# 机器学习笔记

## 前言：

吴恩达的机器学习及深度学习两门课，github上有人做了详尽的笔记

<https://github.com/fengdu78/Coursera-ML-AndrewNg-Notes>

因为有了完整的笔记，这里笔者就不打算记录完整的笔记，而是记录一些对于笔者而言重要的笔记，所以对大家而言就没有什么参考价值了。

## 一、引言

无监督学习

无监督学习中的数据没有任何标签，我们仅有一个数据集，不知如何处理。无监督学习能判断数据中的聚集簇，从而将数据分类，这称之为聚类算法（cluster）。

## 二、单变量线性回归

如果以y = ax + b这样的线性方程来拟合数据，并且以mse（均方误差）为代价函数来计算，那么一下是推导过程。

方程



代价函数

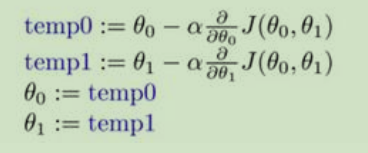


目标：代价函数最小

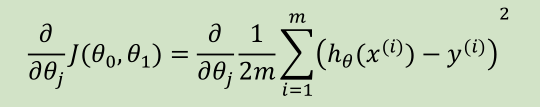
方法：通过更改theta1和theta2的值，使得代价函数降低。而最快的方法则是沿着梯度的方向改变值。Alpha是学习率，那么theta的更新函数如下所示

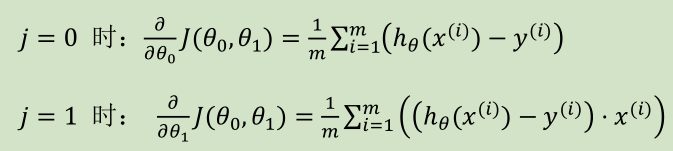


更具体一点，计算过程应该如下：

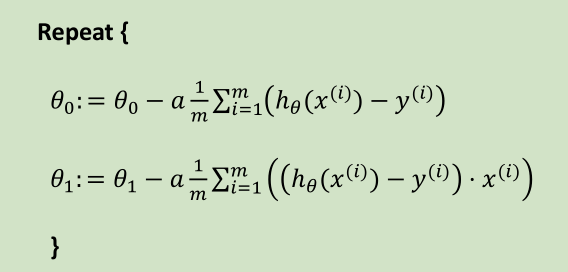


关于偏导数的公式推导：





那算法改写如下：



## 三、线性代数回顾

### 规律

矩阵乘法不满足交换律



矩阵乘法满足结合律



单位矩阵，除对角线元素为1，其余元素都为0的矩阵。一般用I或者E表示。对于单位矩阵，有

AI = IA = A

### 逆矩阵

只有方阵才有逆矩阵。

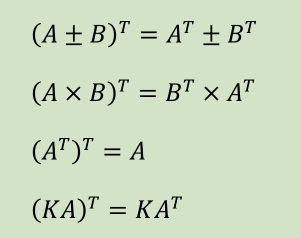
如果有逆矩阵，则

没有逆矩阵的矩阵成为奇异矩阵(Singular)或者退化矩阵(degenerate)

Numpy中求矩阵的额逆的方法

np.linalg.inv()

