Table of Contents

前言

机器学习(`Machine Learning`)是一门多领域交叉学科，涉及概率论、统计学、最优化、算法复杂度理论等多个学科方向。专门研究如何借助计算机模拟实现人类的学习行为，以获取新的知识或技能，或者更好地组织已有的知识结构。机器学习是人工智能的核心，是使计算机具有智能的根本途径之一，其应用遍及人工智能的各个领域。

机器学习正在迅速改变我们的世界，我们几乎每天都会看到机器学习如何改变日常的生活。如果你在淘宝或者京东这样的电商平台购买商品，或者只是进行一次百度搜索，就已经触碰到了机器学习的应用。使用这些服务的用户会产生数据，这些数据会被手机，在进行预处理之后用来训练模型，而模型会通过这些数据来提供更好的用户体验。此外，还有很多使用机器学习技术的产品或服务即将在我们的生活当中普及。可以说如果想要深入机器学习的应用开发当中，现在是一个非常理想的时机。

说明

本书主要介绍一些机器学习的入门知识，例如什么是机器学习，常见的机器学习算法原理，常用的模型性能评估指标，怎样快速入门 `sklearn` 等内容。

若想更加全面，系统的学习机器学习相关知识，可以在本书的最后扫码体验整套机器学习实训课程。该课程是与南京大学合作共建的实训课程，总共有 `65` 个实践任务，涵盖了《机器学习》中的前十章内容，并已在南京大学投入使用。可以通过扫码查看整套课程。

机器学习概述

近年来，全球新一代信息技术创新浪潮迭起。作为全球信息领域产业竞争的新一轮焦点，人工智能的发展也迎来了第三次浪潮，它正在推动工业发展进入新的阶段，掀起第四次工业革命的序幕。而作为人工智能的重要组成部分，机器习也成了炙手可热的概念。本章将向您介绍机器学习的基础知识，为后面的学习打好基础。

本章主要涉及的知识点有：

- 什么是机器学习
- 机器学习的主要任务
- 机器学习中常见的术语
- 机器学习项目的流程

什么是机器学习

机器学习的定义有很多种，但是最准确的定义是：`"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."`

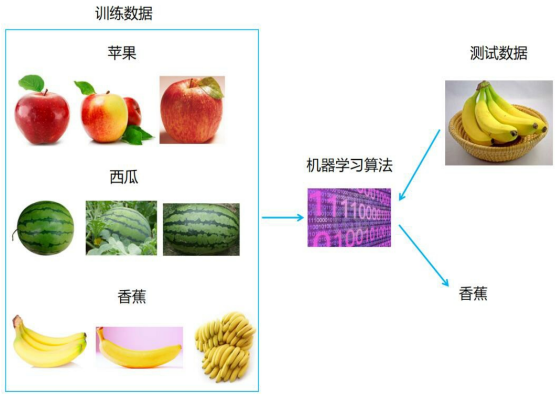
这个定义除了非常押韵之外，还体现了机器学习的几个关键点，即：`task`，`experience` 和 `performance`。

task

task 指的是机器学习所需要完成的任务。机器学习能够完成的任务主要有：分类、回归、聚类。

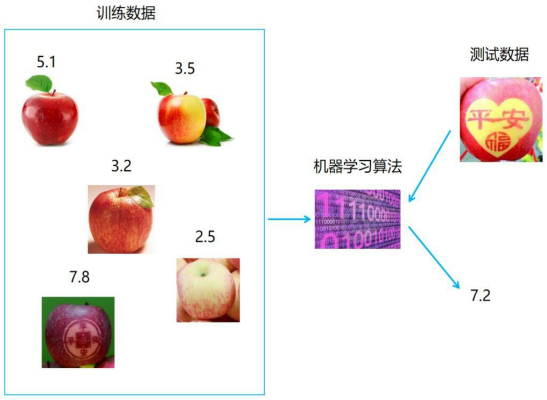
分类

假如现在有一些苹果、西瓜和香蕉的图片作为训练集(有标签)，现在想要机器学习算法能够根据新的测试图片来分辨出该图片中的是苹果、西瓜还是香蕉。像这样的任务我们称为分类任务。



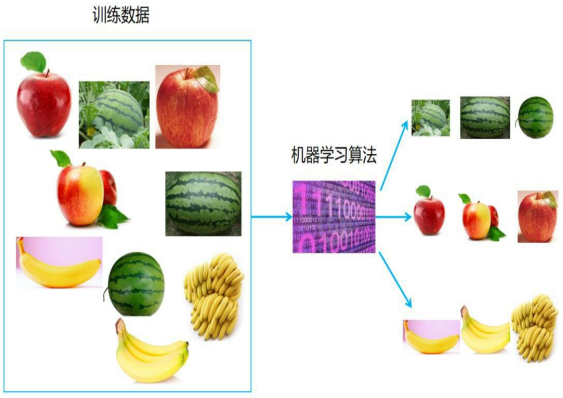
回归

假如现在有一些苹果的售价数据作为训练集(有标签)，现在想要机器学习算法能够根据新的测试图片来分辨出该图片中的苹果能卖多少钱。像这样的任务我们称为回归任务。



聚类

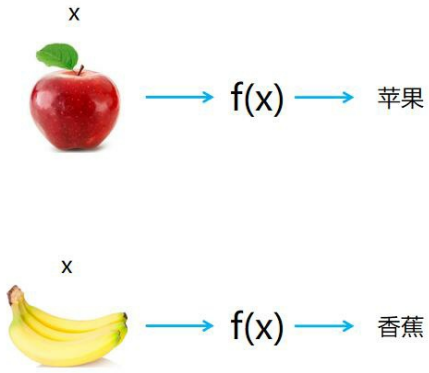
假如现在有一些水果的图片作为训练集(无标签)，现在想要机器学习算法能够根据训练集中的图片将这些图片进行归类，但是并不知道这些类别是什么。像这样的任务我们称为聚类任务。



细心的您可能注意到了，分类和回归问题的训练集中都是带有标签的。也就是说数据已经告诉了机器学习算法我这条数据的答案是这个，那条数据的答案是那个，就像有老师在监督学生做题目一样，一看到学生做错了就告诉他题目做错了，看到学生做对了就鼓励他。所以用来解决分类和回归问题的机器学习算法又称为监督学习。而像用来解决聚类问题的机器学习算法又称为无监督学习。

experience

experience指的根据历史数据总结归纳出规律的过程，即学习过程，或模型的训练过程。模型这个词看上去很高大上，其实我们可以把他看成是一个函数。例如：现在想用机器学习来识别图片里的是香蕉还是苹果，那么机器学习所的事情就是得到一个比较好的函数，当我们输入一张香蕉图片时，能得到识别结果为香蕉的输出，当我们输入一张苹果图片时，能得到识别结果为苹果的输出。



至于这样一个函数(模型)里面长什么样子，这就与具体的机器学习算法有关了。对机器学学习算法感兴趣可以阅读常见机器学习算法章节。

performance

performance指的是模型的性能。对于不同的任务，我们有不同的衡量模型性能的标准。例如分类时可能会根据模型的准确率，精准率，召回率，**AUC**等指标来衡量模型的好坏，回归时会看看模型的**MSE**，**RMSE**，**r2 score**等指标，回归时会以**FM**指数，**DB**指数等指标来衡量聚类的效果怎么样。对各种性能指标感兴趣可以阅读模型评估指标章节。

机器学习常用术语

训练集，测试集，样本，特征

假设我们收集了一份西瓜数据：

色泽	纹理	声音	甜不甜
青绿	清晰	清脆	不甜
青绿	模糊	浑浊	甜
乌黑	清晰	清脆	不甜
乌黑	模糊	浑浊	甜

并假设现在已经使用机器学习算法根据这份数据的特点训练出了一个很厉害的模型，成为了一个挑瓜好手，只需告诉它这个西瓜的色泽，纹理和声音就能告诉你这个西瓜甜不甜。

我们通常将这种喂给机器学习算法来训练模型的数据称为训练集，用来让机器学习算法预测的数据称为测试集。

训练集中的所有行称为样本。由于我们的挑瓜好手需要的西瓜信息是色泽、纹理和声音，所以此训练集中每个样本的前 `3` 列称为特征。挑瓜好手给出的结果是甜或不甜，所以最后 `1` 列称为标签。

因此，这份数据是一个有 `4` 个样本， `3` 个特征的训练集，训练集的标签是“甜不甜”。

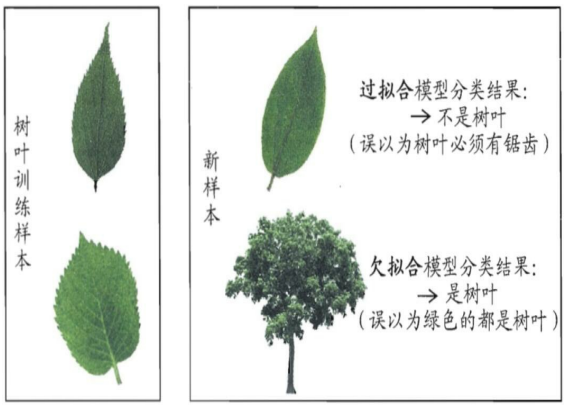
欠拟合与过拟合

最好的情况下，我们的模型应该不管在训练集上还是测试集上，它的性能都不错。但是有的时候，我们的模型在训练集上的性能比较差，那么这种情况我们称为欠拟合。那如果我们的模型在训练集上的性能好到爆炸，但在测试集上的性能却不尽人意，那么这种情况我们称为过拟合。

其实欠拟合与过拟合的区别和我们生活中学生考试的例子很像。如果一个学生在平时的练习中题目的正确率都不高，那么说明这个学生可能基础不牢或者心思没花在学习上，所以这位学生可能欠缺基础知识或者智商可能不太高或者其他种种原因，像这种情况可以看成是欠拟合。那如果这位学生平时练习的正确率非常高，但是他不怎么灵光，喜欢死记硬背，只会做已经做过的题，一碰到没见过的新题就不知所措了。像这种情况可以看成时是过拟合。

那么是什么原因导致了欠拟合和过拟合呢？

当我们的模型过于简单，很可能会导致欠拟合。如果模型过于复杂，就很可能会导致过拟合。



验证集与交叉验证

在真实业务中，我们可能没有真正意义上的测试集，或者说不知道测试集中的数据长什么样子。那么我们怎样在没有测试集的情况下来验证我们的模型好还是不好呢？这个时候就需要验证集了。

那么验证集从何而来，很明显，我们可以从训练集中抽取一小部分的数据作为验证集，用来验证我们模型的性能。

但如果仅仅是从训练集中抽取一小部分作为验证集的话，有可能会让我们对模型的性能有一种偏见或者误解。

比如我们现在要对手写数字进行识别，那么我就可能会训练一个分类模型。但可能模型对于数字 1 的识别准确率比较低，而验证集中没多少个数字为 1 的样本，然后用验证集测试完后得到的准确率为 0.96 。然后您可能觉得哎呀，我的模型很厉害了，但其实并不然，因为这样的验证集让您的模型的性能有了误解。那有没有更加公正的验证算法性能的方法呢？有，那就是**k-**折交叉验证！

在**K-**折交叉验证中，我们把原始训练数据集分割成 K 个不重合的数据集，然后我们做 K 次模型训练和验证。每次，我们使 个数据集验证模型，并使其它 K-1 个数据集来训练模型。在这 K 次训练和验证中，每次来验证模型的数据集都不同。最后，我们对这 K 次在验证集上的性能求平均。

K 的值由我们自己来指定，如以下为 5 折交叉验证。

机器学习项目流程

当我们拿到一个项目需求想要使用机器学习来实现时，我们往往会按照理解问题与问题转换、获取数据、数据清洗与预处理、特征工程、模型训练与调优这`5`个步骤来走。

1.理解实际问题，抽象为机器学习能处理的数学问题

理解实际业务场景问题是机器学习的第一步。机器学习的特征工程和模型训练通常都是一件非常耗时的过程，胡乱尝试时间成本是非常高的。深入理解要处理的问题，能避免走很多弯路。 理解问题，包括明确可以获得什么样的数据，机器学习的目标是一个分类、回归还是聚类。如果都不是的话，要考虑将它们转变为机器学习问题。

2.获取数据

获取数据包括获取原始数据以及从原始数据中经过特征工程从中提取训练、测试数据。机器学习比赛中原始数据都是直接提供的，但是实际问题需要自己获得原始数据。

有句名言是：**“**数据决定机器学习结果的上限，而算法只是尽可能的逼近这个上限**”**，可见数据在机器学习中的作用。总的来说数据要有具有**“**代表性**”**，否则必然会过拟合。对于分类问题，数据偏斜不能过于严重，不同类别的数据数量不要有数个数量级的差距。

3.数据清洗与预处理

数据预处理、数据清洗是很关键的步骤，往往能够使得算法的效果和性能得到显著提高。归一化、离散化、因子化、缺失值处理、去除共线性等，数据挖掘过程中很多时间就花在它们上面。这些工作简单可复制，收益稳定可预期，是机器学习的基础必备步骤。

4.特征工程

特征工程包括从原始数据中特征构建、特征提取、特征选择，非常有讲究。深入理解实际业务场景下的问题，丰富的机器学习经验能帮助我们更好的处理特征工程。特征工程做的好能发挥原始数据的最大效力，往往能够使得算法的效果和性能得到显著的提升，有时能使简单的模型的效果比复杂的模型效果好。

筛选出显著特征、摒弃非显著特征，需要机器学习工程师反复理解业务。这对很多结果有决定性的影响。特征选择需要运用特征有效性分析的相关技术，如相关系数、卡方检验、平均互信息、条件熵、后验概率、逻辑回归权重等方法。

5.模型训练、诊断与调优

现在有很多的机器学习算法的工具包，例如`sklearn`，使用非常方便，真正考验水平的根据对算法的理解调节（超）参数，使模型达到最优。

过拟合、欠拟合的模型状态判断是模型诊断中至关重要的一步。常见的方法如：交叉验证，绘制学习曲线等。过拟合的基本调优思路是增加训练的数据量，降低模型复杂度。欠拟合的基本调优思路是提高特征数量和质量，增加模型复杂度。

诊断后的模型需要进行进一步调优，调优后的新模型需要重新诊断，这是一个反复迭代不断逼近的过程，需要不断的尝试，进而达到最优的状态。

本章主要介绍 10 种常见的机器学习算法(模型)的原理，理解模型的原理对于以后使用一些机器学习库实现业务功能时是有好处的。

本章主要涉及的知识点有：

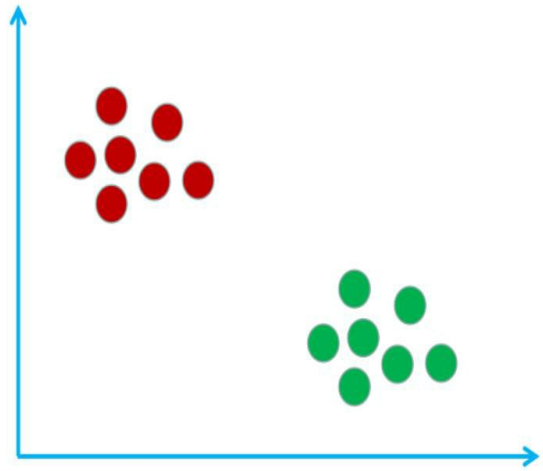
- kNN
- 线性回归
- 逻辑回归
- 多分类学习
- 决策树
- 随机森林
- 朴素贝叶斯分类器
- 支持向量机
- k-Means
- AGNES

近朱者赤近墨者黑-kNN

kNN算法其实是众多机器学习算法中最简单的一种，因为该算法的思想完全可以用 8 个字来概括：“近朱者赤，近墨者黑”。

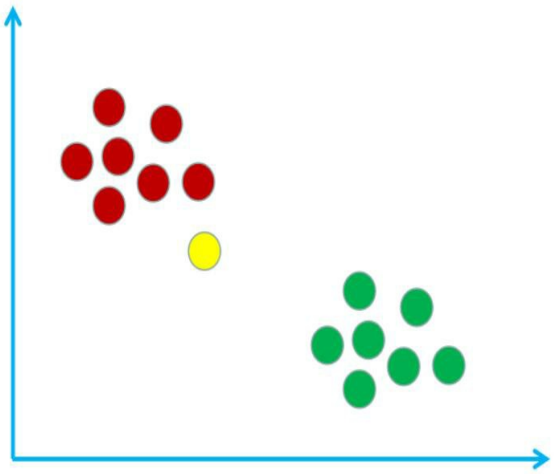
kNN算法解决分类问题

假设现在有这样的一个样本空间(由样本组成的一个空间)，该样本空间里有宅男和文艺青年这两个类别，其中红圈表示宅男，绿圈表示文艺青年。如下图所示：



其实构建出这样的样本空间的过程就是**kNN**算法的训练过程。可想而知**kNN**算法是没有训练过程的，所以**kNN**算法属于懒惰学习算法。

假设我在这个样本空间中用黄圈表示，如下图所示：



现在使用**kNN**算法来鉴别一下我是宅男还是文艺青年。首先需要计算我与样本空间中所有样本的距离。假设计算得到的距离表格如下：

样本编号	**1**	**2**	**...**	**13**	**14**
标签	宅男	宅男	...	文艺青年	文艺青年
距离	11.2	9.5	...	23.3	37.6

然后找出与我距离最小的 k 个样本(k 是一个超参数，需要自己设置，一般默认为 5)，假设与我离得最近的 5 个样本的标签和距离如下：

样本编号	**4**	**5**	**6**	**7**	**8**
标签	宅男	宅男	宅男	宅男	文艺青年
距离	11.2	9.5	7.7	5.8	15.2

最后只需要对这 5 个样本的标签进行统计，并将票数最多的标签作为预测结果即可。如上表中，宅男是 4 票，文艺青年是 1 票，所以我是宅男。

注意：有的时候可能会有票数一致的情况，比如 k = 4 时与我离得最近的样本如下：

样本编号	**4**	**9**	**11**	**13**
标签	宅男	宅男	文艺青年	文艺青年
距离	4.2	9.5	7.7	5.8

可以看出宅男和文艺青年的比分是 2 : 2 ，那么可以尝试将属于宅男的 2 个样本与我的总距离和属于文艺青年的 2 个样本与我的总距离进行比较。然后选择总距离最小的标签作为预测结果。在这个例子中预测结果为文艺青年(宅男的总距离为 4.2 + 9.5 ，文艺青年的总距离为 7.7 + 5.8)。

kNN算法解决回归问题

很明显，刚刚我们使用**kNN**算法解决了一个分类问题，那**kNN**算法能解决回归问题吗？当然可以！

在使用 `kNN` 算法解决回归问题时的思路和解决分类问题的思路基本一致，只不过预测标签值是多少的的时候是将距离最近的 k 个样本的标签值加起来再算个平均，而不是投票。例如离待预测样本最近的 5 个样本的标签如下：

样本编号	**4**	**9**	**11**	**13**	**15**
标签	1.2	1.5	0.8	1.33	1.19

所以待预测样本的标签为： `(1.2+1.5+0.8+1.33+1.19)/5=1.204`

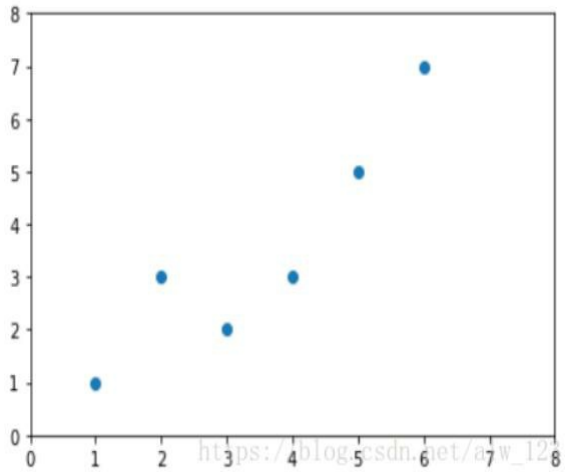
最简单的回归算法-线性回归

什么是线性回归

线性回归是什么意思？我们可以拆字释义。回归肯定不用我多说了，那什么是线性呢？我们可以回忆一下初中时学过的直线方程：$y = k * x + b$

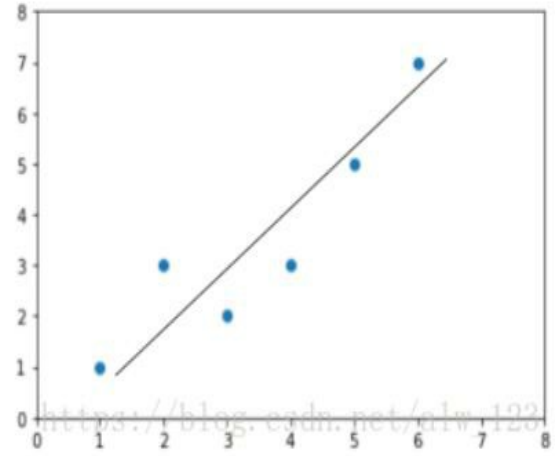
这个式子表达的是，当我知道 k（参数）和 b（参数）的情况下，我随便给一个 x 我都能通过这个方程算出 y 来。而且呢，这个式子是线性的，为什么呢？因为从直觉上来说，你都知道，这个式子的函数图像是条直线。

从理论上来说，这式子满足线性系统的性质(至于线性系统是什么，可以查阅相关资料，这里就不多做赘述了，不然没完没了)。您可能会觉得疑惑，这一节要说的是线性回归，我说个这么 low 直线方程干啥？其实，说白了，线性回归就是在 N 维空间中找一个形式像直线方程一样的函数来拟合数据而已。比如说，我现在有这么一张图，横坐标代表房子的面积，纵坐标代表房价。

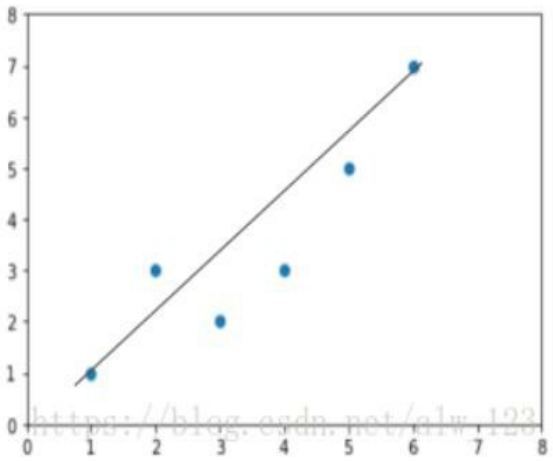


然后呢，线性回归就是要找一条直线，并且让这条直线尽可能地拟合图中的数据点。

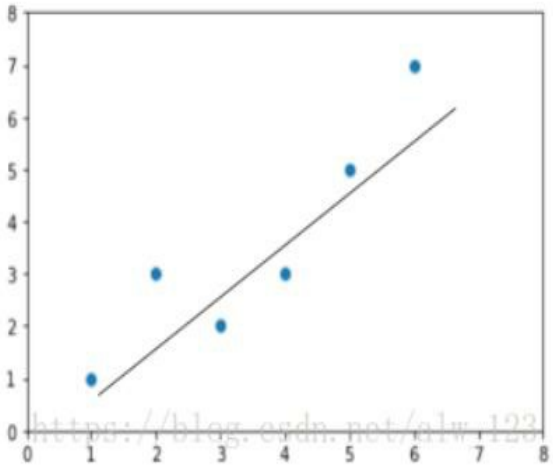
那如果让 1000 位朋友来找这条直线就可能找出 1000 种直线来，比如这样



这样



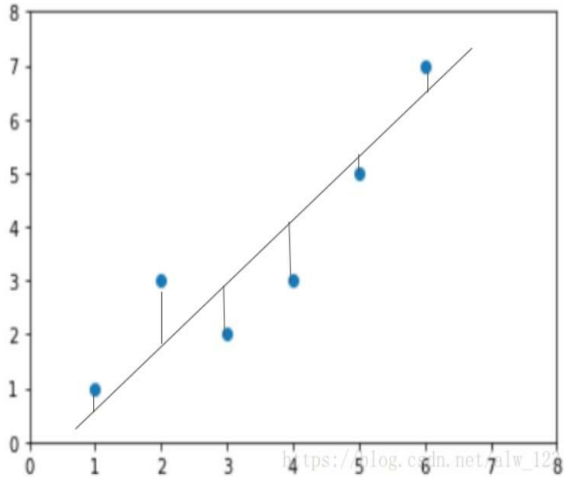
或者这样



喏，其实找直线的过程就是在做线性回归，只不过这个叫法更有高大上而已。

损失函数

那既然是找直线，那肯定是要有一个评判的标准，来评判哪条直线才是最好的。OK，道理我们都懂，那咋评判呢？其实只要算一下实际房价和我找出的直线根据房子大小预测出来的房价之间的差距就行了。说白了就是算两点的距离。当我们把所有实际房价和预测出来的房价的差距（距离）算出来然后做个加和，我们就能量化出现在我们预测的房价和实际房价之间的误差。例如下图中我画了很多条小数线，每一条小数线就是实际房价和预测房价的差距（距离）。



然后把每条小竖线的长度加起来就等于我们现在通过这条直线预测出的房价与实际房价之间的差距。那每条小竖线的长度的加和怎么算？其实就是欧式距离加和，公式为：$\sum_{i=1}^{m}(y^{(i)}-y^{(\hat{i})})^2$(其中$y(i)$表示的是实际房价，$y\hat{(}i)$表示的是预测房价)。

这个欧氏距离加和其实就是用来量化预测结果和真实结果的误差的一个函数。在机器学习中称它为损失函数（说白了就是计算误差的函数）。那有了这个函数，我们就相当于有了一个评判标准，当这个函数的值越小，就越说明我们找到的这条直线越能拟合我们的房价数据。所以说啊，线性回归就是通过这个损失函数做为评判标准来找出一条直线。

如果假设$h_{(\theta)}(x)$表示当权重为θ，输入为x时计算出来的$\hat{y(i)}$，那么线性回归的损失函数$J(\theta)$就是：

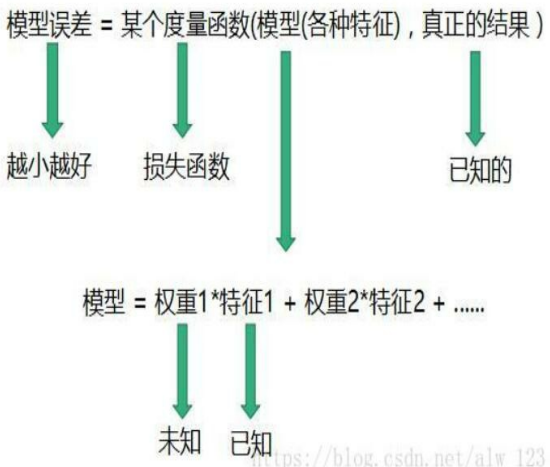
$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$

怎样计算出线性回归的解？

现在您应该已经弄明白了一个事实，那就是我只要找到一组参数（也就是线性方程每一项上的系数）能让我的损失函数的值最小，那我这一组参数就能最好的拟合我现在的训练数据。OK，那怎么来找到这一组参数呢？其实有两种套路，一种就是用大名鼎鼎的梯度下降，其大概思想就是根据每个参数对损失函数的偏导来更新参数。另一种是线性回归的正规方程解，这名字听起来高大上，其实本质就是根据一个固定的式子计算出参数。由于正规方程解在数据量比较大的时候时间复杂度比较高，所以在这一部分中，主要聊聊怎样使用梯度下降的方法来更新参数。

什么是梯度下降

其实梯度下降不是一个机器学习算法，而是一种基于搜索的最优化方法。因为很多算法都没有正规解的，所以需要通过一次一次的迭代来找到找到一组参数能让我们的损失函数最小。损失函数的大概套路可以参看这个图：



所以说，梯度下降的作用是不断的寻找靠谱的权重是多少。

现在我们已经知道了梯度下降就是用来找权重的，那怎么找权重呢？瞎猜？不可能的。。这辈子都不可能猜的。想想都知道，权重的取值范围可以看成是个实数空间，那 100 个特征就对应着 100 个权重，10000 个特征就对应着 10000 个权重。如果靠瞎猜权重的话。应该这辈子都猜不中了。所以找权重的找个套路来找，这个套路就是梯度。梯度其实就是让函数值为 0 时其中各个变量的偏导所组成的向量，而且梯度方向是使得函数值增长最快的方向。

