Machine Learning **2022 年 11 月 13 日**

What is Machine Learning?

lecturer: *Kaidong Wang* *Scriber: Jinyu Zhang*

1 什么是机器学习？

王开东老师在课程中首先讲了比较多的例子引入课程，比如：

Example 1 古代文献修复：原书籍已经变为分散的多个书页，书页数量大，且分布在多处；部分损毁较为严重，字迹模糊，并且需要大量掌握古文字的专业人才。以色列科学家利用一些已经确定相邻和一些已经确定不相邻书页作为训练集训练分类模型，并将机器学习算法应用于大量测试集中，系统的自动判断精度超过 *93%*。

Example 2 美国大选：通过机器学习模型进行选民群体的精准画像，投放广告；分析出哪些政治人物或者明星对某地的选民最优吸引力，并联合明星进行竞选筹资。

通过以上例子我们可以看出得出机器学习的定义，**机器学习**是从人工智能中产生的一个重要科学分支，是实现智能化的关键。通常是指：计算机利用已有的数据（经验），得出了某种模型，并利用此模型预测未来的一种方法。根据维基百科上的定义：

Definition 1 机器学习是人工智能的一个分支。机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。因为学习算法中涉及了大量的统计学理论，机器学习与推断统计学联系尤为密切，也被称为统计学习理论。

2 能否使用机器学习？

机器学习已广泛应用于数据挖掘、计算机视觉、自然语言处理、生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA 序列测序、语音和手写识别、战略游戏和机器人等领域。面对一个新问题（新领域），我们如何判断一个问题能否使用机器学习算法去解决呢？需要满足以下三个条件：

1. 存在一些可学习的模式（规律）————所以表现可以提升
2. 但是无法程序化地定义————所以需要机器学习
3. 同时拥有包含模式的数据————所以可以从中进行学习

比如：预测婴儿十分钟后是否会哭？婴儿的行为通常被认为是不可学习的（脑部未发育完全，行为模式未定型），所以无法使用机器学习模型进行预测。

3 机器学习的一般框架

或者说，机器学习的 Pipeline 究竟是什么样的？机器学习系统由哪几部分组成？机器学习的学习过程是在所有假设组成的空间（Hypothesis Space）中进行搜索的过程，是利用训练样本（训练样本组成的集合称为“训练集”）（Train Dataset）在假设空间中寻求一个损失函数最小/拟合优度最优的函数的过程。

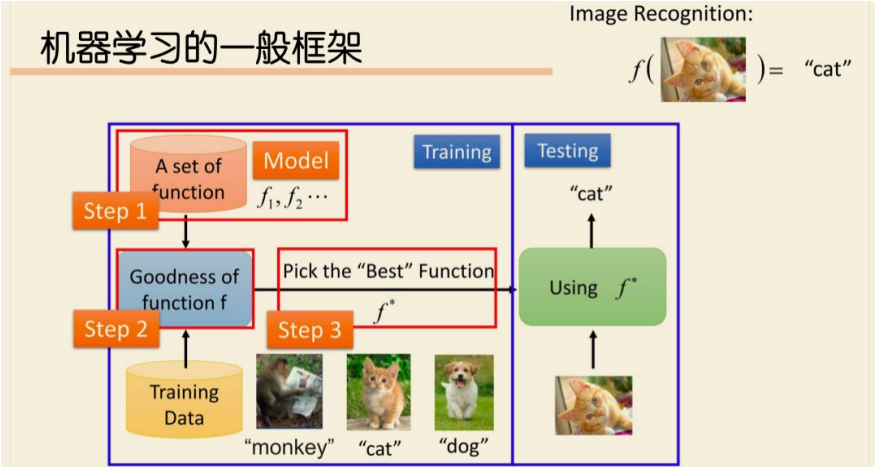

图 1: 机器学习的一般框架

一个机器学习项目通常可以分为四步：

$$\text{数据收集} \longrightarrow \text{数据清洗} \longrightarrow \text{特征工程} \longrightarrow \text{数据建模}$$

从机器学习的框架和机器学习项目的步骤中可以看出，数据对于机器学习算法的重要性不言而喻，**数据就好比是机器学习的燃料，成功的机器学习应用不是拥有最好的算法，而是拥有最多的数据！**

4 机器学习的主要内容

在了解了机器学习项目的一般框架之后，对应的每个步骤中我们应该做些什么？或者说机器学习的主要内容有哪些？

1. 首先要清楚*要学什么？*由机器学习的学习过程中我们可以知道，机器学习其实就是在学习一个**决策函数**。我们根据决策函数输出值是离散型数据还是连续型数据，可以将*有监督学习*划分为*分类问题*和*回归问题*两大类。
2. 然后要清楚*从哪里学？*，数据是机器学习算法的燃料，一个机器学习算法是需要从训练数据中学习，从中不断进行修正。
3. 最后要明确*怎样学习？*从数据中的学习使得决策函数不断修正，这其实就是一个优化问题，我们通过设计优化模型，将决策函数视为目标函数，在添加正则项/或者无约束的条件下，使用一些优化算法，通过训练数据对算法关键参数进行迭代优化（如梯度下降、牛顿法）。

最终的学习结果可能有多种情况，机器学习算法的学习情况也存在两种极端：

- **欠拟合：**算法在训练集和测试集表现效果都很差。模型拟合程度不高，数据距离拟合曲线较远，或指模型没有很好地捕捉到数据特征，不能够很好地拟合数据。
- **过拟合：**算法在训练集上表现的效果远远优于在测试集上表现的效果。模型在训练集上的拟合程度过高，模型对于特征的学习过度了。

Theorem 1 *奥卡姆剃刀：若有多个假设和所观测的结果一致，则选用最简单的那个。“若无必要，勿增实体。”*

以上两种情况都是实际过程中经常会遇到的问题。针对欠拟合，我们可以采用**数据增强、扩大数据集、重采样、合理设置数据权重**的方式，让算法能够充分学习到数据特征。针对过拟合，根据奥卡姆剃刀原理，我们可以采用**修剪特征（特征工程），添加正则项**的方式。

Machine Learning **PLA - 2022 年 11 月 13 日**

Perceptron Linear Algorithm

lecturer: *Kaidong Wang* *Scriber: Jinyu Zhang*

Perceptron Linear Algorithm

1 引入

感知机主要用于解决分类问题，比如如何将垃圾邮件与正常邮件分开？如何判断不同顾客有无资格办理信用卡？我们可以采用线性模型，将收集到的顾客特征（如年龄、工作年限、工资收入等）进行线性组合。规定某一阈值进行类别划分。不妨设 $\mathbf{x} = (x_1, \ldots, x_d)$ 代表测试数据的特征，比如顾客的年龄，性别，工作年限，工资等；设 $y \in \{+1, -1\}$ 代表不同的类别取值，感知机算法的思想就是通过训练数据去学习一个线性组合参数 $w = (w_1, \ldots, w_d)$，使得：

$$
\begin{aligned}
h(x) &= sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right) \\
&= sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold \times 1\right) \\
&令 w_0 = -threshold, x_0 = 1 \\
&= sign\left(\sum_{i=0}^{d} w_i x_i\right) \\
&= sign(\boldsymbol{w}^\top \boldsymbol{x})
\end{aligned}
$$

在训练数据和测试数据中预测样本类别和真实类别的误差较小，可以用于线性分类。

2 感知机理论：

对于一个线性可分的数据集 $\mathbb{T}$。

$$T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

其中，$x \in \mathbb{X} = \mathbb{R}^n$，$y_i \in \mathbb{Y} = \{-1, 1\}$，$i = 1, 2, \ldots, N$，感知机算法是求解参数 w, b，使得损失函数 (1) 极小化：

$$\min_{w,b} L(w, b) = -\sum_{x \in \mathbf{M}} y_i (w \cdot x_i + b) \tag{1}$$

感知机算法是误分类点驱动的，其中 $\mathbf{M}$ 为所有误分类点的集合，即 (x_i, y_i) 满足：

$$y_i (w \cdot x_i + b) < 0$$

我们可以采用随机梯度下降法（SGD）对 $LossFunction$ 进行优化，具体地：初始化 w_0, b_0。然后遍历选取样本集中的点 (x, y)，如果该点满足 $y(w \cdot x + b) < 0$，即误分类，则对超平面参数 w, b 进行更新：

$$w := w + \alpha y x$$
$$b := b + \alpha y$$

其中 $\alpha \in (0, 1]$ 成为学习率。通过迭代，直至损失函数 $L(w, b)$ 不断减少到 0，也即没有误分类点。综上所述得到以下算法：

Algorithm 1 PLA

输入:训练集数据 $\mathbf{T}=\{(x_1, y_1), \ldots, (x_N, y_N)\}$,其中 $x \in \mathbb{X} = \mathbb{R}^n$,$y_i \in \mathbb{Y} = \{-1, 1\}$，$i = 1, 2, \ldots, N$，$\alpha \in (0, 1]$;

1. 初始化超平面，选取参数 w_0, b_0

2. 在训练集中选择数据 (x_i, y_i);

3. if $y_i(w \cdot x_i + b) \leq 0$，

$$w := w + \alpha y x$$
$$b := b + \alpha y$$

4. 转 2，直至训练集数据中没有误分类点。

---

3 PLA 算法的收敛性分析

设训练集数据 $\mathbf{T}=\{(x_1, y_1), \ldots, (x_N, y_N)\}$ 是线性可分的，其中 $x \in \mathbb{X} = \mathbb{R}^n$，$y_i \in \mathbb{Y} = \{-1, 1\}$，$i = 1, 2, \ldots, N$。

Theorem 2 *存在 $\|\hat{w}_{opt}\| = 1$ 的超平面 $\hat{w}_{opt}\hat{x} = w_{opt}x + b_{opt} = 0$ 将训练数据集完全正确分开；且存在 $\gamma > 0$ 对所有的 $i = 1, \ldots, N$*

$$y_i(\hat{w}_{opt}\hat{x}) = y_i(w_{opt}x + b_{opt}) \geq \gamma$$

Theorem 3 *令 $R = \max\limits_{1 \leq i \leq N} \|\hat{x}_i\|$，则感知机算法在训练数据上的误分类次数 k 满足不等式：*

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

定理3表明，误分类次数 k 是有上界的，经过有限次搜索可以找到将训练数据完全正确分开的分离超平面。也就是说，当训练数据集线性可分时，感知机算法原始形式迭代是收敛的。但由于初始参数是随机生成的，线性可分的超平面也不止一条。所以感知机学习算法存在很多个解，这些解既依赖于初值的选择，也依赖于迭代过程中误分类点的选择顺序。当训练集线性不可分时，感知机算法不收敛，迭代结果会发生震荡。

4 Pocket 算法

根据 PLA 算法的收敛性分析板块中，我们知道了*对于线性可分的数据集，PLA 算法是可以收敛的*。如果数据集中添加噪声项，使得原数据集从 D 变为 D'，使得 D' 中数据并非线性可分的，此时 PLA 算法是无法收敛的。此时，修改我们的迭代停止条件，当 w^* 迭代到使得分类错误足够小的时候，停止迭代，Pocket 算法如下：

