Linear Regression Analysis
- Least Sum of Squares Approach -

Tseng-Ching James Shen, Ph.D.

Motivation of Linear Regression

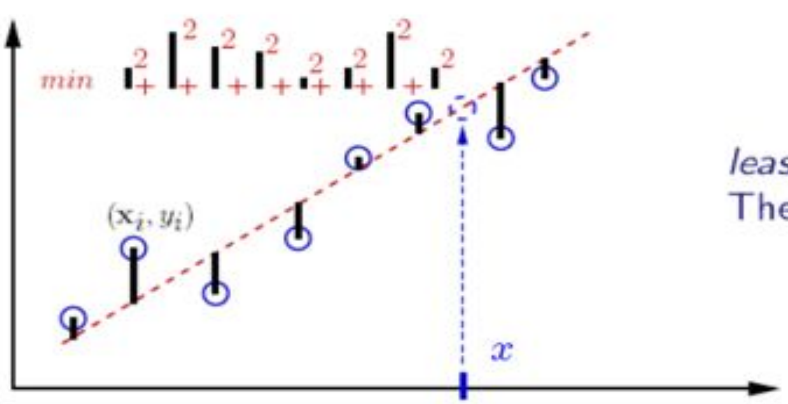
- Given two random variables x, and y, we use the following linear equation to model the correlation between two random variables

  $$y = w\,x + b$$

- The coefficient w and correlation are related in the following way:
  - $w > 0$, positive correlation
  - $w = 0$, no correlation
  - $w < 0$, negative correlation

Motives of Linear Regression (cont.)

- The parameters w and b are unknown and can be determined by fitting the model with the dataset
- Fit model by minimizing the sum of squared errors (i.e. training errors)



least squares (LSQ)
The fitted line is used as a predictor

Linear Regression Model

- Features are random variables $x_1, x_2, \ldots, x_d$
- The model can be generalized to model response y is also a random variable as follows

$$y = w_0 + w_1 x_1 + \ldots. + w_d x_d$$

- It can be also expressed in terms of vector notation

$$y = h(x) = x^T \boldsymbol{w} + w_0$$

Linear Regression Model (cont.)

- Given a set of observed dataset points, we need to determined the coefficients which are best to fit the dataset points ($\boldsymbol{x}_i$, y_i) I = 1, …, N
- For dataset point ($\boldsymbol{x}_i$, y_i) (*revised version*), the estimated y_i

  $$h(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \boldsymbol{w}$$

  $$\mathbf{x}_i = (1, x_{10}, x_{11}, \ldots, x_{1d}), \mathbf{w} = (w_0, w_1, \ldots w_d)$$
- Let $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots \boldsymbol{x}_N)^T$

  $$(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_N))^T = \mathbf{X}\,\mathbf{w}$$

Optimization Problem

- We can use training errors as the objective function to be minimized

$$\frac{1}{N}\sum_{i=1}^{N}(\mathrm{h}(\boldsymbol{x_i}) - y_i)^2$$

- Let $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots \boldsymbol{x_N})^{\mathrm{T}}$

$$(\mathrm{h}(\boldsymbol{x_1}), \ldots, \mathrm{h}(\boldsymbol{x_N}))^{\mathrm{T}} = \mathbf{X}\,\mathbf{w}$$

- The objective function can be re-written as

$$£(\mathbf{w}) = \frac{1}{N}(\mathbf{X}\,\mathbf{w} - \mathbf{y})^{\mathrm{T}}(\mathbf{X}\,\mathbf{w} - \mathbf{y})$$

Closed Form Solution

- Expand the objective function

$$£(\mathbf{w}) = \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{X}\,\mathbf{w} - \frac{2}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{N}\mathbf{y}^T\mathbf{y}$$

