

[TOC]

一、机器学习相关

1、基本概念

- ☒ 1-1 简述解决一个机器学习问题时，你的流程是怎样的？
- ☒ 1-2 损失函数是什么，如何定义合理的损失函数？
- ☐ 1-3 回归模型和分类模型常用损失函数有哪些？各有什么优缺点
- ☐ 1-4 什么是结构误差和经验误差？训练模型的时候如何判断已经达到最优？
- ☐ 1-5 模型的“泛化”能力是指？如何提升模型泛化能力？
- ☒ 1-6 如何选择合适的模型评估指标？AUC、精准度、召回率、F1值都是什么？如何计算？有什么优缺点？
- ☒ 1-7 什么是混淆矩阵？
- ☒ 1-8 ROC曲线如何绘制？相比P-R曲线有什么特点？
- ☒ 1-9 如何评判模型是过拟合还是欠拟合？遇到过拟合或欠拟合时，你是如何解决？
- ☐ 1-10 你是如何针对应用场景选择合适的模型？
- ☒ 1-11 如何选择模型中的超参数？有什么方法，并说说其优缺点
- ☐ 1-12 误差分析是什么？你是如何进行误差分析？
- ☐ 1-13 你是如何理解模型的偏差和方差？什么样的情况是高偏差，什么情况是高方差？
- ☐ 1-14 出现高偏差或者高方差的时候你有什么优化策略？
- ☐ 1-15 奥卡姆剃刀定律是什么？对机器学习模型优化有何启发？举例说明
- ☒ 1-16 线性模型和非线性模型的区别？哪些模型是线性模型，哪些模型是非线性模型？
- ☐ 1-17 生成式模型和判别式模型的区别？哪些模型是生成式模型，哪些模型是判别式模型？

2、经典机器学习

特征工程

- ☐ 2-1-1 你是怎样理解“特征”？
- ☐ 2-1-2 给定场景和问题，你如何设计特征？（特征工程方法论）
- ☐ 2-1-3 开发特征时候如何做数据探索，怎样选择有用的特征？
- ☐ 2-1-4 你是如何做数据清洗的？举例说明

- ☐ 2-1-5 如何发现数据中的异常值，你是如何处理？
- ☐ 2-1-6 缺失值如何处理？
- ☐ 2-1-7 对于数值类型数据，你会怎样处理？为什么要做归一化？归一化有哪些方法？离散些方法，离散化和归一化有哪些优缺点？
- ☐ 2-1-8 标准化和归一化异同？
- ☐ 2-1-9 你是如何处理CTR类特征？
- ☐ 2-1-10 讲解贝叶斯平滑原理？以及如何训练得到平滑参数
- ☐ 2-1-11 类别型数据你是如何处理的？比如游戏品类，地域，设备
- ☐ 2-1-12 序号编码、one-hot编码、二进制编码都是什么？适合怎样的类别型数据？
- ☐ 2-1-13 时间类型数据你的处理方法是什么？原因？
- ☐ 2-1-14 你怎样理解组合特征？举个例子，并说明它和单特征有啥区别
- ☐ 2-1-15 如何处理高维组合特征？比如用户ID和内容ID？
- ☐ 2-1-16 如何理解笛卡尔积、外积、内积？
- ☐ 2-1-17 文本数据你会如何处理？
- ☐ 2-1-18 文本特征表示有哪些模型？他们的优缺点都是什么？
- ☐ 2-1-19 讲解TFF原理，它有什么优点和缺点？针对它的缺点，你有什么优化思路？
- ☐ 2-1-20 N-gram算法是什么？有什么优缺点？
- ☐ 2-1-21 讲解一下word2vec工作原理？损失函数是什么？
- ☐ 2-1-22 讲解一下LDA模型原理和训练过程？
- ☐ 2-1-23 Word2vec和LDA两个模型有什么区别和联系？
- ☐ 2-1-24 Skin-gram和cbow有何异同？
- ☐ 2-1-25 图像数据如何处理？有哪些常用的图像特征提取方法
- ☐ 2-1-26 你是怎样做特征选择的？卡方检验、信息值（IV）、VOE都是如何计算？
- ☐ 2-1-27 计算特征之间的相关性方法有哪些？有什么优缺点

基础算法原理和推导

KNN

- ☒ 2-2-1 Knn建模流程是怎样的？
- ☒ 2-2-2 Knn优缺点是什么？
- ☒ 2-2-3 Knn适合什么样的场景和数据类型？

- ☒ 2-2-4 常用的距离衡量公式都有哪些？具体说明它们的计算流程，以及使用场景？
- ☒ 2-2-5 超参数K值过大或者过小对结果有什么影响，你是如何选择K值？
- ☒ 2-2-6 介绍一下Kd树？如何建树，以及如何搜索最近节点？

支持向量机

- ☐ 2-3-1 简单讲解SVM模型原理？
- ☐ 2-3-2 SVM为什么会对缺失值敏感？实际应用时候你是如何处理？
- ☐ 2-3-3 SVM为什么可以分类非线性问题？
- ☐ 2-3-4 常用的核函数有哪些？你是如何选择不同的核函数的？
- ☐ 2-3-5 RBF核函数一定线性可分么？为什么
- ☐ 2-3-6 SVM属于线性模型还是非线性模型？为什么？
- ☐ 2-3-7 训练误差为0的SVM分类器一定存在吗？说明原因？

朴素贝叶斯模型

- ☐ 2-4-1 讲解贝叶斯定理？
- ☐ 2-4-2 什么是条件概率、边缘概率、联合概率？
- ☐ 2-4-3 后验概率最大化的含义是什么？
- ☐ 2-4-4 朴素贝叶斯模型如何学习的？训练过程是怎样？
- ☐ 2-4-5 你如何理解生成模型和判别模型？
- ☐ 2-4-6 朴素贝叶斯模型“朴素”体现在哪里？存在什么问题？有哪些优化方向？
- ☐ 2-4-7 什么是贝叶斯网络？它能解决什么问题？
- ☐ 2-4-8 为什么说朴素贝叶斯也是线性模型而不是非线性模型呢？

线性回归

- ☐ 2-5-1 线性回归的基本思想是？
- ☐ 2-5-2 什么是“广义线性模型”？
- ☐ 2-5-3 线性回归常用的损失函数有哪些？优化算法有哪些？
- ☐ 2-5-4 线性回归适用什么类型的问题？有哪些优缺点？
- ☐ 2-5-5 请用最小二乘法推倒参数更新公式？

逻辑回归

- ☐ 2-6-1 逻辑回归相比于线性回归有什么异同？
- ☐ 2-6-2 逻辑回归和广义线性模型有何关系？
- ☐ 2-6-3 逻辑回归如何处理多标签分类？
- ☐ 2-6-4 为什么逻辑回归需要进行归一化或者取对数？
- ☐ 2-6-5 为什么逻辑回归把特征离散化之后效果会提升？
- ☐ 2-6-6 类别不平衡问题你是如何处理的？什么是过采样，什么是欠采样？举例
- ☐ 2-6-7 讲解L1和L2正则，它们都有什么作用，解释为什么L1比L2更容易产生稀疏解；对于存在线性相关的一组特征，L1正则如何选择特征？
- ☐ 2-6-8 使用交叉熵作为损失函数，梯度下降作为优化方法，推倒参数更新公式
- ☐ 2-6-9 代码写出训练函数

FM模型

- ☐ 2-7-1 FM模型与逻辑回归相比有什么优缺点？
- ☐ 2-7-2 为什么FM模型计算复杂度时 $O(kn)$ ？
- ☐ 2-7-3 介绍FFM场感知分解机器（Field-aware Factorization Machine），说说与FM异同？
- ☐ 2-7-4 使用FM进行模型训练时候，有哪些核心参数对模型效果影响大？
- ☐ 2-7-5 如何从神经网络的视角看待FM模型？

决策树

- ☒ 2-8-1 讲解完成的决策树的建树过程
- ☒ 2-8-2 你是如何理解熵？从数学原理上解释熵公式可以作为信息不确定性的度量？
- ☒ 2-8-3 联合熵、条件熵、KL散度、信息增益、信息增益比、gini系数都是什么？如何计算？
- ☒ 2-8-4 常用的决策树有哪些？ID3、C4.5、CART有啥异同？
- ☒ 2-8-5 决策树如何防止过拟合？前剪枝和后剪枝过程是怎样的？剪枝条件都是什么

随机森林 (RF)

- ☐ 2-9-1 介绍RF原理和思想
- ☐ 2-9-2 RF是如何处理缺失值？
- ☐ 2-9-3 RF如何衡量特征重要度？
- ☐ 2-9-4 RF“随机”主要体现在哪里？
- ☐ 2-9-5 RF有哪些优点和局限性？
- ☐ 2-9-6 为什么多个弱分类器组合效果会比单个要好？如何组合弱分类器可以获得更好的结果？原因是什么？
- ☒ 2-9-7 Bagging的思想是什么？它是降低偏差还是方差，为什么？
- ☒ 2-9-8 可否将RF的基分类模型由决策树改成线性模型或者knn？为什么？

GBDT

- ☒ 2-10-1 梯度提升和梯度下降有什么区别和联系？
- ☒ 2-10-2 你是如何理解Boosting和Bagging？他们有什么异同？
- ☐ 2-10-3 讲解GBDT的训练过程？
- ☐ 2-10-4 你觉得GBDT训练过程中哪些环节可以平行提升训练效率？
- ☒ 2-10-5 GBDT的优点和局限性有哪些？
- ☐ 2-10-6 GBDT是否对异常值敏感，为什么？
- ☐ 2-10-7 如何防止GBDT过拟合？
- ☐ 2-10-8 在训练过程中哪些参数对模型效果影响比较大？这些参数造成影响是什么？

k-means

- ☐ 2-11-1 简述kmeans建模过程？
- ☐ 2-11-2 Kmeans损失函数是如何定义？
- ☐ 2-11-3 你是如何选择初始类簇的中心点？
- ☐ 2-11-4 如何提升kmeans效率？
- ☐ 2-11-5 常用的距离衡量方法有哪些？他们都适用什么类型问题？
- ☐ 2-11-6 Kmeans对异常值是否敏感？为什么？
- ☐ 2-11-7 如何评估聚类效果？
- ☐ 2-11-8 超参数类的个数k如何选取？
- ☐ 2-11-9 Kmeans有哪些优缺点？是否有了解过改进的模型，举例说明？
- ☐ 2-11-10 试试证明kmeans算法的收敛性
- ☐ 2-11-11 除了kmeans聚类算法之外，你还了解哪些聚类算法？简要说明原理

PCA降维

- ☐ 2-12-1 为什么要对数据进行降维？它能解决什么问题？
- ☐ 2-12-2 你是如何理解维度灾难？
- ☐ 2-12-3 PCA主成分分析思想是什么？
- ☐ 2-12-4 如何定义主成分？
- ☐ 2-12-5 如何设计目标函数使得降维达到提取主成分的目的？
- ☐ 2-12-6 PCA有哪些局限性？如何优化

- ☐ 2-12-7 线性判别分析和主成分分析在原理上有何异同？在目标函数上有何区别和联系？

3、深度学习

DNN

- ☐ 3-1-1 描述一下神经网络？推倒反向传播公式？
- ☒ 3-1-2 讲解一下dropout原理？
- ☐ 3-1-3 梯度消失和梯度膨胀的原因是什么？有什么方法可以缓解？
- ☐ 3-1-4 什么时候该用浅层神经网络，什么时候该选择深层网络
- ☐ 3-1-5 Sigmoid、Relu、Tanh激活函数都有哪些优缺点？
- ☐ 3-1-6 写出常用激活函数的导数
- ☐ 3-1-7 训练模型的时候，是否可以把网络参数全部初始化为0？为什么
- ☐ 3-1-8 Batchsize大小会如何影响收敛速度？

CNN

- ☐ 3-2-1 简述CNN的工作原理？
- ☐ 3-2-2 卷积核是什么？选择大卷积核和小卷积核有什么影响？
- ☐ 3-2-3 你在实际应用中如何设计卷积核？
- ☐ 3-2-4 为什么CNN具有平移不变性？
- ☐ 3-2-5 Pooling操作是什么？有几种？作用是什么？
- ☐ 3-2-6 为什么CNN需要pooling操作？
- ☐ 3-2-7 什么是batchnormalization？它的原理是什么？在CNN中如何使用？
- ☒ 3-2-8 卷积操作的本质特性包括稀疏交互和参数共享，具体解释这两种特性以其作用？
- ☐ 3-2-9 你是如何理解fine-tune？有什么技巧

RNN

- ☐ 3-3-1 简述RNN模型原理，说说RNN适合解决什么类型问题？为什么
- ☐ 3-3-2 RNN和DNN有何异同？
- ☐ 3-3-3 RNN为什么有记忆功能？
- ☐ 3-3-4 长短期记忆网络LSTM是如何实现长短期记忆功能的？
- ☐ 3-3-5 长短期记忆网络LSTM各模块都使用什么激活函数，可以使用其他激活函数么？
- ☐ 3-3-6 GRU和LSTM有何异同

- ☐ 3-3-7 什么是Seq2Seq模型？该模型能解决什么类型问题？
- ☐ 3-3-8 注意力机制是什么？Seq2Seq模型引入注意力机制主要解决什么问题？

4、基础工具

Spark

- ☐ 4-1-1 什么是宽依赖，什么是窄依赖？哪些算子是宽依赖，哪些是窄依赖？
- ☐ 4-1-2 Transformation和action算子有什么区别？举例说明
- ☐ 4-1-3 讲解sparkshuffle原理和特性？shuffle write 和 huffleread过程做些什么？
- ☐ 4-1-4 哪些spark算子会有shuffle？
- ☐ 4-1-5 讲解sparkschedule（任务调度）？
- ☐ 4-1-6 Sparkstage是如何划分的？
- ☐ 4-1-7 Sparkcache一定能提升计算性能么？说明原因？
- ☐ 4-1-8 Cache和persist有什么区别和联系？
- ☐ 4-1-9 RDD是弹性数据集，“弹性”体现在哪里呢？你觉得RDD有哪些缺陷？
- ☐ 4-1-10 当GC时间占比很大可能的原因有哪些？对应的优化方法是？
- ☐ 4-1-11 park中repartition和coalesce异同？coalesce什么时候效果更高，为什么
- ☐ 4-1-12 Groupbykey和reducebykey哪个性能更高，为什么？
- ☐ 4-1-13 你是如何理解caseclass的？
- ☐ 4-1-14 Scala里trait有什么功能，与class有何异同？什么时候用trait什么时候用class
- ☐ 4-1-15 Scala 语法中to 和 until有啥区别
- ☐ 4-1-16 讲解Scala伴生对象和伴生类

Xgboost

- ☒ 4-2-1 你选择使用xgboost的原因是什么？
- ☒ 4-2-2 Xgboost和GBDT有什么异同？
- ☒ 4-2-3 为什么xgboost训练会那么快，主要优化点事什么？
- ☒ 4-2-4 Xgboost是如何处理缺失值的？
- ☒ 4-2-5 Xgboost和lightGBM有哪些异同？
- ☒ 4-2-6 Xgboost为什么要使用泰勒展开式，解决什么问题？
- ☒ 4-2-7 Xgboost是如何寻找最优特征的？

Tensorflow

- ☐ 4-3-1 使用tensorflow实现逻辑回归，并介绍其计算图
- ☐ 4-3-2 sparse_softmax_cross_entropy_with_logits和softmax_cross_entropy_with_logits有何异同？
- ☐ 4-3-3 使用tensorflow过程中，常见调试哪些参数？举例说明
- ☐ 4-3-4 Tensorflow梯度更新是同步还是异步，有什么好处？
- ☐ 4-3-5 讲解一下TFRecords
- ☐ 4-3-6 tensorflow如何使用如何实现超大文件训练？
- ☐ 4-3-7 如何读取或者加载图片数据？

5、推荐系统

- ☐ 5-1-1 你是如何选择正负样本？如何处理样本不均衡的情况？
- ☐ 5-1-2 如何设计推荐场景的特征体系？举例说明
- ☐ 5-1-3 你是如何建立用户模型来理解用户，获取用户兴趣的？
- ☐ 5-1-4 你是如何选择适合该场景的推荐模型？讲讲你的思考过程
- ☐ 5-1-5 你是如何理解当前流行的召回->粗排->精排的推荐架构？这种架构有什么优缺点？什么场景适用使用，什么场景不适合？
- ☐ 5-1-6 如何解决热度穿透的问题？（因为item热度非常高，导致ctr类特征主导排序，缺少个性化的情况）
- ☐ 5-1-7 用户冷启动你是如何处理的？
- ☐ 5-1-8 新内容你是如何处理的？
- ☐ 5-1-9 你们使用的召回算法有哪些？如何能保证足够的召回率？
- ☐ 5-1-10 实时数据和离线数据如何融合？工程上是怎样实现？如何避免实时数据置信度不高带来的偏差问题？
- ☐ 5-1-11 你们是如何平衡不同优化目标的问题？比如：时长、互动等
- ☐ 5-1-12 不同类型内容推荐时候，如何平衡不同类型内容，比如图文、视频；或者不同分类
- ☐ 5-1-13 如何保证线上线下数据一致性？工程上是如何实现？
- ☐ 5-1-14 离线训练效果好，但是上线效果不明显或在变差可能是什么问题？如何解决？
- ☐ 5-1-15 在实际业务中，出现badcase,你是如何快速反查问题的？举例说明
- ☐ 5-1-16 使用ctr预估的方式来做精排，会不会出现相似内容大量聚集？原因是什么？你是如何解决的？
- ☐ 5-1-17 你了解有多少种相关推荐算法？各有什么优缺点

- ☐ 5-1-18 深度学习可以应用到推荐问题上解决哪些问题？为什么比传统机器学习要好？

二、数学相关

6、概率论和统计学

- ☐ 6-1-1 说说你是怎样理解信息熵的？
- ☐ 6-1-2 能否从数据原理熵解析信息熵可以表示随机变量的不确定性？
- ☐ 6-1-3 怎样的模型是最大熵模型？它有什么优点
- ☐ 6-1-4 什么是Beta分布？它与二项分布有什么关系？
- ☐ 6-1-5 什么是泊松分布？它与二项分布有什么关系？
- ☐ 6-1-6 什么是t分布？他与正态分布有什么关系？
- ☐ 6-1-7 什么是多项式分布？具体说明？
- ☐ 6-1-8 参数估计有哪些方法？
- ☐ 6-1-9 点估计和区间估计都是什么？
- ☐ 6-1-10 讲解一下极大似然估计，以及适用场景？
- ☐ 6-1-11 讲解一下最大后验概率估计，以及适用场景？极大似然估计和最大后验概率估计的区别

7、最优化问题

- ☐ 7-1-1 什么是梯度？
- ☐ 7-1-2 梯度下降找到的一定是下降最快的方法？
- ☐ 7-1-3 牛顿法和梯度法有什么区别？
- ☐ 7-1-4 什么是拟牛顿法？
- ☐ 7-1-5 讲解什么是拉格朗日乘子法、对偶问题、kkt条件？
- ☐ 7-1-6 是否所有的优化问题都可以转化为对偶问题？
- ☐ 7-1-7 讲解SMO (Sequential Minimal Optimization) 算法基本思想？
- ☐ 7-1-8 为什么深度学习不用二阶优化？
- ☐ 7-1-9 讲解SGD, ftrl, Adagrad, Adadelata, Adam, Adamax, Nadam优化算法以及他们的联系和优缺点
- ☐ 7-1-10 为什么batch size大，训练速度快or为什么mini-batch比SGD快？
- ☐ 7-1-11 为什么不把batch size设得特别大

解答

一、机器学习相关

1、基本概念

- ☐ 1-1 简述解决一个机器学习问题时，你的流程是怎样的？
 1. 确定问题：有监督问题还是无监督问题？回归问题还是分类问题？

2. 数据收集与处理
3. 特征工程：包括特征构建、特征选择、特征组合等
4. 模型训练、调参、评估：包括模型的选择，选择最优的参数
5. 模型部署：模型在线上运行的效果直接决定模型的成败

- ☐ 1-2 损失函数是什么，如何定义合理的损失函数？

机器学习模型关于单个样本的预测值与真实值的差称为**损失**。用于计算损失的函数称为**损失函数**。损失函数是 $f(x)$ 和 y 的非负实值函数。

- ☐ 1-3 回归模型和分类模型常用损失函数有哪些？各有什么优缺点

回归模型常用的损失函数有：

1. 0-1损失函数： $L(f(x), y) = \begin{cases} 1, & y \neq f(x) \\ 0, & y = f(x) \end{cases}$
2. 绝对损失函数：异常点多的情况下鲁棒性好；但不方便求导 $L(f(x), y) = |f(x) - y|$
3. 平方损失函数：求导方便，能够用梯度下降法优化；对异常值敏感 $L(f(x), y) = (f(x) - y)^2$
4. 对数损失函数/对数似然损失函数： $L(P(Y|X), Y) = -\log P(Y|X)$
5. Huber 损失函数：结合了绝对损失函数和平方损失函数的优点；缺点是需要调整超参数 δ
 $L_{\text{Huber}}(f, y) = \begin{cases} (f - y)^2 & |f - y| \leq \delta \\ 2\delta |f - y| - \delta^2 & |f - y| > \delta \end{cases}$
6. Log-Cosh 损失函数：具有Huber的所有优点，同时二阶处处可微（牛顿法要求二阶可微） $L(f, y) = \log \cosh(f - y)$

- ☐ 1-4 什么是结构误差和经验误差？训练模型的时候如何判断已经达到最优？

经验风险（经验损失）：模型 $f(X)$ 关于训练数据集的平均损失 $R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$
 结构风险：是在经验风险上加上表示模型复杂度的正则化项 $R_{\text{srn}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$
 经验风险最小化的策略认为，经验风险最小的模型是最优的模型。

结构风险最小化是为了防止过拟合而提出的，结构风险最小化等价于正则化。结构风险最小化的策略认为结构风险最小的模型是最优的模型。

- ☐ 1-5 模型的“泛化”能力是指？如何提升模型泛化能力？

通常将模型对未知数据的预测能力称为泛化能力（generalization ability）。现实中采用最多的方法是通过测试误差来评价学习方法的泛化能力。

- ☐ 1-6 如何选择合适的模型评估指标？AUC、精准度、召回率、F1值都是什么？如何计算？有什么优缺点？

TP：将正类预测为正类数

FN：将正类预测为负类数

FP：将负类预测为正类数

TN：将负类预测为负类数

分类任务指标

Accuracy（准确率）：分类正确的样本占总样本个数的比例 $Accuracy = \frac{n_{\text{correct}}}{n_{\text{total}}}$

