机器学习

RandomStar

2021 年 4 月 21 日

目录

1 机器学习的基本概念

1.1 什么是 Machine Learning?

机器学习 (Machine Learning) 实际上就是计算机系统的自我学习，通过输入的数据集来学习出一个“函数 (或者说映射)”，机器学习的三大问题是：

- 监督学习 Supervised Learning：学习已经存在的结构和规则，也就是学习一个映射，即对于有标注数据的学习，常见的有分类和回归
- 非监督学习 Unsupervised Learning：学习过程中由计算机自己发现新的规则和结构，即对于无标注数据的学习，常见的有聚类和降维
- 强化学习：有反馈的学习，但是只会反馈对和错，在机器人中比较常见

监督学习的数据都是有标注的，而非监督学习的数据是没有标准的，需要在学习过程中自己发现数据中存在的一些结构和特征

1.1.1 机器学习中最重要的问题

机器学习问题中最重要并需要花最多时间的事情就是定义模型，机器学习的三个要素是莫名的表示、度量和优化。

1.2 几个机器学习中的基本概念

- Sample，example，pattern 问题案例，样本
- Features，predictor，independent variable 将需要处理的数据用高维向量来表示，一般用 x_i 表示
- State of the nature，lables，pattern class 数据的类型，一般用 ω_i 表示
- Training data：用若干组 (x_i, ω_i) 表示训练数据集
- Test data 测试数据
- Training error & Test error 训练误差和测试误差

1.3 机器学习问题的分类

事实上 1.1 中介绍的三大机器学习问题只是机器学习的一种分类方法，常见的对于机器学习的分类方法还有如下几种：

1.3.1 按模型分类

根据模型的类型，机器学习还存在如下几种分类方式：按照是否为概率模型分类、按照是否为线性模型分类、按照是否为参数化模型分类，每种分类方式的特点如下：

- 概率模型和非概率模型：非概率模型也叫做确定性模型，区别在于概率模型在学习时使用条件概率分布形式，二非概率模型使用函数形式
- 线性模型和非线性模型：主要区别在于模型是否为线性函数，神经网络就是复杂的非线性模型
- 参数化模型和非参数化模型：区别在于参数化模型用确定个参数刻画模型，而非参数化模型的参数维度不固定

1.3.2 按算法分类

可以分为在线学习和批量学习，在线学习是每次接受一个样本进行预测，然后进行学习，并不断重复的过程，而批量学习则是一次性把数据集训练完之后再进行结果的预测

1.4 模型的复杂度 Complexity

随着模型的复杂度提高，其训练误差会不断下降，但是测试误差会先下降再提高。因此模型存在一个最优的复杂度。模型训练的过程中可能会出现过拟合 (overfitting) 的情况.

按照我个人的表达方式，过拟合其实就是对测试数据拟合得太好而对训练数据拟合效果不好。

Definition 1.4.1 泛化 (Generalization) 能力：表示一个模型对未出现过的数据样本的预测能力，我们一般希望泛化能力越大越好。

1.5 模型的评估

1.5.1 样本数据集的表示

我们用 $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ 表示样本数据集其中 y_i 表示 x_i 的真实标记，要评估学习的性能，需要将预测结果和真实标记进行比较

1.5.2 指示函数

指示函数 $\mathbb{I}(e)$ 在表达式 e 的值为真的时候值为 1，在表达式 e 为假的时候值为 0

1.5.3 错误率和准确率

模型训练的错误率定义为

$$E(f; D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(x_i) \neq y_i) \tag{1}$$

精确度的定义为:

$$acc(f; D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(x_i) = y_i) = 1 - E(f; D) \tag{2}$$

1.5.4 查准率和查全率

真实情况	结果为正例	结果为反例
正例	TP	FN
反例	FP	TN

- 查准率 $P = \frac{TP}{TP+FP}$ 表示预测结果为正例中预测正确的比例
- 查全率 $R = \frac{TP}{TP+FN}$ 表示所有正例中被预测对的比例

1.6 No Free Lunch

我们总希望我们的机器学习算法在所有情况下都表现得非常优秀，因为这样可以帮我们省很多事，然而事实上这是不可能的，因为对于同一个问题的两种解决算法 A 和 B，如果 A 在

某些情况下表现比 B 要好，那么 A 就一定会在另一些情况里表现得比 B 要差，这是因为对于一个问题，其所有情况的总误差和算法是没有关系的，也就是说，一个特定问题的所有可能情况的总误差也是一定的。

下面我们可以来简单地证明这一个结论，我们用 X 表示样本空间，H 表示假设空间，并且假设它们都是离散的，令 $P(h|D,\lambda_a)$ 表示算法 a 在训练集 D 下产生假设 h 的概率，再用 f 代表我们希望学习的真实目标函数，则可以用 $C = X - D$ 来表示训练集之外的所有样本，则其产生的误差可以表示为：

$$E(\lambda_a|D,f) = \sum_h \sum_{X\in C} P(x)\mathbb{I}(h(x) \neq f(x))P(h|D,\lambda_a) \tag{3}$$

考虑最简单的二分类问题，并且真实目标函数可以是任何映射到 0 和 1 上的函数，因此可能的函数有 $2^{|X|}$，对所有可能的 f 按照均匀分布求和，有

$$\begin{aligned}
\sum_f E(\lambda_a|D,f) &= \sum_f \sum_h \sum_{X\in C} P(x)\mathbb{I}(h(x) \neq f(x))P(h|D,\lambda_a) \\
&= \sum_{x\in C} P(x) \sum_h P(h|D,\lambda_a) \sum_f \mathbb{I}(h(x) \neq f(x)) \\
&= \sum_{x \int C} P(x) \sum_h P(h|D,\lambda_a) \frac{1}{2} 2^{|X|} \\
&= 2^{|X|-1} \sum_{x\in C} P(x) \sum_h P(h|D,\lambda_a) \\
&= 2^{|X|-1} \sum_{x\in C} P(x)
\end{aligned} \tag{4}$$

我们发现这其实是一个常数，也就是说不管选择了什么算法，其在特定问题和特定数据集下的总误差是一定的，因此两个算法一定会在一些问题上的表现互有胜负，这也就是 There is no free lunch 定理。

1.7 误差，偏差和方差

在机器学习中我们非常关注学习的效果，这可以通过误差的指标来衡量，常见的一种误差就是均方误差，比如在回归问题中，均方误差可以表示为：

$$E(f;\mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2 \tag{5}$$

如果采用概率密度函数，就可以计算连续状态下的均方误差：

$$E(f;\mathcal{D}) = \int_{x\sim D} (f(x_i) - y_i)^2 p(x)dx \tag{6}$$

而均方误差又可以进一步的分解。

1.7.1 方差 variance 和偏差 bias

对于数据集 D 和学习模型 f，学习算法期望预测为：

$$\overline{f}(x) = \mathbb{E}_D(f(x;\mathcal{D})) \tag{7}$$

则根据方差的定义，可以得到方差的表达式：

$$var(x) = \mathbb{E}_D[(f(x;\mathcal{D}) - \overline{f}(x))^2] \tag{8}$$

我们又可以定义**模型的期望预测值和真实标记之间的误差为偏差** (bias)，即

$$bias^2(x) = (\overline{f}(x) - y)^2 \tag{9}$$

则在回归问题的均方误差中，我们可以将均方误差分解为：

$$E(f;\mathcal{D}) = var(x) + bias^2(x) + \epsilon^2 \tag{10}$$

其中 $\epsilon^2 = \mathbb{E}_D[(y_D - y)^2]$ 表示样本产生的噪声 (noise)

1.8 连续形式的 bias 和 variance

变量取值连续的情况下，引入概率密度函数，bias 和 variance 的表达式可以写为：

$$bias^2 = \int (E_D(f(x,D)) - E(y|x))^2 p(x)dx \tag{11}$$

$$var = \int E_D[(f(x,D) - E_D(f(x,D)))^2]p(x)dx \tag{12}$$

2 贝叶斯理论 Bayes' Theory

贝叶斯理论是非常经典的分类算法，我们一般用 x 表示样本，用 ω_j 表示可能的分类类别，则 $P(\omega_j|x)$ 表示 x 属于这一类别的概率。贝叶斯决策论是在概率框架下实施决策的基本方法，本质上是如何基于概率和误判损失来选择最优的类别标记.

2.1 贝叶斯公式的推导

根据条件概率公式，我们有

$$P(AB) = P(A|B)P(B) = P(B|A)P(A) \Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{13}$$

这就是贝叶斯公式最简单的基本形式，其中

- $P(A)$ 是先验概率 (prior), 指的是样本中各种情况出现的概率
- $P(B|A)$ 是似然 (likelihood), 表示 A 发生的条件下，B 出现的概率
- $P(A|B)$ 是后验概率 (posterior)

现在我们假设有若干个特征 $\omega_1, \omega_2, \ldots, \omega_c$, 对于数据集 D 中的一个样本 x，有

$$P(\omega_j|x) = \frac{P(x|\omega_j)P(\omega_j)}{P(x)} \tag{14}$$

$$P(x) = \sum_{j=1}^{c} P(x|\omega_j)P(\omega_j) \tag{15}$$

贝叶斯定理可以用于分类问题的决策，而用贝叶斯定理进行决策的实质是通过概率分布情况来使得分类错误的可能性最小化，上面提到的公式事实上是基于后验概率来进行分类决策的，也称为 Optimal Bayes Decision Rule

贝叶斯决策也可能会碰到一些特殊情况，比如当先验概率相等的时候，只需要比较 likelihood 就可以，当 likelihood 一样大小的时候只需要比较先验概率就可以。

2.2 贝叶斯公式的 Loss

2.2.1 条件风险

可以定义一个 loss function 来估计贝叶斯公式的 loss，我们设 $\lambda(\alpha_i|\omega_j)$ 表示将原本类型为 ω_j 的样本分类成了 α_i 所带来的 loss(也可以叫 risk)，则将数据集 D 中的样本 x 分类为 α_i 所带来的条件风险 (Condition Risk) 可以表示为：

$$R(\alpha_i|x) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) P(\omega_j|x) \tag{16}$$

而对于所有可能的 α_i，可以对其条件风险进行积分，得到总的条件风险

$$R = \int R(\alpha_i|x) p(x) dx \tag{17}$$

可以记 $\lambda_{ij} = \lambda(\alpha_i|\omega_j)$ 则对于一个二分类问题，我们只需要比较 $R(\alpha_i|x)$ 的大小，将其展开之后发现只需要比较 $\frac{P(x|\omega_1)}{P(x|\omega_2)}$ 和 $\frac{\lambda_{12}-\lambda_{22}}{\lambda_{21}-\lambda_{11}} \times \frac{P(\omega_2)}{P(\omega_1)}$ 的大小。

2.2.2 0-1 loss

一种简单的 loss 定义是损失在分类正确的时候为 0，错误的时候为 1，即

$$\lambda_{ij} = \begin{cases} 0 & (i = j) \\ 1 & (i \neq j) \end{cases}$$

将其带入原本的条件风险表达式，我们可以得到

$$R(\alpha_i|x) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) P(\omega_i|x) = \sum_{j \neq i} P(\omega_i|x) = 1 - P(\omega_i|x) \tag{18}$$

此时我们进行决策的话只需要比较 $P(\omega_i|x)$ 的大小，而根据贝叶斯公式，我们只需要比较 $P(x|\omega_i)P(\omega_i)$ 的大小，因此下面我们就来解决这一部分的计算问题。

2.2.3 Coding 体验 I

在学完这一部分的内容之后我尝试了完成蔡登老师的作业，在给定的框架上实现一个最简单形式的贝叶斯二分类器，需要编写的部分包括 likelihood 的计算、posterior 的计算，以及错误分类和 risk 的计算，这个作业主要分成了三个部分，首先是实现 likelihood 并基于 likehood

进行二分类，然后是实现 posterior 并基于 posterior 进行分类，分别计算两种分类的误判数目并比较，然后是计算 risk，其中我有如下几点收获：

- 这个作业的样本分布是离散的，并且有一定的上下界，因此可以先获取样本值的上下界，然后统计样本的分布情况，再进行后续计算
- 在计算 likelihood 的时候，要计算的实际上是当前分类下，此类样本的所占比例，因此每种样本下每种特征的个数除以每种分类下样本总数，就是 likelihood
- 在计算 posterior 的时候，$P(\omega_j)$ 是每种类别占全部样本的比例，而 $P(x)$ 是当前特征属性值 x 的样本的 likelihood 和先验概率的乘积之和，也就是说 $P(x)$ 其实是多个值，并不是一整个值，而是该特征的每一种值都有一个 $P(x)$
- 在计算误判个数的时候要先根据分类依据确定每种特征属性值的最后分类结果，然后遍历所有的测试集找出分类错误的
- 在计算 risk 的时候，也是每种特征属性的值都有一个对应的 risk

2.3 参数估计 Parameter Estimation

经过刚才的推导我们发现，最后只需要计算 $P(x|\omega_i)P(\omega_i)$ 就可以进行贝叶斯决策，而 $P(\omega_i)$ 是可以直接在样本中计算出来的，因为监督学习中每个样本都是有 label 的，可以非常容易地计算出 $P(\omega_i)$，问题就在于如何计算 $P(x|\omega_i)$，也就是类别 ω_i 中出现的样本值为 x 的概率。

我们可以用数据集 D 的样本来对 $P(x|\omega_i)$ 进行估计，而估计的主要方法有极大似然法 (Maximum-Likelihood) 和贝叶斯估计法两种。

2.3.1 正态分布 Normal Distribution

我们需要先回忆一下概率论中学过的正态分布的相关知识，因为后面极大似然估计中会用到。正态分布也叫做高斯分布，很明显这个分布是数学王子高斯发现的，正态分布的形式如下：对于一维变量我们有：

$$P(x|\mu,\sigma^2)=\mathcal{N}(x|\mu,\sigma^2)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\{-\frac{(x-\mu)^2}{2\sigma^2}\} \tag{19}$$

并且 $E(x) = \mu, var(x) = \sigma^2$, 而对于 d 维的向量 x，多元高斯分布的参数是 d 维的均值向量 μ 和 $d \times d$ 的对称正定协方差矩阵 Σ

$$P(\boldsymbol{x}|\mu, \Sigma) = \mathcal{N}(x|\mu, \Sigma^2) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp[-\frac{1}{2}(\boldsymbol{x}-\mu)^T\Sigma^{-1}(\boldsymbol{x}-\mu)] \tag{20}$$

2.3.2 极大似然法

估计模型参数的时候最常用的方法就是极大似然估计，对于一个包含 m 个样本的数据集 X，我们可以假设它是由一个概率分布 $p_{\text{data}}(\text{x})$ 生成的，现在我们要用这个数据集来估计出一个 $p_{\text{model}}(\text{x})$ 来近似地比噢傲视真实的概率模型，我们可以给模型设定一族参数 θ，因此模型可以表示为 $p_{\text{model}}(\text{x}, \theta)$，这样一来，极大似然法可以定义成：

$$\theta_{\text{ML}} = \arg\max_{\theta} p_{\text{model}}(\text{x}, \theta) = \arg\max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(x^{(i)}, \theta) \tag{21}$$

但是多个概率的积不太容易计算，因此可以给目前函数取一个对数：

$$\theta_{\text{ML}} = \arg\max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}, \theta) \tag{22}$$

因为重新缩放代价函数的时候 argmax 不会改变，因此可以除以样本的大小 m 得到和训练数据经验分布相关的期望作为极大似然法的评价准则：

$$\theta_{\text{ML}} = \arg\max \mathbb{E}_{\text{x}-p_{\text{data}}}[\log p_{\text{model}}(x, \theta)] \tag{23}$$

一种解释最大似然估计的观点就是**将极大似然法看作最小化训练集上的经验分布和模型分布之间的差异**，而两者之间的差异可以根据 KL 散度进行度量，即：

$$D_{KL}(\hat{p}_{\text{data}}||p_{\text{model}}) = \mathbb{E}_{\text{x}-p_{\text{data}}}[\log \hat{p}_{\text{data}}(x) - \log p_{\text{model}}(x)] \tag{24}$$

2.3.3 贝叶斯定理的参数估计

我们假定需要进行分类的变量在某一类别下其样本的值是服从高斯分布的，则有

$$P(\omega_i|x) = P(x|\omega_i)P(\omega_i) = P(x|\omega_i, \theta_i)P(\omega_i), \theta_i = (\mu_i, \sigma_i) \tag{25}$$

其中 θ_i 为待估计的参数。我们定义整个数据集 D 中类别为 ω_i 的子集是 D_i, 其对于参数 θ_i 的似然为

$$P(D_i|\theta) = \prod_{x_k \in D_i} P(x_k|\theta_i) \tag{26}$$

极大似然法的基本思路就是让这个数据集的似然达到最大，而达到最大的时候的参数值就是我们要求的参数估计值，因为它使得数据集中可能属于这个类别的样本的概率达到了最大。而为了防止数值过小造成**下溢**，可以采用对数似然

$$l(\theta) = \ln P(D_i|\theta) = \sum_{x_k \in D_i} \ln P(x_k|\theta_i) \tag{27}$$

我们的目标就是 $\theta^* = \arg\max_{\theta} l(\theta)$

我们之前已经假设了某一类别下的 x 服从正态分布，则有

$$\begin{aligned} \ln P(x_k|\theta_i) &= \ln(\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp[-\frac{1}{2}(x_k-\mu_i)^T\Sigma^{-1}(x_k-\mu_i)]) \\ &= -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(x_k-\mu_i)^T\Sigma^{-1}(x_k-\mu_i) \end{aligned} \tag{28}$$

则对 μ_i 求偏导数得到

$$\frac{\partial \ln P(x_k|\theta_i)}{\partial \mu_i} = \Sigma^{-1}(x_k - \mu_i) \tag{29}$$

我们需要让对数似然函数取得最值，则对其求偏导数可得到

$$\sum_{x_k \in D_i} \Sigma^{-1}(x_k - \mu_i) = 0 \Rightarrow \mu_i = \frac{1}{n}\sum_{x_k \in D_i} x_k \tag{30}$$

同理可以对 Σ_i 进行求导可以得到

$$\frac{\partial \ln P(x_k|\theta_i)}{\partial \Sigma_i} = \frac{1}{2\Sigma} + \frac{1}{2\Sigma^2}(x_k-\mu_i)(x_k-\mu_i)^T \tag{31}$$

因此可以求得 Σ 的估计值

$$\Sigma^2 = \sum_{x_k \in D_i}(x_k-\mu_i)(x_k-\mu_i)^T = \sum_{x_k \in D_i}||x_k-\mu_i||^2 \tag{32}$$

这些参数按照上面的估计公式计算之后可以带入原本的 likelihood 表达式计算出 likelihood，进一步计算出 posterior

2.3.4 代码实现

```
1 # 似然的计算，可以直接基于似然进行判别和决策
2 def likelihood(x):
3     """
```

```
    LIKELIHOOD Different Class Feature Likelihood
    INPUT: x, features of different class, C-By-N numpy array
           C is the number of classes, N is the number of different feature

    OUTPUT: l, likelihood of each feature(from smallest feature to biggest feature)
       given by each class, C-By-N numpy array
    """
    C, N = x.shape
    l = np.zeros((C, N))
    # 这里其实给出的样本x的结构是每种分类下面不同特征属性值的分布情况，因此可以先求出每种类别的样本和
    # 再计算得到每种特征属性值对应的分布情况就可以
    class_sum = np.sum(x, axis=1)
    for i in range(C):
        for j in range(N):
            l[i, j] = x[i, j] / class_sum[i]

    return l
```

2.3.5 贝叶斯估计

极大似然法是频率学派的方法，而贝叶斯估计则是贝叶斯派的估计方法，区别在于极大似然法 MLE 认为估计的参数是一个 fixed value 但是贝叶斯派则认为它是随机的变量. 把训练集 D 作为变量，则有

$$P(\omega_i|x,D) = \frac{P(x|\omega_i,D)P(\omega_i,D)}{\sum P(x|\omega_i,D)P(\omega_i,D)} \tag{33}$$

又可以化简为

$$P(\omega_i|x,D) = \frac{P(x|\omega_i,D_i)P(\omega_i)}{\sum P(x|\omega_i,D_i)P(\omega_i)} \tag{34}$$

2.4 朴素贝叶斯 Naive Bayes

朴素贝叶斯分类器的基本思想是，既然我们的困难是 $P(x|\omega_j)$ 涉及到 x 所有属性的联合概率不好估计，那我们就把联合概率的计算难度降到最低，也就是假设 x 的所有属性 (也可以叫做特征) 是互相独立的，此时对于 d 维的样本 $x \in D$, 贝叶斯的公式变成了

$$P(\omega|x) = \frac{P(\omega)P(x|\omega)}{P(x)} = \frac{P(\omega)}{P(x)}\prod_{i=1}^{d} P(x_i|\omega) \tag{35}$$

类似地，对于所有类别来说 P(x) 是相同的，因此朴素贝叶斯分类的目标就是

$$h_{nb}(x) = \arg\max_{c \in Y} P(\omega) \prod_{i=1}^{d} P(x_i|\omega) \tag{36}$$

训练集 D 中，令 D_c 表示第 c 类样本构成的集合，则类的先验概率

$$P(c) = \frac{|D_c|}{|D|} \tag{37}$$

对于离散的属性而言，可以令 D_{c,x_i} 表示 D_c 中第 i 个特征的取值为 x_i 的样本组成的集合，则条件概率可以估计为

$$P(x_i|\omega) = \frac{|D_{c,x_i}|}{|D_c|} \tag{38}$$

Definition 2.4.1 拉普拉斯修正 $LaplasSmoothing$：样本存在局限性，不可能所有的特征都恰好在样本中出现，特别是朴素贝叶斯的完全独立假设使得样本可能的特征数量变得特别多，我们可以假定所有的特征大致上都是均匀分布的，通过在训练集上添加 K 个特征 (K 种不同的特征，每种类型各一个) 使得每种特征都可以出现，此时的先验概率估算公式变成了：

$$P(x_i|\omega) = \frac{|D_{c,x_i}| + 1}{|D_c| + 1} \tag{39}$$

3 线性回归 Linear Regression

前面说到机器学习中的任务主要是分类和回归，分类如前面的贝叶斯定理所展现的那样是输出一个类别 (有限的离散数值)，而回归的任务则是输出一个 real value

3.1 线性回归的基本思路

对于 n 维的样本 $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ 找到一系列线性函数 $\omega = [\omega_1, \omega_2, ..., \omega_n]$ 使得 $f(x) = \omega^T \boldsymbol{x} + b$，这个问题可以变形为 n+1 维使得 $f(x) = \omega^T \boldsymbol{x}$ 其中 $\boldsymbol{x} = [1, x_1, x_2, ..., x_n]$ 而 $\omega = [b, \omega_1, \omega_2, ..., \omega_n]$，这样以来表达式的形式更加统一了，而我们的目标就是训练出这个函数 f，使得对于训练数据 (x_i, y_i)，学习出一个函数 f 使得 $f(x_i) = y_i$，我们要求解的对象就是这个 n+1 维的向量 ω

3.2 线性回归的 loss 函数和解

线性回归常见的 loss 函数定义是最小平方误差 (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i, \omega))^2 \tag{40}$$

也会使用残差平方和 (RSS)，其形式如下：

$$J_n(\alpha) = \sum_{i=1}^{n}(y_i - f(x_i, \omega))^2 = (\boldsymbol{y} - X^T\omega)^T(\boldsymbol{y} - X^T\omega) \tag{41}$$

我们对 RSS 的表达式求梯度有：

$$\nabla J_n(\alpha) = -2X(\boldsymbol{y} - X^T\alpha) = 0 \tag{42}$$

因此可以得到使得 RSS 最小的线性回归的解是:

$$\omega = (XX^T)^{-1}Xy \tag{43}$$

我们需要注意到，每一个样本 x 是 $d \times 1$ 维的向量，因此 X 是 $d \times n$ 维的矩阵，而 y 也是 $n \times 1$ 维的向量，所以当样本数 n 小于特征数 d 的时候，XX^T 是不满秩的，求不出逆矩阵，此时的线性回归有多个解，其实这也很好理解，因为要求解的是 n+1 维的向量 ω，有 n+1 个变量，因此至少需要 n+1 个样本才能确定 n+1 个参数，出现上述情况的时候所有可能的解都可以使得均方误差最小化，此时可以考虑引入正则化项来筛选出需要的结果。

3.3 线性回归的统计模型

真实情况下的数据样本往往会有噪声，比如 $y = f(\boldsymbol{x}, \omega) + \epsilon$ 其中 ϵ 是一个随机噪声，服从 $N(0, \sigma^2)$ 的正态分布，此时可以通过极大似然法来估计 ω，定义

$$P(y|\boldsymbol{x}, \omega, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{1}{2\sigma^2}(y - f(\boldsymbol{x}, \omega))^2] \tag{44}$$

根据极大似然法有

$$L(D, \omega, \sigma) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i, \omega, \sigma) \tag{45}$$

我们的求解目标变成了：

$$\omega = \arg\max L(D, \omega, \sigma) = \arg\max \prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i, \omega, \sigma) \tag{46}$$

取对数似然之后有

$$l(D, \omega, \sigma) = -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f(x_i, \omega)^2) + c(\sigma) \tag{47}$$

到这一步为止我们又回到了 RSS，因此解的表达式依然和上面推出的是一样的。

3.4 岭回归 Ridge Regression

我们发现普通的线性回归很容易出现 overfitting 的情况，比如一些系数 w_i 的值非常极端，仿佛就是为了拟合数据集中的点而生的，对测试集中的数据表现就非常差。为了控制系数的 size，让表达式看起来更像是阳间的求解结果，我们可以引入正则化的方法。

$$\omega^* = \arg\min \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \sum_{j=1}^{d} w_j^2 \tag{48}$$

这实际上就是拉格朗日算子，则我们跟之前一样可以将其写成矩阵形式：

$$(\boldsymbol{y} - X^T\omega)^T(\boldsymbol{y} - X^T\omega) + \lambda\omega^T\omega \tag{49}$$

对其求梯度可以得到

$$\nabla J_n(\alpha) = -2X(\boldsymbol{y} - X^T\alpha) + 2\lambda\omega = 0 \tag{50}$$

则其最终的解就是：

$$\omega^* = (XX^T + \lambda I)^{-1}Xy \tag{51}$$

这就是岭回归 (Ridge Regression) 的方法，其中参数 λ 是可以自己确定的，我们可以保证矩阵 $XX^T + \lambda I$ 是满秩的矩阵。

3.5 贝叶斯线性回归

现在我们重新考虑实际样本中可能会出现的噪声，即 $y = f(\boldsymbol{x}, \omega) + \epsilon$，其中 ϵ 服从 $N(0, \sigma^2)$ 的正态分布。根据贝叶斯定理可以得到：

$$P(\omega|y, x, \sigma) = \frac{P(y|\omega, x, \sigma)P(\omega|x, \sigma)}{P(y|x, \sigma)} \tag{52}$$

即 posterior 正比于 prior 和 likelihood 的乘积，即 $\ln(posterior) \propto \ln(likelihood) \times \ln(prior)$，而在线性回归问题中，我们已经知道了 likelihood 就是

$$l(D, \omega, \sigma) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f(x_i, \omega)^2) + c(\sigma) \tag{53}$$

我们可以用如下方法选择：

$$p(\omega) = N(w|0, \lambda^{-1}I) = \frac{1}{(2\pi)^{\frac{d}{2}}|\lambda^{-1}I|^{\frac{1}{2}}}\exp^{-\frac{1}{2}\omega^T(\lambda^{-1}I)\omega} \tag{54}$$

$$\ln(p(\omega)) = -\frac{\lambda}{2}\omega^T\omega + c \tag{55}$$

因此在贝叶斯理论下的岭回归模型需要优化的目标可以等价于：

$$-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f(x_i, \omega)^2) + c(\sigma) - \frac{\lambda}{2}\omega^T\omega + c \tag{56}$$

3.6 逻辑回归 Logistic Regression

3.6.1 基本概念

逻辑回归往往选择通过一个 sigmod 函数将样本映射到某个区间上，以此来估计样本属于某种类别的概率从而达到分类的目的。我们经常选用的 sigmod 函数是：

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{57}$$

比如对于二分类问题，可以将样本映射到区间 (-1,1) 上，此时如果计算结果为正数则说明样本属于正例，反之就是反例，可以表示为：

$$P(y_i = 1|x_i, \omega) = \sigma(\omega^T x_i) = \frac{1}{1 + e^{-\omega^T x_i}} \tag{58}$$

$$P(y_i = -1|x_i, \omega) = 1 - \sigma(\omega^T x_i) = \frac{1}{1 + e^{\omega^T x_i}} \tag{59}$$

上面的两个式子也可以统一写为:

$$P(y_i = 1|x_i, \omega) = \sigma(y_i \omega^T x_i) = \frac{1}{1 + e^{-y_i \omega^T x_i}} \tag{60}$$

3.6.2 参数估计

我们可以用极大似然法来估计参数 ω，依然用 D 表示数据集，具体的过程如下所示:

$$\begin{aligned} P(D) &= \prod_{i \in I} \sigma(y_i \omega^T x_i) \\ l(P(D)) &= \sum_{i \in I} \ln(\sigma(y_i \omega^T x_i)) = -\sum_{i \in I} \ln(1 + e^{y_i \omega^T x_i}) \end{aligned} \tag{61}$$

因此我们可以将逻辑回归的极大似然法参数估计的 loss 函数定义成:

$$E(\omega) = \sum_{i \in I} \ln(1 + e^{-y_i \omega^T x_i}) \tag{62}$$

对于一个二分类问题，我们如果用 0 和 1 而不是 +1 和-1 来代表两种分类，那么上面的表达式又可以写为:

$$\begin{aligned} E(\omega) &= \sum_{i \in I \cap y_i = 1} \ln(1 + e^{-\omega^T x_i}) + \sum_{i \in I \cap y_i = 0} \ln(1 + e^{\omega^T x_i}) \\ &= \sum_{i \in I \cap y_i = 1} \ln(e^{\omega^T x_i})(1 + e^{\omega^T x_i}) + \sum_{i \in I \cap y_i = 0} \ln(1 + e^{-\omega^T x_i}) \\ &= \sum_{i \in I} \ln(1 + e^{\omega^T x_i}) - \sum_{i \in I \cap y_i = 1} e^{\omega^T x_i} \\ &= \sum_{i \in I} (-y_i \omega^T x_i + \ln(1 + e^{\omega^T x_i})) \end{aligned} \tag{63}$$

我们可以证明 $E(\omega)$ 是一个关于 w 的凸函数，根据凸函数的可加性，我们只需要证明 $-y_i \omega^T x_i + \ln(1 + e^{\omega^T x_i})$ 是关于 w 的凸函数，我们令 $g(\omega) = -y_i \omega^T x_i + \ln(1 + e^{\omega^T x_i})$ 则对其求一阶梯度可以得到:

$$\frac{\partial g(\omega)}{\partial \omega} = -y_i x_i + \frac{x_i e^{\omega^T x_i}}{1 + e^{\omega^T x_i}} \tag{64}$$

能得到这个结果是因为我们有这样一个结论:

Theorem 3.6.1 对于一个 n 维的向量 ω, 我们有 $\frac{\partial \omega^T}{\partial \omega} = I_n$

进一步地，我们对上面求的一阶梯度再求二阶梯度，可以得到：

$$\frac{\partial^2 g(\omega)}{\partial \omega^2} = \frac{x_i^2 e^{w^T x_i}}{(1+e^{w^T x_i})^2} \geq 0 \tag{65}$$

因此我们证明了损失函数是一个凸函数，因此可以用梯度下降的方法求得其最优解，即：

$$\omega^* = \arg\min_{\omega} E(\omega) \tag{66}$$

根据上面求得的一阶梯度，可以得到基于梯度下降法的逻辑回归参数求解迭代方程：

$$\begin{aligned} \omega_{i+1} &= \omega_i - \eta(i) \sum_{i \in I} (-y_i x_i + \frac{x_i e^{\omega_i^T x_i}}{1+e^{\omega_i^T x_i}}) \\ &= \omega_i + \eta(i) \sum x_i (\frac{1}{1+e^{-\omega_i^T x_i}} - y_i) \\ &= \omega_i + \eta(i) X(\sigma(\omega_i, X) - y) \end{aligned} \tag{67}$$

其中 $\sigma(x)$ 是 sigmod 函数，而 $\eta(i)$ 是自己选择的学习率，一半是一个比较小的数字，并且应该不断减小。

4 感知机 Perceptron

4.1 感知机模型的基本概念

4.1.1 定义

感知机模型是一种二分类的线性分类模型，输入 k 维线性空间中的向量，输出一个实例的类别 (正反类别分别用 +1 和-1 来表示)，可以将这个分类过程用一个函数来表示：

$$y = f(x) = \text{sign}(\omega x + b) \tag{68}$$

这里的参数 ω 和 b 就是感知机的模型参数，其中 ω，而实数 b 是一个偏置 (bias)，这里的 sign 函数是一个特殊的函数，当输入的 x 是正数或者 0 的时候函数值就是 1，输入的 x 是负数的时候函数值就是-1.

4.1.2 基于超平面的理解

可以把样本集里面的 N 个 k 维的向量看成是 k 维线性空间中的点，感知机的目标就是找到一个划分这个点集的超平面 S，使得平面 S 的两侧分别是两种类型的点，后面的测试集的分类就基于测试集中数据所对应的点和超平面的位置关系来划分，在正例的一侧 $\omega x + b \geq 0$，反例的一侧则小于 0. 因此感知机的学习目标就是根据样本数据学习出超平面的方程 $\omega x + b = 0$

4.1.3 基于神经元的理解

其实感知机可以看成是一种非常简单的二层神经网络，输入层的内容是 k 维向量的 k 个特征，输出层的结果就是向量的分类情况 (分为 +1 和-1) 两种，而输出层的神经元存在一个阈值 θ，如果超过了这个阈值，神经元就会被激活，神经元存在一个激活函数 $f(x)$，因此这个二层神经网络可以用下面的式子来表示：

$$y = f(\sum_{i=1}^{k} \omega_i x_i - \theta) = f(\omega x - \theta) \tag{69}$$

其中 w_i 就是每个神经元的权重，也对应于定义式中的 ω，而阈值则是定义中的 bias 的相反数，函数 f 被称为激活函数，可以选用 sign 函数，也可以选用 sigmod 函数，这两个函数的区别是 sign 是实数域 $\mathbb{R}$ 不连续的函数，而 sigmod 是连续的，当选取 sign 函数作为激活函数的时候，这个感知机模型的表达式就和超平面的理解中完全一致。

4.2 感知机模型的求解

4.2.1 损失函数

感知机的训练集必须要求是线性可分的，而感知机的的性能评估要考虑被误分类的点的情况，为了选择一个关于参数 ω 和 b 连续的损失函数，一个比较自然的选择就是根据各个点到超平面 S 的距离之和，即有：

$$L = -\frac{1}{||\omega||}\sum_{x_i \in M} y_i(\omega x_i + b) \tag{70}$$

这里的 M 表示数据集 D 中被误分类的点的集合，因此感知机中的损失函数可以定义为：

$$L(\omega, b) = -\sum_{x_i \in M} y_i(\omega x_i + b) \tag{71}$$

因此感知机模型的求解目标就变成了：

$$\min L(\omega, b) = -\sum_{x_i \in M} y_i(\omega x_i + b) \tag{72}$$

4.2.2 基于随机梯度下降法的求解

对损失函数求梯度可以得到：

$$\nabla_\omega L(\omega, b) = -\sum_{x_i \in M} y_i x_i \tag{73}$$

$$\nabla_b L(\omega, b) = -\sum_{x_i \in M} y_i \tag{74}$$

因此可以在学习的过程中，**先确定一组参数的初值，并选择好学习率** eta，(注意学习率不能超过 1)，然后进行如下步骤开始学习：

- 在训练集中选取一组数据 (x_i, y_i)
- 如果这组数据是误分类的，也就是说 $y_i(\omega_k x_i + b_k) \leq 0$，那么就要：

$$w_{k+1} = w_k + \eta y_i x_i \tag{75}$$

$$b_{k+1} = b_k + \eta y_i \tag{76}$$

- 回到第二步继续循环，直到训练集中没有误分类的点，结束模型的训练

4.3 感知机的适用范围

5 支持向量机 SVM 和 Kernel

5.1 支持向量的引入

对于一个分类问题，我们如果可以找到一个向量 ω，使得对于数据集 D 中的任何样本 x，如果 x 是正例 (用 +1 表示) 就有 $\omega^T x > 0$，如果 x 是反例 (用-1 表示) 就有 $\omega^T x < 0$，那么我们就可以很好地对数据集进行分类，判断依据就是 $\omega^T x$

我们也可以这样来考虑，将数据集 D 中的每个样本 x 投射到 d 维的空间上，如果我们可以找到一个 d 维空间里的超平面 (hyperplane)$\omega^T x + b = 0$ 将这个空间划分成两个部分，其中一个部分里的样本 x 全是正例，另一个空间里的样本 x 全是反例，那么这个数据集的分类问题就解决了，但是实际情况并不会这么好，对于 d 维空间里的点 x，其到超平面的距离可以表示为:

$$r = \frac{|\omega^T x + b|}{||\omega||} \tag{77}$$

又为了能正确地进行分类，对于训练集中的数据，需要有:

$$\begin{cases} \omega^T x_i + b \leq -1, y_i = -1 \\ \omega^T x_i + b \geq +1, y_i = +1 \end{cases} \tag{78}$$

并且存在一些样本使得上面的等号成立，我们就称这些使得等号取到的点称为支持向量 (support machine)，每个支持向量到超平面的距离是:

$$r = \frac{1}{||\omega||} \tag{79}$$

则超平面两侧的两个支持向量到超平面的距离之和就是:

$$\gamma = \frac{2}{||\omega||} \tag{80}$$

这个量也被称为间隔 (margin)，为了使得分类效果尽可能地好，我们应该让这个间隔尽可能地大，因此我们的目标可以转化为求 $||\omega||^2$ 的最小值，即

$$\begin{aligned} &\min_{\omega,b} \frac{1}{2}||\omega||^2 \\ &s.t. y_i(\omega^T x_i + b) \geq 1 \end{aligned} \tag{81}$$

这个优化问题实际上就是支持向量机的基本形式

5.2 松弛变量 slack variable

在 SVM 问题求解中我们一般选择整个数据集中最小的一组间隔作为整个数据集的间隔，而我们优化的目标就是让这个最小间隔最大化。但是现实往往没有这么美好，数据的分布不会像我们预想的那么完美，因此我们可以引入松弛变量 (slack variable)，给间隔一个上下浮动的空间，即可以将问题转化成：

$$\begin{aligned} &\min_{\omega,b} \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n}\xi_i \\ &y_i(\omega^T x_i + b) \geq 1 - \xi_i \cap \xi_i \geq 0 \end{aligned} \tag{82}$$

我们可以将上面的约束条件转化为：

$$\xi_i \geq 1 - y_i(\omega^T x_i + b) \cap \xi_i \geq 0 \tag{83}$$

$$\Rightarrow \xi_i = \max\{1 - y_i(\omega^T x_i + b), 0\} \tag{84}$$

则优化的目标可以等价于：

$$\min_{\omega,b}(\sum_{i=0}^{n}\max(1 - y(\omega^T x_i + b), 0) + \frac{1}{2C}||\omega||^2) \tag{85}$$

在求解 SVM 的过程中，可以将这个式子的前半部分作为 loss function，后半部分作为 regularizer

5.3 线性模型的统一性

我们可以把上面的损失函数记作：

$$l(f) = \max[1 - yf, 0] \tag{86}$$

我们称这种类型的损失函数为 hinge loss(铰链损失函数)，因为其函数图像是一个类似于折线的形状。则我们的优化目标可以写成：

$$\min\{\sum_{i=0}^{n} l(f) + \lambda R(f)\} \tag{87}$$

其中 R(f) 是正则项。

事实上前面的所有线性模型，包括线性回归和逻辑回归的优化目标都可以写成上面的形式，区别在于 loss 函数的选择不同，线性回归选择的 loss 是 Square loss，而逻辑回归选择了 Logistic loss，这几种 loss 函数的图像也各有特点：

- 0-1 loss 只有 0 和 1 两种值，在优化目标化简的时候特别方便
- Square loss 的波动幅度比较大，更能反映出极端情况下的损失
- Logistic loss 的变动比较平缓但是永远存在
- Hinge loss 在一定情况下会变成 0，而在非 0 的时候比较平缓

5.4 核 Kernel

5.4.1 核方法 Kernel Method

支持向量机问题中，我们要求解的目标就是一个超平面 $y = \omega^T x + b$，用这个超平面来对数据集 D 进行线性的分割，这时候就出现了一个问题，如果数据不能被线性分割该怎么办？事实上我们之前在 SVM 以及其他的线性模型中都默认了数据集 D 是线性可分的，如果实际情况中碰到的并非这么理想，我们就应该采取一定的办法使得现有的数据变得线性可分。

核方法就可以解决这个问题，核方法通过将原始空间映射到一个更高维的特征空间中，使得数据集 D 中的样本 x 线性可分，而此时样本 x 也从一个 d 维向量映射成了一个更高维度的向量 (可以假定高维特征空间的维度是 χ)，用 $\phi(x)$，则我们需要求解的问题变成了：

$$y = \omega^T \phi(x) + b \tag{88}$$

5.4.2 核函数 Kernel Function

而在使用核方法的时候经常会需要计算两个向量的内积，由于特征空间的维度可能很高甚至是无穷维，因此我们可以用一个求解简单的核函数来代替内积，使得两个向量在特征空间的内积等于其原本的形式通过函数 $K(x_i, x_j)$ 计算出来的结果，这就是 kernel trick

根据向量内积的性质我们可以推测出，和函数一定是对称的，并且运算的结果需要时非负数，即有：

$$K(x_i, x_j) = K(x_j, x_i) \geq 0 \tag{89}$$

而对于 $|D| = m$，$K(x_i, x_j)$ 可以形成 m 维的半正定矩阵，这个矩阵也被称为再生核希尔伯特空间 (RKHS)，常见的核函数有：

- 线性核：$K(x_i, x_j) = x_i^T x_j$
- 多项式核：$K(x_i, x_j) = (x_i^T x_j)^d$
- 高斯核：$K(x_i, x_j) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$
- 拉普拉斯核：$K(x_i, x_j) = \exp(-\frac{||x_i - x_j||}{\sigma})$
- Sigmod 核：$K(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$

借助核方法和核函数我们可以把线性分类器用到非线性分类的问题上，但是具体如何选择核函数是未知的，或许需要自己一个个去尝试。

6 kNN：k-Nearest Neighbor 分类

6.1 基本概念

6.1.1 kNN 的基本思路

kNN：k-Nearest Neighbor 算法是一种分类算法，其核心想法是根据给定的训练集，对于需要判别的测试集中的样本，在训练集中找到与之最接近的 k 个样本，并统计这 k 个样本的分类情况，将出现次数最多的类别作为测试集中的样本的分类结果。如果将 k 个最近的样本用 $N_k(x)$ 来表示，那么 kNN 的分类决策用公式表示就是：

$$y = \arg\max \sum_{x_i \in N_k(x)} I(y_i = c_i) \tag{90}$$

kNN 的基本思路其实就是：如果一个东西看起来像鸭子，走路像鸭子，吃饭也像鸭子，那么它很可能就是一只鸭子。而" 最接近" 这个概念是有待商榷的，因为我们没有统一的距离度量法则，因此需要确定一种度量距离的方法。此外 k 也需要自己选择，k=1 的时候这个算法被称为最近邻算法

6.1.2 距离度量的选择

我们需要在 n 维的特征空间中度量两个点之间的距离，对于 n 维空间中的两个向量 x_i, x_j，我们定义距离：

$$L_p(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|)^{\frac{1}{p}} \tag{91}$$

上面的公式中 p=2 的时候称为欧几里得距离，而 p=1 的时候称为曼哈顿距离，当 p 为无穷大的时候，L 就是每个坐标中距离最大值.

6.1.3 k 的选择

k 的选择也对最终的结果有比较大的影响，k 选择太小的话学习时的误差会减小，但是估计误差会增大，预测结果会对样本的位置关系非常敏感。如果 k 选的太大，则估计误差会降低，但是学习时的误差会增大，二者各有利弊，当然如果 k=N 那么就是按照出现次数最多的作为结果，完全忽略了模型中的大量有用信息。

6.2 kd 树：kNN 求解

kd 树是一种对 k 维空间中的实例点进行存储以便对其进行快速检索的树形数据结构，是一种二叉树，用来表示对 k 维空间的一个划分，使得每个 k 维空间中数据集的点对应到一个 k 维空间超矩形上面去。

6.2.1 kd 树的构造

一般来说构造 kd 树的方法是先选择根节点，然后每次选定一个坐标轴，并用该方向的样本点坐标的中位数作为划分的依据，并沿着垂直于坐标轴的方向作超矩形，然后将 k 维空间分出了新的两个区域，之后就在两个新的区域里面换一个坐标轴重复上面的操作，直到每个样本点都进行了划分，每个子区域中没有样本点的时候停止。

用这种方法构造出来的 kd 树是

6.2.2 kd 树的搜索

对于一棵构造好的 kd 树进行 k 近邻搜索可以省去对大部分数据点的搜索，从而减少搜索的计算量，以最近邻为例，对于给定的目标点，搜索其最邻近的点需要首先找到包含目标点的叶节点 (注意，kd 树里的节点代表的是超平面里的一块超矩形，不是样本点)，然后从该叶节点出发，回退到父节点，并不断查找与目标点最邻近的节点，当确定不可能存在更近的节点的时候终止。

6.3 kNN 实际编码体验

尝试了蔡老师机器学习课程中的 kNN 作业，目标是用 kNN 实现一个简单的验证码数字识别，主要任务其实就是用最原始的方法实现 kNN，然后对数据集进行一定的标注之后用 kNN 来识别数字，可能是因为场景比较简单，最后实现的验证准确率非常高。

7 决策树 Decision Tree

7.1 基本概念

决策树 (Decision Tree) 实际上是通过树形的结构来进行分类或者回归，在分类问题中可以根据样本的不同特征属性的不同属性值来进行多次分类的决策最终确定输入样本的类别，也就相当于执行了一连串嵌套的 if-else 语句，**也可以被认为是定义在特征空间和类空间上的条件概率分布**。决策树的优点是模型具有较好的可读性，分类的速度比较快。

决策树中有一系列节点和有向边，节点又分为内部节点和也节点，内部节点表示一个特征或者属性，叶节点表示一种类别。

7.1.1 决策树模型的建立

决策树主要依靠训练数据，采取损失函数最小化的原则来建立决策树模型，分为特征选择、决策树的生成和剪枝三个步骤。建立决策树的过程用的特点主要有：

- 采用递归的方法构建决策树
- 当所有的样本都属于同一类或者在这一层已经对所有的特征都进行了分类，决策树的建立就停止
- 在决策树的每一层**选取一个最优的特征并用这个特征来进行分类**，这也是决策树算法最关键的地方
- 一般而言我们希望，随着划分过程的不断进行，决策树的分支节点所包含的样本尽可能属于同一类别，也就是节点的纯度越来越高

因为决策树考虑到了样本中的所有点，对所有的样本点都有正确的分类，因此决策树的 bias 实际上是 0，因此评价一棵树的主要标准是 variance，一般来说规模比较小的决策树的 performance 更好。

7.2 最优特征的选取

上面已经说到决策树构建的最重要的部分就是在每一层选择合适的特征来进行划分，常见的算法有熵、信息增益等等。

7.2.1 信息熵 Entropy

熵可以用来表示随机变量的**不确定性**。如果 X 是一个取值个数为有限个值即 $P(X = x_i) = p_i$，那么随机变量 X 的熵的定义就是：

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i \in [0, \log n] \tag{92}$$

而对于随机变量 X 和 Y，如果 $P(X = x_i, Y = y_i) = p_{ij}$，那么在 X 给定的情况下随机变量 Y 的条件熵代表了 X 给定条件下 Y 的条件概率分布的熵对于 X 的数学期望：

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{93}$$

当用数据估计得到熵和条件熵的时候，他们又分别被称为经验熵和经验条件熵。

7.2.2 信息增益

信息增益 (information gain) 表示**已经知道特征 X 的信息而使得类 Y 的信息的不确定度减少的程度**。特征 A 对于数据集 D 的信息增益 $g(D, A)$ 定义为 D 的经验熵和 D 在 A 给定条件下的经验条件熵的差值，即：

$$g(D, A) = H(D) - H(D|A) \tag{94}$$

信息增益又叫做**互信息** (mutual information)，两个概念是等价的。基于信息增益的特征选择方法一般都是对于当前训练集 D，计算其每个特征的信息增益，并比较大小，选择信息增益最大的特征来进行当前层的分类。

假设训练的数据集是 D，有 K 个不同的分类 C_i 满足 $\sum_{i=1}^{K} |C_k| = |D|$，其中特征 A 有 n 个不同的取值 a_i，根据特征的取值将 D 划分成了 n 个不同的子集 D_i，而每个子集 D_{ik} 中表示属于类别 C_k 的样本，那么数据集 D 的经验熵可以表示为：

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|} \tag{95}$$

特征 A 对于数据集 D 的条件经验熵可以表示为：

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|} \tag{96}$$

7.2.3 信息增益比

以信息增益作为数据集划分依据的时候，容易偏向于选择取值更多的特征作为分类标准，但是使用信息增益比，可以校正这一偏差，信息增益比是当前数据集 D 关于特征 A 的信息增益和自身的熵的比值，即：

$$g_R(D,A) = \frac{g(D,A)}{H_A(D)} = \frac{g(D,A)}{-\sum\limits_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}} \tag{97}$$

7.2.4 基尼指数

基尼指数是另一种决策树的评价标准，对于分类问题，假设样本集合中有 K 中不同的类别，每类出现的概率为 p_k 那么基于概率分布的基尼指数可以定义成

$$G(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2 \tag{98}$$

而在样本集合 D 中，基尼指数可以定义为：

$$G(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|} \right)^2 \tag{99}$$

7.3 决策树的剪枝与生成

7.3.1 ID3 算法

ID3 算法的核心是在决策树的各个节点上用信息增益 Information Gain 来进行特征的选择，并且递归地建立决策树：

- 从根结点开始，对于当前节点计算所有可能的特征值的信息赠一，选择信息增益最大的特征作为划分的一句并建立字节点
- 直到所有的特征的信息增益都很小或者没有子节点位置停止调用

7.3.2 C4.5 算法

和 ID3 算法类似，但是采用信息增益比作为选择的特征，其他的好像区别不是很大，但实际上决策树的代码实现应该是一系列统计学习的算法中难度比较大的，因为决策树需要使用树的数据结构，相比于其他算法的一系列矩阵计算在 coding 的过程中可能会有更大的麻烦。

7.3.3 决策树的剪枝 Pruning

我们要明白决策树实际上是一种**基于大量统计样本的贪心算法** (仅根据个人理解，不代表任何工人观点)，在选定了一个决策标准 (信息熵，信息增益，信息增益比等等) 之后，就需要根据训练集来构建一棵决策树，构建的策略就是在每一次作出决策的时候 (对应到树中就是产生了一系列子节点) 都会依照给定的标准，计算每种特征相对于给定标准的值，然后选择当前区分度最大，也就是最优的特征作为当前层的决策依据。

但是这样一来，生成的决策树的复杂度可能是非常高的，因此我们应该想办法对这个决策树进行一定的简化，也就是进行剪枝，决策树的剪枝往往通过极小化决策树整体的损失函数或者代价函数来实现。设决策树 T 的每个叶节点 t 有 N_{kt} 个样本点，其中属于类别 k 的有个，节点 t 的经验熵可以表示为:

$$H_t(T) = -\sum_{k=1}^{n} \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t} \tag{100}$$

那么决策树的损失函数可以定义为:

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T| = -\sum_{t=1}^{|T|} \sum_{k=1}^{n} N_{tk} \log \frac{N_{tk}}{N_t} + \alpha|T| \tag{101}$$

而其中 α 是一个自己确定的参数值，比较大的 α 可以使得损失函数偏向于更简单的模型，而比较小的则似的损失函数的优化偏向于比较复杂的模型，$\alpha = 0$ 的时候完全无视了模型的复杂度，而只考虑模型和训练数据的匹配程度。其实这也就相当于给决策树的损失函数加一个正则项了，我们可以把过分精确的决策树看成是一种过拟合。

这里就不得不提一个冷知识:

Tip 7.3.1 完整的决策树在训练的过程中分类的误差是 0，也就是说不做任何剪枝处理的决策树对训练集的分类是一定准确的。

这也很好理解，因为决策树在构建的过程中并没有舍弃掉一些小概率的样本，而是对其如实进行了分类，虽然最后构建出来的决策树可能非常复杂但是在构建的时候因为我们的策略是贪心的，所以一定会把训练集上的样本分类到对应的叶节点上。

换句话说不剪枝的决策树的 bias 是 0，而 variance 往往比较大，未剪枝的决策树在训练集上表现很好而在测试集上可能不尽人意，因此我们需要通过剪枝使得决策树可以更好地适应测试集，换句话说，**剪枝可以降低决策树模型的** variance

Tip 7.3.2 剪枝过程是一个从底部向上的递归过程，首先需要计算每个节点的经验熵，如果剪掉某一段子树之后损失函数比剪掉之前更小，就进行剪枝，将父节点作为新的叶节点，重复下去直到不能继续为止。

7.4 CART 算法

7.4.1 CART 的定义

CART 算法是 Classification And Regression Tree(分类与回归树) 的简称，从字面意思就可以看出来这种决策树既可以用于分类，也可以用于回归。

- CART 树的一个基本假设就是决策树都是二叉树，**决策的结果只有是与否两种，这等价于递归地二分每个特征**，将输入空间划分成优先个但愿，并在这些单元上确定预测的概率分布，也就是在输入给定的条件下输出的条件概率分布
- CART 算法由生成和剪枝两个部分组成，对于生成的决策树希望它尽可能要大，剪枝的时候使用验证数据集来选择最优子树，这时候用损失函数最小作为剪枝的标准

7.4.2 回归树的生成

回归树对应的是**输入空间的一个划分和输出值的映射关系**，假如说输入的空间被划分成了 M 个单元 R_i，每个分别对应了固定的输出值 c_i，那么回归树模型可以表示为一个函数：

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{I}(x \in R_m) \tag{102}$$

如果输入空间的划分确定，就可以用平方误差来估计训练数据的预测误差，用平方误差最小的准则来求解每个单元上的最优输出值。对于每一个单元 R_i，这个单元对应的输出值就是其中若干个样本的 label 的平均值：

$$c_m = \frac{1}{|R_m|} \sum_{x_i \in R_m} y_i \tag{103}$$

因此问题在于如何对输入空间进行合理的划分，《统计学习方法》这本书里采用了一种启发式的方法，选择选择特征空间中的第 j 个变量 $x^{(j)}$ 和它取的值 s 作为切分变量 (splitting variable) 和切分点。并定义两个区域：

$$R_1(j,s) = \{x|x^{(j)} \leq s\} \quad R_2(j,s) = \{x|x^{(j)} > s\} \tag{104}$$

然后我们可以遍历所有的特征 j 和特征 j 的取值 s 来找到使得平房误差最小的切分点，寻找的依据是：

$$\min_{(j,s)}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_1(j,s)}(y_i-c_2)^2\right] \tag{105}$$

对于一个固定的 j 而言，上面的表达式中 c_1, c_2 的值是确定的，即按照上面所说的取这一范围内的平均值即可，而通过遍历所有的输入变量可以找到最优的切分变量 j，然后就可以根据这样一个二元组将决策树划分成两个区域，然后在每个区域中重复上述操作，最后就可以生成一棵最小二乘回归树

7.4.3 分类树的生成

CART 的分类树用基尼指数作为衡量集合不确定性的标准，因为 CART 默认采用的都是二分类，因此基尼指数可以定义成：

$$G(D,A)=\sum_{i=1}^{2}\frac{|D_i|}{|D|}G(D_i) \tag{106}$$

分类树的构建方法和普通的决策树一样，只不过选用基尼指数作为评价标准并且每次划分只能划分成两个子节点，递归建树的过程和普通决策树是完全一致的。

8 聚类 Clustering

8.1 聚类的基本概念

聚类是一种无监督学习，就是在给定的数据集中根据特征的相似度或者距离，将数据集中的对象归并到若干个类中 (事先并不知道，即数据集是没有 label 的)，因此**解决聚类问题的关键是要先定义距离的度量方式**，因为很多时候数据点并没有物理意义上的“距离”(地图上的不同点拥有的是物理距离)，比如社交网络中人和人之间的关系就是一种广义上的距离，对于不同的场景，我们需要寻找不同的方式来定义数据样本中的“距离”

8.1.1 相似度和距离

用于聚类的样本数据集 D 或者样本集合，假设有 n 个样本，每个样本的有 n 个属性，那么样本的集合就可以用一个大小为 $m \times n$ 的矩阵 X 来表示，矩阵的每一列表示一个样本的 m 个特征，常见的距离定义方法有

- 闵可夫斯基距离：对于两个向量，定义其闵可夫斯基距离为

$$d_{ij} = \left(\sum_{k=1}^{m} |x_{ik} - x_{jk}|^p \right)^{\frac{1}{p}} \tag{107}$$

Tip 8.1.1 闵可夫斯基距离中，当 p=1 时就是曼哈顿距离，p=2 时就是欧式距离，p=∞ 为时就是切比雪夫距离，等价于各个坐标数值差的绝对值

- 马哈拉诺比斯距离：假设样本矩阵 X 的协方差矩阵是 S，则马哈拉诺比斯的距离共识可以表示为

$$d_{ij} = \left[(x_i - x_j)^T S^{-1} (x_i - x_j) \right] \tag{108}$$

Tip 8.1.2 马哈拉诺比斯距离考虑了各个特征之间的相关性，通过协方差矩阵传递了这种相关性到两个样本的距离中，马哈拉诺比斯距离越大说明相似度越小

- 向量夹角的余弦：高中的时候立体几何中学的

$$s_{ij} = \frac{\langle x_i, x_j \rangle}{||x_i||_2 \times ||x_j||_2} = \frac{\sum_{k=1}^{m} x_{ki} x_{kj}}{\left(\sum_{k=1}^{m} x_{ki}^2 \sum_{k=1}^{m} x_{kj}^2 \right)^{\frac{1}{2}}} \tag{109}$$

- 相关系数：可以用相关系数来表示样本之间的相似度，(概率论和数理统计中似乎学过这个只是) 其定义是：

$$r_{ij} = \frac{\sum_{k=1}^{m}(x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)}{\sum_{k=1}^{m}(x_{ki} - \bar{x}_i)^2 \sum_{k=1}^{m}(x_{kj} - \bar{x}_j)^2} \tag{110}$$

8.1.2 类和簇的判定

定义了什么是距离之后，我们遇到的第二个问题就是，对于一个数据集合 G，当其中的点距离满足什么条件的时候可以将一系列数据样本定义成一个类 (也可以叫簇，cluster)，而常见的定义方式有：

- 集合 G 中任意两个点的距离都不超过常数 T
- 对于集合 G 中的任意一个点，都存在另一个样本使得二者的距离不超过常数 T
- 对于 G 中的一个样本点，它和任意一个其他点的距离都不超过 T
- 对于集合 G 中的样本，任意两个点之间的距离不超过 V 并且所有距离的平均值不超过 T

我们可以用一些特征来刻画一个类，比如定义类的均值或者直径：

$$\bar{x}_G = \frac{1}{N_G}\sum_{i=1}^{N_G} x_i \qquad G_D = \max d_{ij} \tag{111}$$

同时类和类之间的距离也可以进行定义，这种距离又可以叫做连接 (linkage)，可以有多种定义方式，比如最短距离，最长距离，中心局距离和平均距离等等。

8.2 层次聚类

层次聚类是假设类别之间存在层次结构，将样本聚合到层次化的类里面去，分为自顶向下的聚类 (分割) 和自底向上的聚类 (聚合)，聚合算法首先要确定三个要素，不同要素进行组合就可以得到一系列聚类算法：

- 定义距离或者相似度
- 确定合并的规则
- 确定算法停止的条件

8.2.1 聚合聚类

聚合聚类会根据输入的 n 个样本点进行聚类，最后输出一个包含所有样本的类和其中的各个子类，算法步骤如下：

- 对于 n 个样本点计算其距离，构成一个 $m \times n$ 的矩阵
- 构造 n 个不同的类，每个类只包含一个样本
- 合并类间距离最小的两个类，以最短距离为类间距离，构造一个新类
- 计算新的类和当前各个类的距离，如果类的个数为 1 则停止计算，否则返回上一步

8.3 K-means 算法

8.3.1 基本介绍

K-Means 聚类 (也叫做 K 均值聚类) 是一种基于样本集合划分的聚类算法，k 均值将样本集合划分称为 k 个子集，每个样本只属于一个类，因此是一种**硬聚类**的算法 (相对的，软聚类算法是结果中每个样本可能属于多个类的算法)

对于给定的样本集合 X，每个样本用特征空间中的特征向量来表示，我们希望将样本分成 k 个不同的类别 G_k 并且满足：

$$G_i \cap G_j = \emptyset \quad \bigcup_{i=1}^{k} G_i = X \tag{112}$$

可以用 C 来表示这样的一个划分关系，而一个划分关系也就对应着一个 K 均值分类的结果，划分 C 是一个函数，可以表示为：

$$C(x_i) = G_k \tag{113}$$

表示一个样本的最终划分结果，而 K 均值算法在训练的过程中采用最小化损失函数的方式来选择最优的划分函数 C

8.3.2 K-Means 的 loss 函数

K-Means 采用欧几里德距离，并且用**所有点到其所在的类的中心的距离的平方和**作为损失函数：

$$W(C) = \sum_{l=1}^{k} \sum_{C(i)=l} ||x_i - \bar{x}_l||^2 \tag{114}$$

因此实际上 K-Means 算法的求解就是一个最优化的问题：

$$C^* = \arg\min W(C) = \arg\min \sum_{l=1}^{k} \sum_{C(i)=l} ||x_i - \bar{x}_l||^2 \tag{115}$$

8.3.3 K-Means 算法的训练过程

根据上面所确定的损失函数，K-Means 算法的训练过程可以这样描述：

1. 首先要随机选 K 个样本作为初始的 K 个类的中心点，然后进行如下步骤
2. 对于每个样本点，计算其与 K 个类的中心点，然后在**选择最近的中心点所在的类作为该样本点的类**
3. 第 2 步的循环完成之后，在每个类中找到使得到该类中的所有样本点距离最小的中心点，使得该类中距离，也就是该类的样本中所有点的均值
4. 然后回到第 2 步用新的 K 个中心点计算出新的分类，重复上述步骤直到划分结果不再改变，就得到了 K 均值聚类的最终结果

8.3.4 K-Means 算法的特点

1. K 均值聚类属于启发式的方法，不能保证收敛到全局的最优，初始中心的选择会直接影响聚类的结果，要注意的时候类中心的移动往往不会有非常大的波动
2. 不同的类别数 K 会带来不同的聚类结果，一般来说分类的平均直径会随着 K 的增大先减小后不变，这个算法中的 K 是一个超参数，也是需要人为进行设定的

8.3.5 k-Means 代码实现

提供一份浙江大学蔡登老师给图灵班开设的 ML 课的作业中的一段 K-means 具体实现的代码，是本人写的因此可能比较垃圾

```
def kmeans(x, k):
    """
    KMEANS K-Means clustering algorithm

        Input: x - data point features, n-by-p maxtirx.
               k - the number of clusters

        OUTPUT: idx - cluster label
                centers - cluster centers, K-by-p matrix.
                iteration_number - cluster centers of each iteration, (iter, k, p)
                      3D matrix.
    """

    # begin answer
    def get_current_center(centers, k, x):
        dist = np.zeros(k)
        for j in range(k):
            dist[j] = np.linalg.norm((centers[j] - x))
        min_idx = np.argmin(dist)
        return min_idx

    def get_new_center(index, x, k):
        count = 0
        sum_x = np.zeros((1, x.shape[1]))
        for i in range(len(index)):
            if index[i] == k:
                count += 1
                sum_x += x[i]
        return sum_x / count

    N, p = x.shape
    max_iter, iteration = 2, 0
    idx = np.zeros(N, dtype=np.int32)
    centers = np.zeros((k, p))
    iteration_number = np.zeros((max_iter + 1, k, p))

```

```
    # 初始化K个不同的center，随机选择数据集中的点作为中心
    first_centers = np.random.randint(0, N, k)
    for i in range(k):
        centers[i] = x[first_centers[i]]
    iteration_number[0] = centers
    while iteration < max_iter:
        # 开始迭代，每次先根据中心求出点与中心的距离，然后选择最小的点
        for i in range(N):
            idx[i] = get_current_center(iteration_number[iteration], k, x[i])
        for i in range(k):
            res = get_new_center(idx, x, i)
            centers[i] = res
        iteration += 1
        iteration_number[iteration] = centers

    # end answer

    return idx, centers, iteration_number
```

9 提升 Boosting 与 Ensemble

9.1 基本介绍

提升方法其实就是用多个不同的决策方法组合而成，根据不同方法的决策情况来进行最后的决策，也就是说不在一棵树上直接吊死，而是多找几棵树看看那棵树的风水最好再吊上去。

9.1.1 强可学习与弱可学习

如果可以用一个多项式复杂度的学习算法学习一个概念，并且正确率很高，那么这个概念就是强可学习的，如果学习的效果仅仅比随机猜测稍微好一点，那就是弱可学习的。如果预测的正确率还不如随机猜测，那这种算法属实没有存在的必要

弱可学习的算法一般比较容易学习，相比于强可学习算法更容易获得并且训练所需要的 cost 往往也更少，因此我们就产生了一个想法，是不是可以通过多个比较弱的学习算法一起预测结果，然后根据这些算法的预测结果再决定最后的分类结果。

- 但是决策的结果往往是少数服从多数，即“极大似然法”，我们暂且不讨论是否会出现多数人的暴政
- 可以给不同的分类器设定一个不同的权重，分类效果比较好的可以设置比较大的权重，这类通过组合多个算法共同预测的机器学习算法就是 ensemble method

9.1.2 ensemble 有效性的证明

这种看似缝合怪的机器学习方法也叫做 Combining Models，顾名思义也就是将多个不同的 model 组合到一起共同预测结果，如果 M 个模型的权重是相同的，那么预测结果可以表示称：

$$y_{COM} = \frac{1}{M}\sum_{i=1}^{M} y_i(x) \tag{116}$$

那么为什么 ensemble method 的集体决策效果要比单个分类器的预测效果好呢？我们可以假设真实的模型是 $h(x)$，那么 M 个不同的模型组合而成的新模型的误差期望是：

$$
\begin{aligned}
E_{COM} = E((y_{COM} - h(x))^2) &= E((\frac{1}{M}\sum_{i=1}^{M} y_i(x) - h(x))^2) \\
&= \frac{1}{M^2}\sum_{i=1}^{M} E((y_i - h(x)^2) = \frac{1}{M}E_{AV}
\end{aligned}
\tag{117}
$$

我们发现通过 ensemble 方法，模型的 variance 变成了原来的 $\frac{1}{M}$，因此这种方法确实可以提高预测的准确性，因为它减小了模型的 variance，使得模型的预测结果更加稳定。

9.2 如何生成多个模型：Bagging

我们现在已经知道模型的组合可以提高预测结果的稳定性，那么我们如何来生成多个模型进行组合呢？常见的方法有两种，一种是 Bagging，另一种是 Boosting

Bagging 是通过不同的数据集划分方式来训练出多个模型，是一种可以并行的模型训练方式，它的两个关键步骤可以概括为：

- Bootstrap sampling：从数据集 D 中生成若干组不同的大小为 N 的训练集，并训练出一系列基学习器 (Base Learner)
- Aggregating：即模型组合的策略，往往在分类模型中采用 voting 策略，即选择可能性最高的，而在回归问题中使用 averaging，取平均数

Bagging 方法在基学习器是弱可学习的时候提升效果非常好，这一点也可以理解，因为 Bagging 方法的本质是生成了一系列乌合之众，然后用 voting 策略来预测结果，但是乌合之众的规模变大的时候准确度也会有所提高。

这里的基学习器可以选择各类线性模型或者神经网络，也可以使用决策树，因为**决策树是一种比较容易获得的非线性分类器**，并且 bias 非常小而 variance 非常高，我们用 ensemble 的方式刚好可以降低决策树的 variance，得到 bias 和 variance 都比较小的模型，而这种使用决策树作为基学习器的 Bagging 模型叫**做随机森林** (Random Forest)，这个名字也比较形象，很多“树”聚集成了森林。

9.3 Boosting 方法与 AdaBoost

与 Bagging 方法不同 Boosting 方法是一种迭代式的训练算法，其中比较著名的就是 AdaBoost，这是一种串行的 Ensemble 方法，通过若干次迭代的训练将训练数据学习成若干个弱

分类器，然后通过调整权重来提高模型预测的准确度。

9.3.1 AdaBoost 算法步骤

1. 将训练集 D 中的 N 个训练数据赋予初始的权重，一般来说是所有样本之间都相等，即

$$D_1 = (w_{11}, w_{12}, \ldots, w_{1N}) \quad w_{1i} = \frac{1}{N} \tag{118}$$

2. 对数据进行一共 M 轮的迭代，第 m 轮的时候训练数据的权重分布是 D_m，并训练得到一个弱分类器 $G_m(x)$，然后计算这个分类器的训练误差：

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} \mathbb{I}(G_m(x_i) \neq y_i) \tag{119}$$

3. 然后计算该分类器的系数：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \tag{120}$$

4. 更新整个训练集的权重分布 $D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \ldots, w_{m+1,N})$，其中 Z_m 叫做规范化因子，很明显这样的规范化因子使得前面的系数的和变成了 1，也就是说权重系数成为了一个概率分布

$$\begin{aligned} w_{m+1,i} &= \frac{\exp(-\alpha_m y_i G_m(x_i))}{Z_m} w_{m,i} \\ Z_m &= \sum_{i=1}^{N} \exp(-\alpha_m y_i G_m(x_i)) w_{m,i} \end{aligned} \tag{121}$$

5. 最后生成一个各个若分类器进行线性组合，形成权重不同的强分类器：

$$\begin{aligned} f(x) &= \sum_{i=1}^{M} \alpha_m G_m(x) \\ G(x) &= \text{sign}(f(x)) = \text{sign}(\sum_{i=1}^{M} \alpha_m G_m(x)) \end{aligned} \tag{122}$$

9.3.2 AdaBoost 误差分析

AdaBoost 存在一个误差上界：

$$\frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(G_m(x_i) \neq y_i) \leq \frac{1}{N} \sum_{i} \exp(-y_i f(x_i)) = \prod_{m} Z_m \tag{123}$$

而对于一个二分类的问题，有：

$$\begin{aligned}\prod_{m=1}^{M} Z_m &= \prod_{m=1}^{M} 2\sqrt{e_m(1-e_m)} \\ &= \prod_{m=1}^{M} \sqrt{1-4\gamma_m^2} \leq \exp(-2\prod_{m=1}^{M}\gamma_m^2)\end{aligned} \tag{124}$$

其中 $\gamma_m = \frac{1}{2} - e_m$

10 EM 算法和高斯混合模型 GMM

EM 算法的全称是 Expectation Maximization，是用于**对含有隐变量的概率模型进行参数极大似然估计的一种迭代算法**，隐变量就是数据集的特征中中没给出但是会影响预测结果的变量。如果没有隐变量，那么在概率模型的估计中直接进行极大似然估计，然后求导即可，而有隐变量存在的时候导数不能直接求出，所以才需要使用 EM 算法。EM 算法主要分成 E 和 M 两个主要步骤：

- E 步骤：求解参数估计值的期望
- M 步骤：使用极大似然法求出期望的最值，然后将得到的结果放到下一轮迭代中去

10.1 EM 算法的推导

如果说一个概率模型中中含有可观测隐变量 Z，那么我们在极大化观测数据 Y 关于模型参数的似然函数的时候，实际上的目标就变成了

$$\begin{aligned} L(\theta) &= \log P(Y|\theta) = \log \sum_Z P(Y,Z|\theta) \\ &= \log \sum_Z P(Y|Z,\theta)P(Z|\theta) \end{aligned} \tag{125}$$

而隐变量是没有观察到的，因此不能直接使用极大似然法进行参数估计，EM 算法要解决的就是在隐变量条件下的参数估计问题，两个步骤的具体过程如下：

10.1.1 EM 算法的输入输出要求

一般来说用 Y 表示观测随机变量的数据，用 Z 来表示隐变量的数据，Y 和 Z 联合在一起一般称为完全数据，观测数据 Y 又叫做不完全数据，EM 算法的求解需要知道观测变量数据 Y，隐变量数据 Z，联合概率分布 $P(Y,Z|\theta)$，条件分布 $P(Z|Y,\theta)$，最终输出的是模型的参数 θ

10.1.2 Q 函数

完全数据的对树似然函数 $\log P(Y,Z|\theta)$ 关于在给定观测数据 Y 和当前参数估计 θ_i 下对于未观测数据 Z 的条件概率分布 $P(Z|Y,\theta)$ 的期望称为 Q 函数，也就是：

$$Q(\theta,\theta_i) = E_Z[\log P(Y,Z|\theta)|Y,\theta_i] \tag{126}$$

10.1.3 EM 算法求解过程

在选定好初始化参数 θ_0 之后，需要进行：

1. E 步骤：假设第 i 次迭代的时候参数为 θ_i，则在第 i+1 次的 E 步骤，计算 Q 函数：

$$\begin{aligned} Q(\theta, \theta_i) &= E_Z[\log P(Y, Z|\theta)|Y, \theta_i] \\ &= \sum_Z \log P(Y, Z|\theta) P(Z|Y, \theta_i) \end{aligned} \tag{127}$$

2. M 步骤：求使得 $Q(\theta, \theta_i)$ 最大化的 θ 作为本次迭代的新估计值：

$$\theta_{i+1} = \arg\max Q(\theta, \theta_i) \tag{128}$$

3. 重复 E 步骤和 M 步骤直到得到的参数收敛

10.1.4 EM 算法的有效性和收敛性证明

这部分内容《统计学习方法》上有比较详细的证明，奈何暂时看不懂，就先不管了，先理解 EM 算法的基本步骤再说。

10.2 高斯混合模型 GMM

高斯混合模型是指如下形式的概率分布模型：

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k) \tag{129}$$

其中 α_i 表示权重系数，$\phi(y|\theta_k), \theta_k = (\mu_k, \sigma_k^2)$ 表示一个高斯分布的密度函数，即：

$$\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(y_k - \mu_k)^2}{2\sigma_k^2}\right) \tag{130}$$

10.2.1 GMM 的隐变量分析

高斯混合模型中的模型参数有:$\theta = (\alpha_a, \cdots \alpha_K, \theta_1, \ldots, \theta_K)$，我们需要估计每个高斯分量的权重和自身的模型参数，而 GMM 中是有隐变量的，我们可以这样来分析：首先根据每个不同的权重 α_k 来选择第 k 个分量计算生成的观测数据 y_j，这个时候的观测结果是已知的，但是反

应观测数据来自第 k 个分量的数据是未知的，因此我们可以用一个隐变量 γ_{jk} 来表示第 j 个观测来自于第 k 的模型，因此该隐变量只能取 0 和 1 两个值。

这里很重要的一点，也是非常容易出现的误区就是，观测数据的生成并不是在 K 个高斯模型中分别生成然后按照权重比例组合，而是以权重比例为概率分布情况，随机选择出一个高斯模型来生成观测数据 y_j，因此隐变量 $\gamma_{jk}(k = 1, 2, \ldots, k)$ 中有且仅有一个是 1，剩下的都是 0

10.2.2 GMM 的求解和参数估计

EM 求解 GMM 需要先输入 N 个观测数据，并输出一个高斯混合模型的参数，训练的过程主要分为 E 步骤和 M 步骤：

- E 步骤：依据当前的模型参数，计算分模型 k 对观测数据的响应度，即：

$$\hat{\gamma}_{jk} = \frac{\alpha_k \phi(y_j|\theta_k)}{\sum_{k=1}^{K} \alpha_k \phi(y_j|\theta_k)} \tag{131}$$

- M 步骤：根据隐变量计算出新一轮迭代所需要的模型参数，模型就根据这些参数产生：

$$\mu_k = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk} y_j}{\sum_{j=1}^{N} \gamma_{jk}} \tag{132}$$

$$\sigma_k^2 = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk} (y_j - \mu_k)^2}{\sum_{j=1}^{N} \gamma_{jk}} \tag{133}$$

$$\alpha_k = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk}}{N} \tag{134}$$

其实这一部分一开始我没怎么看懂，后面有空继续看。

11 主成分分析 PCA

11.1 基本概念

主成分分析 (Principal component analysis, PCA) 是一种常见的无监督学习方式，**利用正交变换把线性相关变量表示的观测数据转换成几个线性无关变量所表示的数据**

- 这些线性无关的变量叫做主成分
- 主成分分析属于降维方法的一种，关于这个，统计学习方法中举了很多例子

在数据总体上进行的主成分分析叫做总体主成分分析，在有限的样本数据上进行的主成分分析称为样本主成分分析

11.2 样本主成分分析

问题的情景：对 m 维的随机变量 x 进行 n 次独立的观测，得到一个大小为 $m \times n$ 的样本矩阵 X，并且可以估计这些样本的均值向量：

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{135}$$

11.2.1 样本的统计量

对于上面的样本，协方差矩阵可以表示为：

$$s_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j) \tag{136}$$

而样本的相关矩阵可以写成 $R = [r_{ij}]_{m\times m}$：

$$r_{ij} = \frac{s_{ij}}{\sqrt{s_{ii}s_{jj}}} \tag{137}$$

11.2.2 主成分的定义

我们可以设一个 m 维的随机变量 x 到 m 维的随机变量 y 的一个线性变换：

$$\mathcal{Y} = (y_1, y_2, \ldots, y_m) = A^T x = \sum_{i=1}^{m} \alpha_i x_i \tag{138}$$

$$y_i = \alpha_i^T x = \sum_{j=1}^{m} \alpha_{ji} x_j \tag{139}$$

11.2.3 主成分的统计量

对于随机变量 $\mathcal{Y} = (y_1, y_2, \ldots, y_m)$，其统计量有：

$$\begin{aligned} \bar{y}_i &= \frac{1}{n} \sum_{j=1}^{n} \alpha_i^T x_j = \alpha_i \bar{x} \\ \text{var}(y_i) &= \alpha_i^T S \alpha_i \\ \text{cov}(y_i, y_j) &= \alpha_i^T S \alpha_j \end{aligned} \tag{140}$$

11.2.4 PCA 的具体算法步骤

- 先将样本矩阵进行 normalization：

$$x_{ij}^* = \frac{x_{ij} - \bar{x}_i}{\sqrt{s_{ij}}} \tag{141}$$

- 计算样本的协方差矩阵 XX^T 并对其进行特征值的分解
- 取最大的 d 个特征值所对应的特征向量 w_j
- 输出投影矩阵 $W^* = (w_1, w_2, \ldots, w_d)$

11.2.5 PCA 代码实现

```
def PCA(data):
    """
    PCA Principal Component Analysis

    Input:
      data - Data numpy array. Each row vector of fea is a data point.
    Output:
      eigvector - Each column is an embedding function, for a new
                  data point (row vector) x, y = x*eigvector
                  will be the embedding result of x.
      eigvalue - The sorted eigvalue of PCA eigen-problem.
    """
```

```
13
14      # Hint: you may need to **normalize** the data before applying PCA
15      # begin answer
16      p, N = data.shape
17      normal_data = data - np.average(data, axis=1).reshape(p, 1)
18      conv = np.matmul(normal_data, normal_data.T) / N
19      eigen_values, eigen_vectors = np.linalg.eig(conv)
20      index = np.argsort(eigen_values)[:: -1]
21      eigen_values = eigen_values[index]
22      eigen_vectors = eigen_vectors[:, index]
23      # end answer
24      return eigen_vectors, eigen_values
```

12 线性判别分析 LDA

线性判别分析 (Linear Disciminant Analysis) 是一种经典的降维方法，并且是一种线性的学习方法，LDA 的基本想法就是，对于给定的训练集，我们可以设法将这些样本投影到一条直线上，并且使得同类样本点的投影尽可能接近而不同类的样本点的投影尽可能远离，这样一来我们实际上就需要训练出一条直线来进行特征空间中的分类，而对于测试集中的数据，可以将其同样投影到这条直线上面去，再根据投影点位置来确定该样本属于哪一类别。

12.1 LDA 问题的定义

我们先来研究二分类问题的 LDA，可以假设问题定义在数据机 D 上，用 X_i, μ_i, Σ_i 分别表示某一类别的数据集合，均值和协方差矩阵，如果能够将数据投影到特征空间中的一条直线 ω 上，那么两类样本的中心在直线上的投影分别是 $\omega^T\mu_0$ 和 $\omega^T\mu_1$，如果将两类样本的所有点都投影到这条直线上面，那么两类样本的协方差是 $\omega^T\Sigma_i\omega$

而我们的目的是希望同类别的点在直线上的投影尽可能靠近而不同类的尽可能远离。因此可以让同类别的投影点的协方差尽可能小，即一个目标可以定义成：

$$\min\left(\omega^T\Sigma_0\omega + \omega^T\Sigma_1\omega\right) \tag{142}$$

而同时我们也希望不同类别的数据样本点的投影尽可能远离，因此可以让类中心之间的距离尽可能大，则另一个目标可以表示为：

$$\max||\omega^T\mu_0 - \omega^T\mu_1||_2^2 \tag{143}$$

这样一来，我们可以结合两个优化目标，定义目标函数：

$$\begin{aligned} J &= \frac{\left\|\omega^T\mu_0 - \omega^T\mu_1\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} = \frac{\left\|\left(\omega^T\mu_0 - \omega^T\mu_1\right)^T\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \\ &= \frac{\left\|\left(\mu_0-\mu_1\right)^T\omega\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} = \frac{\left[\left(\mu_0-\mu_1\right)^T\omega\right]^T\left(\mu_0-\mu_1\right)^T\omega}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \\ &= \frac{\omega^T\left(\mu_0-\mu_1\right)\left(\mu_0-\mu_1\right)^T\omega}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \end{aligned} \tag{144}$$

12.2 LDA 的优化与求解

12.2.1 问题的改写

对于上面得到的优化目标，我们可以定义：

Definition 12.2.1 类内散度矩阵 (within-class scatter matrix)，用来表示同一个类别内的点的接近程度

$$S_w = \Sigma_0 + \Sigma_1 = \sum_{x \in X_0} (x - \mu_0)(x - \mu_0)^T + \sum_{x \in X_1} (x - \mu_1)(x - \mu_1)^T \tag{145}$$

Definition 12.2.2 类间散度矩阵 (between-class scatter matrix)，用来表示不同类别内的点的远离程度

$$S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T \tag{146}$$

这样一来上面的目标函数就可以变成：

$$J = \frac{\omega^T S_b \omega}{\omega^T S_w \omega} \tag{147}$$

这就是一种广义瑞丽商的优化形式。

12.2.2 瑞丽商 Rayleigh quotient

对于 $n \times n$ 的矩阵 A，瑞丽商的定义形式是：

$$R(A, x) = \frac{x^T A x}{x^T x} \tag{148}$$

瑞丽商这种形式的函数很重要的一个性质就是其最大值等于 A 的最大的特征值，而最小值等于 A 最小的特征值，即：

$$\lambda_{\min} \leq R(A, x) \leq \lambda_{\max} \tag{149}$$

可以用拉格朗日乘子法来证明这一结论，我们不妨设 $x^t x = 1$，这样一来根据拉格朗日乘子法，优化的目标函数可以写成：

$$f(x) = x^T A x + \lambda(x^T x - 1) \tag{150}$$

求梯度并令梯度等于 0 可以得到：

$$\frac{\partial f(x)}{\partial x} = Ax - \lambda x = 0 \rightarrow Ax = \lambda x \tag{151}$$

这其实就是矩阵特征值的定义形式，这里所有能够使得梯度为 0 的 x 就是 A 的所有特征值对应的特征向量，这样一来 x^TAx 在极值点的时候的计算结果就是 A 的一系列特征向量，因此该函数的最大值就是 A 最大的特征向量，而最小值就是 A 最小的特征向量。

而广义瑞丽商的定义是:

$$R(A,B,x)=\frac{x^TAx}{x^TBx} \tag{152}$$

广义瑞丽商可以变形为:

$$R(A,B,x)=\frac{x^TAx}{x^TBx}=\frac{\hat{x}^T(B^{-\frac{1}{2}}AB^{-\frac{1}{2}})\hat{x}}{\hat{x}^T\hat{x}} \tag{153}$$

同上面一样可以得到这个函数的最大值和最小值分别是矩阵 $B^{-\frac{1}{2}}AB^{-\frac{1}{2}}$ 的最大的特征值和最小值。

12.2.3 LDA 的求解

根据瑞丽商的性质，我们可以知道 LDA 的求解就是对矩阵 $S_w^{-\frac{1}{2}}S_bS_w^{-\frac{1}{2}}$ 进行特征值分解，最大的特征值对应的就是最大值，最小的特征值对应的就是最小值，同时又可以进行进一步的变形:

$$\omega=S_w^{-1}(\mu_0-\mu_1) \tag{154}$$

同时 LDA 也可以用贝叶斯决策理论来解释，并且可以证明，当两类数据满足同先验、满足高斯分布并且协方差相等的时候，LDA 可以达到最优的分类结果。

12.3 高斯分布下的 LDA

12.3.1 问题的定义

这一节内容从从贝叶斯决策理论来推导高斯分布下的 LDA 模型，如果样本满足高斯分布，我们先考虑二分类时候的情况，可以设两类样本分别满足高斯分布并且共享协方差矩阵，这样一类:

$$P(x|y=c,\theta)=\mathcal{N}(x|\mu_c,\Sigma)\quad c\in\{0,1\} \tag{155}$$

这样一来，根据贝叶斯公式，后验概率分布可以表示为：

$$\begin{aligned} P(y=c|x,\theta) =& \frac{P(x|y=c,\theta)P(y=c,\theta)}{\sum_{c\in C} P(x|y=c,\theta)P(y=c,\theta)} \\ \propto & \quad \pi_c \exp\left[\mu_c^T\Sigma^{-1}x - \frac{1}{2}x^T\Sigma^{-1}x - \frac{1}{2}\mu_c^T\Sigma^{-1}\mu_c\right] \\ =& \exp\left[\mu_c^T\Sigma^{-1}x - \frac{1}{2}\mu_c^T\Sigma^{-1}\mu_c + \log\pi_c\right]\exp\left[-\frac{1}{2}x^T\Sigma^{-1}x\right] \end{aligned} \tag{156}$$

这里我们可以令：

$$\begin{aligned} \gamma_c &= -\tfrac{1}{2}\mu_c^T\Sigma^{-1}\mu_c + \log\pi_c \\ \beta_c &= \Sigma^{-1}\mu_c \end{aligned} \tag{157}$$

这样一来要求的后验概率就可以写成：

$$p(y=c \mid x,\theta) = \frac{e^{\beta_c^T x+\gamma_c}}{\sum_{c'} e^{\beta_{c'}^T x+\gamma_{c'}}} = \mathcal{S}(\eta)_c \tag{158}$$

这其实就是一个 softmax 函数的形式，softmax 函数可以把一个分布标准化成一个和为 1 的概率分布，而 sigmoid **函数其实就是一个一元形式的** softmax **函数**

12.3.2 二分类下的特殊情况

对于二分类问题，我们可以进行这样的变形：

$$\begin{aligned} P(y=1 \mid x,\theta) &= \frac{e^{\beta_1^T x+\gamma_1}}{e^{\beta_1^T x+\gamma_1} + e^{\beta_0^T x+\gamma_0}} \\ &= \frac{1}{1+e^{(\beta_0-\beta_1)^T x+(\gamma_0-\gamma_1)}} = \text{sigmoid}\left(\left(\beta_1-\beta_0\right)^T x + (\gamma_1-\gamma_0)\right) \end{aligned} \tag{159}$$

这样就可以将二分类情况下的高斯分布转化成一个 sigmoid 函数的形式，根据之前的 β,γ 的定义，我们有：

$$\begin{aligned} \gamma_1-\gamma_0 &= -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \log\left(\pi_1/\pi_0\right) \\ &= -\frac{1}{2}\left(\mu_1-\mu_0\right)^T\Sigma^{-1}\left(\mu_1+\mu_0\right) + \log\left(\pi_1/\pi_0\right) \end{aligned} \tag{160}$$

因此我们可以定义：

$$\begin{aligned} \omega &= \beta_1-\beta_0 = \Sigma^{-1}\left(\mu_1-\mu_0\right) \\ x_0 &= \frac{1}{2}\left(\mu_1+\mu_0\right) - \left(\mu_1-\mu_0\right)\frac{\log\left(\pi_1/\pi_0\right)}{\left(\mu_1-\mu_0\right)^T\Sigma^{-1}\left(\mu_1-\mu_0\right)} \end{aligned} \tag{161}$$

这样一来我们就可以得到 (这个比较好推出):

$$\omega^T x_0 = -(\gamma_1 - \gamma_0) \tag{162}$$

因此，后验概率可以被写成:

$$P(y = 1|x, \theta) = \text{sigmoid}(\omega^T(x - x_0)) \tag{163}$$

我们可以将 $\omega^T(x - x_0)$ 看成是特征空间中的点 x 平移了 x_0 个单位之后投影到直线 ω 上面，而 sigmoid 函数则是对投影的结果进行一个分类，看投影点是更靠近哪一类别。这个形式就是线性判别分析的形式，即将样本点投影到一条直线上，然后看距离多个类别的中心点的距离来判断样本的类别。

12.3.3 参数估计和模型优化

我们可以来考虑用极大似然法来训练一个模型，我们可以将该问题的对数似然函数定义为:

$$\log P(\mathcal{D} \mid \boldsymbol{\theta}) = \left[\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbb{I}\left(y_i = c\right)\log \pi_c\right] + \sum_{c=1}^{C}\left[\sum_{i:y_i=c}\log \mathcal{N}\left(x \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right)\right] \tag{164}$$

这里的参数可以用样本的数据进行估计，比如 $\pi_c = \frac{N_c}{N}$，$\mu_c = \frac{1}{N_c}\sum_{i \in N(c)} x_i$，$\Sigma_c = \frac{1}{N}{}_c(x - \mu_c)(x - \mu_c)^T$，这些都是基于极大似然法的常见估计。