90 分钟急救机器学习

Version: 2.0

Update: February 11, 2023

Contents

1 梯度下降

梯度下降是一个最优化算法，规范形式为：

Algorithm 1 梯度下降法

Input: $\boldsymbol{\theta}$，$J(\boldsymbol{\theta})$，α，其中 $\boldsymbol{\theta}$ 中存储的为随机值

Output: $\boldsymbol{\theta}$ 为使 $J(\boldsymbol{\theta})$ 取到最小值时的参数

```
while θ 不收敛 do
    for j = 0 to n do
        θ_j := θ_j − α∇J(θ_j)
    end for
end while
```

2 线性回归

2.1 预测方程

$$h_\theta(\boldsymbol{x}) = \sum_{i=0}^{n} \theta_i x_i \tag{1}$$

其中规定 $x_0 = 1$。

若令

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}, \boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \tag{2}$$

则预测方程为

$$h_\theta(x) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{X} \tag{3}$$

2.2 代价函数

线性回归的代价函数为

$$J(\theta_j) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \tag{4}$$

此时梯度公式为

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}) \boldsymbol{x}_j^{(i)} \tag{5}$$

2.3 归一化

1. 均值归一化为

$$x_i \leftarrow \frac{x_i - \mu_i}{\max - \min} \tag{6}$$

2. min-max 归一化为

$$x_i \leftarrow \frac{x_i - \min}{\max - \min} \tag{7}$$

2.4 学习率

1. 学习率太小时，收敛的速度可能会很慢；

2. 学习率太大时，可能会导致算法不收敛，代价函数 $J(\boldsymbol{\theta})$ 可能不会每次迭代都减小。

2.5 正规方程法求 θ

令

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{m\times(n+1)}, y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m \tag{8}$$

则

$$\boldsymbol{\theta} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}y \tag{9}$$

考虑到矩阵乘法的复杂度为 $\mathcal{O}(n^3)$，所以当 n 不是很大时可以考虑使用正规方程法，否则使用梯度下降法。

$X^{\mathrm{T}}X$ 不可逆的情况有两种：

1. 某些特征之间存在线性关系，导致部分特征是冗余的；

2. 训练集中的样本数目小于特征数目。

3 逻辑回归

3.1 Sigmoid 函数

Sigmoid 函数为

$$g(x) = \frac{1}{1+e^{-x}} \tag{10}$$

性质：$g(x) \in (0,1)$，$g(0)=\dfrac{1}{2}$，$\lim\limits_{x\to-\infty} g(x)=0$，$\lim\limits_{x\to+\infty} g(x)=1$。

3.2 预测函数

$$h_\theta(\boldsymbol{x}) = g(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}+b) \tag{11}$$

3.3 代价函数

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta(\boldsymbol{x}^{(i)}) + (1-y^{(i)})\log(1-h_\theta(\boldsymbol{x}^{(i)}))\right] \tag{12}$$

此代价函数称为**对数似然损失函数**，也称为**交叉熵损失函数**。

此时梯度公式为

$$\nabla J(\theta_j) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)})\boldsymbol{x}_j^{(i)} \tag{13}$$

3.4 多分类问题

对于多分类问题，可以使用一对多的方法，即对于每个类别 k，训练一个逻辑回归分类器。对于一个输入，只需找到输出最大的那个分类器，并将其对应的类别作为输入的类别。

对于第 i 个分类器，输出的意义是**输入 x 属于第 i 类的概率为** $h_\theta^i(x)$。

4 正则化

4.1 概念

过拟合（overfitting）指的是当有多个特征时，预测函数的代价可能会很低，但对于新的预测数据全都错误的情况，这会导致模型的泛化能力较差。通过进行正则化可以提高模型的**泛化能力**。

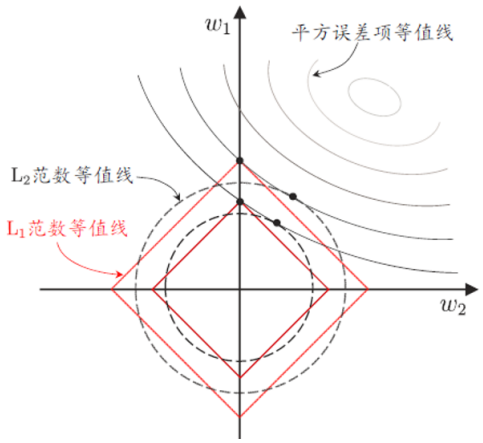
4.2 LASSO

以线性回归为例，以平方误差为损失函数，并将岭回归中的 L2 范数替换为 L1 范数，有

$$\min_{\boldsymbol{w}}\sum_{i=1}^{m}(y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i)^2 + \lambda\|\boldsymbol{w}\|_1 \tag{14}$$

其中正则化参数 $\lambda > 0$，上式称为 LASSO。

LASSO 易获得稀疏解，即它求得的 $\boldsymbol{w}$ 会有更少的非零分量，如下图所示：



可以发现，L1 范数的等值线与平方误差项的等值线的交点“常”出现在坐标轴上，即产生一个 w_1 或 w_2 为 0 的稀疏解，而 L2 范数的等值线与平方误差项的等值线的交点“不太可能”出现在坐标轴上，w_1 和 w_2 都非 0。

4.3 岭回归

以线性回归为例，加入 L2 正则项后的代价函数为

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{15}$$

其中 λ 称为正则化参数，$\lambda\sum_{j=1}^{n}\theta_j^2$ 被称为 L2 正则项。

如果正则化参数过大，惩罚的程度过高，我们会得到一个除了 θ_0 以外，其他的参数都趋于零的结果，相当于把假设函数的全部项都忽略掉了，这就导致了欠拟合的现象，欠拟合会导致**高偏差**。

如果正则化参数过小，惩罚的程度过低，相当于没有进行正则化，这就导致了过拟合的现象，过拟合会导致**高方差**。

4.4 使用岭回归对线性回归进行正则化

1. 梯度下降法：加入正则项后的梯度下降式为

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)} \\ \theta_j &:= \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \ (j = 1, 2, 3, \ldots, n) \\ &:= \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\end{aligned} \tag{16}$$

2. 正规方程法：加入正则项后 $\boldsymbol{\theta}$ 的解为

$$\boldsymbol{\theta} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\begin{bmatrix}0 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1\end{bmatrix}\right)^{-1}\mathbf{X}^{\mathrm{T}}y \tag{17}$$

矩阵为一个主对角线上除了第一行第一列元素为 0，其他全为 1 的 $(n+1)\times(n+1)$ 的矩阵。

4.5 使用岭回归对逻辑回归进行正则化

加入正则项后的梯度下降式为

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_0^{(i)} \\ \theta_j &:= \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j\right], \ (j = 1, 2, 3, \ldots, n) \\ &:= \left(1 - \alpha\frac{\lambda}{m}\right)\theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\end{aligned} \tag{18}$$

5 Softmax 回归

Softmax 回归用于多分类问题，如果分类的类别是互斥的，适于选择 softmax 回归分类器。如果类别之间不是互斥的，而是相互掺杂，则 k 个 logistic 回归分类器更加合适。当 $k = 2$ 时，Softmax 回归退化为逻辑回归。

5.1 假设函数

$$h_\theta\left(x^{(i)}\right)=\begin{bmatrix}p(y^{(i)}=1\mid x^{(i)};\theta)\\p(y^{(i)}=2\mid x^{(i)};\theta)\\\vdots\\p(y^{(i)}=k\mid x^{(i)};\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{k}e^{\theta_j^{\mathrm{T}}x^{(i)}}}\begin{bmatrix}e^{\theta_1^{\mathrm{T}}x^{(i)}}\\e^{\theta_2^{\mathrm{T}}x^{(i)}}\\\vdots\\e^{\theta_k^{\mathrm{T}}x^{(i)}}\end{bmatrix}\tag{19}$$

其中 $\theta_1,\theta_2,\cdots,\theta_k\in\mathbb{R}^{n+1}$。

5.2 代价函数

若令 $\boldsymbol{\theta}=\begin{bmatrix}\theta_1^{\mathrm{T}}\\\theta_2^{\mathrm{T}}\\\vdots\\\theta_k^{\mathrm{T}}\end{bmatrix}_{k\times(n+1)}$，则代价函数为

$$J(\boldsymbol{\theta})=-\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}1\left[y^{(i)}=j\right]\log\frac{e^{\theta_j^{\mathrm{T}}x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^{\mathrm{T}}x^{(i)}}}\right)\tag{20}$$

其中 $1[x]$ 为示性函数，表示 x 为真时为 1，否则为 0。

5.3 梯度下降

在 Softmax 回归中，将 x 归类为类别 j 的概率为

$$p\left(y^{(i)}=j\mid x^{(i)};\theta\right)=\frac{e^{\theta_j^{\mathrm{T}}x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^{\mathrm{T}}x^{(i)}}}\tag{21}$$

则梯度公式为

$$\nabla_{\theta_j}J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}\left(1\left[y^{(i)}=j\right]-p\left(y^{(i)}=j\mid x^{(i)};\theta\right)\right)\right]\tag{22}$$

5.4 权重衰减

所谓权重衰减，事实上就是正则化。

$$J(\boldsymbol{\theta})=-\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}1\left[y^{(i)}=j\right]\log\frac{e^{\theta_j^{\mathrm{T}}x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^{\mathrm{T}}x^{(i)}}}\right)+\frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2\tag{23}$$

此时梯度公式为

$$\nabla_{\theta_j} J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[x^{(i)}\left(1\left[y^{(i)}=j\right]-p\left(y^{(i)}=j \mid x^{(i)};\theta\right)\right)\right]+\lambda\theta_j \tag{24}$$

