Contents

[基本数学和算法 2](#_Toc173155051)

[《机器学习》周志华 4](#_Toc173155052)

[《数据挖掘导论》 4](#_Toc173155053)

[思想 4](#_Toc173155054)

[数据 7](#_Toc173155055)

[数据类型,质量 7](#_Toc173155056)

[预处理 9](#_Toc173155057)

[数据探索 10](#_Toc173155058)

[汇总并可视化 10](#_Toc173155059)

[相似性和相异性的度量 11](#_Toc173155060)

[特征降维和选择 12](#_Toc173155061)

[特征降维 12](#_Toc173155062)

[特征选择与稀疏学习 14](#_Toc173155063)

[模型评估与选择 15](#_Toc173155064)

[划分数据：训练集 + 验证集 + 测试集 15](#_Toc173155065)

[性能度量 17](#_Toc173155066)

[如何比较学习器？ 19](#_Toc173155067)

[监督学习 20](#_Toc173155068)

[线性模型 (具有很好的可解释性) 20](#_Toc173155069)

[线性判别分析LDA = Linear Discriminant Analysis 21](#_Toc173155070)

[决策树 21](#_Toc173155071)

[神经网络 24](#_Toc173155072)

[支撑向量机SVM和支撑向量回归SVR 28](#_Toc173155073)

[Bayes分类器 28](#_Toc173155074)

[集成学习ensemble learning （multi-classifier system, committee-based learning） 30](#_Toc173155075)

[无监督学习 33](#_Toc173155076)

[聚类 33](#_Toc173155077)

[异常检测 36](#_Toc173155078)

[关联分析 36](#_Toc173155079)

[半监督学习 37](#_Toc173155080)

[Active Learning 37](#_Toc173155081)

[概率图模型 40](#_Toc173155082)

[统计语言模型 40](#_Toc173155083)

[Bayes network (也称为belief network) 41](#_Toc173155084)

[隐马尔可夫模型 41](#_Toc173155085)

[实际问题 42](#_Toc173155086)

[多类问题 42](#_Toc173155087)

[不平衡类问题 43](#_Toc173155088)

[在线学习（增量学习 incremental learning） 45](#_Toc173155089)

[Scikit Learn 46](#_Toc173155090)

[Split dataset: StratifiedGroupKFold (preferred) 46](#_Toc173155091)

[Preprocess data 49](#_Toc173155092)

[Estimator (classification, regression, cluster, transformer) 50](#_Toc173155093)

[Supervise-learning: Regression 52](#_Toc173155094)

[Supervise-learning: Classification 66](#_Toc173155095)

[Unsupervise-learning: Cluster 79](#_Toc173155096)

[Feature agglomeration 81](#_Toc173155097)

[Pipeline 82](#_Toc173155098)

[Tuning the hyper-parameters of an estimator 83](#_Toc173155099)

[Model Selection and evaluation 87](#_Toc173155100)

[Application: 87](#_Toc173155101)

[Kaggle 竞赛 87](#_Toc173155102)

[Working With Text Data 87](#_Toc173155103)

[practice 89](#_Toc173155104)

[clean data, benchmark 89](#_Toc173155105)

[数据降维与可视化 90](#_Toc173155106)

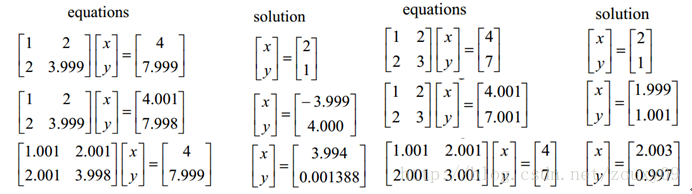
[imbalance 91](#_Toc173155107)

[Feature selection 93](#_Toc173155108)

# 基本数学和算法

**病态ill-condition**

假设我们有个方程组AX=b，我们需要求解X。如果A或者b稍微的改变，会使得X的解发生很大的改变，那么这个方程组系统就是ill-condition的，反之就是well-condition的



左边方程组系统是ill-conditioned病态的:

第一行假设是我们的AX=b，

第二行我们稍微改变下b，得到的x和没改变前的差别很大

第三行我们稍微改变下系数矩阵A，可以看到结果的变化也很大。

换句话来说，这个系统的解对系数矩阵A或者b太敏感了。又因为**一般我们的系数矩阵A和b是从实验数据里面估计得到的，所以它是存在误差的，如果我们的系统对这个误差是可以容忍的就还好，但系统对这个误差太敏感了，以至于我们的解的误差更大，那这个解就太不靠谱了。**

右边是well-condition系统

对于一个ill-condition的系统，我的输入稍微改变下，输出就发生很大的改变，这不好啊，这表明我们的系统不能实用啊。你想想看，例如对于一个回归问题y=f(x)，我们是用训练样本x去训练模型f，使得y尽量输出我们期待的值，例如0。那假如我们遇到一个样本x’，这个样本和训练样本x差别很小，面对他，系统本应该输出和上面的y差不多的值的，例如0.00001，最后却给我输出了一个0.9999，这很明显不对呀。就好像，你很熟悉的一个人脸上长了个青春痘，你就不认识他了，那你大脑就太差劲了

如何量化ill-condition的可信度？condition number

condition number衡量的是输入发生微小变化的时候，输出会发生多大的变化。也就是系统对微小变化的敏感度。condition number值小的就是well-conditioned的，大的就是ill-conditioned的。

如果方阵A是非奇异的，那么A的condition number定义为：

http://img.blog.csdn.net/20140504122518843?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvem91eHkwOQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

《概率与统计》

参数估计

已知某个随机样本满足某种概率分布，但是其中具体的参数不清楚，参数估计就是通过若干次试验，观察其结果，利用结果推出参数的大概值

**最大似然估计**

建立在参数估计的思想上：已知某个参数能使这类样本出现的概率最大

解释：从分布是p(x|θ)的总体样本中抽取到m个样本的概率，也就是样本集X中各个样本的联合概率

http://img.my.csdn.net/uploads/201301/24/1359003923_8916.jpg

这个函数反映的是**在不同的参数θ取值下，取得当前这个样本集的可能性**，因此称为参数θ相对于样本集X的**似然函数（likehood function）**

最大似然估计你可以把它看作是一个反推。多数情况下我们是根据已知条件来推算结果，而**最大似然估计是已经知道了结果，然后寻求使该结果出现的可能性最大的条件，以此作为估计值。**

例子：

身高分布估计

在学校所有男生（身高）中，独立地按照概率密度p(x|θ)抽取100个（身高），组成样本集X，如何通过样本集X估计未知参数θ。假定概率密度p(x|θ)是高斯分布N(u,∂)，则未知参数是θ=[u, ∂]T

**EM算法**

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

例如：在学校所有人（身高）中，独立地抽取200个（身高），组成样本集，**如何通过样本集X估计：高斯混合模型**（一是这个人是男的还是女的？二是男生和女生对应的身高的高斯分布的参数是多少？）

先有鸡还是先有蛋: 两种高斯分布的人混在一块了，我们又不知道哪些人属于第一个高斯分布，哪些属于第二个，所以就没法估计这两个分布的参数。反过来，只有当我们对这两个分布的参数作出了准确的估计的时候，才能知道到底哪些人属于第一个分布，那些人属于第二个分布

EM算法的基本思想：为了解决这个你依赖我，我依赖你的循环依赖问题，总得有一方要先打破僵局，说，不管了，我先随便整一个值出来，看你怎么变，然后我再根据你的变化调整我的变化，然后如此迭代着不断互相推导，最终就会收敛到一个解。

EM算法：假设估计A和B两参数，在开始状态下二者都是未知的，但如果知道A信息就可以得到B信息，反过来知道B也就得到A。可以考虑首先赋予A某种初值，以此得到B的估计值，然后从B的当前值出发，重新估计A的取值，这个过程一直持续到收敛为止。

EM = Expectation Maximization，如身高问题？

**Expectation阶段**：先随便猜一下男生（身高）的正态分布的参数：如均值和方差是多少。例如男生的均值是1米7，方差是0.1米（当然了，刚开始肯定没那么准），然后计算出每个人更可能属于第一个还是第二个正态分布中的（例如，这个人的身高是1米8，那很明显，他最大可能属于男生的那个分布）。

**Maximization**：有了每个人的归属，或者说我们已经大概地按上面的方法将这200个人分为男生和女生两部分，我们就可以根据之前说的最大似然那样，通过这些被大概分为男生的n个人来重新估计第一个分布的参数，女生的那个分布同样方法重新估计。当我们更新了这两个分布的时候，每一个属于这两个分布的概率又变了，那么我们就再需要调整E步……如此往复，直到参数基本不再发生变化为止。

咋知道新的参数的估计就比原来的好?

数学推导，可以证明

# 《机器学习》周志华

# 《数据挖掘导论》

# 思想

在计算机系统中，“经验”通常以数据形式存在，因此，机器学习所研究的主要内容是关于在计算机上从数据中产生“模型”的算法，即学习算法

模型：泛指从数据中学得的结果

模式：局部性结果（如一条规则）

学得模型对应了关于数据的某处潜在的规律，因此亦称为“假设”

机器学习的目标：使学得的模型能很好地适用于”新样本”

**尽管训练集通常只是样本空间的一个很小的采样，我们仍希望它能很好地反映出样本空间的特性，否则就很难期望在训练集上学得的模型能在整个样本空间上都工作得很好**。通常假设样本空间中总体样本服从一个未知“分布”D，我们获得的每个样本都是独立地从这个分布上采样获得的，即独立同分布i.i.d

**学习过程看作一个在所有假设组成的空间中进行搜索的过程，搜索目标是找到与训练集“匹配”的假设，假设的表示一旦确定，假设空间及其规模大小就确定了**。如我们的假设空间由形如“(色泽=?) and (根蒂=?) and (敲声=?)的可能取值所形成的假设组成。可以有许多策略对这个假设空间进行搜索，搜索过程中可以不断删除与正例不一致的假设、和与反倒一致的假设，最终将会获得与训练集一致（即对所有训练样本能够进行正确判断）的假设。但学习过程是基于有限样本训练集进行的，因此，可能有多个假设与训练集一致，应该采用哪一些模型（假设）呢？由机器学习算法在学习过程中对某种类型假设的偏好决定，即归纳偏好(inductive bias)。

**任何一个有效的机器学习算法必有其归纳偏好**。比如若认为相似的样本应有相似的输出（例如，在各种属性上都很相像的西瓜，成熟程度应该比较接近），则对应的学习算法可能偏好比较”平滑“的曲线，而不是比较”崎岖“的曲线。再比如Occam’s razor原则，若有多个假设与观察一致，则选最简单的那个。

归纳偏好对应了学习算法本身所做出的关于”什么样的模型更好“的假设，在具体的现实问题中，这个假设是否成立，即算法的归纳偏好是否与问题本身匹配，大多数时候直接决定了算法能否取得好的性能。

若所有”问题“出现的机会相同，或所有问题同等重要，那么无论学习算法a多聪明，学习算法b多笨拙，它们的期望性能是相同的，这就是”没有免费的午餐“定理。然后实现情形并不是这样，很多时候，我们只关注自己正在试图解决的问题（例如某个具体应用任务）。所以**要谈论算法的相对优劣，必须要针对具体的学习问题**，在某些问题上表现好的学习算法，在另一些问题上却可能不尽如人意，学习算法自身的归纳偏好与问题是否相配，往往会起到决定性的作用。

符号主义人工智能：手动编写明确规则。适合用来解决定义明确的逻辑问题，比如下国际象棋，但它难以给出明确规则来解决更复杂、更模糊的问题，比如图像分类、语音识别或自然语言翻译等。

基于规则的系统很脆弱，比如手写数字识别，每当遇到一个新的手写数字不符合精心设计的规则时，你都不得不添加新的数据变换和新的规则，还要考虑它们与之前每条规则之间的相互作用。

机器学习：不用编写规则，而是将数据和对应答案喂给机器，机器自动找到任务自动化的规则。与统计学不同，机器学习经常要处理复杂的大型数据集（比如包含百万张图片的数据集，每张图片又包含数万像素），用经典的统计分析（比如Bayes）来处理这种数据集是不切实际的。

深度学习：是机器学习的一个分支领域，是从数据中学习表示的一种新方法，强调从连续的层中学习，这些层对应于越来越有意义的表示。深度学习之“深度”并不是说这种方法能够获取更深层次的理解，而是指一系列连续的表示层。数据模型所包含的层数被称为该模型的深度。

可以将深度神经网络看作多级信息蒸馏(information distillation)过程：信息穿过连续的过滤器，其纯度越来越高，对任务的帮助越来越大。

通过将复杂、抽象的表示拆解为多个中间层来学习这些表示，每个中间层仅仅是前一层表示的简单变换

机械学习：没有进行真正的学习，仅是在进行信息存储与检索

**归纳学习：从样例中学习，即从训练样例中归纳出学习结果**

= 符号主义学习 （产生明确的概念，如决策树，基于逻辑学习）

连接主义学习 （产生的是黑箱模型，如神经网络，深度网络）

统计学习 （如SVM 以及更一般的核方法）

数据挖掘：从海量数据中发掘知识 = 数据库+机器学习+统计学

**分类与回归**

分类任务的输入数据是样例的集合，用元组(X, y)表示，其中X是属性的集合，y是样例类标号。类标号必须是离散属性。

回归也是一种预测建模任务，但目标属性是连续

分类技术非常适合预测或描述二元或标称类型的数据集，对于序数分类（如把人分类为高收入，中等收入或低收入组），分类技术不太有效，因为分类技术不考虑隐含在目标类中的序关系。

分类任务是通过学习得到一个目标函数f,把每个属性集X映射到一个预先定义的类标号y. 目标函数也称为分类模型

分类法：根据输入数据集建立分类模型的系统方法。如**决策树，基于规则，神经网络，支持向量机和贝叶斯**。这些技术都使用学习算法确定分类模型，该模型能很好地拟合输入数据中类标号和属性集之间的联系。学习算法很到的模型不仅要很好地拟合输入数据，还要能够正确地预测未知样本的类标号。

应用场景：根据电子邮件的标题和内容检查出垃圾邮件；根据核磁共振扫描的结果区分肿瘤是恶性的还是良性的？

竞争型学习的“可塑性-稳定性窘境”

可塑性指神经网络要有学习新知识的能力，稳定性则指神经网络在学习新知识时要保持对旧知识的记忆。

抛开理解数据的人和数据所处的领域而简单地使用数据分析技术是不可行的

数据挖掘技术可以用来支持广泛的商务智能应用，如顾客分析、定向营销、工作流管理、商店分布和欺诈检测等

**数据挖掘**技术用来探查大型数据库，发现先前未知的有用模式，预测未来观测结果

**信息检索**：使用数据库管理系统查找个别的记录，或通过因特网的搜索引擎查找特定的web页面。它们主要依赖传统的计算机科学技术和数据的明显特征来创建索引结构，从而有效地组织和检索信息

输入**数据存储**：可以以各种形式存储如平展文件、电子数据表或关系表，并且可以驻留在集中的数据存储库中，或分布在多个站点上。

**数据预处理**：融合来自多个数据源的数据，清洗数据以消除噪声或重复的观测值，选择与当前数据挖掘任务相关的记录和特征

**可视化**：从各种不同的视角探查数据和数据挖掘结果。在后处理阶段，还能使用统计度量或假设检验，删除虚假的数据挖掘结果

**数据挖掘的难点：**

可伸缩：处理海量数据集，算法必须是可伸缩的。许多数据挖掘算法使用特殊的搜索策略处理指数级搜索问题

高维性：成百上千属性的数据集

异种数据和复杂数据

**数据挖掘与传统统计方法的比较？**

传统的统计方法基于一种假设检验模式，即提出一种假设，设计实验来收集数据，然后针对假设分析数据。当前的数据分析任务常常需要产生和评估数千种假设，因此需要自动地产生和评估假设。

数据挖掘所分析的数据集通常不是精心设计的实验的结果，并且它们通常代表数据的时机性样本（opportunistic sample），而不是随机样本(random sample).

数据挖掘来自如下领域思想：1统计学的抽样、估计和假设检验。2人工智能、模式识别和机器学习的搜索算法、建模技术和学习理论。3最优化、进化计算、信息论、信号处理、可视化和信息检索

**数据挖掘任务：**

1预测任务

以说明变量函数的方式为目标变量建立模型，有两类预测建模任务：分类（用于预测离散的目标变量）和回归（用于预测连续的目标变量）。目标都是训练一个模型，使目标变量预测值与实际值之间的误差达到最小。常见的应用：检查结果判断病人是否患有某种疾病。

2描述任务

其目标是导出概括数据中潜在联系的模式（相关、趋势、聚类、轨迹和异常）。本质上，描述性数据挖掘任务通常是探查性的，并且常常需要后处理技术验证和解释结果。

# 数据

## 数据类型,质量

**数据类型**

决定我们应使用何种工具和技术来分析数据。知道属性的类型是重要的，因为它告诉我们测量值的哪些性质与属性的基本性质一致，从而使得我们可以避免诸如计算雇员的平均ID这样的愚蠢行为

不同的属性类型

分类的(定性的)

标称 区分对象(=, ~=) 一对一变换

序数 确定对象的序(>, <) 值的保序变换，即新值=f(旧值)，f是单调函数

数值的（定量的）

区间 值之间的差(+, -) 新值=a\*旧值+b

比率 差和比率(\*, /) 新值=a\*旧值

数据集特性：维度，稀疏性和分辨率（常常可以在不同的分辨率下得到数据，并且在不同的分辨率下数据的性质也不同，即数据的模式也依赖于分辨率，如果分辨率太高，模式可能看不出，或者掩埋在噪声中；如果分辨率太低，模式可能不出现）

**数据质量**

如存在噪声和离群点，数据遗漏、不一致或重复，数据有偏差或者不能代表它应该描述的现象或总体情况

数据挖掘常常不能“在数据源头控制质量。相比之下，统计学的实验设计或调查往往其数据质量都达到了一定的要求。由于无法避免数据质量问题，因此数据挖掘着眼于两个方面：1)数据质量问题的检测和纠正. 2)使用可以容忍低质量数据的算法。

测量和数据收集问题：

1) 离群点

在某种意义上具有不同于数据集中其他大部分数据对象的特征的数据对象，或是相对于该属性的典型值来说不寻常的属性值。Outlier可以是合法的数据对象或值。因此不像噪声，离群点本身有时是人们感兴趣的对象

2) 遗漏值

处理遗漏值的策略：a) 删除数据对象或属性. b) 估计遗漏值。如果属性是连续的，则可以使用最近邻的平均属性值；如果属性是分类的，则可以取最近邻中最常出现的属性. c) 在分析时忽略遗漏值。如果某对的一个对象或两个对象都有某些属性有遗漏值，则可以仅使用没有遗漏值的属性来计算相似性.

3) 不一致的值.

4) 重复数据

5) 测量误差和数据收集错误.

6) 噪声和伪像

噪声通常用于包含时间或空间分量的数据。在这些情况下，常常可以使用信号或图像处理技术降低噪声，从而帮助发现可能”淹没在噪声中“的模式（信号）

伪像：数据错误可能是更确定性现象的结果，即数据的某种确定性失真。如一组照片在同一地方出现条纹

7) 精度、偏倚和准确率

数据集 = 数据对象的集合

数据对象 = {属性：值}

记录数据

如：数据表病人信息；事务数据如购物蓝信息；数据矩阵如特征空间；文档-词矩阵

序列数据 = 记录数据 + 时间信息（或空间信息）

**时间自相关**：分析时间数据时，如果两个测量的时间很接近，则这些测量的值通常非常相似

**空间自相关**：物理上靠近的对象趋向于在其他方面也相似。如地球上相互靠近的两个点通常具有相近的气温和降水量

## 预处理

原始数据必须加以处理才能适合于分析。如将连续值属性如长度转换成具有离散的分类值的属性如短中长；如数据集属性的数目常常需要减少，因为属性较少时许多技术用起来更加有效。

**聚集**

将多个对象合并成单个对象，聚集是删除属性（如商品类型）的过程，或者是压缩特定属性不同值个数的过程

好处：1)数据归约导致较小数据集需要较少的内存和处理时间. 2)通过高层而不是低层数据视图，聚集起到了范围或标度转换的作用. 3对象或属性群的行为通常比单个对象或属性的行为更加稳定

缺点：丢失有趣的细节

**抽样**

统计学使用抽样是因为得到感兴趣的整个数据集的费用太高、太费时间；数据挖掘使用抽样是因为处理所有的数据的费用太高、太费时间。

有效抽样：样本是代表性的，前提是它近似地具有与原数据集相同的感兴趣的性质。如果数据对象的均值是感兴趣的性质，而样本具有近似于原数据集的均值，则样本就是有代表性的。我们所能做的最好的抽样方案就是选择一个确保以很高的概率得到有代表性的样本

抽样方法

无放回抽样和有放回抽样：当样本与数据集相比相对较小时，两种方法产生的样本差别不大。但是对于分析，有放回抽样较为简单，因为在抽样过程中，每个对象被选中的概率保持不变

分层抽样：从预先指定的组开始抽样。在最简单的情况下，尽管抽样的大小不同，但是从每组抽取的对象个数相同。另一种变形是从每一组抽取的对象数量正比于该组的大小。

渐进抽样（自适应抽样）：从一个小样本开始，然后增加样本容量直至得到足够容量的样本。尽管这种技术不需要在开始就确定正确的样本容量，但是需要评估样本的方法，确定它是否足够大。尽管预测模型的准确率随样本容量增加，但是在某一点准确率的增加趋于确定

**维归约**

好处：1) 维归约可以删除不相关的特征并降低噪声，一部分是因为维灾难。2) 维归约可以使模型更容易理解，因为模型可能只涉及较小的属性。3) 数据也可通过观察属性或对三元组属性达到可视化4使用维归约降低了数据挖掘算法的时间和内存需求

维灾难：随着数据维度的增加，许多数据分析变得非常困难。特别是随着维度增加，数据在它所占据的空间中越来越稀疏。对于分类，这可能意味着没有足够的数据对象来创建模型，将所有可能的对象可靠地指派到一个类。对于聚类，点之间的密度和距离的定义失去了意义

最常用的方法：线性代数技术，将数据由高维空间投影到低维空间。如主成分分析PCA: 找出新的属性（主成分），这些属性是原属性的线性组合，是相互正交的，并且捕获了数据的最大变差；如奇异值分解

**特征子集选择**

看起来这种方法可能丢失信息，但是在存在冗余或不相关特征的时候，情况并非如此。冗余特征重复了包含在一个或多个其他属性中的许多或所有信息。不相关特征包含对于手头的数据挖掘任务几乎完全没用的信息。冗余和不相关的特征可能降低分类的准确率，影响所发现的聚类的质量

**特征创建**

特征提取：对数据进行处理，获得一些较高层次的特征。最常使用的特征提取技术都是高度针对具体领域

映射数据到新的空间：使用一种完全不同的视角挖掘数据可能揭示出重要和有趣的特征。如果有大量周期模式，并且存在大量噪声，则很难检测这些模式。尽管如此，通过对该时间序列实施傅里叶变换，将它转换成频率信息明显的表示，就能检测到这些模式

特征构造：使用专家的意见构造特征

**离散化和二元化**

非监督离散化: 用于分类的离散化方法之间的根本区别在于使用类信息还是不使用类信息。如果不使用类信息，则常使用一些相对简单的方法，如等宽，等频

监督离散化: 如熵entropy,区间的熵是区间纯度的度量。如果一个区间只包含一个类的值（该区间非常纯），则其熵为0并且不影响总熵。如果一个区间中的值类出现的频率相等（该区间心可能不纯），则其熵最大。

**变量变换**

在统计学中，变量变换（特别是平方根、对数和倒数变换）常用来将不具有高斯（正态）分布的数据变换成具有高斯（正态）分布的数据。

使用变量变换时需要小心，因为它们改变了数据的特性。

标准化或规范化：目标是使整个值的集合具有特定的性质。统计学中术语规范化可能与使变量正态（高斯）的变换相混淆。对变量标准化是为了避免具有较大值域的变量左右计算结果。均值和标准差受离群点的影响很大，因此通常需要修改用中位数取代均值，用绝对标准差取代标准差.