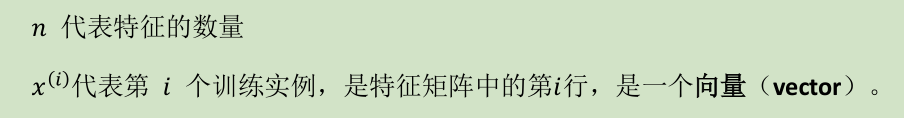
### 矩阵转置

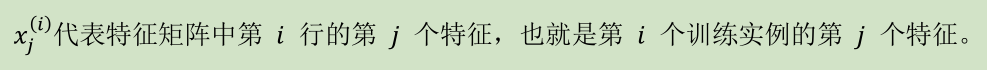


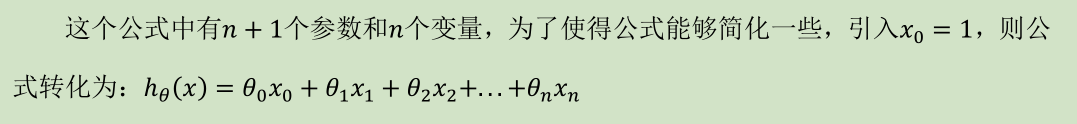
## 四、多变量线性回归

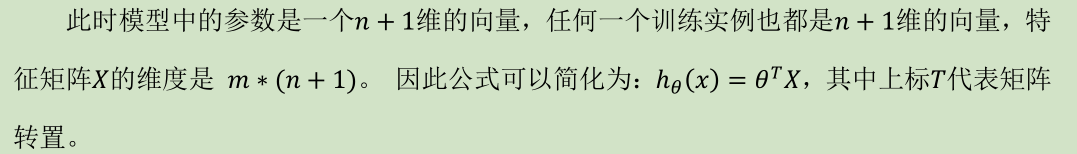
### 符号含义

有了多维特征后，公式推导的符号含义：









### 梯度下降

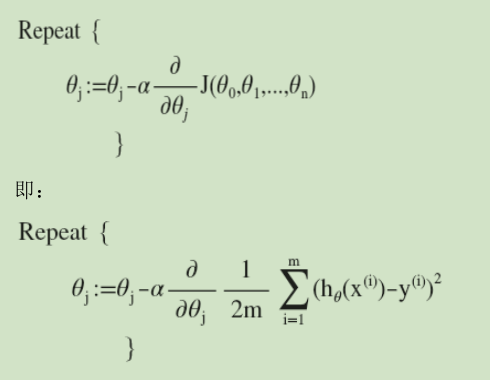
#### 公式



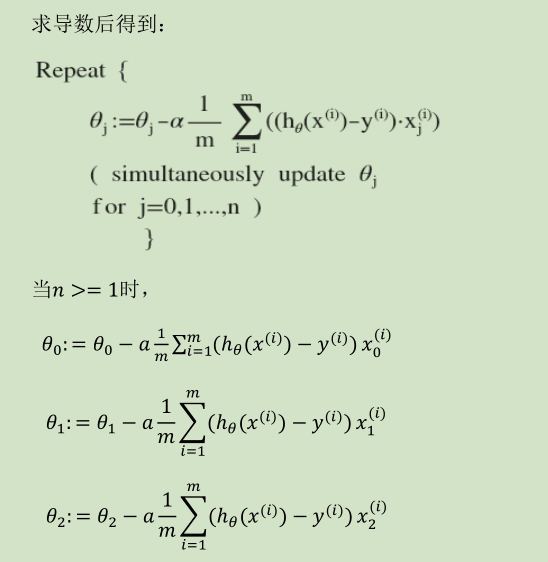
#### 代价函数：



梯度下降算法为：



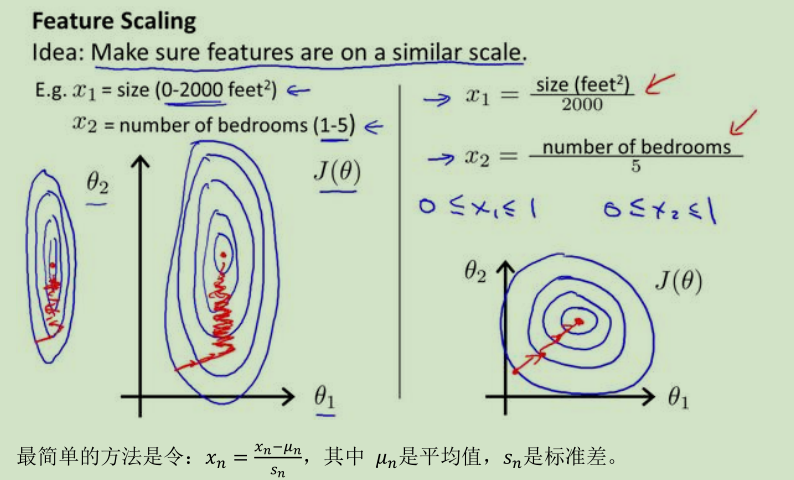
计算后得到



### 特征缩放

在我们面对多维特征问题的时候，我们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。

以房价问题为例，假设我们使用两个特征，房屋的尺寸和房间的数量，尺寸的值为 0-2000 平方英尺，而房间数量的值则是 0-5，以两个参数分别为横纵坐标，绘制代价函数的等高线图能，看出图像会显得很扁，梯度下降算法需要非常多次的迭代才能收敛。

解决的方法是尝试将所有特征的尺度都缩放到一个大致相等的区间之间。如图：

### 学习率

学习率也考虑多测试几个值，Andrew推荐的序列是以3倍递增：

0.001，0.003，0.01，0.03，0.1……

### 多项式回归模型

线性模型有时候并不能满足我们的需要，这时候可以考虑别的曲线来拟合数据，比如二次方模型、三次方模型v





或者：





多项式回归模型的话，特征缩放非常有必要

### 正规方程

梯度下降算法的原理是找到损失函数的最低点；从数学上讲，这个点就是该点梯度为0的地方，而正规方程就是一步到位，求出损失函数梯度为0 的点。

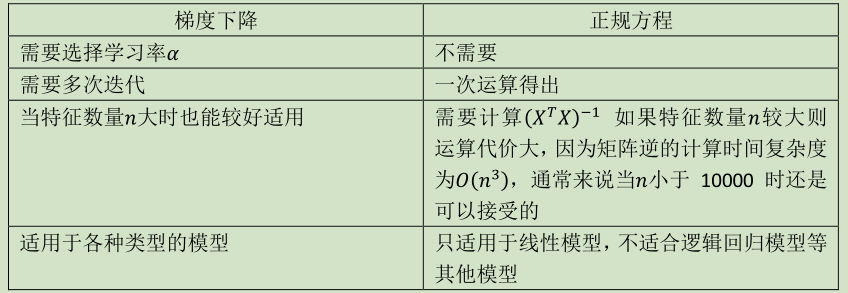
求解公式为：

θ = (XTX)-1XTy

矩阵不可逆通常有2种情况引起：

* 使用了线性相关的特征，比如同时使用m和feet
* 使用了太多特征，比如想用10个数据算出100个特征。

#### 比较



#### 结论

只要特征变量的数目并不大，标准方程是一个很好的计算参数的替代方法。

具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

随着我们要讲的学习算法越来越复杂，例如，当我们讲到分类算法，像逻辑回归算法只要特征变量的数目并不大，标准方程是一个很好的计算参数𝜄的替代方法。对于那些更复杂的学习算法，

我们将不得不仍然使用梯度下降法。

正规方程的 python 实现：

import numpy as np

def normalEqn(X, y):

theta = np.linalg.inv([X.T@X)@X.T@y](mailto:X.T@X)@X.T@y) #X.T@X 等价于 X.T.dot(X)

return theta

## 五、逻辑回归

### 假说表示

逻辑回归模型的假设是





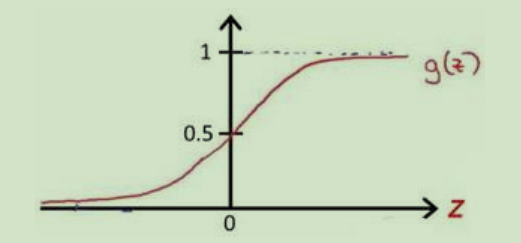
Python代码实现为

import numpy as np

def sigmoid(z):

return 1 / (1 + np.exp(-z))

函数图像



ℎ𝜃 (x)的作用是，对于给定的输入变量，根据选择的参数计算输出变量=1 的可能性

（estimated probablity）即ℎ𝜃 (x) = P(y = 1|x: 𝜃)

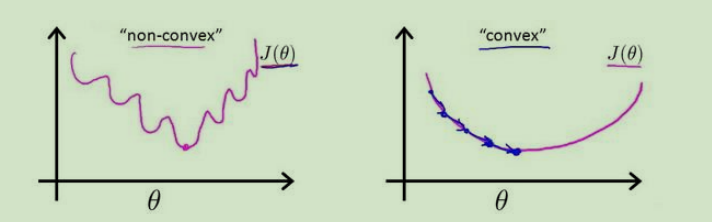
例如，如果对于给定的𝑦，通过已经确定的参数计算得出ℎ 𝜃 (x) = 0.7，则表示有 70%的

几率𝑧为正向类，相应地𝑧为负向类的几率为 1-0.7=0.3

### 代价函数

对于线性回归模型，我们定义的代价函数是所有模型误差的平方和。理论上来说，我们也可以对逻辑回归模型沿用这个定义，但是问题在于，当我们将

带入到这样定义了的代价函数中时，我们得到的代价函数将是一个非凸函数（non-convexfunction）



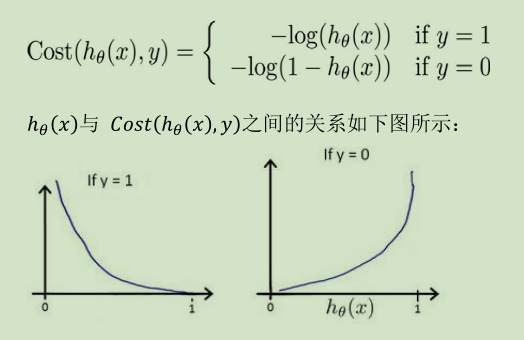
这意味着我们的代价函数有许多局部最小值，这将影响梯度下降算法寻找全局最小值。

最小平方函数不具有单调性，而对数函数具有单调性，能得到凸函数的代价函数，这就是代价函数采取对数函数而不是最小平方法的原因。

重新定义逻辑回归的代价函数为



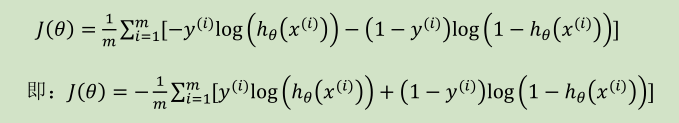
其中



代价函数简化之后如下



带入代价函数中



Python代码实现

import numpy as np

def cost(theta, X, y):

theta = np.matrix(theta)

