法律声明

- 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，小象学院和主讲老师拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意及内容，我们保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

回归与特征选择

小象学院 ChinaHadoop.cn　邹博

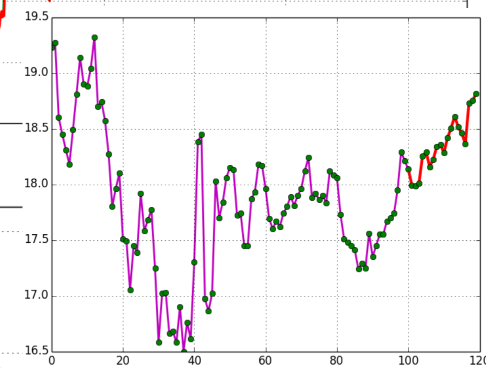
主要内容

- 线性回归
  - 高斯分布
  - 最大似然估计MLE
  - 最小二乘法的本质
- Logistic回归
  - 分类问题的首选算法
- 技术点
  - 梯度下降算法
  - 最大似然估计
  - 特征选择

股价预测

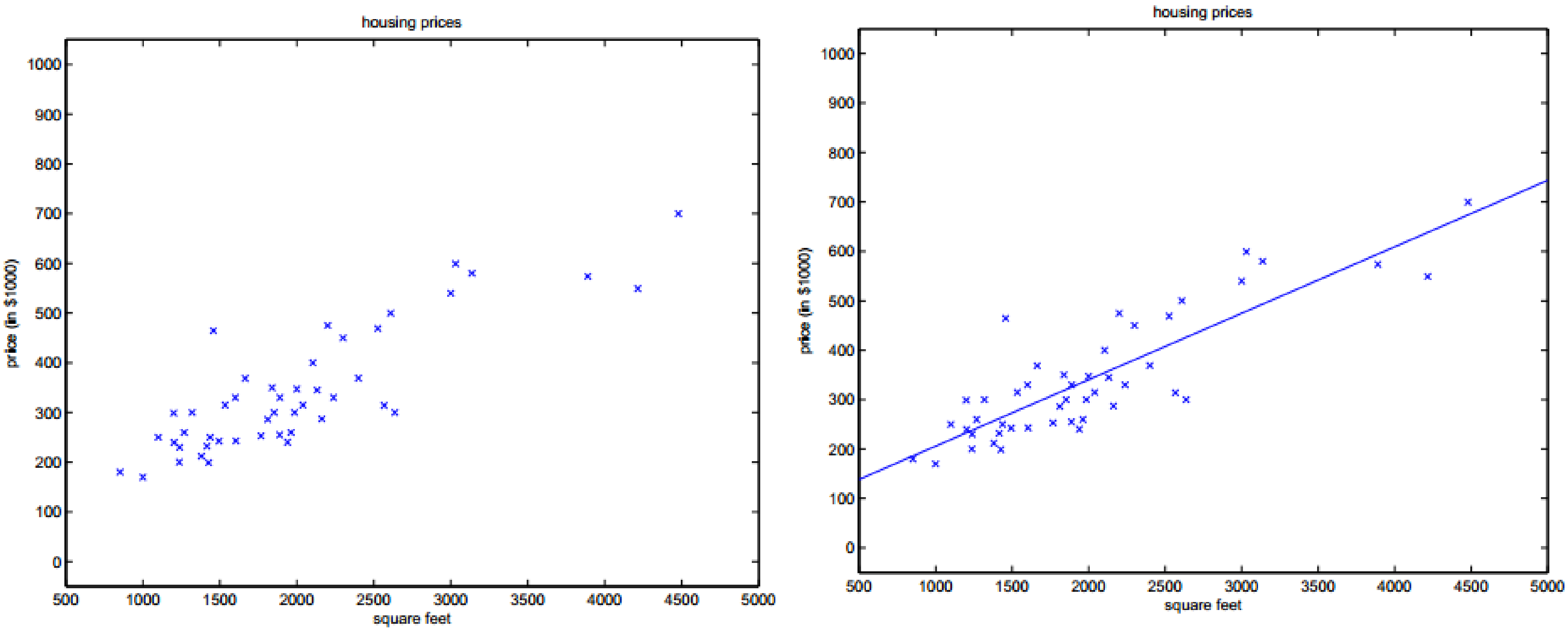
- 方法：自回归
- 参数：100阶

```
if __name__ == "__main__":
    #  日期    开盘    最高    最低    收盘    成交量    成交额
    price = np.loadtxt('sh_600000.txt', delimiter='\t', skiprows=2, usecols=(4,))
    print "原始价格：\n", price
    n = 100         # 阶数
    y = price[n:]
    m = len(y)      # 样本个数
    print "预测价格：\n", y
    x = np.zeros((m, n+1))
    for i in range(m):
        x[i] = np.hstack((price[i:i+n], 1))
    print "自变量：\n", x
    theta = np.linalg.lstsq(x, y)[0]     # theta为回归系数
    print theta
    show(theta, x, y)

    # 预测
    pn = 20     # 预测未来多少天
    x = price[-n:]
    y = np.hstack((price[-n:], np.zeros(pn)))
    for i in range(pn):
        y[n+i] = np.dot(theta, np.hstack((y[i:n+i], 1)))
    show_predict(y, pn)
```



线性回归

- y=ax+b

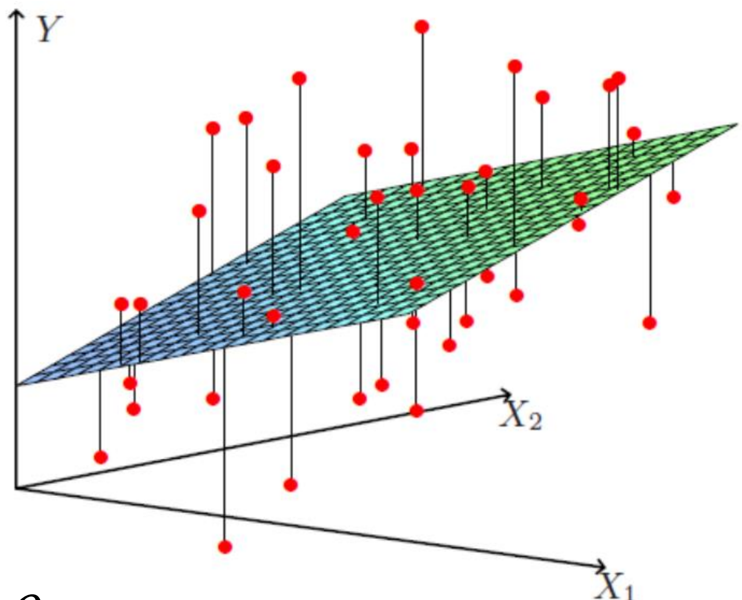


housing prices
price (in $1000)
square feet
0
100
200
300
400
500
600
700
800
900
1000
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
housing prices
price (in $1000)
square feet

多个变量的情形

- 考虑两个变量

Living area (feet^2)	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

使用极大似然估计解释最小二乘

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

the $\epsilon^{(i)}$ are distributed IID (independently and identically distributed) according to a Gaussian distribution (also called a Normal distribution) with mean zero and some variance σ^2

- 误差$\varepsilon^{(i)}(1 \le i \le m)$是独立同分布的，服从均值为0，方差为某定值$\sigma^2$的高斯分布。
  - 原因：中心极限定理

中心极限定理的意义

- 实际问题中，很多随机现象可以看做众多因素的独立影响的综合反应，往往近似服从正态分布。
  - 城市耗电量：大量用户的耗电量总和
  - 测量误差：许多观察不到的、微小误差的总和

  - 注：应用前提是多个随机变量的和，有些问题是乘性误差，则需要鉴别或者取对数后再使用。

似然函数 $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

高斯的对数似然与最小二乘

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)}-\theta^T x^{(i)})^2
\end{aligned}
$$

$$
J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2
$$

话题：聊聊“假设”

- 机器学习中的建模过程，往往充斥着假设，合理的假设是合理模型的必要前提。
- 假设具有三个性质：
  - 内涵性
  - 简化性
  - 发散性

假设的内涵性

- 所谓假设，就是根据常理应该是正确的。
  - 如假定一个人的身高位于区间[150cm,220cm]，这能够使得大多数情况都是对的，但很显然有些篮球运动员已经不属于这个区间。所以，假设的第一个性质：假设往往是正确的但不一定总是正确。
  - 我们可以称之为“假设的内涵性”。

假设的简化性

- 假设只是接近真实，往往需要做若干简化。
  - 如，在自然语言处理中，往往使用词袋模型(Bag Of Words)，认为一篇文档的词是独立的——这样的好处是计算该文档的似然概率非常简洁，只需要每个词出现概率乘积即可。
  - 但我们知道这个假设是错的：一个文档前一个词是“正态”，则下一个词极有可能是“分布”，文档的词并非真的独立。
  - 这个现象可以称之为“假设的简化性”。

假设的发散性

- 在某个简化的假设下推导得到的结论，不一定只有在假设成立时结论才成立。
  - 如，我们假定文本中的词是独立的，通过朴素贝叶斯做分类(如垃圾邮件的判定)。
  - 我们发现：即使使用这样明显不正确的假设，但它的分类效果往往在实践中是堪用的。
  - 这个现象可以称之为“假设的发散性”。

θ的解析式的求解过程

- 将M个N维样本组成矩阵X：
  - X的每一行对应一个样本，共M个样本(measurements)
  - X的每一列对应样本的一个维度，共N维(regressors)
    - 还有额外的一维常数项，全为1
- 目标函数 $J(\theta)=\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2=\frac{1}{2}(X\theta-y)^T(X\theta-y)$
- 梯度：$\nabla_\theta J(\theta)=\nabla_\theta\left(\frac{1}{2}(X\theta-y)^T(X\theta-y)\right)=\nabla_\theta\left(\frac{1}{2}\left(\theta^T X^T-y^T\right)(X\theta-y)\right)$

$$=\nabla_\theta\left(\frac{1}{2}\left(\theta^T X^T X\theta-\theta^T X^T y-y^T X\theta+y^T y\right)\right)$$

$$=\frac{1}{2}\left(2X^T X\theta-X^T y-\left(y^T X\right)^T\right)=X^T X\theta-X^T y\xrightarrow{\text{求驻点}}0$$

最小二乘意义下的参数最优解

- 参数的解析式

$$\theta = \left(X^T X\right)^{-1} X^T y$$

- 若X^TX不可逆或防止过拟合，增加λ扰动

$$\theta = \left(X^T X + \lambda I\right)^{-1} X^T y$$

- “简便”方法记忆结论

$$X\theta = y \Rightarrow X^T X\theta = X^T y$$

$$\Rightarrow \theta = \left(X^T X\right)^{-1} X^T y$$

加入λ扰动后

- X^TX半正定：对于任意的非零向量u

$$u^TX^TXu=(Xu)^TXu\xrightarrow{令v=Xu}v^Tv\geq 0$$

- 对于任意的实数λ>0，$X^TX+\lambda I$正定，

$$u^T\cdot\left(X^TX+\lambda I\right)\cdot u=u^TX^TXu+\lambda u^Tu$$

$$\xrightarrow{u\neq 0,\lambda>0}u^TX^TXu+\lambda u^Tu>0$$

- 从而$X^TX+\lambda I$可逆，保证回归公式一定有意义。 $$\theta=\left(X^TX+\lambda I\right)^{-1}X^Ty$$

线性回归的复杂度惩罚因子

- 线性回归的目标函数为： $J(\vec{\theta})=\frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right)-y^{(i)}\right)^2$
- 将目标函数增加平方和损失：

$$J(\vec{\theta})=\frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right)-y^{(i)}\right)^2+\lambda\sum_{j=1}^{n}\theta_j^2$$

- 本质即为假定参数θ服从高斯分布。
  - Ridge：Hoerl, Kennard,1970
- LASSO：Tibshirani,1996
  - Least Absolute Shrinkage and Selection Operator
  - LARS算法解决Lasso计算，Barsley Efron, 2004
    - Least Angle Regression

