目录

[1 降维 5](#_Toc524119756)

[1.1 PCA 6](#_Toc524119757)

[1.2 LDA 9](#_Toc524119758)

[2常见算法 10](#_Toc524119759)

[LR和SVM比较 10](#_Toc524119760)

[2.1 LR 10](#_Toc524119761)

[2.1.1 LR处理的特征是离散的还是连续的？【网易考拉】 10](#_Toc524119762)

[2.1.2 sigmoid函数。这个函数有什么优点和缺点？为什么不用其他函数？ 11](#_Toc524119763)

[2.1.3优化技巧 11](#_Toc524119764)

[为什么用极大似然估计参数 11](#_Toc524119765)

[3. LR 损失函数为什么用极大似然函数？ 12](#_Toc524119766)

[LR训练过程中如果有的特征高度相关会造成什么影响？ 12](#_Toc524119767)

[2.1.6逻辑回归和最大熵模型的相似点和不同点 12](#_Toc524119768)

[2.2 SVM 12](#_Toc524119769)

[ VC维的理解 16](#_Toc524119770)

[SVM和LR比较 16](#_Toc524119771)

[2.3 决策树 17](#_Toc524119772)

[2.3.1 ID3 使用信息增益作为选择特征的准则； 17](#_Toc524119773)

[2.3.2 C4.5 使用信息增益比作为选择特征的准则； 17](#_Toc524119774)

[2.3.3 CART 使用 Gini 指数作为选择特征的准则。 17](#_Toc524119775)

[2.4 k近邻 18](#_Toc524119776)

[2.5朴素贝叶斯 18](#_Toc524119777)

[2.6 FM 20](#_Toc524119778)

[2.7 EM 21](#_Toc524119779)

[2.8 hmm 23](#_Toc524119780)

[HMM的三个问题 25](#_Toc524119781)

[维特比算法 26](#_Toc524119782)

[2.8 判别式模型和生成式模型 27](#_Toc524119783)

[2.8.1 判别模型是直接对P(Y|X)建模， 27](#_Toc524119784)

[2.8.2 生成式模型 27](#_Toc524119785)

[2.9 LDA隐含狄利克雷分布 27](#_Toc524119786)

[3.模型融合方法 28](#_Toc524119787)

[3.1Boosting和bagging的原理以及异同点 28](#_Toc524119788)

[3.2GBDT 29](#_Toc524119789)

[3.2.1 GBDT如何处理样本不均衡 30](#_Toc524119790)

[3.2.2 GBDT 并行 30](#_Toc524119791)

[3.2.3 AdaBoost V.S. GBDT 30](#_Toc524119792)

[3.3 RF 31](#_Toc524119793)

[RF处理缺失值 31](#_Toc524119794)

[参数 **错误!未定义书签。**](#_Toc524119795)

[3.4 Xgboost的原理 32](#_Toc524119796)

[处理非平衡数据集： 32](#_Toc524119797)

[3.5 Lgbm的原理 33](#_Toc524119798)

[3.6 模型比较 34](#_Toc524119799)

[Lgb，XGB的区别和联系，并行是如何并行的 34](#_Toc524119800)

[GBDT和随机森林的区别 34](#_Toc524119801)

[GBDT和xgboost的区别 34](#_Toc524119802)

[4、深度学习 35](#_Toc524119803)

[权重初始化的方法 35](#_Toc524119804)

[优化方法 36](#_Toc524119805)

[牛顿法 38](#_Toc524119806)

[拟牛顿法 38](#_Toc524119807)

[Momentum 动量 38](#_Toc524119808)

[Adagrad 39](#_Toc524119809)

[RMSprop 39](#_Toc524119810)

[Adadelta 40](#_Toc524119811)

[Adam 40](#_Toc524119812)

[介绍CNN 41](#_Toc524119813)

[介绍RNN 42](#_Toc524119814)

[CNN和RNN的区别？ 43](#_Toc524119815)

[GRU对LSTM做了两个大改动 44](#_Toc524119816)

[神经网络overfiting、Regulation 45](#_Toc524119817)

[Batch Normalization 45](#_Toc524119818)

[6. 评价 46](#_Toc524119819)

[过拟合和欠拟合 46](#_Toc524119820)

[过拟合如何解决 **错误!未定义书签。**](#_Toc524119821)

[欠拟合的解决 **错误!未定义书签。**](#_Toc524119822)

[评价指标 48](#_Toc524119823)

[Logloss 48](#_Toc524119824)

[AUC和ROC 48](#_Toc524119825)

[F1 score 50](#_Toc524119826)

[精确率、召回率 50](#_Toc524119827)

[比较 50](#_Toc524119828)

[Logloss和roc、f1比较 50](#_Toc524119829)

[正则化 51](#_Toc524119830)

[特征选择 52](#_Toc524119831)

[5. 聚类 52](#_Toc524119832)

[5.1 k-means算法 52](#_Toc524119833)

[时间复杂度 53](#_Toc524119834)

[选择聚类个数 K的确定 53](#_Toc524119835)

[5.2基于密度的聚类 53](#_Toc524119836)

[5.2.1 DBSCAN 53](#_Toc524119837)

[5.2.2 OPTICS 58](#_Toc524119838)

[7. NLP 61](#_Toc524119839)

[CNN和RNN的比较（文本中） 62](#_Toc524119840)

[8. 推荐 63](#_Toc524119841)

[9. CTR 63](#_Toc524119842)

[9.1基本概念 63](#_Toc524119843)

[9.2 模型 64](#_Toc524119844)

[1. LR 海量高纬度 离散特征 64](#_Toc524119845)

[2. GBDT 65](#_Toc524119846)

[3. FM 与 FFM 65](#_Toc524119847)

[4. GBDT+(LR, FM, FFM) 66](#_Toc524119848)

[5.MLR 66](#_Toc524119849)

[6.DNN 67](#_Toc524119850)

[7. FNN 68](#_Toc524119851)

[8. DeepFM 68](#_Toc524119852)

[9.wide&deep 69](#_Toc524119853)

[9.3 比赛 70](#_Toc524119854)

[1. 腾讯广告算法大赛 70](#_Toc524119855)

[10. 面经 72](#_Toc524119856)

[360 72](#_Toc524119857)

[最小编辑距离的问题 72](#_Toc524119858)

[如何实现LRU，用到哪些数据结构 72](#_Toc524119859)

# 1 降维

降维的标准为：样本点到这个超平面的距离足够近,或者说样本点在这个超平面上的投影能尽可能的分开。

## 1.1 PCA

PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，可用于提取数据的主要特征分量，常用于高维数据的降维。

PCA本质上是将方差最大的方向作为主要特征，并且在各个正交方向上将数据“离相关”，也就是让它们在不同正交方向上没有相关性。

|  |
| --- |
| **PCA的缺点** |
| 1.依赖于线性假设，数据内在的结构未必是简单的线性  2.PCA是正交的，如果在非正交方向上存在几个方差较大的方向，PCA的效果就大打折扣了。  3.PCA让方差尽量大可能会丢失信息 |
| **为什么要标准化？** |
| PCA通常是用于高维数据的降维，它可以将原来高维的数据投影到某个低维的空间上并使得其方差尽量大。如果数据其中某一特征（矩阵的某一列）的数值特别大，那么它在整个误差计算的比重上就很大，那么可以想象在投影到低维空间之后，为了使低秩分解逼近原数据，整个投影会去努力逼近最大的那一个特征，而忽略数值比较小的特征。因为在建模前我们并不知道每个特征的重要性，这很可能导致了大量的信息缺失。为了“公平”起见，防止过分捕捉某些数值大的特征，我们会对每个特征先进行标准化处理，使得它们的大小都在相同的范围内，然后再进行PCA。  此外，从计算的角度讲，PCA前对数据标准化还有另外一个好处。因为PCA通常是数值近似分解，而非求特征值、奇异值得到解析解，所以当我们使用梯度下降等算法进行PCA的时候，我们最好先要对数据进行标准化，这是有利于梯度下降法的收敛。 |

#### PCA为什么要用SVD求解。

在[主成分分析（PCA）原理总结](http://www.cnblogs.com/pinard/p/6239403.html)中，我们讲到要用PCA降维，需要找到样本协方差矩阵XTX的最大的d个特征向量，然后用这最大的d个特征向量张成的矩阵来做低维投影降维。可以看出，在这个过程中需要先求出协方差矩阵XTX，当样本数多样本特征数也多的时候，这个计算量是很大的。

　　注意到我们的SVD也可以得到协方差矩阵XTX最大的d个特征向量张成的矩阵，但是SVD有个好处，有一些SVD的实现算法可以不求先求出协方差矩阵XTX，也能求出我们的右奇异矩阵V。也就是说，我们的PCA算法可以不用做特征分解，而是做SVD来完成。这个方法在样本量很大的时候很有效。实际上，scikit-learn的PCA算法的背后真正的实现就是用的SVD，而不是我们认为的暴力特征分解。

　　另一方面，注意到PCA仅仅使用了我们SVD的右奇异矩阵，没有使用左奇异矩阵，那么左奇异矩阵有什么用呢？

　　假设我们的样本是m×n的矩阵X，如果我们通过SVD找到了矩阵XXT最大的d个特征向量张成的m×d维矩阵U，则我们如果进行如下处理：

X′d×n=UTd×mXm×n

可以得到一个d×n的矩阵X’,这个矩阵和我们原来的m×n维样本矩阵X相比，行数从m减到了k，可见对行数进行了压缩。也就是说，左奇异矩阵可以用于行数的压缩。相对的，右奇异矩阵可以用于列数即特征维度的压缩，也就是我们的PCA降维。

**左奇异矩阵可以用于行数压缩。**

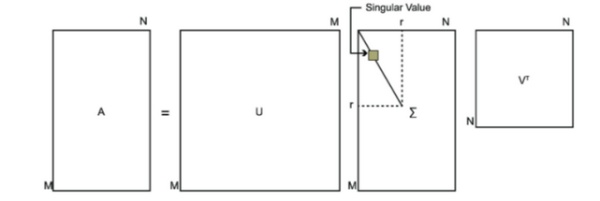
**右奇异矩阵可以用于列数即特征维度的压缩，即PCA降维**

#### SVD

SVD也是对矩阵进行分解，但是和特征分解不同，SVD并不要求要分解的矩阵为方阵。假设我们的矩阵A是一个m×n的矩阵，那么我们定义矩阵A的SVD为：

https://pic2.zhimg.com/v2-a71a3b4be58eaea23992595d495c55ce_b.jpg

其中 U是一个m ╳ m的矩阵,是一个m ╳ n的矩阵，除了主对角线上的元素以外全为0，主对角线上的每个元素都称为奇异值，V是一个 n ╳ n的矩阵。 U和 V都是酉矩阵，即满足。下图可以很形象的看出上面SVD的定义：



那么我们如何求出SVD分解后的U,Σ,V这三个矩阵呢？

如果我们将A的转置和A做矩阵乘法，那么会得到n×n的一个方阵 ATA。既然 ATA是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式：

https://pic2.zhimg.com/v2-fda9b3c4f938e1c71f27d78746477aca_b.jpg

这样我们就可以得到矩阵 ATA的n个特征值和对应的n个特征向量v了。将ATA的所有特征向量张成一个n×n的矩阵V，就是我们SVD公式里面的V矩阵了。一般我们将V中的每个特征向量叫做A的右奇异向量。

如果我们将A和A的转置做矩阵乘法，那么会得到m×m的一个方阵 AAT。既然 AAT是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式：

https://pic1.zhimg.com/v2-b94718895c711a19db641cac4064eae3_b.jpg

这样我们就可以得到矩阵 AAT的m个特征值和对应的m个特征向量u了。将 AAT的所有特征向量张成一个m×m的矩阵U，就是我们SVD公式里面的U矩阵了。一般我们将U中的每个特征向量叫做A的左奇异向量。

U和V我们都求出来了，现在就剩下奇异值矩阵Σ没有求出了.

由于Σ除了对角线上是奇异值其他位置都是0，那我们只需要求出每个奇异值σ就可以了。

我们注意到:

https://pic4.zhimg.com/v2-eab35f0f8896ebe2dbf64d3c0b2bb1da_b.jpg

这样我们可以求出我们的每个奇异值，进而求出奇异值矩阵Σ。

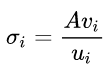
上面还有一个问题没有讲，就是我们说ATA的特征向量组成的就是我们SVD中的V矩阵，而AAT 的特征向量组成的就是我们SVD中的U矩阵，这有什么根据吗？这个其实很容易证明，我们以V矩阵的证明为例。

https://pic3.zhimg.com/v2-51a61b4e3b977ade92b970f486a4aef4_b.jpg

上式证明使用了 可以看出的ATA特征向量组成的的确就是我们SVD中的V矩阵。类似的方法可以得到AAT的特征向量组成的就是我们SVD中的U矩阵。

进一步我们还可以看出我们的特征值矩阵等于奇异值矩阵的平方，也就是说特征值和奇异值满足如下关系：

https://pic2.zhimg.com/v2-987750654ea2a7e2e929fa33661beea9_b.jpg

这样也就是说，我们可以不用来计算奇异值，也可以通过求出 ATA的特征值取平方根来求奇异值。

SVD的缺点是分解出来的矩阵解释性往往不强。

## 1.2 LDA

**核心思想**

希望在新的投影空间中，相同类之间的距离尽可能近，不同类之间距离尽可能的远

**LDA中相近相远的刻画**

* 同类相近。同一个类内部方差小
* 异类相远，不同类之间的均值的差的绝对值尽可能的大

# 2常见算法

## LR和SVM比较

LR就是把y=wx+b的结果放到sigmoid函数里，把sigmoid作为一个分类器，这样做的原因是sigmoid函数能把所有范围的值域控制在（0，1）区间内，然后我们把0.5作为一个分类的阈值，大于0.5的作为正类，小于0.5的作为负类。

SVM是一个利用超平面将数据隔开分类的问题，首先我们在max所有距离平面最近的点的margin，同时subject to y(wx+b)>0,意味着分类正确：

然后：我们可以的到最近点到平面的距离：

我们最后再用拉格朗日乘子式将subjuct to 条件转换成一个等式求解最后的w,b 然后求得最优超平面。我说SVM有很多kernel，kernel是把数据把低维映射到高维，因为有些数据在低维不可分，映射高维可以找到超平面去划分，更好准确。

最后说了LR比较方便计算，SVM 高维kernel计算复杂，但是准确。如果数据多，要求实时得到预测结果，用LR；如果数据不多，要求准确率，我选择SVM。

## 2.1 LR

线性模型—>sigmoid非线性变换—>最大似然—>交叉熵损失

**LR常见问题总结(重要)**：<https://zhuanlan.zhihu.com/p/34670728>

### 2.1.1 LR处理的特征是离散的还是连续的？【网易考拉】

（连续的）；离散化，会有什么影响吗，比如一个特征取值0-1，需要离散化吗？（又是一脸蒙蔽！！！离散化对于树模型来说，应该是比较好的，但是LR，应该不用吧，我没有用过，哎，菜啊）；好，我们想想树是怎么做的，CART对于连续特征，也是二分的，如果我们先做好离散这件事，是不是会更好一点，对于LR来说，也是一样，是会有一定提升的。（没有太理解，想想也是吧，一个特征挤在0-1，可能造成的区分性没有那么明显，而离散化之后，加强了特征对于数据的区分度，不知道这么理解对不对，有大佬的话，不吝赐教）

### 2.1.2 sigmoid函数。这个函数有什么优点和缺点？为什么不用其他函数？

p.s sigmoid函数导数的范围（0-1/4）

我们需要从GLM（广义线性模型）的角度理解logistic regression，GLM是一个比较大的集合，logistic regression和linear regression都属于GLM

GLM由两个因素决定：Y|X的分布、link function

**为什么用log损失，而不是平方损失**

* 最小二乘法得到的目标函数是非凸的，意味着我们的代价函数有许多局部最小值，这将影响梯度下降法寻找全局最小值。采用log作为损失函数，得到的代价函数是一个凸函数，并且没有局部最优值。
* 使用log损失的时候，梯度和sigmoid函数本身的梯度是无关的，而如果使用平方损失，梯度的更新和sigmoid很相关，此时梯度更新的速度很慢。

### 2.1.3优化技巧

* 随机梯度下降法，mini-batch随机梯度下降
* 共轭梯度，局部优化法和有限内存局部优化法（ng 吴）
  + 以上三种算法的优点：
  + 不需要手动的选择学习率a：通过内部的线性搜索尝试不同的学习率，并选择一个最好的
* 特征缩放，也可以加快速度
* 牛顿法求解，但是求hessian矩阵的逆特别复杂

### 为什么用极大似然估计参数

* 我们想让每一个样本的预测都要得到最大的概率，即将所有样本预测后的概率进行相乘都最大，就是极大似然的思想
* 对极大似然取对数以后，相当于对数损失，求解速度快

### 3. LR 损失函数为什么用极大似然函数？

1. 对极大似然函数取对数以后相当于对数损失函数，由上面 梯度更新 的公式可以看出，对数损失函数的训练求解参数的速度是比较快的，而且更新速度只和x，y有关，比较的稳定，
2. 为什么不用平方损失函数   
   如果使用平方损失函数，梯度更新的速度会和 sigmod 函数的梯度相关，sigmod 函数在定义域内的梯度都不大于0.25，导致训练速度会非常慢。   
   而且平方损失会导致损失函数是 theta 的非凸函数，不利于求解，因为非凸函数存在很多局部最优解。

### LR训练过程中如果有的特征高度相关会造成什么影响？

在损失函数最终收敛的情况下，没有影响。加入某个特征重复100次，特征的权重变成原来的1/100

### 2.1.6逻辑回归和最大熵模型的相似点和不同点

逻辑回归跟最大熵模型**没有本质差别**。逻辑回归是最大熵相应类别为**二类**时的特殊情况

**指数簇分布的最大熵**等价于其**指数形式的最大似然**。

**二项式**分布的最大熵解等价于二项式指数形式(**sigmoid**)的最大似然；   
**多项式**分布的最大熵等价于多项式分布指数形式(**softmax**)的最大似然。

## 2.2 SVM

* **为什么要引入对偶问题**

|  |
| --- |
| **减少算法复杂度** |
| 在原问题下，求解算法的复杂度与样本维度（等于权值w的维度）有关；  而在对偶问题下，求解算法的复杂度与样本数量（等于拉格朗日算子a的数量）有关。 |

因此，如果你是做线性分类，且样本维度低于样本数量的话，在原问题下求解就好了，Liblinear之类的线性SVM默认都是这样做的；

但如果你是做非线性分类，那就会涉及到升维（比如使用高斯核做核函数，其实是将样本升到无穷维），升维后的样本维度往往会远大于样本数量，此时显然在对偶问题下求解会更好。

