**第一部分**

**监督学习：**我们数据集中的每个样本都有相应的“正确答案”，再根据这些样本作出预测；处理分类和回归；回归问题，即通过回归来推出一个连续的输出；分类问题，其目标是推出一组离散的结果；如k近邻，线性回归和多项式回归，逻辑回归，SVM，决策树和随机森林。

**无监督学习：**无监督学习中没有任何的标签或者是有相同的标签或者就是没标签。有已知数据集，却不知如何处理，也未告知每个数据点是什么。别的都不知道，就是一个数据集。无监督学习算法可能会把这些数据分成两个不同的簇，所以叫做聚类算法。

**泛化能力（generalization ability）：**由该方法学习到的模型对未知数据的预测能力，是学习方法本质上重要的性质。现实中采用最多的办法是通过测试误差（泛化误差）来评价学习方法的泛化能力。 但这种评价是依赖于测试数据集的。因为测试数据集是有限的，很有可能由此得到的评价结果是不可靠的。

**过拟合：**曲线很好的拟合了样本，太过于注重训练数据的细节，甚至将一些噪声错误的当成特征；模型在训练集上表现好，验证和测试阶段就大不如意了，模型的泛化能力很差；

解决过拟合(高方差)的方法：

1. 增加训练数据数

发生过拟合最常见的现象就是数据量太少而模型太复杂；过拟合是由于模型学习到了数据的一些噪声特征导致，增加训练数据的量能够减少噪声的影响，让模型更多地学习数据的一般特征；增加数据量有时可能不是那么容易，需要花费一定的时间和精力去搜集处理数据；利用现有数据进行扩充或许也是一个好办法，例如在图像识别中，如果没有足够的图片训练，可以把已有的图片进行旋转，拉伸，镜像，对称等，这样就可以把数据量扩大好几倍而不需要额外补充数据；注意保证训练数据的分布和测试数据的分布要保持一致。

2. 使用正则化约束

在代价函数后面添加正则化项，可以避免训练出来的参数过大从而使模型过拟合。使用正则化缓解过拟合的手段广泛应用，不论是在线性回归还是在神经网络的梯度下降计算过程中，都应用到了正则化的方法。常用的正则化有l1l1正则和l2l2正则，具体使用哪个视具体情况而定，一般l2l2正则应用比较多；

3. 减少特征数

欠拟合需要增加特征数，那么过拟合自然就要减少特征数。去除那些非共性特征，可以提高模型的泛化能力

4. 调整参数和超参数

不论什么情况，调参是必须的

5. 降低模型的复杂度

欠拟合要增加模型的复杂度，那么过拟合正好反过来

6. 使用Dropout

这一方法只适用于神经网络中，即按一定的比例去除隐藏层的神经单元，使神经网络的结构简单化

7. 提前结束训练

即early stopping，在模型迭代训练时候记录训练精度(或损失)和验证精度(或损失)，倘若模型训练的效果不再提高，比如训练误差一直在降低但是验证误差却不再降低甚至上升，这时候便可以结束模型训练了

**欠拟合：**模型拟合能力不足，对于细节把握的还不够完善；在训练集、验证集和测试集上均表现不佳。

解决欠拟合(高偏差)的方法：

1. 模型复杂化

对同一个算法复杂化。例如回归模型添加更多的高次项，增加决策树的深度，增加神经网络的隐藏层数和隐藏单元数等；弃用原来的算法，使用一个更加复杂的算法或模型，例如用神经网络来替代线性回归，用随机森林来代替决策树等

2. 增加更多的特征

特征挖掘十分重要，尤其是具有强表达能力的特征，往往可以抵过大量的弱表达能力的特征

特征的数量往往并非重点，质量才是，总之强特最重要，能否挖掘出强特，还在于对数据本身以及具体应用场景的深刻理解，往往依赖于经验

3. 调整参数和超参数

超参数包括：神经网络中的学习率、学习衰减率、隐藏层数、隐藏层的单元数、Adam优化算法中的β1β1和β2β2参数、batch\_size数值等；其他算法中的随机森林的树数量，k-means中的cluster数，正则化参数λλ等