**离散属性数值化**

若属性值间存在序order关系，可通过连续化将其转化为连续值，例如值属性“身高”的取值{高，矮}可转化为{1.0, 0.0}，三值属性”高度“的取值{高，中，低}可转化为{1.0, 0.5, 0.0};

若属性值间不存在序关系，假定有k个属性值，则通常转化为k维向量。例如属性”瓜类“的取值{西瓜，南瓜，黄瓜}可转化为(0, 0, 1), (0, 1, 0), (1, 0, 0)

## 数据探索

### 汇总并可视化

频率：frequency(vi) = 具有属性值vi的对象数/m

众数: 最高频率值

百分位数

位置度量：均值和中位数

均值对离群值很敏感，对于包含离群值的数据，中位数可以更稳健地提供值集中间的估计

散布度量：极差和方差 (表明属性值是否散布很宽，或是否相对集中在均值附近

极差：range(x) = max(x)-min(x)

标准差：与均值差平方和除以(m-1) = 方差的平方根

因为方差用均值计算，因此它也对离群值敏感。

对于多元数据，数据的散布用协方差矩阵表示。协方差矩阵的对角线上是属性的方差。

两个属性的协方差是两个属性一起变化并依赖于变量大小的度量。协方差的值接近

**可视化**

数据对象、它们的属性，以及数据对象之间的联系要转换成诸如点、线、形状和颜色等图形元素。

1. Stem and leaf plot
2. Histogram
3. 2D historgram
4. Box plot
5. Pie chart
6. Scatter plot 散布图的作用：图形化显示两个属性之间的关系。当类标给出时，可以使用散布图考察两个属性将类分开的程度。
7. Contour plot （等高线图）
8. Surface plot （曲面图）

**可视化高维数据**

1. 图像
2. 平等坐标系

记录用线表示，每个属性值对应坐标轴上的点

缺点：图中模式的检测可能取决于坐标轴的序

联机分析处理(OLAP)

在不同的维上或不同的属性值上聚集数据

多维数据分析的关键目标是观察聚集量，如总和或平均值

### 相似性和相异性的度量

邻近度：表示相似性或相异性，由于两个对象之间的邻近度是两个对象对应属性之间的邻近度的函数

相似度：[0, 1]

相异度：>0

相关性：[-1, 1]

如果邻近度度量原来在区间[0, ]上取值，则需要使用非线性变换，d’ = d/(d+1)， 注意logistic函数就是ea/(ea + 1)

任意单调减函数都可以用来将相异度转换到相似度（或相反）

**简单属性的相异度**

标称属性：

序数属性：

, 值映射到整数0~n-1,其中n是值的个数

区间或比率的属性：

d =|x-y|

数据对象之间的相异度

Minkowski distance, 见page42

**记录之间的相似性度量**

对称二元属性：简单匹配系数simple Matching Coefficient

SMC = 值匹配的属性个数/属性个数

SMC可以在一个仅包含是非题的测验中用来发现回答问题相似的学生

**非对称二元属性**：Jaccard系数，忽略0-0度量，因为稀疏性大多0

J = 匹配的个数/不涉及0-0匹配的属性个数

多元属性：余弦相似度

余弦相似度实际上是x和y之间夹角（余弦）的度量，如果余弦相似度为1,则x and y之间夹角是0.余弦相似不考虑两个数据对象的量值。

**记录之间的相关性度量**

Person’s correlation:

连续变量：Corr(x,y) = covariance(x, y) / (standard\_deviation(x) \* standard\_deviation(y))

二元变量：

is defrieved from Pearson’s chi-squared test

如果相关度为0,则两个数据对象的属性之间不存在线性关系。然而仍然可能存在非线性关系。

**邻近度计算**

若属性不同尺度，但属性之间独立，只需要对变量标准化，然后计算欧几里得距离

若属性不同尺度，且属性之间相关，需要计算Mahalanobis距离

# 特征降维和选择

## 特征降维

维数灾难：在高维情形下出现的数据样本稀疏、距离计算困难等问题，是所有机器学习方法共同面临的严重障碍

降维：即通过某种数学变换将原始高维属性空间转变为一个低维“子空间”

为什么能进行降维？

因为在很多时候，人们观测或收集到的数据样本虽是高维的，但与学习任务密切相关的也许仅是某个低维分布，即高维空间中的一个低维“嵌入”embedding

**多维缩放multiple dimensional scaling**：若要求原始空间中样本之间的距离在低维空间中得以保持，具体算法见《机器学习》周志华p229

对降维效果的评估，通常是比较降维前后学习器的性能，若性能有所提高则认为降维起到了作用，若将维数降至二维或三维，则可通过可视化技术来直观地判断降维效果

**PCA：**使所有样本的投影尽可能分开，则需最大化投影点的方差。

PCA是一种线性投影，新空间的轴与最大方差的方向对齐

新的特征是现有特征的线性组合（即旋转）

输入：样本集，低维空间维数d’

算法过程：

1. 对所有样本进行中心化
2. 计算样本的协方差矩阵
3. 对协方差矩阵做特征值分解
4. 取最大的d’个特征值所对应的特征向量

输出：投影矩阵=（特征向量1, 特征向量2, …）

实践中常通过对X进行奇异值分解来代替协方差矩阵的特征值分解，主成分是X的右奇异

如何确定降维后低维空间的维数？

通常是由用户事先指定 （比如交叉验证选取较好的d’值）

还可从重构的角度设置一个重构阈值，降维后的特征值总和比例

PCA仅需保留降维后的特征向量和样本的均值向量，最小的d-d’特征向量被舍弃，舍弃这部分信息往往是必要的？

一方面，舍弃这部分信息之后能使样本的采样密度增大，这正是降维的重要动机

另一方面，当数据受到噪声影响时，最小的特征值所对应的特征向量往往与噪声有关，将它们舍弃能在一定程度上起到去噪的效果

**流行学习manifold learning**

流行: 指在局部具有欧氏空间的性质

等度量映射isometric mapping: 低维嵌入流形上的测地线距离不能用高维空间的直线距离计算（如测地线距离），但能用低邻距离来近似

计算两点之间测地线距离的问题，可以转变为计算近邻连接图上两点之间的最短路径问题

输入：样本集D, 近邻参数k, 低维空间维d’

算法过程：

1. for i = 1, 2, … m do
   1. 确定xi的k近邻
   2. Xi与k近邻点之间的距离设置为欧氏距离，与其他点的距离设置为无穷大
2. 调用最短路径算法计算任意两样本点之间的距离dist(xi, xj)
3. 将dist(xi, xj)作为MDS算法的输入，返回低维空间坐标

输出：样本集在低维空间的投影

Isomap仅是得到了训练样本在低维空间的坐标，对新样本，如何将其映射到低维空间？

常用解决方案是，将训练样本的高维空间坐标作为输入，低维空间坐标作为输出，训练一个回归学习器来对新样本的低维空间坐标进行预测

近邻参数的选择？

若近邻范围指定得较大，则距离很远的点可能被误为近邻，这样就出现“短路”问题；近邻范围指定得较小，则图中有些区域可能与其他区域不存在连接，这样就出现“断路”问题

**Dimensionality Reduction （数据降维）**

High dimensional points (高维数据)-> low dimensional manifold（低维流行）

线性变换:

PCA: 使所有样本的投影尽可能分开，则需最大化投影点的方差。

Multi-Dimensional Scaling (MDS): computes the projection that best preserves the pairwise distances between input points (求原始空间中样本之间的距离在低维空间中得以保持)

非线性变换：

Kernel PCA:

Manifold learning: Siamese network with triplet loss

## 特征选择与稀疏学习

特征选择过程必须确保不丢失重要特征。

冗余特征：如已有特征“长”，“宽”，“高”，那么“面积”，“体积”就是冗余特征。冗余特征很多时候不起作用，去除它们会减轻学习过程的负担。但有时冗余特征会降低学习任务的难度，比如某个冗余特征恰好对应了完成学习任何所需的“中间概念”，（如目标是体积，冗余特征面积），那么该冗余特征就是有用的

特征选择 = 特征子集搜索机制 + 子集评价机制

特征子集搜索机制: 1) 前向搜索：逐渐增加相关特征的策略. 2) 后向搜索：逐渐减少特征的策略

子集评价：对每个候选特征子集，我们可基于训练数据集D来计算其信息增益。一般地，特征子集A实际上确定了对数据集D的一个划分，每个划分区域对应着A上的一个取值，而样本标记信息Y则对应着对D的真实划分，通过估算这两个划分的差异，就能对A进行评价。

三种方法：

过滤式filter：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

包裹式wrapper：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。1) 随机产生特征子集. 2) 基于交叉验证估计学习器误差. 3) 若有改进，更新最优特征子集

嵌入式embedding: 将特征选择过程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，即在学习器训练过程中自动地进行了特征选择. 比如典型例子是基于L1正则化的学习方法

**稀疏表示sparse representation**

特征选择所考虑的问题是特征具有”稀疏性”，即矩阵中的许多列与当前学习任务无关。

数据集具有稀疏性，即D所对应的矩阵中存在很多元素，而且这些0元素并不是以整列、整行形式存在，在不少现实应用中我们会遇到这样的情形，如在文档分类任务中，通常将每个文档看作一个样本，每个字（词）作为一个特征，字（词）在文档中出现的频率或次数作为特征的取值。

当样本具有这样的稀疏表达形式时，对学习任务来说会有不少好处，例如线性支持向量机之所以能在文本数据上有很好的性能，恰是由于文本数据在使用上述的字频表示后具有高度的稀疏性，使大多数问题变得线性可分。

若给定数据集D是稠密的，能否将其转化为稀疏表示呢？

给普通稠密表达的样本找到合适的字典，将样本转化为合适的稀疏表示形式，从而使学习任务得以简化，模型复杂度得以降低，通常称为”字典学习“dictionary learning”，亦称“稀疏编码”sparse coding

在很多应用中均可获得稀疏性的表示，例如图像或声音的数字信号通常在时域上不具有稀疏性，但经过傅里叶变换，余弦变换，小波变换等处理后却会转化为频域上的稀疏信号.

**压缩感知 = 感知测量 + 重构恢复**

在现实任务中，我们常希望根据部分信息来恢复全部信息。如采样后的数字信号恢复成模拟信号（只要满足Nysquist采样定理，采样频率是模拟信号最高频率的两倍以上）

那如果欠采样，能否重构原信号吗？

通过压缩感知技术恢复欠采样信号的前提条件：信号有稀疏表示

如客户对书的喜好程度评分，一般来说相似题材的书籍会有相似的读者，若能将书籍按题材归类，则题材总数必然远远少于书籍总数，因此从题材的角度，评分矩阵是稀疏的

矩阵补全matrix completion技术

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 笑傲江湖 | 万历十五年 | 人间词话 | 神雕侠侣 | 人类的故事 |
| 用户1 | 5 | ？ | ？ | 3 | 2 |
| 用户2 | ？ | 5 | 3 | ？ | 5 |
| 用户3 | 5 | 3 | ？ | ？ | ？ |
| 用户4 | 3 | ？ | 5 | 4 | ？ |

min rank(X) subject to (X)ij = (A)ij i, j 属于A非零元素下标

X表示待恢复的稀疏信号

A表示如表格评分矩阵的已观测信号

理论研究表明，在满足一定条件时，若rank(A) = r, n << m, 则只需观察到O(mr\*logm)个元素就能完美恢复A

# 模型评估与选择

## 划分数据：训练集 + 验证集 + 测试集

经验误差（训练误差）：学习器在训练集上的误差

Overfitting: 应该从训练样本中尽可能学出适用于所有潜在样本的“普遍规律“，如果学习器把训练样本学得“太好”了的话，很可能已经把样本自身的一些特点当作了所有潜在样本都会具有的一般性质，这样就会导致泛化性能下降

**很多因素导致过拟合，最常见的情况是由于学习能力过于强大，以至于把训练样本所包含的不太一般的特性都学到了**，过拟合是无法彻底避免的，我们所能做的只是“缓解”，或者说减小其风险。

测试误差：测试集上的“测试误差”作为泛化误差的近似，通常我们假设测试样本也是从样本真实分布中独立分布采样而得。

训练集与测试集如何划分？ 数据集=训练集+测试集

训练/测试集的划分要尽可能保持数据分布的一致性，避免因数据划分过程引入额外的偏差而对最终结果产生影响

* **分层采样(stratified sampling)**:

保留类别比例的采样方式

* **留出法hold-out**:

单次使用hold-out得到的估计结果往往不够稳定可靠，在使用hold-out时，一般要采用若干次随机划分、重复进行实验评估后取平均值作为hold-out的评估结果。常见做法是将大约2/3-4/5的样本用于训练，剩余样本用于测试

* **交叉验证法cross validation**:

数据集D划分k个大小相似的互斥子集, 每个子集都尽可能保持数据分布的一致性，即从D中通过分层采样得到。通常把交叉验证法称为“k折交叉验证”k-fold cross validation, k最常用的取值是10. 与hold-out相似，将数据集D划分为k个子集同样存在多种划分方式，为减少因样本划分不同而引入的差别，k折交叉验证通常要随机使用不同的划分重复p次，最终结果取均值. 若令k = 数据集数目，则为留一法leave-one-out. 留一法中被实际评估的模型与期望评估的用D训练出的模型很相似。

* **自助法bootstrapping**:

希望评估的是用D训练出的模型，而保留了一部分样本用于测试，因此实际评估的模型所使用的训练集比D小，这必然会引入一些因训练样本规模不同而导致的估计偏差。有没什么办法可以减少样本规模不同造成的影响？

自助法是一个比较好的解决方案，有放回抽样，则样本在m次采样中始终不被采到的概率是(1-1/m)m，取极限得到1/e = 0.368，即通过自助采样，初始数据集D中约有36.8%的样本未出现在采样数据集中。M个样本（含重复样本）作为训练集，未被采样的样本用于测试，亦称包外估计out-of-bag estimate

自助法在数据集较小，难以有效划分训练/测试集时很有用； 此外，自助法能从初始训练集中产生多个不同的训练集，这对集成学习等方法有很大的好处。然后自助法产生的数据集改变了初始数据集的分布，这会引入估计偏差。因此在初始数据量足够时，留出法和交叉验证法更常用一些。

调参与最终模型 训练集 = 训练集 + 验证集

在研究对比不同算法的泛化性能时，我们用测试集上的判别效果来估计模型在实际使用时的泛化能力，而把训练数据另外划分为训练集和验证集，基于验证集上的性能来进行模型选择和调参。模型选择完成后，学习算法和参数配置已选定，此时应该用数据集D重新训练模型。

## 性能度量

假设一共100学生，男女各半，检测的时候男生全部检测正确，女生有25人分为男生。

* Accuracy & Error

分类对的样本除以总样本个数，这个一般是针对于整个分类任务说的

分对了男生50人加女生25人也就是分对了75人，acc=0.75

* Recall (所有好瓜中有多少比例被挑了出来？) = TP / (TP + FN)

针对某一类来说的，他只管自己负责的这一类有多少被检测出来了，而不管其他类有多少误分成我这一类。

男生的recall是1，因为男生全部被检测出来了，女生只有一半被检测出来了，所以她的recall就是0.5

* Precision (挑出的西瓜中有多少比例是好瓜？) = TP (TP + FP)

针对某一类来说的，比recall苛刻，鸡蛋里挑骨头，好不容易有一些分成本类的，她还要挑挑捡捡，看是不是真的本类，也就是代表了被检出的某一类中，真正是该类的样本占的概率

男生的precision是50/75=0.666，女生的precision就是25/25=1

* P-R曲线

预测结果是概率，将测试集的样例预测结果排序，顺序逐个把样本作为正例进行预测，每次计算出当前的precision and recall, 以precision为x轴，以recall为y轴作图

* mAP (mean average precision)

计算出每一类的ap，然后计算所有类别ap的平均值

AP为P-R曲线下的面积

* F1：precision and recall的调和平均

1/F1 = 1/2 \* (1/P + 1/R)

在一些应用中，对precision and recall的重视程度有所不同，例如在商品推荐中， 为了尽可能少打扰用户，更希望推荐内容确是用户感兴趣的，此时precision更重要；而在逃犯信息的信息检索系统中，更希望尽可能少漏掉逃犯，此时recall更重要，需要用加权F1

加权F1: 加权调和平均

1/Fb = 1/(1+b2)(1/P + b2/R)

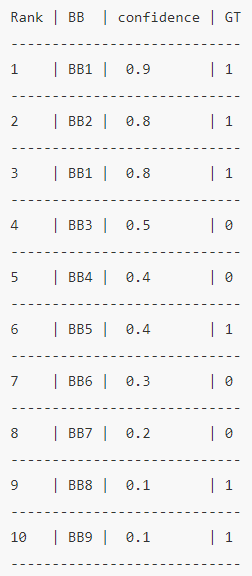
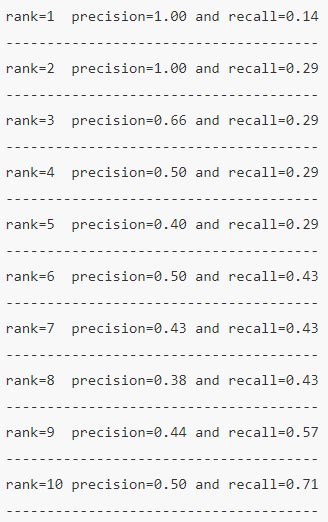
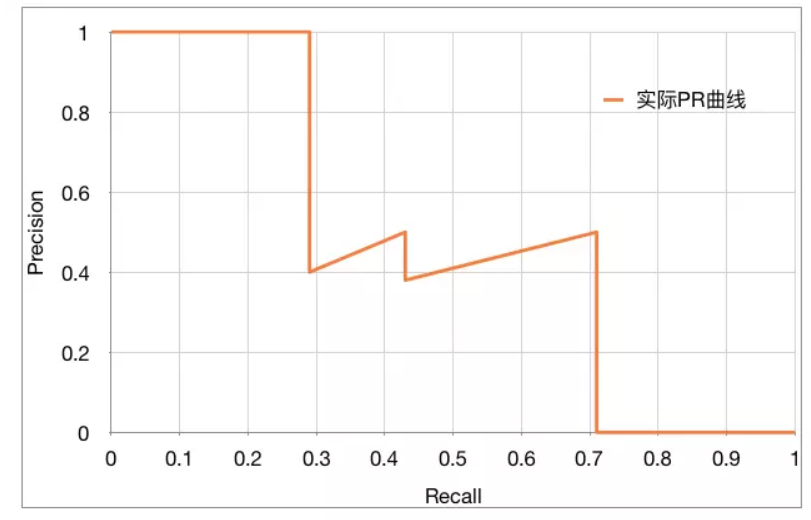
(Note: 与算术平均(P+R)/2 和几何平均(P+R)1/2相比，调和平均更重视较小值

* ROC:

预测结果是概率，将测试集的样例预测结果排序，顺序逐个把样本作为正例进行预测，每次计算出当前的真正率True Positive Rate = TP/(TP+FN), 假正率False Positive Rate = FP/(TN+FP), 以假正率为x轴，以真正率为y轴作图

若object detection， 请参考<https://juejin.im/post/5c696cb6f265da2de9706c0a>

先固定IoU\_threshold，然后排序object\_score从高到低

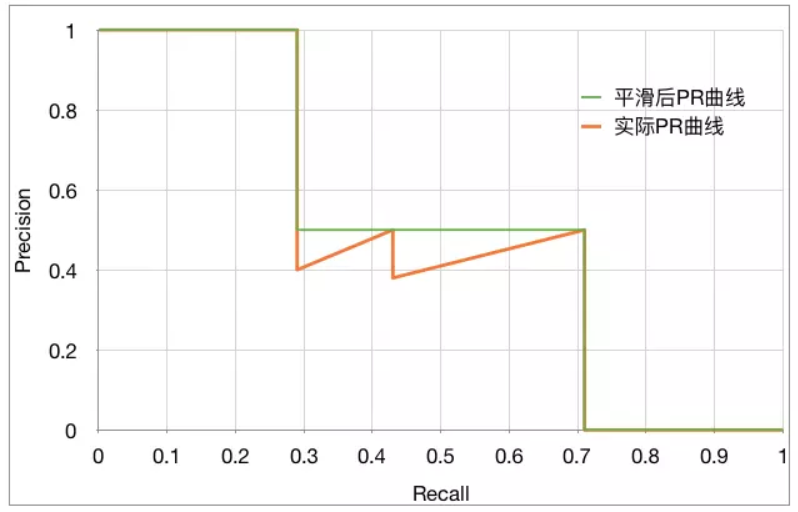
 ->  -> 

随着预测正反分割线的向下移动，Recall稳步变大，Precision整体减小，局部上下跳动

计算AP有两种方法

VOC2010之前的方法：

AP =（平滑后PR曲线上，Recall分别等于0，0.1，0.2，… , 1.0等11处Precision的平均值）。

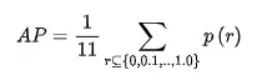


平滑曲线，保证单调递减

方法：

查全率r对应的查准率p，取查全率大于等于r时最大的查准率p。即





AP=(1 + 1 + 1 + 0.5 + 0.5 + 0.5 + 0.5 + 0.5 + 0 + 0 + 0) / 11 = 0.5

VOC2010及以后的方法

AP=平滑后PR曲线下包围的面积

AP = (0.14-0) \* 1 + (0.29-0.14) \* 1 + (0.43-0.29) \* 0.5 + (0.57-0.43) \* 0.5 + (0.71-0.57) \* 0.5 + (1-0.71) \* 0 = 0.5

VOC数据集中的mAP VOC数据集中的mAP计算的是IoU\_threshold=0.5时各个类别AP的均值。

COCO数据集中的mAP 检测是否正确有两个超参数，P\_threshold和IoU\_threshold。

COCO则认为固定了IoU\_threshold的取值，无法衡量IoU\_threshold对模型性能的影响。

COCO在VOC标准的基础上，取IoU\_threshold=0.5，0.55， 0.6，… , 0.95时各个mAP的均值。

饱和曲线

<https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee>

[2012] Predicting sample size required for classification performance

[2016] How much data is needed to train a medical image deep learning system to achieve necessary high accuracy

## 如何比较学习器？

* **基于实验评估方法和性能度量**

见《机器学习》周志华page 37 - ，比想像中的困难，有两个因素：

1. 因为比较的是泛化性能，然后实验评估方法获得的是测试集上的性能
2. 测试集上的性能与测试集本身的选择有很大关系

需要基于统计假设检验hypothesis test，即若在测试集上观察到学习器A比B好，则A的泛化性能是否在统计意义上优于B，以及这个结论的把握有多大？

对单个学习器泛化性能

**假设检验**

假设检验中的”假设“是对学习器泛化错误率分布的某种判断或猜想，例如泛化错误率是否不大于0.3, 即error <= 0.3

已知测试错误率，在一定置信度下，接受还是拒绝假设

估推出泛化错误率的分布

假定泛化错误率为error, 测试错误率为error\_t (意味着在m个测试样本中恰有error\_t \* m个被误分类)

误分类样本数的概率分布满足二项分布

P(error\_t; error) = Cmn \* errorerror\_t\*m \* (1-error)m-(error\_t\*m)

假设error <= e0, 求得在置信度1 – alpha的概率内所能观测到的最大错误率max\_error

若测试错误率error\_t<最大错误率max\_error，则假设error<= e0不能被拒绝，即以1-alpha的置信度认为，学习器的泛化错误率不大于e0

**t检验(t-test)**

若通过多次重复hold-out or cross-validation进行多次训练/测试，会得到多个测试错误率，此时用t检验(t-test)

对多个学习器泛化性能

交叉验证t检验

McNemar检验

* **偏差与方差**

对学习算法除了通过实验估计其泛化性能，还可以借助“偏差-方差分解”

偏差-方差分解试图对学习算法的期望泛化错误率进行拆解。

x: 测试样本

yD: 数据集中的标记

y: x的真实标记

f(x; D)：训练集D上学得模型f在x上的预测输出

f\_t = ED[f(x; D)]: 学习算法的期望预测

**var(x) = ED[(f – f\_t)2]: 使用样本数相同的不同训练集**

e\_n2 = ED[(yD – y )2]: 噪声产生的误差

**bias = f\_t – y: 偏差 （期望输出与真实标记的差别）**

算法的期望泛化误差：

ED[( f(x; D) – yD)2] = bias2 + var(x) + e\_n2

泛化误差可分解为偏差、方差与噪声之和

偏差度量学习算法的期望预测与真实结果的偏离程度，即刻画了学习算法本身的拟合能力

方差度量同样大小训练集的变动导致学习性能的变化，即刻画了数据扰动所造成的影响

噪声则表达了在当前任务上任何学习算法所能达到的期望泛化误差的下界，即刻画了学习问题本身的难度。

偏差-方差分解说明：泛化性能是由学习算法的能力、数据的充分性以及学习任务本身的难度所共同决定的。

给定学习任务，为了取得好的泛化性能，则需使偏差较小，即能够充分拟合数据，并且使方差较小，即使得数据扰动产生的影响小

一般来说，偏差与方差是有冲突的。给定学习任务，若在训练不足时，学习器的拟合能力不强，训练数据的扰动不足以使学习器产生显著变化，此时偏差主导了泛化错误率； 随着训练程度的加深，学习器的拟合能力逐渐增强，训练数据发生的扰动渐渐能被学习器学到，方差逐渐主导了泛化错误率； 在训练程度充足后，学习器的拟合能力非常强，训练数据发生的轻微扰动都会导致学习器发生显著变化，若训练数据自身的、非全局的特性被学习器学到了，则将发生过拟合。

# 监督学习

## 线性模型 (具有很好的可解释性)

线性回归：y = WTx + b

模型权重：Wopt = (XTX)-1XTy

现实任务中XTX往往不是满秩，例如在许多任务中我们会遇到大量的变量，其数目甚至超过样例数，此时可解出多个w, 它们都能使均方误差最小化，选择哪一个解作为输出，将由学习算法的归纳偏好决定，常见的做法是引入正则化regularization项。

稀疏表示sparse representation：本质上对应L0范数的优化，这在通常条件下是NP难题，LASSO通过L1范数来近似L0范数，是求取稀疏解的重要技术

对数线性回归：lny = WTx + b

广义线性模型：y = g-1(WTx + b) g(.)单调可微函数

Sigmoid函数：y = 1/(1+e-z), z = WTx + b

等价于 ln(y/(1-y)) = WTx + b 其中ln(y/(1-y))称为对数几率

**优点：直接对分类可能性进行建模，无需事先假设数据分布，这样就避免了假设分布不准确所带来的问题，它不是仅预测出“类别”，而是可得到近似概率预测**，这对许多需利用概率辅助决策的任务很有用。

## 线性判别分析LDA = Linear Discriminant Analysis

思想：给定训练集，设法将样例投影到一条直线上，使得同类样例的投影点尽可能接近，异类样例的投影点尽可能远。

y = WTx

Wopt = Sw-1 (u0 – u1)

Sw: 类内散度矩阵 within-class scatter matrix

Sb: 类间散度矩阵between-class scatter matrix

考虑到数值解的稳定性，在实践中通常是对Sw进行奇异值分解

## 决策树

决策过程中提出的每个判定问题都是对某个属性的“测试”，例如“色泽=？”； 每个测试的结果或是导出最终结论，或是导出进一步的判定问题，其考虑范围是在上次决策结果的限定范围之内

一棵决策树包含一个根结点、若干个内部结点和若干个叶结点；叶结点对应于决策结果，其他每个结点则对应于一个属性测试。

一般而言，随着划分过程不断进行，我们希望决策树分支结点所包含的样本尽可能属于同一类别，即结点的“纯度”越来越高

度量样本集合纯度purity的指标：

信息熵information entropy: - 类k比例\*log(类k比例)的求和 k个类别

基尼指数Gini index: 反映了从数据集D中随机抽取两个样本，其类别标记不一致的概率

剪枝

决策树分支过多，会把训练样本学得过好，以致于把训练集自身的一些特点当作所有数据都具有的一般性质而导致过拟合，因此可通过主动去掉一些分支来降低过拟合的风险

预剪枝pre-pruning：在决策树生成过程中，对每个结点在划分前先进行估计，若当前结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结点标记为叶结点

有些分支的当前划分虽不能提升泛化性能、甚至可能导致泛化性能暂时下降，但在其基础上进行的后续划分却有可能导致性能显著提高，而预剪枝基于贪心本质禁止这些分支展开，给预剪枝决策树带来了欠拟合的风险

后剪枝post-pruning：先从训练集生成一棵完事的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点

后剪枝决策树通过比预剪枝保留了更多的分支，一般情形下，后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝。

如何判断决策树泛化性能是否提升呢？从训练集中预留一部分用作验证集以进行性能评估

多变量决策树

决策树所形成的分类边界有一个明显的特点：轴平行，即它的分类边界由若干个与坐标轴平行的分段组成。

显然，分类边界的每一段都是与坐标轴平行的，这样的分类边界使得学习结果有较好的可解释性，因为每一段划分都直接对应了某个属性取值。但在学习任务的其实分类边界比较复杂时，必须使用很多段划分才能获得较好的近似。

如何使用斜的划分边界甚至更复杂划分的决策树？

采用多变量决策树，非叶结点不再是仅对某个属性，而是对属性的线性组合（对应斜划分边界）进行测试。

信息增益，增益率，基尼指数等准则对决策树的尺寸有较大影响，但对泛化性能影响有限，大约2%左右

剪枝方法和程度对决策树泛化性能影响相当显著，在数据带有噪声时通过剪枝甚至可将决策树的泛化性能提高25%

决策树是一种由结点和有向边组成的层次结构

在决策树中，每个叶结点都赋予一个类标号。非终结点（包含根结点和内部结点）包含属性测试条件，用以分开具有不同特性的记录。

如何建立决策树？

原则上讲，对于给定的属性集，可以构造的决策树的数目达指数级。由于搜索空间是指数规模的，找出最佳决策树在计算上是不可行的。所以算法的目的是能够在合理的时间内构造出具有一定准确率的局部最优决策树

Hunt算法是许多决策树算法的基础，如CART

1. 假定分类问题的初始决策树只有一个结点，因为根结点包含超过一个类的记录，所以需要进一步的细化
2. 根据一个属性测试条件，数据集被划分为较小的子集
3. 对根结点的每个子女递归地调用Hunt算法

核心：

如何选择一个属性测试条件，作为划分数据的最优标准？也就是说，算法必须提供为不同类的属性指定测试条件的方法，并且提供评估每种测试条件的客观度量

选择最佳划分的度量通常是：根据划分后子女结点不纯性的程度。不纯的程度越低，类分布就越倾斜。均衡分布(0.5, 0.5)的结点具有最高的不纯性

其中t是结点t的总数

其中表示结点t中属性类i的记录所占的比例

例子：如何用决策树分类法区分正常的用户访问，还是由Web爬虫产生的访问？

输入数据：Web服务器日志

Web日志记录的字段：包括客户端的IP地址、请求的时间戳、请求访问的文档的网址、文档的大小、客户的身份（通过用户代理字段获得）。

Web会话是客户在一次网站访问期间发出的请求序列，每个Web会话都可以用有向图来建模，其中结点对应于网页，而有向边对应于连接网页的超链。

目标：对Web会话进行分类

特征构建：构造描述每次会话特性的特征，如遍历的深度和宽度…

训练的模型：（作为描述性模型）

Web机器人的访问倾向于宽而浅，而正常用户访问比较集中（窄而深）

与正常用户不同，Web机器人很少访问与Web文档相关的图片页

Web机器人的会话的长度趋于较长，包含了大量请求页面

Web机器人更可能对相同的文档发出重复的请求，因为正常用户访问的网页常常会被浏览器保存

**决策树优缺点**

优点：容易解释, 对噪声的干扰具有相当好的鲁棒性；不要求任何先验假设，不假定类和其他属性服从一定的概率分布。

缺点：

1) NP完全问题，一般采用启发式方法指导对假设空间的搜索.

