**面经**

<https://blog.csdn.net/yasin0/article/details/82319649>

一、机器学习

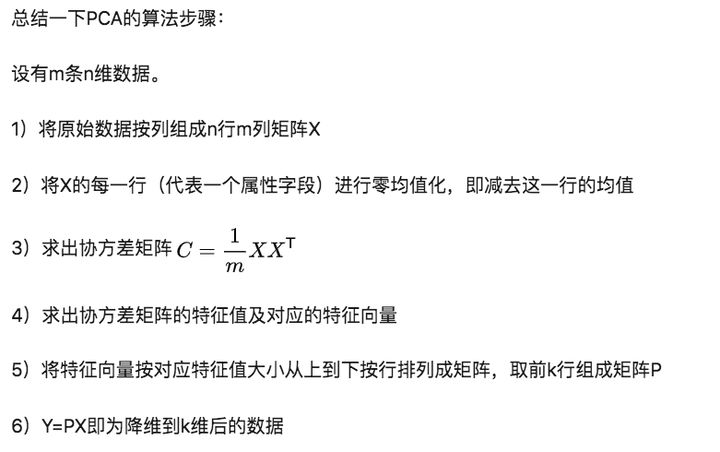
（1）PCA合集

1.1 简述PCA

**PCA是比较常见的线性降维方法,通过线性投影将高维数据映射到低维数据中,所期望的是在投影的维度上,新特征自身的方差尽量大,方差越大特征越有效,尽量使产生的新特征间的相关性越小**。

PCA算法的具体操作为对所有的样本进行中心化操作,计算样本的协方差矩阵,然后对协方差矩阵做特征值分解,取最大的n个特征值对应的特征向量构造投影矩阵。

1.2 PCA具体计算步骤



①相关理解：3中的协方差矩阵的公式与下面公式一样，知识下面的公式自带了中心化（即减去均值，从方差的定义即可知道）



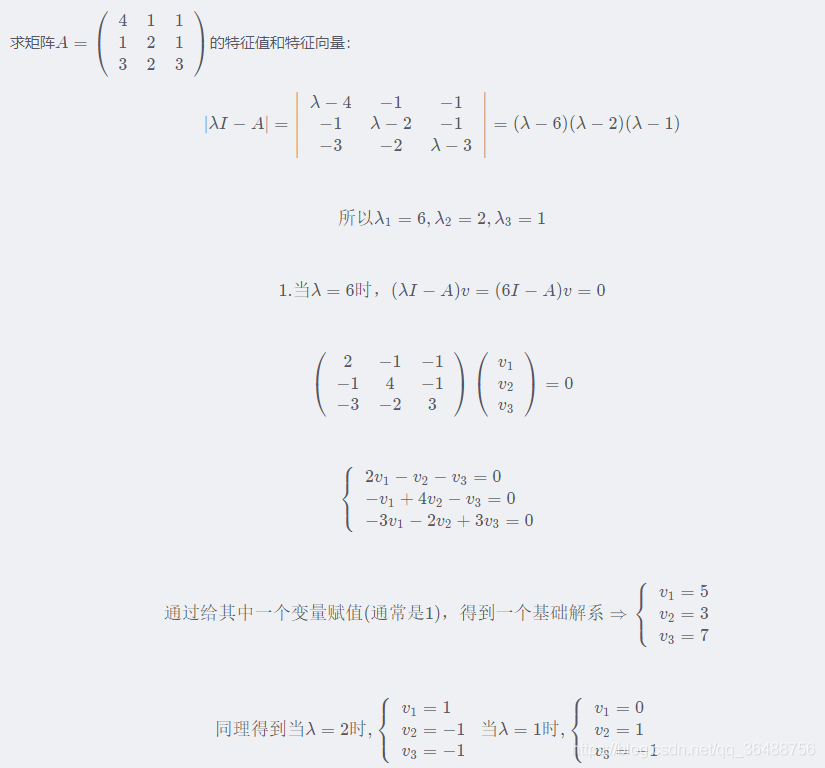
当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个线性变化可能没法通过图片来表示，但是可以想象，这个变换也同样有很多的变换方向，**我们通过特征值分解得到的前N个特征向量，那么就对应了这个矩阵最主要的N个变化方向**。我们利用这前N个变化方向，就可以近似这个矩阵（变换）。也就是之前说的：提取这个矩阵最重要的特征。总结一下，特征值分解可以得到特征值与特征向量，**特征值表示的是这个特征到底有多重要**，**而特征向量表示这个特征是什么，可以将每一个特征向量理解为一个线性的子空间**，我们可以利用这些线性的子空间干很多的事情。

**不过，特征值分解也有很多的局限，比如说变换的矩阵必须是方阵。**

1.3 特征值分解过程

<https://blog.csdn.net/qq_36488756/article/details/110285409>

m维随机变量的协方差矩阵是实对称矩阵，故也为方阵，故可以特征值分解



1.4 PCA为什么要用协方差矩阵的特征向量矩阵来做投影矩阵呢

<https://zhuanlan.zhihu.com/p/58663947?from_voters_page=true>

降维的目的就是“降噪”和“去冗余”。

**“降噪”的目的就是使保留下来的维度间的相关性尽可能小**，而“去冗余”的目的就是**使保留下来的维度含有的“能量”即方差尽可能大**。

我们要最大化方差来保留更多的信息。去噪。

有趣的是，协方差矩阵能同时表现不同维度间的相关性以及各个维度上的方差。

协方差矩阵度量的是维度与维度之间的关系，而非样本与样本之间。协方差矩阵的主对角线上的元素是各个维度上的方差(即能量)，其他元素是两两维度间的协方差(即相关性)。

**先看“降噪”，让保留下的不同维度间的相关性尽可能小，也就是说让协方差矩阵中非对角线元素都基本为零**。达到这个目的的方式——**矩阵对角化**。

**再看“去冗余”，对角化后的协方差矩阵，对角线上较小的新方差对应的就是那些该去掉的维度。我们只取那些含有较大能量(特征值)的维度，其余的就舍掉即可。（所以要按特征值从大到小排序）**

1.5 给你一个数据集。该数据集包含很多变量，你知道其中一些是高度相关的。经理要求你用PCA。你会先去掉相关的变量吗？为什么？

不会。

丢弃相关变量会对PCA有实质性的影响，因为有相关变量的存在，由特定成分解释的方差被放大。

例如：在一个数据集有3个变量，其中有2个是相关的。如果在该数据集上用PCA，第一主成分的方差会是与其不相关变量的差异的两倍。此外，加入相关的变量使PCA错误地提高那些变量的重要性，这是有误导性的。

**一维坐标上有一些点，加入一个相关变量，也就是接近线性相关的特征，此时再用pca则会导致斜线45°才是二维特征的主成分，此时的长度变为原来的根号2倍。方差需要平方，则方差为原来的两倍。**

1.6在PCA中有必要做旋转变换吗？如果有必要，为什么？如果你没有旋转变换那些成分，会发生什么情况？

是的，旋转（正交）是必要的，**因为它把由主成分捕获的方差之间的差异最大化**。这使得主成分更容易解释。但是不要忘记我们做PCA的目的是选择更少的主成分（与特征变量个数相较而言），那些选上的主成分能够解释数据集中最大方差。

通过做旋转，各主成分的相对位置不发生变化，它只能改变点的实际坐标。如果我们没有旋转主成分，PCA的效果会减弱，那样我们会不得不选择更多个主成分来解释数据集里的方差。**（减少主成分就可以实现降维，没有旋转变换就需要更多特征，例如李航298页的斜椭圆，没有变换需要二维特征，如果沿45°可能一维就可以表示大部分信息）**

假设矩阵X，有变换矩阵P，变换后的矩阵为Y = PX。

1. 如何处理过拟合和欠拟合

(数据增强，然后L1正则化，L2正则化；dropout)

<https://zhuanlan.zhihu.com/p/85373781>

过拟合：在训练集上误差小，但在测试集上误差大。

过拟合解决方法：

①正则化(Regularization)；

②降低模型复杂度，如决策树深度，神经网络的层数等等

③获取更多数据，如图像平移、旋转。

深度学习：

①随机失活(dropout)；（深度学习合集中有介绍）

②逐层归一化(batch normalization)：（深度学习中有介绍）Batch Normalization（以下称BN）的主要作用是加快网络的训练速度。如果硬要说是防止过拟合，可以这样理解：BN每次的mini-batch的数据都不一样，但是每次的mini-batch的数据都会对moving mean和moving variance产生作用，可以认为是引入了噪声，这就可以认为是进行了data augmentation，而data augmentation被认为是防止过拟合的一种方法。因此，可以认为用BN可以防止过拟合。但是个人认为对防止过拟合基本没有多大用，还是用来加速网络收敛。

③降低神经网络结构的复杂度

④提前终止(early stopping)；

Bagging（随机森林）

欠拟合：在训练集上训练效果不好（测试集上也不好），准确率不高。

解决方法：

①添加其他特征项；

②为了提高复杂关系的拟合能力， 在特征工程中经常会把一阶离散特征两两组合，构成高阶组合特征；

②增加模型复杂度；

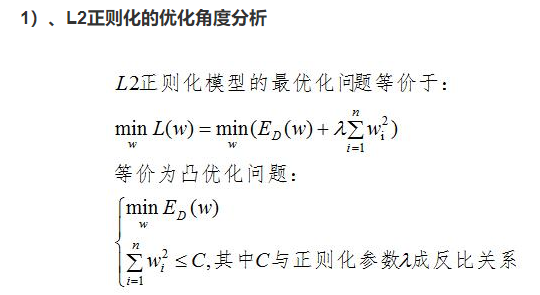
③降低正则化系数；

1. 对L1和L2正则化的理解

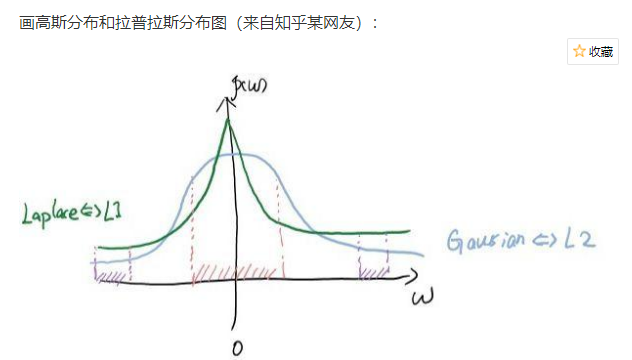
（在解空间碰撞出稀疏解，可以使一些特征参数为零）

<https://baijiahao.baidu.com/s?id=1621054167310242353&wfr=spider&for=pc>





L1是通过稀疏参数（减少参数的数量）来降低复杂度，L2是通过减小参数值的大小来降低复杂度。



**由上图可知，若假设参数w\_{j}服从拉普拉斯分布。拉普拉斯分布在参数w=0点的概率最高，因此L1正则化相比于L2正则化更容易使参数为0；高斯分布在零附近的概率较大，因此L2正则化相比于L1正则化更容易使参数分布在一个很小的范围内。**

1. Svm合集

<https://zhuanlan.zhihu.com/p/49331510?utm_source=wechat_session&utm_medium=social&utm_oi=759055729532346368>

4.1介绍一下SVM

SVM就是一种二分类模型，他的基本模型是的定义在特征空间上的间隔最大的线性分类器，SVM的学习策略就是间隔最大化。

4.2遇到线性不可分怎么办

当训练数据线性可分时，通过硬间隔最大化，学习一个线性的分类器。

当训练数据近似线性可分时，通过软间隔最大化，也可以学习得到一个线性的分类器。

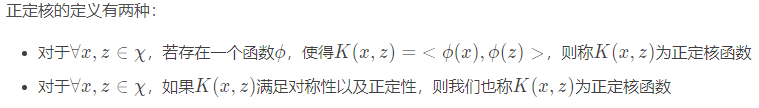
**遇到线性不可分怎么办：通过使用核技巧及软间隔最大化，学习非线性支持向量机**

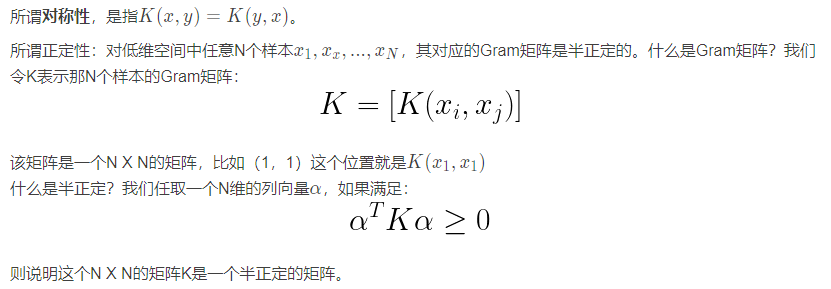
4.3核函数有什么特点

**核函数特点：当输入空间为欧式空间或离散集合，特征空间为希尔伯特空间时，核函数表示输入空间映射到特征空间得到的特征向量之间的內积。**

**正定核的充要条件：李航p139有证明**







4.4 解释下支持向量机对偶算子的用法

即对于原始的凸优化问题，通过求解其对偶问题得到原始问题的最优解。

目的：①对偶问题更容易求解；②自然引入核函数推广到非线性问题。

4.5 svm的求解方法

见统计学习方法的SMO算法

SMO算法：先选择两个变量，固定其他变量，针对这两个变量进行求解。选取变量时必须选择一个违反KKT条件最严重的一个（因为最终解肯定满足KKT条件），就是yi、g（xi）的积和1的情况来判断是否偏离KKT条件。另一个可以选择|E1-E2|最大的来判断（标准是更新后变化足够大）。

4.6支持向量机(SVM)是否适合大规模数据

我理解SVM并不是不适合大规模数据，而应该说，SVM在小样本训练集上能够得到比其它算法好很多的结果。

支持向量机之所以成为目前最常用，效果最好的分类器之一，在于其优秀的泛化能力，这是是因为其本身的优化目标是结构化风险最小，而不是经验风险最小，因此，通过margin的概念，得到对数据分布的结构化描述，因此减低了对数据规模和数据分布的要求。

4.7 svm的损失函数

本身的优化目标是结构化风险最小，名叫合页损失函数



4.8数学基础补充：

欧式空间：欧几里得空间就是在对现实空间的规则抽象和推广（从n<=3推广到有限n维空间）。

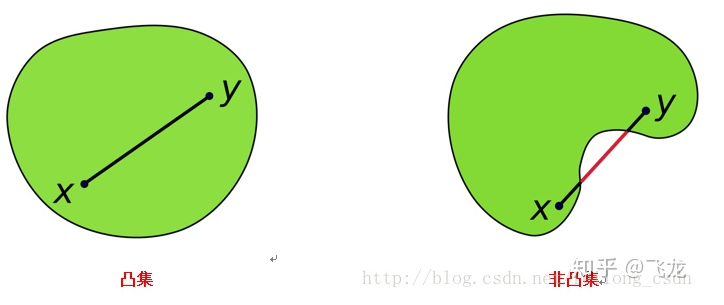
希尔伯特空间：希尔伯特空间也是一个内积空间；具备完备性，其中完备性的意思就是空间中的极限运算不能跑出该空间，如有理数空间中的根号2的小数表示，其极限随着小数位数的增加收敛到根号2，但根号2属于无理数，并不在有理数空间，故不满足完备性。

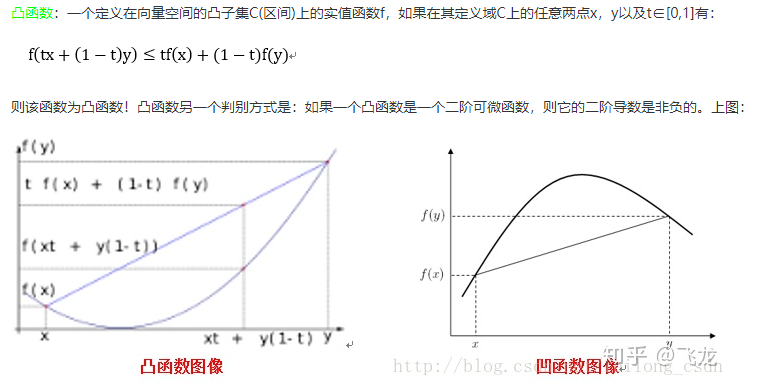
输入空间和特征空间：

输入空间可以看作是未处理的数据集.

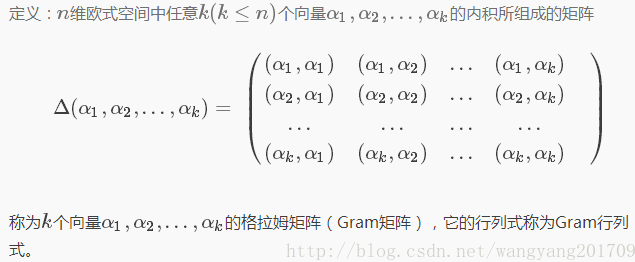
特征空间是指经过特征工程处理过的输入空间，将输入空间中隐藏的特征显现出来。SVM的核函数，将输入从输入空间映射到特征空间得到特征向量之间的内积。通过在这个特征空间上学习线性支持向量机。

凸优化：目标函数和约束函数都是凸函数。凸函数就是一个定义在某个向量空间的凸子集C（区间）上的实值函数。





Gram矩阵：



4.7 核函数怎么进行构造

1. xgboost合集

5.1介绍一下xgb（与gbdt对比）

<https://zhuanlan.zhihu.com/p/142437971?from_voters_page=true>

<https://blog.csdn.net/yinyu19950811/article/details/81079192>

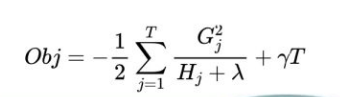
<https://zhuanlan.zhihu.com/p/269201158?utm_source=wechat_session&utm_medium=social&utm_oi=759055729532346368>

1. GBDT在模型训练时只使用了代价函数的一阶导数信息，XGBoost对代价函数进行二阶泰勒展开，可以同时使用一阶和二阶导数：

在GBDT中，我们每次生成下一个弱学习器，都是把损失函数的梯度作为学习目标，相当于利用梯度下降法进行优化来逼近损失函数的最小值，也就是使得损失函数为0，最终学习器尽可能接近真实结果。（梯度下降法就是用一阶泰勒展开来近似函数）

而xgboost中，我们则是把损失函数的二阶泰勒展开的差值作为学习目标，相当于利用牛顿法进行优化，来逼近损失函数的最小值，也就是使得损失函数为0（牛顿法则是用二阶泰勒展开来近似函数）

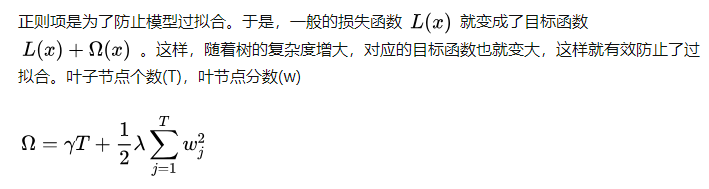
则xgboost的目标函数为



其中G代表一阶导、H代表二阶导。

2、正则项

xgboost优于GBDT的一个特性是它在代价函数中加入了正则化项，用于控制模型的复杂度，新增的是叶子节点输出L2平滑。学习出来的模型更加简单，防止过拟合。



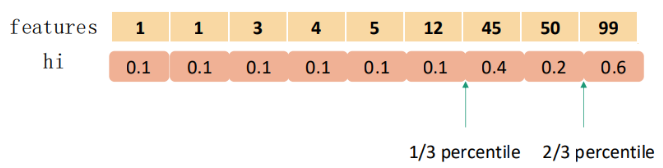
3、树节点的分裂方法

<https://blog.csdn.net/yilulvxing/article/details/106180900>

①精确算法：遍历所有特征的所有可能分割点，来寻找使目标函数最小的分割点。

②近似算法：对于每个特征，只考察分位点，减少计算复杂度。

而XGBoost不是简单地按照样本个数进行分位，而是以二阶导数值作为权重(Weighted Quantile Sketch)，比如:



1. 传统的GBDT采用以CART作为基分类器，XGboost支持多种类型的基分类器，比如线性分类器
2. 传统的GBDT在每轮迭代时采用全部的数据，XGboost则采用了与随机森林相似的策略；支持对数据进行采样
3. 传统的GBDT没有设计对缺失值的处理，XGboost能够自动学习出缺失值的处理策略。

总的来说即xgboost在构建树的时候不考虑有缺失值的样本，在训练集中有缺失值的样本可以分别算左右子树归到最大的增益中去。并将其设置为默认的分支（每一个树结点会记住哪个分支让缺失值进去会更好）。如果仅仅在测试集中有缺失值，即没有学习到默认分支，则直接将其归到默认的右结点。

而CART树这类默认处理缺失值时，如果训练某个样本时，某个存在缺失值的特征恰好是当前的分裂增益最大的特征。则需重新遍历其他特征，去选择另外的特征作为当前结点的分类标准。

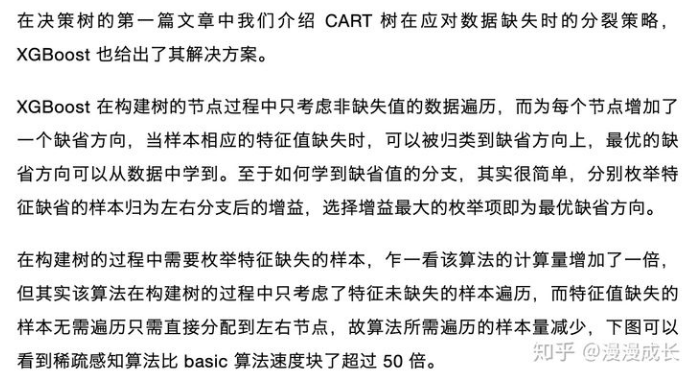
①CART树的缺失值处理：

首先，如果某个存在缺失值的特征恰好是当前的分裂增益最大的特征，那么我们需要遍历剩余的特征，剩余的特征中如果有也存在缺失值的特征，那么这些特征忽略，仅仅在完全没有缺失值的特征上进行选择，我们选择其中能够与最佳增益的缺失特征分裂之后增益最接近的特征进行分裂。

如果我们事先设置了一定的标准仅仅选择仅仅选择差异性在一定范围内的特征作为代理特征进行分裂而导致了没有特征和最佳缺失特征的差异性满足要求，或者所有特征都存在缺失值的情况下，缺失样本默认进入个数最大的叶子节点。

显然这种缺失值的处理方式的计算量是非常大的，我们需要遍历其它的特征来进行代理特征选择，这个在数据量很大的情况下开销太大，而带来的性能提升确很有限，所以后来就不怎么用这种处理方式

②xgboost把缺失值当做稀疏矩阵来对待，本身的在节点分裂时不考虑的缺失值的数值。缺失值数据会被分到左子树和右子树分别计算损失，选择较优的那一个。如果训练中没有数据缺失，预测时出现了数据缺失，那么默认被分类到右子树



1. XGboost可以实现并行处理：块结构可以很好地支持并行计算（原因见：<https://zhuanlan.zhihu.com/p/269201158?utm_source=wechat_session&utm_medium=social&utm_oi=759055729532346368）>





缺点：①虽然利用预排序和近似算法可以降低寻找最佳分裂点的计算量，但在结点分裂过程中依然需要遍历数据集。

②**预排序过程的空间复杂度过高，不仅需要存储特征值，还需要存储特征对应样本的梯度统计值的索引（存梯度信息的目的是每次在制定的缓存区连续寻找样本梯度即可）**，**相当于消耗了两倍的内存（因为每个样本都对应了梯度）**

5.2 xgb的boosting如何体现，有什么特殊含义；

葫芦书P289

Boosting的训练过程:在训练好一个弱分类器后，我们需要计算弱分类器的错误或者残差，作为下一个分类器的输入。这个过程本身就是在不断减小损失函数，来使模型不断逼近靶心。

5.3 xgb能否处理离散特征，为什么，如果要用怎么处理；

<https://blog.csdn.net/zc02051126/article/details/79094655/>

在Xgb中需要将离散特征one-hot编码，和连续特征一起输入训练，这样做是为了达到在cart树中处理离散特征的方式一致，即每次选择一个离散特征对应的样本作为一类，剩下的所有特征值对应的样本作为一类。不按照扫描切分，因为扫描切分会导致后续的子树中特征组合变少。

**无论分类树还是回归树，只是最终预测有离散和连续之分。而特征都是不擅长处理高维稀疏特征的。弄清楚特征和结果的区别，回归问题也可以有离散特征。**

**xgboost 对所有的输入特征都是当做数值型对待，所以你给定的数据也要是指定的数据类型。故一般数不需要独热码，而xgboost需要独热码**

5.4 xgb的分类树也是用残差吗，不是的话是什么

和传统的boosting tree模型一样，xgboost的提升模型也是采用的残差（或梯度负方向），不同的是分裂结点选取的时候不一定是最小平方损失

5.5 xgb在处理分类问题和回归问题的区别

**主要还是体现在目标函数（目标函数一般是损失函数加上正则化）和损失函数**

①objective：目标函数

二分类：常用的是binary:logistic（即sigmoid函数）

多分类：multi:softmax,当是多分类任务时需要指定类别数量，eg:'num\_class':33;

回归任务：reg:linear

**sigmoid将一个real value映射到（0,1）的区间（当然也可以是（-1,1）），这样可以用来做二分类（比如逻辑回归就是用一个线性回归拟合一个概率值，然后利用sigmoid函数映射到非0即1的过程）。**

**而softmax把一个k维的real value向量（a1,a2,a3,a4…）映射成一个（b1,b2,b3,b4…）其中bi是一个0-1的常数，然后可以根据bi的大小来进行多分类的任务，如取权重最大的一维。**

②eval\_metric：评价函数，如果该参数没有指定，缺省值是通过目标函数来做匹配，

二分类：常用auc和logloss

多分类：mlogloss

回归任务：均方误差：mse，均方根误差：rmse, 平均绝对值误差：mae

5.6 Xgboost在多分类与二分类过程中有什么不同？

综上我们可以看出**xgboost多分类问题就是通过softmax函数做映射得到最终的分类标签，而二分类是用sigmoid做映射得到最终的分类标签（分类零则用0去做残差，分类1则用1去做残差，多分类可见20.6详解）**，只是怎么把回归的结果映射成概率矩阵