- 缺点：不同类别的样本比例非常不均衡时，占比大的类别往往成为影响准确率的最主要因素。比如，当负样本占99%时，分类器把所有样本都预测为负样本也可以获得99%的准确率。
- 解决：可以使用每个类别下的样本准确率的算术平均（平均准确率）作为模型评估的指标。

Precision（精确率）：分类正确的正样本个数占分类器判定为正样本的样本个数的比例 $Precision = \frac{TP}{TP+FP}$

Recall（召回率）：分类正确的正样本数占真正的正样本个数的比例 $Recall = \frac{TP}{TP+FN}$

F1-score：precision和recall的调和平均值；当精确率和召回率都高时，F1值也会高 $F1 = \frac{2 \times precision \times recall}{precision + recall}$ 在排序问题中，通常没有一个确定的阈值把得到的结果直接判定为正样本或负样本，而是采用Top N返回结果的Precision和Recall值来衡量排序模型的性能。即认为模型返回的Top N结果就是模型判定的正样本，计算前N个位置的Precision@N和Recall@N。为了综合评估一个排序模型的好坏，不仅要看模型在不同Top N下的Precision@N和Recall@N，而且最好画出模型的P-R曲线。P-R曲线的横轴是Recall，纵轴是Precision。

ROC：横坐标为假阳性率（False Positive Rate, FPR）；纵坐标为真阳性率（True Positive Rate, TPR）

$FPR = \frac{FP}{N}$ $TPR = \frac{TP}{P}$ 其中P是真实的正样本的数量，N是真实的负样本的数量，TP是P个正样本中被分类器预测为正样本的个数，FP是N个负样本中被预测为正样本的个数。

【如何绘制ROC曲线】通过不断移动分类器的“截断点”来生成曲线上的一组关键点。在二分类问题中，模型输出一般是预测样本为正例的概率，在输出最终的正例负例之前，我们需要制定一个阈值。大于该阈值的样本判定为正例，小于该阈值的样本判定为负例。通过动态调整截断点，绘制每个截断点对应位置，再连接所有点得到最终的ROC曲线。

AUC：ROC曲线下的面积大小。计算AUC值只要沿着ROC横轴做积分就可以。AUC取值一般在0.5~1之间。AUC越大，分类性能越好。AUC表示预测的正例排在负例前面的概率。

指标想表达的含义，简单来说其实就是随机抽出一对样本（一个正样本，一个负样本），然后用训练得到的分类器来对这两个样本进行预测，预测得到正样本的概率大于负样本概率的概率

$$AUC = \frac{\sum_{i \in \text{positiveClass}} rank_i - \frac{M(1+M)}{2}}{M \times N}$$

AUC为0.5表明对正例和负例没有区分能力，对于不论真实类别是1还是0，分类器预测为1的概率是相等的。

我们希望分类器达到的效果：对于真实类别为1的样本，分类器预测为1的概率（TPR）要大于真实类别为0而预测类别为1的概率（FPR），即 $y > x$

AUC的计算方法同时考虑了分类器对于正例和负例的分类能力，在样本不平衡的情况下，依然能够对分类器作出合理的评价。

思路：1.首先对预测值进行排序，排序的方式用了python自带的函数sorted，详见注释。

2.对所有样本按照预测值从小到大标记rank，rank其实就是index+1，index是排序后的sorted_pred数组中的索引

3.将所有正样本的rank相加，遇到预测值相等的情况，不管样本的正负性，对rank要取平均值再相加

4.将rank相加的和减去正样本排在正样本之后的情况，再除以总的组合数，得到auc

回归任务指标

RMSE：计算预测值和实际值的平均误差
$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

1-7 什么是混淆矩阵？

混淆矩阵，又称误差矩阵，就是分别统计分类模型归错类，归对类的观测值个数，然后把结果放在一个表里展示出来。这个表就是混淆矩阵。

混淆矩阵是ROC曲线绘制的基础，同时它也是衡量分类型模型准确度中最基本，最直观，计算最简单的方法。

混淆矩阵	预测结果	
真实情况	反例	正例
反例	TN（真反例）	FP（假正例）
正例	FN（假反例）	TP（真正例）

- TN：True Negative，被判定为负样本，事实上也是负样本
- FP：False Positive，被判定为正样本，但事实上是负样本
- FN：False Negative，被判定为负样本，但事实上是正样本
- TP：True Positive，被判定为正样本，事实上也是正样本

1-8 ROC曲线如何绘制？相比P-R曲线有什么特点？

ROC曲线的纵轴为TPR，横轴为FPR，曲线上每个点对应一个TPR和FPR。通过调整模型阈值可以得到一个个点，从而将各个点连起来即可得到ROC曲线。

一般情况下，PR曲线易受样本数量的影响，样本数量不均衡情况下PR曲线会有明显变化，故一般使用ROC曲线。

1-9 如何评判模型是过拟合还是欠拟合？遇到过拟合或欠拟合时，你是如何解决？

过拟合是指学习时选择的模型所包含的参数过多，以至出现这一模型对已知数据预测得很好，但对未知数据预测得很差的现象。

当训练集效果差，欠拟合（如accuracy<0.8）；训练集效果好，测试集效果差，过拟合

欠拟合解决方法：

1. 增加特征
2. 提高模型复杂度：神经网络提高神经元数、增加层数；SVM使用核函数；
3. 减小正则项的系数

过拟合解决方法：

1. 提高样本数量：
 - 神经网络：Data Augmentation（数据增强）

2. 简化模型：

- 神经网络使用 Dropout、Early Stopping
- 决策树剪枝、限制树的深度

3. 加入正则化项（L1或L2）或提高惩罚系数

4. 使用集成学习

5. 神经网络中使用dropout机制

6. early stopping

7. 标签平滑

- ☐ [1-10 你是如何针对应用场景选择合适的模型？](#)
- ☐ [1-11 如何选择模型中的超参数？有什么方法，并说说其优缺点](#)

超参搜索算法一般包括的要素（1）目标函数（2）搜索范围，上限和下限缺点（3）其他参数，如搜索步长。

1. 网格搜索

查找搜索范围内所有的点来确定最优值；实际应用中先用较大搜索范围和较大步长，寻找全局最优值可能位置；然后逐步缩小搜索范围和搜索步长，寻找更精确位置。

- 优点
 - 简单
 - 如果采用较大的搜索范围和较小步长，有很大概率找到全局最优值
- 缺点
 - 耗时
 - 目标函数非凸时，可能错过全局最优解

2. 随机搜索

不再测试上界和下界之间的所有值，而是在搜索范围中随机选取样本点。如果样本点集足够大，通过随机搜索也能大概率找到全局最优解或其近似。

- 优点
 - 更快
- 缺点
 - 可能错过全局最优解

3. 贝叶斯优化算法

对目标函数形状进行学习，找到使目标函数向全局最优值提升的参数。

- 优点
 - 不同于前两种方法测试一个新点时会忽略前一个点的信息；贝叶斯优化算法充分利用之前的信息
- 缺点
 - 容易陷入局部最优值

- ☐ [1-12 误差分析是什么？你是如何进行误差分析？](#)

误差诊断：通过训练误差和测试误差来分析模型是否存在高方差、高偏差。

- 如果训练误差较高：说明模型的偏差较大，模型出现了欠拟合。
- 如果训练误差较低，而测试误差较高：说明模型的方差较大，出现了过拟合。
- 如果训练误差较低，测试误差也较低：说明模型的方差和偏差都适中，是一个比较理想的模型。
- 如果训练误差较高，且测试误差更高：说明模型的方差和偏差都较大。

上述分析的前提是：训练集、测试集的数据来自于同一个分布，且最优误差较小。否则讨论更复杂。

- ☐ 1-13 你是如何理解模型的偏差和方差？什么样的情况是高偏差，什么情况是高方差？

- 高偏差对应于模型的欠拟合：模型过于简单，以至于未能很好的学习训练集，从而使得训练误差过高。

此时模型预测的方差较小，表示预测较稳定。但是模型预测的偏差会较大，表示预测不准确。

- 高方差对应于模型的过拟合：模型过于复杂，以至于将训练集的细节都学到，将训练集的一些细节当做普遍的规律，从而使得测试集误差与训练集误差相距甚远。

- ☐ 1-14 出现高偏差或者高方差的时候你有什么优化策略？

- ☐ 1-15 奥卡姆剃刀定律是什么？对机器学习模型优化有何启发？举例说明

奥卡姆剃刀定律：若有多个假设与观察一致，则选最简单的那个

奥卡姆剃刀原理应用于模型选择时变为以下想法：在所有可能选择的模型中，能够很好地解释已知数据并且十分简单才是最好的，也就是应该选择的模型。

从贝叶斯估计的角度来看，正则化项对应于模型的先验概率。可以假设复杂的模型有较小的先验概率，简单的模型有较大的先验概率。

- ☐ 1-16 线性模型和非线性模型的区别？哪些模型是线性模型，哪些模型是非线性模型？

非概率模型可以分为线性模型和非线性模型。如果函数 $y=f(x)$ 或 $z = g(x)$ 是线性函数，则称模型是线性模型，否则成模型为非线性模型。

线性模型：感知机、线性支持向量机、k近邻、k均值、潜在语义分析

非线性模型：核函数支持向量机、AdaBoost，神经网络

- ☐ 1-17 生成式模型和判别式模型的区别？哪些模型是生成式模型，哪些模型是判别式模型？

监督学习方法分为生成方法（generative approach）和判别方法（discriminative approach）。所学习到的模型分别称为生成模型和判别模型。监督学习的模型一般形式为决策函数： $Y = f(X)$ 或者条件概率分布 $P(Y|X)$ 。

生成方法由数据学习联合概率分布 $P(X,Y)$ ，然后求出条件概率分布 $P(Y|X)$ 作为预测模型： $P(Y|X) = \frac{P(X,Y)}{P(X)}$ 之所以成为生成方法，是因为模型表示了给定输入 X 产生输出 Y 的生成关系。

典型的生成模型：朴素贝叶斯法、隐马尔可夫模型。

判别方法由数据直接学习决策函数 $f(X)$ 或者条件概率分布 $P(X,Y)$ 作为预测的模型，关心的是对给定的输入 X ，应该预测什么样的输出 Y 。

典型的判别模型：k近邻、感知机、决策树、逻辑斯蒂回归、最大熵模型、支持向量机、提升方法、条件随机场。

- ☐ 1-18 概率模型和非概率模型的区别？哪些模型是概率模型，哪些模型是非概率模型？

概率模型与非概率模型的区别在于模型的内在结构。概率模型一定可以表示为联合概率分布的形式，其中的变量表示输入、输出、因变量甚至参数。而针对非概率模型则不一定存在这样的联合概率分布。

统计学习的模型可以分为概率模型（probabilistic model）和非概率模型（non-probabilistic model）或者确定性模型（deterministic model）。在监督学习中，概率模型取条件概率分布形式 $p(y|x)$ ，非概率模型取函数形式 $y=f(x)$ ，其中 x 是输入， y 是输出。在无监督学习中，概率模型取条件概率分布形式 $p(z|x)$ 或 $p(x|z)$ ，其中 x 是输入， z 是输出。在监督学习中，概率模型是生成模型，非概率模型是判别模型。

概率模型：决策树、朴素贝叶斯、隐马尔可夫模型、条件随机场、概率潜在语义分析、潜在狄利克雷分布、高斯混合模型

非概率模型：感知机、支持向量机、k近邻、AdaBoost、k均值、潜在语义分析、神经网络

逻辑斯蒂回归即可看做概率模型，又可看做非概率模型。

- ☐ 1-19 参数化模型和非参数化模型的区别？哪些模型是参数化模型，哪些模型是非参数化模型？

统计学习模型又可以分为参数化模型（parametric model）和非参数化模型（non-parametric model）。参数化模型假设模型参数的维度固定，模型可以由有限维参数完全刻画；非参数模型假设模型参数的维度不固定或者说无穷大，随着训练数据量的增加而不断增大。

参数化模型：感知机、朴素贝叶斯、逻辑斯蒂回归、k均值、高斯混合模型

非参数化模型：决策树、支持向量机、AdaBoost、k近邻、潜在语义分析、概率潜在语义分析、潜在狄利克雷分配

2、经典机器学习

特征工程

- ☐ 2-1-1 你是怎样理解“特征”？
- ☐ 2-1-2 给定场景和问题，你如何设计特征？（特征工程方法论）
- ☐ 2-1-3 开发特征时候如何做数据探索，怎样选择有用的特征？
- ☐ 2-1-4 你是如何做数据清洗的？举例说明
- ☐ 2-1-5 如何发现数据中的异常值，你是如何处理？
- ☐ 2-1-6 缺失值如何处理？

1. 缺失值较多.直接将该特征舍弃掉，否则可能反而会带入较大的噪声，对结果造成不良影响。

2. 缺失值较少,其余的特征缺失值都在10%以内，我们可以采取很多的方式来处理：

1. 把NaN直接作为一个特征，假设用0表示；（离散特征取值k维扩充到k+1维）
2. 用均值填充；（连续特征-均值，离散特征-特征取值的众数）
3. 用随机森林等算法预测填充

- ☐ 2-1-7 对于数值类型数据，你会怎样处理？为什么要做归一化？归一化有哪些方法？离散化方法，离散化和归一化有哪些优缺点？

数值型特征

为了消除数据特征之间的量纲影响，我们需要对特征进行归一化处理，使得不同指标之间具有可比性。以梯度下降过程为例，如果不做归一化，在学习速率相同的情况下，大量纲变量的更新速度会大于小量纲，需要较多迭代才能找到最优解。如果将其归一化到相同的数值区间后，更新速度变得更为一致，容易更快地通过梯度下降找到最优解。

常用的归一化方法

(1) 线性函数归一化 (Min-Max Scaling)

对原始数据进行线性变化，使结果映射到[0, 1]范围，实现对原始数据的等比缩放。
$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$
适用于有固定取值范围的数据，如图像数据[0, 255]；不适用于有异常值的数据，例如有异常大的数值，其他数据会被压缩到很小的数值。

(2) 零均值归一化 (Z-score Normalization)

将原始数据映射到均值为0、标准差为1的分布上。假设原始特征的均值为 μ ，方差为 σ ，那么归一化公式定义为
$$z = \frac{x - \mu}{\sigma}$$

哪些机器学习算法不需要做归一化处理？

概率模型不需要归一化，因为它们不关心变量的值，而是关心变量的分布和变量之间的条件概率，如决策树、RF。而像Adaboost、GBDT、XGBoost、SVM、LR、KNN、KMeans之类的最优化问题就需要归一化。

离散化

分桶

- ☐ [2-1-8 标准化和正则化异同？](#)

简单来说，**标准化**是依照特征矩阵的列处理数据，其通过求z-score/min-max scaling的方法，将样本的特征值转换到同一量纲下。

正则化是依照特征矩阵的行处理数据，其目的在于样本向量在点乘运算或其他核函数计算相似性时，拥有统一的标准，也就是说都转化为“单位向量”。

- ☐ [2-1-9 你是如何处理CTR类特征？](#)
- ☐ [2-1-10 讲解贝叶斯平滑原理？以及如何训练得到平滑参数](#)
- ☐ [2-1-11 类别型数据你是如何处理的？比如游戏品类，地域，设备](#)

one-hot编码

- ☐ [2-1-12 序号编码、one-hot编码、二进制编码都是什么？适合怎样的类别型数据？](#)

序号编码

用于类别间具有大小关系的数据，例如成绩“高>中>低”，分别编码为3、2、1

one-hot编码

One-Hot Encoding采用N位状态位来对N个可能的取值进行编码，每个取值都由独立的状态位来表示，并且在任意时刻只有其中的一位有效。

用于类别间不具有大小关系的特征，比如血型

二进制编码

先用序号编码给每个类别赋予一个类别ID，然后将ID对应的二进制编码作为结果，最终得到0/1特征向量，且维数少于one-hot编码，节省了存储空间。

- ☐ 2-1-13 时间类型数据你的处理方法是什么？原因？
- ☐ 2-1-14 你怎样理解组合特征？举个例子，并说明它和单特征有啥区别
- ☐ 2-1-15 如何处理高维组合特征？比如用户ID和内容ID？
- ☐ 2-1-16 如何理解笛卡尔积、外积、内积？

笛卡尔积

笛卡尔乘积是指在数学中，两个集合X和Y的笛卡尔积（Cartesian product），又称直积，表示为 $X \times Y$ ，第一个对象是X的成员而第二个对象是Y的所有可能有序对的其中一个成员。

例如， $A=\{a,b\}$, $B=\{0,1,2\}$ ，则

$$A \times B = \{(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)\}$$

内积inner product或点积dot product

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

外积outer product

给定两个向量 $\mathbf{u} = (u_1, u_2, \dots, u_m)$ 和 $\mathbf{v} = (v_1, v_2, \dots, v_n)$

它们的外积 $\mathbf{u} \otimes \mathbf{v}$ 定义为 $m \times n$ 的矩阵 $\mathbf{u} \otimes \mathbf{v} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \cdots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \cdots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \cdots & u_m v_n \end{bmatrix}$ 或者用index表示为 $(\mathbf{u} \otimes \mathbf{v})_{ij} = u_i v_j$

内积为外积矩阵的迹

以下是矩阵的运算

哈达玛积Hadamard product（又称作逐元素积）

$$\mathbf{A} \circ \mathbf{B} = \begin{bmatrix} a_{1,1} b_{1,1} & a_{1,2} b_{1,2} & \cdots & a_{1,n} b_{1,n} \\ a_{2,1} b_{2,1} & a_{2,2} b_{2,2} & \cdots & a_{2,n} b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} b_{m,1} & a_{m,2} b_{m,2} & \cdots & a_{m,n} b_{m,n} \end{bmatrix}$$

- ☐ 2-1-17 文本数据你会如何处理？
- ☐ 2-1-18 文本特征表示有哪些模型？他们的优缺点都是什么？

词袋模型 (Bag of Words)：每篇文章看成一袋子词，并忽略每个词出现的顺序。每篇文章可以表示成一个长向量，向量中的每一位代表一个单词，该维对应的权重则反映这个词在原文章中的重要程度，常用TF-IDF来计算权重。

缺点：无法识别出两个不同的词或者词组具有相同的主题。

TF-IDF

N-gram模型：将连续出现的n个词 ($n \leq N$) 组成的词组也作为一个单独的特征放到向量表示中。

缺点：无法识别出两个不同的词或者词组具有相同的主题。

主题模型 (Topic Model)：是一种特殊的概率图模型

词嵌入模型 (Word Embedding)：将词向量化；核心思想是将每个词映射到低维空间 ($K=50 \sim 300$ 维) 上的一个稠密向量。K维空间的每一维也可以看作一个隐含的主题

- ☐ [2-1-19 讲解TFF原理，它有什么优点和缺点？针对它的缺点，你有什么优化思路？](#)

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

其中 $\text{TF}(t, d)$ 为单词t在文档d中出现的频率， $\text{IDF}(t)$ 是逆文档频率，用来衡量单词t对表达语义所起的重要性 $\text{IDF}(t) = \log \frac{\text{文章总数}}{\text{包含单词t的文章总数} + 1}$ 优点：

- ☐ [2-1-20 N-gram算法是什么？有什么优缺点？](#)
- ☐ [2-1-21 讲解一下word2vec工作原理？损失函数是什么？](#)

定义中心词预测上下文 $p(w_{t+1} | w_t)$

$$p(w_O | w_I) = \frac{\exp(v_{w_O}'^\top v_{w_I})}{\sum_{w=1}^W \exp(v_w'^\top v_{w_I})}$$

负采样

$$\log \sigma(v_{w_O}'^\top v_{w_I}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-v_{w_i}'^\top v_{w_I})]$$

- []
- ☐ [2-1-22 讲解一下LDA模型原理和训练过程？](#)
- ☐ [2-1-23 Word2vec和LDA两个模型有什么区别和联系？](#)
- ☐ [2-1-24 Skin-gram和cbow有何异同？](#)

不同点：skip-gram使用中心词预测上下文；CBOW用上下文预测中心词

相同点：都是根据word共现的频率建模word的相似度

- ☐ 2-1-25 图像数据如何处理？有哪些常用的图像特征提取方法
- ☐ 2-1-26 你是怎样做特征选择的？卡方检验、信息值（IV）、VOE都是如何计算？

从给定的特征集合中选出相关特征子集的过程称作特征选择 **feature selection**。

进行特征选择的原因：

- 首先，在现实任务中经常会遇到维数灾难问题，这是由于属性过多造成的。如果能从中选择出重要的特征，使得后续学习过程仅仅需要在一部分特征上构建模型，则维数灾难问题会大大减轻。

从这个意义上讲，特征选择与降维技术有相似的动机。事实上它们是处理高维数据的两大主流技术。

- 其次，去除不相关特征往往会降低学习任务的难度。

要想从初始的特征集合中选取一个包含了所有重要信息的特征子集，如果没有任何领域知识作为先验假设，则只能遍历所有可能的特征组合。

这在计算上是不可行的，因为这样会遭遇组合爆炸，特征数量稍多就无法进行。

一个可选的方案是：

- 产生一个候选子集，评价出它的好坏。
- 基于评价结果产生下一个候选子集，再评价其好坏。
- 这个过程持续进行下去，直至无法找到更好的后续子集为止。

常见的特征选择方法大致可分为三类：过滤式 **filter**、包裹式 **wrapper**、嵌入式 **embedding**。

过滤式选择

- 过滤式方法 先对数据集进行特征选择，然后再训练学习器，特征选择过程与后续学习器无关。

这相当于先用特征选择过程对初始特征进行过滤，再用过滤后的特征来训练模型。

包裹式选择

- 与过滤式特征选择不考虑后续学习器不同，包裹式特征选择直接把最终将要使用的学习器的性能作为特征子集的评价准则。其目的就是为给定学习器选择最有利于其性能、量身定做的特征子集。LVW算法中使用随机策略来进行子集搜索。
 - 优点：由于直接针对特定学习器进行优化，因此从最终学习器性能来看，效果比过滤式特征选择更好。
 - 缺点：需要多次训练学习器，因此计算开销通常比过滤式特征选择大得多。

嵌入式选择

- 嵌入式特征选择是将特征选择与学习器训练过程融为一体，两者在同一个优化过程中完成的。即学习器训练过程中自动进行了特征选择。
- 以线性回归模型为例。引入 L_1 范数（lasso回归）除了降低过拟合风险之外，还有一个好处：它求得的 $\overrightarrow{\mathbf{w}}$ 会有较多的分量为零。即：它更容易获得稀疏解。

