中国科学院大学
University of Chinese Academy of Sciences

《模式识别与机器学习》
Chap 2 课程作业解答

2022 年 9 月 10 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

设以下模式类别具有正态概率密度函数:

$$\omega_1:\left\{(0,0)^{\mathrm{T}},(2,0)^{\mathrm{T}},(2,2)^{\mathrm{T}},(0,2)^{\mathrm{T}}\right\},\quad \omega_2:\left\{(4,4)^{\mathrm{T}},(6,4)^{\mathrm{T}},(6,6)^{\mathrm{T}},(4,6)^{\mathrm{T}}\right\}$$

(1). 设 $P(\omega_1)=P(\omega_2)=1/2$, 求这两类模式之间的贝叶斯判别界面的方程式; (2). 绘出判别界面.

Solution:

(1). 易知正态分布模式的贝叶斯判别函数为

$$d_i(\boldsymbol{x})=\ln P(\omega_i)-\frac{1}{2}\ln|\boldsymbol{C}_i|-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_i)^{\mathrm{T}}\boldsymbol{C}_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i),i=1,2 \tag{1}$$

模式的均值向量 $\boldsymbol{m}_i$ 和协方差矩阵 $\boldsymbol{C}_i$ 可用下式估计:

$$\widehat{\boldsymbol{m}}_i=\frac{1}{N_i}\sum_{j=1}^{N_i}\boldsymbol{x}^{(j)},\quad \widehat{\boldsymbol{C}}_i=\frac{1}{N_i}\sum_{j=1}^{N_i}\left(\boldsymbol{x}^{(j)}-\widehat{\boldsymbol{m}}_i\right)\left(\boldsymbol{x}^{(j)}-\widehat{\boldsymbol{m}}_i\right)^{\mathrm{T}},i=1,2 \tag{2}$$

其中 N_i 为类别 ω_i 中的模式的样本数量, 于是由上式计算出:

$$\widehat{\boldsymbol{m}}_1=\begin{pmatrix}1\\1\end{pmatrix},\widehat{\boldsymbol{m}}_2=\begin{pmatrix}5\\5\end{pmatrix},\widehat{\boldsymbol{C}}_1=\begin{pmatrix}1&0\\0&1\end{pmatrix},\widehat{\boldsymbol{C}}_2=\begin{pmatrix}1&0\\0&1\end{pmatrix} \tag{3}$$

显然 $\widehat{\boldsymbol{C}}_1=\widehat{\boldsymbol{C}}_2=\boldsymbol{C}$, 于是有:

$$\begin{aligned}d_i(\boldsymbol{x})&=\ln P(\omega_i)-\frac{1}{2}\ln|\boldsymbol{C}|-\frac{1}{2}\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}-\underbrace{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_i}_{1\times1\text{的数}}-\underbrace{\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}}_{1\times1\text{的数}}+\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_i\right)\\&=\ln P(\omega_i)-\frac{1}{2}\ln|\boldsymbol{C}|-\frac{1}{2}\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}-\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_i\right)^{\mathrm{T}}-\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}+\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_i\right)\\&=\ln P(\omega_i)-\frac{1}{2}\ln|\boldsymbol{C}|-\left(\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}-\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}+\frac{1}{2}\boldsymbol{m}_i{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_i\right),i=1,2\end{aligned}$$

由于 $P(\omega_1)=P(\omega_2)$, 从而有贝叶斯判别界面的方程式:

$$d_1(\boldsymbol{x})-d_2(\boldsymbol{x})=(\boldsymbol{m}_1-\boldsymbol{m}_2)^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{x}-\frac{1}{2}\boldsymbol{m}_1{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_1+\frac{1}{2}\boldsymbol{m}_2{}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{m}_2=-4x_1-4x_2+24=0 \tag{4}$$

(2). 判别界面方程可化简为 $x_1+x_2=6$, 于是判别界面如下图所示:

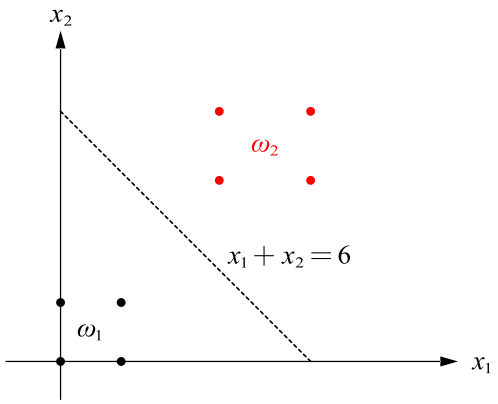


图 1: 判别界面

Problem 2

编写两类正态分布模式的贝叶斯分类程序 (可选例题或上述作业题为分类模式).

正态分布模式的贝叶斯判别 `bayes discrimination()` 函数编码如下 (其中 a 是用来控制例题数据或作业题数据的变量, $a = 1$ 表示以作业题数据作为输入, $a = 0$ 表示例题数据作为输入):

```python
import numpy as np
import sympy as sp

def bayes_discrimination(a, pw1, pw2, X_1, X_2):
    N_1 = (X_1[0].shape)[0]  # 获取样本个数
    N_2 = (X_2[0].shape)[0]  # 获取样本个数

    m_1 = np.mean(X_1,axis=1)  # 计算均值向量 m_1
    m_1 = np.matrix(m_1).T
    m_2 = np.mean(X_2,axis=1)  # 计算均值向量 m_2
    m_2 = np.matrix(m_2).T

    Cov_1 = np.cov(X_1)  # 计算协差阵 Cov_1
    C_1 = Cov_1*(N_1-1)/(N_1)  # 修正协差阵为 C_1
    C_1 = np.matrix(C_1)
    Cov_2 = np.cov(X_2)  # 计算协差阵 Cov_2
    C_2 = Cov_2*(N_2-1)/(N_2)  # 修正协差阵为 C_2
    C_2 = np.matrix(C_2)

    det_C1 = np.linalg.det(C_1) # 计算协差阵的行列式
    det_C2 = np.linalg.det(C_2) # 计算协差阵的行列式

    ### 求取贝叶斯判别函数 d_i(x)###
    if(a == 0):
        x = np.matrix([sp.Symbol('x_1'), sp.Symbol('x_2'), sp.Symbol('x_3')]).T
    elif(a == 1):
        x = np.matrix([sp.Symbol('x_1'), sp.Symbol('x_2')]).T
    D_1 = np.log(pw1) - 0.5 * np.log(det_C1) - 1/2 * (x -
    ↪  m_1).T.dot(C_1.I).dot(x - m_1)  # 判别函数 d_1(x)
    D_2 = np.log(pw2) - 0.5 * np.log(det_C2) - 1/2 * (x -
    ↪  m_2).T.dot(C_2.I).dot(x - m_2)  # 判别函数 d_2(x)
    D = np.log(pw1) - np.log(pw2) + (m_1 - m_2).T.dot(C_1.I).dot(x) + \
        1/2 * m_2.T.dot(C_1.I).dot(m_2) - 1/2 * m_1.T.dot(C_1.I).dot(m_1)  # 特
        ↪  殊情形下简化后的判别界面方程表达式
    print('d_1(x)=', sp.simplify(D_1), sep = '\n')  # 打印判别函数 d_1(x)
    print('d_2(x)=',sp.simplify(D_2), sep = '\n')  # 打印判别函数 d_2(x)
    print('d_1(x)-d_2(x)=',sp.simplify(D_1-D_2), sep = '\n')  # 直接相减得到的
    ↪  通用判别界面方程
    print('D=',sp.simplify(D))  # 特殊情形下判别界面方程的简化表达式
    return
```