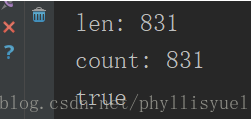
二分类：常用的是binary:logistic

多分类：multi:softmax,当是多分类任务时需要指定类别数量，

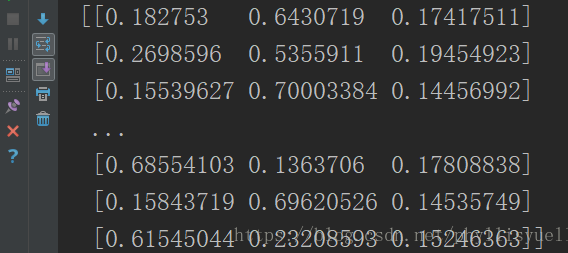


①objective='multi:softmax', # 多分类！！！！！！

multi：softmax是使用softmax后产生的分类结果



而multi:softprob是输出的概率矩阵



5.7 随机森林和XGBoost的区别

（说了他们属于不同的bagging/boosting大家族，弱学习器之间是否有依赖关系，xgb损失函数的优化，泰勒展开；xgb相对于RF并行上面的优化；）

5.8 为什么xgboost不擅长处理高维稀疏的向量（不是不擅长处理离散值）

高维稀疏的ID类特征会使树模型的训练效率变得极为低效，且容易过拟合。树模型训练过程是一个贪婪选择特征的算法，要从候选特征集合中选择一个使分裂后信息增益最大的特征来分裂（**高维稀疏矩阵会增加特征数量**）。按照高维的ID特征做分裂时（独热码转换为多个取值为0、1的连续特征），子树数量非常多，计算量会非常大，训练会非常慢。同时，按ID分裂得到的子树的泛化性也比较弱，由于只包含了对应ID值的样本，**样本稀疏时（即对应的0、1标签某一类很少）**也很容易过拟合。

数学原理补充：

https://blog.csdn.net/mxg1022/article/details/80702264

①牛顿法

https://blog.csdn.net/itplus/article/details/21896453

1. lightgbm合集

<https://blog.csdn.net/abcdefg90876/article/details/106935080>

6.1 lgb能否处理离散特征，模型做了哪些操作

<https://blog.csdn.net/abcdefg90876/article/details/106935080>

<https://www.pianshen.com/article/886263472/>

<https://blog.csdn.net/anshuai_aw1/article/details/83275299>

1. 离散特征建立直方图的过程

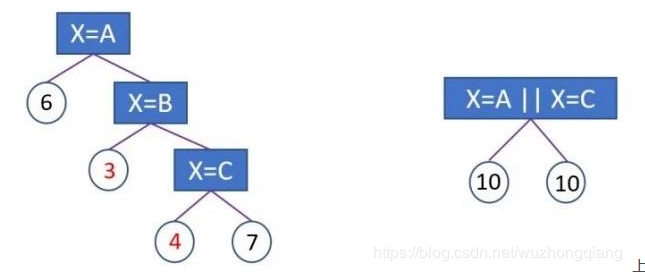
统计该特征下每一种离散值出现的次数，并从高到低排序，并过滤掉出现次数较少的特征值, 然后为每一个特征值，建立一个bin容器, 对于在bin容器内出现次数较少的特征值直接过滤掉，不建立bin容器。（类似于连续值放入直方图）

1. 计算分裂阈值的过程：

a、先看该特征下划分出的bin容器的个数，如果bin容器的数量小于4，直接使用one vs other方式, 逐个扫描每一个bin容器，找出最佳分裂点;

b、对于bin容器较多的情况, 先进行过滤，只让子集合较大的bin容器参加划分阈值计算, 先把直方图按照**每个类别对应的label均值进行排序**；根据该值对bin容器从小到大进行排序，然后分从左到右、从右到左进行搜索，得到最优分裂阈值。但是有一点，没有搜索所有的bin容器，而是设定了一个搜索bin容器数量的上限值，程序中设定是32，即参数max\_num\_cat。

LightGBM中对离散特征实行的是many vs many 策略，这32个bin中最优划分的阈值的左边或者右边所有的bin容器就是一个many集合，而其他的bin容器就是另一个many集合（**many对many可以有效减少树的分支数量，根据label均值分为左边和右边部分**）。**下图左边是正常对独热码进行划分。右边是对many对many。即右图含义是 X=A或者X=C 放到左孩子，其余放到右孩子, 右边的切分方法，数据会被切分到两个比较大的空间，进一步的学习也会更好**



C、对于连续特征，划分阈值只有一个，对于离散值可能会有多个划分阈值，每一个划分阈值对应着一个bin容器编号，当使用离散特征进行分裂时，只要数据样本对应的bin容器编号在这些阈值对应的bin集合之中，这条数据就加入分裂后的左子树，否则加入分裂后的右子树

6.2介绍xgb和lgb，改进了什么

<https://zhuanlan.zhihu.com/p/99069186>

XGBoost的缺点：这种构建决策树的算法基本思想是：首先，对所有特征都按照特征的数值进行预排序。其次，在遍历分割点的时候用O(#data)的代价找到一个特征上的最好分割点。最后，在找到一个特征的最好分割点后，将数据分裂成左右子节点。

XGBoost使用预排序后需要记录特征值及其对应样本的统计值的索引，而 LightGBM 使用了直方图算法将特征值转变为 bin 值，且不需要记录特征到样本的索引，将空间复杂度从 ，极大的减少了内存消耗；

这样的预排序算法的优点是能精确地找到分割点。

但是缺点也很明显：①空间消耗大。这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如，为了后续快速的计算分割点，保存了排序后的索引），这就需要消耗训练数据两倍的内存。

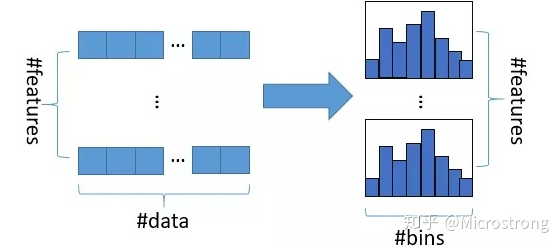
②时间上也有较大的开销，在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。

③对cache优化不友好。在预排序后，特征对梯度的访问是一种随机访问，并且不同的特征访问的顺序不一样，无法对cache进行优化。

Lightgbm算法：

①LightGBM 采用了直方图算法将遍历样本转变为遍历直方图，极大的降低了时间复杂度；（将连续特征的取值转换为分桶的离散特征）

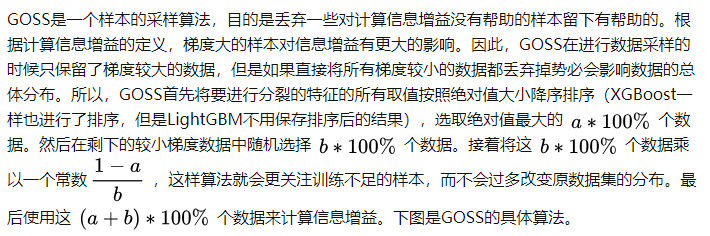
在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点。



（由上分析可知，可以做直方图做差加速）

LightGBM另一个优化是Histogram（直方图）做差加速。一个叶子的直方图可以由它的父亲节点的直方图与它兄弟的直方图做差得到，在速度上可以提升一倍。

②LightGBM 在训练过程中采用单边梯度算法过滤掉梯度小的样本，减少了大量的计算；



③LightGBM 采用了基于 Leaf-wise 算法的增长策略构建树，减少了很多不必要的计算量；

LightGBM采用Leaf-wise的增长策略，该策略每次从当前所有叶子中，找到分裂增益最大的一个叶子，然后分裂，如此循环。因此同Level-wise相比，Leaf-wise的优点是：在分裂次数相同的情况下，Leaf-wise可以降低更多的误差，得到更好的精度；Leaf-wise的缺点是：可能会长出比较深的决策树，产生过拟合。因此LightGBM会在Leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合

XGBoost使用预排序后需要记录特征值及其对应样本的统计值的索引，而 LightGBM 使用了直方图算法将特征值转变为 bin 值，且不需要记录特征到样本的索引，将空间复杂度从  ，极大的减少了内存消耗；

④LightGBM 在训练过程中采用互斥特征捆绑算法减少了特征数量，降低了内存消耗。

<https://blog.csdn.net/abcdefg90876/article/details/106935080>



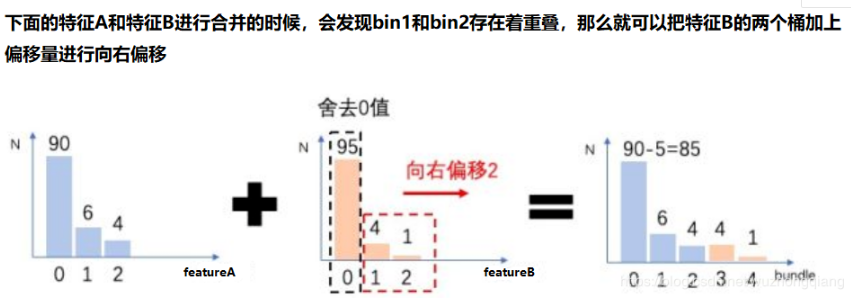
其实说白了，捆绑特征就是在干这样的一件事：

**利用图着色问题的求解来选择哪些特征进行捆绑**

**首先将所有的特征看成图的各个顶点，将不相互独立的特征用一条边连起来，边的权重就是两个相连接的特征的总冲突值（也就是这两个特征上不同时为0的样本个数）。**

**然后按照节点的度对特征降序排序，度越大，说明与其他特征的冲突越大**

**对于每一个特征，遍历已有的特征簇，如果发现该特征加入到特征簇中的矛盾数不超过某一个阈值，则将该特征加入到该簇中。如果该特征不能加入任何一个已有的特征簇，则新建一个簇，将该特征加入到新建的簇中。**



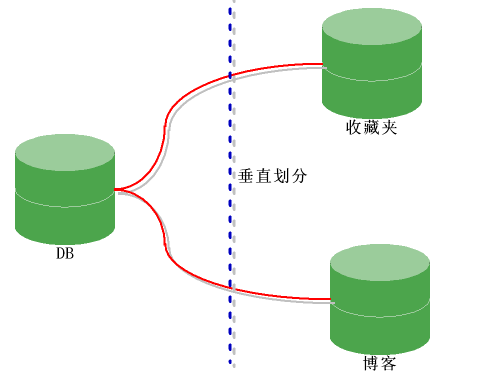
⑤lightgbm可以处理类别型特征

详细见6.1

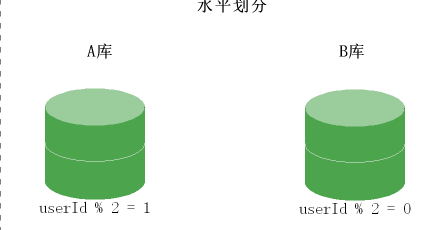
补充知识：

1. 垂直划分与水平划分

①垂直划分：（xgboost）这种特征并行方法有个很大的缺点：就是对数据进行垂直划分，每台机器所含数据不同，然后使用不同机器找到不同特征的最优分裂点，划分结果需要通过通信告知每台机器，增加了额外的复杂度



②水平划分：（lightgbm）LightGBM 则不进行数据垂直划分，而是在每台机器上保存全部训练数据，在得到最佳划分方案后可在本地执行划分而减少了不必要的通信。



6.3人工特征工程和lightgbm模型特征处理的区别；

<https://zhuanlan.zhihu.com/p/26444240>

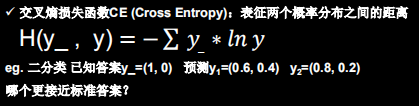
LightGBM和XGBoost都能将NaN作为数据的一部分进行学习，所以不需要处理缺失值。

在决策树系列的算法中（单棵决策树、gbdt、随机森林），每一个样本都会被映射到决策树的一片叶子上。因此，我们可以把样本经过每一棵决策树映射后的index（自然数）或one-hot-vector（哑编码得到的稀疏矢量）作为一项新的特征。

Lightgbm等决策树算法本身就可以进行特征处理。

6.4 lightgbm处理分类问题的目标函数

①Lightgbm中二分类问题的损失函数为“binary\_logloss”（即对数损失函数、或叫交叉熵损失函数）



Ps：注意回顾常用的损失函数；二分类还常用AUC

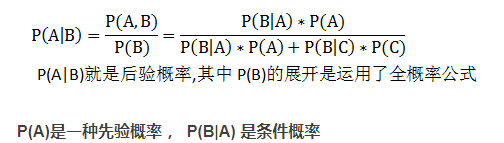
（7）什么是似然估计，什么是先验概率/后验概率，举例说明

<https://segmentfault.com/a/1190000020547595?utm_source=tag-newest>

极大似然估计：就是利用已知的样本结果信息，反推最具有可能（最大概率）导致这些样本结果出现的模型参数值。

先验概率： 事件发生前的预判概率。可以是基于历史数据的统计，可以由背景常识得出，也可以是人的主观观点给出。一般都是单独事件概率，如P(x),P(y)。

后验概率： 事件发生后求的反向条件概率；或者说，基于先验概率求得的反向条件概率。概率形式与条件概率相同。



（8）随机森林合集

8.1 RF与GBDT的区别

<https://blog.csdn.net/qq_14997473/article/details/88877198>

葫芦书P289

①随机森林采用的bagging思想，而GBDT采用的boosting思想。这两种方法都是Bootstrap思想的应用，Bootstrap是一种有放回的抽样方法思想。虽然都是有放回的抽样，但二者的区别在于：Bagging采用有放回的均匀取样，而Boosting根据错误率来取样（Boosting初始化时对每一个训练样例赋相等的权重1／n，然后用该算法对训练集训练t轮，每次训练后，对训练失败的样例赋以较大的权重），因此Boosting的分类精度要优于Bagging。Bagging的训练集的选择是随机的，各训练集之间相互独立，弱分类器可并行，而Boosting的训练集的选择与前一轮的学习结果有关，是串行的。

②组成随机森林的树可以是分类树，也可以是回归树；而GBDT只能由回归树组成。

③**组成随机森林的树可以并行生成；而GBDT只能是串行生成。**

④对于最终的输出结果而言，随机森林采用多数投票等；而GBDT则是将所有结果累加起来，或者加权累加起来。

⑤随机森林对异常值不敏感；GBDT对异常值非常敏感。

⑥随机森林对训练集一视同仁；GBDT是基于权值的弱分类器的集成。

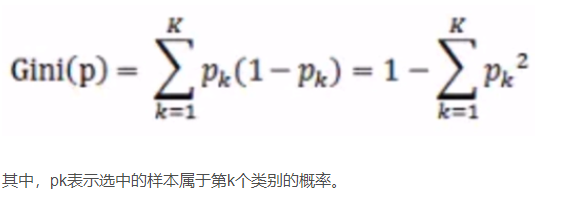
⑦随机森林是通过减少模型方差提高性能；GBDT是通过减少模型偏差提高性能

8.2 Gini的物理含义

<https://blog.csdn.net/weixin_46072771/article/details/106468993>

**基尼指数表示在样本集合中一个随机选中的样本被分错的概率（即纯度，全为0或者全为1则纯度高）**。

CART树是二叉树，对于一个有多个取值（超过2个）的特征，需要计算以每个取值作为划分点，对样本D划分之后子集的纯度Gini(D, Ai)，然后从所有的可能划分的Gini(D, Ai)中找出Gini指数最小的划分，这个划分的划分点，便是使用特征A对样本集合D进行划分的最佳划分点。



8.3 boosting和bagging的区别，重点是方差和偏差（整体和单棵树两个角度）；

葫芦书P289

Bagging能够提高弱分类器性能的原因是降低了方差，Boosting能够提升弱分类器性能的原因是降低了偏差。

Bagging 是 Bootstrap Aggregating 的简称，意思就是再抽样，然后在每 个样本上训练出来的模型取平均。随机森林对n个独立不相关的模型的预测结果取平均， 方差是原来单个模型的1/n。

再看Boosting，大家应该还记得Boosting的训练过程。在训练好一个弱分类器后，我们需要计算弱分类器的错误或者残差，作为下一个分类器的输入。这个过 程本身就是在不断减小损失函数，来使模型不断逼近“靶心”，使得模型偏差不断 降低。

8.4 随机森林的随机怎么理解

1、数据集的随机选取

从原始的数据集中采取有放回的抽样（bagging），构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。

2、待选特征的随机选取

与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。

有了这2个随机的保证，随机森林就**不会产生过拟合**的现象了

随机森林的构建过程：

①从原始训练集中使用Bootstraping方法随机有放回采样取出m个样本，共进行n\_tree次采样。生成n\_tree个训练集。

②对n\_tree个训练集，我们分别训练n\_tree个决策树模型。

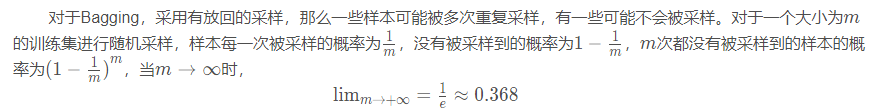
③对于单个决策树模型，假设训练样本特征的个数为n，那么每次分裂时根据信息增益/信息增益比/基尼指数 选择最好的特征进行分裂

④每棵树都已知这样分裂下去，知道该节点的所有训练样例都属于同一类。在决策树的分裂过程中不需要剪枝

⑤将生成的多颗决策树组成随机森林。对于分类问题，按照多棵树分类器投票决定最终分类结果；对于回归问题，由多颗树预测值的均值决定最终预测结果。

8.5 随机森林一个数据被重复采样的概率是多少？

独立重复试验的问题。



（9）决策树合集

9.1 树模型对连续型、离散型特征有一套统一的理论，是什么？

独热码、lightgbm、做一个embedding降维

9.2 决策树是如何分裂节点的，分裂的增益怎么算的；（ID3，C4.5，CART，GBDT，xgboost，lightgbm）；

<https://blog.csdn.net/choven_meng/article/details/82878018>

结合统计学习方法

①ID3：



其中

②C4.5：



③CART（GINI）:



其中

9.3 CART了解吗？怎么做回归和分类的？

回归：

①借助损失函数（残差平方和）最小为标准，遍历求最优切分变量和最优切分点。

②用最优变量和切分点确定各自区域的平均输出值cm

③遍历以上两个步骤直到满足停止条件。

分类：

①遍历特征和取值，分别求出基尼指数，选择基尼指数最小的特征和切分点。

②将数据集分为两半后，重复使用①，直到满足分类条件。

9.4 回归树的目标函数什么样；



9.5 决策树如何剪枝的

①计算各个结点的损失函数，将损失函数加上正则化（决策树的C（T）是gini指数）：



其中，称为经验熵

②递归地从树的叶结点向上回缩，如果递归到父节点之后的整体树大于父节点之前的整体树（即分裂前和分裂后的损失函数对比，若分类前的小则剪枝）

③重复以上步骤，直到损失函数最小为止

9.6 决策树处理缺失值

借CART树的缺失值处理方法回答。

9.7 为什么树模型不需要对特征进行one-hot编码、归一化、和woe编码等预处理呢？（这是站在分类树角度看的，前面的gbdt和xgboost需要独热编码是站在回归树的角度看待问题的，独热码能够将离散特征划分成连续特征）

1、树模型是要寻找最佳分裂点，对于离散特征，树模型会评估每个离散值的信息增益，将信息增益最大的数值作为分裂点，因此，树模型不需要对离散特征进行事先one-hot处理，否则会使特征维度增大且稀疏，不仅会增加模型的计算量，而且会损失数据的信息量造成模型的效果不佳，以及过拟合的风险。

2、**对于连续型特征，树模型对尝试对连续特征分桶**，将信息增益最大的桶边界值最为分裂点。因此也不需要事先对连续特征进行分桶和woe编码。

3、另外，树模型是找最佳分裂点，是否对数据进行归一化，不影响最佳分裂点的计算，因此也无需进行数据归一化处理。

1. 逻辑回归模型

10.1 为什么LR适用于离散型特征而GBDT适用于连续型特征

LR一般需要连续特征离散化原因：

①离散特征的增加和减少都很容易，易于模型快速迭代；

②稀疏向量内积乘法速廈快，计算结果方便存储，容易扩展；

③离散化的特征对异常数据有很强的鲁棒性(比如年龄为300异常值可归为年龄>30这一段)；

④逻辑回归属于广义线性模型，表达能力受限。单变量离散化为N个后，每个变量有单独的权重，相当于对模型引入了非线性，能够提升模型表达能力，加大拟合；

⑤离散化进行特征交叉，由 M+N 个变量为M\*N个变量(将单个特征分成M个取值)，进一步引入非线性，提升表达能力

⑥特征离散化后，模型会更稳定(比如对用户年龄离散化，20-30作为一个区间，不会因为用户年龄，增加一岁变成完全不同的人。

10.2 SVM和logistics regression的区别

联系：

①都是监督的分类算法。

②都是线性分类方法 (不考虑核函数时）。

③都是判别模型。

区别：

① 损失函数的不同，LR是对数损失函数，SVM是hinge损失函数。

② SVM不能产生概率，LR可以产生概率。

③ SVM自带结构风险最小化，LR则是经验风险最小化。（即LR没有自带正则化）

④ SVM会用核函数而LR一般不用核函数。

⑤ LR和SVM在实际应用的区别：根据经验来看，对于小规模数据集，SVM的效果要好于LR，但是大数据中，SVM的计算复杂度受到限制，而LR因为训练简单，可以在线训练，所以经常会被大量采用。

10.3 gbdt与lr的异同

相同点：本质上都是监督学习，判别模型，直接对数据的分布建模，不尝试挖掘隐含变量，这些方面是大体相同的。

不同点：

①一个是线性模型，一个是非线性模型。lr假设数据是线性可分的。

②策略 ，从loss（经验风险最小化）+ 正则（结构风险最小化）的框架：应为lr，的输出是y=1的概率，所以在极大似然下，lr的loss 是 交叉熵（极大似然估计求得），此时lr的准测是最大熵原理，也就是为了追求最小分类误差，追求最大熵loss，本质熵分类器算法，而且对数据的噪声具有高斯假设。而GBDT采用的cart回归树 作为基本分类器，其无论是处理分类 还是回归，都是采用回归拟合（将分类问题通过softmax转换为回归问题），用当前cart树拟合前一轮目标函数与实际的负梯度

③：从特征空间角度：

就是应为lr是特征的线性组合，求交叉熵的最小化，也就是对特征的线性组合做logistic，使用与分类。

而GBDT采用cart树作为基分类其，其每一轮的特征拟合都是对特征空间做平行于坐标轴的空间分割，所以自带特征选择和可解释性。GBDT既可以做分类也可以做回归，只是统一采用回归的思路进行求解。

④从正则的角度：lr采用 l1 l2正则；GBDT采用 弱分类器的个数，也就是迭代轮次T，T的大小影响着算法的复杂度

⑤ 特征组合：GBDT特征的选择方法是采用最下化均方误差，来寻找分裂特征和对用的分裂点，所以会自动在当前根据特征A分裂的自述下寻找其他 能使负梯度最小的 其他特征B ，这样就自动剧本寻求好的特征组合的性能，因此也能给出那些特征比较重要（根据该特征被作为分裂特征的次数）

lr：只是一次性的寻求最大化熵的过载，对每一维的特征都假设独立，因此，只具备对已有特征空间进行分割的能力，更不会对特征空间进行升维（特征组合）

lr不具有特征组合呢能力，并假设各个维度的特征独立，因此只具有线性分界面，实际应用中，多数特征之间又相关性，只有维度特别大的稀疏数据中特征才会近似独立，所以适用于 特征稀疏的数据熵。

而GBDT更适合处理稠密特征，例如GBDT+lr 和facebook论文中，对于连续型的特征导入GBDT做特征组合来代替一部分手工特征，而对于id类特征的做法往往是one-hot之后。直接转入lr，或者先hash，在hot-one，现在主流的做法是直接one-hot+ embeding来讲高维稀疏特征压缩为低维稠密特征，也进一步引入语义信息，有利于特征表达。

10.4 LR的损失函数，LR的损失函数的意义

LR回归的损失函数是极大似然函数的对数似然函数。（推导过程见李航）



其中；

由于

故

10.5 逻辑回归分类非线性的样本分布

对数据进行升维，变成更高一维的线性可分空间。

10.6 为什么逻辑回归是线性模型

判断一个模型是否是线性，是通过分界面是否是线性来判断的。故SVM和逻辑回归是线性模型。

1. stacking，blending

<https://blog.csdn.net/u010412858/article/details/80785429>

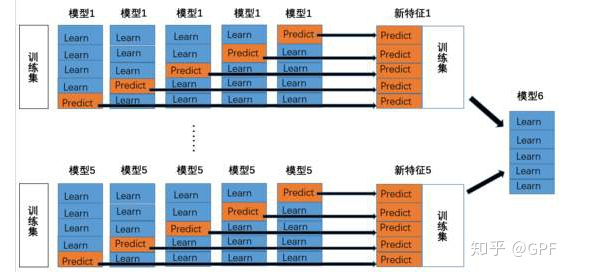
<https://zhuanlan.zhihu.com/p/42229791>

<https://zhuanlan.zhihu.com/p/91836593>

stacking是一种分层模型集成框架(**也是集成学习的一种**)。以两层为例，第一层由多个基学习器组成，其输入为原始训练集，用K折交叉验证得到第二层的输入。第二层的模型则是以第一层基学习器的输出作为特征加入训练集进行再训练，从而得到完整的stacking模型。

stacking模型调参包括对基模型和元模型进行调参。对于基模型，因为我们在生成元特征的时候要使用相同的K折划分，所以我们使用交叉验证+网格搜索来调参时最好使用与生成元特征相同的Kfold。对于元模型的调参，使用交叉验证+网格搜索来调参时，**为了降低过拟合的风险**，我们最好也使用与元特征生成时**同样的Kfold**

**其实就是五折交叉验证，四折用于第一层训练，一折用于第一层的预测；循环五次可以得到，原始数据集在第一层模型的预测值，将该训练集和第一层得到的预测值输入第二个模型，输出最终预测。**



Blending：Blending与Stacking大致相同，只是Blending的主要区别在于训练集不是通过K-Fold的CV策略来获得预测值从而生成第二阶段模型的特征，而是建立一个Holdout集。简单来说，Blending直接用不相交的数据集用于不同层的训练。

**bending是一种模型融合方法，对于一般的blending，主要思路是把原始的训练集先分成两部分，比如70%的数据作为新的训练集，剩下30%的数据作为测试集。第一层我们在这70%的数据上训练多个模型，然后去预测那30%数据的label。在第二层里，我们就直接用这30%数据在第一层预测的结果做为新特征继续训练即可。**

以两层的Blending为例，训练集划分为两部分（d1，d2），测试集为test。

第一层：用d1训练多个模型，将其对d2和test的预测结果作为第二层的New Features。

第二层：用d2的New Features和标签训练新的分类器，然后把test的New Features输入作为最终的测试集，对test预测出的结果就是最终的模型融合的值。