这个性质怎么理解呢？举个栗子。假如我是个想要成为英雄联盟郊区王者的死肥宅，然后要成为郊区王者可能有这么几个因素，一个是英雄池的深浅，一个是大局观，还有一个是骚操作。他们对我成为王者来说都有一定的权重。如图所示，每一个因素的箭头都有方向（也就是因素对于我成为王者的偏导的方向）和长度（偏导的值的大小）。然后在这些因素的共同作用下，我最终会朝着一个方向来训练（好比物理中分力和合力的关系），这个时候我就能以最快的速度向郊区王者更进一步。

也就是说我如果一直朝着最终的那个方向努力的话，理论上来说我就能以最快的速度成为郊区王者。

OK。现在我们知道了梯度的方向是函数增长最快的方向，那我在梯度前面取个负号（反方向），那不就是函数下降最快的方向了么。所以，梯度下降它的本质就是更新权重的时候是沿着梯度的反方向更新。好比下面这个图，假如我是个瞎子，然后莫名其妙的来到了一个山谷里。现在我要做的事情就是走到山谷的谷底。因为我是瞎子，所以我只能一点一点的挪。要挪的话，那我肯定是那我的脚在我四周扫一遍，觉得哪里感觉起来更像是在下山那我就往哪里走。然后这样循环反复一发我最终就能走到山谷的谷底。

所以，梯度下降的伪代码如下：

$$repeat\ unitl\ convergence$$

$$for\ every\ \theta$$

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta_j)}{\partial \theta_j}$$

循环干的事情就相当于我下山的时候在迈步子，代码里的 α 高端点叫学习率，实际上就是代表我下山的时候步子迈多大。值越小就代表我步子迈得小，害怕一脚下去掉坑里。值越大就代表我胆子越大，步子迈得越大，但是有可能会越过山谷的谷底。

使用梯度下降求解线性回归的解

我们知道线性回归的损失函数 J 为：$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$，其中$\theta$为线性回归的解。使用梯度下降来求解，最关键的一步是算梯度(也就是算偏导)，通过计算可知第j个权重的偏导为：

$\partial J(\theta)$

$\frac{\partial J(\theta_j)}{\theta J} = (h_\theta(x) - y)x_j$。

所以很自然的可以想到，使用梯度下降求解线性回归的解的流程如下：

```
循环若干次
    计算当前参数theta对损失函数的梯度 gradient
    theta = theta - alpha * gradient
```

当θ更新好了之后，就相当于得到了一个线性回归模型。也就是说只要将数据放到模型中进行计算就能得到预测输出了。

使用回归的思想进行分类-逻辑回归

逻辑回归是属于机器学习里面的监督学习，它是以回归的思想来解决分类问题的一种非常经典的二分类分类器。由于其训练后的参数有较强的可解释性，在诸多领域中，逻辑回归通常用作 baseline模型，以方便后期更好的挖掘业务相关信息或提升模型性能。

逻辑回归大体思想

什么是逻辑回归

当一看到“回归”这两个字，可能会认为逻辑回归是一种解决回归问题的算法，然而逻辑回归是通过回归的思想来解决二分类问题的算法。

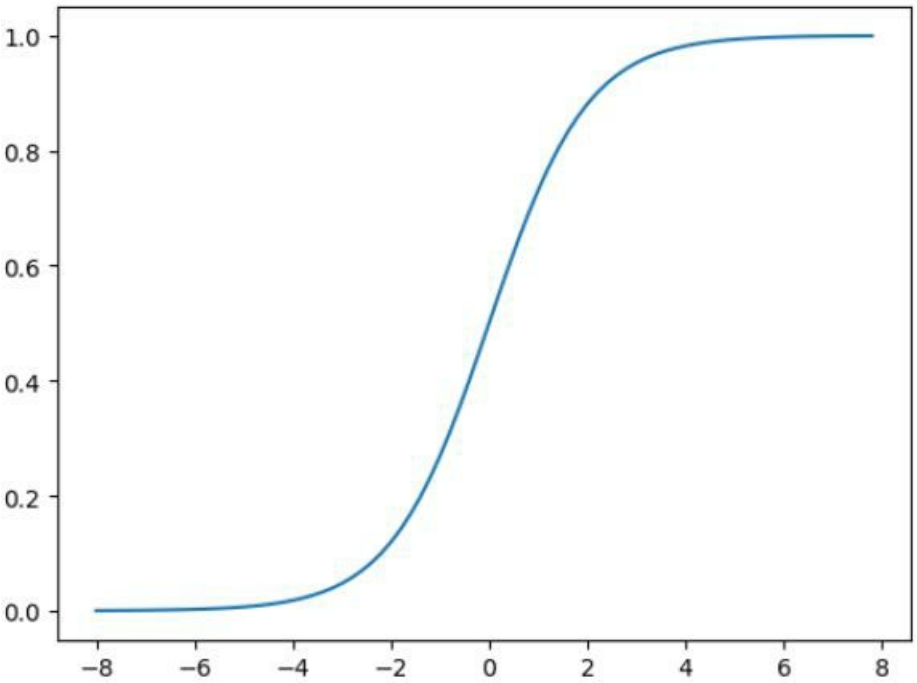
那么问题来了，回归的算法怎样解决分类问题呢？其实很简单，逻辑回归是将样本特征和样本所属类别的概率联系在一起，假设现在已经训练好了一个逻辑回归的模型为$f(x)$，模型的输出是样本x的标签是1的概率，则该模型可以表示成$\hat{p} = f(x)$。若得到了样本x属于标签1的概率后，很自然的就能想到当$\hat{p} > 0.5$时x属于标签1，否则属于标签 0 。所以就有$\hat{y} = \begin{cases} 0 & \hat{p} < 0.5 \\ 1 & \hat{p} > 0.5 \end{cases}$(其中$\hat{y}$为样本 x 根据模型预测出的标签结果，标签 0 和标签 1 所代表的含义是根据业务决定的，比如在癌细胞识别中可以使 0 代表良性肿瘤，1 代表恶性肿瘤)。

由于概率是 0 到 1 的实数，所以逻辑回归若只需要计算出样本所属标签的概率就是一种回归算法，若需要计算出样本所属标签，则就是一种二分类算法。

那么逻辑回归中样本所属标签的概率怎样计算呢？其实和线性回归有关系，学习了线性回归的话肯定知道线性回归就是训练出一组参数 W 和 b 来拟合样本数据，线性回归的输出为$\hat{y} = Wx + b$ 。不过$\hat{y}$的值域是$(-\infty, +\infty)$，如果能够将值域为$(-\infty, +\infty)$ 的实数转换成 $(0, 1)$ 的概率值的话问题就解决了。要解决这个问题很自然地就能想到将线性回归的输出作为输入，输入到另一个函数中，这个函数能够进行转换工作，假设函数为 σ ，转换后的概率为 $\hat{p}$ ，则逻辑回归在预测时可以看成 $\hat{p} = \sigma(Wx + b)$。 σ 其实就是接下来要介绍的$sigmoid$函数。

sigmoid函数

$sigmoid$ 函数的公式为：$\sigma(t) = 1/1 + e^{-t}$。函数图像如下图所示：



从$sigmoid$函数的图像可以看出当t趋近于$-\infty$时函数值趋近于0，当t趋近于$+\infty$时函数值趋近于1。可见$sigmoid$函数的值域是$(0, 1)$，满足我们要将$(-\infty, +\infty)$的实数转换成$(0, 1)$的概率值的需求。因此逻辑回归在预测时可以看成$\hat{p} = 1/(1 + e^{-Wx+b})$，如果$\hat{p} > 0.5$时预测为一种类别，否则预测为另一种类别。

逻辑回归的损失函数

在预测样本属于哪个类别时取决于算出来的$\hat{p}$。从另外一个角度来说，假设现在有一个样本的真实类别为 1 ，模型预测样本为类别 1 的概率为 0.9 的话，就意味着这个模型认为当前样本的类别有 90% 的可能性为 1 ，有 10% 的可能性为 0 。所以从这个角度来看，逻辑回归的损失函数与 $\hat{p}$ 有关。

当然逻辑回归的损失函数不仅仅与 $\hat{p}$ 有关，它还与真实类别有关。假设现在有两种情况，情况 **A** ：现在有个样本的真实类别是 0 ，但是模型预测出来该样本是类别 1 的概率是 0.7（也就是说类别 0 的概率为 0.3 ）；情况 **B** ：现在有个样本的真实类别是 0 ，但是模型预测出来该样本是类别 1 的概率是 0.6（也就是说类别 0 的概率为 0.4 ）；请您思考 2 秒钟，**AB** 两种情况哪种情况的误差更大？很显然，情况 **A** 的误差更大！因为情况 **A** 中模型认为样本是类别 0 的可能性只有 30%，而 情况 **B** 有 40%。

假设现在又有两种情况，情况**A**： 现在有个样本的真实类别是 0 ，但是模型预测出来该样本是类别 1 的概率是 0.7（也就是说类别 0 的概率为 0.3）；情况**B**：现在有个样本的真实类别是 1，但是模型预测出来该样本是类别 1 的概率是 0.3（也就是说类别 0 的概率为 0.7 ）；请您再思考 2 秒钟，**AB**两种情况哪种情况的误差更大？很显然，一样大！

所以逻辑回归的损失函数如下，其中 $cost$ 表示损失函数的值，y 表示样本的真实类别：

$$cost = -ylog(\hat{p}) - (1-y)log(1-\hat{p})$$

知道了逻辑回归的损失函数之后，逻辑回归的训练流程就很明显了，就是寻找一组合适的 W 和 b ，使得损失值最小。找到这组参数后模型就确定下来了。怎么找？很明显，用梯度下降，而且不难算出梯度为：$(\hat{y}-y)x$。

所以逻辑回归梯度下降的代码如下：

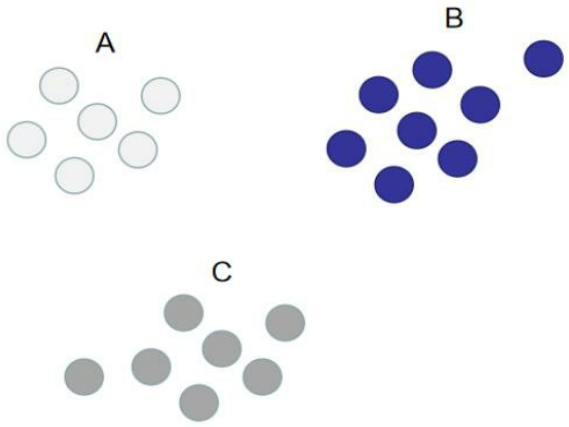
```
#loss
def J(theta, X_b, y):
    y_hat = self._sigmoid(X_b.dot(theta))
    try:
        return -np.sum(y*np.log(y_hat)+(1-y)*np.log(1-y_hat)) / len(y)
    except:
        return float('inf')

# 算theta对loss的偏导
def dJ(theta, X_b, y):
    return X_b.T.dot(self._sigmoid(X_b.dot(theta)) - y) / len(y)

# 批量梯度下降
def gradient_descent(X_b, y, initial_theta, leraning_rate, n_iters=1e4, epsilon=1e-8):
    theta = initial_theta
    cur_iter = 0
    while cur_iter < n_iters:
        gradient = dJ(theta, X_b, y)
        last_theta = theta
        theta = theta - leraning_rate * gradient
        if (abs(J(theta, X_b, y) - J(last_theta, X_b, y)) < epsilon):
            break
        cur_iter += 1
    return theta
```

二分类算法解决多分类问题

现实中常遇到多分类学习任务。有些二分类算法可以直接推广到多分类，但在更多情形下，我们是基于一些策略，利用二分类算法来解决多分类问题。例如：OvO、OvR。

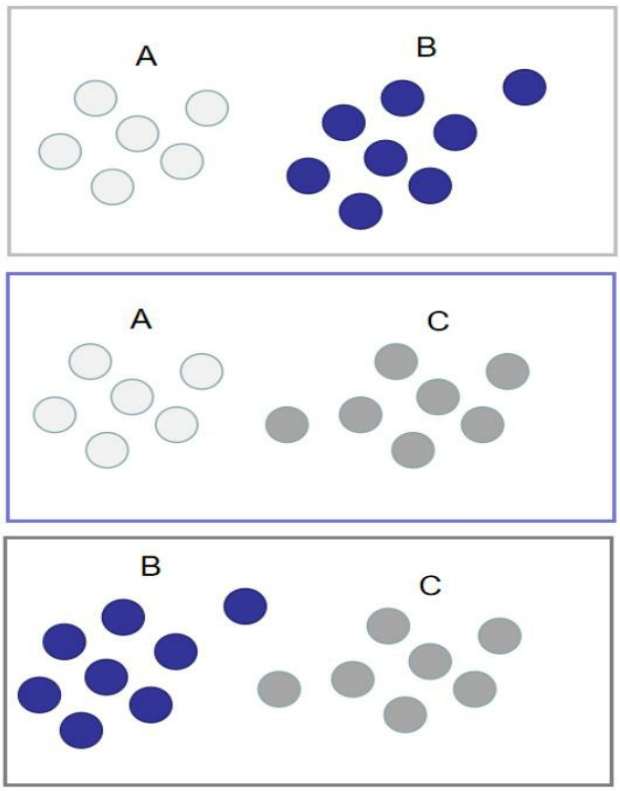
OvO

假设现在训练数据集的分布如下图所示（其中 `A`， `B`， `C` 代表训练数据的类别）：

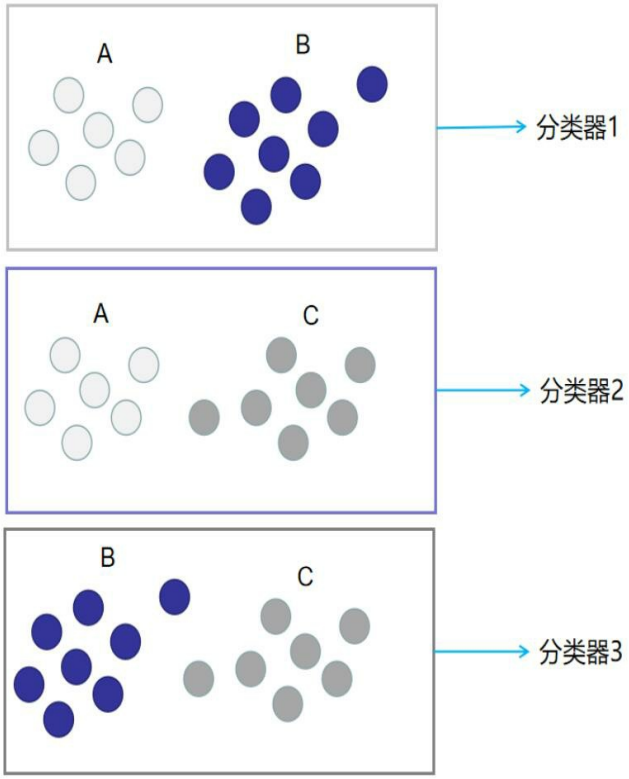


如果想要使用逻辑回归算法来解决这种 `3` 分类问题，可以使用 `OvO` 。 `OvO ( One Vs One )`是使用二分类算法来解决多分类问题的一种策略。从字面意思可以看出它的核心思想就是一对一。所谓的“一”，指的是类别。而“对”指的是从训练集中划分不同的两个类别的组合来训练出多个分类器。

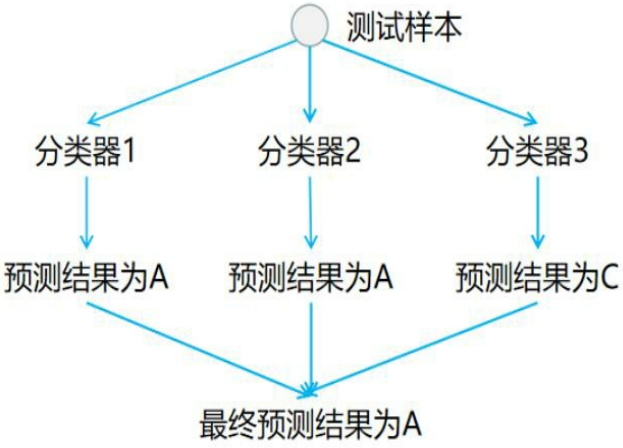
划分的规则很简单，就是组合(C_n^2，其中 `n` 表示训练集中类别的数量，在这个例子中为 `3`)。如下图所示(其中每一个矩形框代表一种划分)：



分别用这 `3` 种划分，划分出来的训练集训练二分类分类器，就能得到 `3` 个分类器。此时训练阶段已经完毕。如下图所示：



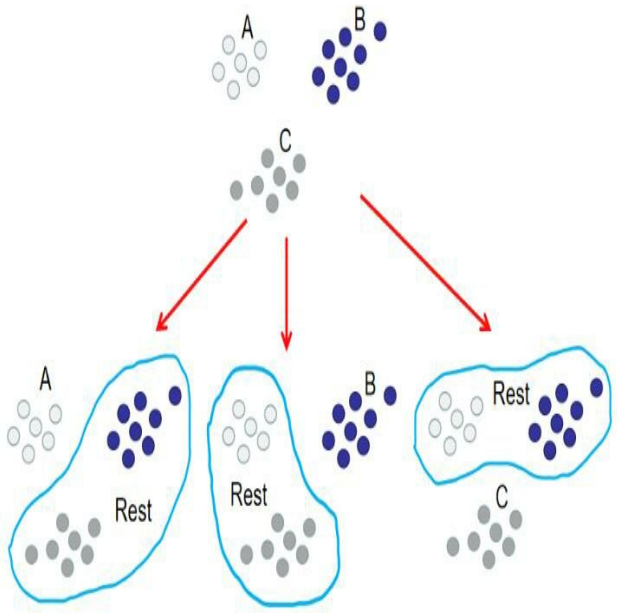
在预测阶段，只需要将测试样本分别扔给训练阶段训练好的 3 个分类器进行预测，最后将 3 个分类器预测出的结果进行投票统计，票数最高的结果为预测结果。如下图所示：



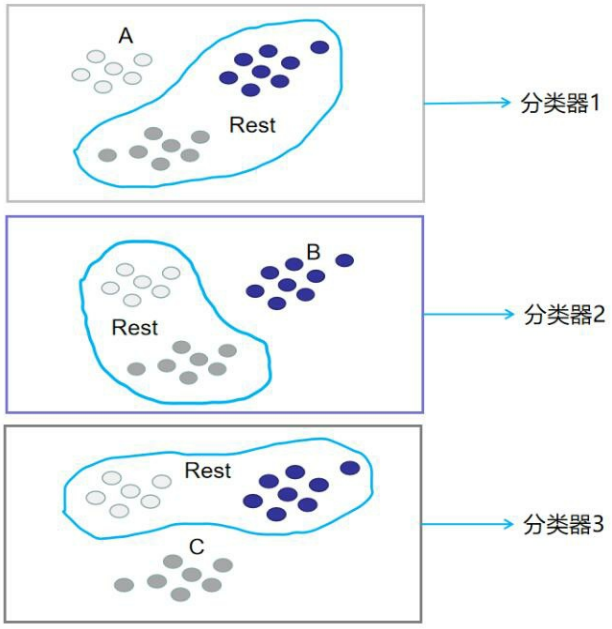
OvR

如果想要使用逻辑回归算法来解决这种 3 分类问题，可以使用 OvR 。 OvR (One Vs Rest)是使用二分类算法来解决多分类问题的一种策略。从字面意思可以看出它的核心思想就是一对剩余。一对剩余的意思是当要对 n 种类别的样本进行分类时，分别取一种样本作为一类，将剩余的所有类型的样本看做另一类，这样就形成了 n 个二分类问题。所以和 OvO 一样，在训练阶段需要进行划分。

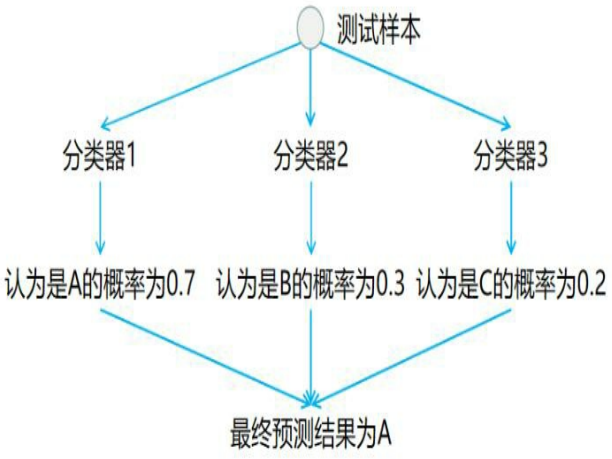
划分也很简单，如下图所示：



分别用这 3 种划分，划分出来的训练集训练二分类分类器，就能得到 3 个分类器。此时训练阶段已经完毕。如下图所示：



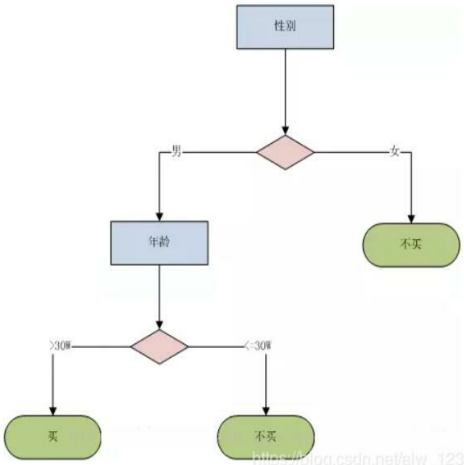
在预测阶段，只需要将测试样本分别扔给训练阶段训练好的 3 个分类器进行预测，最后选概率最高的类别作为最终结果。如下图所示：



最接近人类思维的分类算法-决策树

什么是决策树

决策树说白了就是一棵能够替我们做决策的树，或者说是我们人的脑回路的一种表现形式。比如我看到一个人，然后我会思考这个男人有没有买车。那我的脑回路可能是这样的：



其实这样一种脑回路的形式就是我们所说的决策树。所以从图中能看出决策树是一个类似于人们决策过程的树结构，从根节点开始，每个分枝代表一个新的决策事件，会生成两个或多个分枝，每个叶子代表一个最终判定所属的类别。很明显，如果我现在已经构造好了一颗决策树的话，现在我得到一条数据(男， 29)，我就会认为这个人没有买过车。所以呢，关键问题就是怎样来构造决策树了。

构造决策树时会遵循一个指标，有的是按照信息增益来构建，这种叫**ID3**算法，有的是信息增益比来构建，这种叫**C4.5**算法，有的是按照基尼系数来构建的，这种叫**CART**算法。在这里主要介绍一下**ID3**算法。

ID3算法

整个**ID3**算法其实主要就是围绕着信息增益来的，所以要弄清楚**ID3**算法的流程，首先要弄清楚什么是信息增益，但要弄清楚信息增益之前有个概念必须要懂，就是熵。所以先看看什么是熵。

熵、条件熵、信息增益

在信息论和概率统计中呢，为了表示某个随机变量的不确定性，就借用了热力学的一个概念叫熵。如果假设 X 是一个有限个取值的离散型随机变量的话，很显然它的概率分布或者分布律就是这样的：$P(X=x_i)=p_i, i=1,2,...,n$。

有了概率分布后，则这个随机变量 X 的熵的计算公式就是（PS：这里的 log 是以 2 为底）：$H(X)=-\sum_{i=1}^{n}p_i log p_i$

从这个公式也可以看出，如果我概率是 0 或者是 1 的时候，我的熵就是 0 。（因为这种情况下我随机变量的不确定性是最低的），那如果我的概率是 0.5 也就是五五开的时候，我的熵是最大也就是 1 。（就像扔硬币，你永远都猜不透你下次扔到的是正面还是反面，所以它的不确定性非常高）。所以呢，熵越大，不确定性就越高。

在我们实际情况下，我们要研究的随机变量基本上都是多随机变量的情况，所以假设有随便量(X,Y)，那么它的联合概率分布是这样的：

$P(X=x_i,Y=y_j)=p_{ij}, i=1,2,...,n; j=1,2,...,m$

那如果我想知道在我事件 X 发生的前提下，事件 Y 发生的熵是多少的话，这种熵我们叫它条件熵。条件熵 $H(Y|X)$ 表示随机变量 X 的条件下随机变量 Y 的不确定性。条件熵的计算公式是这样的：$H(Y|X)=\sum_{i=1}^{n}p_i H(Y|X=x_i)$。

当然条件熵的一个性质也熵的性质一样，我概率越确定，条件熵就越小，概率越五五开，条件熵就越大。

OK，现在已经知道了什么是熵，什么是条件熵。接下来就可以看看什么是信息增益了。所谓的信息增益就是表示我已知条件 X 后能得到信息 Y 的不确定性的减少程度。就好比，我在玩读心术。您心里想一件东西，我来猜。我已开始什么都没问你，我要猜的话，肯定是瞎猜。这个时候我的熵就非常高对不对。然后我接下来我会去试着问你是非题，当我问了是非题之后，我就能减小猜测你心中想到的东西的范围，这样其实就是减小了我的熵。那么我熵的减小程度就是我的信息增益。

所以信息增益如果套上机器学习的话就是，如果把特征 A 对训练集 D 的信息增益记为 $g(D,A)$ 的话，那么 $g(D,A)$ 的计算公式就是：$g(D,A)=H(D)-H(D|A)$。

如果看到这一堆公式可能会懵逼，那不如举个栗子来看看信息增益怎么算。假设我现在有这一个数据表，第一列是性别，第二列是活跃度， 第三列是客户是否流失的 $label$。

gender	act_info	is_lost
男	高	0
女	中	0
男	低	1
女	高	0
男	高	0
男	中	0
男	中	1
女	中	0
女	低	1
女	中	0
女	高	0
男	低	1
女	低	1
男	高	0
男	高	0

那如果我要算性别和活跃度这两个特征的信息增益的话，首先要先算总的熵和条件熵。($5/15$ 的意思是总共有 15 条样本里面 $label$ 为 1 的样本有 5 条，$3/8$ 的意思是性别为男的样本有 8 条，然后这 8 条里有 3 条是 $label$ 为 1，其他的数值以此类推)

总熵= (-5/15)*log(5/15)-(10/15)*log(10/15)=0.9182

性别为男的熵= -(3/8)*log(3/8)-(5/8)*log(5/8)=0.9543

性别为女的熵= -(2/7)*log(2/7)-(5/7)*log(5/7)=0.8631

活跃度为低的熵= -(4/4)*log(4/4)-0=0

活跃度为中的熵= -(1/5)*log(1/5)-(4/5)*log(4/5)=0.7219

活跃度为高的熵= -0-(6/6)*log(6/6)=0

现在有了总的熵和条件熵之后我们就能算出性别和活跃度这两个特征的信息增益了。

性别的信息增益=总的熵-(8/15)*性别为男的熵-(7/15)*性别为女的熵=0.0064

活跃度的信息增益=总的熵-(6/15)*活跃度为高的熵-(5/15)*活跃度为中的熵-(4/15)*活跃度为低的熵=0.6776

那信息增益算出来之后有什么意义呢？回到读心术的问题，为了我能更加准确的猜出你心中所想，我肯定是问的问题越好就能猜得越准！换句话来说我肯定是要想出一个信息增益最大的问题来问你，对不对？其实**ID3**算法也是这么想的。**ID3**算法的思想是从训练集 D 中计算每个特征的信息增益，然后看哪个最大就选哪个作为当前节点。然后继续重复刚刚的步骤来构建决策树。

决策树构流程

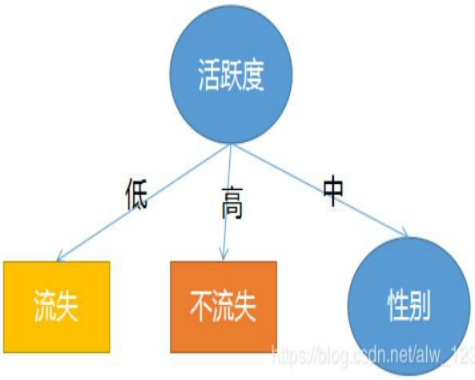
ID3算法其实就是依据特征的信息增益来构建树的。具体套路就是从根节点开始，对节点计算所有可能的特征的信息增益，然后选择信息增益最大的特征作为节点的特征，由该特征的不同取值建立子节点，然后对子节点递归执行上面的套路直到信息增益很小或者没有特征可以继续选择为止。