2) 冗余属性（一个属性在数据中与另一个属性是强相关的）不会对决策树的准确率造成不利的影响。但是如果数据集中含有很多不相关的属性（即对分类任务没有用的属性），则某些不相关属性可能在树的构造过程中偶然被选中，导致决策树过于庞大.

3) 大多数的决策算法采用自顶向下的递归划分方法，因此沿着树向下，记录会越来越少。在叶结点，记录可能太少，对于叶结点代表的类，不能做出具有统计意义的判决，这就是所谓的数据碎片问题**。**

**决策树的物理解释：**

可以将决策树的生长过程看成划分属性空间不相交的区域的过程，直到每个区域都只包含同一类的记录。由于测试条件只涉及单个属性，因此决策边界是直线，即平行于坐标轴，这就限制了决策树对连续属性之间复杂关系建模的表达能力。

**模型过拟合和剪枝**

决策树过大容易受所谓过拟合(overfitting)现象的影响。通过修剪初始决策树的分支，剪枝有助于提高决策树的泛化能力。模型越是复杂，出现过分拟合的几率就越高，可以采用奥卡姆剃刀原则：给定两个具有相同的泛化误差的模型，较简单的模型比较复杂的模型更可取

过分拟合会导致低训练误差，但高泛化误差。

两种可能：噪声导致的过拟合，缺乏代表性样本导致的过拟合

如何解决？

先剪枝（提前终止规则）：当观察到的不纯性度量的增益（或估计的**泛化误差的改进**）低于某个确定的阈值时就停止扩展叶结点。然而很难为提前终止选取正确的阈值，阈值太高将导致拟合不足，阈值太低就不能充分解决过分拟合的问题

后剪枝（常用方法）：初始决策树按照最大规模生长，然后自底向上的方式修剪完全增长的决策树。1) 用新的叶结点替换子树，该叶结点的类标号由子树下记录中的多数类确定；2）用子树中最常使用的分支代替子树。

分类器性能的评估

在检验集上评估模型的性能。

1. Holdout方法

数据集 = 训练集 + 测试集

训练集越小，模型的方差越大；若训练集太大，较小的检验集估计的准确率不太可靠

1. 交叉验证法

K折交叉验证 （若k=2, 则二折交叉；k=10 常见；若k=N，则留一法）

误差为K个检测集的误差总和

## 神经网络

Sigmoid函数：它把可能在较大范围内变化的输入值挤压到(0, 1)输出范围内，因此有时也称为“挤压函数”squashing function

过拟合

BP神经网络，由于其强大的表示能力，BP神经网络经常遭遇过拟合，其训练误差持续降低，但测试误差却可能上升

解决过拟合方法：

以多组不同参数初始化多个神经网络，相当于从多个不同的初始点开始搜索。

**Early stopping：**将数据分为训练集和验证集，验证集误差升高，则停止训练，返回具有最小验证集误差的连接权和阈值

**正则化：**在误差目标函数中增加一个用于描述网络复杂度的部分。如局部权小问题

机器学习中的范数规则化：L0, L1, L2

blog.csdn.net/zouxy09/article/details/24971995

监督机器学习问题无非就是“minimizeyour error while regularizing your parameters”，也就是在规则化参数的同时最小化误差。最小化误差是为了让我们的模型拟合我们的训练数据，而规则化参数是防止我们的模型过分拟合我们的训练数据。多么简约的哲学啊！因为参数太多，会导致我们的模型复杂度上升，容易过拟合，也就是我们的训练误差会很小。但训练误差小并不是我们的最终目标，我们的目标是希望模型的测试误差小，也就是能准确的预测新的样本。所以，我们需要保证模型“简单”的基础上最小化训练误差，这样得到的参数才具有好的泛化性能（也就是测试误差也小），而模型“简单”就是通过规则函数来实现的。另外，规则项的使用还可以约束我们的模型的特性。这样就可以将人对这个模型的先验知识融入到模型的学习当中，强行地让学习到的模型具有人想要的特性，例如稀疏、低秩、平滑等等。要知道，有时候人的先验是非常重要的。前人的经验会让你少走很多弯路，这就是为什么我们平时学习最好找个大牛带带的原因。一句点拨可以为我们拨开眼前乌云，还我们一片晴空万里，醍醐灌顶。对机器学习也是一样，如果被我们人稍微点拨一下，它肯定能更快的学习相应的任务。只是由于人和机器的交流目前还没有那么直接的方法，目前这个媒介只能由规则项来担当了。

还有几种角度来看待规则化的。

1. 规则化符合奥卡姆剃刀(Occam's razor)原理。这名字好霸气，razor！不过它的思想很平易近人：在所有可能选择的模型中，我们应该选择能够很好地解释已知数据并且十分简单的模型。

2. 从贝叶斯估计的角度来看，规则化项对应于模型的先验概率。民间还有个说法就是，规则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项(regularizer)或惩罚项(penalty term)。

一般来说，监督学习可以看做最小化下面的目标函数：

http://img.blog.csdn.net/20140504122253546?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvem91eHkwOQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

其中，第一项L(yi,f(xi;w)) 衡量我们的模型（分类或者回归）对第i个样本的预测值f(xi;w)和真实的标签yi之前的误差。因为我们的模型是要拟合我们的训练样本的嘛，所以我们要求这一项最小，也就是要求我们的模型尽量的拟合我们的训练数据。

但正如上面说言，我们不仅要保证训练误差最小，我们更希望我们的模型测试误差小，所以我们需要加上第二项，也就是对参数w的规则化函数Ω(w)去约束我们的模型尽量的简单。

机器学习的大部分带参模型都和这个神似。是的，其实大部分无非就是变换这两项而已。对于第一项Loss函数，如果是Square loss，那就是最小二乘了；如果是Hinge Loss，那就是著名的SVM了；如果是exp-Loss，那就是牛逼的 Boosting了；如果是log-Loss，那就是Logistic Regression了；还有等等。不同的loss函数，具有不同的拟合特性，这个也得就具体问题具体分析的。但这里，我们先不究loss函数的问题，我们把目光转向“规则项Ω(w)”。

规则化函数Ω(w)也有很多种选择，一般是模型复杂度的单调递增函数，模型越复杂，规则化值就越大。比如，规则化项可以是模型参数向量的范数。然而，不同的选择对参数w的约束不同，取得的效果也不同，但我们在论文中常见的都聚集在：零范数、一范数、二范数、迹范数、Frobenius范数和核范数等等。这么多范数，到底它们表达啥意思？具有啥能力？什么时候才能用？什么时候需要用呢？

L0范数是指向量中非0的元素的个数

L1范数是指向量中各个元素绝对值之和，也叫“稀疏规则算子”（Lasso regularization）

L2范数是指向量各元素的平方和然后求平方根

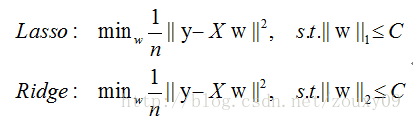
如果我们用L0范数来规则化一个参数矩阵W的话，就是希望W的大部分元素都是0。

为什么L1范数会使权值稀疏？

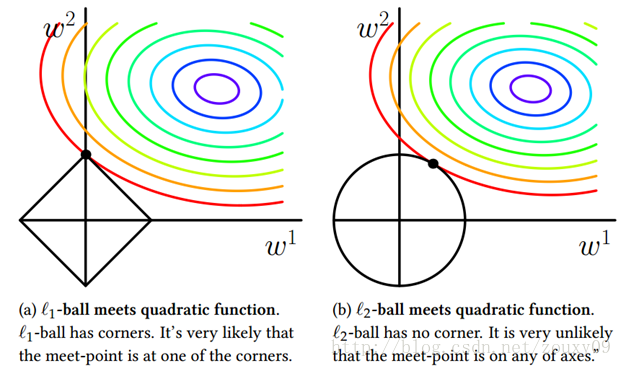
理想的情况是使用L0范数，但L0范数不连续，难以优化求解（NP难问题），而L1范数是L0范数的最优凸近似，且它比L0范数要容易优化求解。

从模型空间的限制角度解释：

L1和L2规则化的代价函数



考虑两维的情况，在(w1, w2)平面上可以画出目标函数的等高线，而约束条件则成为平面上半径为C的一个 norm ball 。等高线与 norm ball 首次相交的地方就是最优解



L1-ball 与L2-ball 的不同就在于L1在和每个坐标轴相交的地方都有“角”出现，而目标函数的测地线除非位置摆得非常好，大部分时候都会在角的地方相交。注意到在角的位置就会产生稀疏性，例如图中的相交点就有w1=0

L2-ball 就没有这样的性质，因为没有角，所以相交的地方出现在具有稀疏性的位置的概率就变得非常小了

让L2范数的规则项||W||2最小，可以使得W的每个元素都很小，都接近于0，但与L1范数不同，它不会让它等于0，而是接近于0

L2正则项的回归叫“岭回归”（Ridge Regression），也称为“权值衰减weight decay”

L1正则项的回归叫LASSO回归

L1会趋向于产生少量的特征，而其他的特征都是0，而L2会选择更多的特征，这些特征都会接近于0。Lasso在特征选择时候非常有用，而Ridge就只是一种规则化而已。

**drops out**[**http://iamtrask.github.io/2015/07/28/dropout/**](http://iamtrask.github.io/2015/07/28/dropout/)



The line represents the error the network generates for every value of a particular weight, The balls in the picture signify various weights, The ball's initial positions are randomly generated, If two balls randomly start in the same colored zone, they will converge to the same point. This makes them redundant! They're wasting computation and memory! This is exactly what happens in neural networks.

Why Dropout: Dropout helps prevent weights from converging to identical positions. It does this by randomly turning nodes off when forward propagating. It then back-propagates with all the nodes turned on

if(do\_dropout):

layer\_1 \*= np.random.binomial([np.ones((len(X),hidden\_dim))],1-dropout\_percent)[0]

\* (1.0/(1-dropout\_percent))

if you're turning off half of your hidden layer, you want to double the values that ARE pushing forward so that the output compensates correctly

Dropout during training. not on your testing dataset

<http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/>

**batch normalization**

xk = xk – E[xk] / sqrt(Var[xk])

yk = rkxk + betak

对mini-batch数据集，沿着第k维进行batch normalization(公式如上)

优点：

improves gradient flow through the networks

allows higher learning rates

reduces the strong dependence on initialization

acts as a form of regularization, slightly reduces the need for dropout

role of bias in Neural Networks

sigmoid(wx+b), w改变的是sigmoid曲线的steepness, b改变的是将x平移

常见神经网络：

RBF网络

单隐层前馈神经网络，使用径向基函数作为隐层神经元激活函数，而输出层则是对隐层神经输出的线性组合。

ART网络 （竞争型学习，无监督学习）

比较层+识别层+识别阈值+重置模块

比较层负责接收输入样本，并将其传递给识别层神经元，识别层神经元对应一个模式类，神经元数目可以训练过程中动态增长以增加新模式类

在接收到比较层的输入信号后，识别层神经元之间相互竞争以产生获胜神经元。竞争的最简单方式：计算输入向量与每个识别层神经元所对应的模式类的代表向量之间的距离，距离最小者胜。若输入向量与获胜神经元的相似度大于识别阈值，则当前输入样本将被归为获胜神经元的向量集，同时，网络连接权将会更新，使得以后在接收到相似输入样本时该模式类会计算出更大的相似度，从而使该获胜神经元有更大可能获胜；若相似度不大于识别阈值，则重置模块奖在识别层增设一个新的神经元，其代表向量就设置为当前输入向量

SOM网络

级联相关网络Cascade-Correlation网络

Recurrent neural networks递归神经网络

隐层神经元的输出被反馈回来，与下一时刻输入层神经元提供的信号一起，作为隐层神经元在下一时刻的输入。

Boltzmann机

深度学习

深层神经网络：通过多层处理，逐渐将初始的“低层”特征表示转化为“高层”特征表示后，用“简单模型”即可完成复杂的分类等学习任务。深度学习也可理解为“特征学习”或表示学习representation learning

参数越多的模型，复杂度越高，容量capacity越大，意味着它能完成更复杂的学习任务

逐层训练 = 预训练pre-training + 微调fine-tuning

每次训练一层隐结点，训练时将上一层隐结点的输出作为输入，而本层隐结点的输出作为下一层隐结点的输入 (预训练)；在预训练全部完成后，再对整个网络进行“微调”

Sigmoid + Sigmoid + mse

batch Gradient decent, mini-batch gradient decent, SGD

<http://iamtrask.github.io/2015/07/12/basic-python-network/> <http://iamtrask.github.io/2015/07/27/python-network-part2/>

<https://iamtrask.github.io/2015/07/27/python-network-part2/>

## 支撑向量机SVM和支撑向量回归SVR

基于训练集D在样本空间中找到一个划分超平面，将不同类别的样本分开，但能将训练样本分开的划分超平面可能有很多

SVM目标：找最大间隔maximum margin

训练完成后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关

在现实任务中，原始样本空间内也许并不存在一个能正确划分两类样本的超平面，幸运的是，如果原始空间是有限维，即属性数有限，那么一定存在一个高维特征空间使样本可分

我们希望样本在特征空间内线性可分，因此特征空间的好坏对支持向量机的性能至关重要。在不知道特征映射的形式时，我们并不知道什么样的核函数是合适的，若核函数选择不合适，则意味着将样本映射到了一个不合适的特征空间

核函数直接决定了支持向量机与核方法的最终性能，但遗憾的是，核函数的选择是一个未决问题

无法SVM or SVR, 学得的模型总能表示成核函数的线性组合

核方法：基于核函数的学习方法

最常见的是，通过“核化”（即引入核函数）来将线性学习器拓展为非线性学习器，如核线性判别分析kernelized linear discriminant analysis

线性核SVM迄今仍是文本分类的首选技术。一个重要原因可能是：若将每个单词作为文本数据的一个属性，则该属性空间维数很高，冗余度很大。其描述能力足以将不同文档打散

支持向量机的求解通常是借助于凸优化技术

## Bayes分类器

决策树、BP神经网络、支持向量机等属于判别式模型

Bayes分类器属于生成式模型

类后验概率p(c | x) = p( c) \* p(x | c) / p(x)

p(x): 归一化因子

p(c): 类先验概率，表达了样本空间中各类样本所占的比例，根据大数定律，当训练集包含充足的独立同分布样本时，p(c)可通过各类样本出现的频率来进行估计

p(x | c):似然（也称为类条件概率），样本x相对类标记c的类条件概率。涉及关于x所有属性的联合概率，直接根据样本出现的频率来估计将会遇到严重的困难。例如，假设样本的d个属性都是二值的，则样本空间将有2d种可能的取值，在现实应用中，这个值往往远大于训练样本数m

**频率主义学派**的极大似然估计maximum likelihood estimation：根据数据采样来估计概率分布参数的经典方法，极大似然估计是试图在参数所有可能的取值中，找到一个能使数据出现的“可能性”最大的值

估计似然的一种常用策略：先假定其具有某种确定的概率分布形式，再基于训练样本对概率分布的参数进行估计，事实上概率模型的训练过程就是参数估计parameter estimation过程。

参数化的方法虽能使似然估计变得相对简单，但估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实数据分布。在现实应用中，欲做出能较好地接近潜在真实分布的假设，往往需要一定程度上利用关于应用任务本身的经验知识

**朴素Bayes分类器**：似然是所有属性上的联合概率，难以从有限的训练样本直接估计而得。为避开这个障碍，朴素Bayes分类器采用了“属性条件独立性假设”，对已知类别，假设所有属性相互独立。

这个假设在现实应用中往往很难成立，但有趣的是，native bayes在很大情形下都能获得相当好的性能，为什么？若属性间依赖对所有类别影响相同，或依赖关系的影响能相互抵消，则属性条件独立性假设在降低计算开销的同时不会对性能产生负面影响。

Native bayes在信息检索领域尤为常用

Bayes network (也称为belief network)

表示任意属性间的依赖性，借助有向无环图来刻画属性之间的依赖关系，并使条件概率表来描述属性的联合概率分布

= 结构学习 + 参数学习

半朴素Bayes分类器

适当考虑一部分属性间的相互依赖信息，从而既不需要进行完全联合概率计算，又不至于彻底忽略了比较强的属性依赖关系。

TAN (Tree Augmented native Bayes) ： 求解强相关属性之间的依赖性

1. 计算任意两个属性之间的条件互信息
2. 以属性为结点构建完全图，任意两个结点之间权重设为条件互信息
3. 构建此完全图的最大带权生成树，挑选根变量，将边置为有向
4. 加入类别结点y, 增加从y到每个属性的有向边

若X表示属性集，Y表示类变量， 若类变量和属性之间的关系不确定，**可以把X和Y看作随机变量**

若假定测试记录有属性集：X=（有房＝否，婚姻状况＝已婚，年收入＝$120k)，预测该记录属于拖欠贷款 YES or 还清贷款NO，即求P(YES|X) and P(NO|X)

P(Y|X) = P(X|Y)\*P(Y) / P(X)

因为先验概率P(Y) and P(X)是常数，所以argmaxP(Y|X) = argmaxP(X|Y)

朴素Bayes分类器：假定属性之间相互独立，则

P(X|Y) = P(x1|Y) \* P(x2|Y) \* … \* P(xd|Y) 假定d个属性

argmaxP(No|X) = argmaxP(X|No) =

argmax(p(有房＝否|No) \* p(婚姻状况＝已婚 | No) \* p(年收入＝$120k|No) )

若属性是分类的：P(xi|Y): 类y中等于xi的训练实例的比例

若属性是连续的：假设连续变量服从某种概率分布，然后使用训练数据估计分布的参数。高斯分布通常被用来表示连续属性的类条件概率分布。该分布有两个参数：mean and std.

P(xi|Y) = exp(-(xi-mean)\*(xi-mean)/(2\*std\*std) / (sqrt(2pi)\*std)

mean: 类yi所有训练记录关于Xi的样本均值

std: 类yi所有训练记录关于Xi的样本标准差

计算结果若P(No|X) > P(Yes|X), 则估计为还清贷款

Bayes特点：

1. 面对孤立的噪声点，朴素Bayes分类器是健壮的。因为在从数据中估计条件概率时，这些点被平均。
2. 若Xi是无关属性，那么P(Xi|Y)几乎变成了均匀分布，不会对总的估计产生影响
3. 相关属性会降低朴素Bayes分类器的性能

## 集成学习ensemble learning （multi-classifier system, committee-based learning）

集成学习通过构建并结合多个学习器来完成学习任务

在一般经验中，如果把好坏不等的东西掺到一起，那么通常结果会是比最坏的要好一些，比最好的要坏一些，集成学习把多个学习器结合起来，如何能获得比最好的单一学习器更好的性能呢？

当然有要求，要获得好的集成，个体学习器应“好而不同”，即学习器要有一定的准确性和多样性（即学习器间具有差异），如下：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 测试例1 | 2 | 3 |  | 1 | 2 | 3 |  | 1 | 2 | 3 |
| H1 | 分类正确 | 正确 | 错误 | H1 | 正确 | 正确 | 错误 | H1 | 正确 | 错误 | 错误 |
| H2 | 分类错误 | 正确 | 正确 | H2 | 正确 | 正确 | 错误 | H2 | 错误 | 正确 | 错误 |
| H3 | 分类正确 | 错误 | 正确 | H3 | 正确 | 正确 | 错误 | H3 | 错误 | 错误 | 正确 |
| 集成 | 分类正确 | 正确 | 正确 | 集成 | 正确 | 正确 | 错误 | 集成 | 错误 | 错误 | 错误 |

左表格集成提升性能，中表格集成不起作用，右表格集成起负作用

如何产生并结合“好而不同”的个体学习器，恰是集成学习研究的核心

**弱学习器：**

泛化性能略优于随机猜测的学习器，如在二分类问题上精度略高于50%的分类器，虽然从理论上来说使用弱学习器集成足以获得好的性能，但在实践中出于种种考虑，例如希望使用较少的个体学习器，人们往往会使用比较强的学习器

**强学习器：组合弱学习器**

序列化方法：个体学习器间存在强依赖关系，如Boosting, AdaBoost

Boosting是一族将弱学习器提升为强学习器的算法，先从初始训练集训练出一个基学习器，再根据其学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器，如此重复进行，直到基学习器数目达到事先指定的值T，最终将这T个基学习器进行加权结合

AdaBoost算法：

1. 初始化样本权值分布
2. 基于分布Dt从数据集D中训练出分类器ht
3. 估计ht的测试误差
4. 确定分类器ht的权重
5. 更新样本分布，使下一轮的基学习器Ht+1能纠正Ht的一些错误

Boosting算法要求基学习器能对特定的数据分布进行学习，这可通过re-weighting实施，即在训练过程的每一轮中，根据样本分布为每个训练样本重新赋予一个权重，对无法接受带权样本的基学习算法，则可通过re-sampling来处理，即在每一轮学习中，根据样本分布对训练集重新进行采样，再用重采样而得的样本集对基学习器进行训练。

Boosting算法在训练的每一轮都要检查当前生成的基学习器是否满足基本条件（是否比随机猜测好）

从偏差-方差分解的角度看，Boosting主要关注降低偏差，因此Boosting能基于泛化性能相当弱的学习器构建出很强的集成

并行化方法：个体学习器间不存在强依赖关系，可并行生成，如Bagging, Random Forest

Bagging

自助采样法采样出T个含m个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习器进行结合

在对预测输出进行结合时，Bagging通常对分类任务使用简单投票法，对回归任务使用简单平均法

自助采样过程还给Bagging带来了另一个优点：由于每个基学习器只使用了初始训练集中约63.2%的样本，剩下约36.8%的样本可用作验证集来对泛化性能进行”包外估计“out-of-bag estimate

从偏差-方法角度看，Bagging主要关注降低方差，因此它在不剪枝决策树、神经网络等易受样本扰动的学习器上效用更为明显

Random Forest

在以决策树为基学习器构建Bagging集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。具体来说，传统决策树在选择划分属性时是在当前结点的属性集合中选择一个最优属性；而在RF中，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含k个属性的子集，然后再从这个子集中选择一个最优属性用于划分。一般推荐值k取属性数目的log2。

随机森林对Bagging只做了小改动，但是与Bagging中基学习器的”多样性“仅通过样本扰动（通过对初始训练集采样）而来不同，随机森林中基学习器的多样性不仅来自样本扰动，还来自属性扰动，这就使得最终集成的泛化性能可通过个体学习器之间差异度的增加而进一步提升。

随机森林的起始性能往往相对较差，特别是在集成中只包含一个基学习器时，这很容易理解，因为通过引入属性扰动，随机森林中个体学习器的性能往往有所降低，然而随着个体学习器数目的增加，随机森林通常会收敛到更低的泛化误差，值得一提的是，随机森林的训练效率常优于Bagging.