4. 增加训练数据往往没有用

欠拟合本来就是模型的学习能力不足，增加再多的数据给它训练它也没能力学习好

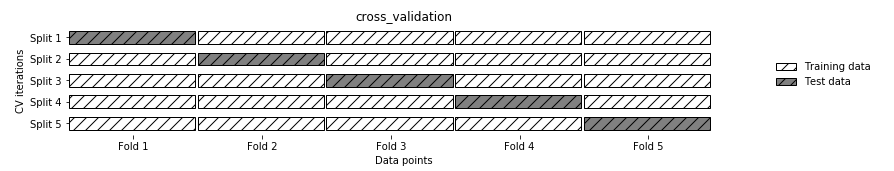
5. 降低正则化约束

正则化约束是为了防止模型过拟合，如果模型压根不存在过拟合而是欠拟合了，那么就考虑是否降低正则化参数λλ或者直接去除正则化项

**交叉验证：**交叉验证是一种评估泛化性能的统计学方法，它比单次划分训练集和测试集的方法更加稳定、全面。最常用的交叉验证是k折交叉验证，在此思想上改进的交叉验证方法还有分层交叉验证、打乱划分交叉验证、分组交叉验证、嵌套交叉验证。

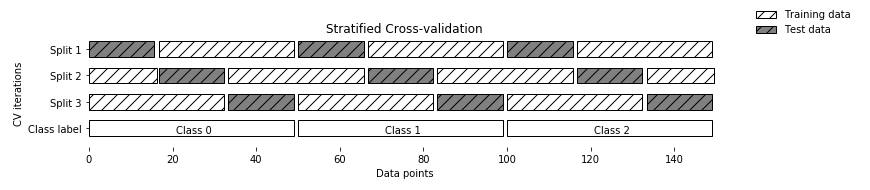
1. k折交叉验证

将数据划分为大致相等的k折（部分），轮流将某一折作为测试集，其它折作为训练集来训练模型和评估精度。



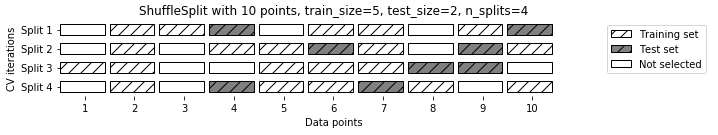
2. 分层交叉验证

分折时，使每个折中类别之间的比例与整个数据集中的比例相同。



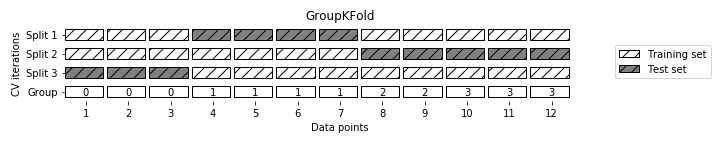
3. 打乱划分交叉验证

将数据打乱来代替分层。每次划分为训练集取样train\_size个点，为测试集取样test\_size个不相交的点，将这一划分方法重复n\_iter次。每次划分的训练集和测试集不相交，但是不同次的划分可能重复选取部分数据作为测试集。



