机器学习中导数最优化方法(基础篇)

1. 前言

熟悉机器学习的童鞋都知道，优化方法是其中一个非常重要的话题，最常见的情形就是利用目标函数的导数通过多次迭代来求解无约束最优化问题。实现简单，coding 方便，是训练模型的必备利器之一。这篇博客主要总结一下使用导数的最优化方法的几个基本方法，梳理梳理相关的数学知识，本人也是一边写一边学，如有问题，欢迎指正，共同学习，一起进步。

2. 几个数学概念

1) 梯度（一阶导数）

考虑一座在 (x1, x2) 点高度是 f(x1, x2) 的山。那么，某一点的梯度方向是在该点坡度最陡的方向，而梯度的大小告诉我们坡度到底有多陡。注意，梯度也可以告诉我们不在最快变化方向的其他方向的变化速度（二维情况下，按照梯度方向倾斜的圆在平面上投影成一个椭圆）。对于一个含有 n 个变量的标量函数，即函数输入一个 n 维 的向量，输出一个数值，梯度可以定义为：

$$\nabla_{\boldsymbol{x}} \stackrel{\text{def}}{=} \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \cdots, \frac{\partial}{\partial x_n}\right]^T = \frac{\partial}{\partial \boldsymbol{x}}$$

2) Hesse 矩阵（二阶导数）

Hesse 矩阵常被应用于**牛顿法解决的大规模优化问题**(后面会介绍)，主要形式如下：

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

当 f(x) 为二次函数时，梯度以及 Hesse 矩阵很容易求得。二次函数可以写成下列形式：

$$f(x) = \frac{1}{2}x^T A x + b^T x + c$$

其中 A 是 n 阶对称矩阵，b 是 n 维列向量， c 是常数。f(x) 梯度是 Ax+b, Hesse 矩阵等于 A。

3) Jacobi 矩阵

Jacobi 矩阵实际上是向量值函数的梯度矩阵，假设F:Rn→Rm 是一个从n维欧氏空间转换到m维欧氏空间的函数。这个函数由m个实函数组成: $y(x) = [y_1(x_1, ..., x_n), \cdots, y_m(x_1, ..., x_n)]$。这些函数的偏导数(如果存在)

可以组成一个m行n列的矩阵(m by n)，这就是所谓的雅可比矩阵：

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

总结一下,

a) 如果 f(x) 是一个标量函数，那么雅克比矩阵是一个向量，等于 f(x) 的梯度， Hesse 矩阵是一个二维矩阵。如果 f(x) 是一个向量值函数，那么Jacobi 矩阵是一个二维矩阵，Hesse 矩阵是一个三维矩阵。

b) 梯度是 Jacobian 矩阵的特例，梯度的 jacobian 矩阵就是 Hesse 矩阵（一阶偏导与二阶偏导的关系）。

---

3. 优化方法

1) Gradient Descent

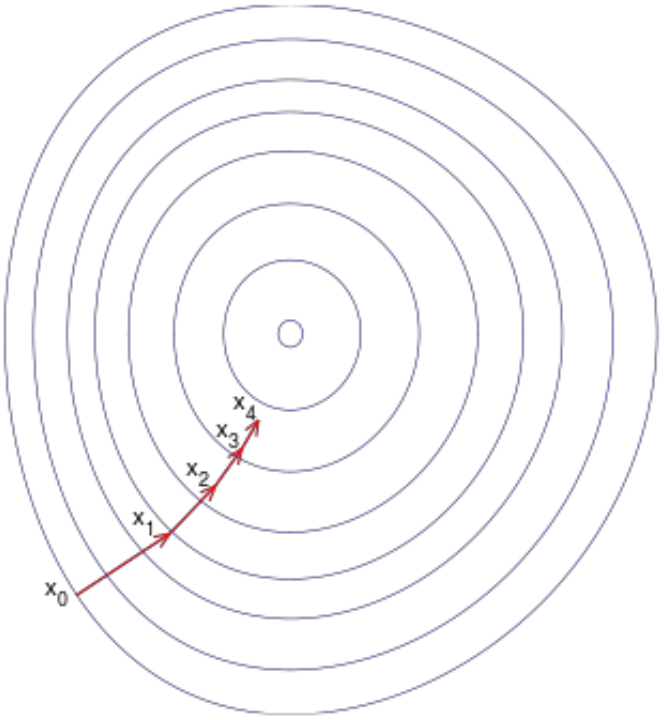
Gradient descent 又叫 steepest descent，是利用一阶的梯度信息找到函数局部最优解的一种方法，也是机器学习里面最简单最常用的一种优化方法。Gradient descent 是 line search 方法中的一种，主要迭代公式如下：