随机森林是一类专门为决策树分类器设计的组合方法，它**组合多棵决策树**作出的预测，其中每棵树都是基于随机向量的一个独立集合的值产生的。与AdaBoost使用的自适应方法不同，AdaBoos中概率分布是变化的，以关注难分类的样本，而随机森林则采用一个固定的概率分布来产生随机向量。

**如何组合多个学习器？**

简单平均法

加权平均法：加权平均法的权重一般是从训练数据中学习而得，现实任务中的训练样本通常不充分或存在噪声，这将使得学出的权重不完全可靠，尤其是对规模比较大的集成来说。一般而言，在个体学习器性能相差较大时宜使用加权平均法，而在个体学习器性能相近时宜使用简单平均法

绝对多数投票法majority voting：若某标记得票过半数，则预测为该标记，否则拒绝预测。majority voting提供了”拒绝预测“选项，这在可靠性要求较高的学习任务中是一个很好的机制

相对多数投标法plurality voting: 预测得票最多的标记，若同时有多个标记获得最高票，先置信度高的

加权投票法weighted voting

学习法: 当训练数据很多时，通过另一个学习器来进行结合，如Stacking

Stacking算法: 初始数据集训练出初始学习器, 生成次级学习器的训练集

次级训练集是由初始学习器的输出产生，作为输入特征，标记仍是初始样本的标记; 若直接用初始学习器的训练集来产生次级训练集，则过拟合风险会比较大；因此一般是通过使用交叉难或留一法这样的方式，用训练初始学习器未使用的样本来产生次级学习器的训练样本.如将初始学习器的输出类概率作为次级学习器的输入属性，用多响应性回归作为次级学习算法

**如何提升多样性？**

一般思路是在学习过程中引入随机性，常见做法主要是对数据样本、输入属性、输出表示、算法参数进行扰动

**数据样本扰动**：常基于采样法，在Bagging中使用自助采样，在AdaBoost中使用序列采样。训练样本稍加变化就会导致学习器有显著变动如决策树、神经网络等，数据样本扰动法对这个的”不稳定学习器“很有效。然而有一些基学习器对数据样本的扰动不敏感，例如线性学习器，支持向量机，朴素贝叶斯，k近邻学习器，这样的基学习器称为稳定学习器

**输入属性扰动**：对包含大量冗余属性的数据，在子空间中训练个体学习器能产生多样性大的个体，由于冗余属性多，减少一些属性后训练出的个体学习器也不至于太差，若数据只包含少量属性，或冗余属性很少，则不宜使用输入属性扰动法

**算法参数扰动**：通过随机设置不同的参数，往往可产生差别’较大的个体学习器。注意，使用单一学习器时通常需使用交叉验证等方法来确定参数，这事实 已使用了不同参数训练出多个学习器，只不过最终仅选择其中一个学习器进行使用，而集成学习则相当于把这些学习器都利用起来。

# 无监督学习

## 聚类

常见应用：对相关的顾客分组、找出显著影响地球气候的海洋区域以及压缩数据等。

聚类分析只是解决其他问题（如数据汇总）的起点。聚类分析仅根据在数据中发现的描述对象及其关系的信息，将数据对象分组。其目标是，组内相似性越大，组间差别越大

人类擅长将对象划分成组（聚类），并将特定的对象指派到这些组（分类），聚类分析是研究自动发现这些类的技术。聚类过程仅能自动形成簇结构，族所对应的概念语义需由使用者来把握和命名。物以类聚，即同一族的样本尽可能彼此相似，不同族的样本尽可能不同。

聚类分析提供由个别数据到数据对象所指派的族的抽象。聚类技术使用族原型（即代表簇中其他对象的数据对象）来刻画簇特征。

族原型的使用：

汇总：许多数据分析技术如回归和PCA，可以将算法用于仅包含族原型的数据集，而不是整个数据集。依赖分析类型，原型个数和原型代表数据的精度，汇总结果可以与使用所有数据得到的结果相媲美

有效地发现最近邻：找出最近邻可能需要计算所有点对点之间的距离。通常，可以更有效地发现簇和族原型。如果对象相对地靠近族的原型，则我们可以使用族原型减少发现对象最近邻所需要计算的距离的数目。为了找出一个对象的最近邻，只需要计算到邻近族中对象的距离

**Voronoi剖分**

对任意样本x，它将被划入与其距离最近的原型向量pi所代表的簇中；换言之，每个原型向量定义了与之相关的一个区域Ri, 该区域中每个样本与pi的距离不大于它与其他原型向量的距离，由此形成了对样本空间的簇划分。

聚类**性能度量**

考虑聚类结果的族划分C = {C1, C2, … Ck}

avg(C ): 簇内样本间的平均距离

diam(C ): 族内样本间的最远距离

dmin(Ci, Cj): 两个簇样本间的最近距离

dcen(Ci, Cj): 两个族中心的距离

指标有DB指数，Dunn指数，见《机器学习》周志华p199

**距离计算**

Minkowski distance: distmk(xi, xj) = (xi – xj)pp

当p=2， 欧式距离

当p=1, 曼哈顿距离

连续值属性和有序属性，均可用Minkowski距离计算

对无序属性，采用以VDM距离(value difference metric) 见《机器学习》周志华p201

加权距离计算：distwmk(xi, xj) = w\*(xi – xj)pp

**如何计算聚类簇之间的距离（即**集合间的距离计算**）？**

最小/最大距离：两个簇的最近/最远样本确定

平均/hausdorff距离：两个簇的平均/见图像分割结果的比较

**原型聚类**

假设聚类结构能通过一组原型向量来刻画，如k均值算法，通过k个簇内均值刻画

1. 初始化k个均值向量 （随机选取样本），代表k个簇
2. 考察每个样本，划分到最近的k个簇 (比较样本与k个均值向量的距离)
3. 更新k个族的刻画（计算族均值向量）
4. 重复上述过程，直到算法停止

**学习向量量化learning vector quantization ： 有监督聚类**

假设数据样本带有类别标记，学习过程利用样本的这些监督信息来辅助聚类

1. 初始化k个原型向量 （分层随机选取样本）
2. 考察每个样本，找到距离最近的原型向量
3. 根据类别标记是否一致，更新原型向量。

若最近的原型向量与样本的类别标记相同，则令原型向量靠拢该样本；

若类别标记不同，则令原型向量远离该样本

1. 重复上述过程

**高斯混合聚类（采用k个高斯分布来刻画）**

假设样本的生成过程由高斯混合分布给出：首先，根据a1, a2, … ak定义的先验分布选择高斯混合成分，其中ai为选择第i个混合成分的概率；然后，根据被选择的混合成分的概率密度函数进行采样，从而生成相应的样本。

1. 初始化高斯混合分布：k个混合系数ai和高斯分布的模型参数（均值向量，协方差矩阵）
2. 由样本计算各混合成分生成的后验概率
3. 更新k个混合系数和高斯分布的模型参数
4. 重复上述过程

**密度聚类**

假设聚类结构能通过样本分布的紧密程度确定。

DBSCAN: 基于一组“邻域”参数(距离，最小样本数)来刻画样本分布的紧密程度

根据邻域参数，找出所有核心对象

以任一核心对象为出发点，找出由密度可达的样本生成聚类簇，直至所有核心对象均被访问。

**层次聚类**

先将数据集中的每个样本看作一个初始聚类簇，然后在算法运行的每一步中找出距离最近的两个聚类簇进行合并，该过程不断重复，直到预设的聚类簇个数

聚类类型：

划分与层次

划分聚类partitional clustering:简单地将数据对象集划分成不重叠的子集（簇），使得每个数据对象恰在一个子集中。

层次聚类hierarchical clustering

互斥，重叠与模糊的

模糊聚类：每个点属于每个簇的概率，点的隶属概率之和为1

完全的与部分的

族类型：

基于原型的

族是对象的集合，其中每个对象到定义刻簇的原型的距离比到其他族的原型的距离更近。对**于具有连续属性的数据，簇的原型通常是质心；**

**对于具有分类属性的数据，簇的原型通常是中心点，即簇中最有代表性的点**

通常把基于原型的簇看作基于中心的簇，这种簇趋向于呈球状

基于图的

若数据用图表示，其中结点是对象，而边代表对象之间的联系。则簇可以定义为连通分支，如基于邻近的簇：两个对象相连仅当它们的距离在指定的范围之内。缺点：容易受噪声干扰

基于密度的

优点：当族不规则或互相盘绕，并且有噪声和离群点时，常常使用密度的簇定义

概念簇：

把簇定义为有某种共同性质的对象的集合。

K均值算法：

随机选择K个点作为初始质心

repeat

将每个点指派到最近的质心，形成K个簇

重新计算每个簇的质心

until 质心不发生变化

指派点到最近的质心？

需要邻近性度量，通过欧氏空间中的点使用欧几里得距离，对文档用余弦相似性。

聚类目标？

通常用一个目标函数表示，该函数依赖于点到簇质心的邻近性，如最小化每个点到最近质心的距离平方和

K均值：常见的邻近度，质心和目标函数组合

|  |  |  |
| --- | --- | --- |
| 邻近度函数 | 质心 | 目标函数 |
| 曼哈顿距离 | 中位数 | 最小化对象到其簇质心的L1距离和 |
| 平方欧几里得距离 | 均值 | 最小化距离平方和 |
| 余弦 | 均值 | 最小化余弦相似度和 |
| Bregman散度 | 均值 | 最小化Bregman散度和 |

余弦相似度或平方欧几里得距离是Bregman散度的特例

使用平方误差标准时，离群点可能过度影响所发现的簇，如何识别离群点？

记录每个点对SSE （sum of the squared error)的影响，删除那些具有异乎寻常影响的点，还可能需要删除很小的簇，因为它们常常代表离群点的组。

**K均值缺陷：**

**当簇具有非球形形状，或具有不同尺寸，或具不同密度时，K均值很难检测到“自然的”的簇。因为K均值目标函数是最小化等尺寸和等密度的球形簇，或者明显分离的簇。**

凝聚层次聚类算法：

计算邻近度矩阵

repeat

合并最接近的两人个簇

更新邻近性矩阵，以反映新的簇与原来的簇之间的邻近性

until仅剩下一个簇

簇之间的邻近性：

MIN：不同的结点子集中两人个结点之间的最短边

MAX: 不同的结点子集中两人个结点之间的最长边

组平均：取自不同簇的所有点对邻近度的平均值

层次聚类的空间复杂度O(m\*m),时间复杂度O(m\*m\*log(m))

## 异常检测

识别其特征显著不同于其他数据的观测值。

常见应用：检测欺诈、网络攻击、疾病的不寻常模式、生态系统扰动等。

常借助于聚类或距离计算进行，如将远离所有簇中心的样本作为异常点，或将密度极低处的样本作为异常点

## 关联分析

用来发现描述数据中**强关联特征的模式**。由于搜索空间是指数规模的，关联分析的目标是以有效的方式提取最有趣的模式。常见的应用：具体有相关功能的基因组、识别用户一起访问的web页面、理解地球气候系统不同元素之间的联系等

# 半监督学习

未标记样本虽未直接包含标记信息，但若它们与有标记样本是从同样的数据源独立同分布采样而来，则它们所包含的关于数据分布的信息对建立模型将大有裨益。

要利用未标记样本，必然要做一些将未标记样本所揭示的数据分布信息与类别标记相联系的假设。如

a. 聚类假设(cluster assumption),即假设数据存在簇结构，同一个簇的样本属于同一个类别

等价于低密度分类假设（Low Density Separation Assumption），即分类决策边界应该穿过稀疏数据区域，而避免将稠密数据区域的样例划分到决策边界两侧。

b. 流形假设(manifold assumption), 即假设数据分布在一个流形结构上，邻近的样本拥有相似的输出值。

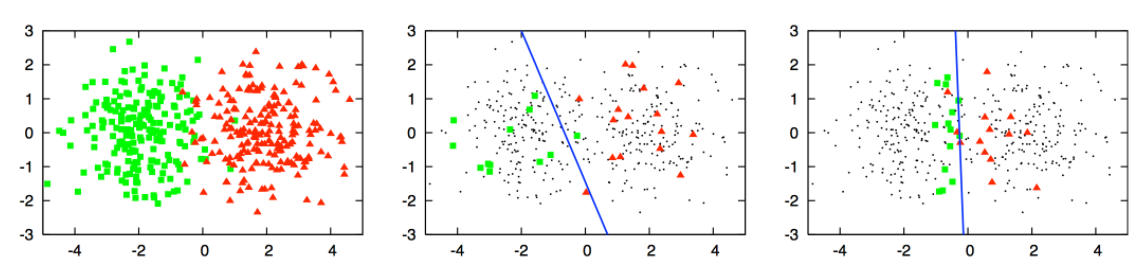
无论聚类假设还是流形假设，其本质都是“相似的样本拥有相似的输出”这个基本假设

## Active Learning

Passive learning: collect data randomly sampled from the underlying distribution

The main difference between an active and a passive learner is the ability to query instances based upon labelled data

应用：最小化标注数量



支撑向量的标注更有信息，如何寻找支撑向量？

Small labelled data

train model

A large pool of unlabelled data

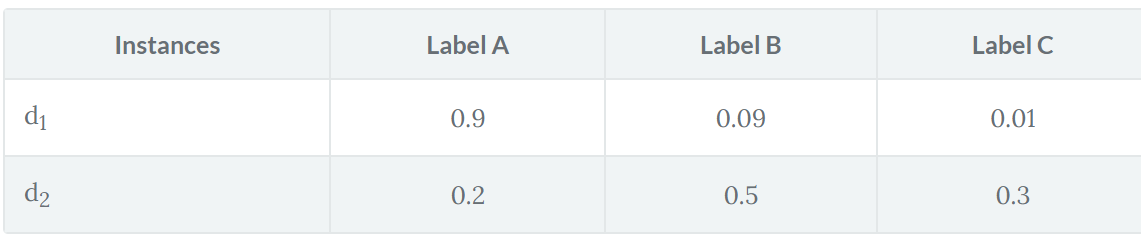
Calc and rank the informativeness measure

Select the best instances (with most informative)

Label selected data and add them into existing labelled data, repeat the above process

如何度量informative?

已训练的模型预测unlabelled data，获得各类的概率，假定如下：



越不确定的instance越有信息

Least Confidence: the learner selects the instance for which it has the least confidence in its most likely label, 比如case 1 0.9, case 2 0.5, 选择case 2 (d1=0.9, d2=0.5, 则选d2)

Margin Sampling: select the instance that has the smallest difference between the first and second most probable labels, 比如d1=0.9-0.09, d2=0.5-0.3, 则选d2

Entropy sampling (最大熵): d1 = sum(- p\*log(p) ) = -(0.9\*log(0.9) + 0.09\*log(0.09) + 0.01\*log(0.01)) = 0.155, d2 = 0.447, 则选最大熵（度量信息最大化）d2

直推学习transductive: 假定学习过程中所考虑的未标记样本恰是待预测数据，学习的目的就是在这些未标记样本上获得最优泛化性能

纯半监督学习：假定训练数据中的未标记样本并非待预测的数据

**self-training算法:**

标签传播算法，它假设距离样本点最近的数据点的标签就是该样本点的标签，需要经过迭代，每次迭代只赋予一个样本点标签。**该算法受离群点和偶然性因素影响太大**，效果不佳。

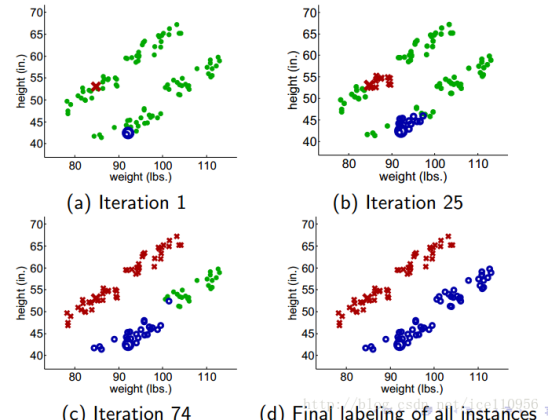
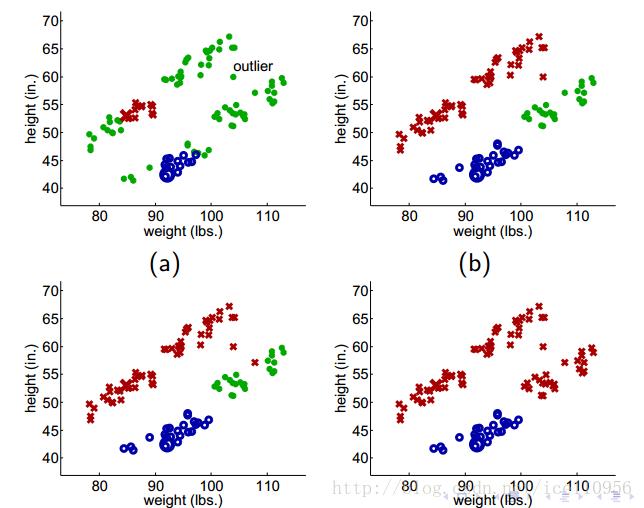
Repeat:

用L（有标记样本集）生成分类策略F;

用F分类U（无标记样本集）,计算误差

选取U的子集u,即误差小的,加入标记.L=L+u;

重复上述步骤,直到U为空集.

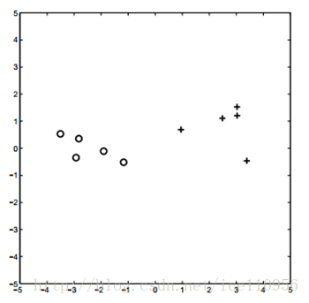
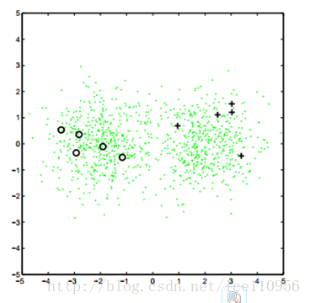
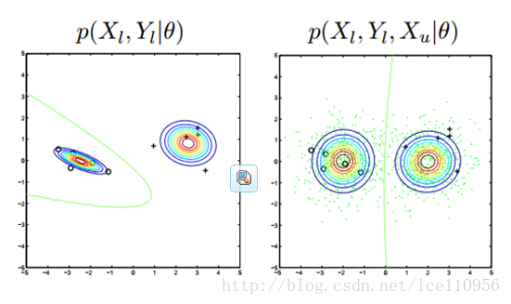
右图有噪声点，导致分类出错

**生成式方法**

假设所有数据（无论是否有标记）都是由同一个潜在的模型“生成”的

未标记数据与学习目标是通过假设同一个潜在模型的参数联系在一起，未标记数据的标记则可看作模型的缺失参数，通常可基于EM算法进行极大似然估计求解

常用的高斯混合模型(GMM)

右图：前者最大化标记样本出现概率,后者加入了无标记样本出现概率.

基于生成式模型的半监督学习方法简单、直观，并且在训练样本，特别是有标记样本极少时能够取得比判别式模型更好的性能，但是**当模型假设与数据分布不一致时，使用大量的未标记数据来估计模型参数反而会降低学得模型的泛化能力**。由于寻找合适的生成式模型来为数据建模需要大量领域知识，这使得基于生成式模型的半监督学习在实际问题中的应用有限

**图半监督学习:** 数据集 -> 图

做法：数据集中每个样本对应于图中一个结点，若两个样本之间的相似度很高，则对应的结点之间存在一条边，边的强度正比于样本之间的相似度。

将有标记样本所对应的结点染颜色，未标记样本样本所对应的结点尚未染色，于是半监督学习就对应于“颜色”在图上扩散或传播的过程

设计能量函数满足：

1. 标记样本的预测与真实标记尽可能相同
2. 相似样本具有相似标记

图的质量极为重要，常见的有高斯距离图，k近邻图等

**半监督SVM** 见《机器学习》周志华p299

试图找到能将两类有标记样本分开，且穿过数据低密度区域的划分超平面

典型算法是Transductive SVM: 试图考虑对未标记样本进行各种可能的标记指派label assignment,即尝试将每个未标记样本分别作为正例或反例，然后在所有这些结果中，寻求一个在所有样本（包括有标记样本和进行了所有标记指派的未标记样本）上间隔最大化的划分超平面

如何设计出高效的优化求解策略，此外，训练数据中存在多个”低密度划分“

**基于分歧的方法**

起源于协同训练co-training，多视图学习multi-view learning的代表

在不少现实应用中，一个数据对象往往同时拥有多个”属性集“，每个属性集构成了一个视图

如对一部分电影来说，它拥有多个属性集：图像画面属性集，声音信息属性信，字幕属性集等

假设数据拥有两个充分且条件独立视图，若两个视图充分具条件独立，则可利用未标记样本通过协同训练将弱分类器的泛化性能提升到任意高

事实上无需数据拥有多视图，仅需弱学习器之间具有显著的分歧（或差异），即可通过相互提供伪标记样本的方式来提升泛化性能（所以叫基于分歧的方法），不同视图、不同算法、不同数据采样、不同参数设置等，都能导致学习器之间的分歧

充分指每个视图都包含足以产生最优学习器的信息

条件独立则是指在给定类别标记条件下两个视图独立

视图的条件独立性在现实任务中通常很难满足，因此性能提升幅度不会那么大。

基本训练过程为：在有类标签的样本的两个不同视图上分别训练，得到两个不同的学习机，然后用这两个学习机预测无类标签的样例的类标签，每个学习机选择标记结果置信度最高的样例和它们的类标签加入另一个学习机的有类标签的样本集中。这个过程反复迭代进行，直到满足停止条件。这个方法需要满足两个假设条件：（1）视图充分冗余假设。给定足够数量的有类标签的样本，基于每个视图都能通过训练得到性能很好的学习机；（2）条件独立假设。每个视图的类标签都条件独立于另一视图给定的类标签。

**半监督聚类**

约束k均值 constrained k-means

和k均值算法流程相同，只是将样本划分时多一个约束：检测将x划入聚类簇是否会违背”必连“集must-link（指样本必属于同一个簇）与勿连集cannot link（样本必不属于同一个簇）

约束种子k均值constrained seed k-means

用有标记样本初始化簇中心，在聚类簇迭代更新过程中，不改变种子样本的簇隶属关系

# 概率图模型

概率模型：将学习任务归结于计算变量的概率分布

假定所关心的变量集合为Y, 可观测变量集合为X, 其他变量的集合为R

生成式模型generative model 考虑联合分布P(Y, X, R)

判别式模型discriminative 考虑条件分布P(Y, R | X)

概率图模型probabilistic graphical model是一类用图来表达变量相关关系的概率模型。结点表示随机变量，边表示变量间的概率相关关系

= 贝叶斯网 + 马尔可夫网

贝叶斯网 = 有向图模型

若变量间存在显示的因果关系

马尔可夫网 = 无向图模型

若变量间存在相关性，但难以获得显示的因果关系

## 统计语言模型

计算机需要知道一个文字序列是否能构成一个大家理解而且有意义的句子？

一个句子是否合理，就看看它的可能性大小如何（即概率），如这句话“用数学的方法描述语言规律”，出现的概率大致是10-20； “描述语言规律用数学的方法”， 出现的概率大致是10-25；”语规言律数…”，出现的概率大致是10-70

语言模型：是给定一个字符串，看它是自然语言的概率

p(w1, w2, w3, ..., wn) = p(w1)p(w2|w1)p(w3|w2,w1)…p(wn | wn-1, …, w2,w1)

其中p(w1)表示第一个词出现的概率

二元模型：

利用马尔可夫假设（即假设任意一个词出现的概率只同它前面的词有关）

p(w1, w2, w3, ..., wn) = p(w1)p(w2|w1)p(w3|w2)…p(wn | wn-1)

p(wn | wn-1) = p(wn, wn-1) / p(wn-1), 由大数定理，相对频度等于概率

N元模型：

N-1阶马尔可夫假设（即假设任意一个词出现的概率只同它前面的N-1个词有关）

N一般取4

是否越高阶模型，结果越准确吗？

因为自然语文中，上下文之间的相关性可能跨度非常大，甚至可以从一个段落跨到另一个段落，这就是马尔可夫假设的局限性

几乎所有的自然语言处理问题都可以等价成通信解码问题，

如机器翻译，接收英语信息，推测汉语意思

如通信解码问题（信息y1, y2, … -> 编码,传递,解码 -> x1, x2, …）

如何根据接收端的观测信号x1, x2, …来推测信号源发送的信息y1, y2, …

即令p(y1, y2, … | x1, x2, …)达到最大的信息串y1, y2, …

y1, y2, … = argmax p(y1, y2, … | x1, x2, …) all y1, y2, …

= argmax p(x1, x2, …| y1, y2, …) p(y1, y2, …)

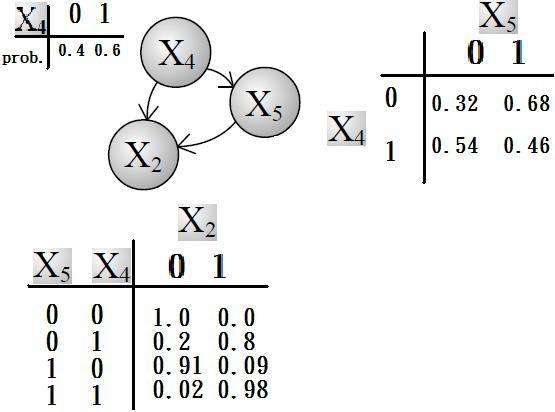
若利用两个假设，则以上优化可以由隐马尔可夫模型估计

## Bayes network (也称为belief network)

表示任意属性间的依赖性，借助有向无环图来刻画属性之间的依赖关系，并使条件概率表来描述属性的联合概率分布

= 结构学习 + 参数学习

为便于贝叶斯网络计算，满足马尔可夫假设（即每个结点仅由相邻的结点有关）



属性的联合概率分布

p(x4, x5, x2) = p(x4)\*p(x5|x4)p(x2|x5,x4)

## 隐马尔可夫模型

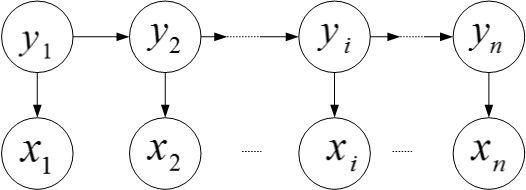
结构最简单的动态贝叶斯网，用于时序数据建模，

两个假设：

1. 马尔可夫假设：系统下一时刻的状态仅由当前状态决定，不依赖于以往的任何状态

p(y1, y2, …) = p(y1)p(y2|y1)…

1. 独立输出假设：系统在每个时刻t输出xt, 且xt仅和yt相关



基于这种依赖关系，所有变量的联合概率分布

p(y1, y2, …, x1, x2, …) = p(y1)p(y2| y1)…p(x2|x1)…

如何训练模型？

Baum-Welch algorithm

**马尔可夫随机场**

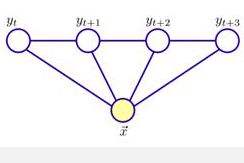
无向图

**条件随机场**

只满足马尔可夫假设（即每个状态仅由相邻的状态有关）

当前观察值不仅与系统当前状态有关，与前后的状态都有关

无向图



# 实际问题

## 多类问题

多分类学习中，每个样本仅属性一个类别。

多标记**(multi-label learning)**学习中，一个样本同时预测出多个类别标记，如一幅图像可同时标注为“蓝天”，“白云”，“羊群”，“自然场景”

基本策略：拆解法，将多分类任务拆为若干个二分类任务求解。 见《机器学习》周志华p63

关键是如何对多分类任务进行拆分，以及如何对多个分类器进行集成。

一对一：将N个类别两两配对，从而产生N(N-1)/2个二分类任务。当为类(yi, yj) 构建分类器，不属于yi or yj的样本忽略。在测试阶段，新样本同时提交给所有分类器，得到N(N-1)/2分类结果，最终结果通过投票产生。

一对多：每次将一个类的样例作为正例，所有其他类作为反例，容易产生不平衡类，从而训练N个分类器。在测试时若仅有一个分类器预测为正类，则对应的类别标记作为最终分类结果，若有多个分类器预测为正类，选择置信度最大的类别标记作为分类结果

多对多：每次将若干个类作为正类，若干个其他类作为反类，需要特殊的设计，不能随意选取，常见的有”纠错输出码”方法

纠错输出编码error-correcting output coding, ECOC

受信息理论中通过噪声信道发送信息的启发。其基本思想是借助于代码字向传输信息中增加一些冗余，从而使得接收方能发现接收信息中的一些错误，而且如果错误量很少，还可能恢复原始信息。

**对于多类学习，每个类yi用一个长度为n的唯一位串来表示，称为它的代码字**。然后训练n个二元分类器，预测代码字串的每个二进位。检验实例的预测类由这样的代码字给出，该代码字到二元分类器产生的代码字海明距离最近。（两个位串之间的海明距离是它们的不同的二进位的数目）

关键是：如何为不同的类设计合适的代码字集合？

对于通信任务，代码字应该最大化各行之间的海明距离，使得纠错可以进行。

对于多类学习要求，将代码字列向和行向的距离很好地分开。较大的列向距离可以确保二元分类器是相互独立的。

|  |  |
| --- | --- |
| 类 | 代码字 |
| y1 | 1 1 1 1 1 |
| y2 | 0 0 0 1 1 |
| y3 | 1 1 1 0 0 |

文献：稀有类，不平衡数据集，代价敏感学习，多类学习

## 不平衡类问题

稀有类的正确分类比多数类的正确分类更有价值。准确率经常用来比较分类器的性能，然而它可能不适合评价从不平衡数据集得到的模型。不平衡类一般用precesion, recall, and F1

检测稀有类的实例好比大海捞针。因为这些实例很少出现，因此描述稀有类的模型趋向于高度特殊性。例如，在基于规则的分类器中，为稀有类提取的规则通常涉及大量的属性，并很难简化为更一般的，具有很高覆盖率的规则。这样的模型很容易受训练数据中噪声的影响。

对于二元分类，稀有类通常记为正类，而多数类被认为是负类。

ROC曲线：显示分类器真正率和假正率之间折中的一种图形化方法。ROC曲线中，真正率沿y轴，假正率沿x轴上，沿着曲线每个点对应于一个分类器

如何解决不平衡类问题？

1. 代价敏感学习

将一个类错分另一个类的代价是不一样的，可以用代价矩阵表达将一个类的记录分类到另一个类的惩罚进行编码　C = [c(i, j)]，

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 预测的类 | |
| 类=+ | 类=- |
| 实际的类 | 类=+ | -1 | 100 |
| 类=- | 1 | 0 |

可以证明　：在0/1代价矩阵中，总代价等于误分类的数目

C = TP\*c(+,+) + FP\*c(-, +) + FN\*c(+,-) + TN\*c(-,-)

　= TP\*0 + FP\*1 + FN\*1 + TN\*0

= FP + FN = N\*Err

医学上，宁可误判为恶性，但不可漏了一个恶性，也就是说假阴性代价远高于假阳性，对应代价矩阵中c(+, -)=100 > c(-, +)=1

如何将代价信息引入分类算法中？

以决策树归纳为例，代价信息修改每个叶结点上的决策规则。

典型的二元分类，若误分代价相同，则将正类批派到结点t

p(+|t) > p(-|t)

=> p(+|t) > 1-p(+|t)

=> p(+|t) > 0.5

从而表明，叶结点的类标号取决于到达该结点的训练记录的多数类

若引入代价信息

C(i|t) = sum( p(j|t)\*c(j, i) ) j=类别

若两类

c(+|t) = p(+|t)\*c(+, +) + p(-|t)\*c(-, +) > c(- |t) = p(+|t)\*c(+, 1) + p(-|t)\*c(-, -)

=> p(+|t) > [c(-,-) - c(-,+)]/[c(+,+) + c(-,-) - c(-,+) - c(+,-)]

1. 基于抽样的方法

降采样：　对多的某类样本执行降采样，从而生成基分类器，多次如此，从而生成多分类器，然后**组合多分类器**

上采样：对少的某类样本执行上采样，如在少样本集中，对某个数据的k近邻，随机生成新的样本。

**上采样缺点：对于噪声数据，上采样可能导致模型过分拟合，因为一些噪声样本也可能被复制多次**。原则上，上采样没有向训练集中添加任何新的信息

How to handle imbalanced learning problems?