本来的算法也可以求解 SVM，但是之所以要用**对偶问题**来求解，优点是：

**一是对偶问题往往更容易求解；**

**二是自然引入核函数，进而推广到非线性分类问题。**

对于线性不可分的问题，**使用核函数可以从原始空间映射到高纬空间，使得问题变得线性可分。核函数还可以使得在高维空间计算的内积在低维空间中通过一个函数来完成。**

线性可分问题的支持向量机学习方法，对现行不可分训练数据是不适用的。所以将间隔函数修改为软间隔，对于函数间隔，在其上加上一个松弛变量，使其满足大于等于 1。

**参数C的理解 ：**<https://www.zhihu.com/question/40217487?sort=created>

|  |
| --- |
| 你可以把这个参数C理解为调节优化方向中两个指标（间隔大小，分类准确度）偏好的权重。soft-margin SVM针对hard-margin SVM容易出现的过度拟合问题，适当放宽了margin的大小，容忍一些分类错误（violation），把这些样本当做噪声处理，本质上是间隔大小和噪声容忍度的一种trade-off，至于具体怎么trade-off，对哪个指标要求更高，那就体现在C这个参数上了。 |

* **如何解决非线性问题**（软间隔和核技巧）

**什么是软间隔？SVM有什么可调的参数呢？**（C）；C=0的情况下会出现什么问题（此时用硬间隔解决非线性，效果应该不会好，感觉这个回答不行，有大佬的话，不吝赐教啊）；核函数有哪些？对于一个问题，如何选取核函数？（问题比较简单的话，选取线性核，复杂的话，选取高斯核。其他核，不太了解，有大佬的话，不吝赐教）

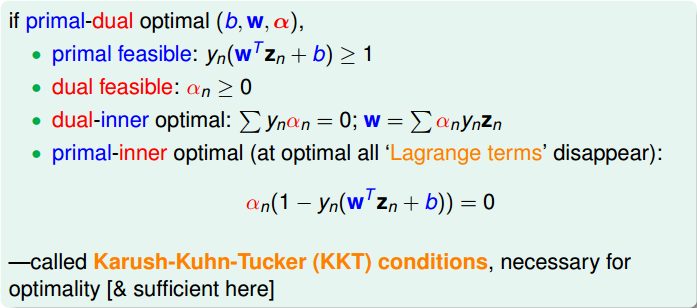
C>0，是一个常数，当C为无穷大的时候。迫使所有的样本均满足约束，(容易过拟合)等价于硬间隔。当C取有限值的时候，允许一些样本不满足约束。

SVM参数调节<https://www.cnblogs.com/pinard/p/6117515.html>

* **SVM怎么防止过拟合**

说了SVM里面的松弛变量。可以忍受异常值的存在。

* **SVM是通过最小间隔最大化寻找超平面；为什么么要最大化最小间隔呢？**
* **KKT条件用哪些，完整描述**



* **SVM核函数的选择？**

1. Linear核：主要用于线性可分的情形。参数少，速度快，对于一般数据，分类效果已经很理想了。

2. RBF核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数。

一般用线性核和高斯核，也就是Linear核与RBF核

需要注意的是需要对数据归一化处理，很多使用者忘了这个小细节

然后一般情况下RBF效果是不会差于Linear。但是时间上RBF会耗费更多

**下面是吴恩达的见解**：

1. 如果Feature的数量很大，跟样本数量差不多，这时候选用LR或者是Linear Kernel的SVM

当训练集不大，feature比较多的时候，用线性的核。因为多feature的情况下就已经可以给线性的核提供不错的variance去fit训练集。

2. 如果Feature的数量比较小，样本数量一般，不算大也不算小，选用SVM+Gaussian Kernel

情况2：当训练集相对可观，而feature比较少，用非线性的核。因为需要算法提供更多的variance去fit训练集。

3. 如果Feature的数量比较小，而样本数量很多，需要手工添加一些feature变成第一种情况

情况3：feature少，训练集非常大，用线性的核。因为非线性的核需要的计算量太大了。而庞大的训练集，本身就可以给非线性的核提供很好的分类效果。

核函数需要计算内积，两两样本都得算，所以样本过多的话时间消耗太大，很明显高斯核比线性核复杂的多

AndrewNg理论1：当数据量足够庞大时，feature足够多时，所有的分类算法最终的效果都差不多。也就是说，不管你选用什么样的核，在训练集够大的情况下都是然并卵。当然，就分类效果来说，非线性的比线性的核好一些。但线性的也能够有很不错的分类效果，而且计算量比非线性小，所以需要具体情况具体分析。

|  |  |  |
| --- | --- | --- |
| **核函数** | **优点** | **缺点** |
| **线性核** | 1.简单，安全  2.不涉及别的问题，所以可以设计特别的QP二次规划的解决办法  3.可以很容易的看出来machine怎么做分类的。哪些点重要 | 有限制，当数据不是线性可分的时候。 |
| **高斯核—**无限多维的转换 | 1.比线性核和多项式核更powerful  2.与多项式核比较参数少  3.只有一个参数，方便调参 | 1. 比较难解释  2. 计算慢  3. 可能过拟合 |
| **多项式核** | 计算有困难，很多参数要选择，比较困难 | |

* **核函数的条件：**

**自己定义的核函数：对称；对于任意的数据D，求出来的核矩阵K是半正定的**

* **SVM如何处理高维度特征(网易有道)**

svm不存在维度灾难。SVM 的最终决策函数只由少数的支持向量所确定,计算的复杂性取决于支持向量的数目,而不是样本空间的维数,这在某种意义上避免了“维数灾难”。

* **sklearn.svm包中的SVC(kernel=”linear“)和LinearSVC的区别**

1. LinearSVC使用的是平方hinge loss，SVC使用的是绝对值hinge loss （我们知道，绝对值hinge loss是非凸的，因而你不能用GD去优化，而平方hinge loss可以）
2. LinearSVC使用的是One-vs-All（也成One-vs-Rest）的优化方法，而SVC使用的是One-vs-One
3. 对于多分类问题，如果分类的数量是N，则LinearSVC适合N模型，而SVC适合N(N-1)/2模型

* VC维的理解：

对一个指标函数集，如果存在h 个样本能够被函数集中的函数按所有可能的2h种形式分开，则称函数集能够把h个样本打散；函数集的VC维就是它能打散的最大样本数目h。若对任意数目的样本都有函数能将它们打散，则函数集的VC维是无穷大，有界实函数的VC维可以通过用一定的阀值将它转化成指示函数来定义。

### SVM和LR比较

 Linear SVM和LR都是线性分类器

 Linear SVM和LR都是[判别模型](https://zhuanlan.zhihu.com/p/32655097)

 Linear SVM不直接依赖数据分布，分类平面不受一类点影响；LR则受所有数据点的影响，如果数据不同类别处于极其不平衡的状态, 一般需要先对数据做平衡处理。

 Linear SVM依赖数据表达的距离测度，所以需要对数据先做标准化；LR不受其影响

 Linear SVM依赖惩罚项的系数，实验中需要做[交叉验证](https://zhuanlan.zhihu.com/p/32627500)

 Linear SVM和LR的执行都会受到异常值的影响，其敏感程度而言，谁更好很难下明确结论。

 Linear SVM和LR损失函数不同, LR为logloss, SVM为hinge loss. 而SVM中的 称为**hinge loss**。

**SVM和LR的应用场景**

（在网上找的一些回答） n特征数量，m样本数

1. 如果Feature的数量很大，跟样本数量差不多，这时候选用LR或者是Linear Kernel的SVM
2. 如果Feature的数量比较小，样本数量一般，不算大也不算小，（n=1-1000，m=10-10000）选用SVM+Gaussian Kernel
3. 如果Feature的数量比较小，而样本数量很多，增加更多的feature然后使用LR算法或者不带核函数的SVM。 logistic regression适合需要得到一个分类概率的场景，SVM则没有分类概率。SVM无法预测概率，而LR可以，所以像点击率预估这种只能用LR

## 2.3 决策树

### 2.3.1 ID3 使用信息增益作为选择特征的准则；

信息增益越大，则意味着使用属性a来进行划分所获得的 “纯度提升” 越大 。也就是说，用属性a来划分训练集，得到的结果中纯度比较高。

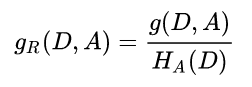
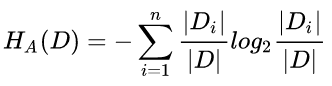
ID3 仅仅适用于二分类问题。ID3 仅仅能够处理离散属性

信息增益表示的是：**得知特征X的信息而使得类Y的信息的不确定性减少的程度。**特征A对训练数据集D的信息增益定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即

一般地，熵**H(Y)**与条件熵**H(Y|X)**之差称为互信息(mutual information).

### 2.3.2 C4.5 使用信息增益比作为选择特征的准则；

C4.5 克服了 ID3 仅仅能够处理离散属性的问题，以及信息增益偏向选择取值较多特征的问题，使用信息增益比来选择特征。**信息增益比 = 信息增益 / 划分前熵** 选择信息增益比最大的作为最优特征。公式：

其中，n是特征A取值的个数

C4.5 处理连续特征是先将特征取值排序，以连续两个值中间值作为划分标准。尝试每一种划分，并计算修正后的信息增益，选择信息增益最大的分裂点作为该属性的分裂点。

### 2.3.3 CART 使用 Gini 指数作为选择特征的准则。

写Gini系数，信息增益，信息增益率的公式

## 2.4 k近邻

K值的减小意味着模型变得复杂，容易发生过拟合。近似误差减小，估计误差增大，预测结果会对近邻的实例点非常敏感。

* **knn如何处理高维特征？**

knn和kmeans维度过高出现距离收敛，证明也没答上来，解决方案是降维、换距离度量。

## 2.5朴素贝叶斯

朴素贝叶斯是生成方法，也就是直接找出特征输出Y和特征X的联合分布P(X,Y)P(X,Y),然后用P(Y|X)=P(X,Y)/P(X)P(Y|X)=P(X,Y)/P(X)得出。

贝叶斯学派的思想可以概括为先验概率+数据=后验概率。也就是说我们在实际问题中需要得到的后验概率，可以通过先验概率和数据一起综合得到。

**流程**

1、计算Y的先验概率分布

2、分别计算第K个类别的第j维特征的第L个取值条件概率

3、对于实例X。计算后验概率。

**使用python进行文本分析流程**

从词向量计算概率

用到了朴素贝叶斯假设。

计算每个类别中的文档数目：

对每篇训练文档：

对每个类别：

如果词条出现在文档中—>增加该词条的计数值

增加所有词条的计数值

对每个类别:

对每个词条:

将该词条的数目除以总词条数目得到的条件概率（P(词条|类别)）

返回该文档属于每个类别的条件概率（P(类别|文档的所有词条)）

def \_trainNB0(trainMatrix, trainCategory):

"""

训练数据原版

:param trainMatrix: 文件单词矩阵 [[1,0,1,1,1....],[],[]...]

:param trainCategory: 文件对应的类别[0,1,1,0....]，列表长度等于单词矩阵数，其中的1代表对应的文件是侮辱性文件，0代表不是侮辱性矩阵

"""

# 文件数

numTrainDocs = len(trainMatrix)

# 单词数

numWords = len(trainMatrix[0])

# 侮辱性文件的出现概率，即trainCategory中所有的1的个数，

# 代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概率

pAbusive = sum(trainCategory) / float(numTrainDocs)

# 构造单词出现次数列表 (**针对每个类别**)

p0Num = zeros(numWords) # [0,0,0,.....]

p1Num = zeros(numWords) # [0,0,0,.....]

# 整个数据集单词出现总数

p0Denom = 0.0

p1Denom = 0.0

for i in range(numTrainDocs):

# 是否是侮辱性文件

if trainCategory[i] == 1:

# 如果是侮辱性文件，对侮辱性文件的向量进行加和

p1Num += trainMatrix[i] #[0,1,1,....] + [0,1,1,....]->[0,2,2,...]

# 对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数

p1Denom += sum(trainMatrix[i])

else:

p0Num += trainMatrix[i]

p0Denom += sum(trainMatrix[i])

# 类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)....]列表

# 即 在1类别下，每个单词出现的概率

p1Vect = log(p1Num / p1Denom)# [1,2,3,5]/90->[1/90,...]

# 类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)....]列表

# 即 在0类别下，每个单词出现的概率

p0Vect = log(p0Num / p0Denom)

return p0Vect, p1Vect, pAbusive

###### P(w|c1) \* P(c1) ，即贝叶斯准则的分子

p1 = sum(vec2Classify \* p1Vec) + log(pClass1)

p0 = sum(vec2Classify \* p0Vec) + log(1.0 - pClass1)

##### P(w|c0) \* P(c0) ，即贝叶斯准则的分子·

if p1 > p0:

return 1

else:

return 0

## 2.6 FM

FM（Factorization Machine）旨在解决稀疏数据下的特征组合问题。

根据paper的描述，FM有以下三个优点：   
1. **可以在非常稀疏的数据中进行合理的参数估计**   
2. **FM模型的时间复杂度是线性的**   
3. **FM是一个通用模型，它可以用于任何特征为实值的情况**

一般的线性模型为：



从上面的式子中看出，一般的线性模型没有考虑特征之间的关联。为了表述特征间的相关性，我们采用多项式模型。在多项式模型中，特征xi与xj的组合用xixj表示。为了简单起见，我们讨论二阶多项式模型。



组合特征的参数一共有 n(n−1)/2 个，任意两个参数都是独立的。然而，在数据稀疏性普遍存在的实际应用场景中，二次项参数的训练是很困难的。其原因是，每个参数 wij 的训练需要大量 xi 和 xj 都非零的样本；由于样本数据本来就比较稀疏，满足“xi 和 xj 都非零”的样本将会非常少。训练样本的不足，很容易导致参数 wij 不准确，最终将严重影响模型的性能。如何解决二次项参数的训练问题呢？矩阵分解提供了一种解决思路。W = VTV,因此FM的模型方程为

其中，vi 是第 i 维特征的隐向量，⟨⋅,⋅⟩ 代表向量点积。隐向量的长度为 k（k远远小于n），包含 k 个描述特征的因子。根据公式(2)，二次项的参数数量减少为 kn个，远少于多项式模型的参数数量。另外，参数因子化使得XhXi的参数和XiXj的参数不再是相互独立的，因此我们可以在样本稀疏的情况下相对合理地估计FM的二次项参数。具体来说，XhXi 和 XiXj 的系数分别为 ⟨Vh,Vi⟩ 和 ⟨Vi,Vj⟩，它们之间有共同项 Vi。也就是说，所有包含“Xi 的非零组合特征”（存在某个 j≠i，使得 XiXj≠0）的样本都可以用来学习隐向量 Vi，这很大程度上避免了数据稀疏性造成的影响。而在多项式模型中，Whi 和 Wij 是相互独立的。

直观上看，FM的复杂度是 O(kn2)。但是，通过公式(3)的等式，FM的二次项可以化简，其复杂度可以优化到 O(kn)[7]。由此可见，FM可以在线性时间对新样本作出预测。

## 2.7 EM

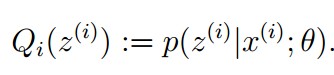
**EM算法（Expectation-maximization）：**

     期望最大算法是一种从不完全数据或有数据丢失的数据集（存在隐含变量）中求解概率模型参数的最大似然估计方法。

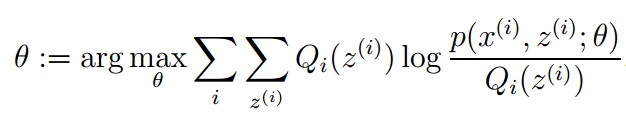
**EM的算法流程：**

初始化分布参数θ；

**重复以下步骤直到收敛：**

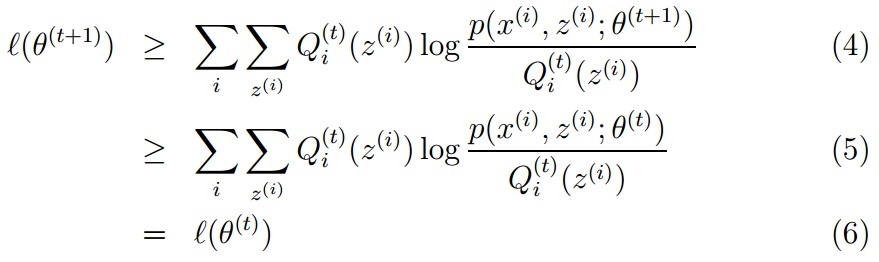
**E步骤：**根据参数初始值或上一次迭代的模型参数来计算出隐性变量的后验概率，其实就是隐性变量的期望。作为隐藏变量的现估计值：       

**M步骤：**将似然函数最大化以获得新的参数值：



这个不断的迭代，就可以得到使似然函数L(θ)最大化的参数θ了。那就得回答刚才的第二个问题了，它会收敛吗？

感性的说，因为下界不断提高，所以极大似然估计单调增加，那么最终我们会到达最大似然估计的最大值。理性分析的话，就会得到下面的东西：



具体如何证明的，看推导过程参考：Andrew Ng《The EM algorithm》

<http://www.cnblogs.com/jerrylead/archive/2011/04/06/2006936.html>

**Jensen不等式**

  回顾优化理论中的一些概念。设f是定义域为实数的函数，如果对于所有的实数x，[clip_image002](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615511363.png)，那么f是凸函数。当x是向量时，如果其hessian矩阵H是半正定的（[clip_image004](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615523937.png)），那么f是凸函数。如果[clip_image006](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615525640.png)或者[clip_image008](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615534069.png)，那么称f是严格凸函数。

      Jensen不等式表述如下：

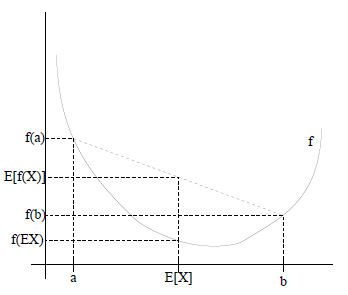
      如果f是凸函数，X是随机变量，那么

[clip_image010](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615545772.png)

      特别地，如果f是严格凸函数，那么[clip_image012](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615544725.png)当且仅当[clip_image014](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615549218.png)，也就是说X是常量。

      这里我们将[clip_image016](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615557299.png)简写为[clip_image018](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615555380.png)。

      如果用图表示会很清晰：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615564890.png)

   图中，实线f是凸函数，X是随机变量，有0.5的概率是a，有0.5的概率是b。（就像掷硬币一样）。X的期望值就是a和b的中值了，图中可以看到[clip_image010[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615563561.png)成立。