X = np.matrix(X)

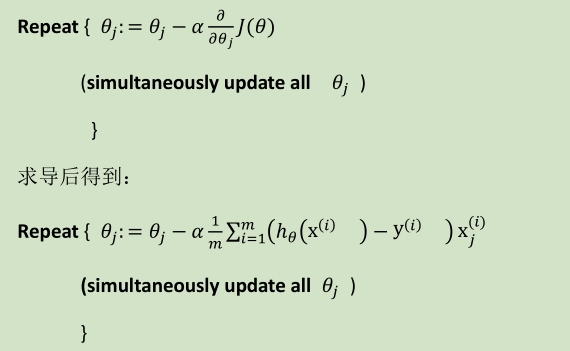
y = np.matrix(y)

first = np.multiply(-y, np.log(sigmoid(X\* theta.T)))

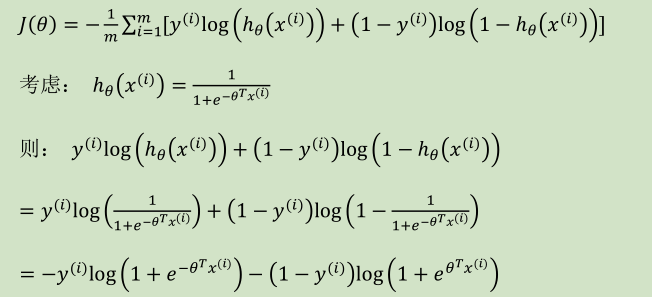
second = np.multiply((1 - y), np.log(1 - sigmoid(X\* theta.T)))

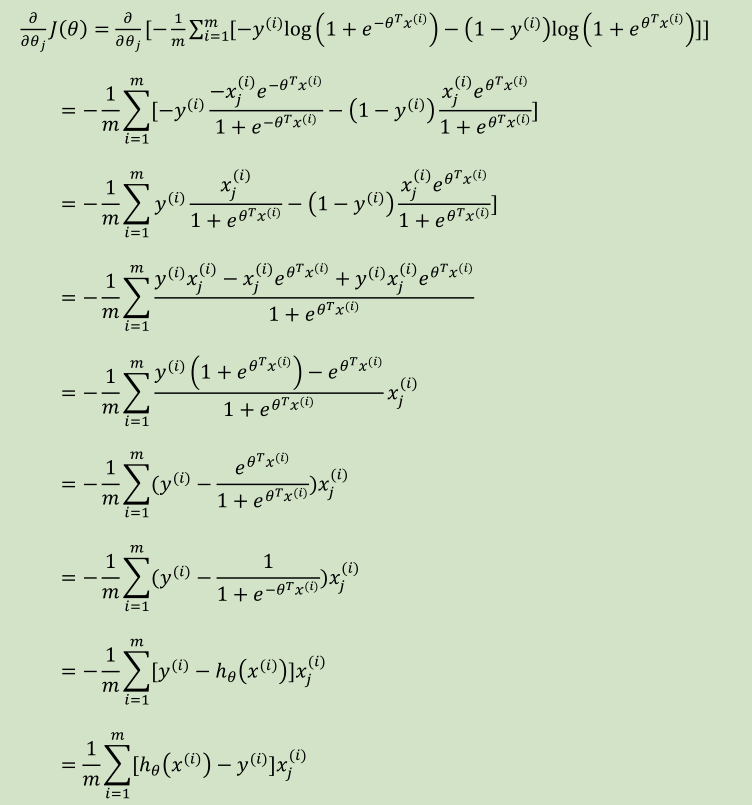
return np.sum(first - second) / (len(X))

梯度下降算法



#### 代价函数推导





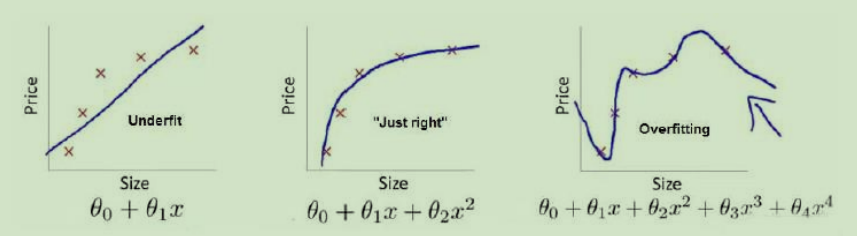
虽然得到的梯度下降算法表面上与线性回归是一样的，但这里的



所以，实际上是不一样的。

此外，进行特征缩放是很有必要的

## 六、正则化



出现了过拟合的问题，我们会采取两种办法：

* 丢弃无用特征， 或者使用模型选择的算法来改善。
* 正则化，保留所有参数，但是减小参数的大小。如第三个overfitting的图，减小所有多项式系数，模型表现好很多。

### 代价函数

图三的模型是，正是高次项导致了过拟合的产生，如果能让高次项的系数接近0的话，就可以很好拟合数了。所以目标就是一定程度上减小这些参数的值，这就是正则化的基本方法。

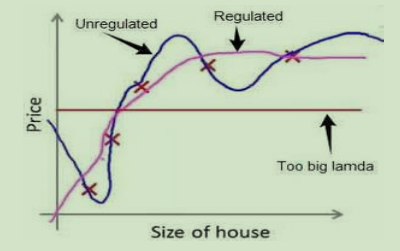
这里决定要减小后两个高阶项的系数，方法就是修改代价函数，对后两个系数进行一点惩罚。修改后的代价函数如下：



在我们不知道那些项是高阶项或者说无法指明要减小哪些参数的时候，可以对所有的参数进行惩罚：



其中λ称为正则化参数。注意：θ0 是常数项，我们不对它进行惩罚。经过正则化的效果图可能如下所示

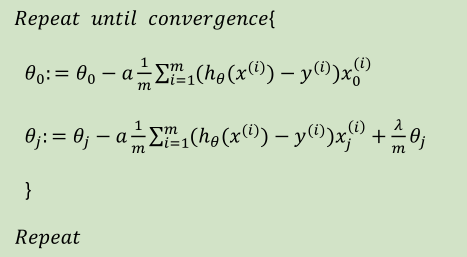


### 正则化线性回归

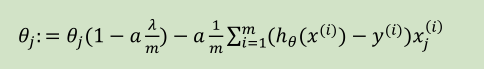
正则化线性回归的代价函数



算法

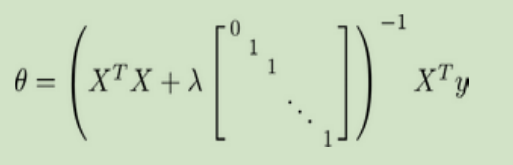


对第二式调整



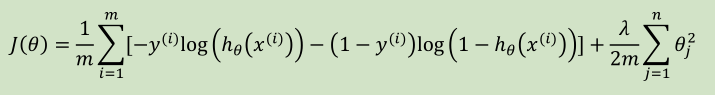
可以看出，正则化线性回归的梯度下降算法的变化在于， 每次都在原有更新规则下令θ减小一个额外的值。