这样看上去可能会懵，不如用刚刚的数据来构建一颗决策树。

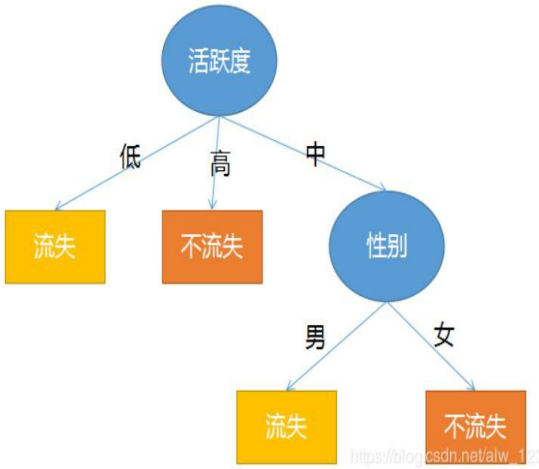
一开始我们已经算过信息增益最大的是活跃度，所以决策树的根节点是活跃度 。所以这个时候树是这样的：

然后发现训练集中的数据表示当我活跃度低的时候一定会流失，活跃度高的时候一定不流失，所以可以先在根节点上接上两个叶子节点。

但是活跃度为中的时候就不一定流失了，所以这个时候就可以把活跃度为低和为高的数据屏蔽掉，屏蔽掉之后 5 条数据，接着把这 5 条数据当成训练集来继续算哪个特征的信息增益最高，很明显算完之后是性别这个特征，所以这时候树变成了这样：



这时候呢，数据集里没有其他特征可以选择了（总共就两个特征，活跃度已经是根节点了），所以就看我性别是男或女的时候那种情况最有可能出现了。此时性别为男的用户中有 1 个是流失，1 个是不流失，五五开。所以可以考虑随机选个结果当输出了。性别为女的用户中有全部都流失，所以性别为女时输出是流失。所以呢，树就成了这样：



好了，决策树构造好了。从图可以看出决策树有一个非常好的地方就是模型的解释性非常强！！很明显，如果现在来了一条数据(男, 高)的话，输出会是不流失。

群众的力量是伟大的-随机森林

既然有决策树，那有没有用多棵决策树组成森林的算法呢？有！那就是随机森林。随机森林是一种叫**Bagging**的算法框架的变体。所以想要理解随机森林首先要理解**Bagging**。

Bagging

什么是**Bagging**

Bagging 是 Bootstrap Aggregating 的英文缩写，刚接触的您不要误认为 **Bagging** 是一种算法，**Bagging** 是集成学习中的学习框架，**Bagging** 是并行式集成学习方法。大名鼎鼎的随机森林算法就是在 $Bagging$ 的基础上修改的算法。

Bagging 方法的核心思想就是三个臭皮匠顶个诸葛亮。如果使用 **Bagging** 解决分类问题，就是将多个分类器的结果整合起来进行投票，选取票数最高的结果作为最终结果。如果使用 **Bagging** 解决回归问题，就将多个回归器的结果加起来然后求平均，将平均值作为最终结果。

那么 **Bagging** 方法如此有效呢，举个例子。狼人杀我相信您应该玩过，在天黑之前，村民们都要根据当天所发生的事和别人的发现来投票决定谁可能是狼人。

如果我们将每个村民看成是一个分类器，那么每个村民的任务就是二分类，假设 $h_i(x)$ 表示第 i 个村民认为 x 是不是狼人(-1 代表不是狼人，1 代表是狼人)，$f(x)$ 表示 x 真正的身份(是不是狼人)，ϵ 表示为村民判断错误的错误率。则有 $P(h_i(x) \neq f(x)) = \epsilon$。

根据狼人杀的规则，村民们需要投票决定天黑前谁是狼人，也就是说如果有超过半数的村民投票时猜对了，那么这一轮就猜对了。那么假设现在有 T 个村民，$H(x)$ 表示投票后最终的结果，则有 $H(x) = sign(\sum_{i=1}^{T} h_i(x))$。

现在假设每个村民都是有主见的人，对于谁是狼人都有自己的想法，那么他们的错误率也是相互独立的。那么根据**Hoeffding**不等式可知，$H(x)$ 的错误率为：

$$P(H(x) \neq f(x)) = \sum_{k=0}^{T/2} C_T^k (1-\epsilon)^k \epsilon^{T-k} \leq exp(-\frac{1}{2}T(1-2\epsilon)^2)$$

根据上式可知，如果 5 个村民，每个村民的错误率为 0.33 ，那么投票的错误率为 0.749 ；如果 20 个村民，每个村民的错误率为 0.33 ，那么投票的错误率为 0.315 ；如果 50 个村民，每个村民的错误率为 0.33 ，那么投票的错误率为 0.056 ；如果 100 个村民，每个村民的错误率为 0.33 ，那么投票的错误率为 0.003 。从结果可以看出，村民的数量越大，那么投票后犯错的错误率就越小。这也是**Bagging**性能强的原因之一。

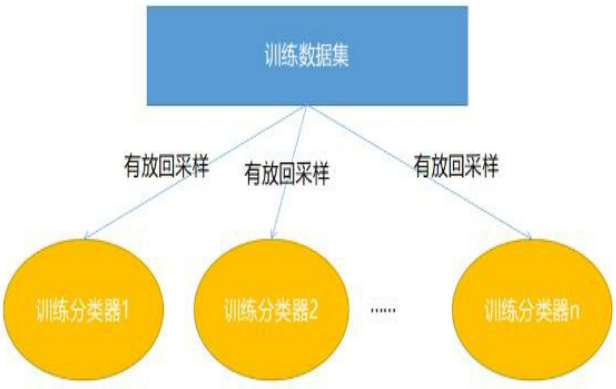
Bagging方法如何训练

Bagging 在训练时的特点就是随机有放回采样和并行。

随机有放回采样： 假设训练数据集有 m 条样本数据，每次从这 m 条数据中随机取一条数据放入采样集，然后将其返回，让下一次采样有机会仍然能被采样。然后重复 m 次，就能得到拥有 m 条数据的采样集，该采样集作为 **Bagging** 的众多分类器中的一个作为训练数据集。假设有 T 个分类器（随便什么分类器），那么就重复 T 此随机有放回采样，构建出 T 个采样集分别作为 T 个分类器的训练数据集。

并行： 假设有 10 个分类器，在**Boosting**中，1 号分类器训练完成之后才能开始2 号分类器的训练，而在**Bagging**中，分类器可以同时进行训练，当所有分类器训练完成之后，整个**Bagging**的训练过程就结束了。

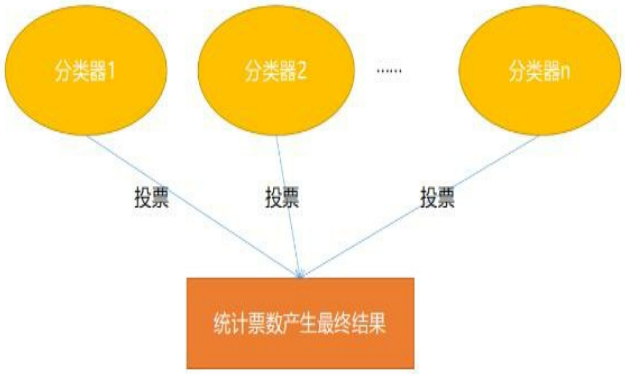
Bagging训练过程如下图所示：



Bagging方法如何预测

Bagging在预测时非常简单，就是投票！比如现在有 5 个分类器，有 3 个分类器认为当前样本属于 A 类，1 个分类器认为属于 B 类，1 个分类器认为属于 C 类，那么**Bagging**的结果会是 A 类（因为 A 类的票数最高）。

Bagging预测过程如下图所示:



随机森林

随机森林是**Bagging**的一种扩展变体，随机森林的训练过程相对与**Bagging**的训练过程的改变有：

- 基学习器：**Bagging**的基学习器可以是任意学习器，而随机森林则是以决策树作为基学习器。
- 随机属性选择：假设原始训练数据集有 10 个特征，从这 10 个特征中随机选取 k 个特征构成训练数据子集，然后将这个子集作为训练集扔给决策树去训练。其中 k 的取值一般为 $log2$ (特征数量)。

这样的改动通常会使得随机森林具有更加强的泛化性，因为每一棵决策树的训练数据集是随机的，而且训练数据集中的特征也是随机抽取的。如果每一棵决策树模型的差异比较大，那么就很容易能够解决决策树容易过拟合的问题。

用概率说话-朴素贝叶斯分类器

朴素贝叶斯分类算法是基于贝叶斯理论和特征条件独立假设的分类算法。对于给定的训练集，首先基于特征条件独立假设学习数据的概率分布。然后基于此模型，对于给定的特征数据 `x`，利用贝叶斯定理计算出标签 `y`。朴素贝叶斯分类算法实现简单，预测的效率很高，是一种常用的分类算法。

条件概率

朴素贝叶斯分类算法是基于贝叶斯定理与特征条件独立假设的分类方法，因此想要了解朴素贝叶斯分类算法背后的算法原理，就不得不用到概率论的一些知识，首当其冲就是条件概率。

什么是条件概率

概率指的是某一事件 `A` 发生的可能性，表示为 `P(A)`。而条件概率指的是某一事件 `A` 已经发生了条件下，另一事件 `B` 发生的可能性，表示为 `P(B|A)`，举个例子：

今天有 `25%` 的可能性下雨，即 `P(下雨)=0.25`；今天 `75%` 的可能性是晴天，即 `P(晴天)=0.75`；如果下雨，我有 `75%` 的可能性穿外套，即 `P(穿外套|下雨)=0.75`；如果下雨，我有 `25%` 的可能性穿T恤，即 `P(穿T恤|下雨)=0.25`；

从上述例子可以看出，条件概率描述的是 `|` 右边的事件已经发生之后，左边的事件发生的可能性，而不是两个事件同时发生的可能性！

怎样计算条件概率

设 `A, B` 是两个事件，且 `P(A)>0`，称 `P(B|A)=P(AB)/P(A)` 为在事件 `A` 发生的条件下，事件 `B` 发生的条件概率。(其中 `P(AB)` 表示事件**A**和事件**B**同时发生的概率)

举个例子，现在有一个表格，表格中统计了甲乙两个厂生产的产品中合格品数量、次品数量的数据。数据如下：

	甲厂	乙厂	合计
合格品	475	644	1119
次品	25	56	81
合计	500	700	1200

现在想要算一下已知产品是甲厂生产的，那么产品是次品的概率是多少。这个时候其实就是在算条件概率，计算非常简单。

假设事件 `A` 为产品是甲厂生产的，事件 `B` 为产品是次品。则根据表中数据可知 `P(AB)=25/1200`， `P(A)=500/1200`。则 `P(B|A)=P(AB)/P(A)=25/500`。

乘法定理

将条件概率的公式两边同时乘以 `P(A)`，就变成了乘法定理，即 `P(AB)=P(B|A)*P(A)`。那么乘法定理怎么用呢？举个例子：

现在有一批产品共 `100` 件，次品有 `10` 件，从中不放回地抽取 `2` 次，每次取 `1` 件。现在想要算一下第一次为次品，第二次为正品的概率。

从问题来看，这个问题问的是第一次为次品，第二次为正品这两个事件同时发生的概率。所以可以用乘法定理来解决这个问题。

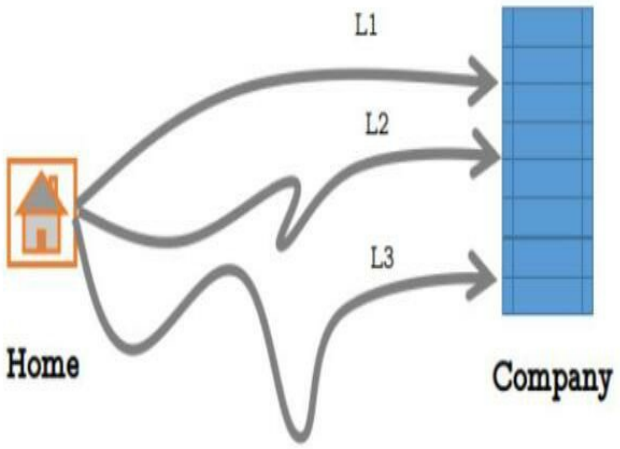
假设事件 `A` 为第一次为次品，事件 `B` 为第二次为正品。则 `P(AB)=P(A)*P(B|A)=(10/100)*(90/99)=0.091`。

全概率公式

贝叶斯公式是朴素贝叶斯分类算法的核心数学理论，在了解贝叶斯公式之前，我们需要先了解全概率公式的相关知识。

引例

小明从家到公司上班总共有三条路可以直达，如下图：



但是每条路每天拥堵的可能性不太一样，由于路的远近不同，选择每条路的概率如下表所示：

L1	L2	L3
0.5	0.3	0.2

每天从上述三条路去公司时不堵车的概率如下表所示：

L1不堵车	**L2**不堵车	**L3**不堵车
0.2	0.4	0.7

如果不堵车就不会迟到，现在小明想要算一算去公司上班不会迟到的概率是多少，应该怎么办呢？

其实很简单，假设事件 `C` 为小明不迟到，事件 `A1` 为小明选 `L1` 这条路并且不堵车，事件 `A2` 为小明选 `L2` 这条路并且不堵车，事件 `A3` 为小明选 `L3` 这条路并且不堵车。那么很显然 `P(C)=P(A1)+P(A2)+P(A3)` 。

那么问题来了， `P(A1)` 、 `P(A2)` 和 `P(A3)` 怎么算呢？其实只要会算 `P(A1)` 其他的就都会算了。我们同样可以假设事件 `D1` 为小明选择 `L1` 路，事件 `E1` 为不堵车。那么 `P(A1)=P(D1)*P(E1)` 。但是在从表格中我们只知道 `P(D1)=0.5` ，怎么办呢？

回忆一下上面介绍的乘法定理，不难想到 `P(A1)=P(D1)*P(E1|D1)` 。从表格中可以看出 `P(E1|D1)=0.2` 。因此 `P(A1)=0.5*0.2=0.1` 。

然后依葫芦画瓢可以很快算出， `P(A2)=0.3*0.4=0.12` ， `P(A3)=0.2*0.7=0.14` 。所以 `P(C)=0.1+0.12+0.14=0.36` 。

全概率公式

当为了达到某种目的，但是达到目的有很多种方式，如果想知道通过所有方式能够达到目的的概率是多少的话，就需要用到全概率公式（上面的例子就是这种情况！）。全概率公式的定义如下：

若事件$B_1,B_2,...,B_n$两两互不相容，并且其概率和为 `1` 。那么对于任意一个事件 `C` 都满足：

$$P(C)=P(B_1)P(C|B_1)+...+P(B_n)P(C|B_n)=\sum_{i=1}^{n}P(B_i)P(C|B_i)$$

引例中小明选择哪条路去公司的概率是两两互不相容的（只能选其中一条路去公司），并且和为 `1` 。所以小明不迟到的概率可以通过全概率公式来计算，而引例中的计算过程就是用的全概率公式。

贝叶斯公式

当已知引发事件发生的各种原因的概率，想要算该事件发生的概率时，我们可以用全概率公式。但如果现在反过来，已知事件已经发生了，但想要计算引发该事件的各种原因的概率时，我们就需要用到贝叶斯公式了。

贝叶斯公式定义如下，其中A表示已经发生的事件，B_i为导致事件A发生的第i个原因：

$$P(B_i|A)=\frac{P(A|B_i)P(B_i)}{\sum_{i=1}^{n}P(A|B_i)P(B_i)}$$

贝叶斯公式看起来比较复杂，其实非常简单，分子部分是乘法定理，分母部分是全概率公式（分母等于$P(A)$）。

如果我们对贝叶斯公式进行一个简单的数学变换（两边同时乘以分母，再两边同时除以$P(B_i)$）。就能够得到如下公式：

$$P(B|A)P(A)$$

$$P(A|B_i) = \frac{P(B_i|A)P(A)}{P(B_i)}$$

贝叶斯算法流程

在炎热的夏天你可能需要买一个大西瓜来解暑，但虽然你的挑西瓜的经验很老道，但还是会有挑错的时候。尽管如此，你可能还是更愿意相信自己经验。假设现在在你面前有一个纹路清晰，拍打西瓜后声音浑厚，按照你的经验来看这个西瓜是好瓜的概率有 `80` %，不是好瓜的概率有 `20` %。那么在这个时候你下意识会认为这个西瓜是好瓜，因为它是好瓜的概率大于不是好瓜的概率。

朴素贝叶斯分类算法的预测流程

朴素贝叶斯分类算法的预测思想和引例中挑西瓜的思想一样，会根据以往的经验计算出待预测数据分别为所有类别的概率，然后挑选其中概率最高的类别作为分类结果。

假如现在一个西瓜的数据如下表所示：

颜色	声音	纹理	是否为好瓜
绿	清脆	清晰	?

若想使用朴素贝叶斯分类算法的思想，根据这条数据中 `颜色` 、 `声音` 和 `纹理` 这三个特征来推断是不是好瓜，我们需要计算出这个西瓜是好瓜的概率和不是好瓜的概率。

假设事件 `A1` 为好瓜，事件 `B` 为绿，事件 `C` 为清脆，事件 `D` 为清晰，则这个西瓜是好瓜的概率为 `P(A1|BCD)` 。根据贝叶斯公式可知:

$$P(A_1|BCD) = \frac{P(A_1)P(B|A_1)P(C|A_1)P(D|A_1)}{P(BCD)}$$

同样，假设事件 `A2` 为好瓜，事件 `B` 为绿，事件 `C` 为清脆，事件 `D` 为清晰，则这个西瓜不是好瓜的概率为 `P(A2|BCD)` 。根据贝叶斯公式可知:

$$‘‘‘latex P(A_2|BCD) = \frac{P(A_2)P(B|A_2)P(C|A_2)P(D|A_2)}{P(BCD)}‘‘‘$$

朴素贝叶斯分类算法的思想是取概率最大的类别作为预测结果，所以如果满足下面的式子，则认为这个西瓜是好瓜，否则就不是好瓜：

$$\frac{P(A_1)P(B|A_1)P(C|A_1)P(D|A_1)}{P(BCD)} > \frac{P(A_2)P(B|A_2)P(C|A_2)P(D|A_2)}{P(BCD)}$$

从上面的式子可以看出， `P(BCD)` 是多少对于判断哪个类别的概率高没有影响，所以式子可以简化成如下形式：

$$P(A_1)P(B|A_1)P(C|A_1)P(D|A_1) > P(A_2)P(B|A_2)P(C|A_2)P(D|A_2)$$

所以在预测时，需要知道 `P(A1)` ， `P(A2)` ， `P(B|A_1)` ， `P(C|A_1)` ， `P(D|A_1)` 等于多少。而这些概率在训练阶段可以计算出来。

朴素贝叶斯分类算法的训练流程

训练的流程非常简单，主要是计算各种条件概率。假设现在有一组西瓜的数据，如下表所示：

编号	颜色	声音	纹理	是否为好瓜
1	绿	清脆	清晰	是
2	黄	浑厚	模糊	否
3	绿	浑厚	模糊	是
4	绿	清脆	清晰	是
5	黄	浑厚	模糊	是
6	绿	清脆	清晰	否

从表中数据可以看出：

`P(是好瓜)=4/6` `P(颜色绿|是好瓜)=3/4` `P(颜色黄|是好瓜)=1/4` `P(声音清脆|是好瓜)=1/2` `P(声音浑厚|是好瓜)=1/2` `P(纹理清晰|是好瓜)=1/2` `P(纹理模糊|是好瓜)=1/2` `P(不是好瓜)=2/6` `P(颜色绿|不是好瓜)=1/2` `P(颜色黄|是好瓜)=1/2` `P(声音清脆|不是好瓜)=1/2` `P(声音浑厚|不是好瓜)=1/2` `P(纹理清晰|不是好瓜)=1/2` `P(纹理模糊|不是好瓜)=1/2`

当得到以上概率后，训练阶段的任务就已经完成了。我们不妨再回过头来预测一下这个西瓜是不是好瓜。

颜色	声音	纹理	是否为好瓜
绿	清脆	清晰	?

假设事件 `A1` 为好瓜，事件 `B` 为绿，事件 `C` 为清脆，事件 `D` 为清晰。则有：

$$P(A_1)P(B|A_1)P(C|A_1)P(D|A_1) = \frac{4}{6} * \frac{3}{4} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}$$

假设事件 `A2` 为不是瓜，事件 `B` 为绿，事件 `C` 为清脆，事件 `D` 为清晰。则有：

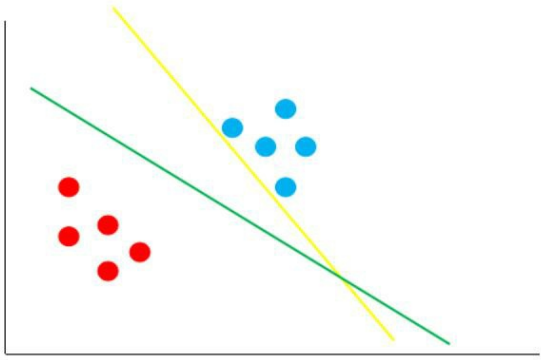
$$P(A_2)P(B|A_2)P(C|A_2)P(D|A_2) = \frac{2}{6} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{24}$$

由于$\frac{1}{8} > \frac{1}{24}$，所以这个西瓜是好瓜。

支持向量机

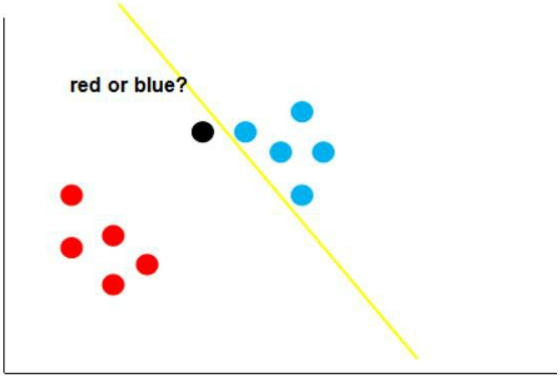
线性可分支持向量机

例如逻辑回归这种广义线性模型对数据进行分类时，其实本质就是找到一条 决策边界 ，然后将数据分成两个类别(边界的一边是一种类别，另一边是另一种类别)。如下图中的两条线是两种模型产生的 决策边界 ：



图中的绿线与黄线都能很好的将图中的红点与蓝点给区分开。但是，哪条线的泛化性更好呢？可能你不太了解泛化性，也就是说，我们的这条直线，不仅需要在训练集 **(**已知的数据**)** 上能够很好的将红点跟蓝点区分开来，还要在测试集 **(**未知的数据**)** 上将红点和蓝点给区分开来。

假如经过训练，我们得到了黄色的这条决策边界用来区分我们的数据，这个时候又来了一个数据，即黑色的点，那么你觉得黑色的点是属于红的这一类，还是蓝色的这一类呢？



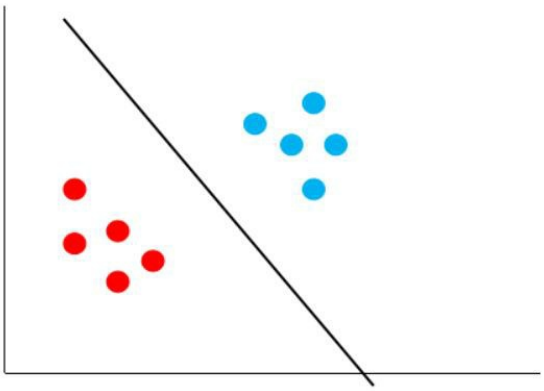
如上图，根据黄线的划分标准，黑色的点应该属于红色这一类。可是，我们肉眼很容易发现，黑点离蓝色的点更近，它应该是属于蓝色的点。这就说明，黄色的这条直线它的泛化性并不好，它对于未知的数据并不能很好的进行分类。那么，如何得到一条泛化性好的直线呢？这个就是支持向量机考虑的问题。

基本思想

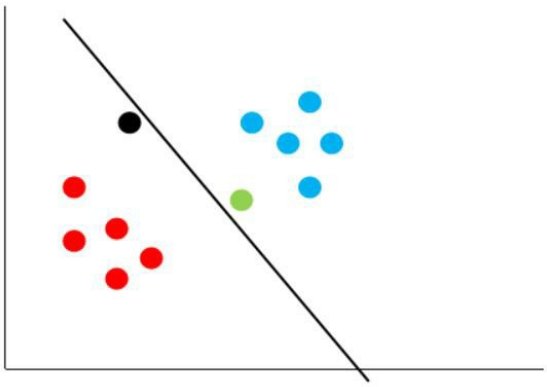
支持向量机的思想认为，一条决策边界它如果要有很好的泛化性，它需要满足一下以下两个条件：

- 能够很好的将样本划分
- 离最近的样本点最远

比如下图中的黑线：



它能够正确的将红点跟蓝点区分开来，而且，它还保证了对未知样本的容错率，因为它离最近的红点跟蓝点都很远，这个时候，再来一个数据，就不会出现之前黄色决策边界的错误了。



无论新的数据出现在哪个位置，黑色的决策边界都能够很好的给它进行分类，这个就是支持向量机的基本思想。

间隔与支持向量

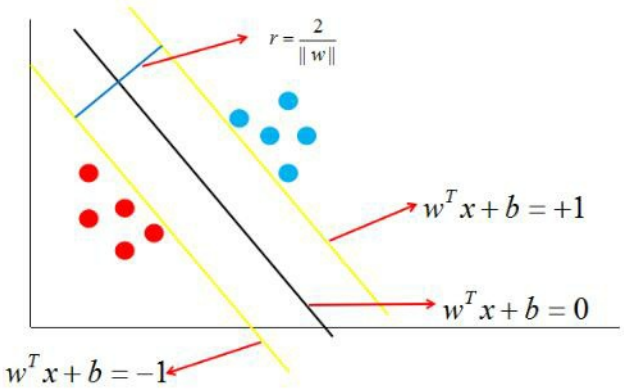
在样本空间中，决策边界可以通过如下线性方程来描述：

$$w^T x + b = 0$$

其中$w = (w_1, w_2, .., w_d)$为法向量，决定了决策边界的方向。b为位移项，决定了决策边界与原点之间的距离。显然，决策边界可被法向量和位移确定，我们将其表示为(w, b)。样本空间中的任意一个点x,到决策边界(w, b)的距离可写为：

$$r = \frac{|w^T x + b|}{||w||}$$

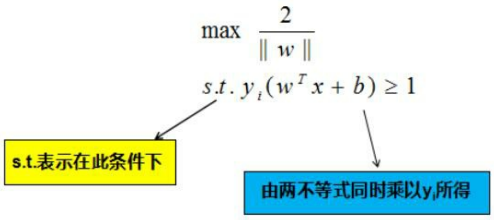
假设决策边界(w, b)能够将训练样本正确分类，即对于任何一个样本点(x_i, y_i),若它为正类，即$y_i = +1$时，$w^T x + b \geq +1$。若它为负类，即$y_i = -1$时，$w^T x + b \leq -1$。



如图中，距离最近的几个点使两个不等式的等号成立，它们就被称为 **支持向量** ，即图中两条黄色的线。两个异类支持向量到超平面的距离之和为：

$$r = \frac{2}{||w||}$$

它被称为 **间隔** ，即蓝线的长度。欲找到具有“最大间隔”的决策边界，即黑色的线，也就是要找到能够同时满足如下式子的w与b:



显然，为了最大化间隔，仅需最大化$||w||^{-1}$,这等价于最小化$||w||^2$，于是，条件可以重写为：

$$min\frac{1}{2}||w||^2$$

$$s.t.y_i(w^Tx+b)\geq 1$$

这就是**线性可分支持向量机**的基本型。 ### 对偶问题 我们已经知道了支持向量机的基本型，问题本身是一个凸二次规划问题，可以用现成的优化计算包求解，不过可以用更高效的方法。支持向量机的模型如下：

$$min\frac{1}{2}||W||^2$$

$$s.t.y_i(w^Tx+b)\geq 1$$

对两式子使用拉格朗日乘子法可得到其 **对偶问题** 。具体为，对式子的每条约束添加拉格朗日乘子$\alpha\geq 0$，则该问题的拉格朗日函数可写为：

$$L(w,b,\alpha)=\frac{1}{2}||w||^2+\sum_{i=1}^m\alpha_i(1-y_i(w^Tx_i+b))\text{.....}(1)$$