当f是（严格）凹函数当且仅当-f是（严格）凸函数。

  Jensen不等式应用于凹函数时，不等号方向反向，也就是[clip_image021](http://images.cnblogs.com/cnblogs_com/jerrylead/201104/201104061615576102.png)。

## 2.8 hmm

**HMM的确定**

**1. 初始概率分布**

 z1可能是状态1，状态2 ... 状态n，于是z1就有个N点分布：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Z1 | 状态1 | 状态2 | ... | 状态n |
| 概率 | P1 | P2 | ... | Pn |

即：Z1对应个n维的向量。

上面这个n维的向量就是初始概率分布，记做π。

**2. 状态转移矩阵**

 但Z2就不能简单的“同上”完事了，因为Z2和Z1不独立，所以Z2是状态1的概率有：Z1是状态1时Z2是状态1，Z1是状态2时Z2是状态1,..., Z1是状态n时Z2是状态1，于是就是下面的表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Z2  Z1 | 状态1 | 状态2 | ... | 状态n |
| 状态1 | P11 | P12 | ... | P1n |
| 状态2 | P21 | P22 | ... | P2n |
| ... | ... | ... | ... | ... |
| 状态n | Pn1 | Pn2 | ... | Pnn |

即：Z1->Z2对应个n\*n的矩阵。

同理：Zi -> Zi+1对应个n\*n的矩阵。

上面这些n\*n的矩阵被称为状态转移矩阵，用An\*n表示。

当然了，真要说的话，Zi -> Zi+1的状态转移矩阵一定都不一样，但在实际应用中一般将这些状态转移矩阵定为同一个，即：只有一个状态转移矩阵。 图1的第一行就搞定了，下面是第二行。

**3. 观测矩阵**

 如果对于zi有：状态1, 状态2, ..., 状态n，那zi的每一个状态都会从下面的m个观测中产生一个：观测1, 观测2, ..., 观测m，所以有如下矩阵：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X  Z | 观测1 | 观测2 | ... | 观测m |
| 状态1 | P11 | P12 | ... | P1m |
| 状态2 | P21 | P22 | ... | P2m |
| ... | ... | ... | ... | ... |
| 状态n | Pn1 | Pn2 | ... | Pnm |

这可以用一个n\*m的矩阵表示，也就是观测矩阵，记做Bn\*m。

  由于HMM用上面的π，A，B就可以描述了，于是我们就可以说：HMM由初始概率分布π、状态转移概率分布A以及观测概率分布B确定，为了方便表达，把A, B, π 用 λ 表示，即：

**λ = (A, B, π)**

### HMM的三个问题

|  |  |
| --- | --- |
| **概率计算问题** | **前向后向算法** |
| 给定模型和观测序列，计算此模型下观测序列出现的概率 | |
| **学习问题(HMM参数估计的方法)** | **Baum-welch算法** |
| 已知观测序列，估计模型的参数，使得在该模型下观测序列的概率最大 | |
| **预测问题/解码问题** | **Viterbi算法(一种动态规划算法)** |
| 给定观测序列，求最有可能的对应的状态序列 | |

1，知道HMM的参数 λ = (A, B, π) 和观测序列O = {o1,o2, ..., oT} ，如何计算模型 λ 下观测序列O出现的概率P(O | λ)。

2，HMM的参数如何确定？

  比如：对于刚才的中文分词的小例子。**初始概率分布π**好确定**：**是不是终结词的概率各是0.5。**观测矩阵B**也好确定：1/65535嘛 但**状态转移矩阵**怎么确定？我怎么知道下个词是终结词的概率是多少？

3，知道HMM的参数 λ = (A, B, π) 和观测序列O = {o1,o2, ..., oT}，如何计算给定观测序列条件概率P(I|O,  λ )最大的状态序列I，即：

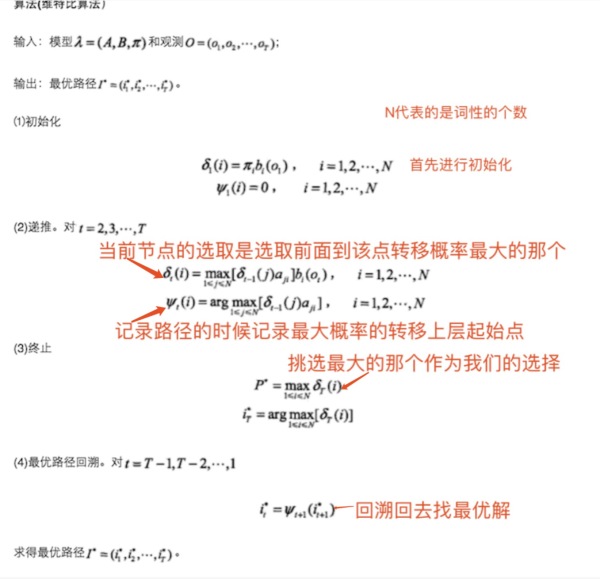
### 维特比算法

1，先从前向后推出一步步路径的最大可能，最终会得到一个从起点连接每一个终点的m条路径(假设有m个终点)。

2，确定终点之后在反过来选择前面的路径。

3，确定最优路径。

<http://f.dataguru.cn/thread-733767-1-1.html>



## 2.8 判别式模型和生成式模型

### 2.8.1 判别模型是直接对P(Y|X)建模，

就是说，直接根据X特征来对Y建模训练。

所以判别式模型的特征总结如下：

1. 对P(Y|X)建模
2. 对所有的样本只构建一个模型，确认总体判别边界
3. 观测到输入什么特征，就预测最可能的label
4. 另外，判别式的优点是：对数据量要求没生成式的严格，速度也会快，小数据量下准确率也会好些。

|  |  |
| --- | --- |
| **判别模型** | **神经网络模型、条件随机场、CART、高斯过程、SVM、LR、线性回归、径向基函数(RBF)** |
| **生成模型** | **混合高斯模型、隐马尔科夫模型、马尔科夫随机场、NB、LDA(主题模型)、KNN、深度信念网** |

### 2.8.2 生成式模型

在模型训练中，我学习到的是X与Y的联合模型P(X,Y)，也就是说，**我在训练阶段是只对**P(X,Y)**建模**，我需要确定维护这个联合概率分布的所有的信息参数。完了之后在inference再对新的sample计算 P(Y|X)，导出Y,但这已经不属于建模阶段了。

所以生成式总结下有如下特点：

1. 对P(X,Y)建模
2. 这里我们主要讲分类问题，所以是要对每个label(yi)都需要建模，最终选择最优概率的label为结果，所以没有什么判别边界。（对于序列标注问题，那只需要构件一个model）
3. 中间生成联合分布，并可生成采样数据。
4. 生成式模型的优点在于，所包含的信息非常齐全，我称之为“上帝信息”，所以不仅可以用来输入label，还可以干其他的事情。生成式模型关注结果是如何产生的。但是生成式模型需要非常充足的数据量以保证采样到了数据本来的面目，所以速度相比之下，慢。

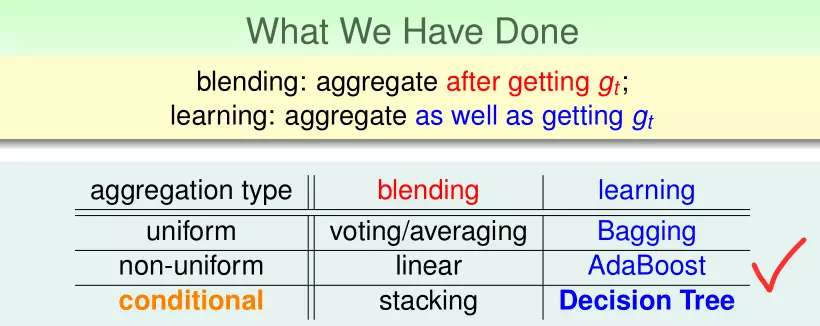
## 2.9 LDA隐含狄利克雷分布

LDA假设文档主题的先验分布是Dirichlet分布，LDA假设主题中词的先验分布是Dirichlet分布。

# 3.模型融合方法

## 3.1Boosting和bagging的原理以及异同点

**bagging.boosting.stacking的区别？**



**Bagging**：通过对原数据进行有放回的抽取，构建出多个样本数据集，然后用这些新的数据集训练多个分类器。因为是有放回的采用，所以一些样本可能会出现多次，而其他样本会被忽略。该方法是通过降低基分类器方差来改善泛化能力,因此Bagging的性能依赖于基分类器的稳定性，如果基分类器是不稳定的Bagging有助于减低训练数据的随机扰动导致的误差，但是如果基分类器是稳定的，即对数据变化不敏感，那么Bagging方法就得不到性能的提升，甚至会降低。

**Boosting**：提升方法是一个迭代的过程，通过改变样本分布，使得分类器聚集在那些很难分的样本上，对那些容易错分的数据加强学习，增加错分数据的权重，这样错分的数据再下一轮的迭代就有更大的作用（对错分数据进行惩罚）。

**Bagging与Boosting的区别**：

二者的主要区别是取样方式不同。Bagging采用均匀取样，而Boosting根据错误率来取样，因此Boosting的分类精度要优于Bagging。Bagging的训练集的选择是随机的，各轮训练集之间相互独立，而Boosting的各轮训练集的选择与前面各轮的学习结果有关；

Bagging的各个预测函数没有权重，而Boosting是有权重的；

Bagging的各个预测函数可以并行生成，而Boosting的各个预测函数只能顺序生成。

bagging是减少variance，而boosting是减少bias。Bagging 是 Bootstrap Aggregating的简称，意思就是再取样 (Bootstrap) 然后在每个样本上训练出来的模型取平均，所以是降低模型的 variance. Bagging 比如 Random Forest 这种先天并行的算法都有这个效果。Boosting 则是迭代算法，每一次迭代都根据上一次迭代的预测结果对样本进行加权，所以随着迭代不断进行，误差会越来越小，所以模型的 bias 会不断降低。这种算法无法并行。

**Stacking**是一个分层的结构模型。

加入了k折交叉验证。

## 3.2GBDT

有哪些可调参数，一般怎么选参数？

|  |  |
| --- | --- |
| **Boosting框架参数** | **弱学习器参数** |
| **n\_estimators**:也就是弱学习器的最大迭代次数，或者说最大的弱学习器的个数 | 划分时考虑的最大特征数**max\_features** |
| **learning\_rate**: 即每个弱学习器的权重缩减系数ν | 决策树最大深度**max\_depth** |
| **Subsample**：子采样比例 | 内部节点再划分所需最小样本数**min\_samples\_split** |
| **loss**:损失函数 | 叶子节点最小的样本权重和**min\_weight\_fraction\_leaf** |
| **GBDT的正则化：** | |
| 1.和Adaboost类似的正则化项，即步长(learning rate)。 | |
| 2.通过子采样比例（subsample） | |
| 3.对于弱学习器即CART回归树进行正则化剪枝。 | |
| **GBDT多分类** | |
| 对于多分类任务，GDBT的做法是采用一对多的策略。 也就是说，对每个类别训练M个分类器。假设有K个类别，那么训练完之后总共有M\*K颗树。 | |

多分类 <https://blog.csdn.net/qq_22238533/article/details/79199605>

多分类softmax，然后更新公式用的近似的一个式子，可以看看原论文

### 3.2.1 GBDT如何处理样本不均衡

<https://www.nowcoder.com/discuss/98955?type=2&order=0&pos=13&page=1>

（3）样本不均衡的话，可以怎么解决？（带权重分类，Aaboost本身就是带权重分类，那我可以直接用Adboost，机智Boy）。Adboost是可以，但是在GBDT框架下，怎么学习呢，会产生怎样的影响（大佬不依不饶，我只有尴尬而不失礼貌地笑了--\_--）：你想啊，拿最简单的回归来看吧，写个平方损失函数吧（我写了MSE，瞬间想到CART每步选取最优分割点是根据MSE的，那我也可以给每个样本加权啊，加权之后，会更加偏向对权重大的样本的拟合，而权重较小的样本有点误差也是可以接受的），我已经忘了我昨天怎么想到了梯度，说权重大的，梯度也会大一点，更新更快（不是梯度下降啊，大兄弟，而且GB也不仅依赖权重啊，自身x，残差）

（4）除了加权之外，还有其他方法吗（降采样和过采样）；采样和加权有什么区别吗（本质上是没有区别的，但是采样的话，需要先进行采样，而且NP比例也是一个超参数，需要验证，而加权的话就直接train好了，其实类别权重也是需要验证的系不系）；采样的话，造成了数据分布的改变，会产生什么问题嘛？有的话，如何应对？（又是一脸蒙蔽，迷之微笑！！！采样确实会造成数据分布不一致，但是模型的分辨能力得到了提升，而不至于在极度不平衡的数据上学不到任何知识，大佬解释了一统）

### 3.2.2 GBDT 并行

1. 按行并行化，将样本按行分成N行，分别在N个节点上做计算；
2. 并行建立一棵树的过程；
   * 在0号节点上对特征随机采样，生成建立一棵树需要用到的特征，并分发到N个节点上；
   * 在0号节点上维护每一维采样特征所有可能的特征值；
   * 将每一维特征的每一个可能的特征值分发到N个节点上；
   * 每一个节点并行计算该节点上所有样本与分发得到的特征值的比较结果，分割成左右子树，并计算增益；
   * 归并所有节点的增益，在0号节点得到每一个特征在每一个特征值的增益(f, v, incr)；
   * 在0号节点上找出最大的(f, v, incr)，并作为本次的最佳裂点，分发到N个节点；
   * N个节点将样本分割成左右子树；
   * 对左右子树继续上面过程，直到叶子节点数目满足要求；
3. 并行建立第二棵树；

因此，GBDT并行化包括了样本并行化与特征分裂点计算的并行化；其中最耗时的仍然是需要遍历特征的所有可能的特征值，并计算增益寻找最优分裂点的过程；可以采用对特征值直方图采样，不用遍历所有特征值来优化。

### 3.2.3 AdaBoost V.S. GBDT

最主要的区别在于两者如何识别模型的问题。AdaBoost用错分数据点来识别问题，通过调整错分数据点的权重来改进模型。Gradient Boosting通过负梯度来识别问题，通过计算负梯度来改进模型。

## 3.3 RF

|  |  |
| --- | --- |
|  |  |
| **n\_estimators**: 也就是弱学习器的最大迭代次数 | RF划分时考虑的最大特征数**max\_features** |
| **oob\_score** :即是否采用袋外样本来评估模型的好坏。 | 决策树最大深度**max\_depth** |
| **criterion:**即CART树做划分时对特征的评价标准。 | 内部节点再划分所需最小样本数**min\_samples\_split** |
|  | 最大叶子节点数**max\_leaf\_nodes** |
|  | 叶子节点最少样本数**min\_samples\_leaf**: |
|  | **class\_weight** |
|  |  |

### RF处理缺失值

随机森林(Random Forests)。随机森林是已故统计学家Leo Breiman提出的，和gradient boosted tree一样，它的基模型是决策树。在介绍RF时，Breiman就提出两种解决缺失值的方法（[Random forests - classification description](https://link.zhihu.com/?target=https%3A//www.stat.berkeley.edu/%7Ebreiman/RandomForests/cc_home.htm%23missing1)）：

* 方法1(快速简单但效果差)：把数值型变量(numerical variables)中的缺失值用其所对应的类别中(class)的*中位数*(median)替换。把描述型变量(categorical variables)缺失的部分用所对应类别中*出现最多的数值*替代(most frequent non-missing value)。
* 方法2(耗时费力但效果好)：虽然依然是使用*中位数*和*出现次数最多的数*来进行替换，方法2引入了权重。即对需要替换的数据先和其他数据做相似度测量(proximity measurement)在补全缺失点时相似的点的数据会有更高的权重W。

参考：[缺失值的处理](https://www.zhihu.com/question/58230411/answer/242037063)

## 3.4 Xgboost的原理

处理非平衡数据集：

会存在一些数据极度不平衡，会影响到xgb模型的训练，有两个方法可以改善这种情况：  
1、如果你关注预测结果的排序结果auc  
    平衡正负样本的权重，通过scale\_pos\_weight来处理。  
2、如果你关注预测正确的概率：  
    这种情况下，你不可以对这个数据进行re-balance,如果你采样或者别的操作，会改变原始数据集合的分布。可以设置参数max\_delta\_step为一个确定的数，这有助于模型收敛。

**xgboost中booster有三个选项，gblinear，gbtree，dart，三者的区别与适用场景？**

gblinear 使用线性方程

gbtree，dart(增加了Dropout)使用的是树模型

dart额外的参数

* sample\_type 采样算法
* rate\_drop 前置树的丢失率，有多少比率的树不进入下一个迭代[0,1]

#### xgboost如何防止过拟合

行抽样、列抽样(借鉴随机森林)、shrink学习率缩减、目标函数中含有正则化项。

#### xgboost的参数

|  |  |  |
| --- | --- | --- |
| 通用参数 | booster参数 | 学习目标参数 |
| 参数控制在提升（boosting）过程中使用哪种booster，常用的booster有树模型（tree）和线性模型（linear model） | 这取决于使用哪种booster。 | 控制学习的场景，例如在回归问题中会使用不同的参数控制排序。 |
|  | eta 学习率 | Objective |
| booster | min\_child\_weight 最小叶子结点样本权重和 | eval\_metric |
| slient | max\_depth | seed |
| nthread | gamma |  |
|  | Lambda、alpha |  |
|  |  |  |

Booster参数：

1. eta 学习率

2. min\_child\_weight 最小叶子结点样本权重和。在节点分裂的时候，如果分裂后所有叶子节点的样本和小于min\_child\_weight，那么就停止分裂，可以用来防止过拟合。