- Find the solution which satisfies the following equation

$$\frac{\partial £}{\partial \mathbf{w}} = 0$$

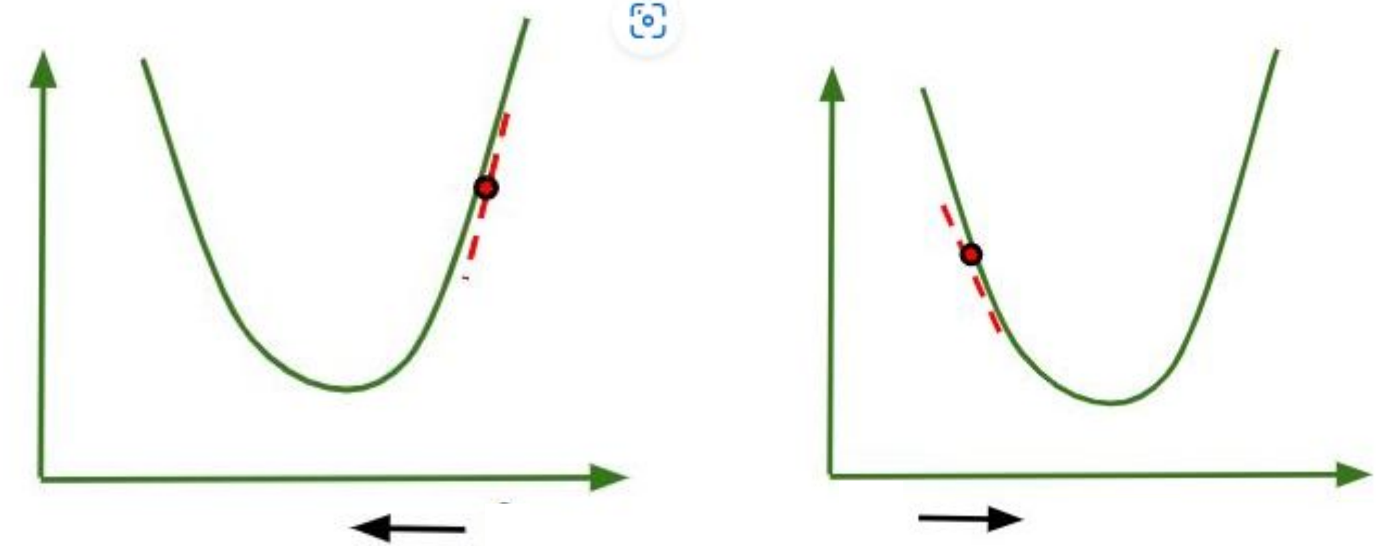
which leads to $\mathbf{X}^T\mathbf{X}\,\mathbf{w} = \mathbf{X}^T\mathbf{y}$

- Therefore, the solution is $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
- Use Python NumPy to implement the solution

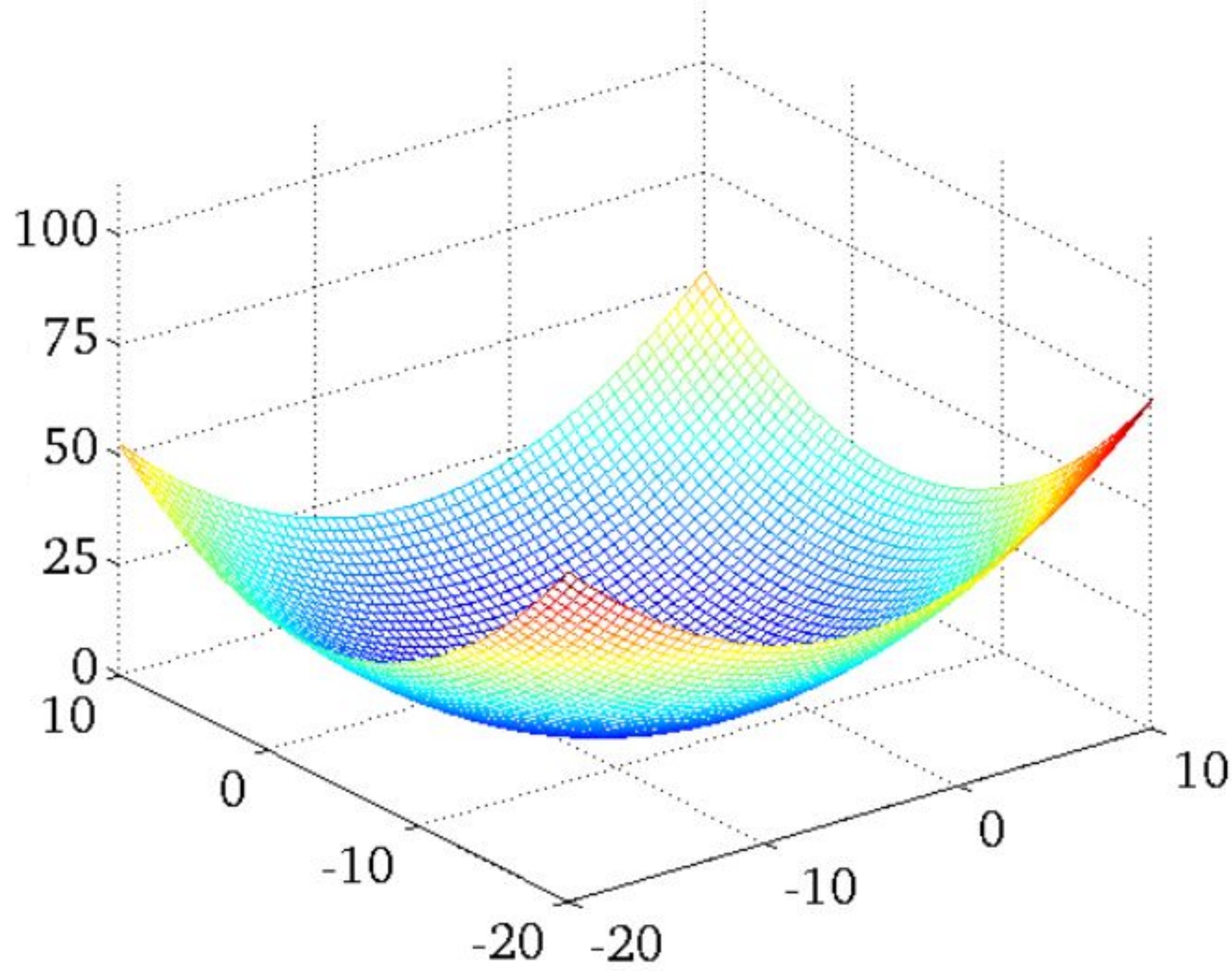
```
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
```