主函数编码如下:

```
if __name__ == "__main__":
    a = input('例题请输入 0, 作业题请输入 1: ')
    a = int(a)
if(a == 0):
    ##### 以下是例题的输入数据 #####
    pw1 = 0.5  ## 这里输入先验概率 pw1
    pw2 = 0.5  ## 这里输入先验概率 pw2
    X_1 = np.array([[1, 0, 1, 1], [0, 0, 1, 0], [1, 0, 0, 0]]) # 输入样本矩阵 X_1
    X_2 = np.array([[0, 0, 0, 1], [0, 1, 1, 1], [1, 1, 0, 1]]) # 输入样本矩阵 X_2
    ##### 以上是例题的输入数据 #####
    bayes_discrimination(a, pw1, pw2, X_1, X_2) ## 输出判别函数和判别界面方程
elif(a == 1):
    ##### 以下是作业题的输入数据 #####
    pw1 = 0.5  # 这里输入先验概率 pw1
    pw2 = 0.5  # 这里输入先验概率 pw2
    X_1 = np.array([[0, 2, 2, 0],[0, 0, 2, 2]]) # 输入样本矩阵 X_1
    X_2 = np.array([[4, 6, 6, 4],[4, 4, 6, 6]]) # 输入样本矩阵 X_2
    ##### 以上是作业题的输入数据 #####
    bayes_discrimination(a, pw1, pw2, X_1, X_2) ## 输出判别函数和判别界面方程
```

上述程序已保存为 chap1.py 脚本, conda 的 base 环境里装了 numpy 和 sympy 库之后, 在终端里执行命令: python chap1.py, 即可输出作业题的判别函数 $d_1(\boldsymbol{x}), d_2(\boldsymbol{x})$ 分别为

$$d_1(\boldsymbol{x}) = d_1(x_1, x_2) = -0.5x_1^2 + 1.0x_1 - 0.5x_2^2 + 1.0x_2 - 1.6931, \tag{5}$$

$$d_1(\boldsymbol{x}) = d_2(x_1, x_2) = -0.5x_1^2 + 5.0x_1 - 0.5x_2^2 + 5.0x_2 - 25.6931 \tag{6}$$

相应的界面判别方程为

$$D(\boldsymbol{x}) = D(x_1, x_2) = d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) = -4.0x_1 - 4.0x_2 + 24.0 = 0 \tag{7}$$

也得到例题的判别函数 $d_1(\boldsymbol{x}), d_2(\boldsymbol{x})$ 分别为

$$d_1(\boldsymbol{x}) = -4.0x_1^2 + 4.0x_1x_2 + 4.0x_1x_3 + 4.0x_1 - 4.0x_2^2 - 4.0x_2x_3 - 4.0x_3^2 + 0.5794 \tag{8}$$

$$d_2(\boldsymbol{x}) = -4.0x_1^2 + 4.0x_1x_2 + 4.0x_1x_3 - 4.0x_1 - 4.0x_2^2 - 4.0x_2x_3 + 8.0x_2 - 4.0x_3^2 + 8.0x_3 - 3.4206 \tag{9}$$

相应的界面判别方程为

$$D(\boldsymbol{x}) = D(x_1, x_2, x_3) = d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) = 8.0x_1 - 8.0x_2 - 8.0x_3 + 4.0 = 0 \tag{10}$$

Problem 3

结合生活中的例子, 出一道用贝叶斯判别及贝叶斯最小风险判别求解的题目.

(1). 假设某地区居民的新冠感染率为 0.005, 居民的状态只有感染 (ω_1) 和非感染 (ω_2) 两种. 现在**国家查得**某核酸机构的数据得知假阳性的比例为 0.05, 假阴性的比例为 0.01. 若已知某个人的核酸检测结果呈阳性, 则他最可能处于什么状态?

Solution:

根据题意易知 $P(\omega_1)=0.005, P(\omega_2)=0.995, p\left(x=\text{阳}|\omega_2\right)=0.05, p\left(x=\text{阴}|\omega_1\right)=0.01$. 根据贝叶斯公式有如下:

$$P\left(\omega_1|x=\text{阳}\right)=\frac{P(\omega_1)\,p\left(x=\text{阳}|\omega_1\right)}{\sum_{i=1}^{2}P(\omega_i)\,p\left(x=\text{阳}|\omega_i\right)}=\frac{0.005\times0.99}{0.005\times0.99+0.995\times0.05}=0.0904936$$

$$P\left(\omega_2|x=\text{阳}\right)=\frac{P(\omega_2)\,p\left(x=\text{阳}|\omega_2\right)}{\sum_{i=1}^{2}P(\omega_i)\,p\left(x=\text{阳}|\omega_i\right)}=\frac{0.05\times0.995}{0.005\times0.99+0.995\times0.05}=0.909506$$

由于 $P\left(\omega_1|x=\text{阳}\right)<P\left(\omega_2|x=\text{阳}\right)\therefore x\in\omega_2$, 因此可以得知该核酸机构是吃干饭的.

(2). 国家为了防止某些检测机构投机倒把、指阳为阴、指阴为阳, 现需要对核酸机构进行相应的罚款来予以匡正. 因此不妨设出一个假阳性的国家罚款为 L_{21}(元)、出一个**假阴性**[1]的国家罚款为 L_{12}(元). 现在国家询问某 UCAS 学子: 应当怎样指定罚款 L_{21} 和 L_{12} 来确保核酸机构很难弄虚作假且精准检测.

Solution:

先计算当拿到阳性报告时的各类平均风险:

$$r_1\left(x=\text{阳}\right)=\underbrace{L_{11}p\left(x=\text{阳}|\omega_1\right)P(\omega_1)}_{L_{11}=0\text{(表示不失分)}}+L_{21}p\left(x=\text{阳}|\omega_2\right)P(\omega_2)=L_{21}\times0.05\times0.995$$

$$r_2\left(x=\text{阳}\right)=L_{12}p\left(x=\text{阳}|\omega_1\right)P(\omega_1)+\underbrace{L_{22}p\left(x=\text{阳}|\omega_2\right)P(\omega_2)}_{L_{22}=0\text{(表示不失分)}}=L_{12}\times0.99\times0.005$$

再计算当拿到阴性报告时的各类平均风险:

$$r_1\left(x=\text{阴}\right)=\underbrace{L_{11}p\left(x=\text{阴}|\omega_1\right)P(\omega_1)}_{L_{11}=0\text{(表示不失分)}}+L_{21}p\left(x=\text{阴}|\omega_2\right)P(\omega_2)=L_{21}\times0.95\times0.995$$

$$r_2\left(x=\text{阴}\right)=L_{12}p\left(x=\text{阴}|\omega_1\right)P(\omega_1)+\underbrace{L_{22}p\left(x=\text{阴}|\omega_2\right)P(\omega_2)}_{L_{22}=0\text{(表示不失分)}}=L_{12}\times0.01\times0.005$$

现在需要使得拿到阳性样本时判别为 ω_1 和拿到阴性样本时判别为 ω_2 的平均风险都最小, 即

$$L_{12}\times0.01\times0.005<L_{21}\times0.95\times0.995,\ \ L_{21}\times0.05\times0.995<L_{12}\times0.99\times0.005$$

解得不等式为 $0<\frac{L_{12}}{18905}<L_{21}<\frac{99L_{12}}{995}$, 于是可选罚款数为 $L_{12}=18905, L_{21}=1881$ 来确保核酸机构不能投机倒把的发国难财且精准检测.

[1]请读者仔细思考为何假阴性的影响比较恶劣 (不论机构是有意的还是无意的), 这在罚款数中也可体现出来.

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 3 课程作业解答

2022 年 10 月 11 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

在一个 10 类的模式识别问题中, 有 3 类单独满足多类情况 1, 其余的类别满足多类情况 2. 问该模式识别问题所需判别函数的最少数目是多少?

Solution: 将 10 类问题可看作 4 类满足多类情况 1 的问题, 可将 3 类单独满足多类情况 1 的类找出来, 剩下的 7 类全部划到 4 类中剩下的一个子类中. 再在此子类中, 运用多类情况 2 的判别法则进行分类, 此时需要 $7*(7-1)/2=21$ 个判别函数. 故共需要 4+21=25 个判别函数.

Problem 2

一个三类问题, 其判别函数如下:

$$d_1(\boldsymbol{x})=-x_1,\ d_2(\boldsymbol{x})=x_1+x_2-1,\ d_3(\boldsymbol{x})=x_1-x_2-1$$

(1). 设这些函数是在多类情况 1 条件下确定的, 绘出其判别界面和每一个模式类别的区域.
(2). 设为多类情况 2, 并使: $d_{12}(\boldsymbol{x})=d_1(\boldsymbol{x}), d_{13}(\boldsymbol{x})=d_2(\boldsymbol{x}), d_{23}(\boldsymbol{x})=d_3(\boldsymbol{x})$. 绘出其判别界面和多类情况 2 的区域.
(3). 设 $d_1(\boldsymbol{x}), d_2(\boldsymbol{x})$ 和 $d_3(\boldsymbol{x})$ 是在多类情况 3 的条件下确定的, 绘出其判别界面和每类的区域.