- 于是基于 L_1 正则化的学习方法就是一种嵌入式特征选择方法，其特征选择过程与学习器训练过程融为一体，二者同时完成。
- $\overrightarrow{\mathrm{w}}$ 取得稀疏解意味着初始的 d 个特征中仅有对应着 $\overrightarrow{\mathrm{w}}$ 的非零分量的特征才会出现在最终模型中。 L_1 正则化问题的求解可以用近端梯度下降 **Proximal Gradient Descent: PGD** 算法求解。

- ☐ 2-1-27 计算特征之间的相关性方法有哪些？有什么优缺点

基础算法原理和推导

KNN

- ☐ 2-2-1 Knn建模流程是怎样的？

(1) 根据给定的距离度量，在训练集 T 中找出与 x 最邻近的 k 个点，涵盖这 k 个点的邻域记作 $N_k(x)$ ；

(2) 在 $N_k(x)$ 中根据分类决策规则（如多数表决）决定 x 的类别 y ： $y = \arg \max \{c_j\} \mid \sum_{x_i \in N_k(x)} I(y_i = c_j) \geq \frac{k}{2}, \quad i=1, 2, \dots, N_i \quad j=1, 2, \dots, K$ 在上式中， I 为指示函数，即当 $y_i = c_j$ 时为1，否则为0

- ☐ 2-2-2 Knn优缺点是什么？

knn优点：

1. 理论成熟，思想简单，既可以用来做分类又可以做回归
2. KNN是一种在线技术，新数据可以直接加入数据集而不必进行重新训练
3. 可用于非线性分类（数据集不要求线性可分）
4. 和朴素贝叶斯之类的算法比，对数据没有假设，准确度高，对异常点不敏感

knn缺点：

1. 计算量大，尤其是数据集非常大的时候
2. 样本不平衡的时候，对稀有类别的预测准确率低
3. KD树，球树之类的模型建立需要大量的内存
4. k 值大小的选择很重要

- ☐ 2-2-3 Knn适合什么样的场景和数据类型？

通常最近邻分类器使用于特征与目标类之间的关系为比较复杂的数字类型，或者说二者关系难以理解，但是相似类间特征总是相似。

数据要求归一化，统一各个特征的量纲。

- ☐ 2-2-4 常用的距离衡量公式都有哪些？具体说明它们的计算流程，以及使用场景？

特征空间 \mathcal{X} 是 n 维实数向量空间 \mathbf{R}^n ， $x_i, x_j \in \mathcal{X}$ ， $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)})$ ， $x_j = (x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(n)})$ 。则 x_i, x_j 的 L_p 距离

(闵可夫斯基距离) 定义为 $L_p(x_i, x_j) = (\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|)^{\frac{1}{p}}$ 这里 $p \geq 1$ 。

1. 欧式距离

当 $p=2$ 时, 称为欧氏距离, 强调数值上的绝对误差

是严格定义的距离, 满足正定性、对称性、三角不等式 $L_2(x_i, x_j) = (\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|)^{\frac{1}{2}}$

2. 曼哈顿距离 ($p=1$) $L_1(x_i, x_j) = \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|$

3. 切比雪夫距离 ($p = \infty$), 各个坐标距离数值差的绝对值的最大值 $L_{\infty}(x_i, x_j) = \max_l |x_i^{(l)} - x_j^{(l)}|$

4. 马氏距离

考虑各个分量 (特征) 之间的相关性并与各个分量的尺度无关。给定一个样本集合 X , $X = (x_{ij})_{m \times n}$, 其协方差矩阵记为 S 。样本 x_i 与样本 x_j 之间的马氏距离 d_{ij} 定义为 $d_{ij} = [(x_i - x_j)^T S^{-1} (x_i - x_j)]^{\frac{1}{2}}$ 当 S 为单位矩阵时, 即样本数据的各个分量互相独立且各个分量的方差为1时, 马氏距离就是欧氏距离。

汉明距离

两个等长字符串之间的汉明距离是两个字符串对应位置的不同字符的个数

1011101 与 1001001 之间的汉明距离是 2。

2143896 与 2233796 之间的汉明距离是 3。

"toned" 与 "roses" 之间的汉明距离是 3。

5. 相关系数 (correlation coefficient)

相关系数的绝对值越接近1, 表示样本越相似; 越接近0, 表示样本越不相似。

x_i 与 x_j 之间的相关系数定义为 $r_{ij} = \frac{\sum_{k=1}^m (x_{ki} - \overline{x_i})(x_{kj} - \overline{x_j})}{\sqrt{[\sum_{k=1}^m (x_{ki} - \overline{x_i})^2] [\sum_{k=1}^m (x_{kj} - \overline{x_j})^2]}}$

$\overline{x_i} = \frac{1}{m} \sum_{k=1}^m x_{ki}$, $\overline{x_j} = \frac{1}{m} \sum_{k=1}^m x_{kj}$

4. 余弦相似度

强调方向上的相对误差

不是严格定义的距离, 满足正定性、对称性, 不满足三角不等式 $\cos(A, B) = \frac{A \cdot B}{\|A\|_2 \|B\|_2}$

5. KL 散度

计算两个分布的差异性

不是严格定义的距离, 满足正定性, 不满足对称性、三角不等式

使用场景

欧氏距离: 适用于向量各分量的度量标准统一的情况; 当某些特征比其他特征取值大很多时, 精确度会变差, 很多特征值为0, 即稀疏矩阵, 结果不准, 数据点的分布是某个圆心的半径, 用欧式距离就不能比较了。

曼哈顿距离：适用于计算类似街区距离这样的实际问题。异常值对分类结果影响比欧式距离小。量纲不同时使用曼哈顿距离比欧式距离好。

总结

用距离度量相似度时，距离越小样本越相似；用相关系数时，相关系数越大样本越相似。

- 2-2-5 超参数K值过大或者过小对结果有什么影响，你是如何选择K值？

如果选择较小的K值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，K值的减小就意味着整体模型变得复杂，容易发生过拟合；

如果选择较大的K值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且K值的增大就意味着整体的模型变得简单。

$K=N$ ，则完全不足取，因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的类，模型过于简单，忽略了训练实例中大量有用信息。

在实际应用中，K值一般取一个比较小的数值，例如采用交叉验证法（简单来说，就是一部分样本做训练集，一部分做测试集）来选择最优的K值。

- 2-2-6 介绍一下Kd树？如何建树，以及如何搜索最近节点？

Kd树是一种对k维空间中的实例点进行存储，以便对其进行快速检索的树形数据结构。Kd树是二叉树，表示对k维空间的一个划分。构造Kd树相当于不断地用垂直于坐标轴的超平面将k维空间切分，构成一系列的k维超矩形区域。Kd树的每个节点对应于一个k维超矩形区域。

构造平衡Kd树过程：

输入：k维空间数据集 $T = \{x_1, x_2, \dots, x_N\}$ ，其中 $x_i = (x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(k)})^T$ ， $i=1, 2, \dots, N$

输出：Kd树

(1) 开始：构造根节点，根节点对应于包含T的k维空间的超矩形区域

选择 $x^{(1)}$ 为坐标轴，以T中所有实例的 $x^{(1)}$ 坐标的中位数为切分点，将根节点对应的超矩形区域且分为两个子区域。切分由通过切分点并与坐标轴 $x^{(1)}$ 垂直的超平面实现。

由根节点生成深度为1的左、右子节点：左子节点对应坐标 $x^{(1)}$ 小于切分点的子区域，右子节点对应于坐标 $x^{(1)}$ 大于切分点的子区域。

将落在切分超平面上的实例点保存在根节点。

(2) 重复：对深度为j的节点，选择 $x^{(l)}$ 为切分的坐标轴， $l = j \bmod k + 1$ ，以该节点的区域中所有实例的 $x^{(l)}$ 坐标的中位数为切分点，将该节点对应的超矩形区域切分为两个子区域。切分由通过切分点并与坐标轴 $x^{(l)}$ 垂直的超平面实现。

由该节点生成深度为j+1的左、右子节点：左子节点对应坐标 $x^{(l)}$ 小于切分点的子区域，右子节点对应坐标 $x^{(l)}$ 大于切分点的子区域。

将落在切分超平面上的实例点保存在根节点。

(3) 直到两个子区域没有实例存在时停止，从而形成kd树的区域划分

例 3.2 给定一个二维空间的数据集：

$$T = \{(2,3)^T, (5,4)^T, (9,6)^T, (4,7)^T, (8,1)^T, (7,2)^T\}$$

构造一个平衡 *kd* 树^③。

解 根结点对应包含数据集 *T* 的矩形，选择 $x^{(1)}$ 轴，6 个数据点的 $x^{(1)}$ 坐标的中位数是 7，以平面 $x^{(1)} = 7$ 将空间分为左、右两个子矩形（子结点）；接着，左矩形以 $x^{(2)} = 4$ 分为两个子矩形，右矩形以 $x^{(2)} = 6$ 分为两个子矩形，如此递归，最后得到如图 3.3 所示的特征空间划分和如图 3.4 所示的 *kd* 树。 ■

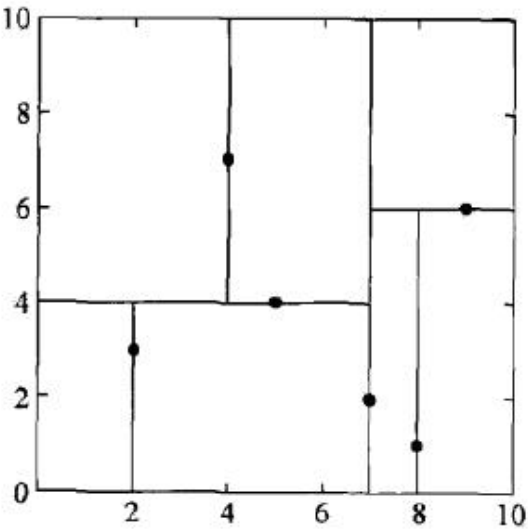


图 3.3 特征空间划分

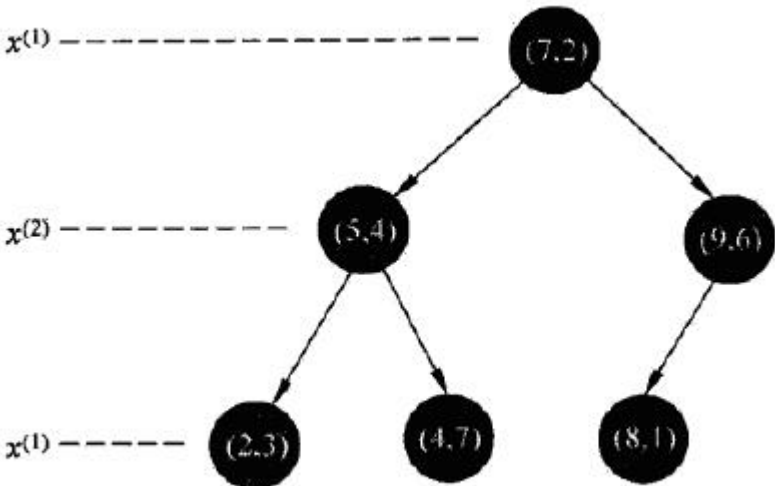


图 3.4 *kd* 树示例

算法：用kd树的最近邻搜索

输入：已构造的kd树：目标点 x ；

输出： x 的最近邻

(1) 在kd树中找出包含目标点 x 的叶节点：从根节点出发，递归地向下访问kd树。若目标点 x 当前维的坐标小于切分点的坐标，则移动到左子节点，否则移动到右子节点。直到子节点为叶节点为止。

(2) 以此叶节点为“当前最近点”。

(3) 递归地向上回退，在每个节点进行以下操作：

(a) 如果该节点保存的实例点比当前最近点距离目标点更近，则以该实例点作为“当前最近点”

(b) 当前最近点一定存在于该节点一个子节点对应的区域。检查该子节点的父节点的另一子节点（兄弟节点）对应区域是否有更近的点。具体地，检查另一子节点对应的区域是否与以目标点为球心、以目标点与“当前最近点”间的距离为半径的超球体相交。

如果相交，可能在另一个子节点对应的区域内存在距目标点更近的点，移动到另一个子节点。接着递归地进行最近邻搜索；

如果不相交，向上回退

(4) 当回退到根节点时，搜索结束。最后的“当前最近点”即为 x 的最近邻点。

如果实例点是随机分布的，kd树搜索的平均计算复杂度是 $O(\log N)$ ，这里 N 是训练实例数。****kd树更适用与训练实例数远大于空间维数时的k近邻搜索。当空间维数接近训练实例数时，它的效率会迅速下降，几乎接近线性扫描。**

例 3.3 给定一个如图 3.5 所示的 kd 树，根结点为 A ，其子结点为 B, C 等。树上共存储 7 个实例点；另有一个输入目标实例点 S ，求 S 的最近邻。

解 首先在 kd 树中找到包含点 S 的叶结点 D （图中的右下区域），以点 D 作为近似最近邻。真正最近邻一定在以点 S 为中心通过点 D 的圆的内部。然后返回结点 D 的父结点 B ，在结点 B 的另一子结点 F 的区域内搜索最近邻。结点 F 的区域与圆不相交，不可能有最近邻点。继续返回上一级父结点 A ，在结点 A 的另一子结点 C 的区域内搜索最近邻。结点 C 的区域与圆相交；该区域在圆内的实例点有点 E ，点 E 比点 D 更近，成为新的最近邻近似。最后得到点 E 是点 S 的最近邻。 ■

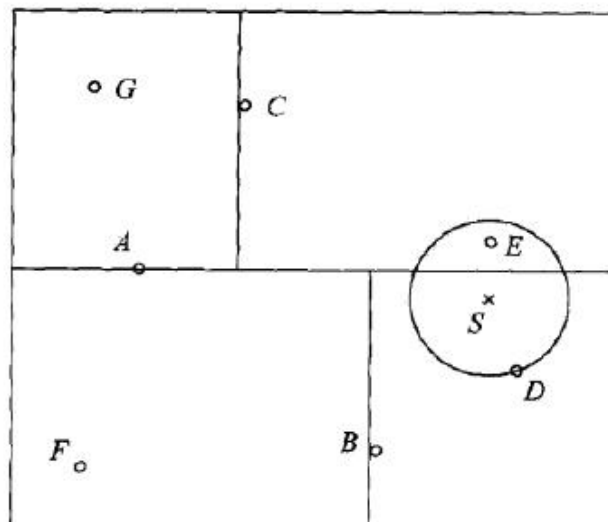


图 3.5 通过 kd 树搜索最近邻

支持向量机

hinge loss + kernel method

2-3-1 简单讲解SVM模型原理？

支持向量机是一种二类分类模型，它的基本模型是定义在特征空间的**间隔最大的线性分类器**，间隔最大，间隔最大使它有别于感知机；支持向量机还包括**核技巧**，这使它成为实质上的非线性分类器。支持向量机的学习策略是**间隔最大化**，可形式化为求解凸二次规划的问题，也等价于正则化的合页损失函数最小化问题。

线性可分支持向量机：当训练数据线性可分，通过硬间隔最大化，学习一个线性的分类器

线性支持向量机：当训练数据近似线性可分，通过软间隔最大化，学习一个线性的分类器

非线性支持向量机：当训练数据线性不可分，通过使用核技巧及软间隔最大化，学习非线性分类器

2-3-2 SVM为什么会对缺失值敏感？实际应用时候你是如何处理？

涉及到距离度量(distance measurement)时，如计算两个点之间的距离，缺失数据就变得比较重要。如果缺失值处理不当就会导致效果很差，如SVM，KNN。

常用的缺失值处理方法：

- (1) 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)。【快速简单但效果差】（平均数、中位数、众数、插值等）
- (2) 将缺失值当成新的数值，NaN
- (3) 忽略该项数据（当缺失少时）

2-3-3 SVM为什么可以分类非线性问题？

原输入空间是一个非线性可分问题，能用一个超曲面将正负例正确分开；

通过核技巧的非线性映射，将输入空间的超曲面转化为特征空间的超平面，原空间的非线性可分问题就变成了新空间的线性可分问题。低维映射到高维。

在核函数 $K(x, z)$ 给定的条件下，可以利用解线性分类问题的方法求解非线性分类问题的支持向量机。学习是隐式地在特征空间进行的，在学习和预测中只定义核函数 $K(x, z)$ ，而不需要显式地定义特征空间和映射函数 ϕ ，这样的技巧成为核技巧。通常直接计算 $K(x, z)$ 比较容易，而通过 $\phi(x)$ 和 $\phi(z)$ 计算 $K(x, z)$ 并不容易。

对于给定核 $K(x, z)$ ，特征空间和映射函数的取法并不唯一。

2-3-4 常用的核函数有哪些？你是如何选择不同的核函数的？

核函数定义： 设 \mathcal{X} 是输入空间，又设 \mathcal{H} 为特征空间，如果存在一个从 \mathcal{X} 到 \mathcal{H} 的映射 $\phi : \mathcal{X} \rightarrow \mathcal{H}$ 使得对所有 $x, z \in \mathcal{X}$ ，函数 $K(x, z)$ 满足条件 $K(x, z) = \phi(x) \cdot \phi(z)$ 则称 $K(x, z)$ 为核函数， $\phi(x)$ 为映射函数，式中 $\phi(x) \cdot \phi(z)$ 为 $\phi(x)$ 和 $\phi(z)$ 的内积

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

多项式核函数 (polynomial kernel function) $K(x, z) = (x \cdot z + 1)^p$ 对应的支持向量机是一个p次多项式分类器。分类决策函数为 $f(x) = \text{sign} \left(\sum_{i=1}^{N_s} a_i y_i \left(x \cdot x_i + 1 \right)^p + b \right)$ 多项式核函数可以实现将低维的输入空间映射到高维的特征空间，但是多项式核函数的参数多，当多项式的阶数比较高的时候，核矩阵的元素值将趋于无穷大或者无穷小，计算复杂度会大到无法计算。

高斯核函数 (Gaussian kernel function) $K(x, z) = \exp(-\frac{1}{2} \|x - z\|^2) = \phi(x) \cdot \phi(z)$ 对应的支持向量机是高斯径向基函数 (radial basis function) 分类器，分类决策函数为 $f(x) = \text{sign} \left(\sum_{i=1}^{N_s} a_i y_i \exp \left(-\frac{1}{2} \|x - x_i\|^2 \right) + b \right)$ 高斯径向基函数是一种局部性强的核函数，其可以将一个样本映射到一个更高维的空间内，该核函数是应用最广的一个，无论大样本还是小样本都有比较好的性能，而且其相对于多项式核函数参数要少，因此大多数情况下在不知道用什么核函数的时候，优先使用高斯核函数。

Sigmoid核函数 $K(x, z) = \tanh(\eta x \cdot z + \theta)$ 总结

- 如果特征的数量大到和样本数量差不多，则选用LR或者线性核的SVM；
 - (特征维度高，往往线性可分，SVM解决非线性分类问题的思路就是将样本映射到更高维的特征空间中)
- 如果样本数量很多，由于求解最优化问题的时候，目标函数涉及两两样本计算内积，使用高斯核明显计算量会大于线性核，所以手动添加一些特征，使得线性可分，然后用LR或者线性核的SVM
- 如果特征的数量小，样本的数量正常，则选用SVM+高斯核函数；

2-3-5 RBF核函数一定线性可分么？为什么

根据Cover定理，从低维度映射到高维度后，线性可分的可能性比较大。

而RBF核函数将原始空间映射到无穷维的特征空间，基本线性可分（也有线性不可分的情况，如加入噪声：同一样本不同标签）。如果忽略噪声，同时允许一定限度的误差，可以说升到足够高的维度，几乎所有数据集都是线性可分了。

同时，维度特别高，几乎一定线性可分，也以为这模型特别复杂，几乎一定会碰到过拟合问题。

2-3-6 SVM属于线性模型还是非线性模型？为什么？

基本模型是一个线性分类器；而通过使用核函数可以学习非线性支持向量机

2-3-7 训练误差为0的SVM分类器一定存在吗？说明原因？

对于训练一个不加入松弛变量的SVM模型时，一定存在。百面p55

对于加入松弛变量的SVM的训练误差不一定能达到0

2-3-8 当用支持向量机进行分类时，支持向量越多越好还是越少越好

结论：在n维特征空间中，线性SVM一般会产生n+1个支持向量（不考虑退化情况）

通常的SVM的使用会伴随着核技巧（kernel），这用于将低维空间映射到一个更高维的空间，使得原本非线性可分的数据点变得在高维空间中线性可分。虽然这种映射是隐式的，我们通常并不知道映射到的空间是什

么样子。但是根据之前的结论，我们可以认为如果训练出来的SVM有 $d+1$ 个支持向量，这个kernel在这个任务里就讲原来的数据映射到了一个 d 维的空间中，并使得其线性可分。

更高的维度通常意味着更高的模型复杂度，所以支持向量越多，表示着训练得到的模型越复杂。根据泛化理论，这意味着更有过拟合的风险。

如果在性能一致的情况下，更少支持向量可能是更好的。但是这一点其实不绝对，因为泛化理论仅仅是误差的上界，实际的泛化情况的决定因素比较复杂，也可能取决于kernel的性质。所以还是自己做cross validation比较好。

- 2-3-9 多类分类问题

1. 某些算法原生的支持多分类，如：决策树、最近邻算法等。但是有些算法只能求解二分类问题，如：支持向量机。
2. 对于只能求解二分类问题的算法，一旦遇到问题是多类别的，那么可以将多分类问题拆解成二分类任务求解。

即：

- 先对原问题进行拆分，然后为拆出的每个二分类任务训练一个分类器。
 - 测试时，对这些二分类器的预测结果进行集成，从而获得最终的多分类结果。
3. 多分类问题有三种拆解方式：
 - 一对其余(One-vs-rest: OvR) 。
 - 为每一对类别训练一个分类器。
 - 一对一(one-vs-one: OvO) 。
 - 训练 $k(k-1)$ 个分类器
 - 多对多(many-vs-many: MvM) 。

朴素贝叶斯模型

- 2-4-1 讲解贝叶斯定理？
$$P(B_i | A) = \frac{P(B_i) P(A | B_i)}{\sum_{j=1}^n P(B_j) P(A | B_j)}$$
- 2-4-2 什么是条件概率、边缘概率、联合概率？

条件概率：条件概率表示在条件 $Y=b$ 成立的情况下， $X=a$ 的概率，记作 $P(X=a|Y=b)$ 或 $P(a|b)$ 。它具有如下性质：“在条件 $Y=b$ 下 X 的条件分布”也是一种“ X 的概率分布”，因此穷举 X 的可取值之后，所有这些值对应的概率之和为1即： $\sum_a P(X=a | Y=b)=1$