Example 5.1 证明 *logistic* 回归是 *softmax* 回归在 $k=2$ 时的特例。

Proof. 令 $k=2$，则

$$h_\theta\left(x^{(i)}\right)=\frac{1}{e^{\theta_1^{\mathrm{T}}x^{(i)}}+e^{\theta_2^{\mathrm{T}}x^{(i)}}}\begin{bmatrix}e^{\theta_1^{\mathrm{T}}x^{(i)}}\\ e^{\theta_2^{\mathrm{T}}x^{(i)}}\end{bmatrix} \tag{25}$$

令 θ_1，θ_2 均减去 θ_1，得到

$$\begin{aligned}
h_\theta\left(x^{(i)}\right) &= \frac{1}{e^{\vec{0}^{\mathrm{T}}x^{(i)}}+e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}}\begin{bmatrix}e^{\vec{0}^{\mathrm{T}}x^{(i)}}\\ e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}\end{bmatrix}\\
&= \frac{1}{1+e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}}\begin{bmatrix}1\\ e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}\end{bmatrix}\\
&= \begin{bmatrix}\dfrac{1}{1+e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}}\\ \dfrac{e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}}{1+e^{(\theta_2-\theta_1)^{\mathrm{T}}x^{(i)}}}\end{bmatrix}\\
&= \begin{bmatrix}1-p\\ p\end{bmatrix}.
\end{aligned} \tag{26}$$

6 决策树

6.1 基本算法流程

Algorithm 2 决策树学习基本算法 TreeGenerate(D, A)

Input: 训练集 $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$

属性集 $A = \{a_1, a_2, \ldots, a_d\}$

Output: 以 node 为根结点的一棵决策树

生成结点 node;
if D 中样本全属于同一类别 C **then**
  将 node 标记为 C 类叶结点;
  return
end if
if $A = \varnothing$ **OR** D 中样本在 A 上取值相同 **then**
  将 node 标记为叶结点，其类别标记为 D 中样本数最多的类;
  return
end if
从 A 中选择最优划分属性 a_*;
for a_* 中的每一个值 a_*^v **do**
  为 node 生成一个分支;
  令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
  if D_v 为空 **then**
    将分支结点标记为叶结点，其类别标记为 D 中样本最多的类;
    return
  else
    以 TreeGenerate $(D_v, A \backslash \{a_*\})$ 为分支结点;
  end if
end for

6.2 划分选择

6.2.1 信息增益

令信息熵

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2(p_k) \tag{27}$$

假设离散属性 a 有 V 个可能的取值 $\{a^1, a^2, \ldots, a^V\}$，如果使用属性 a 为样本集进行划分，则会产生 V 个分支结点，其中第 v 个分支结点包含了 D 中所有属性为 a^v 的样本，记为 D^v。我们用下式表示属性 a 对样本集 D 进行划分所获得的“信息增益”:

$$\mathrm{Gain}(D, a) = \mathrm{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v) \tag{28}$$

选择最优属性时我们可以采用属性 $a_* = \arg\max_{a \in A} \text{Gain}(D, a)$。

6.2.2 增益率

由于信息增益准则对**可取值数目较多**的属性有所偏好，为了避免这种影响，由 Quinlan 在 1993 年提出的 C4.5 决策树算法不直接使用信息增益，而是使用“增益率”来选择最优划分属性，定义为：

$$\text{Gain\_ratio}(D, a) = \frac{\text{Gain}(D, a)}{\text{IV}(a)} \tag{29}$$

其中，$\text{IV}(a)$ 表示属性 a 的“固有值”。属性 a 的可能取值数目越多（即 V 越大），则 $\text{IV}(a)$ 的值通常会越大：

$$\text{IV}(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \tag{30}$$

需注意的是，增益率准则对**可取值数目较少**的属性有所偏好，因此，C4.5 算法采用了一种启发式的做法：先找出信息增益高于平均水平的属性，再从其中选择增益率最高的。

6.2.3 基尼指数

数据集 D 的纯度可用基尼值来度量：

$$\begin{aligned} \text{Gini}(D) &= \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} \\ &= 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2 \end{aligned} \tag{31}$$

直观上，$\text{Gini}(D)$ 反映了从数据集 D 中随机抽取两个样本，其类别标记不一致的概率，所以 $\text{Gini}(D)$ 越小，则数据集 D 的纯度越高。

属性 a 的基尼指数定义为

$$\text{Gini\_index}(D, a) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} \text{Gini}(D^v) \tag{32}$$

我们选择划分后基尼指数最小的属性作为最优划分属性，即 $a_* = \arg\min_{a \in A} \text{Gini\_index}(D, a)$。

6.3 剪枝

剪枝分为“预剪枝”和“后剪枝”两种。

“预剪枝”是指在决策树生成过程中，对每个结点在划分前进行估计，如果划分后泛化能力无法提升，则停止划分并将当前结点标记为叶结点；

“后剪枝”则是先从训练集生成一棵完整的决策树，然后自下向上地对非叶结点进行考察，如果把该结点的子树替换为叶结点能提高泛化能力，则将该子树替换为叶结点。

对比：

- 时间开销：
  - 预剪枝：训练时间开销降低，测试时间开销降低
  - 后剪枝：训练时间开销增加，测试时间开销降低
- 过/欠拟合风险：
  - 预剪枝：过拟合风险降低，欠拟合风险增加
  - 后剪枝：过拟合风险降低，欠拟合风险基本不变
- 泛化性能：后剪枝通常优于预剪枝

6.4 连续值处理

设给定样本集 D 和连续属性 a，a 在 D 上出现了 n 个不同的取值，从小到大排序后记为 $\{a^1, a^2, \ldots, a^n\}$。

我们可考察包含 $n-1$ 个元素的候选划分点集合（即将区间的中点作为候选划分点）

$$T_a = \left\{ \frac{a^i + a^{i+1}}{2}, 1 \leq i \leq n-1 \right\} \tag{33}$$

对于信息增益，我们可以稍加改造得到属性连续时的公式：

$$\begin{aligned} \mathrm{Gain}(D, a) &= \max_{t \in T_a} \mathrm{Gain}(D, a, t) \\ &= \max_{t \in T_a} \mathrm{Ent}(D) - \sum_{\lambda \in \{-,+\}} \frac{|D_t^\lambda|}{|D|} \mathrm{Ent}(D_t^\lambda) \end{aligned} \tag{34}$$

6.5 缺失值处理

缺失值的处理涉及到两个问题：

（1）如何在属性值缺失的情况下进行划分属性选择?

（2）给定划分属性，若样本在该属性上的值缺失，如何对样本进行划分?

给定训练集 D 和属性 a，令 $\tilde{D}$ 表示 D 中在属性 a 上没有缺失值的样本子集。对于问题（1），我们可以仅通过 $\tilde{D}$ 来判断属性的优劣。

假定 a 有 V 个可取值 $\{a^1, a^2, \ldots, a^V\}$，令 $\tilde{D}^v$ 表示 $\tilde{D}$ 中在属性 a 上取值为 a^v 的样本子集，

$\tilde{D}_k$ 表示 $\tilde{D}$ 中属于第 k 类（$k=1,2,\ldots,|\mathcal{Y}|$）的样本子集，则显然有

$$\begin{aligned}\tilde{D}&=\bigcup_{k=1}^{|\mathcal{Y}|}\tilde{D}_k\\ \tilde{D}&=\bigcup_{v=1}^{V}\tilde{D}^v\end{aligned}\tag{35}$$

假定我们为每个样本 x 赋予一个权重 w_x，并定义

$$\begin{aligned}\rho&=\frac{\sum_{x\in\tilde{D}}w_x}{\sum_{x\in D}w_x}\\ \tilde{p}_k&=\frac{\sum_{x\in\tilde{D}_k}w_x}{\sum_{x\in D}w_x}\quad(1\le k\le|\mathcal{Y}|)\\ \tilde{r}_v&=\frac{\sum_{x\in\tilde{D}^v}w_x}{\sum_{x\in D}w_x}\quad(1\le v\le V)\end{aligned}\tag{36}$$

ρ 表示无缺失值样本所占的比例，$\tilde{p}_k$ 表示无缺失值样本中第 k 类所占比例，$\tilde{r}_v$ 表示无缺失值样本中在属性 a 上取值为 a^v 的样本所占的比例。

基于上述定义，我们可将信息增益的计算式推广为

$$\begin{aligned}\mathrm{Gain}(D,a)&=\rho\times\mathrm{Gain}(\tilde{D},a)\\ &=\rho\times\left(\mathrm{Ent}\left(\tilde{D}\right)-\sum_{v=1}^{V}\tilde{r}_v\,\mathrm{Ent}\left(\tilde{D}^v\right)\right)\end{aligned}\tag{37}$$

其中

$$\mathrm{Ent}(\tilde{D})=-\sum_{k=1}^{|\mathcal{Y}|}\tilde{p}_k\log_2\tilde{p}_k\tag{38}$$

对问题（2）：

- 若样本 x 在划分属性 a 上的取值已知，则将 x 划入与其取值对应的子结点，且样本权值在子结点中保持为 w_x；
- 若样本 x 在划分属性 a 上的取值未知，则将 x 同时划入所有子结点，且样本权值在与属性值 a^v 对应的子结点中调整为 $\tilde{r}_v\cdot w_x$。

6.6 从“树”到“规则”，从“单变量”到“多变量”

可以发现，一颗决策树对应着一个“规则集”，每个从根结点到叶结点的分支路径对应于一条规则。将决策树转化为一个规则集可以改善可理解性，进一步提升泛化能力。

在每个非叶结点仅考虑一个划分属性时得到的决策树称为“**单变量决策树**”，产生的是“轴平行”分类面，当分类边界比较复杂时，单变量决策树必须使用很多段划分才能获得较好的近似，此时决策树会相当复杂，时间开销会很大。若能够使用斜的划分边界，则模型将大为简