### 正规方程正则化



图中矩阵尺寸为(n+1)\*(n+1)

### 逻辑回归模型正则化



Python代码

import numpy as np

def costReg(theta, X, y, learningRate):

theta = np.matrix(theta)

X = np.matrix(X)

y = np.matrix(y)

first = np.multiply(-y, np.log(sigmoid(X\*theta.T)))

second = np.multiply((1 - y), np.log(1 - sigmoid(X\*theta.T)))

reg = (learningRate / (2 \* len(X))\* np.sum(np.power(theta[:,1:the

ta.shape[1]],2))

return np.sum(first - second) / (len(X)) + reg

### 总结

总结来讲，正则化就是通过减小参数值，防止某些权值对网络影响过大而造成过拟合。

## 七、神经网络

### 1、非线性假设

线性回归和逻辑回归都有一个缺点，当特征数量变多时，计算负荷会特别大。

假设我们有100个特征变量，我们希望用这100个特征来构建一个非线性的多项式回归模型，即便我们只采用两两特征的组合，(𝑦 1 𝑦 2 +𝑦 1 𝑦 3 + 𝑦 1 𝑦 4 +...+𝑦 2 𝑦 3 + 𝑦 2 𝑦 4 +...+𝑦 99 𝑦 100 )，也有5000个特征，这对逻辑回归来说太多了。

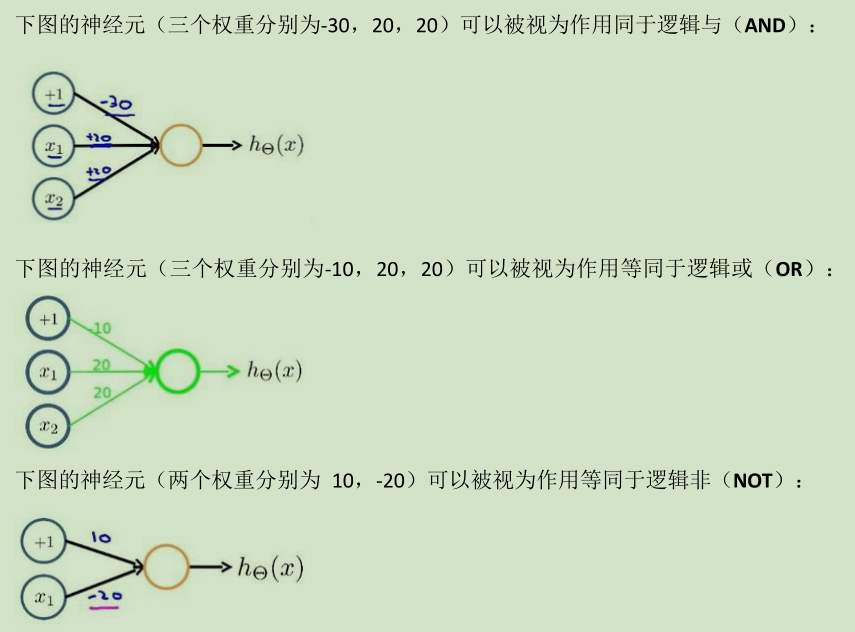
### 2、神经元和大脑

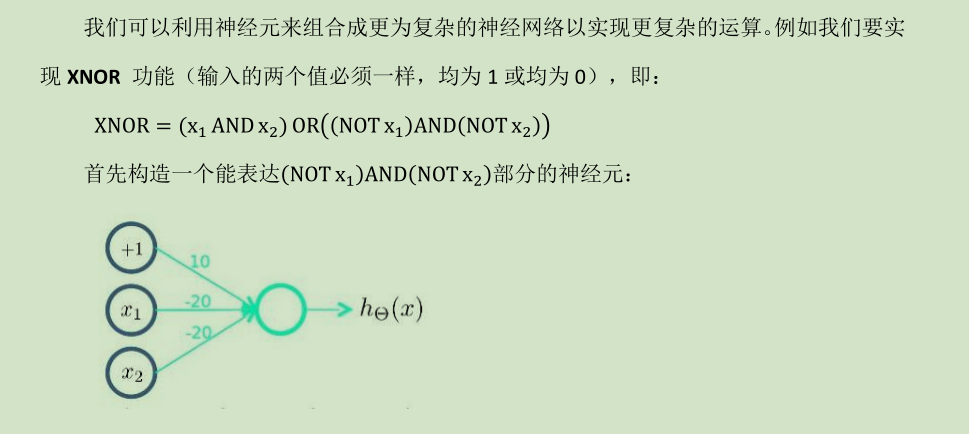
动物的大脑可以胜任多种工作。比如把动物的听觉神经到大脑的连接切断，把视神经连接到这部分，结果表明这部分听觉皮层学会了处理图像信息。把摄像机捕捉到的灰度图像通过舌头发送给大脑，那舌头也学会了看路。