边缘概率：仅与单个随机变量有关的概率称为边缘概率，如 $P(X=a)$ 或 $P(Y=b)$

联合概率：联合概率指的是包含多个条件且所有条件同时成立的概率，记作 $P(X=a,Y=b)$ 或 $P(a,b)$

联合概率、边缘概率与条件概率的关系： $P(X=a | Y=b) = \frac{P(X=a, Y=b)}{P(Y=b)}$

- 2-4-3 后验概率最大化的含义是什么？

朴素贝叶斯法将实例分到后验概率最大的类中。后验概率最大化这等价于期望风险最小化。

假设选择0-1损失函数： $L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$ 其中 $f(X)$ 是分类决策函数。这是期望风险函数为 $R_{\text{cap}}(f) = E[L(Y, f(X))]$ 期望是对联合分布 $P(X, Y)$ 取的。由此取条件期望 $R_{\text{exp}}(f) = E_X \left[\sum_{k=1}^K L(c_k, f(X)) P(c_k | X) \right]$ 为了使期望奉献最小化，只需对 $X=x$ 逐个最小化，由此得到
$$f(x) = \arg \min_{f \in \mathcal{Y}} \sum_{k=1}^K L(c_k, f(x)) P(c_k | X=x) = \arg \min_{f \in \mathcal{Y}} \sum_{k=1}^K P(y \neq c_k | X=x) = \arg \min_{f \in \mathcal{Y}} \left(1 - \sum_{k=1}^K P(y = c_k | X=x) \right) = \arg \max_{f \in \mathcal{Y}} \sum_{k=1}^K P(y = c_k | X=x)$$
 这样一来，根据期望风险最小化准则就得到了后验概率最大化准则： $f(x) = \arg \max_{c_k} P(c_k | X=x)$ 即朴素贝叶斯法所采用原理

2-4-4 朴素贝叶斯模型如何学习的？训练过程是怎样？

对于给定的训练数据集，首先基于特征条件独立性假设学习输入输出的联合概率分布；

然后基于此模型，对给定的输入 x ，利用贝叶斯定理求出后验概率最大的输出 y 。

训练过程：

(1) 计算先验概率及条件概率
$$P(Y=c_k) = \frac{\sum_{i=1}^N I(Y=c_k)}{N}, \quad k=1, 2, \dots, K$$

$$P(X^{(j)}=a_{lj} | Y=c_k) = \frac{\sum_{i=1}^N I(X^{(j)}=a_{lj}, Y=c_k)}{\sum_{i=1}^N I(Y=c_k)}, \quad j=1, 2, \dots, n; \quad l=1, 2, \dots, S_j; \quad k=1, 2, \dots, K$$
 (2) 对于给定实例 $x = (x^{(1)}, x^{(2)}, \dots, x^{(n)})^T$ ，计算
$$P(Y=c_k) \prod_{j=1}^n P(X^{(j)}=x^{(j)} | Y=c_k), \quad k=1, 2, \dots, K$$
 (3) 确定实例 x 的类（最大后验概率）
$$y = \arg \max_{c_k} P(Y=c_k) \prod_{j=1}^n P(X^{(j)}=x^{(j)} | Y=c_k)$$

2-4-5 你如何理解生成模型和判别模型？

生成方法由数据学习联合概率分布 $P(X, Y)$ ，然后求出条件概率分布 $P(Y|X)$ 作为预测模型：
$$P(Y|X) = \frac{P(X, Y)}{P(X)}$$
 之所以成为生成方法，是因为模型表示了给定输入 X 产生输出 Y 的生成关系。

典型的生成模型：朴素贝叶斯法、隐马尔可夫模型。

判别方法由数据直接学习决策函数 $f(X)$ 或者条件概率分布 $P(X, Y)$ 作为预测的模型，关心的是对给定的输入 X ，应该预测什么样的输出 Y 。

典型的判别模型：k近邻、感知机、决策树、逻辑斯蒂回归、最大熵模型、支持向量机、提升方法、条件随机场。

2-4-6 朴素贝叶斯模型“朴素”体现在哪里？存在什么问题？有哪些优化方向？

“朴素”体现在朴素贝叶斯模型对条件概率分布作了**条件独立性假设**，这是一个较强的假设。
$$P(X=x | Y=c_k) = P(X^{(1)}=x^{(1)}, \dots, X^{(n)}=x^{(n)} | Y=c_k) = \prod_{j=1}^n P(X^{(j)}=x^{(j)} | Y=c_k)$$
 存在问题：当特征分布不满足条件独立性假设时，分类的性能不高

优化方法：贝叶斯网络

- 2-4-7 什么是贝叶斯网络？它能解决什么问题？

朴素贝叶斯法假设输入变量都是条件独立的，如果假设他们之间存在概率依存关系，模型就被成了贝叶斯网络。

贝叶斯网络也称为“信念网”，借助有向无环图来刻画属性之间的依赖关系，并使用条件概率表来描述属性的联合概率分布。贝叶斯网结构有效地表达了属性的条件独立性。

具体来说，一个贝叶斯网B由结构G和参数 θ 表示，即 $B = \langle G, \theta \rangle$ 。网络结构G是一个邮箱无环图，其每个节点对应于一个属性，若两个属性有直接依赖关系，则它们由一条边连接起来；参数 θ 定量描述这种依赖关系，假设属性 x_i 在G中的父节点集为 π_i ，则 θ 包含了每个属性的条件概率 $\theta_{x_i|\pi_i} = P_B(x_i|\pi_i)$ 。

给定父节点集，贝叶斯网假设每个属性与它的非后裔属性独立，于是将属性的联合概率分布定义为

$$P_B(x_1, x_2, \dots, x_d) = \prod_{i=1}^d P_B(x_i | \pi_i) = \prod_{i=1}^d \theta_{x_i|\pi_i}$$

以上图为例，联合概率分布定义为

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1) P(x_2) P(x_3 | x_1) P(x_4 | x_1, x_2) P(x_5 | x_2)$$

贝叶斯网学习过程

精确求解NP难，所以（1）贪心法：从某个网络结构出发，每次调整一条边，直到评分函数值不再降低（2）给网络结构施加约束条件：如限定为树形结构等。

贝叶斯网推断

理想情况根据贝叶斯网定义的联合概率分布来精确计算后验概率，但这样的精确推断时NP难的。近似推断采用吉布斯采样法，通过随机游走，使马尔可夫链趋于平稳分布。

- 2-4-8 为什么说朴素贝叶斯也是线性模型而不是非线性模型呢？

线性分类器是通过特征的线性组合来做出分类决定的分类器。本质上，朴素贝叶斯分类器是一种线性分类器。

朴素贝叶斯分类器是建立在属性变量相互独立的基础上，后验概率为判定准则的分类器。不等式1成立，则样例 $x=[x_1, \dots, x_n]$ 为正类。否则，样例为负类。

(1)

$$\prod_{i=1}^n p(x_i|T)p(T) - \prod_{i=1}^n p(x_i|F)p(F) > 0$$

线性分类器直观地来说，是在高维样本空间中找到一组超平面，将样本空间划分了两个区域。每个区域对应于不同的类别。数学上来说，线性分类器能找到权值向量 w ，使得判别公式可以写成特征值的线性加权组合。

(2)

$$\sum_{i=1}^n w_i x_i - w_0 > 0$$

如果公式2成立，则样本属于正类；反之，则样本属于负类。

离散特征的朴素贝叶斯分类器

一般离散特征的取值范围有两种， $\{-1,1\}$ 或者 $\{0,1\}$ 。这两种取值方式不会影响分析。不妨假设离散特征的取值范围为 $\{-1,1\}$ 。下面的不等式成立，样例 $x=[x_1, \dots, x_n]$ 为正类。(3)

$$\prod_{i=1}^n p(x_i|T)p(T) - \prod_{i=1}^n p(x_i|F)p(F) > 0$$

$$\Rightarrow \sum_{i=1}^n [\ln p(x_i|T) - \ln p(x_i|F)] > \ln p(F) - \ln p(T)$$

对于某个特征 x ，我们很容易推导出下面的公式

(4)

$$p(x|T)$$

$$= \frac{1}{2}[p(x=1|T) - p(x=-1|T)] * x + \frac{1}{2}[p(x=1|T) + p(x=-1|T)]$$

其中 $p(x|F)$ 也有类似的结果，从而有 (5)

$$p(x|T) - p(x|F)$$

$$= \frac{1}{2}[p(x=1|T) - p(x=1|F) + p(x=-1|F) - p(x=-1|T)]$$

$$+ \frac{1}{2}[p(x=1|T) + p(x=-1|T) - p(x=-1|T) - p(x=-1|F)]$$

将公式5带入朴素贝叶斯分类器的公式3，得到下面的公式 (6)

$$\sum_{i=1}^n \frac{1}{2}[p(x_i=1|T) - p(x_i=1|F) + p(x_i=-1|F) - p(x_i=-1|T)]x_i$$

$$+ \sum_{i=1}^n \frac{1}{2}[p(x_i=1|T) + p(x_i=-1|T) - p(x_i=-1|T) - p(x_i=-1|F)]$$

$$> \ln p(F) - \ln p(T)$$

根据公式6，离散特征的朴素贝叶斯分类器判别公式能够写成特征值的加权线性组合。也就是说，离散特征的朴素贝叶斯分类器本质上是线性分类器。

- 2-4-9 如何解决用极大似然法可能出现所要估计的概率为0的情况？

分子加1，分母加可能情况数

用极大似然法估计可能会出现所要估计的概率值为0的情况。这是会影响到后验概率的计算结果，使分类产生偏差。解决这一问题的方法是采用贝叶斯估计。具体地，条件概率的贝叶斯估计是 $P_{\lambda}(X^{(j)}=a_{ij} | Y=c_k) = \frac{\sum_{i=1}^N I(x_i^{(j)}=a_{ij}, y_i=c_k) + \lambda}{\sum_{i=1}^N I(y_i=c_k) + S_j}$ 式中 $\lambda \geq 0$ 。等价于在随机变量各个取值的频数上赋予一个正数 λ 。当 $\lambda=0$ 时就是极大

似然估计。常取 $\lambda=1$ ，这时称为拉普拉斯平滑 (laplacian smoothing)。显然，对任何 $l=1,2,\dots, S_{\{j\}}, \quad k=1,2,\dots, K$ 有
$$P_{\lambda}\left(X^{\{j\}}=a_{\{j\}} \mid Y=c_{\{k\}}\right) > 0 \quad \left\{ \sum_{l=1}^{S_{\{j\}}} P_{\lambda}\left(X^{\{j\}}=a_{\{j\}} \mid Y=c_{\{k\}}\right) = 1 \right\}$$

表明上式确实为一种概率分布。同样，先验概率的贝叶斯估计是

$$P_{\lambda}\left(Y=c_{\{k\}}\right)=\frac{\sum_{i=1}^N I\left(y_{\{i\}}=c_{\{k\}}\right)+\lambda}{N+K \lambda}$$

线性回归

- ☐ 2-5-1 线性回归的基本思想是？
- ☐ 2-5-2 什么是“广义线性模型”？

考虑单调可微函数 $g(\cdot)$ ，令 $g(y)=\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}+b$ ，这样得到的模型称作广义线性模型 (generalized linear model)。其中函数 $g(\cdot)$ 称作联系函数 (link function)。

- ☐ 2-5-3 线性回归常用的损失函数有哪些？优化算法有哪些？
- ☐ 2-5-4 线性回归适用什么类型的问题？有哪些优缺点？
- ☐ 2-5-5 请用最小二乘法推倒参数更新公式？

逻辑回归

- ☐ 2-6-1 逻辑回归相比于线性回归有什么异同？

不同点：

- 逻辑回归处理的是分类问题，线性回归处理的是回归问题；
- 逻辑回归中认为 y 是因变量，即逻辑回归的因变量是离散的，线性回归的因变量是连续的。

相同点：

- 二者都使用了极大似然估计来对训练样本进行建模
- 求解超参数过程中，都可以使用梯度下降的方法

联系：

如果把一个事件的几率 (odds) 定义为该事件发生的概率与不发生概率的比值 $\frac{p}{1-p}$ ，那么逻辑回归可以看做是对于 $y=1|x$ 这一事件的对数几率的线性回归
$$\log \frac{p}{1-p} = \theta^T x$$
，其中 $p = P(y=1|x)$

- ☐ 2-6-2 逻辑回归和广义线性模型有何关系？

可以看做广义线性模型在因变量 y 服从二元分布时的一个特殊情况。

- ☐ 2-6-3 逻辑回归如何处理多标签分类？

如果一个样本只对应于一个标签（多分类问题）：假设每个样本属于不同标签的概率服从几何分布，使用 softmax regression 进行分类：

$$h_{\theta} = \begin{bmatrix} p(y=1|x;\theta) \\ p(y=2|x;\theta) \\ \vdots \\ p(y=k|x;\theta) \end{bmatrix}$$

$$\frac{1}{\sum_{j=1}^k e^{\theta_j^T x}}$$

$$\begin{bmatrix} e^{\theta_1^T x} & e^{\theta_2^T x} & \dots & e^{\theta_k^T x} \end{bmatrix}$$

其中 $\theta_1, \theta_2, \dots, \theta_k \in \mathbb{R}^n$

如果存在样本可能属于多个标签的情况时，可以训练 k 个二分类的逻辑回归分类器。第 i 个分类器用以区分每个样本是否可以归为第 i 类。

- ☐ 2-6-4 为什么逻辑回归需要进行归一化或者取对数？
 - 由于概率的乘积会因为很多原因不便使用（如容易出现数值下溢出），因此转换为对数的形式
- ☐ 2-6-5 为什么逻辑回归把特征离散化之后效果会提升？
 1. 在工业界很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列 0/1 的离散特征。

其优势有：

- 离散化之后得到的稀疏向量，内积乘法运算速度更快，计算结果方便存储。
 - 离散化之后的特征对于异常数据具有很强的鲁棒性。
- 如：销售额作为特征，当销售额在 $[30, 100)$ 之间时，为 1，否则为 0。如果未离散化，则一个异常值 10000 会给模型造成很大的干扰。由于其数值较大，它对权重的学习影响较大。
- 逻辑回归属于广义线性模型，表达能力受限，只能描述线性关系。特征离散化之后，相当于引入了非线性，提升模型的表达能力，增强拟合能力。

假设某个连续特征 x_j ，它离散化为 M 个 0/1 特征 $x_{j_1}, x_{j_2}, \dots, x_{j_M}$ 。则：

$$w_j \cdot x_j \rightarrow w_{j_1} \cdot x_{j_1}^{\prime} + w_{j_2} \cdot x_{j_2}^{\prime} + \dots + w_{j_M} \cdot x_{j_M}^{\prime}$$

其中 $x_{j_1}^{\prime}, \dots, x_{j_M}^{\prime}$ 是离散化之后的新的特征，它们的取值空间都是 $\{0, 1\}$ 。

上式右侧是一个分段线性映射，其表达能力更强。

- 离散化之后可以进行特征交叉。假设有连续特征 x_j ，离散化为 M 个 0/1 特征；连续特征 x_k ，离散化为 N 个 0/1 特征，则分别进行离散化之后引入了 $M+N$ 个特征。

假设离散化时，并不是独立进行离散化，而是特征 $M+N$ 联合进行离散化，则可以得到 $M \times N$ 个组合特征。这会进一步引入非线性，提高模型表达能力。

- 离散化之后，模型会更稳定。

如对销售额进行离散化， $[30, 100)$ 作为一个区间。当销售额在40左右浮动时，并不会影响它离散化后的特征的值。

但是处于区间连接处的值要小心处理，另外如何划分区间也是需要仔细处理。

2. 特征离散化简化了逻辑回归模型，同时降低模型过拟合的风险。

能够对抗过拟合的原因：经过特征离散化之后，模型不再拟合特征的具体值，而是拟合特征的某个概念。因此能够对抗数据的扰动，更具有鲁棒性。

另外它使得模型要拟合的值大幅度降低，也降低了模型的复杂度。

□ 2-6-6 类别不平衡问题你是如何处理的？什么是过采样，什么是欠采样？举例

这里讨论中，假设正类样本偏少、反类样本偏多。

1. 对于类别不平衡问题，常用的有三种方法：

- 基于再缩放策略进行决策，称之为阈值移动 **threshold-moving**。
- 直接对训练集里的反类样本进行欠采样 **undersampling**。
- 直接对训练集里的正类样本进行过采样 **oversampling**。

再缩放

1. 假设对样本 $\overrightarrow{\mathbf{x}}$ 进行分类时，预测为正类的概率为 p 。常规的做法是将 p 与一个阈值，比如 0.5，进行比较。如果 $p > 0.5$ 时，就判别该样本为正类。

概率 p 刻画了样本为正类的可能性，几率 $\frac{p}{1-p}$ 刻画了正类可能性与反类可能性的比值。

2. 当存在类别不平衡时，假设 N^+ 表示正类样本数目， N^- 表示反类样本数目，则观测几率是 $\frac{N^+}{N^-}$ 。

假设训练集是真实样本总体的无偏采样，因此可以用观测几率代替真实几率。于是只要分类器的预测几率高于观测几率就应该判断为正类。即如果 $\frac{p}{1-p} > \frac{N^+}{N^-}$ ，则预测为正类。

3. 通常分类器都是基于概率值来进行预测的，因此需要对其预测值进行调整。在进行预测的时候，令：

$$\frac{\overline{p}}{1-\overline{p}} = \frac{p}{1-p} \times \frac{N^-}{N^+}$$
 然后再将 \overline{p} 跟阈值比较。这就是类别不平衡学习的一个基本策略：再缩放 **rescaling**。

4. 再缩放虽然简单，但是由于“训练集是真实样本总体的无偏采样”这个假设往往不成立，所以无法基于训练集观测几率来推断出真实几率。

欠采样（下采样）

1. 欠采样会去除一些反类使得正、反类数目接近。
2. 欠采样若随机抛弃反类，则可能丢失一些重要信息。

常用方法是将反类划分成若干个集合供不同学习器使用，这样对每个学习器来看都是欠采样，但是全局来看并不会丢失重要信息。

过采样（上采样）

1. 过采样会增加一些正类使得正、反类数目接近。
2. 过采样不能简单的对原始正类进行重复采样，否则会导致严重的过拟合。

通常在原始正类之间插值来生成额外的正类。

3. 常见的有以下过采样策略：

- **SMOTE**方法：对于每个正类样本 \vec{x}_i ，从它的 k 近邻中随机选取一个样本点 \hat{x}_i ，然后根据下式生成一个新的正类样本：
$$\vec{x}_{n \in w} = \vec{x}_i + \left(\hat{x}_i - \vec{x}_i \right) \times \delta$$
，其中 $\delta \in [0, 1]$ 是随机数。（插值方法）

该方法有两个问题：

- 增加了正类样本之间重叠的可能性。
- 生成了一些没有提供有益信息的样本。
- ☐ 2-6-7 讲解L1和L2正则，它们都有什么作用，解释为什么L1比L2更容易产生稀疏解；对于存在线性相关的一组特征，L1正则如何选择特征？

L1和L2正则，都可以防止过拟合，增强模型的泛化能力；区别在于L1使参数更稀疏，达到特征选取的作用；L2使参数更接近于0

从解空间的形状来看：

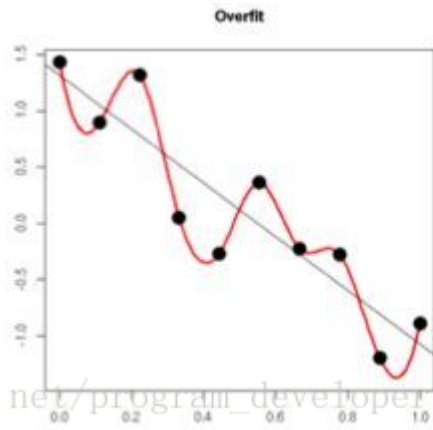
L1正则项约束后的解空间是多边形，而L2正则项约束后的解空间是圆形。而多边形的解空间更容易在尖角处与等高线碰撞出稀疏解。

对于存在线性相关的一组特征，L1正则会使部分参数为0

从函数叠加的观点：

- ☐ 权重衰减（L2正则化的作用）

作用：权重衰减（L2正则化）可以避免模型过拟合问题。**思考：**L2正则化项有让w变小的效果，但是为什么w变小可以防止过拟合呢？**原理：**（1）从模型的复杂度上解释：更小的权值w，从某种意义上说，表示网络的复杂度更低，对数据的拟合更好（这个法则也叫做奥卡姆剃刀），而在实际应用中，也验证了这一点，L2正则化的效果往往好于未经正则化的效果。（2）从数学方面的解释：过拟合的时候，拟合函数的系数往往非常大，为什么？如下图所示，过拟合，就是拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。而正则化是通过约束参数的范数使其不要太大，所以可以在一定程度上减少过拟合情况。



- 2-6-8 使用交叉熵作为损失函数，梯度下降作为优化方法，推倒参数更新公式

似然函数： $L(w) = \prod \left[\pi \left(x_i \right) \right]^{y_i} \left[1 - \pi \left(x_i \right) \right]^{1 - y_i}$ 对

数似然函数： $\begin{aligned} \ln L(w) &= \sum \left[y_i \ln \pi \left(x_i \right) + \left(1 - y_i \right) \ln \left(1 - \pi \left(x_i \right) \right) \right] \\ &= \sum \left[y_i \ln \frac{\pi \left(x_i \right)}{\pi \left(x_i \right) + \left(1 - \pi \left(x_i \right) \right)} + \left(1 - y_i \right) \ln \frac{\left(1 - \pi \left(x_i \right) \right)}{\pi \left(x_i \right) + \left(1 - \pi \left(x_i \right) \right)} \right] \\ &= \sum \left[y_i \ln \left(w \cdot x_i \right) - \ln \left(1 + e^{w \cdot x_i} \right) \right] \end{aligned}$ 损失函数：负对数似然函数 $J(W) = -\frac{1}{n} \ln L(w)$ 使用梯度下降法求解逻辑回归参数估计

求 $J(W)$ 梯度： $g(w)$: $\begin{aligned} J \left(w_{\{k\}} \right) &= -\frac{1}{N} \ln L \left(w_{\{k\}} \right) \\ \rightarrow -\ln L \left(w_{\{k\}} \right) &= -\sum \left[y_i \ln \left(w_{\{k\}} \cdot x_i \right) - \ln \left(1 + e^{w_{\{k\}} \cdot x_i} \right) \right] \end{aligned}$

求 $J(W)$ 梯度： $g(w)$: $\begin{aligned} J \left(w_{\{k\}} \right) &= -\frac{1}{N} \ln L \left(w_{\{k\}} \right) \\ \rightarrow -\ln L \left(w_{\{k\}} \right) &= -\sum \left[y_i \ln \left(w_{\{k\}} \cdot x_i \right) - \ln \left(1 + e^{w_{\{k\}} \cdot x_i} \right) \right] \end{aligned}$