Gradient Descent

- The objective function is a differential and convex function
- Gradient descent is a iterative process
  - Pick a starting point
  - At each iteration move to the next point by applying the slope of current point (derivative of the function)



3-D Convex Function



Derivative of Function

- The derivative of the function is

$$\frac{\partial}{\partial \boldsymbol{w}} £(\mathbf{w}) = \frac{\partial}{\partial \boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} (\mathrm{h}(\boldsymbol{x}_i) - y_i)^2$$

- Since $\frac{\partial}{\partial \boldsymbol{w}} £(\mathbf{w}) = (\frac{\partial}{\partial w_0} £(\mathbf{w}), \ldots, \frac{\partial}{\partial w_d} £(\mathbf{w}))$, we will find $\frac{\partial}{\partial w_j} £(\mathbf{w})$ j = 0, 1, 2, ..., d
- The equation for $\frac{\partial}{\partial w_j} £(\mathbf{w})$

$$\frac{\partial}{\partial w_j} £(\mathbf{w}) = \frac{\partial}{\partial w_j} \frac{1}{N} \sum_{i=1}^{N} (\sum_{k=0}^{k=d} w_d \, x_{id} - y_i)^2$$

Derivation of Function

- Apply the Derivative Chain Rule:

$$\frac{\partial}{\partial w_j}\pounds(\mathbf{w}) = \frac{2}{N}\sum_{i=1}^{N}\left(\sum_{k=0}^{k=d} w_d\, x_{id} - y_i\right)\frac{\partial}{\partial w_j}\left(\sum_{k=0}^{k=d} w_d\, x_{id} - y_i\right)$$

- We will get the following equation

$$\frac{\partial}{\partial w_j}\pounds(\mathbf{w}) = \frac{2}{N}\sum_{i=1}^{N}\left(\sum_{k=0}^{k=d} w_d\, x_{id} - y_i\right) x_{ij}$$

Gradient Descent Algorithm

- Initialize **w**
- Update w_j at each iteration using the following equation until **w** converges

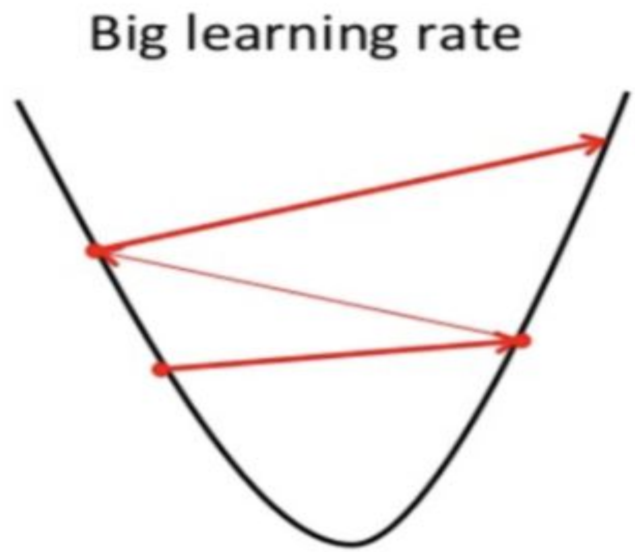
$$w_j \leftarrow w_j + \alpha \left(\frac{2}{N} \sum_{i=1}^{N} \left(\sum_{k=0}^{k=d} w_d \, x_{id} - y_i\right) x_{ij}\right)$$