$$x_{k+1} = x_k + \alpha_k \mathbf{p}_k$$

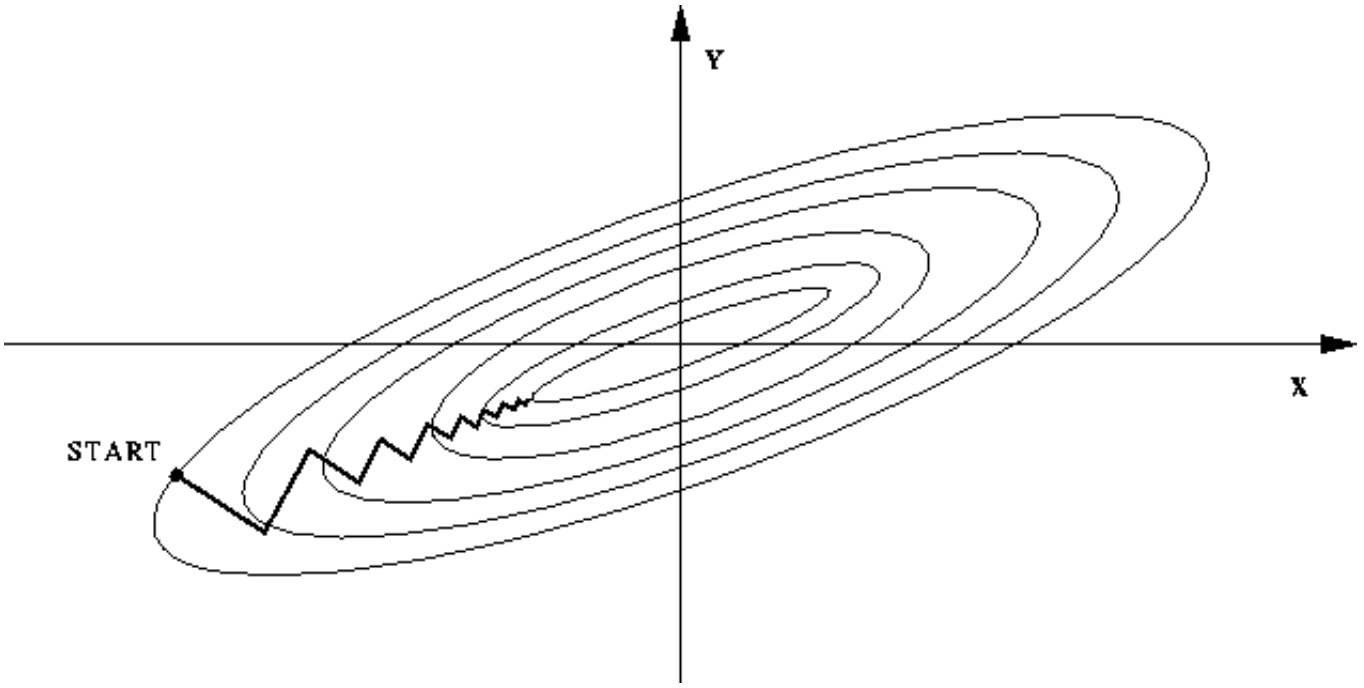
其中，$\mathbf{p}_k$是第 k 次迭代我们选择移动的方向，在 steepest descent 中，移动的方向设定为梯度的负方向， α_k是第 k 次迭代用 line search 方法选择移动的距离，每次移动的距离系数可以相同，也可以不同，有时候我们也叫学习率（learning rate）。在数学上，移动的距离可以通过 line search 令导数为零找到该方向上的最小值，但是在实际编程的过程中，这样计算的代价太大，我们一般可以将它设定位一个常量。考虑一个包含三个变量的函数 $f(\mathbf{x}) = 0.5x_1^2 + 0.2x_2^2 + 0.6x_3^2$，计算梯度得到 $\nabla f(\mathbf{x}) = (x_1, 0.4x_2, 1.2x_3)$。设定 learning rate = 1，算法代码如下：