Algorithm 2 Pocket-PLA

输入:训练集数据 $\mathbf{T}=\{(x_1,y_1),\ldots,(x_N,y_N)\}$,其中 $x\in\mathbb{X}=\mathbb{R}^n$,$y_i\in\mathbb{Y}=\{-1,1\}$，$i=1,2,\ldots,N$，$\alpha\in(0,1]$;

1. 初始化超平面，选取参数 w_0,b_0
2. while 迭代次数小于规定值：
3. 在训练集中随机选择数据 (x_i,y_i);
4. if $y_i(w\cdot x_i+b)\leq 0$，

$$w^{t+1}=w^t+\alpha yx$$
$$b^{t+1}=b^t+\alpha y$$

5. if $\sum_{n=1}^{N}\left[y_n\neq sign(w^{t+1\top}x_n)\right]<\sum_{n=1}^{N}\left[y_n\neq sign(w^{t\top}x_n)\right]$:
6. $w^*=w^{t+1}$

---

python 代码：

写出 PLA 算法，并取 iris 数据集的 Speal.Length，Speal.Width 作为特征，便于可视化，用 python 实现 PLA、pocket 算法以及可视化代码如下：

```
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
class Perceptron:
def __init__(self, X, y):
    '''
    :param X: shape ——> (m,n) m个数据 n个参数
    :param y: shape ——> m
    '''
    self.X = X
    self.y = y
    self.m = np.shape(X)[0]
    self.n = np.shape(X)[1]
    self.w = np.random.random(self.n)
```

```
16      self.b = 0
17
18  def loss(self):
19      res = self.y*np.sign(self.X.dot(self.w) + self.b)
20      mistake = np.where(res < 0)[0]
21      loss = len(mistake)
22      return loss
23
24  def train(self,n_iter=10000,a=1):
25      cnt=0
26      while self.loss() > 0 and cnt<n_iter:
27          res = self.y * np.sign(self.X.dot(self.w) + self.b)
28          mistake = np.where(res < 0)[0]
29          temp = self.y[mistake[0]] * self.X[mistake[0]]
30          self.w = self.w+a*temp
31          self.b = self.b+a*y[mistake[0]]
32          cnt+=1
33      return self.loss(), self.w, self.b
34
35  if __name__ == '__main__':
36  def load_data():
37  # 加载iris数据集s
38      iris = load_iris()
39      # 转换成df格式，然后将列名设置为对应的标签名
40      df = pd.DataFrame(iris.data, columns=iris.feature_names)
41      df['label'] = iris.target
42      df.columns = ['sepal length', 'sepal width', 'petal length', 'petal
            width', 'label']
43      # 为了方便可视化，只使用sepal length 和 sepal width作为特征
44      return df
45  df = load_data()
46  #选取前100个数据，选择的列为第0列，第1列，最后一列（标签）
47  data = np.array(df.iloc[:100,[0,1,-1]])
48  X,y = data[:,:-1],data[:,-1]
49  #标签标准化为{-1，1}
50  y[np.where(y==0)]=-1
51  #positive point
52  p=X[np.where(y==1)]
53  #negetive point
54  n=X[np.where(y==-1)]
55  #设置坐标轴范围
56  plt.xlim(min(X[:,0])-1,max(X[:,0])+1)
57  plt.ylim(min(X[:,1])-1,max(X[:,1])+1)
```

```
58 #不同颜色画点
59 plt.scatter(p[:,0],p[:,1],c='r',label='positive')
60 plt.scatter(n[:,0],n[:,1],c='b',label='negetive')
61 # print(X)
62 pla = Perceptron(X, y)
63
64 loss,w,b=pla.train()
65 print(loss,w,b)
66 #斜截式画直线方程
67 plt.plot([min(X[:,0])-2,max(X[:,0])+2],[(-w[0]*(min(X[:,0])-2))/w[1]-b/w
       [1],-w[0]*(max(X[:,0])+2)/w[1]-b/w[1]],c='g')
68 plt.legend()
69 plt.show()
```

最终的运行结果如下:

$$loss = 0.0$$

$$w = [77.93862747, -99.9041861]$$

$$b = -121.0$$

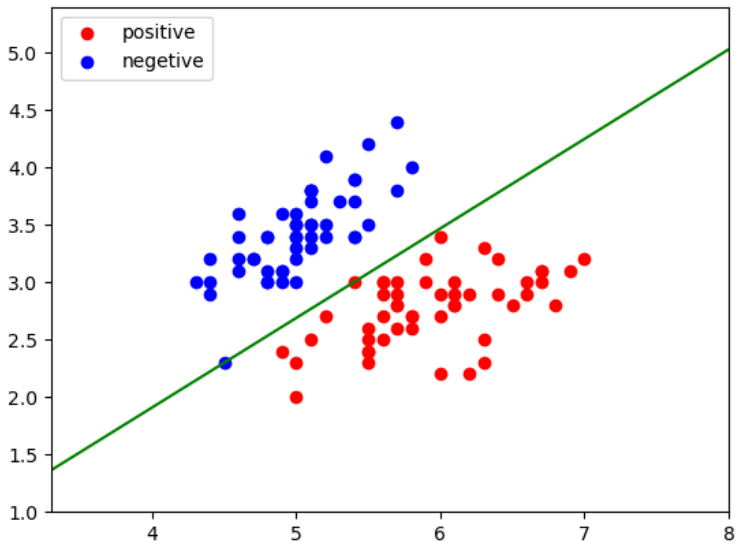

图 2: iris 数据集可视化结果

简单的学习实例

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 线性回归

有监督学习是机器学习的一个重要分支，根据输出标签类型是否连续，可以将有监督学习分为**回归问题**和**分类问题**。其中，线性回归是一类最简单的回归模型。

Definition 2 线性回归 *(Linear Regression)* 是一种通过属性的线性组合来进行预测的线性模型，其目的是找到一条直线或者一个平面或者更高维的超平面，使得预测值和真实值之间的误差最小化。

线性回归的算法流程

线性回归拟合值可以表示为：

$$h(x) = w_0 + w_1 x_1 + \ldots + w_n x_n = \hat{y}$$

线性回归的损失函数通常取平方和损失函数，从统计意义上来说，这是一种残差平方和最小的优化方法（MSR）：

$$\begin{aligned} l(x^i) &= \frac{1}{2}\left(h(x^i) - y^i\right)^2 \\ \text{要找到一组} w &= (w_0, w_1, \ldots,) \\ L(w) &= \frac{1}{2}\sum_{i=1}^{m}(h(x^i) - y^i)^2 \text{最小} \end{aligned} \tag{2}$$

从统计角度来看，式2也被称为最小二乘法，若系数矩阵 $(X\top X)^{-1}$ 可逆，则可以求得闭式解：

$$\begin{aligned} \frac{\partial J}{\partial w} &= \frac{1}{2}\frac{\partial}{\partial w}(Xw - Y)^\top (Xw - Y) \\ &= X^\top X w - X^\top Y \\ w &= (X^\top X)^{-1} X^\top Y \end{aligned} \tag{3}$$

式3通常也被称为 *normal equation*，但若数据集过大，根据数值计算相关知识，求解矩阵逆和矩阵乘法的运算复杂度达到了 $O(n^3)$，直接对大规模数据集求解析解会花费大量时间，通常来说如果 n 小于 10000 还是可以接受的。但如果训练数据集过大，不利于机器学习算法的实际部署。所以，通常使用优化迭代算法求解机器学习模型，常用的迭代优化算法有梯度下降，牛顿法等。在机器学习课程中我们主要使用了随机梯度下降的方法。

梯度下降与随机梯度下降

梯度下降是应用最广泛的迭代优化算法之一，由于其形式简单，计算量相对较小，从而广泛使用于人工智能领域。梯度下降有以下的三种形式：

1. 批量梯度下降（Batch Gradient Descent,BGD）

$$w^{t+1} = w^t - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\left(h(x^t) - y^t \right) x^t \right)$$

   在梯度下降的每一步中，都用到了所有的训练样本。

2. 随机梯度下降（Stochastic Gradient Descent,SGD）

$$w^{t+1} = w^t - \alpha \left(h(x^t) - y^t \right) x^t$$

   梯度下降的每一步中，用到一个样本，在每一次计算之后便更新参数，而不需要首先将所有的训练样本求和。

3. 小批量梯度下降（Mini-Batch Gradient Descent,MBGD）

$$w^{t+1} = w^t - \alpha \frac{1}{B} \sum_{k=i}^{i+b-1} \left(h(x^t) - y^t \right) x^t$$

   梯度下降的每一步中，都用到了一定批量的训练样本。

数据标准化与数据归一化

在机器学习模型中为什么需要数据标准化与归一化？

- **提高模型精度：**不同维度之间的特征在数值上有一定比较性，可以大大提高分类器的准确性。
- **加速模型收敛：**最优解的寻优过程明显会显得平缓，更容易正确的收敛到最优解。

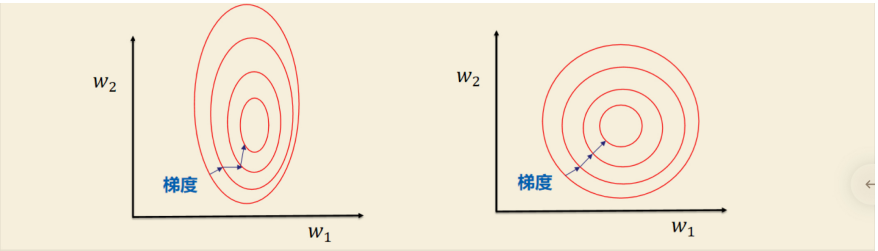

图 3: 平方损失函数优化示意图

如图，以一个线性回归的平方损失函数为例，这是一个经典的二次规划问题，其损失函数的等值线是一条一条的椭圆，当数据尺度差异过大，椭圆就会变得十分扁平，在求梯度下降的过程中会产生震荡的路径，导致迭代次数过多。当数据归一化之后，损失函数等值线会变为趋向于圆形，可以加速梯度下降算法的收敛。

– 归一化

$$x^* = \frac{x - x_{min}}{x_{max} - x_{min}}$$

数据归一化的目的是使得个特征对目标变量的影响一致，但会对特征数据进行伸缩变化，所以数据归一化是会改变特征数据分布的。

– Z-Score 标准化

$$x^* = \frac{x - \mu}{\sigma}$$

处理后的数据均值为 0，方差为 1。数据标准化为了不同特征之间具备可比性，经过标准化变换之后的特征数据分布没有发生改变，当数据特征取值范围过大或者单位差异较大时，最好做一下标准化处理。

什么模型需要做数据归一化和标准化

– **需要做数据归一化/标准化**
线性模型，基于距离度量的模型，比如 KNN、K-means 聚类、感知机和 SVM。另外，线性回归类的几个模型一般情况下也是需要做数据归一化/标准化处理的。

– **不需要做数据归一化/标准化**
决策树、基于决策树的 Boosting 和 Bagging 等集成学习模型对于特征取值大小并不敏感，如随机森林、XGBoost、LightGBM 等树模型，以及朴素贝叶斯，以上这些模型一般不需要做数据归一化/标准化处理。

Machine Learning **Logistic Regression- 2022 年 11 月 13 日**

Logistic Regression

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 引入

上一节中的线性回归模型是指使用线性模型进行回归训练，而在上节中我们也讲到过监督学习模型可以分为分类和回归两大任务，如何使用线性模型进行回归是本节的重要主题，我们引入逻辑回归模型来实现。