正则项与防止过拟合 $\begin{cases}\lambda > 0 \\ \rho \in [0,1]\end{cases}$

- L2-norm：

$$J(\vec{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

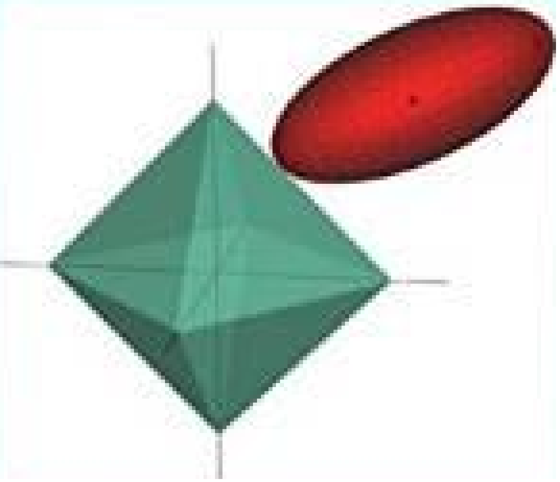
- L1-norm：

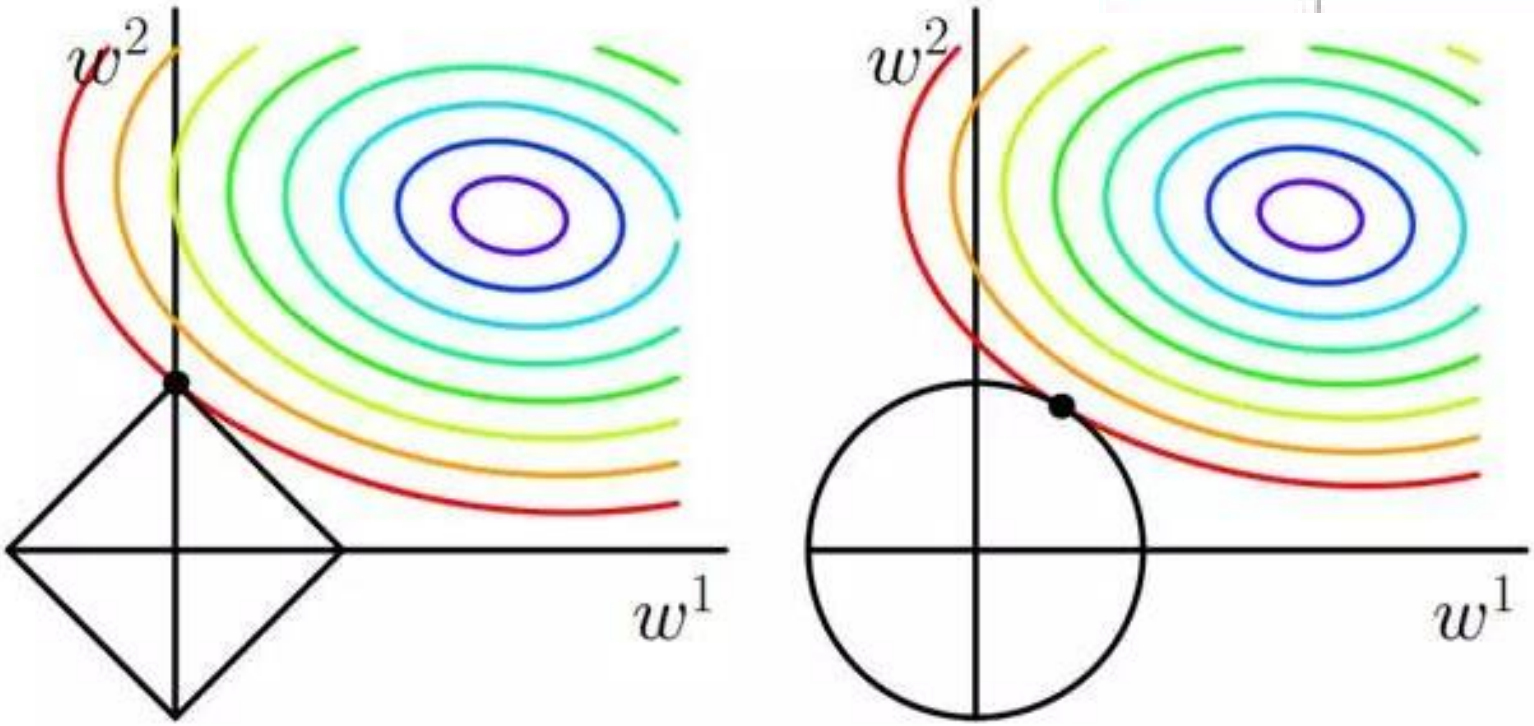
$$J(\vec{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\left|\theta_j\right|$$

- Elastic Net：

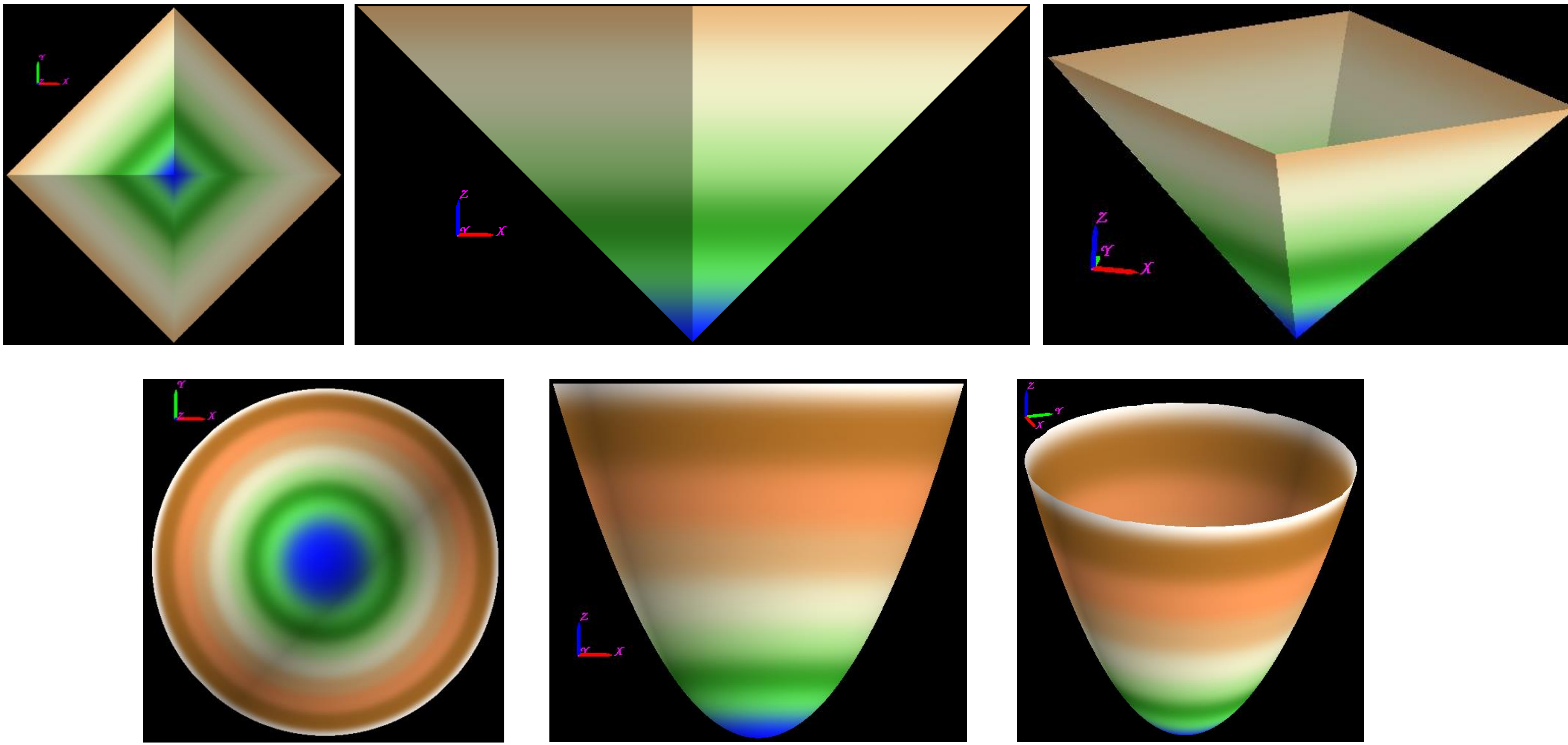
$$J(\vec{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\left(\rho\cdot\sum_{j=1}^{n}\left|\theta_j\right| + (1-\rho)\cdot\sum_{j=1}^{n}\theta_j^2\right)$$

正则化与稀疏





$|w|$与$|w|^2$



L1-norm如何处理梯度？

- 目标函数： $J(\vec{\theta}) = \frac{1}{2}\sum_{i=1}^{m}\left(h_{\vec{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\left|\theta_j\right|$
- 给定： $f(x;\alpha) = x + \frac{1}{\alpha}\log\left(1+\exp(-\alpha x)\right),\ x \geq 0$
- 近似： $|x| \approx f(x;\alpha) + f(-x;\alpha) = \frac{1}{\alpha}\log\left(1+\exp(-\alpha x) + 1 + \exp(\alpha x)\right)$
- 梯度： $\nabla|x| \approx \frac{1}{1+\exp(-\alpha x)} - \frac{1}{1+\exp(\alpha x)}$
- 二阶导： $\nabla^2|x| \approx \frac{2\alpha\exp(\alpha x)}{\left(1+\exp(\alpha x)\right)^2}$
- 实践中，对于一般问题，如取： $\alpha = 10^6$

机器学习与数据使用

训练数据 → θ

训练数据 → θ | 测试数据

训练数据 → θ | 验证数据 → λ | 测试数据

- 交叉验证
  - 如：十折交叉验证

Moore-Penrose广义逆矩阵(伪逆)

- 若A为非奇异矩阵，则线性方程组Ax=b的解为 $x=(A^TA)^{-1}A^T\cdot b$ ，从方程解的直观意义上，可以定义： $A^+=(A^TA)^{-1}A^T$
- 若A为可逆方阵，$A^+=(A^TA)^{-1}A^T$ 即为 A^{-1}

$$\left(A^TA\right)^{-1}A^T=A^{-1}\left(A^T\right)^{-1}A^T=A^{-1}$$

- 当A为矩阵(非方阵)时，称A^+称为A的广义逆(伪逆)。
  - 奇异值分解SVD

SVD计算矩阵的广义逆

- 对于m×n的矩阵A，若它的SVD分解为：

$$A = U \cdot \Sigma \cdot V^T$$

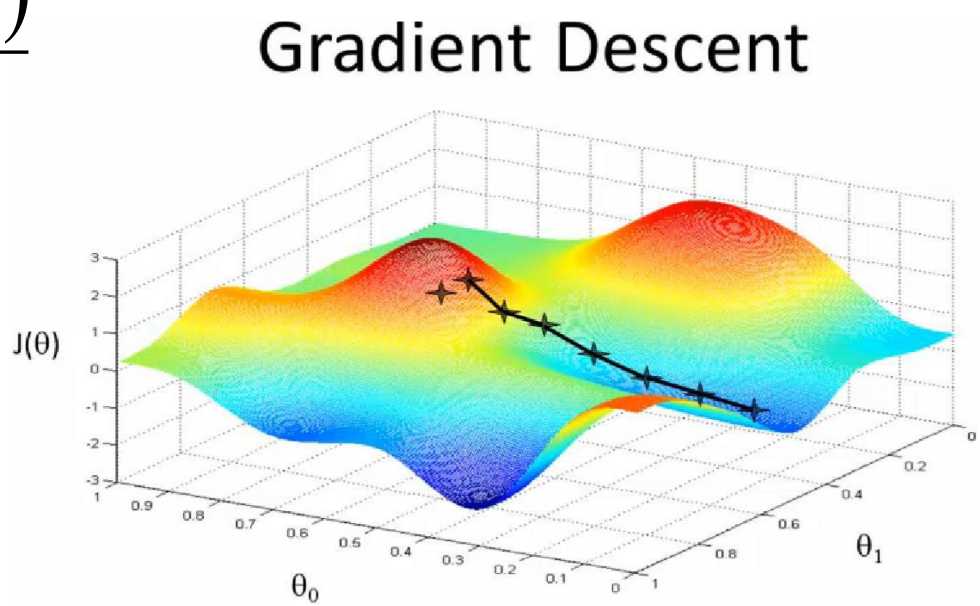
- 则，A的广义逆为：$A^+ = V \cdot \Sigma^{-1} \cdot U^T$
  - 可以验证，若A是n×n的可逆阵，则 $A \cdot A^+ = I$
  - 若A是不可逆阵或m≠n，则 $A^+ = \left(A^T \cdot A\right)^{-1} A$

梯度下降算法 $J(\theta)=\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$

- 初始化θ(随机初始化)
- 沿着负梯度方向迭代，更新后的θ使J(θ)更小

$$\theta=\theta-\alpha\cdot\frac{\partial J(\theta)}{\partial\theta}$$

  - α：学习率、步长



梯度方向

$$
\begin{aligned}
\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}\left(h_\theta(x)-y\right)^2 \\
&= 2\cdot\frac{1}{2}\left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(h_\theta(x)-y\right) \\
&= \left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i-y\right) \\
&= \left(h_\theta(x)-y\right)x_j
\end{aligned}
$$