化，“多变量决策树”可以做到这一点。在该模型中，每个非叶结点不再是仅对某个属性，而是**构造一个线性分类器**，如此决策树模型可以大为简化。更复杂地，我们可以在结点上嵌入神经网络或其他非线性模型，获得更优的决策边界。

7 数据降维-主成分分析

7.1 PCA 算法

主成分分析（Principal Component Analysis，PCA）是最常用的一种降维方法。

对于正交属性空间中的样本点，如果用一个超平面对所有样本进行恰当地表达，它应当具有下列性质：

- **最近重构性**：样本点到这个超平面的距离都足够近；
- **最大可分性**：样本点到这个超平面上的投影能尽可能分开。

Algorithm 3 PCA 算法

Input: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_m\}$, 低维空间维数 d'

Output: 投影矩阵 $W=\{\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_{d'}\}$

1: 对所有样本进行中心化：$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i$

2: 计算样本的协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$

3: 对协方差矩阵 $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ 做特征值分解

4: 选取前 d' 个最大的特征值对应的特征向量作为投影矩阵 W

7.2 特征值分解

特征值分解：

- 先求 A 的特征多项式 $f(\lambda)=|A-\lambda E|=|A-\Lambda|$，其中 Λ 是一个主对角线上全为 λ 的方阵；
- 求特征方程 $|A-\lambda E|=0$ 的全部解，他们就是 A 的全部特征值；
- 对于一个特征值 λ_i，求出相应的特征方程组 $(A-\Lambda_i)X=0$ 的一组特解 $\xi_1,\xi_2,\cdots,\xi_t$，则这组解对应的向量即为一个特征向量。

降维后低维空间的维数 d' 通常是由用户事先指定，或通过在 d' 值不同的低维空间中对 k 近邻分类器（或其他开销较小的学习器）进行交叉验证来选取较好的 d' 值。对 PCA，还可以还可从重构的角度设置一个重构阈值，例如 $t=95\%$，然后选取使下式成立的最小 d' 值：

$$\frac{\sum_{i=1}^{d'}\lambda_i}{\sum_{i=1}^{d}\lambda_i}\geq t \tag{39}$$

PCA 仅需保留 $\mathbf{W}$ 与样本的均值向量，即可通过简单的向量减法和矩阵-向量乘法将新样本

投影至低维空间中。

降维虽然会导致信息的损失，但一方面舍弃这些信息后能**使得样本的采样密度增大**，另一方面，当数据受到噪声影响时，最小的特征值所对应的特征向量往往与噪声有关，**舍弃可以起到去噪效果**。

8 聚类分析

8.1 聚类性能度量

首先是下面会用到的几个量：

$$\begin{aligned}
\mu_i &= \frac{1}{|C|}\sum_{1\le i\le |C|}\boldsymbol{x}_i \\
\operatorname{avg}(C) &= \frac{2}{|C|(|C|-1)}\sum_{1\le i<j\le |C|}\operatorname{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j) \\
\operatorname{diam}(C) &= \max_{1\le i<j\le |C|}\operatorname{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j) \\
d_{\min}(C_i,C_j) &= \min_{\boldsymbol{x}_i\in C_i,\boldsymbol{x}_j\in C_j}\operatorname{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j) \\
d_{\text{cen}}(C_i,C_j) &= \operatorname{dist}(\mu_i,\mu_j)
\end{aligned} \tag{40}$$

其中，$\operatorname{avg}(C)$ 表示簇 C 内样本间的平均距离，$\operatorname{diam}(C)$ 表示簇 C 内样本间的最远距离，$d_{\min}(C_i,C_j)$ 表示簇 C_i 和 C_j 最近样本间的距离，$d_{\text{cen}}(C_i,C_j)$ 表示簇 C_i 和 C_j 中心点间的距离，μ_i 表示簇 C 的中心点（质心）。

- DB 指数：

$$\text{DBI} = \frac{1}{k}\sum_{i=1}^{k}\max_{j\ne i}\left(\frac{\operatorname{avg}(C_i)+\operatorname{avg}(C_j)}{d_{\text{cen}}(\mu_i,\mu_j)}\right) \tag{41}$$

  DB 指数越小，聚类效果越好。

- Dunn 指数：

$$\text{DI} = \min_{1\le i\le k}\left\{\min_{j\ne i}\left(\frac{d_{\min}(C_i,C_j)}{\max\limits_{1\le l\le k}\operatorname{diam}(C_l)}\right)\right\} = \frac{\min_{1\le k<k'\le m}d_{\min}(C_k,C_{k'})}{\max_{1\le l'\le m}\operatorname{diam}(C_{l'})} \tag{42}$$

  Dunn 指数越大，聚类效果越好。

在衡量两个样本 $\boldsymbol{x}_i=(x_{i1};x_{i2};\ldots;x_{in})$ 和 $\boldsymbol{x}_j=(x_{j1};x_{j2};\ldots;x_{jn})$ 的距离时，常用的是“闵可夫斯基距离”：

$$\operatorname{dist}_{\text{mk}}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \left(\sum_{u=1}^{n}|x_{iu}-x_{ju}|^p\right)^{\frac{1}{p}} \tag{43}$$

上式即为 $\boldsymbol{x}_i-\boldsymbol{x}_j$ 的 $\boldsymbol{L}_p$ 范数 $\|\boldsymbol{x}_i-\boldsymbol{x}_j\|_p$。当 $p=1$ 时，称为“曼哈顿距离”；当 $p=2$ 时，称为“欧式距离”。

8.2 原型聚类 - Kmeans 算法

此类算法假设聚类结构能通过一组原型刻画，在现实聚类任务中极为常用。通常情况下，算法先对原型进行初始化，再对原型进行迭代更新求解。

Kmeans 算法针对聚类所得的簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ 最小化均方误差

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} ||\boldsymbol{x} - \mu_i||_2^2 \tag{44}$$

其中 $\mu_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$ 是簇 C_i 的均值向量，即“原型”。E 值在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度，E 值越小，则簇内样本相似度越高。

Algorithm 4 k-Means 算法

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

Output: 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k\}$
while 当前均值向量被更新或第一次进入循环 **do**
  令 $C_i = \varnothing (1 \leq i \leq k)$
  for $j = 1, 2, \cdots, m$ **do**
    计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i (1 \leq i \leq k)$ 的距离：$d_{ji} = \|\boldsymbol{x}_j - \boldsymbol{\mu}_i\|_2$
    根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记：$\lambda_j = \arg\min_{i \in \{1,2,\cdots,k\}} d_{ji}$
    将样本 $\boldsymbol{x}_j$ 划入对应的簇：$C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$
  end for
  for $i = 1, 2, \cdots, k$ **do**
    计算新的均值向量 $\boldsymbol{\mu}_i' = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$
    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
end while

8.3 密度聚类 - DBSCAN 算法

此类算法假设聚类结构能通过样本分布的紧密程度来确定。通常情况下，密度聚类算法从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇来获得最终的聚类结果。

DBSCAN 算法是一种著名的密度聚类算法，它基于一组“邻域”参数 $(\epsilon, MinPts)$ 来刻画样本分布的紧密程度。给定数据集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$，定义下面几个概念：

- ϵ-邻域：对 $\boldsymbol{x}_j \in D$，其 ϵ-邻域包含样本集 D 中与 $\boldsymbol{x}_j$ 的距离不大于 ϵ 的样本，即 $N_\epsilon(\boldsymbol{x}_j) = \{\boldsymbol{x}_i \in D \mid \text{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) \leqslant \epsilon\}$；
- 核心对象：若 $\boldsymbol{x}_j$ 的 ϵ-邻域至少包含 $MinPts$ 个样本，即 $|N_\epsilon(\boldsymbol{x}_j)| \geq MinPts$，则 $\boldsymbol{x}_j$ 是一个核心对象；
- 密度直达：若 $\boldsymbol{x}_j$ 在 $\boldsymbol{x}_i$ 的 ϵ-邻域中，且 $\boldsymbol{x}_i$ 是核心对象，则称 $\boldsymbol{x}_j$ 由 $\boldsymbol{x}_i$ 密度直达；
- 密度可达：对 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$，若存在样本序列 $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n$，其中 $\boldsymbol{p}_1 = \boldsymbol{x}_i$，$\boldsymbol{p}_n = \boldsymbol{x}_j$ 且 $\boldsymbol{p}_{i+1}$ 由 $\boldsymbol{p}_i$ 密度直达，则称 $\boldsymbol{x}_j$ 由 $\boldsymbol{x}_i$ 密度可达；
- 密度相连：对 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$，若存在 $\boldsymbol{x}_k$ 使得 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$ 均由 $\boldsymbol{x}_k$ 密度可达，则称 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$ 密度相连。

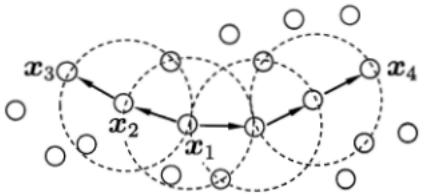


图 9.8 DBSCAN 定义的基本概念($MinPts = 3$): 虚线显示出 ϵ-邻域, $\boldsymbol{x}_1$ 是核心对象, $\boldsymbol{x}_2$ 由 $\boldsymbol{x}_1$ 密度直达, $\boldsymbol{x}_3$ 由 $\boldsymbol{x}_1$ 密度可达, $\boldsymbol{x}_3$ 与 $\boldsymbol{x}_4$ 密度相连.