其中$\alpha=(\alpha_1,\alpha_2,..,\alpha_m)$。令$L(w,b,\alpha)$对$w$和$b$的偏导为零可得：

$$w=\sum_{i=1}^m\alpha_iy_ix_i\text{.....}(2)$$

$$0=\sum_{i=1}^m\alpha_iy_i\text{.....}(3)$$

将式子2带入式子1，则可将w和b消去，再考虑式子3的约束，就可得到原问题的 **对偶问题** ：

$$max(\sum_{i=1}^m\alpha_i-\frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m\alpha_i\alpha_jy_iy_jx_i^Tx_j)$$

$$s.t.\sum_{i=1}^m\alpha_iy_i=0$$

$$\alpha_i\geq 0$$

解出α后，求出w与b即可得到模型：

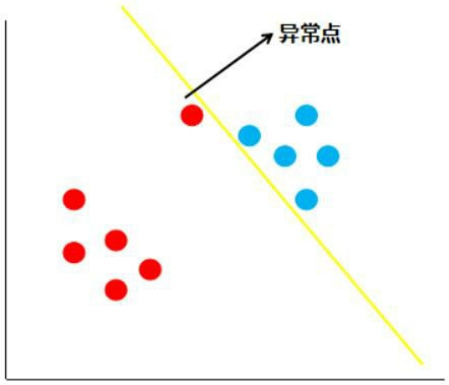
$$f(x)=w^Tx+b=\sum_{i=1}^m\alpha_iy_ix_i^Tx+b\text{.....}(4)$$

从对偶问题解出的α_i是拉格朗日乘子，它恰对应着训练样本(x_i,y_i)，由于原问题有不等式的约束，因此上述过程需满足$KKT(Karush-Kuhn-Tucker)$条件,即要求：

$$\begin{cases}\alpha_i\geq 0\\ y_if(x_i)-1\geq 0\\ \alpha_i(y_if(x_i)-1)=0\end{cases}$$

于是，对任意训练样本(x_i,y_i)，总有$\alpha_i=0$或$y_if(x_i)=1$。若$\alpha_i=0$，则该样本将不会出现在式子4中，也就不会对$f(x)$，有任何影响。若

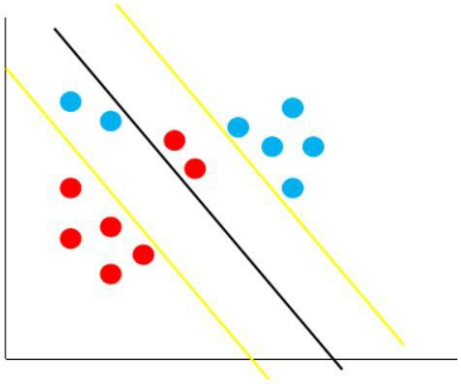
$\alpha_i>0$，则必有$y_if(x_i)=1$，所对应的样本点位于最大间隔边界上，是一个支持向量。这显示出支持向量机的一个重要性质：**训练完后，大部分的训练样本都不需要保留，最终模型仅与支持向量有关** 。 ## 线性支持向量机 假如现在有一份数据分布如下图：



按照线性可分支持向量机的思想，黄色的线就是最佳的决策边界。很明显，这条线的泛化性不是很好，造成这样结果的原因就是数据中存在着异常点，那么如何解决这个问题呢，支持向量机引入了 **软间隔最大化** 的方法来解决。 所谓的 **软间隔** ，是相对于硬间隔说的，刚刚在 **间隔与支持向量** 中提到的间隔就是硬间隔。 接着我们再看如何可以软间隔最大化呢？SVM对训练集里面的每个样本(x_i,y_i)引入了一个松弛变量$\xi_i \geq 0$,使函数间隔加上松弛变量大于等于1，也就是说：

$$y_i(w^Tx+b) \geq 1-\xi$$

对比硬间隔最大化，可以看到我们对样本到超平面的函数距离的要求放松了，之前是一定要大于等于1，现在只需要加上一个大于等于0的松弛变量能大于等于1就可以了。也就是允许支持向量机在一些样本上出错，如下图：



当然，松弛变量不能白加，这是有成本的，每一个松弛变量ξ_i, 对应了一个代价ξ_i，这个就得到了我们的 **软间隔最大化的支持向量机** ，模型如下：

$$min\frac{1}{2}||w||^2+C\sum_{i=1}^m\xi_i$$

$$s.t. y_i(w^Tx_i+b) \geq 1-\xi_i$$

$$\xi_i \geq 0$$

这里，$C>0$为惩罚参数，可以理解为我们一般回归和分类问题正则化时候的参数。C越大，对误分类的惩罚越大，C越小，对误分类的惩罚越小。也就是说，我们希望权值的二范数尽量小，误分类的点尽可能的少。C是协调两者关系的正则化惩罚系数。在实际应用中，需要调参来选择。 同样的，使用 **拉格朗日函数** 将软间隔最大化的约束问题转化为无约束问题，利用相同的方法得到数学模型：

$$max(\sum_{i=1}^m\alpha_i-\frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m\alpha_i\alpha_jy_iy_jx_i^Tx_j)$$

$$s.t. \sum_{i=1}^m\alpha_iy_i=0$$

$$0 \leq \alpha_i \leq C$$

这就是软间隔最大化时的 **线性支持向量机** 的优化目标形式，和之前的硬间隔最大化的线性可分支持向量机相比，我们仅仅是多了一个约束条件:

$$0 \leq \alpha_i \leq C$$

smo算法 **序列最小优化算法** (smo算法)是一种启发式算法， SVM中的模型参数可以使用该算法进行求解。其基本思路是：如果所有变量的解都满足此最优化问题的KKT条件，那么这个最优化问题的解就得到了。因为KKT条件是该最优化问题的充分必要条件。否则,选择两个变量，固定其他变量，针对这两个变量构建一个二次规划问题。这个二次规划问题关于这两个变量的解应该更接近原始二次规划问题的解，因为这会使得原始二次规划问题的目标函数值变得更小。重要的是，这时子问题可以通过解析方法求解，这样就可以大大提高整个算法的计算

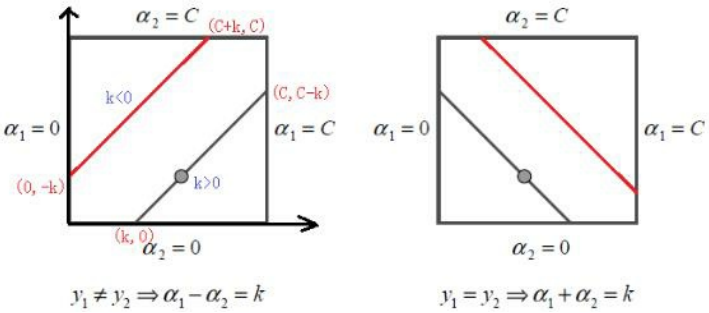
速度。子问题有两个变量,一个是违反KKT条件最严重的那一个，另一个由约束条件自动确定。如此，smo算法将原问题不断分解为子问题并对子问题求解，进而达到求解原问题的目的。具体如下： 选择两个变量α_1, α_2，其它变量$\alpha_i, i = 3, 4, ..m$是固定的。 于是，smo的最优化问题的子问题可以写成：

$$min(\frac{1}{2}k_{11}\alpha_1^2 + \frac{1}{2}k_{22}\alpha 2^2 + y_1y_2k_{12}\alpha_1\alpha_2 - (\alpha_1 + \alpha_2) + y_1\alpha_1\sum_{i=3}^m y_i\alpha_ik_{i1} + y_2\alpha_2\sum_{i=3}^m y_i\alpha_ik_{i2})$$

$$s.t. \alpha_1y_1 + \alpha_2y_2 = -\sum_{i=3}^m y_i\alpha_i = \xi$$

$$0 \le \alpha_i \le C, i = 1, 2$$

其中：$k_{ij} = k(x_i, x_j)$ ### smo算法目标函数的优化 为了求解上面含有这两个变量的目标优化问题，我们首先分析约束条件，所有的α_1, α_2都要满足约束条件，然后在约束条件下求最小。 根据上面的约束条件$\alpha_1y_1 + \alpha_2y_2 = \xi, 0 \le \alpha_i \le C, i = 1, 2$，又由于$y_1, y_2$均只能取值$1$或者$-1$, 这样$\alpha_1, \alpha_2$在$[0, C]$和$[0, C]$形成的盒子里面，并且两者的关系直线的斜率只能为$1$或者$-1$，也就是说$\alpha_1, \alpha_2$的关系直线平行于$[0, C]$和$[0, C]$形成的盒子的对角线，如下图所示：



由于α_1, α_2的关系被限制在盒子里的一条线段上，所以两变量的优化问题实际上仅仅是一个变量的优化问题。不妨我们假设最终是α_2的优化问题。由于我们采用的是启发式的迭代法，假设我们上一轮迭代得到的解是$\alpha_1^{old}, \alpha_2^{old}$，假设沿着约束方向$\alpha_2$未经剪辑的解是$\alpha_2^{new,unc}$。本轮迭代完成后的解为$\alpha_1^{new}, \alpha_2^{new}$。 由于$\alpha_2^{new}$必须满足上图中的线段约束。假设$L$和$H$分别是上图中$\alpha_2^{new}$所在的线段的边界。那么很显然我们有：

$$L \le \alpha_2^{new} \le H$$

而对于L和H，我们也有限制条件如果是上面左图中的情况，则：

$$L = max(0, \alpha_2^{old} - \alpha_1^{old}), H = min(C, C + \alpha_2^{old} - \alpha_1^{old})$$

如果是上面右图中的情况，我们有：

$$L = max(0, \alpha_2^{old} + \alpha_1^{old} - C), H = min(C, \alpha_2^{old} + \alpha_1^{old})$$

也就是说，假如我们通过求导得到的$\alpha_2^{new,unc}$，则最终的α_2^{new}应该为：

$$\alpha_2^{new,unc} = \begin{cases} H, \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, L \le \alpha_2^{new,unc} \le H \\ L, \alpha_2^{new,unc} < L \end{cases}$$

那么如何求出α_2^{new}呢？很简单，我们只需要将目标函数对$\alpha 2$求偏导数即可。首先我们整理下我们的目标函数。为了简化叙述，我们令:

$$E_i = g(x_i) - y_i$$

其中：

$$g(x) = w.\emptyset(x) + b$$

$$v_i = g(x_i) - \sum_{j=1}^2 y_j\alpha_jk(x_i, x_j) - b$$

最终可以计算得：

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{k11 + k22 - 2k12}$$

smo算法两个变量的选择 smo算法需要选择合适的两个变量做迭代，其余的变量做常量来进行优化，那么怎么选择这两个变量呢？ 一般来说，我们首先选择违反$0 < \alpha_i < C$这个条件的点。如果这些支持向量都满足KKT条件，再选择违反$\alpha_i = 0$和$\alpha_i = C$的点。 smo算法称选择第二一个变量为内层循环，假设我们在外层循环已经找到了α_1, 第二个变量α_2的选择标准是让$|E1 - E2|$有足够大的变化。由于α_1定了的时候,$E1$也确定了，所以要想$|E1 - E2|$最大，只需要在$E1$为正时，选择最小的Ei作为$E2$， 在$E1$为负时，选择最大的Ei作为$E2$，可以将所有的Ei

保存下来加快迭代。 如果内存循环找到的点不能让目标函数有足够的下降， 可以采用遍历支持向量点来做α_2,直到目标函数有足够的下降， 如果所有的支持向量做α_2都不能让目标函数有足够的下降，可以跳出循环，重新选择α_1。 ### 计算阈值b和差值Ei 在每次完成两个变量的优化之后，需要重新计算阈值b。当$0 < \alpha_1^{new} < C$时，我们有:

$$y_1 - \sum_{i=1}^{m} \alpha_i y_i k_{i1} - b_1 = 0$$

于是新的b_1^{new}为：

$$b_1^{new} = y_1 - \sum_{i=3}^{m} \alpha_i y_i k_{i1} - \alpha_1^{new} y_1 k_{11} - \alpha 2^{new} y_2 k_{21}$$

用E_1表示为：

$$b_1^{new} = -E_1 - y_1 k_{11}(\alpha_1^{new} - \alpha_1^{old}) - y_2 k_{21}(\alpha_2^{new} - \alpha_2^{old}) - b^{old}$$

同样的，如果$0 < \alpha_2^{new} < C$，则：

$$b_2^{new} = -E_2 - y_1 k_{12}(\alpha_1^{new} - \alpha_1^{old}) - y_2 k_{22}(\alpha_2^{new} - \alpha_2^{old}) - b^{old}$$

最终：

$$b^{new} = \frac{b_1^{new} + b_2^{new}}{2}$$

继续更新得：

$$E_i = \sum_s y_j \alpha_j k(x_i, x_j) + b^{new} - y_i$$

其中 s 为所有支持向量x_j的集合。

序列最小优化算法流程

1 .取初始值$\alpha^0 = 0, k = 0$ 2 .计算$\alpha_2^{new,unc}$，即：

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{k_{11} + k_{22} - 2k_{12}}$$

3 .计算α_2^{k+1}，即：

$$\alpha_2^{k+1} = \begin{cases} H, \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, L \le \alpha_2^{new,unc} \le H \\ L, \alpha_2^{new,unc} < L \end{cases}$$

4 .利用α_2^{k+1}和α_1^{k+1}关系求出α_1^{k+1}

5 .计算b^{k+1}和E_i 6 .在精度范围内检查是否满足如下的终止条件：

$$\sum_{i=1}^{m} \alpha_i y_i = 00 \le \alpha_i \le C \alpha_i^{k+1} = 00 < \alpha_i^{k+1} < C \alpha_i^{k+1} = C$$

7 .如果满足则结束，返回α^{k+1}，否则转向步骤 2

物以类聚人以群分-k Means

k Means是属于机器学习里面的非监督学习，通常是大家接触到的第一个聚类算法，其原理非常简单，是一种典型的基于距离的聚类算法。距离指的是每个样本到质心的距离。那么，这里所说的质心是什么呢？

其实，质心指的是样本每个特征的均值所构成的一个坐标。举个例子：假如有两个数据 $(1,1)$ 和$(2,2)$ 则这两个样本的质心为 $(1.5,1.5)$。

同样的，如果一份数据有 m 个样本，每个样本有 n 个特征，用 x_i^j 来表示第 j 个样本的第 i 个特征，则它们的质心为：

$Cmass = (\frac{\sum_{j=1}^{m} x_1^j}{m}, \frac{\sum_{j=1}^{m} x_2^j}{m}, ..., \frac{\sum_{j=1}^{m} x_n^j}{m})$。

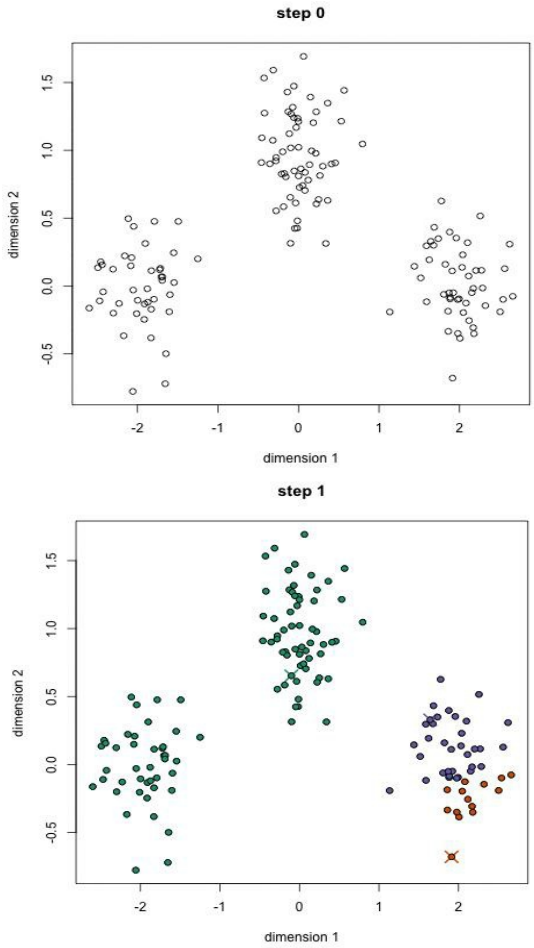
知道什么是质心后，就可以看看**k Means**算法的流程了。

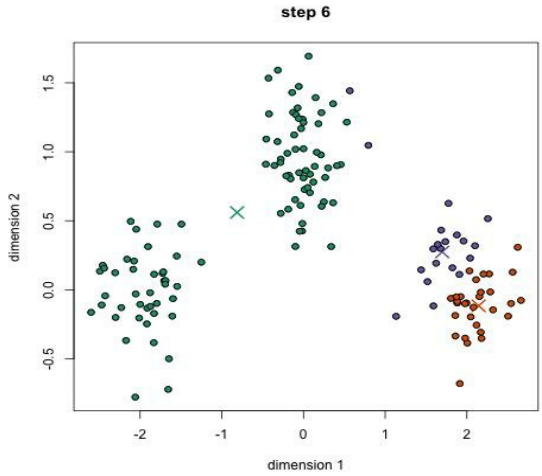
k Means算法流程

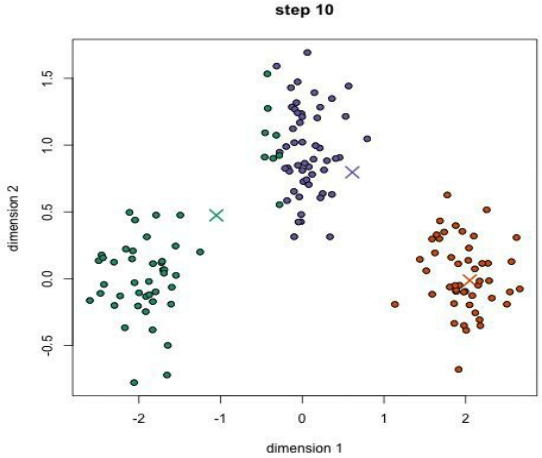
使用**k Means**来聚类时需要首先定义参数**k**，**k**的意思是我想将数据聚成几个类别。假设**k=3**，就是将数据划分成**3**个类别。接下来就可以开始**k Means**算法的流程了，流程如下：

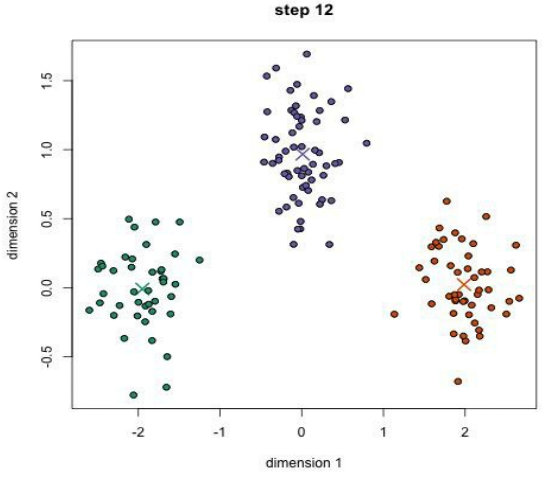
1. 随机初始**k**个样本，作为类别中心。 2. 对每个样本将其标记为距离类别中心最近的类别。 3. 将每个类别的质心更新为新的类别中心。 4. 重复步骤 2 、 3 ，直到类别中心的变化小于阈值。

过程示意图如下（其中 X 表示类别的中心，数据点的颜色代表不同的类别，总共迭代 12 次，下图为部分迭代的结果）：




step 6
dimension 2
dimension 1
-2
-1
0
1
2
-0.5
0.0
0.5
1.0
1.5


step 10
dimension 2
dimension 1
-2
-1
0
1
2
-0.5
0.0
0.5
1.0
1.5


step 12
dimension 2
dimension 1
-2
-1
0
1
2
-0.5
0.0
0.5
1.0
1.5

以距离为尺-AGNES算法

AGNES 算法是一种聚类算法，最初将每个对象作为一个簇，然后这些簇根据某些距离准则被一步步地合并。两个簇间的相似度有多种不同的计算方法。聚类的合并过程反复进行直到所有的对象最终满足簇数目。所以理解 AGNES 算法前需要先理解一些距离准则。

距离准则

为什么需要距离

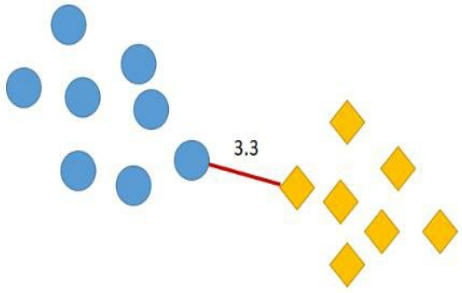
AGNES 算法是一种自底向上聚合的层次聚类算法，它先会将数据集中的每个样本看作一个初始簇，然后在算法运行的每一步中找出距离最近的两个簇进行合并，直至达到预设的簇的数量。所以AGNES算法需要不断的计算簇之间的距离，这也符合聚类的核心思想（物以类聚，人以群分），因此怎样度量两个簇之间的距离成为了关键。

距离的计算

衡量两个簇之间的距离通常分为最小距离、最大距离和平均距离。在 AGNES 算法中可根据具体业务选择其中一种距离作为度量标准。

最小距离

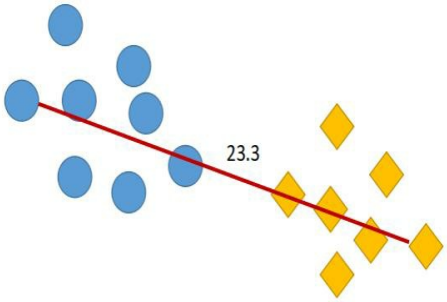
最小距离描述的是两个簇之间距离最近的两个样本所对应的距离。例如下图中圆圈和菱形分别代表两个簇，两个簇之间离得最近的样本的欧式距离为 3.3 ，则最小距离为 3.3。



假设给定簇C_i与C_j，则最小距离为：$d_{min} = min_{x \in i, z \in j} dist(x, z)$

最大距离

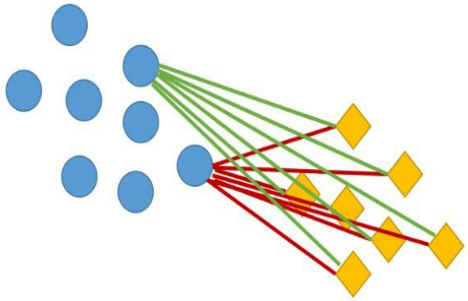
最大距离描述的是两个簇之间距离最远的两个样本所对应的距离。例如下图中圆圈和菱形分别代表两个簇，两个簇之间离得最远的样本的欧式距离为 23.3 ，则最大距离为 23.3 。



假设给定簇C_i与C_j，则最大距离为：$d_{min} = max_{x \in i, z \in j} dist(x, z)$

平均距离

平均距离描述的是两个簇之间样本的平均距离。例如下图中圆圈和菱形分别代表两个簇，计算两个簇之间的所有样本之间的欧式距离并求其平均值。



假设给定簇C_i与C_j，$|C_i|,|C_j|$分别表示簇 i 与簇 j 中样本的数量，则平均距离为：$d_{min} = \frac{1}{|C_i||C_j|}\sum_{x \in i}\sum_{z \in j} dist(x,z)$

AGNES 算法流程

AGNES 算法是一种自底向上聚合的层次聚类算法，它先会将数据集中的每个样本看作一个初始簇，然后在算法运行的每一步中找出距离最近的两个簇进行合并，直至达到预设的簇的数量。

举个例子，现在先要将西瓜数据聚成两类，数据如下表所示：

编号	体积	重量
1	1.2	2.3
2	3.6	7.1
3	1.1	2.2
4	3.5	6.9
5	1.5	2.5

一开始，每个样本都看成是一个簇(1 号样本看成是 1 号簇， 2 号样本看成是 2 号簇，...， 5 号样本看成是 5 号簇)，假设簇的集合为 C=[[1], [2], [3], [4], [5]] 。

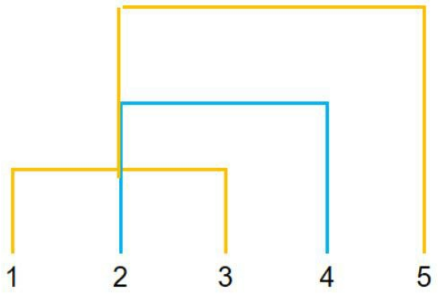
假设使用簇间最小距离来度量两个簇之间的远近，从表中可以看出 1 号簇与 3 号簇的簇间最小距离最小。因此需要将 1 号簇和 3 号簇合并，那么此时簇的集合 C=[[1, 3], [2], [4], [5]]。

然后继续看这 4 个簇中哪两个簇之间的最小距离最小，我们发现 2 号簇与 4 号簇的最小距离最小，因此我们要进行合并，合并之后 C=[[1, 3], [2, 4], [5]]。

然后继续看这 3 个簇中哪两个簇之间的最小距离最小，我们发现 5 号簇与 [1, 3] 簇的最小距离最小，因此我们要进行合并，合并之后 C=[[1, 3, 5], [2, 4]]。

这个时候 C 中只有两个簇了，达到了我们的预期目标（想要聚成两类），所以算法停止。算法停止后会发现，我们已经将 5 个西瓜，聚成了两类，一类是小西瓜，另一类是大西瓜。

如果将整个聚类过程中的合并，与合并的次序可视化出来，就能看出为什么说 AGNES 是自底向上的层次聚类算法了。



所以 AGNES 伪代码如下：

```
#假设数据集为D，想要聚成的簇的数量为k
def AGNES(D, k):
    #C为聚类结果
    C = []
    #将每个样本看成一个簇
    for d in D:
        C.append(d)
```

```
#C中簇的数量
q=len(C)
while q > k:
    寻找距离最小的两个簇a和b
    将a和b合并，并修改C
    q = len(C)
return C
```

当模型训练好之后，我们需要有指标来量化我们的模型的性能好不好。而且模型的性能不单单只有一个维度，所以模型的好坏通常会用多个指标来进行衡量。例如，现在想要衡量一个分类模型的性能，您可能第一时间会想到用准确率来衡量模型的好坏，但是准确率高并不一定就代表模型的性能高，因此可能会需要使用如`f1 score`、`AUC`等指标来衡量。所以在什么情况下使用什么样的性能评估指标，每个指标的数值的含义时什么，是我们在评估模型性能时必须要学会的知识。

本章主要涉及的知识点有：

- 常用分类性能评估指标
- 常用回归性能评估指标
- 常用聚类性能评估指标

分类模型性能评估指标

准确度的缺陷

准确度这个概念相信对于大家来说肯定并不陌生，就是正确率。例如模型的预测结果与数据真实结果如下表所示：

编号	预测结果	真实结果
1	1	2
2	2	2
3	3	3
4	1	1
5	2	3

很明显，连小朋友都能算出来该模型的准确度为 3/5 。

那么准确对越高就能说明模型的分类性能越好吗？非也！举个例子，现在我开发了一套癌症检测系统，只要输入你的一些基本健康信息，就能预测出你现在是否患有癌症，并且分类的准确度为 0.999 。您认为这样的系统的预测性能好不好呢？

您可能会觉得，哇，这么高的准确度！这个系统肯定很牛逼！但是我们知道，一般年轻人患癌症的概率非常低，假设患癌症的概率为 0.001 ，那么其实我这个癌症检测系统只要一直输出您没有患癌症，准确度也可能能够达到 0.999 。

假如现在有一个人本身已经患有癌症，但是他自己不知道自己患有癌症。这个时候用我的癌症检测系统检测发现他没有得癌症，那很显然我这个系统已经把他给坑了（耽误了治疗）。

看到这里您应该已经体会到了，一个分类模型如果光看准确度是不够的，尤其是对这种样本极度不平衡的情况（ 10000 条健康信息数据中，只有 1 条的类别是患有癌症，其他的类别都是健康）。

混淆矩阵

想进一步的考量分类模型的性能如何，可以使用其他的一些性能指标，例如精准率和召回率。但这些指标计算的基础是混淆矩阵。

继续以癌症检测系统为例，癌症检测系统的输出不是有癌症就是健康，这里为了方便，就用 1 表示患有癌症， 0 表示健康。假设现在拿 10000 条数据来进行测试，其中有 9978 条数据的真实类别是 0 ， 系统预测的类别也是 0 ，有 2 条数据的真实类别是 1 却预测成了 0 ，有 12 条数据的真实类别是 0 但预测成了 1 ，有 8 条数据的真实类别是 1 ，预测结果也是 1 。