Solution: 三种情况分别如下图1中所示:

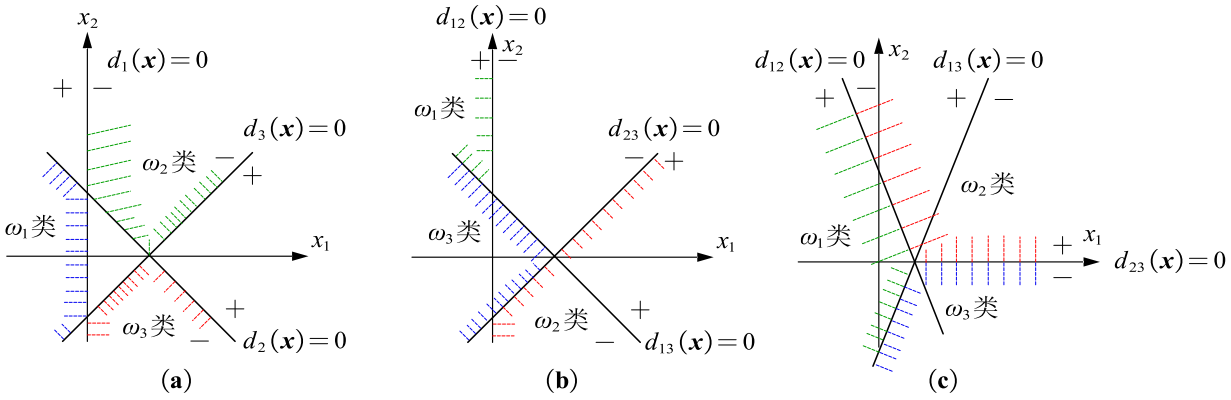


图 1: 判别界面与区域

Problem 3

两类模式, 每类包括 5 个 3 维不同的模式, 且良好分布. 如果它们是线性可分的, 问权向量至少需要几个系数分量? 假如要建立二次的多项式判别函数, 又至少需要几个系数分量? (设模式的良好分布不因模式变化而改变)

Solution: (1). 若是线性可分的, 则权向量需要至少 $n+1=4$ 个系数分量;
(2). 根据公式易知此情形下的系数分量个数至少为 $N_\omega=C_{n+r}^r=\dfrac{(n+r)!}{r!n!}=\dfrac{5!}{2!3!}=10.$

Problem 4

1. 用感知器算法求下列模式分类的解向量 $\boldsymbol{w}$:

$$\omega_1 = \left\{(0,0,0)^{\mathrm{T}}, (1,0,0)^{\mathrm{T}}, (1,0,1)^{\mathrm{T}}, (1,1,0)^{\mathrm{T}}\right\}$$
$$\omega_2 = \left\{(0,0,1)^{\mathrm{T}}, (0,1,1)^{\mathrm{T}}, (0,1,0)^{\mathrm{T}}, (1,1,1)^{\mathrm{T}}\right\}$$

Solution: 将属于 ω_2 的训练样本乘以 (-1), 并写成增广向量的形式:

$$x_1 = (0,0,0,1)^{\mathrm{T}}, x_2 = (1,0,0,1)^{\mathrm{T}}, x_3 = (1,0,1,1)^{\mathrm{T}}, x_4 = (1,1,0,1)^{\mathrm{T}}$$
$$x_5 = (0,0,-1,-1)^{\mathrm{T}}, x_6 = (0,-1,-1,-1)^{\mathrm{T}}, x_7 = (0,-1,0,-1)^{\mathrm{T}}, x_8 = (-1,-1,-1,-1)^{\mathrm{T}}$$

迭代选取 $C = 1, w_1 = (0,0,0,0)^{\mathrm{T}}$, 则迭代过程中权向量变换如下:

$$w(2) = (0,0,0,1)^{\mathrm{T}}, w(3) = (0,0,-1,0), w(4) = (0,-1,-1,-1)^{\mathrm{T}}, w(5) = (0,-1,-1,0)^{\mathrm{T}}$$
$$w(6) = (1,-1,-1,1)^{\mathrm{T}}, w(7) = (1,-1,-2,0)^{\mathrm{T}}, w(8) = (1,-1,-2,1)^{\mathrm{T}}, w(9) = (2,-1,-1,2)^{\mathrm{T}}$$
$$w(10) = (2,-1,-2,1)^{\mathrm{T}}, w(11) = (2,-2,-2,0)^{\mathrm{T}}, w(12) = (2,-2,-2,1)^{\mathrm{T}} \text{(此时已收敛)}$$

所以最终得到解向量 $\boldsymbol{w} = (2,-2,-2,1)^{\mathrm{T}}$, 相应的判别函数为 $d(\boldsymbol{x}) = 2x_1 - 2x_2 - 2x_3 + 1$.

2. 编写求解上述问题的感知器算法程序.

Solution: C++ 代码如下:

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <numeric>
using namespace std;

int main() {
    int d, n1, n2, C;
    cout << " 请依次输入样本维数 d, w1 类的样本数 n1, w2 类的样本数 n2, 迭代步长 C" << endl;
    cin >> d >> n1 >> n2 >> C;
    vector<vector<float>> omega_1(n1, vector<float>(d));
    vector<vector<float>> omega_2(n2, vector<float>(d));
    cout << " 请依次输入 w1 类的模式样本" << endl;
    for(int i = 0; i < n1; i++) {
        for(int j = 0; j < d; j++) {
            cout << " omega_1[" << i << "][" << j << "] : ";
            cin >> omega_1[i][j];
        }
    }
    cout << " 请依次输入 w2 类的模式样本" << endl;
    for(int i = 0; i < n2; i++) {
        for(int j = 0; j < d; j++) {
            cout << " omega_2[" << i << "][" << j << "] : ";
            cin >> omega_2[i][j];
        }
    }

```