Blending与stacking相比优点在于：

1.比stacking简单（因为不用进行k次的交叉验证来获得新特征）

2.由于两层使用的数据不同，所以避免了一个信息泄露的问题。

3.在团队建模过程中，不需要给队友分享自己的随机种子。

而缺点在于：

1.由于blending对数据集这种划分形式，第二层的数据量比较少。

2.由于第二层数据量比较少所以可能会过拟合。

3.stacking使用多次的CV会比较稳健

（12）常见损失函数及其对应模型

<https://www.cnblogs.com/guoyaohua/p/9217206.html>

（13）评价指标合集

13.1 AUC

（准确度， 精确度，混淆矩阵（即TP、FP、FN、TN四个组成的），AUC， F1-score）

<https://www.zhihu.com/question/39840928>

FPR表示，在所有的实际为恶性肿瘤中，被预测成良性的比例（即假正率）。(**注意TP和FP加了个率就是不同含义。FP率和TP率的分母都为实际为负和实际为正**）（所以全把预测为正，100个病人，99个正常，1个癌症，假正率为0，还是没有面积）

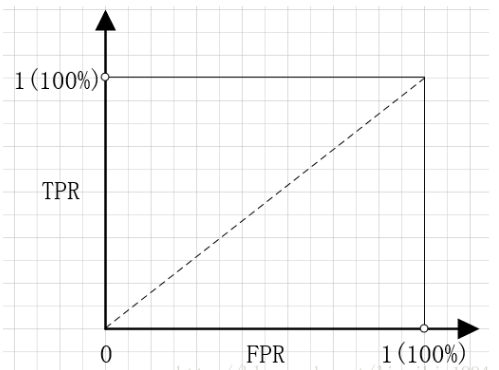
TPR表示，在所有良性肿瘤中，被预测为良性的比例，称为真阳性率。



FPR：在所有实际为阴性的样本中，被错误地判断为阳性之比率



下图直线表示的意义是：对于不论真实类别是1还是0的样本，分类器预测为1的概率是相等的。因为FPRate = TPRate。预测标签为正的概率为0.5，预测标签为负的也是0.5，结合TPR为实际为正又预测为正。



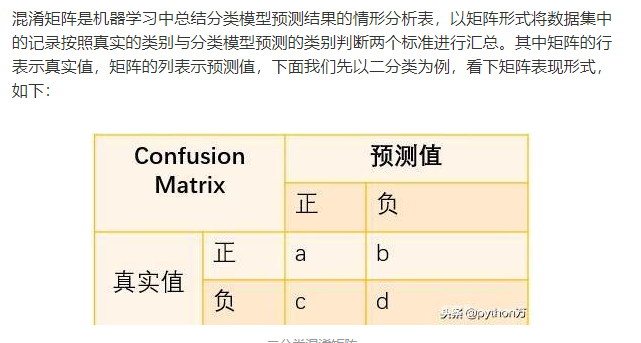
点(0,1)，即FPR=0，TPR=1。FPR=0说明FP=0，也就是说，没有假正例。TPR=1说明，FN=0，也就是说没有假反例。这不就是最完美的情况吗？所有的预测都正确了。良性的肿瘤都预测为良性，恶性肿瘤都预测为恶性，分类百分之百正确。这也体现了FPR 与TPR的意义。就像前面说的我们本来就希望FPR越小越好，TPR越大越好。

点(1,0)，即FPR=1，TPR=0。这个点与上面那个点形成对比，刚好相反。所以这是最糟糕的情况。所有的预测都预测错了。

点(0,0)，即FPR=0，TPR=0。也就是FP=0，TP=0。所以这个点的意义是所有的样本都预测为恶性肿瘤。也就是说，无论给什么样本给我，我都无脑预测成恶性肿瘤就是了。

点(1,1)，即FPR=1，TPR=1。显然，这个点跟点(0,0)是相反的，这个点的意义是将所有的样本都预测为良性肿瘤。

在二分类（0，1）的模型中，一般我们最后的输出是一个概率值，表示结果是1的概率。那么我们最后怎么决定输入的x是属于0或1呢？我们需要一个阈值，超过这个阈值则归类为1，低于这个阈值就归类为0。所以，不同的阈值会导致分类的结果不同，也就是混淆矩阵不一样了，FPR和TPR也就不一样了。所以当阈值从0开始慢慢移动到1的过程，就会形成很多对(FPR, TPR)的值，将它们画在坐标系上，就是所谓的ROC曲线了。



我们来举一个例子。比如我们有5个样本：

真实的类别(label)为y = c(1,1,0,0,1).

一个分类器预测样本为1的概率为p=c(0.5, 0.6, 0.55, 0.4, 0.7).

正如上面说的，我们需要有阈值，才能将概率转换为类别，才能得到FPR和TPR。而选定不同的阈值会得到不同的FPR和TPR。假设我们现在选定的阈值为0.1,那么5个样本都被归类为1。如果选定0.3，结果仍然一样。如果选了0.45作为阈值，那么只有样本4被分进0，其余都进入1类。当我们不断改变阈值，就会得到不同的FPR和TPR。然后我们将得到的(FPR , TPR)连接起来，就得到了ROC曲线了。

**ROC曲线上的每一个点对应于一个threshold，对于一个分类器，每个threshold下会有一个TPR和FPR。**

**比如Threshold最大时，TP=FP=0，对应于原点；Threshold最小时，TN=FN=1，对应于右上角的点(1,1)。**

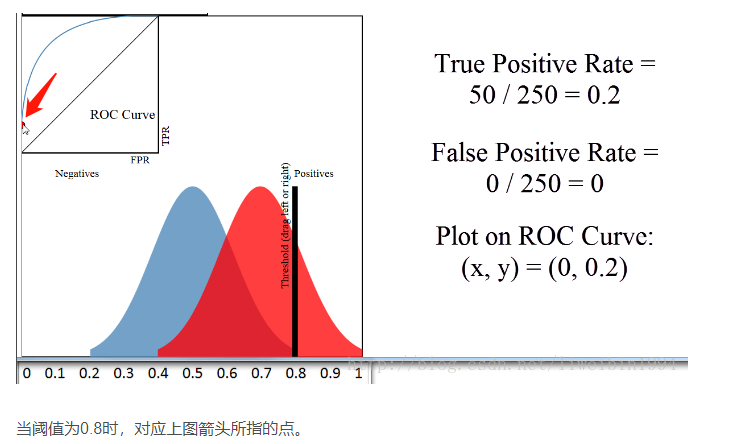
**随着阈值theta增加，TP和FP都减小，TPR和FPR也减小，ROC点向左下移动**。

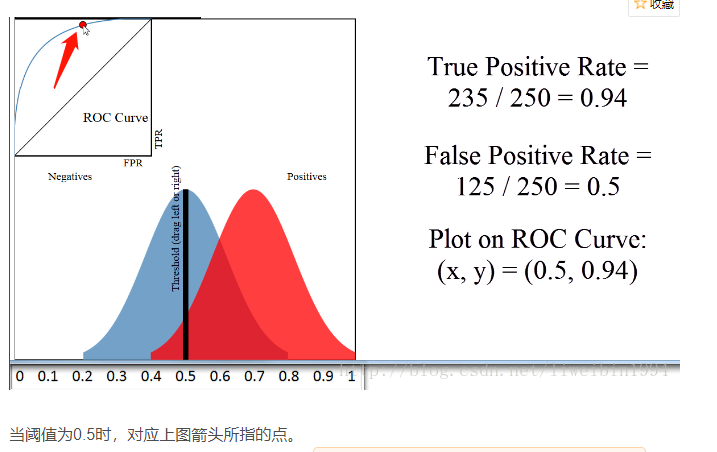
如果在给定的样本中，我都随机预测，也就是0.5概率预测为良性肿瘤，0.5概率预测为恶性肿瘤。那么这条曲线会是怎样的呢？可以想象，如果数据是均匀，那么这条曲线就是y=x。

注意曲线一定是从(0,0)开始最终到达(1,1)的。理解了上面四个点的意义就知道了。

事实上，ROC曲线不是光滑的，而是阶梯型的。为什么呢？因为样本的数量是有限的，而FPR和TPR的变化需要至少有一个样本变化了，在没有变化的间隙里，就不会有变化。也就是说，步进是1/样本数。

每一个阈值对应ROC曲线上的一点，连接所有阈值对应的点得到ROC曲线。故**所以全部预测为正的话，FPR不管什么阈值始终为1，面积为0，此类情况横坐标上无积累。**





**如果我们想要用ROC来评估分类器的分类质量，我们就可以通过计算AUC（ROC曲线下的面积）来评估了，这就是AUC目的。**

**其实，AUC表示的是正例排在负例前面的概率。**

由如下例子可知，选了一个正一个负组成样本对，正在负前面的意思则是预测结果也为先正后负，与实际挑选的样本对预测一致，代表预测正确。即代表预测正确的概率值。

**AUC的含义：从所有1（0）样本中随机选择一个样本，放入分类器进行预测，预测1-->1的概率为p1，预测0-->1的概率为p0. p1>p0的概率就是AUC。当AUC为0.5时，p1=p0=0.5所以两者概率相等。**

<https://www.zhihu.com/question/39840928>

13.2 Recall

TP、True Positive 真阳性：预测为正，实际也为正

FP、False Positive 假阳性：预测为正，实际为负

FN、False Negative 假阴性：预测与负、实际为正

TN、True Negative 真阴性：预测为负、实际也为负

即召回率是预测为真且实际为真/实际为真。

Ps：某池塘有1400条鲤鱼，300只虾，300只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了700条鲤鱼，200只虾，100只鳖。那么，这些指标分别如下：

精确率 = 700 / (700 +200 + 100) = 70%

召回率 = 700 / 1400 =50%

F1分数（F1-Score），又称为平衡F分数（BalancedScore），它被定义为精确率和召回率的调和平均数。



13.3 回归问题有啥评价指标

<https://blog.csdn.net/weixin_41108334/article/details/84502204>

<https://www.cnblogs.com/wzdLY/p/9771460.html>

①平均绝对误差，也叫L1范数损失，公式：MAE = 1/N·Σ|Yi-Pi|,其中，N为样本数，Yi为第i条样本的真实值，Pi为第i条样本的预测值。**模型使用MAE作为损失函数是对数据分布的中值进行拟合（13.6有推导理解）**。但某些模型如XGBoost必须要求损失函数有二阶导数，所以不能直接优化MAE。

②均方根误差的公式：RMSE =√( 1/N·Σ|Yi-Pi|2)，RMSE代表的是预测值与真实值差值的样本标准差。和MAE对比，RMSE对大误差样本有更大的惩罚，但它对离群点敏感，健壮性不如MAE。模型使用RMSE作为损失函数是对数据分布的平均值进行拟合。

③MSE：平方误差

13.4 分类问题的评价指标

①准确率Accruacy

②精确率Precision和召回率Recall

精确率表示预测结果中，预测为正样本的样本中，正确预测为正样本的概率；

召回率表示在原始样本的正样本中，最后被正确预测为正样本的概率；

③F-1 Score



④ROC与AUC （只能用于二分类）

13.5 比较AUC和F1-score

（说AUC是真阳性，与假阳性之间的平衡，面积越大越好，然后F1-score,是平衡p和R）

<https://blog.csdn.net/Jerry_Lu_ruc/article/details/107912462>

①关于如何选择这两个指标，网上很多人的意见是：auc不容易受样本不平的影响，所以对于imbalance的情况优先使用auc。对于负样本少的场景，F1分数依然会高，但是AUC不会很高。

②两者除了召回率外，AUC的另一个目标是：降低非真样本呈阳性的比例(假阳性)希望训练一个尽量不误报的模型，比较保守。而F1score的另一个优化目标是希望提高检验呈阳性的样本中实际为真的比例，也就是提高检验的准确率/命中率

**如果我们犯检验误报错误的成本很高，那么我们选择auc是更合适的指标。如果我们犯有漏网之鱼错误的成本很高，那么我们倾向于选择f1score。**

**放到实际中，对于检测传染病，相比于放过一个可能的感染者，我们愿意多隔离几个疑似病人，所以优选选择F1-score作为评价指标。而对于推荐这种场景，由于现在公司的视频或者新闻库的物料总量是很大的，潜在的用户感兴趣的item有很多，所以我们更担心的是给用户推荐了他不喜欢的视频，导致用户体验下降，而不是担心漏掉用户可能感兴趣的视频。所以推荐场景下选择auc是更合适的。**

13.6 为什么MSE损失的收敛目标是均值，而MAE的是中位数

MSE

IMG_256

这是MSE损失函数的计算公式。m是样本量，score和y分别是模型预估值和样本真实label。我们用t表示模型预估值score1, score2, score3...的均值，然后用t替换上式中的score\_i, 看下在该损失函数下模型收敛值取多少时损失最小:

IMG_257

求导:

IMG_258

IMG_259

也就是说，在损失函数为 MSE 时，模型收敛值为样本label的均值。如果样本x在训练数据中出现了3次，3次对应的label由于各种现实原因可能并不相同，如分别是2, 4, 9， 那么在mse下训练的模型预估值应该在(2+4+9)/3=5左右。

****MAE****

IMG_256

这是MAE损失函数的计算公式, 符号意义同MSE。同样用t表示模型预估值score1, score2, score3...的均值:

IMG_257

求导分两种情况，考虑求和的子项 |t-y\_i|，当 t > y\_i 时t梯度为1，相反则为-1，那么有:

IMG_258

**其中sgn(t-y)表示t-y的正负符号。显然，在t > y\_i和t < y\_i一样多，也就是t取中位数时损失取得极小值**。如果样本x在样本中的label取值为2, 4, 9，那么在 MAE 下训练的模型预估值应该是中位数4。

****区别与取舍****

从以上推导可以看出, MSE、MAE损失函数的收敛目标分别对应着label的均值和中位数。**均值受异常值影响较大，而中位数则相对稳定**。因此从这一角度来看，如果训练数据中存在较多异常点，MSE 受影响较大，毕竟其损失函数中平方项会给异常值较大比重。**不过尽管MAE较稳定，但其缺点是梯度大小始终不变(1或者-1)，从以上推导也能看出这点。而实际中，在预估与label已经很接近的时候，梯度应该更小才合理**。当然也可以梯度不够学习率来凑，MAE 可以手动调整学习率来缓解这一问题。具体二者的选择也不能一概而论，还是需要结合具体使用场景、数据分布情况等来确定。

**某些模型如XGBoost必须要求损失函数有二阶导数，所以不能直接优化MAE**

（14）怎么衡量两个分布的差异？KL散度和交叉熵损失有什么不同？关系是啥？

14.1必备知识

<https://blog.csdn.net/am290333566/article/details/81187124>

信息熵：信息是个很抽象的概念。人们常常说信息很多，或者信息较少，但却很难说清楚信息到底有多少。比如一本五十万字的中文书到底有多少信息量。

直到1948年，香农提出了“信息熵”的概念，才解决了对信息的量化度量问题。香农用信息熵的概念来描述信源的不确定度。

信息熵的定义公式：

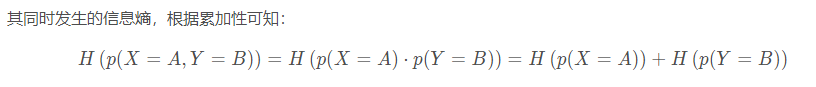


信息熵的三个性质：

单调性，发生概率越高的事件，其携带的信息量越低；举一个极端的例子，“太阳从西边升起”所携带的信息量就远大于“太阳从东边升起”，因为后者是一个万年不变的事实，不用特意述说大家都知道；而前者是一个相当不可能发生的事情，如果发生了，那代表了太多的可能性，可能太阳系有重大变故，可能物理法则发生了变化，等等

非负性，信息熵可以看作为一种广度量，非负性是一种合理的必然；

累加性，即多随机事件同时发生存在的总不确定性的量度是可以表示为各事件不确定性的量度的和，这也是广度量的一种体现。对数可以由乘转加。



14.2 KL散度（Kullback–Leibler divergence）

<https://blog.csdn.net/leviopku/article/details/81388306>

<https://blog.csdn.net/weixinhum/article/details/85064685>

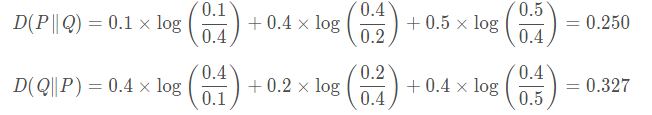
相对熵，又被称为KL散度或信息散度，是两个概率分布间差异的非对称性度量 。在信息论中，相对熵等价于两个概率分布的信息熵的差值，若其中一个概率分布为真实分布，另一个为理论（拟合）分布，则此时相对熵等于交叉熵与真实分布的信息熵之差，表示使用理论分布拟合真实分布时产生的信息损耗。



为真实事件的概率分布，为理论拟合出来的该事件的概率分布

也就是在理论拟合出来的事件概率分布跟真实的一模一样的时候，相对熵等于0。而拟合出来不太一样的时候，相对熵大于0。这个性质很关键，因为它正是深度学习梯度下降法需要的特性。假设神经网络拟合完美了，那么它就不再梯度下降，而不完美则因为它大于0而继续下降。

但它有不好的地方，就是它是不对称的。举个例子，比如随机变量取值为1,2,3时的概率分别为[0.1,0.4,0.5]，随机变量取值为1,2,3时的概率分别为[0.4,0.2,0.4]:

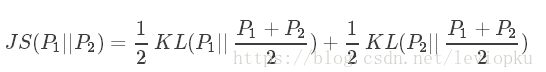


也就是用 P来拟合Q和用Q来拟合P的相对熵居然不一样，而他们的距离是一样的。这也就是说，相对熵的大小并不跟距离有一一对应的关系。这点蛮头疼的，因为一般我们希望距离越远下降越快，而相对熵取哪个为参考在同等距离情况下下降的速度都不一样，这就非常尴尬了。



可以看到它的两头异常的平，也就是说在那些地方的导数接近于0。而反向传播是需要求导的，用了均方差损失函数之后求导结果包含 y ( y − 1 ) y(y-1) y(y−1)（可参考这篇文章），这在 y 接近于0或者1的时候都趋于0，会导致梯度消失，网络训练不下去。但如果用相对熵衍生出来的交叉熵作为损失函数则没有这个问题。

14.3 JS散度（Jensen-Shannon divergence）

JS 散度度量了两个概率分布的相似度，基于KL散度的变体，解决了KL散度非对称的问题。一般地，JS散度是对称的，其取值是0到1之间。

KL 散度和JS散度度量的时候有一个问题：如果两个分配P,Q离得很远，完全没有重叠的时候，那么KL散度值是没有意义的，而JS散度值是一个常数。这在学习算法中是比较致命的，这就意味这这一点的梯度为0。梯度消失了。

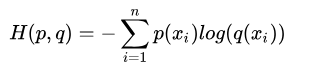
14.4交叉熵（Cross Entropy）

<https://zhuanlan.zhihu.com/p/74075915>

我们将KL散度公式进行变形



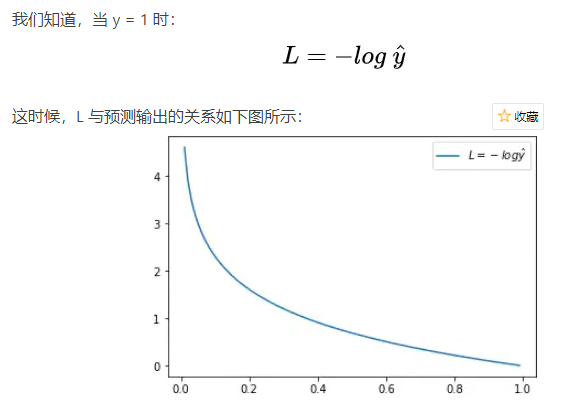
等式的前一部分恰巧就是p的熵，等式的后一部分，就是交叉熵：

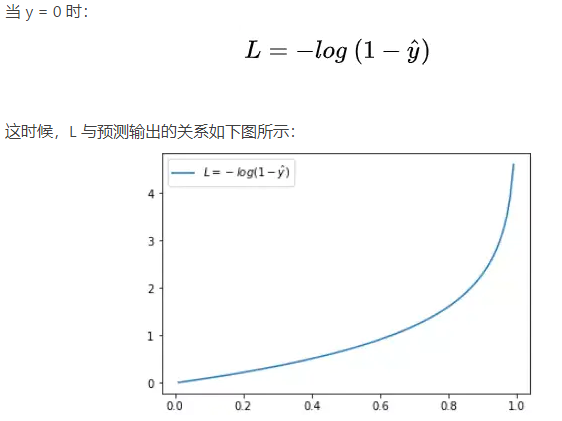


在机器学习中，我们需要评估label和predicts之间的差距，使用KL散度刚刚好，即，由于KL散度中的前一部分−H(y)不变，故在优化过程中，只需要关注交叉熵就可以了。所以一般在机器学习中直接用用交叉熵做loss，评估模型

当实际标签为y=1时，-logy在0到1内是一个递减函数，符合损失函数的定义，越与实际接近则越小。

当实际标签y=0时，

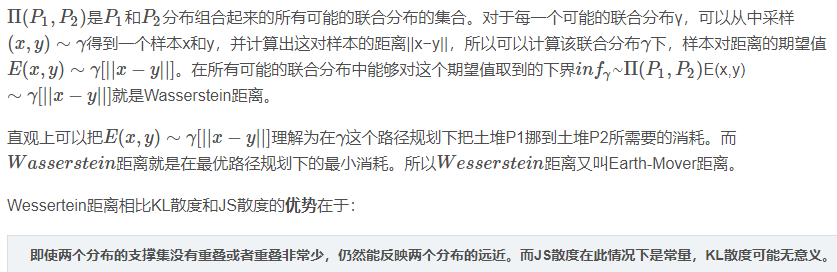




14.5 Wasserstein距离

Wasserstein 距离度量两个概率分布之间的距离，定义如下：





1. 变量间的相关关系

15.1如何判断变量之间的相关性

可以用简单相关系数来进行分析，应用场景非常广泛。皮尔逊相关系数的结果总是落在-1和1之间就可以了。

结果为正数时表示两个变量成正相关，即一个变量增大时另一个变量也增大，比如气温越高，冷饮的销量就越多，这是正相关关系；

结果为负数时两个变量呈负相关，即一个变量增大时另一个变量减小，例如海拔越高时，空气中的氧气含量就越少。

如果为0，则表示两个变量不为线性关系，有可能两者不相关，但也有可能两者有更加复杂的关系。



15.2那怎么判断非线性相关

<https://blog.csdn.net/qq_27586341/article/details/90603140>

MIC，即（Maximal Information Coefficient）最大信息系数，属于Maximal Information-based Nonparametric Exploration (MINE) 最大的基于信息的非参数性探索，用于衡量两个变量X和Y之间的关联程度，线性或非线性的强度，常用于机器学习的特征选择。

互信息（Mutual Information）是信息论里一种有用的信息度量，它可以看成是一个随机变量中包含的关于另一个随机变量的信息量，或者说是一个随机变量由于已知另一个随机变量而减少的不肯定性。

1. 分类和聚类的区别与联系
2. HMM合集（nlp会比较多问）

17.1 HMM介绍

<https://blog.csdn.net/hudashi/article/details/87867916>

统计学习方法

隐马尔可夫模型(Hidden Markov model, HMM)是一种结构最简单的动态贝叶斯网的生成模型，它也是一种著名的有向图模型。它是典型的自然语言中处理标注问题的统计机器学模型，本文将重点介绍这种经典的机器学习模型。

隐马尔科夫模型是关于时序的概率模型，描述由一个隐藏的马尔可夫链随机生成不可观测的状态随机序列，再由各个状态生成一个可观测的随机序列的过程，隐藏的马尔可夫链随机生成的状态序列，称为状态序列(也就上面例子中的D6，D8等)；每个状态生成一个观测，而由此产生的观测随机序列，称为观测序列(也就上面例子中的1，6等)。序列的每个位置又可以看作是一个时刻。

隐马尔可夫模型由初始的概率分布、状态转移概率分布以及观测概率分布确定。A、B和C也被称为隐马尔科夫模型的三要素。（具体的定义见链接中的介绍）

两个基本假设：

①马尔可夫性假设，即假设隐藏的马尔可夫链在任意时刻t的状态只依赖于其前一时刻的状态，与其它时刻的状态及观测无关，也与时刻t无关。



②观测独立性假设，即假设任意时刻的观测只依赖于该时刻的马尔可夫链的状态，与其他观测及状态无关。



隐马尔科夫模型的三个基本问题：

概率计算问题、学习问题、预测问题；（解决方法和过程见统计学习方法）

17.2和CRF的区别（判别模型和生成模型的区别）、并且举例

1.HMM是生成模型，CRF是判别模型

2.HMM是概率有向图，CRF是概率无向图

3.HMM求解过程可能是局部最优，CRF可以全局最优

4.CRF概率归一化较合理，HMM则会导致label bias 问题

并且要举例说明：

HMM用在了分词，CRF用在了词性标注

17.3 最成功的应用

1. CRF合集

18.1 CRF的损失函数

18.2 条件随机场以及其势函数的定义

（19）什么时候要归一化（normalization）

不同评价指标（即特征向量中的不同特征就是所述的不同评价指标）往往具有不同的量纲和量纲单位，这样的情况会影响到数据分析的结果，为了消除指标之间的量纲影响，需要进行数据标准化处理，以解决数据指标之间的可比性。原始数据经过数据标准化处理后，各指标处于同一数量级，适合进行综合对比评价。

用于特征标准化的途径有两种：

①min max normalization,他会将所有特征数据按比例缩放到0-1的这个取值区间.有时也可以是-1到1的区间.

②standard deviation normalization, 他会将所有特征数据缩放成 平均值为0, 方差为1.