如果我们把这些结果组成如下矩阵，则该矩阵就成为混淆矩阵。

真实\预测	**0**	**1**
0	9978	12
1	2	8

混淆矩阵中每个格子所代表的的意义也很明显，意义如下：

真实\预测	**0**	**1**
0	预测 0 正确的数量	预测 1 错误的数量
1	预测 0 错误的数量	预测 1 正确的数量

如果将正确看成是 True ，错误看成是 False ， 0 看成是 Negtive ， 1 看成是 Positive 。然后将上表中的文字替换掉，混淆矩阵如下：

真实\预测	**0**	**1**
0	TN	FP
1	FN	TP

因此 TN 表示真实类别是 Negtive ， 预测结果也是 Negtive 的数量； FP 表示真实类别是 Negtive ， 预测结果是 Positive 的数量； FN 表示真实类别是 Positive ， 预测结果是Negtive 的数量； TP 表示真实类别是 Positive ， 预测结果也是 Positive 的数量。

很明显，当 FN 和 FP 都等于 0 时，模型的性能应该是最好的，因为模型并没有在预测的时候犯错误。即如下混淆矩阵：

真实\预测	**0**	**1**

0	9978	0
1	0	22

所以模型分类性能越好，混淆矩阵中非对角线上的数值越小。

精准率

精准率**(Precision)**指的是模型预测为 Positive 时的预测准确度，其计算公式如下：

$Precisioin = \frac{TP}{TP+FP}$

假如癌症检测系统的混淆矩阵如下：

真实\预测	**0**	**1**
0	9978	12
1	2	8

则该系统的精准率=8/(8+12)=0.4 。

0.4 这个值表示癌症检测系统的预测结果中如果有 100 个人被预测成患有癌症，那么其中有 40 人是真的患有癌症。也就是说，精准率越高，那么癌症检测系统预测某人患有癌症的可信度就越高。

召回率

召回率**(Recall)**指的是我们关注的事件发生了，并且模型预测正确了的比值，其计算公式如下：

$Recall = \frac{TP}{FN+TP}$

假如癌症检测系统的混淆矩阵如下：

真实\预测	**0**	**1**
0	9978	12
1	2	8

则该系统的召回率=8/(8+2)=0.8。

从计算出的召回率可以看出，假设有 100 个患有癌症的病人使用这个系统进行癌症检测，系统能够检测出 80 人是患有癌症的。也就是说，召回率越高，那么我们感兴趣的对象成为漏网之鱼的可能性越低。

精准率与召回率之间的关系

假设有这么一组数据，菱形代表 Positive ，圆形代表 Negtive 。

现在需要训练一个模型对数据进行分类，假如该模型非常简单，就是在数据上画一条线作为分类边界。模型认为边界的左边是 Negtive ，右边是 Positive 。如果该模型的分类边界向左或者向右移动的话，模型所对应的精准率和召回率如下图所示：

从上图可知，模型的精准率变高，召回率会变低，精准率变低，召回率会变高。

F1 Score

上一关中提到了精准率变高，召回率会变低，精准率变低，召回率会变高。那如果想要同时兼顾精准率和召回率，这个时候就可以使用**F1 Score**来作为性能度量指标了。

F1 Score 是统计学中用来衡量二分类模型精确度的一种指标。它同时兼顾了分类模型的准确率和召回率。F1 Score 可以看作是模型准确率和召回率的一种加权平均，它的最大值是 1 ，最小值是0 。其公式如下：

$F1 = \frac{2*precision*recall}{precision+recall}$

- 假设模型 A 的精准率为 0.2 ，召回率为 0.7 ，那么模型 A 的 F1 Score 为 0.31111 。
- 假设模型 B 的精准率为 0.7 ，召回率为 0.2 ，那么模型 B 的 F1 Score 为 0.31111 。
- 假设模型 C 的精准率为 0.8 ，召回率为 0.7 ，那么模型 C 的 F1 Score 为 0.74667 。
- 假设模型 D 的精准率为 0.2 ，召回率为 0.3 ，那么模型 D 的 F1 Score 为 0.24 。

从上述 4 个模型的各种性能可以看出，模型C的精准率和召回率都比较高，因此它的 F1 Score 也比较高。而其他模型的精准率和召回率要么都比较低，要么一个低一个高，所以它们的 F1 Score 比较低。

这也说明了只有当模型的精准率和召回率都比较高时 F1 Score 才会比较高。这也是 F1 Score 能够同时兼顾精准率和召回率的原因。

ROC曲线

ROC曲线(Receiver Operating Characteristic Curve)描述的 TPR（True Positive Rate）与 FPR（False Positive Rate）之间关系的曲线。

TPR 与 FPR 的计算公式如下：

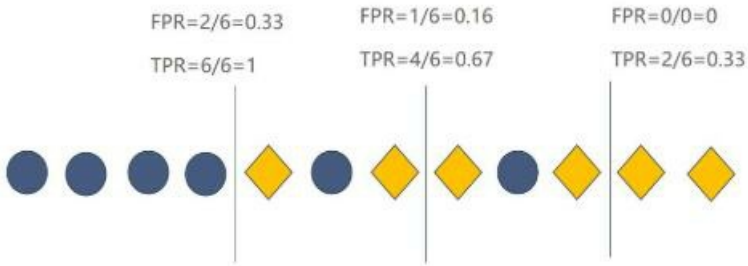
$TPR = \frac{TP}{TP+FN}$

$FPR = \frac{FP}{FP+TN}$

其中 TPR 的计算公式您可能有点眼熟，没错！就是召回率的计算公式。也就是说 **TPR** 就是召回率。所以 **TPR** 描述的是模型预测 **Positive** 并且预测正确的数量占真实类别为 **Positive** 样本的比例。而 **FPR** 描述的模型预测 **Positive** 并且预测错了的数量占真实类别为 **Negtive** 样本的比例。

和精准率与召回率一样， TPR 与 FPR 之间也存在关系。假设有这么一组数据，菱形代表 Positive ，圆形代表 Negtive 。

现在需要训练一个逻辑回归的模型对数据进行分类，假如将从 0 到 1 中的一些值作为模型的分类阈值。若模型认为当前数据是 Positive 的概率小于分类阈值则分类为 Negtive ，否则就分类为 Positive （假设分类阈值为 **0.8** ，模型认为这条数据是 **Positive** 的概率为 **0.7** ， **0.7** 小于 **0.8** ，那么模型就认为这条数据是 **Negtive**）。在不同的分类阈值下，模型所对应的 TPR 与 FPR 如下图所示（竖线代表分类阈值，模型会将竖线左边的数据分类成 Negtive ，竖线右边的分类成 Positive ）：

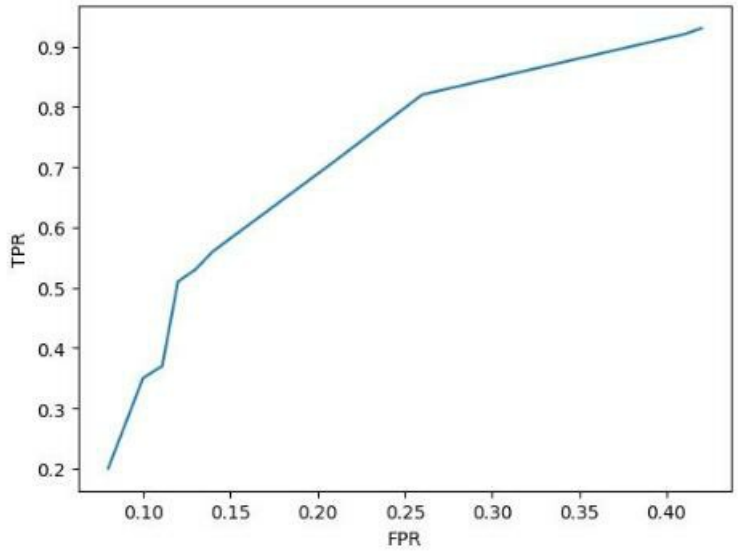


从图中可以看出，当模型的 **TPR** 越高 **FPR** 也会越高， **TPR** 越低 **FPR** 也会越低。这与精准率和召回率之间的关系刚好相反。并且，模型的分类阈值一但改变，就有一组对应的 TPR 与 FPR 。假设该模型在不同的分类阈值下其对应的 TPR 与 FPR 如下表所示：

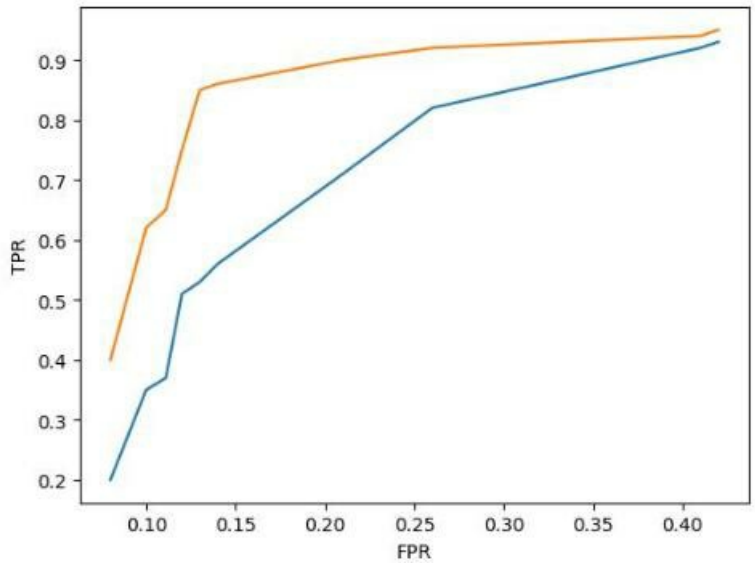
TPR	FPR
0.2	0.08
0.35	0.1
0.37	0.111
0.51	0.12

0.53	0.13
0.56	0.14
0.71	0.21
0.82	0.26
0.92	0.41
0.93	0.42

若将 FPR 作为横轴， TPR 作为纵轴，将上面的表格以折线图的形式画出来就是 ROC曲线 。



假设现在有模型 A 和模型 B ，它们的 ROC 曲线如下图所示(其中模型 A 的 ROC曲线 为黄色，模型 B 的 ROC曲线 为蓝色)：



那么模型 A 的性能比模型 B 的性能好，因为模型 A 当 FPR 较低时所对应的 TPR 比模型 B 的低 FPR 所对应的 TPR 更高。由由于随着 FPR 的增大， TPR 也会增大。所以 ROC 曲线与横轴所围成的面积越大，模型的分类性能就越高。而 ROC曲线 的面积称为 AUC。

AUC

很明显模型的 AUC 越高，模型的二分类性能就越强。 AUC 的计算公式如下：

$$AUC = \frac{\sum_{iepositiveclass} rank_i - \frac{M(M+1)}{2}}{M*N}$$

其中 M 为真实类别为 Positive 的样本数量， N 为真实类别为 Negtive 的样本数量。 ranki 代表了真实类别为 Positive 的样本点额预测概率从小到大排序后，该预测概率排在第几。

举个例子，现有预测概率与真实类别的表格如下所示（其中 0 表示 Negtive ， 1 表示 Positive ）：

编号	预测概率	真实类别
1	0.1	0
2	0.4	0
3	0.3	1
4	0.8	1

想要得到公式中的 rank ，就需要将预测概率从小到大排序，排序后如下：

编号	预测概率	真实类别
1	0.1	0
3	0.3	1
2	0.4	0
4	0.8	1

排序后的表格中，真实类别为 Positive 只有编号为 3 和编号为 4 的数据，并且编号为 3 的数据排在第 2 ，编号为 4 的数据排在第 4 。所以 rank=[2, 4]。又因表格中真是类别为 Positive 的数据有 2 条，Negtive 的数据有 2 条。因此 M 为2，N 为2。所以根据 AUC 的计算公式可知：

$AUC = \frac{(2+4)-\frac{2(2+1)}{2}}{2*2} = 0.75$。

回归模型性能评估指标

MSE

MSE（Mean Squared Error）叫做均方误差，其实就是线性回归的损失函数。公式如下：

$\frac{1}{m}\sum_{i=1}^{m}(y^i-p^i)^2$

其中y^i表示第 i 个样本的真实标签，p^i表示模型对第 i 个样本的预测标签。线性回归的目的就是让损失函数最小。那么模型训练出来了，我们在测试集上用损失函数来评估模型就行了。

RMSE

RMSE（Root Mean Squard Error）均方根误差，公式如下：

$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y^i-p^i)^2}$

RMSE 其实就是**MSE**开个根号。有什么意义呢？其实实质是一样的。只不过用于数据更好的描述。

例如：要做房价预测，每平方是万元，我们预测结果也是万元。那么差值的平方单位应该是千万级别的。那我们不太好描述自己做的模型效果。怎么说呢？我们的模型误差是多少千万？于是干脆就开个根号就好了。我们误差的结果就跟我们数据是一个级别的了，在描述模型的时候就说，我们模型的误差是多少万元。

MAE

MAE(Mean Aboslute Error)，公式如下：

$\frac{1}{m}\sum_{i=1}^{m}|y^i-p^i|$

MAE 虽然不作为损失函数，确是一个非常直观的评估指标，它表示每个样本的预测标签值与真实标签值的 L1 距离。

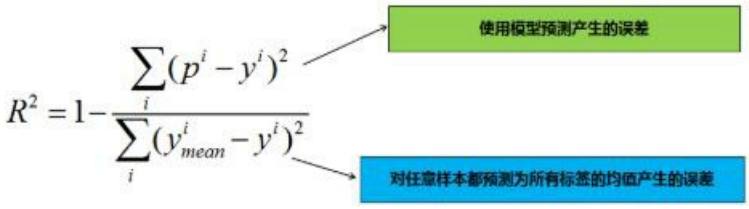
R-Squared

上面的几种衡量标准针对不同的模型会有不同的值。比如说预测房价 那么误差单位就是万元。数子可能是 3 ， 4 ， 5 之类的。那么预测身高就可能是 0.1 ，0.6 之类的。没有什么可读性，到底多少才算好呢？不知道，那要根据模型的应用场景来。 看看分类算法的衡量标准就是正确率，而正确率又在 0～1 之间，最高百分之百。最低 0 。如果是负数，则考虑非线性相关。很直观，而且不同模型一样的。那么线性回归有没有这样的衡量标准呢？

R-Squared 就是这么一个指标，公式如下：

$R^2=1-\frac{\sum_i(p^i-y^i)^2}{\sum_i(y^i_{mean}-y^i)^2}$

其中y_{mean}表示所有测试样本标签值的均值。为什么这个指标会有刚刚我们提到的性能呢？我们分析下公式：



其实分子表示的是模型预测时产生的误差，分母表示的是对任意样本都预测为所有标签均值时产生的误差，由此可知：

1. $R^2\leq 1$,当我们的模型不犯任何错误时，取最大值 1 。
2. 当我们的模型性能跟基模型性能相同时，取 0 。
3. 如果为负数，则说明我们训练出来的模型还不如基准模型，此时，很有可能我们的数据不存在任何线性关系。

聚类模型性能评估指标

聚类的性能度量大致分为两类：一类是将聚类结果与某个参考模型作为参照进行比较，也就是所谓的外部指标；另一类是则是直接度量聚类的性能而不使用参考模型进行比较，也就是内部指标。

外部指标

外部指标通常使用 **Jaccard Coefficient(JC系数)**、**Fowlkes and Mallows Index(FM指数)**以及 **Rand index**（**Rand**指数）。

想要计算上述指标来度量聚类的性能，首先需要计算出a，c，d，e。假设数据集$E=\{x_1,x_2,...,x_m\}$。通过聚类模型给出的簇划分为$C=\{C_1,C_2,...C_k\}$，参考模型给出的簇划分为$D=\{D_1,D_2,...D_s\}$。λ与λ^*分别表示C与D对应的簇标记，则有:

$a=|\{(x_i,x_j)|\lambda_i=\lambda_j,\lambda_i^*=\lambda_j^*,i<j\}|$

$b=|\{(x_i,x_j)|\lambda_i=\lambda_j,\lambda_i^*\neq\lambda_j^*,i<j\}|$

$c=|\{(x_i,x_j)|\lambda_i\neq\lambda_j,\lambda_i^*=\lambda_j^*,i<j\}|$

$d=|\{(x_i,x_j)|\lambda_i\neq\lambda_j,\lambda_i^*\neq\lambda_j^*,i<j\}|$

举个例子，参考模型给出的簇与聚类模型给出的簇划分如下：

编号	参考簇	聚类簇
1	0	0
2	0	0
3	0	1
4	1	1
5	1	2
6	1	2

那么满足a的样本对为$(1,2)$(因为1号样本与2号样本的参考簇都为0，聚类簇都为0)，$(5,6)$(因为5号样本与6号样本的参考簇都为1，聚类簇都为2)。总共有2个样本对满足a，因此$a=2$。

满足b的样本对为$(3,4)$(因为3号样本与4号样本的参考簇不同，但聚类簇都为1)。总共有1个样本对满足b，因此$b=1$。

那么满足c的样本对为$(1,3)$(因为1号样本与3号样本的聚类簇不同，但参考簇都为0)，$(2,3)$(因为2号样本与3号样本的聚类簇不同，但参考簇都为0)，$(4,5)$(因为4号样本与5号样本的聚类簇不同，但参考簇都为1)，$(4,6)$(因为4号样本与6号样本的聚类簇不同，但参考簇都为1)。总共有4个样本对满足c，因此$c=4$。

满足d的样本对为$(1,4)$(因为1号样本与4号样本的参考簇不同，聚类簇也不同)，$(1,5)$(因为1号样本与5号样本的参考簇不同，聚类簇也不同)，$(1,6)$(因为1号样本与6号样本的参考簇不同，聚类簇也不同)，$(2,4)$(因为2号样本与4号样本的参考簇不同，聚类簇也不同)，$(2,5)$(因为2号样本与5号样本的参考簇不同，聚类簇也不同)，$(2,6)$(因为2号样本与6号样本的参考簇不同，聚类簇也不同)，$(3,5)$(因为3号样本与5号样本的参考簇不同，聚类簇也不同)，$(3,6)$(因为3号样本与6号样本的参考簇不同，聚类簇也不同)。总共有8个样本对满足d，因此$d=8$。

JC系数

JC系数根据上面所提到的a，b，c来计算，并且值域为$[0,1]$，值越大说明聚类性能越好，公式如下：

$JC=\frac{a}{a+b+c}$

因此刚刚的例子中，$JC=\frac{2}{2+1+4}=\frac{2}{7}$

FM指数

FM指数根据上面所提到的a，b，c来计算，并且值域为$[0,1]$，值越大说明聚类性能越好，公式如下：

$FMI=\sqrt{\frac{a}{a+b}*\frac{a}{a+c}}$

因此刚刚的例子中，$FMI=\sqrt{\frac{2}{2+1}*\frac{2}{2+4}}=\sqrt{\frac{4}{18}}$

Rand指数

Rand指数根据上面所提到的a和d来计算，并且值域为$[0,1]$，值越大说明聚类性能越好，假设m为样本数量，公式如下：

$RandI = \frac{2(a+d)}{m(m-1)}$

因此刚刚的例子中，$RandI = \frac{2*(2+8)}{6*(6-1)} = \frac{2}{3}$。

内部指标

内部指标通常使用 **Davies-Bouldin Index (DB**指数**)**以及 **Dunn Index**（**Dunn**指数）。

DB指数

DB指数又称 DBI ，计算公式如下：

$DBI = \frac{1}{k}\sum_{i=1}^{k} max(\frac{avg(C_i)+avg(C_j)}{d_c(\mu_i,\mu_j)}), i \neq j$

公式中的表达式其实很好理解，其中k代表聚类有多少个簇，μ_i代表第i个簇的中心点，$avg(C_i)$代表C_i第i个簇中所有数据与第i个簇的中心点的平均距离。$d_c(\mu_i,\mu_j)$代表第i个簇的中心点与第j个簇的中心点的距离。

举个例子，现在有6条西瓜数据$\{x_1, x_2, ..., x_6\}$，这些数据已经聚类成了2个簇。

编号	体积	重量	簇
1	3	4	1
2	6	9	2
3	2	3	1
4	3	4	1
5	7	10	2
6	8	11	2

从表格可以看出：

$k = 2$

$\mu_1 = (\frac{(3+2+3)}{3}, \frac{(4+3+4)}{3}) = (2.67, 3.67)$

$\mu_2 = (\frac{(6+7+8)}{3}, \frac{(9+10+11)}{3}) = (7, 10)$

$d_c(\mu_1,\mu_2) = \sqrt{(2.67-7)^2 + (3.67-10)^2} = 7.67391$

$avg(C_1) = (\sqrt{(3-2.67)^2 + (4-3.67)^2} + \sqrt{(2-2.67)^2 + (3-3.67)^2} + \sqrt{(3-2.67)^2 + (4-3.67)^2})/3 = 0.628539$

$avg(C_2) = (\sqrt{(6-7)^2 + (9-10)^2} + \sqrt{(7-7)^2 + (10-10)^2} + \sqrt{(8-7)^2 + (11-10)^2})/3 = 0.94281$

因此有：

$DBI = \frac{1}{k}\sum_{i=1}^{k} max(\frac{avg(C_i)+avg(C_j)}{d_c(\mu i,\mu j)}) = 0.204765$

DB指数越小就越就意味着簇内距离越小同时簇间距离越大，也就是说**DB**指数越小越好。

Dunn指数

Dunn指数又称DI，计算公式如下：

$DI = min_{1 \leq i \leq k}\{min_{i \neq j}(\frac{d_min(C_i,C_j)}{max_{1 \leq l \leq k} diam(C_l)})\}$

公式中的表达式其实很好理解，其中k代表聚类有多少个簇，$d_{min}(C_i,C_j)$代表第i个簇中的样本与第j个簇中的样本之间的最短距离，$diam(C_l)$代表第l个簇中相距最远的样本之间的距离。

还是这个例子，现在有 6 条西瓜数据$\{x_1, x_2, ..., x_6\}$，这些数据已经聚类成了 2 个簇。

编号	体积	重量	簇
1	3	4	1
2	6	9	2
3	2	3	1
4	3	4	1
5	7	10	2
6	8	11	2

从表格可以看出：

$k=2$

$d_{min}(C_1,C_2)=\sqrt{(3-6)^2+(4-9)^2}=5.831$

$diam(C_1)=\sqrt{(3-2)^2+(4-2)^2}=1.414$

$diam(C_2)=\sqrt{(6-8)^2+(9-11)^2}=2.828$

因此有：

$DI=min_{1\le i\le k}\{min_{i\ne j}(\frac{d_min(C_i,C_j)}{max_{1\le l\le k}diam(C_l)})\}=2.061553$

Dunn指数越大意味着簇内距离越小同时簇间距离越大，也就是说**Dunn**指数越大越好。

使用**sklearn**进行机器学习

写在前面

这是一个 sklearn 的 hello world 级教程，想要更加系统更加全面的学习 sklearn 建议查阅 sklearn 的官方网站。

sklearn简介

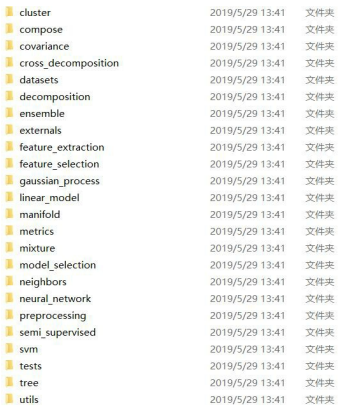
scikit-learn(简记sklearn)，是用 python 实现的机器学习算法库。sklearn 可以实现数据预处理、分类、回归、降维、模型选择等常用的机器学习算法。基本上只需要知道一些 python 的基础语法知识就能学会怎样使用 sklearn 了，所以 sklearn 是一款非常好用的 python 机器学习库。

sklearn的安装

和安装其他第三方库一样简单，只需要在命令行中输入 `pip install scikit-learn` 即可。

sklearn的目录结构

sklearn 提供的接口都封装在不同的目录下的不同的 py 文件中，所以对 sklearn 的目录结构有一个大致的了解，有助于我们更加深刻地理解 sklearn 。目录结构如下：

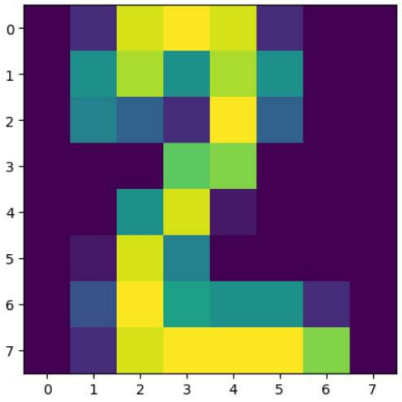


其实从目录名字可以看出目录中的 py 文件是干啥的。比如 cluster 目录下都是聚类算法接口， ensem 目录下都是集成学习算法的接口。

使用**sklearn**识别手写数字

接下来不如通过一个实例来感受一下 sklearn 的强大。

想要识别手写数字，首先需要有数据。sklearn 中已经为我们准备好了一些比较经典且质量较高的数据集，其中就包括手写数字数据集。该数据集有 1797 个样本，每个样本包括 8*8 像素（实际上是一条样本有 64 个特征，每个像素看成是一个特征，每个特征都是 float 类型的数值）的图像和一个 [0, 9] 整数的标签。比如下图的标签是 2 ：



想要使用这个数据很简单，代码如下：

```
from sklearn import datasets

# 加载手写数字数据集
digits = datasets.load_digits()

# X表示特征，即1797行64列的矩阵
X = digits.data
# Y表示标签，即1797个元素的一维数组
y = digits.target
```

得到 X，y 数据之后，我们还需要将这些数据进行划分，划分成两个部分，一部分是训练集，另一部分是测试集。因为如果没有测试集的话，我们并不知道我们的手写数字识别程序识别得准不准。数据集划分代码如下：

```
# 将X，y划分成训练集和测试集，其中训练集的比例为80%，测试集的比例为20%
# X_train表示训练集的特征，X_test表示测试集的特征，y_train表示训练集的标签，y_test表示测试集的标签
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

接下来，可以使用机器学习算法来实现手写数字识别了，例如想要使用随机森林来进行识别，那么首先要导入随机森林算法接口。

```
# 由于是分类问题，所以导入的是RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
```

导入好接口后，就可以创建随机森林对象了。随机森林对象有用来训练的函数 `fit` 和用来预测的函数 `predict`。`fit` 函数需要训练集的特征和训练集的标签作为输入，`predict` 函数需要测试集的特征作为输入。所以代码如下：

```
# 创建一个有50棵决策树的随机森林，n_estimators表示决策树的数量
clf = RandomForestClassifier(n_estimators=50)
# 用训练集训练
clf.fit(X_train, Y_train)
# 用测试集测试，result为预测结果
result = clf.predict(X_test)
```

得到预测结果后，我们需要将其与测试集的真实答案进行比对，计算出预测的准确率。sklearn 已经为我们提供了计算准确率的接口，使用代码如下：

```
# 导入计算准确率的接口
from sklearn.metrics import accuracy_score

# 计算预测准确率
acc = accuracy_score(y_test, result)
# 打印准确率
print(acc)
```

此时您会发现我们短短的几行代码实现的手写数字识别程序的准确率高于 **0.95**。

而且我们不仅可以使用随机森林来实现手写数字识别，我们还可以使用别的机器学习算法实现，比如逻辑回归，代码如下:

```
from sklearn.linear_model import LogisticRegression

# 创建一个逻辑回归对象
clf = LogisticRegression()
# 用训练集训练
clf.fit(X_train, Y_train)
# 用测试集测试，result为预测结果
result = clf.predict(X_test)
```

细心的您可能已经发现，不管使用哪种分类算法来进行手写数字识别，不同的只是创建的算法对象不一样而已。有了算法对象后，就可以 `fit`，`predict` 大法了。

下面是使用随机森林识别手写数字的完整代码：

```
from sklearn import datasets
# 由于是分类问题，所以导入的是RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
# 导入计算准确率的接口
from sklearn.metrics import accuracy_score

# 加载手写数字数据集
digits = datasets.load_digits()

# X表示特征，即1797行64列的矩阵
```

```
X = digits.data
# Y表示标签，即1797个元素的一维数组
y = digits.target

# 将X，y划分成训练集和测试集，其中训练集的比例为80%，测试集的比例为20%
# X_train表示训练集的特征，X_test表示测试集的特征，y_train表示训练集的标签，y_test表示测试集的标签
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# 创建一个有50棵决策树的随机森林，n_estimators表示决策树的数量
clf = RandomForestClassifier(n_estimators=50)
# 用训练集训练
clf.fit(X_train, Y_train)
# 用测试集测试，result为预测结果
result = clf.predict(X_test)