```
    vector<vector<float>> sample_1(n1, vector<float>(d + 1));
    vector<vector<float>> sample_2(n2, vector<float>(d + 1));
    /* 增广后的 w1 类的模式样本 */
    for(int i = 0; i < n1; i++) {
        copy(omega_1[i].begin(), omega_1[i].end(), sample_1[i].begin());
        sample_1[i][d] = 1;
    }
    /* 增广后的 w2 类的模式样本 */
    for(int i = 0; i < n2; i++) {
        copy(omega_2[i].begin(), omega_2[i].end(), sample_2[i].begin());
        sample_2[i][d] = 1;
    }
    for(int i = 0; i < n2; i++) {
        for(int j = 0; j < d + 1; j++) {
            sample_2[i][j] = (-1.0) * sample_2[i][j];  //增广的 w2 训练样本乘以-1
        }
    }
    int n = n1 + n2;
    vector<vector<float>> sample(n, vector<float>(d + 1));
    copy(sample_1.begin(), sample_1.end(), sample.begin());
    copy(sample_2.begin(), sample_2.end(), sample.begin() + n1);
    vector<float> w(d + 1, 0);
    int cnt = 0;
    while (cnt != n) { //当被正确分类的样本数 cnt 等于总样本数 n 时，结束循环
        cnt = 0; //计数变量清零
        for(int i = 0; i < n; i++) {
            if(inner_product(w.begin(), w.end(), sample[i].begin(), 0.0) > 0) {
                cnt++;  //若被正确分类，则权向量不变且对应的样本数自增一
            }
            else {
                for(int j = 0; j < d + 1; j++) {  //若 x(k) 被错误分类，
                    w[j] = w[j] + C * sample[i][j];  //则 w(k+1) = w(k) + C * x(k)
                }
            }
        }
    }
    cout << " 解向量 w 的分量分别为" << endl;
    for(int i = 0; i < d + 1; i++) {
        cout << " w[" << i << "]"  << " = " << w[i];
    }
    return 0;
}
```

程序执行结果如下图2所示:

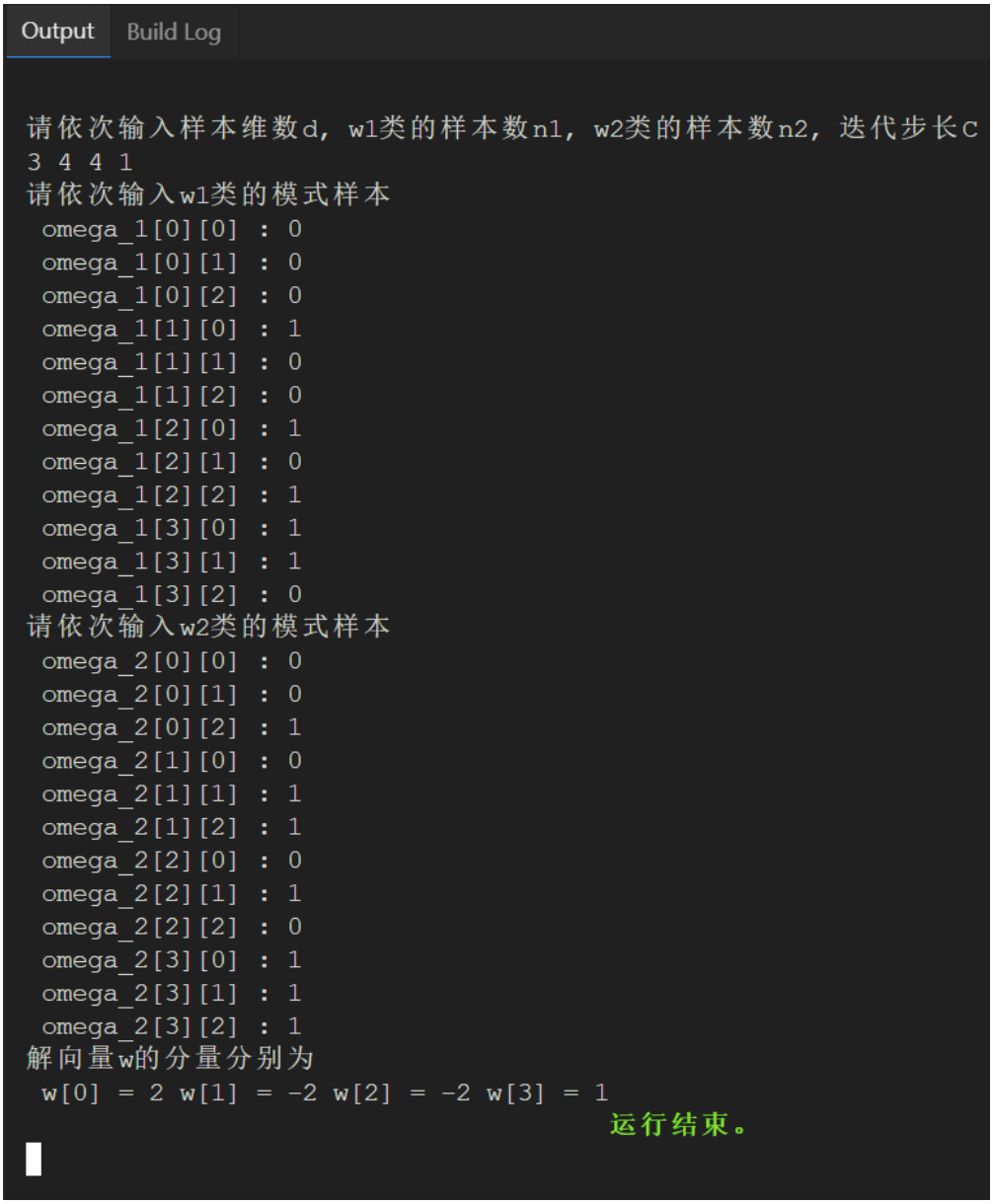


图 2: Problem 4 程序运行结果

Problem 5

用多类感知器算法求下列模式的判别函数:

$$\omega_1 : (-1,-1)^{\mathrm{T}}, \omega_2 : (0,0)^{\mathrm{T}}, \omega_3 : (1,1)^{\mathrm{T}}$$

Solution: 采用一般化的感知器算法, 将模式样本写成增广形式, 即

$$x_1 = (-1,-1,1)^{\mathrm{T}}, x_2 = (0,0,1)^{\mathrm{T}}, x_3 = (1,1,1)^{\mathrm{T}}$$

取初始值 $w_1 = w_2 = w_3 = (0,0,0)^{\mathrm{T}}$, 取 $C = 1$.

以 x_1 为训练样本来进行第一次迭代, 计算内积得到 $d_1(1) = d_2(1) = d_3(1) = 0$, 故

$$w_1(2) = (-1,-1,1)^{\mathrm{T}}, w_2(2) = (1,1,-1)^{\mathrm{T}}, w_3(2) = (1,1,-1)^{\mathrm{T}}$$

以 x_2 为训练样本来进行第 2 次迭代, 计算内积得到 $d_1(2) = 1, d_2(2) = -1, d_3(2) = -1$, 故

$$w_1(3) = (-1,-1,0)^{\mathrm{T}}, w_2(3) = (1,1,0)^{\mathrm{T}}, w_3(3) = (1,1,-2)^{\mathrm{T}}$$

以 x_3 为训练样本来进行第 3 次迭代, 计算内积得到 $d_1(3) = -2, d_2(3) = 2, d_3(3) = 0$, 故

$$w_1(4) = (-1,-1,0)^{\mathrm{T}}, w_2(4) = (0,0,-1)^{\mathrm{T}}, w_3(4) = (2,2,-1)^{\mathrm{T}}$$

以 x_1 为训练样本来进行第 4 次迭代, 计算内积得到 $d_1(4) = 2, d_2(4) = -1, d_3(4) = -5$, 显然 x_1 被正确分类了, 故权向量此时不用更新;

以 x_2 为训练样本来进行第 5 次迭代, 计算内积得到 $d_1(5) = 0, d_2(5) = -1, d_3(5) = -1$, 故

$$w_1(6) = (-1,-1,-1)^{\mathrm{T}}, w_2(6) = (0,0,0)^{\mathrm{T}}, w_3(6) = (2,2,-2)^{\mathrm{T}}$$

以 x_3 为训练样本来进行第 6 次迭代, 计算内积得到 $d_1(6) = -3, d_2(6) = 0, d_3(6) = 2$, 显然 x_3 被正确分类了, 故权向量此时不用更新;

以 x_1 为训练样本来进行第 7 次迭代, 计算内积得到 $d_1(7) = 1, d_2(7) = 0, d_3(7) = -6$, 显然 x_1 被正确分类了, 故权向量此时不用更新;

以 x_2 为训练样本来进行第 8 次迭代, 计算内积得到 $d_1(8) = -1, d_2(8) = 0, d_3(8) = -2$, 显然 x_2 被正确分类了, 故权向量此时不用更新. 由于第 6,7,8 次迭代中对 x_1, x_2, x_3 均以正确分类, 故权向量的解为 $w_1 = (-1,-1,-1)^{\mathrm{T}}, w_2 = (0,0,0)^{\mathrm{T}}, w_3 = (2,2,-2)^{\mathrm{T}}$ 且判别函数为 $d_1(\boldsymbol{x}) = -x_1 - x_2 - 1, d_2(\boldsymbol{x}) = 0, d_3(\boldsymbol{x}) = 2x_1 + 2x_2 - 2$. 类似 Problem4, 可以编写出多类感知器算法的 C++ 程序 (见后页), 程序执行结果如下图3中所示:

```
请依次输入样本维数d, 样本类别数M, 迭代步长C
2 3 1
请依次输入各类别的样本个数
1 1 1
请按照w1,w2,...,wM的类别顺序依次输入各类的模式样本
 sample[0][0] = -1
 sample[0][1] = -1
 sample[1][0] = 0
 sample[1][1] = 0
 sample[2][0] = 1
 sample[2][1] = 1
解向量w[1]的分量分别为:
 w[1][1] = -1,
 w[1][2] = -1,
 w[1][3] = -1,
解向量w[2]的分量分别为:
 w[2][1] = 0,
 w[2][2] = 0,
 w[2][3] = 0,
解向量w[3]的分量分别为:
 w[3][1] = 2,
 w[3][2] = 2,
 w[3][3] = -2,