4. 分组交叉验证

对于每次划分，每个分组都是都是整体出现在训练集或测试集中。



5. 嵌套交叉验证

外层for循环将原始数据使用交叉验证进行多次划分，内层for循环在划分好的训练集中再使用交叉验证进行多次划分，主要用于网格搜索里面。

重要知识点：

①Scikit-Learn是利用model\_selection模块中的cross\_val\_score函数来实现交叉验证的。

②总结交叉验证精度的一种常用方法是计算平均值。

③使用交叉验证可以消除偶然性得分，使得对模型的评估更具准确性。

④交叉验证的主要缺点是增加了计算成本。

⑤交叉验证不会返回一个模型，其目的是评估给定算法在特定数据集上训练后的泛化能力。

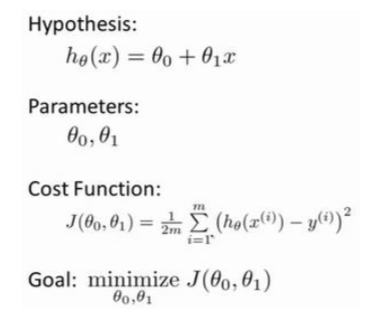
⑥打乱划分交叉验证中允许在每次迭代中仅使用部分数据。

**单变量线性回归(LinearRegressionwithOneVariable)：**监督学习回归问题；hθ(x)=θ0+θ1x只含有一个特征/输入变量，这样的问题叫作单变量线性回归问题。

代价函数（Cost Function）：

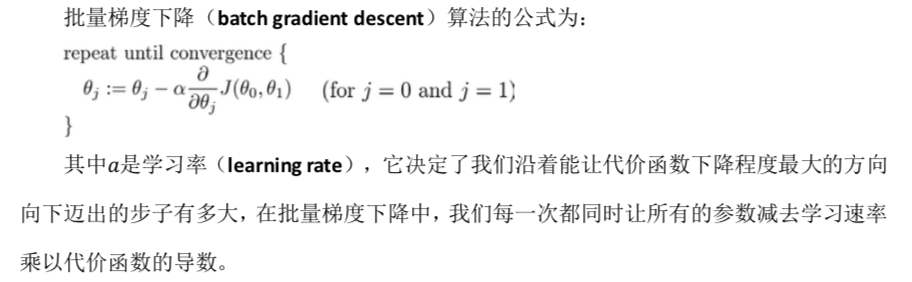


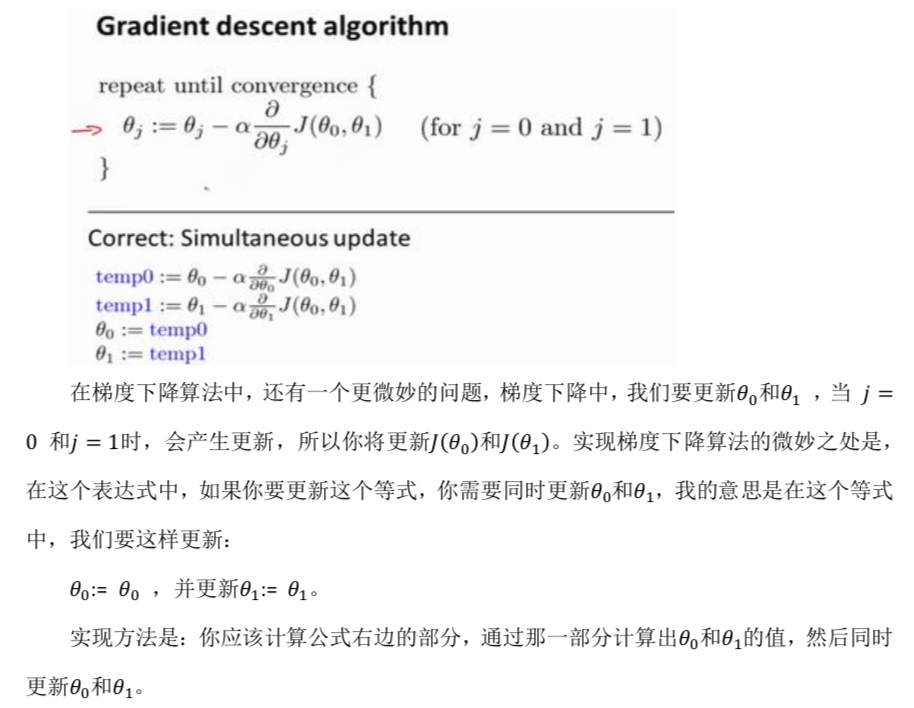
代价函数也被称作平方误差函数，有时也被称为平方误差代价函数。还有其他的代价函数也能很好地发挥作用，但是平方误差代价函数可能是解决回归问题最常用的手段了。



