【传统 CV 算法】机器学习算法

目 录

第一章 概述

事实上机器学习可以被认为是一个独立的学科分支存在。它与计算机视觉存在一些交集，但机器学习具有更广泛的应用，例如：数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA 序列测序、语音和手写识别、战略游戏和机器人运用等等。因此，如果想完整地学习机器学习需要阅读一些更周到详细的书籍资料。而本书的核心内容在于计算机视觉，因此对于机器学习不会作详尽介绍。本书重点在于提炼机器学习当中的重要算法，并介绍这些算法在计算机视觉中的应用。

1. 机器学习概述

机器学习(Machine Learning, ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。

它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。

2. 机器学习发展史

机器学习是人工智能研究较为年轻的分支，它的发展过程大体上可分为 4 个时期。

第一阶段是在 20 世纪 50 年代中叶到 60 年代中叶，属于热烈时期。

第二阶段是在 20 世纪 60 年代中叶至 70 年代中叶，被称为机器学习的冷静时期。

第三阶段是从 20 世纪 70 年代中叶至 80 年代中叶，称为复兴时期。

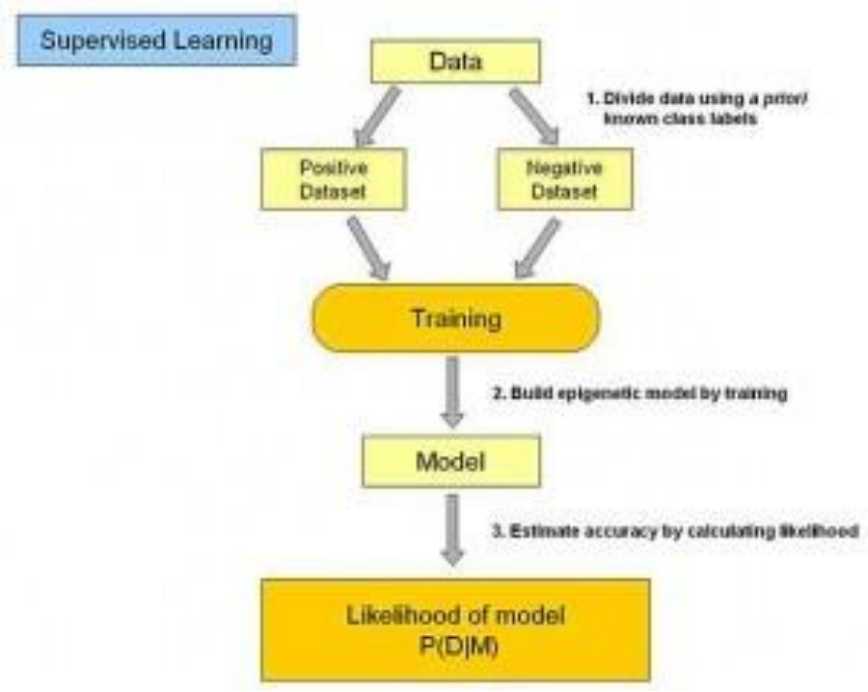
机器学习的最新阶段始于 1986 年。

更详细的发展史介绍请参阅：
https://blog.csdn.net/u012328159/article/details/52462433

3. 机器学习算法分类

根据数据类型的不同，对一个问题的建模有不同的方式。在机器学习或者人工智能领域，人们首先会考虑算法的学习方式。将算法按照学习方式分类是一个不错的想法，这样可以让人们在建模和算法选择的时候考虑能根据输入数据来选择最合适的算法来获得最好的结果。在机器学习领域，有几种主要的学习方式：**监督式学习，非监督式学习，半监督式学习和强化学习**。

3.1 监督式学习



在监督式学习下，输入数据被称为“训练数据”，每组训练数据有一个明确的标识或结果，如对防垃圾邮件系统中“垃圾邮件”“非垃圾邮件”，对手写数字识别中的“1“，”2“，”3“，”4“等。在建立预测模型的时候，监督式学习建立一个学习过程，将预测结果与“训练数据”的实际结果进行比较，不断的调整预测模型，直到模型的预测结果达到一个预期的准确率。监督式学习的常见应用场景如分类问题和回归问题。常见监督式学习算法有决策树学习(ID3,C4.5 等)，朴素贝叶斯分类，最小二乘回归，逻辑回归（Logistic Regression），支撑矢量机，集成方法以及反向传递神经网络（Back Propagation Neural Network）等等。

3.2 非监督式学习

在非监督式学习中，数据并不被特别标识，学习模型是为了推断出数据的一些内在结构。常见的应用场景包括关联规则的学习以及聚类等。常见非监督学习算法包括奇异值分解、主成分分析，独立成分分析，Apriori 算法以及 k-Means 算法等等。

3.3 半监督式学习

在此学习方式下，输入数据部分被标识，部分没有被标识，这种学习模型可以用来进行预测，但是模型首先需要学习数据的内在结构以便合理的组织数据来进行预测。应用场景包括分类和回归，算法包括一些对常用监督式学习算法的延伸，这些算法首先试图对未标识数据进行建模，在此基础上再对标识的数据进行预测。如图论推理算法（Graph Inference）或者拉普拉斯支持向量机（Laplacian SVM）等。

3.4 强化学习

在这种学习模式下，输入数据作为对模型的反馈，不像监督模型那样，输入数据仅仅是作为一个检查模型对错的方式，在强化学习下，输入数据直接反馈到模型，模型必须对此立刻作出调整。常见的应用场景包括动态系统以及机器人控制等。常见算法包括 Q-Learning 以及时间差学习（Temporal difference learning）。在企业数据应用的场景下， 人们最常用的可能就是监督式学习和非监督式学习的模型。 在图像识别等领域，由于存在大量的非标识的数据和少量的可标识数据， 目前半监督式学习是一个很热的话题。 而强化学习更多的应用在机器人控制及其他需要进行系统控制的领域。

4. 机器学习常见算法

挖掘主题	算法	发表时间
分类	C4.5	1993
聚类	K-Means	1967
统计学习	SVM	1995
关联分析	Apriori	1994
统计学习	EM	2000
链接挖掘	PageRank	1998
集装与推进	AdaBoost	1997
分类	kNN	1996
分类	Naive Bayes	2001
分类	CART	1984

5.*机器学习的典型应用

- 购物篮分析：纸尿布和啤酒
    - **关联规则**

- 用户细分精准营销：神州大众卡，全球通，动感地带，神州行
    - **聚类**

- 垃圾邮件
    - **朴素贝叶斯**

- 信用卡欺诈
  - **决策树**

- 互联网广告
  - **ctr 预估**

- 推荐系统
  - **协同过滤**

- 自然语言处理
  - **情感分析**
  - **实体识别**

- 图像识别
  - **深度学习**

- 语音识别

- 个性化医疗

- 情感分析

- 人脸识别

- 自动驾驶

第二章 监督式学习

本章节主要介绍机器学习传统算法的监督学习部分。监督学习算法主要解决回归和分类两大问题。只能做回归的算法是线性回归，只能做分类的算法是逻辑回归和贝叶斯分类。其他的算法既可以做回归又可以做分类。

1. 线性回归

线性回归属于监督学习，因此方法和监督学习是一样的，先给定一个训练集，根据这个训练集学习出一个线性函数，然后测试这个函数训练的好不好（即此函数是否足够拟合训练集数据），挑选出最好的函数（cost function 最小）即可。

1.1 线性回归分类

(1) 单变量回归

我们能够给出单变量线性回归的模型：

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|$$

我们需要使用到 Cost Function（代价函数），代价函数越小，说明线性回归地越好（和训练集拟合地越好），当然最小就是 0，即完全拟合。

(2) 多变量回归

多变量线性回归之前必须要 Feature Scaling。思想：将各个 feature 的值标准化，使得取值范围大致都在-1<=x<=1 之间。

这里我们可以定义出多变量线性回归的模型：

$$h_\theta(x)=\theta_0+\theta_1x_1+\theta_2x_2+\cdots+\theta_nx_n$$

1.2 求拟合方程方法

(1) 最小二乘法

“最小二乘法”的核心就是保证所有数据偏差的平方和最小。（“平方”的在古时侯的称谓为“二乘”）。

$$\sum_{i=1}^{n}\left(y_i-\sum_{j=0}^{p}w_jx_{ij}\right)^2$$

(2) 岭回归

- 预测精度：这里要处理好这样一对问题，即样本的数量 n 和特征的数量 p
  - n>>p 时，最小二乘法回归会有较小的方差
  - n≈p 时，容易产生过拟合
  - n<<p 时，最小二乘回归得不到有意义的结果

岭回归(Ridge Regression)是在平方误差的基础上增加正则项。通过确定 lamda 的值可以使得在方差和偏差之间达到平衡。

$$\sum_{i=1}^{n}\left(y_i-\sum_{j=0}^{p}w_jx_{ij}\right)^2+\lambda\sum_{j=0}^{p}w_j^2 \quad ,\lambda>0$$

岭回归优于最小二乘回归的原因在于方差-偏倚选择。随着 lambda 的增大，模型方差减小而偏倚（轻微的）增加。

岭回归的一个缺点：在建模时，同时引入 p 个预测变量，罚约束项可以收缩这些预测变量的待估系数接近 0,但并非恰好是 0（除非 lambda 为无穷大）。这个缺点对于模型精度影响不大，但给模型的解释造成了困难。这个缺点可以由 lasso 来克服。(所以岭回归虽然减少了模型的复杂度，并没有真正解决变量选择的问题)

(3) Lasso 回归

lasso 是在 RSS 最小化(Residual Sum of Squares)的计算中加入一个 l1 范数作为罚约束：

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_jx_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| \;=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|$$

l1 范数的好处是当 lambda 充分大时可以把某些待估系数精确地收缩到 0。

2. 逻辑回归

（1 ）逻辑回归模型

简单来说线性回归就是直接将特征值和其对应的概率进行相乘得到一个结果，逻辑回归则是这样的结果上加上一个逻辑函数，这里选用的就是 Sigmoid 函数。逻辑回归分为二分类和多分类。

假设我们的样本是{x, y}，y 是 0 或者 1，表示正类或者负类，x 是我们的 m 维的样本特征向量。那么这个样本 x 属于正类，也就是 y=1 的“概率”可以通过下面的逻辑函数来表示：

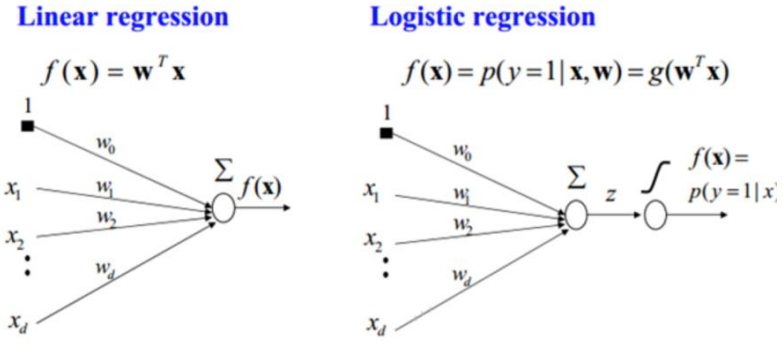
$$p(y=1|\mathbf{x};\theta)=\sigma(\theta^{\top}\mathbf{x})=\frac{1}{1+\exp(-\theta^{\top}\mathbf{x})}$$

这里 θ 是模型参数，也就是回归系数， σ 是 sigmoid 函数。实际上这个函数是由下面的对数几率（也就是 x 属于正类的可能性和负类的可能性的比值的对数）变换得到的：

$$\log it(\mathrm{x})=\ln(\frac{P(\mathrm{y}=1|\mathrm{x})}{P(\mathrm{y}=0|\mathrm{x})})$$

$$=\ln(\frac{P(\mathrm{y}=1|\mathrm{x})}{1-P(\mathrm{y}=1|\mathrm{x})})=\theta_0+\theta_1x_1+\theta_2x_2+\ldots+\theta_mx_m$$

所以说上面的 logistic 回归就是一个线性分类模型，它与线性回归的不同点在于：为了将线性回归输出的很大范围的数，例如从负无穷到正无穷，压缩到 0 和 1 之间，这样的输出值表达为“可能性”才能说服广大民众。当然了，把大值压缩到这个范围还有个很好的好处，就是可以消除特别冒尖的变量的影响（不知道理解的是否正确）。而实现这个伟大的功能其实就只需要平凡一举，也就是在输出加一个 logistic 函数。另外，对于二分类来说，可以简单的认为：如果样本 x 属于正类的概率大于 0.5，那么就判定它是正类，否则就是负类。



所以说，LogisticRegression 就是一个被 logistic 方程归一化后的线性回归，仅此而已。

(2) 代价函数

假设我们有 n 个独立的训练样本{(x1,y1), (x2,y2),……, (xn,yn)},y={0,1},那每一个观察到的样本(xi,yi)出现的概率是：

$$P(\mathrm{y}_i,\mathrm{x}_i)=P(\mathrm{y}_i=1\mid \mathrm{x}_i)^{y_i}(1-P(\mathrm{y}_i=1\mid \mathrm{x}_i))^{1-y_i}$$

上面为什么是这样呢？当 y=1 的时候，后面那一项是不是没有了，那就只剩下 x 属于 1 类的概率，当 y=0 的时候，第一项是不是没有了，那就只剩下后面那个 x 属于 0 的概率（1 减去 x 属于 1 的概率）。所以不管 y 是 0 还是 1，上面得到的数，都是(x,y)出现的概率。那我们的整个样本集，也就是 n 个独立的样本出现的思然函数为（因为每个样本都是独立的，所以 n 个样本出现的概率就是他们各自出现的概率相乘）：

$$L(\theta)=\prod P(\mathrm{y}_i=1\mid \mathrm{x}_i)^{y_i}(1-P(\mathrm{y}_i=1\mid \mathrm{x}_i))^{1-y_i}$$

那最大似然法就是求模型中使得似然函数最大的系数取值 θ *。这个最大似然函数就是我们的代价函数（cost function）了。

那代价函数有了，我们下一步要做的就是优化求解了。我们先尝试对上面的代价函数求导，看导数为 0 的时候可不可以解出来，也就是有没有解析解，有这个解的时候，就皆大欢喜了，一步到位。如果没有就需要通过迭代了，耗时耗力。

我们先变换下 L(θ)：取自然对数，然后化简（不要看到一堆公式就害怕哦，很简单的哦，只需要耐心一点点，自己动手推推就知道了。注：有 xi 的时候，表示它是第 i 个样本，下面没有做区分了，相信你的眼睛是雪亮的），得到：

$$\begin{aligned}
&L(\theta)=\log(\prod P(\mathrm{y}_i=1\mid \mathrm{x}_i)^{y_i}(1-P(\mathrm{y}_i=1\mid \mathrm{x}_i))^{1-y_i})\\
&=\sum_{i=1}^{n} y_i\log p(\mathrm{y}_i=1\mid \mathrm{x}_i)+(1-y_i)\log(1-p(\mathrm{y}_i=1\mid \mathrm{x}_i))\\
&=\sum_{i=1}^{n} y_i\log\frac{p(\mathrm{y}_i=1\mid \mathrm{x}_i)}{1-p(\mathrm{y}_i=1\mid \mathrm{x}_i)}+\sum_{i}^{n}\log(1-p(\mathrm{y}_i=1\mid \mathrm{x}_i))\\
&=\sum_{i=1}^{n} y_i(\theta_0+\theta_1\mathrm{x}_1+\ldots+\theta_m\mathrm{x}_m)+\sum_{i}^{n}\log(1-p(\mathrm{y}_i=1\mid \mathrm{x}_i))\\
&=\sum_{i=1}^{n} y_i(\theta^T\mathrm{x}_i)-\sum_{i}^{n}\log(1+e^{\theta^T x_i})
\end{aligned}$$