Sampling strategies

Over-sampling, Under-sampling, JOUS-Boost

Synthetic data generation

SMOTE, SMOTE-Boost, DataBoost-IM

Cost-sensitive learning

Use a cost-matrix for different types of errors or instance to facilitate learning from imbalanced data sets.

Instance-weighting method, metacost (a general cost-sensitive learning framework)

Active learning

Instead of searching the entire training data space, this method can effectively select informative instances from a random set of training populations

Kernel-based methods.

Reference:

[2008] ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning

**类别不平衡问题** 则《机器学习》周志华p66

一个基本策略：再缩放rescaling

线性分类器，预测出的y值与一个阈值进行比较，通常y>0.5时判别为正例，否则为反例。阈值为0.5表明分类器认为真实正反例可能性相同，即分类器决策规则为若y / (1-y) > 1， 则预测为正例

若类别不平均，要强调类少的重要性，用观测几率代表真实几度，即只要分类器的预测几率高于观测几率就应判定为正例，即若y / ( 1-y) > m+ / m-, 则预测为正例

相当于对预测值进行调整

y’ / (1 – y’) = 1 / (1 – y) \* m+ / m-

rescaling不常用，因为“训练集是真实样本总体的无偏采样”这个假设往往并不成立，也就是说，我们未必能有效地基于训练集观测几率来推断出真实几率。

常采用做法：

1. 欠采样：

丢弃很多反倒，当然不能随机丢弃反倒，因为可能丢失一些重要信息；欠采样法的代表性算法EasyEnsemble[Liu et al., 2009]则是利用集成学习机制，将反倒划分为若干个集合供不同学习器使用，这样对每个学习器来看都进行了欠采样，但在分局来看却不会丢失重要信息。

1. 过采样：

不能简单地对初始正例样本进行重复采样，否则会招致严重的过拟合

阈值移动threshold-moving: 直接基于原始训练集进行学习，但在用训练好的分类器进行预测时，将rescaling嵌入到其决策过程中。

## 在线学习（增量学习 incremental learning）

在学得模型后，再接收到训练样例时，仅需根据新样例对模型进行更新，不必重新训练整个模型，并且先前学得的有效信息不会被“冲掉”

# Scikit Learn

A black background with purple rectangles

Description automatically generated

Source code:

## Split dataset: StratifiedGroupKFold (preferred)

**KFold cross-validation**

A screenshot of a computer

Description automatically generated

**KFold/RepeatedKFold, StratifiedKFold/ RepeatedStratifiedKFold**

**KFold:** Splits it into K folds, trains on K-1 and then tests on the left-out.

from sklearn.model\_selection import KFold

X = np.array([[0., 0.],

[1., 1.],

[-1., -1.],

[2., 2.]])

y = np.array([0, 1, 0, 1])

kf = KFold(n\_splits=2)

for train, test in kf.split(X):

X\_train, X\_test, y\_train, y\_test = X[train], X[test], y[train], y[test]

**RepeatedKFold:** repeats K-Fold n times.

from sklearn.model\_selection import RepeatedKFold

rkf = RepeatedKFold(n\_splits=2, n\_repeats=2, random\_state=random\_state)

for train, test in rkf.split(X):

X\_train, X\_test, y\_train, y\_test = X[train], X[test], y[train], y[test]

**StratifiedKFold**: Same as K-Fold but preserves the class distribution within each fold.

**RepeatedStratifiedKFold**：

from sklearn.model\_selection import StratifiedKFold,RepeatedStratifiedKFold

**ShuffleSplit**: Samples are first shuffled and then split into a pair of train and test sets.

**StratifiedShuffleSplit**: Same as shuffle split but preserves the class distribution within each iteration.

from sklearn.model\_selection import ShuffleSplit, StratifiedShuffleSplit

X = np.arange(10)

ss = ShuffleSplit(n\_splits=5, test\_size=0.25, random\_state=0)

for train\_index, test\_index in ss.split(X):

print("%s %s" % (train\_index, test\_index))

A graph with different colored bars

Description automatically generated with medium confidence

**Leave one out (LOO) and Leave P Out (LPO)**

In terms of accuracy, LOO often results in high variance as an estimator for the test error. Intuitively, since n-1 of the samples are used to build each model, models constructed from folds are virtually identical to each other and to the model built from the entire training set. LOO can overestimate the generalization error. As a general rule, most authors, and empirical evidence, suggest that 5- or 10- fold cross validation should be preferred to LOO.

from sklearn.model\_selection import LeaveOneOut

LeaveOneOut ()

**GroupKFold/GroupShuffleSplit**: the same group is not in both testing and training sets.

An example would be when there is medical data collected from multiple patients, with multiple samples taken from each patient. the patient id for each sample will be its group identifier. we need to ensure that all the samples in the validation fold come from groups that are not represented at all in the paired training fold.

For example if the data is obtained from different subjects with several samples per-subject and if the model is flexible enough to learn from highly person specific features it could fail to generalize to new subjects. GroupKFold makes it possible to detect this kind of overfitting situations.

Each subject is in a different testing fold, and the same subject is never in both testing and training. Notice that the folds do not have exactly the same size due to the imbalance in the data.

from sklearn.model\_selection import GroupKFold, GroupShuffleSplit

X = [0.1, 0.2, 2.2, 2.4, 2.3, 4.55, 5.8, 8.8, 9, 10]

y = ["a", "b", "b", "b", "c", "c", "c", "d", "d", "d"]

groups = [1, 1, 1, 2, 2, 2, 3, 3, 3, 3]

gkf = GroupKFold(n\_splits=3)

for train, test in gkf.split(X, y, groups=groups):

X\_train, X\_test, y\_train, y\_test = X[train], X[test], y[train], y[test]

**StratifiedGroupKFold**: preserve the distribution of classes in each split while keeping each group within a single split. That might be useful when you have an unbalanced dataset so that using just GroupKFold might produce skewed splits.

from sklearn.model\_selection import StratifiedGroupKFold

X = list(range(18))

y = [1] \* 6 + [0] \* 12

groups = [1, 2, 3, 3, 4, 4, 1, 1, 2, 2, 3, 4, 5, 5, 5, 6, 6, 6]

sgkf = StratifiedGroupKFold(n\_splits=3)

for train, test in sgkf.split(X, y, groups=groups):

X\_train, X\_test, y\_train, y\_test = X[train], X[test], y[train], y[test]

LeaveOneGroupOut: each split holds out samples belonging to one specific group

LeavePGroupsOut

**Time Series Split:** first k folds as train set and the k+1 th fold as test set.

A graph with different colored bars

Description automatically generated

from sklearn.model\_selection import TimeSeriesSplit

X = np.array([[1, 2],

[3, 4],

[1, 2],

[3, 4],

[1, 2],

[3, 4]])

y = np.array([1, 2, 3, 4, 5, 6])

tscv = TimeSeriesSplit(n\_splits=3)

for train, test in tscv.split(X):

print("%s %s" % (train, test))

output:

[0 1 2] [3]

[0 1 2 3] [4]

[0 1 2 3 4] [5]

## Preprocess data

Select features and output

Fill missing values and normalize the features

from sklearn.preprocessing import MinMaxScaler

from sklearn.impute import SimpleImputer

**# normalize the features (continuous numetric)**

scaler = MinMaxScaler()

scaler.fit(features\_train)

features\_train = scaler.transform(features\_train)

features\_test = scaler.transform(features\_test)

**# Fill missing values**

imputer = SimpleImputer(missing\_values=-1, strategy='median')

imputer.fit(labels\_train)

labels\_train = imputer.transform(labels\_train)

labels\_test = imputer.transform(labels\_test)

**# convert category to onehot for dataframe**

for col in df\_train.columns:

df\_train[col] = df\_train[col].astype('category')

categories = df\_train[col].cat.categories

df\_test[col] = pd.Categorical(df\_test[col], categories)

df\_train\_dummies = pd.get\_dummies(df\_train)

df\_test\_dummies = pd.get\_dummies(df\_test)

**# convert y to multilabel (sample, n\_classes),**

from sklearn.preprocessing import MultiLabelBinarizer

X = [[1, 2], [2, 4], [4, 5], [3, 2], [3, 1]]

y = [[0, 1], # 表示sample 1 属于类0 and 类1

[0, 2], # 表示sample 1 属于类0 and 类2

[1, 3],

[0, 2, 3],

[2, 4]]

y = MultiLabelBinarizer().fit\_transform(y)

[[1 1 0 0 0], # 有几类，就有几列，0表示不属于，1表示属于该类

[1 0 1 0 0],

[0 1 0 1 0],

[1 0 1 1 0],

[0 0 1 0 1]]

classif.fit(X, y).predict(X)

## Estimator (classification, regression, cluster, transformer)

An **estimator** is any object that learns from data

classification,

regression

clustering algorithm

transformer that extracts/filters useful features from raw data.

scikit-learn 中所有实现机器学习方法的类都来自BaseEstimator,

.fit(x, y)

.predict(x)

.predict\_proba(x) 预测的概率

.predict\_log\_proba(x) 预测概率的对数

# Loading an example dataset

from sklearn import datasets

iris = datasets.load\_iris()

digits = datasets.load\_digits() #手写体

iris\_X, iris\_y = digits.data, digits.target

# split train and test dataset

np.random.seed(0)

indices = np.random.permutation(len(iris\_X))

iris\_X\_train, iris\_y\_train = iris\_X[indices[:-10]], iris\_y[indices[:-10]]

iris\_X\_test,iris\_y\_test = iris\_X[indices[-10:]], iris\_y[indices[-10:]]

# Learning and predicting

from sklearn import svm

clf = svm.SVC(gamma=0.001, C=100.)

clf.fit(digits.data[:-1], digits.target[:-1])

clf.predict(digits.data[-1:])

# Model persistence

import pickle

s = pickle.dumps(clf) #to a string

clf2 = pickle.loads(s)

from sklearn.externals import joblib # more efficient on big data

joblib.dump(clf, 'filename.pkl') #can only to file

clf = joblib.load('filename.pkl')

Note:

Unless otherwise specified, input will be cast to float64

Regression targets are cast to float64, classification targets are maintained

clf.fit(iris.data, iris.target)

list(clf.predict(iris.data[:3])) #output is [0, 0, 0]

clf.fit(iris.data, iris.target\_names[iris.target])

list(clf.predict(iris.data[:3])) #['setosa', 'setosa', 'setosa']

Refitting and updating parameters

clf.set\_params(kernel='linear').fit(X, y)

clf.predict(X\_test)

clf.set\_params(kernel='rbf').fit(X, y)

clf.predict(X\_test)

**Cross-validated estimators**

Cross-validation: evaluating estimator performance estimators that set their parameter automatically by cross-validation, these estimators are called similarly to their counterparts, with ‘CV’ appended to their name.

from sklearn import linear\_model, datasets

lasso = linear\_model.LassoCV()

diabetes = datasets.load\_diabetes()

X\_diabetes = diabetes.data

y\_diabetes = diabetes.target

lasso.fit(X\_diabetes, y\_diabetes)

lasso.alpha\_ # The estimator chose automatically its lambda:

**Pipelining: combined estimators**

from sklearn.pipeline import Pipeline

from sklearn import linear\_model, decomposition

from sklearn.model\_selection import GridSearchCV

logistic = linear\_model.LogisticRegression()

pca = decomposition.PCA()

pipe = Pipeline(steps=[('pca', pca), ('logistic', logistic)])

# Prediction

n\_components = [20, 40, 64]

Cs = np.logspace(-4, 4, 3)

# Parameters of pipelines can be set using ‘\_\_’ separated parameter names:

estimator = GridSearchCV(pipe, dict(pca\_\_n\_components=n\_components, logistic\_\_C=Cs))

estimator.fit(X\_digits, y\_digits)

estimator.best\_estimator\_.named\_steps['pca'].n\_components

### Supervise-learning: Regression

data format: X = (n\_samples, n\_features), y = (n\_samples, k), while k>1 for multi-out

key:

normalize(X) like MinMaxScaler, StandardScaler

the output y is not necessarily transformed, like predict time 20s, or house price 200K.

**Appication:**

predict AP onset time and peak time

step 1: CNN+LSTM+classifier, like keypoint detection, input is loop, output is gaussian(or triangle function) signal, value around onset time, others 0. (same to task peak time)

the input is video features extracted by DL model (video len, 1) + [fps, image start index], output is onset time. (same to task peak time)

Regression: map feature (video len, 1) + [fps, image start index] to onset time

Features = [[f1, f2, f3, …, fn],

[f1, f2, f3, …, fn],

… ]] with (m, n)

Output = [t1, t2, …, tm] with (m, 1) while m is the sample number

**Refer to section Gradient Boosting regression which has best result**

Knowledge engineering for feature transformation:

output is gaussian function of image index

if input is gaussian signal (outputted by DL), then is transformed into image index via inversed gaussian function: feature\_onset = sqrt(-ln(feature\_onset))

#### Linear Regression

A line with dots and a blue line

Description automatically generated



**LinearRegression**

The least squares solution is computed using the singular value decomposition of X and assume sample number is larger than feature number.

The coefficient estimates for Ordinary Least Squares rely on the independence of the features. When features are correlated and the columns of the feature matrix have an approximately linear dependence, the feature matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed target, producing a large variance.

diabetes = datasets.load\_diabetes() # Diabetes dataset (442 patients, 10 physiological)

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train).score(diabetes\_X\_test, diabetes\_y\_test)

**Ridge:** LinearRegression + L2 regularization (alpha越大，权重越小)

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:



The complexity parameter α>=0 controls the amount of shrinkage: the larger α, the greater shrinkage and thus the coefficients become more robust to collinearity.

Shrinkage is a form of regularization used to improve the estimation of covariance matrices in situations where training samples number is small compared to features number.

regr = linear\_model.Ridge(alpha=.1)

RidgeCV implement ridge regression with built-in cross-validation of the alpha parameter, It work in the same way as GridSearchCV except that it defaults to efficient Leave-One-Out cross-validation.

reg = linear\_model.RidgeCV(alphas=np.logspace(-6, 6, 13))

reg.fit(X\_train, y\_train)

reg = linear\_model.RidgeCV(alphas=np.logspace(-6, 6, 13), cv=10) # 10-fold cross-validation

**Lasso:** LinearRegression + L1 regularization (alpha越大，权重越稀疏)

The Lasso is a linear model that estimates sparse coefficients, least absolute shrinkage and selection operator, can set some coefficients to zero. Such methods are called sparse method and sparsity can be seen as an application of Occam’s razor: prefer simpler models.

As the Lasso regression yields sparse models, it can thus be used to perform feature selection, as detailed in L1-based feature selection.

Mathematically, it consists of a linear model with an added regularization term. The objective function to minimize is:



The parameter α controls the degree of sparsity of the estimated coefficients.

regr = linear\_model.Lasso(alpha=0.1)

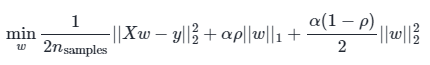
# For high-dimensional datasets with many collinear features

regr = linear\_model. LassoCV(alpha=0.1)

**Elastic-Net = Ridge + Lasso**

Elastic-net is useful when there are multiple features that are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

The objective function to minimize is in this case:



regr = linear\_model. enet\_path(alpha=0.1, l1\_ratio=0.8)

regr = linear\_model. ElasticNetCV(alpha=0.1, l1\_ratio=0.8)

**SGD : Stochastic Gradient Descent**

It is particularly useful when samples number (>10^5 training examples) and features number (>10^5 features) is very large.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing.

SGD is merely an optimization technique to fit linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

For example, using SGDClassifier(loss='log\_loss') results in logistic regression, i.e. a model equivalent to LogisticRegression which is fitted via SGD instead of being fitted by one of the other solvers in LogisticRegression. Similarly, SGDRegressor (loss='squared\_error', penalty='l2') and Ridge solve the same optimization problem, via different means.

The disadvantage: 1) SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations. 2) SGD is sensitive to feature scaling.

Note:

scale each attribute on the input vector X to [0,1] or [-1, +1], or standardize it to have mean 0 and variance 1

Empirically, we found that SGD converges after observing approximately 10^6 training samples. Thus, a reasonable first guess for the number of iterations is max\_iter = np.ceil(10\*\*6 / n), where n is the size of the training set.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train) # Don't cheat - fit only on training data

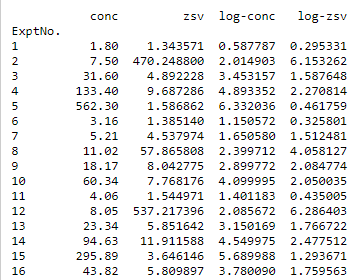
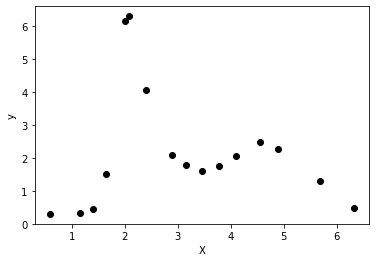
X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test) # apply same transformation to test data

SGDRegressor (loss='squared\_error', penalty='l2')

#### Nonlinear regression

Application: fit curve as shown in figure

Nonlinear models are harder to fit, harder to interpret, and more complex than linear models. You should think hard about whether a nonlinear model is the right next step forward. Some alternatives include using better features with linear models.

Two methods:

* transform X with domain knowledge or poly(X), then non-linear regression will become linear regression

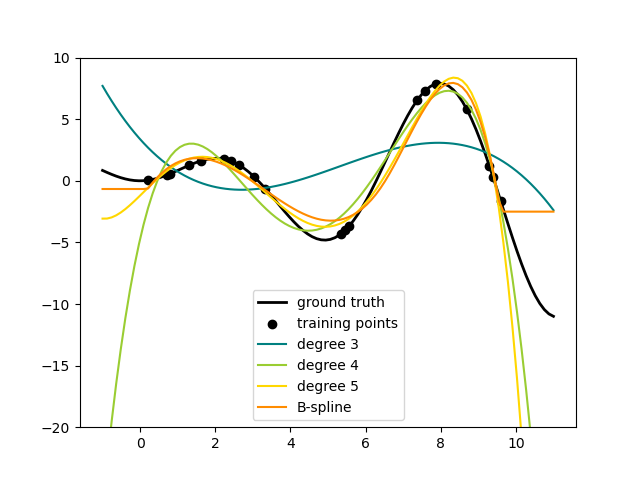
e.g.,

images -> gaussian bell -> image idx

gauss\_pred = model(images)

image\_idx = inverse\_guass(gauss\_pred)

e.g., 所有的非线性拟合都可以用将特征通过多项式变换，然后用线性拟合解决



If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:



We could still use linear model if we create a new set of features:



With this re-labeling of the data, our problem can be written:



* nonlinear model (including kernel + linear model)

**Polynomial regression: extending linear models with basis functions**

X = [ [f1, f2],

[f1, f2],

…]

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)

X\_new = poly.fit\_transform(X)

The features of X have been transformed from [x1, x2] to [1, x1, x2, x12, x1x2, x22], and can now be used within any linear model

**Neural networks**

Not every fit is successful. When I run this several times, some work, and some don't.

Hyperparameters:

* The number of nodes in each layer and The number of layers
* The activation function
* What kind of regularization to use
* Which optimizer and its tuning parameters
* How to standardize the data input, and what data input to us

from sklearn.neural\_network import MLPRegressor

model = MLPRegressor(hidden\_layer\_sizes=(3,), activation='tanh', solver='lbfgs')

**Gradient Boosting regression (preferred)**

Example:<https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html>

from sklearn import datasets, ensemble

params = {

"n\_estimators": 500,

"max\_depth": 4,

"min\_samples\_split": 5,

"learning\_rate": 0.01,

"loss": "squared\_error",

}

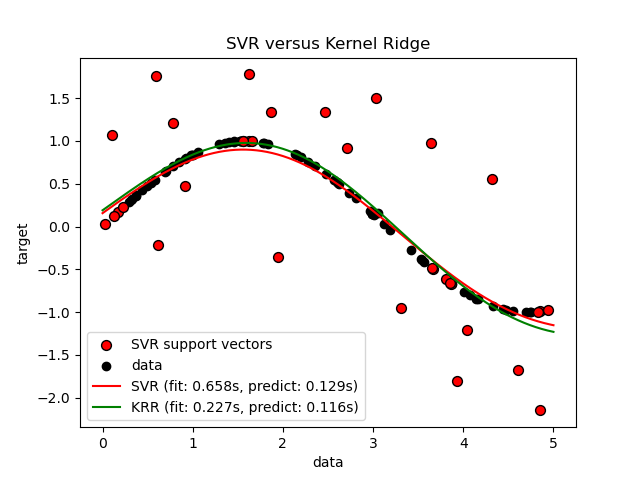
reg = ensemble.GradientBoostingRegressor(\*\*params)

reg.fit(X\_train, y\_train)

**SVR (Support Vector Regression) and Kernel ridge regression**

Both kernel ridge regression (KRR) and SVR learn a non-linear function by employing the kernel trick, i.e., they learn a linear function in the space induced by the respective kernel which corresponds to a non-linear function in the original space.

In contrast to SVR, fitting a KRR can be done in closed-form and is typically faster for medium-sized datasets. On the other hand, the learned model is non-sparse and thus slower than SVR at prediction-time.



from sklearn.kernel\_ridge import KernelRidge

from sklearn.svm import SVR

from sklearn.model\_selection import GridSearchCV

svr = GridSearchCV(

SVR(kernel="rbf", gamma=0.1),

param\_grid={"C": [1e0, 1e1, 1e2, 1e3], "gamma": np.logspace(-2, 2, 5)},

)

svr.fit(X\_train, y\_train)

kr = GridSearchCV(

KernelRidge(kernel="rbf", gamma=0.1),

param\_grid={"alpha": [1e0, 0.1, 1e-2, 1e-3], "gamma": np.logspace(-2, 2, 5)},

)

**Gaussian Process Regression**

Gaussian Processes (GP) are a nonparametric supervised learning method used to solve regression and probabilistic classification problems.

The prediction interpolates the observations (at least for regular kernels).

from sklearn.gaussian\_process import GaussianProcessRegressor

from sklearn.gaussian\_process.kernels import RBF

gp = GaussianProcessRegressor(alpha=1e-1, kernel= RBF())

gp = gp.fit(X, y)

# Test data

x1 = np.linspace(-1, 8)

y1, y1\_std = gp.predict(x1[:, np.newaxis], return\_std=True)

plt.plot(X, y, 'ko', label = 'Training Data')

plt.plot(x1, y1, 'b-', label = "Predicted Function Mean")

y1 = y1.flatten()

plt.fill\_between(x1, y1 - y1std, y1 + y1std, alpha=0.3, color='k', label="Uncertainty")

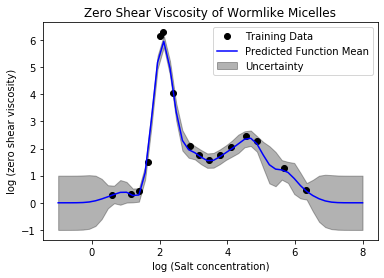


Figure. Unlike other models, this model also provides an estimate of the uncertainty on the predictions y1std. With the uncertainty regions along the function, we would know which areas would require more data and can target gathering data (in this case through experiments) only in the regions of high uncertainty. The final acceptable uncertainty in the model is upto the user to decide.

Adjust hyperparameter:

kernel = RBF(length\_scale\_bounds = (0.1, 1.0))

gp = GaussianProcessRegressor(alpha = 0.1, kernel = kernel)

gp = gp.fit(X, y)

Note:

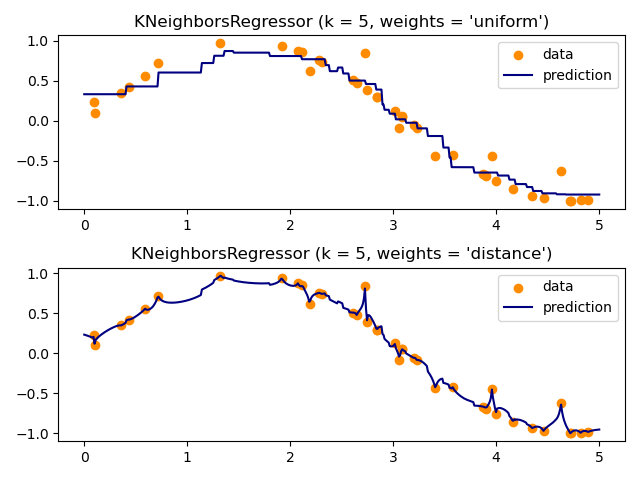
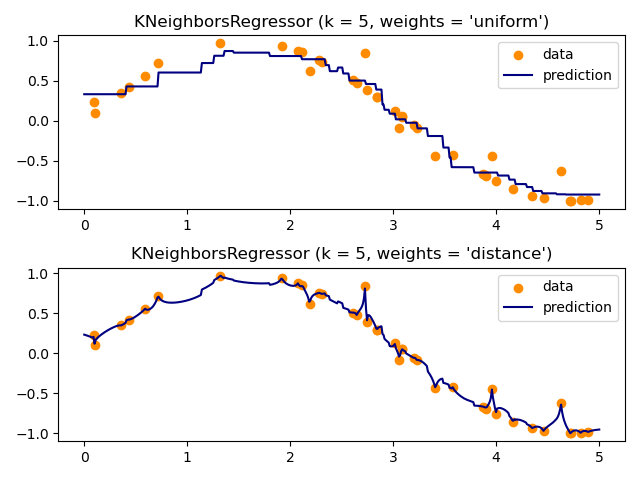
The RBF is a generic kernel that always works when you have enough data. It is similar to using tanh in a neural network, which also always works when you have enough data.