批量梯度下降算法

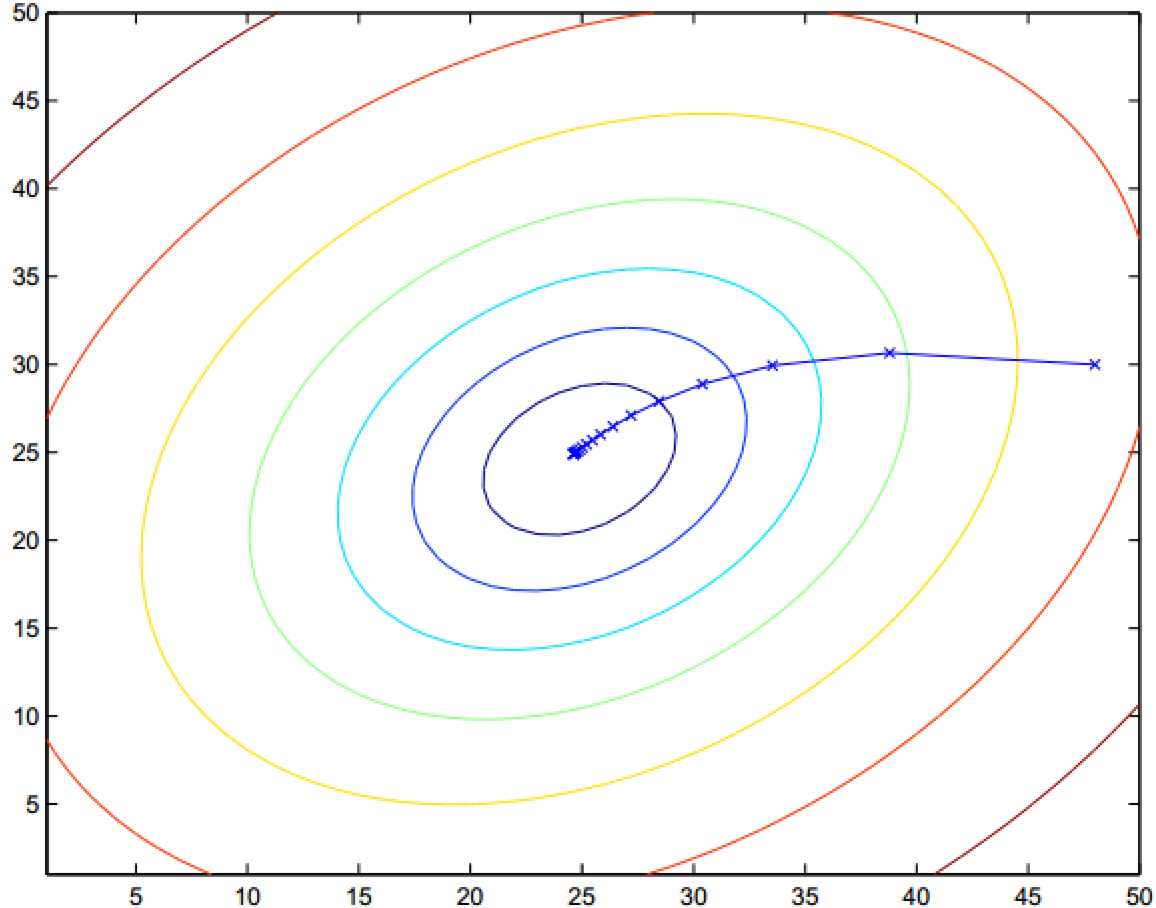
Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

gradient descent. Note that, while gradient descent can be susceptible to local minima in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate α is not too large) to the global minimum. Indeed, J is a convex quadratic function.

批量梯度下降图示



随机梯度下降算法

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

}

This algorithm is called **stochastic gradient descent** (also **incremental gradient descent**). Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent. (Note however that it may never "converge" to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$; but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.[2]) For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

折中：mini-batch

- 如果不是每拿到一个样本即更改梯度，而是若干个样本的平均梯度作为更新方向，则是mini-batch梯度下降算法。

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

}

}

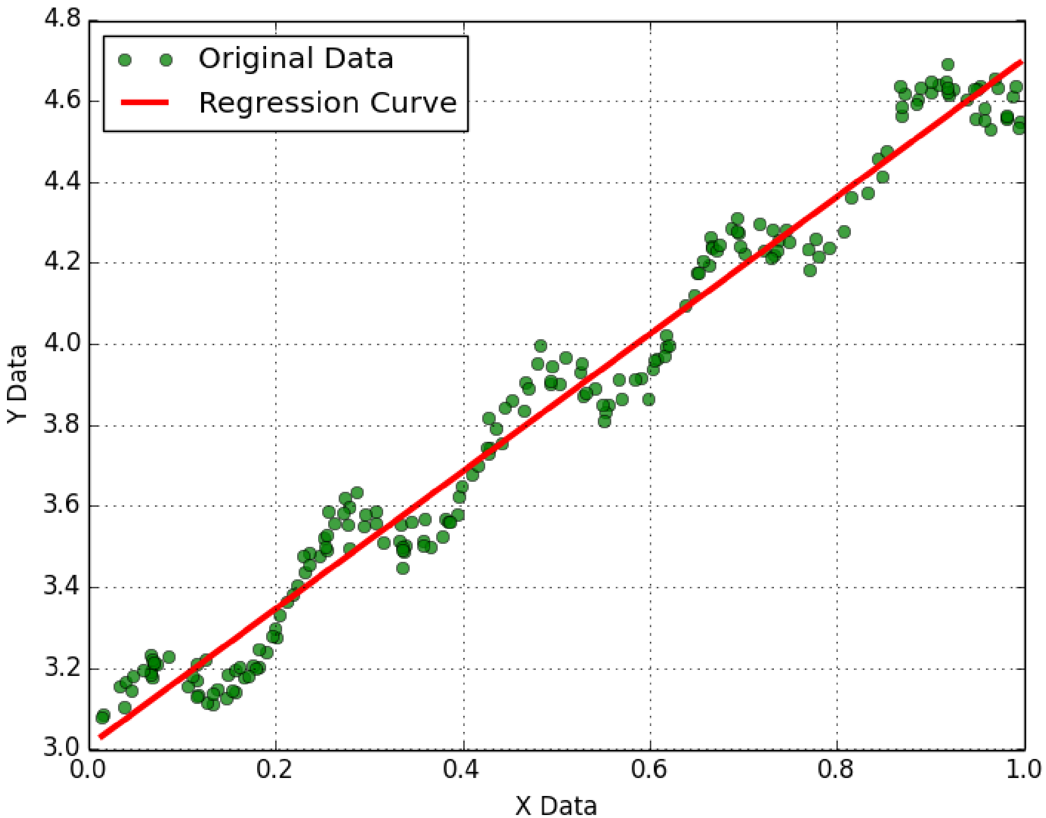
回归Code

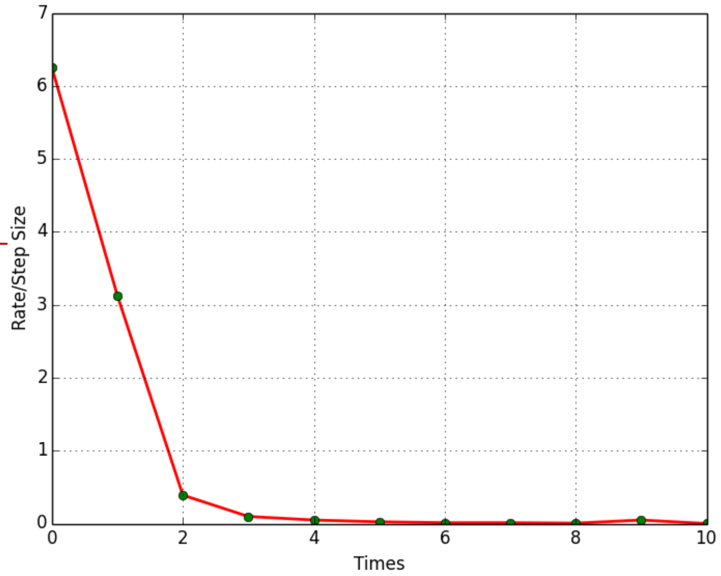
```python
def calcCoefficient(data, listA, listW, listLostFunction):
    N = len(data[0])  # 维度
    w = [0 for i in range(N)]
    wNew = [0 for i in range(N)]
    g = [0 for i in range(N)]

    times = 0
    alpha = 100.0  # 学习率随意初始化
    while times < 10000:
        j = 0
        while j < N:
            g[j] = gradient(data, w, j)
            j += 1
        normalize(g)  # 正则化梯度
        alpha = calcAlpha(w, g, alpha, data)
        numberProduct(alpha, g, wNew)

        print "times,alpha,fw,w,g:\t", times, alpha, fw(w, data), w, g
        if isSame(w, wNew):
            break
        assign2(w, wNew)  # 更新权值
        times += 1

        listA.append(alpha)
        listW.append(assign(w))
        listLostFunction.append(fw(w, data))
    return w
```

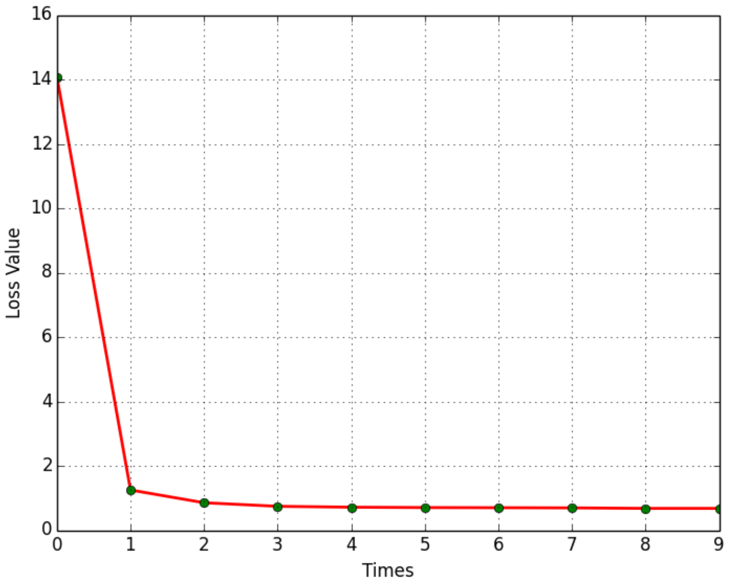
附：学习率Code

```python
# w当前值；g当前梯度方向；a当前学习率；data数据
def calcAlpha(w, g, a, data):
    c1 = 0.3
    now = fw(w, data)
    wNext = assign(w)
    numberProduct(a, g, wNext)
    next = fw(wNext, data)
    # 寻找足够大的a，使得h(a)>0
    count = 30
    while next < now:
        a *= 2
        wNext = assign(w)
        numberProduct(a, g, wNext)
        next = fw(wNext, data)
        count -= 1
        if count == 0:
            break

    # 寻找合适的学习率a
    count = 50
    while next > now - c1*a*dotProduct(g, g):
        a /= 2
        wNext = assign(w)
        numberProduct(a, g, wNext)
        next = fw(wNext, data)

        count -= 1
        if count == 0:
            break
    return a
```