3. max\_depth

4. gamma 在节点分裂的时候只有分裂后，损失函数的值下降了，才会分裂。Gamma指定了节点分裂的所需最小损失函数下降值。

5. lambda 权重l2正则化项

6. alpha 权重l1正则化项

7. scale\_pos\_weight 空值样本不均衡的比例。

学习目标参数：

通用参数：

1. booster：每次迭代的模型。

2. slient

3. nthread 线程数

学习目标参数：

1. Objective 二分类。多分类。

2. eval\_metric 度量方法，回归问题 rmse，分类问题 logloss

3. seed随机种子数。

## 3.5 Lgbm的原理

Lgb的并行：

- 特征并行

- 数据并行

- 投票并行

## 3.6 模型比较

### Lgb，XGB的区别和联系，并行是如何并行的

|  |  |  |
| --- | --- | --- |
|  | Xgboost | Light gbm |
| 决策树的生长策略 | Level-wise的 | Leaf-wise的 |
| 分裂点的查找方法 | 预排序的方法 | 直方图的方法 |
| 内存开销 | 2\*#feature\*#data\*4bytes | #feature\*#data\*1bytes |
| 类别特征的处理 | One-hot | 直接支持 |
| 缓存的利用 |  | 更高的缓存利用率 |

xgboost这个并行化是特征粒度上的并行化：划分节点时，每个特征并行计算，同时每个特征的划分节点也是并行计算（这是加速最猛的处理）

### GBDT和随机森林的区别

**GBDT 和随机森林的相同点：**

* + 都是由多棵树组成；
  + 最终的结果都由多棵树共同决定。

**GBDT 和随机森林的不同点：**

* + 组成随机森林的可以是分类树、回归树；组成 GBDT 只能是回归树；
  + 组成随机森林的树可以并行生成（Bagging）；GBDT 只能串行生成（Boosting）；这两种模型都用到了Bootstrap的思想。
  + 对于最终的输出结果而言，随机森林使用多数投票或者简单平均；而 GBDT 则是将所有结果累加起来，或者加权累加起来；
  + **随机森林对异常值不敏感，GBDT 对异常值敏感**；
  + 随机森林对训练集一视同仁权值一样，GBDT 是基于权值的弱分类器的集成；
  + 随机森林通过减小模型的方差提高性能，GBDT 通过减少模型偏差提高性能。

### GBDT和xgboost的区别

* + 损失函数的正则添加、残差原来从一阶换成泰勒展开成二阶、这里可以细说）
  + **传统的GBDT以CART树作为基学习器，XGBoost还支持线性分类器，**这个时候XGBoost相当于L1和L2正则化的逻辑斯蒂回归（分类）或者线性回归（回归）；
  + 传统的GBDT在优化的时候只用到一阶导数信息，XGBoost则对代价函数进行了二阶泰勒展开，得到一阶和二阶导数；
  + **XGBoost在代价函数中加入了正则项，用于控制模型的复杂度**。从权衡方差偏差来看，它降低了模型的方差，使学习出来的模型更加简单，放置过拟合，这也是XGBoost优于传统GBDT的一个特性
  + **列抽样。**XGBoost借鉴了随机森林的做法，支持列抽样，不仅防止过 拟合，还能减少计算；
  + **对缺失值的处理。**对于特征的值有缺失的样本，XGBoost还可以自动 学习出它的分裂方向；xgboost考虑了训练数据为稀疏值的情况，可以为缺失值或者指定的值指定分支的默认方向，这能大大提升算法的效率，paper提到50倍.
  + **XGBoost工具支持并行。**Boosting不是一种串行的结构吗?怎么并行 的？注意XGBoost的并行不是tree粒度的并行，XGBoost也是一次迭代完才能进行下一次迭代的（第t次迭代的代价函数里包含了前面t-1次迭代的预测值）。XGBoost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代 中重复地使用这个结构，大大减小计算量。这个block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。 （58面试问过：xgboost的特征并行是怎么做到的）
  + 可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。
  + **xgboost在优化目标函数的同时相当于做了预剪枝。另外，上式中还有一个系数lambda，是正则项里leaf score的L2模平方的系数，对leaf score做了平滑，也起到了防止过拟合的作用，这个是传统GBDT里不具备的特性。**
* gbdt、adaboost、随机森林中哪个用的是梯度上升法（PDD）

Adaboost、gbdt

# 4、深度学习

## 权重初始化的方法

|  |  |
| --- | --- |
| Zero initialization |  |
| Random |  |
| He initialization  (Relu) | 初始化为 |
| Xavier初始化  tanh激活函数 | 初始化为 |
|  |

## 优化方法

梯度下降法，牛顿法，拟牛顿法区别(阿里)  
SGD,ADAM区别(百度)

|  |
| --- |
| GD的步骤 |
| 1. 计算目标函数关于参数的梯度 |
| 2. 根据历史梯度计算一阶动量 |
| 根据历史梯度计算二阶动量 |
| 3. 更新模型参数 |

|  |  |  |
| --- | --- | --- |
| 优化算法 | 公式 | 特点 |
| SGD |  | SGD 的缺点在于收敛速度慢，可能在鞍点处震荡。并且，如何合理的选择学习率是 SGD 的一大难点。 |
| Moment+SGD |  | 在梯度下降过程中引入惯性，加速收敛减小震荡。 |
| NAG |  |  |
| Adagrad |  | 不同参数的学习率不同 |
| RMSprop |  | 在 Adagrad 中，是单调递增的，使得学习率逐渐递减至 0，可能导致训练过程提前结束。为了改进这一缺点，可以考虑在计算二阶动量时不累积全部历史梯度，而只关注最近某一时间窗口内的下降梯度。根据此思想有了 RMSprop |
|  |  |  |

**梯度下降法**实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。**梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，最速下降法越接近目标值，步长越小，前进越慢。**

由于批量梯度下降法在更新每一个参数时，都需要所有的训练样本，所以训练过程会随着样本数量的加大而变得异常缓慢。随机梯度下降法（Stochastic Gradient Descent，简称SGD）正是为了解决批量量梯度下降法这一弊端⽽而提出的。

随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已将迭代到最优解了。SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化⽅方向。其优缺点如下：

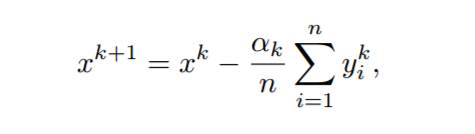
优点：训练速度快；

缺点：准确度下降，并不是全局最优；不易于并行实现。

梯度下降法的缺点

1. 首先选择一个合适的学习速率很难。若学习速率过⼩小，则会导致收敛速度很慢。如果学习速率过大，那么会阻碍收敛，即在极值点附近振荡
2. 学习速率调整（又称学习速率调度，Learning rate schedules）试图在每次更新过程中，改变学习速率，如模拟退火按照预先设定的调度算法或者当相邻的迭代中目标变化小于一个阈值时候减小学习速率。但是梯度下降算法的调度和阈值需要预先设置，无法对数据集特征进行自适应。
3. 模型所有的参数每次更新都是使用相同的学习速率。如果我们的数据很稀疏并且我们的特征出现的次数不同，我们可能不会希望所有的参数以某种相同的幅度进行更新，而是针对很少出现的特征进行一次大幅度更新。
4. 陷入鞍点。

|  |  |
| --- | --- |
| 随机梯度下降sgd  我们知道sgd是当前权重减去步长乘以梯度，得到新的权重。sag中的a，就是平均的意思，具体说，就是在第k步迭代的时候，我考虑的这一步和前面n-1个梯度的平均值，当前权重减去步长乘以最近n个梯度的平均值。n是自己设置的，当n=1的时候，就是普通的sgd。 | |
| 随机平均梯度法sag | |
|  | |
|  | 一般的 mini-batch 大小为 64 到 512 |
|  | |
|  | |



### 牛顿法

在当前迭代点对目标函数进行泰勒展开，得到目标函数的二次近似。将二次近似的极小点作为下一个迭代点。

优点：二阶收敛速度很快。

缺点：

要求目标函数必须一阶二阶可导。

每次都要求海森矩阵，计算量大。

BFGS它用低秩分解来代替求逆。后来还有的L-BFGS算法，降低了算法对内存的需求。对于很多多维机器学习问题，这个优势是很重要的。

### 拟牛顿法

BFGS 近似海森矩阵。

采用迭代的方法求Bk

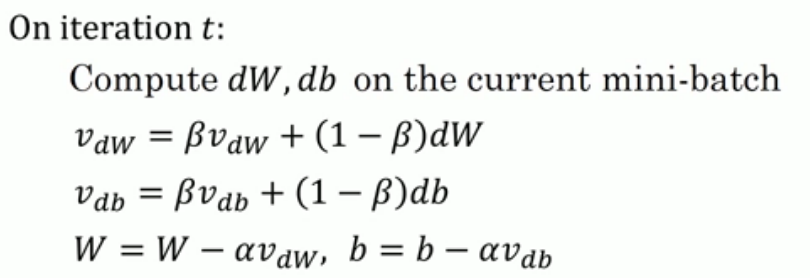
DFP 近似海森矩阵的逆。

通过迭代对海森矩阵的逆做近似。

### Momentum 动量

在更新参数时，除了考虑到梯度以外，还考虑了上一时刻参数的历史变更幅度。

(指数加权平均的思想)



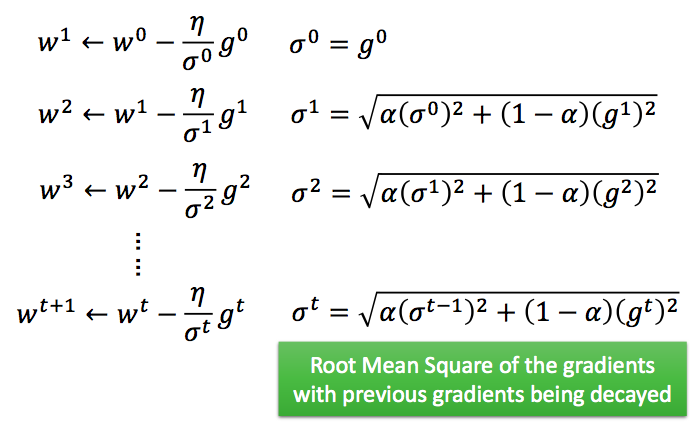
### Adagrad

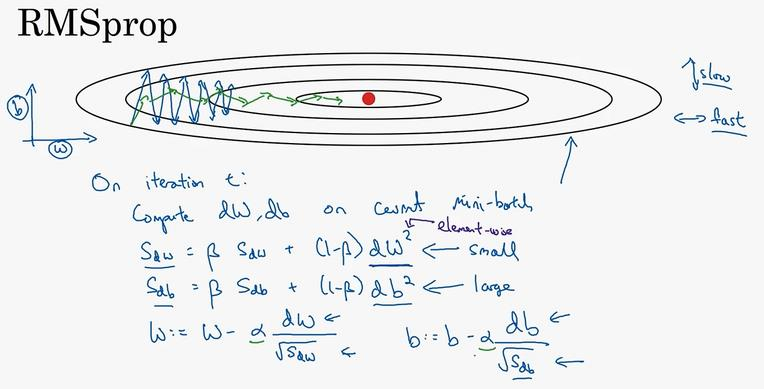
之前的方法中所有参数在更新时均使用同一个Learning rate。而Learning rate调整是一个非常耗费计算资源的过程，所以如果能够自适应地对参数进行调整的话，就大大降低了成本。在Adagrad的每一个参数的每一次更新中都使用不同的learning rate。实质上是对学习率形成了一个约束项regularizer。

**缺点：**由公式可以看出，仍依赖于⼈工设置一个全局学习率； 设置过大的话，会使得regularizer过于敏感，对梯度的调节太大；中后期，分母上梯度平⽅方的累加将会越来越大，使得梯度为0，训练提前结束。

### RMSprop

用了一种很简单的方式修改了Adagrad方法，让它不过于激进而过早停止学习。具体说来就是，它使用了一个梯度平方的滑动平均，仍然是基于梯度的大小来对每个权重的学习率进行修改，效果不错。但是和Adagrad不同的是，其更新不会让学习率单调变⼩。





### Adadelta

是Adagrad的一种扩展，以缓解Adagrad学习速率单调递减问题的算法。Adadelta不是对过去所有时刻的梯度平方进行累加，而是将累加时刻限制在窗⼝⼤⼩为的W区间。

### Adam

**RMSProp + Momentum**

手动调整 SGD(随机梯度下降)动量(momentum)的方法可与目前最先进的自适应方法(如 Adam)竞争,那么 SGD 动量能不能自动调节呢?斯坦福大学的研究人员近日提出了 YellowFin,一种自动调整 SGD 动量超参数的方法。

|  |  |  |
| --- | --- | --- |
|  | **公式** | **特点** |
|  |  |  |
| Adagrad |  | Adagrad的每一个参数的每一次更新中都使用不同的learning rate。 |
| Adadelta |  |  |
| Rmsprop |  |  |

## 梯度消失和爆炸

|  |  |  |
| --- | --- | --- |
|  | 梯度消失 | 梯度爆炸 |
| 原因 | 每次通过一次sigmoid，影响就衰减一次 |  |
|  |  |  |
| 解决 | 改变激活函数 （改成relu的） | 梯度裁剪 |
|  | Layers-retrainning。先learn好第一个layer，再learn第二个layer |  |
|  |  |  |
|  | 残差网络针对梯度消失和梯度爆炸提出来的 |  |

## 介绍CNN

原理、作用、多层CNN提取全局信息。池化、怎么gpu加速？一维CNN，二维CNN，球面卷积CNN。

#### 卷积

**卷积的作用是提取特征，前面的卷积提取类似于人眼能识别的初步特征，后面的卷积是能够提取更加不容易发现但是真实存在的特征。**

Pooling 用过，max pooling， average pooling， global average pooling。再问这个两个分别有什么用？

max pooling其实就是最具有代表性的特征；average pooling提取的是比较general 的特征；global average pooling用来分类的，因为后面网络的加深，full connected layer参数太多了，不容易训练，为了快速准确得到结果，采用global average pooling，没有参数，但是得到的分类效果跟FC差不多。

**单个卷积层的时间复杂度：Time~O(M^2 \* K^2 \* Cin \* Cout)**

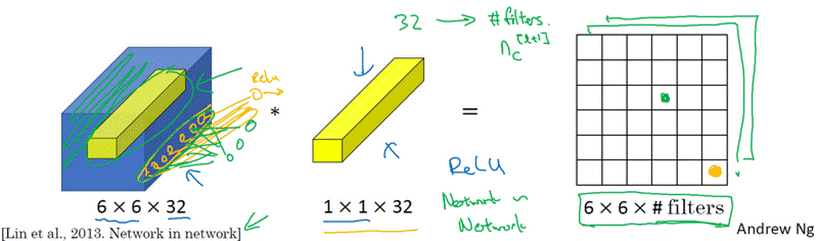
注1：为了简化表达式中的变量个数，这里统一假设输入和卷积核的形状都是正方形。

注2：严格来讲每层应该还包含1个Bias参数，这里为了简洁就省略了。

* M:输出特征图（Feature Map）的尺寸。
* K:卷积核（Kernel）的尺寸。
* Cin:输入通道数。
* Cout:输出通道数。
* 输出特征图尺寸又由输入尺寸X、卷积核尺寸K、Padding、 Stride 这四个参数所决定，表示如下：

**M=(X - K + 2\*Padding) / Stride + 1**

#### 1×1  卷积



可以做到压缩图片的高度。池化层，只是压缩了高度和宽度。

1×1 卷积层就是这样实现了一些重要功能的（doing something pretty non-trivial），它给**神经网络添加了一个非线性函数，从而减少或保持输入层中的通道数量不变**，当然如果你愿意，也可以增加通道数量。