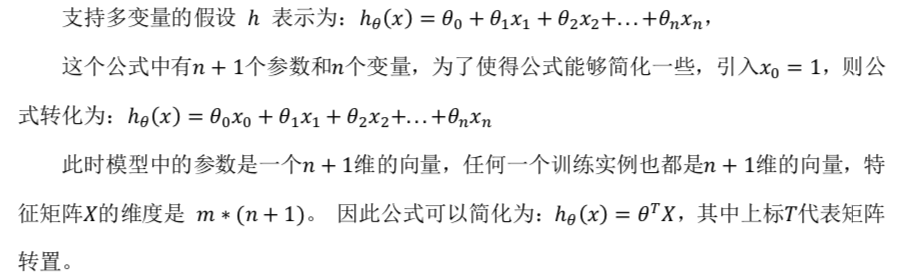
**梯度下降：**

开始时随机选择一个参数的组合(θ0, θ1, . . . . . . , θn)，计算代价函数，然后寻找下一个能让代价函数值下降最多的参数组合。持续这么做直到到到一个局部最小值(local minimum)，不确定是不是全局最小值(global minimum)，选择不同的初始参数组合，可能会找到不同的局部最小值。

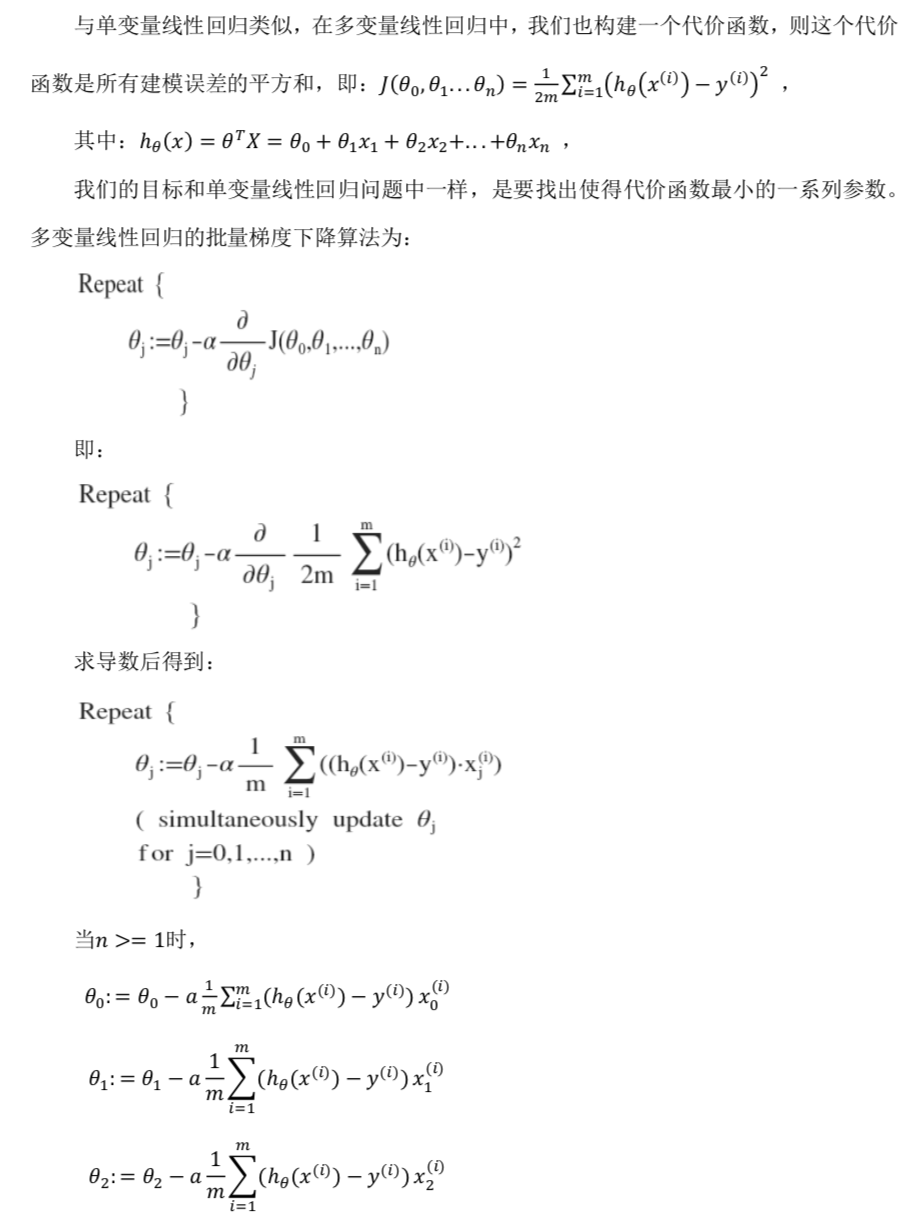