求 $J(W)$ 梯度： $g(w)$: $\begin{aligned} J \left(w_{\{k\}} \right) &= -\frac{1}{N} \ln L \left(w_{\{k\}} \right) \\ \rightarrow -\ln L \left(w_{\{k\}} \right) &= -\sum \left[y_i \ln \left(w_{\{k\}} \cdot x_i \right) - \ln \left(1 + e^{w_{\{k\}} \cdot x_i} \right) \right] \end{aligned}$

求 $J(W)$ 梯度： $g(w)$: $\begin{aligned} g \left(w_{\{k\}} \right) &= -\frac{1}{N} \sum_i \left[x_i \cdot y_i - \frac{x_i \cdot e^{w_{\{k\}} \cdot x_i}}{1 + e^{w_{\{k\}} \cdot x_i}} \right] \\ &= -\frac{1}{N} \sum_i x_{\{ik\}} \left[y_i - \pi \left(w_{\{k\}} \cdot x_i \right) \right] \end{aligned}$

- 2-6-9 代码写出训练函数

FM模型

- 2-7-1 FM模型与逻辑回归相比有什么优缺点？

优点：

- 存储空间和计算复杂度减小
- 解决样本过于稀疏训练不充分的问题

缺点：

- 记忆能力不如LR

- 2-7-2 为什么FM模型计算复杂度时 $O(kn)$ ？
- 2-7-3 介绍FFM场感知分解机器（Field-aware Factorization Machine），说说与FM异同？
- 2-7-4 使用FM进行模型训练时候，有哪些核心参数对模型效果影响大？
- 2-7-5 如何从神经网络的视角看待FM模型？

决策树

- 2-8-1 讲解完成的决策树的建树过程

自上而下，对样本数据进行树形分类的过程。每个内部节点表示一个特征，叶节点表示一个类别。从顶部根节点开始，所有的样本聚在一起。经过根节点的划分，样本被分到不同的子节点，再根据子节点的特征进一步划分，直至所有样本被归到某一个类别（叶节点）中。

- 2-8-2 你是如何理解熵？从数学原理上解释熵公式可以作为信息不确定性的度量？

熵（entropy）是表示随机变量不确定性的度量， X 是一个取有限个值的离散随机变量，其概率分布为 $P(X = x_i) = p_i, i=1,2,\dots,n$ 则随机变量 X 的熵定义为 $H(X) = -\sum_{i=1}^n p_i \log p_i$ 熵越大，随机变量的不确定性就越大。

而熵其实表示的是一个系统的平均信息量。自信息量是用来描述某一条信息的大小 $I = -\log p_i$ 通常以2为底，单位是bit；含义是用多少位二进制可以表示衡量该信息的大小。而通常我们衡量整个系统的信息量，系统存在多个事件 $X=\{x_1,\dots,x_n\}$ ，每个事件的概率分布 $P=\{p_1,\dots,p_n\}$ ，熵是整个系统的平均信息量。

- 2-8-3 联合熵、条件熵、KL散度、信息增益、信息增益比、gini系数都是什么？如何计算？

联合熵：将一维随机变量分布推广到多维随机变量分布 $H(X,Y) = -\sum \lim_{x,y} p(x,y) \log p(x,y)$

条件熵：某个特征A对于数据集D的经验条件熵 $H(D|A)$ 为 $H(D|A) = -\sum_{i=1}^n \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}$

信息增益： $g(D,A)$ 定义为数据集D的经验熵 $H(D)$ 与特征A给定条件下D的经验条件熵 $H(D|A)$ 的差 $g(D,A) = H(D) - H(D|A)$

信息增益比：特征A对于数据集D的信息增益比定义为 $g_R(D|A) = \frac{g(D,A)}{H_A(D)}$ 其中 $H_A(D) = -\sum_{i=1}^n \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}$ 为数据集D关于A的取值熵；n为特征A在D上的取值数目；

Gini系数：描述数据的不确定性。数据集D的Gini系数为 $\text{Gini}(D) = 1 - \sum_{k=1}^K (\frac{|C_k|}{|D|})^2$ 其中 C_k 是 D 中第k类的样本子集，K是类的个数。例如二分类问题，K=2。基尼系数越大，样本集合的不确定性也就越大，这一点与熵相似。基尼系数 $\text{Gini}(D,A)$ 表示经 $A=a$ 分割后集合D的不确定性。

交叉熵：刻画两个概率分布之间的距离，通过q来表示p的交叉熵为；一般 $p(x)$ 为真实分布， $q(x)$ 为预测分布

交叉熵不对称。交叉熵越小，概率分布越接近 $H(p,q) = -\sum \lim_{x,y} p(x) \log q(x)$ **KL散度/**

相对熵： $D_{KL}(p||q) = \sum_{i=1}^n p(x_i) \log \frac{p(x_i)}{q(x_i)}$ n表示事件可能发生的情况总数，KL散度的值越小表示两个分布越接近。 $D_{KL}(p||q) = H(p,q) - H(p)$

机器学习中，我们常常使用KL散度来评估predict和label之间的差别，但是由于KL散度的后半部分是一个常量，所以我们常常将前半部分的交叉熵作为损失函数，其实二者是一样的。

- 2-8-4 常用的决策树有哪些？ID3、C4.5、CART有啥异同？

不同点	ID3	C4.5	CART
原则	信息增益最大	信息增益比最大	划分后集合基尼指数最小
用途	分类	分类	分类、回归

不同点	ID3	C4.5	CART
输入取值	离散	离散、连续	离散、连续
树结构	多叉树	多叉树	二叉树
	特征在层级间不复用	特征在层级间不复用	每个特征可被重复利用
	对样本特征缺失值敏感		

ID3 最大信息增益

信息增益 $g(D,A)$ 定义为数据集D的经验熵 $H(D)$ 与特征A给定条件下D的经验条件熵 $H(D|A)$ 的差 $g(D,A) = H(D) - H(D|A)$ 选择 $g(D,A)$ 最大的特征，所有样本根据此特征，划分到不同的节点上。在经验熵不为0的节点中继续生长。ID3算法只有树的生成，容易产生过拟合。

C4.5 最大信息增益比

因为信息增益对取值数目多的属性有所偏好，为了减少这种偏好带来的影响，使用信息增益比来选择最优划分属性。

CART 基尼指数

基尼系数Gini（D）用来表示集合D的不确定性。CART在每一次迭代中选择划分后**基尼指数最小**的特征及其对应的切分点进行分类。CART是一颗二叉树，每次将数据按特征A的区分分成两份，分别进入左右子树。

- [2-8-5 决策树如何防止过拟合？前剪枝和后剪枝过程是怎样的？剪枝条件都是什么](#)

通过**剪枝**防止过拟合。

预剪枝是指在决策树生成的过程中，对每个节点在划分前先进行估计，若当前节点的划分不能带来决策树泛化性能提升，则停止划分，并将当前节点标记为叶子节点；此时可能存在不同类别的样本同时存于同个节点中，按照多数投票的原则判断节点所属类别

预剪枝对于何时停止决策树的生长：

- (1) 当树达到一定深度
- (2) 当到达当前节点的样本数量小于某个阈值
- (3) 计算每次分裂对测试集的准确度提升，小于某个阈值时停止

后剪枝则是先从训练集生成一棵完整的决策树，然后自底向上地对**非叶子节点**进行考察，若该节点对应的子树**替换成叶子结点**能带来泛化性能提升，则将该子树替换为叶子节点。

随机森林（RF）

- [2-9-1 介绍RF原理和思想](#)
- [2-9-2 RF是如何处理缺失值？](#)
- [2-9-3 RF如何衡量特征重要度？](#)
- [2-9-4 RF“随机”主要体现在哪里？](#)
- [2-9-5 RF有哪些优点和局限性？](#)

- [□ 2-9-6 为什么多个弱分类器组合效果会比单个要好？如何组合弱分类器可以获得更好的结果？原因是什么？](#)
- [□ 2-9-7 Bagging的思想是什么？它是降低偏差还是方差，为什么？](#)

Bagging的思想是通过对数据再抽样，然后在每组样本上训练出来的模型取平均。Bagging是降低方差，防止过拟合。可以这样理解，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 $\frac{1}{n}$ 。

- [□ 2-9-8 可否将RF的基分类模型由决策树改成线性模型或者knn？为什么？](#)

随机森林属于bagging类的集成学习方法，主要好处是减小集成后分类器的方差，比基分类器的方差小。所以Bagging所采用的的基分类器最好是本身对样本分布较为敏感（不稳定分类器），这样bagging才能体现效果。而线性分类器和KNN属于较为稳定的分类器，本身方差不大，所以将他们作为基分类器使用bagging不能再原基分类器的基础上获得更好的表现。相反地，可能因为bagging的采样而使得训练中难以收敛从而增大集成分类器的偏差。

GBDT

梯度提升**Gradient Boosting**的基本思想是根据当前模型损失函数**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**的形式结合到现有的模型中。

GBDT（梯度提升决策树）的核心思想：每一棵树学的是之前所有树结果和的残差，这个残差就是加上预测之后能得到真实值的累加量。

- [□ 2-10-1 梯度提升和梯度下降有什么区别和联系？](#)

两者都是在每一轮迭代中，利用损失函数相对于模型的负梯度方向的信息来对当前模型进行更新，只不过在梯度下降中，模型是以参数化的形式表示，从而模型的更新等价于参数的更新。

而在梯度提升中，模型并不需要参数化表示，而是直接定义在函数空间中，从而大大扩展了可以使用的模型种类。

- [□ 2-10-2 你是如何理解Boosting和Bagging？他们有什么异同？](#)

Bagging通过模型集成降低方差，提高弱分类器的性能。

Boosting通过模型集成降低偏差，提高弱分类器的性能。

	Bagging	Boosting
降低	方差	偏差
训练	各个弱分类器可独立训练	弱分类器需要依次生成
典型方法	随机森林	Adaboost, GBDT, XGBoost

- [□ 2-10-3 讲解GBDT的训练过程？](#)

用每个样本的残差训练下一棵树，直到残差收敛到某个阈值以下，或者树的总数达到某个上限为止。

- [□ 2-10-4 你觉得GBDT训练过程中哪些环节可以平行提升训练效率？](#)

决策树内部局部并行。

- [2-10-5 GBDT的优点和局限性有哪些？](#)

优点

- (1) 预测阶段计算速度快，树与树之间可并行化计算
- (2) 在分布稠密的数据集上，泛化能力和表达能力都很好
- (3) 采用决策树作为弱分类器使得GBDT模型具有较好的可解释性和鲁棒性，能够自动发现特征间的高阶关系，并且不需要对数据进行特殊的预处理如归一化等

缺点

- (1) GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络
- (2) GBDT在处理文本分类特征问题上，优势不如在处理数值特征时明显
- (3) 训练过程需要串行训练，只能在决策树内容采用一些局部并行手段提高训练速度

- [2-10-6 GBDT是否对异常值敏感，为什么？](#)

GBDT对异常值敏感。对于回归类的问题，如果采用平方损失函数。当出现异常值时，后续模型会对异常值关注过多。

- [2-10-7 如何防止GBDT过拟合？](#)

1. 在工程应用中，通常利用下列公式来更新模型：
$$f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + \eta h_m(\mathbf{x}; \Theta_m)$$
，其中 $0 < \eta \leq 1$ 。

其中 η 称作**学习率**。

学习率是正则化的一部分，它可以降低模型更新的速度（需要更多的迭代）。

- 经验表明：一个小的学习率 ($\eta < 0.1$) 可以显著提高模型的泛化能力（相比较于 $\eta = 1$ ）。
- 如果学习率较大会导致预测性能出现较大波动。

2. **Freidman** 从**bagging** 策略受到启发，采用**随机梯度提升**来修改了原始的梯度提升树算法。

- 每一轮迭代中，新的决策树拟合的是原始训练集的一个子集（而并不是原始训练集）的残差。

这个子集是通过对原始训练集的无放回随机采样而来。

- 子集的占比 f 是一个超参数，并且在每轮迭代中保持不变。
 - 如果 $f=1$ ，则与原始的梯度提升树算法相同。
 - 较小的 f 会引入随机性，有助于改善过拟合，因此可以视作一定程度上的正则化。
 - 工程经验表明， $0.5 \leq f \leq 0.8$ 会带来一个较好的结果。
- 这种方法除了改善过拟合之外，另一个好处是：未被采样的另一部分子集可以用来计算包外估计误差。

因此可以避免额外给出一个独立的验证集。

3. 梯度提升树会限制每棵树的叶子结点包含的样本数量至少包含 m 个样本，其中 m 为超参数。在训练过程中，一旦划分结点会导致子结点的样本数少于 m ，则终止划分。

这也是一种正则化策略，它会改善叶结点的预测方差。

- ☐ [2-10-8 在训练过程中哪些参数对模型效果影响比较大？这些参数造成影响是什么？](#)

减小步长需要对应增加最大迭代次数

1. **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。一般来说 n_estimators 太小，容易欠拟合，n_estimators 太大，又容易过拟合，一般选择一个适中的数值。默认是100。在实际调参的过程中，我们常常将 n_estimators 和下面介绍的参数 learning_rate 一起考虑。
2. **learning_rate**: 即每个弱学习器的权重缩减系数 α ，也称作步长，在原理篇的正则化章节我们也讲到了，加上了正则化项
3. **subsample**: 即我们在原理篇的正则化章节讲到的子采样，取值为 $(0,1]$ 。注意这里的子采样和随机森林不一样，随机森林使用的是放回抽样，而这里是不放回抽样。如果取值为1，则全部样本都使用，等于没有使用子采样。如果取值小于1，则只有一部分样本会去做GBDT的决策树拟合。选择小于1的比例可以减少方差，即防止过拟合，但是会增加样本拟合的偏差，因此取值不能太低。推荐在 $[0.5, 0.8]$ 之间，默认是1.0，即不使用子采样。

k-means

kmean的总体特点

- 基于划分的聚类方法
- 类别k事先指定
- 以欧式距离平方表示样本之间的距离
- 以中心或样本的均值表示类别
- 以样本和其所属类的中心之间的距离总和为最优化的目标函数
- 得到的类别是平坦的、非层次化的
- 算法是迭代算法，不能保证全局最优
- ☐ [2-11-1 简述kmeans建模过程？](#)

kmeans聚类是基于样本集合划分的聚类算法

- (1) 首先随机选择k个样本点作为初始聚类中心
- (2) 计算每个样本到类中心的距离，将样本逐个指派到与其最近的类中，得到一个聚类结果
- (3) 更新每个类的样本的均值，作为新的中心
- (4) 重复以上步骤，知道划分不再改变，收敛为止

kmeans的算法复杂度是 $O(mnk)$ ，其中 m 是样本位数， n 是样本个数， k 是类别个数。比层次聚类复杂度低。

- 2-11-2 Kmeans损失函数是如何定义？

样本与所属类的中心之间的距离的总和为损失函数 $W(C) = \sum_{i=1}^k \sum_{C(i)=j} \|x_i - \overline{x}_j\|$ 其中 \overline{x}_j 是第 j 个类的均值或中心。相似的样本被聚到同类时，损失函数值最小。但是这是一个组合优化问题， n 个样本分到 k 个类，可能的分法数目是指数级的，NP难问题。采样迭代的方法求解。

- 2-11-3 你是如何选择初始类族的中心点？

初始类中心点的选择

(1) 可以用层次聚类对样本进行聚类，得到 k 个类时停止。然后从每个类中选取一个与中心距离最近的点。

类别数 k 的选择

k 值需要预先指定，而在实际应用中最优的 k 值是不知道的。解决这个问题一个方法是尝试用不同的 k 值聚类，检验各自得到聚类结果的质量，推测最优的 k 值

(1) 一般地，类别数变小时，平均直径会增加；类别数变大超过某个值后，平均直径会不变；而这个值是最优的 k 值。实验时可以采用二分查找，快速找到最优的 k 值。

- 2-11-4 如何提升kmeans效率？

kmeans时间复杂度 $O(nkt)$ n , k , t 分别为样本数，聚类中心数，迭代轮次

(1) 我们可以使用kd树以及ball 树（数据结构）来提高k-means算法的效率。（KNN计算的优化）

(2) 并行计算

- 2-11-5 常用的距离衡量方法有哪些？他们都适用什么类型问题？

见 2-2-3

- 2-11-6 Kmeans对异常值是否敏感？为什么？

敏感，因为需要计算距离，使用传统的欧式距离度量方式。所以需要预处理离群点、数据归一化。

- 2-11-7 如何评估聚类效果？

知乎：<https://zhuanlan.zhihu.com/p/53840697>

(1) 纯度 (Purity)

我们把每个簇中最多的类作为这个簇所代表的类，然后计算正确分配的类的数量，然后除以 N 。

$$|\Omega, \mathbb{C}| = \frac{1}{N} \sum_k \max_j |\left\{ \omega_k \cap c_j \right\}|$$
 其中

$\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$ 是聚类结果的集合

ω_k 表示第 k 个聚类的集合； $\mathbb{C} = \{c_1, c_2, \dots, c_J\}$ 是原始分类的集合， c_j 表示第 j 个分类的集合。

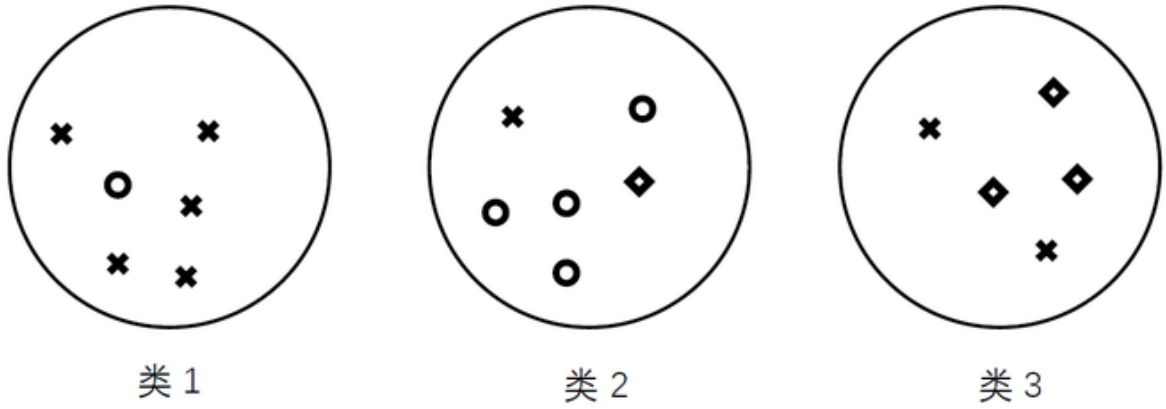


图 1 纯度作为一个聚类质量的外部评估标准。在这三个簇中，多数类以及多数类的成员数量分别为：叉（5 个，类 1）；圈（4 个，类 2）；菱形（3 个，类 3）。纯度为 $(1/17) \times (5 + 4 + 3) \approx 0.71$

purity优点：方便计算，值在0~1之间；

缺点：当簇的数量很多的时候，容易达到较高的纯度——特别是，如果每个文档都被分到独立的一个簇中，那么计算得到的纯度就会是1。因此，不能简单用纯度来衡量聚类质量与聚类数量之间的关系。

(2) 归一化互信息 (NMI, Normalized Mutual Information)

NMI越大，聚类效果越好

$$\text{NMI}(\Omega, C) = \frac{I(\Omega; C)}{(H(\Omega) + H(C))/2}$$

其中, I 表示互信息 (Mutual Information), H 为熵, 当 \log 取 2 为底时, 单位为 bit, 取 e 为底时单位为 nat。

$$\begin{aligned} I(\Omega; C) &= \sum_k \sum_j P(w_k \cap c_j) \log \frac{P(w_k \cap c_j)}{P(w_k)P(c_j)} \\ &= \sum_k \sum_j \frac{|w_k \cap c_j|}{N} \log \frac{N|w_k \cap c_j|}{|w_k||c_j|} \end{aligned}$$

其中, $P(w_k), P(c_j), P(w_k \cap c_j)$ 可以分别看作样本 (document) 属于聚类簇 w_k , 属于类别 c_j , 同时属于两者的概率。第二个等价式子则是由概率的极大似然估计推导而来。

$$\begin{aligned} H(\Omega) &= - \sum_k P(w_k) \log P(w_k) \\ &= - \sum_k \frac{|w_k|}{N} \log \frac{|w_k|}{N} \end{aligned}$$

互信息 $I(\Omega; C)$

表示给定类簇信息 C 的前提下, 类别信息 Ω 的增加量, 或者说其不确定度的减少量。直观地, 互信息还可以写出如下形式:

$$I(\Omega; C) = H(\Omega) - H(\Omega|C)$$

(3) 兰德指数 (RI, Rand Index) 能度量聚类过程中的假阳性和假阴性结果的惩罚

兰德指数 (Rand index, RI), 将聚类看成是一系列的决策过程,即对文档集上所有 $N(N-1)/2$ 个文档 (documents) 对进行决策。当且仅当两篇文档相似时,我们将它们归入同一簇中。

Positive:

- TP 将两篇相似文档归入一个簇 (同 - 同)
- TN 将两篇不相似的文档归入不同的簇 (不同 - 不同)

Negative:

- FP 将两篇不相似的文档归入同一簇 (不同 - 同)
- FN 将两篇相似的文档归入不同簇 (同 - 不同) (worse)

RI 则是计算「正确决策」的比率(精确率, accuracy).

$$RI = \frac{TP + TN}{TP + FP + TF + FN} = \frac{TP + TN}{C_N^2}$$

(4) 熵 (Entropy)

- ☐ 2-11-8 超参数类的个数k如何选取?

(1) 手肘法

尝试不同的K值, 并将不同K值所对应的损失函数化成折线, 横轴为K的取值, 纵轴为误差平方和所定义的损失函数。K值越大, 距离和越小。当K=K'时, 存在一个拐点, 像人的肘部。当 $K \in (1, K')$, 曲线急速下降; 当 $K > K'$, 曲线趋于平稳。手肘法认为拐点就是K的最佳值。

(2) Gap Statistic

手肘法是一个经验方法, 缺点是不够自动化。Gap Statistic方法的优点是, 不需要肉眼判断, 只需要找到最大Gap statistic对应的K即可。因此该方法适用于批量化作业。 $\text{Gap}(K) = E(\log D_k) - \log D_k$ 当分为K簇时, 对应的损失函数记为 D_k , $E(\log D_k)$ 是 $\log D_k$ 的期望, 通过蒙特卡洛模拟产生。 $\text{Gap}(K)$ 的物理含义是随机样本的损失与实际样本的损失之差。当 $\text{Gap}(K)$ 取最大时, 是样本的损失应该相对较小。

- ☐ 2-11-9 Kmeans有哪些优缺点? 是否了解过改进的模型, 举例说明?