`steepest.py`

Steepest gradient 方法得到的是局部最优解，如果目标函数是一个凸优化问题，那么局部最优解就是全局最优解，理想的优化效果如下图，值得注意一点的是，每一次迭代的移动方向都与出发点的等高线垂直：



需要指出的是，在某些情况下，最速下降法存在**锯齿现象**（ zig-zagging）将会导致收敛速度变慢:



粗略来讲，在二次函数中，椭球面的形状受 hesse 矩阵的条件数影响，长轴与短轴对应矩阵的最小特征值和最大特征值的方向，其大小与特征值的平方根成反比，最大特征值与最小特征值相差越大，椭球面越扁，那么优化路径需要走很大的弯路，计算效率很低。

2) Newton's method

在最速下降法中，我们看到，该方法主要利用的是目标函数的局部性质，具有一定的“盲目性”。牛顿法则是利用局部的一阶和二阶偏导信息，推测整个目标函数的形状，进而可以求得出近似函数的全局最小值，然后将当前的最小值设定近似函数的最小值。相比最速下降法，牛顿法带有一定对全局的预测性，收敛性质也更优良。牛顿法的主要推导过程如下：

第一步，利用 Taylor 级数求得原目标函数的二阶近似：

$$f(\mathbf{x}) \approx \phi(\mathbf{x}) = f(\mathbf{x}^{(k)}) + \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$

第二步，令一阶偏导为 0 ，求近似函数的最小值：

$$\mathbf{p}_k = -(\nabla^2 f(\mathbf{x_k}))^{-1} \nabla f(\mathbf{x_k})$$

即：

$$\mathbf{p}_k = -Hesse^{-1}\, Jacobi$$

第三步，将当前的最小值设定近似函数的最小值。

与 **1)** 中优化问题相同，牛顿法的代码如下：

`Newton.py`

上面例子中由于目标函数是二次凸函数，Taylor 展开等于原函数，所以能一次就求出最优解。

牛顿法主要存在的问题是：

1. Hesse 矩阵不可逆时无法计算
2. 矩阵的逆计算复杂为 n 的立方，当问题规模比较大时，计算量很大，解决的办法是采用拟牛顿法如 BFGS, L-BFGS, DFP, Broyden's Algorithm 进行近似。
3. 如果初始值离局部极小值太远，Taylor 展开并不能对原函数进行良好的近似

3) Levenberg–Marquardt Algorithm

Levenberg–Marquardt algorithm 能结合以上两种优化方法的优点，并对两者的不足做出改进。与 line search 的方法不同，LMA 属于一种“信赖域法”(trust region)，牛顿法实际上也可以看做一种信赖域法，即利用局部信息对函数进行建模近似，求取局部最小值。所谓的信赖域法，就是从初始点开始，先假设一个可以信赖的最大位移 s（牛顿法里面 s 为无穷大），然后在以当前点为中心，以 s 为半径的区域内，通过寻找目标函数的一个近似函数（二次的）的最优点，来求解得到真正的位移。在得到了位移之后，再计算目标函数值，如果其使目标函数值的下降满足了一定条件，那么就说明这个位移是可靠的，则继续按此规则迭代计算下去；如果其不能使目标函数值的下降满足一定的条件，则应减小信赖域的范围，再重新求解。

LMA 最早提出是用来解决最小二乘法曲线拟合的优化问题的，对于随机初始化的已知参数 beta， 求得的目标值为：

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} [y_i - f(x_i,\ \boldsymbol{\beta})]^2$$