**Inception 网络或 Inception 层的作用就是代替人工来确定卷积层中的过滤器类型，或者确定是否需要创建卷积层或池化层**

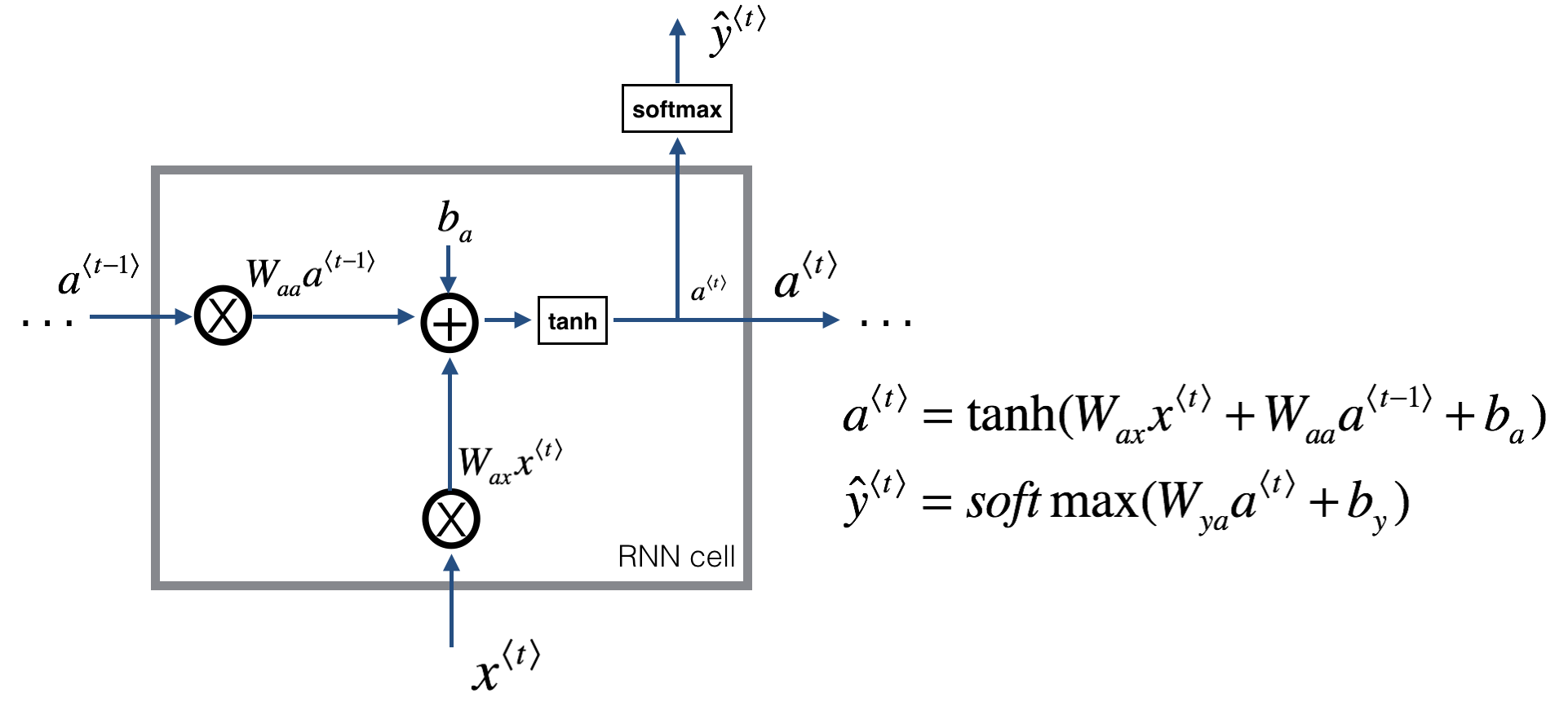
## 介绍RNN

Rnn、lstm、gru

写lstm的结构和前向传播的公式

attention的时候问query key value各是啥（头条）

RNN

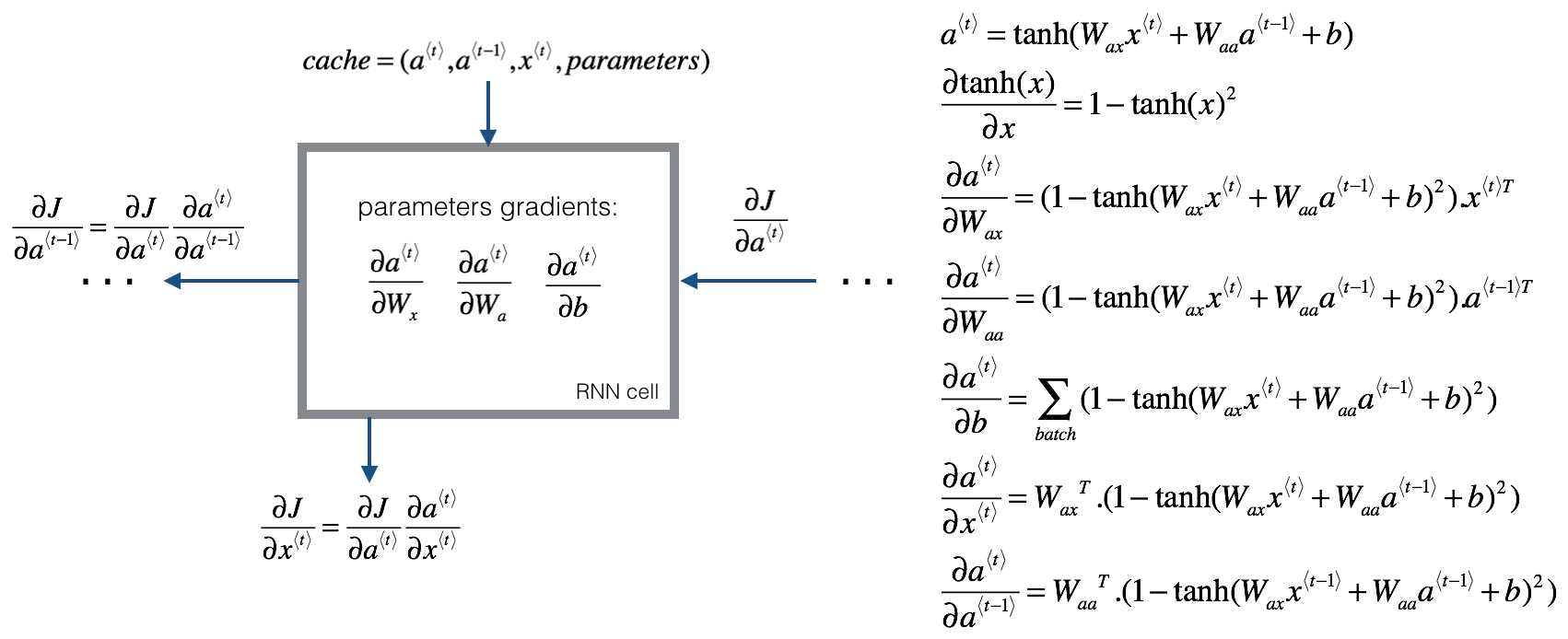


### CNN和RNN的区别？

全连接DNN的结构里下层神经元和所有上层神经元都能够形成连接，带来的潜在问题是参数数量的膨胀。通过“卷积核”作为中介。同一个卷积核在所有图像内是共享的，图像通过卷积操作后仍然保留原先的位置关系。

DNN的无法对时间序列上的变化进行建模，所以出现了RNN，神经元的输出可以在下一个时间戳直接作用到自身。但是RNN出现梯度消失，长短时记忆单元LSTM，通过门的开关实现时间上记忆功能，并防止梯度消失。

RNN既然能继承历史信息，是不是也能吸收点未来的信息呢？双向RNN、双向LSTM，同时利用历史和未来的信息。

LSTM

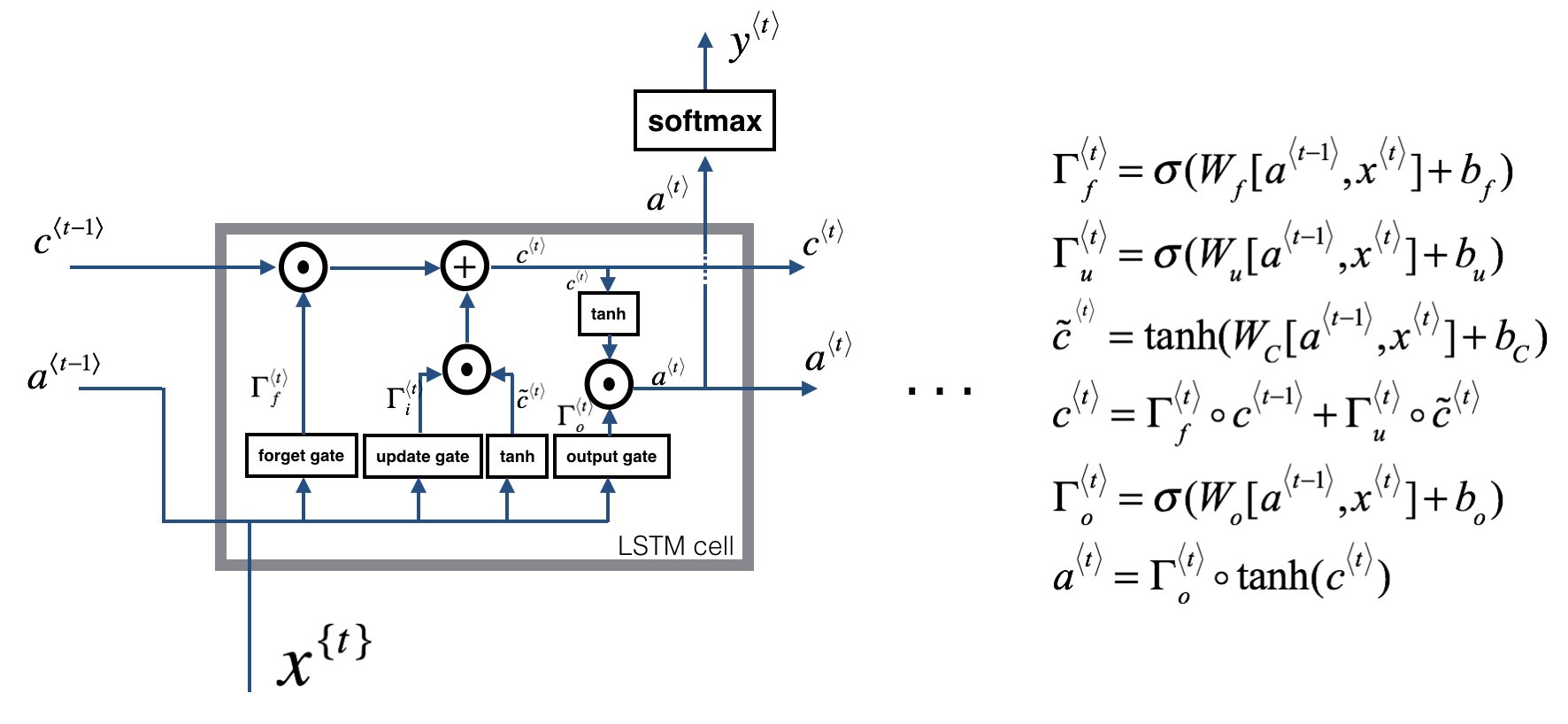
梯度消散是原生RNN中一个很大的问题，也就是后面时间的节点对于前面时间的节点感知力下降，也就是忘事儿。Hochreiter et al., 1997[5] 提出了LSTM，它的设计初衷就是来解决梯度消散问题。在标准的RNN中，这个重复的模块只有一个非常简单的结构，例如一个tanh层。LSTM同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互。如下图所示，一个LSTM块有四个输入。

（1）输入（Input）：模块的输入；

（2）输入门（Input Gate）：控制输入；

（3）遗忘门（Forget Gate）：控制是否更新记忆单元（Memory Cell）；

（4）输出门（Output Gate）：控制输出。



### GRU对LSTM做了两个大改动

1.将输入门、遗忘门、输出门变为两个门：更新门（Update Gate）和重置门（Reset Gate）。

2.将单元状态与输出合并为一个状态：。

GRU只用了两个gates，将LSTM中的输入门和遗忘门合并成了更新门。并且并不把线性自更新建立在额外的memory cell上，而是直接线性累积建立在隐藏状态上，并靠gates来调控。

RNN的输入： **(input, h\_0)** - input (seq\_len, batch, input\_size): 保存输入序列特征的tensor。input可以是被填充的变长的序列。细节请看torch.nn.utils.rnn.pack\_padded\_sequence()

* h\_0 (num\_layers \* num\_directions, batch, hidden\_size): 保存着初始隐状态的tensor

RNN的输出： **(output, h\_n)**

* output (seq\_len, batch, hidden\_size \* num\_directions): 保存着RNN最后一层的输出特征。如果输入是被填充过的序列，那么输出也是被填充的序列。
* h\_n (num\_layers \* num\_directions, batch, hidden\_size): 保存着最后一个时刻隐状态。

RNN模型参数:

* weight\_ih\_l[k] – 第k层的 input-hidden 权重， 可学习，形状是(input\_size x hidden\_size)。
* weight\_hh\_l[k] – 第k层的 hidden-hidden 权重， 可学习，形状是(hidden\_size x hidden\_size)
* bias\_ih\_l[k] – 第k层的 input-hidden 偏置， 可学习，形状是(hidden\_size)
* bias\_hh\_l[k] – 第k层的 hidden-hidden 偏置， 可学习，形状是(hidden\_size)

## 神经网络Overfitting、Regulation

解决Overfitting、regulation的方法：regulation，我才总结下了，大致主要聊了2点：

**1）dropout,介绍dropout的概念啊，问了下train和test阶段的不一样过程细节**

**主要讲了下test把active function的输出都乘以P，这样就把train和test的输出期望都scale到了一个range，这样才是更加准确的**

**2）Batch Normalisation:BN，BN的原理，问什么好，好在哪里？**

**1.降低了样本之间的差异，scale到（0，1）**

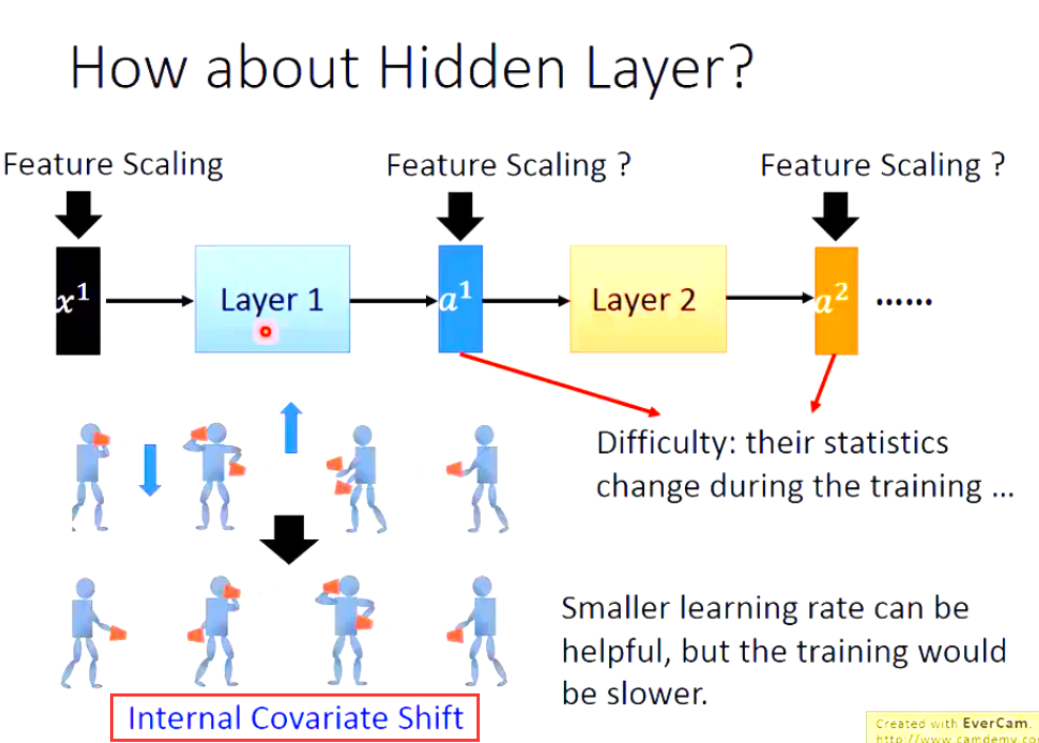
**2，降低了层层之间的依赖，主要体现在前一层的输出是下一层的输入，那么我们把所有数据都scale到了（0，1）的distribution，那么降低了层层之间的依赖关系，从而使数据更加准确**

## Batch Normalization

* 可以使不同的特征Have same scaling，加快训练
* 特征维度上的进行的。
* 每个layer之前都做，在激活函数之前。

深度学习中解决了internal Convariate shift

在深度网络的训练中，每一层网络的输入都会因为前一层网络参数的变化导致其分布发生改变，这就要求我们必须使用一个很小的学习率和对参数很好的初始化，但是这么做会让训练过程变得慢而且复杂。作者把这种现象称作Internal Covariate Shift。通过Batch Normalization可以很好的解决这个问题，



# 6. 评价

## 过拟合和欠拟合

|  |  |  |
| --- | --- | --- |
|  | 过拟合 | 欠拟合 |
| 原因 |  |  |
|  |  |  |
| 解决措施 | 获取更多的数据（数据增强） | 1. 增加模型迭代次数 |
|  | 减少特征数量 | 2. 训练一个复杂的模型（比如神经网络中增加神经网络层数、在SVM中用非线性核函数） |
|  | 采用正则化的方法 | 获取更多的特征以供训练 |
|  | 决策树---剪枝 | 降低正则化权重。 |
|  | 神经网络里面：采用dropout，BacthNormal，earling-stopping。 |  |

模型在新的数据上表现的很差。说明出现过拟合。

* 学习曲线

画出不同训练集大小时，训练集和交叉验证集的准确率，可以看到模型在新数据集上的表现，进而判断模型的情况（高偏差或高方差）

吴恩达视频 高偏差-模型不能很好的拟合数据，欠拟合

高方差-数据过度拟合，过拟合 通过看**训练集误差和验证集误差。和人类水平比较就可以得出模型是否具有高方差.**

* 交叉验证

模型的Error = Bias+Variance

Error反应的是模型的准确度

Bias反应的是模型在训练样本上的输出和真实值之间的误差，模型本身的精确度

Variance反映的是模型每一次输出结果与模型输出期望之间的误差，模型的稳定性。

训练误差Jtrain，交叉验证误差Jcv，测试误差Jtest，如果Jcv>>Jtrain,过拟合

## 样本不均衡

|  |  |
| --- | --- |
| 在样本不均衡的情况下使用F值或者AUC值是更好的选择 | |
|  | |
| 解决方法 | 两方面：改变数据分布、改变分类算法 |
|  | |

SMOTE算法：(记少数样本为s)

1. 选定一个阳性样本s。

2. 找到s最近的k个样本，k可以取5，10之类。这k个样本可能有阳性的也有阴性的。

3. 从这k个样本中随机挑选一个样本，记为r。

4. 合成一个新的阳性样本s′，s′=λs+(1−λ)r，λ是(0,1)之间的随机数。换句话说，新生成的点在r与s之间的连线上。

|  |
| --- |
| **SMOTE如何处理非连续的特征（categorical feature)。** |
| 论文里讲了三种方法，实际上就是两种思路：  思路1：先只考虑continuous feature，然后算出样本点和周围点的距离。如果样本点和周围点的categorical feature不同，那么就增加一个正则项，作为惩罚。原始距离加上惩罚项，就得到最后的近邻。  思路2：同样，只考虑continuous feature，然后算出近邻。categorical feature取这些近邻的众数。 |

## 评价指标

### Logloss

log loss就是交叉熵，交叉熵就是衡量预测的结果和真实结果之间的差异；越小越好。

### AUC和ROC

AUC的意义：roc曲线下的面积

AUC，它的含义是，随机给定一个正样本，一个负样本，将正样本预测为正类的概率p1，将负样本预测为正样本的概率p2。这个p1>p2的概率就是AUC。所以AUC反映的是分类器对样本的排序能力，另外，AUC对样本比例是够均衡不敏感。

ROC：横坐标为假正率，纵坐标为真正率，越靠近左上角越好

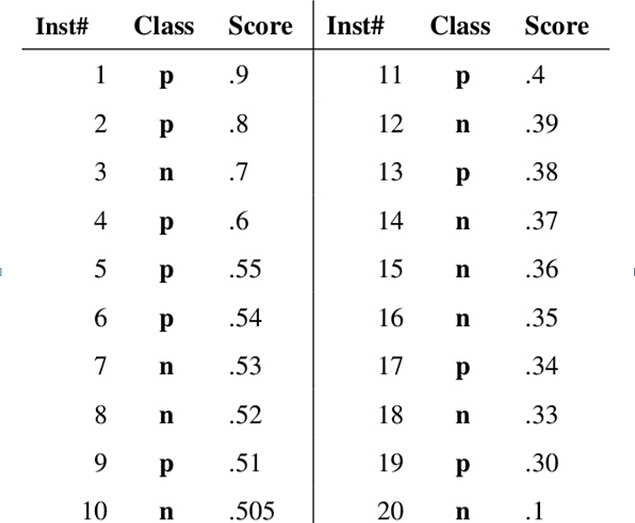
(1)真正类率(True Postive Rate)TPR: **TP/(TP+FN)**,代表分类器预测的**正类中**实际正实例占所有正实例的比例。Sensitivity

(2)假正类率(False Postive Rate)FPR:**FP/(FP+TN)**，代表分类器预测的**正类中**实际负实例占所有负实例的比例。1-Specificity

(3)真负类率(True Negative Rate)TNR:**TN/(FP+TN)**,代表分类器预测的**负类中**实际负实例占所有负实例的比例，**TNR=1-FPR**。Specificity

绘制方式：

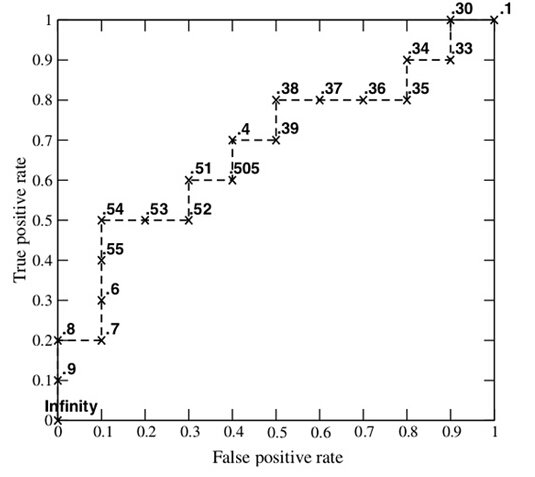
假设已经得出一系列样本被划分为正类的概率，然后按照大小排序，下图是一个示例，图中共有20个测试样本，“Class”一栏表示每个测试样本真正的标签（p表示正样本，n表示负样本），“Score”表示每个测试样本属于正样本的概率。