DBSCAN 将“簇”定义为：由密度可达关系导出的最大的密度相连样本集合。

形式化地，给定邻域参数 $(\epsilon, MinPts)$，簇 $C \subseteq D$ 是满足以下性质的非空样本子集：

- 连接性：$\boldsymbol{x}_i \in C$，$\boldsymbol{x}_j \in C \Longrightarrow \boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 密度相连
- 最大性：$\boldsymbol{x}_i \in C$，$\boldsymbol{x}_j$ 由 $\boldsymbol{x}_i$ 密度可达 $\Longrightarrow \boldsymbol{x}_j \in C$

实际上，若 $\boldsymbol{x}$ 为核心对象，则 $\boldsymbol{x}$ 密度可达的所有样本的集合记为

$$X = \left\{\boldsymbol{x}' \in D \mid \boldsymbol{x}'\text{由}\boldsymbol{x}\text{密度可达}\right\} \tag{45}$$

则 X 即为满足连接性和最大性的簇。

Algorithm 5 DBSCAN 算法

Input: 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$，邻域参数 $(\epsilon, MinPts)$

Output: 簇划分 $C = \{C_1, C_2, \cdots, C_k\}$

1: 初始化核心对象集合：$\Omega = \varnothing$
2: **for** $j = 1, 2, \cdots, m$ **do**
3: 确定 $\boldsymbol{x}_j$ 的 ϵ-邻域 $N_\epsilon(\boldsymbol{x}_j)$
4: **if** $|N_\epsilon(\boldsymbol{x}_j)| \geq MinPts$ **then**
5: 将 $\boldsymbol{x}_j$ 加入核心对象集合：$\Omega = \Omega \bigcup \{\boldsymbol{x}_j\}$
6: **end if**
7: **end for**
8: 初始化聚类簇数：$k = 0$
9: 初始化未访问样本集合：$\Gamma = D$
10: **while** $\Omega \neq \varnothing$ **do**
11: 记录当前未访问样本集合：$\Gamma_{\text{old}} = \Gamma$
12: 随机选取一个核心对象 $\boldsymbol{o} \in \Omega$，初始化队列 $Q =< \boldsymbol{o} >$
13: $\Gamma = \Gamma \setminus \{\boldsymbol{o}\}$
14: **while** $Q \neq \varnothing$ **do**
15: 取出队首 $\boldsymbol{q}$
16: **if** $N_\epsilon(\boldsymbol{q}) \geq MinPts$ **then**
17: 令 $\Delta = N_\epsilon(\boldsymbol{q}) \bigcap \Gamma$
18: 将 Δ 中的样本加入队列 Q
19: $\Gamma = \Gamma \setminus \Delta$
20: **end if**
21: **end while**
22: $k = k + 1$，生成聚类簇 $C_k = \Gamma_{\text{old}} \setminus \Gamma$
23: $\Omega = \Omega \setminus C_k$
24: **end while**

1-7 行中，算法先根据给定的邻域参数找出所有核心对象；10-24 行中，以任一核心对象为出发点，找出由其密度可达的样本生成聚类簇，直到所有核心对象均被访问过为止。

8.4 层次聚类 - AGNES 算法

层次聚类试图在不同层次对数据集进行划分，从而形成树形的聚类结构。数据集划分既可采用“自底向上”的聚合策略，也可采用“自顶向下”的分拆策略。

AGNES 算法是一种基于贪心的采取自底向上聚合策略的层次聚类算法：首先，将样本中的每一个样本看做一个初始聚类簇，然后在算法运行的每一步中找出距离最近的两个聚类簇进行合并，该过程不断重复，直到达到预设的聚类簇的个数。

两个聚类簇 C_i 和 C_j 之间的距离可通过下面的公式得到：

- 最小距离：$d_{\min}(C_i, C_j) = \min\limits_{\boldsymbol{x}\in C_i, \boldsymbol{z}\in C_j} \operatorname{dist}(\boldsymbol{x}, \boldsymbol{z})$；
- 最大距离：$d_{\max}(C_i, C_j) = \max\limits_{\boldsymbol{x}\in C_i, \boldsymbol{z}\in C_j} \operatorname{dist}(\boldsymbol{x}, \boldsymbol{z})$；
- 平均距离：$d_{\text{avg}}(C_i, C_j) = \dfrac{1}{|C_i|\,|C_j|} \sum\limits_{\boldsymbol{x}\in C_i} \sum\limits_{\boldsymbol{z}\in C_j} \operatorname{dist}(\boldsymbol{x}, \boldsymbol{z})$。

当聚类簇距离由 $d_{\min}$，$d_{\max}$ 或 d_{avg} 计算时，AGNES 算法被相应地称为“**单链接**”、“**全链接**”或“**均链接**”算法。

Algorithm 6 AGNES 算法

Input: 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$，聚类簇距离度量函数 d，聚类簇数 k

Output: 簇划分 $C = \{C_1, C_2, \cdots, C_k\}$

1: **for** $j = 1, 2, \cdots, m$ **do**
2: $\quad C_j = \{\boldsymbol{x}_j\}$
3: **end for**
4: **for** $i = 1, 2, \cdots, m$ **do**
5: $\quad$ **for** $j = 1, 2, \cdots, m$ **do**
6: $\quad\quad M(i, j) = d(C_i, C_j)$
7: $\quad\quad M(j, i) = M(i, j)$
8: $\quad$ **end for**
9: **end for**
10: 设置当前聚类簇个数：$q = m$
11: **while** $q > k$ **do**
12: $\quad$ 找出两个距离最近的聚类簇 C_{i^*} 和 C_{j^*}
13: $\quad$ 合并 C_{i^*} 和 C_{j^*}：$C_{i^*} = C_{i^*} \bigcup C_{j^*}$
14: $\quad$ **for** $j = j^* + 1, j^* + 2, \cdots, q$ **do**
15: $\quad\quad$ 将聚类簇 C_j 重编号为 C_{j-1}
16: $\quad$ **end for**
17: $\quad$ 删除距离矩阵 M 的第 j^* 行和第 j^* 列
18: $\quad$ **for** $j = 1, 2, \cdots, q - 1$ **do**
19: $\quad\quad M(i^*, j) = d(C_{i^*}, C_j)$
20: $\quad\quad M(j, i^*) = M(i^*, j)$
21: $\quad$ **end for**
22: $\quad q = q - 1$
23: **end while**

1-9 行，算法先对仅含一个样本的初始聚类簇和相应的距离矩阵进行初始化；11-23 行，AGNES 不断合并距离最近的聚类簇，并对合并得到的距离矩阵进行更新；上述过程不断重复，直到达到预设的聚类簇数。

9 支持向量机

9.1 基本概念

- 支持向量：距离超平面最近的训练样本点；
- 间隔：两个异类支持向量之间的距离 $\gamma = \frac{2}{\|\boldsymbol{w}\|}$

9.2 不等式约束的最优化问题 - KKT 条件

此类最优化问题的标准形式为

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ &\quad \text{s.t. } g_i(\boldsymbol{x}) \leq 0, h_j(\boldsymbol{x}) = 0 \quad i \in [1, m], j \in [1, p] \end{aligned} \tag{46}$$

其中 $g_i(x)$ 为不等式约束，$h_j(x)$ 为等式约束，m 和 p 为约束个数。

定义拉格朗日函数

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) + \sum_{k=1}^{p} \mu_k h_k(\boldsymbol{x}) \tag{47}$$

如果存在一组解 $\boldsymbol{x}^*$ 满足

$$\begin{cases} \nabla_x L = \dfrac{\partial L}{\partial x_i} = 0 \\ h_k(\boldsymbol{x}^*) = 0, \ k = 1, 2, \cdots, p \\ g_j(\boldsymbol{x}^*) \leq 0 \\ \mu_j \geq 0 \\ \mu_j g_j(\boldsymbol{x}^*) = 0, \ j = 1, 2, \cdots, m \end{cases} \tag{48}$$

则这组解 $\boldsymbol{x}^*$ 为满足条件的一组可行解。

9.3 硬间隔 SVM

hard-margin SVM 的最优化问题为

$$\begin{aligned} &\arg\min_{\boldsymbol{w}, b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\quad \text{s.t. } y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1, \ i = 1, 2, \ldots, m \end{aligned} \tag{49}$$

构造拉格朗日函数

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \sum_{i=1}^{m} \alpha_i (1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b)) \tag{50}$$

分别对 $\boldsymbol{w}$、b 求偏导可得

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Longrightarrow \boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i \\ \frac{\partial L}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{m} \alpha_i y_i = 0\end{aligned} \tag{51}$$

将上述两式代入拉格朗日函数得

$$\begin{aligned}L(\alpha) &= \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i y_i \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - \sum_{i=1}^{m}\alpha_i y_i b \\ &= \frac{1}{2}\left(\sum_{i=1}^{m}\alpha_i y_i \boldsymbol{x}_i^{\mathrm{T}}\right)\left(\sum_{j=1}^{m}\alpha_j y_j \boldsymbol{x}_j\right) + \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i y_i \left(\sum_{j=1}^{m}\alpha_j y_j \boldsymbol{x}_j^{\mathrm{T}}\right)\boldsymbol{x}_i \\ &= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j\end{aligned} \tag{52}$$

结合上面对 b 的偏导得到的约束，我们得到原最优化问题的对偶问题

$$\begin{aligned}\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j \\ \text{s.t. } \sum_{i=1}^{m}\alpha_i y_i = 0, \alpha_i \geq 0,\ i = 1, 2, \cdots, m\end{aligned} \tag{53}$$

略去上式必定存在最大值的证明，通过工具包计算得到 $\boldsymbol{\alpha}$ 后，可得到 $\boldsymbol{w}$ 和 b，从而得到分离超平面。

9.4 核化法

在现实任务中，原始的样本空间可能是非线性可分的，即找不到一个能正确划分两类样本的超平面。

此时我们通过核化法将数据映射到一个更高维的**特征空间**，使得样本在特征空间中线性可分，从而完成分类任务。

Theorem 9.1 如果原始空间是有限维的，那么必定存在一个高维特征空间使样本线性可分。

设样本 $\boldsymbol{x}$ 映射后的向量为 $\phi(\boldsymbol{x})$，划分超平面为 $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b$，则原始的最优化问题变为

$$\begin{aligned}\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t. } y_i(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i) + b) \geq 1,\ i = 1, 2, \cdots, m\end{aligned} \tag{54}$$