对拟合曲线函数进行一阶 Jacobi 矩阵的近似：

$$f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta}) \approx f(x_i, \boldsymbol{\beta}) + J_i \boldsymbol{\delta}$$

进而推测出 S 函数的周边信息：

$$S(\boldsymbol{\beta} + \boldsymbol{\delta}) \approx \|\mathbf{y} - \mathbf{f}(\boldsymbol{\beta}) - \mathbf{J}\boldsymbol{\delta}\|^2$$

位移是多少时得到 S 函数的最小值呢？通过几何的概念，当残差 $\mathbf{y} - f(\beta) - J\delta$ 垂直于 J 矩阵的 span 空间

时， S 取得最小（至于为什么？请参考之前博客的最后一部分)

$$(\mathbf{J^T J})\boldsymbol{\delta} = \mathbf{J^T}[\mathbf{y} - \mathbf{f}(\boldsymbol{\beta})]$$

我们将这个公式略加修改，加入阻尼系数得到：

$$(\mathbf{J^T J} + \lambda \mathbf{I})\boldsymbol{\delta} = \mathbf{J^T}[\mathbf{y} - \mathbf{f}(\boldsymbol{\beta})]$$

就是莱文贝格－马夸特方法。这种方法只计算了一阶偏导，而且不是目标函数的 Jacobia 矩阵，而是拟合函数的 Jacobia 矩阵。当 λ 大的时候可信域小，这种算法会接近最速下降法， λ 小的时候可信域大，会接近高斯-牛顿方法。

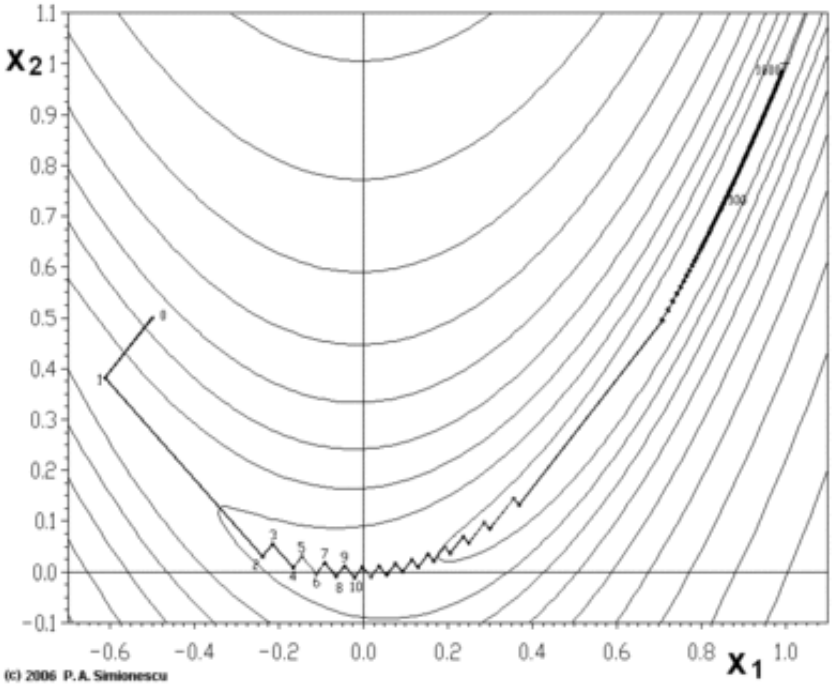
算法过程如下：

1. 给定一个初识值 x0
2. 当 $\mathbf{J}^T[\mathbf{y} - \mathbf{f}(\beta)] > tolerance$ 并且没有到达最大迭代次数时
3. 重复执行:
    - 算出移动向量δ
    - 计算更新值：$\mathbf{x_{new}} = \mathbf{x} + \delta$
    - 计算目标函数真实减少量与预测减少量的比率 ρ
    - if $0 < \rho < 0.25$，接受更新值
    - else if $\rho > 0.25$，说明近似效果很好，接受更新值，扩大可信域（即减小阻尼系数）
    - else: 目标函数在变大，拒绝更新值，减小可信域（即增加阻尼系数）
4. 直到达到最大迭代次数