接下来，我们从高到低，依次将“Score”值作为阈值threshold，当测试样本属于正样本的概率大于或等于这个threshold时，我们认为它为正样本，否则为负样本。举例来说，对于图中的第4个样本，其“Score”值为0.6，那么样本1，2，3，4都被认为是正样本，因为它们的“Score”值都大于等于0.6，而其他样本则都认为是负样本。每次选取一个不同的threshold，我们就可以得到一组FPR和TPR，即ROC曲线上的一点。这样一来，我们一共得到了20组FPR和TPR的值，将它们画在ROC曲线的结果如下图：

**ROC曲线优点：**

当测试集中的正负样本的分布变换的时候，ROC曲线可以保持不变



### F1 score

### 精确率、召回率

**准确率（accuracy）： (TP + TN )/( TP + FP + TN + FN)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 预测 | | |
| 实际 |  | **1** | **0** |
| **1** | **TP** | **FN** |
| **0** | **TF** | **TN** |

|  |  |
| --- | --- |
| **精准率（precision）** | TP / (TP + FP) |
| 针对预测结果而言的，预测为正的，有多少是对的 | |
| **召回率（recall）** | TP / (TP + FN) |
| 针对样本而言的，样本的正例中有多少被预测了。 | |
|  |  |

### Logloss和roc、f1比较

针对平衡数据：

1. 如果关心的是真实的概率差异。使用logloss作为评价指标。
2. 如果只关心最终的预测类别，使用AUC。
3. F1-score对阈值敏感，使用前要先调整。

不平衡数据：

1. Logloss无效；

2. 如果关注的类是少数类，(不管是1还是0，此时建议使用roc-auc)

3. 对于极度不平衡的类(100:10K)，例如欺诈识别，建议选择f1,更能体现出模型的性能。

多分类：(F1-score, Average Accuracy, Log-loss)

Micro

Macro

auc 更多的关注的是排序的结果。

logloss 则是越小越好。

auc 与 logloss 关系

比如 1 1 0 1 预测值 为 0.5 0.5 0.3 0.5

那么 auc 是 1

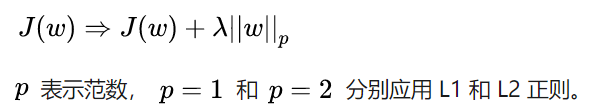
我们提升预测值到 0.7 0.7 0.4 0.7

那么 auc 依然是1

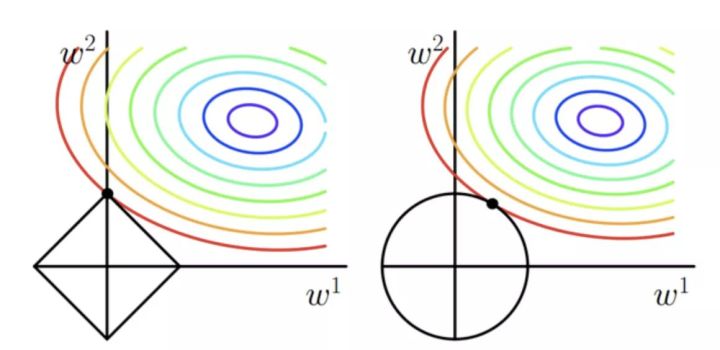
但是 logloss 有了很大的提升。

## 正则化

正则化为了解决过拟合问题。分为 L1 和 L2 正则化。主要通过修正损失函数，加入模型复杂性评估；  
正则化是符合奥卡姆剃刀原理：在所有可能的模型中，能够很好的解释已知数据并且十分简单的才是最好的模型。



* L1 正则化。向量中各元素绝对值之和 。又叫做稀疏规则算子（Lasso regularization）。关键在于能够实现特征的自动选择，参数稀疏可以避免非必要的特征引入的噪声；
* L2 正则化。使得每个元素都尽可能的小，但是都不为零。在回归里面，有人把他的回归叫做岭回归（Ridge Regression），也有人叫他 “权值衰减”（weight decay）。



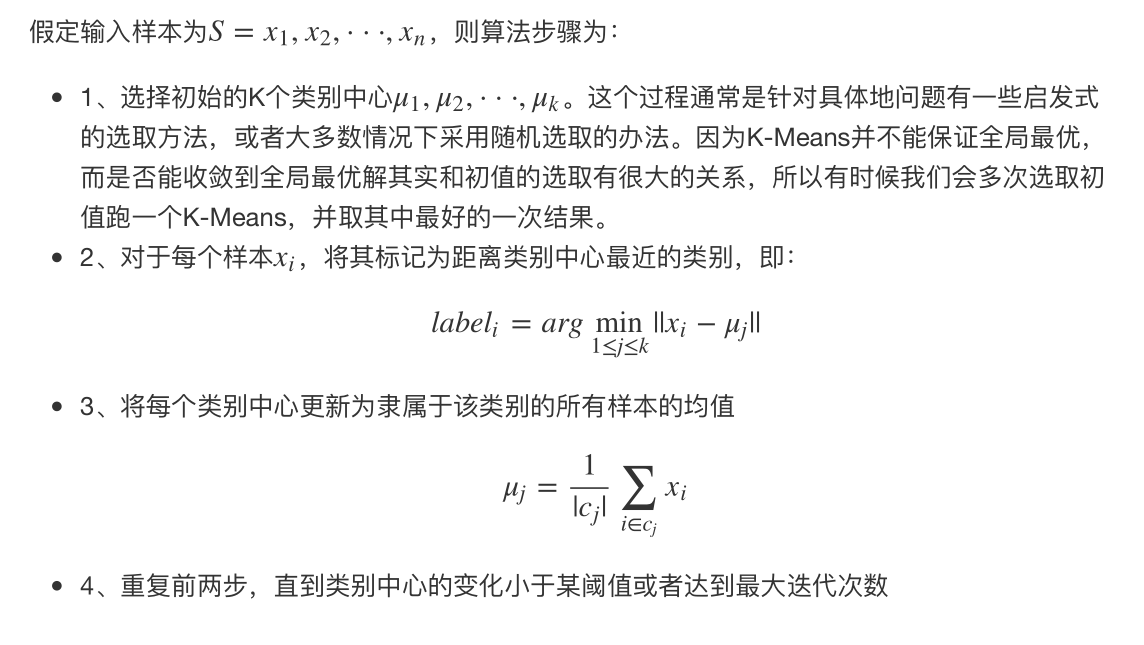
一句话总结就是：L1 会趋向于产生少量的特征，而其他的特征都是 0，而 L2 会选择更多的特征，这些特征都会接近于 0.

## 特征选择

特征选择的几种方式有没有特定的应用场景

# 5. 聚类

## 5.1 k-means算法

kmeans的计算方法如下

### 时间复杂度

聚类数K，迭代次数T，样本容量N。k-means的时间复杂度为O(N\*K\*T)。

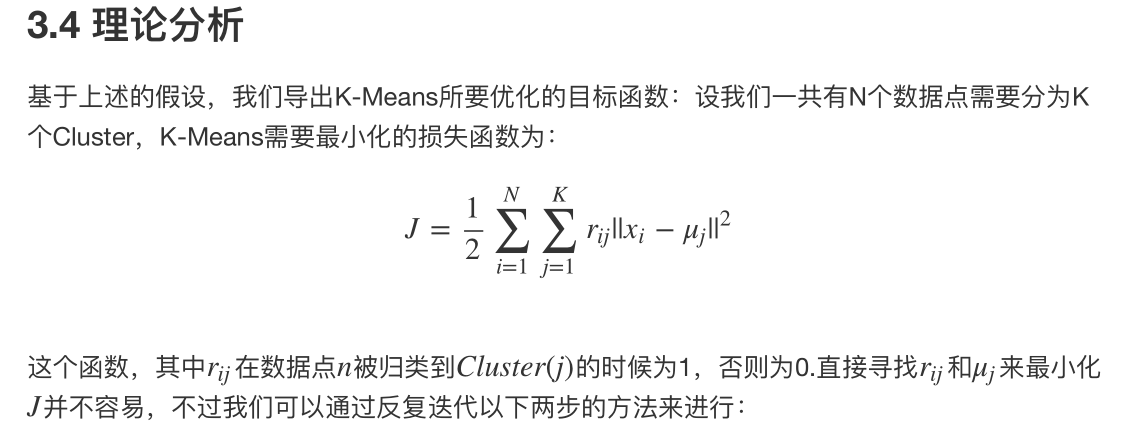
|  |  |
| --- | --- |
| 问题 | 对应的算法 |
| 初始值敏感问题的 | k-means++ 方法 |
| 必须指定聚类的个数 | isodata方法 |
| 噪声和离群值非常敏感 | k-mediods和k-medians |
| 不适用于category类型 | k-modes。 |
| 不能解决非凸问题 | kernel k-means。 |

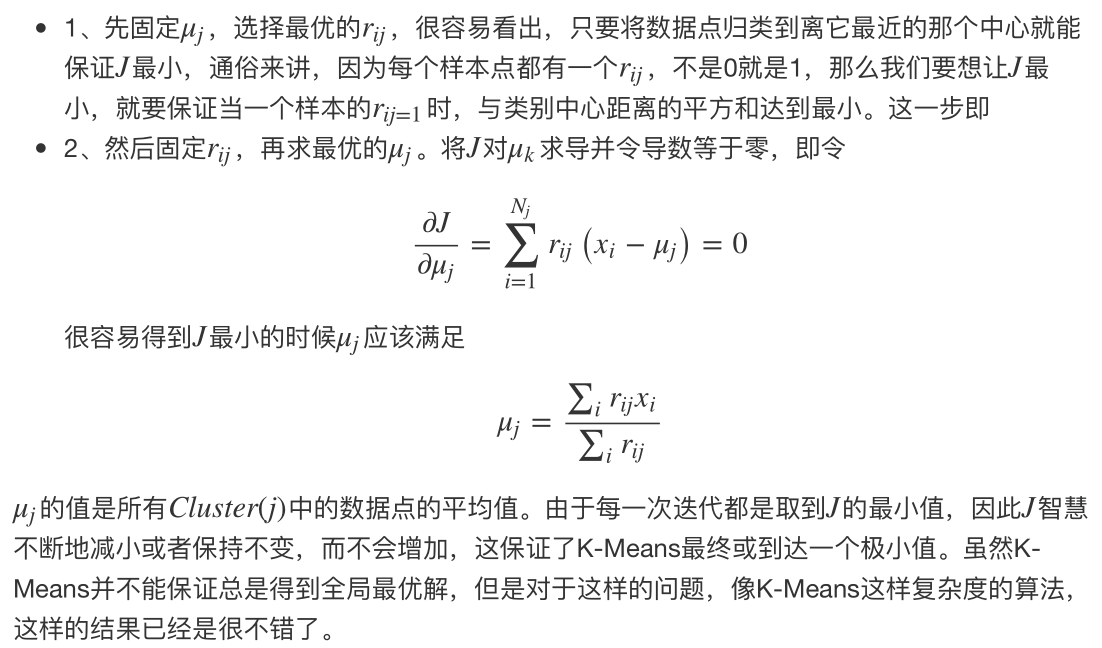
**ISODATA算法**：当两类聚中心小于某个阀值时，将它们合并为一类。当某类的标准差大于某一 阀值时或其样本数目超过某一 阀值时，将其分裂为两类，在某类样本数目小于某一 阀值时，将其取消。这样根据初始类聚中心和设定的类别数目等参数迭代，最终得到一个比较理想的分类结果。

* K-means如何处理高维特征？k-means如何处理异常点？

k-NN和k-means维度过高出现距离收敛，证明也没答上来，解决方案是降维、换距离度量。处理异常值问题可以做异常值检测，用各种优化后的k-means。

### 理论分析





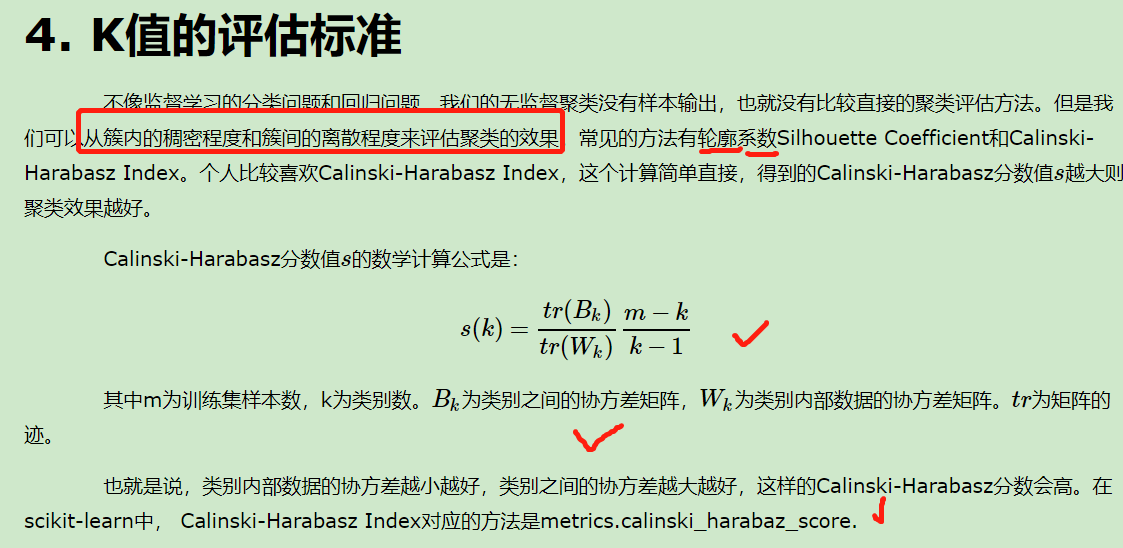
### 选择聚类个数 K的确定

1. 肘部原则——可以尝试的方法，选择下降的骤变点（快——慢）

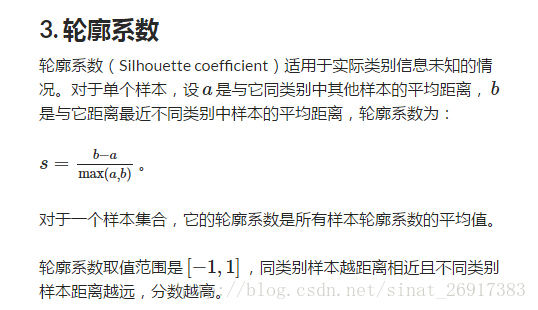
2. 轮廓系数法。

* DBSCAN原理和算法伪代码，与kmeans，OPTICS区别

### 评价指标



轮廓系数

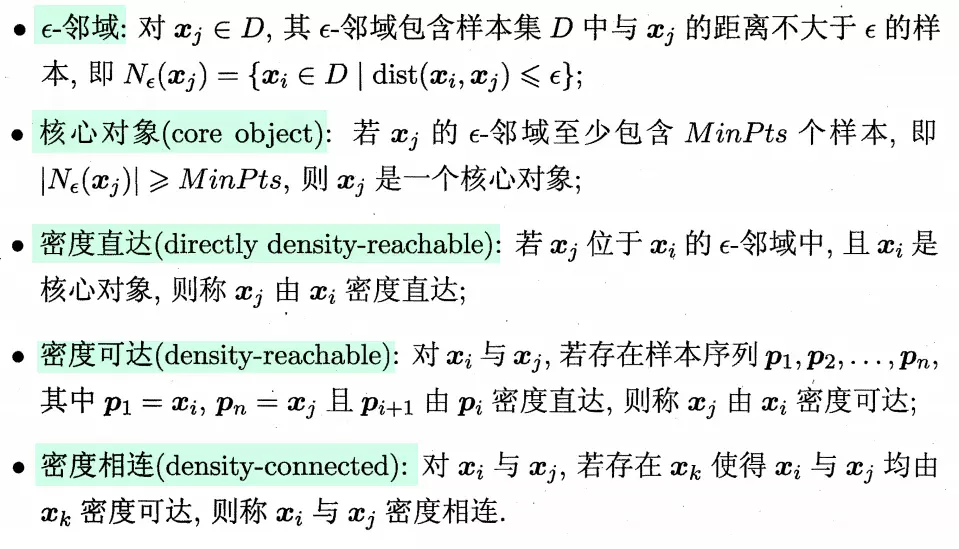


## 5.2基于密度的聚类

### 5.2.1 DBSCAN

DBSCAN是一种基于密度的空间的数据聚类方法，核心思想:从某个选定的核心点出发，不断向密度可达的区域扩张，从而得到一个包含核心点和边界点的最大化区域，区域中任意两点密度相连。

**基本概念**：如下基本概念在中有更为详细说明介绍。



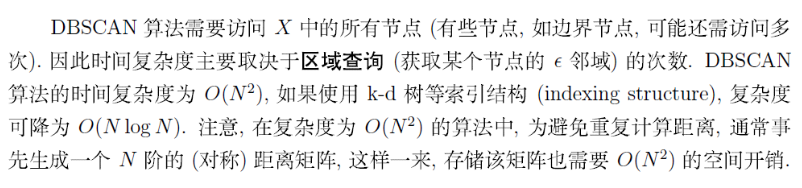
簇的定义：由密度可达关系导出的最大的密度相连样本集合。

优点：

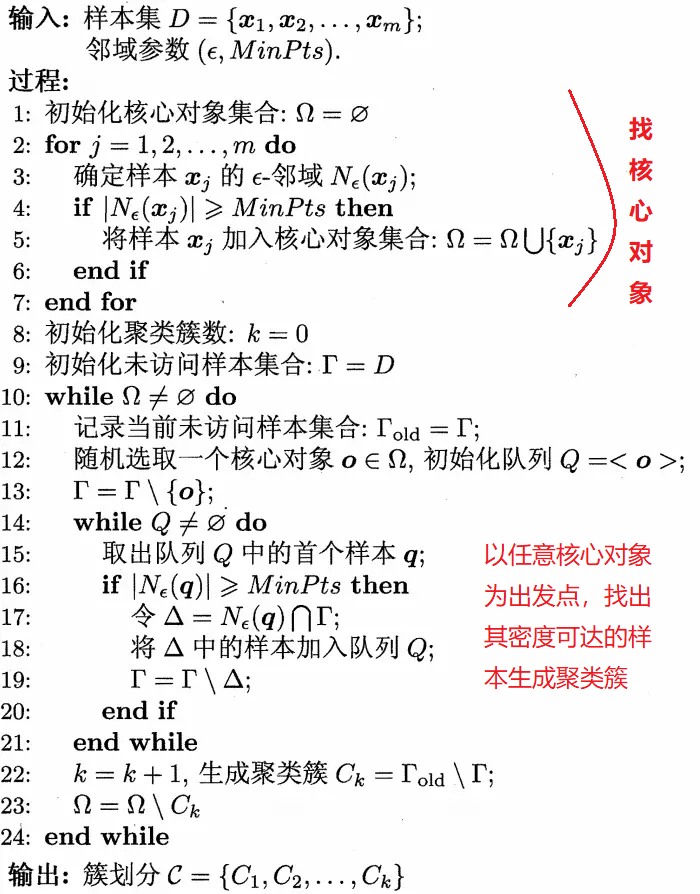
1. 不需要事先指定cluster的数目
2. 可以发现任意形状的cluster
3. 能够找出数据中的噪声，且对噪声不敏感
4. 聚类结果几乎不依赖节点遍历的顺序

缺点：

1. 效果依赖于距离度量的选取，最常用的是欧式距离，但是对于高位数据，会有维数灾难的问题。
2. DBSCAN不适合数据密度差异很大的场景，因为这种情况下，参数ε和м(MinPts)的选取是困难的。
3. **时间复杂度：**



**伪代码**：



**参数设置**

**DBSCAN共包括3个输入数据：数据集D，给定点在邻域内成为核心对象的最小邻域点数：MinPts,邻域半径：Eps，**其中Eps和MinPts需要根据具体应用人为设定。

(1) Eps的值可以使用绘制k-距离曲线(k-distance graph)方法得当，在k-距离曲线图明显拐点位置为对应较好的参数。若参数设置过小，大部分数据不能聚类；若参数设置过大，多个簇和大部分对象会归并到同一个簇中。

**K的选取**[**DBSCAN**](https://blog.csdn.net/zhouxianen1987/article/details/68945844)

(2) MinPts的选取有一个指导性的原则（a rule of thumb），MinPts≥dim+1,其中dim表示待聚类数据的维度。MinPts设置为1是不合理的，因为设置为1，则每个独立点都是一个簇，MinPts≤2时，与层次距离最近邻域结果相同，因此，MinPts必须选择大于等于3的值。若该值选取过小，则稀疏簇中结果由于密度小于MinPts，从而被认为是边界点儿不被用于在类的进一步扩展；若该值过大，则密度较大的两个邻近簇可能被合并为同一簇。因此，该值是否设置适当会对聚类结果造成较大影响。

**Sklearn里面的DBSCAN**

DBSCAN类的重要参数也分为两类，一类是DBSCAN算法本身的参数，一类是最近邻度量的参数，下面我们对这些参数做一个总结。

eps： DBSCAN算法参数，即我们的ϵ-邻域的距离阈值，和样本距离超过ϵ的样本点不在ϵ-邻域内。默认值是0.5.一般需要通过在多组值里面选择一个合适的阈值。eps过大，则更多的点会落在核心对象的ϵ-邻域，此时我们的类别数可能会减少， 本来不应该是一类的样本也会被划为一类。反之则类别数可能会增大，本来是一类的样本却被划分开。

min\_samples:DBSCAN算法参数，即样本点要成为核心对象所需要的ϵ-邻域的样本数阈值。默认值是5. 一般需要通过在多组值里面选择一个合适的阈值。通常和eps一起调参。在eps一定的情况下，min\_samples过大，则核心对象会过少，此时簇内部分本来是一类的样本可能会被标为噪音点，类别数也会变多。反之过小的话，则会产生大量的核心对象，可能会导致类别数过少。

metric：最近邻距离度量参数。可以使用的距离度量较多，一般来说DBSCAN使用默认的欧式距离（即p=2的闵可夫斯基距离）就可以满足我们的需求。可以使用的距离度量参数有：

algorithm：最近邻搜索算法参数，算法一共有三种，第一种是蛮力实现，第二种是KD树实现，第三种是球树实现。对于这个参数，一共有4种可选输入，{brute:蛮力实现，kd\_tree:KD树实现，ball\_tree:球树实现，auto:会在上面三种算法中做权衡，选择一个拟合最好的最优算法}。需要注意的是，如果输入样本特征是稀疏的时候，无论我们选择哪种算法，最后scikit-learn都会去用蛮力实现‘brute’。个人的经验，一般情况使用默认的 ‘auto’就够了。 如果数据量很大或者特征也很多，用"auto"建树时间可能会很长，效率不高，建议选择KD树实现‘kd\_tree’，此时如果发现kd\_tree速度比较慢或者已经知道样本分布不是很均匀时，可以尝试用‘ball\_tree’。而如果输入样本是稀疏的，无论你选择哪个算法最后实际运行的都是‘brute’。

**leaf\_size：**最近邻搜索算法参数，为使用KD树或者球树时。停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30. 因为这个值一般只影响算法的运行速度和使用内存大小，因此一般情况下可以不管它。

p: 最近邻距离度量参数。只用于闵可夫斯基距离和带权重闵可夫斯基距离中p值的选择，p=1为曼哈顿距离， p=2为欧式距离。如果使用默认的欧式距离不需要管这个参数。

### 5.2.2 OPTICS

改进：对输入参数不敏感。一个可达距离的升序排序列表。

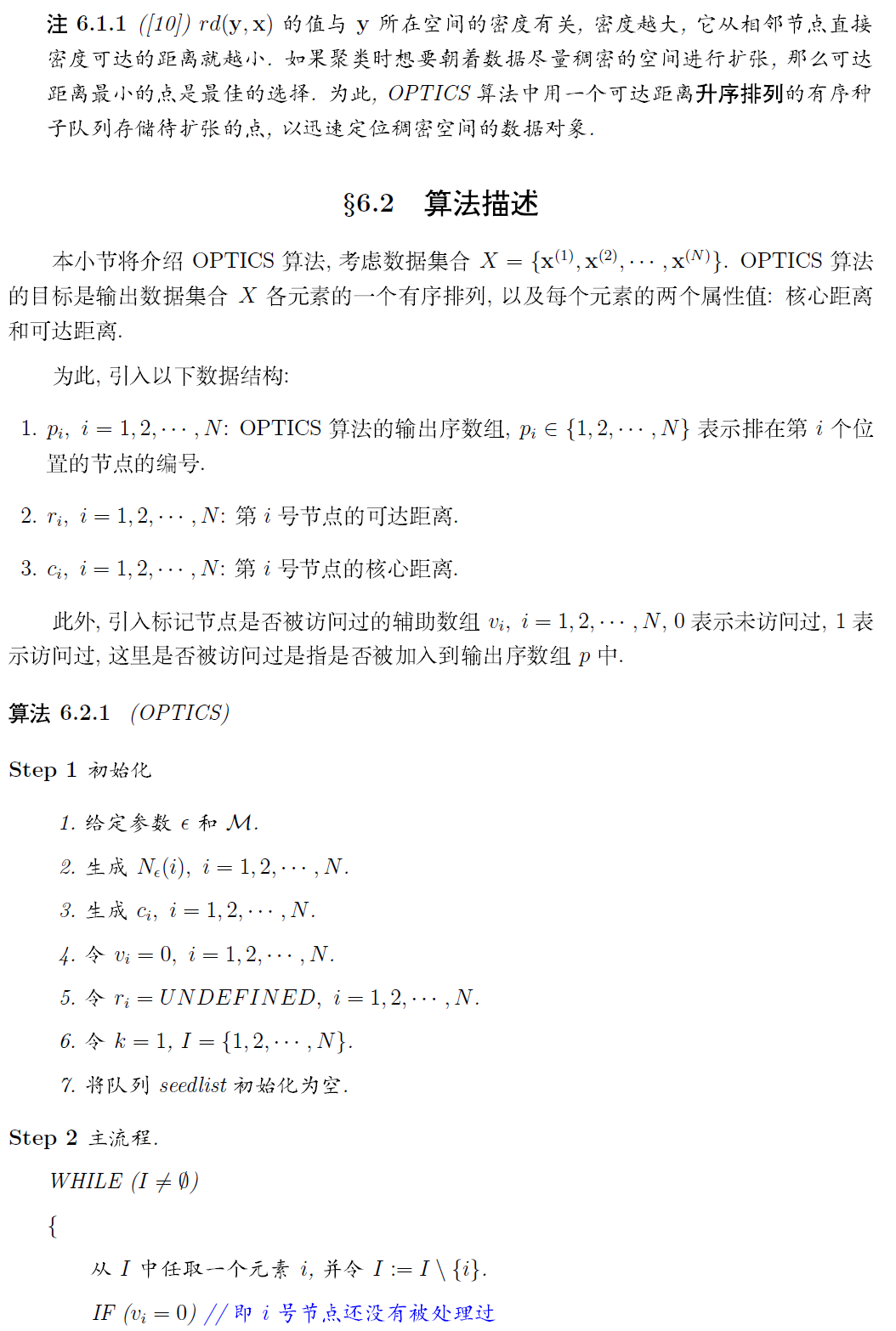
概念：

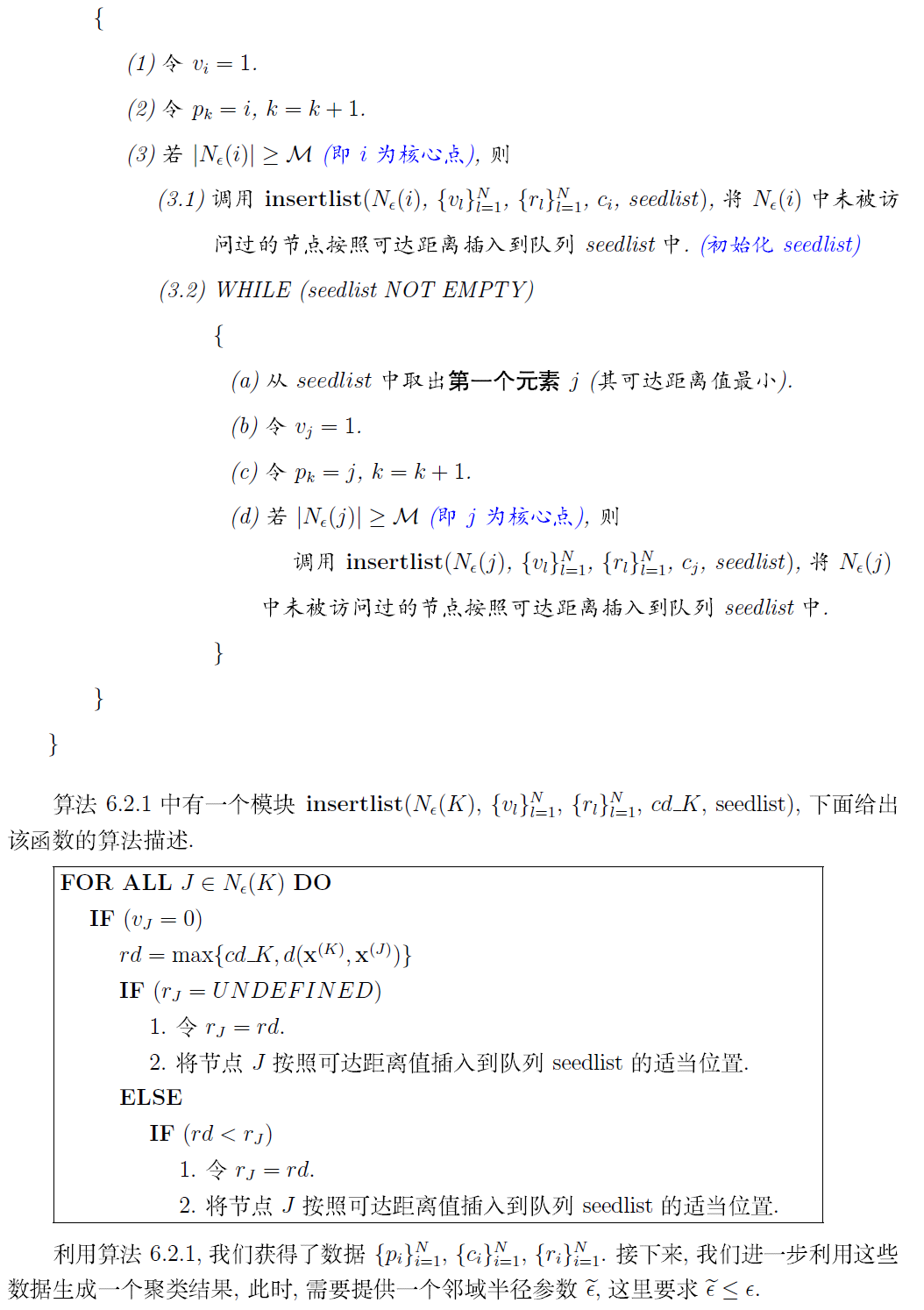
1.核心距离

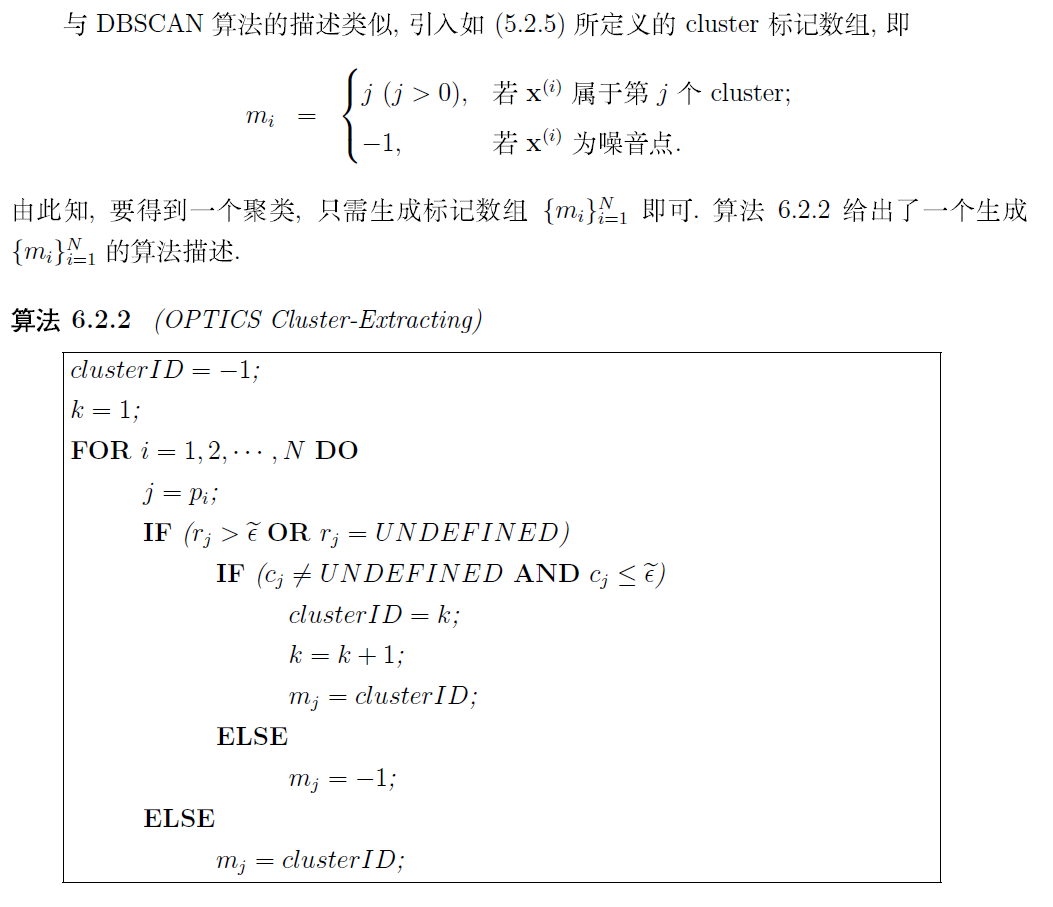
使得x称为核心点的最小邻域半径为x的核心距离。记为cd[x]

2.可达距离

X和y的可达距离表示：使得x成为核心点，且y从x直接密度可达的距离。







# 7. NLP

## TFIDF

|  |  |
| --- | --- |
| TF词频 | idf逆文档频率 |
|  |  |
|  |  |
|  |  |

## FastText

fastText的核心思想就是：将整篇文档的词及n-gram向量叠加平均得到文档向量，然后使用文档向量做softmax多分类。这中间涉及到两个技巧：字符级n-gram特征的引入以及分层Softmax分类。

我记得fastText作者的实现应该是句子中的单词再加上n-gram作为输入的。并且为了节省内存，n-gram需要经过哈希处理，哈希到同一个位置的不同n-gram是会共享一个embedding的。

### 字符级别的n-gram

word2vec把语料库中的每个单词当成原子的，它会为每个单词生成一个向量。这忽略了单词内部的形态特征，比如：“apple” 和“apples”，“达观数据”和“达观”，这两个例子中，两个单词都有较多公共字符，即它们的内部形态类似，但是在传统的word2vec中，这种单词内部形态信息因为它们被转换成不同的id丢失了。

这带来两点**好处**：

1. 对于低频词生成的词向量效果会更好。因为它们的n-gram可以和其它词共享。

2. 对于训练词库之外的单词，仍然可以构建它们的词向量。我们可以叠加它们的字符级n-gram向量。

**用你所学的知识说下怎么计算文本相似度**【百度】

1. 无监督：词袋模型、word embedding→sentence embedding，doc2vec，autoencoder
2. 有监督：深度语义匹配（DSMM、CDSSM、MVDSSM或其他，孪生网络：bilstm、交互式Attention、交叉熵&对比损失、数据不均衡、数据增强等方面说了）

## CNN和RNN的比较（文本中）

CNN：可以捕捉局部相关性，在文本分类任务中，CNN可以提取类似n-gram的关键信息。

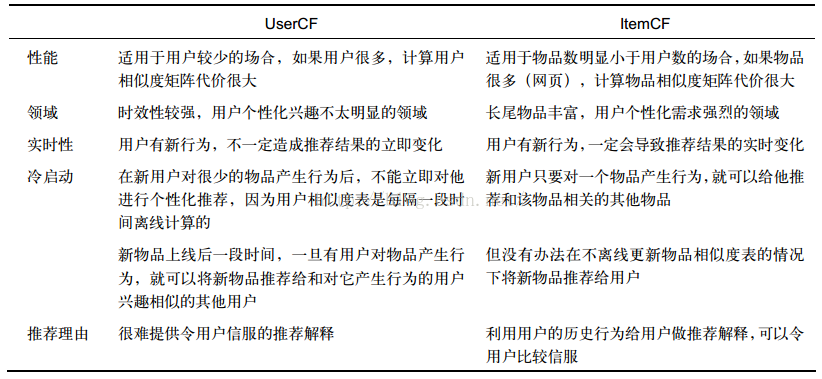
CNN在对进空间信息建模的时候，需要对特征检测器进行复制，降低了模型的效率。CNN对空间不敏感。对于语序的损失很大。很显然，我今天出去玩很不开心和我今天不出玩很开心明显有不同的感情色彩，这种句子是字典方法和CNN based model的杀手。

RNN能够很好的捕捉上下文的信息。

# 8. 推荐

**协同过滤的itemCF，userCF区别适用场景**

UserCF给用户推荐那些和他有共同兴趣爱好的用户喜欢的商品，而ItemCF给用户推荐那些和他之前喜欢的商品类似的商品；



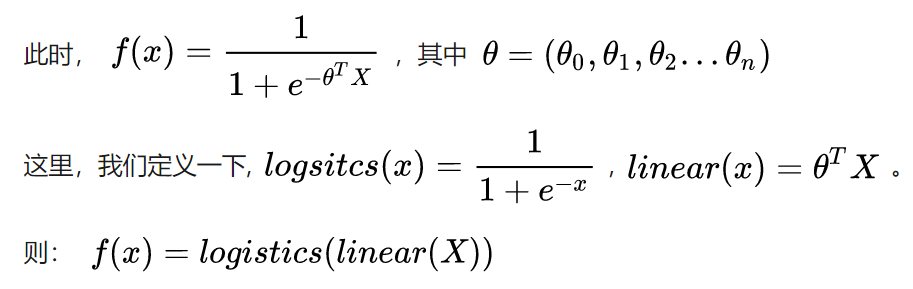
# 9. CTR

## 9.1基本概念

ctr即广告点击率，

## 9.2 模型

### 1. LR 海量高纬度 离散特征



|  |  |  |
| --- | --- | --- |
| 模型 | 优点 | 缺点 |
| LR | 1.处理离散化特征；2.而且模型十分简单，很容易实现分布式计算。  3.LR的变种：FTRL，其实这些变种都可以看成：LR+正则化+特定优化方法。 | 特征与特征之间在模型中是独立的，对于一些存在交叉可能性的特征（比如: 衣服类型与性别，这两个特征交叉很有意义），需要进行大量的人工特征工程进行交叉。虽然模型简单了，但是人工的工作却繁重了很多。而且LR需要将特征进行离散化，归一化，在离散化过程中也可能出现边界问题。 |
| GBDT | 1. 处理连续值特征；  2. 具有一定的组合特征的能力，模型的表达能力要比LR强；  3. 对特征的数值线性变化不敏感； | 1. 不善于挖掘长尾特征；  2. 特征组合能力有限，不及DNN  3. 推荐系统中出现的大多是离散特征。需要将其变成连续的，耗费时间 |
| FM | 1.相对于LR在于拥有处理二次交叉特征的能力，  2.它是可以实现线性的时间复杂度的，模型训练的快 |  |
| FMM | 在FM的基础上，考虑了特征交叉的field的特点，效果比FM好 | 没有办法实现线性的时间复杂度，模型训练要比FM慢一个量级 |
| MLR | 相对于LR拥有更好的非线性表达能力 | 1.需要较大的特征工程处理；  2.这个模型本身属于非凸模型，可能出现不收敛的情况。 |
| Dnn | 1. 模型表达能力强，能够学习出高阶非线性特征。  2. 容易扩充其他类别的特征，比如在特征拥有图片，文字类特征的时候。 |  |

### 2. GBDT

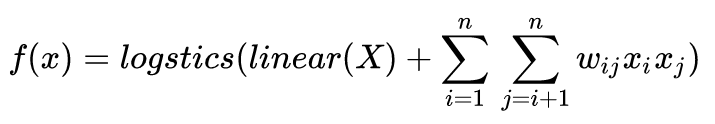
梯度提升决策树，是一种表达能力比较强的非线性模型。



**GBDT的优势**在于处理连续值特征，比如用户历史点击率，用户历史浏览次数等连续值特征。而且由于树的分裂算法，它具有一定的组合特征的能力，模型的表达能力要比LR强。GBDT对特征的数值线性变化不敏感，它会按照目标函数，自动选择最优的分裂特征和该特征的最优分裂点，而且根据特征的分裂次数，还可以得到一个特征的重要性排序。所以，使用GBDT减少人工特征工程的工作量和进行特征筛选。

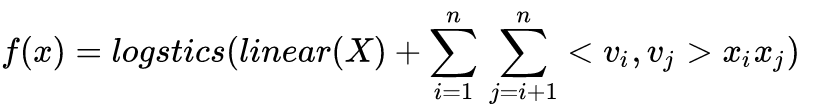
GBDT善于处理连续值特征，但是推荐系统的绝大多数场景中，出现的都是大规模离散化特征，如果我们需要使用GBDT的话，则需要将很多特征统计成连续值特征（或者embedding），这里可能需要耗费比较多的时间。同时，因为GBDT模型特点，它具有很强的记忆行为，不利于挖掘长尾特征，而且GBDT虽然具备一定的组合特征的能力，但是组合的能力十分有限，远不能与dnn相比。

### 3. FM 与 FFM

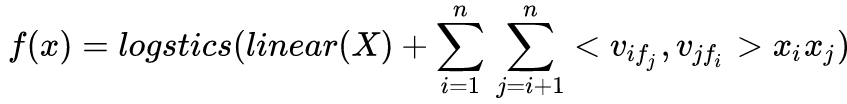
在LR部分，我们提及到了LR不能自动处理交叉特征。而FM则是在这个基础上进改进。FM的模型公式为: 

但是二次项权重  需要我们去存储一个二维矩阵的变量，而因为特征是大规模离散的，这个二维矩阵的维度可能很大。而FM的作者利用矩阵分解的原理，将这个权重矩阵进行分解，即  。

则FM的公式变为：



但是这里，对于特征都是使用做内积。但是对于不同的特征组合，比如天气与地点，天气与性别，关联的程度是不一样的，都使用同样的向量去与不同的特征做内积，会带来明显的信息损失。所以引出了FFM（field FM）。

FFM的模型公式为：

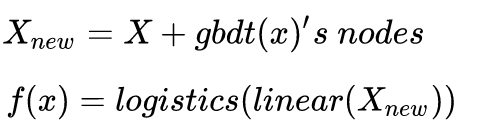
### 4. GBDT+(LR, FM, FFM)

前面提及到了**GBDT适合处理连续值特征，而LR，FM，FFM更加适合处理离散化特征。GBDT可以做到一定程度的特征组合，**而且GBDT的特征组合是多次组合的，不仅仅是FM与FFM这样的二阶组合而已。同时，**GBDT具备一定的特征选择能力（选择最优的特征进行分裂）**

在facebook 2014年的一篇论文中，提及到GBDT+LR的解决方案。即先使用GBDT对一些稠密的特征进行特征选择，得到的叶子节点，再拼接离散化特征放进去LR进行训练。在方案可以看成，利用GBDT替代人工实现连续值特征的离散化，而且同时在一定程度组合了特征，可以改善人工离散化中可能出现的边界问题，也减少了人工的工作量。

而kaggle的ctr预估比赛中，台大的团队参考fb,使用了gbdt+ffm的方案夺冠。

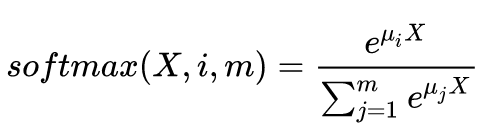
此时GBDT+LR模型的公式为：



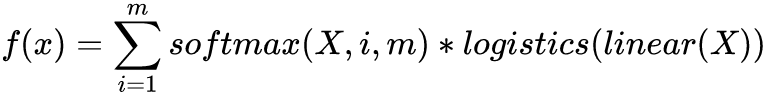
### 5.MLR

MLR是由阿里团队提出的一种非线性模型。它等价于聚类+LR的形式。

我们定义：



则MLR的公式为：



相当于将X进行聚类，聚成m类，然后每一个聚类单独训练一个LR。

MLR相对于LR拥有更好的非线性表达能力，算是LR的一种拓展。

但是MLR与LR一样，也需要较大的特征工程处理，而且这个模型本身属于非凸模型，需要预训练，可能出现不收敛的情况。

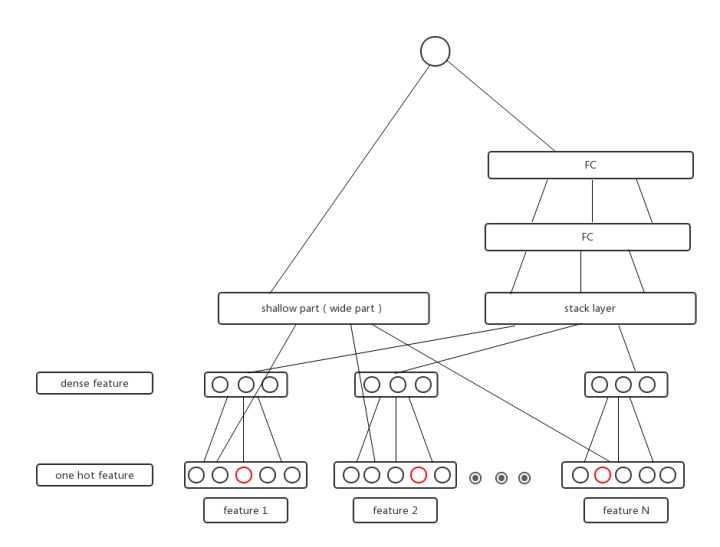
### 6.DNN

目前主流处理ctr预估问题的dnn框架，基本都是由shallow part + deep part组合而成的。

其通用公式可表示为： 

|  |  |  |
| --- | --- | --- |
|  | shallow part | stack layer |
| deep & wide | LR | concatenate（拼接embedding后的向量） |
| DeepFM | fm | 同上 |
| NFM model | lr | Bi-interaction，实际上就是先做向量的内积（与FM类似，但没有累加），然后再做累加。 |
| NFFM模型 | lr | ffm embedding + fcs（全连接层） |

ctr预估问题中dnn模型的主要架构



基本上目前已有的dnn模型基本都是和这个框架类似，只是stack layer的方法不一样，有无shallow part，或者shallow part异同（lr or fm等部分）。

其他模型也是与这些模型类似。其实dnn做ctr问题，关键的部分无非三部分：

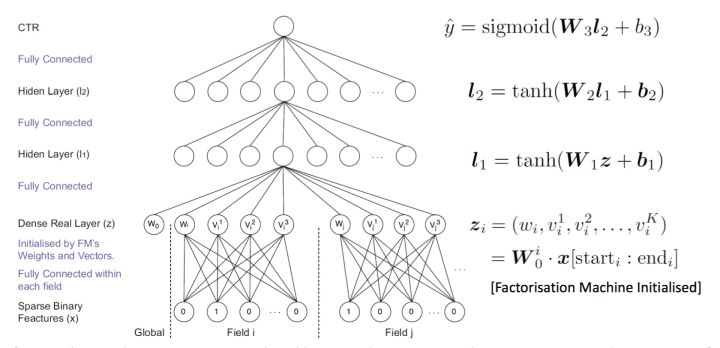
1. 是否保留浅层模型 （存在不保留shallow part的model，比如FNN，PNN）

2. 如何体现特征交叉性，即stack layer的做法是什么。concatenate？Bi-interaction？直接向量加法？。。。等等

3. embedding + fc（全连接层）是基本标配。

### 7. FNN

FNN模型只包含了deep部分，没有shallow部门（lr or fm），而特征之间的交叉提现为拼接（concatenate），然后利用几个fcs加深模型。



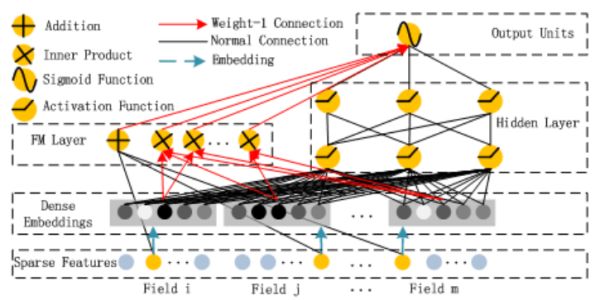
FNN是简单的embedding+fcs，只是使用了FM做预训练。站在宏观的ctr的dnn模型中来看，FNN没有shallow part(lr or fm/ffm)，只有deep part，而特征之间的交叉性提现在了拼接(concatenate)上。FNN的效果上限是很大程度取决于FM的预训练效果的，如果不做预训练的话，就更加难出很好的效果。而且特征的交叉的方式是使用拼接(concatenate)来提现的，其实拼接后，再接fcs，虽然使用了激活函数增加了非线性，实际上是对特征进行了加权组合(add 操作)。

### 8. DeepFM

FM是一个简单的三层网络，它的核心部分是第二层的FM layer，FM layer由两部分组成，一部分是lr部分，令一部分是inner product部分。图中的红色箭头代表权重为1的连接，是不更新的(在实际的代码实现中不需要管，直接内积或者求和就好，这里只是为了更加直观地从神经网络的角度了解FM)。

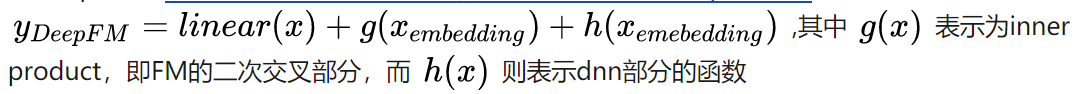
而Deep network，则是一个简单的embedding+fcs，不需要过多讲解。

DeepFM模型其实就是将这两个模型组合起来了。



红色的线是权重为1的线，不需要更新。

DeepFM神经网络部分，隐含层的激活函数用ReLu和Tanh做信号非线性映射，Sigmoid函数做CTR预估的输出函数。



### 9.wide&deep

Wide：高维特征+特征组合的LR

Deep就是前馈神经网络，

联合训练是指同时训练Wide模型和Deep模型，并将两个模型的结果的加权和作为最终的预测结果

训练的方法：

* Wide模型：FTRL
* Deep模型：AdaGrad

## 9.3 比赛

### 1. 腾讯广告算法大赛

#### 特征部分

广告特征

特征

onehot特征

用户特征：

不定长的向量特征

**特征分析的思路**：看分布。

**特征挖掘思路**：越细粒度的特征越重要，时间越近的特征越重要，未来的特征简直无敌。

粗粒度的特征：年龄，性别等。

这次比赛中的：interest、topic、keyword

**粒度太细的特征，是不可以直接放进去模型的。**比如uid，假设将uid进行onehot编码，因为很多uid都只出现了一次，在一轮训练中，每个uid只会被训练一次，显然特征对应的权重的置信度是很低的。所以对于很细粒度的特征，又要利用一些手段来进行**维度的压缩。**其实统计uid的出现次数就是对uid的一个维度压缩，即利用**uid的出现次数**来表达对应的这个uid。然而，压缩必然导致信息损失，如果仅仅使用uid的出现次数来表示uid，那么uid的label信息就没有得到利用了。所以，还需要考虑**使用uid的转化率或者正样本数**来对uid来进行表示。

知乎Jachin特征方案。

本次比赛中，在DNN模型中，我的特征主要包含一下6类:

（1）原始onehot特征，比如aid，age，gender等。  
（2）向量特征，比如interest1，interest2，topic1，kw1等  
（3）向量长度统计特征：interest1，interest2，interest5的长度统计。  
（4）uid类的统计特征，uid的出现次数，uid的正样本次数，以及uid与ad特征的组合出现次数，组合正样本次数。  
（5）**uid的序列特征**，比如uid=1时，总共出现了5次，序列为[-1,1,-1,-1,-1]，  
则第一次出现时，特征为【】  
第二次出现时，特征为【-1】  
第三次出现时，特征为【-1，1】  
第四次出现时，特征为【-1，1，-1】  
第五次出现时，特征为【-1，1，-1，-1】  
（6）组合特征：age与aid的组合，gender与aid的组合，interest1与aid的组合，interest2与aid的组合，topic1与topic2的组合，LBS与kw1的组合。

出现的次数和转化率。和其他特征做组合，用lgb训练，进行特征重要性排序，选出topK。 进行相关系数分析，过滤掉相关程度很强的特征。

比赛Tips：

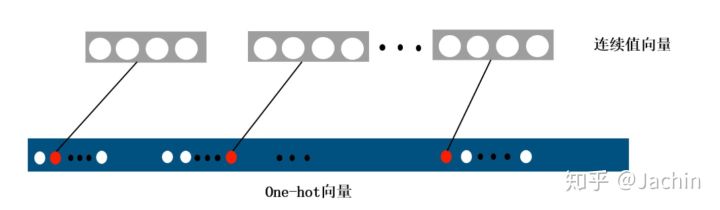
1. 线上和线下效果不一致的情况。选择2%-5%的训练集数据作为验证集；最好是构造两套特征存储，一套拿来看线下验证集，一套训练来提交。

2. k-fold统计。

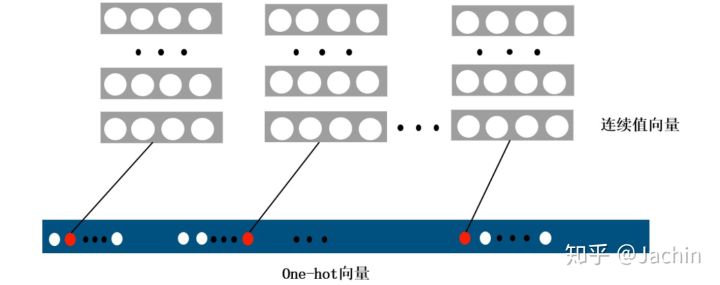
3. 不定长特征的统计表达。在interest，keyword，topic等不定长的特征中，一般可以取max来进行特征表达。比如interest1有在某一行数据中，有5个值【1，10，30，50，100】，每个值对应的转化率为【0.001，0.02，0.05，0.0015，0.1】，那么就用0.1来表示就好了。 当然，如果觉得信息损失很大，也就取topk，比如top3就是[0.1, 0.05, 0.02]

#### 模型：NFFM模型

FM的embedding如下图：



FFM的embedding方法如下图：



其实简单点来讲，FM对于每一个特征只会生成一个连续值特征向量，与其他特征做交叉时，都会使用这一个特征向量。而FFM中，每一个特征会生成多个特征向量，与不同的特征做组合时，会用不同的特征向量做组合。从本质来讲，FFM比FM的效果要好，其实就是参数的数量更多了。但是它又和仅仅增加embedding size来达到增加参数的数量不一样，它的效果是要略好于单纯增加embedding size的。

# 10. 面经

## 360

### 最小编辑距离的问题

### 如何实现LRU，用到哪些数据结构