# 计算预测准确率
acc = accuracy_score(y_test, result)
# 打印准确率
print(acc)
```

更好地验证算法性能

在划分训练集与测试集时会有这样的情况，可能模型对于数字 1 的识别准确率比较低 ，而测试集中没多少个数字为 1 的样本，然后用测试集测试完后得到的准确率为 0.96 。然后您可能觉得哎呀，我的模型很厉害了，但其实并不然，因为这样的测试集让您的模型的性能有了误解。那有没有更加公正的验证算法性能的方法呢？有，那就是**k-**折验证！

k-折验证的大体思路是将整个数据集分成 k 份，然后试图让每一份子集都能成为测试集，并循环 k 次，总后计算 k 次模型的性能的平均值作为性能的估计。一般来说 k 的值为 5 或者 10。

k-折验证的流程如下：

1. 不重复抽样将整个数据集随机拆分成 k 份
2. 每一次挑选其中 1 份作为测试集，剩下的 k-1 份作为训练集 2.1. 在每个训练集上训练后得到一个模型 2.2. 用这个模型在相应的测试集上测试，计算并保存模型的评估指标
3. 重复第 2 步 k 次，这样每份都有一次机会作为测试集，其他机会作为训练集
4. 计算 k 组测试结果的平均值作为算法性能的估计。

sklearn 为我们提供了将数据划分成 k 份类 KFold ，使用示例如下：

```
# 导入KFold
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# 创建一个将数据集随机划分成5份
kf = KFold(n_splits = 5)

mean_acc = 0

# 将整个数据集划分成5份
# train_index表示从5份中挑出来4份所拼出来的训练集的索引
# test_index表示剩下的一份作为测试集的索引
for train_index, test_index in kf.split(X):
    X_train, y_train = X[train_index], y[train_index]
    X_test, y_test = X[test_index], y[test_index]
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)
    result = rf.predict(X_test)
    mean_acc = accuracy_score(y_test, result)

# 打印5折验证的平均准确率
print(mean_acc/5)
```

完整代码如下：

```
from sklearn import datasets
# 由于是分类问题，所以导入的是RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
# 导入计算准确率的接口
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

# 加载手写数字数据集
digits = datasets.load_digits()
```

```
# X表示特征，即1797行64列的矩阵
X = digits.data
# Y表示标签，即1797个元素的一维数组
y = digits.target

# 创建一个将数据集随机划分成5份
kf = KFold(n_splits = 5)

mean_acc = 0

# 将整个数据集划分成5份
# train_index表示从5份中挑出来4份所拼出来的训练集的索引
# test_index表示剩下的一份作为测试集的索引
for train_index, test_index in kf.split(X):
    X_train, y_train = X[train_index], y[train_index]
    X_test, y_test = X[test_index], y[test_index]
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)
    result = rf.predict(X_test)
    mean_acc = accuracy_score(y_test, result)

# 打印5折验证的平均准确率
print(mean_acc/5)
```

综合实战案例

纸上得来终觉浅，绝知此事要躬行。当我们学会了机器学习算法原理，知道了怎样使用 `sklearn` 来快速编写机器学习程序之后，我们需要进行大量的实战演练来提升我们的技能和实操能力。所以在本章中提供了两个综合实战案例：一个是 `kaggle` 中最为经典的也是最适合入门的机器学习比赛---泰坦尼克生还预测。另一个是使用强化学习来让程序自己玩乒乓球游戏。希望通过这两个实战案例，能够提升您对机器学习的兴趣和实战技能。

泰坦尼克生还预测

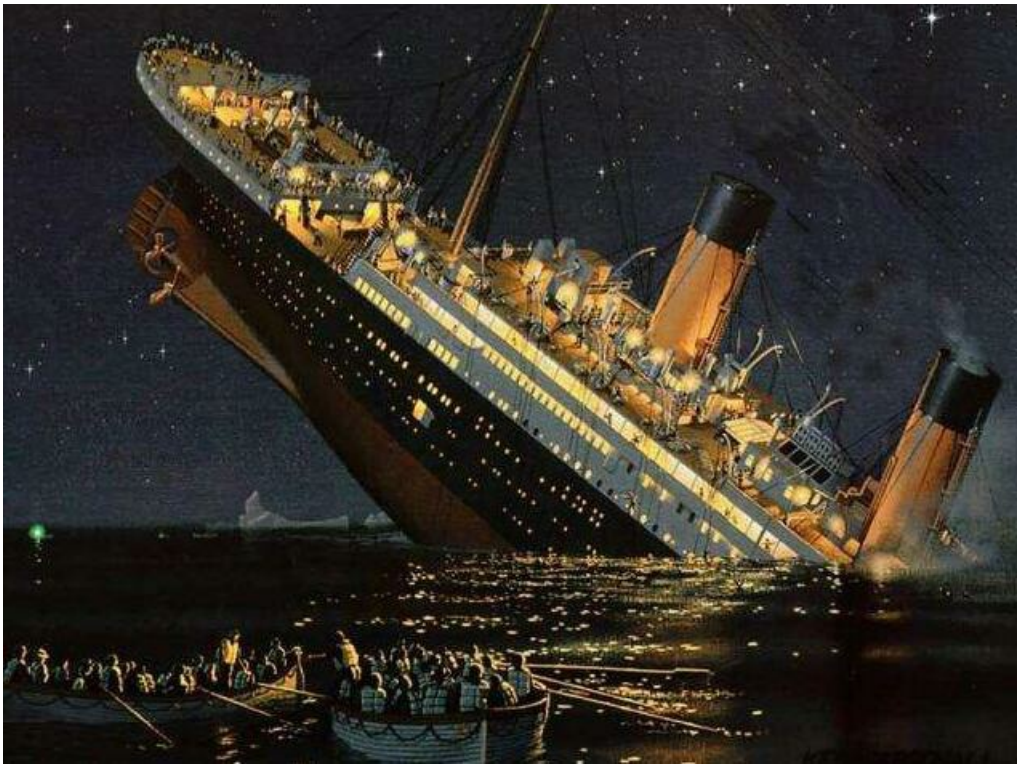
写在前面的话

怎样处理数据，使用什么样的机器学习模型并没有所谓的正确答案。这篇文章只是抛砖引玉，若您是刚刚接触数据科学，我相信这一篇不错的指引；若您已经是老手，我相信文中的一些技巧您肯定也用过，可以温故而知新；所以希望这篇文章对您或多或少的有所帮助。

泰坦尼克生还问题简介

泰坦尼克号的沉船事件是是历史上最臭名昭著的沉船事件之一。`1912`年`4`月`15`日，泰坦尼克在航线中与冰山相撞，`2224`名乘客中有`1502`名乘客丧生。

泰坦尼克号数据集是目标是给出一个模型来预测某位泰坦尼克号的乘客在沉船事件中是生还是死。而且该数据集是一个非常好的数据集，能够让您快速的开始数据科学之旅。



探索性数据分析(EDA)

探索性数据分析(EDA)说白了就是通过可视化的方式来看看数据中特征与特征之间，特征与目标之间的潜在关系，看看有什么有用的线索可以挖掘，例如哪些数据是噪声，有哪些特征的相关性比较低，后续可以造出哪些新的特征等。

初窥

当然，在EDA之前先要加载数据，我们不妨先将训练集train.csv读到内存中，并看一看。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data=pd.read_csv('./Titanic/train.csv')

# 看看data的前5行
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

从图中可以看出数据是由 11 个特征和 1 个标签(Survived)组成的。其中各个特征和标签的意义如下:

特征	意义
Survived	是否生还，1表示是，0表示否
PassengerId	乘客ID
Pclass	船票类型， 总共3种类型：1(一等舱)，2(二等舱)，3(三等舱)
Name	船客姓名
Sex	船客性别：female，male
Age	船客年龄
SibSp	船客的兄弟姐妹妻子丈夫的数量
Parch	船客的父母，孩子的数量
Ticket	船票
Fare	船客在船上所花的钱
Cabin	船客的船舱号
Embarked	船客登船的口岸：C，Q，S

了解了数据种各个属性的含义之后，我们可以看看这个数据集中有没有缺失值。

```
data.isnull().sum()
```

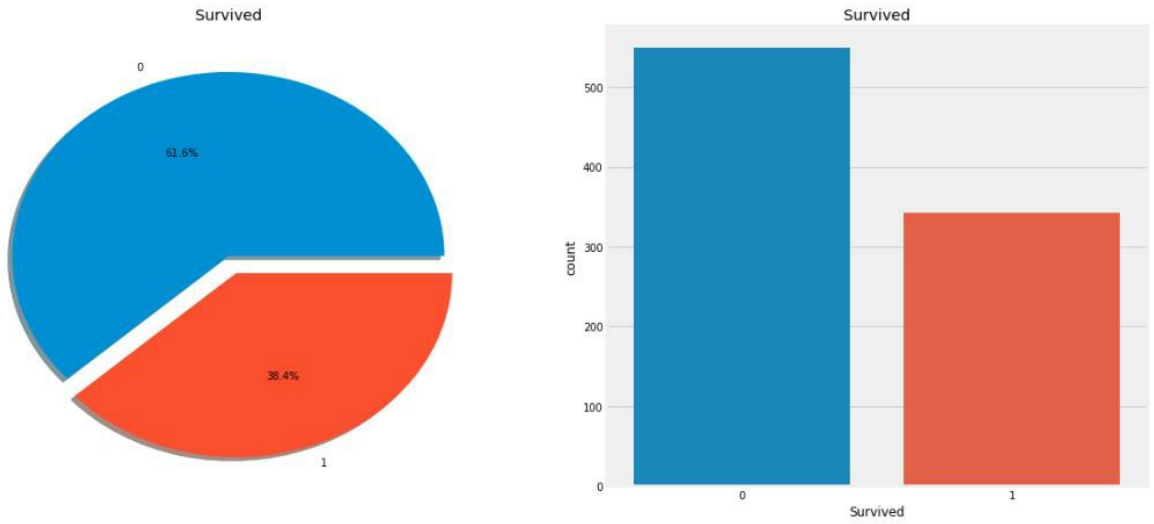
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

可以看出 Age，Cabin 和 Embarked 这三个特征中有缺失值，我们需要处理这些缺失值。怎样处理呢？先不着急，我们可以先看看数据中有哪些信息可以挖掘。

有多少人活了下来

我们首先可以看看训练集中有多少人活了下来。

```
f,ax=plt.subplots(1,2,figsize=(18,8))
# 生还比例饼图
data['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Survived')
ax[0].set_ylabel('')
# 生还数量直方图
sns.countplot('Survived',data=data,ax=ax[1])
ax[1].set_title('Survived')
plt.show()
```



从图中可以看出泰坦尼克沉船事件中还是凶多吉少的。因为在 891 名船客中，只有约 38% 左右的人幸免于难，那么接下来尝试使用数据集中不同的特征，来看看他们的生还率有多少。其实这样一个过程我们可以看出大概有哪些类型的船客活了下来。

性别与生还率的关系

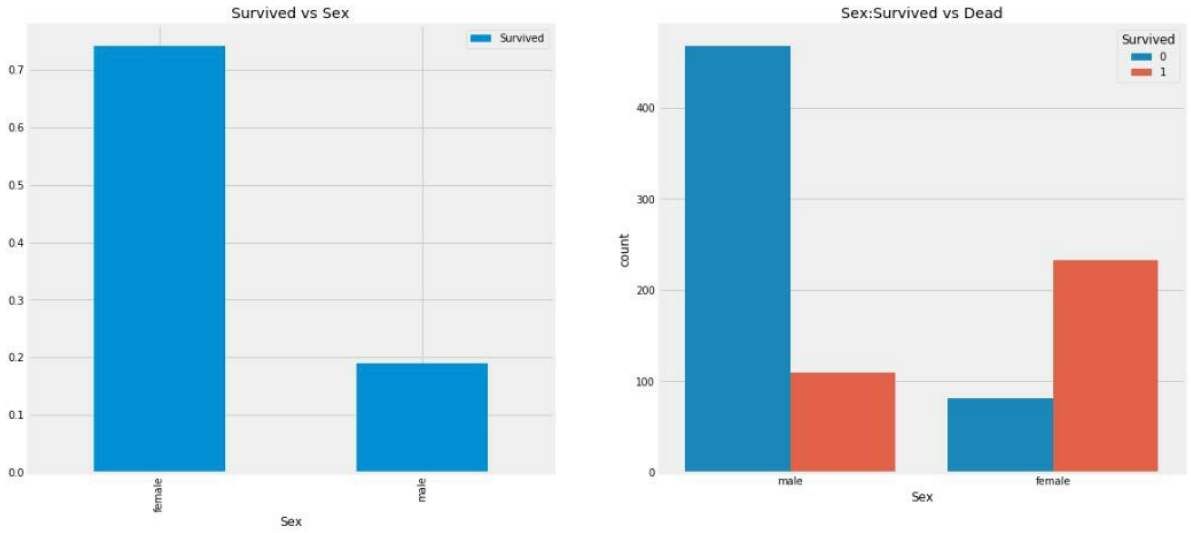
首先，看看不同性别的生还者数量。

```
data.groupby(['Sex','Survived'])['Survived'].count()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

看上去好想女性船客的生还率高一些，我们不妨再可视化一下。

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data[['Sex','Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survived vs Sex')
sns.countplot('Sex',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Sex:Survived vs Dead')
plt.show()
```

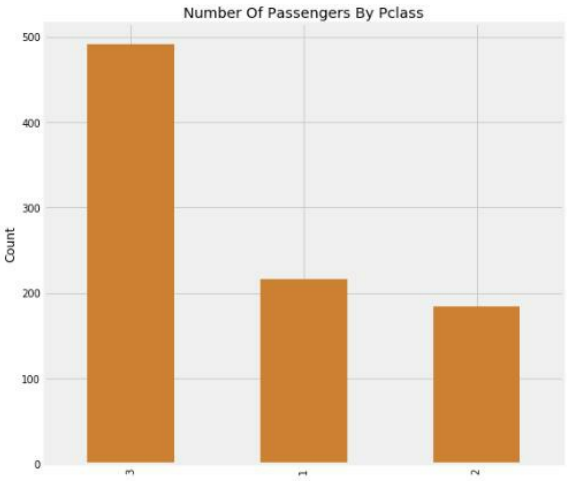


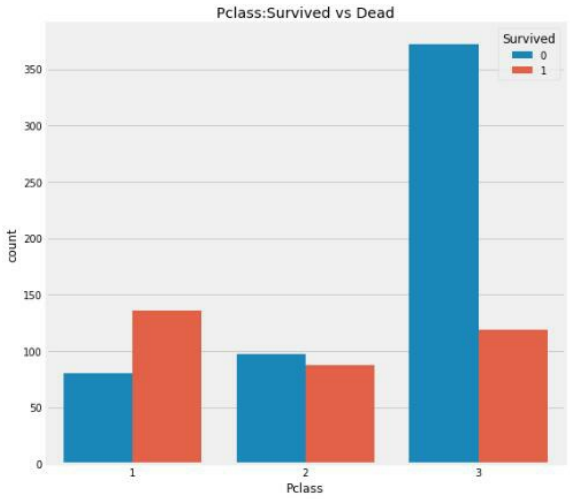
从图中可以看出一个比较有趣的现象，船上的男人是比女人多了 200 多人，但是女人生还的人数几乎是男人生还的人数的两倍，女人的存活率约为 75% ，而男人的存活率约为 19% 的样子。所以 Sex 这个特征应该是一个能够很好的区分一个人是否生还的特征。而且对于生还来说，好像是女士优先。

船票类型与生还率的关系

船票类型分三个档次，其中 1 为一等舱， 2 为二等舱， 3 为三等舱。既然船舱分三六九等，那么是不是越高级的舱，它的生还率越高呢？

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Pclass'].value_counts().plot.bar(ax=ax[0])
ax[0].set_title('Number Of Passengers By Pclass')
ax[0].set_ylabel('Count')
sns.countplot('Pclass',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Pclass:Survived vs Dead')
plt.show()
```



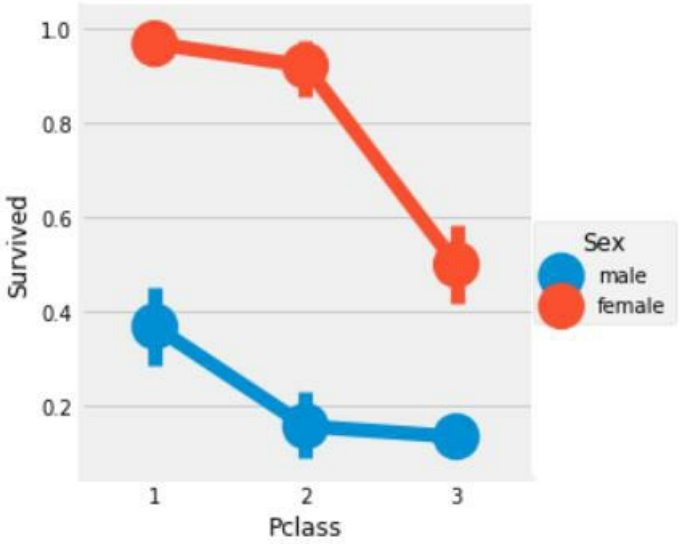


虽然说钱不是万能的，但从可视化结果可以看出，一等舱的生还率最高，大于为 63%，二等舱的生还率约为 48% ，而且虽然三等舱的船客人数是最多的，但生还率确是最低的。所以不难看出，金钱地位还是很重要的，也许一等舱周围有比较多的救生设备。

上流女性与生还率的关系

从前两次可视化结果可以看出，女性，上流人士成为了是否能够活下来的关键，那么上流女性(两者的结合)的生还率会不会很高呢？

```
sns.factorplot('Pclass','Survived',hue='Sex',data=data)
plt.show()
```



从这张图可以看出一等舱的女性(上流女性)的生还率非常高！几乎接近了百分之百！而且二等舱和三等舱的女性的生还率也远比男性的生还率高。这也验证了我们的猜测，在沉船后是优先女性和一等舱的船客的。

年龄与生还率的关系

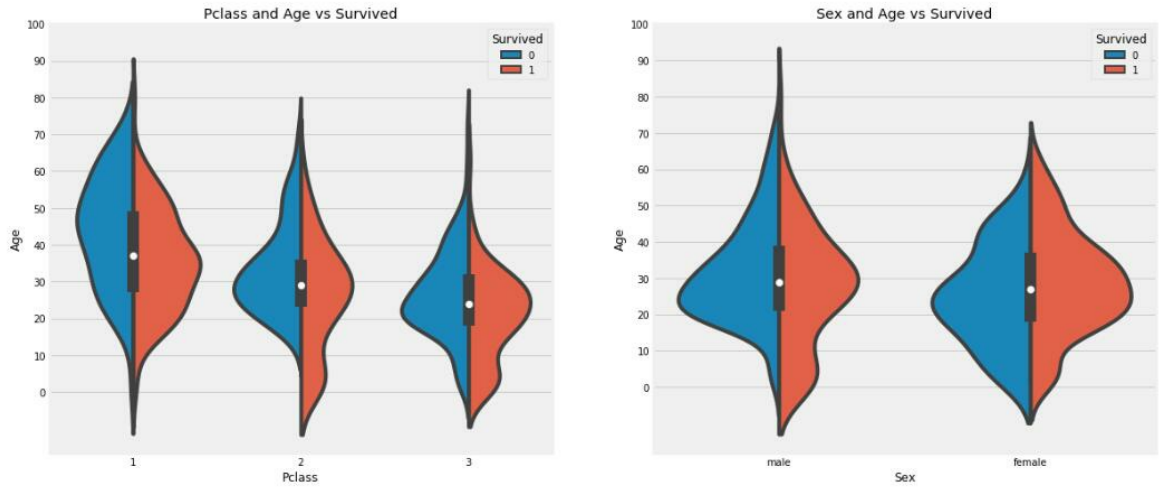
首先可以先看一下训练集中船客的年龄的最值和均值。

```
print('Oldest Passenger was of:',data['Age'].max(),'Years')
print('Youngest Passenger was of:',data['Age'].min(),'Years')
print('Average Age on the ship:',data['Age'].mean(),'Years')
```

```
Oldest Passenger was of: 80.0 Years
Youngest Passenger was of: 0.42 Years
Average Age on the ship: 29.69911764705882 Years
```

年纪最大的是80岁的老爷爷或者老太太，最小的是刚出生的小 baby， 平均年龄快 30 岁。这个还是符合常理的。接下来我们看看船舱等级，年龄和生还率的关系，以及性别，年龄和生还率的关系。

```
f,ax=plt.subplots(1,2,figsize=(18,8))
sns.violinplot("Pclass","Age", hue="Survived", data=data,split=True,ax=ax[0])
ax[0].set_title('Pclass and Age vs Survived')
ax[0].set_yticks(range(0,110,10))
sns.violinplot("Sex","Age", hue="Survived", data=data,split=True,ax=ax[1])
ax[1].set_title('Sex and Age vs Survived')
ax[1].set_yticks(range(0,110,10))
plt.show()
```



从可视化结果可以看出：

- 儿童的数量随着船舱等级的增加而增加，10 岁以下的小朋友存活率仿佛都还挺高的，跟船舱等级好像没有太大关系。
- 来自一等舱的 20-50 岁的船客的存活率很高，而且对女性的生还率一如既往的高。
- 对于男性来说，年纪越大，生还率越低。

不过我们的年龄是有缺失值的，如果图简单，可以使用平均年龄来填充缺失的年龄。但是这样做并不合适，比如人家只是个 5 岁的小屁孩，但是你把人家强行改成 29 岁显然是不合适的。那有没有能够更加准确地知道缺失的年龄是多少的方法呢？有！我们可以根据姓名来推断缺失的年龄，因为姓名中有很多类似 Mr 或者 Mrs 这样的前缀，所以我们可以根据姓名的前缀来填充缺失的年龄。

填充缺失年龄

外国人的姓名和我们中国人的姓名不太一样，一般都会有 Mr 、 Mrs 、Miss 、Dr 等特殊前缀。所以我们可以先提取姓名中的前缀。

```
data['Initial']=0
for _ in data:
    data['Initial']=data.Name.str.extract('([A-Za-z]+)\.')
```

这样我们能够提取出诸如：Capt 、Col 、Don 、Lady 、Major 、Sir 等前缀，接着我们可以将这些前缀替换成 Miss 、 Mr 、 Mrs 、 Other 这四个类别，并统计这四个类别的平均年龄。

```
data['Initial'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt','Sir','Don'],['Miss','Miss','Miss','Mr','Mr'
,'Mrs','Mrs','Other','Other','Other','Mr','Mr','Mr'],inplace=True)

data.groupby('Initial')['Age'].mean()
```

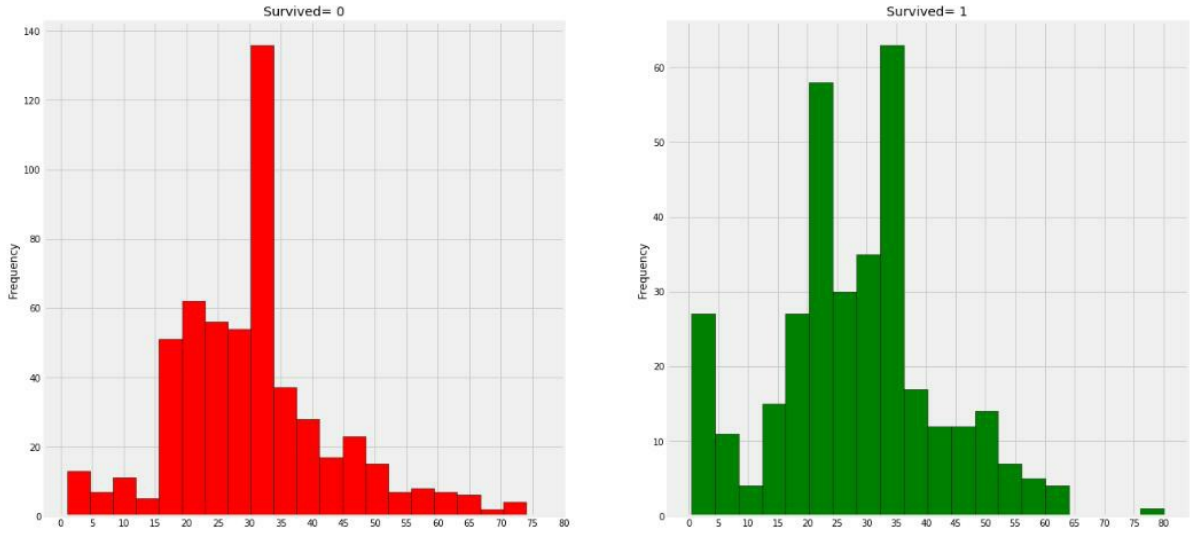
```
Initial
Master     4.574167
Miss      21.860000
Mr        32.739609
Mrs       35.981818
Other     45.888889
Name: Age, dtype: float64
```

接着可以根据前缀来填充缺失的年龄。

```
data.loc[(data.Age.isnull())&(data.Initial=='Mr'),'Age']=33
data.loc[(data.Age.isnull())&(data.Initial=='Mrs'),'Age']=36
data.loc[(data.Age.isnull())&(data.Initial=='Miss'),'Age']=22
data.loc[(data.Age.isnull())&(data.Initial=='Other'),'Age']=46
```

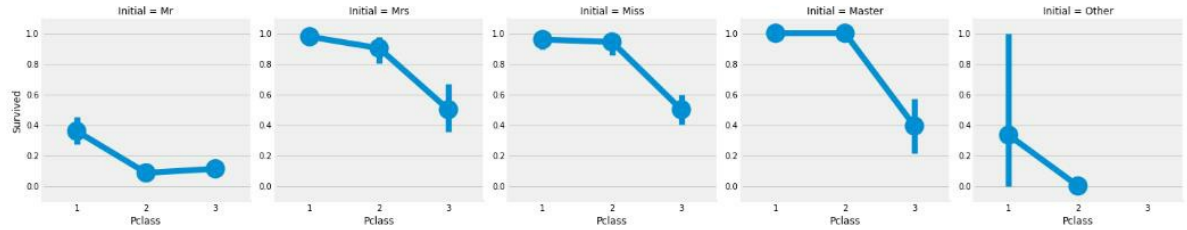
填充完缺失值后，可以尝试可视化一下。

```
f,ax=plt.subplots(1,2,figsize=(20,10))
data[data['Survived']==0].Age.plot.hist(ax=ax[0],bins=20,edgecolor='black',color='red')
ax[0].set_title('Survived= 0')
x1=list(range(0,85,5))
ax[0].set_xticks(x1)
data[data['Survived']==1].Age.plot.hist(ax=ax[1],color='green',bins=20,edgecolor='black')
ax[1].set_title('Survived= 1')
x2=list(range(0,85,5))
ax[1].set_xticks(x2)
plt.show()
```



从图中可以看出 5 岁以下的小屁孩的生还率比较高，80 岁的老人活下来了。

```
sns.factorplot('Pclass','Survived',col='Initial',data=data)
plt.show()
```