维基百科在介绍 Gradient descent 时用包含了细长峡谷的 Rosenbrock function

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

展示了 zig-zagging 锯齿现象：



用 LMA 优化效率如何。套用到我们之前 LMA 公式中，有：

$$\begin{aligned} y &= 0 \\ f(\beta) &= (10(x_2 - x_1^2), \quad 1 - x_1)^T \\ J &= \begin{pmatrix} -20x_1 & 10 \\ -1 & 0 \end{pmatrix} \end{aligned}$$

代码如下：

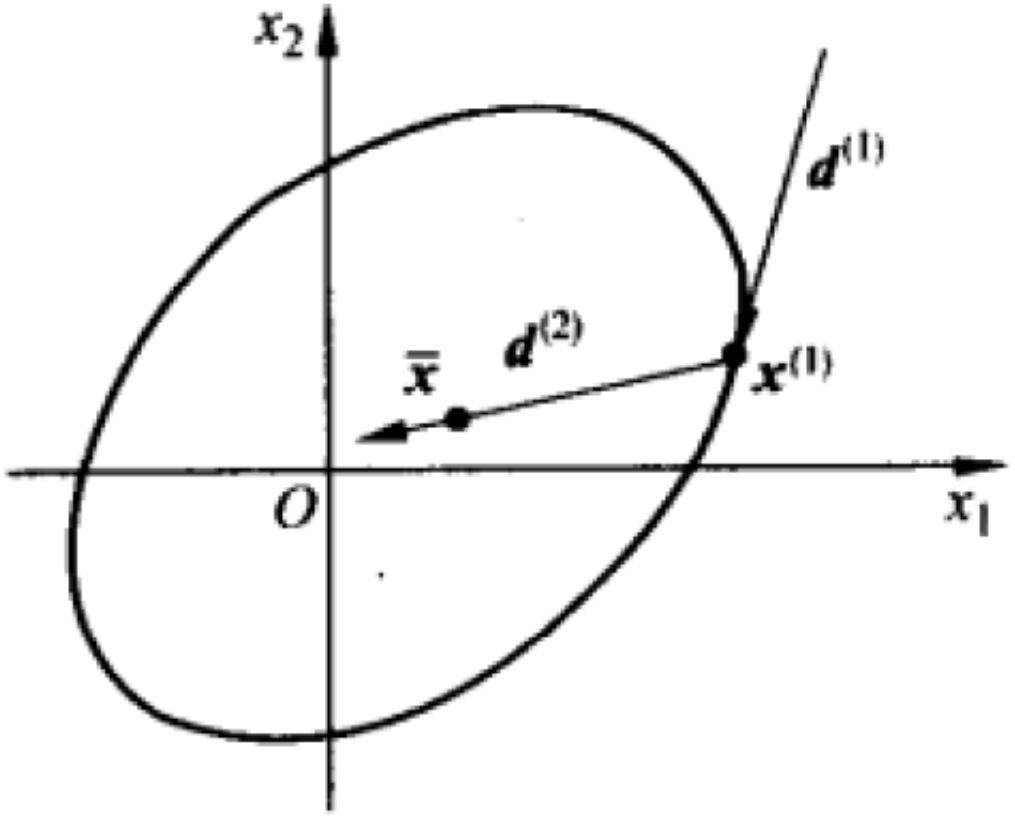
`LevenbergMarquardt.py`

大概 5 次迭代就可以得到最优解 (1, 1).