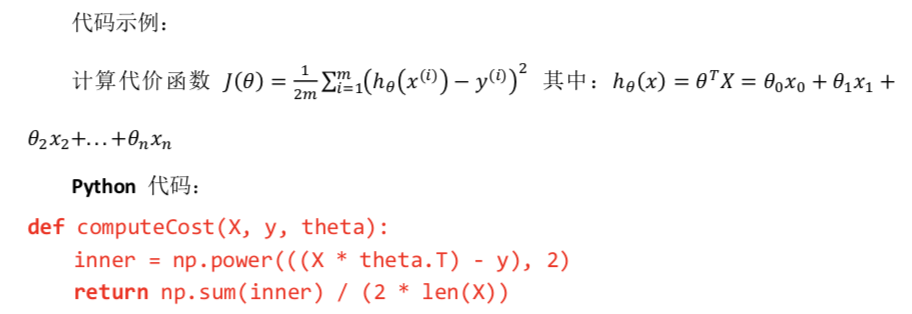


**多变量线性回归(LinearRegressionwithMultipleVariables)：**



**多变量梯度下降：**

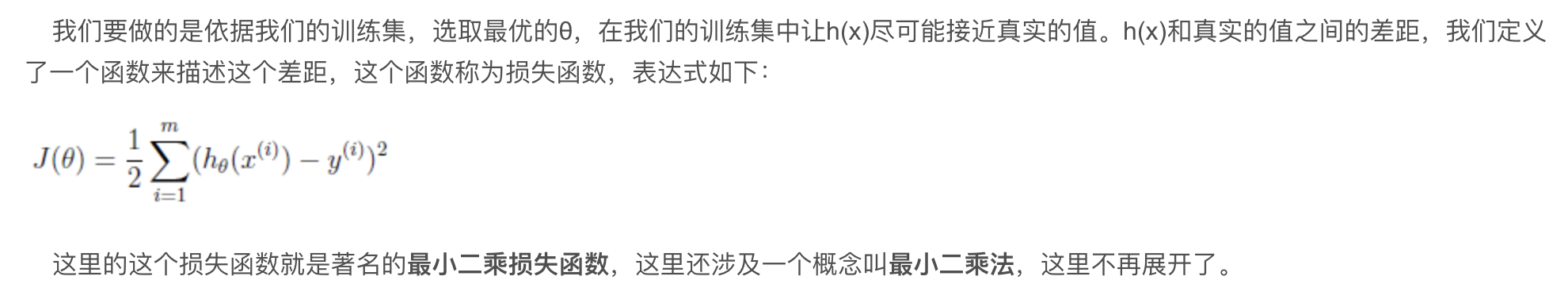




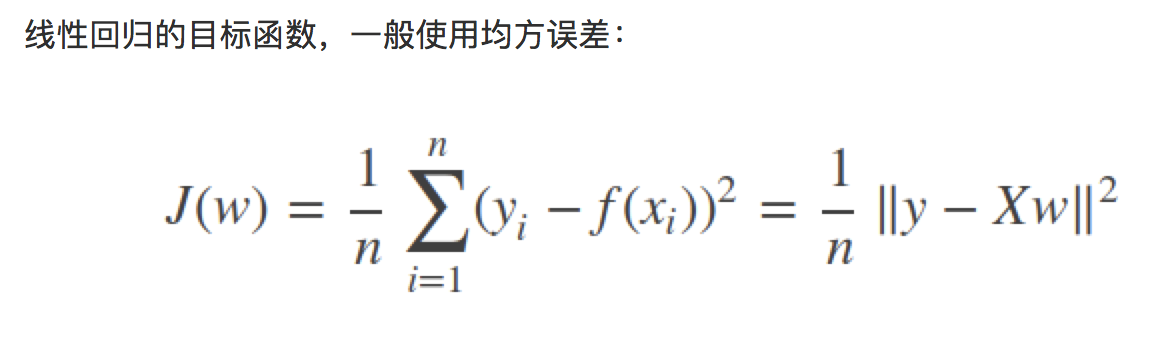
特征缩放：在面对多维特征问题的时候，要保证这些特征都具有相近的尺度，这将帮助梯度下降算法更快地收敛。