（20）聚类算法。

2.1聚类算法有哪些

<https://blog.csdn.net/bxg1065283526/article/details/79650042>

K-Means(K均值)聚类、CLIQUE算法----基于网格的聚类算法、基于密度的聚类方法(DBSCAN)。

2.2 KNN、K-Means的原理

①k越大，模型越简单，当k为N时，简单到极致，直接选取标签较多的。

k越小，学习的误差变小，预测结果会对邻近的点非常敏感。

②kd树的构建：李航P54

搜索：<https://blog.csdn.net/qq_36955294/article/details/106441925>。

Kmeans是无监督学习方法，随机选取k个样本中心、计算每个样本到类中心的距离，将每个样本指派到与其最近的中心的类中。计算各个类中样本的均值，作为新的聚类中心。

2.3 KNN有哪些局限性？

①当样本不平衡时，比如一个类的样本容量很大，其他类的样本容量很小，输入一个样本的时候，K个临近值中大多数都是大样本容量的那个类，这时可能就会导致分类错误。改进方法是对K临近点进行加权，也就是距离近的点的权值大，距离远的点权值小。

②效率低下:

对于每一个预测数据都需要O(mxn)的时间复杂度，可以对其利用kd树结构进行优化，不过即使优化之后其效率也是比较低下的；

③需要事先知道多少种类。（K-Means）

2.4 有缺失值怎么办

对于离散变量而言，经常有缺失值，对于缺失值有如下处理方法：

1. 如果数据集很大，缺失值很少，可以删除缺失值；

2. 如果这个属性对结果的影响很大，可以用均值或者众数来代替；（**离散值用众数、连续值用均值或者中位数**）

如果属性是连续型随机变量可以考虑用回归、随机森林等方法来预测缺失值；

（20）gbdt合集

20.1 gbdt的思想

统计学习方法

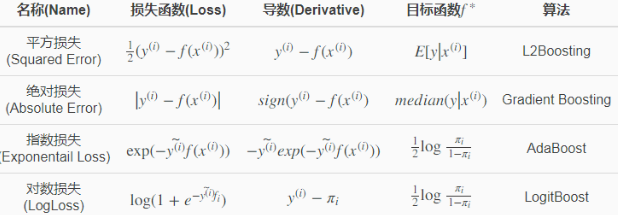
其基本思想是根据当前模型损 失函数的负梯度信息来训练新加入的弱分类器，然后将训练好的弱分类器以累加 的形式结合到现有模型中。

20.2 不同损失函数的gbdt有什么不同

<https://blog.csdn.net/qfikh/article/details/102884930>

①分类算法的损失函数：指数损失函数、对数损失函数：二元分类的对数函数、多元分类的对数函数

②回归算法的损失函数：均方损失函数、绝对值损失函数、Huber损失函数、分位数损失函数



20.3 Boosting是在对上一次训练结果的残差进行拟合，为什么这么说呢？

20.4 Adaboost/GBDT分别拟合了什么残差？

AdaBoost 只能是指数损失，其他gbdt可以用其他损失函数

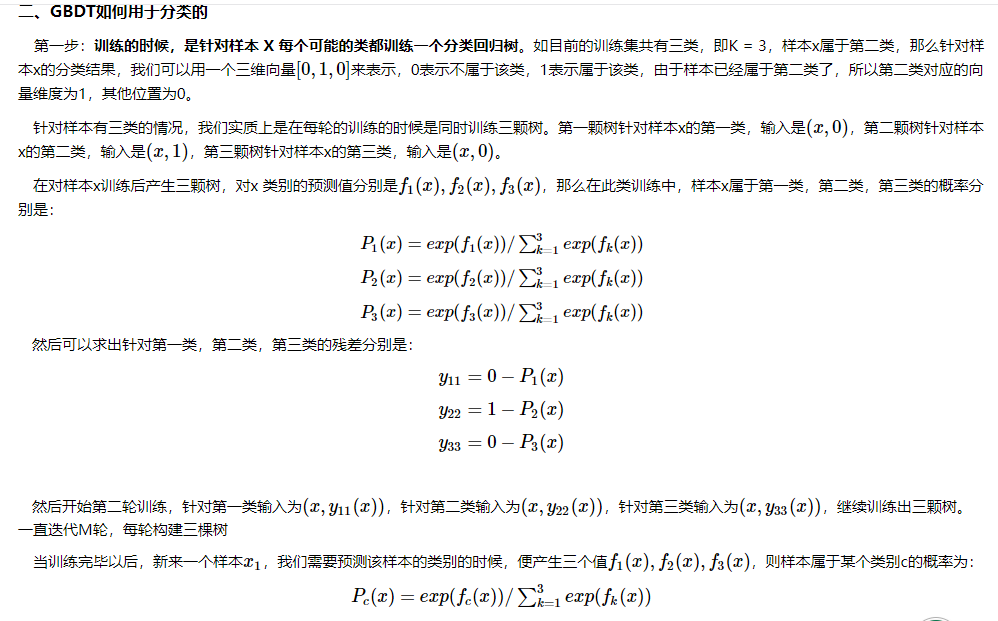
20.5你说GBDT回归的时候拟合的是具体的数值，那分类的时候怎么实现的呢

（分类的时候应该就是拟合概率值把）

20.6面试官继续问GBDT用于三分类的时候，怎么实现的

<https://www.cnblogs.com/always-fight/p/9400346.html>

每轮同时训练三棵树，不再是每轮一棵树，用[0,1,0]减去拟合的概率值作为下轮训练的残差



（21）EM算法合集

21.1 EM算法用在哪些问题上

1. SVM，LR，GBDT相关的模型对比

22.1她们更有什么优缺点和应用场景吗

（我说SVM可能因为超平面的关系更加适用于多分类，然后LR实现简单，应该效率比较高，适用于样本比较好；然后GBDT他是个集成学习，效果应该比前面两个好把，...现场瞎扯，因为从来没有放在一起比过这几个）

<https://juejin.cn/post/6844903990228942861>

①LR

LR基本可以被划分为线性模型，模型本身并不能完全解决非线性问题。**但是我们在使用LR的过程中，往往要对数据进行稀疏化，例如one-hot操作。这样操作会将特征的向量空间进行升维，使得问题变得线性可分。**我们在工业级的数据中，往往输入模型的特征维度过百万，千万甚至上亿，很多都是这种特征稀疏化造成的。

②SVM

SVM本身是一个线性模型，其非线性拟合能力与核函数有关。在SVM的核函数为线性核的时候，其性能与LR很类似，在少量数据集上比LR多了些泛化能力，这一点是支持向量造成的。当SVM采用一些非线性核，比如径向基或者高斯核的时候，其非线性拟合能力比较好。在中小规模的数据集上，SVM分类效果很强的，相比于GBDT，其能够更好的控制过拟合，这一点是其原理造成的。其也可以使用在高维稀疏数据集上，这一点是GBDT的硬伤。但是其缺点也很明显，**SVM对缺失值很敏感，需要标准化（归一化）操作，在数据规模大的时候速度效率低等缺点（速度效率与核函数和支持向量数量有关）**

③GBDT

GBDT是一个非线性模型，它是决策树出身，天生就可以拟合各种非线性问题。GBDT受数据量大小的影响相对来说较小，因为GBDT复杂度随着数据发生变化。数据复杂它可以生成复杂的树，数据简单可以生成简单的树，这里的数据简单和复杂是对其分类难度来说的。但其缺点也是比较明显，很容易过拟合，而且由于其对列进行操作，GBDT对高维稀疏向量效果就很不好。百万维的数据，GBDT会不知道从哪里下手。当然也正是由于其对列操作，所以其对缺失值不敏感，最坏情况就是那一列舍弃不要，不会造成局部噪声过大影响全局的情况。因为其切分树枝操作，只要找到合理的损失函数，GBDT对离散数据和连续数据都很好使用，这里的离散数据指的是稀疏化之前，比如班级序号这种数据，不像LR这种线性模型只对高维稀疏线性可分数据效果明显。

22.2 有些模型为什么只适用于正态分布

数据整体服从正态分布，那样本均值和方差则相互独立。正态分布具有很多好的性质，很多模型假设数据服从正态分布。例如线性回归(linear regression)，它假设误差服从正态分布，从而每个样本点出现的概率就可以表示成正态分布的形式，将多个样本点连乘再取对数，就是所有训练集样本出现的条件概率，最大化这个条件概率就是LR要最终求解的问题。这里这个条件概率的最终表达式的形式就是我们熟悉的误差平方和。总之， ML中很多model都假设数据或参数服从正态分布。

（贝叶斯、逻辑回归）

二、深度学习

（1）LSTM合集

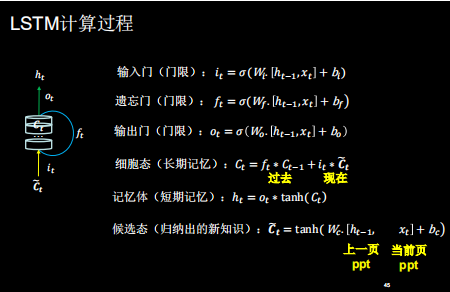
1.1 介绍LSTM

①传统RNN可以通过记忆体实现短期记忆进行连续数据的预测，但是当连续数据的序列变长时，会使展开时间步过长，在反向传播更新参数时梯度要按时间步连续相乘，会导致梯度消失。故提出长短记忆网络LSTM。

②长短记忆网络中引入了三个门限，三个都是当前时刻的输入特征xt和上个时刻的短期记忆ht-1的函数，wi、wf、wo都是待训练的参数矩阵。bi、bf、bo是带训练的偏置项。都经过sigmod激活函数，使门限的范围在0-1之间。

③细胞态表示长期记忆，等于上一时刻的长期记忆乘遗忘门 加上 当前时刻归纳出的新知识乘以输入门。

④记忆体表示短期记忆，属于长期记忆的一部分，是细胞态经过tanh激活函数乘以输出门的结果。

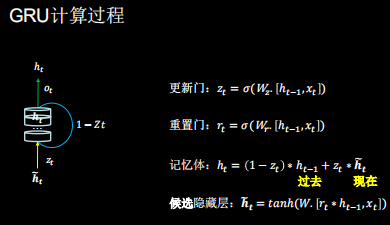
⑤候选态表示归纳出的待存入细胞态的新知识，是当前时刻的输入特征和上个时刻短期记忆ht-1的函数，wc和bc分别是带训练的参数矩阵和偏置项。

1.2介绍LSTM及其变种

<https://zhuanlan.zhihu.com/p/82385476>

<https://blog.csdn.net/u012223913/article/details/77724621>

GRU网络时LSTM的简化，GRU是记忆体ht融合了长期记忆和短期记忆，ht包含了过去信息ht-1和现在信息ht波浪号。现在信息ht波浪号是过去信息ht-1过重置门与当前输入共同决定。两个门限的取值范围都是0-1。



GRU和LSTM的性能在很多任务上不分伯仲。

**GRU 参数更少因此更容易收敛，但是数据集很大的情况下，LSTM表达性能更好**。

从结构上来说，GRU只有两个门（update和reset），LSTM有三个门（forget，input，output），GRU直接将hidden state 传给下一个单元，而LSTM则用memory cell 把hidden state 包装起来。

lstm为三个输入xt，ht-1， ct-1，两个输出。 gru为两个输入xt， ht-1，一个输出ht，输出即state。

lstm有三个门，输入输出忘记门。 gru有两个门，reset，update 门。

update 类似于 input gate和forget gate

1.3介绍一下LSTM，LSTM解决了RNN的什么问题，如何解决

长短期记忆（Long short-term memory, LSTM）是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

LSTM为长短期记忆，是一种变种的RNN，在RNN的基础上引入了细胞状态，根据细胞状态可决定哪些状态应该保留下来，哪些状态应该被遗忘，并使用输入门、遗忘门、输出门三种门来保持和控制信息

1.4 lstm为什么可以解决梯度消失



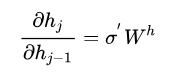
<https://www.zhihu.com/question/44895610>

<https://zhuanlan.zhihu.com/p/28749444>左边链接是以下解释的具体推导

对于RNN来说，前后两个step的hidden state中间经过了一层sigmoid，所以后向传播的时候梯度会乘上一个sigmoid的导数值；对于LSTM来说，前后两个step的hidden cell没有经过一个sigmoid层，而是乘了一个sigmoid的函数值 / 激活值（即LSTM的forget gate），所以后向传播的时候梯度也会乘上一个sigmoid的函数值。

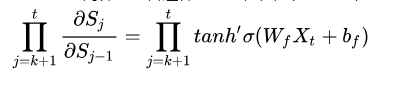
故每次对wh求偏导会乘sigmoid的导数，



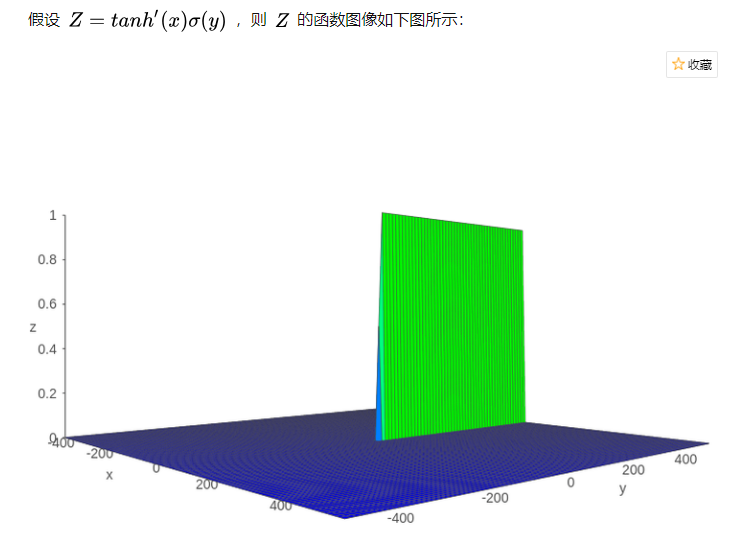


而对于lstm：对长期记忆体求导得到的是sigmoid本身相乘，故不会梯度消失。





也就是说，RNN在后向的时候梯度是不断乘以tanh激活函数的导数值，LSTM在后向的时候梯度是不断乘以sigmoid的函数值乘以tanh激活函数的导数值，而这两者的数值分布存在着显著差异。



可以看到该函数值基本上不是0就是1

1.5 LSTM的门控机制

各门控单元0/1输出。 门控单元输出是[0，1]实数区间的原因是阶跃激活函数无法反向传播进行优化， 所以各门控单元使用sigmoid激活函数去近似阶跃函数。 因此， 为了理解方便， 我们只需要考虑理想情况， 即各门控单元是{0，1}二值输出的，即门控单元扮演了电路中”开关”的角色， 用于控制信息传输的通断

（2）梯度消失、梯度爆炸的解决方法

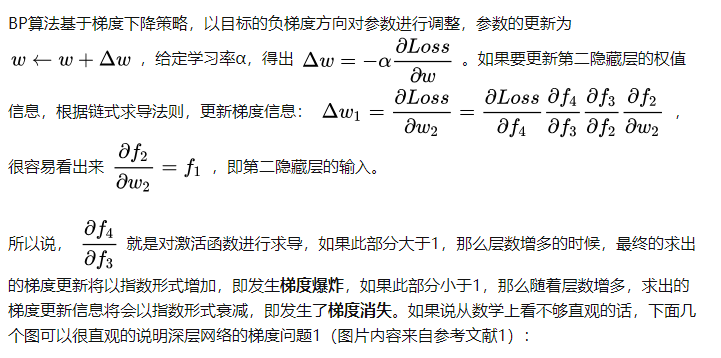
<https://www.jianshu.com/p/3f35e555d5ba>

<https://zhuanlan.zhihu.com/p/33006526>

梯度消失：对于一个含有三层隐藏层的简单神经网络来说，当梯度消失发生时，接近于输出层的隐藏层由于其梯度相对正常，所以权值更新时也就相对正常，但是当越靠近输入层时，由于梯度消失现象，会导致靠近输入层的隐藏层权值更新缓慢或者更新停滞。这就导致在训练时，只等价于后面几层的浅层网络的学习。

为什么越靠近输入层越会梯度消失

（https://zhuanlan.zhihu.com/p/33006526）

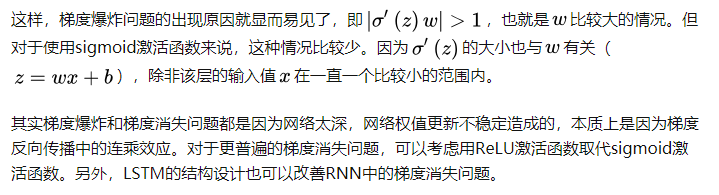


可以得到它Sigmoid函数图像，呈现一个驼峰状（很像高斯函数），从求导结果可以看出，Sigmoid导数的取值范围在0~0.25之间，而我们初始化的网络权值通常都小于1，因此，当层数增多时，小于0的值不断相乘，最后就导致梯度消失的情况出现。

**当神经网络层数很多的时候，因为需要反向求梯度，所以如果激活函数是sigmoid就会产生梯度消失，tanh也是一样，他们两个的梯度都小于1'而relu函数梯度等于1解决了梯度消失问题**

**两种情况下梯度消失经常出现，一是在深层网络中，二是采用了不合适的损失函数，比如sigmoid。梯度爆炸一般出现在深层网络和权值初始化值太大的情况下，**

同理，梯度爆炸的问题也就很明显了，就是当权值过大时，导致 ，最后大于1的值不断相乘，就会产生梯度爆炸。



解决方法：

2.1换用Relu、LeakyRelu、Elu等激活函数

ReLu：让激活函数的导数为1

LeakyReLu：包含了ReLu的几乎所有有点，同时解决了ReLu中0区间带来的影响

ELU：和LeakyReLu一样，都是为了解决0区间问题，相对于来，elu计算更耗时一些

2.2 BatchNormalization

见下一个问题

2.3 ResNet残差结构

2.4 LSTM结构

2.5 预训练加finetunning

此方法来自Hinton在06年发表的论文上，其基本思想是每次训练一层隐藏层节点，将上一层隐藏层的输出作为输入，而本层的输出作为下一层的输入，这就是逐层预训练。

训练完成后，再对整个网络进行“微调（fine-tunning）”。

此方法相当于是找全局最优，然后整合起来寻找全局最优，但是现在基本都是直接拿imagenet的预训练模型直接进行finetunning。

2.6梯度剪切、正则

这个方案主要是针对梯度爆炸提出的，其思想是设值一个剪切阈值，如果更新梯度时，梯度超过了这个阈值，那么就将其强制限制在这个范围之内。这样可以防止梯度爆炸。

另一种防止梯度爆炸的手段是采用权重正则化，正则化主要是通过对网络权重做正则来限制过拟合，但是根据正则项在损失函数中的形式

（3）batch normalization（批标准化）的过程和优缺点

<https://zhuanlan.zhihu.com/p/26138673>

一般来说，如果模型的输入特征不相关且满足标准正态分布时，模型的表现一般较好。在训练神经网络模型时，我们可以事先将特征去相关并使得它们满足一个比较好的分布，这样，模型的第一层网络一般都会有一个比较好的输入特征，但是随着模型的层数加深，网络的非线性变换使得每一层的结果变得相关了，且不再满足正态分布。更糟糕的是，可能这些隐藏层的特征分布已经发生了偏移。

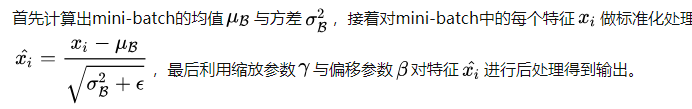
他们提出在层与层之间加入Batch Normalization层。训练时，BN层利用隐藏层输出结果的均值与方差来标准化每一层特征的分布，并且维护所有mini-batch数据的均值与方差，最后利用样本的均值与方差的无偏估计量用于测试时使用。

BN层的作用：①使得模型训练收敛的速度更快

②（防止梯度消失和梯度爆炸） ③模型隐藏输出特征的分布更稳定，更利于模型的学习

过程：

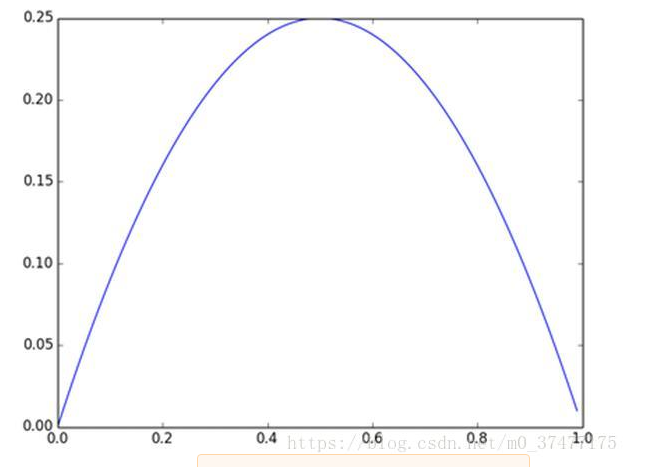
前向传播：



假设没有经过BN调整前x的原先正态分布均值是-6，方差是1，那么意味着95%的值落在了[-8,-4]之间，那么对应的Sigmoid（x）函数的值明显接近于0，这是典型的梯度饱和区，在这个区域里梯度变化很慢，为什么是梯度饱和区？请看下sigmoid(x)如果取值接近0或者接近于1的时候对应导数函数取值，接近于0，

3.2 batchnorm能提升效果的原理，从梯度下降法角度解释

假设没有经过BN调整前x的原先正态分布均值是-6，方差是1，那么意味着95%的值落在了[-8,-4]之间，那么对应的Sigmoid（x）函数的值明显接近于0，这是典型的梯度饱和区，在这个区域里梯度变化很慢，为什么是梯度饱和区？请看下sigmoid(x)如果取值接近0或者接近于1的时候对应导数函数取值，接近于0。 而假设经过BN后，均值是0，方差是1，那么意味着95%的x值落在了[-2,2]区间内，很明显这一段是sigmoid(x)函数接近于线性变换的区域，意味着x的小变化会导致非线性函数值较大的变化，也即是梯度变化较大，对应导数函数图中明显大于0的区域，就是梯度非饱和区。



3.3 layer normalization

但是有些场景是不能使用BN的，例如batchsize较小或者在RNN中，这时候可以选择使用LN，LN得到的模型更稳定且起到正则化的作用。

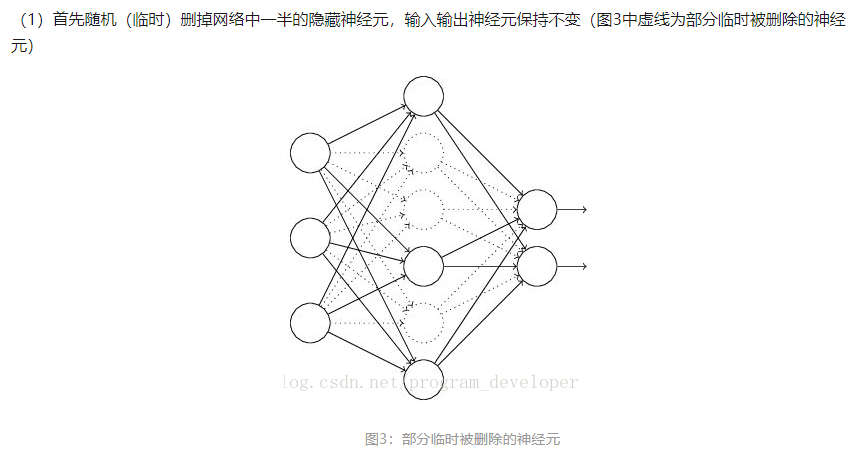
不同的是BN取的是不同样本的同一个特征，而LN取的是同一个样本的不同特征。在BN和LN都能使用的场景中，BN的效果一般优于LN，原因是基于不同数据，同一特征得到的归一化特征更不容易损失信息。

（4）Dropout合集

4.1神经网络利用dropout和多项式回归利用正则项减轻过拟合的本质是什么？

Dropout：就是在神经网络的Dropout层，为每个神经元结点设置一个随机消除的概率，对于保留下来的神经元，我们得到一个节点较少，规模较小的网络进行训练。（注意消除的是隐藏神经元）

Dropout说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征



4.2为什么说Dropout可以解决过拟合

**①取平均的作用**： 先回到标准的模型即没有dropout，我们用相同的训练数据去训练5个不同的神经网络，一般会得到5个不同的结果，此时我们可以采用 “5个结果取均值”或者“多数取胜的投票策略”去决定最终结果。例如3个网络判断结果为数字9,那么很有可能真正的结果就是数字9，其它两个网络给出了错误结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。dropout掉不同的隐藏神经元就类似在训练不同的网络，随机删掉一半隐藏神经元导致网络结构已经不同，整个dropout过程就相当于对很多个不同的神经网络取平均。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。

**②减少神经元之间复杂的共适应关系**： 因为dropout程序导致两个神经元不一定每次都在一个dropout网络中出现。这样权值的更新不再依赖于有固定关系的隐含节点的共同作用， 阻止了某些特征仅仅在其它特定特征下才有效果的情况 。迫使网络去学习更加鲁棒的特征 ，这些特征在其它的神经元的随机子集中也存在。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的特征。从这个角度看dropout就有点像L1，L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高。

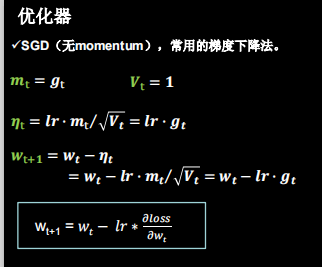
（5）说下adam的思想

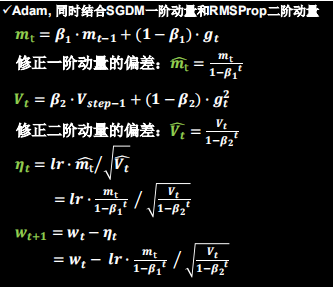
**<https://zhuanlan.zhihu.com/p/32698042?utm_source=wechat_session&utm_medium=social&utm_oi=759055729532346368>**

**<https://zhuanlan.zhihu.com/p/27449596?utm_source=wechat_session&utm_medium=social&utm_oi=759055729532346368>**

**答主的参考答案：**

答：它是个**自适应优化**器，对更新的步长计算，能够从**梯度均值及梯度平方**两个角度进行自适应地调节，而不是直接由当前梯度决定。它的每个step都会限制在一个范围内，所以它对每一步走得会很谨慎，不容易发生学习不起来的情况。





**优化算法的功能，是通过改善训练方式，来最小化(或最大化)损失函数E(x)**。

传统的批量梯度下降将计算整个数据集梯度，但只会进行一次更新，因此在处理大型数据集时速度很慢且难以控制，甚至导致内存溢出。

使用标准形式的批量梯度下降（使用所有样本数求均值）还有一个问题，就是在训练大型数据集时存在冗余的权重更新。

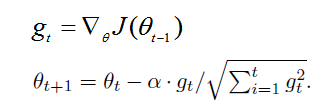
一、两种改进的梯度下降法:

①随机梯度下降(SDG)：在每次更新时用1个样本，可以看到多了随机两个字，随机也就是说我们用样本中的一个例子来近似我所有的样本，来调整w。

②小批量梯度下降：在每次更新时用b个样本,其实批量的梯度下降就是一种折中的方法，他用了一些小样本来近似全部的，其本质就是我1个指不定不太准，那我用个30个50个样本那比随机的要准不少了吧，而且批量的话还是非常可以反映样本的一个分布情况的。

1. 自适应优化

①AdaGrad：针对简单的SGD及Momentum存在的问题，2011年John Duchi等发布了AdaGrad优化算法(Adaptive Gradient，自适应梯度)，它能够对每个不同的参数调整不同的学习率，对频繁变化的参数以更小的步长进行更新，而稀疏的参数以更大的步长进行更新。