运行结束。
```

图 3: Problem 5 程序运行结果

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <numeric>
using namespace std;

int main() {
    int d, M, C;
    cout << " 请依次输入样本维数 d, 样本类别数 M, 迭代步长 C" << endl;
    cin >> d >> M >> C;
    vector<int> N(M);

    cout << " 请依次输入各类别的样本个数" << endl;
    for(int i = 0; i < M; i++) {
        cin >> N[i];
    }

    int n = accumulate(N.begin(), N.end(), 0);
    vector<vector<float>> sample_init(n, vector<float>(d));
    cout << " 请按照 w1,w2,...,wM 的类别顺序依次输入各类的模式样本" << endl;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < d; j++) {
            cout << " sample[" << i << "][" << j << "] = ";
            cin >> sample_init[i][j];
        }
    }

    vector<vector<float>> sample(n, vector<float>(d + 1));
    for(int i = 0; i < n; i++) {
        copy(sample_init[i].begin(), sample_init[i].end(), sample[i].begin());
        sample[i][d] = 1;
    }
    vector<vector<float>> w(M, vector<float>(d + 1, 0));
    vector<float> D(M);
    int cnt = 0;
    while (cnt != n) { //当被正确分类的样本数 cnt 等于总样本数 n 时, 结束循环
        cnt = 0; //每一轮迭代都需要把计数变量 cnt 清零
        for(int i = 0; i < n; i++) {
            for(int m = 0; m < M; m++) {
                D[m] = inner_product(w[m].begin(), w[m].end(), sample[i].begin(), 0.0);
            }
            int index = max_element(D.begin(), D.end()) - D.begin();
            int flag = count(D.begin(), D.end(), D[index]);
            if(i <= N[0] - 1 && i >= 0) { //第 1 类需要单独判断, 否则会越界
                if(index + 1 == 1 && flag == 1) {
                    cnt++;  //若当前样本被正确分类, 则权向量不变且对应的样本数自增一
                }
                else {  //若当前样本没被正确分类, 则按算法规则调整权向量
                    for(int m = 0; m < M; m++) {
                        if(D[m] >= D[0] && m != 0) {
                            for(int j = 0; j < d + 1; j++) {
                                w[m][j] = w[m][j] - C * sample[i][j];
                            }
                        }
```

```
                        else if(D[m] < D[0]) {
                            ;
                        }
                        else if(m == 0) {
                            for(int j = 0; j < d + 1; j++) {
                                w[m][j] = w[m][j] + C * sample[i][j];
                            }
                        }
                    }
                }
            }
            else {
                int Class = 2;
                for(int k = 2; k <= M; k++) {
                    int left = accumulate(N.begin(), N.begin() + k - 1, 0);
                    int right = accumulate(N.begin(), N.begin() + k, 0) - 1;
                    if(i >= left && i <= right) {
                        Class = k;  //求出当前样本所在的真实类别 (即第 k 类)
                        break;
                    }
                }

                if(index + 1 == Class && flag == 1) {
                    cnt++;  //若当前样本被正确分类, 则权向量不变且对应的样本数自增一
                }

                else {  //若当前样本没被正确分类, 则按规则调整权向量
                    for(int m = 0; m < M; m++) {
                        if(D[m] >= D[Class - 1] && m != Class - 1) {
                            for(int j = 0; j < d + 1; j++) {
                                w[m][j] = w[m][j] - C * sample[i][j];
                            }
                        }
                        else if(D[m] < D[Class - 1]) {
                            ;
                        }
                        else if(m == Class - 1) {
                            for(int j = 0; j < d + 1; j++) {
                                w[m][j] = w[m][j] + C * sample[i][j];
                            }
                        }
                    }
                }
            }
        }
    }
    for(int i = 0; i < M; i++) {
        cout << " 解向量 w[" << i + 1 << "] 的分量分别为: " << endl;
        for(int j = 0; j < d + 1; j++) {
            cout << " w[" << i + 1 << "][" << j + 1 << "] = " << w[i][j] << "," << endl;
        }
    }
    return 0;
}
```

Problem 6

采用梯度法和准则函数 $J(w,x,b)=\dfrac{1}{8\|x\|^2}\left[\left(w^{\mathrm{T}}x-b\right)-\left|w^{\mathrm{T}}x-b\right|\right]^2$(其中 $b>0$), 试导出两类模式的分类算法.

Solution: 上述式子对 w 求导可得

$$\frac{\partial J}{\partial w}=\frac{1}{4\|x\|^2}\left[\left(w^{\mathrm{T}}x-b\right)-\left|w^{\mathrm{T}}x-b\right|\right]\cdot\left[x-x\cdot\operatorname{sgn}\left(w^{\mathrm{T}}x-b\right)\right]$$

其中 sgn 表示符号函数, 即 $\operatorname{sgn}\left(w^{\mathrm{T}}x-b\right)=\begin{cases}1, w^{\mathrm{T}}x-b>0\\ -1, w^{\mathrm{T}}x-b\le 0\end{cases}$.

于是可得到迭代式

$$w(k+1)=w(k)+C\frac{\partial J}{\partial w(k)}=w(k)+\begin{cases}0, w^{\mathrm{T}}(k)x-b>0\\ \frac{w^{\mathrm{T}}(k)x-b}{\|x\|^2}Cx, w^{\mathrm{T}}(k)x-b\le 0\end{cases}$$

至此, Chap 3 的作业解答完毕.

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 4 课程作业解答

2022 年 10 月 14 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

设有如下三类模式样本集 ω_1, ω_2 和 ω_3, 其先验概率相等, 求 S_w 和 S_b.

$$\begin{aligned}
&\omega_1 : \left\{(1,0)^{\mathrm{T}}, (2,0)^{\mathrm{T}}, (1,1)^{\mathrm{T}}\right\};\\
&\omega_2 : \left\{(-1,0)^{\mathrm{T}}, (0,1)^{\mathrm{T}}, (-1,1)^{\mathrm{T}}\right\};\\
&\omega_3 : \left\{(-1,-1)^{\mathrm{T}}, (0,-1)^{\mathrm{T}}, (0,-2)^{\mathrm{T}}\right\}
\end{aligned}$$

Solution: 易知 S_w, S_b 的计算公式为

$$S_b = \sum_{i=1}^{M} P(\omega_i)(\boldsymbol{m}_i - \boldsymbol{m}_0)(\boldsymbol{m}_i - \boldsymbol{m}_0)^{\mathrm{T}}, \tag{1}$$

$$S_w = \sum_{i=1}^{M} P(\omega_i)\,\mathbf{E}\left\{(\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m}_i)^{\mathrm{T}} | x \in \omega_i\right\} = \sum_{i=1}^{M} P(\omega_i)\boldsymbol{C}_i \tag{2}$$

根据题意有: $P(\omega_1) = P(\omega_2) = P(\omega_3) = \frac{1}{3}$, 而且计算可知:

$$\boldsymbol{m}_0 = \begin{pmatrix} 1/9 \\ -1/9 \end{pmatrix}, \boldsymbol{m}_1 = \begin{pmatrix} 4/3 \\ 1/3 \end{pmatrix}, \boldsymbol{m}_2 = \begin{pmatrix} -2/3 \\ 2/3 \end{pmatrix}, \boldsymbol{m}_3 = \begin{pmatrix} -1/3 \\ -4/3 \end{pmatrix}$$

于是根据 (1) 式可知:

$$\begin{aligned}
S_b &= \frac{1}{3}\left[\begin{pmatrix} 11/9 \\ 4/9 \end{pmatrix}\begin{pmatrix} 11/9 & 4/9 \end{pmatrix} + \begin{pmatrix} -7/9 \\ 7/9 \end{pmatrix}\begin{pmatrix} -7/9 & 7/9 \end{pmatrix} + \begin{pmatrix} -4/9 \\ -11/9 \end{pmatrix}\begin{pmatrix} -4/9 & -11/9 \end{pmatrix}\right]\\
&= \frac{1}{81}\begin{pmatrix} 62 & 13 \\ 13 & 62 \end{pmatrix}
\end{aligned}$$

根据协方差矩阵的估计式:

$$\boldsymbol{C}_i = \frac{1}{N_i}\sum_{j=1}^{N_i}\left(\boldsymbol{x}^{(j)} - \boldsymbol{m}_i\right)\left(\boldsymbol{x}^{(j)} - \boldsymbol{m}_i\right)^{\mathrm{T}} (i = 1,2,3) \tag{3}$$