学习率：梯度下降算法收敛所需要的迭代次数根据模型的不同而不同，我们不能提前预知，可以绘制迭代次数和代价函数的图表来观测算法在何时趋于收敛；梯度下降算法的每次迭代受到学习率的影响，如果学习率过小，则达到收敛所需的迭代次数会非常高；如果学习率过大，每次迭代可能不会减小代价函数，可能会越过局部最 小值导致无法收敛。

**损失函数：**



**目标函数：**



**牛顿法：**

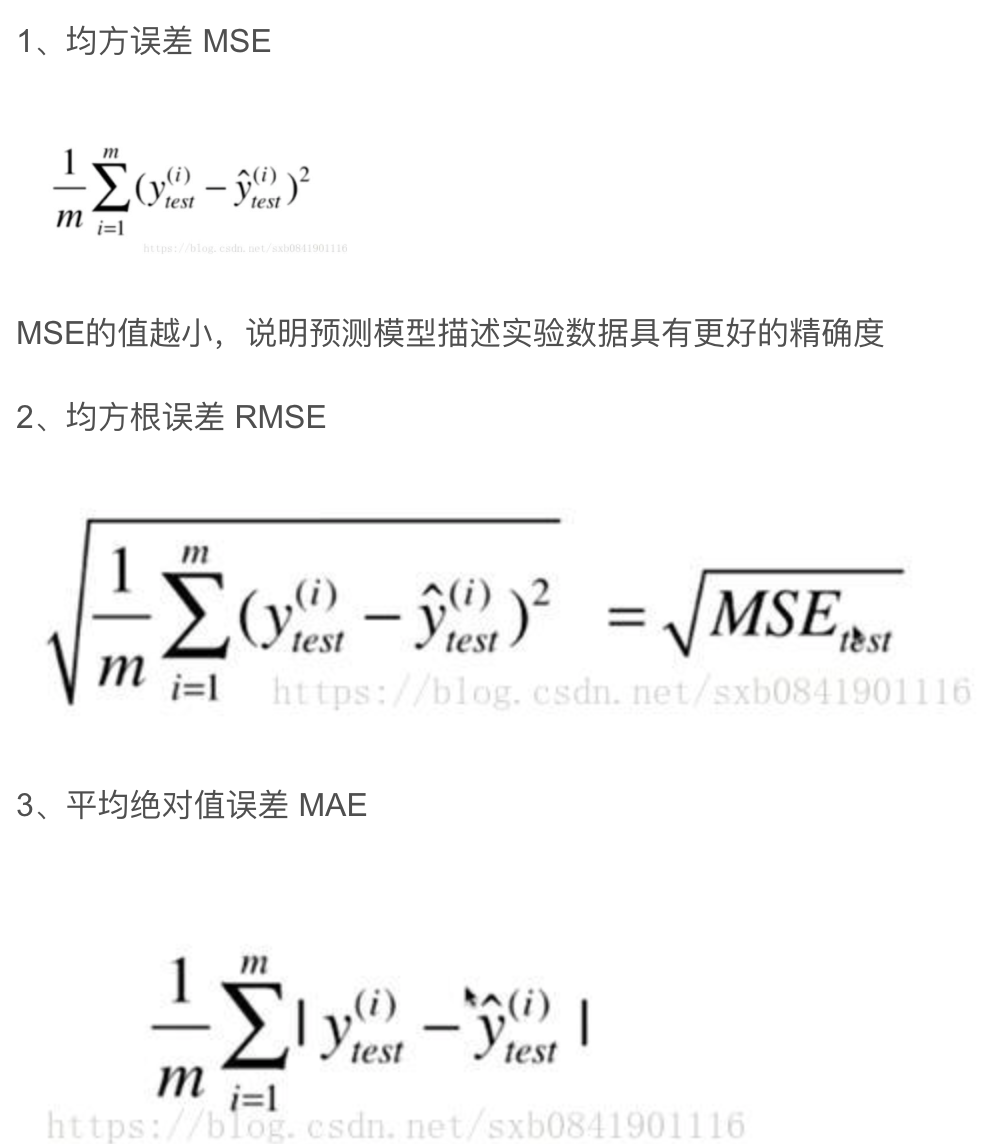
求解无约束最优化问题的常用方法，有收敛速度快的优点，也属于迭代算法，每一步需要求解目标函数的海塞矩阵的逆矩阵；基本思想是利用迭代点处的一阶导数(梯度)和二阶导数(Hessen矩阵)对目标函数进行二次函数近似，然后把二次模型的极小点作为新的迭代点，并不断重复这一过程，直至求得满足精度的近似极小值。牛顿法的速度相当快，而且能高度逼近最优值。牛顿法分为基本的牛顿法和全局牛顿法。