优点

- (1) 对于大数据集, kmeans聚类算法相对是可伸缩和高效的, 它的计算复杂度是 $O(NKt)$ 接近线性, 其中N是样本数, K是聚类的簇数, t是迭代的轮数。
- (2) 尽管算法经常以局部最优解结束, 但一般情况下达到的局部最优已经可以满足聚类的需求

缺点

- (1) 受初值和离群点的影响, 每次的结果不稳定
- (2) 结果通常不是全局最优而是局部最优解

- (3) 无法很好地解决数据簇分布差别比较大的情况（比如一类是另一类样本数量的100倍）
- (4) 不太适用于离散分类
- (5) K值需要人工预先确定，且该值和真实的数据分布未必吻合【最大缺点】
- (6) 样本点只能被划分到单一的类中

如何对Kmeans进行调优

- (1) 数据归一化和离群点处理

Kmeans聚类本质上是一种基于欧式距离度量的数据划分方法，均值和方差大的维度对聚类结果产生决定性影响。未做归一化无法直接参与计算；离群点或少量噪声会对均值产生较大影响，导致中心偏移。

- (2) 合理选择K值

手肘法、Gap Statistic（不需要肉眼判断，只需要找到最大Gap statistic对应的K）

- (3) 采用核函数

核聚类。通过非线性映射，将输入空间中的数据点映射到高维的特征空间中，并在新的特征空间中进行聚类。非线性映射增加了数据点线性可分的概率，从而在经典的聚类算法失效的情况下，通过引入核函数可以达到更为准确的聚类效果

改进的模型

- (1) kmeans++（对初始值选择的改进）

原始的kmeans算法最开始随机选取数据集中K个点作为聚类中心，而kmeans++按照以下思想选取k个聚类中心：假设已经选取了n个初始聚类中心（ $0 < n < k$ ），则在选取第n+1个聚类中心时，距离当前n个聚类中心越远的点会有更高的概率被选为第n+1个聚类中心。在选取第一个聚类中心时同样通过随机的方法。这符合我们的直觉：聚类中心互相离得越远越好。后续步骤与kmeans一致。

- (2) ISODATA（K值不确定时）

ISODATA全称是 迭代自组织数据分析法。在kmeans算法中，聚类个数K值需要预先人为确定，并且在整个算法过程中无法更改。当遇到高维度、海量数据时，难以准确估计出K的大小。ISODATA的思想是：当属于某个类别的样本数过少时，把该类别去除（合并操作）；当属于某个类别的样本数过多、分散程度大时，把该类别分为两个子类别（分裂操作）。

缺点是需要制定的参数比较多，4个：预期聚类中心数目 K_0 、每个类所要求的最少样本数目 N_{\min} 、最大方差 σ 、两个聚类中心之间所允许最小距离 D_{\min} 。

- (3) 模糊C均值，高斯混合模型

样本不划分到单一的类中，即软聚类。

高斯混合模型：假设不同簇中的样本各自服从不同的高斯分布，由此得到的聚类算法称为高斯混合模型。在该假设下，每个单独的分模型都是标准高斯模型，其均值 μ_i 和方差 Σ_i 是待估计的参数。高斯混合模型还有一个参数 π_i ，可以理解为权重或者生成数据的概率。高斯混合模型是一个生成式模型。

$$p(x) = \sum_{i=1}^K \pi_i N(x|\mu_i, \Sigma_i)$$

相比Kmeans的优点：可以给出一个样本属于某类的概率是多少；不仅仅可以用于聚类，还可以用于概率密度的估计；并且可以用于生成新的样本点。

- ☐ 2-11-10 试试证明kmeans算法的收敛性

kmeans聚类属于启发式方法，不能保证收敛到全局最优，初始中心的选择会直接影响聚类结果。

类中心在聚类过程中会发生移动，但是往往不会移动太大，因为在每一步，样本被分到与其最近的中心的类中。

- ☐ 2-11-11 除了kmeans聚类算法之外，你还了解哪些聚类算法？简要说明原理

层次聚类（hierarchical clustering）假设类别之间存在层次结构，分为聚合（自下而上）和分裂（自上而下）两种方法。**聚合法**开始讲每个样本各自分到一个类；之后将相邻最近的两类合并（根据类间距离最小合并规则），建立一个新的类，重复此操作直到满足停止条件（类的个数达到阈值/类的直径超过阈值）；得到层次化的分类。**分裂法**开始讲所有样本分到一个类；之后将已有类中相距最远的样本分到两个新的类，重复此操作直到满足停止条件；得到层次化的分类。

层次聚类算法复杂度： $O(n^3m)$ 其中n是样本个数，m是样本维数。

k-means聚类是基于中心的聚类方法，通过迭代，将样本分到k个类别中，使得每个样本与其所属类的中心或均值最近；得到k个平坦的、非层次化的类别。

基于密度的方法DBSCAN

谱聚类

- ☐ kmeans初始化为什么要从数据中随机挑k个，可以生成k个随机点吗？

不可以；如果是用随机值，可能某个簇在第一轮就没有任何节点，无法继续计算

PCA降维

- ☐ 2-12-1 为什么要对数据进行降维？它能解决什么问题？

高维（多变量）数据，很难观察变量的样本区分能力，也很难观察样本之间的关系。降维是将样本集合中的样本从高维空间转换到低维空间。假设样本原本存在于低维空间，或近似存在与低维空间，通过降维则可以更好地表示样本数据的结构，即更好地表示样本之间的关系。降维有线性降维和非线性降维。

维度灾难

- ☐ 2-12-2 你是如何理解维度灾难？

特征数量超过一定值的时候，分类器的效果反而下降。原因：特征数过多，过拟合

- ☐ 2-12-3 PCA主成分分析思想是什么？

变量之间可能存在相关性，以致增加了分析的难度。考虑用少数不想管的变量来替代相关的变量，用来表示数据，并且要求能保留数据中的大部分信息。

PCA利用正交变换把线性相关变量表示的观测数据转换为少数几个由线性无关变量表示的数据，线性无关的变量称为主成分。PCA属于降维方法。

- ☐ 2-12-4 如何定义主成分？

协方差矩阵的特征向量

- ☐ 2-12-5 如何设计目标函数使得降维达到提取主成分的目的？

PCA目标函数：最大化投影方差。因为方差表示新变量的信息量大小。

- ☐ 2-12-6 PCA有哪些局限性？如何优化

(1) 无法进行非线性降维

通过核映射对PCA进行扩展得到核主成分分析（KPCA）

通过流形映射的降维方法，比如等距映射、局部线性嵌入（LLE）、拉普拉斯特征映射等

(2) 无监督的，算法没有考虑数据的标签，只是把把元数据映射到方差比较大的方向

有监督的降维方法：线性判别分析 LDA

- ☐ 2-12-7 线性判别分析和主成分分析在原理上有何异同？在目标函数上有何区别和联系？

PCA选择的是投影后数据方差最大的方向。由于PCA是无监督的，因此假设方差越大，信息量越多，用主成分来表示原始数据可以去除冗余的维度，达到降维。

LDA选择的是投影后类内方差最小，类间方差最大的方向。其用到了类别标签信息，为了找到数据中具有判别性的维度，使得原始数据在这些方向投影后，不同类别尽可能区分开。

3、深度学习

DNN

- ☐ 3-1-1 描述一下神经网络？推倒反向传播公式？

- ☐ 3-1-2 讲解一下dropout原理？

在神经网络前向传播的时候，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。

- ☐ 3-1-3 梯度消失和梯度膨胀的原因是什么？有什么方法可以缓解？

(1) 深度学习的网络层数太多，在进行反向传播时根据链式法则，要连乘每一层梯度值

(2) 每一层的梯度值是由，非线性函数的导数以及本层的权重相乘得到的，这样非线性的导数的大小和初始化权重的大小会直接影响是否发生梯度弥散或者梯度爆炸

注：任何网络都有可能发生梯度弥散或者梯度爆炸，这是深度学习的基本性质决定的，无法避免。

梯度消失（梯度弥散）的原因：

解决方法：

梯度爆炸的原因：

解决方法：

- ☐ 3-1-4 什么时候该用浅层神经网络，什么时候该选择深层网络
- ☐ 3-1-5 Sigmoid、Relu、Tanh激活函数都有哪些优缺点？

Sigmoid $f(x) = \frac{1}{1 + \exp(-x)}$ $f'(x) = f(x)(1-f(x))$
$$\begin{aligned} f'(z) &= \left(\frac{1}{1+e^{-z}}\right)' \\ &= \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\ &= \frac{1}{(1+e^{-z})} (1-\frac{1}{(1+e^{-z})}) = f(z)(1-f(z)) \end{aligned}$$

优点：

缺点：（1）需要计算指数，速度慢（2）会产生梯度消失问题

Tanh $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$f'(x) = 1 - (f(x))^2$ 优点：

缺点：（1）需要计算指数，速度慢（2）会产生梯度消失问题

Relu $f(x) = \max(x, 0)$
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$
 优点：（1）从计算的角度上，sigmoid和tanh都需要计算指数，复杂度高，而ReLU只需要一个阈值就可以得到激活值

（2）ReLU的非饱和性可以有效解决梯度消失的问题，提供相对宽的激活边界

（3）ReLU的单侧抑制提供了网络的稀疏表达能力（防止过拟合）

缺点：（1）训练过程中会导致神经元死亡的问题

缺点：（1）训练过程中会导致神经元死亡的问题

Leaky ReLU
$$f(x) = \begin{cases} x, & x > 0 \\ ax, & x \leq 0 \end{cases}$$

$f'(x) = \begin{cases} 1, & x > 0 \\ a, & x \leq 0 \end{cases}$ 优点：实现单侧抑制，又保留了部分附体度信息以致不完全消失

缺点：a值需要人工选择

- ☐ 3-1-6 写出常用激活函数的导数

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh 函数	$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU	$f(x) = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

- ☐ 3-1-7 训练模型的时候，是否可以把网络参数全部初始化为0？为什么

不可以；参数全部为0时，网络不同神经元的输出必然相同，相同输出则导致梯度更新完全一样，会使得更新后的参数仍然保持完全相同。从而使得模型无法训练。

- ☐ 3-1-8 Batchsize大小会如何影响收敛速度？

CNN

- ☐ 3-2-1 简述CNN的工作原理？

CNN利用了图像的三个性质：

- (1) 图像的pattern通常比整张图像小
- (2) 通用的patterns会出现在图像的不同区域
- (3) 对图像进行子采样并不影响图像的认识

CNN通过卷积层+pooling层不断堆积，从小的pattern开始不断识别到大的pattern，从而识别整张图像。

CNN适合处理什么问题

具有以上三个特性的问题

- ☐ 3-2-2 卷积核是什么？选择大卷积核和小卷积核有什么影响？
- ☐ 3-2-3 你在实际应用中如何设计卷积核？
- ☐ 3-2-4 为什么CNN具有平移不变性？

参数共享的物理意义是使得卷积层具有平移等变性。在卷积神经网络中，卷积核中的每一个元素将作用于每一次局部输入的特定位置上。

假如图像中有一只猫，无论它出现在图像中的任何位置，我们都应该将它识别为猫，也就是说神经网络的输出对于平移变换来说应当是等变的。

- ☐ 3-2-5 Pooling操作是什么？有几种？作用是什么？

将Pooling核覆盖区域中所有值的平均值（最大值）作为汇合结果

average-pooling, Max-polling, stochastic-polling（对输入数据中的元素按照一定概率大小随机选择，元素值大的activation被选中的概率大）

Pooling操作后的结果相比起输入减小了，是一种降采样操作。


pooling层的作用

1. **特征不变性（feature invariant）**。pooling操作使模型更关注是否存在某些特征而不是特征具体的位置。
 2. **数据降维**。降采样的操作使模型可以抽取更广范围的特征，同时减小了下一层输入大小，进而减少计算量和参数个数
- ☐ 3-2-6 为什么CNN需要pooling操作？

- 3-2-7 什么是batchnormalization? 它的原理是什么? 在CNN中如何使用?

为了解决内部协方差偏移 (ICS) 问题提出的。高层网络需要不断适应底层输出的分布, 导致

对于某一层某个神经元d维输入 $\mathbf{x} = (x^{(1)} \dots x^{(d)})$, 对每一个维度进行批标准化 $\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbf{E}[x^{(k)}]}{\sqrt{\operatorname{Var}[x^{(k)}]}}$

$\{\sqrt{\operatorname{Var}[x^{(k)}]}\}$  1567511107747

Input: Network N with trainable parameters Θ ;
subset of activations $\{x^{(k)}\}_{k=1}^K$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

- 1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
- 2: **for** $k = 1 \dots K$ **do**
- 3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. II)
- 4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
- 5: **end for**
- 6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
- 7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen parameters
- 8: **for** $k = 1 \dots K$ **do**
- 9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
- 10: Process multiple training mini-batches \mathcal{B} , each of size m , and average over them:

$$\mathbf{E}[x] \leftarrow \mathbf{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\text{Var}[x] \leftarrow \frac{m}{m-1} \mathbf{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$
- 11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma, \beta}(x)$ with

$$y = \frac{\gamma}{\sqrt{\text{Var}[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathbf{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \right)$$
- 12: **end for**

Algorithm 2: Training a Batch-Normalized Network

BN训练和测试的区别

先说结论: 并不是测试时的mean,var的计算方式与训练时不同, 而是测试时的mean,var在训练完成整个网络中就全部固定了。

由于在优化网络的时候, 我们一般采用的是batch梯度下降。所以在训练过程中, 只能计算当前batch样本上的mean和var。但是我们做的normalization是对于整个输入样本空间, 因此需要对每个batch的mean, var做指数加权平均来将batch上的mean和var近似成整个样本空间上的mean和var。

而在测试Inference过程中，一般不必要也不合适去计算测试时的batch的mean和var，比如测试仅对单样本输入进行测试时，这时去计算单样本输入的mean和var是完全没有意义的。因此会直接拿训练过程中对整个样本空间估算的mean和var直接来用。此时对于inference来说，BN就是一个线性变换。

- 3-2-8 卷积操作的本质特性包括稀疏交互和参数共享，具体解释这两种特性以其作用？

稀疏交互：每个神经元的只跟上一层的某些神经元连接（vs DNN全连接），用到较少参数

参数共享：同一层的不同神经元之间共享部分权重，用到比原来更少的参数

- 3-2-9 你是如何理解fine-tune？有什么技巧
- 3-2-10 怎么观察CNN每个神经元学到了什么

假设第k个filter是一个11 x 11 的矩阵（一个神经元），可以用以下系数来表示第k个filter被激活的程度 $a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$ 并通过梯度上升找到使 a^k 最大的x，该x表示的图像表示该filter对应的检测纹路。 $x^* = \mathop{\arg \max}_x a^k$

- resnet skip-connection

假如，我们在这个block的旁边加了一条“捷径”（如图5橙色箭头），也就是常说的“skip connection”。假设左边的上一层输入为x，虚线框的输出为f(x)，上下两条路线输出的激活值相加为h(x)，即 $h(x) = F(x) + x$ ，得出的h(x)再输入到下一层。

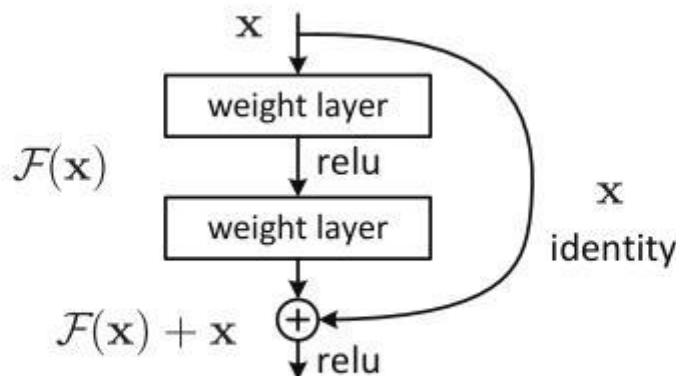


Figure 2. Residual learning: a building block.

图6

当进行后向传播时，右边来自深层网络传回来的梯度为1，经过一个加法门，橙色方向的梯度为 $dh(x)/dx=1$ ，蓝色方向的梯度也为1。这样，经过梯度传播后，现在传到前一层的梯度就变成了[1, 0.0001, 0.01]，多了一个“1”！正是由于多了这条捷径，来自深层的梯度能直接畅通无阻地通过，去到上一层，使得浅层的网络层参数等到有效的训练！

RNN

- 3-3-1 简述RNN模型原理，说说RNN适合解决什么类型问题？为什么

RNN能够很好地处理文本数据变长并且有序的输入序列。它模拟了人阅读一篇文章的顺序，从前到后阅读文章中的每一个单词，将前面阅读到的有用信息编码到状态变量中去，从而拥有一定的记忆能力，可以更好地理解之后的文本。

不定长；下一时刻的状态于当前输入以及当前状态有关

- ☐ 3-3-2 RNN和DNN有何异同？

相同点：一个长度为T的序列用 $x_{1:T}$

不同点：RNN的循环性，序列的每个时刻都执行相同的任务，每个时刻的输出依赖于当前时刻的输入和上一时刻的隐藏状态

- ☐ 3-3-3 RNN为什么有记忆功能？

RNN是包含循环的网络，将前面输入的有用信息编码到状态变量中去，从而拥有了一定的记忆功能。

- ☐ 3-3-4 长短期记忆网络LSTM是如何实现长短期记忆功能的？

LSTM可以对有价值的信息进行长期记忆。

与RNN不同的是，LSTM记忆单元 c 的转移不一定完全取决于激活函数计算得到的状态，还由输入门和遗忘门共同控制。

在一个训练好的LSTM模型中，当输入序列中没有重要信息时，遗忘门的值接近于1，输入门的值接近于0，表示过去的记忆被完整保存，而输入信息被放弃，从而实现长期记忆功能。

当输入序列中存在重要信息时，LSTM应把他存入记忆中，此时输入门接近于1；

当输入序列中存在重要信息且该信息意味着之前的记忆不再重要时，输入门的值接近1，遗忘门的值接近0。

- ☐ 3-3-5 长短期记忆网络LSTM各模块都使用什么激活函数，可以使用其他激活函数么？

百面p245。输入门、输出门、遗忘门使用sigmoid函数作为激活函数；在生成候选记忆时，使用双曲正切函数Tanh作为激活函数。

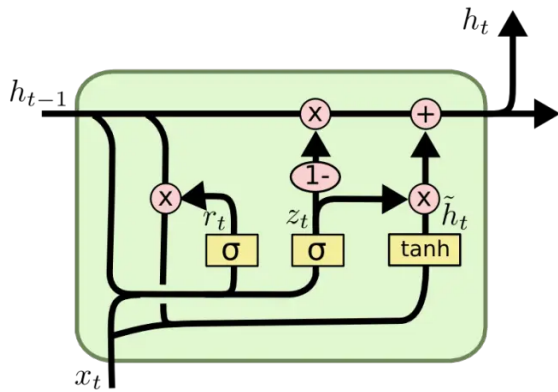
首先这两个激活函数都是饱和的，如果使用非饱和函数如ReLU，将难以实现门控的效果。

sigmoid作为门控信号的原因：sigmoid函数的输出在0~1之间，符合门控的物理定义。并且是饱和的，当输入较大或者较小时，其输出会非常接近1或0，从而保证该门的开或关。

Tanh用于生成候选记忆的原因：输出在 -1~1之间，这与大多数场景下特征分布式以0为中心相吻合；并且Tanh函数在输入为0附近相比sigmoid有更大的梯度，通常使模型收敛更快。

- ☐ 3-3-6 GRU和LSTM有何异同

GRU：（1）将遗忘门和输入门合成了一个单一的更新门，只有两个门：更新门、复位门。（2）GRU不再区分cell的状态 $\overrightarrow{\mathrm{C}}$ 和cell的输出 $\overrightarrow{\mathbf{h}}$ 。



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

reset gate r_t : 计算候选隐层 \tilde{h}_t 时用来控制需要保留多少之前的记忆 h_{t-1} , 比如如果 r_t 为0, 那么 \tilde{h}_t 只包含当前词的信息。

update gate z_t : 控制需要从前一时刻的隐藏层 h_{t-1} 中遗忘多少信息, 需要加入多少当前时刻的隐藏层信息 \tilde{h}_t , 最后得到 h_t

一般来说那些具有短距离依赖的单元reset gate比较活跃 (如果 r_t 为1, 而 z_t 为0 那么相当于变成了一个标准的RNN, 能处理短距离依赖), 具有长距离依赖的单元update gate比较活跃。

相同点

(1) 都会有门操作, 决定是否保留上时刻的状态, 和是否接收此时刻的外部输入, LSTM 是用遗忘门 (forget gate f_t) 和输入门 (input gate i_t) 来做到的, GRU 则是只用了一个更新门 (update gate z_t)

(2) 遗忘门或者更新门选择不重写 (overwritten) 内部的 memory, 那么网络就会一直记住之前的重要特征, 那么会对当前或者未来继续产生影响。缓解梯度消失。

不同点

(1) 首先就是 LSTM 有一个输出门来控制 memory content 的曝光程度 (exposure), 而 GRU 则是直接输出。

(2) 另一点是要更新的新 memory content 的来源也不同。 \tilde{h}_t 会通过重置门 (reset gate) 控制从 h_{t-1} 中得到信息的力度, 而 \tilde{c}_t 则没有, 而是直接输入 h_{t-1} 。

(3) 相同个数参数的情况下, GRU 会比 LSTM 稍好一些

• ☐ 3-3-7 什么是Seq2Seq模型? 该模型能解决什么类型问题?

seq2seq模型是将一个序列信号, 通过编码和解码生成一个新的序列信号, 输入和输出序列的长度实现并不知道。seq2seq的模型的核心思想由编码输入和解码输出两个环节构成。在经典的实现中, 编码器和解码器各由一个循环神经网络构成, 两个循环神经网络是共同训练的。

解决问题: 机器翻译、语音识别、自动对话

• ☐ 3-3-8 注意力机制是什么? Seq2Seq模型引入注意力机制主要解决什么问题?