其对偶问题为

$$\begin{aligned}\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \phi\left(\boldsymbol{x}_i\right)^{\mathrm{T}} \phi\left(\boldsymbol{x}_j\right) \\ \text{s.t.} \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, \cdots, m\end{aligned} \tag{55}$$

预测方程为

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{x}) + b = \sum_{i=1}^{m} \alpha_i y_i \phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}) + b \tag{56}$$

注意到高维向量维数可能很高，且高维向量只以内积 $\phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$ 出现，引入核函数 $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$，通过此我们可以绕过显式考虑特征映射和计算高维内积困难的问题。

Table 1: 常用的核函数

名称	表达式	参数
线性核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$	
多项式核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + r)^d$	r 为偏移量，$d \geq 1$ 为多项式次数
高斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$	$\sigma > 0$ 为高斯核的带宽
拉普拉斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma}\right)$	$\sigma > 0$
Sigmoid 核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j + \theta)$	$\beta > 0$，$\theta < 0$

Example 9.1 假设 $r = 1$，$d = 2$，则多项式核函数为

$$\begin{aligned}(a \times b + 1)^2 &= (a \times b + 1)(a \times b + 1) \\ &= a^2 b^2 + 2ab + 1 \\ &= (\sqrt{2}a, a^2, 1) \cdot (\sqrt{2}b, b^2, 1)\end{aligned} \tag{57}$$

映射函数为

$$\begin{cases}\phi : \mathbb{R}^1 \to \mathbb{R}^3 \\ (x) \to (z_1, z_2, z_3) = (\sqrt{2}x, x^2, 1)\end{cases} \tag{58}$$

Example 9.2 当映射函数 $\phi(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ 时，多项式核函数的 r 和 d 分别是多少？

写做两个向量内积形式为

$$\begin{aligned}\phi(\boldsymbol{x})\phi(\boldsymbol{x}') &= x_1 x_1'^2 + 2x_1 x_2 x_1' x_2' + x_2 x_2'^2 \\ &= (x_1 x_1' + x_2 x_2')^2 \\ &= (\boldsymbol{x} \times \boldsymbol{x}')^2\end{aligned} \tag{59}$$

所以 $r = 0$，$d = 2$。

根据核函数我们可以推导得到特征空间中两个向量间的距离和夹角：

- 两向量间的距离为

$$\begin{aligned}\|\phi(\boldsymbol{x})-\phi(\boldsymbol{x}')\| &= (\phi(\boldsymbol{x})-\phi(\boldsymbol{x}'))^{\mathrm{T}}(\phi(\boldsymbol{x})-\phi(\boldsymbol{x}')) \\ &= \phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{x})-\phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{x}')-\phi(\boldsymbol{x}')^{\mathrm{T}}\phi(\boldsymbol{x})+\phi(\boldsymbol{x}')^{\mathrm{T}}\phi(\boldsymbol{x}') \\ &= \kappa(\boldsymbol{x},\boldsymbol{x})-\kappa(\boldsymbol{x},\boldsymbol{x}')-\kappa(\boldsymbol{x}',\boldsymbol{x})+\kappa(\boldsymbol{x}',\boldsymbol{x}') \\ &= \kappa(\boldsymbol{x},\boldsymbol{x})-2\kappa(\boldsymbol{x},\boldsymbol{x}')+\kappa(\boldsymbol{x}',\boldsymbol{x}')\end{aligned} \tag{60}$$

- 两向量间的夹角余弦为

$$\begin{aligned}\cos\theta &= \frac{\phi(\boldsymbol{x})\cdot\phi(\boldsymbol{x}')}{\|\phi(\boldsymbol{x})\|\|\phi(\boldsymbol{x}')\|} \\ &= \frac{\phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{x}')}{\sqrt{\phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{x})}\sqrt{\phi(\boldsymbol{x}')^{\mathrm{T}}\phi(\boldsymbol{x}')}} \\ &= \frac{\kappa(\boldsymbol{x},\boldsymbol{x}')}{\sqrt{\kappa(\boldsymbol{x},\boldsymbol{x})}\sqrt{\kappa(\boldsymbol{x}',\boldsymbol{x}')}}\end{aligned} \tag{61}$$

映射函数 ϕ 不是必须的，只有核矩阵

$$\begin{bmatrix}\kappa(x_1,x_1) & \kappa(x_1,x_2) & \cdots & \kappa(x_1,x_n) \\ \kappa(x_2,x_1) & \kappa(x_2,x_2) & \cdots & \kappa(x_2,x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n,x_1) & \kappa(x_n,x_2) & \cdots & \kappa(x_n,x_n)\end{bmatrix} \tag{62}$$

是半正定时，$\kappa(\cdot,\cdot)$ 才是一个可使用的核函数；给定一个 ϕ 也能找到其对应的 κ，给定一个 κ 也能找到一个对应的特征空间使得 κ 对应空间中的向量内积。

Definition 9.1 半正定：给定一个大小为 $n\times n$ 的实对称矩阵 A，若对于任意长度为 n 的向量 x，有 $x^{\mathrm{T}}Ax\geq 0$ 恒成立，则称矩阵 A 是一个半正定矩阵。

9.5 软间隔 SVM

由于 hard-margin SVM 无法容忍无法线性可分的情况，可能在确定超平面时出现过拟合的情况，于是我们允许一部分异类样本落入另一侧的区域，形成 soft-margin SVM。此时最优化问题可写为

$$\begin{aligned}&\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\ \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}+C\sum_{i=1}^{m}\xi_i, C>0 \\ &\text{s.t. } \xi_i\geq 0, y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)\geq 1-\xi_i,\ i=1,2,\cdots,m\end{aligned} \tag{63}$$

其中 ξ_i 被称为**松弛变量**。当 C 增大时，$\sum\xi_i$ 必定减小，ξ_i 必定减小，则由限制可知间隔减小。

上问题的拉格朗日函数为

$$L(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})=\frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}+C\sum_{i=1}^{m}\xi_i+\sum_{i=1}^{m}\alpha_i[1-\xi_i-y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)]-\sum_{i=1}^{m}\beta_i\xi_i \tag{64}$$

令上式分别对 $\boldsymbol{w}$、b 和 $\boldsymbol{\xi}$ 求偏导可得

$$\begin{aligned}
&\frac{\partial L}{\partial \boldsymbol{w}} = 0 \Longrightarrow \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i = \boldsymbol{w} \\
&\frac{\partial L}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \\
&\frac{\partial L}{\partial \xi_i} = 0 \Longrightarrow C - \alpha_i - \beta_i = 0 \rightarrow 0 \leq \alpha_i, \beta_i \leq C
\end{aligned} \tag{65}$$

10 神经网络

10.1 神经网络的基本概念

神经网络包含一个输入层，若干个隐藏层和一个输出层。每一层的每一个神经元都会经过**激活函数**从而输出一个值，常见的激活函数包括 Sigmoid 函数、ReLU 函数等。

下面对神经网络中一些记号进行解释：

- $a_i^{(j)}$ 表示第 j 层第 i 个神经元或单元的激活项，激活项是指由一个具体的神经元计算并输出的值；
- $\Theta^{(j)}$ 表示控制从第 j 层到第 $j+1$ 层的映射的权重矩阵，$\Theta_{ij}^{(k)}$ 表示 $a_j^{(k-1)}$ 和 $a_i^{(k)}$ 之间的权重。

如果第 j 层有 s_j 个单元，$j+1$ 层有 s_{j+1} 个单元，则 $\Theta^{(j)}$ 的维数为 $s_{j+1} \times (s_j + 1)$。

10.2 神经网络的前向传播

在神经网络中，信息从上一个神经元直接流转到下一个神经元，直到输出，依据每一个神经元的输入并根据相应规则可以计算出输出，最终得到在当前参数下的损失函数的过程，称为**前向传播**。其计算公式为：

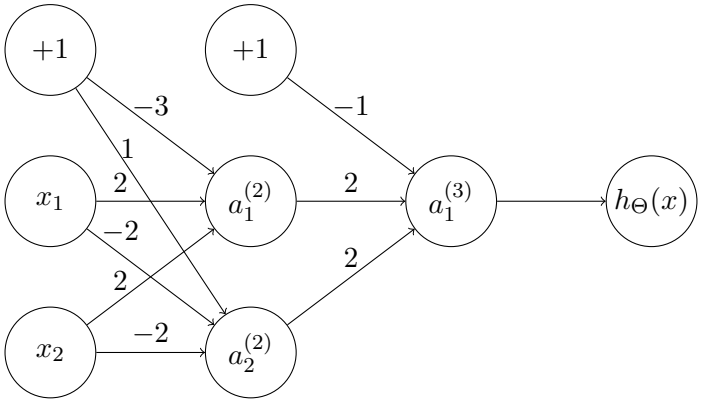
$$\begin{cases}
z^{(i+1)} = \Theta^{(i)} a^{(i)} \\
a^{(i+1)} = g\left(z^{(i+1)}\right)
\end{cases} \tag{66}$$

其中 $g(x)$ 为激活函数，$a^{(i)}$ 为第 i 层的激活项，$z^{(i+1)}$ 为第 $i+1$ 层的输入项。

Example 10.1 构造一个神经网络用以计算 $x_1 \operatorname{XNOR} x_2$。

考虑逻辑推导同或，由于 $A \bigoplus B = \overline{A}B + A\overline{B}$，由德摩根律得 $A \operatorname{XNOR} B = \overline{\left(\overline{A}B + A\overline{B}\right)} = \overline{\overline{A}B} + AB$。