The cost of evaluating a NN is a constant and does not depend on the size of the training data. In contrast, the cost of evaluating a GP depends on the size of the training data because you have to compute the correlation between the new point and all of the existing points. This can be expensive for large data sets.

**Nearest Neighbors Regression (refers to Nearest Neighbors Classification)**

Neighbors-based is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data.

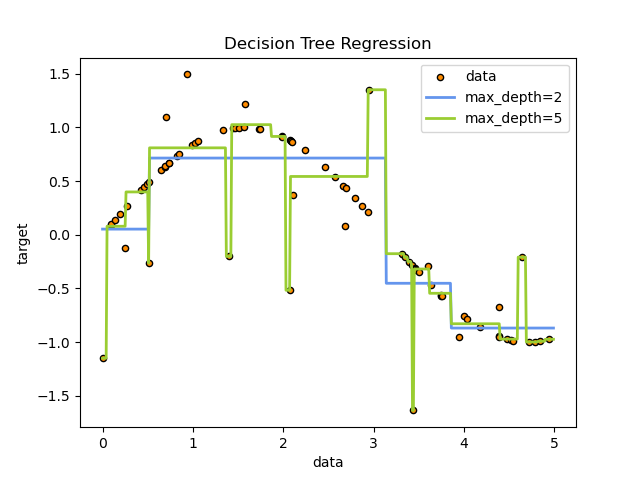
The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.



**Decision Trees**

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

A tree can be seen as a piecewise constant approximation. The deeper the tree, the more complex the decision rules and the fitter the model.



The advantages:

* Simple to understand and to interpret. Trees can be visualized.
* Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Some tree and algorithm combinations support missing values.
* The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
* Able to handle both numerical and categorical data. However, the scikit-learn implementation does not support categorical variables for now.
* Able to handle multi-output problems.

The disadvantages:

* Decision-tree learners is easy overfitting. Hyperparameters: pruning, setting the minimum samples number required at a leaf node or setting the maximum depth of the tree.
* Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. Prefer decision tree ensemble.
* Predictions of decision trees are neither smooth nor continuous, but piecewise constant. Therefore, they are not good at extrapolation.
* The problem of learning an optimal decision tree is known to be NP-complete. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
* Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Tips on practical use:

Decision trees tend to overfit on data with a large number of features. Consider performing dimensionality reduction (PCA, ICA, or Feature selection) beforehand

Use max\_depth (initial value: 3) to control the size of the tree to prevent overfitting

Use min\_samples\_split (initial value: 1) or min\_samples\_leaf (initial value: 5) to ensure that multiple samples inform every decision in the tree

from sklearn import tree

clf = tree.DecisionTreeRegressor()

**Gradient-boosted trees**

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is an excellent model for both regression and classification, in particular for tabular data.

Hyperparameter: tree number (n\_estimators), learning\_rate, control tree size (max\_leaf\_nodes, max\_depth, min\_samples\_leaf). The l2\_regularization parameter acts as a regularizer for the loss function

Monotonic Constraints(单调性约束): you may have prior knowledge indicating that a given feature should in general have a positive (or negative) effect on the target value. For example, all else being equal, a higher credit score should increase the probability of getting approved for a loan.

In a binary classification context, imposing a monotonic increase (decrease) constraint means that higher values of the feature are supposed to have a positive (negative) effect on the probability of samples to belong to the positive class.

# monotonic increase, monotonic decrease, and no constraint on the 3 features

# 特征1变大，输出变大；特征2变大，输出变小；

from sklearn.ensemble import GradientBoostingRegressor

GradientBoostingRegressor(monotonic\_cst=[1, -1, 0])

**Random Forests** (refer to classification)

from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max\_depth=2, random\_state=0)

**Extremely Randomized Trees** (refer to classification)

from sklearn.ensemble import ExtraTreesRegressor

reg = ExtraTreesRegressor(n\_estimators=100, random\_state=0).fit(

X\_train, y\_train)

**Voting Regressor** (refer to classification)

return the average predicted values

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import VotingRegressor

reg1 = GradientBoostingRegressor(random\_state=1)

reg2 = RandomForestRegressor(random\_state=1)

reg3 = LinearRegression()

ereg = VotingRegressor(estimators=[('gb', reg1), ('rf', reg2), ('lr', reg3)])

**Stacked generalization** (StackingClassifier or StackingRegressor)

combining estimators to reduce their biases.

the predictions of each individual estimator are stacked together and used as input to a final estimator to compute the prediction. This final estimator is trained through cross-validation.

In practice, a stacking predictor predicts as good as the best predictor of the base layer and even sometimes outperforms it by combining the different strengths of the these predictors. However, training a stacking predictor is computationally expensive.

from sklearn.linear\_model import RidgeCV, LassoCV

from sklearn.neighbors import KNeighborsRegressor

estimators = [('ridge', RidgeCV()),

('lasso', LassoCV(random\_state=42)),

('knr', KNeighborsRegressor(n\_neighbors=20,

metric='euclidean'))]

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.ensemble import StackingRegressor

final\_estimator = GradientBoostingRegressor(

n\_estimators=25, subsample=0.5, min\_samples\_leaf=25, max\_features=1,

random\_state=42)

reg = StackingRegressor(

estimators=estimators,

final\_estimator=final\_estimator)

reg.fit(X\_train, y\_train)

During training, the estimators are fitted on the whole training data X\_train. To generalize and avoid over-fitting, the final\_estimator is trained on out-samples using sklearn.model\_selection.cross\_val\_predict internally.

AdaBoost (refer to classification)

sklearn.ensemble.AdaBoostRegressor

#### Multioutput regression

multiple regression problems jointly: y shape (n\_samples, n\_tasks)

Features = [[f1, f2, f3, …, fn],

[f1, f2, f3, …, fn],

… ]] with (m, n)

Output = [[t1, t2],

…,

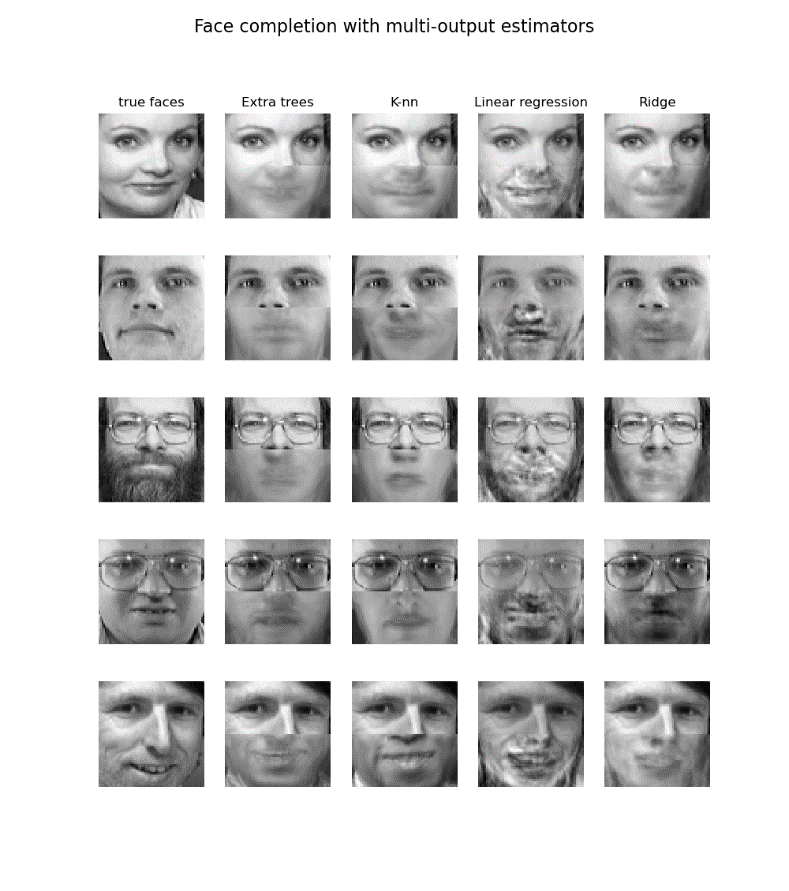
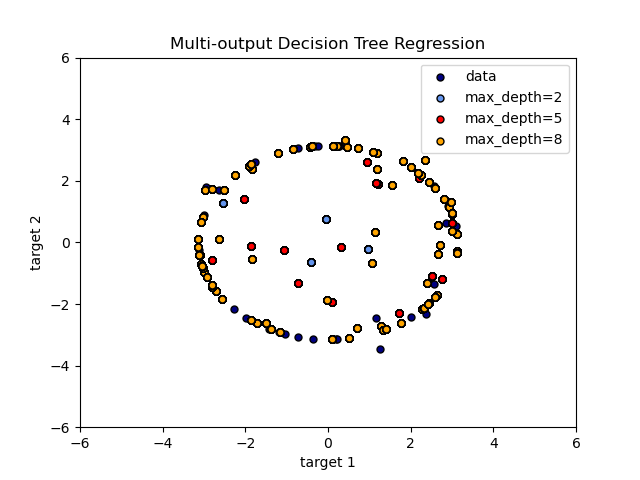
[t1, t2] with (m, 2) while m is the sample number

A multi-output problem is a supervised learning problem with several outputs to predict, that is when Y is a 2d array of shape (n\_samples, n\_outputs).

When there is no correlation between the outputs, a very simple way to solve this kind of problem is to build n independent models, i.e. one for each output, and then to use those models to independently predict each one of the n outputs. However, because it is likely that the output values related to the same input are themselves correlated, an often-better way is to build a single model capable of predicting simultaneously all n outputs. First, it requires lower training time since only a single estimator is built. Second, the generalization accuracy of the resulting estimator may often be increased.

Application:

* 1. the input X is a single real value and the outputs Y are the sine and cosine of X.



* 1. the inputs X are the pixels of the upper half of faces and the outputs Y are the pixels of the lower half of those faces.

cross\_decomposition.CCA

dummy.DummyRegressor

linear\_model.ElasticNet, Lars, Lasso, LassoLars, LinearRegression, MultiTaskElasticNet,

MultiTaskElasticNetCV, MultiTaskLasso, MultiTaskLassoCV, OrthogonalMatchingPursuit,

RANSACRegressor, Ridge, RidgeCV

tree.ExtraTreeRegressor, DecisionTreeRegressor

ensemble.ExtraTreesRegressor, RandomForestRegressor

gaussian\_process.GaussianProcessRegressor

neighbors.KNeighborsRegressor, RadiusNeighborsRegressor

kernel\_ridge.KernelRidge

multioutput.MultiOutputRegressor, RegressorChain

cross\_decomposition.PLSCanonical, PLSRegression

compose.TransformedTargetRegressor

from sklearn.tree import DecisionTreeRegressor

regr = DecisionTreeRegressor(max\_depth=8)

regr.fit(X\_train, y\_train)

from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(n\_estimators=100, max\_depth=30, random\_state=0)

from sklearn.linear\_model import MultiTaskLasso

MultiTaskLasso(alpha=1.0)

from sklearn.linear\_model import MultiTaskElasticNet, MultiTaskElasticNetCV

regr = MultiTaskElasticNet(alpha=[0.1, 0.1], l1\_ratio=[0.8, 0.8])

regr = MultiTaskElasticNetCV(alpha=[0.1, 0.1], l1\_ratio=[0.8, 0.8])

reg = LassoLars(alpha=.1)

reg = BayesianRidge()

**Multioutput + any regressor for multi regressor**

**Multioutput** regression support can be added to any regressor with MultiOutputRegressor. This strategy consists of fitting one regressor per target. Since each target is represented by exactly one regressor it is possible to gain knowledge about the target by inspecting its corresponding regressor. As MultiOutputRegressor fits one regressor per target it can not take advantage of correlations between targets.

regr = MultiOutputRegressor(GradientBoostingRegressor(random\_state=0))

regr.fit(X, y).predict(X)

**finetune hyper-parameter :**

pipe = Pipeline([('Regressor', MultiTaskElasticNet ())])

params = [{

'Regressor': [linear\_model.MultiTaskElasticNet()],

"Regressor\_\_alpha": np.logspace(1, 3, 3),

"Regressor\_\_l1\_ratio": np.linspace(0, 1, 5), }]

optimize = BayesSearchCV(pipe, params, scoring='neg\_mean\_absolute\_error',

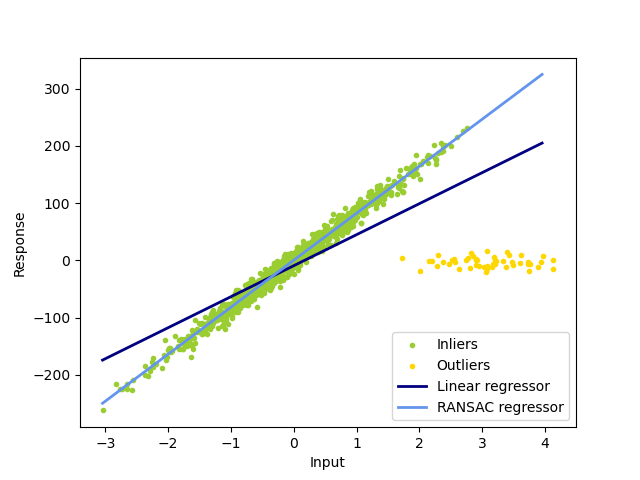
n\_iter=50\*len(params), cv=3, n\_jobs=-1)

optimize.fit(X\_train, y\_train)

model = optimize.best\_estimator\_

#### Robustness regression: outliers and modeling errors

The ordinary linear regressor is sensitive to outliers, and the fitted line can easily be skewed away from the true underlying relationship of data. As shown in following figure.

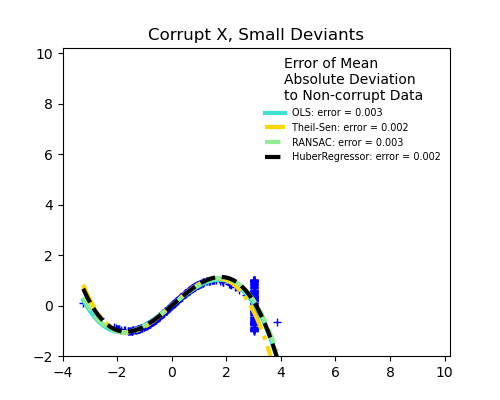
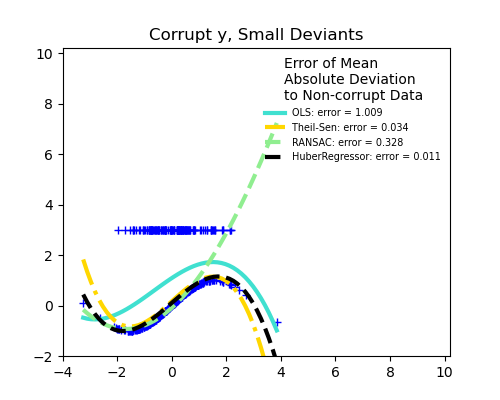


Robust regression aims to fit a regression model in the presence of corrupt data: either outliers, or error in the model.

Note that in general, robust fitting in high-dimensional setting (large n\_features) is very hard.

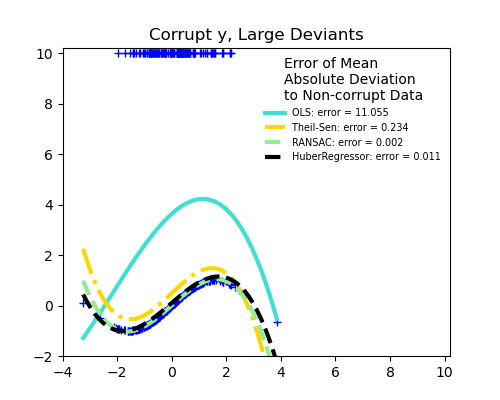
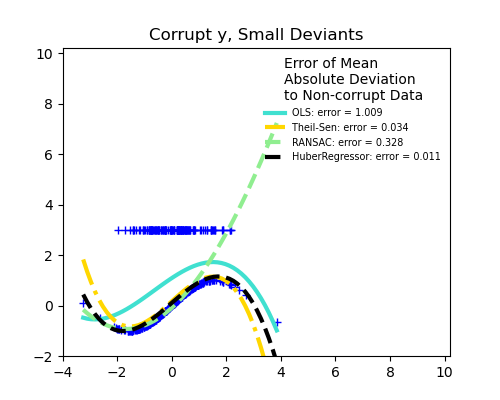
When dealing with data corrupted by outliers, two questions `Outliers in X or in y?` and `how much outliers` are very important.

Outliers in X or in y?



Fraction of outliers versus amplitude of error

The number of outlying points matters, but also how much they are outliers.



3 robust regression estimators: **RANSAC, Theil Sen and HuberRegressor.**

HuberRegressor for the default parameters is most robust.

RANSAC will deal better with large outliers in the y direction (most common situation).

Theil Sen will cope better with medium-size outliers in the X direction, but this property will disappear in high-dimensional settings.

**RANSAC: RANdom SAmple Consensus**

RANSAC fits a model from random subsets of inliers from the complete data set.

The algorithm splits the complete input sample data into a set of inliers, which may be subject to noise, and outliers, which are e.g. caused by erroneous measurements or invalid hypotheses about the data. The resulting model is then estimated only from the determined inliers.

from sklearn import linear\_model

ransac = linear\_model.RANSACRegressor() # Robustly fit linear model with RANSAC algorithm

ransac.fit(X, y)

inlier\_mask = ransac.inlier\_mask\_

outlier\_mask = np.logical\_not(inlier\_mask)

**Theil-Sen estimator: generalized-median-based estimator**

The TheilSenRegressor estimator uses a generalization of the median in multiple dimensions. It is thus robust to multivariate outliers.

It loses its robustness properties and becomes no better than an ordinary least squares in high dimension

**Huber Regression**

The HuberRegressor applies a linear loss to samples that are classified as outliers. A sample is classified as an inlier if the absolute error of that sample is lesser than a certain threshold. It differs from TheilSenRegressor and RANSACRegressor because it does not ignore the effect of the outliers but gives a lesser weight to them.

from sklearn.linear\_model import HuberRegressor, RANSACRegressor, TheilSenRegressor,

from sklearn.pipeline import make\_pipeline

from sklearn.preprocessing import PolynomialFeatures

model = make\_pipeline(PolynomialFeatures(3), TheilSenRegressor(random\_state=42))

model = make\_pipeline(PolynomialFeatures(3), RANSACRegressor(random\_state=42))

model = make\_pipeline(PolynomialFeatures(3), HuberRegressor (random\_state=42))

model.fit(this\_X, this\_y)

mse = mean\_squared\_error(model.predict(X\_test), y\_test)

### Supervise-learning: Classification

data format:

X = (n\_samples, n\_features),

y = (n\_samples, k), while k>1 for multi-out

For multiclass classification, the problem is treated as multi-output regression, and the predicted class corresponds to the output with the highest value.

Benchmark:

**DummyClassifier**: predicts the most prevalent class in a dataset

from sklearn.dummy import DummyClassifier

dummy\_clf = DummyClassifier()

# X\_train可以设为y\_train, 因为dummy分类器不会用到X\_train

y\_pred = dummy\_clf.fit(X\_train, y\_train).predict(X\_test)

from sklearn.metrics import accuracy\_score

acc = accuracy\_score(y\_test, y\_pred)

from sklearn.metrics import classification\_report

classification\_report(y\_test, y\_pred)

#### Binaryclass: (sample, feature dim) -> (sample, )

X = [[1, 2],

[2, 4],

[4, 5],

[3, 2],

[3, 1]]

y = [0, 0, 1, 1, 0]

**Logistic regression:** maximum-entropy classification (MaxEnt) or the log-linear classifier

This implementation can fit binary, One-vs-Rest, or multinomial logistic regression with optional

L1, l2 or Elastic-Net regularization.



The objective function:



A screenshot of a math problem

Description automatically generated

Note that the scale of the class weights and the sample weights will influence the optimization problem. For instance, multiplying the sample weights by a constant b>0 is equivalent to multiplying the (inverse) regularization strength C by b.

from sklearn.linear\_model import LogisticRegression

clf\_l1\_LR = LogisticRegression(C=C, penalty="l1", tol=0.01)

clf\_l2\_LR = LogisticRegression(C=C, penalty="l2", tol=0.01)

clf\_en\_LR = LogisticRegression(C=C, penalty="elasticnet", l1\_ratio=l1\_ratio, tol=0.01)

LogisticRegressionCV(…)

RidgeClassifierCV implements ridge classification with built-in cross-validation of the alpha parameter (refers to Regression/Ridge for more info).

**SGD: Stochastic Gradient Descent**

It is particularly useful when the number of samples (and the number of features) is very large. The partial\_fit method allows online/out-of-core learning.

from sklearn.linear\_model import SGDClassifier

clf = SGDClassifier(max\_iter=1000, tol=1e-3)

Linear and Quadratic Discriminant Analysis

LDA (Linear Discriminant Analysis) can only learn linear boundaries, while Quadratic Discriminant Analysis can learn quadratic boundaries and is therefore more flexible.

LDA can be used to perform supervised dimensionality reduction, by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes. The dimension of the output is necessarily less than the number of classes, so this is in general a rather strong dimensionality reduction, and only makes sense in a multiclass setting.

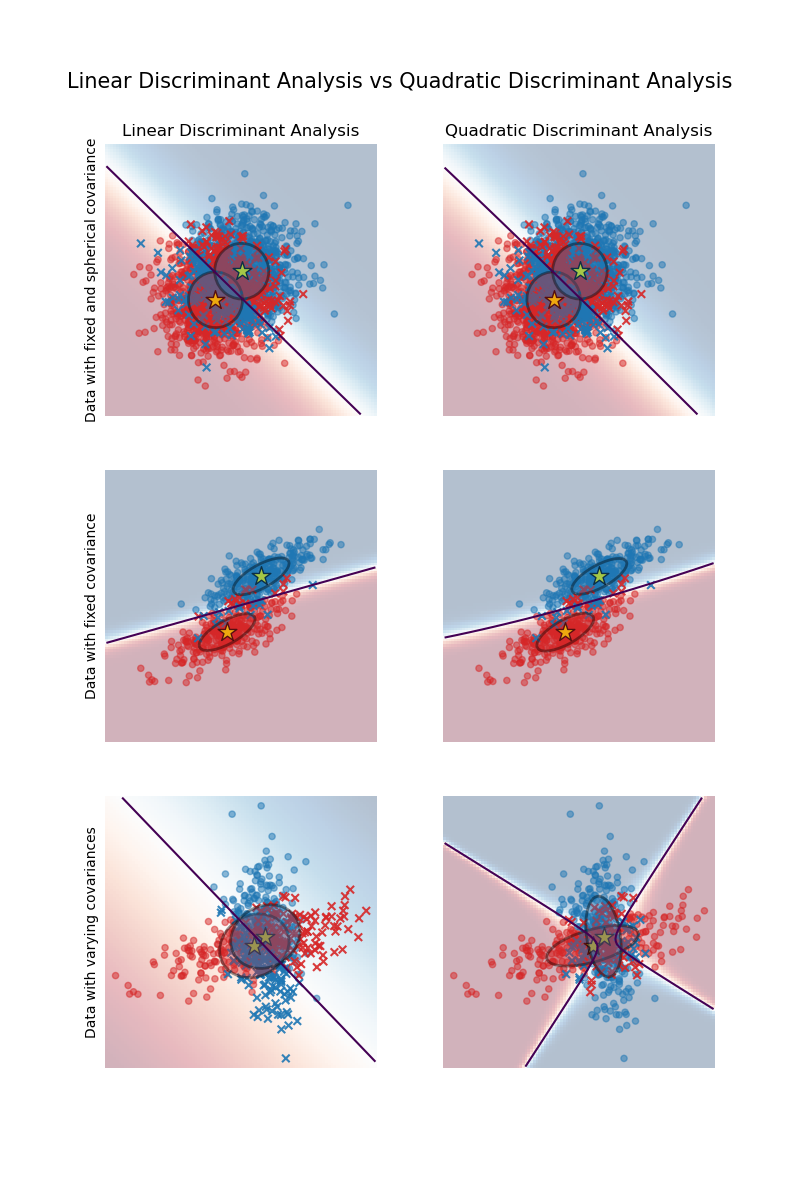


Figure. LDA (left) and Quadratic DA (right)

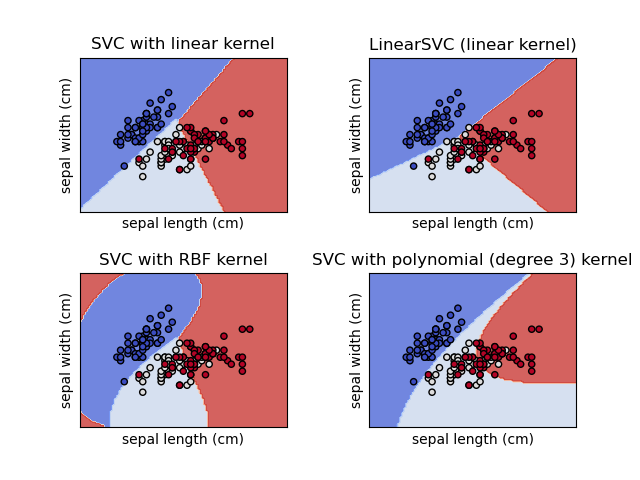
**Support vector machines (SVMs)**

SVC (Support Vector Classification), NuSVC and LinearSVC are classes capable of performing binary and multi-class classification on a dataset.

Support Vector Machines belong to the discriminant model family: they try to find a combination of samples to build a plane maximizing the margin between the two classes. Regularization is set by the C parameter**:** a small value for C means the margin is calculated using many or all of the observations around the separating line (more regularization); a large value for C means the margin is calculated on observations close to the separating line (less regularization).

Classes are not always linearly separable in feature space. The solution is to build a decision function that is not linear but may be polynomial instead. This is done using the kernel trick that can be seen as creating a decision energy by positioning kernels on observations

SVMs decision function depends on some subset of the training data, called the support vectors.



from sklearn import svm

clf = svm.SVC(kernel='linear') # 线性可分

clf = svm.SVC(kernel='poly', degree=3) # 非线性可分

clf = svm.SVC(kernel='rbf') # 非线性可分

clf.fit(X, y)

clf.support\_vectors\_ # get support vectors

Unbalanced problems: In problems where it is desired to give more importance to certain classes or certain individual samples, the parameters class\_weight and sample\_weight can be used.