where α is the *learning rate*

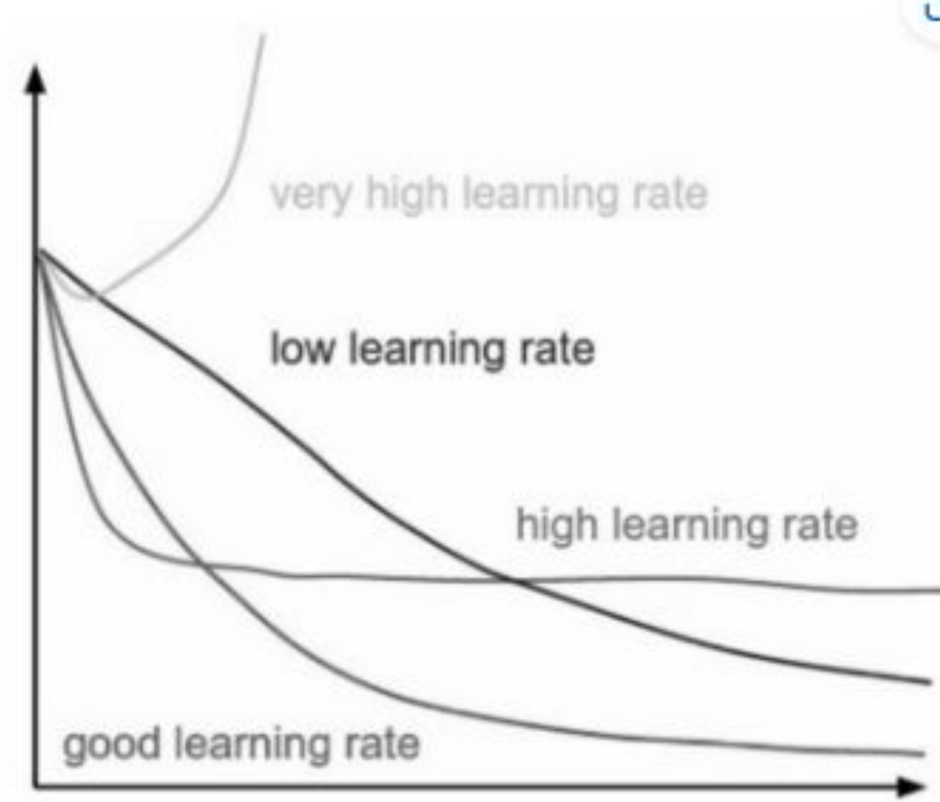
The Learning Rate

- If the learning rate is too high, we might **OVERSHOOT** the minima and keep bouncing, without reaching the minima

- If the learning rate is too small, the training might turn out to be too long



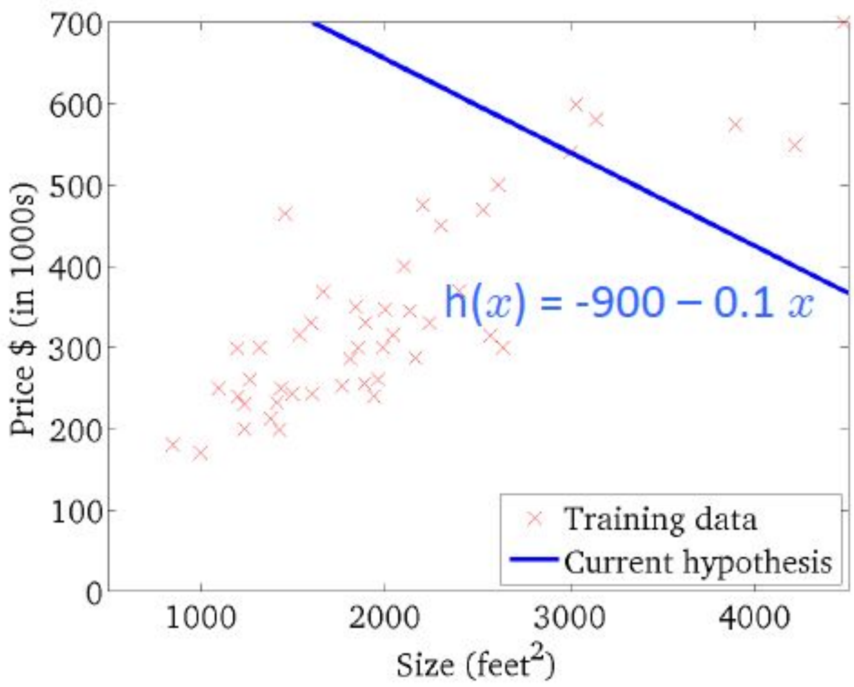
Learning Rate (cont.)