根据此，我们可以构造一个神经网络：



通过简单的计算我们得到，$a_1^{(2)}$ 计算得到的结果是 AB，$a_2^{(2)}$ 计算得到的结果是 $\overline{AB}$，$a_1^{(3)}$ 计算得到的结果是 $\overline{AB}+AB$，是我们需要的结果。

10.3 神经网络解决多分类问题

在神经网络中，我们可以通过将输出层的神经元个数设置为类别的个数，从而解决多分类问题。例如，我们有三个类别，那么我们可以将输出层的神经元个数设置为 3，并且将每个神经元的激活函数设置为 Sigmoid 函数，从而得到每个类别的概率。

之后，我们对所有输出层的输出值求 max，得到最大的概率对应的类别即为预测结果。

10.4 神经网络的代价函数

在神经网络中，我们使用交叉熵作为代价函数：假设 $h_\Theta(x)\in\mathbb{R}^K$，$(h_\Theta(x))_i$ 表示神经网络中输出向量的第 i 个输出，即向量中的第 i 项，则代价函数表示为

$$\begin{aligned} J(\Theta) = &-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\log(h_\Theta(x^{(i)})_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] \\ &+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2 \end{aligned} \tag{67}$$

Proof. 对于此函数，直观的理解是将其与逻辑回归的代价函数做对比，由于神经网络中输出是一个 $\mathbb{R}^K$ 的向量，于是必须存在一个求和将这些输出单元的代价加起来，这就有了内层的求和号；

对于第 k 个输出单元，它预测的类别是 $h_\Theta(x^{(i)})_k$，实际的类别是 $(y^{(i)})_k$，将其在逻辑回归的代价函数中做替换即可得到神经网络代价函数中的前一项；

而后一项明显为 L2 正则项，逻辑回归中是将除 θ_0 以外的所有参数平方后求和，类比到神经网络中，以 $a_1^{(i+1)}$ 举例，它会和 $a_1^{(i)}, a_2^i, \ldots, a_{s_i}^{(i)}$ 这 s_i 个神经元相连，相连时的参数分别为

$\Theta_{11}^{(i)}, \Theta_{12}^{(i)}, \ldots, \Theta_{1s_i}^{(i)}$，那么这个神经元表示的逻辑回归即为

$$a_1^{(i+1)} = \sum_{j=0}^{s_i} \Theta_{1j}^{(i)} a_j^{(i)} \tag{68}$$

那么这一个逻辑回归对应的 L2 正则项为

$$\sum_{j=1}^{s_i} \left(\Theta_{1j}^{(i)}\right)^2 \tag{69}$$

将第 $i+1$ 层的 s_{i+1} 个神经元都考虑进来得到

$$\sum_{j=1}^{s_i} \sum_{k=1}^{s_{i+1}} \left(\Theta_{kj}^{(i)}\right)^2 \tag{70}$$

再把第 1 层到第 $L-1$ 层考虑进来便得到神经网络代价函数中的后一项。

10.5 反向传播算法

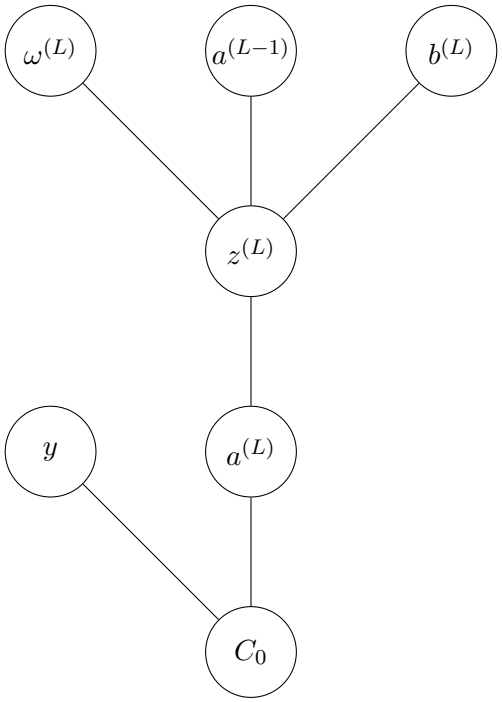
反向传播算法是一个让代价函数最小化的算法。为了使用梯度下降法来找到 $\arg\min_{\Theta} J(\Theta)$，我们需要计算 $J(\Theta)$ 和 $\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ 的值，从而能够使用梯度下降。

10.5.1 每层只有一个神经元的情况

考虑最后一层的激活值是如何计算的：

$$a^{(L)} = g\left(\omega^{(L)} a^{(L-1)} + b^{(L)}\right) = g\left(z^{(L)}\right) \tag{71}$$

这个过程如下图所示：



下面假设代价函数为均方误差。考虑计算当 $\omega^{(L)}$ 变化时，代价函数的变化程度，即

$$\frac{\partial}{\partial \omega^{(L)}} C_0 \tag{72}$$

根据求导的链式法则有

$$\frac{\partial}{\partial \omega^{(L)}} C_0 = \frac{\partial z^{(L)}}{\partial \omega^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} \tag{73}$$

下面计算右侧各式的值:

$$\frac{\partial C_0}{\partial a^{(L)}} = \frac{\partial}{\partial a^{(L)}} \frac{1}{2} \left(a^{(L)} - y \right)^2 = \left(a^{(L)} - y \right)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \frac{\partial}{\partial z^{(L)}} g\left(z^{(L)} \right) = \frac{\partial}{\partial z^{(L)}} z^{(L)} \cdot g'\left(z^{(L)} \right) = g'\left(z^{(L)} \right) \tag{74}$$

$$\frac{\partial z^{(L)}}{\partial \omega^{(L)}} = \frac{\partial}{\partial \omega^{(L)}} \left(\omega^{(L)} a^{(L-1)} + b^{(L)} \right) = a^{(L-1)}$$

10.5.2 每层有多个神经元的情况

我们令 j 表示最后一层的第 j 个神经元，k 表示倒数第二层的第 k 个神经元，此时最后一层的代价为

$$C_0 = \frac{1}{2} \sum_{j=0}^{s_L - 1} \left(a_j^{(L)} - y_j \right)^2 \tag{75}$$

再次提醒，$\omega_{jk}^{(L)}$ 表示 $L-1$ 层的第 k 个神经元连向 L 层的第 j 个神经元的权重，与单神经元类似地，令

$$z_j^{(L)} = \sum_{k=0}^{s_{L-1} - 1} \omega_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)} \tag{76}$$

则最后一层的激活值为

$$a_j^{(L)} = g\left(z_j^{(L)} \right) \tag{77}$$

其中 $g(x)$ 为 Sigmoid 函数。再次利用求导的链式法则，我们得到

$$\frac{\partial C_0}{\partial \omega^{(L)}} = \sum_{k=0}^{s_{L-1} - 1} a_k^{(L-1)} g'\left(z^{(L)} \right) \left(a^{(L)} - y \right)$$

$$\frac{\partial C_0}{\partial b^{(L)}} = 1 g'\left(z^{(L)} \right) \left(a^{(L)} - y \right) \tag{78}$$

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{s_L - 1} \omega_{jk}^{(L)} g'\left(z^{(L)} \right) \left(a^{(L)} - y \right)$$

10.5.3 梯度下降的具体过程

- 先进行一次前向传播

$$\begin{cases} z^{(l)} = \omega^{(l)} a^{(l-1)} + b^{(l)} \\ a^{(l)} = g\left(z^{(l)}\right) \end{cases} \tag{79}$$

- 计算输出层误差方程，按定义有

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} \tag{80}$$

由求导的链式法则得到

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}} g'\left(z_j^{(L)}\right) \quad \text{(BP1)} \tag{81}$$

写作矩阵形式为

$$\delta^{(L)} = \nabla_a C \odot g'\left(z^{(L)}\right) \tag{82}$$

其中 $\odot$ 表示按元素的乘积，有时也被称为 Hadamard 乘积或 Schur 乘积。举例：

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

- 再通过下一层的误差 $\delta^{(l+1)}$ 表示当前层的误差 $\delta^{(l)}$

$$\delta^{(l)} = \left((\omega^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right) \odot g'\left(z^{(l)}\right) \quad \text{(BP2)} \tag{83}$$

- 通过梯度下降更新权重 ω 和偏置值 b

$$\begin{cases} \omega_{jk}^{(l)} := \omega_{jk}^{(l)} - a_k^{(l-1)} \delta_j^{(l)} \\ \\ b_j^{(l)} := b_j^{(l)} - \delta_j^{(l)} \end{cases} \tag{84}$$

Proof. 下面给出式 83 的证明：

$$\begin{aligned} \delta^{(l)} &= \frac{\partial C}{\partial z^{(l)}} \\ &= \frac{\partial C}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \\ &= (\delta^{(l+1)} \cdot \omega^{(l+1)}) \odot g'\left(z^{(l)}\right) \end{aligned} \tag{85}$$

由于 $\omega^{(l+1)}$ 维数为 $s_l \times s_{l+1}$，故上式也可以写作

$$\delta^{(l)} = ((\omega^{l+1})^{\mathrm{T}} \delta^{(l+1)}) \odot g'\left(z^{(l)}\right) \tag{86}$$

得证。

10.6 梯度检查

梯度检查是一个用来保证每次的前向传播、反向传播都是正确的算法。当 $\theta \in \mathbb{R}^n$ 时，此时 $\theta = [\theta_1, \theta_2, \dots, \theta_n]$，我们只需要对向量中每一个参数的偏导数都进行估计即可。具体地，

$$\frac{\partial}{\partial \theta_i} J(\theta) = \lim_{\epsilon \to 0} \frac{J(\dots, \theta_{i-1}, \theta_i + \epsilon, \theta_{i+1}, \dots) - J(\dots, \theta_{i-1}, \theta_i - \epsilon, \theta_{i+1}, \dots)}{2\epsilon} \tag{87}$$