线性回归、rate、Loss





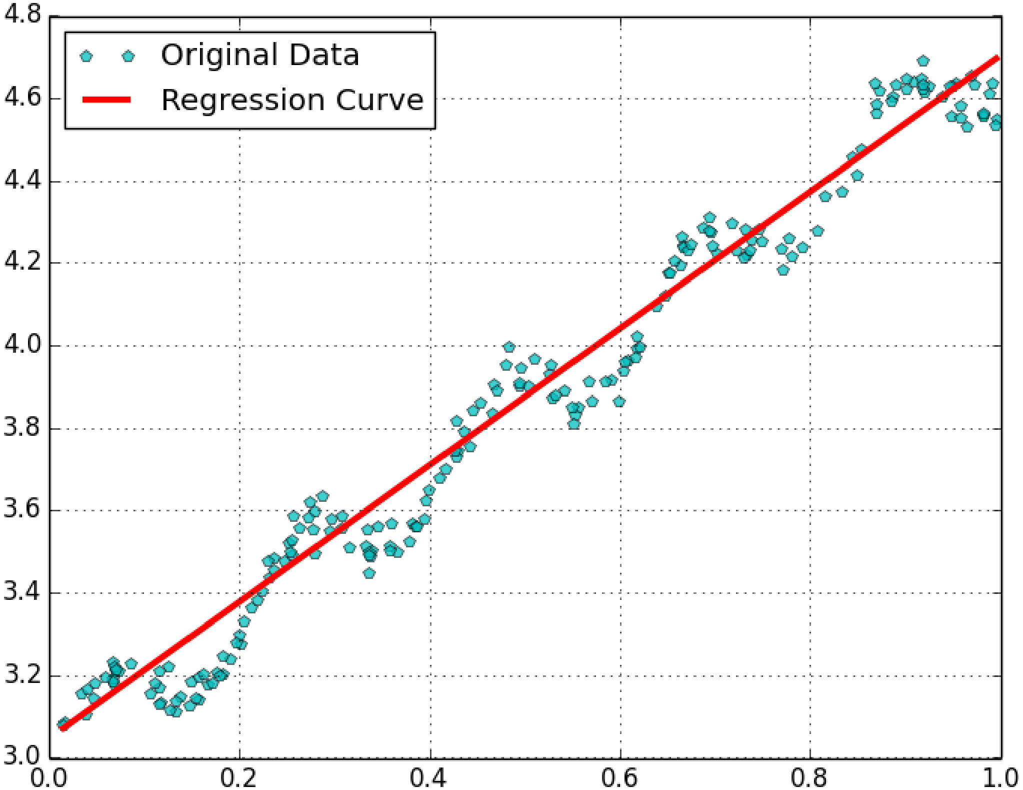


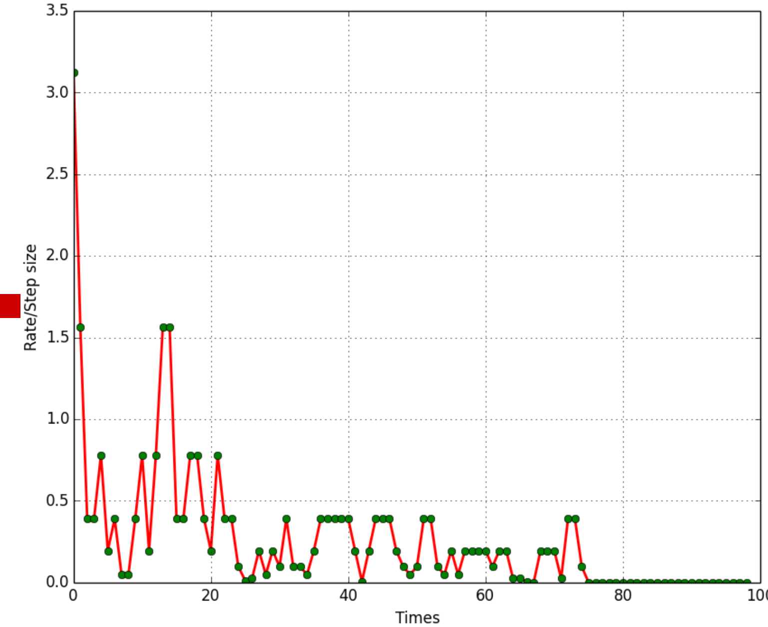
SGD与学习率

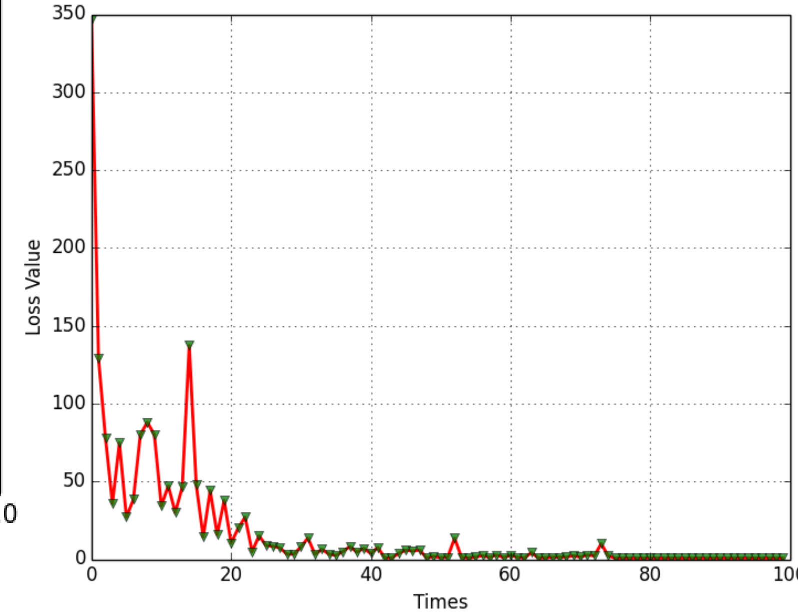
```python
# w当前值；g当前梯度方向；a当前学习率；data数据
def calcAlphaStochastic(w, g, a, data):
    c1 = 0.01   # 因为是每个样本都下降，所以参数运行度大些，即：激进
    now = fwStochastic(w, data)
    wNext = assign(w)
    numberProduct(a, g, wNext)
    next = fwStochastic(wNext, data)
    # 寻找足够大的a，使得h(a)>0
    count = 30
    while next < now:
        if a < 1e-10:
            a = 0.01
        else:
            a *= 2
        wNext = assign(w)
        numberProduct(a, g, wNext)
        next = fwStochastic(wNext, data)
        count -= 1
        if count == 0:
            break

    # 寻找合适的学习率a
    count = 50
    while next > now - c1*a*dotProduct(g, g):
        a /= 2
        wNext = assign(w)
        numberProduct(a, g, wNext)
        next = fwStochastic(wNext, data)

        count -= 1
        if count == 0:
            break
    return a
```

```python
def calcCoefficient(data, listA, listW, listLostFunction):
    M = len(data)        # 样本数目
    N = len(data[0])     # 维度
    w = [0 for i in range(N)]
    wNew = [0 for i in range(N)]
    g = [0 for i in range(N)]

    times = 0
    alpha = 100.0  # 学习率随意初始化
    same = False
    while times < 10000:
        i = 0
        while i < M:
            j = 0
            while j < N:
                g[j] = gradientStochastic(data[i], w, j)
                j += 1
            normalize(g)  # 正则化梯度
            alpha = calcAlphaStochastic(w, g, alpha, data[i])
            #alpha = 0.01
            numberProduct(alpha, g, wNew)

            print "times,i, alpha,fw,w,g:\t", times, i, alpha, fw(w, data), w, g
            if isSame(w, wNew):
                if times > 5:  #防止训练次数过少
                    same = True
                    break
            assign2(w, wNew)  # 更新权值

            listA.append(alpha)
            listW.append(assign(w))
            listLostFunction.append(fw(w, data))

            i += 1
        if same:
            break
        times += 1
    return w
```

随机梯度下降SGD


Original Data
Regression Curve
4.8
4.6
4.4
4.2
4.0
3.8
3.6
3.4
3.2
3.0
0.0
0.2
0.4
0.6
0.8
1.0


Rate/Step size
Times
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100


Loss Value
Times
350
300
250
200
150
100
50
0
0
20
40
60
80
100

批量与随机梯度下降

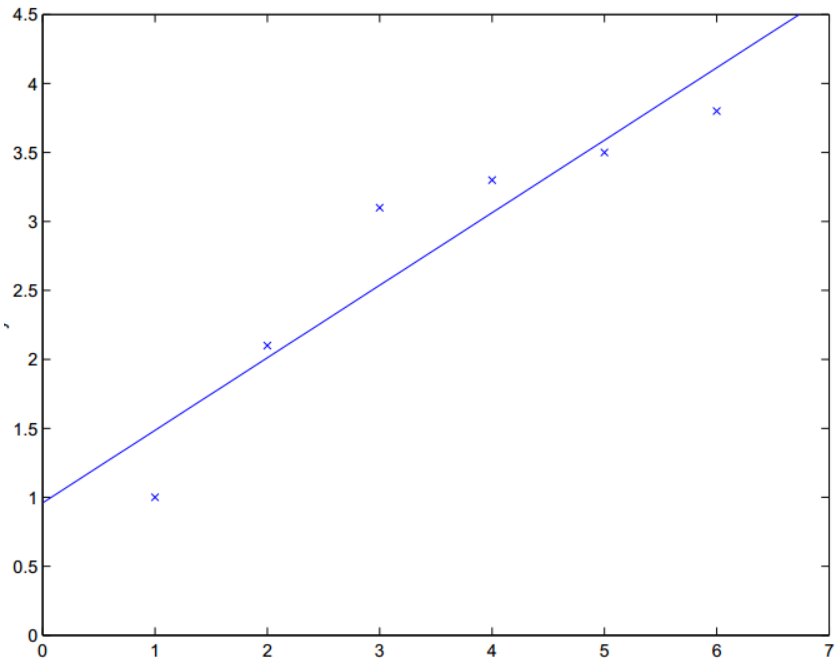
```
times, alpha, fw, w, g: 1 3.125 984.612994627 [2.9098051520832042, 5.531322986131802] [-0.462463597293110 3, -0.886638
times, alpha, fw, w, g: 2 0.390625 14.0797532659 [1.464606410542234 5, 2.7605783935270636] [0.4772346439139256 7, 0.878
times, alpha, fw, w, g: 3 0.09765625 1.24904918001 [1.6510261933211117, 3.1038502321821975] [-0.4008445229943951 6, -0
times, alpha, fw, w, g: 4 0.048828125 0.85339951 [1.6118812203724402, 3.0143828400389685] [0.5719292366989982, 0.8203
times, alpha, fw, w, g: 5 0.0244140625 0.742294709799 [1.6398074526331334, 3.0544366955679614] [-0.23913106662126155,
times, alpha, fw, w, g: 6 0.01220703125 0.714627298341 [1.6339692918269504, 3.030730950986636] [0.7783350337309733, 0
times, alpha, fw, w, g: 7 0.01220703125 0.703424432171 [1.6434704519066743, 3.03839512537648] [0.3237021548469936, -0
times, alpha, fw, w, g: 8 0.006103515625 0.699055652793 [1.647421894226584, 3.0268453324982114] [0.821739362051493 9,
times, alpha, fw, w, g: 9 0.048828125 0.693596841711 [1.6524373932625427, 3.0303235033400355] [0.8678067307461911, -0
times, alpha, fw, w, g: 10 0.000762939453125 0.678000227512 [1.6948107687872591, 3.0060607162442174] [0.4549874445298
times, alpha, fw, w, g: 11 1.73472347598e-16 0.677742186906 [1.6951578966593674, 3.0067401121888184] [0.4481438000238
```

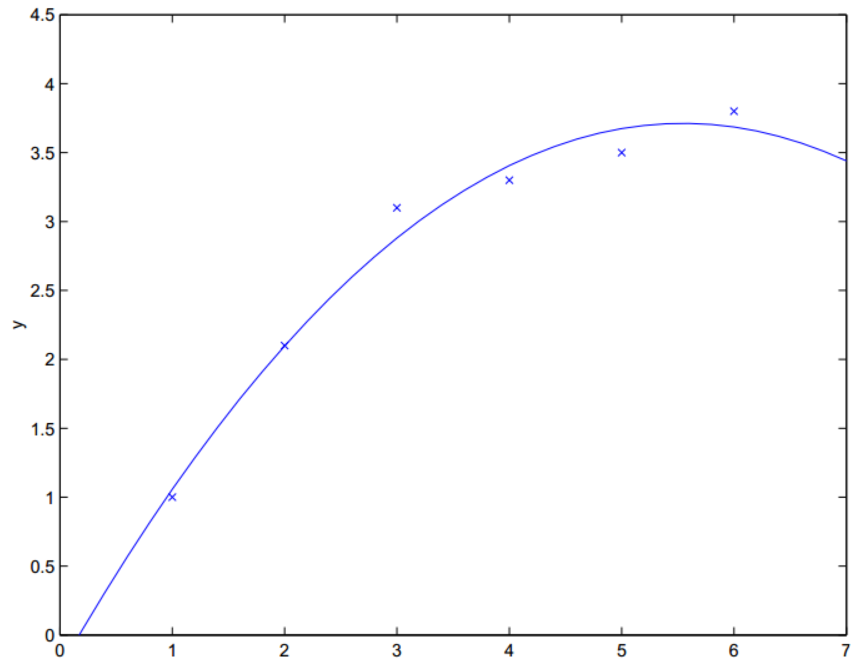
SGD SGD

```
times, i, alph                                                                                     7134124439004281]
times, i, alpha, fw, w, g:                                                                          -0.7881658308341754]
times, i, alpha, fw, w, g:  5 189 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.5464975452139291, 0.8374607053916915]
times, i, alpha, fw, w, g:  5 190 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.6563783605915454, 0.7544318708453104]
times, i, alpha, fw, w, g:  5 191 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [-0.5174844022485295, -0.8556926395788865]
times, i, alpha, fw, w, g:  5 192 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.2472857765429518, 0.9689425910339319]
times, i, alpha, fw, w, g:  5 193 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [-0.5898990792303563, -0.8074769819153842]
times, i, alpha, fw, w, g:  5 194 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.4427816384461959, 0.8966294779087415]
times, i, alpha, fw, w, g:  5 195 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.24892670275384426, 0.968522326359129]
times, i, alpha, fw, w, g:  5 196 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [-0.6403664519380511, -0.7680695328108464]
times, i, alpha, fw, w, g:  5 197 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.0699234327658517, 0.9975523613075352]
times, i, alpha, fw, w, g:  5 198 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.46626882838072714, 0.8846430803891838]
times, i, alpha, fw, w, g:  5 199 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [-0.11538710100614581, -0.9933206012770487]
times, i, alpha, fw, w, g:  6 0 5.29395592034e-21 0.730581114754 [1.6578292262575833, 3.0462889160238773] [0.06757716806139506, 0.997714050395604]
```