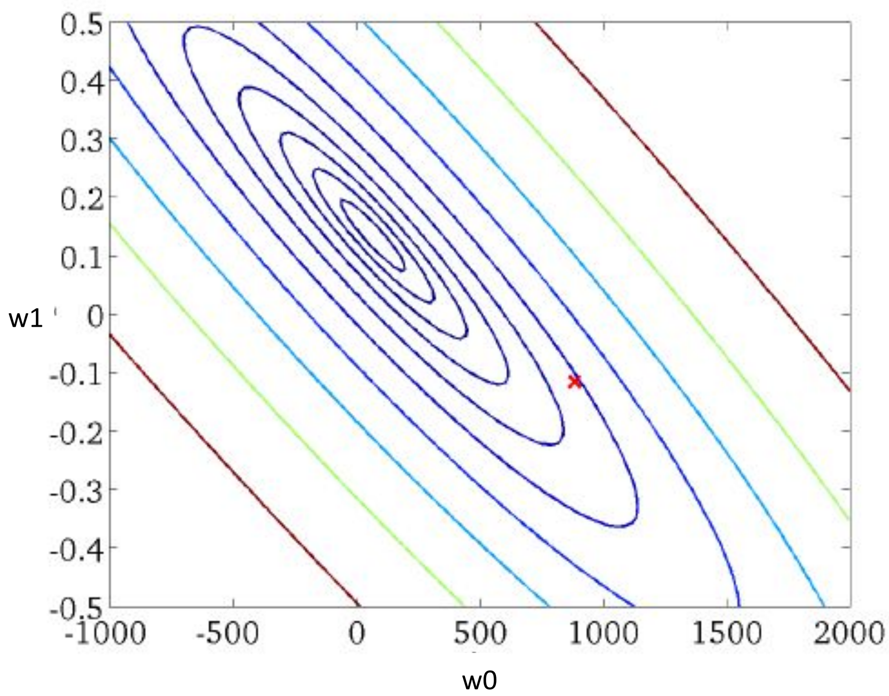


Loss value v.s. the number of iterations

Gradient Descent

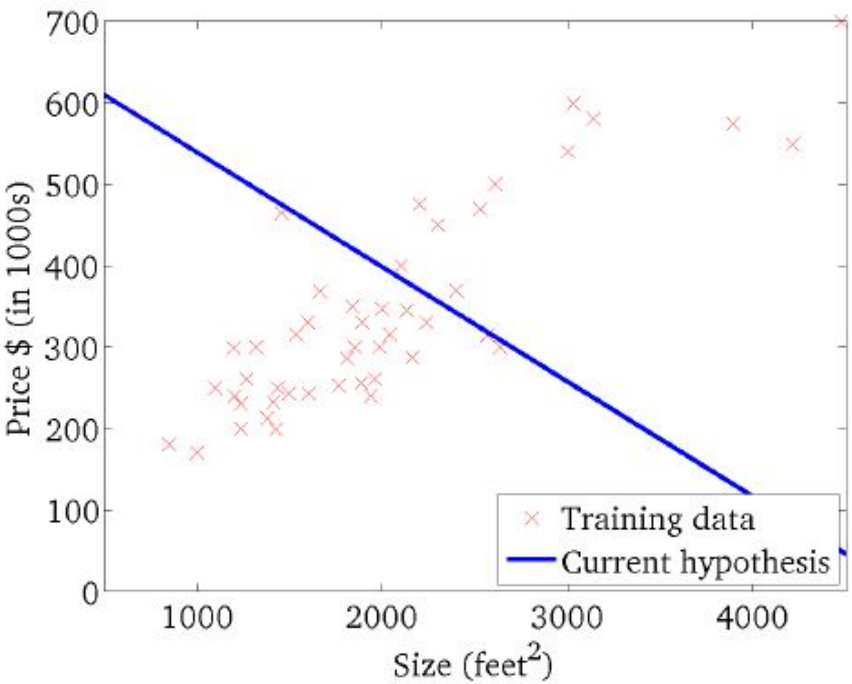
Initial Iteration

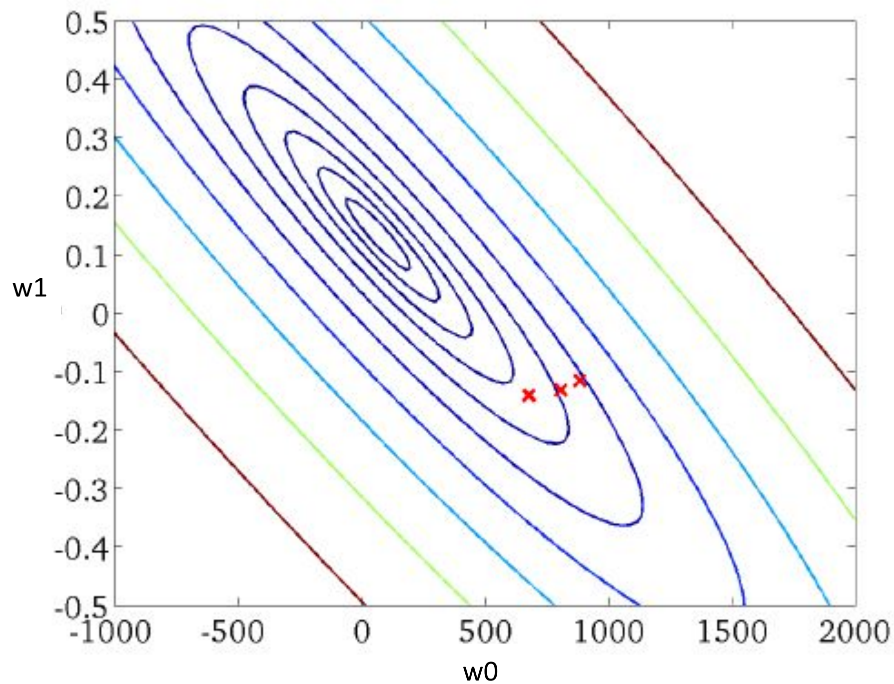




Gradient Descent

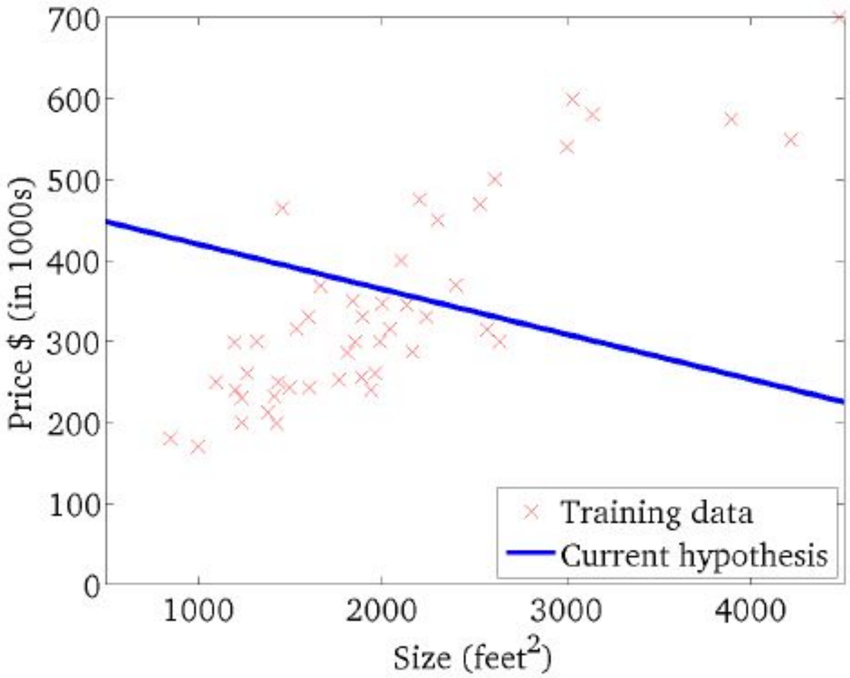
The 3rd Iteration

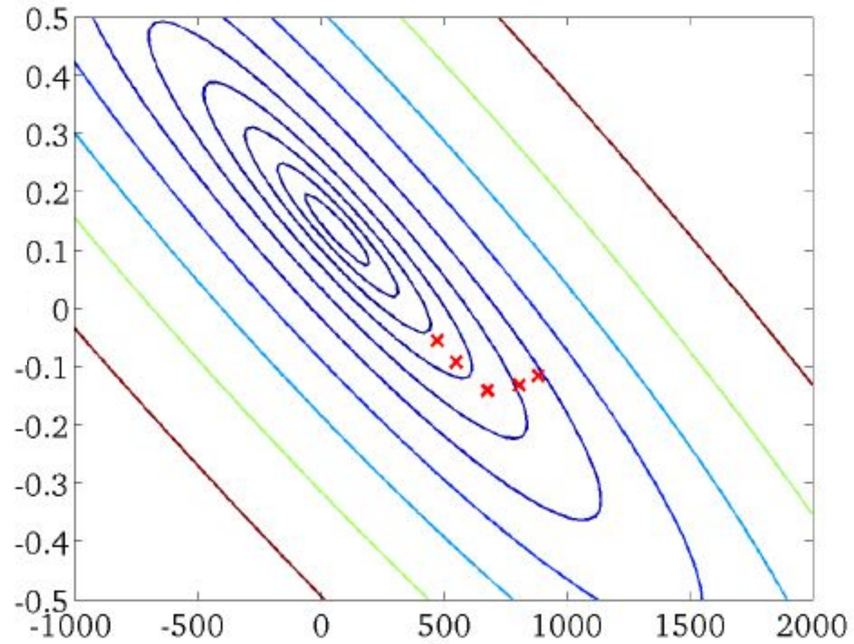




Gradient Descent

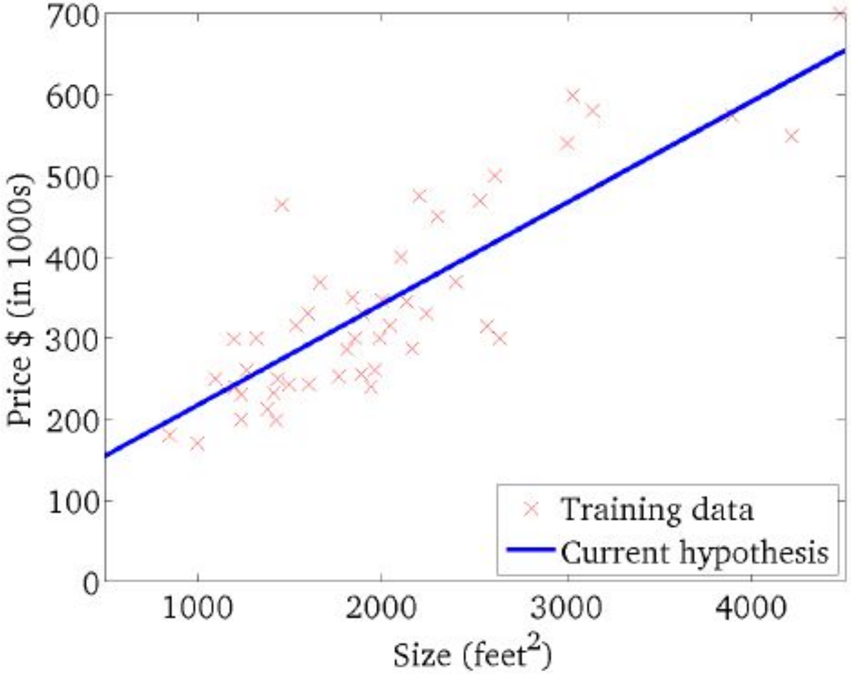
The 5th Iteration

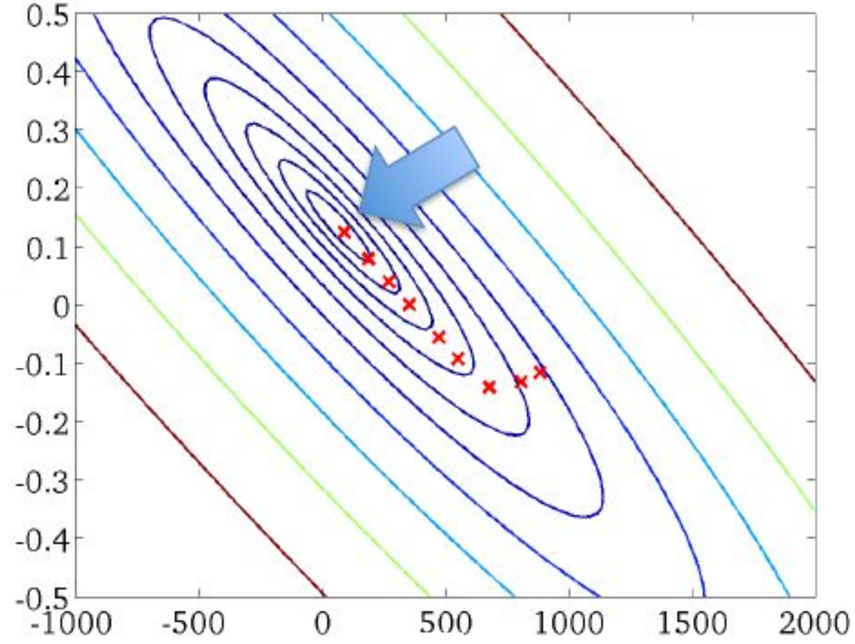




Gradient Descent

The 9th Iteration





Example

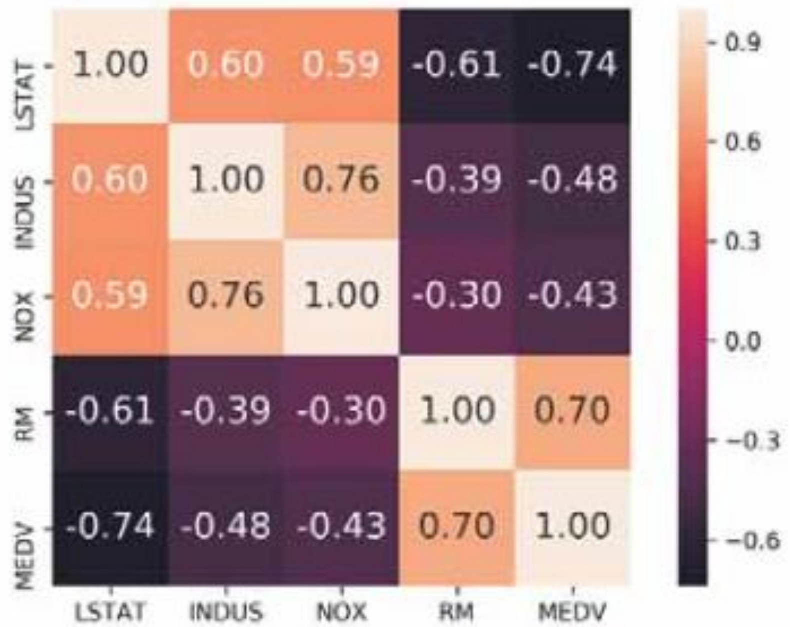
- Chapter 10 in Python Machine Learning, Sebastian Raschka Vahid Mirjalili Packt Publishing Ltd.
- Data preparation

```
>>> import pandas as pd
>>> df = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
...                  'python-machine-learning-book-2nd-edition'
...                  '/master/code/ch10/housing.data.txt',
...                  header=None,
...                  sep='\s+')
>>> df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
...               'NOX', 'RM', 'AGE', 'DIS', 'RAD',
...               'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
>>> df.head()
```

Example (cont.)

- Correlation analysis

```
>>> import numpy as np
>>> cm = np.corrcoef(df[cols].values.T)
>>> sns.set(font_scale=1.5)
>>> hm = sns.heatmap(cm,
...                  cbar=True,
...                  annot=True,
...                  square=True,
...                  fmt='.2f',
...                  annot_kws={'size': 15},
...                  yticklabels=cols,
...                  xticklabels=cols)
>>> plt.show()
```



Example (cont.)

- Feature selection
- Feature scaling

```
>>> X = df[['RM']].values
>>> y = df['MEDV'].values
>>> from sklearn.preprocessing import StandardScaler
>>> sc_x = StandardScaler()
>>> sc_y = StandardScaler()
>>> X_std = sc_x.fit_transform(X)
>>> y_std = sc_y.fit_transform(y[:, np.newaxis]).flatten()
```

Example (cont.)