其中 ϵ 一般取 10^{-4} 即可。

在正式训练中不要使用梯度检查，因为计算代价相比于反向传播而言非常大。

10.7 随机初始化

在神经网络中，对参数进行初始化时，我们需要使用随机初始化的思想。

具体地，令每个 $\Theta_{ij}^{(l)}$ 等于 $[-\epsilon, \epsilon]$ 内的一个随机数，即

$$-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon \tag{88}$$

10.8 总结

- 选择一个合适的神经网络架构：
  - 输入层单元个数为特征向量的维度；
  - 输出层单元个数为所要区分的类别个数。注意，每个数据的类别应当是一个**向量**，而不是一个实数。
  - 一般来说，默认只用一个隐藏层，如果使用多个隐藏层的话，默认每个隐藏层都具有相同数量的神经元。
- 训练神经网络：
  - 随机地初始化权重矩阵；
  - 对于每个训练数据 $x^{(i)}$，执行前向传播以得到输出 $h_\Theta(x^{(i)})$；
  - 计算代价函数 $J(\Theta)$；
  - 执行反向传播计算所有的偏导数 $\dfrac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$；
  - 使用一次梯度检查来保证计算结果的正确性，然后禁用梯度检查；
  - 通过梯度下降或其他方法求出 $\arg\min_\Theta J(\Theta)$。

11 卷积神经网络

卷积神经网络（CNN）是一个专门用于图像处理的神经网络。

11.1 卷积层

在 CNN 中，我们设计一个特定大小的区域称作“感受野”（Receptive field），每一个神经元都只需要关心自己的感受野里面有什么东西。举例：感受野为 3×3，则一个神经元只需要输入一个 $3 \times 3 \times 3 = 27$ 维的向量，权重数目明显减少；

多个感受野的范围是可以重叠的，不同的神经元也可以有相同的感受野。

最经典的感受野设定：

- 考虑一张图片所有的通道，而不是只考虑部分通道；
- 感受野的长和宽被称作 kernel size，如上方讨论的感受野的 kernel size 为 3×3，一般不会设置很大的 kernel size；
- 同一个感受野会对应一组神经元，例如 64 个或 128 个；
- 对于两个相邻的感受野，将偏移量（采样间隔）称为 stride（步幅），这是一个超参数，通常不会很大，一般设置为 1 或 2 即可；
- 当感受野超出图像范围时，我们可以向感受野中做 padding（填补），一般补充为 0。

相同的特征可能会出现在图像的不同区域中，但是这些相同的特征对应的是不同的神经元，但他们做的是相同的工作。我们是否需要让每一个区域都放若干个对应不同特征的神经元?

考虑共享参数，两个对应不同感受野的对应相同特征的神经元，让他们共享相同的参数，这样参数的数量将会大大减少。由于两个神经元对应的感受野不同，他们的输入不一致，所以它们的输出也不会一样。

常见的共享参数的方法：对于同一层上的神经元，每一个感受野对应的神经元都只有一组参数，这组参数被称作 filter。

加入上述两个限制之后，我们得到的就是卷积层 Convolutional Layer。

另一种常见的解释方式是，定义若干个 $3 \times 3 \times$ channel 的 filter，将每个 filter 在输入的图片上做卷积操作，每次将 filter 移动 stride 长度，如果超出图像范围时，我们可以选择进行 padding。

卷积后得到的结果被称做 Feature map。卷积之后得到的 Feature map 可以看做是一张新的图片，只是通道数不再是输入的通道数（1 个通道 + n 个卷积核得到 n 个通道）。

当然，卷积层也是可以叠加很多层的。卷积层叠的越多，考虑的范围就会越来越大。

有时还可以在卷积之后做一次 ReLU 操作，使 Feature map 中不出现负数值结果：

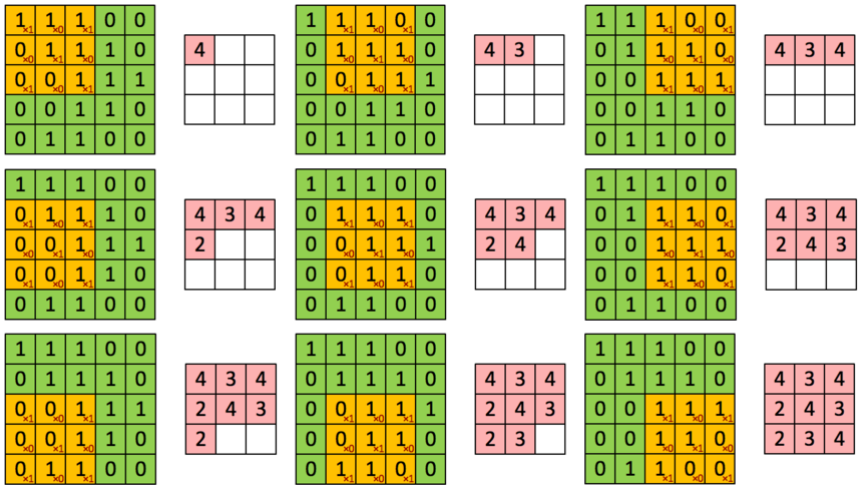
$$\mathrm{ReLU}(x) = \max(0, x) \tag{89}$$

卷积后图像大小由下面的公式进行计算：

$$\text{new\_feature\_size} = \lfloor \text{img\_size} - \text{filter\_size} \rfloor / \text{stride} + 1 \tag{90}$$

其中，img_size 表示原图大小，filter_size 表示卷积核大小，stride 表示卷积核移动的步长。

Figure 1: 卷积示意图



Example 11.1 输入图片规格经过预处理后尺寸为 $227 \times 227 \times 3$，卷积核大小为 $11 \times 11 \times 3$，步长为 4，则经过卷积后得到的特征图大小为

$$\lfloor 227 - 11 \rfloor / 4 + 1 = 55$$

即得到的特征图的尺寸为 $55 \times 55 \times 3$。

11.2 池化层

我们观察到，有时对于一张图片，我们对其做下采样（按照一定规律在图片中采样，使图片尺寸缩小），得到的图片特征是不变的。

根据此，我们可以对卷积得到的 Feature map 通过池化层进行压缩。

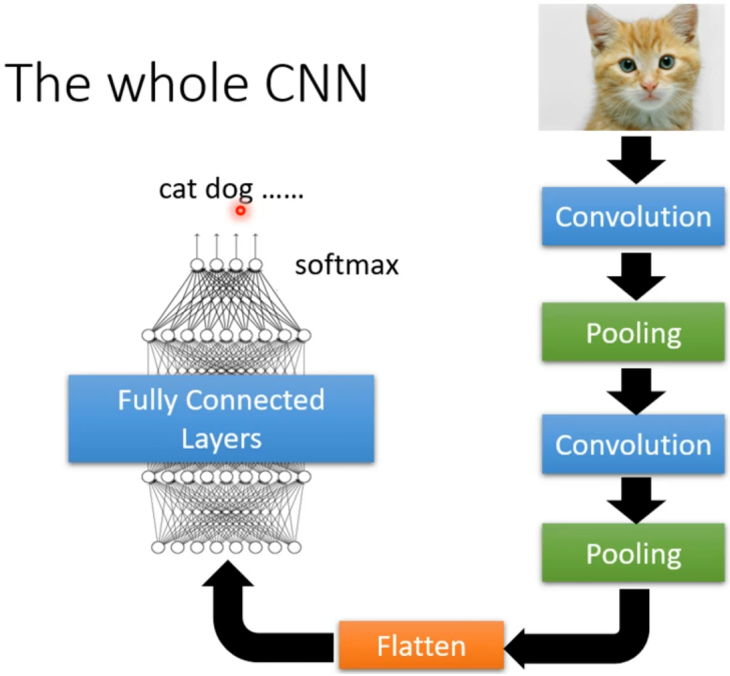
我们选取一个窗口大小（通常是 2×2 或 3×3），然后选择步幅（通常选择 2），对每个窗口中的所有数取 max，最终得到一个特征不变但是尺寸变小的一个 Feature map。

池化可以减少运算量，但相应的可能会损失掉一些细微的特征。池化可以改善结果，使之不容易过拟合。

经过几次卷积和池化后，我们将得到的 Feature map “拉直”（flatten）为一个向量，丢进全连接层，最后通过一个 softmax 回归器来得到最终的投票结果。

11.3 超参数

- 卷积层：卷积核的数量；卷积核的大小；卷积核移动的步长
- 池化层：窗口的大小；窗口移动的步长
- 全连接层：神经元个数



11.4 卷积神经网络的推广

不仅是图片，卷积神经网络也可以用于其他类型的数据，比如文本、语音、视频等。只要数据可以用矩阵表示，就可以用卷积神经网络来处理。

12 稀疏自编码器

12.1 自编码器

传统反向神经网络缺点：

- 梯度越来越稀疏（梯度弥散）：从顶层越往下，误差校正信号越来越小；
- 容易收敛到局部最小值，尤其是从远离最优区域开始的时候（将参数初始化为随机值会导致这种情况发生）；
- 只能进行有监督训练，但大部分的数据是没有标签的，而人类的大脑可以从没有标签的数据中学习。

1986 年，**Rumelhart** 提出了自动编码器的概念，并将其用于高维复杂数据的处理，促进了神经网络的发展。自编码神经网络是一种**无监督**（自监督）学习算法，它使用了反向传播算法，并让**目标值等于输入值**，即自编码神经网络尝试学习一个 $h_{W,b}(x) \approx x$ 的函数。

自编码器分类：

- 浅层自编码（Autoencoder）
- 稀疏自编码（Sparse Autoencoder）

- 栈式自编码（Stacked Autoencoder）
- 去噪自编码（Denoising Autoencoder）
- 变分自编码（Variational Autoencoder）