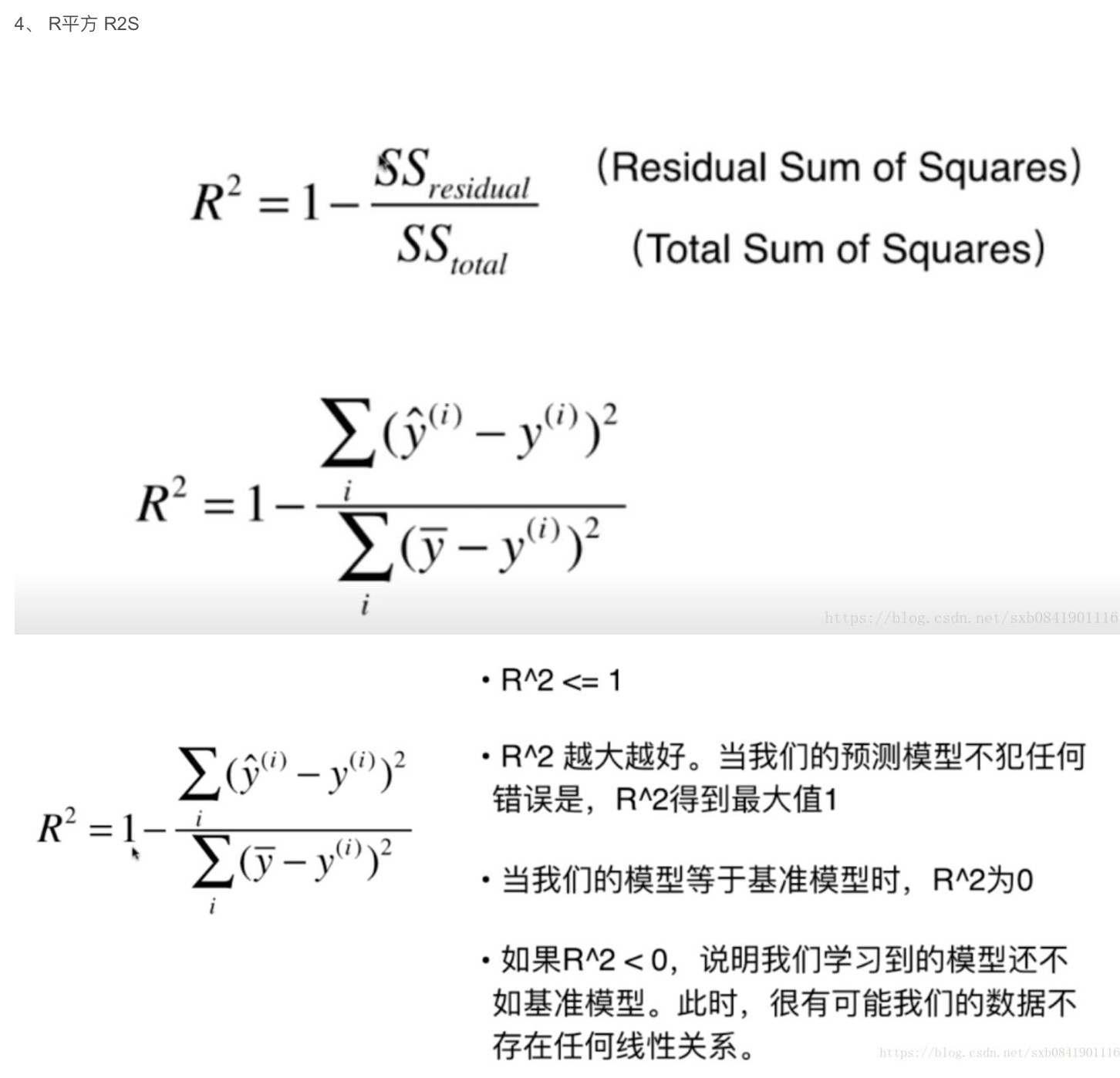
**拟牛顿法：**

拟牛顿法是在牛顿法的基础上引入了Hessian矩阵的近似矩阵，避免每次迭代都计算Hessian矩阵的逆，它的收敛速度介于梯度下降法和牛顿法之间。拟牛顿法跟牛顿法一样，也是不能处理太大规模的数据，因为计算量和存储空间会开销很多。拟牛顿法虽然每次迭代不像牛顿法那样保证是最优化的方向，但是近似矩阵始终是正定的，因此算法始终是朝着最优化的方向在搜索。

**线性回归的评估指标：**

评价线性回归的指标有四种，均方误差（Mean Squared Error）、均方根误差（Root Mean Squared Error）、平均绝对值误差（Mean Absolute Error）以及R Squared方法。





代码实现：

def mean\_squared\_error(y\_true, y\_predict):

"""计算y\_true和y\_predict之间的MSE"""

assert len(y\_true) == len(y\_predict), \

"the size of y\_true must be equal to the size of y\_predict"

return np.sum((y\_true - y\_predict)\*\*2) / len(y\_true)

def root\_mean\_squared\_error(y\_true, y\_predict):

"""计算y\_true和y\_predict之间的RMSE"""

return sqrt(mean\_squared\_error(y\_true, y\_predict))

def mean\_absolute\_error(y\_true, y\_predict):

"""计算y\_true和y\_predict之间的RMSE"""

assert len(y\_true) == len(y\_predict), \

"the size of y\_true must be equal to the size of y\_predict"

return np.sum(np.absolute(y\_true - y\_predict)) / len(y\_true)

def r2\_score(y\_true, y\_predict):

"""计算y\_true和y\_predict之间的R Square"""

return 1 - mean\_squared\_error(y\_true, y\_predict)/np.var(y\_true)

**sklearn参数详解：**

class sklearn.linear\_model.LogisticRegression(penalty=’l2’, dual=False, tol=0.0001, C=1.0,fit\_intercept=True, intercept\_scaling=1, class\_weight=None, random\_state=None,solver=’liblinear’, max\_iter=100, multi\_class=’ovr’, verbose=0,warm\_start=False, n\_jobs=1)

penalty : str, ‘l1’or ‘l2’, default: ‘l2’

fit\_intercept:布尔值，指定是否需要计算线性回归中的截距，即b值。如果为False,那么不计算b值。

normalize:布尔值。如果为False，那么训练样本会进行归一化处理；当为True的时候，则回归量X将在回归之前通过减去平均值并除以I2范数来归一

copy\_X：布尔值。如果为True，会复制一份训练数据。

n\_jobs:一个整数。任务并行时指定的CPU数量。如果取值为-1则使用所有可用的CPU。

coef\_:权重向量

intercept\_:截距b值

参考连接：https://blog.csdn.net/weixin\_41712499/article/details/82526483