- Implementation of Gradient Descent

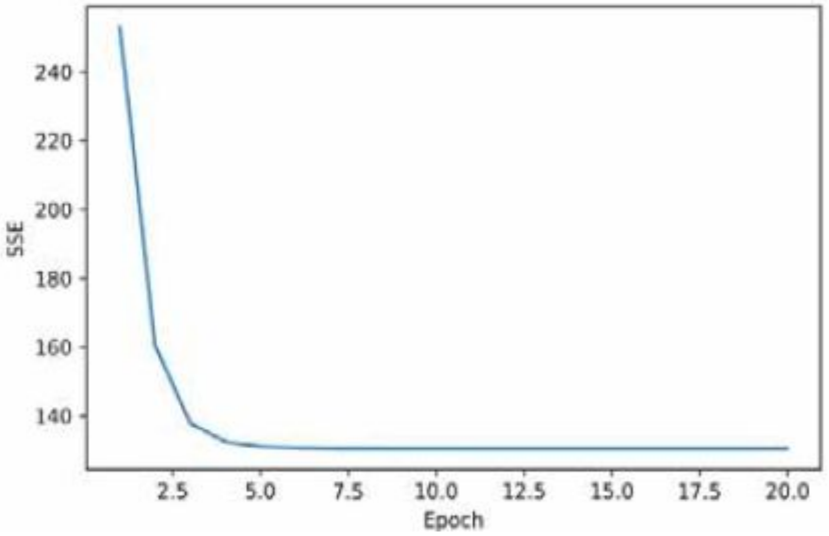
```
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)
```

Example (cont.)

- Usage of GD method

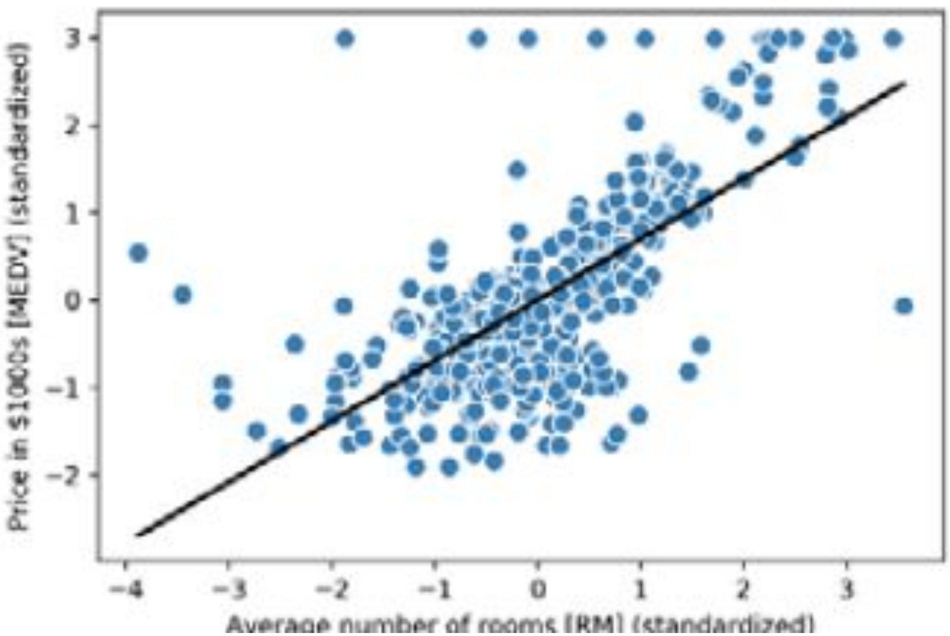
```
>>> lr = LinearRegressionGD()
>>> lr.fit(X_std, y_std)
```

```
>>> sns.reset_orig() # resets matplotlib style
>>> plt.plot(range(1, lr.n_iter+1), lr.cost_)
>>> plt.ylabel('SSE')
>>> plt.xlabel('Epoch')
>>> plt.show()
```



Example (cont.)

```
>>> lin_regplot(X_std, y_std, lr)
>>> plt.xlabel('Average number of rooms [RM] (standardized)')
>>> plt.ylabel('Price in $1000s [MEDV] (standardized)')
>>> plt.show()
```



Example (cont.)

- Scikit-learn library

```
>>> from sklearn.linear_model import LinearRegression
>>> slr = LinearRegression()
>>> slr.fit(X, y)
>>> print('Slope: %.3f' % slr.coef_[0])
Slope: 9.102
>>> print('Intercept: %.3f' % slr.intercept_)
Intercept: -34.671
```

Gradient Descent vs Closed Form

Gradient Descent	**Closed Form Solution**
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}$ is roughly $O(n^3)$

Kernel Regression Analysis

- Linear functions might not be good enough for some datasets
- Kernel method could be used to fit dataset with nonlinear functions
- Apply the kernel method and trick to get

$$y_i = \sum_{j=1}^{N} \boldsymbol{\alpha}_j \, \phi^T(\mathbf{x}_j) \, \phi(\mathbf{x}_i)$$

- Kernel matrix $\mathbf{K}(i, j) = \phi^T(\mathbf{x}_i) \, \phi(\mathbf{x}_j)$