Levenberg–Marquardt algorithm 对局部极小值很敏感，维基百科举了一个二乘法曲线拟合的例子，当使用不同的初始值时，得到的结果差距很大，我这里也有 python 代码，就不细说了。

4) Conjugate Gradients

共轭梯度法也是优化模型经常经常要用到的一个方法，背后的数学公式和原理稍微复杂一些，光这一个优化方法就可以写一篇很长的博文了，所以这里并不打算详细讲解每一步的推导过程，只简单写一下算法的实现过程。与最速梯度下降的不同，共轭梯度的优点主要体现在选择搜索方向上。在了解共轭梯度法之前，我们首先简单了解一下共轭方向：

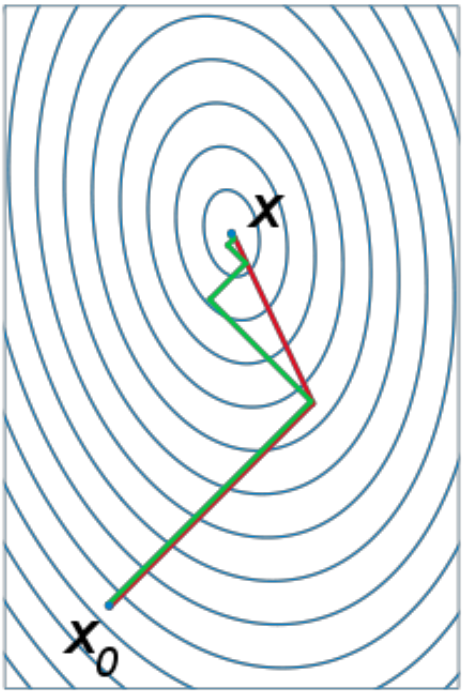


共轭方向和马氏距离的定义有类似之处，他们都考虑了全局的数据分布。如上图，d(1) 方向与二次函数的等值线相切，d(1) 的共轭方向 d(2) 则指向椭圆的中心。所以对于二维的二次函数，如果在两个共轭方向上进行一维搜索，经过两次迭代必然达到最小点。前面我们说过，等值线椭圆的形状由 Hesse 矩阵决定，那么，上图的两个方向关于 Hessen 矩阵正交，共轭方向的定义如下：

$$\mathbf{d}^{(1)T}\mathbf{A}\mathbf{d}^{(2)} = 0$$

如果椭圆是一个正圆， Hessen 矩阵是一个单位矩阵，上面等价于欧几里得空间中的正交。

在优化过程中，如果我们确定了移动方向（GD：垂直于等值线，CG：共轭方向），然后在该方向上搜索极小值点（恰好与该处的等值线相切），然后移动到最小值点，重复以上过程，那么 Gradient Descent 和 Conjugate gradient descent 的优化过程可以用下图的**绿线**与**红线**表示：



讲了这么多，共轭梯度算法究竟是如何算的呢？

1. 给定一个出发点 x0 和一个停止参数 e, 第一次移动方向为**最速下降方向**: $\mathbf{p}_0 = -\nabla f(\mathbf{x})$
2. $\mathbf{p}_{new} = \mathbf{p}_0$
3. while $\mathbf{p}_{new} > e^2\mathbf{p}_0$:
    - 用 Newton-Raphson 迭代计算移动距离，以便在该搜索方向移动到极小，公式就不写了，具体思路就是利用一阶梯度的信息向极小值点跳跃搜索
    - 移动当前的优化解 x： $\mathbf{x}_{new} = \mathbf{x} + \alpha_k\mathbf{p}$
    - 用 Gram-Schmidt 方法构造下一个共轭方向，即 $\mathbf{p}_{new} = \nabla f(\mathbf{x}_{new}) + \beta_{k+1}\mathbf{p}$, 按照 β 的确定公式又可以分为 FR 方法和 PR 和 HS 等。

在很多的资料中，介绍共轭梯度法都举了一个求线性方程组 Ax = b 近似解的例子，实际上就相当于这里所说的 $\min f(\mathbf{x}) = \frac{1}{2}||\mathbf{Ax} - \mathbf{b}||^2$

还是用最开始的目标函数 $f(\mathbf{x}) = 0.5x_1^2 + 0.2x_2^2 + 0.6x_3^2$ 来编写共轭梯度法的优化代码：

`CG.py`

参考资料：

[1] Machine Learning: An Algorithmic Perspective, chapter 11
[2] 最优化理论与算法（第2版），陈宝林
[3] wikipedia