线性回归的进一步分析

- 可以对样本是非线性的，只要对参数θ线性



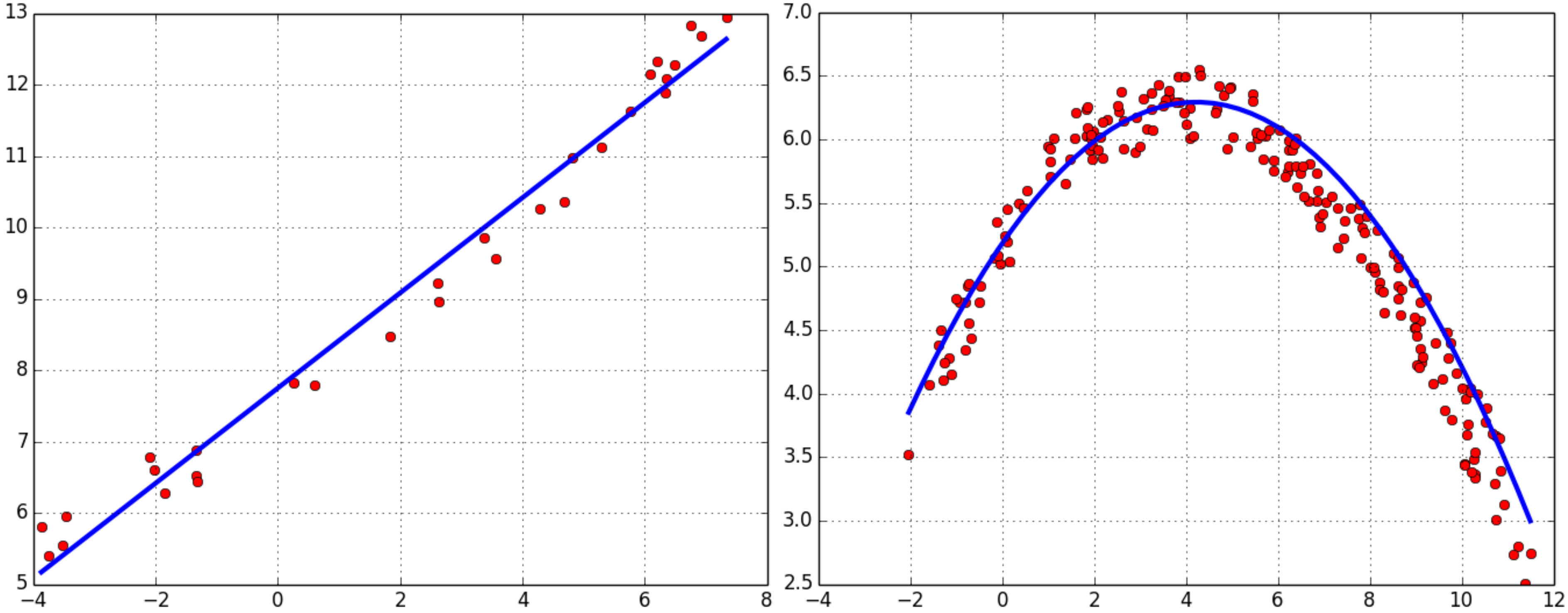


$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

Code

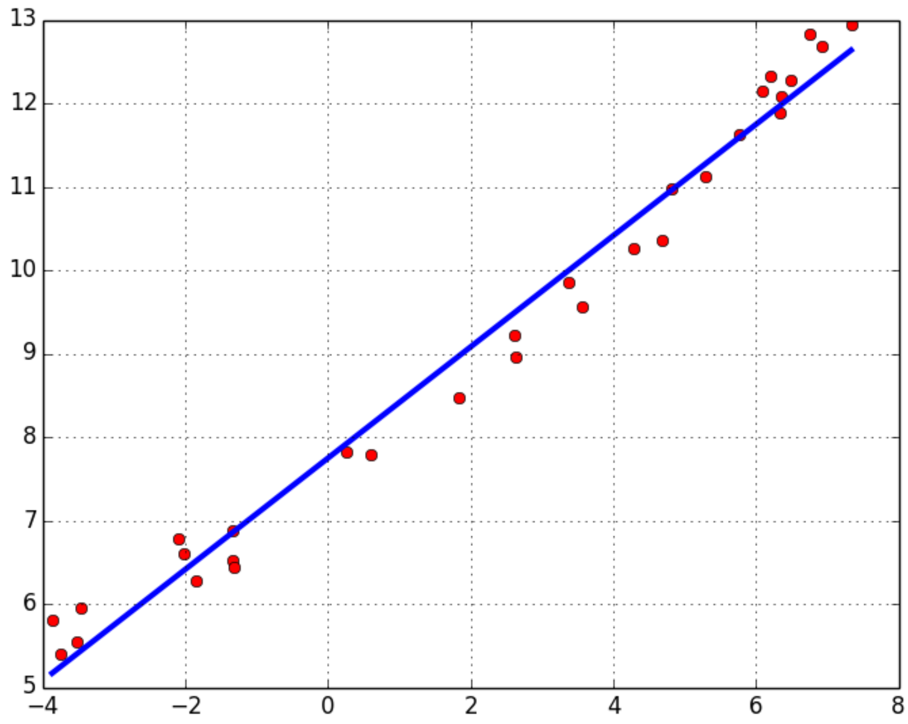
```
def regression(data, alpha, lamda):
    n = len(data[0]) - 1
    theta = np.zeros(n)
    for times in range(100):
        for d in data:
            x = d[:-1]
            y = d[-1]
            g = np.dot(theta, x) - y
            theta = theta - alpha * g * x + lamda * theta
        print times, theta
    return theta
```

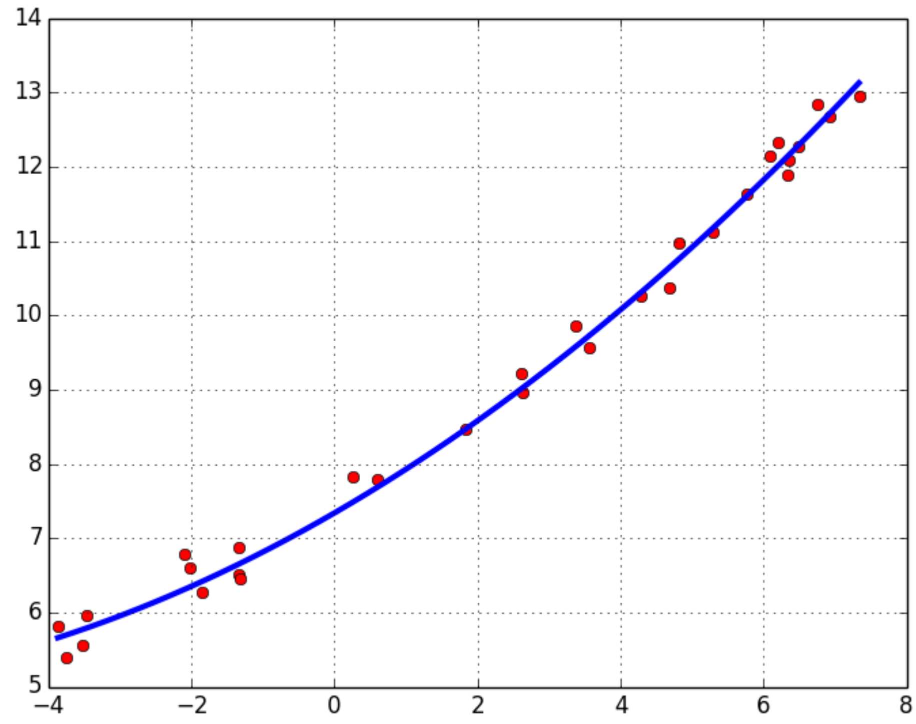
线性回归



13
12
11
10
9
8
7
6
5
−4
−2
0
2
4
6
8
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
−4
−2
0
2
4
6
8
10
12