编码-解码架构的主要缺点: 编码器RNN输出的上下文C的维度太小, 难以恰当的概括一个长的输入序列的完整信息。

◦ ☐ 注意力机制主要解决问题:

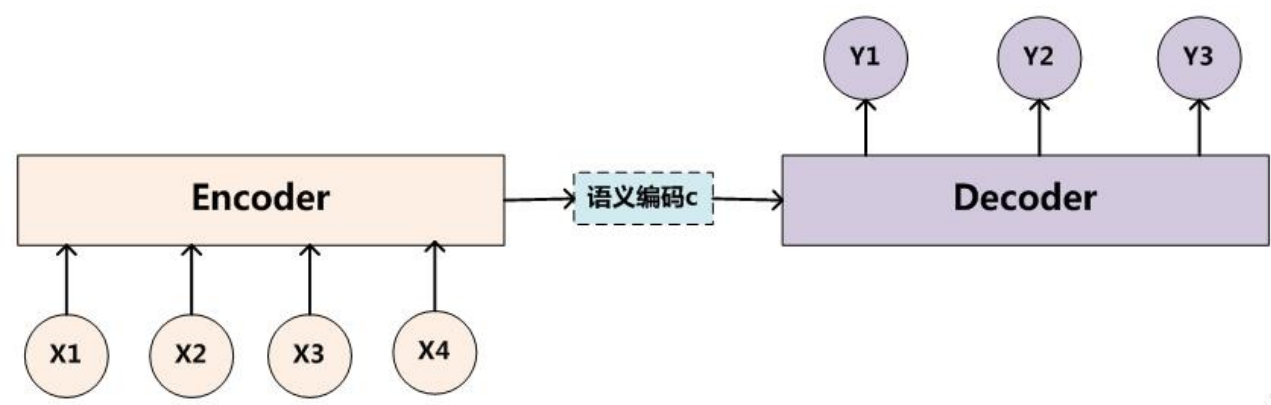
(1) 随着输入序列增长，模型性能发生显著下降。

因为编码时输入序列的全部信息压缩到了一个定长向量表示中。随着序列增长，句子越前面的词的信息丢失就越严重。

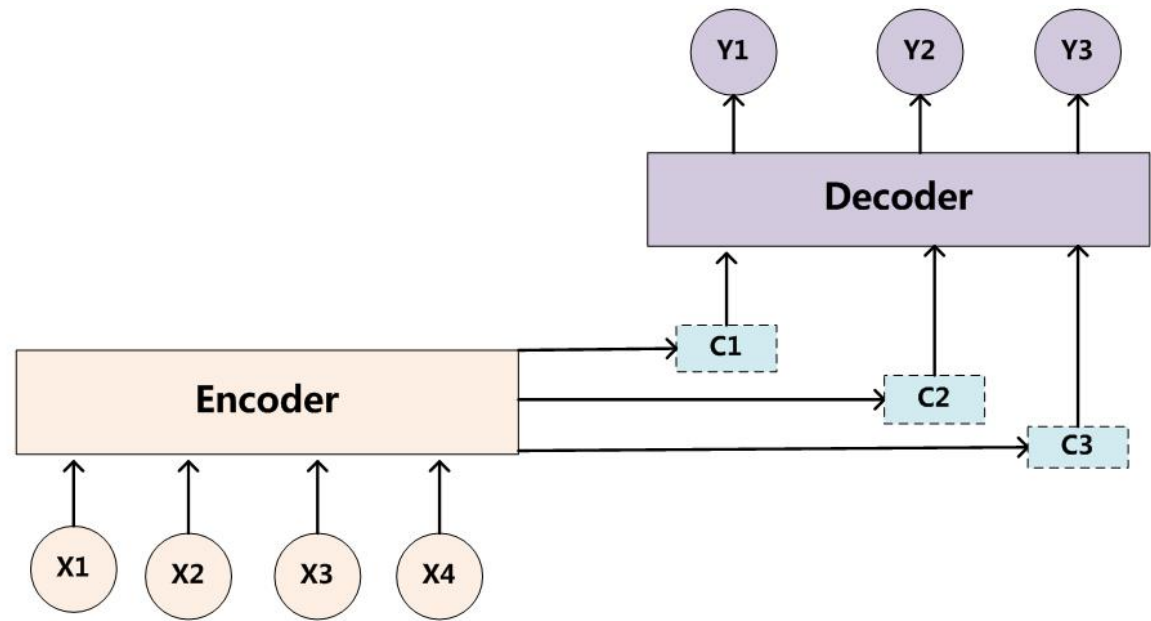
(2) seq2seq输出序列中，常常会损失部分输入序列的信息。

这是因为在解码时，当前词及对应的源语言词的上下文信息和位置信息在编解码过程中丢失了。

原编码-解码架构

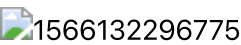


加入attention



attention 本身可以理解作为一种对齐关系，给出了模型输入、输出之间的对齐关系，解释了模型到底学到了什么知识。

- ☐ 注意力打分函数的计算方式



双线性模型

$$s(\mathbf{x}_i, \mathbf{q}) = \mathbf{x}_i^T \mathbf{W} \mathbf{q},$$

(8.5)

- ☐ attention的分类

global attention vs local attention

上述的 **attention** 机制中为了计算上下文向量 $\overrightarrow{\mathbf{c}}_i$ ，需要考虑 **encoder** 的所有隐向量（**global attention**）。当输入序列较长时（如一段话或一篇文章），计算效率较低。

local attention 在计算上下文向量 $\overrightarrow{\mathbf{c}}_i$ 时只需要考虑 **encoder** 的部分隐向量：首选预测 **encoder** 端对齐的位置 p_i ，然后基于位置 p_i 选择一个窗口来计算上下文向量 $\overrightarrow{\mathbf{c}}_i$ 。

self attention

传统的 **attention** 是基于 **encoder** 端和 **decoder** 端的隐向量来计算 **attention** 的，得到的是输入序列的每个 **input** 和输出序列的每个 **output** 之间的依赖关系。

self attention 计算三种 **attention**：

- 在 **encoder** 端计算自身的 **attention**，捕捉 **input** 之间的依赖关系。
- 在 **decoder** 端计算自身的 **attention**，捕捉 **output** 之间的依赖关系。
- 将 **encoder** 端得到的 **self attention** 加入到 **decoder** 端得到的 **attention** 中，捕捉输入序列的每个 **input** 和输出序列的每个 **output** 之间的依赖关系。

❑ RNN的长期依赖（Long-Term Dependencies）问题是什么？怎么解决

长期依赖问题是：随着输入序列的增长，模型的性能发生显著下降，RNN难以捕捉长距离输入之间的依赖。

从结构上来看，理论上RNN可以学习捕捉到长距离依赖，但是实践中使用BPTT算法学习的RNN并不能成功捕捉长距离的依赖关系，主要源于神经网络中的**梯度消失**。

解决方法：

(1) LSTM、GRU等模型加入门控机制，捕捉长期记忆，很大程度上弥补了梯度消失

(2) 残差结构

(2) 设计多个时间尺度的模型：在细粒度的时间尺度上处理近期信息、在粗粒度时间尺度上处理远期的信息。得到粗粒度时间尺度方法1跳跃链接：增加从远期的隐变量到当前隐变量的直接连接；2. 是删除连接：主动删除时间跨度为 1 的连接，并用更长的连接替换。

❑ RNN为什么会产生梯度消失或者梯度爆炸

主要由于权重矩阵 W 在不同时间步被重复使用，导致形成 W 的幂乘

1. 长期依赖的问题是深度学习中的一个主要挑战，其产生的根本问题是：经过许多阶段传播之后，梯度趋向于消失或者爆炸。

- 长期依赖的问题中，梯度消失占大部分情况，而梯度爆炸占少数情况。但是梯度爆炸一旦发生，就优化过程影响巨大。
- RNN** 涉及到许多相同函数的多次复合作用，每个时间步一次。这种复合作用可以导致极端的非线性行为。因此在**RNN** 中，长期依赖问题表现得尤为突出。

2. 考虑一个没有非线性、没有偏置非常简单的循环结构：

$\vec{\mathbf{h}}^{(t)} = \mathbf{W} \vec{\mathbf{h}}^{(t-1)}$ 。则有：

$$\begin{aligned}\vec{\mathbf{h}}^{(t)} &= \mathbf{W}^t \vec{\mathbf{h}}^{(0)} \\ \frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t-1)}} &= \mathbf{W}, \quad \frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(0)}} = \mathbf{W}^t \\ \nabla_{\vec{\mathbf{h}}^{(0)}} L &= \frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(0)}} \nabla_{\vec{\mathbf{h}}^{(t)}} L = \mathbf{W}^t \nabla_{\vec{\mathbf{h}}^{(t)}} L\end{aligned}$$

设 \mathbf{W} 可以正交分解时： $\mathbf{W} = \mathbf{Q} \mathbf{\Lambda}$

\mathbf{Q}^T 。其中 \mathbf{Q} 为正交矩阵， $\mathbf{\Lambda}$ 为特征值组成的三角阵。则：

$$\begin{aligned}\vec{\mathbf{h}}^{(t)} &= \mathbf{Q} \mathbf{\Lambda}^t \vec{\mathbf{h}}^{(0)} \\ \frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(0)}} &= \mathbf{Q} \mathbf{\Lambda}^t \mathbf{Q}^T \\ \nabla_{\vec{\mathbf{h}}^{(0)}} L &= \mathbf{Q} \mathbf{\Lambda}^t \nabla_{\vec{\mathbf{h}}^{(t)}} L\end{aligned}$$

■ 前向传播：

- 对于特征值的幅度不到 1 的特征值对应的 $\vec{\mathbf{h}}^{(0)}$ 的部分将随着 t 衰减到 0。
- 对于特征值的幅度大于 1 的特征值对应的 $\vec{\mathbf{h}}^{(0)}$ 的部分将随着 t 指数级增长。

■ 反向传播：

- 对于特征值幅度不到 1 的梯度的部分将随着 t 衰减到 0。
- 对于特征值幅度大于 1 的梯度的部分将随着 t 指数级增长。

3. 若考虑非线性和偏置，即： $\vec{\mathbf{h}}^{(t+1)} = \tanh$

$$\left(\vec{\mathbf{b}} + \mathbf{W} \vec{\mathbf{h}}^{(t)} + \mathbf{U} \vec{\mathbf{x}}^{(t+1)} \right)$$

■ 前向传播：

由于每一级的 $\vec{\mathbf{h}}$ 的幅度被 \tanh 函数限制在 $(-1, 1)$ 之间，因此前向传播并不会指数级增长。

这也是为什么 RNN 使用 \tanh 激活函数，而不使用 relu 的原因。

■ 反向传播：

由于隐状态的幅度被 $\tanh(\cdot)$ 函数限制在 $(-1, 1)$ 之间，因此 $\frac{\partial \left(1 - \left(\overrightarrow{\mathbf{h}}^{(t+1)}\right)^2\right)}{\partial \mathbf{W}}$ 对 \mathbf{W} 进行了一定程度上的缩小。
 $\overrightarrow{\mathbf{h}}^{(t+1)}$ 越大，结果越小。

- 如果 \mathbf{W} 的特征值经过这样的缩小之后，在每个时刻都远小于1（因为每个时刻缩小的比例会变化），则该梯度部分将衰减到 0。
- 如果 \mathbf{W} 的特征值经过这样的缩小之后，在每个时刻都远大于1，则该梯度部分将指数级增长。
- 如果 \mathbf{W} 的特征值经过这样的缩小之后，在不同的时刻有时候小于1有时候大于1（因为每个时刻缩小的比例会变化），则该梯度部分将比较平稳。

□ RNN如何解决梯度爆炸问题

梯度截断：对梯度值进行缩放，使得梯度的模不超过 η 。假设 g 是梯度向量， $|g| > \eta$ ，那么 $g = \frac{\eta g}{|g|}$

□ RNN如何解决梯度消失问题

(1) LSTM、GRU等模型加入门控机制，捕捉长期记忆，很大程度上弥补了梯度消失。

背后的思路是让路径的梯度乘积接近1

(2) 多时间尺度（增加跳跃连接、删除连接）、泄露单元（线性自连接单元）

(3) 引入残差结构

(4) RNN+初始化权重矩阵为单位矩阵

□ LSTM的结构，每个门的作用，计算公式

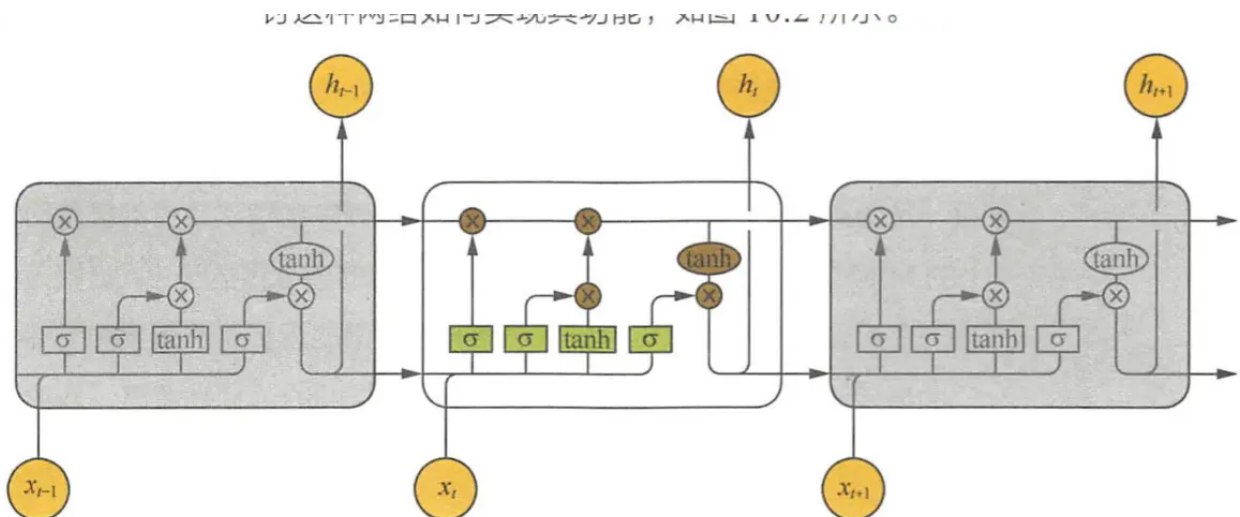
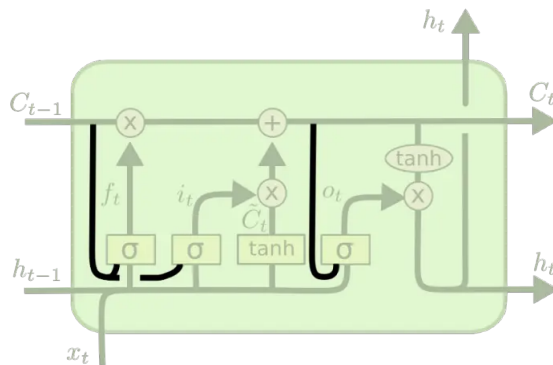


图 10.2 长短时记忆模型内部结构示意图

经典的LSTM中第 t 步计算公式为
$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1}) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

◦ ☐ LSTM变种有哪些

“**peephole connection**”：让门控层 也会接受细胞状态的输入。

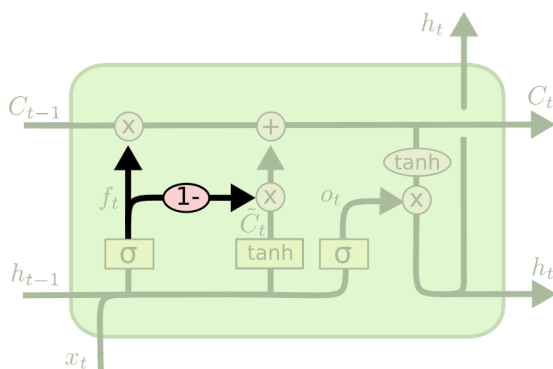


$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

coupled forget and input gates：将输入门和遗忘们耦合在一起，输入和遗忘是同步的。



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

GRU

◦ ☐ LSTM为什么可以解决梯度消失

将连乘关系转换为相加的线性关系

在LSTM中 $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ ，其中 c_{t-1} 是此前的信息， $\tilde{\mathbf{c}}_t$ 是当前即刻的新信息， c_t 是最终的信息。可以看到 c_t 和 c_{t-1} 此时是线性关系，不再是RNN中的连乘关系，梯度以线性在中间节点流动，因此可以保证很长时间的记忆。

进一步地，如果门控信号考虑bias，同时忽略输入变量 h_{t-1} 的作用，隐含层关系表示为：

$$c_j = \sigma(W^f X_j + b^f) c_{j-1} + \sigma(W^i X_j + b^i) \tilde{c}_j$$

于是，需要连乘的项表示为： $\frac{\partial c_j}{\partial c_{j-1}} = \sigma(W^f X_j + b^f)$ 该值范围在0~1之间。但是在实际参数更新中，可以通过控制bias比较大，使得该值接近于1；在这种情况下，即使通过很多次连乘的操作，梯度也不会消失，仍然可以保持“长距”连乘项的存在。即总可以通过选择合适的参数，在不发生梯度爆炸的情况下，找到合理的梯度方向来更新参数，而且这个方向可以充分地考虑远距离的隐含层信息的传播影响。

4、基础工具

Spark

- ☐ 4-1-1 什么是宽依赖，什么是窄依赖？哪些算子是宽依赖，哪些是窄依赖？

- ☐ 4-1-2 Transformation和action算子有什么区别？举例说明
- ☐ 4-1-3 讲解sparkshuffle原理和特性？ shuffle write 和 huffleread过程做些什么？
- ☐ 4-1-4 哪些spark算子会有shuffle？
- ☐ 4-1-5 讲解sparkschedule（任务调度）？
- ☐ 4-1-6 Sparkstage是如何划分的？
- ☐ 4-1-7 Sparkcache一定能提升计算性能么？说明原因？
- ☐ 4-1-8 Cache和persist有什么区别和联系？
- ☐ 4-1-9 RDD是弹性数据集，“弹性”体现在哪里呢？你觉得RDD有哪些缺陷？
- ☐ 4-1-10 当GC时间占比很大可能的原因有哪些？对应的优化方法是？
- ☐ 4-1-11 park中repartition和coalesce异同？coalesce什么时候效果更高，为什么
- ☐ 4-1-12 Groupbykey和reducebykey哪个性能更高，为什么？
- ☐ 4-1-13 你是如何理解caseclass的？
- ☐ 4-1-14 Scala里trait有什么功能，与class有何异同？什么时候用trait什么时候用class
- ☐ 4-1-15 Scala 语法中to 和 until有啥区别
- ☐ 4-1-16 讲解Scala伴生对象和伴生类

Xgboost

- ☐ 4-2-1 你选择使用xgboost的原因是什么？

xgboost的优点：

1. 速度非常快
2. XGBoost支持多种类型的基分类器，比如线性分类器
3. XGBoost采用了多种策略防止过拟合：（1）目标函数正则项相当于预剪枝（2）每轮迭代支持数据行采样（bagging），还有列采样
4. 对缺失值自动处理

- ☐ 4-2-2 Xgboost和GBDT有什么异同？

1. GBDT是机器学习算法，XGBoost是该算法的工程实现
2. 传统GBDT以CART作为基分类器，XGBoost还支持**线性分类器**，这个时候XGBoost相当于带L1和L2正则化项的Logistic回归（分类问题）或者线性回归（回归问题）。
3. 传统的GBDT只用了一阶导数信息（使用牛顿法的除外），而XGBoost对损失函数做了**二阶泰勒展开**。并且XGBoost支持自定义损失函数，只要损失函数一阶、二阶可导。
4. 在使用CART作为基分类器时，XGBoost的目标函数多了**正则项控制模型复杂度**，相当于**预剪枝**，使得学习出来的模型更加不容易过拟合。
5. 传统的GBDT在每轮迭代时使用全部数据，XGBoost则采用了与随机森林相似的策略，支持对数据进行采样（**行采样和列采样**）。
6. **对缺失值的处理**。传统的GBDT没有涉及对缺失值进行处理，XGBoost能够自动学习出缺失值的处理策略。
7. XGBoost工具**支持并行**。当然这个并行是在**特征的粒度**上，而非**tree粒度**，因为本质还是boosting算法。我们知道，决策树的学习最耗时的一个步骤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可能。在进行节点分裂时，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。
8. 可并行的近似直方图计算。

- ❑ 4-2-3 为什么xgboost训练会那么快，主要优化点是什么？

1. 当数据集大的时候使用**近似算法**：在特征分裂时，根据特征k的分布确定 S 个候选切分点。根据这些切分点把相应的样本放入对应的桶中，对每个桶的 G,H 进行累加，最后通过遍历所有的候选分裂点来找到最佳分裂点。我们对这么多个桶进行分支判断，显然比起对 n 个样本找分裂节点更快捷。
2. Block与并行。**分块并行**，针对的是寻找最优切分点过程中的排序部分。
3. CPU cache 命中优化。对于exact greedy算法中, 使用**缓存预取**。具体来说，对每个线程分配一个连续的buffer，读取梯度信息并存入Buffer中（这样就实现了非连续到连续的转化），然后再统计梯度信息。
4. Block预取、Block压缩、Block Sharding等

- ❑ 4-2-4 Xgboost是如何处理缺失值的？

XGBoost能对缺失值自动进行处理，其思想是**对于缺失值自动学习出它该被划分的方向**（左子树or右子树）：

注意，**上述的算法只遍历非缺失值**。划分的方向怎么学呢？很naive但是很有效的方法：

1. 让特征k的所有缺失值的都到右子树，然后和之前的一样，枚举划分点，计算最大的gain
2. 让特征k的所有缺失值的都到左子树，然后和之前的一样，枚举划分点，计算最大的gain

这样最后求出最大增益的同时，也知道了缺失值的样本应该往左边还是往右边。使用了该方法，相当于比传统方法多遍历了一次，但是它只在非缺失值的样本上进行迭代，因此其复杂度与非缺失值的样本成线性关系。

- ❑ 4-2-5 Xgboost和lightGBM有哪些异同？



不同点：

1. 由于在决策树在每一次选择节点特征的过程中，要遍历所有的属性的所有取值并选择一个较好的。**XGBoost**使用的是近似算法，**先对特征值进行排序Pre-sort**，然后根据二阶梯度进行分桶，能够更精确地找到数据分割点；但是复杂度较高。

LightGBM使用的是**直方图算法**，只需要将数据分割成不同的段即可，不需要进行预先的排序。占用内存更低，数据分隔的复杂度更低**。**

2. 决策树的生长策略

XGBoost使用的是Level-wise的树生长策略；LightGBM使用的是leaf-wise的生长策略。

在XGBoost中，树是按层生长的，称为**Level-wise tree growth**，同一层的所有节点都做分裂，最后剪枝，如下图所示：



Level-wise过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，也好控制模型复杂度，不容易过拟合。但实际上Level-wise是一种低效的算法，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销，因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。

而LightGBM采用的是**Leaf-wise tree growth**:

Leaf-wise则是一种更为高效的策略，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，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。