于是根据上述 (3) 式有:

$$\boldsymbol{C}_1 = \frac{1}{3}\left\{\begin{pmatrix} -1/3 & -1/3 \end{pmatrix}\begin{pmatrix} -1/3 \\ -1/3 \end{pmatrix} + \begin{pmatrix} 2/3 & -1/3 \end{pmatrix}\begin{pmatrix} 2/3 \\ -1/3 \end{pmatrix} + \begin{pmatrix} -1/3 & 2/3 \end{pmatrix}\begin{pmatrix} -1/3 \\ 2/3 \end{pmatrix}\right\} = \frac{1}{9}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

$$\boldsymbol{C}_2 = \frac{1}{3}\left\{\begin{pmatrix} -1/3 & -2/3 \end{pmatrix}\begin{pmatrix} -1/3 \\ -2/3 \end{pmatrix} + \begin{pmatrix} 2/3 & 1/3 \end{pmatrix}\begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix} + \begin{pmatrix} -1/3 & 1/3 \end{pmatrix}\begin{pmatrix} -1/3 \\ 1/3 \end{pmatrix}\right\} = \frac{1}{9}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\boldsymbol{C}_3 = \frac{1}{3}\left\{\begin{pmatrix} -2/3 & 1/3 \end{pmatrix}\begin{pmatrix} -2/3 \\ 1/3 \end{pmatrix} + \begin{pmatrix} 1/3 & 1/3 \end{pmatrix}\begin{pmatrix} 1/3 \\ 1/3 \end{pmatrix} + \begin{pmatrix} 1/3 & -2/3 \end{pmatrix}\begin{pmatrix} 1/3 \\ -2/3 \end{pmatrix}\right\} = \frac{1}{9}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

因而求得

$$S_w = \sum_{i=1}^{M} P(\omega_i)\boldsymbol{C}_i = \frac{1}{3}\sum_{i=1}^{3}\boldsymbol{C}_i = \frac{1}{27}\begin{pmatrix} 6 & -1 \\ -1 & 6 \end{pmatrix}$$

Problem 2

设有如下两类样本集, 其出现的概率相等:

$$\omega_1 : \left\{ \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}^{\mathrm{T}} \right\}$$
$$\omega_2 : \left\{ \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^{\mathrm{T}} \right\}$$

运用 *Karhunen-Loeve* (*K-L*) 变换, 分别把特征空间维数降到 2 维和 1 维, 并画出样本在该空间中的位置.

Solution: 先计算样本均值

$$\boldsymbol{m} = \sum_{i=1}^{2} P\left(\omega_i\right) \boldsymbol{m}_i = \frac{1}{2} \begin{pmatrix} 3/4 \\ 1/4 \\ 1/4 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 1/4 \\ 3/4 \\ 3/4 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

再平移样本 ($\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{m}$) 以符合最佳条件:

$$\omega_1' : \left\{ \begin{pmatrix} -1/2 & -1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & -1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & -1/2 & 1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & 1/2 & -1/2 \end{pmatrix}^{\mathrm{T}} \right\}$$
$$\omega_2' : \left\{ \begin{pmatrix} -1/2 & -1/2 & 1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} -1/2 & 1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} -1/2 & 1/2 & 1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & 1/2 & 1/2 \end{pmatrix}^{\mathrm{T}} \right\}$$

再计算 $\boldsymbol{z}$ 的自相关矩阵:

$$\boldsymbol{R} = \sum_{i=1}^{2} P\left(\omega_i\right) \mathbf{E}\left(\boldsymbol{z}\boldsymbol{z}^{\mathrm{T}}\right) = \frac{1}{2}\left[\frac{1}{4} \sum_{\boldsymbol{z}^j \in \omega_1'} \boldsymbol{z}^j \left(\boldsymbol{z}^j\right)^{\mathrm{T}}\right] + \frac{1}{2}\left[\frac{1}{4} \sum_{\boldsymbol{z}^j \in \omega_2'} \boldsymbol{z}^j \left(\boldsymbol{z}^j\right)^{\mathrm{T}}\right] = \frac{1}{4} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

显然 $\boldsymbol{R}$ 的特征值为 $1/4, 1/4, 1/4$. 对应的特征向量分别为

$$\boldsymbol{\varphi}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{\varphi}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \boldsymbol{\varphi}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

若要将特征空间维数 (即 3) 降到 2, 则做如下变换即可:

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \left[\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2\right]^{\mathrm{T}} \left(\boldsymbol{x} - \boldsymbol{m}\right) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 - 1/2 \\ x_2 - 1/2 \\ x_3 - 1/2 \end{pmatrix} = \begin{pmatrix} x_1 - 1/2 \\ x_2 - 1/2 \end{pmatrix}$$

故得到降维后的样本为:

$$\omega_1 : \left\{ \begin{pmatrix} -1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & 1/2 \end{pmatrix}^{\mathrm{T}} \right\}$$
$$\omega_2 : \left\{ \begin{pmatrix} -1/2 & -1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} -1/2 & 1/2 \end{pmatrix}^{\mathrm{T}}, \begin{pmatrix} 1/2 & 1/2 \end{pmatrix}^{\mathrm{T}} \right\}$$

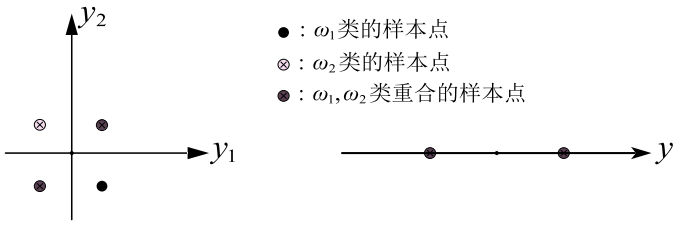
若要将特征空间维数 (即 3) 降到 1, 则做如下变换即可:

$$y = \boldsymbol{\varphi}_1{}^{\mathrm{T}} \left(\boldsymbol{x} - \boldsymbol{m}\right) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 - 1/2 \\ x_2 - 1/2 \\ x_3 - 1/2 \end{pmatrix} = x_1 - 1/2$$

故得到降维后的样本为:

$$\omega_1 : \{-1/2, 1/2\}; \quad \omega_2 : \{-1/2, 1/2\}$$

降维后的样本在空间中的分布位置如下图1中所示.



(**a**). 降到2维后的样本分布 (**b**). 降到1维后的样本分布

图 1: 样本在降维空间中的位置

至此, Chap 4 的作业解答完毕.

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 5 课程作业解答

2022 年 10 月 24 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

自己编程实现课堂上 Polynomial Curve Fitting 的例子, 体会过拟合.

Solution: 先根据均匀分布 $U(0,1)$ 随机产生 10 个数据点, 再利用函数 $y(x) = \sin(2\pi x)$ 并添加高斯噪声 (噪声均值为 0, 方差为 $\sigma^2 = 0.08^2$) 产生对应的 y 数据. 我们分别用 $M = 0, 2, 5, 9$ 次多项式对上述产生的数据点进行多项式拟合. 具体的 Matlab 代码如下所示:

```
N = 10; % 产生的数据点的个数
x = rand(10,1); % 产生均匀分布 U(0,1) 的 10 个点
noise_sigma = 0.08; % 噪声的方差为 noise_sigma^2
M = 9;
y = sin(2*pi*x) + randn(10,1)*noise_sigma; %randn 产生 N(0,1) 正态分布的数据
figure (1)
axis([0 1 -2 2])
plot(x, y, 'b.')
x_r = 0: 0.01: 1;
y_r = sin(2*pi*x_r);
hold on
plot(x_r, y_r, 'b'); % 数据的真实曲线 (蓝色的)
p_x = [];

for m = 0 : M
    p_x = [p_x, x.^m]; % 产生 [x^0;x^1;x^2,...,x^M]
end
p_x = p_x';
w = pinv(p_x*p_x')*p_x*y;
y_est = w'*p_x;
figure(1);
hold on
x_cur = 0:0.01:1;

y_cur = zeros(size(x_cur));
for m = 0 : M
    y_cur = y_cur + w(m + 1)*(x_cur.^m);
end
axis([0 1 -2 2])
plot(x_cur, y_cur, 'r') % 画出红色的拟合曲线
```

代码的运行结果和具体的多项式曲线拟合情况见后页图1,2,3,4,5,6中所示. 可以看出的是: 当 $M = 3, 4, 5$ 时, 过拟合效应较小, 当 $M = 9$(过大) 时, 多项式拟合出现了强烈的过拟合现象.