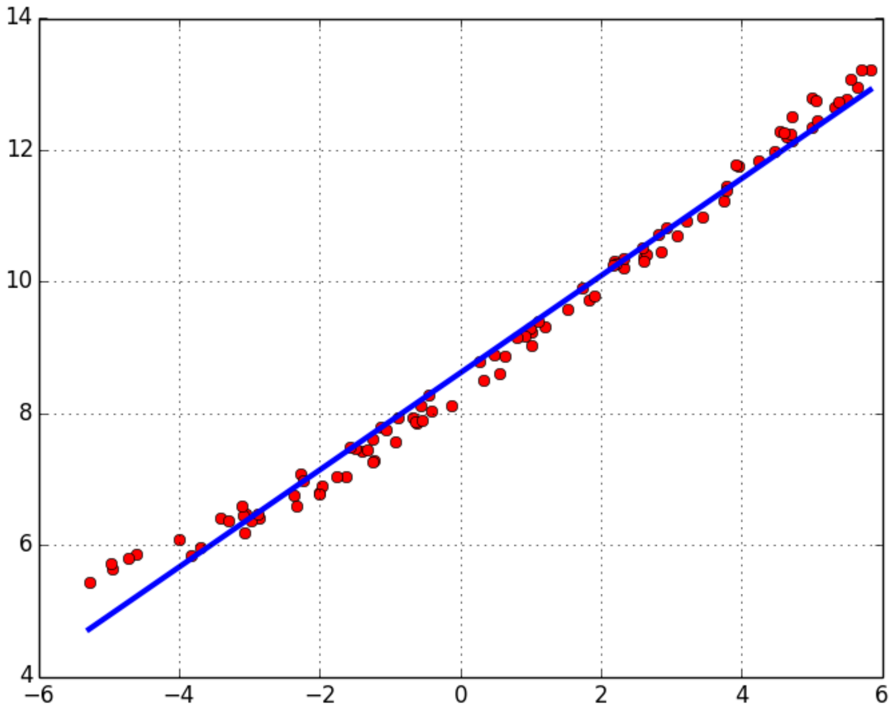
线性回归



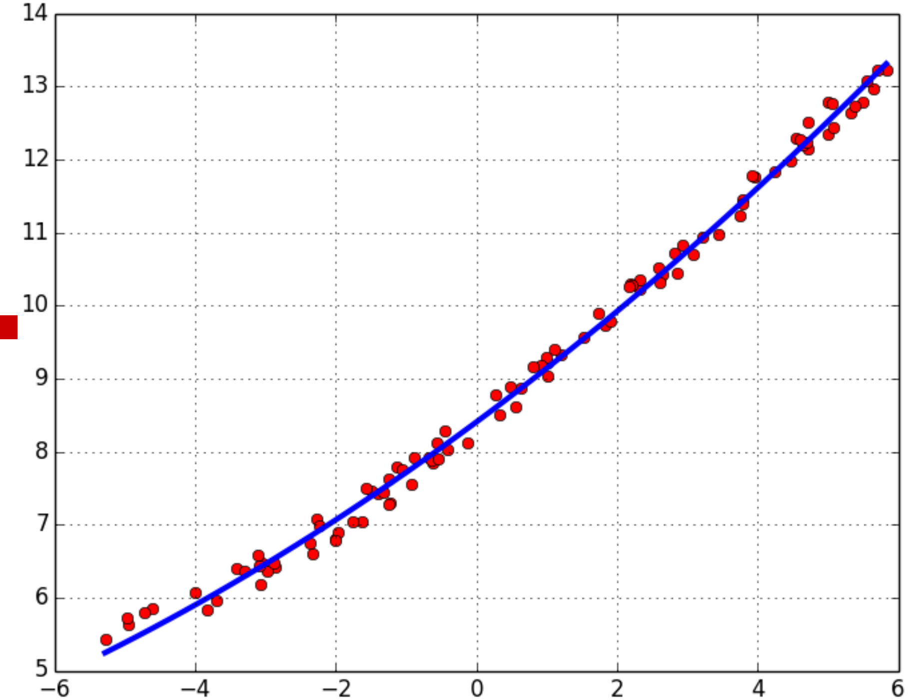


特征选择

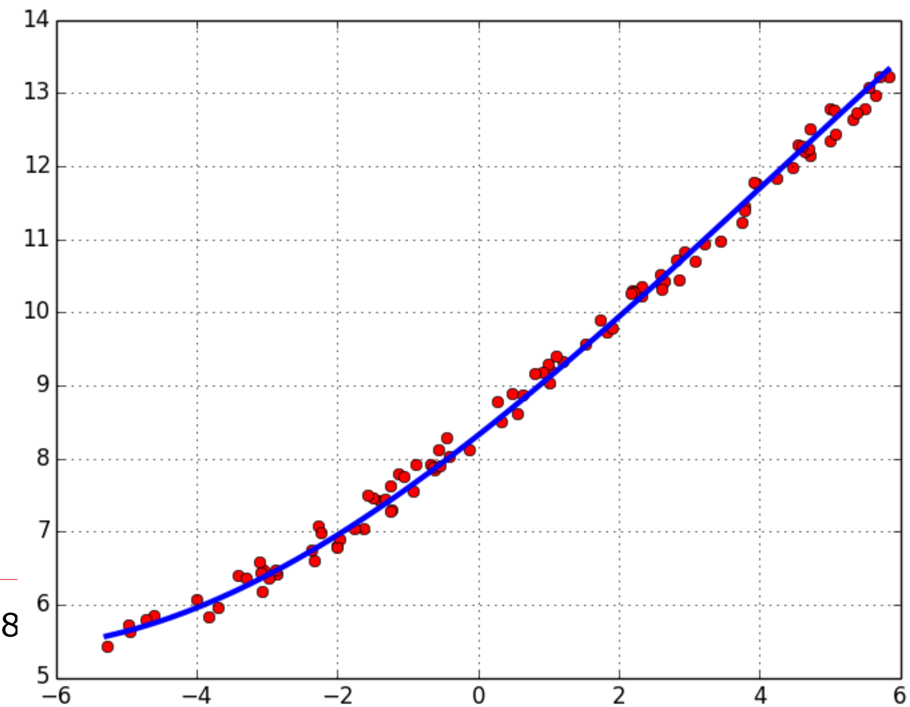
1 2 3



14
12
10
8
6
4
−6
−4
−2
0
2
4
6

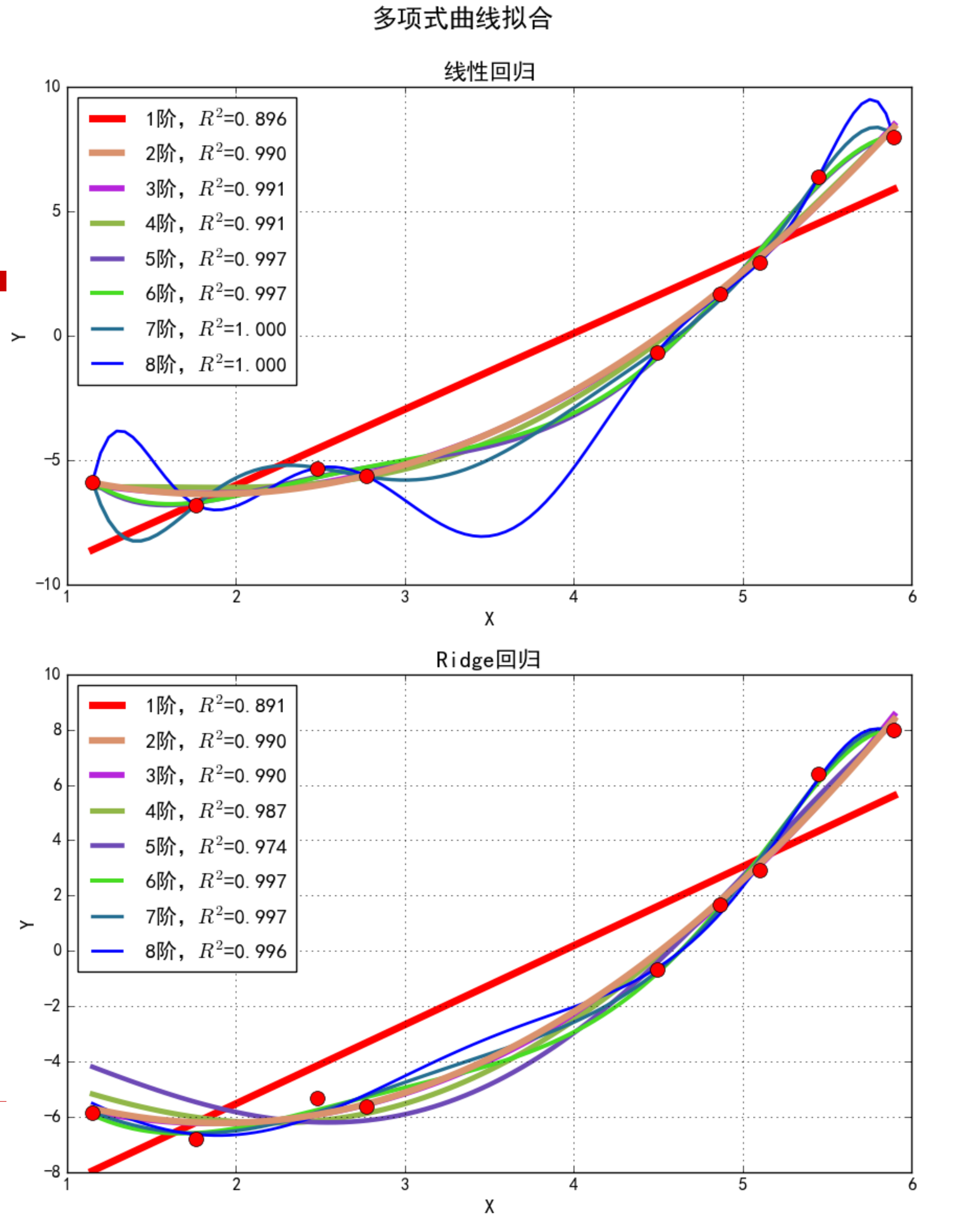


14
13
12
11
10
9
8
7
6
5
−6
−4
−2
0
2
4
6

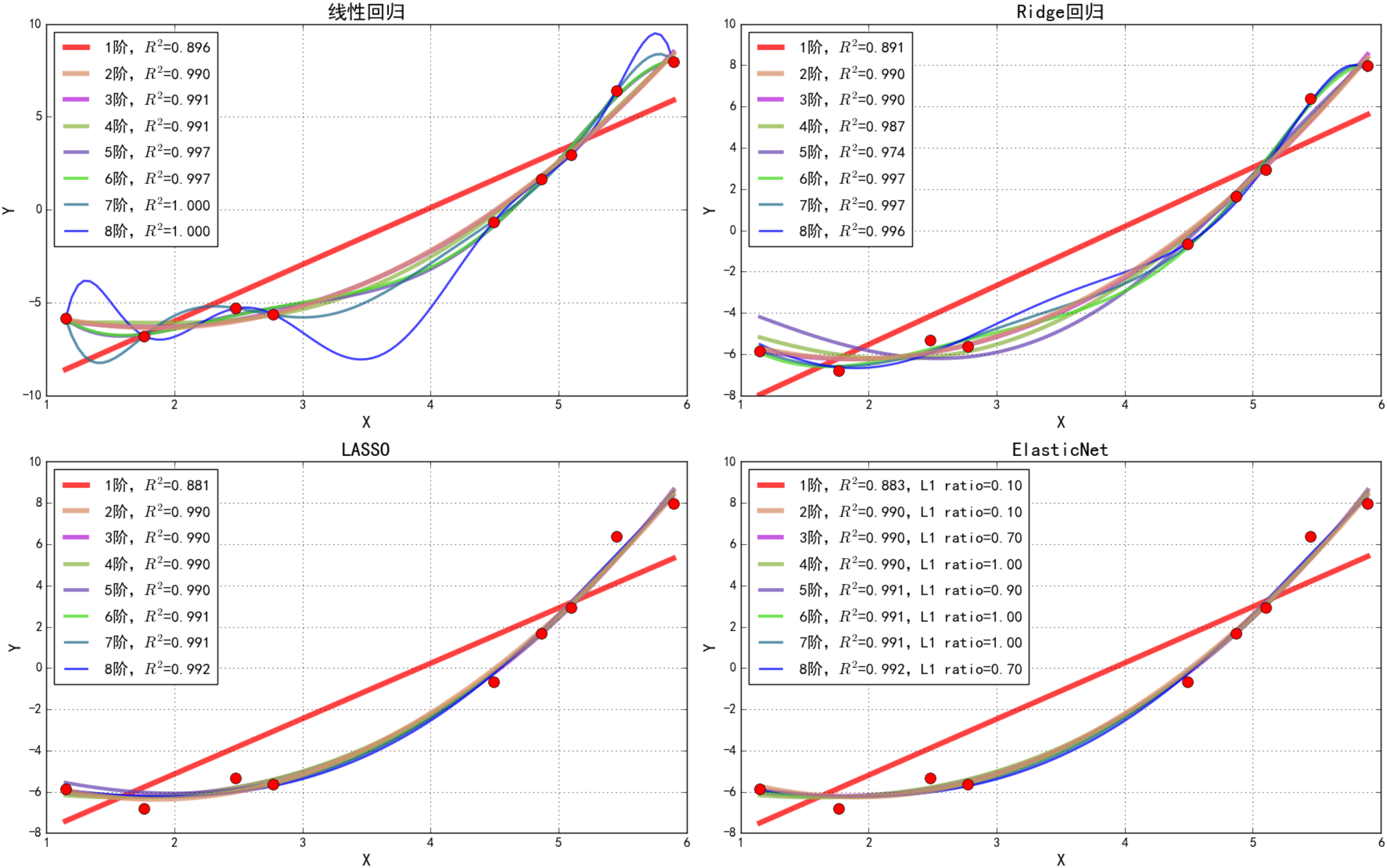


14
13
12
11
10
9
8
7
6
5
−6
−4
−2
0
2
4
6

超参与过拟合


多项式曲线拟合
线性回归
1阶，R^2=0.896
2阶，R^2=0.990
3阶，R^2=0.991
4阶，R^2=0.991
5阶，R^2=0.997
6阶，R^2=0.997
7阶，R^2=1.000
8阶，R^2=1.000
X
Y
Ridge回归
1阶，R^2=0.891
2阶，R^2=0.990
3阶，R^2=0.990
4阶，R^2=0.987
5阶，R^2=0.974
6阶，R^2=0.997
7阶，R^2=0.997
8阶，R^2=0.996
X
Y

多项式曲线拟合比较



线性回归
1阶，R^2=0.896
2阶，R^2=0.990
3阶，R^2=0.991
4阶，R^2=0.991
5阶，R^2=0.997
6阶，R^2=0.997
7阶，R^2=1.000
8阶，R^2=1.000
Ridge回归
1阶，R^2=0.891
2阶，R^2=0.990
3阶，R^2=0.990
4阶，R^2=0.987
5阶，R^2=0.974
6阶，R^2=0.997
7阶，R^2=0.997
8阶，R^2=0.996
LASSO
1阶，R^2=0.881
2阶，R^2=0.990
3阶，R^2=0.990
4阶，R^2=0.990
5阶，R^2=0.990
6阶，R^2=0.991
7阶，R^2=0.991
8阶，R^2=0.992
ElasticNet
1阶，R^2=0.883，L1 ratio=0.10
2阶，R^2=0.990，L1 ratio=0.10
3阶，R^2=0.990，L1 ratio=0.70
4阶，R^2=0.990，L1 ratio=1.00
5阶，R^2=0.991，L1 ratio=0.90
6阶，R^2=0.991，L1 ratio=1.00
7阶，R^2=0.991，L1 ratio=1.00
8阶，R^2=0.992，L1 ratio=0.70
X
Y

