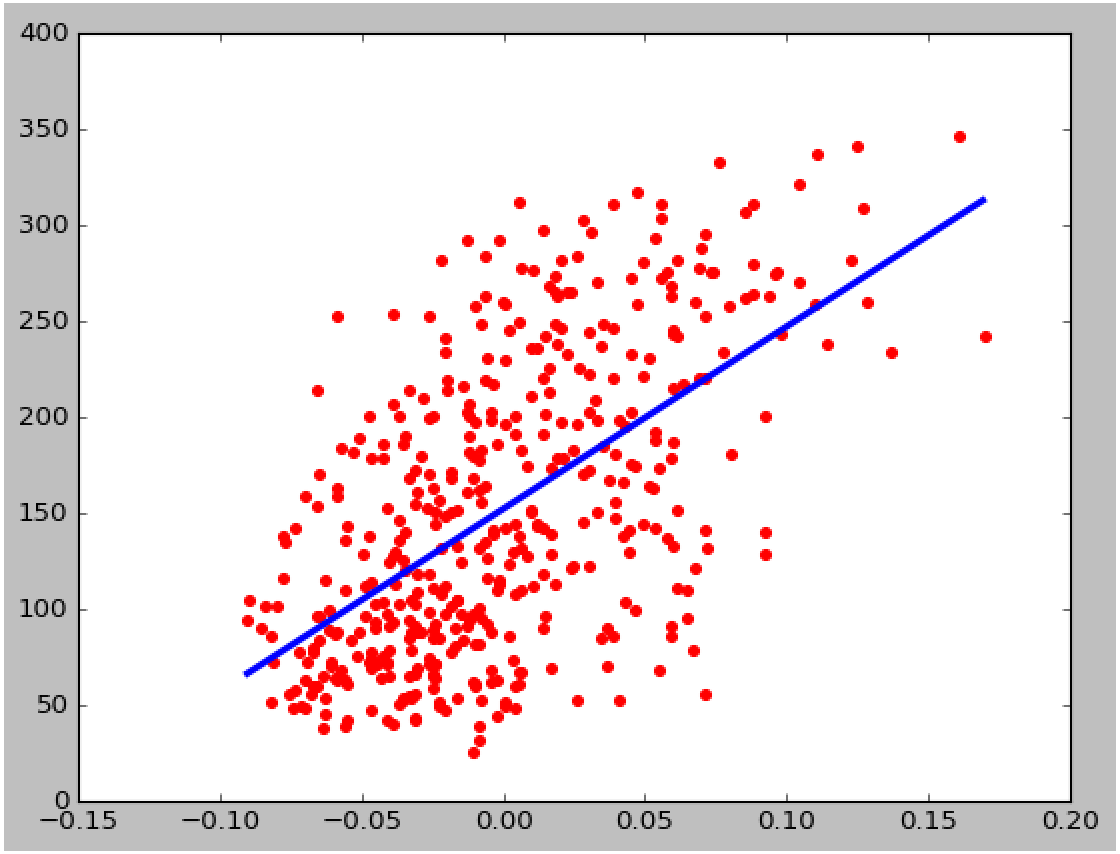
```
from sklearn import datasets
diabetes = datasets.load_diabetes()
x = diabetes.data[:,:4]
y = diabetes.target
regr = linear_model.LinearRegression()
regr.fit(x, y)
sgd = linear_model.SGDRegressor(n_iter=100000, penalty='none')
sgd.fit(x, y)
regr.score(x, y)
sgd.score(x, y)
print(regr.coef_, regr.intercept_)
print(sgd.coef_, sgd.intercept_)
```

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



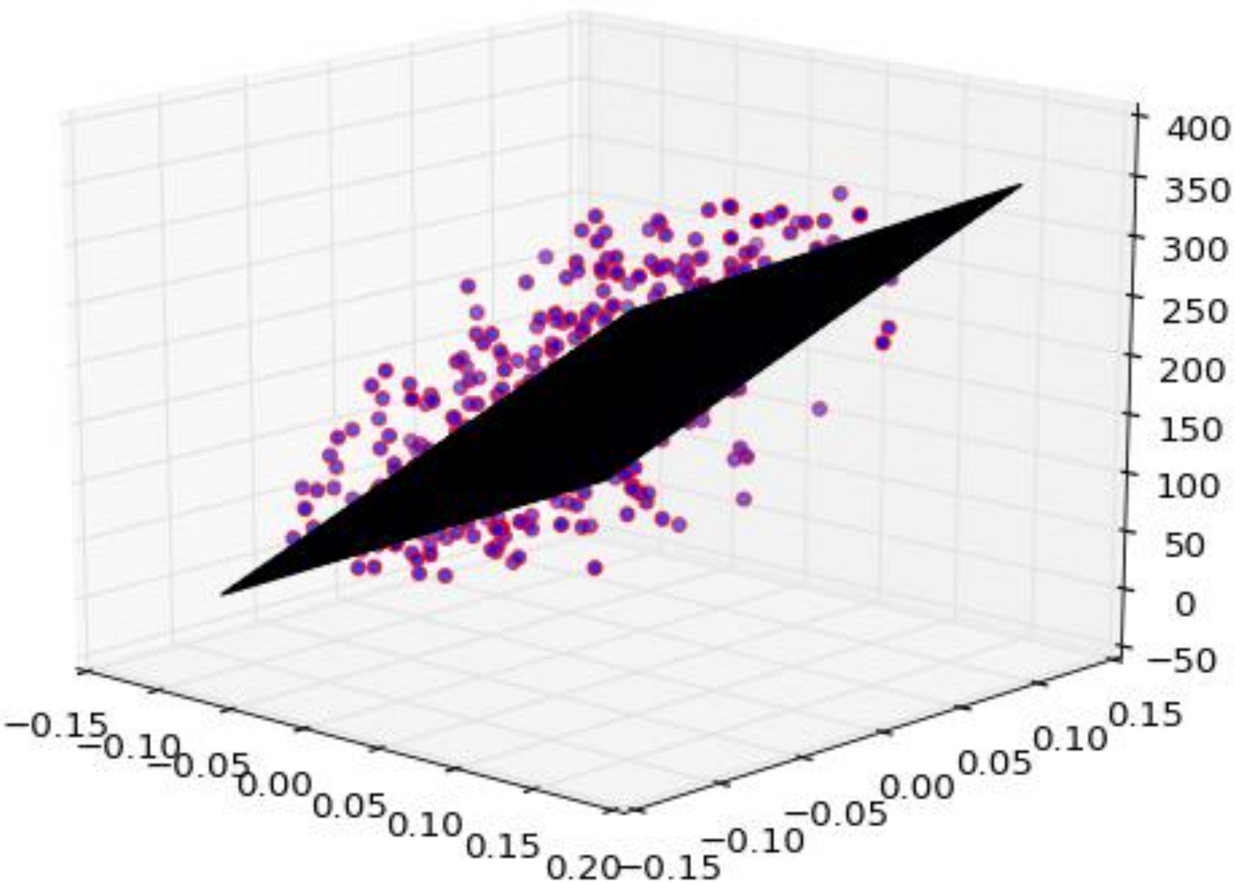
Linear Regression Visualization

```
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



QUESTIONS?!