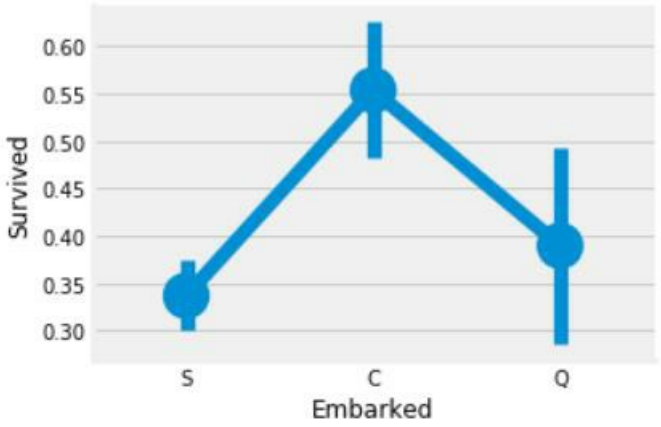


嗯，女性和小孩的生还率比较高。

登船口岸与生还率的关系

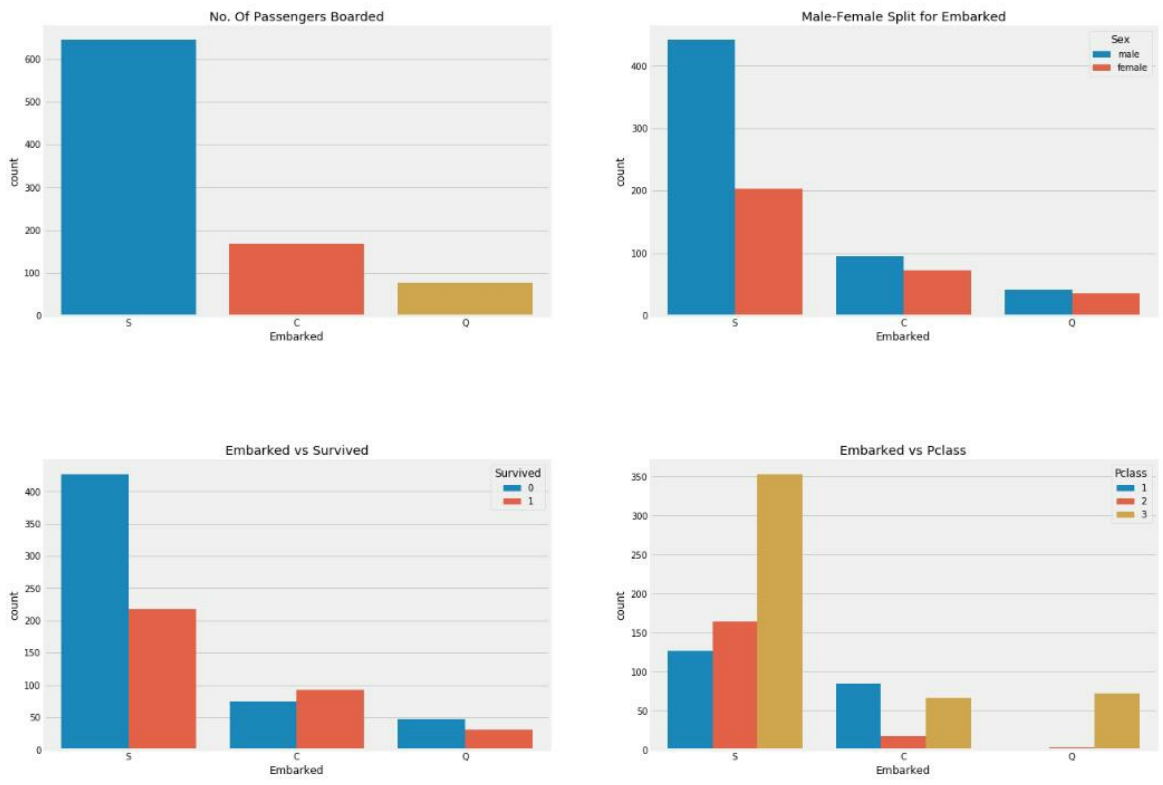
先把口岸和生还率的关系画出来。

```
sns.factorplot('Embarked','Survived',data=data)
fig=plt.gcf()
fig.set_size_inches(5,3)
plt.show()
```



可以看出从 C 号口岸上船的生还率最高，最低的是 S 号口岸。嗯，好像并没有什么线索，我们可以再深入一点。

```
f,ax=plt.subplots(2,2,figsize=(20,15))
sns.countplot('Embarked',data=data,ax=ax[0,0])
ax[0,0].set_title('No. Of Passengers Boarded')
sns.countplot('Embarked',hue='Sex',data=data,ax=ax[0,1])
ax[0,1].set_title('Male-Female Split for Embarked')
sns.countplot('Embarked',hue='Survived',data=data,ax=ax[1,0])
ax[1,0].set_title('Embarked vs Survived')
sns.countplot('Embarked',hue='Pclass',data=data,ax=ax[1,1])
ax[1,1].set_title('Embarked vs Pclass')
plt.subplots_adjust(wspace=0.2,hspace=0.5)
plt.show()
```

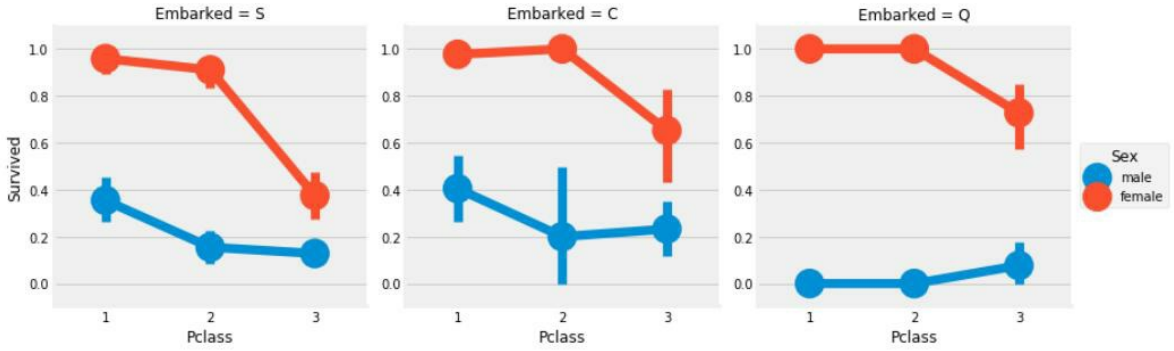


现在能看出很多信息了：

- 上船人数最多的口岸是 S 号口岸，而且在 S 号口岸上船的人大多数都是三等舱的船客。

- C 号口岸上船的生还率最高，可能大部分 C 口岸上船的人是一等舱和二等舱船客吧。
- 虽然有很多一等舱的土豪们基本上都是在 S 口岸上船的，但是 S 口岸的的生还率最低。这是因为 S 口岸上船的人中有很多都是三等舱的船客。
- Q 号口岸上船的人中有 90% 多都是三等舱的船客。

```
sns.factorplot('Pclass','Survived',hue='Sex',col='Embarked',data=data)
plt.show()
```



我们可以看出：

- 一等舱和二等舱的女性的生还率几乎为 100%， 这与女性是一等舱还是二等舱没啥关系。
- S 号口岸上船并且是三等舱的，不管是男的还是女的，生还率都很低。金钱决定命运。。。
- Q 号口岸上船的男性几乎团灭，因为Q 号口岸上船的基本上都是三等舱船客。

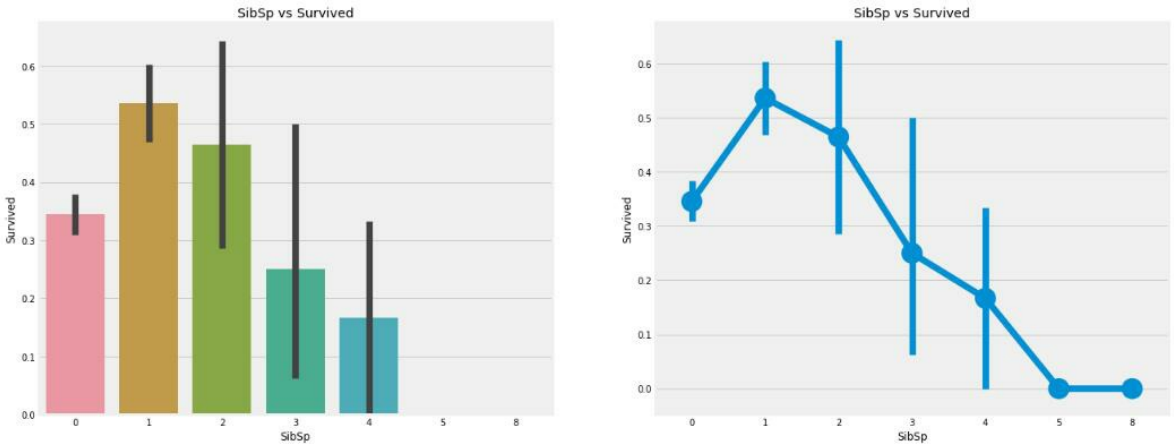
填充缺失口岸

由于大多数人都是从 S 号口岸上的船，我们可以假设由于人多，所以在 S 口岸登记信息时漏了几位船客，所以不妨用 S 号口岸填充缺失值。

```
data['Embarked'].fillna('S',inplace=True)
```

兄弟姐妹的数量与生还率的关系

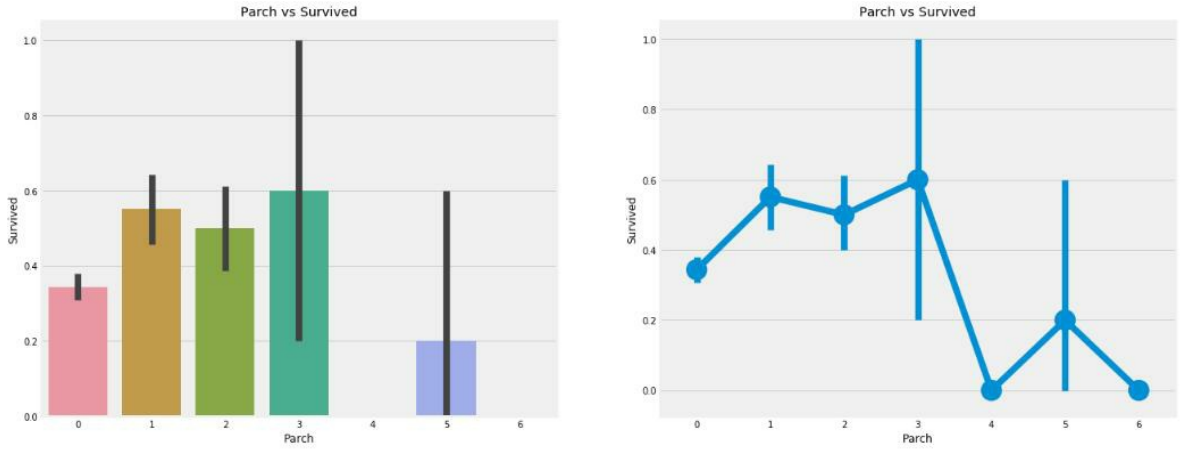
```
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot('SibSp','Survived',data=data,ax=ax[0])
ax[0].set_title('SibSp vs Survived')
sns.factorplot('SibSp','Survived',data=data,ax=ax[1])
ax[1].set_title('SibSp vs Survived')
plt.close(2)
plt.show()
```



从图可以看出，如果一位船客是单独一个人上船旅游，没有兄弟姐妹而且是单身，那么他有大约 34% 的生还率，生还率比较低。如果兄弟姐妹的数量变多，那么生还率还是呈下降趋势的。这其实挺合理的，因为如果是一个家庭在船上的话，可能会设法救他们而不是救自己，这样一来可能谁都救不了。

父母的数量与生还率的关系

```
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot('Parch','Survived',data=data,ax=ax[0])
ax[0].set_title('Parch vs Survived')
sns.factorplot('Parch','Survived',data=data,ax=ax[1])
ax[1].set_title('Parch vs Survived')
plt.close(2)
plt.show()
```



从图上看会发现结果和上面的比较相似，父母在船上的船客有更大的生还机会。而且对于那些在船上有 1-3 个父母的人来说，生还率还是比较高的。

花费与生还率的关系

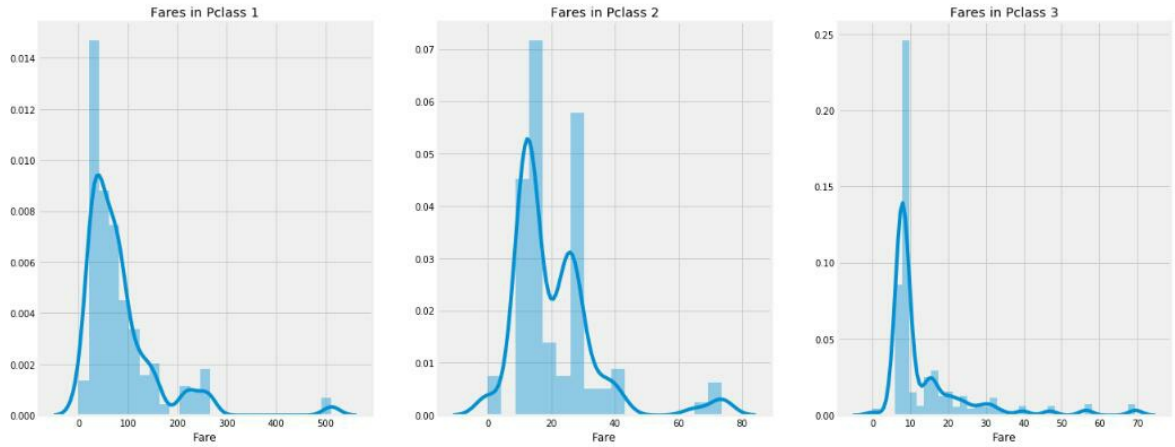
首先，先看一下花费的最值和均值。

```
print('Highest Fare was:',data['Fare'].max())
print('Lowest Fare was:',data['Fare'].min())
print('Average Fare was:',data['Fare'].mean())
```

```
Highest Fare was: 512.3292
Lowest Fare was: 0.0
Average Fare was: 32.2042079685746
```

惊奇的发现，居然有人可以享受免费豪华邮轮！！！！

```
f,ax=plt.subplots(1,3,figsize=(20,8))
sns.distplot(data[data['Pclass']==1].Fare,ax=ax[0])
ax[0].set_title('Fares in Pclass 1')
sns.distplot(data[data['Pclass']==2].Fare,ax=ax[1])
ax[1].set_title('Fares in Pclass 2')
sns.distplot(data[data['Pclass']==3].Fare,ax=ax[2])
ax[2].set_title('Fares in Pclass 3')
plt.show()
```



从图中可以看出平均花费其实是二等舱的普遍消费水平，但是三等舱的人数是最多的，而三等舱的人群中花费人数最多的是 10 左右，因此平均 32 的花费是被有钱的大佬给提上去的。

简单总结一下

看了这么多特征对于生还的影响，可能有点懵，不妨先简单总结一下根据可视化结果所获得的信息。

- 性别：女性的生还率高
- 船舱等级：越有钱越容易活下来，头等舱的生还率最高，三等舱的生还率最低。
- 年龄：10 岁以下的小朋友的存活率比较高，15-35 岁的年轻人存活率低。可能年轻人就是炮灰吧。
- 口岸：即使大多数一等舱的船客在 S 号口岸上的船， 但生还率不是最高的。 Q 号口岸的基本上是三等舱的船客。
- 兄弟姐妹父母爱人数量：有 1-2 个兄弟姐妹，配偶在船上，或 1-3 个父母的生还率比较高，独自一人或者一个大家庭都在船上的生还率比较低。

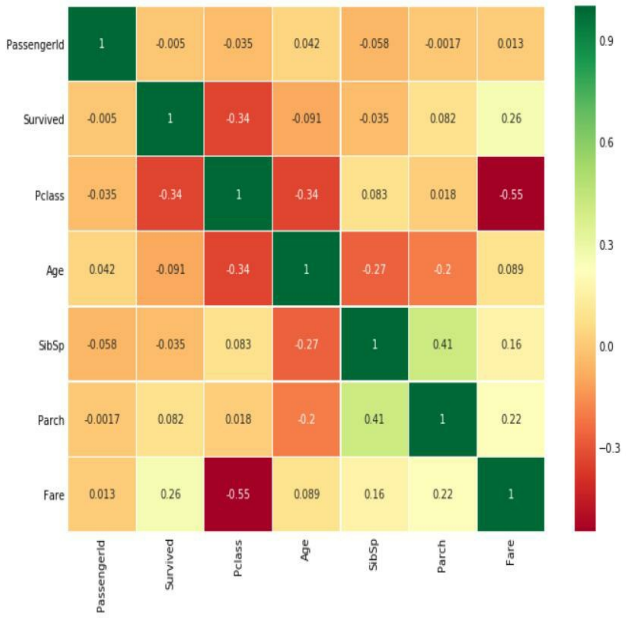
特征之间的相关性系数

相关性分为正相关与负相关，正相关指的是：如果特征 A 的数值变大会导致特征 B 的数值变大；负相关指的是：如果特征 A 的数值变小会导致特征 B 的数值变大。通常使用 [-1, 1] 的数值来表示两个特征之间的相关性，这个值称为相关性系数。若该系数为 1 那么表示两个特征之间完全正相关，若为 -1 则表示完全负相关，若为 0 则表示两个特征之间没有相关性(线性的)。

如果现在两个特征高度相关或者完全相关，这就意味着这两个特征都包含高度相似的信息，并且信息的差异非常小，所以其中一个特征是多余的。在构建模型时，我们应该尽量消除这种多余的特征，因为这样能减少训练的时间，也可以在某种程度上缓解过拟合。

所以接下来用热力图对相关性系数进行可视化。

```
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-->correlation matrix
fig=plt.gcf()
fig.set_size_inches(10,8)
plt.show()
```



从热力图上可以看出这些特征之间没有太大的相关性，最高的也就 SibSp与Parch，值为 0.41 。

特征工程

什么是特征工程？其实每当我们拿到数据时，并不是所有的特征都是有用的，可能有许多冗余的特征需要删掉，或者根据 EDA 的结果，我们可以根据已有的特征来添加新的特征，这其实就是特征工程。

接下来我们来尝试对一些特征进行处理。

年龄离散化

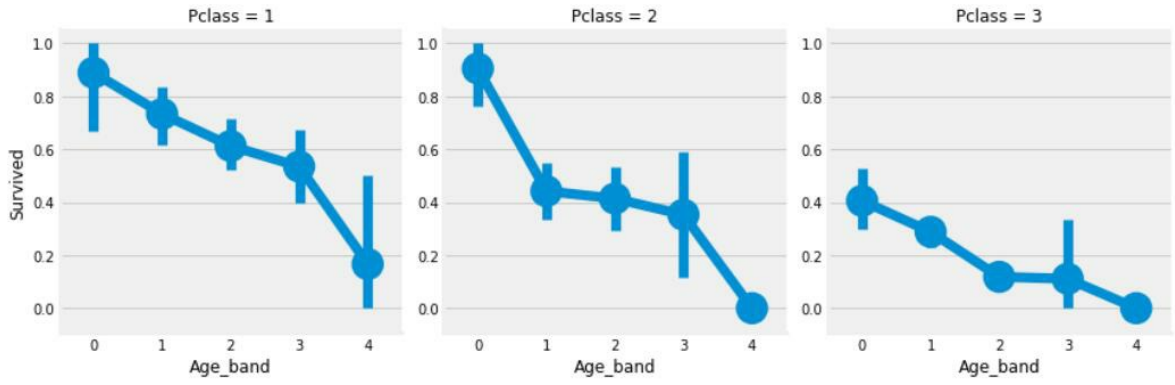
年龄是一个连续型的数值特征，有的机器学习算法对于连续性数值特征不太友好，例如决策树、随机森林等 tree-base model。所以我们可以考虑将年龄转换成年龄段。例如将年龄小于 16 的船客置为 0 ， 16 到 32 岁之间的置为 1 等。

```
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Initial	Age_band
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	Mr	1
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs	2

我们可以看一下转换成年龄段后，年龄段与生还率的关系。

```
sns.factorplot('Age_band','Survived',data=data,col='Pclass')
plt.show()
```



可以看出和我们之前 EDA 的结果相符，年龄越大，生还率越低。

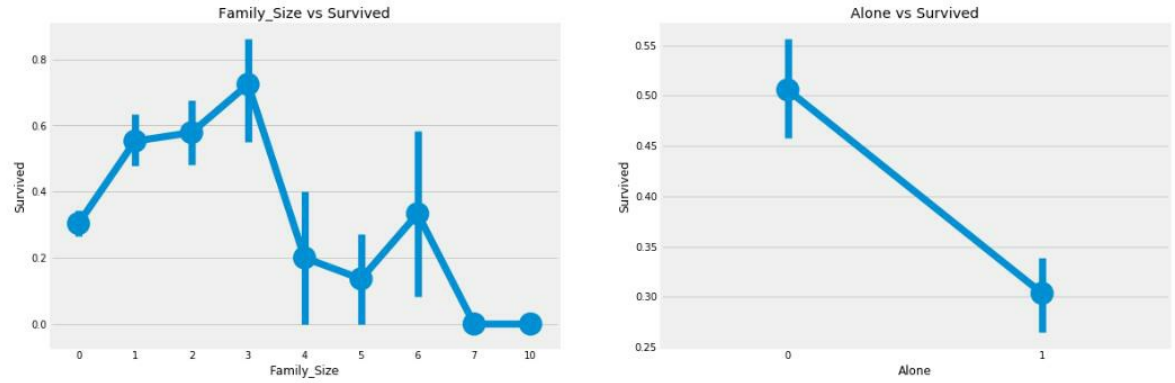
家庭成员数量与是否孤身一人

由于家庭成员数量和是否孤身一人好想对于是否生还有影响，所以我们不妨添加新的特征。

```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1
```

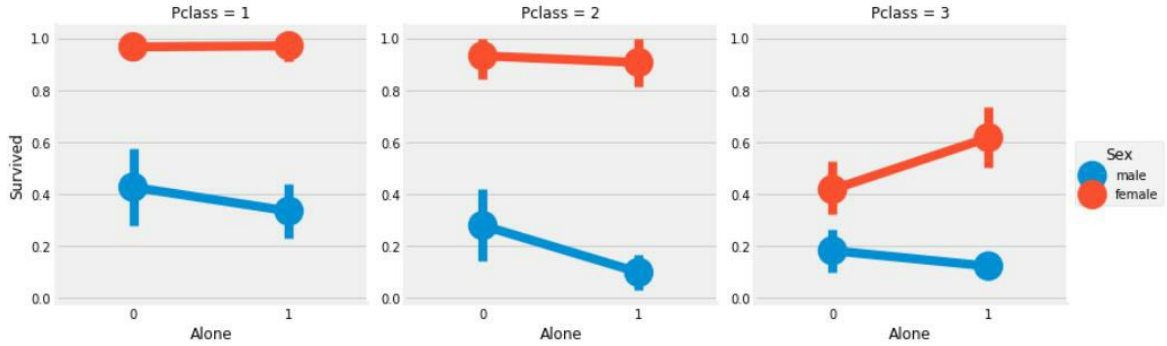
然后再可视化看一下

```
f,ax=plt.subplots(1,2,figsize=(18,6))
sns.factorplot('Family_Size','Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.factorplot('Alone','Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```



从图中可以很明显的看出，如果你是一个人，那么生还的几率比较低，而且对于人数大于 4 人的家庭来说生还率也比较低。感觉，这可能也是一个比较好的特征，可以再深入的看一下。

```
sns.factorplot('Alone','Survived',data=data,hue='Sex',col='Pclass')
plt.show()
```

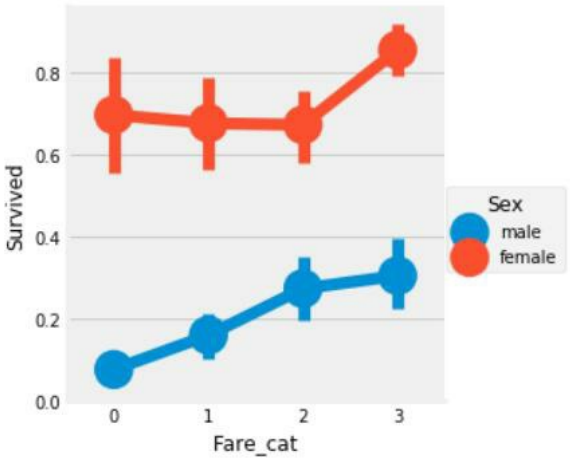


可以看出，除了三等舱的单身女性的生还率比非单身女性的生还率高外，单身并不是什么好事。

花费离散化

和年龄一样，花费也是一个连续性的数值特征，所以我们不妨将其离散化。

```
data['Fare_cat']=0
data.loc[data['Fare']<=7.91,'Fare_cat']=0
data.loc[(data['Fare']>7.91)&(data['Fare']<=14.454),'Fare_cat']=1
data.loc[(data['Fare']>14.454)&(data['Fare']<=31),'Fare_cat']=2
data.loc[(data['Fare']>31)&(data['Fare']<=513),'Fare_cat']=3

sns.factorplot('Fare_cat','Survived',data=data,hue='Sex')
plt.show()
```



很明显，花费越多生还率越高，金钱决定命运。

将字符串特征转换为数值型特征

由于我们的机器学习模型不支持字符串，所以需要将一些有用的字符串类型的特征转换成数值型的特征，比如：性别，口岸，姓名前缀。

```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
```

删掉没多大用处的特征

- 姓名：难道姓名和生死有关系？这也太玄乎了，我不信，所以把它删掉
- 年龄：由于已经根据年龄生成了新的特征“年龄段”，所以这个特征也需要删除。
- 票：票这个特征感觉是一堆随机的字符串，所以删掉。
- 花费：和年龄一样，删掉。
- 船舱：由于有很多缺失值，不好填充，所以可以考虑删掉。
- 船客ID：ID和生死应该没啥关系，所以删掉。

```
data.drop(['Name','Age','Ticket','Fare','Cabin','PassengerId'],axis=1,inplace=True)
```

构建模型进行预测

做好数据预处理后，可以将数据喂给我们的机器学习模型来进行训练和预测了。不过在构建模型之前，我们要使用处理训练集数据的方式来处理测试集。

```
test_data=pd.read_csv('./Titanic/test.csv')

test_data['Initial']=0
for i in test_data:
    test_data.loc[:, 'Initial'] = test_data.Name.str.extract('([A-Za-z]+)\.',expand=False) #lets extract the Salutations

test_data.loc[:, 'Initial'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt','Sir','Don'],['Miss','Miss','Miss','Other','Mr','Mrs','Mrs','Other','Other','Other','Mr','Mr','Mr'],inplace=True)

test_data.loc[(test_data.Age.isnull())&(test_data.Initial=='Mr'),'Age']=33
test_data.loc[(test_data.Age.isnull())&(test_data.Initial=='Mrs'),'Age']=36
test_data.loc[(test_data.Age.isnull())&(test_data.Initial=='Miss'),'Age']=22
test_data.loc[(test_data.Age.isnull())&(test_data.Initial=='Other'),'Age']=46

test_data['Embarked'].fillna('S', inplace=True)

test_data['Age_band']=0
test_data.loc[test_data['Age']<=16,'Age_band']=0
test_data.loc[(test_data['Age']>16)&(test_data['Age']<=32),'Age_band']=1
test_data.loc[(test_data['Age']>32)&(test_data['Age']<=48),'Age_band']=2
test_data.loc[(test_data['Age']>48)&(test_data['Age']<=64),'Age_band']=3
test_data.loc[test_data['Age']>64,'Age_band']=4

test_data['Family_Size']=0
test_data['Family_Size']=test_data['Parch']+test_data['SibSp']+1
test_data['Alone']=0
test_data.loc[test_data.Family_Size==1,'Alone']=1

test_data['Fare_cat']=0
test_data.loc[test_data['Fare']<=7.91,'Fare_cat']=0
test_data.loc[(test_data['Fare']>7.91)&(test_data['Fare']<=14.454),'Fare_cat']=1
test_data.loc[(test_data['Fare']>14.454)&(test_data['Fare']<=31),'Fare_cat']=2
test_data.loc[(test_data['Fare']>31)&(test_data['Fare']<=513),'Fare_cat']=3

test_data['Sex'].replace(['male','female'],[0,1],inplace=True)
test_data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
test_data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
test_data['Cabin'].replace(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'T'], [0, 1, 2, 3, 4, 5, 6, 7], inplace=True)

test_data.drop(['Name','Age','Ticket','Fare','Fare_Range','PassengerId'],axis=1,inplace=True)
```

然后可以使用机器学习模型来训练并预测了，这里使用的是随机森林。

```
Y_train = data['Survived']
X_train = data.drop(['Survived'], axis=1)

Y_test = test_data['Survived']
X_test = test_data.drop(['Survived'], axis=1)

clf = RandomForestClassifier(n_estimators=10)
clf.fit(X_train, Y_train)
predict = clf.predict(X_test)
print(accuracy_score(Y_test, predict))
```

此时看到预测的准确率达到了 0.8275 。

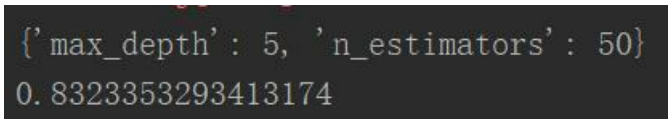
调参

很多机器学习算法有很多可以调整的参数(即超参数)，例如我们用的随机森林需要我们指定森林中有多少棵决策树，没棵决策树的最大深度等。这些超参数都或多或少的会影响这模型的性能。那么怎样才能找到合适的超参数，来让我们的模型性能达到比较好的效果呢？可以使用网格搜索!