3. 并行策略

XGBoost的并行主要集中在特征并行上，而LightGBM的并行策略分特征并行，数据并行以及投票并行。

- [4-2-6 Xgboost为什么要使用泰勒展开式，解决什么问题？](#)

简短来说，就是为了统一损失函数求导的形式以支持自定义损失函数。

作者原回答

因为这样做使得我们可以很清楚地理解整个目标是什么，并且一步一步推导出如何进行树的学习。这一个抽象的形式对于实现机器学习工具也是非常有帮助的。传统的GBDT可能大家可以理解如优化平方残差，但是这样一个形式包含所有可以求导的目标函数。也就是说有了这个形式，我们写出来的代码可以用来求解包括回归，分类和排序的各种问题，正式的推导可以使得机器学习的工具更加一般。

实际上使用二阶泰勒展开是为了xgboost能够【自定义loss function】，如果按照最小二乘法的损失函数直接推导，同样能够得到xgboost最终的推导式子：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

二阶泰勒展开实际不是 \approx 最小二乘法，平方损失函数的二阶泰勒展开=最小二乘法。但作者为何想用二阶泰勒展开呢，一种猜想为了xgboost库的可扩展性，因为任何损失函数只要二阶可导即能【复用】文章中的的任何推导。而且泰勒的本质是尽量去模仿一个函数，我猜二阶泰勒展开已经足以近似大量损失函数了，典型的还有基于分类的对数似然损失函数。嘿，这样同一套代码就能完成回归或者分类了，而不是每次都推导一番，重写训练代码。

xgboost使用了一阶和二阶偏导，二阶导数有利于梯度下降的更快更准。使用泰勒展开取得函数做自变量的二阶导数形式，可以在不选定损失函数具体形式的情况下，仅仅依靠输入数据的值就可以进行叶子分裂优化计算，本质上也就把损失函数的选取和模型算法优化/参数选择分开了。这种去耦合增加了xgboost的适用性，使得它按需选取损失函数，可以用于分类，也可以用于回归。

- [4-2-7 Xgboost是如何寻找最优特征的？](#)

通常采用贪心法，每次尝试分裂一个叶节点，计算分裂后的增益，选增益最大的。这个方法在之前的决策树算法中大量被使用。而增益的计算方式比如ID3的信息增益，C4.5的信息增益率，CART的Gini系数等。那XGBoost呢？

XGBoost使用下面的公式计算增益： $\text{Gain} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} \right] \text{左子树分数} + \frac{1}{2} \left[\frac{G_R^2}{H_R + \lambda} \right] \text{右子树分数} - \frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \text{分裂前分数} - \gamma \text{新叶节点复杂度}$ 分裂后 - 分裂前

的分数。**Gain**值越大，说明分裂后能使目标函数减少越多，就越好。在寻找最优分裂点时，遍历得到增益最大的点作为分裂点。注意分裂不一定会使情况变好，因为有一个引入新叶子的惩罚项 γ ，优化这个目标相当于进行树的剪枝。当引入的分裂带来的增益小于一个阈值的时候，不进行分裂操作。

- 4-2-8 Xgboost的缺点？

1. 空间开销大。需要保存数据的特征值。XGBoost采用Block结构，存储指向样本的索引，需要消耗两倍的内存。
2. 时间开销大（相对于lightGBM而言）。在寻找最优切分点时，要对每个特征都进行排序，还要对每个特征的每个值都进行了遍历，并计算增益。
3. 对Cache不友好（相对于lightGBM而言）。使用Block块预排序后，特征对梯度的访问是按照索引来获取的，是一种随机访问，而不同特征访问顺序也不一样，容易造成命中率低的问题。同时，在每一层长树的时候，需要随机访问一个行索引到叶子索引的数组，并且不同特征访问的顺序也不一样，也会造成较大的Cachemiss。

Tensorflow

- 4-3-1 使用tensorflow实现逻辑回归，并介绍其计算图
- 4-3-2 sparse_softmax_cross_entropy_with_logits和softmax_cross_entropy_with_logits有何异同？
- 4-3-3 使用tensorflow过程中，常见调试哪些参数？举例说明
- 4-3-4 Tensorflow梯度更新是同步还是异步，有什么好处？
- 4-3-5 讲解一下TFRecords
- 4-3-6 tensorflow如何使用如何实现超大文件训练？
- 4-3-7 如何读取或者加载图片数据？

5、推荐系统

- 5-1-1 你是如何选择正负样本？如何处理样本不均衡的情况？

样本不均衡处理：

- 1) 上采样和子采样；
- 2) 修改权重（修改损失函数）；
- 3) 集成方法：bagging，类似随机森林、自助采样；
- 4) 多任务联合学习；

- 5-1-2 如何设计推荐场景的特征体系？举例说明
- 5-1-3 你是如何建立用户模型来理解用户，获取用户兴趣的？
- 5-1-4 你是如何选择适合该场景的推荐模型？讲讲你的思考过程
- 5-1-5 你是如何理解当前流行的召回->粗排->精排的推荐架构？这种架构有什么优缺点？什么场景适用使用，什么场景不适合？
- 5-1-6 如何解决热度穿透的问题？（因为item热度非常高，导致ctr类特征主导排序，缺少个性化的情况）
- 5-1-7 用户冷启动你是如何处理的？
- 5-1-8 新内容你是如何处理的？
- 5-1-9 你们使用的召回算法有哪些？如何能保证足够的召回率？

- ❑ 5-1-10 实时数据和离线数据如何融合？工程上是怎样实现？如何避免实时数据置信度不高带来的偏差问题？
- ❑ 5-1-11 你们是如何平衡不同优化目标的问题？比如：时长、互动等

multi-task learning

- ❑ 5-1-12 不同类型内容推荐时候，如何平衡不同类型内容，比如图文、视频；或者不同分类
- ❑ 5-1-13 如何保证线上线下数据一致性？工程上是如何实现？
- ❑ 5-1-14 离线训练效果好，但是上线效果不明显或在变差可能是什么问题？如何解决？
- ❑ 5-1-15 在实际业务中，出现badcase,你是如何快速反查问题的？举例说明
- ❑ 5-1-16 使用ctr预估的方式来做精排，会不会出现相似内容大量聚集？原因是什么？你是如何解决的？
- ❑ 5-1-17 你了解有多少种相关推荐算法？各有什么优缺点

	优点	缺点	适用场景
item-based CF	注重个性化，发掘长尾商品	推荐结果过于热门，需要惩罚	item更新频率低的（如购物网站、图书、电影网站）
user-based CF	社交网络		item更新频率高（新闻）

|

- ❑ 5-1-18 深度学习可以应用到推荐问题上解决哪些问题？为什么比传统机器学习要好？
- ❑ 为什么CTR任务中损失函数一般选择交叉熵作为损失函数（分类）而不用平方损失函数（回归）

一方面点击-不点击本身是一个二分类的任务，分类任务采用交叉熵作为损失函数是常规做法。另一方面考虑参数更新。

深度学习同样有许多损失函数可供选择，如平方差函数（Mean Squared Error），绝对平方差函数（Mean Absolute Error），交叉熵函数（Cross Entropy）等。而在理论与实践，我们发现Cross Entropy相比于在线性模型中表现比较好的平方差函数有着比较明显的优势。其主要原因是在深度学习通过反向传递更新W和b的同时，激活函数Sigmoid的导数在取大部分值时会落入左、右两个饱和区间，造成参数的更新非常缓慢。具体的推导公式如下：

一般的MSE被定义为：

$$C = \frac{1}{2}(a - y)^2$$

其中 y 是我们期望的输出， a 为神经元的实际输出 $a=\sigma(Wx+b)$ 。由于深度学习反向传递的机制，权值 W 与偏移量 b 的修正公式被定义为：

$$\frac{\partial C}{\partial W} = (a - y)\sigma'(a)x^T$$

$$\frac{\partial C}{\partial b} = (a - y)\sigma'(a)$$

因为Sigmoid函数的性质，导致 $\sigma'(z)$ 在 z 取大部分值时会造成饱和现象。

Cross Entropy的公式为：

$$H(y, a) = - \sum y_i \log(a_i)$$

如果有多个样本，则整个样本集的平均交叉熵为：

$$H(y, a) = -\frac{1}{n} \sum \sum y_{i,n} \log(a_{i,n})$$

其中 n 表示样本编号， i 表示类别编号。如果用于Logistic分类，则上式可以简化成：

$$H(y, a) = -\frac{1}{n} \sum y \log(a) + (1 - y) \log(1 - a)$$

与平方损失函数相比，交叉熵函数有个非常好的特质：

$$H(y, a) = \frac{1}{n} \sum (a_n - y_n) = \frac{1}{n} \sum (\sigma(z_n) - y_n)$$

可以看到，由于没有了 σ' 这一项，这样一来在更新 w 和 b 就不会受到饱和性的影响。当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。

- ☐ 深度学习模型有什么缺点？

单纯的DNN模型对于CTR的提升并不明显。而且单独的DNN模型本身也有一些瓶颈例如，当用户本身是非活跃用户时，由于其自身与Item之间的交互比较少，导致得到的特征向量会非常稀疏，而深度学习模型在处理这种情况时有可能会有过度的泛化，导致推荐与该用户本身相关较少的Item（FM模型也存在该问题）。因此，我们将广泛线性模型与深度学习模型相结合，同时又包含了一些组合特征，以便更好的抓住Item-Feature-Label三者之间的共性关系。我们希望在宽深度模型中的宽线性部分可以利用交叉特征去有效地记忆稀疏特征之间的相互作用，而在深层神经网络部分通过挖掘特征之间的相互作用，提升模型之间的泛化能力。

二、数学相关

6、概率论和统计学

伯努利分布（0-1分布）

单个二值随机变量的分布， x 的取值为0或者1；随机变量为1的概率 p ，为0的概率是 $1-p$ $p(\{x\} = x) = p^x(1-p)^{1-x}$

- ☐ 6-1-1 说说你是怎样理解信息熵的？
- ☐ 6-1-2 能否从数据原理熵解析信息熵可以表示随机变量的不确定性？
- ☐ 6-1-3 怎样的模型是最大熵模型？它有什么优点
- ☐ 6-1-4 什么是Beta分布？它与二项分布有什么关系？

参考知乎回答：<https://www.zhihu.com/question/30269898>

beta分布可以看做一个概率的概率分布，当你不知道一个东西的具体概率是多少时，它可以给出了所有概率出现的可能性大小。它是对二项分布中成功概率 p 的概率分布的描述。它的形式如下：

beta分布与二项分布是共轭先验的（[Conjugate prior](#)）。所谓共轭先验就是先验分布是beta分布，而后验分布同样是beta分布。以棒球运动员击球为例，结果很简单：

$$\text{Beta}(\alpha_0 + \text{hits}, \beta_0 + \text{misses})$$

- ☐ 6-1-5 什么是泊松分布？它与二项分布有什么关系？
- ☐ 6-1-6 什么是t分布？他与正态分布有什么关系？
- ☐ 6-1-7 什么是多项式分布？具体说明？
- ☐ 6-1-8 参数估计有哪些方法？

极大似然估计MLE

最大后验概率估计MAP

期望极大化EM

EM 算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含参数（EM 算法的 E 步），接着基于观察数据和猜想的隐含参数一起来极大化对数似然，求解我们的模型参数（EM 算法的 M 步）。由于我们之前的隐含参数是猜想的，所以此时得到的模型参数一般还不是我们想要的结果。我们基于当前得到的模型参数，继续猜想隐含参数（EM 算法的 E 步），然后继续极大化对数似然，求解我们的模型参数（EM 算法的 M 步）。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

一个最直观了解 EM 算法思路的是 K-Means 算法。在 K-Means 聚类时，每个聚类簇的质心是隐含数据。我们会假设 K 个初始化质心，即 EM 算法的 E 步；然后计算得到每个样本最近的质心，并把样本聚类到最近的这个质心，即 EM 算法的 M 步。重复这个 E 步和 M 步，直到质心不再变化为止，这样就完成了 K-Means 聚类。

EM 算法和极大似然估计的前提是一样的，都要假设数据总体的分布，如果不知道数据分布，是无法使用 EM 算法的。

EM 算法是通过不断求解下界的极大化逼近求解对数似然函数极大化的算法

- ☐ [6-1-9 点估计和区间估计都是什么？](#)
- ☐ [6-1-10 讲解一下极大似然估计，以及适用场景？](#)

在统计学中，常常使用极大似然估计法来估计参数。即找到一组参数，使得在这组参数下，我们数据的似然度（概率）最大。（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，即‘模型已定，参数未知’）

极大似然估计的前提一定是要假设数据总体的分布，如果不知道数据分布，是无法使用极大似然估计的

求极大似然估计的步骤

- (1) 写出似然函数；
- (2) 对似然函数取对数，并整理；
- (3) 求导数，令导数为 0，得到似然方程；
- (4) 解似然方程，得到的参数。

应用场景

- (1) 回归问题中的极小化平方和
- (2) 分类问题中极小化交叉熵

- ☐ [6-1-11 讲解一下最大后验概率估计，以及适用场景？](#)

极大似然估计中采样需满足一个重要的假设，就是所有的采样都是独立同分布的。

那么我们就知道了极大似然估计的核心关键就是对于一些情况，样本太多，无法得出分布的参数值，可以采样小样本后，利用极大似然估计获取假设中分布的参数。

极大似然估计就是经验风险最小化的一个例子。当模型是条件概率分布、损失函数是对数损失函数时，经验风险最小化就等价于极大似然估计。

最大后验概率是计算给定数据条件下模型的条件概率，即后验概率。使用模型的先验分布是贝叶斯学习的特点。

- 频率学派和贝叶斯学派什么区别？

频率学派

频率学派是上帝视角，认为频率是固定的，事件在多次重复实验中趋于一个稳定的值 p ，那么这个值就是该事件的概率。

他们认为模型参数是个定值，希望通过类似解方程组的方式从数据中求得该未知数。这就是频率学派使用的参数估计方法-极大似然估计（MLE），这种方法往往在大数据量的情况下可以很好的还原模型的真实情况。

贝叶斯派

他们认为世界是不确定的，因获取的信息不同而异。假设对世界先有一个预先的估计，然后通过获取的信息来不断调整之前的预估计。他们认为模型参数源自某种潜在分布，希望从数据中推知该分布。对于数据的观测方式不同或者假设不同，那么推知的该参数也会因此而存在差异。这就是贝叶斯派视角下用来估计参数的常用方法-最大后验概率估计（MAP）

这种方法在先验假设比较靠谱的情况下效果显著，随着数据量的增加，先验假设对于模型参数的主导作用会逐渐削弱，相反真实的数据样例会大大占据有利地位。极端情况下，比如把先验假设去掉，或者假设先验满足均匀分布的话，那她和极大似然估计就如出一辙了。

- MLE和MAP的联系

MLE与MAP的联系

-在介绍经验风险与结构风险最小化的时候以具体的逻辑回归（LR）与概率矩阵分解（PMF）模型来介绍MLE和MAP，接下里从宏观的角度，不局限于具体的某个模型来推导MLE与MAP。

-假设数据 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ 是满足独立同分布（i.i.d.）的一组抽样 $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ ，接下来就利用两种参数估计方法来求解。

- MLE对参数 θ 的估计方法可以如下：

$$\begin{aligned}\hat{\theta}_{\text{MLE}} &= \arg \max P(\mathbf{X}; \theta) \\ &= \arg \max P(\mathbf{x}_1; \theta) P(\mathbf{x}_2; \theta) \cdots P(\mathbf{x}_n; \theta) \\ &= \arg \max \log \prod_{i=1}^n P(\mathbf{x}_i; \theta) \\ &= \arg \max \sum_{i=1}^n \log P(\mathbf{x}_i; \theta) \\ &= \arg \min - \sum_{i=1}^n \log P(\mathbf{x}_i; \theta)\end{aligned}$$

知乎 @张小磊

- MAP对 θ 的估计方法可以如下推导：

$$\begin{aligned}\hat{\theta}_{\text{MAP}} &= \arg \max P(\theta|X) \\ &= \arg \min -\log P(\theta|X) \\ &= \arg \min -\log P(X|\theta) - \log P(\theta) + \log P(X) \\ &= \arg \min -\log P(X|\theta) - \log P(\theta) \quad \text{知乎 @张小磊}\end{aligned}$$

***所以MAP和MLE在优化时的不同就是在于增加了一个先验项 $-\log P(\theta)$ 。

***通过以上的分析可以大致给出他们之间的联系： $MAP(\theta) \approx MLE(\theta) + P(\theta)$ 。

- ☐ 大数定理和中心极限定理

7、最优化问题

- ☐ 7-1-1 什么是梯度？

梯度的方向是函数值增加最快的方向，梯度的相反方向是函数值减小的最快的方向。

- ☐ 为什么梯度的负方向是局部下降最快方向？

对 $f(x+v)$ 在 x 处进行泰勒一阶展开 $f(x+v) \approx f(x) + \nabla f(x)^T v$ 这里有 $f(x) - f(x+v) \approx -\nabla f(x)^T v$ 其中 $\nabla f(x)^T$ 和 v 均为向量， $-\nabla f(x)^T v$ 就是两个向量进行内积，而向量进行内积的最大值就是两者共线的时候，也就是 v 的方向和 $-\nabla f(x)^T$ 方向相同时，内积最大。该内积值代表了最大的下降量。

- ☐ 7-1-2 梯度下降找到的一定是下降最快的方法？

梯度下降法并不一定是全局下降最快的方向，它只是目标函数在当前的点的切平面（当然高维问题不能叫平面）上下降最快的方向。在practical implementation中，牛顿方向（考虑海森矩阵）才一般被认为是下降最快的方向，可以达到superlinear的收敛速度。

- ☐ 7-1-3 牛顿法和梯度法有什么区别？

梯度下降法： $\overrightarrow{\mathbf{x}}_{k+1} = \overrightarrow{\mathbf{x}}_k - \epsilon \overrightarrow{\mathbf{g}}$

牛顿法： $\overrightarrow{\mathbf{x}}_{k+1} = \overrightarrow{\mathbf{x}}_k - \mathbf{H}^{-1} \overrightarrow{\mathbf{g}}$

1. 梯度法对目标函数进行一阶泰勒展开，梯度就是目标函数的一阶信息；
2. 牛顿法对目标函数进行二阶泰勒展开，Hessian矩阵就是目标函数的二阶信息。
3. 牛顿法的收敛速度一般要远快于梯度法，但是在高维情况下Hessian矩阵求逆的计算复杂度很大，而且当目标函数非凸时，牛顿法有可能会收敛到鞍点。
4. 因为梯度法旨在朝下坡移动，而牛顿法目标是寻找梯度为0的点。
5. 位于一个极小值点附近时，牛顿法比梯度下降法能更快地到达极小值点。

如果在一个鞍点附近，牛顿法效果很差，因为牛顿法会主动跳入鞍点。而梯度下降法此时效果较好（除非负梯度的方向刚好指向了鞍点）。

6. 梯度下降法中，每一次 $\overrightarrow{\mathbf{x}}$ 增加的方向一定是梯度相反的方向 $-\epsilon_k \nabla_{\mathbf{x}} f(\mathbf{x}_k)$ 。增加的幅度由 ϵ_k 决定，若跨度过大容易引发震荡。

而牛顿法中，每一次 $\overrightarrow{\mathbf{x}}$ 增加的方向是梯度增速最大的反方向 $-\mathbf{H}^{-1} \nabla f(\mathbf{x}_k)$ （它通常情况下与梯度不共线）。增加的幅度已经包含在 \mathbf{H}^{-1} 中（也可以乘以学习率作为幅度的系数）。

7-1-4 什么是拟牛顿法？

在牛顿法的迭代中，需要计算海森矩阵的逆矩阵 \mathbf{H}^{-1} ，这一计算比较复杂。可以考虑用一个 $n \times n$ 阶矩阵 $\mathbf{G}_k = \mathbf{G}(\overrightarrow{\mathbf{x}}^k)$ 来近似代替。

如果选择 \mathbf{G}_k 作为 \mathbf{H}^{-1} 的近似时， \mathbf{G}_k 同样要满足两个条件：

- \mathbf{G}_k 必须是正定的。
- \mathbf{G}_k 满足下面的拟牛顿条件： $\mathbf{G}_{k+1} \overrightarrow{\mathbf{y}}_k = \mathbf{y}_k$

因为 \mathbf{G}_0 是给定的初始化条件，所以下标从 $k+1$ 开始。

按照拟牛顿条件选择 \mathbf{G}_k 作为 \mathbf{H}^{-1} 的近似或者选择 \mathbf{B}_k 作为 \mathbf{H} 的近似的算法称为拟牛顿法

按照拟牛顿条件，在每次迭代中可以选择更新矩阵 $\mathbf{G}_{k+1} = \mathbf{G}_k + \Delta \mathbf{G}_k$

• []

7-1-5 讲解什么是拉格朗日乘子法、对偶问题、kkt条件？

凸优化问题

拉格朗日乘子法

对偶问题

将极小极大的原问题转为极大极小的对偶问题；通常对偶问题更好求解，可以通过求解对偶问题而得到原始问题的解，进而确定分离超平面和决策函数（SVM）。

KKT条件

7-1-6 是否所有的优化问题都可以转化为对偶问题？

对所有实数域上的优化问题都有其对偶问题

7-1-7 讲解SMO (Sequential Minimal Optimization) 算法基本思想？

SMO（序列最小优化算法）是一种启发式算法，是支持向量机学习的一种快速算法，其特点是不断地将原二次规划问题分解为只有两个变量的二次规划子问题，并对子问题进行解析求解，直到所有变量满足KKT条件为止，这时这个最优化问题的解就得到了。这样通过启发式的方法得到原二次规划问题

的最优解。因为子问题有解析解，所以每次计算子问题都很快，虽然计算子问题次数很多，但在总体上还是高效的。

整个SMO算法包括两部分：（1）求解两个变量二次规划的解析方法（2）选择变量的启发式方法

- ☐ 7-1-8 为什么深度学习不用二阶优化？

深度学习的目标函数复杂，非凸；目标函数非凸时，牛顿法有可能会收敛到鞍点

- ☐ 7-1-9 讲解SGD, ftrl, Adagrad, Adadelata, Adam, Adamax, Nadam优化算法以及他们的联系和优缺点
- ☐ 7-1-10 为什么batch size大，训练速度快or为什么mini-batch比SGD快？

(1) 从矩阵计算角度

mini-batch不同的输入可以拼成一个矩阵，和W计算矩阵相乘可以并行地计算，比SGD一个一个计算快。对于GPU，矩阵相乘可以并行地处理，速度快。

- ☐ 7-1-11 7-1-11 为什么不把batch size设得特别大

(1) GPU内存限制

(2) 性能差；容易卡在局部极小值