这时候，用 L(θ)对 θ 求导，得到：

$$\frac{\partial L(\theta)}{\partial\theta}=\sum_{i=1}^{n} y_i\mathrm{x}_i-\sum_{i}^{n}\frac{e^{\theta^T x_i}}{1+e^{\theta^T x_i}}\mathrm{x}_i=\sum_{i=1}^{n}(y_i-\sigma(\theta^T x_i))\mathrm{x}_i$$

然后我们令该导数为 0，你会很失望的发现，它无法解析求解。不信你就去尝试一下。所以没办法了，只能借助高大上的迭代来搞定了。运用用了经典的梯度下降算法就可以了。

3. 贝叶斯分类

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。本文作为分类算法的第一篇，将首先介绍分类问题，对分类问题进行一个正式的定义。然后，介绍贝叶斯分类算法的基础——贝叶斯定理。最后，通过实例讨论贝叶斯分类中最简单的一种：朴素贝叶斯分类。

(1) 贝叶斯定理

这个定理解决了现实生活里经常遇到的问题：已知某条件概率，如何得到两个事件交换后的概率，也就是在已知 P(A|B)的情况下如何求得 P(B|A)。这里先解释什么是条件概率：

P（A|B）表示事件 B 已经发生的前提下，事件 A 发生的概率，叫做事件 B 发生下事件 A 的条件概率。其基本求解公式为：

$$P(A|B) = \frac{P(AB)}{P(B)}$$

贝叶斯定理之所以有用，是因为我们在生活中经常遇到这种情况：我们可以很容易直接得出 P（A|B），P（B|A）则很难直接得出，但我们更关心 P（B|A），贝叶斯定理就为我们打通从 P（A|B）获得 P（B|A）的道路。

下面不加证明地直接给出贝叶斯定理：

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

(2) 朴素贝叶斯分类

朴素贝叶斯分类是一种十分简单的分类算法，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素，朴素贝叶斯的思想基础是这样的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。通俗来说，就好比这么个道理，你在街上看到一个黑人，我问你你猜这哥们哪里来的，你十有八九猜非洲。为什么呢？因为黑人中非洲人的比率最高，当然人家也可能是美洲人或亚洲人，但在没有其它可用信息下，我们会选择条件概率最大的类别，这就是朴素贝叶斯的思想基础。

朴素贝叶斯的分类的正式定义如下：

1、设$x = \{a_1, a_2, ..., a_m\}$为一个待分类项，而每个a为x的一个特征属性。

2、有类别集合$C = \{y_1, y_2, ..., y_n\}$。

3、计算$P(y_1|x), P(y_2|x), ..., P(y_n|x)$。

4、如果$P(y_k|x) = max\{P(y_1|x), P(y_2|x), ..., P(y_n|x)\}$，则$x \in y_k$。

那么现在的关键就是如何计算第 3 步中的各个条件概率。我们可以这么做：

1. 找到一个已知分类的待分类项集合，这个集合叫做训练样本集。
2. 统计得到在各类别下各个特征属性的条件概率估计。即

$$P(a_1|y_1), P(a_2|y_1), ..., P(a_m|y_1); P(a_1|y_2), P(a_2|y_2), ..., P(a_m|y_2); ...; P(a_1|y_n), P(a_2|y_n), ..., P(a_m|y_n)$$

3. 如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

$$P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$$

因为分母对于所有类别为常数，因此我们只要将分子最大化即可。又因为各特征属性是条件独立的，所以有：

$$P(x|y_i)P(y_i) = P(a_1|y_i)P(a_2|y_i)...P(a_m|y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j|y_i)$$

下面讨论 P（a|y）的估计。

由上文看出，计算各个划分的条件概率 P（a|y）是朴素贝叶斯分类的关键性步骤，当特征属性为离散值时，只要很方便的统计训练样本中各个划分在每个类别中出现的频率即可用来估计 P（a|y）。下面重点讨论特征值是连续值的情况。

当特征属性为连续值时，通常假定其值服从高斯分布（也称正态分布）。即：

$$g(x, \eta, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\eta)^2}{2\sigma^2}}$$

而 $P(a_k|y_i) = g(a_k, \eta_{y_i}, \sigma_{y_i})$

因此只要计算出训练样本中各个类别中此特征项划分的各均值和标准差，代入上述公式即可得到需要的估计值。均值与标准差的计算在此不再赘述。

另一个需要讨论的问题就是当 P（a|y）=0 怎么办，当某个类别下某个特征项划分没有出现时，就会产生这种现象，这会令分类器质量大大降低。为了解决这个问题，我们引入 Laplace 校准，它的思想非常简单，就是对没类别下所有划分的计数加 1，这样如果训练样本集数量充分大时，并不会对结果产生影响，并且解决了上述频率为 0 的尴尬局面。

4. 决策树

决策树（Decision Tree）是一种简单但是广泛使用的分类器。通过训练数据构建决策树，可以高效的对未知的数据进行分类。决策数有两大优点：1）决策树模型可以读性好，具有描述性，有助于人工分析；2）效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

4.1 决策树的构造

不同于贝叶斯算法，决策树的构造过程不依赖领域知识，它使用属性选择度量来选择将元组最好地划分成不同的类的属性。所谓决策树的构造就是进行属性选择度量确定各个特征属性之间的拓扑结构。

构造决策树的关键步骤是分裂属性。所谓分裂属性就是在某个节点处按照某一特征属性的不同划分构造不同的分支，其目标是让各个分裂子集尽可能地“纯”。尽可能“纯”就是尽量让一个分裂子集中待分类项属于同一类别。分裂属性分为三种不同的情况：

1、属性是离散值且不要求生成二叉决策树。此时用属性的每一个划分作为一个分支。

2、属性是离散值且要求生成二叉决策树。此时使用属性划分的一个子集进行测试，按照“属于此子集”和“不属于此子集”分成两个分支。

3、属性是连续值。此时确定一个值作为分裂点 split_point，按照>split_point 和 <=split_point 生成两个分支。

构造决策树的关键性内容是进行属性选择度量，属性选择度量是一种选择分裂准则，是将给定的类标记的训练集合的数据划分 D“最好”地分成个体类的启发式方法，它决定了拓扑结构及分裂点 split_point 的选择。

属性选择度量算法有很多，一般使用自顶向下递归分治法，并采用不回溯的贪心策略。这里介绍 ID3 和 C4.5 两种常用算法。

4.2 ID3 算法

从信息论知识中我们知道，期望信息越小，信息增益越大，从而纯度越高。所以 ID3 算法的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。下面先定义几个要用到的概念。

设 D 为用类别对训练元组进行的划分，则 D 的熵（entrophy）表示为：

$$info(D) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

其中 pi 表示第 i 个类别在整个训练元组中出现的频率，可以用属于此类别元素的数量除以训练元组元素总数量作为估计。熵的实际意义表示是 D 中元组的类标号所需要的平均信息量。

现在我们假设将训练元组 D 按属性 A 进行划分，则 A 对 D 划分的期望信息为：

$$info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} info(D_j)$$

而信息增益即为两者的差值：

$$gain(A) = info(D) - info_A(D)$$

ID3 算法就是在每次需要分裂时，计算每个属性的增益率，然后选择增益率最大的属性进行分裂。下面我们继续用 SNS 社区中不真实账号检测的例子说明如何使用 ID3 算法构造决策树。为了简单起见，我们假设训练集合包含 10 个元素：

日志密度	好友密度	是否使用真实头像	账号是否真实
s	s	no	no
s	l	yes	yes
l	m	yes	yes
m	m	yes	yes
l	m	yes	yes
m	l	no	yes
m	s	no	no
l	m	no	yes
m	s	no	yes
s	s	yes	no

其中 s、m 和 I 分别表示小、中和大。

设 L、F、H 和 R 表示日志密度、好友密度、是否使用真实头像和账号是否真实，下面计算各属性的信息增益。

$$info(D) = -0.7log_20.7 - 0.3log_20.3 = 0.7 * 0.51 + 0.3 * 1.74 = 0.879$$

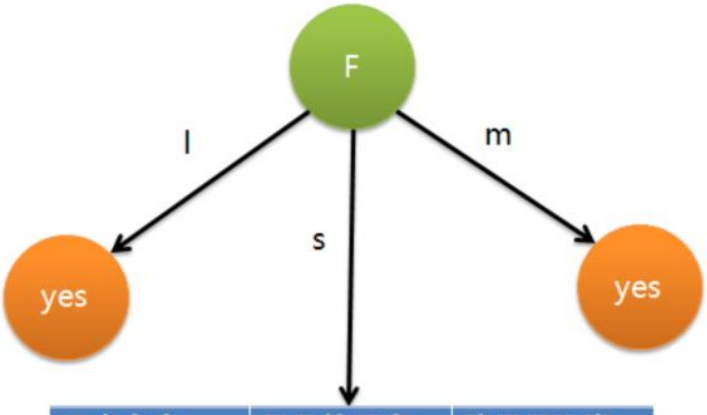
$$info_L(D) = 0.3 * (-\frac{0}{3}log_2\frac{0}{3} - \frac{3}{3}log_2\frac{3}{3}) + 0.4 * (-\frac{1}{4}log_2\frac{1}{4} - \frac{3}{4}log_2\frac{3}{4}) + 0.3 * (-\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3}) = 0 + 0.326 + 0.277 = 0.603$$

$$gain(L) = 0.879 - 0.603 = 0.276$$

因此日志密度的信息增益是 0.276。

用同样的方法得到 H 和 F 的信息增益分别是 0.033 和 0.553。

因为 F 具有最大的信息增益，所以第一次分裂选择 F 为分裂属性，分裂后的结果如下图所示：



日志密度	是否使用真实头像	账号是否真实
s	no	no
m	no	no
m	no	yes
s	yes	no

在上图的基础上，再递归使用这个方法计算子节点的分裂属性，最终就可以得到整个决策树。

4.3 C4.5 算法

ID3 算法存在一个问题，就是偏向于多值属性，例如，如果存在唯一标识属性 ID，则 ID3 会选择它作为分裂属性，这样虽然使得划分充分纯净，但这种划分对分类几乎毫无用处。ID3 的后继算法 C4.5 使用增益率（gain ratio）的信息增益扩充，试图克服这个偏倚。

C4.5 算法首先定义了“分裂信息”，其定义可以表示成：

$$split\_info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} log_2(\frac{|D_j|}{|D|})$$

其中各符号意义与 ID3 算法相同，然后，增益率被定义为：

$$gain\_ratio(A) = \frac{gain(A)}{split\_info(A)}$$

C4.5 选择具有最大增益率的属性作为分裂属性，其具体应用与 ID3 类似，不再赘述。

4.4 量化纯度

前面讲到，决策树是根据“纯度”来构建的，如何量化纯度呢？这里介绍三种纯度计算方法。如果记录被分为 n 类，每一类的比例 P（i）=第 i 类的数目/总数目。还是拿上面的例子，10 个数据中可以偿还债务的记录比例为 P（1）=7/10=0.7，无法偿还的为 P（2）=3/10=0.3,N=2。

Gini 不纯度

$$Gini = 1 - \sum_{i=1}^{n} P(i)^2$$

熵（Entrophy）

$$Entropy = -\sum_{i=1}^{n} P(i) * \log_2^{P(i)}$$

错误率

$$Error = 1 - \max\{p(i) | \ i \text{ in } [1, n]\}$$

上面的三个公式均是值越大，表示越“不纯”，越小表示越“纯”。三种公式只需要取一种即可，实践证明三种公司的选择对最终分类准确率的影响并不大，一般使用熵公式。

纯度差，也成为信息增益（Information Gain）,公示如下：

$$\Delta = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} * I(v_j)$$

其中，I 代表不纯度（也就是上面三个公式的任意一种），K 代表分割的节点数，一般 K=2，Vj 表示子节点中的记录数目。上面公式实际上就是当前节点的不纯度减去子节点不纯度的加权平均数，权重由子节点记录数与当前节点记录数的比例决定。

4.5 停止条件

决策树的构建过程是一个递归的过程，所以需要确定停止条件，否则过程将不会结束。一种最直观的方式是当每个子节点只有一种类型的记录时停止，但是这样往往会使得树的节点过多，导致过拟合问题（Overfitting）。另一种可行的方法是当前节点中的记录数低于一个最小的阀值，那么就停止分割，将 max(P(i))对应的分类作为当前叶节点的分类。

4.6 优化方案

采用上面算法生成的决策树在事件中往往会导致过滤拟合。也就是该决策树对训练数据可以得到很低的错误率，但是运用到测试数据上却得到非常高的错误率。因此我们可以采用如下的一些优化方案：

（1）修剪枝叶

决策树过渡拟合往往是因为太过“茂盛”，也就是节点过多，所以需要裁剪（Prune Tree）枝叶。裁剪枝叶的策略对决策树正确率的影响很大。主要有两种裁剪策略。

前置裁剪

在构建决策树的过程时，提前停止。那么，会将切分节点的条件设置的很苛刻，导致决策树很短小。结果就是决策树无法达到最优。实践证明这中策略无法得到较好的结果。

后置裁剪

决策树构建好后，然后才开始裁剪。采用两种方法：1）用单一叶节点代替整个子树，叶节点的分类采用子树中最主要的分类；2）将一个子树完全替代另外一颗子树。后置裁剪有个问题就是计算效率，有些节点计算后就被裁剪了，导致有点浪费。

(2) K-Fold Cross Validation

首先计算出整体的决策树 T，叶节点个数记作 N，设 i 属于[1,N]。对每个 i，使用 K-Fold Cross Validation 方法计算决策树，并裁剪到 i 个节点，计算错误率，最后求出平均错误率。这样可以用具有最小错误率对应的 i 作为最终决策树的大小，对原始决策树进行裁剪，得到最优决策树。

(3) Random Forest

Random Forest 是用训练数据随机的计算出许多决策树，形成了一个森林。然后用这个森林对未知数据进行预测，选取投票最多的分类。实践证明，此算法的错误率得到了经一步的降低。这种方法背后的原理可以用“三个臭皮匠定一个诸葛亮”这句谚语来概括。一颗树预测正确的概率可能不高，但是集体预测正确的概率却很高。

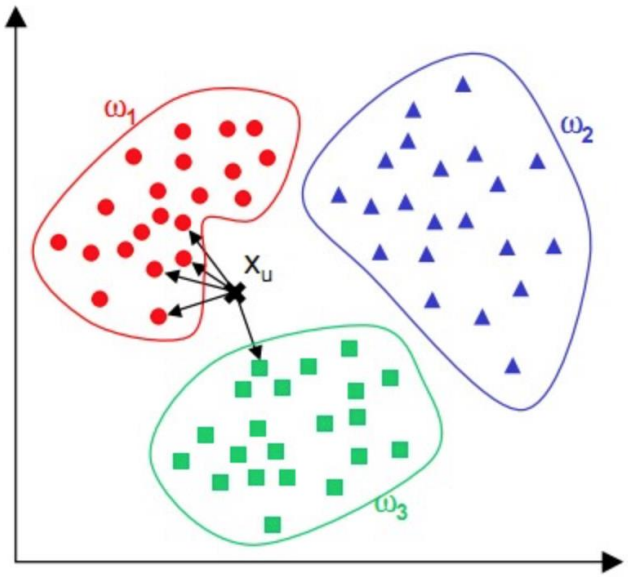
5. K 最近邻法（KNN）

K 最近邻(k-Nearest Neighbor，KNN)分类算法，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN 方法虽然从原理上也依赖于极限定理，但在类别决策时，只与极少量的相邻样本有关。由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。

KNN 算法不仅可以用于分类，还可以用于回归。通过找出一个样本的 k 个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权值(weight)，如权值与距离成正比（组合函数）。

该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。 该算法只计算“最近的”邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行结果。可以采用权值的方法（和该样本距离小的邻居权值大）来改进。该方法的另一个不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的 K 个最近邻点。目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

KNN 可以说是一种最直接的用来分类未知数据的方法。基本通过下面这张图跟文字说明就可以明白 KNN 是干什么的。



简单来说，KNN 可以看成：有那么一堆你已经知道分类的数据，然后当一个新数据进入的时候，就开始跟训练数据里的每个点求距离，然后挑离这个训练数据最近的 K 个点看看这几个点属于什么类型，然后用少数服从多数的原则，给新数据归类。

算法步骤：

step.1---初始化距离为最大值

step.2---计算未知样本和每个训练样本的距离dist

step.3---得到目前K个最临近样本中的最大距离maxdist

step.4---如果dist小于maxdist，则将该训练样本作为K-最近邻样本

step.5---重复步骤2、3、4，直到未知样本和所有训练样本的距离都算完

step.6---统计K-最近邻样本中每个类标号出现的次数

step.7---选择出现频率最大的类标号作为未知样本的类标号

6. 支持向量机（SVM）

6.1 简介

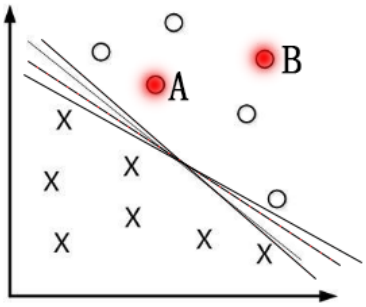
支持向量机（support vector machines）是一种二分类模型，它的目的是寻找一个超平面来对样本进行分割，分割的原则是间隔最大化，最终转化为一个凸二次规划问题来求解。由简至繁的模型包括：

- 当训练样本线性可分时，通过硬间隔最大化，学习一个线性可分支持向量机；
- 当训练样本近似线性可分时，通过软间隔最大化，学习一个线性支持向量机；
- 当训练样本线性不可分时，通过核技巧和软间隔最大化，学习一个非线性支持向量机；

6.2 线性可分支持向量机

间隔最大化和支持向量

如果一个线性函数能够将样本分开，称这些数据样本是线性可分的。那么什么是线性函数呢？其实很简单，在二维空间中就是一条直线，在三维空间中就是一个平面，以此类推，如果不考虑空间维数，这样的线性函数统称为**超平面**。我们看一个简单的二维空间的例子，O 代表正类，X 代表负类，样本是线性可分的，但是很显然不只有这一条直线可以将样本分开，而是有无数条，我们所说的线性可分支持向量机就对应着能将数据正确划分并且**间隔最大**的直线。



那么我们考虑第一个问题，**为什么要间隔最大呢？**一般来说，一个点距离分离超平面的远近可以表示分类预测的确信度，如图中的 A B 两个样本点，B 点被预测为正类的确信度要大于 A 点，所以 SVM 的目标是寻找一个超平面，使得离超平面较近的异类点之间能有更大的间隔，即不必考虑所有样本点，只需让求得的超平面使得离它近的点间隔最大。
接下来考虑第二个问题，**怎么计算间隔？**只有计算出了间隔，才能使得间隔最大化。在样本空间中，划分超平面可通过如下线性方程来描述：

$$w^T x + b = 0 \qquad (1)$$

其中 w 为法向量，决定了超平面的方向，b 为位移量，决定了超平面与原点的距离。假设超平面能将训练样本正确地分类，即对于训练样本(xi,yi)，满足以下公式：

$$\begin{cases} w^T x_i + b \geq +1, y_i = +1 \\ w^T x_i + b \leq -1, y_i = -1 \end{cases} \qquad (2)$$

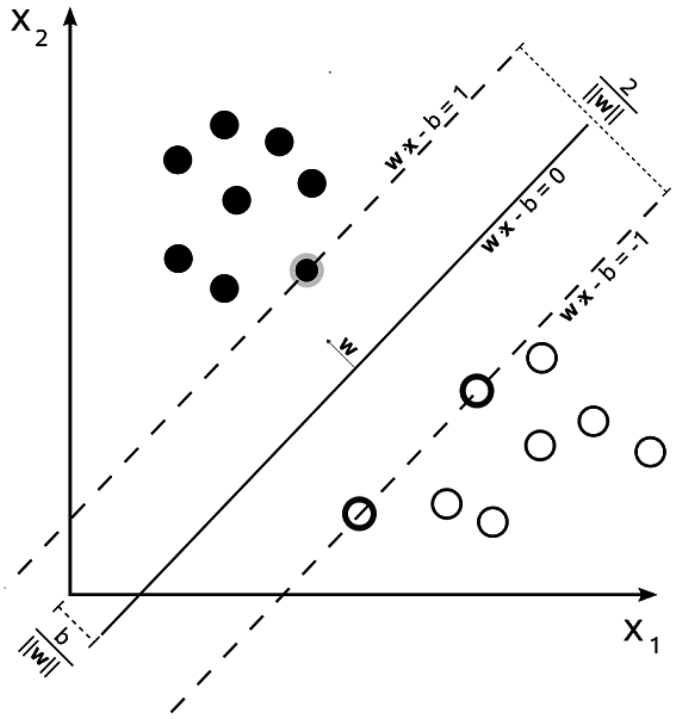
公式（2）称为最大间隔假设，yi=+1 表示样本为正样本，yi=−1 表示样本为负样本，式子前面选择大于等于+1，小于等于-1 只是为了计算方便，原则上可以是任意常数，但无论是多少，都可以通过对 w 的变换使其为 +1 和 -1 ，此时将公式（2）左右都乘以 yi，得到如下：

$$\begin{cases} y_i(w^T x_i + b) \geq (+1) * (+1), y_i = +1 \\ y_i(w^T x_i + b) \geq (-1) * (-1), y_i = -1 \end{cases}$$

实际上等价于：

$$y_i(w^T x_i + b) \geq 1 \qquad （3）$$

训练集中的所有样本都应满足公式（3）。如下图所示，距离超平面最近的这几个样本点满足 $y_i(w^T x_i + b) = 1$ ，它们被称为**“支持向量”**。虚线称为边界，两条虚线间的距离称为**间隔（margin）**。



下面我们开始计算间隔，其实间隔就等于两个异类支持向量的差在 w 上的投影，即：

$$\gamma = \frac{(\vec{x}_+ - \vec{x}_-)\cdot \vec{w}^T}{\|\mathrm{w}\|} = \frac{\vec{x}_+ \cdot \vec{w}^T - \vec{x}_- \cdot \vec{w}^T}{\|\mathrm{w}\|} \qquad (4)$$

其中 $\vec{x}_+$ 和 $\vec{x}_-$ 分别表示两个正负支持向量，因为 $\vec{x}_+$ 和 $\vec{x}_-$ 满足 $y_i(w^T x_i + b) = 1$，即：

$$\begin{cases} 1*(w^T x_+ + b) = 1, y_i = +1 \\ -1*(w^T x_- + b) = 1, y_i = -1 \end{cases},$$

推出：

$$\begin{cases} w^T x_+ = 1 - b \\ w^T x_- = -1 - b \end{cases},$$

代入公式（4）中可以得到：

$$\gamma = \frac{1 - b + (1 + b)}{\|\mathrm{w}\|} = \frac{2}{\|\mathrm{w}\|} \qquad (5)$$

至此，我们求得了间隔，SVM 的思想是使得间隔最大化，也就是：

$$\max_{w,b} \frac{2}{\|\mathrm{w}\|}, s.t. y_i(w^T x_i + b) \geq 1 \quad (i = 1, 2, ..., m) \qquad (6)$$

显然，最大化 $\frac{2}{||w||}$ 相当于最小化 ||w||，为了计算方便，将公式（6）转化成如下：

$$\min_{w,b} \frac{1}{2} \|\mathrm{w}\|^2, s.t. y_i(w^T x_i + b) \geq 1 \quad (i = 1, 2, ..., m) \qquad (7)$$

公式（7）即为支持向量机的基本型。

对偶问题

公式（7）本身是一个凸二次规划问题，可以使用现有的优化计算包来计算，但我们选择更为高效的方法。对公式（7）使用拉格朗日乘子法得到其对偶问题，该问题的拉格朗日函数可以写为：

$$L(w,b,\alpha)=\frac{1}{2}\|\mathrm{w}\|^2+\sum_{i=1}^{m}\alpha_i(1-y_i(w^Tx_i+b)) \qquad (8)$$

公式（8）分别对 w 和 b 求偏导：

$$\begin{cases}\dfrac{\partial L}{\partial w}=w-\sum_{i=1}^{m}\alpha_i y_i x_i\\ \dfrac{\partial L}{\partial b}=\sum_{i=1}^{m}\alpha_i y_i\end{cases}$$

令其分别为 0，可以得到：

$$\begin{cases}w=\sum_{i=1}^{m}\alpha_i y_i x_i & (9)\\ \sum_{i=1}^{m}\alpha_i y_i=0 & (10)\end{cases}$$

将公式（9）（10）代入公式（8），可得：

$$\begin{aligned}&L(w,b,\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i x_j\\ &s.t.\quad \sum_{i=1}^{m}\alpha_i y_i=0,\quad \alpha_i\geq 0,\quad i=1,2,..,m\end{aligned} \qquad (11)$$

此时，原问题就转化为以下仅关于 α 的问题：

$$\begin{aligned}&\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i x_j\\ &s.t.\quad \sum_{i=1}^{m}\alpha_i y_i=0,\quad \alpha_i\geq 0,\quad i=1,2,...,m\end{aligned} \qquad (12)$$

解出 α 之后，根据公式（9）可以求得 w ， 进而求得 b，可以得到模型：

$$f(x)=w^Tx+b=\sum_{i=1}^{m}\alpha_i y_i x_i^T x+b \qquad (13)$$

上述过程的 KKT 条件为：

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

我们分析一下，对于任意的训练样本 (xi,yi)，

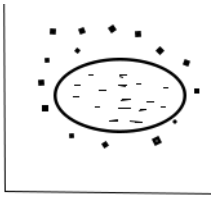
若 αi=0，则其不会在公式（13）中的求和项中出现，也就是说，它不影响模型的训练；

若 αi>0，则 $y_i f(x_i)$ −1=0，也就是 $y_i f(x_i)$ =1，即该样本一定在边界上，是一个支持向量。

这里显示出了支持向量机的重要特征：当训练完成后，大部分样本都不需要保留，最终模型只与支持向量有关。

6.3 非线性支持向量机和核函数

对于非线性问题，线性可分支持向量机并不能有效解决，要使用非线性模型才能很好地分类。先看一个例子，如下图，很显然使用直线并不能将两类样本分开，但是可以使用一条椭圆曲线（非线性模型）将它们分开。非线性问题往往不好求解，所以希望能用解线性分类问题的方法求解，因此可以采用非线性变换，将非线性问题变换成线性问题。



对于这样的问题，可以将训练样本从原始空间映射到一个更高维的空间，使得样本在这个空间中线性可分，如果原始空间维数是有限的，即属性是有限的，那么一定存在一个高维特征空间是样本可分。令 ϕ(x)表示将 x 映射后的特征向量，于是在特征空间中，划分超平面所对应的的模型可表示为：

$$f(x)=w^T\phi(x)+b \qquad (14)$$

于是有最小化函数：

$$\min_{w,b} \frac{1}{2}\|w\|^2, \quad s.t. \quad y_i(w^T\phi(x_i)+b) \geq 1 \ \ (i=1,2,...,m) \qquad (15)$$

其对偶问题为：

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \\ &s.t. \ \sum_{i=1}^{m} \alpha_i y_i = 0, \ \alpha_i \geq 0, \quad i = 1,2,...,m \end{aligned} \tag{16}$$

若要对公式（16）求解，会涉及到计算 ϕ(xi)Tϕ(xj)，这是样本 xi 和 xj 映射到特征空间之后的内积，由于特征空间的维数可能很高，甚至是无穷维，因此直接计算 ϕ(xi)Tϕ(xj)通常是困难的，于是想到这样一个函数：

$$\kappa(x_i, x_j) = < \phi(x_i), \phi(x_j) > = \phi(x_i)^T \phi(x_j) \tag{17}$$

即 xi 和 xj 在特征空间中的内积等于他们在原始样本空间中通过函数 κ(xi,xj)计算的函数值，于是公式（16）写成如下：

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa(x_i, x_j) \\ &s.t. \ \sum_{i=1}^{m} \alpha_i y_i = 0, \ \alpha_i \geq 0, \quad i = 1,2,...,m \end{aligned} \tag{18}$$

求解后得到：

$$\begin{aligned} f(x) &= w^T \phi(x) + b \\ &= \sum_{i=1}^{m} \alpha_i y_i \phi(x_i)^T \phi(x_j) + b \\ &= \sum_{i=1}^{m} \alpha_i y_i \kappa(x_i, x_j) + b \end{aligned} \tag{19}$$

这里的函数 κ(xi,xj)就是核函数，在实际应用中，通常人们会从一些常用的核函数里选择（根据样本数据的不同，选择不同的参数，实际上就得到了不同的核函数），下面给出常用的核函数：

- 线性核：

$$\kappa(x_i, x_j) = x_i^T x_j$$

- 多项式核（d 是多项式的次数，d=1 是退化为线性核）：

$$\kappa(x_i, x_j) = (x_i^T x_j)^d$$

- 高斯核（σ>0）：

$$\kappa(x_i, x_j) = \exp(-\frac{\| x_i - x_j \|^2}{2\sigma^2})$$

- 拉普拉斯核（σ>0）：

$$\kappa(x_i, x_j) = \exp(-\frac{\| x_i - x_j \|}{\sigma})$$

- sigmiod 核（β>0,θ>0）：

$$\kappa(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$$

- 此外，核函数也可以通过组合得到，在此不再赘述。

6.4 线性支持向量机（软间隔支持向量机）与松弛变量

线性支持向量机

在前面的讨论中，我们假设训练样本在样本空间或者特征空间中是线性可分的，但在现实任务中往往很难确定合适的核函数使训练集在特征空间中线性可分，退一步说，即使瞧好找到了这样的核函数使得样本在特征空间中线性可分，也很难判断是不是由于过拟合造成。

线性不可分意味着某些样本点 (xi,yi)不能满足间隔大于等于 1 的条件，样本点落在超平面与边界之间。为解决这一问题，可以对每个样本点引入一个松弛变量 ξi≥0，使得间隔加上松弛变量大于等于 1，这样约束条件变为：

$$y_i(w^T x_i + b) \geq 1 - \xi_i \qquad (20)$$

同时，对于每一个松弛变量 ξi≥0，支付一个代价 ξi≥0，目标函数变为：

$$\frac{1}{2}\| w \|^2 + C\sum_{i=1}^{m} \xi_i \qquad (21)$$

其中 C>0 为惩罚参数，C 值大时对误分类的惩罚增大， C 值小时对误分类的惩罚减小，公式（21）包含两层含义：使 $\frac{1}{2}||w||^2$ 尽量小即间隔尽量大，同时使误分类点的个数尽量小，C 是调和两者的系数。

有了公式（21），可以和线性可分支持向量机一样考虑线性支持向量机的学习过程，此时，线性支持向量机的学习问题变成如下凸二次规划问题的求解（原始问题）：

$$\min_{w,b}\frac{1}{2}\|w\|^2+C\sum_{i=1}^{m}\xi_i,\quad s.t.\ \ y_i(w^Tx_i+b)\geq 1-\xi_i\ \ (i=1,2,...,m) \tag{22}$$

对偶问题

与线性可分支持向量机的对偶问题解法一致，公式（22）的拉格朗日函数为：

$$L(w,b,\alpha,\xi,\mu)=\frac{1}{2}\|w\|^2+C\sum_{i=1}^{m}\xi_i+\sum_{i=1}^{m}\alpha_i(1-\xi_i-y_i(w^Tx_i+b))-\sum_{i=1}^{m}\mu_i\xi_i \tag{23}$$

其中 αi≥0,μi≥0 是拉格朗日乘子。

令 L(w,b,α,ξ,μ)对 w,b,ξ 的偏导数为 0 可得如下：

$$\begin{cases} w=\sum_{i=1}^{m}\alpha_i y_i x_i & (24)\\ \sum_{i=1}^{m}\alpha_i y_i=0 & (25)\\ C=\alpha_i+\mu_i & (26)\end{cases}$$

将公式（24）（25）（26）代入公式（23）得对偶问题：

$$\begin{aligned}&\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i x_j\\ &s.t.\quad \sum_{i=1}^{m}\alpha_i y_i=0,\ \alpha_i\geq 0,\ \mu_i\geq 0,\ C=\alpha_i+\mu_i,\ i=1,2,...,m\end{aligned} \tag{27}$$

解出 α 之后，根据公式（9）可以求得 w ， 进而求得 b，可以得到模型：

$$f(x)=w^Tx+b=\sum_{i=1}^{m}\alpha_i y_i x_i{}^Tx+b \qquad （13）$$

上述过程的 KKT 条件为：

$$\begin{cases}\alpha_i\geq 0\\ y_i f(x_i)\geq 1-\xi_i\\ \alpha_i(y_i f(x_i)-1+\xi_i)=0\\ \xi_i\geq 0,\mu_i\xi_i=0\end{cases} \tag{28}$$

我们分析一下，对于任意的训练样本 (xi,yi)，总有 αi=0 或者 yif(xi)−1+ξi=0 。

- 若 αi=0，则该样本不出现在公式（13）中，不影响模型。
- 若 αi>0，必有 $y_i f(x_i)$ −1+ξi=0，也就是 $y_i f(x_i)$ =1-ξi，此时该样本为支持向量。

由于 C=αi+μi（公式 26）

- 若 αi<C， 则必有 μi>0，根据公式（28）知 ξi=0 ，即该样本恰好落在最大间隔的边界上；
- 若 αi=C，则 μi=0，此时若 ξi≤1 则该样本在最大间隔内部，若 ξi>1 则样本分类错误。

6.5 总结

至此，关于 SVM 的三类问题：线性可分支持向量机与硬间隔最大化，非线性支持向量机与核函数，线性支持向量机与软间隔最大化一一介绍完毕，最后我们来做一小段总结：

我们所面对的所有的机器学算法，都是有适用范围的，或者说，我们所有的机器学习算法都是有约束的优化问题。而这些约束，就是我们在推导算法之前所做的假设。

比如：Logistics Regression，在 Logistics Regression 中，假设后验概率为 Logistics 分布；再比如：LDA 假设 fk(x)fk(x)是均值不同，方差相同的高斯分布；这些都是我们在推导算法之前所做的假设，也就是算法对数据分布的要求。

而对于 SVM 而言，它并没有对原始数据的分布做任何的假设，这就是 SVM 和 LDA、Logistics Regression 区别最大的地方。这表明 SVM 模型对数据分布的要求低，那么其适用性自然就会更广一些。如果我们事先对数据的分布没有任何的先验信息，即，不知道是什么分布，那么 SVM 无疑是比较好的选择。

但是，如果我们已经知道数据满足或者近似满足高斯分布，那么选择 LDA 得到的结果就会更准确。如果我们已经知道数据满足或者近似满足 Logistics 分布，那么选择 Logistics Regression 就会有更好的效果。

7. AdaBoost

Adaboost 是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器（弱分类器），然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。Adaboost 算法本身是通过改变数据分布来实现的，它根据每次训练集之中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值。将修改过权值的新数据集送给下层分类器进行训练，最后将每次得到的分类器最后融合起来，作为最后的决策分类器。

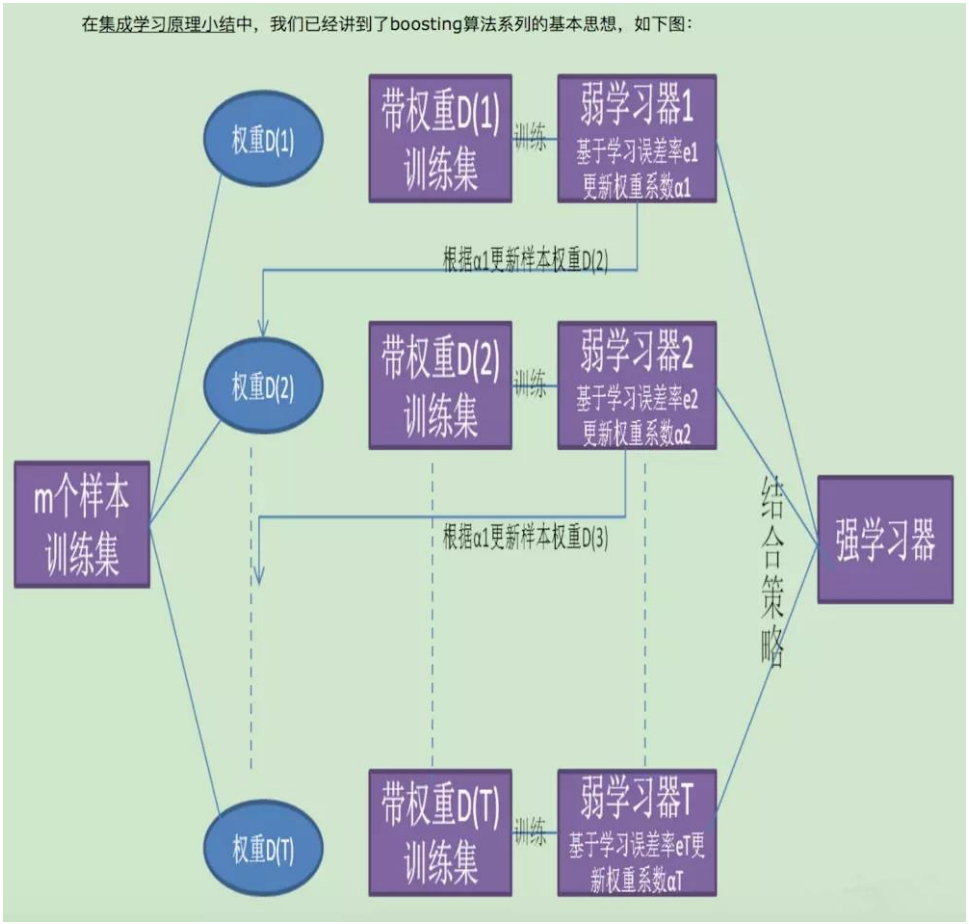
7.1 算法概述

1、先通过对 N 个训练样本的学习得到第一个弱分类器；

2、将分错的样本和其他的新数据一起构成一个新的 N 个的训练样本，通过对这个样本的学习得到第二个弱分类器；

3、将 1 和 2 都分错了的样本加上其他的新样本构成另一个新的 N 个的训练样本，通过对这个样本的学习得到第三个弱分类器

4、最终经过提升的强分类器。即某个数据被分为哪一类要由各分类器权值决定。



7.2 Adaboost 算法的基本思路

假设我们的训练样本集是

$$T = \{(x,y_1),(x_2,y_2),\ldots(x_m,y_m)\}$$

训练集的在第 k 个弱学习器的输出权重为

$$D(k) = (w_{k1}, w_{k2}, \ldots w_{km});\ \ w_{1i} = \frac{1}{m};\ \ i = 1,2\ldots m$$

首先我们看看 Adaboost 的分类问题。

分类问题的误差很好理解和计算。由于多元分类是二元分类的推广，这里我们假设是二元分类问题，输出为{-1,1}，则第 k 个弱分类器 Gk（x）在训练机上的加权误差率为

$$e_k = P(G_k(x_i) \neq y_i) = \sum_{i=1}^{m} w_{ki} I(G_k(x_i) \neq y_i)$$

接着我们看弱学习器的权重系数，对于二元分类问题，第 k 个弱分类器 Gk（x）的权重系数为

$$\alpha_k = \frac{1}{2} log \frac{1-e_k}{e_k}$$

为什么这样计算弱学习器权重系数？从上式可以看出，如果分类误差率e_k越大，则对应的弱分类器权重系数a_k越小。也就是说，误差率小的弱分类器权重系数越大。具体为什么采用这个权重系数公式，我们在讲 Adaboost 的损失函数优化时再讲。

第三个问题，更新样本权重 D。假设第 k 个弱分类器的样本集权重系数为 D（k）=（$w_{k1}, w_{k2}, \ldots\ldots w_{km}$）,则对应的第 k+1 个弱分类器的样本集权重系数为：

$$w_{k+1,i} = \frac{w_{ki}}{Z_K} exp(-\alpha_k y_i G_k(x_i))$$

这里Z_k是规范化因子

$$Z_k = \sum_{i=1}^{m} w_{ki} exp(-\alpha_k y_i G_k(x_i))$$

从$w_{k+1,i}$计算公式可以看出，如果第 i 个样本分类错误，则$y_i G_k(x_i) < 0$，导致样本的权重在第 k+1 个弱分类器中增大，如果分类正确，则权重在第 k+1 个弱分类器中减少，具体为什么采用样本权重更新公式，我们在讲 Adaboost 的损失函数优化时再讲。

最后一个问题是集合策略。Adaboost 分类采用的是加权平均法，最终的强分类器为

$$f(x) = sign(\sum_{k=1}^{K} \alpha_k G_k(x))$$

接着我们看看 Adaboost 的回归问题。由于 Adaboost 的回归问题有很多变种，这里我们以 Adaboost R2 算法为准。

我们先看看回归问题的误差率的问题，对于第 k 个弱学习器，计算他在训练集上的最大误差

$$E_k = max|y_i - G_k(x_i)| \; i = 1, 2...m$$

然后计算每个样本的相对误差

$$e_{ki} = \frac{|y_i - G_k(x_i)|}{E_k}$$

这里是误差损失为线性时的情况，如果我们用平方误差，则$e_{ki} = \frac{(y_i - G_k(x_i))^2}{E_k^2}$，如果我们用的是指数误差，则$e_{ki} = 1 - exp\left(\frac{-y_i + G_k(x_i))}{E_k}\right)$

最终得到第 k 个弱学习器的误差率

$$e_k = \sum_{i=1}^{m} w_{ki} e_{ki}$$

我们再来看看如何得到弱学习器权重系数 α 。这里有：

$$\alpha_k = \frac{e_k}{1 - e_k}$$

对于更新样本权重 D，第 k+1 个弱学习器的样本集权重系数为

$$w_{k+1,i} = \frac{w_{ki}}{Z_k} \alpha_k^{1-e_{ki}}$$

这里Z_k是规范化因子

$$Z_k = \sum_{i=1}^{m} w_{ki} \alpha_k^{1-e_{ki}}$$

最后是结合策略，和分类问题一样，采用的也是加权平均法，最终的强回归器为

$$f(x)=\sum_{k=1}^{K}(ln\frac{1}{\alpha_k})G_k(x)$$

7.3 优缺点分析

优点：

1)Adaboost 是一种有很高精度的分类器

2)可以使用各种方法构建子分类器,Adaboost 算法提供的是框架

3)当使用简单分类器时，计算出的结果是可以理解的。而且弱分类器构造极其简单

4)简单，不用做特征筛选

5)不用担心 overfitting(过度拟合)

缺点：

1)容易受到噪声干扰，这也是大部分算法的缺点

2)训练时间过长

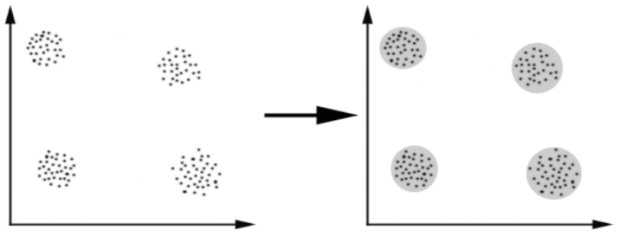
3)执行效果依赖于弱分类器的选择

第三章 非监督式学习

这章节主要介绍机器学习传统算法的非监督学习的部分。是否有监督（supervised），就看输入数据是否有标签（label）。输入数据有标签，则为有监督学习，没标签则为无监督学习。

1. K-means

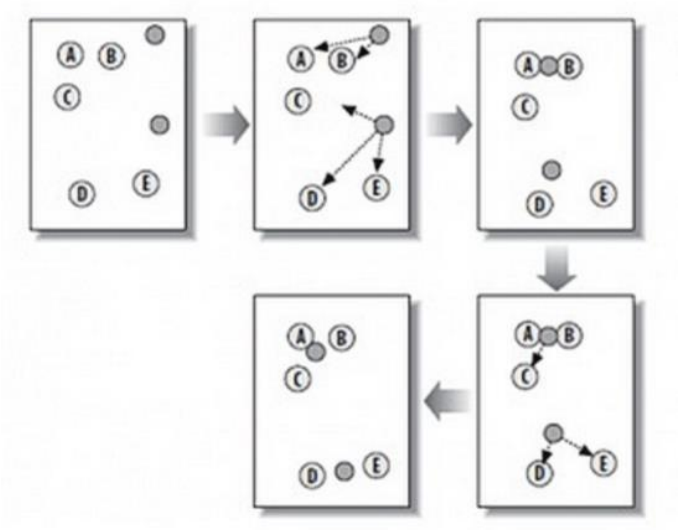
K-Means 算法主要解决的问题如下图所示。我们可以看到，在图的左边有一些点，我们用肉眼可以看出来有四个点群，但是我们怎么通过计算机程序找出这几个点群来呢？于是就出现了我们的 K-Means 算法。



K-Means要解决的问题

1.1 算法

这个算法其实很简单，如下图所示：



从上图中，我们可以看到，A、B、C、D、E 是五个在图中的点，而灰色的点是我们的种子点，也就是我们用来找点群的点。有两个种子点，所以 K=2.

然后，K-Means 的算法如下：

1. 随机在图中取 K（这里 K=2）个种子点。

2. 然后对图中的所有点求到这 K 个种子点的距离，假如点 Pi 离种子点 Si 最近，那么 Pi 属于 Si 点群。（上图中，我们可以看到 A，B 属于上面的种子点，C，D，E 属于下面中部的种子点）

3. 接下来，我们要移动种子点到属于他的“点群”的中心。（见图上的第三步）

4. 然后重复第 2）和第 3）步，直到，种子点没有移动（我们可以看到图中的第四步上面的种子点聚合了 A，B，C，下面的种子点聚合了 D，E）。

1.2 求点群中心的算法

一般来说，求点群中心点的算法你可以很简的使用各个点的 X/Y 坐标的平均值。不过，我这里想告诉大家另三个求中心点的的公式：

（1）Minikowski Distance 公式——λ可以随意取值，可以是负数，也可以是正数，或是无穷大。

$$d_{ij} = \sqrt[\lambda]{\sum_{k=1}^{n} \left| x_{ik} - x_{jk} \right|^{\lambda}}$$

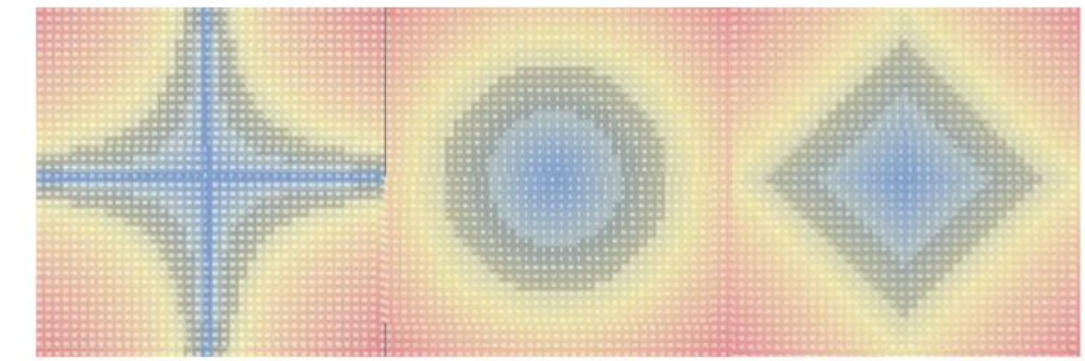
（2）Euclidean Distance 公式——也就是第一个公式λ=2 的情况

$$d_{ij} = \sqrt{\sum_{k=1}^{n} \left(x_{ik} - x_{jk} \right)^2}$$

（3）CityBlock Distance 公式——也就是第一个公式λ=1 的情况

$$d_{ij} = \sum_{k=1}^{n} \left| x_{ik} - x_{jk} \right|$$

这三个公式的求中心点有一些不一样的地方，我们看下图（对于第一个λ在 0-1 之间）：



（1）Minkowski Distance　（2）Euclidean Distance　（3）CityBlock Distance

1.3 K-means 的改进

K-Means 主要有两个最重大的缺陷——都和初始值有关：

(1) K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。（ISODATA 算法通过类的自动合并和分裂，得到较为合理的类型数目 K）。

(2) K-Means 算法需要用初始随机种子点来搞，这个随机种子点太重要，不同的随机种子点会有得到完全不同的结果。（K-Means++算法可以用来解决这个问题，其可以有效地选择初始点）。

我在这里重点说一下 K-Means++算法步骤：

先从我们的数据库随机挑个随机点当“种子点”。

对于每个点，我们都计算其和最近的一个“种子点”的距离 D(x)并保存在一个数组里，然后把这些距离加起来得到 Sum(D(x))。

然后，再取一个随机值，用权重的方式来取计算下一个“种子点”。这个算法的实现是，先取一个能落在 Sum(D(x))中的随机值 Random，然后用 Random -= D(x)，直到其<=0，此时的点就是下一个“种子点”。

重复第（2）和第（3）步直到所有的 K 个种子点都被选出来。

进行 K-Means 算法。

2. 高斯混合模型(Gaussian mixture model)

GMM 和 k-means 其实是十分相似的，区别仅仅在于对 GMM 来说，我们引入了概率。说到这里，我想先补充一点东西。统计学习的模型有两种，一种是概率模型，一种是非概率模型。所谓概率模型，就是指我们要学习的模型的形式是 P(Y|X)，这样在分类的过程中，我们通过未知数据 X 可以获得 Y 取值的一个概率分布，也就是训练后模型得到的输出不是一个具体的值，而是一系列值的概率（对应于分类问题来说，就是对应于各个不同的类的概率），然后我们可以选取概率最大的那个类作为判决对象（算软分类 soft assignment）。而非概率模型，就是指我们学习的模型是一个决策函数 Y=f(X)，输入数据 X 是多少就可以投影得到唯一的一个 Y，就是判决结果（算硬分类 hard assignment）。

回到 GMM，学习的过程就是训练出几个概率分布，所谓混合高斯模型就是指对样本的概率密度分布进行估计，而估计的模型是几个高斯模型加权之和（具体是几个要在模型训练前建立好）。每个高斯模型就代表了一个类（一个 Cluster）。对样本中的数据分别在几个高斯模型上投影，就会分别得到在各个类上的概率。然后我们可以选取概率最大的类所为判决结果。

得到概率有什么好处呢？我们知道人很聪明，就是在于我们会用各种不同的模型对观察到的事物和现象做判决和分析。当你在路上发现一条狗的时候，你可能光看外形好像邻居家的狗，又更像一点点女朋友家的狗，你很难判断，所以从外形上看，用软分类的方法，是女朋友家的狗概率 51%，是邻居家的狗的概率是 49%，属于一个易混淆的区域内，这时你可以再用其它办法进行区分到底是谁家的狗。而如果是硬分类的话，你所判断的就是女朋友家的狗，没有“多像”这个概念，所以不方便多模型的融合。

从中心极限定理的角度上看，把混合模型假设为高斯的是比较合理的，当然也可以根据实际数据定义成任何分布的 Mixture Model,不过定义为高斯的在计算上有一些方便之处，另外，理论上可以通过增加 Model 的个数，用 GMM 近似任何概率分布。

混合高斯模型的定义为：

$$p(x)=\sum_{k=1}^{K}\pi_k\,p(x\mid k)$$

其中 K 为模型的个数，πk 为第 k 个高斯的权重，则为第 k 个高斯的概率密度函数，其均值为 μk，方差为 σk。我们对此概率密度的估计就是要求 πk，μk 和 σk 各个变量。当求出表达式后，求和式的各项的结果就分别代表样本 x 属于各个类的概率。

在做参数估计的时候，常采用的方法是最大似然。最大似然法就是使样本点在估计的概率密度函数上的概率值最大。由于概率值一般都很小，N 很大的时候这个连乘的结果非常小，容易造成浮点数下溢。所以我们通常取 log，将目标改写成：

$$\max\sum_{i=1}^{N}\log p(x_i)$$

也就是最大化 log-likelyhood function，完整形式则为：

$$\max \sum_{i=1}^{N} \log(\sum_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \sigma_k))$$

一般用来做参数估计的时候，我们都是通过对待求变量进行求导来求极值，在上式中，log 函数中又有求和，你想用求导的方法算的话方程组将会非常复杂，所以我们不好考虑用该方法求解（没有闭合解）。可以采用的求解方法是 EM 算法——将求解分为两步：

第一步是假设我们知道各个高斯模型的参数（可以初始化一个，或者基于上一步迭代结果），去估计每个高斯模型的权值；

第二步是基于估计的权值，回过头再去确定高斯模型的参数。重复这两个步骤，直到波动很小，近似达到极值（注意这里是个极值不是最值，EM 算法会陷入局部最优）。

具体表达如下：

1.对于第 i 个样本 xi 来说，它由第 k 个 model 生成的概率为：

$$\varpi_i(k) = \frac{\pi_k N(x_i \mid \mu_k, \sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i \mid \mu_j, \sigma_j)}$$

在这一步，我们假设高斯模型的参数和是已知的（由上一步迭代而来或由初始值决定）。

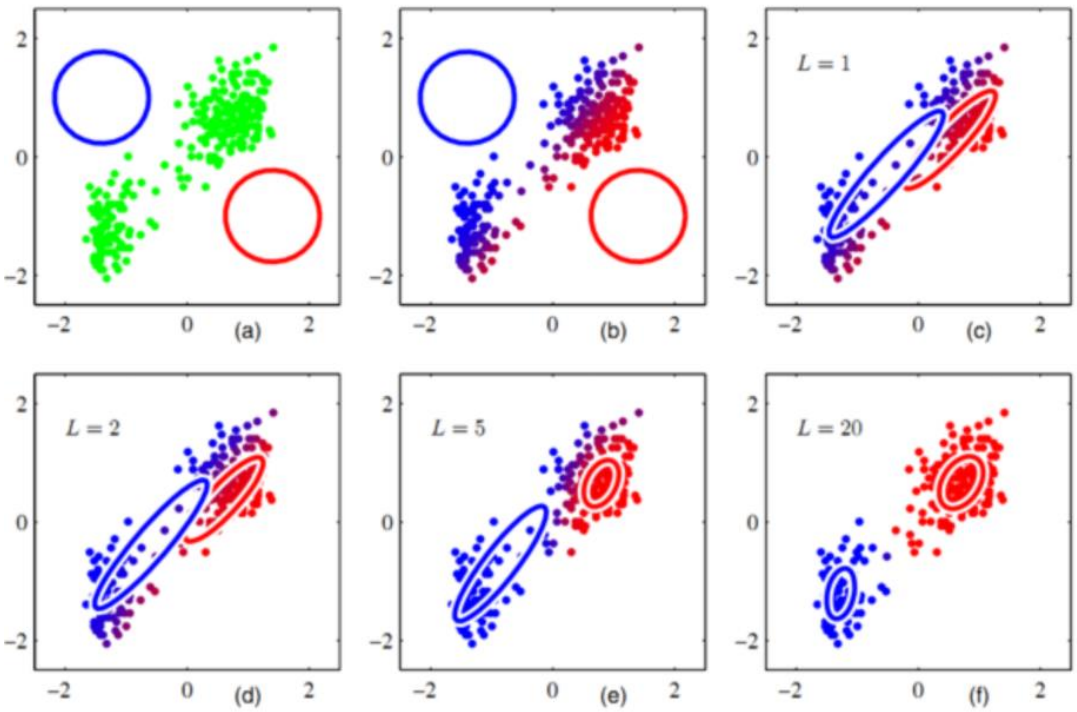
2.得到每个点的$w_i(k)$后，我们可以这样考虑：对样本x_i来说，它的$w_i(k)x_i$的值是由第 k 个高斯模型产生的。换句话说，第 k 个高斯模型产生了$w_i(k)x_i$(i=1,2,……,N)这些数据。这样在估计第 k 个高斯模型的参数时，我们就用$w_i(k)x_i$(i=1,2,……,N)这些数据去做参数估计。和前面提到的一样采用最大似然的方法去估计：

$$\mu_k = \frac{1}{N} \sum_{i=1}^{N} \varpi_i(k) x_i$$

$$\sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \varpi_i(k)(x_i - \mu_k)(x_i - \mu_k)^T$$

$$N_k = \sum_{i=1}^{N} \varpi_i(k)$$

3.重复上述两步骤直到算法收敛（这个算法一定是收敛的，至于具体的证明请回溯到 EM 算法中去）



最后总结一下，用 GMM 的优点是投影后样本点不是得到一个确定的分类标记，而是得到每个类的概率，这是一个重要信息。GMM 每一步迭代的计算量比较大，大于 k-means。GMM 的求解办法基于 EM 算法，因此有可能陷入局部极值，这和初始值的选取十分相关了。GMM 不仅可以用在聚类上，也可以用在概率密度估计上。

3. 层次聚类(Hierarchical Clustering)

不管是 GMM，还是 k-means，都面临一个问题，就是 k 的个数如何选取？比如在 bag-of-words 模型中，用 k-means 训练码书，那么应该选取多少个码字呢？为了不在这个参数的选取上花费太多时间，可以考虑层次聚类。

3.1 层次聚类算法

假设有 N 个待聚类的样本，对于层次聚类来说，基本步骤就是：

1、（初始化）把每个样本归为一类，计算每两个类之间的距离，也就是样本与样本之间的相似度；

2、寻找各个类之间最近的两个类，把他们归为一类（这样类的总数就少了一个）；

3、重新计算新生成的这个类与各个旧类之间的相似度；

4、重复 2 和 3 直到所有样本点都归为一类，结束。

整个聚类过程其实是建立了一棵树，在建立的过程中，可以通过在第二步上设置一个阈值，当最近的两个类的距离大于这个阈值，则认为迭代可以终止。另外关键的一步就是第三步，如何判断两个类之间的相似度有不少种方法。

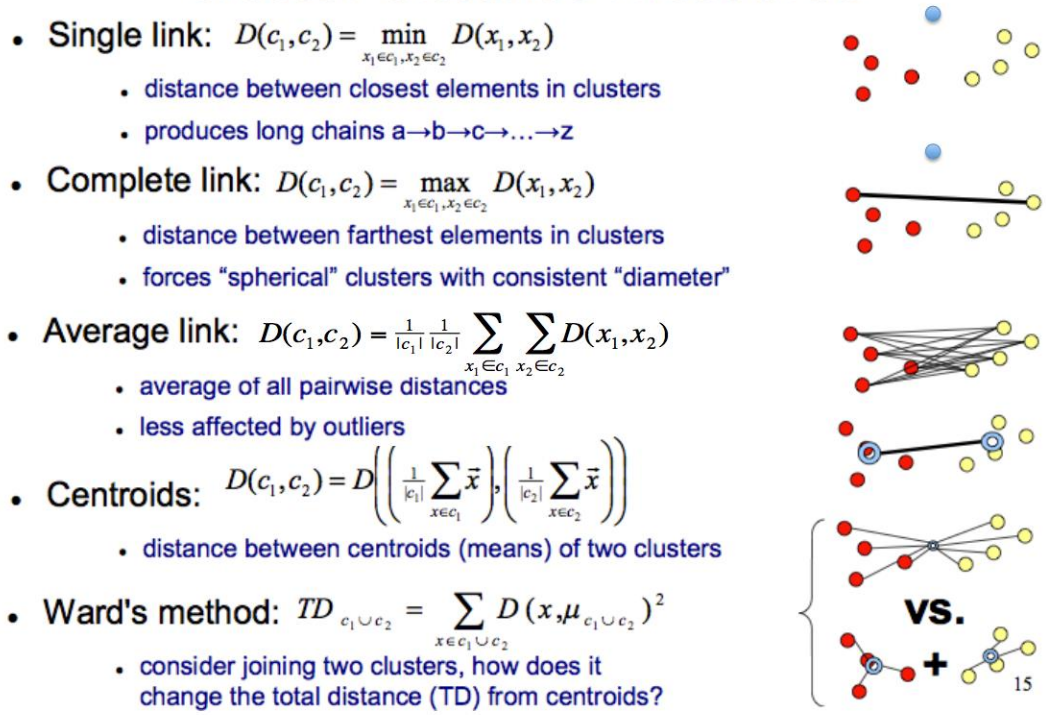
3.2 相似度计算

1. SingleLinkage：又叫做 nearest-neighbor ，就是取两个类中距离最近的两个样本的距离作为这两个集合的距离，也就是说，最近两个样本之间的距离越小，这两个类之间的相似度就越大。容易造成一种叫做 Chaining 的效果，两个 cluster 明明从“大局”上离得比较远，但是由于其中个别的点距离比较近就被合并了，并且这样合并之后 Chaining 效应会进一步扩大，最后会得到比较松散的 cluster 。

2. CompleteLinkage：这个则完全是 Single Linkage 的反面极端，取两个集合中距离最远的两个点的距离作为两个集合的距离。其效果也是刚好相反的，限制非常大，两个 cluster 即使已经很接近了，但是只要有不配合的点存在，就顽固到底，老死不相合并，也是不太好的办法。这两种相似度的定义方法的共同问题就是指考虑了某个有特点的数据，而没有考虑类内数据的整体特点。

3. Average-linkage：这种方法就是把两个集合中的点两两的距离全部放在一起求一个平均值，相对也能得到合适一点的结果。average-linkage 的一个变种就是取两两距离的中值，与取均值相比更加能够解除个别偏离样本对结果的干扰。

4. Centroid linkage： 定义类间距离为类间质心的距离，质心为类中所有成员的原始数据的均值。



3.3 自顶而下 / 自下而上

上面介绍的这种聚类的方法叫做 agglomerative hierarchical clustering（自下而上）的，描述起来比较简单，但是计算复杂度比较高，为了寻找距离最近/远和均值，都需要对所有的距离计算个遍，需要用到双重循环。另外从算法中可以看出，每次迭代都只能合并两个子类，这是非常慢的。

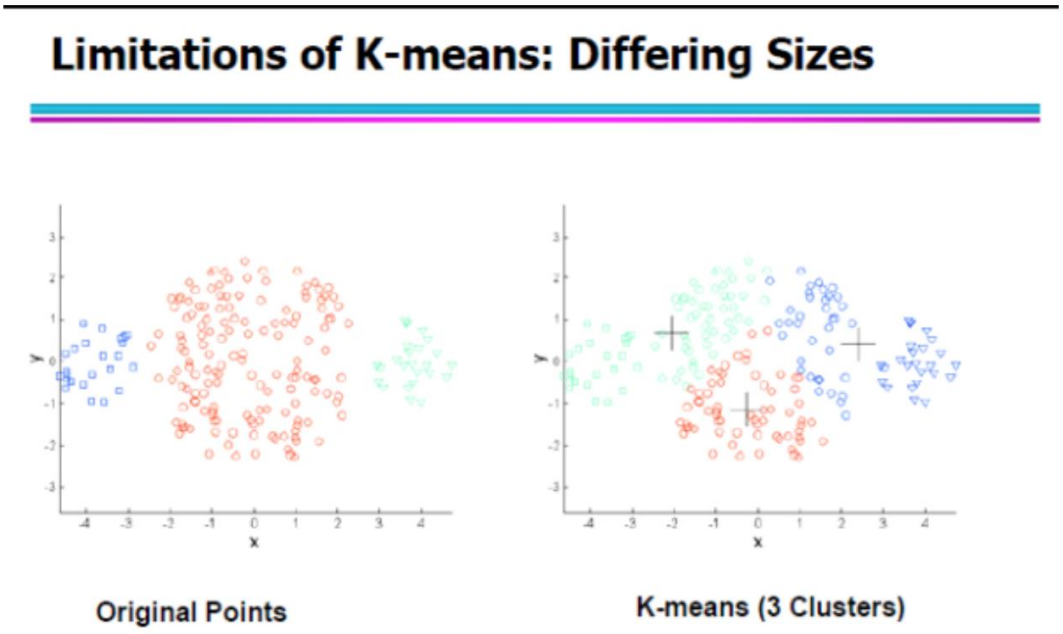
另外有一种聚类方法叫做 divisivehierarchical clustering（自顶而下），过程恰好是相反的，一开始把所有的样本都归为一类，然后逐步将他们划分为更小的单元，直到最后每个样本都成为一类。在这个迭代的过程中通过对划分过程中定义一个松散度，当松散度最小的那个类的结果都小于一个阈值，则认为划分可以终止。这种方法用的不普遍。

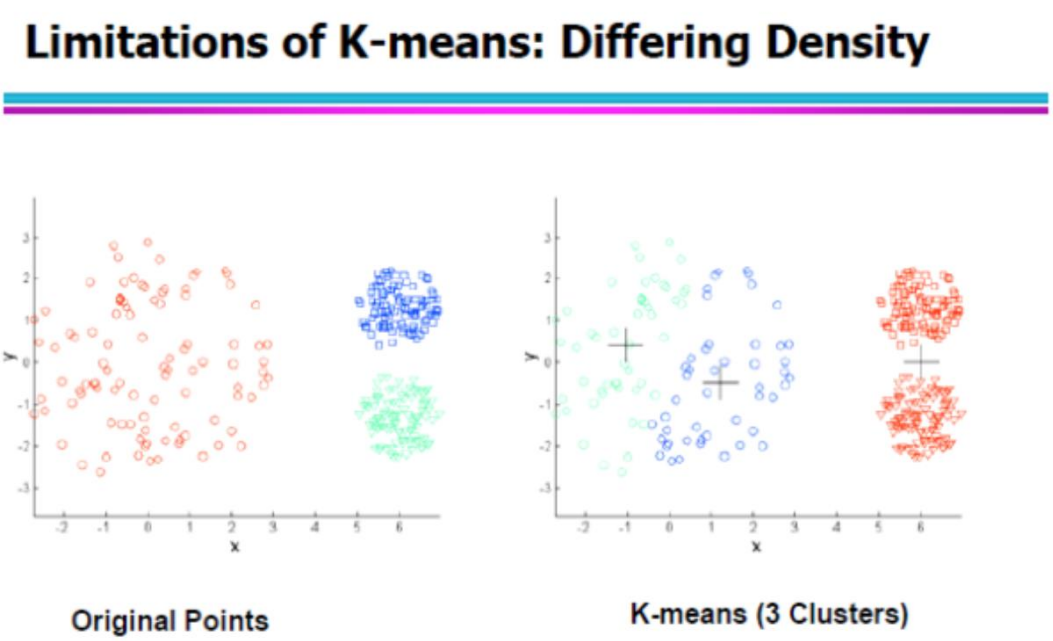
4. 三种方法对比

(1) K-means

优点：简单、时间复杂度、空间复杂度低

缺点：随机初始化的中心点对结果影响很大；hold 不住族之间的 size 或密度差别较大的情况，因为 K-means 的目标函数是距离和，导致最后必然出来一种很社会主义的结果。





(2) 层次聚类

优点：可解释性好（如当需要创建一种分类法时）；还有些研究表明这些算法能产生高质量的聚类，也会应用在上面说的先取 K 比较大的 K-means 后的合并阶段；还有对于 K-means 不能解决的非球形族就可以解决了。

Original Points　　Six Clusters

• Can handle non-elliptical shapes

缺点：时间复杂度高啊，o(m^3)，改进后的算法也有 o(m^2lgm)，m 为点的个数；贪心算法的缺点，一步错步步错；同 K-means，difficulty handling different sized clusters and convex shapes。

(3) 高斯混合模型

优点：投影后样本点不是得到一个确定的分类标记，而是得到每个类的概率，这是一个重要信息。

缺点：GMM 每一步迭代的计算量比较大，大于 k-means。GMM 的求解办法基于 EM 算法，因此有可能陷入局部极值，这和初始值的选取十分相关了。

第四章 半监督式学习与强化学习

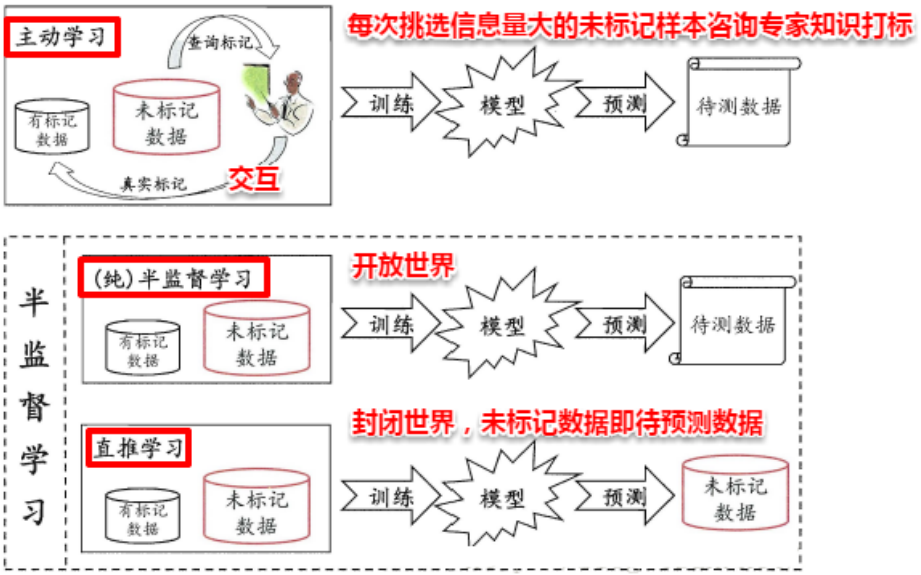
1. 半监督式学习

前面我们一直围绕的都是监督学习与无监督学习，监督学习指的是训练样本包含标记信息的学习任务，例如：常见的分类与回归算法；无监督学习则是训练样本不包含标记信息的学习任务，例如：聚类算法。在实际生活中，常常会出现一部分样本有标记和较多样本无标记的情形，例如：做网页推荐时需要让用户标记出感兴趣的网页，但是少有用户愿意花时间来提供标记。若直接丢弃掉无标记样本集，使用传统的监督学习方法，常常会由于训练样本的不充足，使得其刻画总体分布的能力减弱，从而影响了学习器泛化性能。那如何利用未标记的样本数据呢？

一种简单的做法是通过专家知识对这些未标记的样本进行打标，但随之而来的就是巨大的人力耗费。若我们先使用有标记的样本数据集训练出一个学习器，再基于该学习器对未标记的样本进行预测，从中挑选出不确定性高或分类置信度低的样本来咨询专家并进行打标，最后使用扩充后的训练集重新训练学习器，这样便能大幅度降低标记成本，这便是主动学习（active learning），其目标是使用尽量少的/有价值的咨询来获得更好的性能。

显然，主动学习需要与外界进行交互/查询/打标，其本质上仍然属于一种监督学习。事实上，无标记样本虽未包含标记信息，但它们与有标记样本一样都是从总体中独立同分布采样得到，因此它们所包含的数据分布信息对学习器的训练大有裨益。如何让学习过程不依赖外界的咨询交互，自动利用未标记样本所包含的分布信息的方法便是半监督学习（semi-supervised learning），即训练集同时包含有标记样本数据和未标记样本数据。

此外，半监督学习还可以进一步划分为纯半监督学习和直推学习，两者的区别在于：前者假定训练数据集中的未标记数据并非待预测数据，而后者假定学习过程中的未标记数据就是待预测数据。主动学习、纯半监督学习以及直推学习三者的概念如下图所示：



1.1 生成式方法

生成式方法（generative methods）是基于生成式模型的方法，即先对联合分布 P（x,c）建模，从而进一步求解 P（c | x），**此类方法假定样本数据服从一个潜在的分布，因此需要充分可靠的先验知识**。例如：前面已经接触到的贝叶斯分类器与高斯混合聚类，都属于生成式模型。现假定总体是一个高斯混合分布，即由多个高斯分布组合形成，从而一个子高斯分布就代表一个类簇（类别）。高斯混合分布的概率密度函数如下所示：

$$p(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \cdot p(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

混合系数 均值向量 协方差矩阵

不失一般性，假设类簇与真实的类别按照顺序一一对应，即第 i 个类簇对应第 i 个高斯混合成分。与高斯混合聚类类似地，这里的主要任务也是估计出各个高斯混合成分的参数以及混合系数，不同的是：对于有标记样本，不再是可能属于每一个类簇，而是只能属于真实类标对应的特定类簇。

$$LL(D_l \cup D_u) = \sum_{(\boldsymbol{x}_j, y_j) \in D_l} \ln \left(\sum_{i=1}^{N} \alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \, p(y_j \mid \Theta = i, \boldsymbol{x}_j) \right) + \sum_{\boldsymbol{x}_j \in D_u} \ln \left(\sum_{i=1}^{N} \alpha_i \cdot p(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right)$$

当且仅当i=j时取1

有类标样本只在特定类簇中出现

无类标样本可能在所有类簇中出现

直观上来看，**基于半监督的高斯混合模型有机地整合了贝叶斯分类器与高斯混合聚类的核心思想**，有效地利用了未标记样本数据隐含的分布信息，从而使得参数的估计更加准确。同样地，这里也要召唤出之前的 EM 大法进行求解，首先对各个高斯混合成分的参数及混合系数进行随机初始化，计算出各个 PM（即 γji，第 i 个样本属于 j 类，有标记样本则直接属于特定类），再最大化似然函数（即 LL（D）分别对 α、u 和∑求偏导 ），对参数进行迭代更新。

$$\boldsymbol{\mu}_i = \frac{1}{\sum_{\boldsymbol{x}_j \in D_u} \gamma_{ji} + l_i} \left(\sum_{\boldsymbol{x}_j \in D_u} \gamma_{ji} \boldsymbol{x}_j + \sum_{(\boldsymbol{x}_j, y_j) \in D_l \wedge y_j = i} \boldsymbol{x}_j \right)$$

li指的是第i类有标记样本数目

$$\boldsymbol{\Sigma}_i = \frac{1}{\sum_{\boldsymbol{x}_j \in D_u} \gamma_{ji} + l_i} \left(\sum_{\boldsymbol{x}_j \in D_u} \gamma_{ji} (\boldsymbol{x}_j - \boldsymbol{\mu}_i)(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^{\mathrm{T}} + \sum_{(\boldsymbol{x}_j, y_j) \in D_l \wedge y_j = i} (\boldsymbol{x}_j - \boldsymbol{\mu}_i)(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^{\mathrm{T}} \right)$$

$$\alpha_i = \frac{1}{m} \left(\sum_{\boldsymbol{x}_j \in D_u} \gamma_{ji} + l_i \right)$$

当参数迭代更新收敛后，对于待预测样本 x，便可以像贝叶斯分类器那样计算出样本属于每个类簇的后验概率，接着找出概率最大的即可：

$$p(\Theta = i \mid \boldsymbol{x}) = \frac{\alpha_i \cdot p(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{i=1}^{N} \alpha_i \cdot p(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}$$

$$f(\boldsymbol{x}) = \arg\max_{j \in \mathcal{Y}} \sum_{i=1}^{N} p(y = j \mid \Theta = i, \boldsymbol{x}) \cdot p(\Theta = i \mid \boldsymbol{x})$$

可以看出：基于生成式模型的方法十分依赖于对潜在数据分布的假设，即假设的分布要能和真实分布相吻合，否则利用未标记的样本数据反倒会在错误的道路上渐行渐远，从而降低学习器的泛化性能。因此，**此类方法要求极强的领域知识和掐指观天的本领**。

1.2 半监督SVM

监督学习中的 SVM 试图找到一个划分超平面，使得两侧支持向量之间的间隔最大，即“**最大划分间隔**”思想。对于半监督学习，S3VM 则考虑超平面需穿过数据低密度的区域。TSVM 是半监督支持向量机中的最著名代表，其核心思想是：尝试为未标记样本找到合适的标记指派，使得超平面划分后的间隔最大化。TSVM 采用局部搜索的策略来进行迭代求解，即首先使用有标记样本集训练出一个初始 SVM，接着使用该学习器对未标记样本进行打标，这样所有样本都有了标记，并基于这些有标记的样本重新训练 SVM，之后再寻找易出错样本不断调整。整个算法流程如下所示：

$$\min_{\boldsymbol{w},b,\hat{\boldsymbol{y}},\boldsymbol{\xi}} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C_l\sum_{i=1}^{l}\xi_i + C_u\sum_{i=l+1}^{m}\xi_i$$

松弛变量 hinge损失

$$\begin{aligned}
\text{s.t.}\ \ & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geqslant 1-\xi_i, \ \ i = 1,2,\ldots,l,\\
& \hat{y}_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geqslant 1-\xi_i, \ \ i = l+1,l+2,\ldots,m,\\
& \xi_i \geqslant 0, \ \ i = 1,2,\ldots,m,
\end{aligned}$$

输入： 有标记样本集 $D_l = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_l, y_l)\}$;
未标记样本集 $D_u = \{\boldsymbol{x}_{l+1}, \boldsymbol{x}_{l+2}, \ldots, \boldsymbol{x}_{l+u}\}$;
折中参数 C_l, C_u.

过程：

用 D_l 训练一个 SVM_l; 初始SVM

用 SVM_l 对 D_u 中样本进行预测，得到 $\hat{\boldsymbol{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \ldots, \hat{y}_{l+u})$; 伪标记

初始化 $C_u \ll C_l$;

while $C_u < C_l$ **do**

  基于 $D_l, D_u, \hat{\boldsymbol{y}}, C_l, C_u$ 求解式(13.9)，得到 $(\boldsymbol{w}, b)$, $\boldsymbol{\xi}$;

  while $\exists\{i,j \mid (\hat{y}_i\hat{y}_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2)\}$ **do**

    $\hat{y}_i = -\hat{y}_i$;

    $\hat{y}_j = -\hat{y}_j$;

    松弛变量越大表示离超平面越近，越容易分错

    基于 $D_l, D_u, \hat{\boldsymbol{y}}, C_l, C_u$ 重新求解式(13.9)，得到 $(\boldsymbol{w}, b)$, $\boldsymbol{\xi}$

  end while

  $C_u = \min\{2C_u, C_l\}$ 逐渐增大Cu

end while

输出： 未标记样本的预测结果: $\hat{\boldsymbol{y}} = (\hat{y}_{l+1}, \hat{y}_{l+2}, \ldots \hat{y}_{l+u})$ 最终调整后的结果

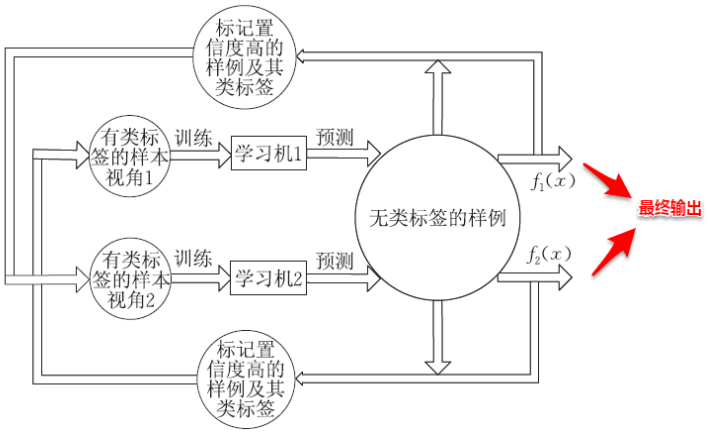
1.3 基于分歧的方法

基于分歧的方法通过多个学习器之间的**分歧（disagreement）/多样性（diversity）**来利用未标记样本数据，协同训练就是其中的一种经典方法。**协同训练最初是针对于多视图（multi-view）数据而设计的，多视图数据指的是样本对象具有多个属性集，每个属性集则对应一个试图**。例如：电影数据中就包含画面类属性和声音类属性，这样画面类属性的集合就对应着一个视图。首先引入两个关于视图的重要性质：

相容性：即使用单个视图数据训练出的学习器的输出空间是一致的。例如都是{好，坏}、{+1,-1}等。

互补性：即不同视图所提供的信息是互补/相辅相成的，实质上这里体现的就是集成学习的思想。

协同训练正是很好地利用了多视图数据的“**相容互补性**”，其基本的思想是：首先基于有标记样本数据在每个视图上都训练一个初始分类器，然后让每个分类器去挑选分类置信度最高的样本并赋予标记，并将带有伪标记的样本数据传给另一个分类器去学习，从而**你依我依/共同进步**。



输入：有标记样本集 $D_l=\{(\langle \boldsymbol{x}_1^1,\boldsymbol{x}_1^2\rangle,y_1),\ldots,(\langle \boldsymbol{x}_l^1,\boldsymbol{x}_l^2\rangle,y_l)\}$；
未标记样本集 $D_u=\{\langle \boldsymbol{x}_{l+1}^1,\boldsymbol{x}_{l+1}^2\rangle,\ldots,\langle \boldsymbol{x}_{l+u}^1,\boldsymbol{x}_{l+u}^2\rangle\}$；
缓冲池大小 s；
每轮挑选的正例数 p；
每轮挑选的反例数 n；
基学习算法 $\mathfrak{L}$；
学习轮数 T。

设置缓冲池，减少了每轮计算置信度的次数

过程：

从 D_u 中随机抽取 s 个样本构成缓冲池 D_s；
$D_u=D_u\setminus D_s$;
for $j=1,2$ **do**
  $D_l^j=\{(\boldsymbol{x}_i^j,y_i)\mid(\langle \boldsymbol{x}_i^j,\boldsymbol{x}_i^{3-j}\rangle,y_i)\in D_l\}$; 各视图的有标记样本
end for
for $t=1,2,\ldots,T$ **do**
  for $j=1,2$ **do**
    $h_j\leftarrow\mathfrak{L}(D_l^j)$; 基于每个视图训练初始学习器
    考察 h_j 在 $D_s^j=\{\boldsymbol{x}_i^j\mid\langle \boldsymbol{x}_i^j,\boldsymbol{x}_i^{3-j}\rangle\in D_s\}$ 上的分类置信度，挑选 p 个正例置信度最高的样本 $D_p\subset D_s$、n 个反例置信度最高的样本 $D_n\subset D_s$;
    由 D_p^j 生成伪标记正例 $\tilde{D}_p^{3-j}=\{(\boldsymbol{x}_i^{3-j},+1)\mid\boldsymbol{x}_i^j\in D_p^j\}$;
    由 D_n^j 生成伪标记反例 $\tilde{D}_n^{3-j}=\{(\boldsymbol{x}_i^{3-j},-1)\mid\boldsymbol{x}_i^j\in D_n^j\}$;
    $D_s=D_s\setminus(D_p\bigcup D_n)$; 两个学习器挑选的不会有重复
  end for
  if h_1,h_2 均未发生改变 **then**
    break
  else
    for $j=1,2$ **do**
      $D_l^j=D_l^j\bigcup\left(\tilde{D}_p^j\bigcup\tilde{D}_n^j\right)$; 加入打过伪标的未标记样本
    end for
    从 D_u 中随机抽取 $2p+2n$ 个样本加入 D_s 补充缓冲池
  end if
end for
输出：分类器 h_1, h_2 最终输出两个分类器做集成

1.4 半监督聚类

前面提到的几种方法都是借助无标记样本数据来辅助监督学习的训练过程，从而使得学习更加充分/泛化性能得到提升；半监督聚类则是借助已有的监督信息来辅助聚类的过程。一般而言，监督信息大致有两种类型：

必连与勿连约束：必连指的是两个样本必须在同一个类簇，勿连则是必不在同一个类簇。

标记信息：少量的样本带有真实的标记。

下面主要介绍两种基于半监督的 K-Means 聚类算法：第一种是数据集包含一些必连与勿连关系，另外一种则是包含少量带有标记的样本。两种算法的基本思想都十分的简单：对于带有约束关系的 k-均值算法，在迭代过程中对每个样本划分类簇时，需要**检测当前划分是否满足约束关系**，若不满足则会将该样本划分到距离次小对应的类簇中，再继续检测是否满足约束关系，直到完成所有样本的划分。算法流程如下图所示：

输入：样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$;
必连约束集合 $\mathcal{M}$;
勿连约束集合 $\mathcal{C}$;
聚类簇数 k.

过程：
从 D 中随机选取 k 个样本作为初始均值向量$\{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\ldots,\boldsymbol{\mu}_k\}$;
repeat
  $C_j=\varnothing\ (1\leqslant j\leqslant k)$;
  for $i=1,2,\ldots,m$ **do**
    计算样本 $\boldsymbol{x}_i$ 与各均值向量 $\boldsymbol{\mu}_j\ (1\leqslant j\leqslant k)$ 的距离: $d_{ij}=||\boldsymbol{x}_i-\boldsymbol{\mu}_j||_2$;
    $\mathcal{K}=\{1,2,\ldots,k\}$;
    is_merged=false;
    while ¬ is_merged **do**
      基于 $\mathcal{K}$ 找出与样本 $\boldsymbol{x}_i$ 距离最近的簇: $r=\arg\min_{j\in\mathcal{K}} d_{ij}$;
      检测将 $\boldsymbol{x}_i$ 划入聚类簇 C_r 是否会违背 $\mathcal{M}$ 与 $\mathcal{C}$ 中的约束;
      if ¬ is_voilated **then**
        $C_r=C_r\bigcup\{\boldsymbol{x}_i\}$;
        is_merged=true
      else
        $\mathcal{K}=\mathcal{K}\setminus\{r\}$; 若不满足则寻找距离次小的类簇
        if $\mathcal{K}=\varnothing$ **then**
          break并返回错误提示
        end if
      end if
    end while
  end for
  for $j=1,2,\ldots,k$ **do**
    $\boldsymbol{\mu}_j=\frac{1}{|C_j|}\sum_{\boldsymbol{x}\in C_j}\boldsymbol{x}$;
  end for
until 均值向量均未更新
输出：簇划分 $\{C_1,C_2,\ldots,C_k\}$

对样本进行划分时，需检测是否满足约束关系，其它步骤均相同

对于带有少量标记样本的 k-均值算法，则可以**利用这些有标记样本进行类中心的指定，同时在对样本进行划分时，不需要改变这些有标记样本的簇隶属关系**，直接将其划分到对应类簇即可。算法流程如下所示：

输入： 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
少量有标记样本 $S = \bigcup_{j=1}^{k} S_j$;
聚类簇数 k .

过程：

```
for j = 1,2,...,k do
    μ_j = (1/|S_j|) Σ_{x∈S_j} x
end for
```
使用带标记样本各类别的均值向量作为初始类中心

```
repeat
    C_j = ∅ (1 ⩽ j ⩽ k);
    for j = 1,2,...,k do
        for all x ∈ S_j do
            C_j = C_j ⋃ {x}
        end for
    end for
```
带标记样本直接划入对应类簇

```
    for all x_i ∈ D \ S do
        计算样本 x_i 与各均值向量 μ_j (1 ⩽ j ⩽ k) 的距离: d_ij = ||x_i − μ_j||_2 ;
        找出与样本 x_i 距离最近的簇: r = arg min_{j∈{1,2,...,k}} d_ij ;
        将样本 x_i 划入相应的簇: C_r = C_r ⋃ {x_i}
    end for
```
划分无标记样本

```
    for j = 1,2,...,k do
        μ_j = (1/|C_j|) Σ_{x∈C_j} x;
    end for
```
重新计算类中心

```
until 均值向量均未更新
```

输出： 簇划分 $\{C_1, C_2, \ldots, C_k\}$

2. 强化学习

强化学习（Reinforcement Learning，简称 **RL**）是机器学习的一个重要分支，前段时间人机大战的主角 AlphaGo 正是以强化学习为核心技术。在强化学习中，包含两种基本的元素：**状态**与**动作**，**在某个状态下执行某种动作，这便是一种策略**，学习器要做的就是通过不断地探索学习，从而获得一个好的策略。例如：在围棋中，一种落棋的局面就是一种状态，若能知道每种局面下的最优落子动作，那就攻无不克/百战不殆了~

若将状态看作为属性，动作看作为标记，易知：**监督学习和强化学习都是在试图寻找一个映射，从已知属性/状态推断出标记/动作**，这样强化学习中的策略相当于监督学习中的分类/回归器。但在实际问题中，**强化学习并没有监督学习那样的标记信息**，通常都是在**尝试动作后才能获得结果**，因此强化学习是通过反馈的结果信息不断调整之前的策略，从而算法能够学习到：在什么样的状态下选择什么样的动作可以获得最好的结果。

2.1 基本要素

强化学习任务通常使用**马尔可夫决策过程**（Markov Decision Process，简称 **MDP**）来描述，具体而言：机器处在一个环境中，每个状态为机器对当前环境的感知；机器只能通过动作来影响环境，当机器执行一个动作后，会使得环境按某种概率转移到另一个状态；同时，环境会根据潜在的奖赏函数反馈给机器一个奖赏。综合而言，强化学习主要包含四个要素：状态、动作、转移概率以及奖赏函数。

状态（X）：机器对环境的感知，所有可能的状态称为状态空间；
动作（A）：机器所采取的动作，所有能采取的动作构成动作空间；
转移概率（P）：当执行某个动作后，当前状态会以某种概率转移到另一个状态；
奖赏函数（R）：在状态转移的同时，环境给反馈给机器一个奖赏。

因此，**强化学习的主要任务就是通过在环境中不断地尝试，根据尝试获得的反馈信息调整策略，最终生成一个较好的策略 π，机器根据这个策略便能知道在什么状态下应该执行什么动作**。常见的策略表示方法有以下两种：

确定性策略：π（x）=a，即在状态 x 下执行 a 动作；
随机性策略：P=π（x,a），即在状态 x 下执行 a 动作的概率。

一个策略的优劣取决于长期执行这一策略后的累积奖赏，换句话说：可以使用累积奖赏来评估策略的好坏，最优策略则表示在初始状态下一直执行该策略后，最后的累积奖赏值最高。长期累积奖赏通常使用下述两种计算方法：

- **T步累积奖赏**：$\mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t]$，即执行该策略T步的平均奖赏的期望值。
- **r折扣累积奖赏**：$\mathbb{E}[\sum_{t=0}^{+\infty} \gamma^t r_{t+1}]$，一直执行到最后，同时越往后的奖赏权重越低。

2.2 K 摇摆赌博机

首先我们考虑强化学习最简单的情形：仅考虑一步操作，即在状态 x 下只需执行一次动作 a 便能观察到奖赏结果。易知：欲最大化单步奖赏，我们需要知道每个动作带来的期望奖赏值，这样便能选择奖赏值最大的动作来执行。若每个动作的奖赏值为确定值，则只需要将每个动作尝试一遍即可，但大多数情形下，一个动作的奖赏值来源于一个概率分布，因此需要进行多次的尝试。

单步强化学习实质上是 **K-摇臂赌博机**（K-armed bandit）的原型，一般我们**尝试动作的次数是有限的**，那如何利用有限的次数进行有效地探索呢？这里有两种基本的想法：

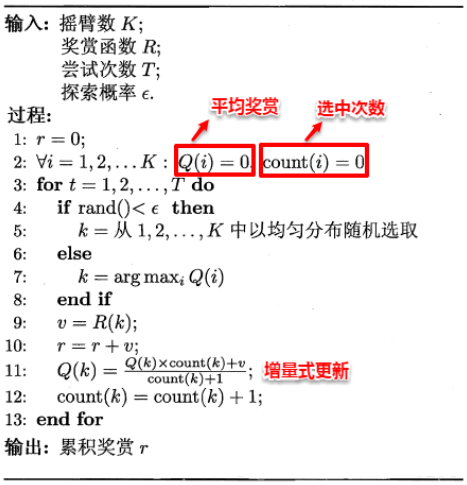
仅探索法：将尝试的机会平均分给每一个动作，即轮流执行，最终将每个动作的平均奖赏作为期望奖赏的近似值。

仅利用法：将尝试的机会分给当前平均奖赏值最大的动作，隐含着让一部分人先富起来的思想。

可以看出：上述**两种方法是相互矛盾的**，仅探索法能较好地估算每个动作的期望奖赏，但是没能根据当前的反馈结果调整尝试策略；仅利用法在每次尝试之后都更新尝试策略，符合强化学习的思（tao）维（lu），但容易找不到最优动作。因此需要在这两者之间进行折中。

2.2.1 ε-贪心法

ε-贪心法基于一个概率来对探索和利用进行折中，具体而言：在每次尝试时，以 ε 的概率进行探索，即以均匀概率随机选择一个动作；以 1-ε 的概率进行利用，即选择当前最优的动作。ε-贪心法只需记录每个动作的当前平均奖赏值与被选中的次数，便可以增量式更新。



2.2.2 Softmax

Softmax 算法则基于当前每个动作的平均奖赏值来对探索和利用进行折中，Softmax 函数将一组值转化为一组概率，**值越大对应的概率也越高，因此当前平均奖赏值越高的动作被选中的几率也越大。Softmax 函数如下所示：**

$$P(k)=\frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}},$$

T：温度
T->0:越放大差距
T->inf：越缩小差距

输入： 摇臂数 K;
奖赏函数 R;
尝试次数 T;
温度参数 τ.

过程：

```
r = 0;
∀i = 1,2,...K : Q(i) = 0, count(i) = 0;
for t = 1,2,...,T do
    k = 从1,2,...,K 中根据式(16.4)随机选取
    v = R(k);
    r = r + v;
    Q(k) = (Q(k)×count(k)+v)/(count(k)+1);
    count(k) = count(k) + 1;
end for
```

即根据Softmax函数

输出： 累积奖赏 r

2.3 有模型学习

若学习任务中的四个要素都已知，即状态空间、动作空间、转移概率以及奖赏函数都已经给出，这样的情形称为“**有模型学习**”。假设状态空间和动作空间均为有限，即均为离散值，这样我们不用通过尝试便可以对某个策略进行评估。

2.3.1 策略评估

前面提到：**在模型已知的前提下，我们可以对任意策略的进行评估**（后续会给出演算过程）。一般常使用以下两种值函数来评估某个策略的优劣：

状态值函数（V）：V（x），即从状态 x 出发，使用 π 策略所带来的累积奖赏；

状态-动作值函数（Q）：Q（x,a），即从状态 x 出发，执行动作 a 后再使用 π 策略所带来的累积奖赏。

根据累积奖赏的定义，我们可以引入 T 步累积奖赏与 r 折扣累积奖赏：

$$\begin{cases} V_T^\pi(x) = \mathbb{E}_\pi\left[\frac{1}{T}\sum_{t=1}^{T} r_t \mid x_0 = x\right], & T \text{ 步累积奖赏;} \\ V_\gamma^\pi(x) = \mathbb{E}_\pi\left[\sum_{t=0}^{+\infty}\gamma^t r_{t+1} \mid x_0 = x\right], & \gamma \text{ 折扣累积奖赏.} \end{cases}$$

$$\begin{cases} Q_T^\pi(x,a) = \mathbb{E}_\pi[\frac{1}{T}\sum_{t=1}^{T} r_t \mid x_0 = x, a_0 = a]; \\ Q_\gamma^\pi(x,a) = \mathbb{E}_\pi[\sum_{t=0}^{+\infty}\gamma^t r_{t+1} \mid x_0 = x, a_0 = a]. \end{cases}$$

由于 MDP 具有马尔可夫性，即现在决定未来，将来和过去无关，我们很容易找到值函数的递归关系：

T步累积奖赏

$$V_T^\pi(x) = \mathbb{E}_\pi\left[\frac{1}{T}\sum_{t=1}^{T} r_t \mid x_0 = x\right] = \mathbb{E}_\pi\left[\frac{1}{T}r_1 + \frac{T-1}{T}\frac{1}{T-1}\sum_{t=2}^{T} r_t \mid x_0 = x\right]$$

$$= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P_{x\to x'}^a\left(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}\mathbb{E}_\pi\left[\frac{1}{T-1}\sum_{t=1}^{T-1} r_t \mid x_0 = x'\right]\right)$$ → 全概率展开

$$= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P_{x\to x'}^a\left(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}V_{T-1}^\pi(x')\right)$$ 递归关系

类似地，对于 r 折扣累积奖赏可以得到：

$$V_\gamma^\pi(x) = \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P_{x\to x'}^a\left(R_{x\to x'}^a + \gamma V_\gamma^\pi(x')\right).$$

易知：**当模型已知时，策略的评估问题转化为一种动态规划问题**，即以填表格的形式自底向上，先求解每个状态的单步累积奖赏，再求解每个状态的两步累积奖赏，一直迭代逐步求解出每个状态的 T 步累积奖赏。算法流程如下所示：

输入： MDP 四元组 $E = \langle X, A, P, R\rangle$;
被评估的策略 π;
累积奖赏参数 T.

过程：

$\forall x \in X: V(x) = 0;$ → 即最后一个状态的值函数，由于不再执行动作/转移，因此值函数为0

for $t = 1, 2, \ldots$ **do**
  $\forall x \in X: V'(x) = \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P_{x\to x'}^a\left(\frac{1}{t}R_{x\to x'}^a + \frac{t-1}{t}V(x')\right)$;
  if $t = T+1$ **then**
    break
  else
    $V = V'$ → 记录每个状态的t步累积奖赏值
  end if
end for

输出： 状态值函数 V

对于状态-动作值函数，只需通过简单的转化便可得到：

$$\begin{cases} Q_T^\pi(x,a) = \sum_{x'\in X} P_{x\to x'}^a(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}V_{T-1}^\pi(x')); \\ Q_\gamma^\pi(x,a) = \sum_{x'\in X} P_{x\to x'}^a(R_{x\to x'}^a + \gamma V_\gamma^\pi(x')). \end{cases}$$

2.3.2 策略改进

理想的策略应能使得每个状态的累积奖赏之和最大，简单来理解就是：不管处于什么状态，只要通过该策略执行动作，总能得到较好的结果。因此对于给定的某个策略，我们需要对其进行改进，从而得到**最优的值函数**。

$$\begin{cases} V_T^*(x) = \max\limits_{a\in A} \sum\limits_{x'\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \frac{T-1}{T} V_{T-1}^*(x')\right); \\ V_\gamma^*(x) = \max\limits_{a\in A} \sum\limits_{x'\in X} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma V_\gamma^*(x')\right). \end{cases}$$

选择当前最优的动作

$$\begin{cases} Q_T^*(x,a) = \sum\limits_{x'\in X} P_{x\to x'}^a \left(\frac{1}{T} R_{x\to x'}^a + \frac{T-1}{T} \max\limits_{a'\in A} Q_{T-1}^*(x',a')\right); \\ Q_\gamma^*(x,a) = \sum\limits_{x'\in X} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma \max\limits_{a'\in A} Q_\gamma^*(x',a')\right). \end{cases}$$

最优 Bellman 等式改进策略的方式为：**将策略选择的动作改为当前最优的动作**，而不是像之前那样对每种可能的动作进行求和。易知：选择当前最优动作相当于将所有的概率都赋给累积奖赏值最大的动作，因此每次改进都会使得值函数单调递增。

$$\pi'(x) = \arg\max_{a\in A} Q^\pi(x,a)$$

将策略评估与策略改进结合起来，我们便得到了生成最优策略的方法：先给定一个随机策略，现对该策略进行评估，然后再改进，接着再评估/改进一直到策略收敛、不再发生改变。这便是策略迭代算法，算法流程如下所示：

输入： MDP 四元组 $E = \langle X, A, P, R\rangle$;
累积奖赏参数 T.
过程：
$\forall x \in X:\ V(x) = 0,\ \pi(x,a) = \frac{1}{|A(x)|}$; 随机策略
loop
  for $t = 1, 2, \ldots$ **do**
    $\forall x \in X: V'(x) = \sum_{a\in A} \pi(x,a) \sum_{x'\in X} P_{x\to x'}^a \left(\frac{1}{t} R_{x\to x'}^a + \frac{t-1}{t} V(x')\right)$; 使用动态规划法计算
    if $t = T+1$ **then**
      break
    else
      $V = V'$
    end if
  end for 得到当前策略的T步状态值函数
  $\forall x \in X: \pi'(x) = \arg\max_{a\in A} Q(x,a)$; 依据状态值函数更新策略
  if $\forall x: \pi'(x) = \pi(x)$ **then**
    break
  else
    $\pi = \pi'$
  end if
end loop
输出： 最优策略 π

可以看出：策略迭代法在每次改进策略后都要对策略进行重新评估，因此比较耗时。若从最优化值函数的角度出发，即先迭代得到最优的值函数，再来计算如何改变策略，这便是值迭代算法，算法流程如下所示：

输入： MDP 四元组 $E=\langle X,A,P,R\rangle$;
累积奖赏参数 T;
收敛阈值 θ.
过程：
$\forall x\in X: V(x)=0$;
for $t=1,2,\ldots$ **do**
  $\forall x\in X: V'(x)=\max_{a\in A}\sum_{x'\in X}P^a_{x\to x'}\left(\frac{1}{t}R^a_{x\to x'}+\frac{t-1}{t}V(x')\right)$; 每次都选择最优动作
  if $\max_{x\in X}|V(x)-V'(x)|<\theta$ **then**
    break
  else
    $V=V'$
  end if
end for → 得到了最优的值函数
输出： 策略 $\pi(x)=\arg\max_{a\in A}Q(x,a)$ → 根据最优值函数来改变策略

2.4 蒙特卡罗强化学习

在现实的强化学习任务中，**环境的转移函数与奖赏函数往往很难得知**，因此我们需要考虑在不依赖于环境参数的条件下建立强化学习模型，这便是**免模型学习**。蒙特卡罗强化学习便是其中的一种经典方法。

由于模型参数未知，状态值函数不能像之前那样进行全概率展开，从而运用动态规划法求解。一种直接的方法便是通过采样来对策略进行评估/估算其值函数，**蒙特卡罗强化学习正是基于采样来估计状态-动作值函数**：对采样轨迹中的每一对状态-动作，记录其后的奖赏值之和，作为该状态-动作的一次累积奖赏，通过多次采样后，使用累积奖赏的平均作为状态-动作值的估计，并**引入 ε-贪心策略保证采样的多样性**。

输入： 环境 E;
动作空间 A;
起始状态 x_0;
策略执行步数 T.
过程：
$Q(x,a)=0,\ \mathrm{count}(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$; 随机策略
for $s=1,2,\ldots$ **do**
  在 E 中执行策略 π 产生轨迹
  $<x_0,a_0,r_1,x_1,a_1,r_2,\ldots,x_{T-1},a_{T-1},r_T,x_T>$;
  for $t=0,1,\ldots,T-1$ **do**
    $R=\frac{1}{T-t}\sum_{i=t+1}^{T}r_i$; 状态-动作的T-t步累积奖赏
    $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times\mathrm{count}(x_t,a_t)+R}{\mathrm{count}(x_t,a_t)+1}$; 更新状态-动作值函数
    $\mathrm{count}(x_t,a_t)=\mathrm{count}(x_t,a_t)+1$
  end for
  对所有已见状态 x:

$$\pi(x,a)=\begin{cases}\arg\max_{a'}Q(x,a'), & \text{以概率 } 1-\epsilon;\\ \text{以均匀概率从 } A \text{ 中选取动作}, & \text{以概率 } \epsilon.\end{cases}$$

end for
输出： 策略 π

在上面的算法流程中，被评估和被改进的都是同一个策略，因此称为**同策略蒙特卡罗强化学习算法**。引入 ε-贪心仅是为了便于采样评估，而在使用策略时并不需要 ε-贪心，那能否仅在评估时使用 ε-贪心策略，而在改进时使用原始策略呢？这便是**异策略蒙特卡罗强化学习算法**。

输入： 环境 E;
动作空间 A;
起始状态 x_0;
策略执行步数 T.

过程：

$Q(x,a)=0,\ \mathrm{count}(x,a)=0,\ \pi(x,a)=\frac{1}{|A(x)|}$;
for $s=1,2,\ldots$ **do**
  在 E 中执行 π 的 ϵ-贪心策略产生轨迹
  $< x_0,a_0,r_1,x_1,a_1,r_2,\ldots,x_{T-1},a_{T-1},r_T,x_T >$;
  $p_i=\begin{cases}1-\epsilon+\epsilon/|A|, & a_i=\pi(x);\ \text{选择当前最优动作的概率}\\ \epsilon/|A|, & a_i\neq\pi(x),\ \text{选择随机动作的概率}\end{cases}$
  for $t=0,1,\ldots,T-1$ **do**
    $R=\frac{1}{T-t}\sum_{i=t+1}^{T}(r_i\times\prod_{j=i}^{T-1}\frac{1}{p_j})$; 修正的累积奖赏
    $Q(x_t,a_t)=\frac{Q(x_t,a_t)\times\mathrm{count}(x_t,a_t)+R}{\mathrm{count}(x_t,a_t)+1}$;
    $\mathrm{count}(x_t,a_t)=\mathrm{count}(x_t,a_t)+1$
  end for
  $\pi(x)=\arg\max_{a'}Q(x,a')$ → 根据估计的状态-动作值函数改变策略
end for

输出： 策略 π

2.5 AlphaGo 原理浅析

本篇一开始便提到强化学习是 AlphaGo 的核心技术之一，刚好借着这个东风将 AlphaGo 的工作原理了解一番。正如人类下棋那般“**手下一步棋，心想三步棋**”，Alphago 也正是这个思想，**当处于一个状态时，机器会暗地里进行多次的尝试/采样，并基于反馈回来的结果信息改进估值函数，从而最终通过增强版的估值函数来选择最优的落子动作。**

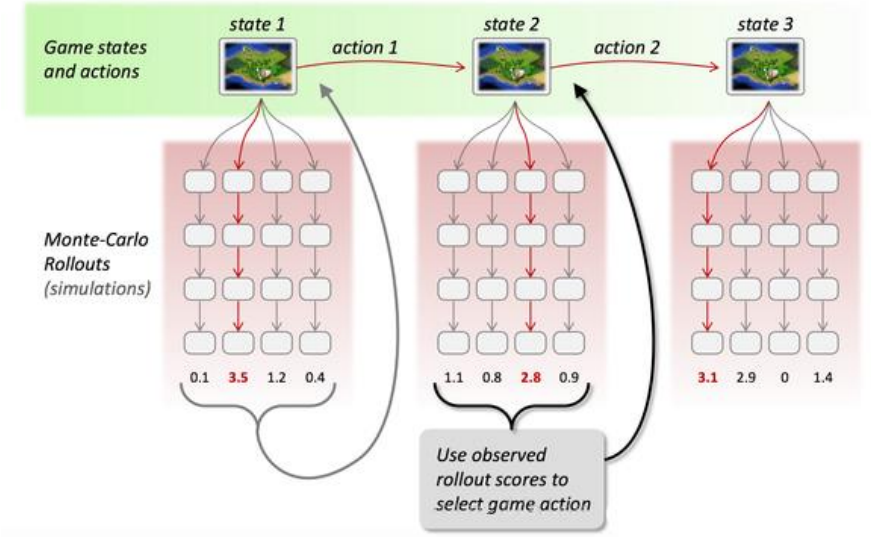
其中便涉及到了三个主要的问题：**（1）如何确定估值函数（2）如何进行采样（3）如何基于反馈信息改进估值函数**，这正对应着 AlphaGo 的三大核心模块：**深度学习、蒙特卡罗搜索树、强化学习**。

2.5.1 深度学习（拟合估值函数）

由于围棋的状态空间巨大，像蒙特卡罗强化学习那样通过采样来确定值函数就行不通了。在围棋中，**状态值函数可以看作为一种局面函数，状态-动作值函数可以看作一种策略函数，**若我们能获得这两个估值函数，便可以根据这两个函数来完成：(1)衡量当前局面的价值；(2)选择当前最优的动作。那如何精确地估计这两个估值函数呢？**这就用到了深度学习，通过大量的对弈数据自动学习出特征，从而拟合出估值函数。**

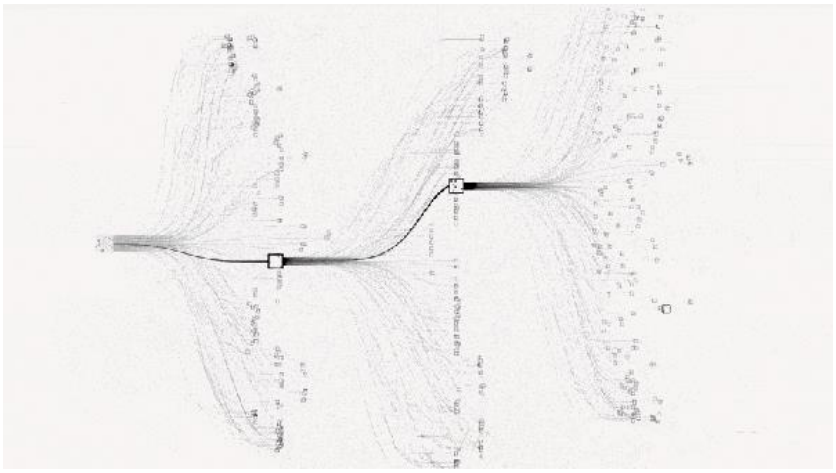
2.5.2 蒙特卡罗搜索树（采样）

蒙特卡罗树是一种经典的搜索框架，它通过反复地采样模拟对局来探索状态空间。具体表现在：从当前状态开始，利用策略函数尽可能选择当前最优的动作，同时也引入随机性来减小估值错误带来的负面影响，从而模拟棋局运行，使得棋盘达到终局或一定步数后停止。



2.5.3 强化学习（调整估值函数）

在使用蒙特卡罗搜索树进行多次采样后，每次采样都会反馈后续的局面信息（利用局面函数进行评价），根据反馈回来的结果信息自动调整两个估值函数的参数，这便是强化学习的核心思想，最后基于改进后的策略函数选择出当前最优的落子动作。



第五章 致谢及引用

声明：本资料全部整理自互联网，仅用作学习。现注明转载出处如下：

致谢：

第一章·概述

网址：https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/217599?fr=aladdin

网址：https://blog.csdn.net/jiejinquanil/article/details/52270178

网址：https://blog.csdn.net/u011826404/article/category/6528944

第二章·监督式学习

网址：https://www.jianshu.com/p/c7be04c50659

网址：https://blog.csdn.net/sinat_20177327/article/details/79729551

第三章·非监督式学习

网址：https://www.jianshu.com/p/e98fe7163a25

第四章·半监督式学习与强化学习

网址：https://blog.csdn.net/u011826404/article/details/74358913

网址：https://blog.csdn.net/u011826404/article/details/75576856

更多资料：

1.《机器学习》——周志华（西瓜书）

网址：http://www.gwylab.com/download/机器学习——周志华.pdf

2.《统计学习方法》——李航

网址：http://www.gwylab.com/download/统计学习方法——李航.pdf