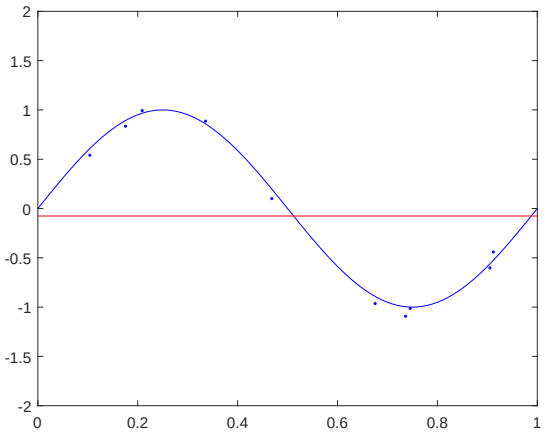


图 1: $M = 0$ 时的拟合情况

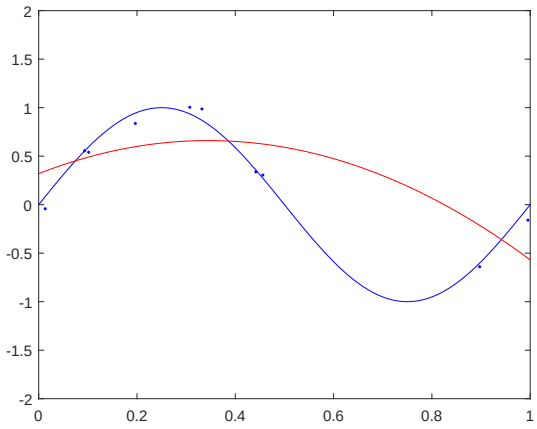


图 2: $M = 2$ 时的拟合情况

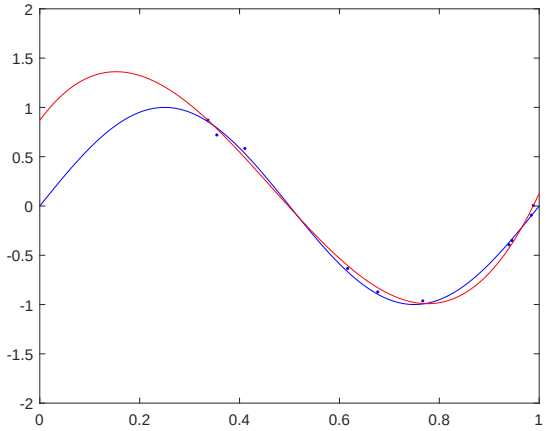


图 3: $M = 3$ 时的拟合情况

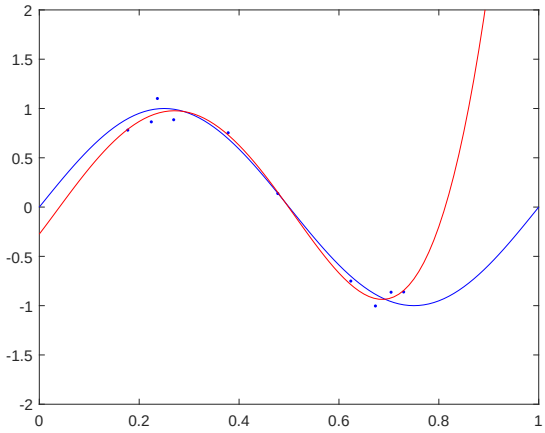


图 4: $M = 4$ 时的拟合情况

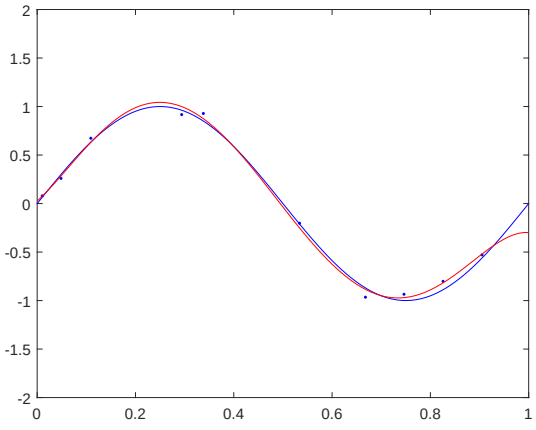


图 5: $M = 5$ 时的拟合情况

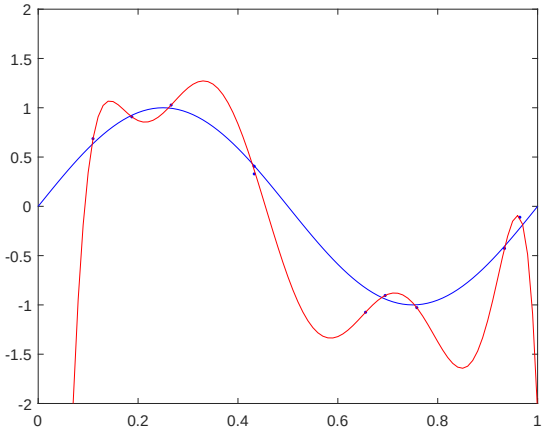


图 6: $M = 9$ 时的拟合情况

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 6 课程作业解答

2022 年 11 月 3 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

给定训练数据集 $\boldsymbol{X}=\begin{pmatrix}1 & 2 & 5 & 4\\ 2 & 5 & 1 & 2\end{pmatrix}, \boldsymbol{y}=\begin{pmatrix}19 & 26 & 19 & 20\end{pmatrix}^{\mathrm{T}}$, 令 $\alpha=0.001, \boldsymbol{w}_0=\begin{pmatrix}1 & 1\end{pmatrix}^{\mathrm{T}}$. 编程实现 SGD 和 GD 算法, 求解 $\boldsymbol{w}$.

Solution: 最优化问题 (即代价函数) 为

$$\min_{\boldsymbol{w}} J(\boldsymbol{w})=\frac{1}{2N}\sum_{i=1}^{N}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)}\right)^2$$

于是批梯度下降 (BGD) 和梯度迭代更新规则分别为 (具体的 python 训练代码见如下)

$$\frac{\partial J(\boldsymbol{w})}{\partial w_j}=\frac{1}{N}\sum_{i=1}^{N}x_j^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)}\right),\quad w_j\leftarrow w_j-\frac{\alpha}{N}\sum_{i=1}^{N}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)}\right)x_j^{(i)}, \alpha>0$$

```python
# 批量梯度下降 BGD
# 拟合函数为: y = theta * x
# 代价函数为: J = 1 / (2 * m) * ((theta * x) - y) * ((theta * x) - y).T;
# 梯度迭代为: theta = theta - alpha / m * (x * (theta * x - y).T);
import time
import numpy as np
# 1、多元线性回归的 BGD 程序
def bgd_multi():
    # 训练集, 每个样本有 2 个分量
    x = np.array([(1, 2), (2, 5), (5, 1), (4, 2)])
    y = np.array([19, 26, 19, 20])
    # 初始化
    m, dim = x.shape
    theta = np.ones(dim)  # 参数
    alpha = 0.001  # 学习率
    threshold = 0.0001  # 停止迭代的错误阈值
    iterations = 1500  # 迭代次数
    error = 0  # 初始错误为 0
    # 迭代开始
    for i in range(iterations):
        error = 1 / (2 * m) * np.dot((np.dot(x, theta) - y).T, (np.dot(x, theta) - y))
        # 迭代停止
        if abs(error) <= threshold:
            break
        theta -= alpha / m * (np.dot(x.T, (np.dot(x, theta) - y)))
    print('BGD 的迭代次数为%d,' % (i + 1), 'theta:', theta, ', error: %f' % error)
if __name__ == '__main__':
    start = time.time()
    bgd_multi()
    end = time.time()
    print('运行时间为: ', (end - start) * 1000, 'ms')
```

上述代码的输出结果为 $\boldsymbol{w}=\begin{pmatrix}2.868 & 4.565\end{pmatrix}^{\mathrm{T}}$, 循环迭代次数为 1500, 循环终止时的线性拟合误差为 $\epsilon=6.995$, BGD 算法运行时间为 13.84ms.