网格搜索的意思其实就是遍历所有我们想要尝试的参数组合，看看哪个参数组合的性能最高，那么这组参数组合就是模型的最佳参数。

sklearn 为我们提供了网格搜索的接口，我们能很方便的进行网格搜索。

```
from sklearn.model_selection import GridSearchCV

# 想要调整的参数的字典，字典的key为参数名字，value为想要尝试参数值
param_grid = {'n_estimators': [10, 20, 50, 100, 150, 200],'max_depth': [5, 10, 15, 20, 25, 30]}

# 采用5折验证的方式进行网格搜索，分类器为随机森林
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, Y_train)

# 打印最佳参数组合
print(grid_search.best_params_)
# 打印最佳参数组合时模型的最佳性能
print(grid_search.best_score_)
```



可以看到经过调参之后，我们的随机森林模型的性能提高到了 0.8323 ，提升了接近 1% 的准确率。然后我们使用最佳参数构造随机森林，并对测试集测试会发现，测试集的准确率达到了 0.8525。

```
Y_train = data['Survived']
X_train = data.drop(['Survived'], axis=1)

Y_test = test_data['Survived']
X_test = test_data.drop(['Survived'], axis=1)

clf = RandomForestClassifier(n_estimators=50, max_depth=5)
clf.fit(X_train, Y_train)
predict = clf.predict(X_test)
print(accuracy_score(Y_test, predict))
```

什么是强化学习

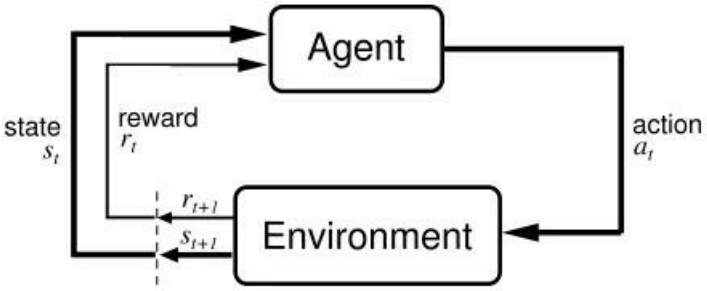
强化学习是一类算法，是让计算机实现从一开始完全随机的进行操作，通过不断地尝试，从错误中学习，最后找到规律，学会了达到目的的方法。这就是一个完整的强化学习过程。让计算机在不断的尝试中更新自己的行为，从而一步步学习如何操自己的行为得到高分。

它主要包含四个元素，Agent、环境状态、行动、奖励，强化学习的目标就是获得最多的累计奖励。

让我们想象一下比赛现场：

计算机有一位虚拟的裁判，这个裁判他不会告诉你如何行动，如何做决定，他为你做的事只有给你的行为打分，最开始，计算机完全不知道该怎么做，行为完全是随机的，那计算机应该以什么形式学习这些现有的资源，或者说怎么样只从分数中学习到我应该怎样做决定呢？很简单，只需要记住那些高分，低分对应的行为，下次用同样的行为拿高分, 并避免低分的行为。

计算机就是 Agent，他试图通过采取行动来操纵环境，并且从一个状态转变到另一个状态，当他完成任务时给高分(奖励)，但是当他没完成任务时，给低分(无奖励)。这也是强化学习的核心思想。

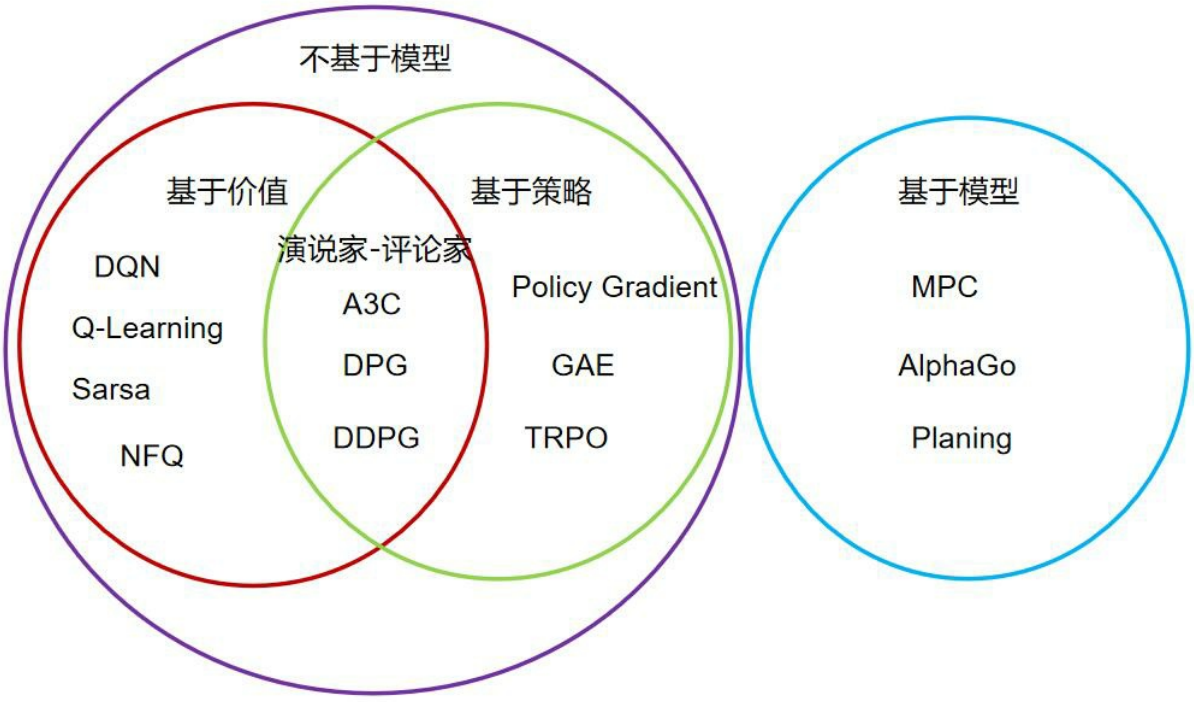


在强化学习中有很多算法，如果按类别划分可以划分成 model-based (基于模型)和 model-free (不基于模型)两大类。

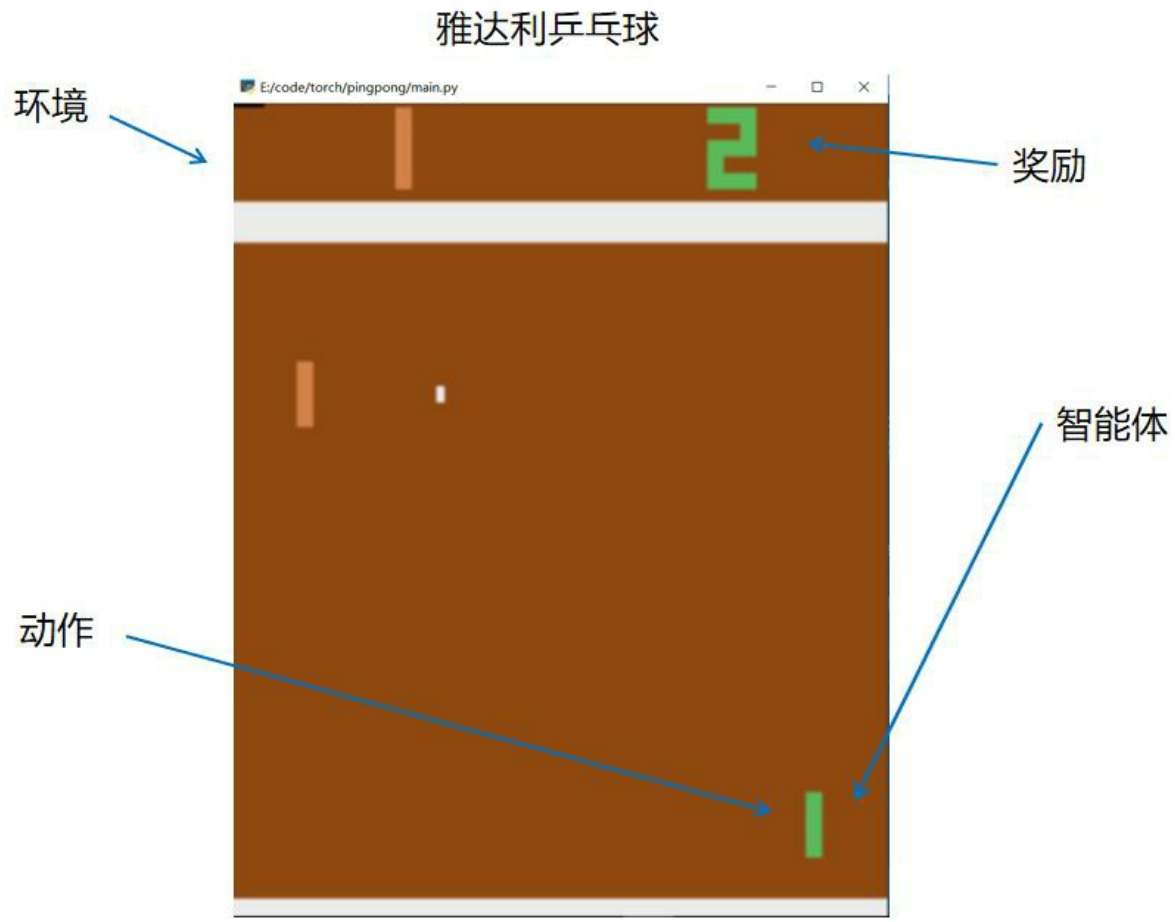
如果我们的 Agent 不理解环境，环境给了什么就是什么，我们就把这种方法叫做 model-free，这里的 model 就是用模型来表示环境，理解环境就是学会了用一个模型来代表环境，所以这种就是 model-based 方法。

Model-free 的方法有很多, 像 Q learning、Sarsa、Policy Gradients 都是从环境中得到反馈然后从中学习。而 model-based 只是多了一道程序，为真实世界建模，也可以说他们都是 model-free 的强化学习, 只是 Model-based 多出了一个虚拟环境，我们可以先在虚拟环境中尝试，如果没问题，再拿到现实环境中来。

model-free 中, Agent 只能按部就班，一步一步等待真实世界的反馈，再根据反馈采取下一步行动。而 model-based，能通过想象来预判断接下来将要发生的所有情况，然后选择这些想象情况中最好的那种，并依据这种情况来采取下一步的策略，这也就是围棋场上 AlphaGo 能够超越人类的原因。



在这里主要介绍一下 model-free 中基于策略的一种算法，Policy Gradient。在介绍该算法之前，我们先要明确一下这个雅达利乒乓球游戏中的环境状态是游戏画面，**Agent**是我们操作的挡板，奖励是分数，动作是上或者下。



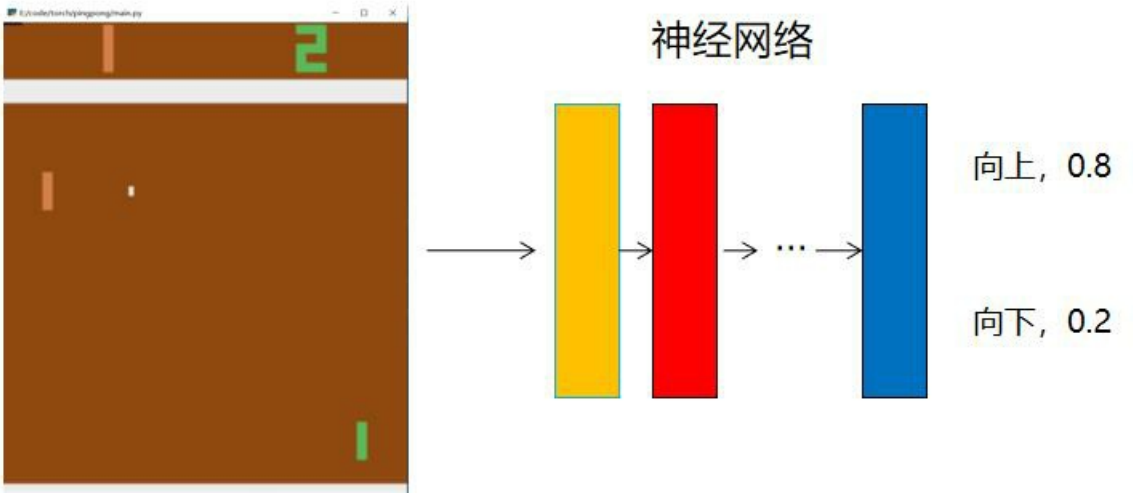
Policy Gradient

Policy Gradient的核心思想

其实 Policy Gradient 的核心思想非常简单，就是找一个函数π，这个函数π能够根据现在环境的状态(state)来产生接下来要采取的行动或者动作(action)。即$\pi(state) \rightarrow action$。

函数π其实可以看成是一个模型，那么想在无数次尝试中寻找出能让 Agent 尽量拿高分的模型应该怎样来找呢？我相信您应该猜到了！没错！就是神经网络！

我们可以将游戏画面传给神经网络作为输入，然后神经网络预测一下当前游戏画面下，下一步动作的概率分布。



细心的您可能会发现，如果每次取概率最高的动作作为下一步的动作，那不就成分类了么。其实 Policy Gradient 的并不是每次都选取概率最高的动作，而是根据动作的概率分布进行采样。也就是说就算我预测出来的向上挪的概率为 80% ，也不一定会向上挪。

那么为什么采样而不是直接选取概率最大的呢？因为这样很有灵性。可以想象一下，我们和别人下棋的时候，如果一直按照套路来下，那么对手很可能能够猜到我们下一步棋会怎么走，从而占据主动。如果我们时不时地不按套路出牌，但是这种不按套路的动作不会降低太多对于我们能够赢下这一局棋的几率。那么对手很可能会不知所措，主动权就掌握在我们手里。就像《天龙八部》中虚竹大破珍珑棋局时一样，可能有灵性一点，会有意想不到的效果。

Policy Gradient 的原理

现在已经知道 Policy Gradient 是通过神经网络来训练模型，该模型需要根据环境状态来预测出下一步动作的概率分布，并根据这个概率分布进行采样，将采样到的动作作为下一步的动作。

那么会有一个灵魂拷问，就是怎样来鉴定我的神经网络是好还是坏呢？很显然，当然是赢的越多越好了！所以我们不妨假设，让计算机玩 10 把乒乓球游戏，那么可能会有这样的一个统计结果。

那么怎样评价这 10 把游戏打的好还是不好呢？也很明细，把 10 把游戏的所有反馈全部都加起来就好了。如果把这些反馈的和称为总反馈(总得分)，那么就有总反馈**(**总得分**)=**第**1**把反馈**1+**第**1**把反馈**2+...+**第**10**把反馈**m**。也就是说总反馈越高越好。

说到这，有一个问题需要弄清楚：假设总共玩了 100 把，每 10 把计算一次总反馈，那么这 10 次的总反馈会不会是一模一样的呢？其实仔细想想会发现不会一摸一样，因为：

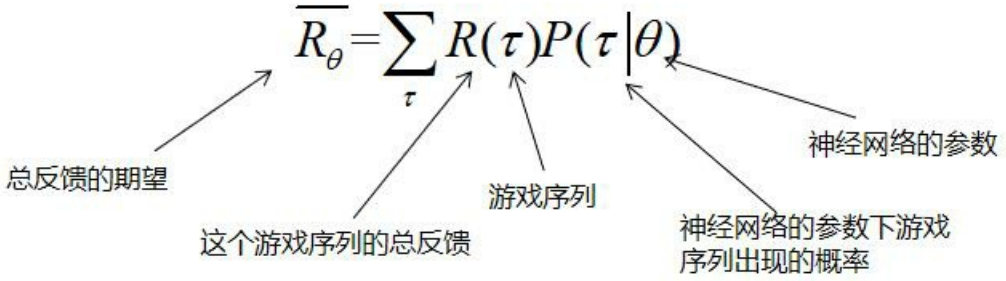
- 游戏的状态实时在变，所以环境状态不可能一直是一样的。
- 动作是从一个概率分布中采样出来的。

既然总反馈一直会变，那么我们可以尝试换一种思路，即计算总反馈的期望，即总反馈的期望越高越好。那这个期望怎么算呢？

首先我们可以将每一把游戏看成一个游戏序列(状态**1->**动作**1->**反馈**1->**状态**2->**动作**2->**反馈**2 ...** 状态**N->**动作**N->**反馈**N**)。那么每一个游戏序列(即每一把游戏)的反馈**=**反馈**1+**反馈**2+...+**反馈**N**。因此，若假设$R(\tau)$表示游戏序列τ的反馈，则有:$R(\tau)=\sum_{n=1}^{N}\tau_n$。

如果我们把整个乒乓球游戏所有可能出现的状态，动作，反馈组合起来看成是玩了 N(N很大很大) 把游戏，就会有 N 个游戏序列(游戏序列**1**，游戏序列**2**，游戏序列**3, ... ,** 游戏序列**N**)。那么我们在玩游戏时所得到的游戏序列实际上就是从这 N 个游戏序列中采样得到的。

所以我们游戏的总的反馈期望$\overline{R_\theta}$可表示为：$\overline{R_\theta}=\sum_\tau R(\tau)P(\tau|\theta)$。这个公式看起来复杂，其实不难理解。



假设我们玩了 10 把游戏，就相当于得到了 10 个游戏序列$[\tau_1,\tau_2,...,\tau_{10}]$。这 10 个游戏序列就相当于从 P 中采样了 10 次τ。所以总反馈期望$\overline{R_\theta}$又可以近似的表示为：

$$\overline{R_\theta}\approx\frac{1}{N}\sum_{n=1}^{N}R(\tau^n)$$

由于$\overline{R_\theta}$的值越大越好，所以我们可以使用梯度上升的方式来更新θ。所以就有如下数学推导：

$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \overline{R}_{\theta^{old}}$$

$$\overline{R_\theta} = \sum_\tau R(\tau) P(\tau|\theta)$$

$$\nabla \overline{R_\theta} = \sum_\tau R(\tau) \nabla P(\tau|\theta)$$

$$\nabla \overline{R_\theta} = \sum_\tau R(\tau) P(\tau|\theta) \frac{\nabla P(\tau|\theta)}{P(\tau|\theta)}$$

$$\downarrow \quad \frac{d \log(f(x))}{dx} = \frac{1}{f(x)} \frac{df(x)}{dx}$$

$$\nabla \overline{R_\theta} = \sum_\tau R(\tau) P(\tau|\theta) \nabla \log P(\tau|\theta)$$

又由于：

$$\overline{R_\theta} = \sum_\tau R(\tau) P(\tau|\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n)$$

所以就有：

$$\nabla \overline{R_\theta} \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla log P(\tau^n|\theta)$$

您会发现$\sum_{n=1}^{N} R(\tau^n)$很好算，只要把反馈全部加起来就完事了，难算的是$\nabla log P(\tau^n|\theta)$。所以我们来看一下$\nabla log P(\tau^n|\theta)$应该怎么算。

由于一个游戏序列τ是由多个状态，动作，反馈构成的，即：

$$\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, ..., s_T, a_T, r_T\}$$

所以：

$$P(\tau|\theta) = P(s_1) P(a_1|s_1, \theta) P(r_1, s_2|s_1, a_1) P(a_2|s_2, \theta) P(r_2, s_3|s_2, a_2) ...$$

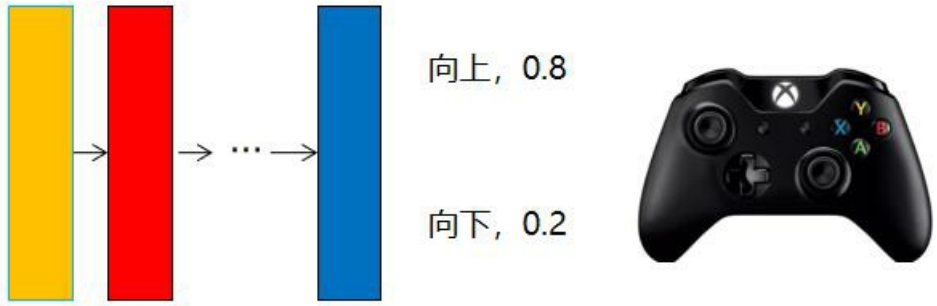
稍微整理一下可知：

$$P(\tau|\theta) = P(s_1) \prod_{t=1}^{T} P(a_t|s_t, \theta) P(\tau_t, s_{t+1}|s_t, a_t)$$

然后两边取log会得到：

$$log P(\tau|\theta) = \sum_{t=1}^{T} \nabla log P(a_t|s_t, \theta)$$

$P(a_t|s_t, \theta)$其实就是我们神经网络根据环境状态预测出来的下一步的动作概率分布。



OK，到这里，Policy Gradient 的数学推导全部推导完毕了。我们不妨用一张图来总结一下 Policy Gradient 的算法流程。流程如下：

使用**Policy Gradient**玩乒乓球游戏

安装 **gym**

想要玩乒乓球游戏，首先得有乒乓球游戏。OpenAI 的 gym 为我们提供了模拟游戏的环境。使得我们能够很方便地得到游戏的环境状态，并作出动作。想要安装 gym 非常简单，只要在命令行中输入`pip install gym`即可。

安装 **atari_py**

由于乒乓球游戏是雅达利游戏机上的游戏，所以需要安装 atari_py 来实现雅达利环境的模拟。安装 atari_py 也很方便，只需在命令行中输入`pip install --no-index -f https://github.com/Kojoley/atari-py/releases atari_py`即可。

开启游戏

当安装好所需要的库之后，我们可以使用如下代码开始游戏：

```
# 开启乒乓球游戏环境
import gym

env = gym.make('Pong-v0')

# 一直渲染游戏画面
while True:
    env.render()
    # 随机做动作，并得到做完动作之后的环境(observation)，反馈(reward)，是否结束(done)
    observation, reward, done, _ = env.step(env.action_space.sample())
```

游戏画面预处理

由于`env.step`返回出来的 observation 是一张RGB的三通道图，而且我们的挡板怎么移动只跟挡板和球有关系，所以我们可以尝试将三通道图转换成一张二值化的图，其中挡板和球是 1，背景是 0 。

```


# 游戏画面预处理
def prepro(I):
    I = I[35:195]  #不要上面的记分牌
    I = I[::2, ::2, 0]  #scale 0.5，所以I是高为80，宽为80的单通道图
    I[I == 144] = 0  # 背景赋值为0
    I[I == 109] = 0  # 背景赋值为0
    I[I != 0] = 1  # 目标为1
    return I.astype(np.float).ravel() #将二维图压成一维的数组

# cur_x为预处理后的游戏画面
cur_x = prepro(observation)
```

游戏的画面是逐帧组成的，如果我们将当前帧和上一帧的图像相减就能得到能够表示两帧之间的变化的帧差图，将这样的帧差图作为神经网络的输入的话会是个不错的选择。

```
# x为帧差图
x = cur_x - prev_x
# 将当前帧更新为上一帧
prev_x = cur_x
```

搭建神经网络

神经网络可以根据自己的喜好来搭建，在这里我使用最简单的只有两层全连接层的网络模型来进行预测，由于我们挡板的动作只有上和下，所以最后的激活函数为 sigmoid 函数。

```
# 神经网络中神经元的参数
model = {}
# 随机初始化第一层的神经元参数，总共200个神经元
model['W1'] = np.random.randn(H, D) / np.sqrt(D)
```

```
# 随机初始化第二层的神经元参数，总共200个神经元
model['W2'] = np.random.randn(H) / np.sqrt(H)

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

# 神经网络的前向传播，x为输入的帧差图
def policy_forward(x):
    h = np.dot(model['W1'], x)
    # relu
    h[h < 0] = 0
    logp = np.dot(model['W2'], h)
    # sigmoid激活
    p = sigmoid(logp)
    # p为下一步要往下挪的概率，h为隐藏层中神经元的参数
    return p, h


# 算每层的参数偏导，eph为一个游戏序列的隐藏层中神经元的参数，epdlogp为一个游戏序列中反馈期望的偏导。
def policy_backward(eph, epdlogp):
    dW2 = np.dot(eph.T, epdlogp).ravel()
    dh = np.outer(epdlogp, model['W2'])
    dh[eph <= 0] = 0
    dW1 = np.dot(dh.T, epx)
    return {'W1': dW1, 'W2': dW2}
```

训练神经网络

```
while True:
    env.render()

    # 游戏画面预处理
    cur_x = prepro(observation)
    # 得到帧差图
    x = cur_x - prev_x if prev_x is not None else np.zeros(D)
    # 将上一帧更新为当前帧
    prev_x = cur_x

    #前向传播
    aprob, h = policy_forward(x)
    #从动作概率分布中采样，action=2表示往上挪，action=3表示往下挪
    action = 2 if np.random.uniform() < aprob else 3

    # 环境
    xs.append(x)
    # 隐藏层状态
    hs.append(h)
    # 将2和3改成1和0，因为sigmoid函数的导数为f(x)*(1-f(x))
    y = 1 if action == 2 else 0
    dlogps.append(y - aprob)

    # 把采样到的动作传回环境
    observation, reward, done, info = env.step(action)
    # 如果得一分则reward为1，丢一份则reward为-1
    reward_sum += reward

    # 记录反馈
    drs.append(reward)

    # 当有一方得到21分后游戏结束
    if done:
        episode_number += 1

        epx = np.vstack(xs)
        eph = np.vstack(hs)
        epdlogp = np.vstack(dlogps)
        epr = np.vstack(drs)
        discounted_epr = discount_rewards(epr)
        # 将反馈进行zscore归一化，有利于训练
        discounted_epr -= np.mean(discounted_epr)
        discounted_epr /= np.std(discounted_epr)

        #算期望
        epdlogp *= discounted_epr
        #算梯度
        grad = policy_backward(eph, epdlogp)
        for k in model:
            grad_buffer[k] += grad[k]
```

```
        # 每batch_size次游戏更新一次参数
        if episode_number % batch_size == 0:
            #rmsprop梯度上升
            for k, v in model.items():
                g = grad_buffer[k]
                rmsprop_cache[k] = decay_rate * rmsprop_cache[k] + (1 - decay_rate) * g ** 2
                model[k] += learning_rate * g / (np.sqrt(rmsprop_cache[k]) + 1e-5)
                grad_buffer[k] = np.zeros_like(v)

        # 每100把之后保存模型
        if episode_number % 100 == 0:
            pickle.dump(model, open('save.p', 'wb'))
        reward_sum = 0
        # 重置游戏
        observation = env.reset()
        prev_x = None
```

加载模型玩游戏

经过漫长的训练过程后，我们可以将训练好的模型加载进来开始玩游戏了。

```
import numpy as np
import pickle
import gym

model = pickle.load(open('save.p', 'rb'))

env = gym.make("Pong-v0")
observation = env.reset()

while True:
    env.render()
    cur_x = prepro(observation)
    x = cur_x - prev_x if prev_x is not None else np.zeros(80*80)
    prev_x = cur_x
    aprob, h = policy_forward(x)
    #从动作概率分布中采样
    action = 2 if np.random.uniform() < aprob else 3
    observation, reward, done, info = env.step(action)

    if done:
        observation = env.reset()
        prev_x = None
```

实训推荐

关于本书的实验与涉及的案例均可以在平台进行体验，名称与链接如下：

名称	链接
《机器学习》---绪论	https://www.educoder.net/shixuns/4fhemfr9/challenges
《机器学习》---模型评估与选择	https://www.educoder.net/shixuns/cbsfh3r5/challenges
《机器学习》---线性回归	https://www.educoder.net/shixuns/4awq25iv/challenges
《机器学习》---逻辑回归	https://www.educoder.net/shixuns/tw9up75v/challenges
《机器学习》---kNN算法	https://www.educoder.net/shixuns/aw9bxy75/challenges
《机器学习》---决策树	https://www.educoder.net/shixuns/hl7wacq5/challenges
《机器学习》---随机森林	https://www.educoder.net/shixuns/ya8h7utx/challenges
《机器学习》---k-means	https://www.educoder.net/shixuns/k6fp4saq/challenges
《机器学习》---AGNES	https://www.educoder.net/shixuns/qy9gozt8/challenges
泰坦尼克号生还预测	https://www.educoder.net/shixuns/kz3fixv9/challenges

也可通过扫码查看整套课程，二维码如下: