BP 算法　神经网络训练　K-means

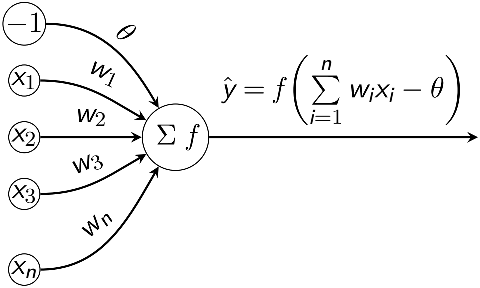
杨　硕

2022 年 06 月 03 日

目　录

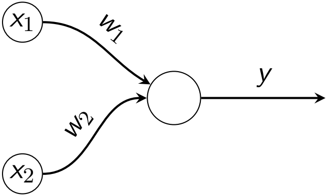
目　录

神经元模型



- 神经元模型: $\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$
- x_i 来自第 i 个神经元的输入
- w_i 第 i 个神经元的连接权重
- f 激活函数 activation function
- θ 偏置单元 *bias*, 阈值
- $\hat{y}$ 当前神经元的输出
- 总输入值 $\sum_{i=1}^{n} w_i x_i$ 超过阈值 θ, 神经元兴奋, 否则抑制

单层感知机



$$\widehat{y} = sgn(w_1 \cdot x_1 + w_2 \cdot x_2 - \theta)$$

- 只有输出层有神经元
- 激活函数是阶跃函数
- 可以解决线性可分问题，比如与，或，非
- 无法解决非线性可分问题，比如异或 (XOR) (多层感知机可以解决)
- 感知机预测正确，权重不发生调整更新

单层感知机

感知机权重调整规则

$$
\begin{aligned}
w_i &\leftarrow w_i + \Delta w_i \qquad (1) \\
\Delta w_i &= \eta(y - \widehat{y}\,)x_i \qquad (2)
\end{aligned}
$$

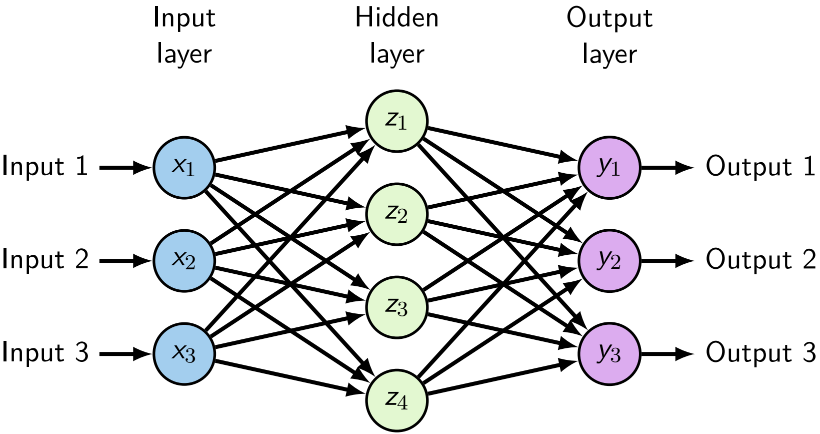
$\eta \in (0,1)$ 被称作学习率

单层感知机无法解决异或划分问题

$$
\begin{cases}
1 \cdot w_1 + 1 \cdot w_2 - \theta < 0 \\
0 \cdot w_1 + 0 \cdot w_2 - \theta < 0 \\
1 \cdot w_1 + 0 \cdot w_2 - \theta > 0 \\
0 \cdot w_1 + 1 \cdot w_2 - \theta > 0
\end{cases}
\Longrightarrow
\begin{cases}
w_1 + w_2 < \theta \\
\theta > 0 \\
w_1 > \theta \\
w_2 > \theta
\end{cases}
$$

多层感知机

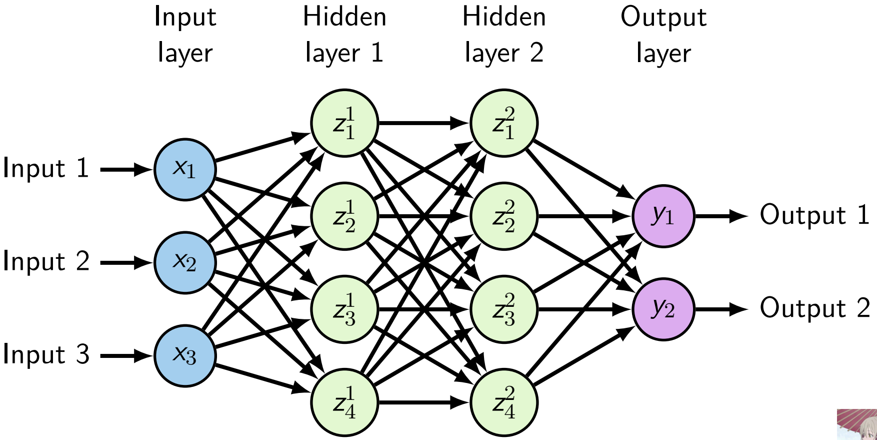
单隐层网络



- x_i: 输入层神经元，接收外界的输入
- z_h: 隐层神经元，对输入信号加工处理
- y_j: 输出层神经元，输出最终结果
- 神经网络的隐层通常有多个，这样能够学到数据之间复杂的关系.

多层感知机

双隐层网络


Input layer
Hidden layer 1
Hidden layer 2
Output layer
Input 1
Input 2
Input 3
x_1
x_2
x_3
z_1^1
z_2^1
z_3^1
z_4^1
z_1^2
z_2^2
z_3^2
z_4^2
y_1
y_2
Output 1
Output 2

目　录

激活函数

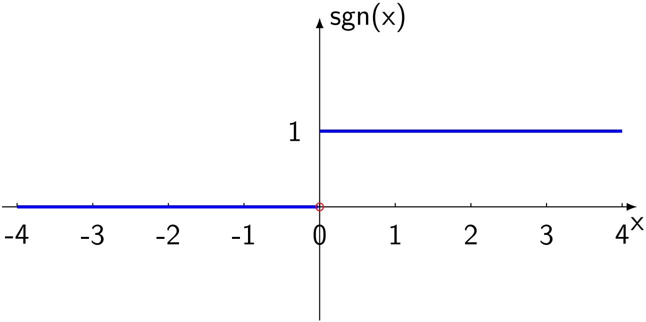
增强神经元的表达能力，表达非线性模型
激活函数应该满足：

- 非线性：使用非线性激活函数的多层神经网络，可逼近所有函数
- 可微性：使用的优化方法是基于梯度更新参数的 (此时是必须的)
- 单调性：保证单层网络的损失函数是凸函数
- 近似恒等性：$f(x) \approx x$ 参数初始化为随机小值时，神经网络更稳定

激活函数输出值的范围：

- 有限值：基于梯度的优化方法更稳定
- 无限值：调小学习率

$sgn(x)$

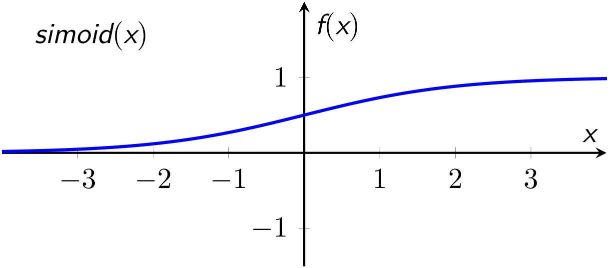


阶跃函数 $sgn(x) =$

$$\begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- $y \in \{0, 1\}$ 1 代表神经元兴奋，0 代表神经元抑制
- 在 $x = 0$ 处不连续，不可导

sigmoid



$$sigmoid(x) = \frac{1}{1 + \exp -x}$$

优点:

- 将输出值映射到 (0,1) 之间，单调连续
- 输出范围有限，优化稳定，可作为输出层
- 求导容易 $\nabla f(x) = f(x)[1 - f(x)]$

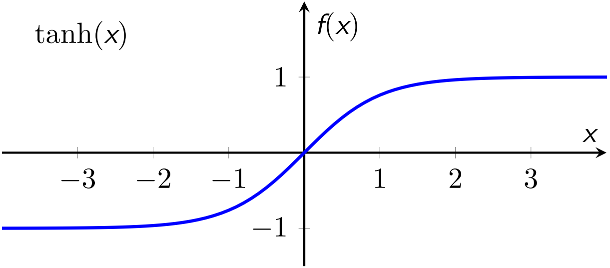
缺点:

- 非 0 均值输出，梯度始终是正的，收敛慢
- 指数运算慢，训练时间长
- 饱和性问题，以及梯度消失

sigmoid 只能用作二分类问题的输出激活，对于多分类问题，无法解决，通常使用 softmax 作为最后分类层输出的激活函数

`tf.nn.sigmoid(x)`, `tf.math.sigmoid(x)`, `tf.sigmoid(x)`

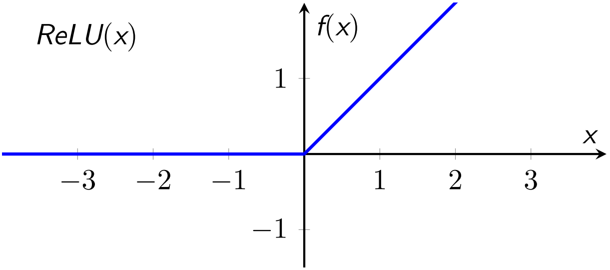
$\tanh(x)$



$$\tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2sigmoid(2x) - 1$$

解决 *sigmoid* 激活函数非 0 均值输出的问题 → $tanh(x)$ 是 0 均值的，比 sigmoid 收敛快
当输入很大很小时，输出几乎平滑，梯度消失，不利于权重更新
`tf.nn.tanh(x)`, `tf.math.tanh(x)`, `tf.tanh(x)`

ReLU(Rectified Liner Unit) 激活函数



$$ReLU(x) = \max(0, x)$$

优点:

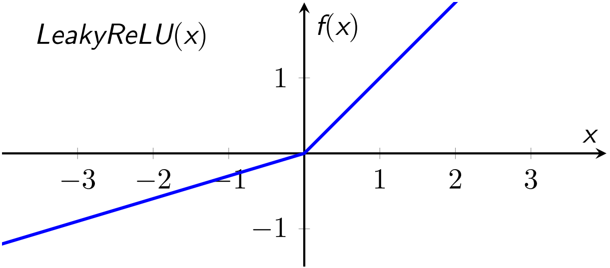
- 解决 *tanh* 梯度消失 (正区间)
- 输出范围无限
- 不涉及指数运算，计算速度快

```
tf.nn.relu(x)
```

缺点:

- 非 0 均值输出，收敛慢
- Dead ReLU 学习率很大，反向传播后的参数可能为负数，导致下一轮正向输入为负数，神经元不被激活，参数不被更新

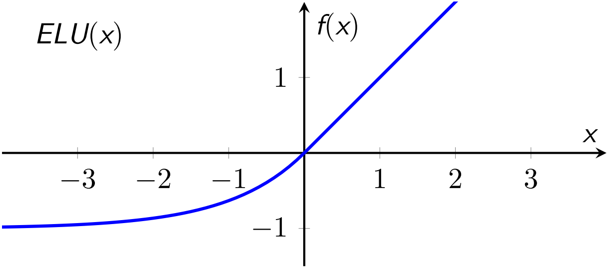
Leaky ReLU 函数



$$LeakyReLU(x) = max(\alpha x, x), \alpha \in (0, 1)$$

解决 $ReLU(x)$ 在 $x < 0$ 时候不被激活的问题，避免 $ReLU$ 死掉.
α 是一个可调参数，每个通道一个，可以通过反向传播训练得到.
在实际操作中，并没有完全证明 Leaky ReLU 总是好于 ReLU.
`tf.nn.leaky_relu(x)`

ELU(Exponential Linear Unit) 函数



$$ELU(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \le 0 \end{cases}$$

- α 是一个可调参数，控制 ELU 在负值区间的饱和位置
- ELU 的输出均值接近于 0，所以收敛速度更快
- 右侧线性部分是的 ELU 能够缓解梯度消失
- 左侧软饱能让 ELU 对输入变化或噪声更鲁棒，避免神经元死掉

```
tf.nn.elu(x)
```

激活函数

神经网络训练建议:

- 首选 ReLU 做激活函数;
- 学习率设置为较小值;
- 输入特征标准化 (满足均值为 0, 标准差为 1 的高斯分布)
- 初始参数中心化

目　录

损失函数

- 同一个算法的损失函数不是唯一的
- 损失函数是参数 (w, α) 的函数
- 损失函数用来评价网络模型的好坏，损失函数越小说明模型和参数越符合训练样本
- 损失函数是一个标量
- 选择损失函数时，要选择对参数可微的函数
- 损失函数也称作代价函数，目标函数

损失函数

1. 均方误差 (Mean Square Error) 损失函数是回归问题最常见的损失函数.

$$MSE(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i\ )^2$$

   `tf.keras.losses.MSE`

2. 交叉熵 (cross entropy) 表征两个概率分布之间的距离，交叉熵越小说明二者分布越接近，是分类问题常使用的损失函数

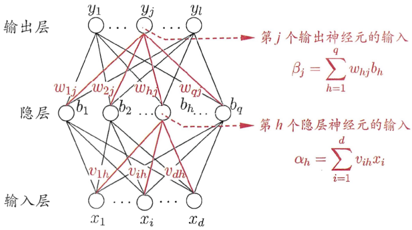
$$H(y,\hat{y}) = -\sum y \times \ln \hat{y}$$

   y 表示数据实际值,$\hat{y}$ 表示网络的预测输出值.
   `tf.keras.losses.categorical_crossentropy`

3. 自定义损失函数
   目标检测 Object detection：分类损失 Classificition Loss，回归损失 Bounding Box Regeression Loss
   针对特定的背景，具体的任务设计损失函数

目　录

BP 算法 error BackPropagation



- 输入层神经元个数 b, 隐层神经元个数 h, 输出层神经元个数 l.
- γ_h 隐层第 h 个神经元的阈值, θ_j, 输出层第 j 个神经元的阈值.
- v_{ih} : 输入层第 i 神经元和隐层第 h 个神经元之间的连接权重.
- w_{hj} : 隐层第 h 神经元和输出层第 j 个神经元之间的连接权重.
- $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$: 隐层第 h 神经元的输入.
- b_h : 隐层第 h 个神经元的输出
- $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$: 输入层第 j 个神经元的输入.
- 训练集 $\mathcal{D} = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), \cdots, (\mathbf{x_m}, \mathbf{y_m})\}, \mathbf{x_i} \in \mathbb{R}^d, \mathbf{y_i} \in \mathbb{R}^l$

BP 算法推导

样本实例:$(\mathbf{x_k}, \mathbf{y_k})$，输入 $\mathbf{x_k}$ 的输出:

$$
\begin{aligned}
\hat{\mathbf{y}}^{\mathbf{k}} &= (\hat{y}_1^k, \hat{y}_2^k, \cdots, \hat{y}_l^k) \\
f(x) &= \frac{1}{1+e^{-x}} \\
\hat{y}_j^k &= f(\beta_j - \theta_j) \\
E_k &= \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2 \\
\beta_j &= \sum_{h=1}^{q} w_{hj} b_h \\
b_h &= f(\alpha_h - \gamma_h) \\
\alpha_h &= \sum_{i=1}^{d} v_{ih} x_i \\
\Delta w_{hj} &= -\eta \frac{\partial E_k}{\partial w_{hj}}, \Delta v_{ih} = -\eta \frac{\partial E_k}{\partial v_{ih}} \\
\nabla f(x) &= f(x)[1 - f(x)]
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E_k}{\partial w_{hj}} &= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \\
\frac{\partial E_k}{\partial \hat{y}_j^k} &= \hat{y}_j^k - y_j^k \\
\frac{\partial \hat{y}_j^k}{\partial \beta_j} &= \hat{y}_j^k(1 - \hat{y}_j^k), \frac{\partial \hat{y}_j^k}{\partial \theta_j} = -\hat{y}_j^k(1 - \hat{y}_j^k) \\
\frac{\partial \beta_j}{\partial w_{hj}} &= b_h \\
\frac{\partial E_k}{\partial w_{hj}} &= (\hat{y}_j^k - y_j^k) \cdot \hat{y}_j^k(1 - \hat{y}_j^k) \cdot b_h \\
&= -\hat{y}_j^k(y_j^k - \hat{y}_j^k)(1 - \hat{y}_j^k) \cdot b_h \\
\frac{\partial E_k}{\partial \theta_j} &= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \theta_j} \\
&= \hat{y}_j^k(y_j^k - \hat{y}_j^k)(1 - \hat{y}_j^k)
\end{aligned}
$$

BP 算法推导

$$\frac{\partial b_h}{\partial \alpha_h} = b_h(1-b_h), \frac{\partial \alpha_h}{\partial v_{ih}} = x_i, \frac{\partial \beta_j}{\partial b_h} = w_{hj}$$

$$\begin{aligned}
\frac{\partial E_k}{\partial v_{ih}} &= \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial v_{ih}} \\
&= \left(\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \right) \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}} \\
&= \left(\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \right) \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}} \\
&= b_h(1-b_h) \left(\sum_{i=1}^{l} -\hat{y}_j^k (y_j^k - \hat{y}_j^k)(1-\hat{y}_j^k) w_{hj} \right) x_i
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E_k}{\partial \gamma_h} &= \left(\sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \right) \cdot \frac{\partial b_h}{\partial \gamma_h} \\
&= b_h(1-b_h) \left(\sum_{i=1}^{l} \hat{y}_j^k (y_j^k - \hat{y}_j^k)(1-\hat{y}_j^k) w_{hj} \right)
\end{aligned}$$

BP 算法推导

$$g_j = -\hat{y}_j^k(y_j^k - \hat{y}_j^k)(1 - \hat{y}_j^k) \tag{3}$$

$$e_h = b_h(1 - b_h)\left(\sum_{i=1}^{l} \hat{y}_j^k(y_j^k - \hat{y}_j^k)(1 - \hat{y}_j^k) w_{hj}\right) \tag{4}$$

$$\Delta w_{hj} = \eta g_j \cdot b_h \tag{5}$$

$$\Delta \theta_j = -\eta g_j \tag{6}$$

$$w_{hj} = w_{hj} - \Delta w_{hj} \tag{7}$$

$$\theta_j = \theta_j - \Delta \theta_j \tag{8}$$

$$\Delta v_{ih} = \eta e_h \cdot x_i \tag{9}$$

$$\Delta \gamma_h = -\eta e_h \tag{10}$$

$$v_{ih} = v_{ih} - \Delta v_{ih} \tag{11}$$

$$\gamma_h = \gamma_h - \Delta \gamma_h \tag{12}$$

BP 算法伪代码

 # Input： $D=\{(x_k,y_k)\}_{k=1}^{m}$
学习率 η
过程:
在 (0~1) 范围内随机初始化网络中所有连接权和阈值
repeat:
    for all $\{(x_k,y_k)\}\in D$ do
    根据当前参数和式$\hat{y}_j^k=f(\beta_j-\theta_j)$ 计算样本的前向输出$\hat{y}$
    根据公式(3) 计算输出层梯度项g_j
    根据公式(4) 计算隐层神经元梯度项e_h
    根据公式(5), (6), (7), (8) 计算输出层更新后的连接权重w_{hj} 和阈值θ_j
    根据公式(9), (10), (11), (12) 计算隐层更新后的连接权重v_{ih} 和阈值γ_h
    end **for**
until 达到停止条件
 # Output: 连接权重和阈值确定的前反馈神经网络.

BP 算法

- 标准 BP 算法 vs. 累计 BP 算法
  损失函数的不同: 标准 BP 基于单个样本的误差, 累计 BP 基于训练集样本的累计误差.
  标准 BP 算法会出现" 抵消" 现象, 更新频率快; 累计 BP 更新慢.
- Hornik 证明, 只需一个包含足够多神经元的隐层, 多层前馈网络就能以任意精度逼近任意复杂度的连续函数.
- BP 神经存在" 过拟合" 问题
  1. 早停 (early stopping) 划分训练集和验证集
  2. 正则化 (regularization) 添加一个描述网络复杂度的正则化项:

$$E = \lambda \frac{1}{m} \sum_{k=1}^{m} E_k + (1-\lambda) \sum_i w_i^2$$

训练过程将偏好较小的参数值, 使得网络更加" 平滑",λ 用来调整误差和网络复杂度的折中的参数

RBF 网络

- RBF Radial Basis Function, 径向基函数
- 通常是一种单隐层神经网络
  1. 隐层的激活函数是径向基函数
  2. 输出层是对隐层神经元的线性组合
- $\varphi(x) = \sum_{i=1}^{q} w_i \rho(x, c_i)$
  q 为隐层神经元个数,c_i 为第 i 个隐层神经元对应的中心.
  $\rho(x, c_i)$ 是径向基函数, 是一种沿径向对称的标量函数
  通常定义为样本 x 到数据中心 c_i 之间欧式距离的单调函数
- 高斯径向基函数: $\rho(x, c_i) = e^{-\beta_i ||x-c_i||^2}$
- RBF 网络的训练步骤:
  1. 通过随机采样, 聚类等方法确定数据中心 c_i.
  2. 使用 BP 算法来确定参数 w_i 和 β_i.

ART 网络 Adaptive Resonance Theory

- 竞争型学习　无监督学习
  输出神经元相互竞争，某个时刻只有一个神经元被激活　”胜者通吃”
- 组成：比较层，识别层，识别阈值，重置模块
- 缓解了”可塑性-稳定性窘境”stability-plasticity dilemma
- 增量学习 incremental learning / 在线学习 online learning

Boltzmann Machine

深度学习概述

- 提高模型复杂度的方法
  1. 增加隐层数目 (深度)
  2. 增加隐层神经元数目 (宽度)
- 多隐层的神经网络很难用 BP 算法等经典算法训练, 存在梯度消失和梯度爆炸的问题.
- 通过 Normalized initialization, intermediate normalization layers 以及使用 BP 的随机梯度下降的方法能够训练数十层深的网络
- 无监督逐层训练 预训练 + 微调代表: 深度置信网络 deep belief network DBN(RBM 堆叠)
- 权共享 weights sharing 代表 LeCNN 卷积层和采样层提取信息, 全连接层实现对输出的映射每一组神将元采用相同的连接权
- 多隐层的深度学习是一种特征学习, 将底层特征表示为转换为高层特征表示

目　录

聚类 Clustering

聚类是一种无监督学习的任务，将无标记训练样本划分为若干不相交的子集，每个子集称为一个簇 cluster

聚类形式化定义

样本集 $D = \{x_1, x_2, \dots, x_m\}$

样本 $\mathbf{x_i} = (x_{i1}; x_{i2}; \dots; x_{in})$

聚类的目标：划分 k 个不相交的簇

$\{C_l | l = 1, 2, \dots, k\}, C_{l'} \cap_{l' \neq l} C_l = \varnothing, D = \cup_{l=1}^{k} C_l$

$\lambda_j \in \{1, 2, \dots, k\}$ 表示样本 $\mathbf{x_j}$ 的簇标记，$\mathbf{x_j} \in C_{\lambda_j}$

聚类结果：$\lambda = (\lambda_1; \lambda_2; \dots; \lambda_m)$

聚类性能度量

有效性指标 validity index
1. 评估聚类结果的好坏;
2. 作为聚类过程的优化目标;
希望”簇内相似度高”且”簇间相似度”低
外部指标 vs. 内部指标

外部指标

定义记号

数据集:$D=\{x_1,x_2,\ldots,x_m\}$, 划分 $\mathbf{C}=\{C_1,C_2,\ldots,C_k\}$, 簇标记向量 λ
参考模型划分簇 $C^\star=\{C_1^\star,C_2^\star,\ldots,C_s^\star\}$, 簇标记向量 $\lambda^\star$.

$$
\begin{aligned}
a &= |SS|, SS=\{(x_i,x_j)|\lambda_i=\lambda_j,\lambda_i^\star=\lambda_j^\star,i<j\}\\
b &= |SD|, SD=\{(x_i,x_j)|\lambda_i=\lambda_j,\lambda_i^\star\neq\lambda_j^\star,i<j\}\\
c &= |DS|, DS=\{(x_i,x_j)|\lambda_i\neq\lambda_j,\lambda_i^\star=\lambda_j^\star,i<j\}\\
d &= |DD|, DD=\{(x_i,x_j)|\lambda_i\neq\lambda_j,\lambda_i^\star\neq\lambda_j^\star,i<j\}
\end{aligned}
$$

- Jaccard 系数 (Jaccard Cofficient) $JC=\frac{a}{a+b+c}$
- FM 指数 (Fowlkes and Mallows Index) $FMI=\sqrt{\frac{a}{a+b}\cdot\frac{a}{a+c}}$
- Rand 指数 (Rand Index) $RI=\frac{2(a+d)}{m(m-1)}$
- 上面的性能度量结果均在 (0,1) 区间值越大越好.

内部指标

定义记号

聚类结果划分 $C = \{C_1, C_2, \ldots, C_k\}$
dist 用于计算两个样本之间的距离;
μ 代表簇 C 的中心点 $\mu = \frac{1}{|C|}\sum_{1\le i\le |C|} x_i$

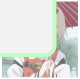
$$avg(C) = \frac{2}{|C|(|C|-1)} \sum_{1\le i\le j\le C} dist(x_i, x_j)$$

$$diam(C) = max_{1\le i\le j\le |C|} dist(x_i, x_j)$$

$$d_{min}(C_i, C_j) = min_{x_i\in C_i, x_j\in C_j} dist(x_i, x)$$

$$d_{cen}(C_i, C_j) = dist(\mu_i, \mu_j)$$

1. $avg(C)$ 是簇 C 内样本间的平均距离;
2. $diam(C)$ 是簇 C 内样本间的最远距离;
3. $d_{min}(C_i, C_j)$ 是簇 C_i 与簇 C_j 最近样本间的距离;
4. $d_{cen}(C_i, C_j)$ 是簇 C_i 与簇 C_j 中心点间的距离.



内部指标

- DB 指数 (Davies-Bouldin Index)

$$DBI = \frac{1}{k}\sum_{i=1}^{k} max_{j\neq i}\left(\frac{avg(C_i)+avg(C_j)}{d_{cen}(C_i, C_j)}\right)$$

  DBI 值越小越好

- Dunn 指数 (Dunn Index)

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

  DI 值越大越好

距离计算

距离度量满足的性质:

- 非负性: $dist(x_i, x_j) \geq 0$
- 同一性: $dist(x_i, x_j) = 0 iff. x_i = x_j$
- 对称性: $dsit(x_i, x_j) = dist(x_j, x_i)$
- 三角不等式: $dist(x_i, x_j) \leq dist(x_i, x_k) + dist(x_k, x_j)$

闵可夫斯基距离 Minkowski distance

样本 $\mathrm{x_i} = (x_{i1}; x_{i2}; \ldots; x_{in}), \mathrm{x_j} = (x_{j1}; x_{j2}; \ldots; x_{jn})$

$$dist_{mk}(\mathrm{x_i}, \mathrm{x_j}) = \left(\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}, p \geq 1$$

特别地, 当 $p = 1$ 时闵可夫斯基距离即使曼哈顿距离 (Manhattan distance)

$$\mathrm{dist}_{\mathrm{man}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1 = \sum_{u=1}^{n} |x_{iu} - x_{ju}|$$

当 $p = 2$ 时闵可夫斯基距离就是欧式距离 (Eulclidean distance)

$$\mathrm{dist}_{\mathrm{ed}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2}$$

距离计算

- 无序属性 VDM(Value Difference Metric)

$$\mathrm{VDM}_p(a, b) = \sum_{i=1}^{k} \left| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right|^p$$

- 闵可夫斯基 VDM

$$\mathrm{MinkovDM}_p\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left(\sum_{u=1}^{n_c} \left|x_{iu} - x_{ju}\right|^p + \sum_{u=n_c+1}^{n} \mathrm{VDM}_p\left(x_{iu}, x_{ju}\right) \right)^{\frac{1}{p}}$$

- 加权闵可夫斯基距离

$$\mathrm{dist}_{\mathrm{wmk}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left(w_1 \cdot \left|x_{i1} - x_{j1}\right|^p + \ldots + w_n \cdot \left|x_{in} - x_{jn}\right|^p\right)^{\frac{1}{p}}, w_i \geqslant 0(i =$$

K-means 算法

K-均值算法 (K-means) 是一种原型聚类算法

K-means

给定样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$
最小化平方误差:

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2$$

$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$ 是簇 C_i 的均值向量.

Input: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$，聚类簇数 k

从D 中随机选择k 个样本作为初始均值向量$\{\boldsymbol{\mu_1},\boldsymbol{\mu_2},\ldots,\boldsymbol{\mu_k}\}$

repeat

    令$C_i=\varnothing(1\leq i\leq k)$

    for $j=1,2,\ldots,m$ **do**

        计算样本$\boldsymbol{x_j}$ 与各均值向量$\mu_i(1\leq i\leq k)$ 的距离:$d_{ji}=\parallel \boldsymbol{x_j}-\boldsymbol{\mu_i}\parallel_2$;

        根据距离最近的均值向量确定$\boldsymbol{x_j}$ 的簇标记:$\lambda_j=argmin_{i\in\{1,2,\ldots,k\}}d_{ji}$;

        将样本$\boldsymbol{x_j}$ 划分到相应的簇$C_{\lambda_j}=C_{\lambda_j}\cup\{\boldsymbol{x_j}\}$;

    end for

    for $i=1,2,\ldots,k$ **do**

        计算新均值向量: $\boldsymbol{\mu}_i'=\frac{1}{|C_i|}\sum_{x\in C_i}\boldsymbol{x}$;

        if $\boldsymbol{\mu}_i'\neq\boldsymbol{\mu}_i$ **then**

            将当前均值向量$\boldsymbol{\mu}_i$ 更新为$\boldsymbol{\mu}_i'$

        else

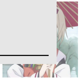
            保持当前均值向量不变

        end if

    end for

until 当前均值向量均为更新

Outpu: 簇划分 $C=\{C_1,C_2,\ldots,C_k\}$



谢　谢