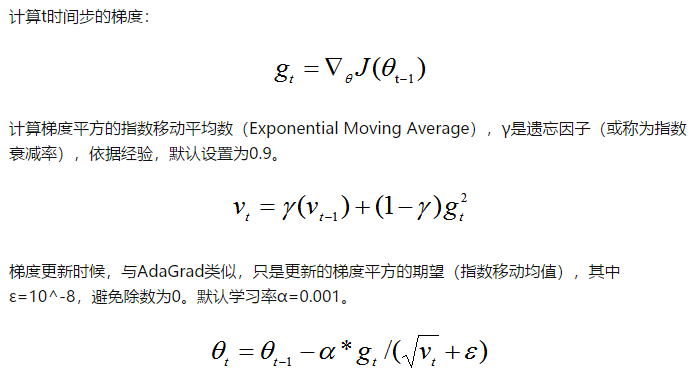


与SGD的核心区别在于计算更新步长时，增加了分母：梯度平方累积和的平方根。此项能够累积各个参数的历史梯度平方，频繁更新的梯度，则累积的分母项逐渐偏大，那么更新的步长(stepsize)相对就会变小，而稀疏的梯度，则导致累积的分母项中对应值比较小，那么更新的步长则相对比较大。

优势：在数据分布稀疏的场景，能更好利用稀疏梯度的信息，比标准的SGD算法更有效地收敛。

缺点：主要缺陷来自分母项的对梯度平方不断累积，随之时间步地增加，分母项越来越大，最终导致学习率收缩到太小无法进行有效更新。

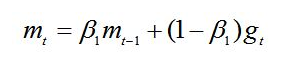
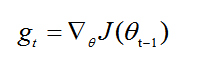
②RMSProp：是Geoffrey Hinton教授在教案中提到的算法，结合梯度平方的指数移动平均数来调节学习率的变化。能够在不稳定（Non-Stationary）的目标函数情况下进行很好地收敛。



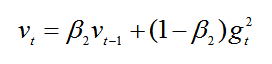
优势：能够克服AdaGrad梯度急剧减小的问题，在很多应用中都展示出优秀的学习率自适应能力。尤其在不稳定(Non-Stationary)的目标函数下，比基本的SGD、Momentum、AdaGrad表现更良好。

③Adam优化器： Kingma和Lei Ba两位学者提出了Adam优化器，结合AdaGrad和RMSProp两种优化算法的优点。对梯度的一阶矩估计（First Moment Estimation，即梯度的均值）和二阶矩估计（Second Moment Estimation，即梯度的未中心化的方差）进行综合考虑，计算出更新步长。

①计算梯度的指数移动平均数，m0 初始化为0。类似于Momentum算法，综合考虑之前时间步的梯度动量。β1 系数为指数衰减率，控制权重分配（动量与当前梯度），通常取接近于1的值。



②其次，计算梯度平方的指数移动平均数，v0初始化为0。β2 系数为指数衰减率，控制之前的梯度平方的影响情况。类似于RMSProp算法，对梯度平方进行加权均值。默认为0.999



③由于m0初始化为0，会导致mt偏向于0，尤其在训练初期阶段。所以，此处需要对梯度均值mt进行偏差纠正，降低偏差对训练初期的影响。

④与m0 类似，因为v0初始化为0导致训练初始阶段vt 偏向0，对其进行纠正。



⑤更新参数，初始的学习率α乘以**梯度均值 与梯度方差 的平方根**之比。其中默认学习率α=0.001。ε=10^-8，避免除数变为0。

由表达式可以看出，对更新的步长计算，能够从梯度均值及梯度平方两个角度进行自适应地调节，而不是直接由当前梯度决定。



（6）CNN合集

6.1卷积层的基本介绍

①输入特征图的深度决定当前层的卷积核深度。如果输入特征是单通道灰度图，使用深度为1的单通道卷积核；如果输入特征是三通道彩色图，可以使用`3\*3\*3`的卷积核（或者5×5×3的卷积核），总之要使\*\*卷积核的通道数与输入特征图的通道数一致\*\*，因为要想让卷积核与输入特征图对应点匹配上，必须让卷积核的深度与输入特征图的深度一致

②当前卷积核的个数决定了当前层输出特征图的深度；由于每个卷积核在卷积计算后，会得到一张输出特征图，所以当前层使用了几个卷积核，就会有几个输出特征图

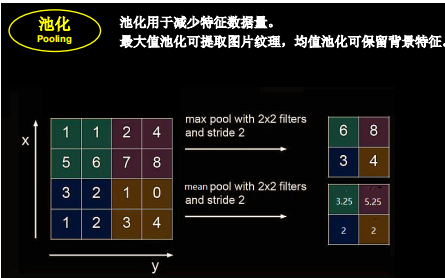
③如果你觉得某层模型的特征提取能力不足，可以在这一层多用几个卷积核提高这一层的特征提取能力；

每次在执行卷积计算时，卷积核里的参数固定，在每次反向传播时，小颗粒存储的待训练参数会被梯度下降法更新，卷积则是用立体卷积核实现了参数共享。



用2×2的池化层对输入图片以2为步长进行池化，输出图片将变为输入图片的1/4大小

最大池化是用2×2的池化核框柱四个像素点选择最大的6输出，步长为2.滑到红色位置输出8。均值池化类似。



6.2卷积层相比全连接层的优点

①卷积层相较于全连接层需要训练的参数更少，所以神经网络的设计离不开卷积层

②卷积层通过参数共享和稀疏连接两种方式来保证单层卷积中的训练参数少

③由上，也可以推出：卷积核（即上述所说的特征检测器或者权重）的大小会影响训练参数数量，但相较于全连接来说是小巫见大巫

6.3 CNN卷积层的作用

提取特征。“不全连接（dropout），参数共享（九个合成一个共享一个参数）”的特点大大降低了网络参数，保证了网络的稀疏性，防止过拟合。之所以可以“参数共享”，是因为样本存在局部相关的特性。

6.4 池化的作用

减少特征数量，最大值池化可以提取图片纹理，均值池化可以保留背景特征

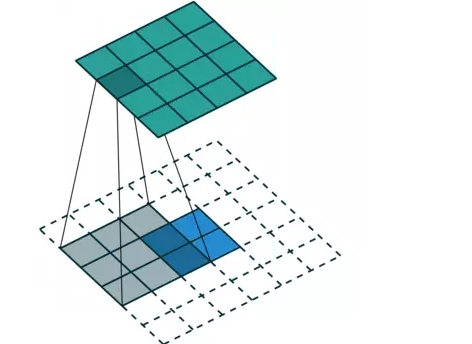
6.5 我所知道的卷积种类

<https://my.oschina.net/u/876354/blog/3064227>

卷积现在已衍生出了各种类型，包括标准卷积、反卷积、可分离卷积、分组卷积等等。

①转置卷积（反卷积）：卷积是对输入图像提取出特征（可能尺寸会变小），而所谓的“反卷积”便是进行相反的操作。但这里说是“反卷积”并不严谨，因为并不会完全还原到跟输入图像一样，一般是还原后的尺寸与输入图像一致，主要用于向上采样。从数学计算上看，“反卷积”相当于是将卷积核转换为稀疏矩阵后进行转置计算，因此，也被称为“转置卷积”。

如下图，在2x2的输入图像上应用步长为1、边界全0填充的3x3卷积核，进行转置卷积（反卷积）计算，向上采样后输出的图像大小为4x4



②空洞卷积（膨胀卷积）

为扩大感受野，在卷积核里面的元素之间插入空格来“膨胀”内核，形成“空洞卷积”（或称膨胀卷积），并用膨胀率参数L表示要扩大内核的范围，即在内核元素之间插入L-1个空格。当L=1时，则内核元素之间没有插入空格，变为标准卷积。

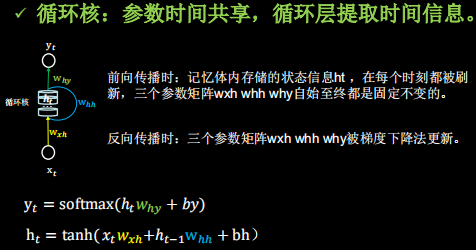
（7）RNN合集

7.1 RNN介绍

7.1.1

①有些数据是与时间序列相关的，可以根据上文预测出下文，这要借助循环核具有的记忆力，实现时间序列的信息提取，输入xt和yt的维度由记忆体的个数确定，ht表示每个时刻记忆体内的存储信息，IMG_256 表当前时刻记忆体存储信息的计算公式，即上一时刻的状态信息乘whh加上这一时刻的输入和参数加上偏置。

②输出特征如下所示IMG_257 （softmax是使输出满足概率分布）



7.1.2

横向增加whh作为两层的连接，是乘前面记忆体提取到的特征。

总的过程还是训练参数矩阵，得到训练最好的参数矩阵后，输入x，得到预测输出的y。

与脑的预测相似，你当前的预测推理判断，是根据你以往的知识积累训练得到的。

循环神经网络，就是借助循环核提取时间特征后，送入全连接网络。实现连续数据的预测



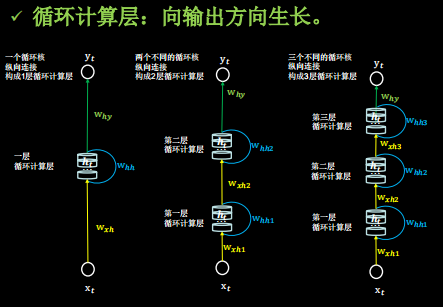
7.1.3

纵向增加是增加记忆体的个数

①每个循环核构成一个循环计算层，循环计算层的层数是向输出方向增长的，下图从左往右依次有1、2、3层循环计算层（1、2、3个循环核）。

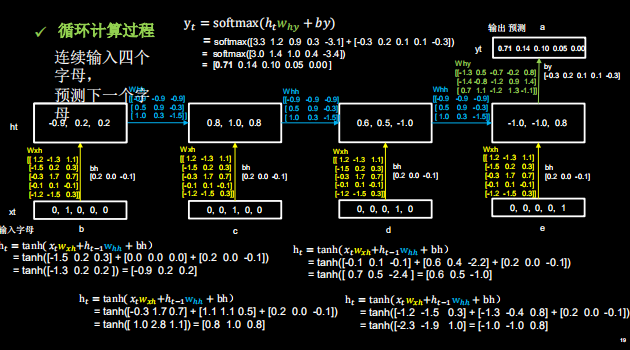
②每个循环核中记忆体的个数，由自身的需求而定

③卷积层实现的是空间抽取，循环层实现的是时间抽取



7.1.4

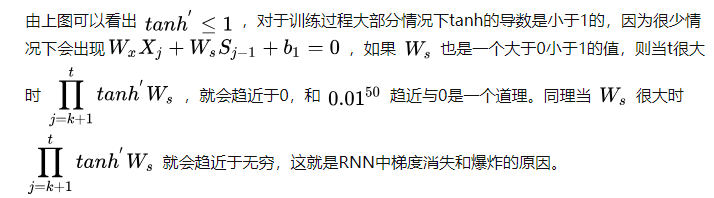
感受把时间核按时间步展开，连续输入多个字母预测下一个字母,前面几层不用输出ht,最后一层要输出ht给全连接层。



7.2产生梯度消失和爆炸的原因

<https://zhuanlan.zhihu.com/p/28687529>

S2、s1也是关于wx和ws的函数，会一直按状态连乘，并且tanh是小于1的



（8）模型权重初始化的方法

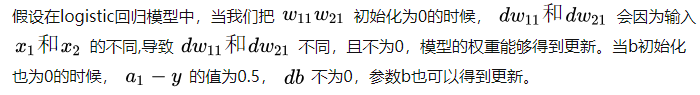
<https://zhuanlan.zhihu.com/p/133190518>

①把w初始化为0

<https://zhuanlan.zhihu.com/p/75879624>

我们在线性回归，logistics回归的时候，基本上都是把参数初始化为0，我们的模型也能够很好的工作。然后在神经网络中，把w初始化为0是不可以的。这是因为如果把w初始化0，那么每一层的神经元学到的东西都是一样的（输出是一样的），而且在bp的时候，每一层内的神经元也是相同的，因为他们的gradient相同（**输出一样则下一层的输入也一样，当wi和bi都初始化为同样值，每个梯度也一样**）。

**神经网络与之相比的不同点在于，神经网络的隐藏层存在，导致隐藏层对应的每个神经元都是对所有输入xi的求和，而对于逻辑回归来说，不同xi对应的参数不同，故可以将参数设为相同的**

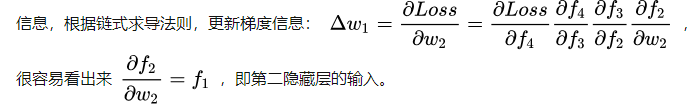


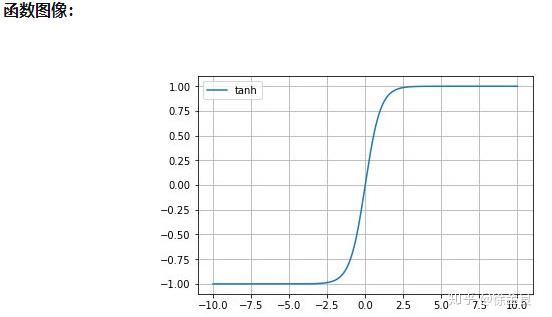
②对w随机初始化

但是随机初始化也有缺点，np.random.randn()其实是一个均值为0，方差为1的高斯分布中采样。当神经网络的层数增多时，会发现越往后面的层的激活函数（使用tanH）的输出值几乎都接近于0。

根据f=W.X+b可知反向传播时对权重W求偏导时，当前层输出的数值X即是反向传播时计算的梯度中的乘积因子，导致梯度非常小，使得参数更新困难。

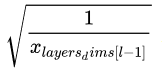
下图改为w2





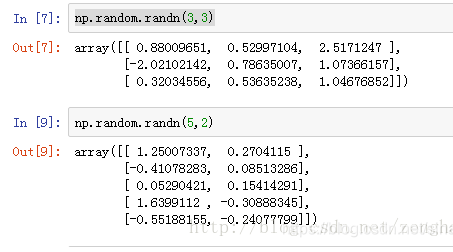
sigmoid和tanh激活函数有共同的缺点：即在z很大或很小时，梯度几乎为零，因此使用梯度下降优化算法更新网络很慢。

③Xavier initialization

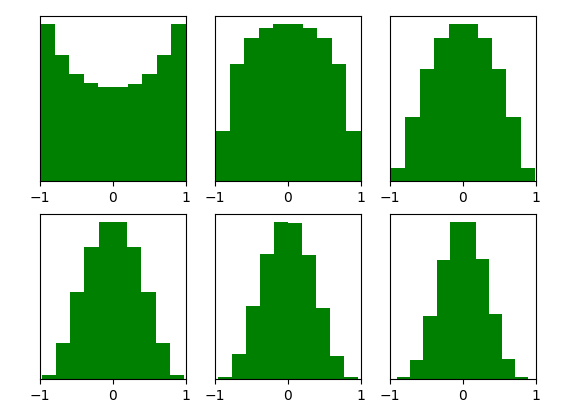
xavier initialization是Glorot等人为了解决随机初始化的问题提出来的另一种初始化方法，思想就是尽可能的让输入和输出服从相同的分布，即保持输入和输出的方差一致。即将随机初始化的值乘以缩放因子：

Tf的实现过程

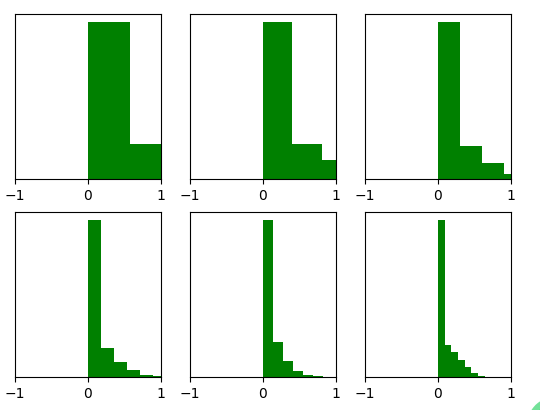




来看下Xavier initialization后每层的激活函数输出值的分布：



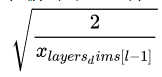
能够看出，深层的激活函数输出值还是非常漂亮的服从标准高斯分布。虽然Xavier initialization能够很好的 tanH 激活函数，但是对于目前神经网络中最常用的ReLU激活函数，还是无能能力。

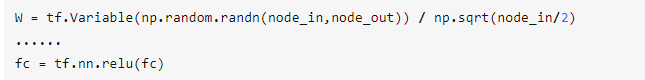


当达到5，6层后几乎又开始趋向于0，更深层的话很明显又会趋向于0。

④He initialization

He初始化可以解决上面在ReLU激活函数是Xavier效果不好的问题。其思想是：在ReLU网络中，假定每一层有一半的神经元被激活，另一半为0，要保持方差不变，只需要在Xavier的基础上再除以2。即缩放因子变为：

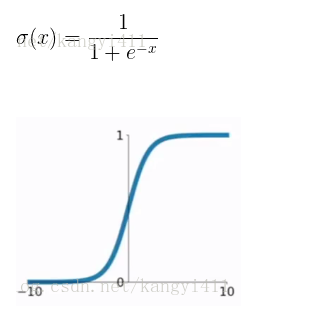




现在神经网络中，隐藏层常使用ReLU，权重初始化常用He initialization这种方法。

1. 激活函数合集

①sigmoid激活函数



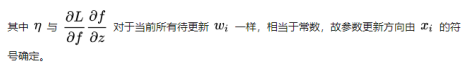
缺点：

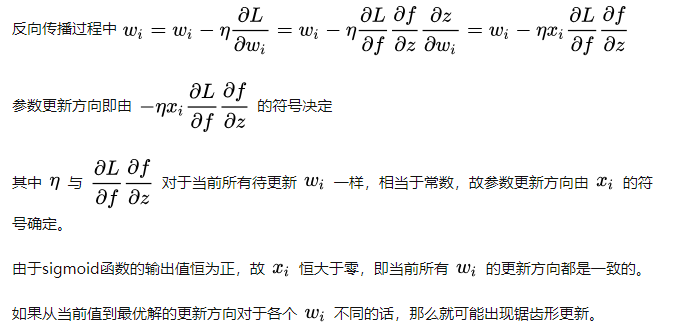
1. 当输入稍微远离了坐标原点，函数的梯度就变得很小了，几乎为零。在神经网络反向传播的过程中，我们都是通过微分的链式法则来计算各个权重w的微分的。当反向传播经过了sigmod函数，这个链条上的微分就很小很小了，况且还可能经过很多个sigmod函数，最后会导致权重w对损失函数几乎没影响，这样不利于权重的优化，这个问题叫做梯度饱和，也可以叫梯度弥散。

2.函数输出不是以0为中心的，这样会使权重更新效率降低。对于这个缺陷，在斯坦福的课程里面有详细的解释。

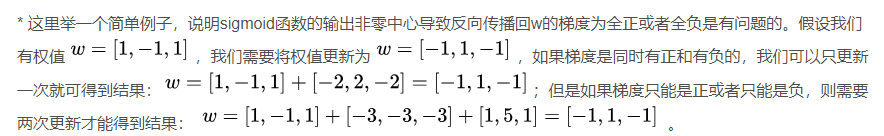
<https://blog.csdn.net/weixin_38646522/article/details/79534677>

即w向量中每一维的梯度更新方向是一致的，由于sigmoid的输出全大于0，故下一层输入x向量的各个分量xi都大于0，导致w向量的各个分量的更新方向一样。

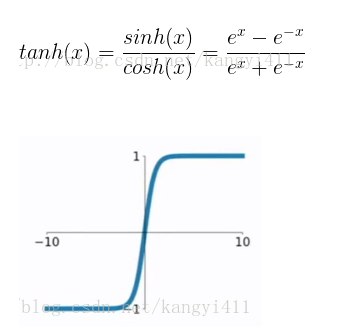




因为多层神经网络时，上一层的输出是sigmoid，故下一层的输入始终为正，而故更新方向要么全为正要么全为负。



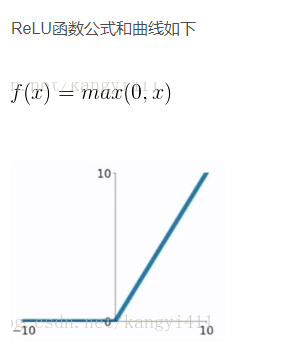
②tanh激活函数



sigmoid和tanh激活函数有共同的缺点：即在z很大或很小时，梯度几乎为零，因此使用梯度下降优化算法更新网络很慢。

不同的是输出区间，tanh的输出区间是在(-1,1)之间，而且整个函数是以0为中心的，这个特点比sigmod的好。

③relu激活函数



1) 在输入为正数的时候，不存在梯度饱和问题。

2) 计算速度要快很多。ReLU函数只有线性关系，不管是前向传播还是反向传播，都比sigmod和tanh要快很多。（sigmod和tanh要计算指数，计算速度会比较慢）

缺点也是有的：

1) 当输入是负数的时候，ReLU是完全不被激活的，这就表明一旦输入到了负数，ReLU就会死掉。这样在前向传播过程中，还不算什么问题，有的区域是敏感的，有的是不敏感的。但是到了反向传播过程中，输入负数，梯度就会完全到0，这个和sigmod函数、tanh函数有一样的问题。

1. 我们发现ReLU函数的输出要么是0，要么是正数，这也就是说，ReLU函数也不是以0为中心的函数。

④

1. tensorflow实现一个dnn的流程



三、推荐系统

（1）推荐问题中的几种排序

<https://blog.csdn.net/seoyundu/article/details/101848917>

1.1协同过滤

1.2矩阵分解

1.3逻辑回归

1.4 FM及FFM

1.5 GBDT + LR

1.6 LS-PLM

（2）是否了解过一些深度学习的排序方法

2.1 DEEP CROSSING模型

2.2 NeuralCF模型

2.3 PNN模型

2.4 Wide&deep模型

2.5 deepfm

2.6 AFM

2.7 DIN

2.8 DIEN

2.9强化学习

（3）常用的推荐系统离线评估方法

①Holdout 检验、交叉检验和自助法：Holdout 检验是最基础，最常用的离线评估方法，它将原始的样本集合随机划分为训练集和测试集两部分。

虽然 Holdout 检验很简单实用，但它的缺点也很明显，就是评估的结果有一定随机性，因为训练集和验证集的划分是随机的，所以如果只进行少量的 Holdout 检验，得到的评估指标会存在一定的波动。那为了消除这种随机性，我们就要使用“交叉检验”的方法。

②时间切割

说完了前三种方法，我们再来看时间切割法。在“模型实战准备（二）”那节课里，我们曾经讲过一个概念，叫“未来信息”。它是说，如果我们在 t 时刻进行模型预测，那么 t+1 时刻的信息就是未来信息。在构建特征工程的时候，我们要避免引入“未来信息”。

其实，在进行模型评估的时候，我们同样不应该在训练集中包含“未来”的样本。怎么理解这句话呢？比如，我们所有的样本数据分布在 t0到 tn这样的时间轴上，如果训练样本是通过随机采样得到的，那么训练数据也会分布在 t0到 tn上，同样，测试数据也会分布在 t0到 tn上。

如果你细想，这个事情其实是有点反常理的。因为训练模型的时候，我们已经使用了 tn这个时间窗口的数据，结果你却用它来预测 t0的事件，这不是很荒谬吗？这就相当于你有一个时光机，已经穿越到了明天，知道股票会涨，结果你又穿越回来，预测说明天股票会涨，这哪是预测呢？这就是“作弊”

（4）怎么进行召回

<https://zhuanlan.zhihu.com/p/115690499?from_voters_page=true>

召回是推荐系统的第一阶段，负责将海量的候选集快速缩小为几百到几千的规模。

4.1多路召回

多路召回是指采用不同策略、特征或者简单模型分别召回同一部分候选集，然后把候选集混合在一起供后续排序模型使用。

各个简单模型保证候选集的快速召回，从不同角度设计的策略保证召回率接近理想的状态。

常见的有：热门新闻、兴趣标签、协同过滤、最近流行、朋友喜欢等等

4.2 基于embedding的召回方法

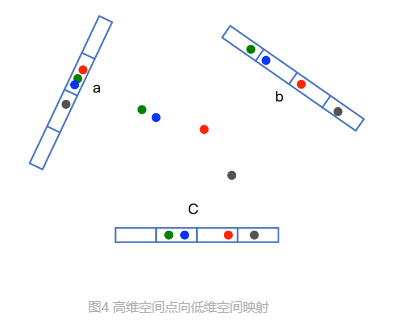
第一步，我们获取用户的 Embedding。第二步，我们获取所有物品的候选集，并且逐一获取物品的 Embedding，计算物品 Embedding 和用户 Embedding 的相似度。第三步，我们根据相似度排序，返回规定大小的候选集。

在这三步之中，最主要的时间开销在第二步，虽然它的时间复杂度是线性的，但当物品集过大时（比如达到了百万以上的规模），线性的运算也可能造成很大的时间开销。于是乎可以使用局部敏感hash进行快速的embedding最近邻计算。

局部敏感哈希的基本思想是希望让相邻的点落入同一个“桶”，这样在进行最近邻搜索时，我们仅需要在一个桶内，或相邻的几个桶内的元素中进行搜索即可。如果保持每个桶中的元素个数在一个常数附近，我们就可以把最近邻搜索的时间复杂度降低到常数级别。

如果将高维空间中的点向低维空间进行映射，其欧式相对距离是不是会保持不变呢？以图 4 为例，图 4 中间的彩色点处在二维空间中，当我们把二维空间中的点通过不同角度映射到 a、b、c 这三个一维空间时，可以看到原本相近的点，在一维空间中都保持着相近的距离。而原本远离的绿色点和红色点在一维空间 a 中处于接近的位置，却在空间 b 中处于远离的位置。

因此我们可以得出一个定性的结论：欧式空间中，将高维空间的点映射到低维空间，原本接近的点在低维空间中肯定依然接近，但原本远离的点则有一定概率变成接近的点。



利用低维空间可以保留高维空间相近距离关系的性质，我们就可以构造局部敏感哈希“桶”。对于 Embedding 向量来说，由于 Embedding 大量使用内积操作计算相似度，因此我们也可以用内积操作来构建局部敏感哈希桶。假设 v 是高维空间中的 k 维 Embedding 向量，x 是随机生成的 k 维映射向量。那我们利用内积操作可以将 v 映射到一维空间，得到数值 h(v)=v⋅x。

而且，我们刚刚说了，一维空间也会部分保存高维空间的近似距离信息。因此，我们可以使用哈希函数 h(v) 进行分桶，公式为：hx,b(v)=⌊wx⋅v+b​] 。其中， ⌊⌋ 是向下取整操作， w 是分桶宽度，b 是 0 到 w 间的一个均匀分布随机变量，避免分桶边界固化。

不过，映射操作会损失部分距离信息，如果我们仅采用一个哈希函数进行分桶，必然存在相近点误判的情况，因此，我们可以采用 m 个哈希函数（**更改****相应的参数再內积**）同时进行分桶。如果两个点同时掉进了 m 个桶，那它们是相似点的概率将大大增加。通过分桶找到相邻点的候选集合后，我们就可以在有限的候选集合中通过遍历找到目标点真正的 K 近邻了。（利用局部敏感hash筛选掉大部分，在精密查找）

更详细的内容见：

<https://time.geekbang.org/column/article/301739?code=egRnBwnBkdTKZHG8xAk6IS5Ox3x-yu812NEgUpSwhok%3D>

（5）DIN和DIEN合集

5.1 DIN基本介绍

Base Model 是一个典型的 Embedding MLP 的结构。它的输入特征有用户属性特征（User Proflie Features）、用户行为特征（User Behaviors）、候选广告特征（Candidate Ad）和场景特征（Context Features）。

与 Base Model 相比，DIN 为每个用户的历史购买商品加上了一个激活单元（Activation Unit），这个激活单元生成了一个权重，这个权重就是用户对这个历史商品的注意力得分，权重的大小对应用户注意力的高低。

它的输入是当前这个历史行为商品的 Embedding，以及候选广告商品的 Embedding。我们把这两个输入 Embedding，与它们的外积结果连接起来形成一个向量，再输入给激活单元的 MLP 层，最终会生成一个注意力权重，这就是激活单元的结构。简单来说，激活单元就相当于一个小的深度学习模型，它利用两个商品的 Embedding，生成了代表它们关联程度的注意力权重

5.2 DIEN的基本介绍

5.3 两者区别

1. FM合集

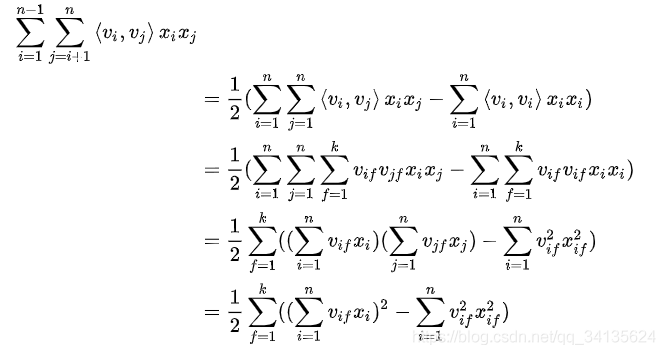
推导过程：<https://zhuanlan.zhihu.com/p/109980037>

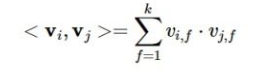
6.1 FM了解么，具体怎么做的

用两个特征向量对应隐权重向量（隐权重向量类似于embedding向量,是通过矩阵分解得到的）的內积作为两个特征交叉的权重，送入逻辑回归中训练。

6.2 怎么解决权重系数难训练的问题

FM通过特征隐向量的方式，将级别的权重参数，减少到O(nk),其中**n为特征的维度，k是隐向量的维度**。过程见fm的推导。





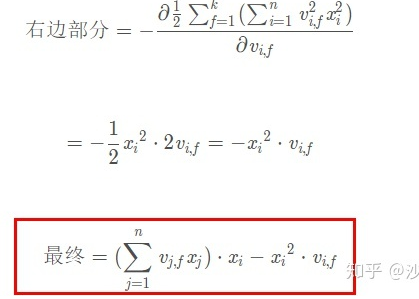
使用梯度下降法训练时，也可以将其降为O(nk),过程推导。

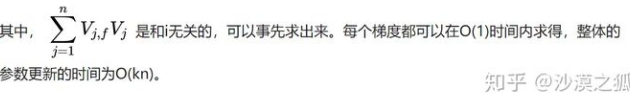
6.3 梯度怎么更新的

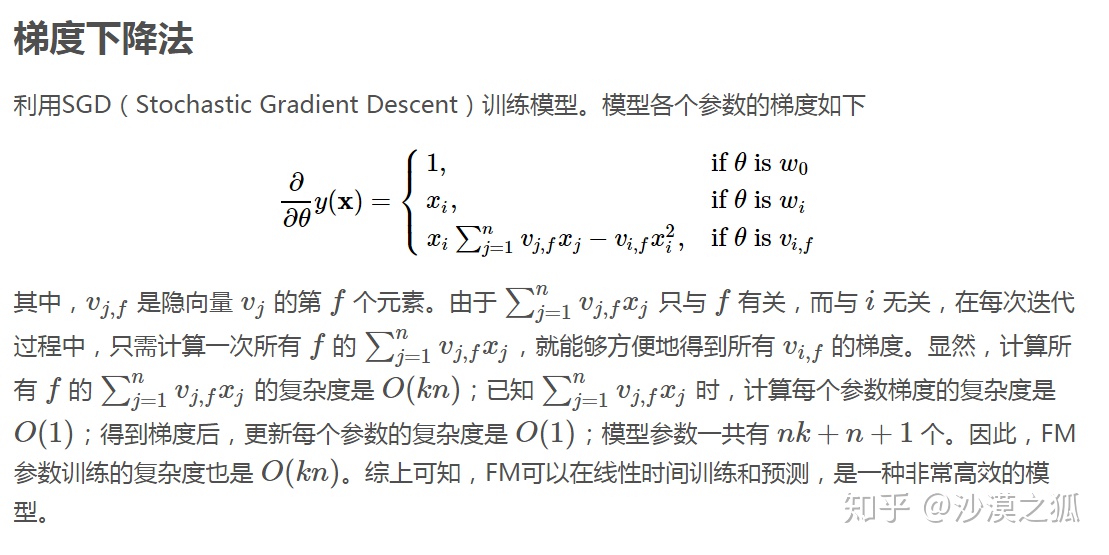
左边部分的对变量f求和直接去掉了，因为是对Vif求导，其他f不一样的直接忽略，故只剩下f对得上的，求和符号就消失了。

注意求和符号的括号是在求和符号外面，故事相乘的关系，最终写成j









6.4 相较于LR的优势在哪里呢？

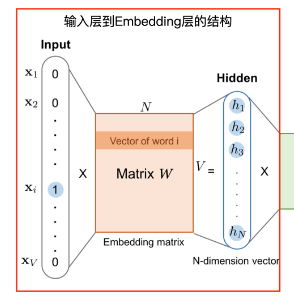
具备特征交叉功能。最后还是将特征交叉输入模型

（7）DeepFM

7.1 embedding层是怎么训练的

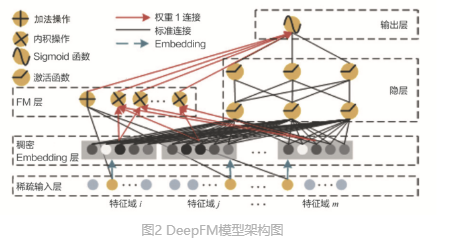
Embedding 层。Embedding 层就是为了把稀疏的 One-hot 向量转换成稠密的 Embedding 向量而设置的，我们需要注意的是，Embedding 层并不是全部连接起来的，而是每一个特征对应一个 Embedding 层，不同 Embedding 层之间互不干涉。

Embedding 层的内部结构：Embeding 层的结构就是 Word2vec 模型中从输入神经元到隐层神经元的部分（如图 2 红框内的部分）。参照下面的示意图，我们可以看到，这部分就是一个从输入层到隐层之间的全连接网络。



一般来说，Embedding 向量的维度应远小于原始的稀疏特征向量，按照经验，几十到上百维就能够满足需求，这样它才能够实现从稀疏特征向量到稠密特征向量的转换。

7.2 结构是什么样的



（8）协同过滤和lmf算法合集（矩阵分解合集）

8.1协同过滤介绍

利用用户对商品的历史评价信息生成共现矩阵，找到与用户X兴趣最相似的n个用户，综合相似用户对某个物品的评价，做出预测。

其中相似度计算（判断两个向量相似度）常用的有：欧式距离、余弦相似度计算、皮尔逊相关系数。

综合用户评价方法：加权平均之和（相似度和评分的加权平均和）。

更常用的是物品itemcf，即基于物品相似度进行推荐，计算共现矩阵（m×n）中物品列向量的相似度得到物品之间的相似矩阵（n×n），再找到**用户的历史正反馈物品**的相似物品进行推荐

Usercf场景：新闻推荐（适用于发现热点、跟踪热点的趋势）

Itemcf场景：电商（找到选择的相似物品）

Usercf是计算行与行间的相似度，Itemcf是计算列与列之间的相似矩阵。

协同过滤的缺点： ①热门物品具有很强的头部效应，容易跟大量物品相似（也可以理解为处理稀疏矩阵的能力弱，因为稀疏矩阵中的1少，即喜欢它的用户少，不热门）；（具体理解见深度学习推荐系统P20）

8.2 lmf算法计算矩阵有几种方式

Lmf基本介绍：**由协同过滤的共现矩阵生成用户矩阵（m×k）×物品矩阵（k×n）。用户和物品的隐向量（用户与物品的隐向量在同一坐标系中）（都是k维，用户每一行就是一个用户的隐向量，物品的每一列就是一个物品隐向量），找到与用户隐向量距离最接近的topk物品隐向量。**

共现矩阵分解过程：用户矩阵（m×k）×物品矩阵（k×n）。k的大小决定了隐向量表达能力的强弱，k值越小、信息越少，泛化能力越高。

利用用户矩阵U和物品矩阵V，用户u对物品i的预估评分：



其中是用户u在用户矩阵U中对应的行向量。是物品i在物品矩阵中的列向量。

矩阵分解的三种方法：特征值分解、奇异值分解、梯度下降。特征值分解只能作用于方阵。

①奇异值分解（SVD）：



取对角矩阵中较大的k个元素作为隐含特征，故可以分解为



缺点：①要求共现矩阵是稠密的；②计算复杂度太高O(mn2)

②梯度下降法：

梯度下降法的目标函数由原始评分与预测评分的差值、正则化两部分组成。



分别对和向量求偏导完成梯度下降更新参数。

8.3 矩阵分解的优缺点

优点：①泛化能力强（能一定程度解决数据稀疏的问题）；

②空间复杂度低：不需要存储较大的用户相似性和物品性矩阵，只需要存用户和物品的隐向量，由n平方复杂度降到(n+m)×k。

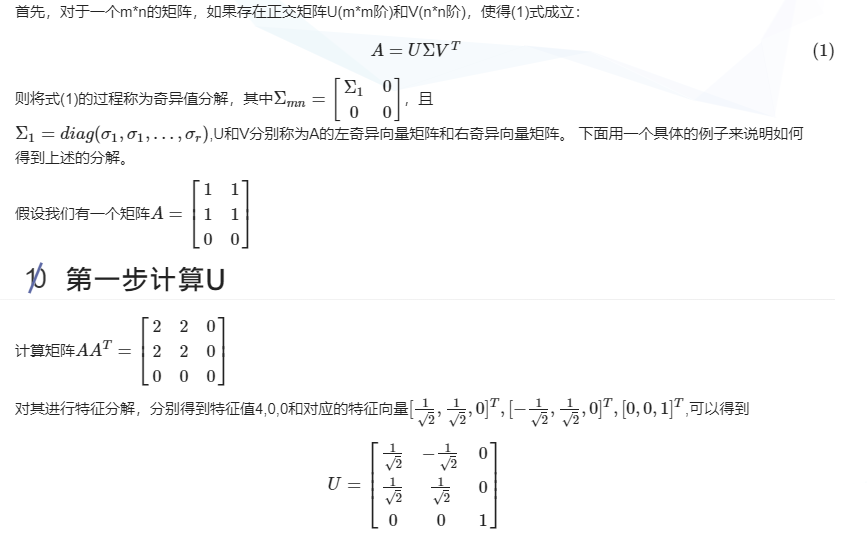
③与embedding相似，分解结果便于与其他特征组合，便于与深度学习结合。

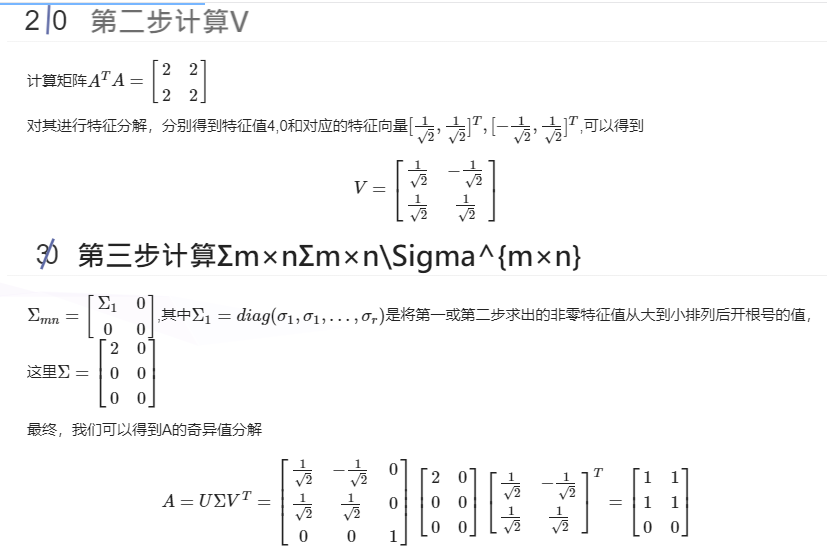
缺点：①缺乏用户历史行为时难以预测；②丢失很多组合特征，上下文特征等信息。

8.4 奇异值分解求解过程（矩阵分解）

<https://www.cnblogs.com/marsggbo/p/10155801.html>

特征值只能作用在一个m×m的正方矩阵上，而奇异值分解则可以作用在一个m×n的长方矩阵上。其次，奇异值分解同时包含了旋转、缩放和投影三种作用，(1)式中，U和V都起到了对A旋转的作用，而Σ起到了对A缩放的作用。特征值分解只有缩放的效果。





1. 推荐系统的框架了解（召回和ranking）
2. Embedding技术合集

10.1 基本介绍

embedding就是一个低维稠密的向量“表示”一个对象，Embedding向量能够表达相应对象的某些特征，同时向量之间的距离反映了对象之间的相似性。

在推荐系统中的应用：

①直接应用：Embedding层负责将高维稀疏特征向量转换为稠密低维特征向量。利用物品 Embedding 间的相似性实现相似物品推荐。

②另外往往会与其他推荐系统特征连接后一同输入后续深度学习网络进行训练。指的是在我们预先训练好物品和用户的 Embedding 之后，不直接应用，而是把这些 Embedding 向量作为特征向量的一部分，跟其余的特征向量拼接起来，作为推荐模型的输入参与训练。

②embedding对物品、用户相似度的计算是常用的推荐系统召回层技术。在局部敏感哈希等快速最近邻搜索技术应用于推荐系统后，embedding 更适合对海量备选物品快速初筛。

10.2 Word2vec技术——经典embedding方法

见自然语言处理的word2vec专题

10.3 Item2Vec：Word2vec 方法的推广

既然 Word2vec 可以对词“序列”中的词进行 Embedding，那么对于用户购买“序列”中的一个商品，用户观看“序列”中的一个电影，也应该存在相应的 Embedding 方法。

微软于 2015 年提出了 Item2Vec 方法，它是对 Word2vec 方法的推广，使 Embedding 方法适用于几乎所有的序列数据。Item2Vec 模型的技术细节几乎和 Word2vec 完全一致，只要能够用序列数据的形式把我们要表达的对象表示出来，再把序列数据“喂”给 Word2vec 模型，我们就能够得到任意物品的 Embedding 了

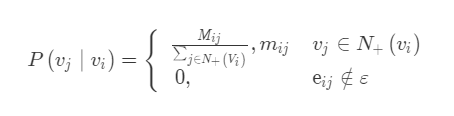
10.4 Graph Embedding技术

但是，互联网的数据可不仅仅是序列数据那么简单，越来越多的数据被我们以图的形式展现出来。这个时候，基于序列数据的 Embedding 方法就显得“不够用”了。

①基于随机游走的 Graph Embedding 方法，Deep Walk

它的主要思想是在由物品组成的图结构上进行随机游走，产生大量物品序列，然后将这些物品序列作为训练样本输入 Word2vec 进行训练，最终得到物品的 Embedding。因此，DeepWalk 可以被看作连接序列 Embedding 和 Graph Embedding 的一种过渡方法。

在上述 DeepWalk 的算法流程中，唯一需要形式化定义的就是随机游走的跳转概率，也就是到达节点 vi后，下一步遍历 vi 的邻接点 vj 的概率。如果物品关系图是有向有权图，那么从节点 vi 跳转到节点 vj 的概率定义如下：



其中，N+(vi) 是节点 vi所有的出边集合，Mij是节点 vi到节点 vj边的权重，即 DeepWalk 的跳转概率就是跳转边的权重占所有相关出边权重之和的比例。如果物品相关图是无向无权重图，那么跳转概率将是上面这个公式的一个特例，即权重 Mij将为常数 1，且 N+(vi) 应是节点 vi所有“边”的集合，而不是所有“出边”的集合。

②在同质性和结构性间权衡的方法，Node2vec

Node2vec 通过调整随机游走跳转概率的方法，让 Graph Embedding 的结果在网络的同质性（Homophily）和结构性（Structural Equivalence）中进行权衡，可以进一步把不同的 Embedding 输入推荐模型，让推荐系统学习到不同的网络结构特。

详细见：

<https://time.geekbang.org/column/article/296672?code=egRnBwnBkdTKZHG8xAk6IS5Ox3x-yu812NEgUpSwhok%3D>

10.5 Graph Embedding和GNN（图神经网络）的区别

1. 逻辑回归模型族

11.1 基本介绍

将推荐问题看成一个分类问题，通过预测正样本的概率对物品进行排序，转换成一个点击率预估（CTR）问题。

基本假设是满足伯努利分布（0、1分布）。

11.2 训练方法

梯度下降法、牛顿法、拟牛顿法。

①应用梯度下降法可以找到一个函数的局部最小值；具体过程：先应用极大似然估计得到目标函数（对数损失函数）；取对数并乘-1/m，转换为求极小值；

②牛顿法和拟牛顿法见数学模块

11.3 优缺点

优点：可解释下强（权值较大的特征更重要）；模型简单、训练开销小。

缺点：表达能力不强、无法进行特征交叉等操作。

四、自然语言处理

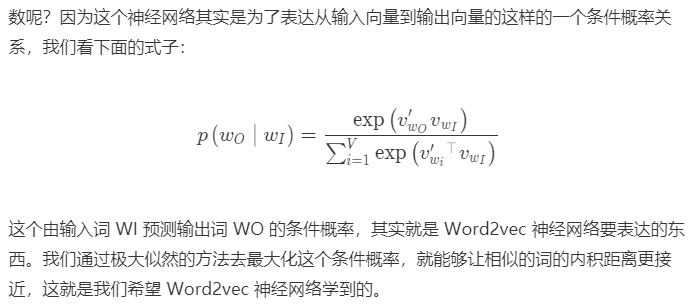
1. word2vec合集

<https://time.geekbang.org/column/article/295939?code=egRnBwnBkdTKZHG8xAk6IS5Ox3x-yu812NEgUpSwhok%3D>

1.1介绍word2vec、训练得到的word2vec的本质、介绍word2vec的原理

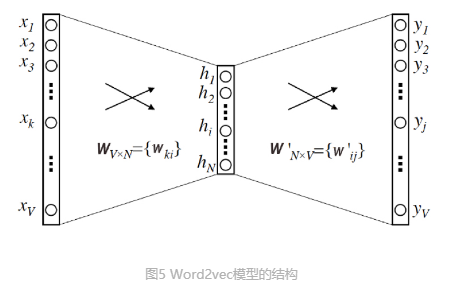
Word2vec 就是词嵌入（embedding) 的一种 ，是“word to vector”的简称，顾名思义，它是一个生成对“词”的向量表达的模型。

想要训练 Word2vec 模型，我们需要准备由一组句子组成的语料库。假设其中一个长度为 T 的句子包含的词有 w1,w2……wt，并且我们假定每个词都跟其相邻词的关系最密切。



我们从语料库中抽取一个句子，选取一个长度为 2c+1（目标词前后各选 c 个词）的滑动窗口，将滑动窗口由左至右滑动，每移动一次，窗口中的词组就形成了一个训练样本。根据 Skip-gram 模型的理念，中心词决定了它的相邻词，我们就可以根据这个训练样本定义出 Word2vec 模型的输入和输出，输入是样本的中心词，输出是所有的相邻词。

有了训练样本之后，我们最关心的当然是 Word2vec 这个模型的结构是什么样的。

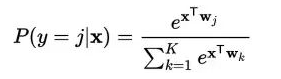


它的输入层和输出层的维度都是 V，这个 V 其实就是语料库（即已知的所有词向量）词典的大小。假设语料库一共使用了 10000 个词，那么 V 就等于 10000。根据图 4 生成的训练样本，这里的输入向量自然就是由输入词转换而来的 One-hot 编码向量，输出向量则是由多个输出词转换而来的 Multi-hot 编码向量，显然，**基于 Skip-gram 框架的 Word2vec 模型解决的是一个多分类问题**。

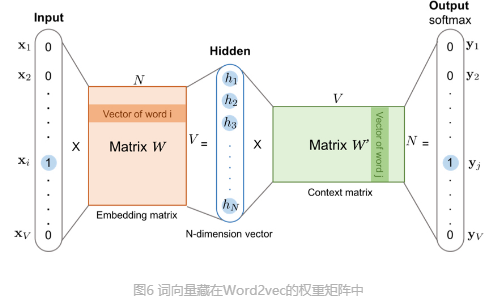
隐层的维度是 N，N 的选择就需要一定的调参能力了，我们需要对模型的效果和模型的复杂度进行权衡，来决定最后 N 的取值，并且最终每个词的 Embedding 向量维度也由 N 来决定。

最后是激活函数的问题，这里我们需要注意的是，**隐层神经元是没有激活函数的，或者说采用了输入即输出的恒等函数作为激活函数**，而输出层神经元采用了 softmax 作为激活函数（作用是将输入）。

Softmax函数：

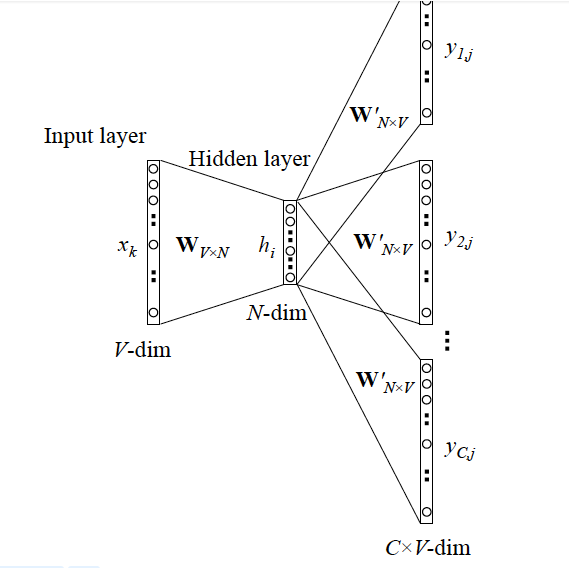


在训练完 Word2vec 的神经网络之后，可能你还会有疑问，我们不是想得到每个词对应的 Embedding 向量嘛，这个 Embedding 在哪呢？其实，**它就藏在输入层到隐层的权重矩阵 WVxN 中。**

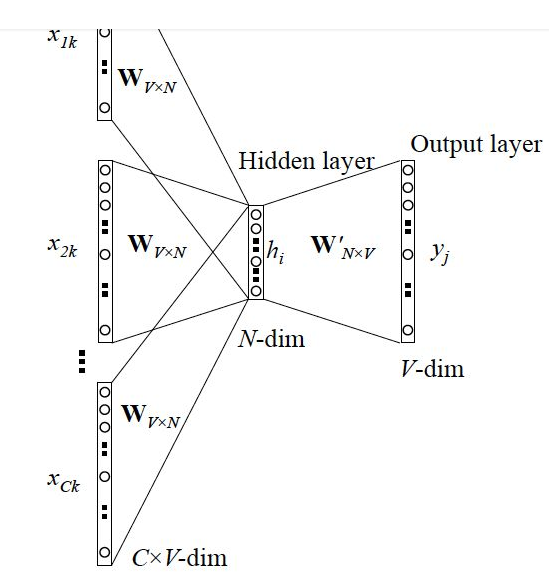


Input是。输入向量矩阵 WVxN 的每一个行向量对应的就是我们要找的“词向量”。比如我们要找词典里第 i 个词对应的 Embedding，因为输入向量是采用 One-hot 编码的，所以输入向量的第 i 维就应该是 1，**那么输入向量矩阵 WVxN 中第 i 行的行向量自然就是该词的 Embedding** 。

上面讨论的是最简单情形，即 y 只有一个词，当 y 有多个词时



CBOW模型：Skip-gram 是预测一个词的上下文，而 CBOW 是用上下文预测这个词。



1.2 word2vec负采样

当你阅读word2vec中的skip-gram模型导学的时候，你会发现那个神经网络实在是太巨大了。

在我给的这个例子下面，每个词向量由300个元素组成（即embedding的维度），并且一个单词表中包含了10000个单词。回想神经网络中有两个权重矩阵——一个在隐藏层，一个在输出层。这两层都具有300 x 10000 = 3,000,000个权重！

使用梯度下降法在这种巨大的神经网络下面进行训练是很慢的。并且可能更糟糕的是，你需要大量的训练数据来调整这些权重来避免过拟合。

**降低计算量三个方法**：

①对于常见的单词对或者短语，在模型中将他们视为单个的单词。

②对常见单词进行二次采样来减少他们在训练样本中的数量。

**我们将会有多样本（“the”，....），远超过我们需要训练“the”的样本数量。**

**Word2Vec通过“二次采样”方案来解决上述问题。对于出现在训练文中的每个单词，都会有一个从文本删除的概率，这个概率取决于相应单词的词频。**

③使用所谓的“负采样”（negative sampling）来改进优化对象，这将造成每一个训练的样本只会更对模型权重的很小一个比例的更新。

训练一个神经网络意味着使用一个训练样本就要稍微调整一下所有的神经网络权重，这样才能够确保预测训练样本更加精确。换句话说，每个训练样本都会改变神经网络中的权重。

单词表的大小意味着我们的skip-gram神经网络拥有非常庞大的权重数，所有权重都会被十亿个样本中的一个稍微地进行更新。

负采样通过使每一个训练样本仅仅改变一小部分的权重而不是所有权重，从而解决这个问题。下面介绍它是如何进行工作的。

当通过（”fox”, “quick”)词对来训练神经网络时，我们回想起这个神经网络的“标签”或者是“正确的输出”是一个one-hot向量。也就是说，对于神经网络中对应于”quick”这个单词的神经元对应为1，而其他上千个的输出神经元则对应为0。

**使用负采样，我们通过随机选择一个较少数目（比如说5个）的“负”样本来更新对应的权重。**(在这个条件下，“负”单词就是我们希望神经网络输出为0的神经元对应的单词）。并且我们仍然为我们的“正”单词更新对应的权重（也就是当前样本下”quick”对应的神经元）

论文说选择5~20个单词对于较小的样本比较合适，而对于大样本，我们可以悬着2~5个单词。

回想一下，我们模型的输出层有大约300 x 10,000维度的权重矩阵。所以我们只需要更新正确的输出单词”quick”的权重，加上额外的5个其他应该输出为0的单词的权重。也就是总共6个输出神经元，和总共1800个的权重值。这些总共仅仅是输出层中3百万个权重中的0.06%。

从本质上来说，选择一个单词来作为负样本的概率取决于它出现频率，对于更经常出现的单词，我们将更倾向于选择它为负样本。

1.3 word2vector损失函数

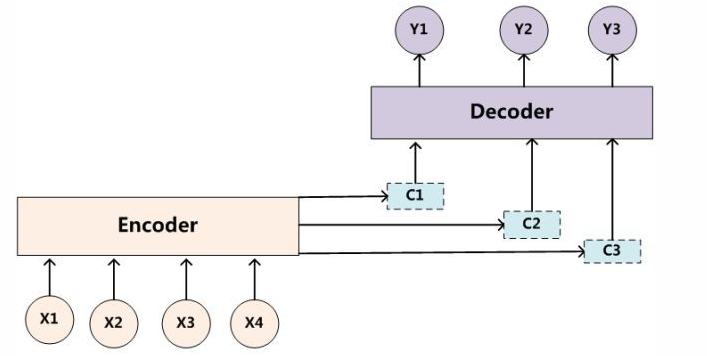
多元的交叉熵损失函数，本质还是预测一个长序列的标签。（softmax和sigmoid是激活函数，不是损失函数）

1. attention机制

<https://www.cnblogs.com/ydcode/p/11038064.html>

<https://zhuanlan.zhihu.com/p/35571412>

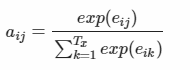
带有 Attention 机制的 Encoder-Decoder 模型则是要从序列中学习到每一个元素的重要程度，然后按重要程度将元素合并。因此，注意力机制可以看作是 Encoder 和 Decoder 之间的接口，它向 Decoder 提供来自每个 Encoder 隐藏状态的信息。通过该设置，模型能够选择性地关注输入序列的有用部分，从而学习它们之间的“对齐”。这就表明，在 Encoder 将输入的序列元素进行编码时，得到的不在是一个固定的语义编码 C ，而是存在多个语义编码，且不同的语义编码由不同的序列元素以不同的权重参数组合而成。一个简单地体现 Attention 机制运行的示意图如下：



在 Attention 机制下，语义编码 C 就不在是输入序列 X 的直接编码了，而是各个元素按其重要程度加权求和得到的，即：

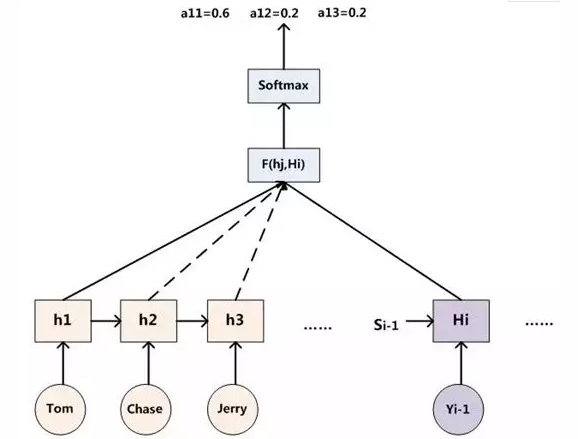


在公式（6）中，参数 i 表示时刻， j 表示序列中的第 j 个元素， Tx 表示序列的长度， f(⋅) 表示对元素 xj 的编码。aij 可以看作是一个概率，反映了元素 hj 对 Ci 的重要性，可以使用 softmax 来表示：

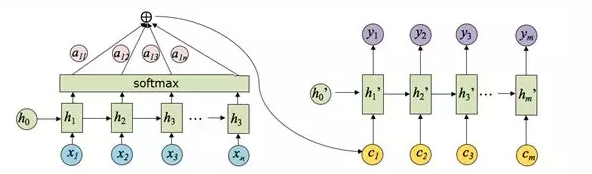


这里 eij 正是反映了待编码的元素和其它元素之间的匹配度，当匹配度越高时，说明该元素对其的影响越大，则 aij 的值也就越大。

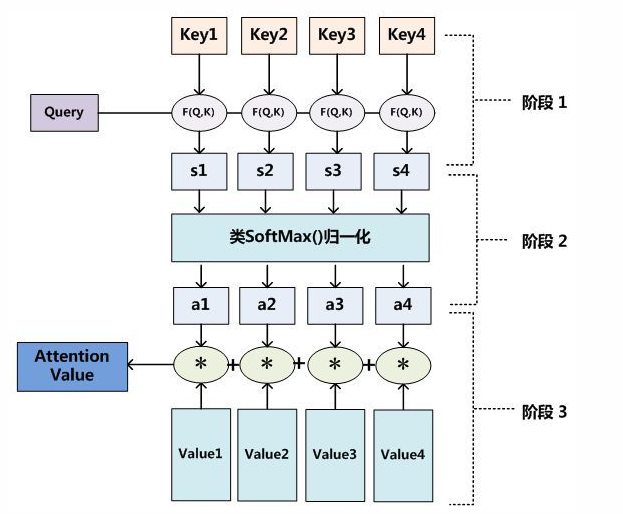
因此，得出 aij 的过程如下图：



其中，hi 表示 Encoder 的转换函数，F(hj,Hi) 表示预测与目标的匹配打分函数。将以上过程串联起来，则注意力模型的结构如下图所示：



Attention 机制的一个重点就是获得 attention value，即机器翻译中的语义编码 Ci。在上一节中我们知道该值是通过输入元素按照不同的权重参数组合而成的，所以我们可以将其定义为一个 attention 函数，比较主流的 attention 函数的机制是采用键值对查询的方式，其工作实质如下图所示：



attention函数共有三步完成得到attention value。

- Q与K进行相似度计算得到权值

- 对上部权值归一化

- 用归一化的权值与V加权求和。

自然语言任务中，往往 Key 和 Value 是相同的。需要注意的是，计算出来的 attention value 是一个向量，代表序列元素 xj 的编码向量，包含了元素 xj 的上下文关系，即同时包含全局联系和局部联系。全局联系很好理解，因为在计算时考虑了该元素与其他所有元素的相似度计算；而局部联系则是因为在对元素 xj 进行编码时，重点考虑与其相似度较高的局部元素，尤其是其本身。

上面从attention函数得到了attention机制工作过程。现在换一个角度来理解，我们将attention机制看做软寻址。就是说序列中每一个元素都由key(地址)和value(元素)数据对存储在存储器里，当有query=key的查询时，需要取出元素的value值(也即query查询的attention值)，与传统的寻址不一样，它不是按照地址取出值的，它是通过计算key与query的相似度来完成寻址。这就是所谓的软寻址，它可能会把所有地址(key)的值(value)取出来，上步计算出的相似度决定了取出来值的重要程度，然后按重要程度合并value值得到attention值，此处的合并指的是加权求和。

1. Transformer合集

3.1 Transformer在训练过程中有哪些可以 调整的超参

3.2 Multi-head attention的原理

3.3 seq2seq跟transformer相比有什么不同吗？

（seq2seq有长距离依赖问题，那transformer encoder方面引入了self-attention机制，可以更好的关注上下文，seq2seq相当于只能关注前面的信息，而transormer相当于可以关注上下文，相当于双向语言模型；为了解决transformer位置编码的问题，transformer引入了位置编码）

3.4 transformer的注意力机制你知道怎么实现的吗

（Q,K,V,Q作用域K，然后sigmoid取均值，输出加权平均和，面试官纠正你刚刚说sigmoid?我说说错了，应该是softmax。。。）

3.5 你知道Q,K,V有什么意义吗，怎么得到的

（分别乘以3个不同的权重矩阵）

3.6 那Multi-attention怎么得到的

（那就用多个不同的Q,K,V映射矩阵，然后映射到不同的Q,K,V value），面试官多头注意力机制是为了什么（我说是为了从不同的角度关注当前词和上下文的关系，避免信息挖掘的不全，为了更好的关注不同的语义信息）

3.7 你刚刚说了除以根号d，你知道为什么要除以根号d吗

（我..可能是为了归一化把？？？？后面想到可能是为了降低数值，面试官后来问有没有把 attention is all you need 捡起来从头到晚读一遍...）

3.8 Transformer中的Scaled Dot-Product Attention为什么要缩放（两点）

3.9 Transformer中的Position Embedding是怎么实现的？为什么？

1. BERT合集

五、实战

（1）如何处理样本不均衡问题

（说了过采样，欠采样，问还有没有其他的，没答上来）

<https://www.cnblogs.com/lyr2015/p/8711120.html>

1.1样本的过采样和欠采样

过采样：将稀有类别的样本进行**复制**，通过增加此稀有类样本的数量来平衡数据集。该方法适用于数据量较小的情况。

使用SMOTE方法来构造样本：SMOTE算法是一种过采样的算法。这个算法不是简单的复制已有的数据，而是在原有数据基础上，通过算法产生新生数据。

算法思想：

1、基于距离度量的方式计算两个或多个稀有类样本之间的相似性；

2、然后选择其中的一个样本作为基础样本

3、再在邻居样本中随机选取一定数量的样本对那个基础样本的一个属性进行噪声。每次处理一个属性，通过这样的方式产生新生数据。

欠抽样：从丰富类别的样本中随机选取和稀有类别相同数目的样本，通过减少丰富类的样本量啦平衡数据集。该方法适用于数据量较大的情况。

过采样：可能会存在过拟合问题。（可以使用SMOTE算法，增加随机的噪声的方式来改善这个问题）

欠采样：可能会存在信息减少的问题。因为只是利用了一部分数据，所以模型只是学习到了一部分模型。

1.2 使用多个分类器进行分类

通过训练多个模型的方式解决数据不均衡的问题，是指将多数类数据随机分成少数类数据的量N份，每一份与全部的少数类数据一起训练成为一个分类器，这样反复训练会生成很多的分类器。

最后再用组合的方式(bagging或者boosting)对分类器进行组合，得到更好的预测效果。简单来说若是分类问题可采用投票法，预测问题可以采用平均值。这个解决方式需要很强的计算能力以及时间，但效果较好，相当于结合了组合分类器的优势。

1.3 将二分类问题转换成其他问题

可以将不平衡的二分类问题转换成异常点检测，或者一分类问题（可使用one-class svm建模）

1.4 改变正负类别样本在模型中的权重

使用代价函数学习得到每个类的权值，大类的权值小，小类的权值大。刚开始，可以设置每个类别的权值与样本个数比例的倒数，然后可以使用过采样进行调优（**在权重初始化时这样做**）。

例如scikit-learn中的SVM算法，也称作penalized-SVM，可以手动设置权重。若选择balanced，则算法会设定样本权重与其对应的样本数量成反比。

（2）给一个训练样本，其中有一个离散特征，取值有100W维，怎么解决

1.1根据类别特征的意义进行合并（分桶）

1.2 将类别按频次排序，频次特别低的一部分合并

1.3 Embedding：而深度学习的特点以及工程方面的原因使其不利于稀疏特征向量的处理。因此如果能把物体编码为一个低维稠密向量再喂给DNN（深度神经网络），自然是一个高效的基本操作。

（3）训练时发现模型不收敛的原因

学习率过大或者过小、梯度消失、数据量不够等等

（4）深度学习相比于机器学习有什么优点

1.数据依赖性

传统机器学习和深度学习之间最重要的区别在于数据规模升级时的性能。当数据很小时，深度学习算法不能很好地运行，因为它们需要大数据来完美地识别和理解它。然而，机器学习算法能在这种情况下工作。

2.硬件依赖性

深度学习算法在很大程度上依赖于高端机器，因为深度学习包括GPU，它是其工作中不可或缺的一部分。由于深度学习通过大量矩阵乘法进行遗传操作，因此通过使用专为此目的而构建的GPU，可以高效地优化这些操作。与此相比，传统的机器学习算法可以在低端机器上运行。

3.特征工程

特征工程是指在创建特征提取器时放置领域知识的过程，以便降低数据复杂性并使模式对学习算法可见，以便它们可以工作。整个过程非常昂贵且困难，需要大量的时间和专业知识。在传统的机器学习中，所有应用的特征都由专家识别，后者根据数据类型和域对其进行手工编码。例如，特征可以是形状，像素值，纹理，方向和位置。机器学习算法的性能取决于识别和提取的特征的准确性。另一方面，深度学习算法从数据中识别这些高级特征，因此减少了为每个问题开发全新特征提取器的工作量。

4.逻辑的解释性

解释性也是深度学习在应用于行业之前必须考虑的因素之一。例如，如果我们使用深度学习为任何论文提供自动评分。虽然它的表现非常出色，但不会透露给出该分数的原因。您可以随时在数学上找出在评分时激活的深层神经网络的节点，但您永远不会知道这些神经元的模型是什么以及它们共同做了什么。如此深入的学习使我们无法解释结果。然而，机器学习算法为我们提供了一套清晰的规则，根据这些规则选择了分数。因此，解释它背后的逻辑变得容易

六、场景题

<https://blog.csdn.net/u010601183/article/details/56481868>

1. 类似基数[排序]，先是问所有[排序]中时间复杂度最坏情况最好的是什么，然后结合场景给出最快的方案（开大数组），面试官问了如何优化内存（位计数），然后给我介绍了公司如何继续优化的方向

该方法与排序方法类似，用一个容器保存前10000个数，然后将剩余的所有数字——与容器内的最小数字相比，如果所有后续的元素都比容器内的10000个数还小，那么容器内这个10000个数就是最大10000个数。如果某一后续元素比容器内最小数字大，则删掉容器内最小元素，并将该元素插入容器，最后遍历完这1亿个数，得到的结果容器中保存的数即为最终结果了。

1. 场景题：给一个文件，里面存储着一个类别ID以及该类别的父类别ID，要求写函数处理文件，并能够根据查询的类别ID输出其所有子类别的ID。

题主的理解：是把文件构造成多叉树，根据输入的节点输出其所有子节点。或者建个图，然后深度优先搜索。

1. 场景题，百度有海量的搜索词记录，返回TopK个高频词

<https://blog.csdn.net/qq1404510094/article/details/80323168>

分治+Trie树/hash+小顶堆（就是上面提到的最小堆），即先将数据集按照Hash方法分解成多个小数据集（**hash后取模则能得到多个小数据集**），然后使用Trie树或者Hash统计每个小数据集中的query词频，之后用小顶堆求出每个数据集中出现频率最高的前K个数，最后在所有top K中求出最终的top K。

建堆时间复杂度是O（mlogm），算法的时间复杂度为O（nmlogm）

（我的回答是经典的分治 + 归并，但是最优解是字典树）

1. 蓄水池算法合集

<https://www.jianshu.com/p/7a9ea6ece2af>

4.1 给定一个数据流，数据流长度N很大，且N直到处理完所有数据之前都不可知，请问如何在只遍历一遍数据（O(N)）的情况下，能够随机选取出m个不重复的数据。

这个场景强调了3件事：

①数据流长度N很大且不可知，所以不能一次性存入内存。

②时间复杂度为O(N)。

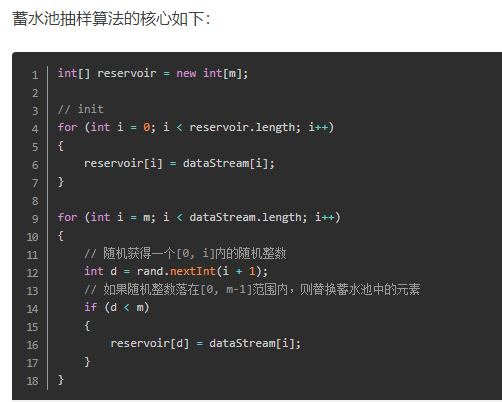
③随机选取m个数，每个数被选中的概率为m/N。

第1点限制了不能直接取N内的m个随机数，然后按索引取出数据。

第2点限制了不能先遍历一遍，然后分块存储数据，再随机选取。

第3点是数据选取绝对随机的保证。讲真，在不知道蓄水池算法前，我想破脑袋也不知道该题做何解。

类似问题：怎么用最短时间最少空间去随机抽取一个未知长度的链表



注：这里使用已知长度的数组dataStream来表示未知长度的数据流，并假设数据流长度大于蓄水池容量m。

算法思路大致如下：

如果接收的数据量小于m，则依次放入蓄水池。

当接收到第i个数据时，i >= m，在[0, i]范围内取以随机数d，若d的落在[0, m-1]范围内，则用接收到的第i个数据替换蓄水池中的第d个数据。

重复步骤2。

算法的精妙之处在于：当处理完所有的数据时，蓄水池中的每个数据都是以m/N的概率获得的。

下面用白话文推导验证该算法。假设数据开始编号为1.

**第i个接收到的数据最后能够留在蓄水池中的概率=第i个数据进入过蓄水池的概率\*之后第i个数据不被替换的概率（第i+1到第N次处理数据都不会被替换）。**

当i<=m时，数据直接放进蓄水池，所以第i个数据进入过蓄水池的概率=1。

**(i<=m时，进入概率为1，不被换出概率为m/N,故相乘是m/N)**当i<=m时，程序从接收到第m+1个数据时开始执行替换操作，第m+1次处理会替换池中数据的为m/(m+1)（若取m之外的数就不替换了），会替换掉第i个数据的概率为1/m，则第m+1次处理替换掉第i个数据的概率为(m/(m+1))\*(1/m)=1/(m+1)，不被替换的概率为1-1/(m+1)=m/(m+1)。依次，第m+2次处理不替换掉第i个数据概率为(m+1)/(m+2)...第N次处理不替换掉第i个数据的概率为(N-1)/N。所以，之后第i个数据不被替换的概率=m/(m+1)\*(m+1)/(m+2)\*...\*(N-1)/N=m/N。

当i>m时，在[1,i]内选取随机数d，如果d<=m，则使用第i个数据替换蓄水池中第d个数据，因此第i个数据进入过蓄水池的概率=m/i。**(根据上段同理往下推即可）**

（5）处理上亿条数据时，内存不够load文件怎么办？

综合所有的方法来看，主要有hash成小文件、位图等等。

（6）海量日志数据，提取出某日访问百度次数最多的那个IP。（在无法全部存入内存时统计大量数据次数的方法）

再详细介绍下此方案：首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有个2^32个 IP。同样可以采用映射的方法，比如模1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP（可以采用hash\_map进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

（7）假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），请你统计最热门的10个查询串，要求使用的内存不能超过1G。

（哈希加TOPK，哈希的思想和mapreduce差不多）

①典型的Top K算法，还是在这篇文章里头有所阐述。 文中，给出的最终算法是：第一步、先对这批海量数据预处理，在O（N）的时间内用Hash表完成排序；然后，第二步、借助堆这个数据结构，找出Top K，时间复杂度为N‘logK。 即，借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比所以，我们最终的时间复杂度是：O（N） + N'\*O（logK），（N为1000万，N’为300万）

②采用trie树，关键字域存该查询串出现的次数，没有出现为0。最后用10个元素的最小推来对出现频率进行排序。

**Query即用户在搜索引擎输入查询条件。在通用搜索引擎中，一般是指输入的关键词。而在各类行业或者垂直搜索引擎，还可以输入类目，如优酷网站中可以选择“电影”、“电视剧”这样的类目。在电子商务网站中，各种产品品牌、型号、款式、价格等也是常见的查询条件**

（8）有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。

（**不足以存进去时，则使用先hash转换为整数，再用取余操作转换为小文件，再对小文件进行top排序**）

　　方案：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件（记为x0,x1,...x4999）中。这样每个文件大概是200k左右。

如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。 对每个小文件，统计每个文件（一定每个文件都要能统计，这一步也是与第二题中全部能统计的区别）中出现的词以及相应的频率（可以采用trie树/hash\_map等），并取出出现频率最大的100个词（可以用含100个结点的最小堆），并把100个词及相应的频率存入文件，这样又得到了5000个文件。下一步就是把这5000个文件进行归并（类似与归并排序）的过程了。

（9）给定a、b两个文件，各存放50亿个url，每个url各占64字节，内存限制是4G，让你找出a、b文件共同的url

方案1：

可以估计每个文件安的大小为5G×64=320G，远远大于内存限制的4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

　　遍历文件a，对每个url求取hash(url)%1000，然后根据所取得的值将url分别存储到1000个小文件（记为a0,a1,...,a999）中。这样每个小文件的大约为300M。

　　遍历文件b，采取和a相同的方式将url分别存储到1000小文件（记为b0,b1,...,b999）。这样处理后，所有可能相同的url都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）中，不对应的小文件不可能有相同的url。然后我们只要求出1000对小文件中相同的url即可。

　　求每对小文件中相同的url时，可以把其中一个小文件的url存储到hash\_set中。然后遍历另一个小文件的每个url，看其是否在刚才构建的hash\_set中，如果是，那么就是共同的url，存到文件里面就可以了。

方案2：

<https://zhuanlan.zhihu.com/p/85083632>

如果允许有一定的错误率，可以使用Bloom filter，4G内存大概可以表示340亿bit（**前面处理小文件的思路还是一样，只是针对查找集合的部分**）。将其中一个文件中的url使用Bloom filter映射为这340亿bit，然后挨个读取另外一个文件的url，检查是否与Bloom filter，如果是，那么该url应该是共同的url（注意会有一定的错误率）。

Bloom-Filter，即布隆过滤器，是一种多哈希函数映射的快速查找算法。通常应用在一些需要快速检测一个元素是否在一个集合中，但是并不严格要求100%正确的场合。Bloom Filter的空间利用效率很高，使用位数组表示一个待检测集合，使用它可以大大节省存储空间。利用这个算法我们可以实现去重效果。

Bloom Filter有可能会出现错误判断，但不会漏掉判断。也就是Bloom Filter判断元素不在集合，那肯定不在。如果判断元素存在集合中，有一定的概率判断错误（**不在的一定不在，在的不一定在**）

**在判断y是否属于这个集合时，我们对y应用k次哈希函数，如果所有hi(y)的位置都是1（1≤i≤k），那么我们就认为y是集合中的元素，否则就认为y不是集合中的元素。下图中y1就不是集合中的元素。y2或者属于这个集合，或者刚好是一个false positive。**

（10）在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数？

方案1：

**（如果位图也不足以存储，则还是需要用“hash（x）%k”划分为k个小文件的方法）**

采用2-Bitmap（每个数分配2bit，00表示不存在，01表示出现一次，10表示多次，11无意义）进行，共需内存内存，还可以接受**（因为是找整数，整数本身就是key，而键就是该元素）**。然后扫描这2.5亿个整数，查看Bitmap中相对应位，如果是00变01，01变10，10保持不变。所描完事后，查看bitmap，把对应位是01的整数输出即可。

方案2：

也可采用与第1题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

（11）怎么在海量数据中找出重复次数最多的一个？

　　 方案1：先做hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）

七、C++

（1）C++结构体初始化时，什么时候不写构造函数会报错

（2）C++的vector底层实现

它就是使用 3 个迭代器（可以理解成指针）来表示的。

\_Myfirst 指向的是 vector 容器对象的起始字节位置；\_Mylast 指向当前最后一个元素的末尾字节；\_myend 指向整个 vector 容器所占用内存空间的末尾字节。

在此基础上，将 3 个迭代器两两结合，还可以表达不同的含义，例如：

\_Myfirst 和 \_Mylast 可以用来表示 vector 容器中目前已被使用的内存空间；

\_Mylast 和 \_Myend 可以用来表示 vector 容器目前空闲的内存空间；

\_Myfirst 和 \_Myend 可以用表示 vector 容器的容量。

vector扩大容量的本质：另外需要指明的是，当 vector 的大小和容量相等（size==capacity）也就是满载时，如果再向其添加元素，那么 vector 就需要扩容。

vector 容器扩容的过程需要经历以下 3 步：

1完全弃用现有的内存空间，重新申请更大的内存空间；

2将旧内存空间中的数据，按原有顺序移动到新的内存空间中；

3最后将旧的内存空间释放。

这也就解释了，为什么 vector 容器在进行扩容后，与其相关的指针、引用以及迭代器可能会失效的原因。

由此可见，vector 扩容是非常耗时的。为了降低再次分配内存空间时的成本，每次扩容时 vector 都会申请比用户需求量更多的内存空间（这也就是 vector 容量的由来，即 capacity>=size），以便后期使用。

vector 容器扩容时，不同的编译器申请更多内存空间的量是不同的。以 VS 为例，它会扩容现有容器容量的 50%。

（3）C++的sort底层实现

毫无疑问是用到了快速排序，但不仅仅只用了快速排序，还结合了插入排序和堆排序。

序列式容器中的stack、queue和priority-queue都有特定的出入口，不允许用户对元素排序。

剩下的vector、deque，适用sort算法。

STL的sort算法，数据量大时采用QuickSort快排算法，分段。一旦分段后的数据量小于某个门槛（16），为避免QuickSort快排的递归调用带来过大的额外负荷，就改用Insertion Sort插入排序。如果递归层次过深，还会改用HeapSort堆排序。

（4）C++指针与引用的区别

4.1指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。

4.2引用不可以为空，当被创建的时候，必须初始化，而指针可以是空值，可以在任何时候被初始化。

4.3 可以有const指针，但是没有const引用；

4.4 如果返回动态内存分配的对象或者内存，必须使用指针，引用可能引起内存泄漏

（5）面向对象的三大要素

封装、继承、多态

封装最好理解了。封装是面向对象的特征之一，是对象和类概念的主要特性。

封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

其余两个见上面

（6）多态

<https://www.cnblogs.com/alinh/p/9636352.html>

**多态两种方式：覆盖，重载。**

**C++的多态性用一句话概括就是：在基类的函数前加上virtual关键字，在派生类中重写该函数，运行时将会根据对象的实际类型来调用相应的函数。如果对象类型是派生类，就调用派生类的函数；如果对象类型是基类，就调用基类的函数。**

如果不加virtual：c++编译器在编译的时候，要确定每个对象调用的函数（非虚函数）的地址，这称为早期绑定，当我们将Son类的对象son的地址赋给pFather时，c++编译器进行了类型转换，此时c++编译器认为变量pFather保存的就是Father对象的地址，当在main函数中执行pFather->Say(),调用的当然就是Father对象的Say函数。

如果加virtual：前面输出的结果是因为编译器在编译的时候，就已经确定了对象调用的函数的地址，要解决这个问题就要使用晚绑定，当编译器使用晚绑定时候，就会在运行时再去确定对象的类型以及正确的调用函数，而要让编译器采用晚绑定，就要在基类中声明函数时使用virtual关键字，这样的函数我们就称之为虚函数，一旦某个函数在基类中声明为virtual，那么在所有的派生类中该函数都是virtual，而不需要再显式地声明为virtual。

（7）继承

继承允许我们依据另一个类来定义一个类，这使得创建和维护一个应用程序变得更容易。这样做，也达到了重用代码功能和提高执行效率的效果。

**当创建一个类时，您不需要重新编写新的数据成员和成员函数，只需指定新建的类继承了一个已有的类的成员即可。这个已有的类称为基类，新建的类称为派生类。**

1. 公有继承

当类的继承方式为公有继承时，基类的公有和保护成员的访问属性在派生类中保持不变，而基类的私有成员不可访问。

即基类的公有成员和保护成员被继承到派生类中仍作为派生类的公有和保护成员，派生类的其他成员可以直接访问它们；

其他外部使用者只能通过派生类的对象访问继承来的公有成员；

而无论派生类的成员还是对象都无法访问基类的私有成员。

2. 私有继承

当类的继承方式为私有继承时，基类的公有和保护成员都以私有成员身份出现在派生类中，而基类的私有成员在派生类中不可访问。

即基类的公有成员和保护成员被继承到派生类中作为派生类的私有成员，派生类的其他成员可以直接访问它们；

但是在类外部通过派生类的对象无法访问；

而无论派生类的成员还是对象都无法访问基类的私有成员。

3. 保护继承

当类的继承方式为保护继承时，基类的公有和保护成员都以保护成员身份出现在派生类中，而基类的私有成员在派生类中不可访问。

即基类的公有成员和保护成员被继承到派生类中作为派生类的保护成员，派生类的其他成员可以直接访问它们；

但是在类外部通过派生类的对象无法访问；

而无论派生类的成员还是对象都无法访问基类的私有成员。

无论哪种继承方式，基类的公有和保护成员都可以被派生类的成员访问。

无论哪种继承方式，基类的私有成员都不可以被派生类的成员和对象访问。

只有公有继承时，基类的公有成员才可以被派生类的对象访问。

（8）模板

7.1前置知识

<https://www.cnblogs.com/skynet/archive/2010/09/05/1818636.html>

**重载：函数重载是指在同一作用域内，可以有一组具有相同函数名，不同参数列表的函数，这组函数被称为重载函数。重载函数通常用来命名一组功能相似的函数，这样做减少了函数名的数量，避免了名字空间的污染，对于程序的可读性有很大的好处。**

（9）C语言中结构体struct{int i; bool b}一共占几个字节

如果int类型占4个字节的话，那么这个结构体一共需要8个字节

1. offset(b)在结构体中偏移几个字节

4个字节

1. Union

和struct类似，但是是共享内存

1. Python

（1）Python列表与元祖的区别

列表是动态数组，它们可变且可以重设长度（改变其内部元素的个数）。

元组是静态数组，它们不可变，且其内部数据一旦创建便无法改变。

元组缓存于Python运行时环境，这意味着我们每次使用元组时无须访问内核去分配内存

1. python装饰器

2.1前置知识

2.1.1嵌套函数

如果在一个函数的内部还定义了另一个函数(注意: 是定义，不是引用!），这个函数就叫嵌套函数。外部的我们叫它外函数，内部的我们叫他内函数。

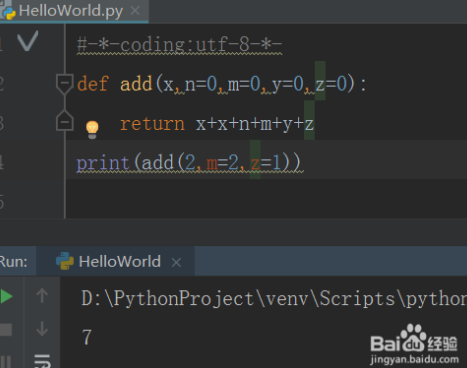
2.1.2 python的参数设置

python函数的普通参数,默认参数,可变参数,关键字参数。

①默认参数：在调用函数时如果不指定某个参数，Python 解释器会抛出异常。为了解决这个问题，Python 允许为参数设置默认值，即在定义函数时，直接给形式参数指定一个默认值。这样的话，即便调用函数时没有给拥有默认值的形参传递参数，该参数可以直接使用定义函数时设置的默认值。

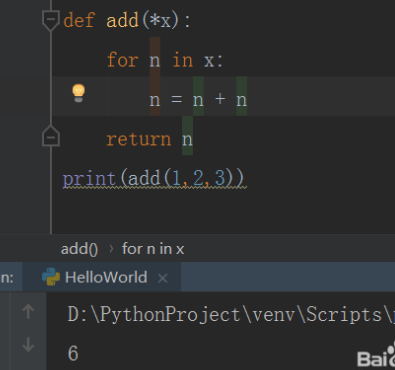


如果我们有很多默认参数，但调用的时候只想传入其中几个，就可以指定参数名称传入值，不用像java一样写一堆重载函数，或者为了调用多参数函数而写上一长串参数。



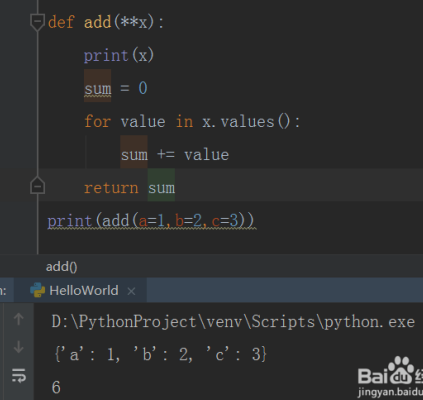
②可变参数

除了默认参数以外，python还提供可变参数，可变参数用\*号表示,传入的元素会被打包成一个tuple（元组）。



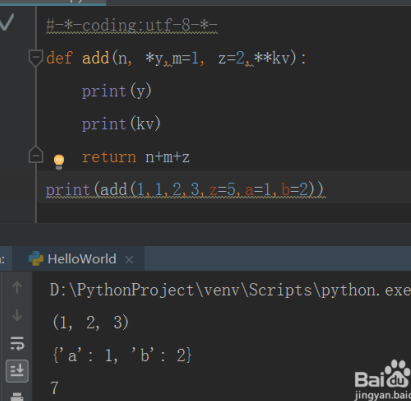
③关键字参数

如果你希望参数被打包成一个dict,这样你就可以传入键值对了,你就需要用到关键字参数，关键字参数用 \*\* 表示。



参数的定义顺序有一定要求:普通参数、可变参数、默认参数、关键字参数。

1会被普通参数吸收,之后的1,2,3会被可变参数打包成tuple,之后的指定型参数会被默认参数吸收掉对应的部分,剩下的会被关键字参数打包为dict。



2.1.3闭包(Closure)

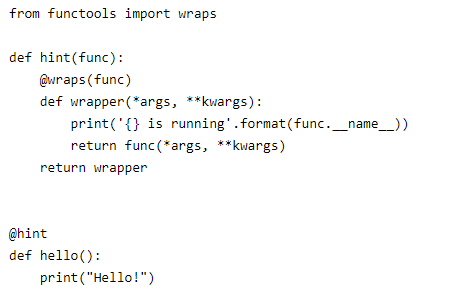
闭包是Python编程一个非常重要的概念。如果一个外函数中定义了一个内函数，且内函数体内引用到了体外的变量，这时外函数通过return返回内函数的引用时，会把定义时涉及到的外部引用变量和内函数打包成一个整体（闭包）返回。

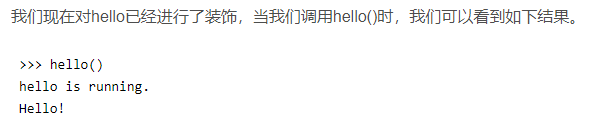
2.2装饰器

<https://blog.csdn.net/weixin_42134789/article/details/84635252?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromBaidu-1.control>

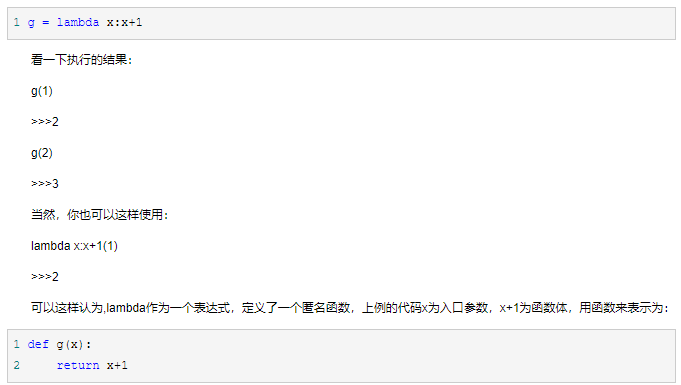
Python的装饰器本质上是一个嵌套函数，它接受被装饰的函数(func)作为参数，并返回一个包装过的函数。这样我们可以在不改变被装饰函数的代码的情况下给被装饰函数或程序添加新的功能。

下列代码先执行hint中的print函数，再执行定义的print hello



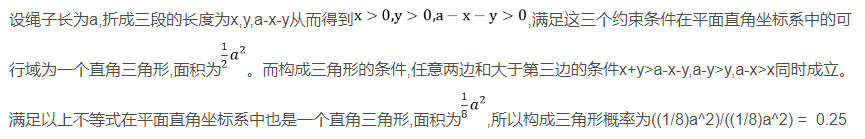


1. 介绍lambda表达式



非常容易理解，在这里lambda简化了函数定义的书写形式。是代码更为简洁，但是使用函数的定义方式更为直观，易理解。

1. 概率
2. 一根绳子分成三段，能围成三角形的概率



1. 概率题：N枚真硬币是一面图案一面字，M枚假硬币是两面图案，选了一枚抛K次都是图案，问是真硬币的概率——贝叶斯



（3）如果明天下午的概率是80%，然后明天每个小时下雨的概率都是相等的， 求明天0点到8点的不下雨的概率（有一个小时下雨就算下雨）

1. 项目
2. 你的特征为什么要这样构造？
3. 时序特征怎么处理的？
4. 为什么这样构造数据集？
5. 为什么召回的数量级小，排序模型的效果就好
6. 为什么这样设置召回数量级
7. 代码
8. 代码题：给个矩阵，0代表可以通行，1代表死路，求一条从左上到右下的路径
9. 代码题：给定一个目标值M的数组，返回数组是否存在和为M子集
10. 代码题：有序数组寻找目标值最后出现的位置
11. 从海量数据中寻找频数前1000的数据
12. 实现函数 int sqrt(int x). 要求复杂度O（lgn）

即用二分法开平方：



1. 快排，面试官要求只能c++，问选择pivot的时候有没有加速的手段
2. 二分查找翻转数组指定元素
3. 链表找环
4. 第k大数
5. 旋转数组中搜索某个目标值
6. 最长回文字串
7. 枚举全排列
8. 最近公共祖先
9. 间隔反转列表，要求时间复杂度为O(n)
10. 把数字转换成中文汉字给定输入”28024“，输出”两万八千零二十四“ ，这个没有什么算法知识，但有一些比较tricky的处理技巧
11. 非递归前序遍历、中序、后序
12. tensorflow实现deepfm
13. 智力题
14. 智力题：25匹马，5个赛道，最多几次可以知道前三名

(1)求前3名要7次；

将马分成A、B、C、D、E五组。

第1-5次比赛：各组分别进行比赛，决出各组名次,取每组前三名

A1、A2、A3，

B1、B2、B3，

C1、C2、C3，

D1、D2、D3，

E1、E2、E3。

第6次比赛：A1、B1、C1、D1、E1

剩下A2、A3、B1、B2、C1是有希望冲进前三的

1. 数据结构
2. 哈希表了解吗？有哪些解决冲突方法
3. 递归有什么缺点

（3）为什么快排比堆排快？

1. 其他
2. 多进程和多线程的区别

<https://blog.csdn.net/weixin_46178557/article/details/104845827>

多线程：同一时刻执行多个线程。如，用浏览器一边下载，一边听歌，一边看视频，一边看网页......

多进程：同时执行多个程序。如，同事运行YY，QQ，以及各种浏览器。

1. 在浏览器输入网址后的一个过程

<https://www.cnblogs.com/jiqianqian/p/6587620.html>

<https://blog.csdn.net/weixin_38892128/article/details/103144135>

1. 浏览器查找该域名的IP地址
2. 浏览器根据解析得到的IP地址向服务器发送一个HTTP请求
3. 服务器可能会发生重定向响应
4. 浏览器跟踪重定向地址
5. 服务器收到请求并进行处理
6. 服务器返回一个HTML响应
7. 浏览器开始显示HTML

另一角度：

1.解析域名的IP地址（在浏览器缓存或hosts或DNS服务器查找）

2.发起连接请求，TCP三次握手

3.发送HTTP请求信息

4.接受服务器返回的数据并渲染到页面

5.断开TCP连接，四次挥手

十五、公式推导

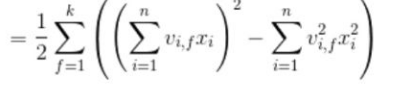
FM

<https://www.cnblogs.com/zhangchaoyang/articles/7897085.html>

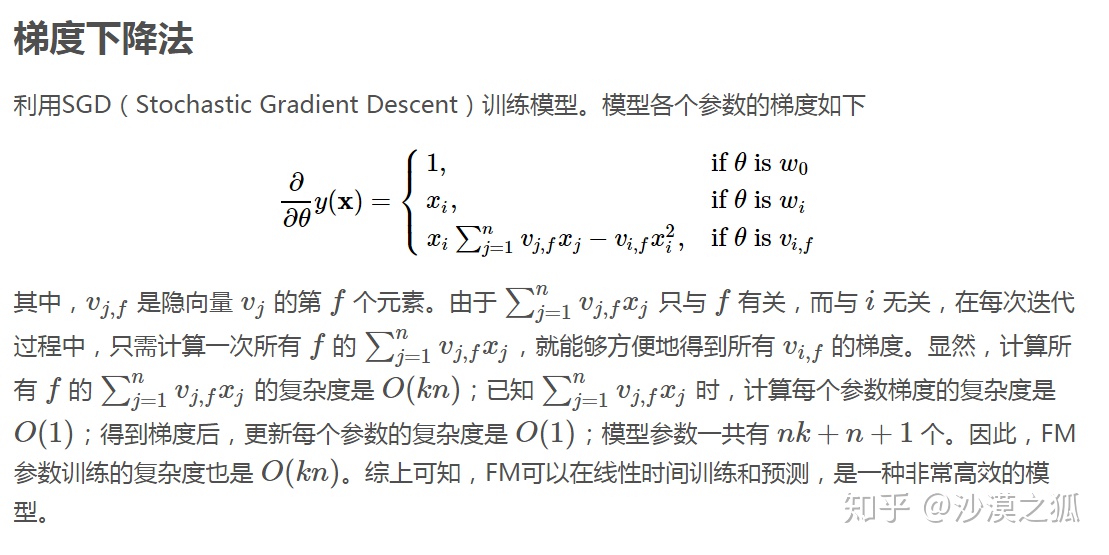
<https://zhuanlan.zhihu.com/p/109980037>

其中，公式等号左边其实 就是矩阵 对角线上方的元素 == （整个矩阵 减去对角线的元素）/2 。 不是求和式是一个矩阵，而是对上三角矩阵求和（不包括对角线），等于对整个矩阵求和 减去对角线元素，根据对称属于相当于加了两次，故还要除以二。

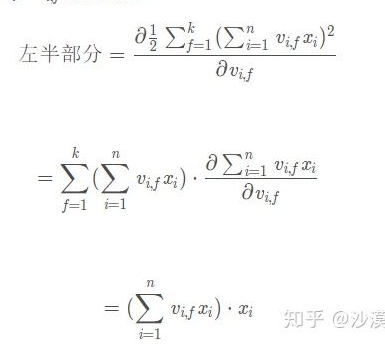
梯度下降法的训练过程:



对上式求导，得出下式，前两种是非交叉特征的偏导，最后一项是交叉特征的导数。



其中相关推导，下式对vif求导，求和符号之所以消失，是因为求和过程中对于非v1f的求导都当成常数处理了；而f到k的求和符号之所以消失也是同理。



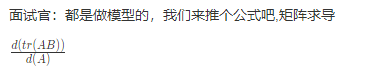
Svm

决策树

LR回归

深度学习推荐系统的31页有逻辑回归梯度下降法求解的推导。

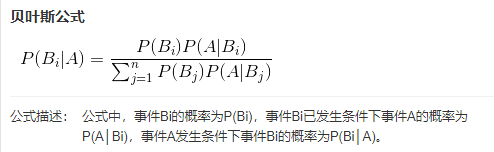
朴素贝叶斯



十六、Spark

（1）spark中数据倾斜如何引起，怎么解决

1. 数学
2. 贝叶斯公式你给我写一下



全概率公式是贝叶斯公式的分母部分。

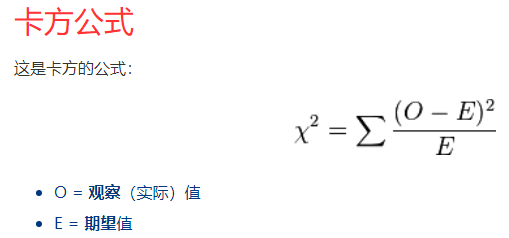
1. 卡方检验知道吗，你知道里面的P值是什么吗

卡方检验是一种用途很广的计数资料的假设检验方法。属于非参数检验，主要是比较两个及两个以上样本率（构成比）以及两个分类变量的关联性分析。根本思想在于比较理论频数和实际频数的吻合程度或者拟合优度问题。

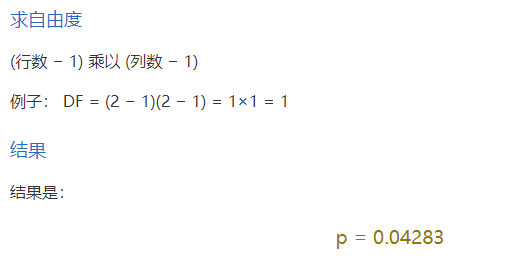
**卡方检验只适用于分类数据（分成不同类别的数据），像性别 {Men, Women} 或颜色 {Red, Yellow, Green, Blue} 等等，而不适用于数值数据，例如身高、体重等等**。

**"p" 是变量是独立的 概率。通常 p < 0.05 代表变量是相依的。在这例子里，p 大于 0.05，所以我们相信变量是独立（没关联）的**。我们怎样计算 p值？用卡方检验！

通常情况下，卡方检验用于对两个特性的独立性检验，其原假设为这里两个特性是独立的，如果卡方检验的p值大于0.5，说明原假设下，我们抽到这样的样本是一个大概率事件，说明原假设是成立的，即被检验的两个特性是独立的。



由卡方到 p，用卡方的值来求 p值是个复杂的算法，但你可以去查表或用 卡方计算器。但先需要求 "自由度" (DF)



1. 凸优化合集

22.1凸优化有哪些方法

①梯度下降法

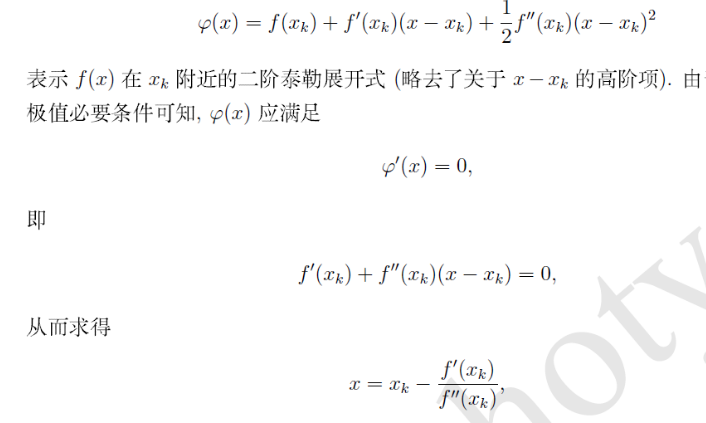
②牛顿迭代法

牛顿法（Newton method）和拟牛顿法（quasi Newton method）是求解无约束最优化问题的常用方法，有收敛速度快的优点。牛顿法是迭代算法，每一步都需求解目标函数的海塞矩阵（Hessian Matrix），计算比较复杂。拟牛顿法通过正定矩阵近似海塞矩阵的逆矩阵或海塞矩阵，简化了这一计算过程。

<https://blog.csdn.net/itplus/article/details/21896453>

<https://zhuanlan.zhihu.com/p/46536960>

当目标函数是二次函数时，海塞矩阵退化成一个常数矩阵，从任一初始点出发，牛顿法可一步到达，因此它是一种具有二次收敛性的算法。对于非二次函数，若函数的二次性态较强，或迭代点已进入极小点的邻域，则其收敛速度也是很快的，这是牛顿法的主要优点。



由上图可知，可用泰勒求导，用二阶导

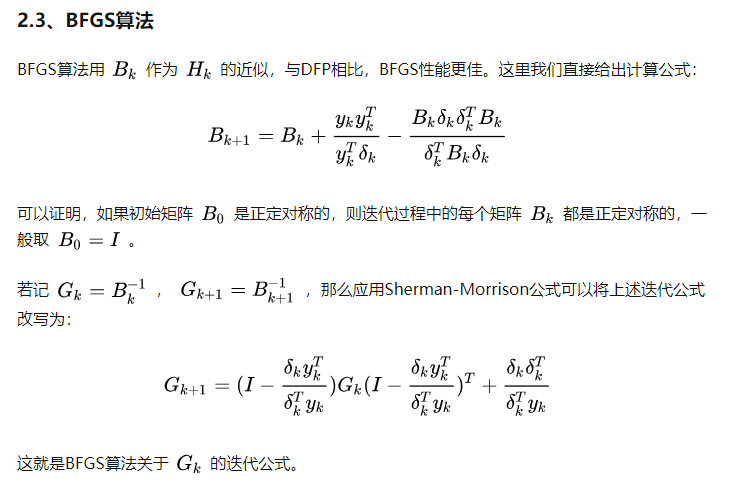
牛顿法的另一个弊病在于，每一次迭代都要计算 ，这一步计算比较复杂，下一节的拟牛顿法将解决这个问题。

③拟牛顿法

牛顿法虽然收敛速度快，但是需要计算海塞矩阵的逆矩阵 [公式] ，而且有时目标函数的海塞矩阵无法保持正定，从而使得牛顿法失效。为了克服这两个问题，人们提出了拟牛顿法。这个方法的基本思想是：不用二阶偏导数而构造出可以近似海塞矩阵（或海塞矩阵的逆）的正定对称阵。不同的构造方法就产生了不同的拟牛顿法。

DFP算法和BFGS算法都可以近似海塞矩阵。

22.2 拟牛顿法的bfgs的初始点满足什么要求



二者都是求解无约束最优化问题的常用方法，牛顿法是二阶收敛，梯度下降法是一阶收敛，所以牛顿法更快。

深度学习中，往往采用梯度下降法作为优化算子，而很少采用牛顿法，主要原因有以下几点：

1. 神经网络通常是非凸的，这种情况下，牛顿法的收敛性难以保证；

2. 即使是凸优化，只有在迭代点离全局最优很近时，牛顿法才会体现出收敛快的优势；

3. 可能被鞍点吸引。

22.3 机器学习哪些问题可以用凸优化方法解决

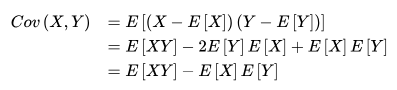
凸优化问题：逻辑回归、支持向量机

非凸优化：神经网络、主成分分析。

1. 协方差

协方差（Covariance）在概率论和统计学中用于衡量两个变量的总体误差。而方差是协方差的一种特殊情况，即当两个变量是相同的情况。

协方差表示的是两个变量的总体的误差，这与只表示一个变量误差的方差不同。 **如果两个变量的变化趋势一致，也就是说如果其中一个大于自身的期望值，另外一个也大于自身的期望值，那么两个变量之间的协方差就是正值。 如果两个变量的变化趋势相反，即其中一个大于自身的期望值，另外一个却小于自身的期望值，那么两个变量之间的协方差就是负值。（协方差与相关系数有关）**



协方差表示的是两个变量总体误差的期望

1. 伯努利分布

伯努利分布指的是对于随机变量X有, 参数为p(0<p<1)，如果它分别以概率p和1-p取1和0为值。EX= p,DX=p(1-p)。伯努利试验成功的次数服从伯努利分布,参数p是试验成功的概率。伯努利分布是一个离散型机率分布，是N=1时二项分布的特殊情况。

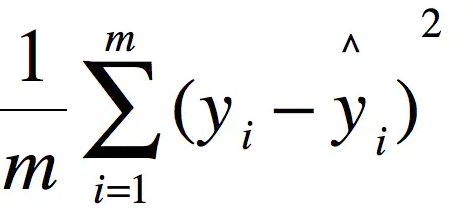
1. 假设检验的两类错误

假设检验及其两类错误是数理统计学中的名词。在进行假设检验时提出原假设和备择假设，原假设实际上是正确的，但我们做出的决定是拒绝原假设，此类错误称为第一类错误。原假设实际上是不正确的，但是我们却做出了接受原假设的决定，此类错误称为第二类错误。

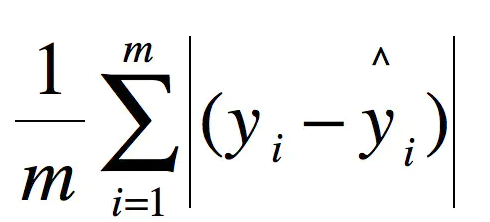


1. MSE、MAE、贝叶斯估计的区别

①均方误差(MSE)是最常用的回归损失函数：



②平均绝对误差（MAE）是另一种用于回归模型的损失函数。



即L2损失函数和L1损失函数。

③贝叶斯估计（即朴素贝叶斯的思路）：

贝叶斯估计（Bayesian estimation）是利用贝叶斯定理结合新的证据及以前的先验概率，来得到新的概率。它提供了一种计算假设概率的方法，基于假设的先验概率、给定假设下观察到不同数据的概率以及观察到的数据本身。