考虑一个二分类任务，输出标签为 $y \in \{0,1\}$，而线性回归模型产生的预测值 $z = w^\top x + b$ 是实值，于是我们需要将实值 z 转化为 0/1 值，最理想的是单位跃阶函数。但单位跃阶函数显然是不连续的，因此不能直接用作我们的损失函数，因此我们采用单位跃阶函数的一个最常用的替代函数，对数几率函数，或者称为 sigmoid 函数。

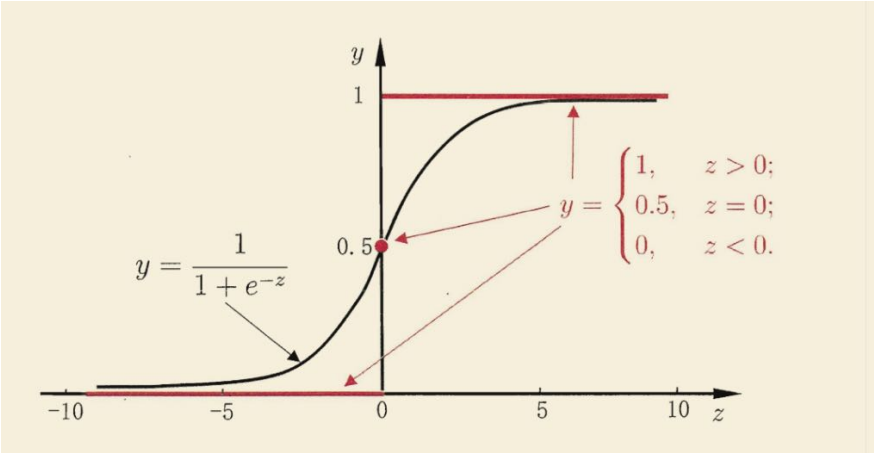

图 4: 对数几率函数与单位阶跃函数

2 逻辑回归

逻辑回归是一种有监督二分类模型。对于给定数据集 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，逻辑回归将二分类问题转化为软二分类问题将输出转化为 01 之间的概率值，即从假设空间中找到最优的函数 h，按照条件概率的大小将样本点分类到概率值较大的那一类。也即：

$$P(y = 1|x; \theta) = h_\theta(x)$$
$$P(y = 0|x; \theta) = 1 - h_\theta(x)$$

可以简写为：

$$P(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

逻辑回归是一种广义线性模型，可以使用 $Sigmoid$ 函数将线性函数值 $\theta^T\boldsymbol{x}$ 映射在 $[0,1]$ 之间，即令：

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T\boldsymbol{x}}}$$

逻辑回归的核心是设计损失函数并求解最优的参数值。假设 m 个训练样本是独立同分布的，对参数使用极大似然的方法进行估计：

$$\max_\theta \ L(\theta)$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N}(p(x_i))^{y_i}(1-p(x_i))^{1-y_i} \\ logL(\theta) = l(\theta) &= \sum_{i=1}^{N}(y_i \ln h_\theta(x_i) + (1-y_i)\ln(1-h_\theta(x_i))) \\ 可以化简为：l(\theta) &= \sum_{i=1}^{n}(-y_i x_i^T\theta + \log(1+e^{x_i^T\theta})) \end{aligned} \tag{4}$$

式4所示的损失函数也称为交叉熵 $(cross-entropy)$。我们可以采用梯度下降或者牛顿法对损失函数进行优化。而梯度下降方法又可以分为 batch-GD 与 S-GD。面对大规模数据集，我们常常选用随机梯度下降的方法进行优化损失函数，本文中我使用的方法是随机梯度下降（SGD）。

损失函数优化：随机梯度下降

对于式4的无约束优化问题，我们首先考虑随机梯度下降的方法进行优化，也即：

Algorithm 3 Stochastic Gradient Descent of Logistic loss

输入： 目标函数 $l(\theta) = \sum_{i=1}^{n}(-y_i x_i^T\theta + \log(1+e^{x_i^T\theta})) : \mathbb{R}^n \to \mathbb{R}$

1. 初始化点 $\theta^0 \in \mathbb{R}^n$
2. while $\|\nabla l(\theta^t)\| \geq 0$:(考虑到计算精度问题在实际代码编写时一般都规定一个 ε,求解 $\varepsilon-suboptimal$)

$$\theta^{t+1} = \theta^t - s^t\nabla l_{i_k}(\theta^t)$$

where $i_k \in \{1,\ldots,n\}$is some index chosen randomly at iteration k.

$$k \leftarrow k+1$$

end while

输出： 最优解 θ^*（近似最优解）与最优值 $l(\theta^*)$

我们利用 $iris$ 数据集的前 100 个样本进行测试（因为 $iris$ 数据集的前 100 个样本只有 0 和 1 两种类

别，可以使用 *Logistic Regression* 模型进行测试）。编写代码并可视化分类结果如图5，模型输出结果如图6：

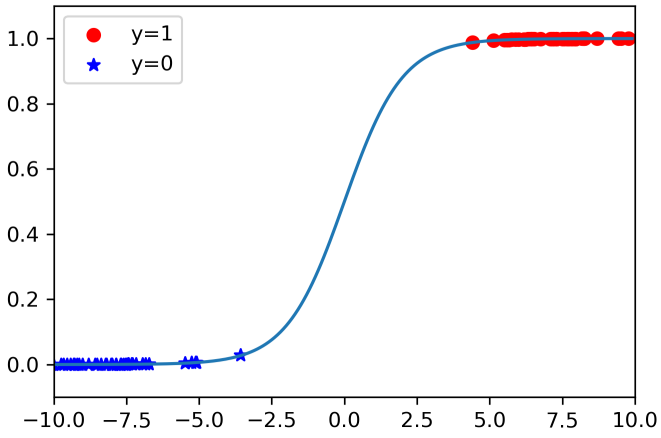

图 5: Logistic Regression 分类结果可视化

```
t=991,loss:0.13917945921558764,w=[ 0.62233975  2.87811295 -4.67444029 -1.2902682   1.18302959]
t=992,loss:0.13954114982973367,w=[ 0.62108907  2.87761268 -4.67531577 -1.29051834  1.18277945]
t=993,loss:0.13665446360270875,w=[ 0.63043418  2.88377196 -4.67234233 -1.29009356  1.18490334]
t=994,loss:0.136678510289955,w=[ 0.63032436  2.88372262 -4.67242031 -1.29011743  1.18488742]
t=995,loss:0.13317618861730507,w=[ 0.64842908  2.89654679 -4.66525386 -1.28936307  1.18865924]
t=996,loss:0.13318961037244553,w=[ 0.64819267  2.89643253 -4.66543117 -1.28942217  1.18861984]
t=997,loss:0.1324870854426093,w=[ 0.65881532  2.90330601 -4.66189029 -1.28838074  1.19070271]
t=998,loss:0.13248245950274087,w=[ 0.65869229  2.90324954 -4.66198508 -1.28840494  1.19068254]
t=999,loss:0.13244977737130104,w=[ 0.66000352  2.90419653 -4.66157229 -1.28830781  1.19092536]
t=1000,loss:0.13242651307176143,w=[ 0.66194198  2.90548884 -4.66100215 -1.28823179  1.19130545]
```

图 6: 随机梯度下降结果

从二分类到多分类

一对多策略

多分类问题往往有大于 2 个类别，传统的 Logistic 回归只能处理简单的二分类任务。我们利用数学中化归的思想，考虑将多分类问题转化为二分类问题。设给定数据集 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$ 表示共有 K 个类别，一对多策略（OVA）的中心思想即为，对于每一个类别 k，仅考虑两种类别，即属于类别 k 的样本点和不属于 k 的样本点。对于每一个类别均使用逻辑回归算法判断其落入 k 类别的概率 $P(y = k|x)$，最终使得分类结果为后验概率最大的类。即 $k^* = \underset{y \in \mathcal{Y}}{arg\max}\ P(y = k|x)$。OVA 算法的流程如下：

Algorithm 4 OVA Logistic

输入： 样本数据 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$

1. 初始化点 $\theta^0 \in \mathbb{R}^n$

2. for $y \in \mathcal{Y}$

3. 重新构造数据集的标签，即 $D_K = \{x_i, y_i^{'} = \mathbf{21}(y_i = k) - 1\}_{n=1}^{N}$

4. 对构造的临时数据集 D_k 使用 Logistic Regression 算法，并记录 $P(y = k|x)$ 的值

5. return $k^* = \underset{y \in \mathcal{Y}}{arg\max}\ P(y = k|x)$

输出：最优解分类结果 k^*

一对一策略

在一对多学习算法中，经常会出现由于“一个类别”对应“多个类别”而导致的**样本不平衡问题**，而如果使用一对一学习策略，能够更好的缓解样本不平衡的问题。设给定数据集 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$ 表示共有 K 个类别，一对一策略（OVO）的中心思想即为，对于 k 个类别，考虑 C_k^2 个类别组合。对于每一个类别组合均使用逻辑回归算法判断其落入哪个类别中，最终令得到的 C_k^2 个分类器进行 voting，投票结果最多的类即为最优的类。OVO 算法的流程如下：

Algorithm 5 OVA Logistic

输入：样本数据 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$

1. 初始化点 $\theta^0 \in \mathbb{R}^n$

2. for $y \in \mathcal{Y}$

3. 重新构造数据集的标签，即 $D_{kl} = \{x_i, y_i^{'} = \mathbf{21}(y_i = k) - 1 | y = k \ or \ y = l\}_{n=1}^{N}$

4. 对构造的临时数据集 D_{kl} 使用 Logistic Regression 算法，并记录落入哪个类的概率更大

5. return k^*，其中 k^* 是 C_n^2 个分类器进行 voting，投票结果最多的类即为最优的类。**输出：**最优解分类结果 k^*

Appendix

A. 随机梯度下降求解逻辑回归

```
import numpy as np
from math import exp,log
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt

def load_data():
    # 加载iris数据集s
        iris = load_iris()
```

```
10          # 转换成df格式，然后将列名设置为对应的标签名
11          df = pd.DataFrame(iris.data, columns=iris.feature_names)
12          df['label'] = iris.target
13          df.columns = ['sepal length', 'sepal width', 'petal length', '
                petal width', 'label']
14          return df
15
16  class logistic_regression:
17      def __init__(self,X,y,max_iter=1000) -> None:
18          # 带标签的训练数据
19          '''
20          X m,n  m个  n个特征
21          y m,1  标签
22          '''
23          self.X=np.concatenate([X,np.ones((X.shape[0],1))],axis=1)
24          self.y=y
25          self.iter=max_iter
26          self.m=self.X.shape[0]
27          self.n=self.X.shape[1]
28          self.w=np.random.random(self.n)
29      def logit_func(self,x):
30          return 1/(1+exp(-self.w@x))
31      # 定义损失函数
32      def loss_func(self):
33          loss=0
34          for i in range(self.m):
35              loss+=-(self.y[i]*log(self.logit_func(self.X[i,:]))+(1-self.y[
                    i])*log(1-self.logit_func(self.X[i,:])))
36          return loss
37      def dloss(self):
38          d=0
39          for i in range(self.m):
40              d+=(self.y[i]-self.logit_func(self.X[i,:]))*self.X[i,:]
41          return d
42      def train(self,sk=0.5):
43          cnt=0
44          while cnt<1000:
45              #随机梯度下降
46              i=np.random.randint(self.m)
47              self.w=self.w-sk*(self.y[i]-self.logit_func(self.X[i,:]))*self
                    .X[i,:]
48              cnt+=1
49              print(f't={cnt},loss:{self.loss_func()},w={self.w}')
```

```
50          return self.w
51
52  if __name__=='__main__':
53      df = load_data()
54      X,y = df.iloc[:100,:-1].values,df.iloc[:100,-1].values
55
56      logit_solver=logistic_regression(X,y)
57      w=logit_solver.train()
58      y_pred=[]
59      X=np.concatenate([X,np.ones((X.shape[0],1))],axis=1)
60      for i in range(X.shape[0]):
61          y_pred.append(1/(1+exp(-w@X[i,:])))
62      plt.figure(figsize=(5,5),dpi=300)
63      plt.xlim([-10,10])
64      plt.ylim([-0.1,1.1])
65      plt.plot(np.arange(-10,10,0.1),[1/(1+exp(-i)) for i in np.arange
            (-10,10,0.1)])
66      neg=X[np.where(y==0)[0],:]
67      pos=X[np.where(y==1)[0],:]
68      plt.scatter([w@neg[i,:] for i in range(neg.shape[0])],[1/(1+exp(-w@neg
            [i,:])) for i in range(neg.shape[0])],color='r',marker='o',label='y
            =1')
69      plt.scatter([w@pos[i,:] for i in range(pos.shape[0])],[1/(exp(-w@pos[i
            ,:])) for i in range(pos.shape[0])],color='b',marker='*',label='y=0
            ')
70      plt.legend()
71      plt.show()
```

Machine Learning **nonlinear transformation-2022 年 11 月 13 日**

非线性变换

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 引入

在前几节的学习中，我们学习了线性模型，并且介绍了有监督学习模型中的分类与回归问题。在回归问题中，我们主要讨论了线性回归与最小二乘法，在分类问题中，我们讨论了感知机算法 (PLA) 和逻辑回归。但上述两种分类问题的学习中我们主要分析了可严格线性可分的数据集，对多分类问题和非严格线性可分的情况仅做了简要的介绍。

在本节中，我们重点讨论不能线性可分的数据集。如何使得线性不可分数据集能够线性区分呢？

2 非线性变换

以一个典型的非线性可分的例子————圆形可分为例：采用化归的思想，用高维的非线性变换将圆形

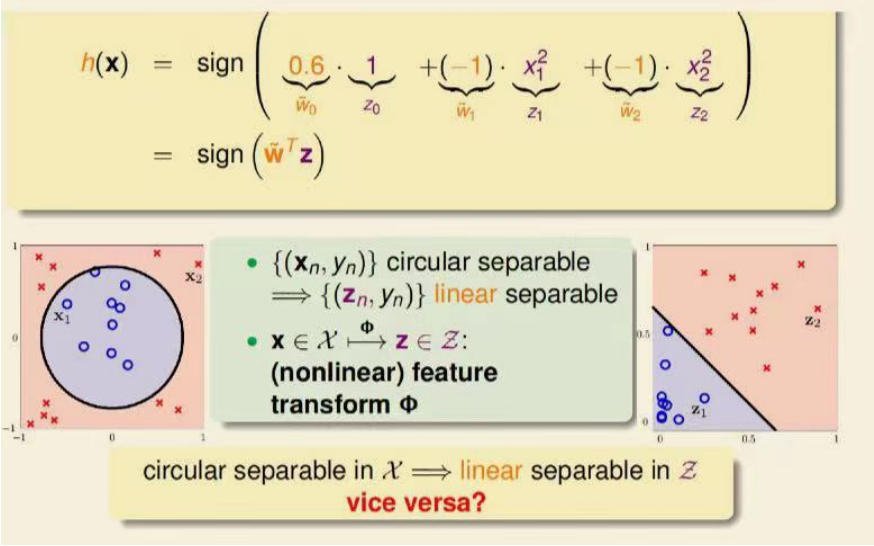

图 7: 圆形可分 vs 线性可分

可分的数据集化为线性可分的数据集。

$$\{\boldsymbol{x_i}, y_i\}_i^N \text{ 圆形可分} \longrightarrow \{\boldsymbol{z_i}, y_i\}_i^N \text{ 线性可分}$$
$$\text{即寻找非线性特征变换 } \boldsymbol{\Phi}\text{，使得 } x \in \mathcal{X} \xrightarrow{\Phi} y \in \mathcal{Y}$$

如何寻找一个非线性变换？以一个二元圆形可分的数据集为例，如果寻找一个高维非线性变换 $\boldsymbol{\Phi}$，使其

变为线性变换。则 $\Phi = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$，这个变换对于 $x \in \mathcal{R}^2$ 的数据点来说是完备的。若对于一个 $x \in \mathcal{R}^n$ 的数据集，能够通过一个二次变换将二次可分的数据集转换为线性可分的数据集。则需要 1 个常数项、C_n^2 个交互项以及 n 个二次项。非线性变换 Φ 的维数至多为 $\frac{n^2}{2} + \frac{3d}{2} + 1$。

模型的评估

正如同4 机器学习的主要内容中讲的那样，模型特征的选择往往会导致欠拟合或者过拟合，由上面的讨论我们可以知道非线性变换的维数（或者说特征）至多为 $\frac{n^2}{2} + \frac{3d}{2} + 1$ 个。然而对于实际数据集来说，过多的非线性变换特征维数，往往会导致过拟合。如何进行有效的特征选择？

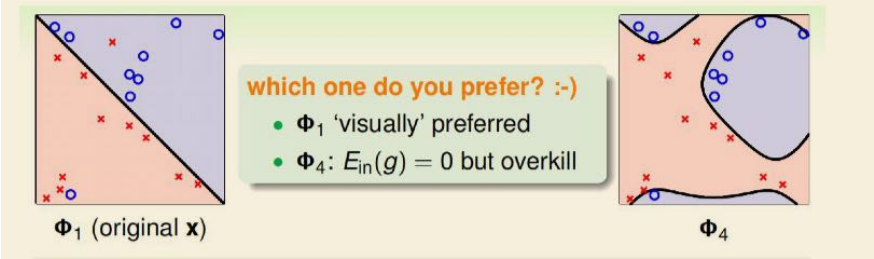

图 8: 过拟合

我们应该使用训练数据与测试数据进行模型选择。用带标签的训练数据训练模型，并用一些未知的数据进行模型的验证。选取泛化误差最小的模型。

> 泛化误差描述的是即从样本数据中推导出的规则能够适用于新的数据的能力。泛化误差越小，这种能力越强。

但我们手中并没有大量新的数据...，通常采用将样本数据划分为训练集与测试集的方法进行模型的训练与评估。常见的方法有：**留出法、交叉验证法、自助法**。

- **留出法**
  测试集留出法即从样本数据中随机采样/分层采样的方式抽取一部分数据进行测试。训练集不能太大，通常取 $\frac{1}{5}$ $\frac{1}{3}$ 的数据。

- **交叉验证法**
  k 折交叉验证法，即平均将数据集分为 k 份。每次使用 $\frac{1}{k}$ 的数据集进行测试，$\frac{k-1}{k}$ 的数据集进行训练。经过 k 轮的重复验证，计算误差的均值作为模型的泛化误差。

- **自助法**
  在统计学中，自助法（Bootstrap Method，Bootstrapping 或自助抽样法）是一种从给定训练集中有放回的均匀抽样，也就是说，每当选中一个样本，它等可能地被再次选中并被再次添加到训练集中。对于小数据集，自助法效果很好。
  假设给定的数据集包含 d 个样本。该数据集有放回地抽样 m 次，产生 m 个样本的训练集。这样原数据样本中的某些样本很可能在该样本集中出现多次。没有进入该训练集的样本最终形成检验集（测试集）。显然每个样本被选中的概率是 $1/m$，因此未被选中的概率就是 $(1 - 1/m)$，这样一

个样本在训练集中没出现的概率就是 m 次都未被选中的概率，即 $(1-1/m)^m$。当 m 趋于无穷大时，这一概率就将趋近于 $\frac{1}{e}=0.368$，所以留在训练集中的样本大概就占原来数据集的 63.2%。自助法在数据集较小、难以有效划分训练集和测试集时很有用；此外，自助法能从初始数据集中产生多个不同的训练集，这对集成学习等方法有很大的好处。然而，自助法产生的数据集改变了初始数据集的分布，这会引入估计偏差。因此，在初始数据量足够时，留出法和交叉验证法更常用一些。

超参数调参与验证集

超参数是算法中一般由人工设定的参数，如正则化项的系数。模型的参数一般是由学习过程决定的，比如说线性回归，感知机算法中的 $w^\top$。怎样确定超参数也是模型最终效果的重要一环，参数调的好不好对最终性能往往有关键影响。

调参过程通常是设置一系列超参数的值，得到若干模型，然后基于某种评估方式进行选择。验证集 (validation set) 就是用于训练确定超参数的数据集。

性能度量

性能度量是衡量模型泛化能力的评价标准。回归任务常用均方误差：

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

分类问题常用错误率与精度度量：

$$\begin{aligned}\text{错误率：}\mathcal{E}(f;D)&=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f(x_i)\neq y_i\right)\\ \text{精度：}acc(f;D)&=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left(f(x_i)=y_i\right)\\ &=1-\mathbb{E}(f;D)\end{aligned}$$

对于一个二分类问题的混淆矩阵：其中 TP,FP,TN,FN 分别表示真正例、假正例、真负例和假负例。查

表 1: 混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP	FN
反例	FP	TN

准率 P 与查全率 R 分别定义为：

$$\text{查准率：} P = \frac{TP}{TP + FP}$$

$$\text{查全率：} P = \frac{TP}{TP + FN}$$

$$\text{F1 分数：} F_1 = \frac{2 \times P \times R}{P + R}$$

若对查准率/查全率有不同偏好：

$$F_\beta = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P) + R}$$

其中 $\beta > 1$ 时查全率有更大的影响，$\beta < 1$ 时查准率有更大的影响。

ROC 曲线与 AUC

在介绍 ROC 曲线之前，首先先介绍准确率与精确率两个概念。真正例率（TPR）表示正确预测的正例占所有预测为正例数量的比值。假正例率（FPR）表示错误预测的正例数量占负类的比值。画 ROC 曲线时对于规定的阈值，给出不同阈值下的 TPR 与 FPR，以 FPR 为横坐标，TPR 为纵坐标画图。曲线下方的面积即为 AUC。

以我做过的一道题目为例：

下表中数据对象已经按分类器预测为 P 类的概率值递减排序。

1. 若阈值设置为 0.55，试给出混淆矩阵。
2. 绘制 ROC 曲线。

ID	预测概率	实际类
1	0.95	P
2	0.85	N
3	0.78	P
4	0.66	P
5	0.60	N
6	0.55	P
7	0.53	N
8	0.52	N
9	0.51	N
10	0.40	P

图 9: 第二题样本数据

解

1. 由题目数据可知，若阈值为 0.55，则给出上表中最终分类结果：由上表结果可得：

表 2: 分类结果表

ID	1	2	3	4	5	6	7	8	9	10
预测结果	P	P	P	P	P	P	N	N	N	N
实际	P	N	P	P	N	P	N	N	N	P

$$TP = 4 \quad FN = 1$$
$$FP = 2 \quad TN = 3$$

混淆矩阵即为：

$$\begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$$

2. ROC 曲线即“受试者工作特征曲线”，根据学习器的预测结果对样例进行排序，按照顺序逐个把样本作为正例进行预测，每次计算出两个重要量的值，分别以它们为横纵坐标绘图就得到了 ROC 曲线，ROC 曲线的纵轴为 TPR（真正例率），横轴为 FPR（假正例率），其计算公式为：

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

利用 python 程序绘制 ROC 曲线如下：

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df=pd.DataFrame([[0.95,1],[0.85,0],[0.78,1],[0.66,1],[0.6,0],[0.55,1],
[0.53,0],[0.52,0],[0.51,0],[0.4,1]])
tpr=[]
fpr=[]
for a in np.arange(0,1,0.01):
    df[2]=df[0].map(lambda x: 1 if x>a else 0)
    tpr.append(np.where((df[2]==1) & (df[1]==1))[0].shape[0]/(np.where
        ((df[2]==1) & (df[1]==1))[0].shape[0]+np.where((df[1]==1) & (df
        [2]==0))[0].shape[0]))
    fpr.append(np.where((df[1]==0)&(df[2]==1))[0].shape[0]/(np.where((
        df[1]==0)&(df[2]==1))[0].shape[0]+np.where((df[2]==0) & (df
        [1]==0))[0].shape[0]))

plt.figure(dpi=250)
```

```
plt.plot(fpr,tpr,'*-')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.title('ROC curve')
plt.legend(['ROC'])
plt.show()
```

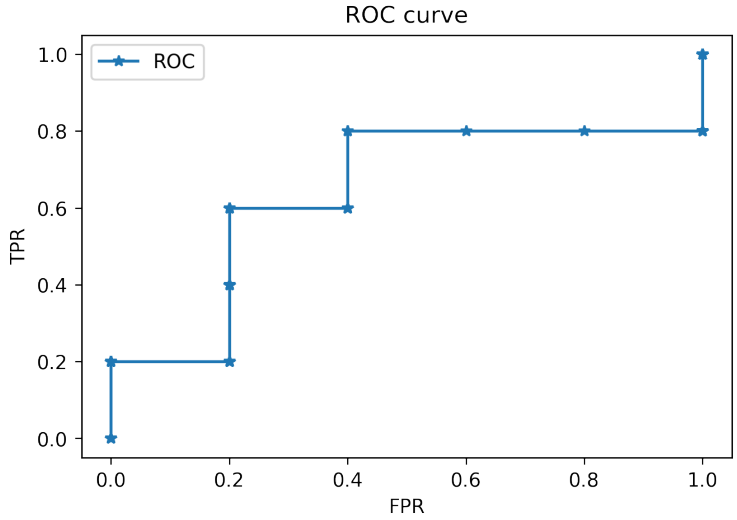

图 10: ROC 曲线

Machine Learning **Decision Tree-2022 年 11 月 13 日**

决策树

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 引入

决策树是一种常见的分类模型，通过作出一系列决策来对数据进行划分，这类似于针对一系列问题进行选择。从根节点开始，测试待分类项中对应的特征属性，并按照其值选择输出分支，直到叶子节点，将叶子节点的存放的类别作为决策结果。是一种判别模型。

决策树的生成算法是一个递归的过程，1. 当前样本集合全部属于同一类别，2. 当前属性集为空，3. 当前节点包含的样本集为空。以上三种情况会跳出递归的树生成算法。

2 划分选择

决策树生成算法中最关键的一步就是“从属性集中选择划分属性 a^*”。如何选择最优的属性成为决策树算法的核心。根据划分选择属性指标的不同，可以将决策树划分选择算法分为三类：

1. ID3

   (a) 信息熵：信息熵是度量样本集合纯度的最常用的一种指标，假定当前样本集合 D 中第 k 类别所占的比例为 $p_k(k = 1, \ldots, |\mathcal{Y}|)$，则 D 的信息熵可以定义为：

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

   若信息熵 $Ent(D)$ 的值越小，则 D 的纯度越高。

   (b) 信息增益：假设离散属性 a 有 $|V|$ 个可能的取值集合 $\{a^1, \ldots, a^{|V|}\}$，对于单独属性 a 的信息增益 (information gain)。

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

   一般来说信息增益越大，则意味着使用属性 a 来进行划分所获得的“纯度提升”越大，因此我们可以使用**信息增益**进行属性划分。即选择 $a^* = \underset{a \in \mathcal{A}}{arg\max}\ Gain(D, a)$。著名的 ID3 决策树学习算法就是以信息增益来选择划分属性。

2. CD4.5

(a) 信息增益准则对可取值数目较多的属性有所偏好，比如如果数据集中有“id”，那么信息增益准则可能会偏好以“id”为划分属性进行划分，从而将决策树分为 N 个分支。然而从实际意义出发，以 id 为划分准则是没有意义的。

为了减少信息增益准则对可取值数目较多的属性有所偏好可能带来的不利影响。著名的 C4.5 算法不使用信息增益而是使用增益率来选择划分属性。

$$Gain_ratio(D,a) = \frac{Gain(D,a)}{IV(a)}$$

其中

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

称为属性 a 的“固有值”，属性 a 的可能取值数目越多 (即 V 越大)，则 $|IV(a)|$。CD4.5 算法选择增益率最大的属性进行划分，也即 $a^* = \underset{a \in \mathcal{A}}{arg\max}\, Gain\_index(D,a)$

3. CART

CART 决策树使用“基尼指数”，作为划分属性集的指标。

$$Gini(D) = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2$$

直观来说，Gini 反映了从数据集中随机抽取两个样本，其类别标记不一致的概率，因此当 Gini 越小，数据集 D 的纯度越高。CART 算法采用 Gini_index 作为属性划分准则。

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

每次分支时选择划分后基尼指数最小的属性作为最优划分属性:即 $a^* = \underset{a \in \mathcal{A}}{arg\min}\, Gini\_index(D,a)$

3 剪枝操作

研究表明，不同属性划分算法会对决策树的尺寸有较大影响，但对泛化性能的影响却十分有限。在决策树分支（学习）过程中，为了尽可能正确分类训练样本，节点划分过程将不断重复，有时可能会出现将数据的所有特征都学的“太好”的情况，我们可以采用剪枝的方法主动去掉一些分支，增加模型的泛化性。同时也是对付“过拟合”的主要手段。

剪枝操作主要可以分为预剪枝和后剪枝。

- 预剪枝是指在分支生成前进行估计，若当前节点的划分不能带来决策树泛化性能的提升，则停止分支
- 后剪枝是指在决策树划分完成之后，从叶子节点开始，自底向上地对非叶子节点进行考察，若该节点不分支能带来模型泛化性能的提升则不分枝

预剪枝和后剪枝两者的对比：

表 3: 预剪枝与后剪枝对比

分支方法	预剪枝	后剪枝
时间开销	训练时间开销降低 测试时间开销降低	训练时间开销增加 测试时间开销降低
欠/过拟合	过拟合风险降低 欠拟合风险增加	过拟合风险降低 欠拟合风险基本不变
泛化性能	通常低于后剪枝	通常优于预剪枝

代码实现决策树分类算法

以 2022 年全国大学生数学建模 C 题为例，我在比赛中提出使用决策树算法来区分铅钡玻璃与高钾玻璃。

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split


data = pd.read_excel('data/data.xlsx', sheet_name=1).fillna(0).values
data_x = data[..., 2:15]
data_y = data[..., 16]
train_x, test_x, train_y, test_y = train_test_split(data_x, data_y,
    test_size=0.3)

clf = tree.DecisionTreeClassifier()
clsf = clf.fit(train_x, train_y)
result = clsf.score(test_x, test_y)
```

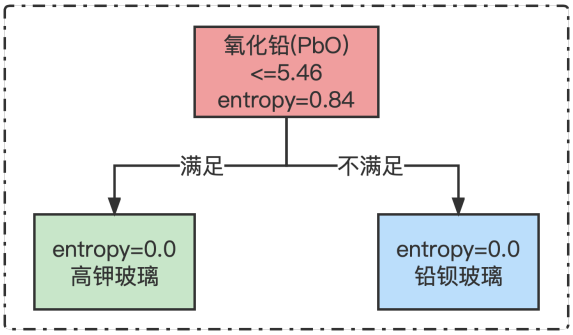

图 11: 算法可视化图

又以我做过的一道决策树题目为例，手算 CART 分支准则的决策树：一个用于决策树分类模型训练的数据集如下：

Tid	属性A	属性B	类标号
1	Y	N	1
2	Y	Y	1
3	Y	Y	1
4	Y	N	0
5	Y	Y	1
6	N	N	0
7	N	N	0
8	N	N	0
9	Y	Y	0
10	Y	N	0

图 12: 决策树数据集

用决策树分类算法建模时需要通过选定的属性来划分对象到不同的子节点。试分别计算划分前、按照属性 A 和按照 B 划分时的节点 GINI 值，并判断决策树分类算法将会在 A 和 B 中先选择哪个属性作为划分条件。

解：数据集的纯度可以用 $gini-value$ 来度量。对于数据集 D 有：

$$Gini\,(D)=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2$$

其中 p_k 表示样本集合中第 k 类样本所占的比例 $p_k(k=1,\ldots,|\mathcal{Y}|)$。假设离散属性 a 有 $|V|$ 个可能的取值集合 $\{a^1,\ldots,a^{|V|}\}$，对于单独属性 a 的 $Gini-index$，定义为：

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

在本样例中：

$$\begin{aligned}Gini\_index(D,\text{属性 A})&=\frac{7}{10}Gini(\text{属性 A}=Y)+\frac{3}{10}Gini(\text{属性 A}=N)\\&=\frac{7}{10}\left(1-(\frac{4}{7})^2-(\frac{3}{7})^2\right)+\frac{3}{10}\left(1-1^2\right)\\&=0.3429\\Gini\_index(D,\text{属性 B})&=\frac{4}{10}Gini(\text{属性 B}=Y)+\frac{6}{10}Gini(\text{属性 B}=N)\\&=\frac{4}{10}\left(1-(\frac{3}{4})^2-(\frac{1}{4})^2\right)+\frac{6}{10}\left(1-(\frac{1}{6})^2-(\frac{5}{6})^2\right)\\&=0.3167\end{aligned}$$

故决策树 CART 算法会首先选择特征使得：

$$a^*=\underset{a\in\mathcal{A}}{arg\,\min}\;Gini\_index(D,a)$$

故会选择属性 B 进行分支。

Machine Learning **Ensemble Learning-2022 年 11 月 13 日**

集成学习

lecturer：*Kaidong Wang* *Scribe: Jinyu Zhang*

1 引入

如果你有 T 个朋友，他们对于股票涨停的预测表现为 $g_1(x), g_2(x), \ldots, g_T(x)$，常见的决策方法有：

- 从他们以前的预测（经验预测）结果中，选择一个你最值得信赖的朋友：

$$G(x) = g_t(x)，其中\ t = \underset{t\in 1,\ldots,T}{arg\min}\ E_{val}(g_t)，即经验预测分类误差最小的朋友。$$

- 将所有朋友的预测取平均值（回归）/令每位朋友进行投票表决（分类）

$$回归问题：G(x) = \frac{1}{T}\sum_t g_t(x)$$
$$分类问题：G(x) = sign(\sum_t g_t(x))$$

- 将所有朋友的预测取加权平均值（回归）/令每位朋友进行不均等投票（分类）

$$回归问题：G(x) = \frac{1}{T}\sum_t w_t g_t(x)$$
$$分类问题：G(x) = sign(\sum_t w_t g_t(x))$$

- 根据朋友的预测结果进行权重的动态调整进行赋权

$$回归问题：G(x) = \frac{1}{T}\sum_t w_t(x) g_t(x)$$
$$分类问题：G(x) = sign(\sum_t w_t(x) g_t(x))$$

为什么多个朋友预测的结果就会比单独一个朋友预测的结果要好呢？或者说为什么集成学习的学习效果通常会优于单独学习器的学习效果呢？有一个重要的前提假定是，各个个体学习其应该“好而不同”，即个体学习器要有一定的“准确性”，也要有“多样性”。

考虑一个二分类任务，输出标签为 $y \in \{-1, 1\}$ 和真实函数 f，假设基分类器的错误率为 ε，对于每个基分类器 h_i。

$$P(h_i(x) \neq f(x)) = \varepsilon$$

集成学习的错误率为：

$$P(H(x) \neq f(x)) = \sum_{k=0}^{[T/2]} C_T^k (1-\varepsilon)^k \varepsilon^{T-k}$$
$$由\ \text{Hoeffding}\ 不等式：\ \leq exp\left(-\frac{1}{2}T(1-2\varepsilon)^2\right)$$

以上分析有一个关键假设，就是基学习器的误差相互独立。然而在现实任务中，基学习器是同一个问题训练出来的，他们显然不可能相互独立！按照目前集成学习的算法主要可以分为两大类，一类是个体学

习器之间存在强依赖关系，必须串行生成的序列化方法 Boosting，一种是个体学习器之间不存在强依赖关系，可以同时生成的并行化方法 Bagging/Blending。

Bagging

从集成学习的理论分析中我们可以得知，要想得到学习效果好的学习器，通常需要保证基学习器有效的同时且多元化（Diversity）。而 Bagging 就是这样一种，利用数据集抽样的随机性实现模型的多样性。

Bootstrap 采样

给定一个训练数据集，一种可能的做法是对训练样本进行采样，产生出若干个不同的子集，再从每个数据子集中训练出一个基学习器。这样，由于训练数据不同，我们获得的基学习器可望具有比较大的差异。然而，为获得好的集成，我们同时还希望个体学习器不能太差。如果采样出的每个子集都完全不同，则每个基学习器只用到了一小部分训练数据，甚至不足以进行有效学习，这显然无法确保产生出比较好的基学习器。为解决这个问题，我们可考虑使用相互有交叠的采样子集。

在统计学中，自助法（Bootstrap Method，Bootstrapping 或自助抽样法）是一种从给定训练集中有放回的均匀抽样，也就是说，每当选中一个样本，它等可能地被再次选中并被再次添加到训练集中。对于小数据集，自助法效果很好。

假设给定的数据集包含 d 个样本。该数据集有放回地抽样 m 次，产生 m 个样本的训练集。这样原数据样本中的某些样本很可能在该样本集中出现多次。没有进入该训练集的样本最终形成检验集（测试集）。显然每个样本被选中的概率是 $1/m$，因此未被选中的概率就是 $(1-1/m)$，这样一个样本在训练集中没出现的概率就是 m 次都未被选中的概率，即 $(1-1/m)^m$。当 m 趋于无穷大时，这一概率就将趋近于 $\frac{1}{e}=0.368$，所以留在训练集中的样本大概就占原来数据集的 63.2%。

Blending

Bagging 算法的名称其实就是Bootstrap aggregating的缩写。Bagging 是并行式集成学习算法中最著名的代表。从名字可以看出它使用的是自助采样法进行抽样。如果抽样进行 T 次，则可以得到 T 个含 m 个训练样本的采样集，然后基于每一个训练样本的采样集进行训练得到 T 个基学习器，再通过 Blending 方法进行训练。这就是 Bagging 的流程。

Blending 学习方法可以按照基学习器的权重分为 **Uniform Blending、Linear Blending、Any Blending (Stacking)**。

- Uniform Blending

$$\text{回归问题：} G(x)=\frac{1}{T}\sum_t g_t(x)$$
$$\text{分类问题：} G(x)=sign(\sum_t g_t(x))$$

理论分析：对一个固定的 x，设最优基学习器为 $f(x)$:

$$\begin{aligned} avg(g_t(x) - f(x))^2 &= avg\left(g_t^2 - 2g_t^2 f + f^2\right) \\ &= avg(g_t^2) - 2Gf + f^2 \\ &= avg(g_t^2) - G^2 + (G - f)^2 \\ &= avg(g_t^2 - 2g_t G + G^2) + (G - f)^2 \\ &= avg\left((g_t - G)^2\right) + (G - f)^2 \\ avg(E_{out}(g_t)) &= avg(g_t - G)^2 + E_{out}(G) \geq E_{out}(G) \end{aligned}$$

可知集成学习后的学习器 G 的期望误差要小于任意一次抽样的基学习器 g_t。

- Linear Blending
  将已有的 T 个基学习器 $g_1, \ldots, g_T$ 进行不一致投票，或者线性加权平均。

$$\begin{aligned} &\text{回归问题：} G(x) = \tfrac{1}{T}\textstyle\sum_t w_t g_t(x) \\ &\text{分类问题：} G(x) = sign(\textstyle\sum_t w_t g_t(x)) \end{aligned}$$

- Any Blending
  将已有的 T 个基学习器进行动态调整赋权

$$\begin{aligned} &\text{回归问题：} G(x) = \tfrac{1}{T}\textstyle\sum_t w_t(x) g_t(x) \\ &\text{分类问题：} G(x) = sign(\textstyle\sum_t w_t(x) g_t(x)) \end{aligned}$$

Bagging 方法中最具代表性的集成学习方法为随机森林。

随机森林

利用 Bootstrap 抽样法，从原始数据集中生成 k 个数据集，并且每个数据集都含有 N 个观测和 P 个自变量。针对每一个数据集，构造一棵 CART 决策树，在构建子树的过程中，并没有将所有自变量用作节点字段的选择，而是随机选 p 个字段。让每一棵决策树尽可能地充分生长，使得树中的每个节点尽可能“纯净”，即随机森林中的每一棵子树都不需要剪枝。针对 K 棵 CART 树的随机森林，对分类问题利用投票法，将最高得票的类别用于最终的判断结果; 对回归问题利用均值法，将其用作预测样本的最终结果。

Boosting

本部分重点讲了 AdaBoost 算法，梯度 Boosting 算法。

AdaBoost

从上面的 Bagging 算法中我们知道，集成学习的基学习器应该做到“好而不同”。Bagging 方法主要是通过 Bootstrap 抽样得到不同的子数据进行训练模型，从而得到不同的基学习器。

有没有可能通过某种加权方式进一步提升多样性？于是我们引入 AdaBoost 算法。

- 如何保证训练的基学习器不要太差？（准确性）
  设在第 t 步，每个训练数据 (x_i, y_i) 的权重为 $w_i^{(t)}$。

$$f_t = \underset{f \in \mathcal{H}}{arg\min} \sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}$$

- 分类器不要太像（多样性）
  在第 $t+1$ 步，f_t 在权重 v^{t+1} 下的性能近似为随机猜测：

$$\text{分错样本的权重和：} \sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}$$

$$\text{分对样本的权重和：} \sum_{i=1}^{N} w_i^{(t)} I\{y_i = f(x_i)\}$$

$$\text{也即：} \frac{\sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}}{\sum_{i=1}^{N} w_i^{(t)}} = \frac{\sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}}{\sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\} + \sum_{i=1}^{N} w_i^{(t)} I\{y_i = f(x_i)\}} = \varepsilon_t$$

我们将注意力多放在错误分类的样本上，定义 $s_t = \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$。则第 i 个样本的下一次学习的权重为：

$$v_i^{t+1} = \begin{cases} v_n^t / s_t, \ if \ y_i = f_t(x_i) \\ v_n^t \times s_t, \ if \ y_i \neq f_t(x_i) \end{cases}$$

其含义是，对于分错的样本我们增大权重，增大模型的注意力。

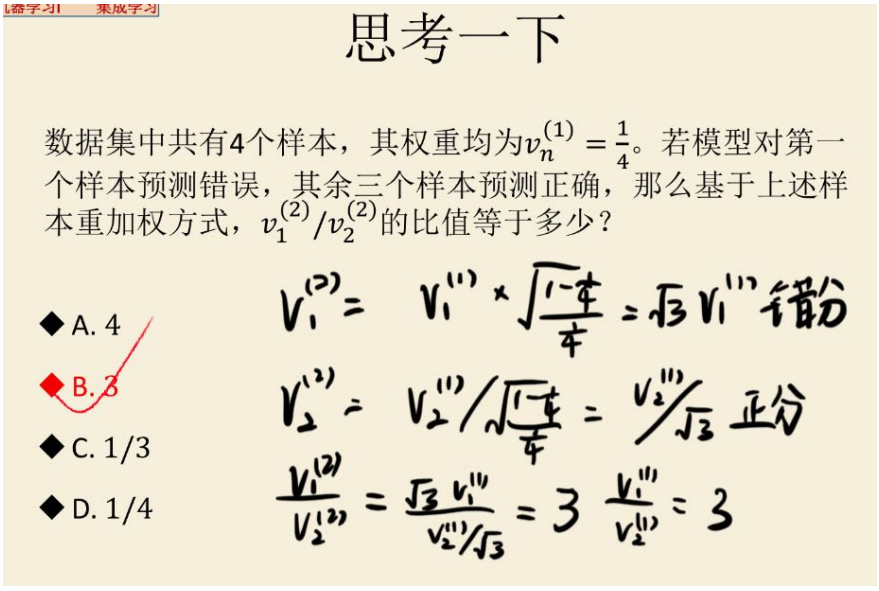

图 13: adaboost 例题

- 如何集成？

对于好的学习器我们进行强化，对于效果差的学习器进行弱化：

$$\alpha_t = ln(s_t) \quad = ln\left(\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}\right)$$

对于好的学习器，当 ε_t 较小，从而 α 较大。对于差的学习器，当 ε_t 较大，从而 α 较小。$\varepsilon_t = 1/2$ 时，则基学习器相当于随即猜测，正分和误分的概率相同，学习器权重 $\alpha = 0$。$\varepsilon_t = 0$ 时，即分类正确率 100%，此时学习器权重 $\to \infty$。AdaBoost 集成学习器的形式为：

$$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

AdaBoost 算法的流程如下：

Algorithm 6 AdaBoost

输入：样本数据 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$；弱分类器个数 T，函数空间 $\mathcal{H}$(算法 $\mathcal{A}$)。

1. 初始化样本权重 $v^0 = (\frac{1}{N}, \ldots, \frac{1}{N})$
2. for $t = 1, 2, \ldots, T$
3. $\quad f_t = \underset{f \in \mathcal{H}}{arg\min} \sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}$
4. $\quad$ 计算 $\varepsilon_t = \frac{\sum_{i=1}^{N} w_i^{(t)} I\{y_i \neq f(x_i)\}}{\sum_{i=1}^{N} w_i^{(t)}}$ 和 $s_t = \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}$
5. $\quad$ 计算 $\alpha_t = ln(s_t)$
6. $\quad v_i^{t+1} = \begin{cases} v_n^t / s_t, \ if \ y_i = f_t(x_i) \\ v_n^t \times s_t, \ if \ y_i \neq f_t(x_i) \end{cases}$

输出：$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$

- AdaBoost 的损失函数：
  adaboost 对于误分类点权重进行放大,对于正确分类点权重进行缩小。其定义其 $margin = y_i \sum_{i=1}^{T} \alpha_t f_t(x_i)$，如果 $margin > 0$ 为正分类点，如果 $margin < 0$ 为误分类点。AdaBoost 可以视为放大误分类点的 $loss$，可以考虑指数损失函数：如果记 $F_t(x) = \sum_{t=1}^{T} = F_{t-1}(x) + \alpha_t f_t(x)$

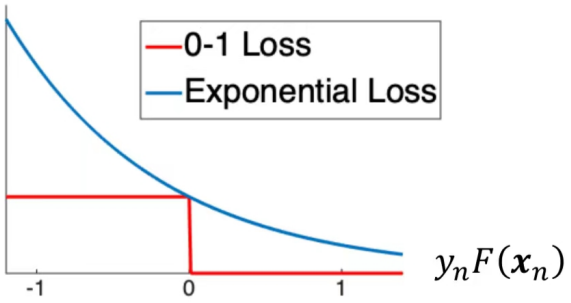

图 14: 指数损失函数

就像是一个以 $f_t(x)$ 为方向的梯度提升算法。于是我们引入梯度 Boosting 算法。

梯度 Boosting 算法

考虑一个常见的优化问题：

$$\text{优化目标：} \min_{\theta} f(\theta)$$

$$\text{优化过程：} \theta_t = \theta_{t-1} - \alpha_t d$$

而对于上述的 AdaBoost 算法来说：

$$\text{优化目标：} \min_{F} \sum_{n=1}^{N} exp(-yF(x_n))$$

$$\text{优化过程：} F_t = F_{t-1} + \alpha_t f_t\text{，可以将} -f_t\text{看作下降方向！}$$

梯度 Boosting 算法的思路与梯度下降法类似，都分为确定提升（下降）方向（f_t）和确定提升步长（α_t）两步。

1. 确定提升方向 f_t。

$$-\left[\frac{dl(F(x_n), y_n)}{dF(x_n)}\right]_{F(x)=F_{t-1}(x)} = r_{nt}$$

$$f_t = \underset{f}{argmin} \sum_{n=1}^{N} \frac{1}{2}\|f(x_n) - r_{nt}\|^2$$

2. 确定提升步长 α_t。

$$\alpha_t = \underset{\alpha}{argmin} \sum_{n=1}^{N} \lessdot (F_{t-1} + \alpha f_t, y_i)$$

故梯度 Boosting 算法的伪代码如下：

Algorithm 7 AdaBoost

输入：样本数据 $\{\boldsymbol{x}, y\}_{i=1}^{N}$，其中 $\boldsymbol{x} \in \mathcal{R}^n$，$y \in \mathcal{R}$，其中 $y \in \{1, \ldots, K\}$；弱分类器个数 T，函数空间 $\mathcal{H}$(算法 $\mathcal{A}$)，初始弱分类器 f_0。

1. 初始化点 $v^0 = (\frac{1}{N}, \ldots, \frac{1}{N})$
2. for $t = 1, 2, \ldots, T$
3. $\quad r_{nt} = -\left[\frac{dl(F(x_n), y_n)}{dF(x_n)}\right]_{F(x)=F_{t-1}(x)}$
4. $\quad f_t = \underset{f}{argmin} \sum_{n=1}^{N} \frac{1}{2}\|f(x_n) - r_{nt}\|^2$
5. $\quad \alpha_t = \underset{\alpha}{argmin} \sum_{n=1}^{N} \lessdot (F_{t-1} + \alpha f_t, y_i)$

输出：$F_t(x) = F_{t-1}(x) + \alpha_t f_t(x)$

Machine Learning **Support Vector Machine-2022 年 11 月 13 日**

支持向量机

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 线性支撑向量机（硬间隔支持向量机）

对于一个二分类问题：$\{x_n, y_n\}_{n=1}^N$，$x_n \in \mathcal{R}^n$，$y \in \{-1, 1\}$。假设数据集线性可分，硬间隔支撑向量机的目标就是寻找一个超平面，使得所有分类点被正确分类的前提下，保证“间隔”最大。

什么是“间隔”（margin）

样本空间中的任意一点 (x_i, y_i) 到超平面 (w, b) 的距离可以写为：

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

假设该超平面真的能够将所有的样本点正确分类，则有：

$$y_i \left(w^\top x_i + b\right) \geq 0,\ \forall (x_i, y_i) \in D$$

距离超平面最近的几个训练样本被称为“支撑向量”。而两个异类支撑向量到超平面的距离之和称为间隔，也即：

$$\min_{(x_i, y_i) \in D} \frac{2 \times y_i (w^\top x_i + b)}{|w|}$$

不妨定义间隔为 γ，也即：$\gamma = \min\limits_{(x_i, y_i) \in D} \frac{2 \times y_i (w^\top x_i + b)}{|w|}$，所以有：

$$\frac{2 \times y_i (w^\top x_i + b)}{|w|} \geq \gamma$$
$$2 \times y_i (w^\top x_i + b) \geq \|w\| \gamma$$

总存在 (w, b) 的放缩变换使得：$2 \times y_i (w^\top x_i + b) \geq \|w\| \gamma = 1$ 成立。故间隔可以定义为 $\gamma = \frac{1}{\|w\|}$。

所以找到最大间隔的支撑超平面也就是寻找，满足：

$$\min_{w,b} \frac{1}{\|w\|}$$
$$s.t.\ y_i \left(w^\top x_i + b\right) \geq 1, \forall i = 1, 2, \ldots, m$$

优化问题等价于一个二次规划问题 QP:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$s.t.\ y_i\left(w^\top x_i + b\right) \geq 1, \forall i = 1, 2, \ldots, m$$

这就是线性支撑向量机模型。

理解支撑向量机：由硬间隔支撑向量机的形式可以看出，分离超平面的确定可能仅与部分离超平面最近的样本点决定（这些距离分离超平面最近的样本点也叫“支持向量（support vector）”）。其他样本点的微小改变不影响分类面。

2 软间隔支撑向量机

对于非线性可分的数据集，无法再通过硬间隔支撑向量机的方法找到一个分离超平面将两类数据完全分开了，于是我们的优化目标就变为，再最大化间隔的同时，不满足约束的样本尽可能少。于是优化目标可以修改为：

$$\min_{w,b} \frac{1}{2}w^2 + C\sum_{i=1}^{N} l_{0/1}\left(y_i(w^\top x_i + b) - 1\right)$$

其中 $l_{0/1}$ 为“0/1 损失函数”，$C > 0$。然而 $l_{0/1}$ 具有非凸、非连续的数学性质，使得此优化问题不易直接求解，可以用一些其他函数来代替 $l_{0/1}$，在软间隔支持向量机中最长使用的替代损失函数为 hinge 损失：

$$l_{hinge}(z) = max(0, 1 - z)$$

则软间隔优化问题可改写为：

$$\min_{w,b} \frac{1}{2}w^2 + C\sum_{i=1}^{N} max\left(0, 1 - y_i(w^\top x_i + b)\right)$$

引入松弛变量 $\xi_i \geq 0$ 优化问题可改写为：

$$\min_{w,b,\xi} \frac{1}{2}w^2 + C\sum_{i=1}^{N} \xi_i$$
$$s.t.\ y_i(w^\top x_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

即为一个软间隔支持向量机，可知其仍然是一个 QP 问题，此时的支持向量为间隔边界与间隔中的所有样本点。$\{x_i | y_i(w^\top x_i + b) = 1 - \xi_i\}$（由于 ξ_i 在一定范围有随机性，该集合实际上表示硬间隔之间的所有样本点，都算支撑向量）。

3 核方法

当数据集线性不可分，但是能非线性可分时。可对数据集的特征进行高维映射，使得映射变换后的数据集能够在高纬空间线性可分，则可以转化为硬间隔支撑向量机进行求解。

$$\begin{aligned}&\min_{w,b,} \frac{1}{2}w^2\\ &s.t.\ y_i(w^\top\phi(x_i)+b)\geq 1-\xi_i, \forall i=1,\ldots,N\end{aligned}$$

其中 $\phi(x_i):\mathcal{R}^n \longrightarrow \mathcal{R}^D(D>n)$ 是一种高维映射。

4 支持向量机的对偶问题

软间隔支持向量机

在**常象宇老师**的最优化课程作业中，曾推到过软间隔支持向量机的对偶形式，这里直接引用：

对于软间隔支持向量机问题

$$\begin{aligned}&\min_{\boldsymbol{w},\boldsymbol{b},\xi} \frac{\|\boldsymbol{w}\|^2}{2}+C\sum_{i=1}^{n}\xi_i\\ &s.t.\ y_i\left(\langle \boldsymbol{w},\boldsymbol{x}_i\rangle+b\right)\geq 1-\xi_i,\\ &\xi_i\geq 0, i=1,...,n,\end{aligned}$$

写出 *Lagrange* 函数如下：

$$L(\lambda,\mu)=\frac{\|w\|^2}{2}+C\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\lambda_i\left\{\left[y_i\left(<\boldsymbol{w},\boldsymbol{x_i}>+\boldsymbol{b}\right)\right]+\xi_i-1\right\}-\sum_{i=1}^{n}\mu_i\xi_i \tag{5}$$

目标函数 $\frac{\|\boldsymbol{w}\|^2}{2}+C\sum_{i=1}^{n}\xi_i$ 为凸函数，约束函数 $s.t.\ y_i\left(\langle \boldsymbol{w},\boldsymbol{x}_i\rangle+b\right)\geq 1-\xi_i$ 与 $\xi_i\geq 0, i=1,...,n,$ 均为线性仿射函数，该凸优化问题满足 Slater 条件，也满足 KKT 条件。

Lagrange 对偶函数满足：

$$d=\inf_{\boldsymbol{w},\boldsymbol{b},\xi} L(\lambda,\mu) \tag{6}$$

由 KKT 条件：

$$\begin{cases}\nabla_{\boldsymbol{w}}L=\boldsymbol{w}-\sum_{i=1}^{n}\lambda_i y_i x_i=0\\ \nabla_{\boldsymbol{b}}L=-\sum_{i=1}^{n}\lambda_i y_i=0\\ \nabla_{\xi}L=C\mathbb{1}-\boldsymbol{\lambda}-\boldsymbol{\mu}=0\end{cases}$$

其中 $\mathbb{1}=(1,\ldots,1)^T$，$\boldsymbol{\lambda}=(\lambda_1,\ldots,\lambda_n)^T$，$\boldsymbol{\mu}=(\mu_1,\ldots,\mu_n)^T$。将驻点条件带入式6可得：

$$\begin{aligned} d=&\inf_{\boldsymbol{w},\boldsymbol{b},\xi}\ \frac{\|w\|^2}{2}+C\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\lambda_i\left\{\left[y_i\left(<\boldsymbol{w},\boldsymbol{x_i}>+\boldsymbol{b}\right)\right]+\xi_i-1\right\}-\sum_{i=1}^{n}\mu_i\xi_i \\ =&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j}-\sum_{i=1}^{n}\lambda_i y_i\boldsymbol{x_i}\sum_{j=1}^{n}\lambda_j y_j\boldsymbol{x_j}+\sum_{i=1}^{n}\lambda_i \\ =&-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j}+\sum_{i=1}^{n}\lambda_i \end{aligned} \tag{7}$$

所以，写出 *Lagrange* 对偶问题如下：

$$\begin{aligned} &\min_{\boldsymbol{\lambda}}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j}-\sum_{i=1}^{n}\lambda_i \\ s.t.\ &\sum_{i=1}^{n}\lambda_i y_i=0 \\ &0\le\lambda_i\le C\qquad i=1,\ldots,n \end{aligned} \tag{8}$$

硬间隔支持向量机

对于硬间隔支持向量机，同理：

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{b},\xi}\frac{\|\boldsymbol{w}\|^2}{2} \\ s.t.\ &y_i\left(\langle\boldsymbol{w},\boldsymbol{x}_i\rangle+b\right)\ge 1, \end{aligned}$$

其 Lagrange 对偶问题为：

$$\begin{aligned} &\min_{\boldsymbol{\lambda}}\ \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j}-\sum_{i=1}^{n}\lambda_i \\ s.t.\ &\sum_{i=1}^{n}\lambda_i y_i=0 \\ &\lambda_i\ge 0\qquad i=1,\ldots,n \end{aligned} \tag{9}$$

为什么要写出对偶问题？

根据化成的对偶问题我们可以发现，该对偶问题仍然是二次规划问题，根据该问题的本身特性，有一种很高效的方法可以求解该问题，**SMO** 方法，可以高效求解 SVM 问题。

核化支持向量机

对于对偶问题，如果引入核方法对数据集进行高维映射，可以将原本线性不可分的数据化为线性可分的数据：

$$\begin{aligned} &\min_{\boldsymbol{\lambda}} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j \boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) - \sum_{i=1}^{n}\lambda_i \\ s.t.\ &\sum_{i=1}^{n}\lambda_i y_i = 0 \\ &\lambda_i \geq 0 \qquad i = 1, \ldots, n \end{aligned} \tag{10}$$

什么样的函数可以作为核函数呢？

Theorem 4 对于任意数据集，$D = \{x_i, y_i\}_{i=1}^{N}$，$\begin{bmatrix} K(x_1,x_1) & \ldots & K(x_1,x_N) \\ \vdots & \ddots & \vdots \\ K(x_N,x_1) & \ldots & K(x_N,x_N) \end{bmatrix}$ 为半正定矩阵，则 $K()$ 为核函数。常见的核函数有：

- 线性核：$K(x, x') = x^\top x'$
- 高斯核：$K(x, x') = exp(-\frac{\|x-x'\|^2}{2\sigma^2})$
- *Laplace* 核：$K(x, x') = exp(-\frac{\|x-x'\|}{\sigma})$
- 多项式核：$K(x, x') = (x^\top x')^d$
- *Sigmoid* 核：$K(x, x') = tanh(\beta x^\top x' + \theta)$

Machine Learning **KNN、Naive Bayes & K-Means-2022 年 11 月 13 日**

最近邻算法、朴素贝叶斯与 K-Means 聚类

lecturer: *Kaidong Wang* *Scribe: Jinyu Zhang*

1 KNN 算法

KNN 算法是一种有监督学习算法，该算法可以用于分类与回归问题。

对于分类问题：如果一个样本点在特征空间中与另外 k 个样本点距离较近（与其 k 个最邻近的样本点），那么这 k 个样本点中最多的类别即为该未知样本点的类别。该分类思路类似于投票法（voting）。

对于回归问题，对于新的样本，根据其最近邻的 k 个训练样本标签值的均值作为预测值。

距离的度量

KNN 中的“距离”并非我们常识中的距离，我们最常见的距离度量方式为欧式距离 $d = \sqrt{\sum(x_i - y_i)^2}$，除此之外还有：

- 欧氏距离：$d(x_i, x_j) = \sqrt{\sum(x_i - x_j)^2}$
- 曼哈顿距离：$d(x_i, x_j) = |x_i - x_j|$
- 切比雪夫距离：$d(x_i, x_j) = \max\limits_i |x_i - x_j|$
- 闵可夫斯基距离：$d(x_i, x_j) = (\sum_{ij} |x_i - x_j|^p)^{\frac{1}{p}}$

2 朴素贝叶斯

贝叶斯决策论

贝叶斯决策论是概率框架下实施决策的基本方法。考虑多分类问题，假设有 N 中可能的类别标记，$\mathcal{Y} = \{c_1, \ldots, c_N\}$，$\lambda_{ij}$ 是将一个真实标记为 c_j 的样本误分类为 c_i 所产生的损失，基于后验概率（已知样本，其类别的概率）可得将样本分类为 c_i 所产生的期望损失，即在样本 x 上的条件风险：

$$R(c_i|x) = \sum_{j=1}^{N} \lambda_{ij} P(c_j|x)$$

我们的任务是寻找一个判定准则 $h : \mathcal{X} \longrightarrow \mathcal{Y}$ 使得总体风险最小化：

$$R(h) = \mathbb{E}_x \left[R\left(h(x)|x\right) \right]$$

贝叶斯判定准则也即: 为了最小化总体风险，只需在每个样本上选择那个能使条件风险 $R(c|x)$ 最小的类别标记，即：

$$h^*(x) = arg\min_{c\in\mathcal{Y}} R(c|x)$$

然而后验概率 $P(c|x)$ 在现实中难以直接获得，从这个角度看，机器学习所要实现的是基于有限的训练样本尽可能准确地估计出后验概率。从而衍生出两种学习方法：

1. 判别式学习
   直接对 $P(c|x)$ 建模，即给定样本点数据，通过算法直接求解其类别概率，比如 logit 回归，svm，pla，决策树等都是判别式模型

2. 生成式学习
   不通过直接对后验概率 $P(c|x)$ 建模的方式，而是通过对联合概率分布建模，并通过贝叶斯公式得到后验概率，代表就是本节学的朴素贝叶斯分类器。

而根据贝叶斯定理来看，生成式学习的最大困难是估计似然。我们可以采用极大似然估计的方法：

朴素贝叶斯之所以“朴素”，是因为它有一条非常重要的假设，这条假设虽然很强，但是对于一些问题的结果却非常好。那就是在给定类别的条件下，各特征的条件概率相互独立。

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

$$h(x) = arg\max_{c\in\mathcal{Y}} P(c)\prod_{i=1}^{d} P(x_i|c)$$

以我做过的一道题目为例：

Example 3 给定一个数据集和一个测试对象：$X = (Refund = No, Married, Male)$，试用朴素贝叶斯方法预测对象 X 的类标号。

Tid	Refund	Marital Status	Sex	Evade
1	Yes	Single	Male	No
2	No	Married	Male	No
3	No	Single	Female	No
4	Yes	Married	Male	No
5	No	Divorced	Male	Yes
6	No	Married	Female	No
7	Yes	Divorced	Male	No
8	No	Single	Male	Yes
9	No	Married	Female	No
10	No	Single	Male	Yes

图 15: 样例三

解：根据样本数据构建一个朴素贝叶斯分类器，首先计算类先验概率：

$$P(Evade = Yes) = \frac{3}{10}$$
$$P(Evade = No) = \frac{7}{10}$$

其次计算样本 $X = (Refund = No, Married, Male)$ 各个属性下的条件概率 $P(a|Evade)$。

$$P(Refund = No|Evade = Yes) = 1$$
$$P(Refund = No|Evade = No) = \frac{4}{7}$$
$$P(Marital\ Status = Married|Evade = Yes) = 0$$
$$P(Marital\ Status = Married|Evade = No) = \frac{4}{7}$$
$$P(Sex = Male|Evade = Yes) = 1$$
$$P(Sex = Male|Evade = No) = \frac{4}{7}$$

故构建朴素贝叶斯分类器：

$$\begin{aligned} Evade^* &= arg\max\ P(Evade|Refund = No, Married, Male) \\ &= arg\max\ \frac{P(Evade)P(Refund = No, Married, Male|Evade)}{P(Refund = No, Married, Male)} \\ &= arg\max\ P(Evade)P(Refund = No|Evade)P(Married|Evade)P(Male|Evade) \end{aligned}$$

经过计算：

$$P(Evade = Yes)P(Refund = No|Evade)P(Married|Evade)P(Male|Evade) = 0$$
$$P(Evade = No)P(Refund = No|Evade)P(Married|Evade)P(Male|Evade) = \frac{32}{245}$$

故样本分类为 $Evade = No$

朴素贝叶斯可能会有条件概率为 0 的情况（样本中没有出现这种情况），故引入拉普拉斯平滑。Laplace 平滑是一种用于平滑分类数据的技术，引入 Laplace 平滑法是用来解决 0 概率问题。

$$P_\lambda(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k) + \lambda}{N + k\lambda}$$
$$P_\lambda(x_i = a_j|y = c_k) = \frac{\sum_{i=1}^{N} I(x_i = a_j, y_i = c_k) + \lambda}{\sum_{i=1}^{N} I(y_i = c_k) + A\lambda}$$

其中 K 表示类别数量，A 表示 a_j 中不通知的数量，通常为 1。引入 Laplace 平滑后避免了概率为 0 的情况，又保证了最终概率和为 1。

1　3 K-means 聚类方法

无监督学习

无监督学习主要可以分为四类 聚类、降维、关联规则挖掘和推荐系统

K-means 算法概述

它是一种迭代过程，在过程中，数据集被分组为 k 个（超参数）预定义的不重叠的聚类或子组，使得簇的内部点尽可能相似，同时试图保持簇在不同的空间，使得簇的质心与数据点之间的平方距离之和最小。Kmeans 算法的伪代码如下：

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

过程：

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
repeat
　令 $C_i = \varnothing \ (1 \leqslant i \leqslant k)$
　for $j = 1, 2, \ldots, m$ **do**
　　计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i \ (1 \leqslant i \leqslant k)$ 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: $\lambda_j = \arg\min_{i \in \{1,2,\ldots,k\}} d_{ji}$;
　　将样本 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$;
　end for
　for $i = 1, 2, \ldots, k$ **do**
　　计算新均值向量: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
　　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
until 当前均值向量均未更新

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

图 9.2　k 均值算法

肘部法则

如何确定超参数 k？通常根据肘部图，柑橘 loss function value 急剧下降变得缓慢的一个肘点作为超参数 k 的点。