自编码器的共同点：学习一个**与输入相同的输出**，并尽可能的让其具有较强的**泛化能力**。

自编码器的构成：

- 编码器是把高维度数据**有损的**映射成低维度数据，减少数据量，要实现这种映射关系需要学习数据间的相关性，对输入 x 编码，得到新的特征 y；
- 解码器和编码器完全相反，是把低维度数据映射成高维度数据，**增加数据量**，使经过压缩的数据恢复成原来的样子，即利用新的特征 y，得到一个重构的数据 x'；
- 损失函数用来衡量由于压缩而损失掉的信息。编码器和解码器一般都是参数化的方程，且关于损失函数**可导**，典型的情况是使用最小均方误差

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x - x')^2 \tag{91}$$

12.2 深度自编码器

2006 年，**Hinton** 对原型自动编码器的结构进行了改进，进而产生了 DAE，先用**无监督逐层贪心训练算法**完成对隐含层的**预训练**，然后用**反向传播**算法对整个神经网络进行系统性参数优化调整，显著降低了神经网络的训练复杂程度，有效改善了反向传播算法容易陷入局部最小的不良状况。

将三层浅层自编码器扩展成具有多个隐藏层的编码器即为深度自编码器，能更好的学习数据更高维度的特征。

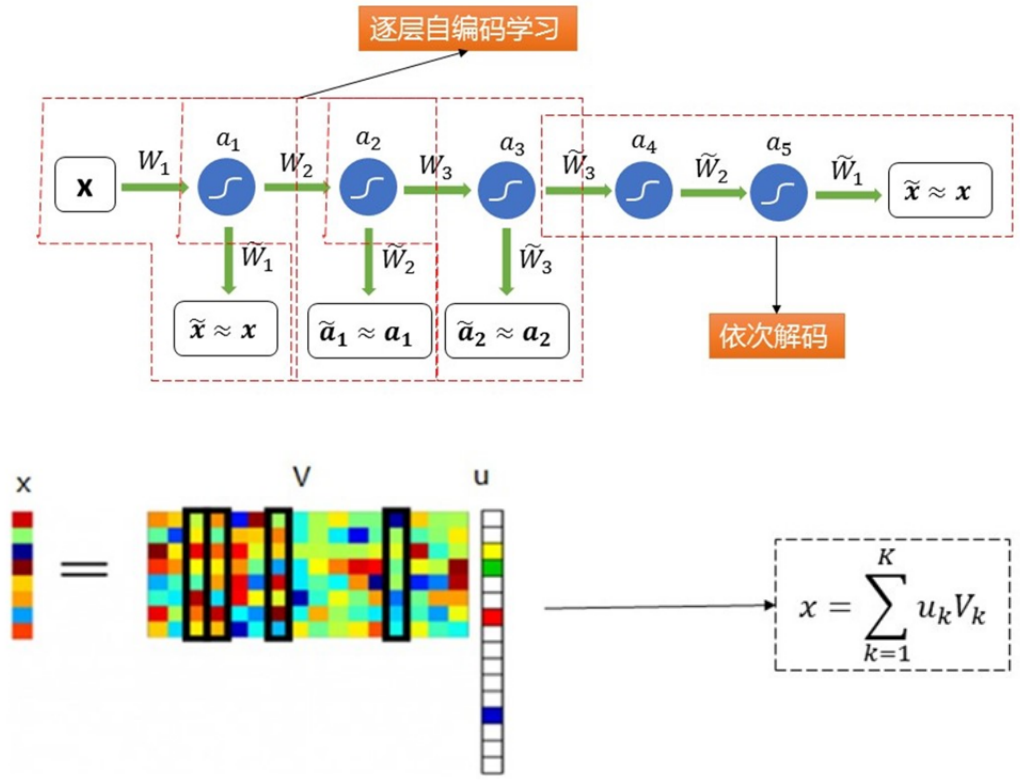
12.2.1 逐层学习

在深度自编码器中，对其中的多个隐藏层，我们需要进行逐层自编码学习。具体地，每一个编码层中都会包含一个解码器，用来输出当前层的解码结果，我们贪心地希望每一层的解码结果都与前一层向其输入的结果尽可能的相似：

深度自编码器特点：具有强大的**表达能力**及深度神经网络的**所有优点**。它通常能够获取到输入的“层次性分组”或者“部分-整体分解”结构。自编码器倾向于学习得到能更好地**表示输入数据的特征**。

12.3 稀疏表示

我们可以寻找一个系数矩阵 u 和一个字典矩阵 V，使得 $V \times u$ 尽可能的还原 x，且 u 尽可能地稀疏，则 u 就是 x 的稀疏表示。我们将一个大矩阵变成两个小矩阵，从而达到压缩的目的。



给定数据集 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$，字典学习的最简单形式为

$$J(\mathbf{B}, \boldsymbol{\alpha}) = \min_{\mathbf{B}, \boldsymbol{\alpha}_i} \sum_{i=1}^{m} \|x_i - \mathbf{B}\boldsymbol{\alpha}_i\|_2^2 + \lambda \sum_{i=1}^{m} \|\boldsymbol{\alpha}_i\|_1 \tag{92}$$

其中 $\mathbf{B} \in \mathbb{R}^{d \times k}$ 为字典矩阵，k 称为字典的**词汇量**，通常由用户指定。$\boldsymbol{\alpha}_i \in \mathbb{R}^k$ 则是样本 $\boldsymbol{x}_i \in \mathbb{R}^d$ 的稀疏表示。显然，上式的第一项是希望 $\boldsymbol{\alpha}_i$ 能较好地重构 $\boldsymbol{x}_i$，第二项则是希望 $\boldsymbol{\alpha}_i$ 尽量稀疏，可以再加入一项 $+\beta \sum_r \sum_c \mathbf{B}_{rc}^2$，表示希望字典中每项也尽可能小。

为了较为简单的最小化上述二元函数，我们通过变量交替优化的策略来进行：

- 首先随机初始化字典矩阵 $\mathbf{B}$；
- 然后通过梯度下降，得到 $\mathbf{B}$ 确定时使得函数 $J(\cdot, \cdot)$ 取得最小值时的 $\boldsymbol{\alpha}$；
- 再通过梯度下降，得到 $\boldsymbol{\alpha}$ 确定时使得函数 $J(\cdot, \cdot)$ 取得最小值时的 $\mathbf{B}$；
- 不断重复上述两步，直至结果确定。

12.4 稀疏自编码

一个神经网络中，使得神经元在大部分的时间内都是被**抑制**的限制称为**稀疏性限制**。

考虑一个最简单的自编码器，其神经网络只有三层，那么 $a_j^{(2)}$ 表示隐藏层中的隐藏神经元 j 的激活值，但这种表示方法无法表示是哪一个输入 x 带来的激活值，所以我们改用 $a_j^{(2)}(x)$ 表示在给定输入为 x 的情况下，隐藏层中的隐藏神经元 j 的激活值。

令

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} a_j^{(2)}(x^{(i)}) \tag{93}$$

表示隐藏神经元 j 在 m 个输入样本上的平均激活度。

再近似的加入一条限制：$\hat{\rho}_j = \rho$，其中 ρ 被称为稀疏性参数。为了满足这一限制，将会在优化目标函数中加入一个惩罚因子，这将惩罚那些 $\hat{\rho}_j$ 和 ρ 有显著不同的情况，从而使得隐藏神经元的平均活跃度保持在较小范围内。

12.4.1 KL 散度

定义两分布 P，Q 之间的相对熵（KL 散度）为

$$D(P\|Q) = \sum_x E\left[\ln p(x) - \ln q(x)\right] = \sum_x p(x)\ln\frac{p(x)}{q(x)} \tag{94}$$

其中 $p(x)$，$q(x)$ 为两分布的概率密度函数。相对熵可以解释为“通过 q 的编码去编码 p 后，信息丢失数的期望”，可以用相对熵来表示两个分布之间的差异。

把隐层的每个神经元的激活和未激活态看做服从**二项分布**，则 KL 散度可以表示为一个以 ρ 为均值和一个以 $\hat{\rho}_j$ 为均值的两个二项分布之间的相对熵：

$$\mathrm{KL}(\rho\|\hat{\rho}_j) = \rho\ln\frac{\rho}{\hat{\rho}_j} + (1-\rho)\ln\frac{1-\rho}{1-\hat{\rho}_j} \tag{95}$$

KL 散度的三个特点：

- $\mathrm{KL}(\rho\|\hat{\rho}_j) \geq 0$，当且仅当 $\rho = \hat{\rho}_j$ 时取等号；
- 非对称性：$\mathrm{KL}(\rho\|\hat{\rho}_j) \neq \mathrm{KL}(\hat{\rho}_j\|\rho)$；
- 当 $\hat{\rho}_j \to 0$ or 1 时，$\mathrm{KL}(\rho\|\hat{\rho}_j) \to +\infty$。

现在，代价函数可以表示为

$$J_{\text{sparse}}(\mathbf{B},\boldsymbol{\alpha}) = J(\mathbf{B},\boldsymbol{\alpha}) + \beta\sum_{j=1}^{s_2}\mathrm{KL}(\rho\|\hat{\rho}_j) \tag{96}$$

注意到

$$\frac{\partial\,\mathrm{KL}(\rho\|\hat{\rho}_j)}{\partial\hat{\rho}_j} = -\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j} \tag{97}$$

于是在神经网络的反向传播中，当前层误差 $\delta^{(l)}$ 更新为

$$\delta^{(l)} = \left((\omega^{(l+1)})^{\mathrm{T}}\delta^{(l+1)} + \beta\left(-\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j}\right)\right)\odot g'\left(z^{(l)}\right) \tag{98}$$

即可，其中 $l = n_l - 1, n_l - 2, \ldots, 2$。注意，在进行稀疏自编码器的训练过程中，必须要先进行一次前向传播以得到所有神经元的平均激活度 $\hat{\rho}$。