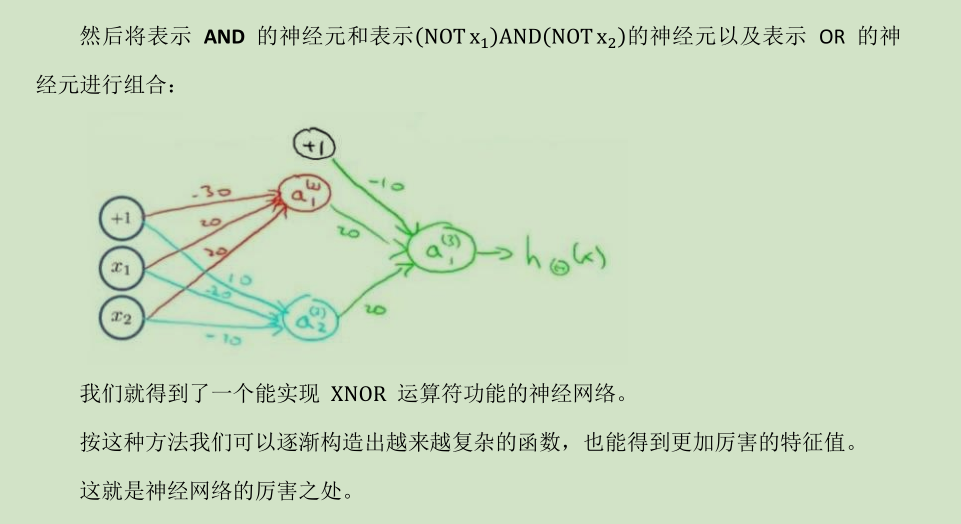
凡此种种，不一而足。

### 3、感知机

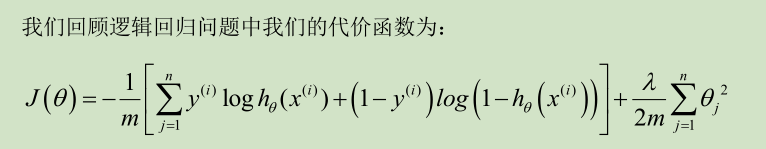
刚能明白，感知机是简化版本的神经网络，它能够以简单的线性神经元描述非线性问题。



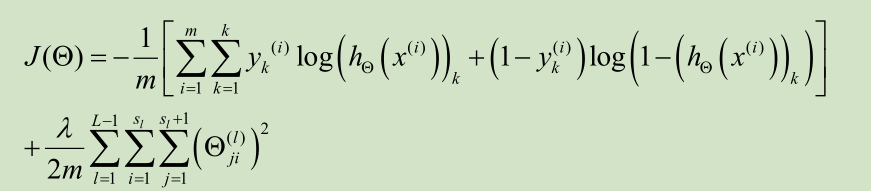


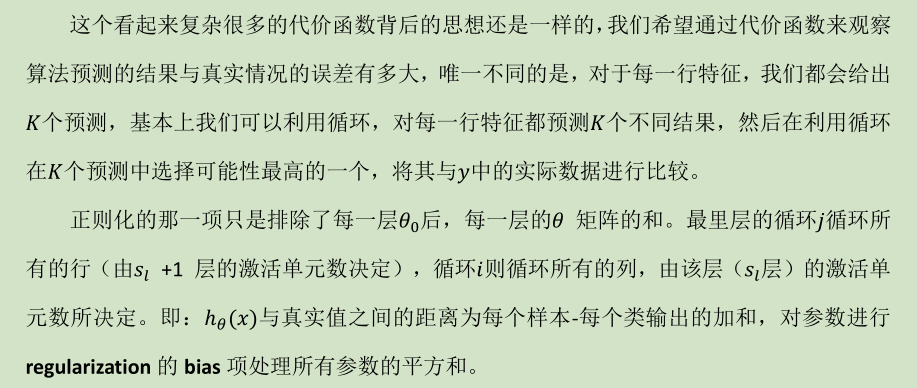


## 八、反向传播









这部分代码到uda的课内代码实现吧

## 九、应用机器学习的建议

总结一下

欠拟合/偏差，增加模型复杂程度

* + 获得更多特征，即输入加入更多特征项
  + 增加多项式特征，即及增加隐藏层（提高复杂度）
  + 减小正则化

过拟合/方差，说明模型过于复杂了，要减小模型复杂程度

* + - 获得更多数据
    - 减少特征数量，即减少输入x的数量
    - 增加正则化程度

关于正则化：

增加数据量：如果数据量小而模型过于复杂，那么模型中的参数会很轻松记住这些数据，因此模型泛化能力很差，所以增加数据量是有用的。

关于特征数量，只要是对模型有用的特征，个人感觉不要剔除，采用别的方法就好了。

## 十、机器学习系统设计

第一步，先写出一个模型训练。然后根据表现，选择提升模型表现的思路，因为你并不能提前知道你是否需要复杂的特征变量，或者你是否需要更多的数据，还是别的什么。这种理念是：我们必须用证据来领导我们的决策，怎样分配自己的时间来优化算法，而不是仅仅凭直觉，凭直觉得出的东西一般总是错误的。

### 1、误差分析

构建一个学习算法的推荐方法为：

1. 从一个简单的能快速实现的算法开始，实现该算法并用交叉验证集数据测试这个算法

2. 绘制学习曲线，决定是增加更多数据，或者添加更多特征，还是其他选择

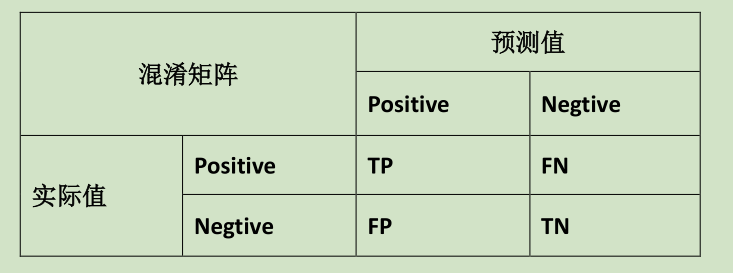
3. 进行误差分析：人工检查交叉验证集中我们算法中产生预测误差的实例，看看这些实例是否有某种系统化的趋势。

总结一下，当你在研究一个新的机器学习问题时，我总是推荐你实现一个较为简单快速、即便不是那么完美的算法。我几乎从未见过人们这样做。大家经常干的事情是：花费大量的时间在构造算法上，构造他们以为的简单的方法。因此，不要担心你的算法太简单，或者太不完美，而是尽可能快地实现你的算法。当你有了初始的实现之后，它会变成一个非常有力的工具，来帮助你决定下一步的做法。因为我们可以先看看算法造成的错误，通过误差分析，来看看他犯了什么错，然后来决定优化的方式。另一件事是：假设你有了一个快速而不完美的算法实现，又有一个数值的评估数据，这会帮助你尝试新的想法，快速地发现你尝试的这些想法是否能够提高算法的表现，从而你会更快地做出决定，在算法中放弃什么，吸收什么。

### 2、类偏斜的误差度量—查准率和查全率

偏斜类（skewed classes）表现为我们的训练集中有非常多的同一种类的实例，只有很少或没有其他类的实例。

一个二分类问题：



precision查准率=TP/(TP+FP)。例，在所有我们预测有恶性肿瘤的病人中，实际上有恶性肿瘤的病人的百分比，越高越好。

recall查全率=TP/(TP+FN)。例，在所有实际上有恶性肿瘤的病人中，成功预测有恶性肿瘤的病人的百分比，越高越好。

如果根据查准率和查全率来选择模型，一般会用的评分机制是Fscore



该值越高越好。

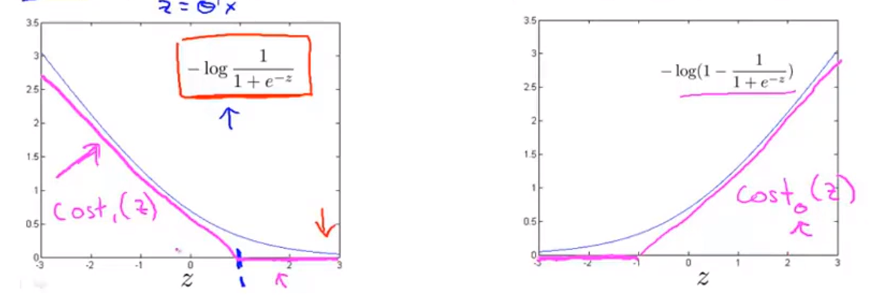
3、机器学习数据

It’s not who has the best algorithm win, it’s who has the most data.

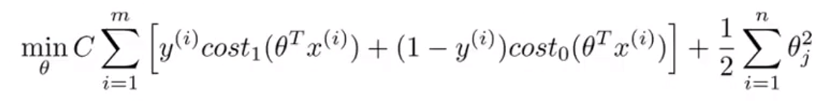
## 十一、支持向量机

### 优化目标

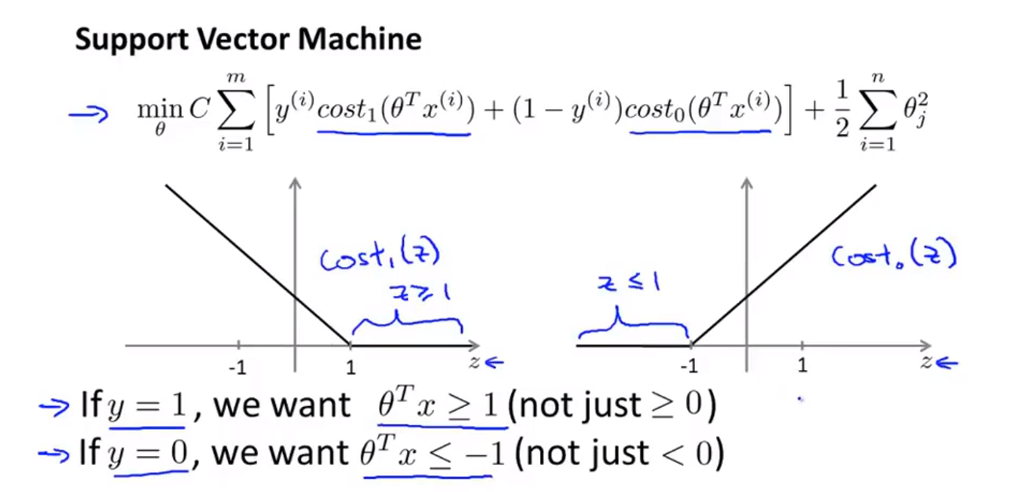
损失函数图



损失函数



### 大间隔的直观理解



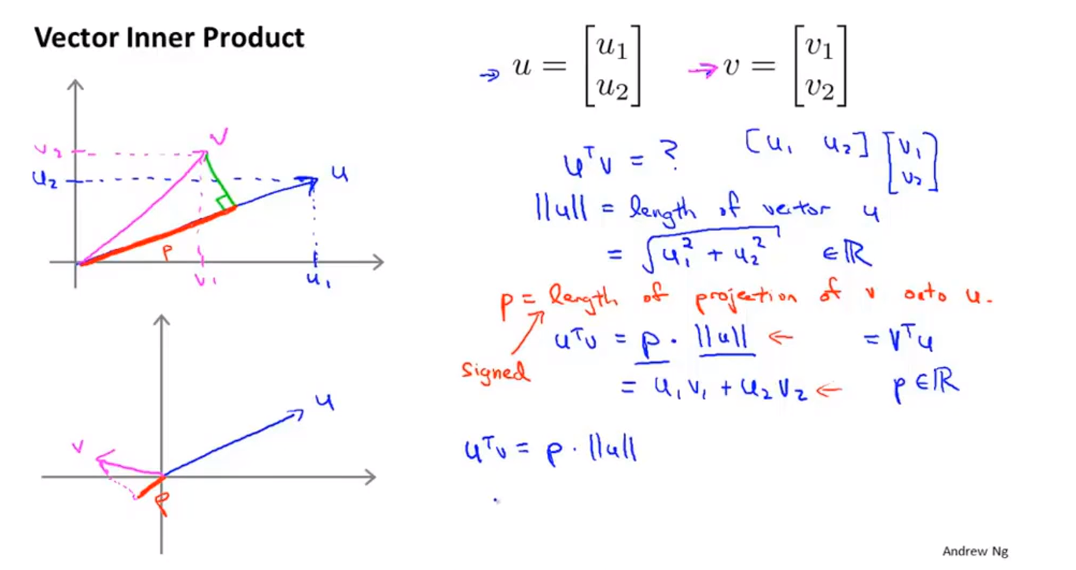
相比于逻辑回归，

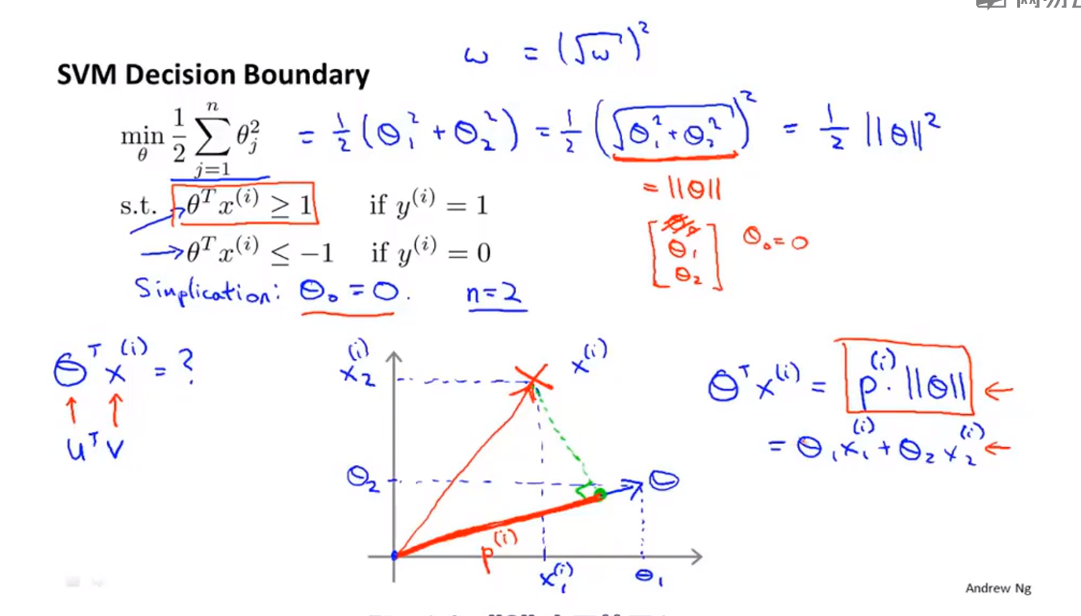
C较大时，算法注重拟合数据，低偏差，但是又过拟合风险，容易高方差

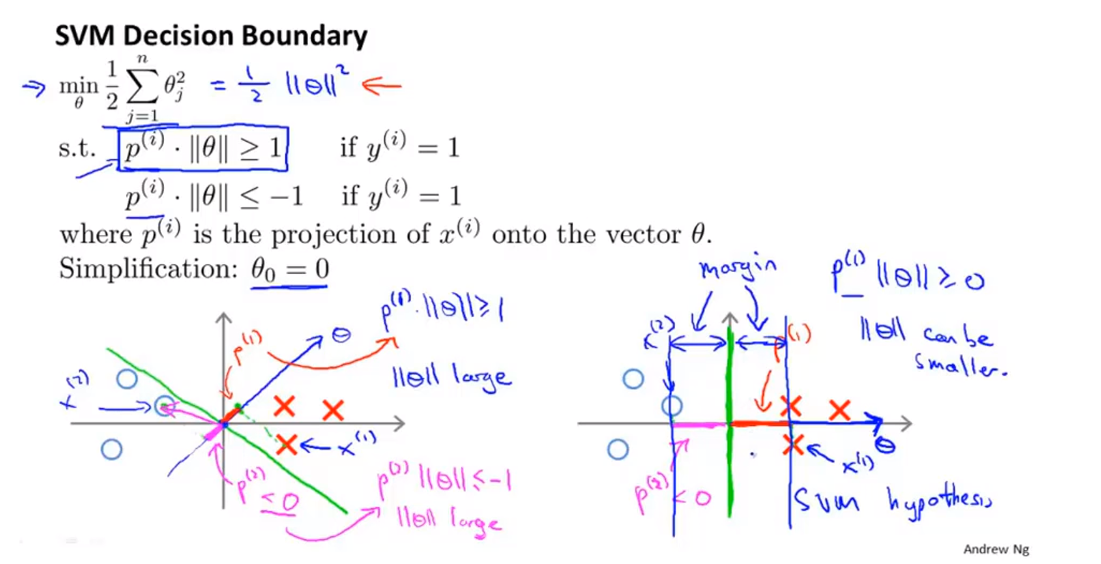
C较小时，算法注重正则化，泛化能力较好，低方差，但是容易高偏差。

### 大间隔的数学原理

向量内积的几何表示

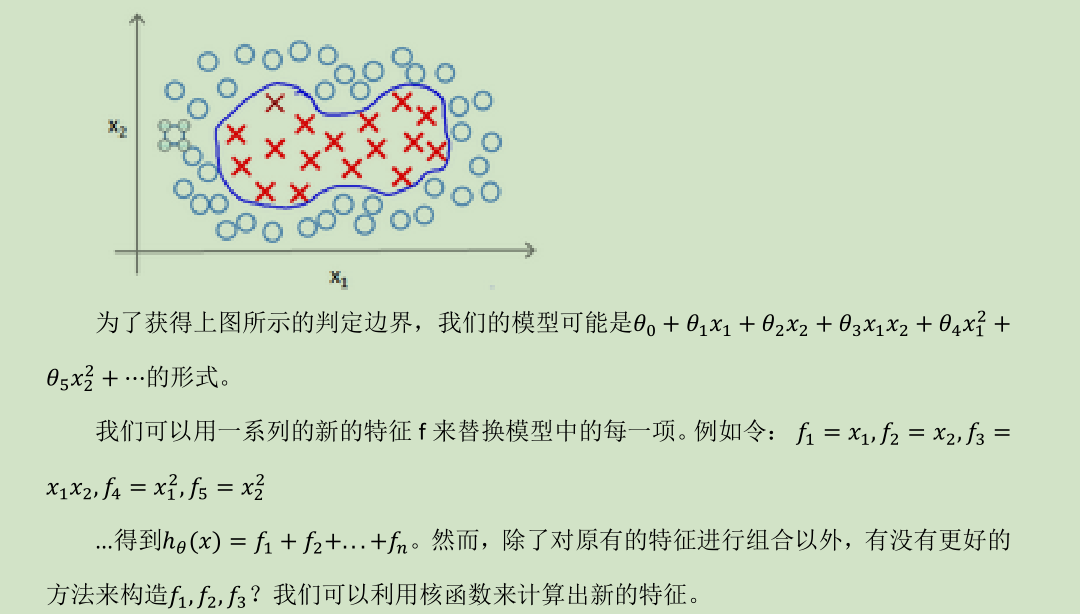


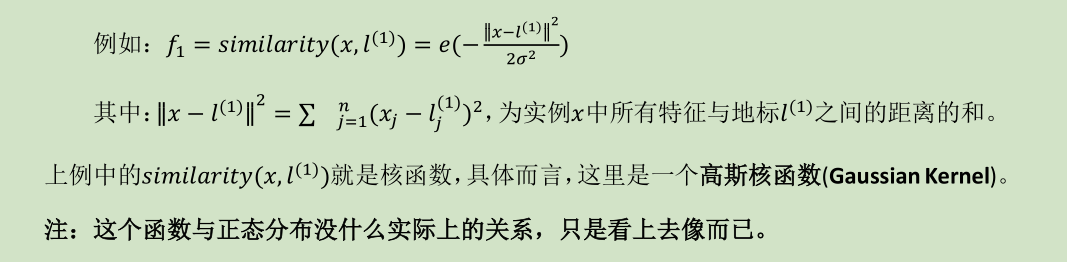




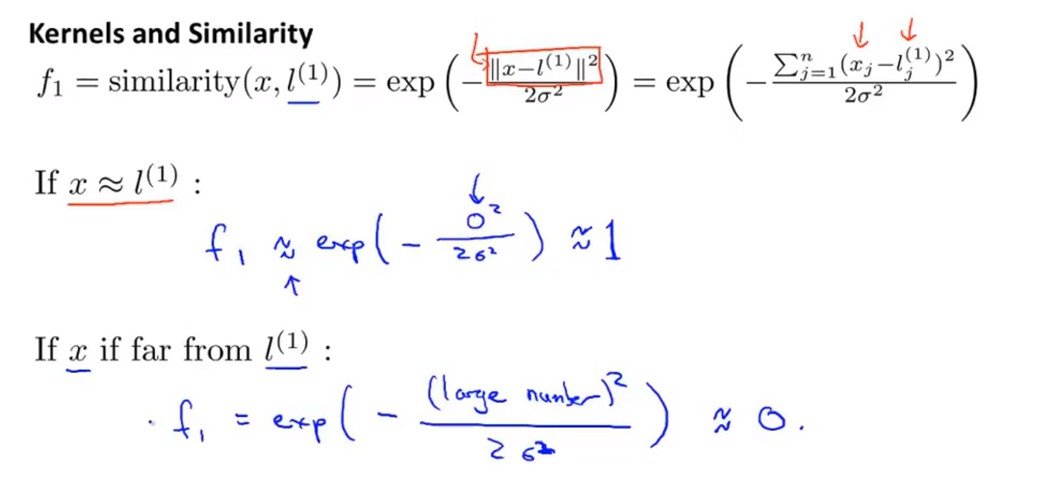
### 核函数

核函数即用来解决多项式回归模型有过多高阶项，导致运算量爆炸的问题。

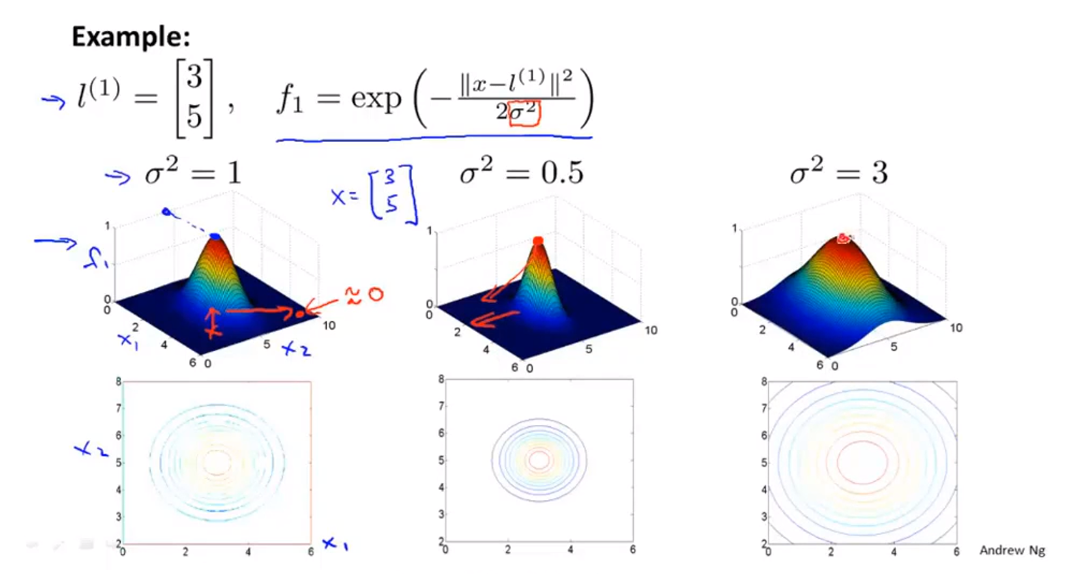




核函数的表现如下：



参数σ的影响



可以看到

σ较大的情况，f平滑，不容易过拟合，会高偏差，低方差。

σ较小的情况，f陡峭，容易完美拟合数据，容易低偏差，高方差。

**总结来讲，支持向量机就是先构造一个核函数，核函数的数量就等于训练样本的数量。然后基于这m个核函数构造一个线性回归的模型，进行回归训练。**不过这个模型的损失函数跟logits的损失函数有一点点的不同，在计算上也进行了一些速度上的优化。

#### 线性核函数

不使用核函数的svm也称为使用线性核函数。其核函数如下



线性核，主要用于线性可分的情况，我们可以看到特征空间到输入空间的维度是一样的，其参数少速度快，对于线性可分数据，其分类效果很理想，因此我们通常首先尝试用线性核函数来做分类，看看效果如何，如果不行再换别的

#### 多项式核函数



多项式核函数可以实现将低维的输入空间映射到高纬的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算

#### 高斯（RBF）核函数



高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

#### sigmoid核函数



采用sigmoid核函数，支持向量机实现的就是一种多层神经网络。

#### 参数调整

相比于逻辑回归，

C较大时，算法注重拟合数据，低偏差，但是又过拟合风险，容易高方差

C较小时，算法注重正则化，泛化能力较好，低方差，但是容易高偏差。

如果选择高斯核

σ较大的情况，f平滑，不容易过拟合，会高偏差，低方差。

σ较小的情况，f陡峭，容易完美拟合数据，容易低偏差，高方差。

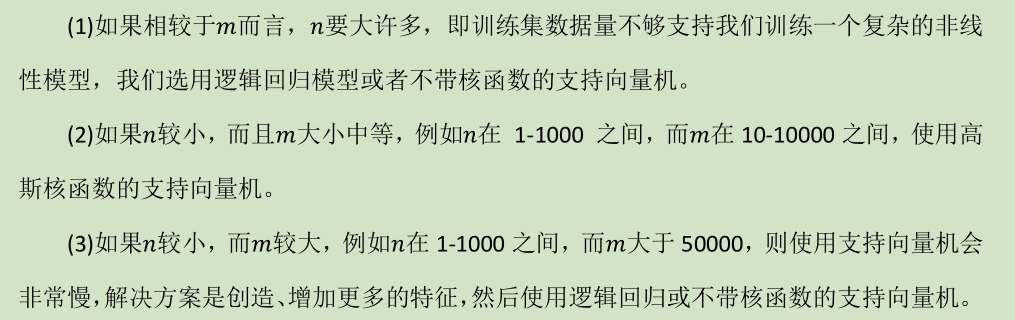
对于sklearn的svc的gamma值，

Gamma越大，越容易过拟合。

#### 如何选择

因此，在选用核函数的时候，如果我们对我们的数据有一定的先验知识，就利用先验来选择符合数据分布的核函数；如果不知道的话，通常使用交叉验证的方法，来试用不同的核函数，误差最下的即为效果最好的核函数，或者也可以将多个核函数结合起来，形成混合核函数。

* 如果特征的数量大到和样本数量差不多，则选用LR或者线性核的SVM；
* 如果特征的数量小，样本的数量正常，则选用SVM+高斯核函数；
* 如果特征的数量小，而样本的数量很大，则需要手工添加一些特征从而变成第一种情况。



### TIPS

值得一提的是，神经网络在以上三种情况下都可能会有较好的表现，但是训练神经网络可能非常慢，选择支持向量机的原因主要在于它的代价函数是凸函数，不存在局部最小值。

逻辑回归和不带核函数的支持向量机它们都是非常相似的算法，不管是逻辑回归还是不带核函数的 SVM，通常都会做相似的事情，并给出相似的结果。但是根据你实现的情况，其中一个可能会比另一个更加有效。

与神经网络相比，对于许多这样的问题，神经网络训练起来可能会特别慢，但是如果你有一个非常好的 SVM 实现包，它可能会运行得比较快比神经网络快很多，尽管我们在此之前没有展示，但是事实证明，SVM 具有的优化问题，是一种凸优化问题。因此，好的 SVM优化软件包总是会找到全局最小值，或者接近它的值。对于 SVM 你不需要担心局部最优。（这个时候GPU还不流行？）

其实，相比于选择哪个模型，更重要的是你有多少数据，你有多熟练是否擅长做误差分析和排除学习算法，指出如何设定新的特征变量和找出其他能决定你学习算法的变量等方面