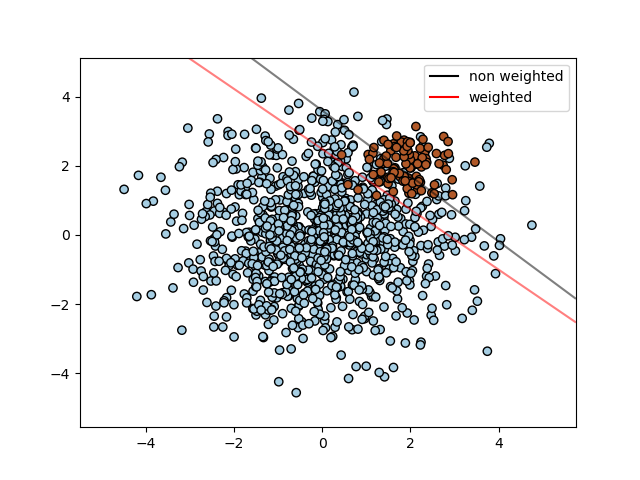
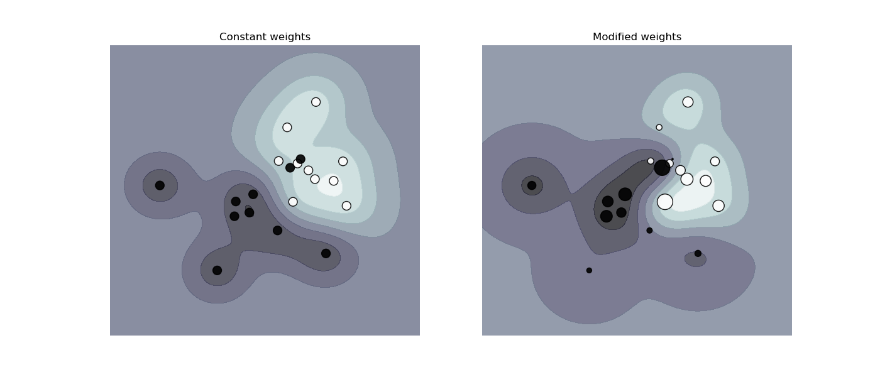
 

Figure. the decision boundary of an unbalanced problem, with and without weight correction (left), the effect of sample weighting on the decision boundary. The size of the circles is proportional to the sample weights:

wclf = svm.SVC(kernel="linear", class\_weight={1: 10})

wclf.fit(X, y)

The sample weighting rescales the C parameter, which means that the classifier puts more emphasis on getting these points right. The effect might often be subtle.

sample\_weight\_last\_ten = abs(np.random.randn(len(X)))

sample\_weight\_last\_ten[15:] \*= 5 # add bigger weights to some outliers

sample\_weight\_last\_ten[9] \*= 15 # add bigger weights to some outliers

clf\_weights = svm.SVC(gamma=1)

clf\_weights.fit(X, y, sample\_weight=sample\_weight\_last\_ten)

#### Multiclass: (sample, feature dim) -> (sample, )

X = [[1, 2],

[2, 4],

[4, 5],

[3, 2],

[3, 1]]

y = [0, 0, 1, 1, 2]

It is possible to parameterize a K-class classification model using only K-1 weight vectors, leaving one class probability fully determined by the other class probabilities by leveraging the fact that all class probabilities must sum to one. We deliberately choose to overparameterize the model using K weight vectors for ease of implementation.

**Inherently multiclass:**

from sklearn.linear\_model import LogisticRegression, LogisticRegressionCV,

RidgeClassifier, RidgeClassifierCV

from sklearn.naive\_bayes import BernoulliNB, GaussianNB

from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier

from sklearn.ensemble.ExtraTreesClassifier, RandomForestClassifier

from sklearn. neural\_network import MLPClassifier

from sklearn.neighbors import KNeighborsClassifier, NearestCentroid, RadiusNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis,

QuadraticDiscriminantAnalysis

from sklearn. semi\_supervised import LabelPropagation, LabelSpreading

**Multiclass as One-Vs-One:**

from sklearn.svm import NuSVC, SVC

from sklearn.gaussian\_process import GaussianProcessClassifier

GaussianProcessClassifier(multi\_class = “one\_vs\_one”)

**Multiclass as One-Vs-The-Rest:**

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import SGDClassifier, Perceptron, PassiveAggressiveClassifier

from sklearn.gaussian\_process import GaussianProcessClassifier

GaussianProcessClassifier (multi\_class = “one\_vs\_rest”)

from sklearn.svm import LinearSVC

LinearSVC (multi\_class=”ovr”)

**LogisticRegression**

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(C=50.0 / train\_samples, penalty="l1", solver="saga", tol=0.1)

clf.fit(X, y)

**SVC**

SVC and NuSVC implement the “one-versus-one” approach for multi-class classification. In total, n\_classes \* (n\_classes - 1) / 2 classifiers are constructed and each one trains data from two classes.

from sklearn import svm

clf = svm.SVC(decision\_function\_shape='ovo')

clf.fit(X, Y)

**Generalized Linear Models**

Generalized Linear Models (GLM) extend linear models in two ways:

1. the predicted values y are linked to a linear combination of the input variables X via an inverse link function h:



1. the squared loss function is replaced by the unit deviance d of a distribution in the exponential family (or more precisely, a reproductive exponential dispersion model (EDM)

A black and white math symbol

Description automatically generated

Where α is the L2 regularization penalty. When sample weights are provided, the average becomes a weighted average.

A table with math equations

Description automatically generated

* If the target values y are counts (non-negative integer valued) or relative frequencies (non-negative), you might use a Poisson distribution with a log-link.
* If the target values y are probabilities, you can use the Bernoulli distribution. The Bernoulli distribution with a logit link can be used for binary classification. The Categorical distribution with a softmax link can be used for multiclass classification.

Examples of use cases:

* Agriculture / weather modeling: number of rain events per year (Poisson), amount of rainfall per event (Gamma), total rainfall per year (Tweedie / Compound Poisson Gamma).
* Risk modeling / insurance policy pricing: number of claim events / policyholder per year (Poisson), cost per event (Gamma), total cost per policyholder per year (Tweedie / Compound Poisson Gamma).
* Credit Default: probability that a loan can’t be paid back (Bernoulli).
* Fraud Detection: probability that a financial transaction like a cash transfer is a fraudulent transaction (Bernoulli).
* Predictive maintenance: number of production interruption events per year (Poisson), duration of interruption (Gamma), total interruption time per year (Tweedie / Compound Poisson Gamma).
* Medical Drug Testing: probability of curing a patient in a set of trials or probability that a patient will experience side effects (Bernoulli).
* News Classification: categories classification of news articles into three categories namely Business News, Politics and Entertainment news (Categorical).

X = [[1, 2],

[2, 4],

[4, 5],

[3, 2],

[3, 1]]

y = [0, 0, 1, 1, 2]

from sklearn.linear\_model import TweedieRegressor

reg = TweedieRegressor(power=1, alpha=0.5, link='log')

reg.fit(X, y)

**Nearest Neighbors Classification**

Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

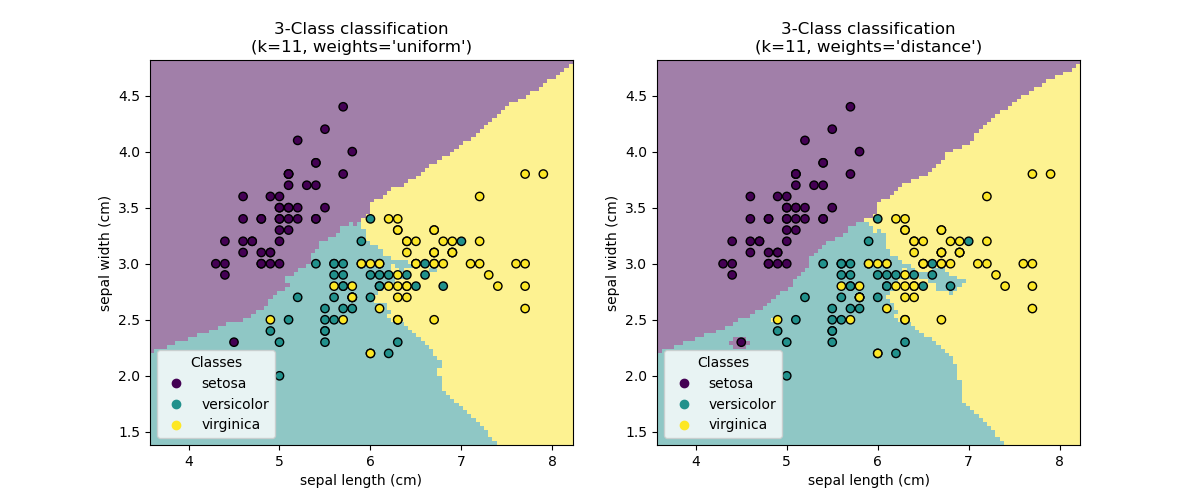
KNeighborsClassifier implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user.

The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

RadiusNeighborsClassifier implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

In cases where the data is not uniformly sampled, radius-based neighbors classification can be a better choice. The user specifies a fixed radius r, such that points in sparser neighborhoods use fewer nearest neighbors for the classification. For high-dimensional parameter spaces, this method becomes less effective due to the so-called “curse of dimensionality”.

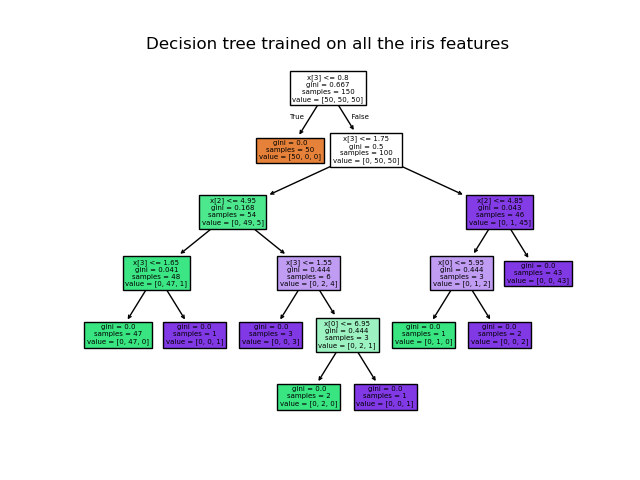
it is better to weight the neighbors such that nearer neighbors contribute more to the fit. The default value, weights = 'uniform', assigns uniform weights to each neighbor. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point



from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=3)

**DecisionTreeClassifier**



from sklearn import tree

clf = tree.DecisionTreeClassifier()

clf = clf.fit(X, y)

tree.plot\_tree(clf)

**GradientBoostingClassifier**

For datasets with categorical features, using the native categorical support is often better than relying on one-hot encoding (OneHotEncoder), because one-hot encoding requires more tree depth to achieve equivalent splits. It is also usually better to rely on the native categorical support rather than to treat categorical features as continuous (ordinal), which happens for ordinal-encoded categorical data, since categories are nominal quantities where order does not matter.

GradientBoostingClassifier supports both binary and multi-class classification.

Tips:

Empirical evidence suggests that small values of learning\_rate favor better test error. [HTF] recommend to set the learning rate to a small constant (e.g. learning\_rate <= 0.1) and choose n\_estimators large enough that early stopping applies

Subsampling with shrinkage can further increase the accuracy of the model

categorical\_features : pass a list of integers indicating the indices of the categorical features

GradientBoostingClassifier(categorical\_features=[0])

from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(n\_estimators=100,

learning\_rate=1.0,

max\_depth=1,

random\_state=0).fit(X\_train, y\_train)

Interpretation with feature importance

Individual decision trees intrinsically perform feature selection by selecting appropriate split points. This information can be used to measure the importance of each feature; the basic idea is: the more often a feature is used in the split points of a tree the more important that feature is. This notion of importance can be extended to decision tree ensembles by simply averaging the impurity-based feature importance of each tree.

clf.feature\_importances\_

**Random forests and other randomized tree ensembles**

RandomForestClassifier supports both binary and multi-class classification.

In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

individual decision trees typically exhibit high variance and tend to overfit. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model.

In contrast to the original publication, the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n\_estimators=10)

Extremely Randomized Trees相比Random forests,更加随机性建树

from sklearn.ensemble import ExtraTreesClassifier

clf = ExtraTreesClassifier(n\_estimators=10, max\_depth=None,

min\_samples\_split=2, random\_state=0)

hyperparameter:

n\_estimators (tree number, the larger the better)

max\_features (radom subset size of feature, The lower the greater the reduction of variance, but also the greater the increase in bias)

max\_features=1.0 (default) equivalent to bagged trees

max\_features=None (all features) and min\_samples\_split=2 for regression problems,

max\_features="sqrt" (random subset of size sqrt(n\_features)) for classification tasks

max\_features=0.3 (smaller values more randomness)

The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features. By averaging the estimates of predictive ability over several randomized trees one can reduce the variance of such an estimate and use it for feature selection.

clf.feature\_importances\_

**Bagging meta-estimator**

boosting methods which usually work best with weak models (e.g., shallow decision trees).

to reduce the variance of a base estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. bagging methods work best with strong and complex models (e.g., fully developed decision trees)

max\_samples and max\_features control the size of the subsets (in terms of samples and features), while bootstrap and bootstrap\_features control whether samples and features are drawn with or without replacement.

from sklearn.ensemble import BaggingClassifier

from sklearn.neighbors import KNeighborsClassifier

bagging = BaggingClassifier(KNeighborsClassifier(),

max\_samples=0.5, max\_features=0.5)

**Voting Classifier**

The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels.

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import VotingClassifier

clf1 = LogisticRegression(random\_state=1)

clf2 = RandomForestClassifier(n\_estimators=50, random\_state=1)

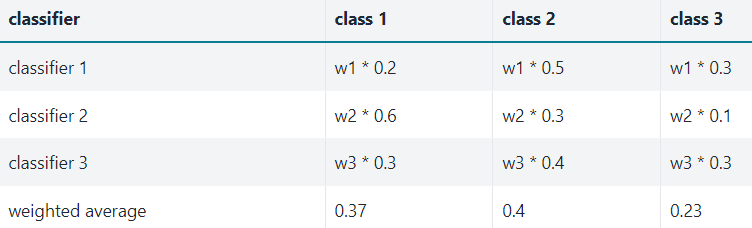
clf3 = GaussianNB()

eclf = VotingClassifier(

estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)],

voting='hard')

Weighted Average Probabilities (Soft Voting)



the predicted class label is 2, since it has the highest average probability.

clf1 = DecisionTreeClassifier(max\_depth=4)

clf2 = KNeighborsClassifier(n\_neighbors=7)

clf3 = SVC(kernel='rbf', probability=True)

eclf = VotingClassifier(estimators=[('dt', clf1), ('knn', clf2), ('svc', clf3)],

voting='soft', weights=[2, 1, 2])

**AdaBoost** (Adaptive Boosting)

Principle: fit a sequence of weak learners (e.g. Decision Trees) on repeatedly re-sampled versions of the data. Each sample carries a weight that is adjusted after each training step, such that misclassified samples will be assigned higher weights. The re-sampling process with replacement takes into account the weights assigned to each sample. Samples with higher weights have a greater chance of being selected multiple times in the new data set, while samples with lower weights are less likely to be selected. This ensures that subsequent iterations of the algorithm focus on the difficult-to-classify samples.

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

weak\_learner = DecisionTreeClassifier(max\_leaf\_nodes=8)

n\_estimators = 300

adaboost\_clf = AdaBoostClassifier(

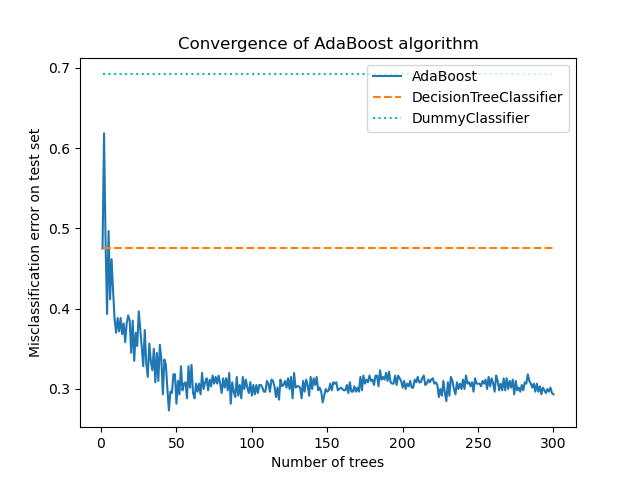
estimator=weak\_learner,

n\_estimators=n\_estimators,

algorithm="SAMME",

random\_state=42,

).fit(X\_train, y\_train)



AdaBoost远比决策树结果好，决策树比随机分类器好

**Gaussian Process Classification (GPC)**

GaussianProcessClassifier supports multi-class classification by performing either one-versus-rest or one-versus-one based training and prediction. In one-versus-rest, one binary Gaussian process classifier is fitted for each class, which is trained to separate this class from the rest. In “one\_vs\_one”, one binary Gaussian process classifier is fitted for each pair of classes, which is trained to separate these two classes. The predictions of these binary predictors are combined into multi-class predictions.

from sklearn.gaussian\_process import GaussianProcessClassifier

from sklearn.gaussian\_process.kernels import RBF

kernel = 1.0 \* RBF(1.0)

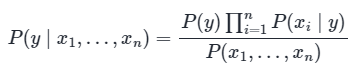
gpc = GaussianProcessClassifier(kernel=kernel, random\_state=0)

**Naive Bayes**

given class variable y and dependent feature vector x1 through xn :



Using the naive conditional independence assumption that, for all i, this relationship is simplified



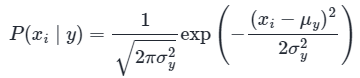
 => A black text on a white background

Description automatically generated

we can use Maximum A Posteriori (MAP) estimation to estimate P(y) and P(xi |y), p(y) is the relative frequency of class y in the training set. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi |y).

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

**GaussianNB**



The parameters and are estimated using maximum likelihood.

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

OneVsRestClassifier (depreciated)

from sklearn.multiclass import OneVsRestClassifier

clf = SVC(random\_state=0)

classif = OneVsRestClassifier(clf)

classif.fit(X, y)

OneVsOneClassifier (depreciated)

constructs one classifier per pair of classes, At prediction time, the class which received the most votes is selected.

from sklearn.multiclass import OneVsOneClassifier

OneVsOneClassifier(LinearSVC(random\_state=0)).fit(X, y).predict(X)

#### Multilabel: (sample, feature dim) -> (sample, class num)

y(t, k) belong {0, 1}, 0 means no belong to class k, 1 means belong to class k

It is equal to n\_classes binary classification tasks.

Features = [[f1, f2, f3, …, fn],

[f1, f2, f3, …, fn],

… ]] with (m, n)

Output = [[y1, …, yk],

…,

[y1, …, yk] with (m, k)

while m is the sample number, k is the class number

y1 belong to {0, 1}, 0 means no label, 1 means label

e.g.,

For example, prediction of the topics relevant to a text document or video. The document or video may be about one of ‘religion’, ‘politics’, ‘finance’ or ‘education’, several of the topic classes or all of the topic classes.

y = np.array([[1, 0, 0, 1], # 4列表示4类，0表示不属于对应类别，1表示属于对应类别

[0, 0, 1, 1],

[0, 0, 0, 0]])

Each column represents a class. The 1’s in each row denote the positive classes a sample has been labeled with.

Sample 1 belong to religion and education

Sample 2 belong to finance and education

Sample 3 belong to none.

from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier

from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import RidgeClassifier, RidgeClassifierCV

**finetune hyper-parameter :**

pipe = Pipeline([(Classifier, MultiOutputClassifier(ensemble.GradientBoostingClassifier()))])

params = [{

'Classifier': [MultiOutputClassifier(ensemble.GradientBoostingClassifier())],

"Classifier\_\_estimator\_\_n\_estimators": [200, 300, 500],

}]

optimize = BayesSearchCV(pipe, params, scoring='neg\_mean\_absolute\_error',

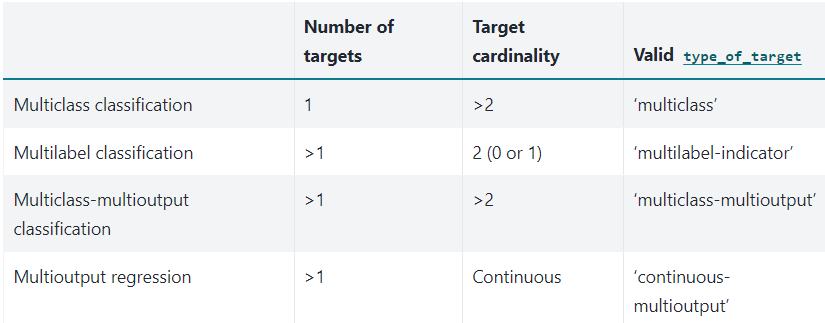
n\_iter=50\*len(params), cv=3, n\_jobs=-1)

optimize.fit(X\_train, y\_train)

model = optimize.best\_estimator\_

#### Multiclass-multioutput classification (multitask classification): (sample, feature dim) -> (sample, task num)

Output has multi-tasks, each tasks has > 2 classes



Features = [[f1, f2, f3, …, fn],

[f1, f2, f3, …, fn],

… ]] with (m, n)

Output = [[y1, …, yt],

…,

[y1, …, yt] with (m, t) while m is the sample number

t taks classification, each task maybe has >2 classes

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier

from sklearn.multioutput import MultiOutputClassifier

y = np.array([['apple', 'green'],

['orange', 'orange'],

['pear', 'green']])

n\_samples, n\_features = X.shape

n\_outputs = Y.shape[1]

n\_classes = 3

forest = RandomForestClassifier(random\_state=1)

multi\_target\_forest = MultiOutputClassifier(forest, n\_jobs=2)

multi\_target\_forest.fit(X, Y).predict(X)

### Unsupervise-learning: Cluster

**Nearest Neighbors**: unsupervised and supervised learning

Principle: find a predefined number of training samples closest in distance to the new point, and predict the label from these.

Two options:

* k-nearest neighbor learning: user-defined samples number
* radius-based neighbor learning: based on the local density of points. Common metric measure: standard Euclidean distance.

Neighbors-based methods are known as non-generalizing machine learning methods, since they simply “remember” all of its training data

# Finding the Nearest Neighbors (unsupervised)

from sklearn.neighbors import NearestNeighbors

import numpy as np

X = np.array([[-1, -1],

[-2, -1],

[-3, -2]]])

nbrs = NearestNeighbors(n\_neighbors=2, algorithm='ball\_tree').fit(X)

distances, indices = nbrs.kneighbors(X)

nbrs.kneighbors\_graph(X).toarray() # connections between neighboring points

The NearestCentroid classifier is a simple algorithm that represents each class by the centroid of its members. In effect, this makes it similar to the KMeans algorithm.

from sklearn.neighbors import NearestCentroid

import numpy as np

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

y = np.array([1, 1, 1, 2, 2, 2])

clf = NearestCentroid()

**K-means clustering**

Don’t over-interpret clustering results. There is absolutely no guarantee of recovering a ground truth.

First, choosing the right number of clusters is hard.

Second, the algorithm is sensitive to initialization, and can fall into local minima

from sklearn import cluster, datasets

iris = datasets.load\_iris()

X\_iris = iris.data

y\_iris = iris.target

k\_means = cluster.KMeans(n\_clusters=3)

k\_means.fit(X\_iris)

print(k\_means.labels\_[::10])

print(y\_iris[::10])

Application example: vector quantization

KMeans, in particular, can be seen as a way of choosing a small number of exemplars to compress the information

Hierarchical clustering

A Hierarchical clustering method is a type of cluster analysis that aims to build a hierarchy of clusters

Agglomerative - bottom-up approaches: each observation starts in its own cluster, and clusters are iteratively merged in such a way to minimize a linkage criterion, When the number of clusters is large, it is much more computationally efficient than k-means.

Divisive - top-down approaches: all observations start in one cluster, which is iteratively split as one moves down the hierarchy. For estimating large numbers of clusters, this approach is both slow and statistically ill-posed.

Connectivity-constrained clustering

to retrieve connected regions (sometimes also referred to as connected components) when clustering an image

### Feature agglomeration

We have seen that sparsity could be used to mitigate the curse of dimensionality,

an insufficient amount of observations compared to the number of features.

Another approach is to merge together similar features: feature agglomeration. This approach can be implemented by clustering in the feature direction, in other words clustering the transposed data.

**Principal component analysis: PCA**

Principal component analysis (PCA) selects the successive components that explain the maximum variance in the signal.

from sklearn import decomposition

pca = decomposition.PCA()

pca.fit(X) # X(100, 3)

print(pca.explained\_variance\_)

[ 2.18565811e+00 1.19346747e+00 8.43026679e-32]

# As we can see, only the 2 first components are useful

pca.n\_components = 2

X\_reduced = pca.fit\_transform(X) # (100, 2)

Application: Face recognition with eigenfaces

<http://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html#pipelining>

**Independent Component Analysis: ICA**

Independent component analysis (ICA) selects components so that the distribution of their loadings carries a maximum amount of independent information. It can recover non-Gaussian independent signals

# Generate sample data

import numpy as np

from scipy import signal

def real\_data():

time = np.linspace(0, 10, 2000)

s1 = np.sin(2 \* time) # Signal 1 : sinusoidal signal

s2 = np.sign(np.sin(3 \* time)) # Signal 2 : square signal

s3 = signal.sawtooth(2 \* np.pi \* time) # Signal 3: saw tooth signal

S = np.c\_[s1, s2, s3]

S += 0.2 \* np.random.normal(size=S.shape) # Add noise

S /= S.std(axis=0) # Standardize data

return S

def observation(S):

A = np.array([[1, 1, 1], [0.5, 2, 1], [1.5, 1, 2]]) # Mixing matrix

X = np.dot(S, A.T) # Generate observations

return X

def recover\_signal():

ica = decomposition.FastICA()

S\_ = ica.fit\_transform(X) # Get the estimated sources

A\_ = ica.mixing\_.T

np.allclose(X, np.dot(S\_, A\_) + ica.mean\_)

return S\_

**Cross decomposition**: supervised dimensionality reduction

Cross decomposition algorithms find the fundamental relations between two matrices (X and Y).

PLS projects both X and Y into a lower-dimensional subspace such that the covariance between transformed(X) and transformed(Y) is maximal.

Principal Component Regression (PCR) would keep the features with the most variance, but it’s possible that features with a small variances are relevant from predicting the target. In a way, PLS allows for the same kind of dimensionality reduction, but by taking into account the targets y

## Tuning the hyper-parameters of an estimator

Note that it is common that a small subset of those parameters can have a large impact on the predictive or computation performance of the model while others can be left to their default values.

Fine tune hyperparameters for estimators on the test set would lead to a risk of overfitting because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance.

Cross validation for hyper-parameter, then the best model is frozed for test dataset.