高阶系数与过拟合

```
线性回归：1阶，系数为： [-12.12113792   3.05477422]
线性回归：2阶，系数为： [-3.23812184 -3.36390661  0.90493645]
线性回归：3阶，系数为： [-3.90207326 -2.61163034  0.66422328  0.02290431]
线性回归：4阶，系数为： [-8.20599769  4.20778207 -2.85304163  0.73902338 -0.05008557]
线性回归：5阶，系数为： [ 21.59733285 -54.12232017  38.43116219 -12.68651476   1.98134176  -0.11572371]
线性回归：6阶，系数为： [ 14.73304784 -37.87317493  23.67462341  -6.07037979   0.42536833   0.06803132  -0.00859246]
线性回归：7阶，系数为： [ 314.30344774 -827.89447318  857.3329359  -465.46543854  144.21883916  -25.67294689    2.44658613   -0.09675941]
线性回归：8阶，系数为： [-1189.50153111   3643.69120893 -4647.92955114   3217.22824043 -1325.87388079    334.32869997    -50.57119257      4.21251829     -0.14852101]
Ridge回归：1阶，alpha=0.109854，系数为： [-11.21592213   2.85121516]
Ridge回归：2阶，alpha=0.138950，系数为： [-2.90423989 -3.49931368  0.91803171]
Ridge回归：3阶，alpha=0.068665，系数为： [-3.47165245 -2.85078293  0.69245987  0.02314415]
Ridge回归：4阶，alpha=0.222300，系数为： [-2.84560266 -1.99887417 -0.40628792  0.33863868 -0.02674442]
Ridge回归：5阶，alpha=1.151395，系数为： [-1.68160373 -1.52726943 -0.8382036   0.2329258   0.03934251 -0.00663323]
Ridge回归：6阶，alpha=0.001000，系数为： [ 0.53724068 -6.00552086 -3.75961826  5.64559118 -2.21569695  0.36872913 -0.02221332]
Ridge回归：7阶，alpha=0.033932，系数为： [-2.38021238 -2.26383055 -1.47715232  0.00763115  1.12242917 -0.52769633  0.09199201 -0.00560201]
Ridge回归：8阶，alpha=0.138950，系数为： [-2.19299093 -1.91896884 -1.21608489 -0.19314178  0.49300277  0.05452898 -0.09690455  0.02114434 -0.00140202]
LASSO：1阶，alpha=0.222300，系数为： [-10.41556797   2.66199326]
LASSO：2阶，alpha=0.001000，系数为： [-3.29932625 -3.31989869  0.89878903]
LASSO：3阶，alpha=0.013257，系数为： [-4.83524033 -1.48721929  0.29726322  0.05804667]
LASSO：4阶，alpha=0.002560，系数为： [-5.08513199 -1.41147772  0.3380565   0.0440427   0.00099807]
LASSO：5阶，alpha=0.042919，系数为： [-4.11853758 -1.8643949   0.2618319   0.07954732  0.00257481 -0.00069093]
LASSO：6阶，alpha=0.001000，系数为： [-4.53546398 -1.70335188  0.29896515  0.05237738  0.00489432  0.00007551 -0.00010944]
LASSO：7阶，alpha=0.001000，系数为： [-4.51456835 -1.58477275  0.23483228  0.04900369  0.00593868  0.00044879 -0.00002625 -0.00002132]
LASSO：8阶，alpha=0.001000，系数为： [-4.62623251 -1.37717809  0.17183854  0.04307765  0.00629505  0.00069171  0.0000355  -0.00000875 -0.00000386]
ElasticNet：1阶，alpha=0.021210，l1_ratio=0.100000，系数为： [-10.74762959   2.74580662]
ElasticNet：2阶，alpha=0.013257，l1_ratio=0.100000，系数为： [-2.95099269 -3.48472703  0.91705013]
ElasticNet：3阶，alpha=0.013257，l1_ratio=1.000000，系数为： [-4.83524033 -1.48721929  0.29726322  0.05804667]
ElasticNet：4阶，alpha=0.010481，l1_ratio=0.950000，系数为： [-4.8799192  -1.5317438   0.3452403   0.04825571  0.00049763]
ElasticNet：5阶，alpha=0.004095，l1_ratio=0.100000，系数为： [-4.07916291 -2.18606287  0.44650232  0.05102669  0.00239164 -0.00048279]
ElasticNet：6阶，alpha=0.001000，l1_ratio=1.000000，系数为： [-4.53546398 -1.70335188  0.29896515  0.05237738  0.00489432  0.00007551 -0.00010944]
ElasticNet：7阶，alpha=0.001000，l1_ratio=1.000000，系数为： [-4.51456835 -1.58477275  0.23483228  0.04900369  0.00593868  0.00044879 -0.00002625 -0.00002132]
ElasticNet：8阶，alpha=0.001000，l1_ratio=0.500000，系数为： [-4.53761647 -1.45230301  0.18829714  0.0427561   0.00619739  0.00068209  0.00003506 -0.00000869 -0.00000384]
```

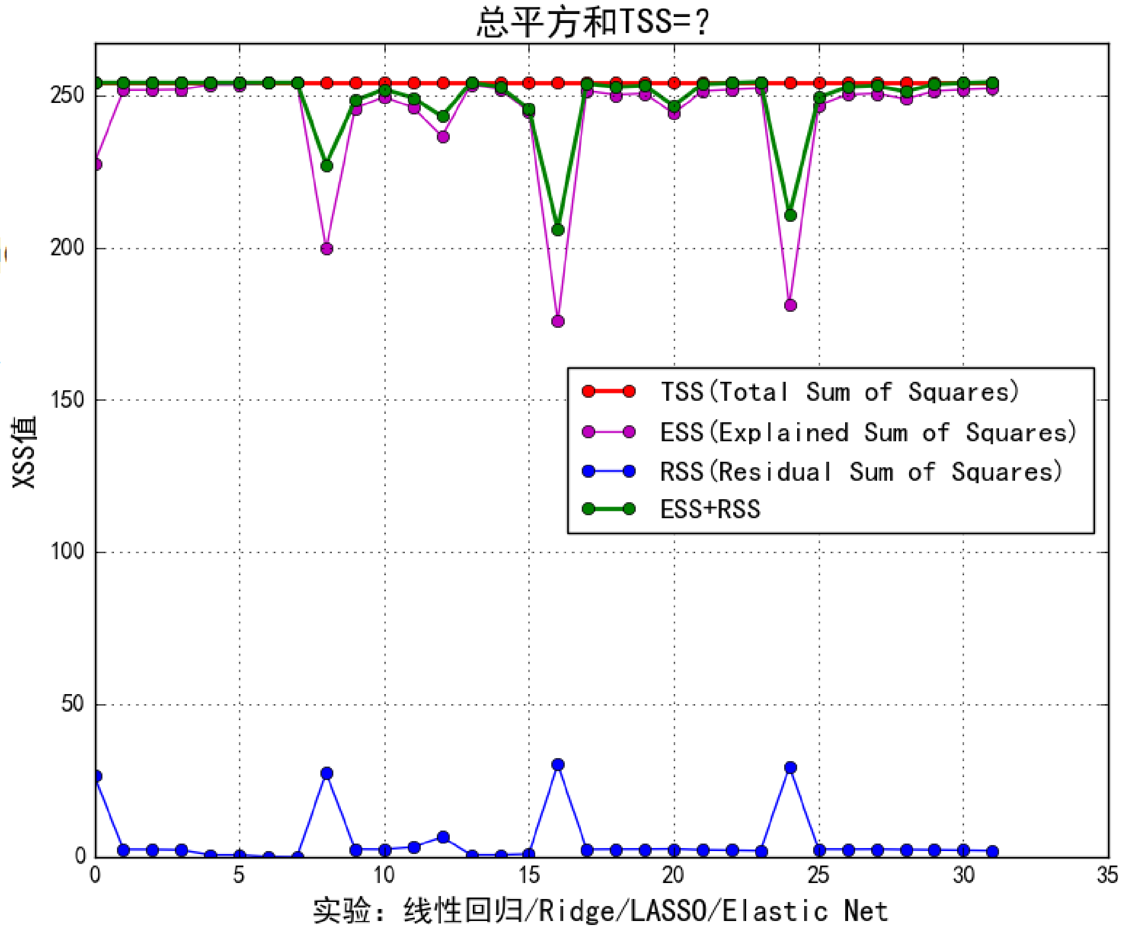
Coefficient of Determination

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$$

- 对于m个样本 $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \cdots, (\vec{x}_m, y_m)$
- 某模型的估计值为 $(\vec{x}_1, \hat{y}_1), (\vec{x}_2, \hat{y}_2), \cdots, (\vec{x}_m, \hat{y}_m)$
- 计算样本的总平方和TSS(Total Sum of Squares)：$TSS = \sum_{i=1}^{m}(y_i - \bar{y})^2$
  - 即样本伪方差的m倍 $Var(Y) = TSS / m$
- 计算残差平方和RSS(Residual Sum of Squares)：$RSS = \sum_{i=1}^{m}(\hat{y}_i - y_i)^2$
  - 注：RSS即误差平方和SSE(Sum of Squares for Error)
- 定义 $R^2 = 1 - RSS / TSS$
  - R^2越大，拟合效果越好
  - R^2的最优值为1；若模型预测为随机值，R^2有可能为负
  - 若预测值恒为样本期望，R^2为0
- 亦可定义ESS(Explained Sum of Squares)：$ESS = \sum_{i=1}^{m}(\hat{y}_i - \bar{y})^2$
  - TSS = ESS + RSS
    - 只有在无偏估计时上述等式才成立，否则，$TSS \ge ESS + RSS$
  - ESS又称回归平方和SSR(Sum of Squares for Regression)

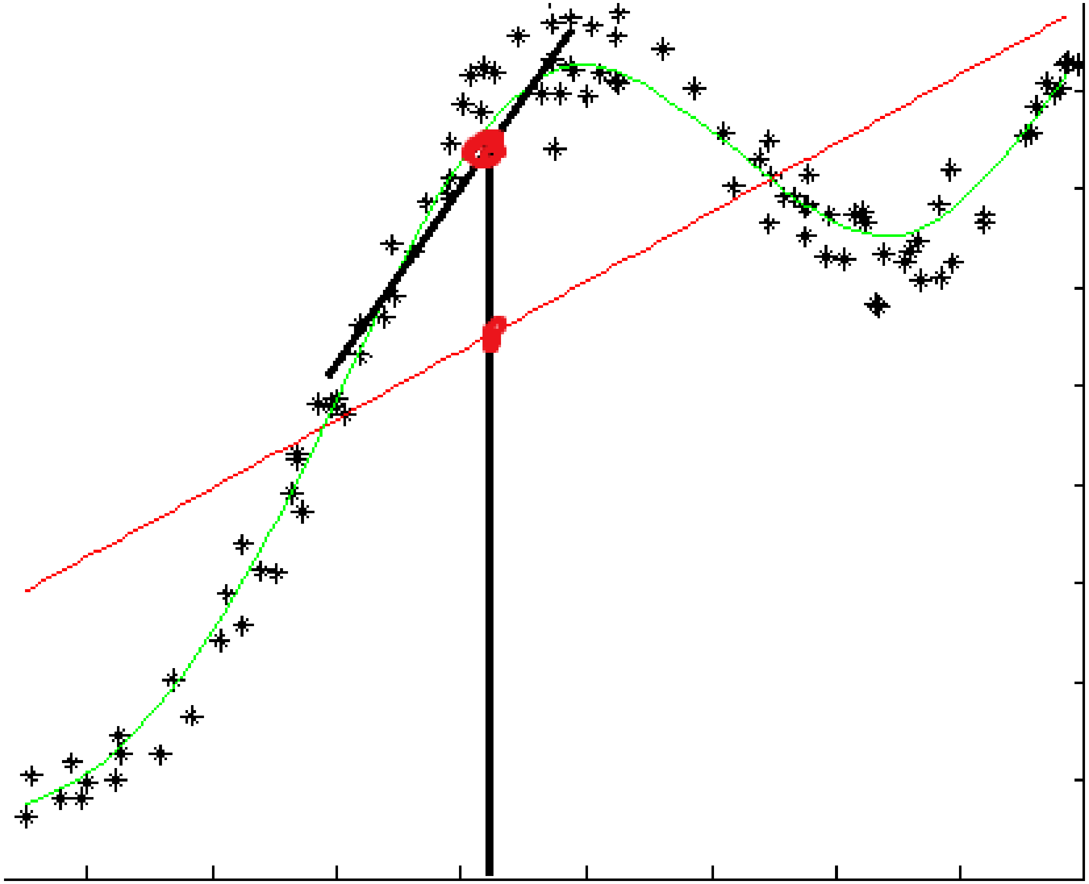
$TSS \geq ESS + RSS$

```python
def xss(y, y_hat):
    y = y.ravel()
    y_hat = y_hat.ravel()
    # Version 1
    tss = ((y - np.average(y)) ** 2).sum()
    rss = ((y_hat - y) ** 2).sum()
    ess = ((y_hat - np.average(y)) ** 2).sum
    r2 = 1 - rss / tss
    print 'RSS:', rss, '\t ESS:', ess
    print 'TSS:', tss, 'RSS + ESS = ', rss +
    tss_list.append(tss)
    rss_list.append(rss)
    ess_list.append(ess)
    ess_rss_list.append(rss + ess)
    # Version 2
    # tss = np.var(y)
    # rss = np.average((y_hat - y) ** 2)
    # r2 = 1 - rss / tss
    corr_coef = np.corrcoef(y, y_hat)[0, 1]
    return r2, corr_coef
```



局部加权回归

- 黑色是样本点
- 红色是线性回归曲线
- 绿色是局部加权回归曲线



局部加权线性回归

- LWR：Locally Weighted linear Regression

1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$
2. Output $\theta^T x$.

1. Fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$
2. Output $\theta^T x$.

权值的设置

- ω的一种可能的选择方式(高斯核函数):

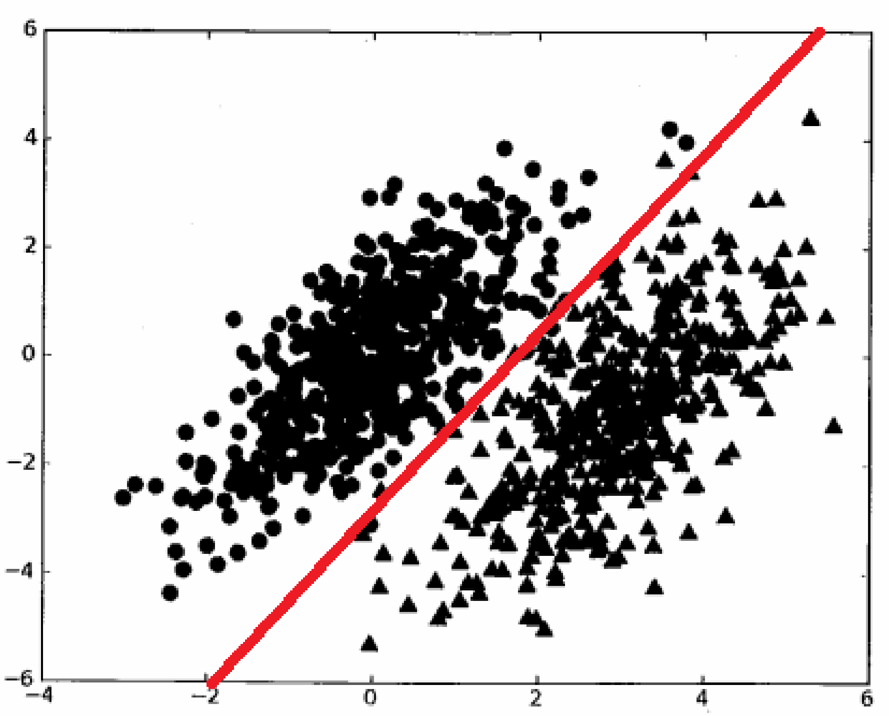
$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

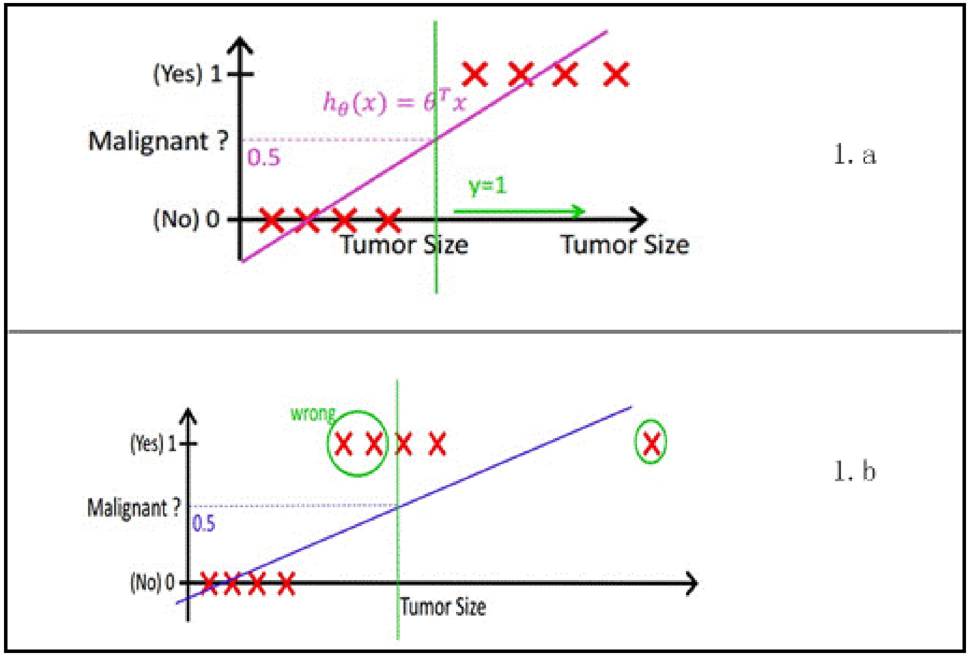
- τ称为带宽，它控制着训练样本随着与$x^{(i)}$距离的衰减速率。
- 多项式核函数

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + R)^d$$

  - 在SVM章节继续核函数的讨论。

思考：用回归解决分类问题？





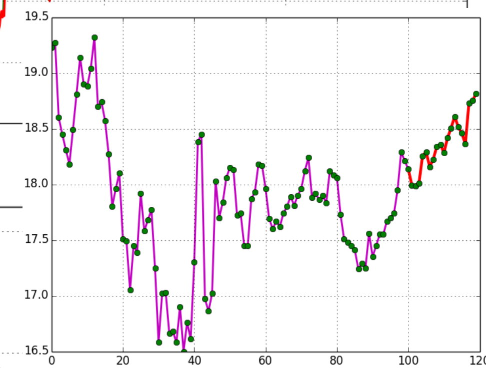
总结和思考

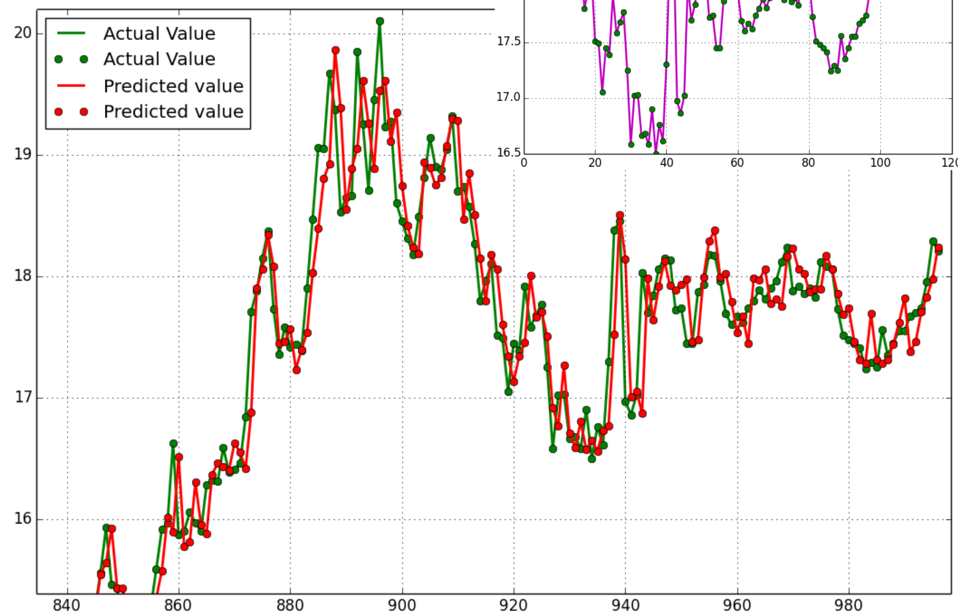
- 特征选择很重要，除了人工选择，还可以用其他机器学习方法，如随机森林、PCA、LDA等。
- 梯度下降算法是参数优化的重要手段，尤其SGD。
  - 适用于在线学习
  - 跳出局部极小值
- 思考：计算可逆方阵的逆，可否使用梯度下降算法？

股价拟合

- 方法：自回归
- 参数：100阶

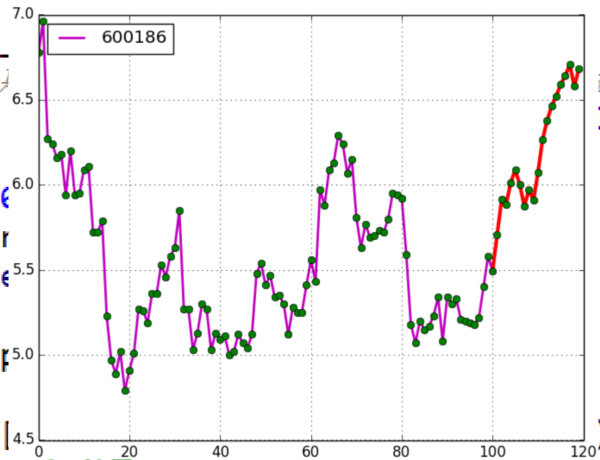
```python
if __name__ == "__main__":
    #   日期      开盘      最高      最低      收盘      成交量          成交额
    price = np.loadtxt('sh_600000.txt', delimiter='\t', skiprows=2, usecols=(4,))
    print "原始价格：\n", price
    n = 100            # 阶数
    y = price[n:]
    m = len(y)         # 样本个数
    print "预测价格：\n", y
    x = np.zeros((m, n+1))
    for i in range(m):
        x[i] = np.hstack((price[i:i+n], 1))
    print "自变量：\n", x
    theta = np.linalg.lstsq(x, y)[0]     # theta为回归系数
    print theta
    show(theta, x, y)

    # 预测
    pn = 20      # 预测未来多少天
    x = price[-n:]
    y = np.hstack((price[-n:], np.zeros(pn)))
    for i in range(pn):
        y[n+i] = np.dot(theta, np.hstack((y[i:n+i], 1)))
    show_predict(y, pn)
```

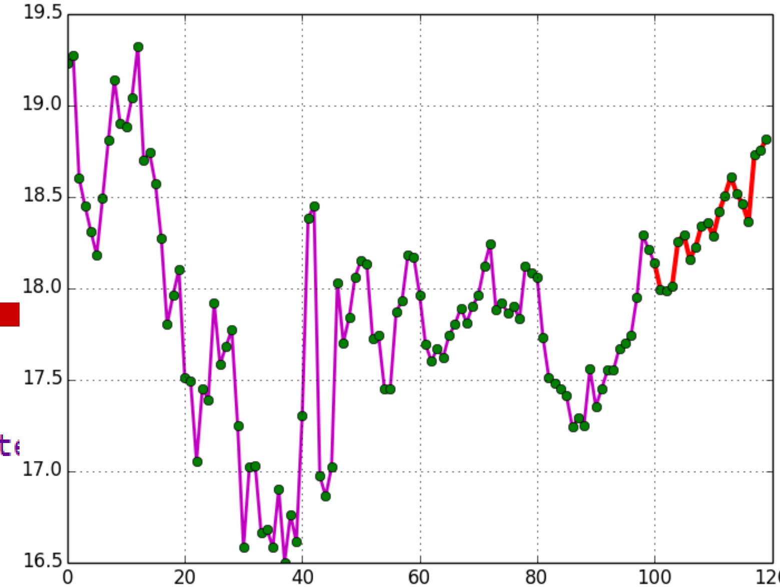




股价预测

```
if __name_
    # 日
    price
    print
    n = 16
    y = pr
    m = le
    print
    x = np
    for i
        x[
    print "自变量: \n", x
    theta
    print
    show(t

    # 预测
    pn = 2
    x = pr
    y = np
    for i
        y[
    show_p
```





SH_600000.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

600000 浦发银行 日线 前复权

日期	开盘	最高	最低	收盘	成交量	成交额
2016/6/1	18.3	18.35	18.14	18.21	10436700	190116112
2016/5/31	17.93	18.4	17.92	18.29	35631526	650177344
2016/5/30	17.84	17.97	17.68	17.95	17538184	313324800
2016/5/27	17.69	17.77	17.63	17.74	11621242	205801168
2016/5/26	17.65	17.86	17.65	17.7	11065062	196266944
2016/5/25	17.68	17.73	17.6	17.67	13766238	243549456
2016/5/24	17.48	17.6	17.43	17.55	10725938	187905440
2016/5/23	17.5	17.68	17.44	17.55	14080659	247441296
2016/5/20	17.31	17.5	17.27	17.45	11140621	194223008
2016/5/19	17.44	17.49	17.33	17.35	10380826	180686336
2016/5/18	17.25	17.62	17.02	17.56	30962453	536935488
2016/5/17	17.29	17.34	17.16	17.25	9765153	168387280
2016/5/16	17.23	17.35	17.21	17.29	11045624	190872656
2016/5/13	17.37	17.46	17.22	17.24	11013829	190929408
2016/5/12	17.43	17.5	17.2	17.41	14668954	254410736
2016/5/11	17.55	17.57	17.4	17.45	11941164	208655792
2016/5/10	17.38	17.56	17.38	17.48	16846648	294675488

第 1100 行，第 1 列

作业

- 解释线性回归中使用误差平方和作为目标函数的原因。
- 请描述BGD和SGD的区别，并指出SGD的优势有哪些？
- 特征选择后如果得到共线性特征，应该如何处理？

参考文献

- Prof. Andrew Ng. *Machine Learning*. Stanford University
- 李航，统计学习方法，清华大学出版社，2012
- Stephen Boyd, Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004
  - 中译本：王书宁，许鋆，黄晓霖，凸优化，清华大学出版社，2013

我们在这里

- http://wenda.ChinaHadoop.cn
  - 视频/课程/社区
- 微博
  - @ChinaHadoop
  - @邹博_机器学习
- 微信公众号
  - 小象学院
  - 大数据分析挖掘

感谢大家！

恳请大家批评指正！