**Computing cross-validated metrics**

from sklearn.model\_selection import cross\_val\_score

svc = svm.SVC(C=1, kernel='linear')

METHOD1: splitting the data, fitting a model and computing the score 5 consecutive times

scores = cross\_val\_score(clf, X, y, cv=5 , scoring='f1\_macro')

When the cv argument is an integer, cross\_val\_score uses the KFold or StratifiedKFold strategies by default, the latter being used if the estimator derives from ClassifierMixin.

from sklearn.model\_selection import ShuffleSplit

cv = ShuffleSplit(n\_splits=5, test\_size=0.3, random\_state=0)

cross\_val\_score(clf, X, y, cv=cv)

from sklearn.metrics import recall\_score

scoring = ['precision\_macro', 'recall\_macro']

scores = cross\_validate(clf, X, y, scoring=scoring)

METHOD2:

from sklearn.model\_selection import KFold, cross\_val\_score

X = ["a", "a", "b", "c", "c", "c"]

k\_fold = KFold(n\_splits=3)

for train, test in k\_fold.split(X\_digits):

svc.fit(X\_digits[train], y\_digits[train]).score(X\_digits[test], y\_digits[test])

**GridSearchCV (HalvingGridSearchCV)** & **RandomizedSearchCV (HalvingRandomSearchCV)**

By default, the GridSearchCV uses a 3-fold cross-validation. However, if it detects that a classifier is passed, rather than a regressor, it uses a stratified 3-fold.

By default, parameter search uses the score function of the estimator to evaluate a parameter setting. These are the sklearn.metrics.accuracy\_score for classification and sklearn.metrics.r2\_score for regression.

in unbalanced classification, the accuracy score is often uninformative

refit=True will build the best\_estimator\_ on the whole dataset with best\_params\_.

**GridSearchCV (HalvingGridSearchCV)** exhaustively considers all parameter combinations

**RandomizedSearchCV (HalvingRandomSearchCV)** can sample a given number of candidates from a parameter space with a specified distribution.

HalvingXXXCV class is able to find parameter combinations that are just as accurate as GridSearchCV, in much less time (about 4 times).

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

tuned\_parameters = [

{"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},

{"kernel": ["linear"], "C": [1, 10, 100, 1000]},

]

grid\_search = GridSearchCV(

SVC(), tuned\_parameters, scoring=scores, refit=refit\_strategy

)

grid\_search = HalvingGridSearchCV(

SVC(), tuned\_parameters, factor=2, random\_state=rng

)

grid\_search.fit(X\_train, y\_train)

RandomizedSearchCV implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values. This has two main benefits over an exhaustive search: 1) A budget can be chosen independent of the number of parameters and possible values. 2) Adding parameters that do not influence the performance does not decrease efficiency.

The randomized search and the grid search explore exactly the same space of parameters. The result in parameter settings is quite similar, while the run time for randomized search is drastically lower.

from sklearn.linear\_model import SGDClassifier

from sklearn.model\_selection import RandomizedSearchCV

import scipy.stats as stats

param\_dist = {

"average": [True, False],

"l1\_ratio": stats.uniform(0, 1),

"alpha": stats.loguniform(1e-2, 1e0),

}

random\_search = RandomizedSearchCV(clf, param\_distributions=param\_dist, n\_iter=15)

from scipy.stats import expon, gamma, uniform, loguniform, randint

params = {

'C': scipy.stats.expon(scale=100), # 最好用连续分布函数

'gamma': scipy.stats.expon(scale=.1),

'kernel': ['rbf'],

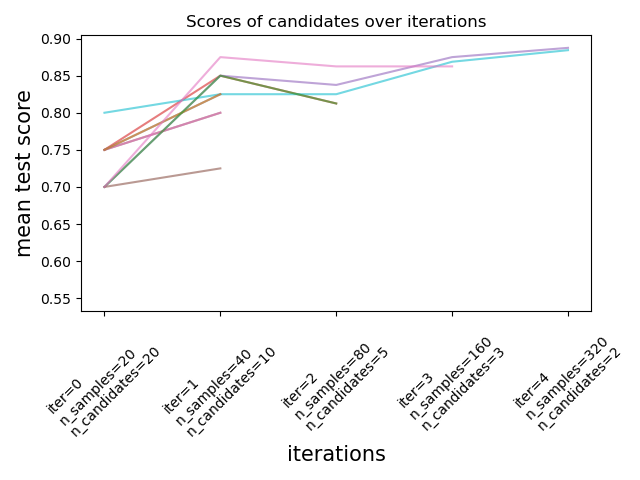
'class\_weight':['balanced', None]

}

For continuous parameters, such as C above, it is important to specify a continuous distribution to take full advantage of the randomization. This way, increasing n\_iter will always lead to a finer search.

**HalvingRandomSearchCV (preferred)**

Successive halving (SH) is like a tournament among candidate parameter combinations. SH is an iterative selection process where all candidates (the parameter combinations) are evaluated with a small amount of resources at the first iteration. Only some of these candidates are selected for the next iteration, which will be allocated more resources. SH 是一个迭代选择过程，其中所有候选（参数组合）在第一次迭代时使用少量资源进行评估。只有其中一些候选会被选中进行下一次迭代，并为其分配更多资源。



As illustrated in the figure below, only a subset of candidates ‘survive’ until the last iteration.

from sklearn.experimental import enable\_halving\_search\_cv

from sklearn.model\_selection import HalvingRandomSearchCV

grid\_search = HalvingRandomSearchCV (

SVC(), tuned\_parameters, factor=2, random\_state=rng

)

The factor (> 1) parameter controls the rate at which the resources grow, and the rate at which the number of candidates decreases. In each iteration, the number of resources per candidate is multiplied by factor and the number of candidates is divided by the same factor.

GridSearchCV and RandomizedSearchCV allow searching over parameters of composite or nested estimators such as Pipeline, ColumnTransformer, VotingClassifier or CalibratedClassifierCV using a dedicated <estimator>\_\_<parameter> syntax:

from sklearn.calibration import CalibratedClassifierCV

from sklearn.ensemble import RandomForestClassifier

calibrated\_forest = CalibratedClassifierCV(RandomForestClassifier(n\_estimators=10))

param\_grid = {

'estimator\_\_max\_depth': [2, 4, 6, 8]

}

search = GridSearchCV(calibrated\_forest, param\_grid, cv=5)

**example: select best model with best hyper-parameters**

<https://learnwithnas.medium.com/tuning-multiple-ml-models-using-gridsearchcv-a-sample-project-of-predicting-steel-optimal-8d1c86f7f4f0>

(X\_train, y\_train), (X\_test, y\_test)

pipe = Pipeline([('Regressor', RandomForestRegressor(random\_state=random\_state))])

params = [

{

'Regressor' : [RandomForestRegressor(random\_state=random\_state)],

'Regressor\_\_criterion': ['friedman\_mse'],

'Regressor\_\_n\_estimators': [50, 100, 500],

},

{

'Regressor' : [XGBRegressor(random\_state=random\_state)],

'Regressor\_\_learning\_rate' : [0.01, 0.1, 0.2, 0.5],

}

]

optimize = BayesSearchCV(pipe, params, scoring='neg\_mean\_absolute\_error',

n\_iter=50\*len(params), cv=3, n\_jobs=-1)

optimize.fit(X\_train, y\_train)

best\_model = optimize.best\_estimator\_

best\_params = optimize.best\_params\_

best\_score = optimize.best\_score\_

cv\_results\_df = pd.DataFrame(optimize.cv\_results\_)

print(cv\_results\_df)

A screenshot of a computer

Description automatically generated

Each row corresponds to a given parameter combination (a candidate) and a given iteration. The iteration is given by the iter column. The n\_resources column tells you how many resources were used.

## Pipeline

A sequence of data transformers with an optional final predictor.

The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.

Intermediate steps of the pipeline must be ‘transforms’, that is, they must implement fit and transform methods.

from sklearn.pipeline import Pipeline

pipe = Pipeline([

('scaler', StandardScaler()),

('svc', SVC())

])

pipe.fit(X\_train, y\_train).score(X\_test, y\_test)

pipe.set\_params(svc\_\_C=10).fit(X\_train, y\_train).score(X\_test, y\_test)

复杂例子：feature generated with hyper-parameter

class FeatureExtractor(BaseEstimator, TransformerMixin):

def \_\_init\_\_(self, mini\_loop\_num):

self.mini\_loop\_num = mini\_loop\_num

def fit(self, patients\_train, y=None):

return self

def transform(self, patients\_train, y=None):

features, patients = prepare\_feature(self.mini\_loop\_num)

sub\_features = filter(features, patients, patients\_train)

return sub\_features

pipe = Pipeline(steps=[

('fe', FeatureExtractor(mini\_loop\_num=10)),

('Regressor', ensemble.RandomForestRegressor(random\_state=random\_state))

])

params = [{

'fe\_\_mini\_loop\_num': Integer(10, 500),

'Regressor': [linear\_model.ElasticNet(random\_state=random\_state)],

'Regressor\_\_alpha': Real(1e-2, 1e+3, prior='log-uniform'),

'Regressor\_\_l1\_ratio': Real(0, 1) },

{

'fe\_\_mini\_loop\_num': Integer(10, 500),

'Regressor': [ensemble.RandomForestRegressor(random\_state=random\_state)],

'Regressor\_\_criterion': ['friedman\_mse'],

'Regressor\_\_n\_estimators': [50, 100, 500] },

]

optimize = BayesSearchCV(pipe, params, cv=3, n\_jobs=-1)

optimize.fit(patients\_train, y\_train)

model = optimize.best\_estimator\_

preds = model.predict(patients\_test)

Note: patients\_train and y\_train will split to cv fold of (train, val)

Each train dataset or val will call FeatureExtractor.transform(patients\_train) to generate features with different mini\_loop\_num for different hyper-parameters

## Model Selection and evaluation

## Application:

## Kaggle 竞赛

From The Official Blog of Kaggle.com

blog.kaggle.com/2016/07/21/approaching-almost-any-machine-learning-problem-abhishek-thakur/

### Working With Text Data

# Loading the 20 newsgroups dataset

from sklearn.datasets import fetch\_20newsgroups

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

twenty\_train = fetch\_20newsgroups(subset='train', categories=categories, shuffle=True, random\_state=42)

print("\n".join(twenty\_train.data[0].split("\n")[:3])) #content

print(twenty\_train.target\_names[twenty\_train.target[0]]) #category

METHOD1: IN DETAIL

# Extracting features from text files

# Tokenizing text, filtering of stopwords => 词频矩阵

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

词频矩阵X\_train\_counts, shape=(n\_samples, n\_words),

注意n\_words是基于所有文档

元素a[i][j]表示j词在i文档下的词频

X\_train\_counts在python是用scipy.sparse.csr\_matrix来存储稀疏矩阵

#Term Frequency times Inverse Document Frequency

from sklearn.feature\_extraction.text import TfidfTransformer

tf\_transformer = TfidfTransformer(use\_idf=False).fit(X\_train\_counts)

X\_train\_tf = tf\_transformer.transform(X\_train\_counts)

TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

# Training a classifier

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target)

# predict

docs\_new = ['God is love', 'OpenGL on the GPU is fast']

X\_new\_counts = count\_vect.transform(docs\_new)

X\_new\_tfidf = tfidf\_transformer.transform(X\_new\_counts)

predicted = clf.predict(X\_new\_tfidf)

for doc, category in zip(docs\_new, predicted):

print('%r => %s' % (doc, twenty\_train.target\_names[category]))

METHOD2: PIPELINE vectorizer => transformer => classifier

from sklearn.pipeline import Pipeline

text\_clf = Pipeline([('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB()),

])

text\_clf.fit(twenty\_train.data, twenty\_train.target)

# Evaluation of the performance on the test set

import numpy as np

twenty\_test = fetch\_20newsgroups(subset='test',

categories=categories, shuffle=True, random\_state=42)

predicted = text\_clf.predict(twenty\_test.data)

np.mean(predicted == twenty\_test.target)

from sklearn import metrics

metrics.confusion\_matrix(twenty\_test.target, predicted)

## practice

clean data, benchmark

**随机生成两个类别的点**

从一个具有特定协方差矩阵和特定均值的随机分布中抽取坐标来生成每一类别。协方差矩阵描述了点云的形状，均值则描述了点云在平面上的位置。

A yellow and purple dots

Description automatically generated

import numpy as np

import matplotlib.pyplot as plt

def make\_binary(num\_samples\_per\_class=1000):

samples\_neg = np.random.multivariate\_normal(mean=[0, 3], cov=[[1, 0.5], [0.5, 1]],

size=num\_samples\_per\_class)

samples\_pos = np.random.multivariate\_normal(mean=[3, 0], cov=[[1, 0.5], [0.5, 1]],

size=num\_samples\_per\_class)

inputs = np.vstack((samples\_neg, samples\_pos)).astype(np.float32)

targets = np.vstack((np.zeros((num\_samples\_per\_class, 1), dtype="float32"),

np.ones((num\_samples\_per\_class, 1), dtype="float32")))

return inputs, targets

inputs, targets = make\_binary()

plt.scatter(inputs[:, 0], inputs[:, 1], c=targets[:, 0])

plt.show()

**随机基准模型**

def random\_classifier(test\_labels):

test\_labels\_copy = copy.copy(test\_labels)

np.random.shuffle(test\_labels\_copy)

hits = np.array(test\_labels) == np.array(test\_labels\_copy)

accuracy = hits.mean()

return accuracy

inputs, targets = make\_binary()

accuracy = random\_classifier(y)

如果是均衡10分类，随机分类达到10% accuracy.

如果是平均衡10分类，随机分类精度需要用以上函数估计

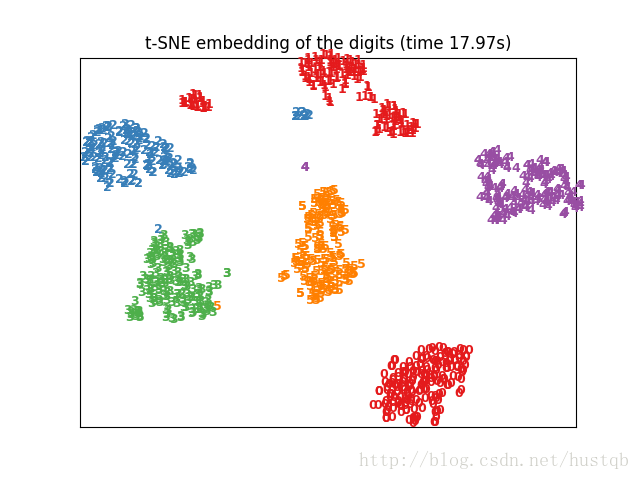
### 数据降维与可视化

思路：对高维数据进行分类，想大致判断数据集有没有很好的可分性（即同类之间间隔小，异类之间间隔大），可以通过t-SNE投影到2维或者3维的空间中观察一下。如果在低维空间中具有可分性，则数据是可分的；如果在低维空间中无法区分，可能低维不足以准确地表示数据的内部结构。

* t-SNE：非线性降维，主要关注数据的局部结构，更适合作为高维数据的可视化。缺点：计算量大
* PCA：线性变换，速度快。手写体降维，PCA结果会导致重叠
* Isomap, LLE and variants适合展开单个连续低维的manifold.

t-SNE原理：将数据点之间的相似度转换为概率，原始空间中的相似度由高斯联合概念表示，嵌入空间的相似度由t分布表示，通过梯度下降最小化原始空间和嵌入空间的联合概率的KL散度。

例子：手写体mnist数据降维可视化



from sklearn.manifold import TSNE

from sklearn import datasets

def get\_data():

digits = datasets.load\_digits(n\_class=6)

data = digits.data

label = digits.target

n\_samples, n\_features = data.shape

return data, label, n\_samples, n\_features

data, label, n\_samples, n\_features = get\_data()

# 高维 -> 2D

tsne = TSNE(n\_components=2, init='pca', random\_state=0)

transformed\_data = tsne.fit\_transform(data)

# in table format

tsne\_df = pd.DataFrame(

np.column\_stack((transformed\_data, label)),

columns=["x", "y", "targets"])

tsne\_df.loc[:, "targets"] = tsne\_df.targets.astype(int)



# visualize

grid = sns.FacetGrid(tsne\_df, hue="targets", size=8)

grid.map(plt.scatter, "x", "y").add\_legend()

### imbalance

kaggle竞赛指导

<https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>

df\_train = pd.read\_csv('../input/train.csv')

# 画两类图

target\_count = df\_train.target.value\_counts()

target\_count.plot(kind='bar', title='Count (target)');

# 画confusion matrix

from sklearn.metrics import confusion\_matrix

from matplotlib import pyplot as plt

conf\_mat = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print('Confusion matrix:\n', conf\_mat)

labels = ['Class 0', 'Class 1']

fig = plt.figure()

ax = fig.add\_subplot(111)

cax = ax.matshow(conf\_mat, cmap=plt.cm.Blues)

fig.colorbar(cax)

ax.set\_xticklabels([''] + labels)

ax.set\_yticklabels([''] + labels)

plt.xlabel('Predicted')

plt.ylabel('Expected')

plt.show()

**weakness of simplest resampling**

* The simplest implementation of over-sampling is to duplicate random records from the minority class, which can cause overfitting.
* In under-sampling, the simplest technique involves removing random records from the majority class, which can cause loss of information

**more sophisticated resapling techniques**

* we can cluster the records of the majority class and do the under-sampling by removing records from each cluster, thus seeking to preserve information.
* In over-sampling, instead of creating exact copies of the minority class records, we can introduce small variations into those copies, creating more diverse synthetic samples.

Imbalanced-learn (scikit-learn-contrib)

**Under-sampling: Tomek links**

from imblearn.under\_sampling import TomekLinks

tl = TomekLinks(return\_indices=True, ratio='majority')

X\_tl, y\_tl, id\_tl = tl.fit\_sample(X, y)

**Under-sampling: Cluster Centroids**

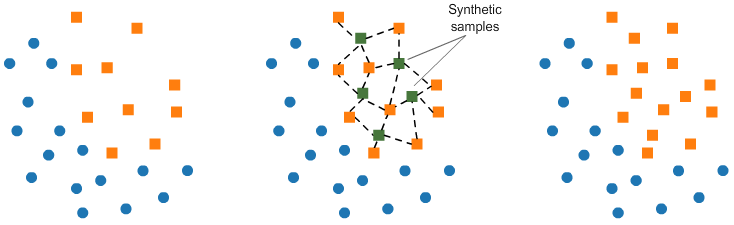
from imblearn.under\_sampling import ClusterCentroids

cc = ClusterCentroids(ratio={0: 10})

X\_cc, y\_cc = cc.fit\_sample(X, y)

**Over-sampling: SMOTE**

SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class, based on those that already exist. It works randomly picingk a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.



from imblearn.over\_sampling import SMOTE

smote = SMOTE(ratio='minority')

X\_sm, y\_sm = smote.fit\_sample(X, y)

**Oversampling + undersampling**

from imblearn.combine import SMOTETomek

smt = SMOTETomek(ratio='auto')

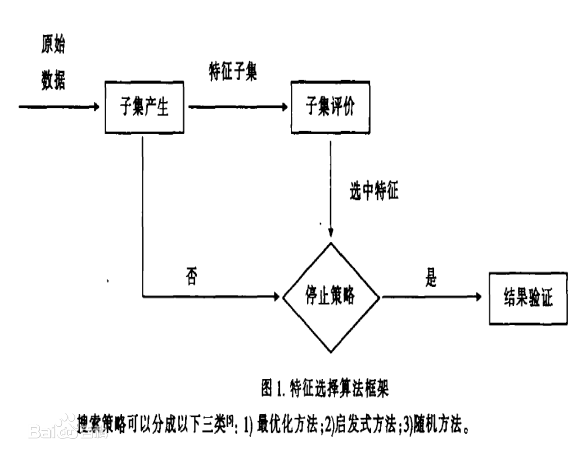
X\_smt, y\_smt = smt.fit\_sample(X, y)

Ensemble learning

from imblearn.ensemble import EasyEnsemble

from imblearn.ensemble import BalanceCascade

### Feature selection



评价函数scoring function很重要，如相关性p-value, 互信息，分类器指标等

1. Filter Method

其实就是预处理，利用训练集自身的特点筛选出特征子集后再送入分类器进行学习，与分类器的选择无关

Removing features with low variance

removes all features whose variance doesn’t meet some threshold

from sklearn.feature\_selection import VarianceThreshold

{X, y}

**sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))**

**sel.fit\_transform(X)**

Univariate feature selection

selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator.

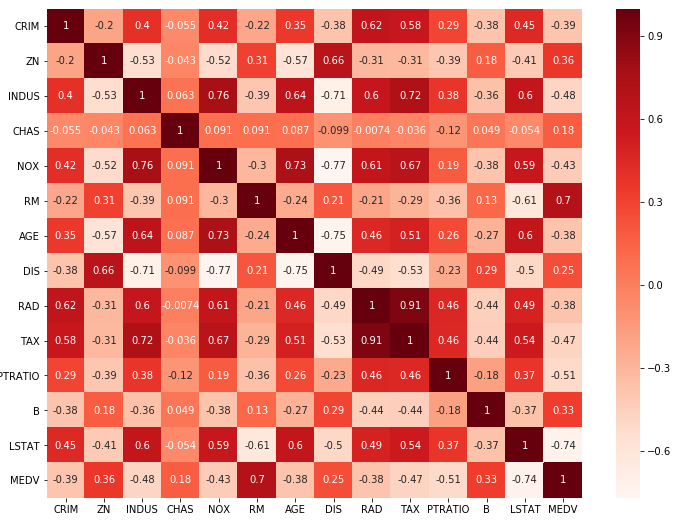
# Using Pearson Correlation

plt.figure(figsize=(12,10))

cor = df.corr()

sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)

plt.show()



SelectKBest removes all but the highest scoring features

SelectPercentile removes all but a user-specified highest scoring percentage of features

using common univariate statistical tests for each feature: false positive rate SelectFpr, false discovery rate SelectFdr, or family wise error SelectFwe.

GenericUnivariateSelect allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

from sklearn.feature\_selection import SelectKBest, chi2, SelectPercentile, SelectFpr, SelectFdr, SelectFwe, GenericUnivariateSelect

**X\_new = SelectKBest(chi2, k=2).fit\_transform(X, y)**

注意scoring function需要满足：

a scoring function that returns univariate scores and p-values

For regression: f\_regression, mutual\_info\_regression

For classification: chi2, f\_classif, mutual\_info\_classif

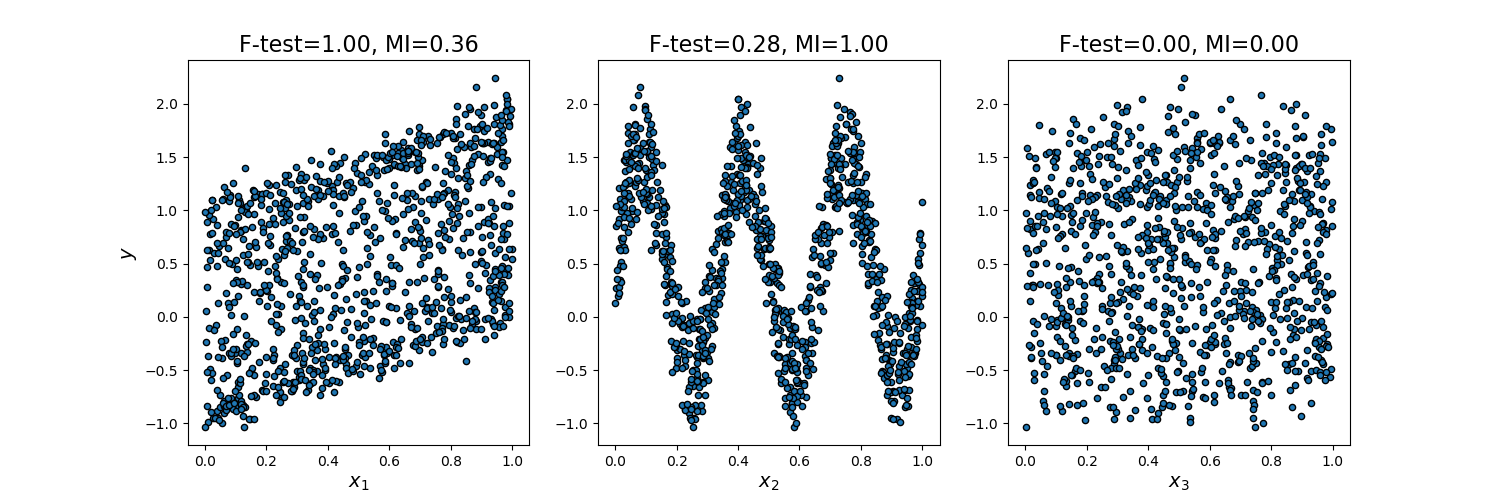
Example:

X = np.random.rand(1000, 3)

y = X[:, 0] + np.sin(6 \* np.pi \* X[:, 1]) + 0.1 \* np.random.randn(1000)

f\_test, \_ = f\_regression(X, y)

mi = mutual\_info\_regression(X, y)



y = x1 + sin(6 \* pi \* x2) + 0.1 \* N(0, 1)

F-test captures only linear dependency, it rates x1 as the most discriminative feature.

mutual information can capture any kind of dependency between variables, it rates x2 as the most discriminative feature

both methods correctly marks x3 as irrelevant.

**Feature selection using SelectFromModel**

remove unimportant features by threshold which can be found by built-in heuristics like "mean", "median" or "0.1\*mean"

from sklearn.feature\_selection import SelectFromModel

lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)

model = SelectFromModel(lsvc, prefit=True)

X\_new = model.transform(X)

1. Wrapper Method

封装器用选取的特征子集对训练集进行分类，分类的精度作为衡量特征子集好坏的标准。

**RFE (Recursive feature elimination)**

RFE:

First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a coef\_ attribute or through a feature\_importances\_attribute.

Then, the least important features are pruned from current set of features.

That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

RFECV:

是基于RFE取得score结果（从1个特征到所有特征），进行求和，然后求最大，得出最佳特征数是k (注意：此刻仍不知道是哪k个)

然后对全体训练集求k个最佳特征

from sklearn.feature\_selection import RFE, RFECV

# Initialization

n\_features\_to\_select = 1

n\_features = X\_train.shape[1]

support\_ = np.ones(n\_features, dtype=np.bool)

ranking\_ = np.ones(n\_features, dtype=np.int)

scores = []

# Elimination

while np.sum(support\_) > n\_features\_to\_select:

# Remaining features

features = np.arange(n\_features)[support\_]

# Rank the remaining features

score, feature\_importants = ImbalanceClassifier(X\_train[:, features], y\_train, X\_test[:, features], y\_test, build\_model).calc\_model\_performance()

coefs = feature\_importants # Get coefs

ranks = np.argsort(safe\_sqr(coefs)) # Get ranks

# Eliminate the worse features

threshold = min(1, np.sum(support\_) - n\_features\_to\_select)

support\_[features[ranks][:threshold]] = False

ranking\_[np.logical\_not(support\_)] += 1

设置n\_features\_to\_select = 1, 从而获取所有特征到一个最佳特征的特征mask(support\_)

**Backward Elimination, Forward Selection, Bidirectional Elimination**

Forward Selection: 特征子集X从空集开始，每次选择能使得评价函数J(X)最优的一个特征x加入，其实就是贪心算法，缺点是只加不减

Backward Elimination: 和FS相反，从特征全集开始，每次选择使评价函数J(X)最差的特征x剔除，也是贪心，缺点是只减不增

1. Embedded Method