而对于 SGD 算法, 最优化问题 (即代价函数) 仍为

$$\min_{\boldsymbol{w}} J(\boldsymbol{w})=\frac{1}{2N}\sum_{i=1}^{N}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}-y^{(i)}\right)^2$$

于是随机梯度下降 (SGD) 和梯度迭代更新规则分别为 (具体的 python 训练代码见如下)

$$\frac{\partial J(\boldsymbol{w})}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)} - y^{(i)}\right), \quad w_j \leftarrow w_j - \alpha\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)} - y^{(i)}\right)x_j^{(i)}, \alpha > 0$$

```python
# 随机梯度下降 SGD
import time
import numpy as np
# 2、多元线性回归的 SGD 程序
def sgd():
    # 训练集，每个样本有 2 个分量
    x = np.array([(1, 2), (2, 5), (5, 1), (4, 2)])
    y = np.array([19, 26, 19, 20])
    # 初始化
    m, dim = x.shape
    theta = np.ones(dim)  # 参数
    alpha = 0.001  # 学习率
    threshold = 0.0001  # 停止迭代的错误阈值
    iterations = 1500  # 迭代次数
    error = 0  # 初始错误为 0
    # 迭代开始
    for i in range(iterations):
        error = 1 / (2 * m) * np.dot((np.dot(x, theta) - y).T, (np.dot(x,
        ↪ theta) - y))
        # 迭代停止
        if abs(error) <= threshold:
            break
        j = np.random.randint(0, m)
        theta -= alpha * (x[j] * (np.dot(x[j], theta) - y[j]))
    print('SGD 的迭代次数为%d,' % (i + 1), 'theta:', theta, ', error: %f' %
    ↪ error)
if __name__ == '__main__':
    start = time.time()
    sgd()
    end = time.time()
    print('运行时间为: ', (end - start) * 1000, 'ms')
```

上述代码的输出结果为 $\boldsymbol{w} = \begin{pmatrix}2.880 & 4.613\end{pmatrix}^{\mathrm{T}}$, 循环迭代次数为 1500, 循环终止时的线性拟合误差为 $\epsilon = 7.01$, SGD 算法的运行时间为 13.51ms. 可以看出 SGD 算法的速度是略快于 BGD 算法的.

Problem 2

利用下面表格1中的训练数据训练朴素贝叶斯分类器. 给定测试样本 $\boldsymbol{x} = (2, S)^{\mathrm{T}}$ 和 $\boldsymbol{x} = (1, N)^{\mathrm{T}}$, 请预测他们的标签.

表 1: Problem 2 的训练数据

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_1	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
x_2	S	M	M	S	S	S	M	M	L	L	L	M	M	L	L
y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

由于测试样本中出现了训练样本中未出现的特征分量, 所以需要对朴素贝叶斯分类器参数的极大似然估计做 *Laplace* 平滑 (其中平滑系数 λ 选取为 1):

$$p(y=1) = \frac{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + 1}{N + K} = \frac{9+1}{15+2} = \frac{10}{17} \Rightarrow p(y=-1) = 1 - \frac{10}{17} = \frac{7}{17}$$

第 1 个特征分量的似然函数为

$$p(x_1 = 1|y=1) = \frac{\sum_{i=1}^{N} I_{\{x_1^{(i)}=1 \wedge y^{(i)}=1\}} + 1}{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + S_1} = \frac{2+1}{9+3} = \frac{1}{4},$$

$$p(x_1 = 2|y=1) = \frac{\sum_{i=1}^{N} I_{\{x_1^{(i)}=2 \wedge y^{(i)}=1\}} + 1}{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + S_1} = \frac{3+1}{9+3} = \frac{1}{3},$$

$$p(x_1 = 3|y=1) = \frac{\sum_{i=1}^{N} I_{\{x_1^{(i)}=3 \wedge y^{(i)}=1\}} + 1}{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + S_1} = \frac{4+1}{9+3} = \frac{5}{12}$$

第 2 个特征分量的似然函数为:

$$p(x_2 = S|y=1) = \frac{\sum_{i=1}^{N} I_{\{x_2^{(i)}=S \wedge y^{(i)}=1\}} + 1}{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + S_2} = \frac{1+1}{9+4} = \frac{2}{13},$$

$$p(x_2 = M|y=1) = \frac{\sum_{i=1}^{N} I_{\{x_2^{(i)}=M \wedge y^{(i)}=1\}} + 1}{\sum_{i=1}^{N} I_{\{y^{(i)}=1\}} + S_2} = \frac{4+1}{9+4} = \frac{5}{13},$$

$$p\left(x_2=L|y=1\right)=\frac{\sum_{i=1}^{N} I_{\left\{x_2^{(i)}=L\wedge y^{(i)}=1\right\}}+1}{\sum_{i=1}^{N} I_{\left\{y^{(i)}=1\right\}}+S_2}=\frac{4+1}{9+4}=\frac{5}{13},$$

$$p\left(x_2=N|y=1\right)=\frac{\sum_{i=1}^{N} I_{\left\{x_2^{(i)}=N\wedge y^{(i)}=1\right\}}+1}{\sum_{i=1}^{N} I_{\left\{y^{(i)}=1\right\}}+S_2}=\frac{0+1}{9+4}=\frac{1}{13}$$

预测表达式为

$$p\left(y=1|\boldsymbol{x}\right)=\frac{p\left(\boldsymbol{x}|y=1\right)p\left(y=1\right)}{p\left(\boldsymbol{x}\right)}=\frac{\prod_{j=1}^{D} p\left(x_j|y=1\right)p\left(y=1\right)}{\prod_{j=1}^{D} p\left(x_j|y=1\right)p\left(y=1\right)+\prod_{j=1}^{D} p\left(x_j|y=-1\right)p\left(y=-1\right)}$$

于是对于样本 $\boldsymbol{x}^{(1)}=(2,S)^{\mathrm{T}}$, 预测概率为

$$p\left(y=1|\boldsymbol{x}^{(1)}\right)=\frac{\frac{4}{12}\times\frac{2}{13}\times\frac{10}{17}}{\frac{4}{12}\times\frac{2}{13}\times\frac{10}{17}+\frac{2+1}{6+3}\times\frac{3+1}{6+4}\times\frac{7}{17}}=0.35461$$

$$p\left(y=-1|\boldsymbol{x}^{(1)}\right)=\frac{\frac{2+1}{6+3}\times\frac{3+1}{6+4}\times\frac{7}{17}}{\frac{4}{12}\times\frac{2}{13}\times\frac{10}{17}+\frac{2+1}{6+3}\times\frac{3+1}{6+4}\times\frac{7}{17}}=0.64539$$

所以样本 $\boldsymbol{x}^{(1)}=(2,S)^{\mathrm{T}}$ 的标签预测为 $y=-1$.

而对于样本 $\boldsymbol{x}^{(2)}=(1,N)^{\mathrm{T}}$, 预测概率为

$$p\left(y=1|\boldsymbol{x}^{(2)}\right)=\frac{\frac{2+1}{9+3}\times\frac{0+1}{9+4}\times\frac{10}{17}}{\frac{2+1}{9+3}\times\frac{0+1}{9+4}\times\frac{10}{17}+\frac{3+1}{6+3}\times\frac{0+1}{6+4}\times\frac{7}{17}}=0.382003$$

$$p\left(y=-1|\boldsymbol{x}^{(2)}\right)=\frac{\frac{3+1}{6+3}\times\frac{0+1}{6+4}\times\frac{7}{17}}{\frac{2+1}{9+3}\times\frac{0+1}{9+4}\times\frac{10}{17}+\frac{3+1}{6+3}\times\frac{0+1}{6+4}\times\frac{7}{17}}=0.617997$$

所以样本 $\boldsymbol{x}^{(2)}=(1,N)^{\mathrm{T}}$ 的标签预测为 $y=-1$. 上述算法可以用 Python 代码实现, 不妨将 S,M,L,N 分别记为 $1,2,3,4$, 具体代码如下:

```python
import numpy as np
from sklearn.naive_bayes import MultinomialNB
X = np.array([[1, 1], [1, 2], [1, 2], [1, 1], [1, 1], [2, 1], [2, 2], [2, 2],
              [2, 3], [2, 3], [3, 3], [3, 2], [3, 2], [3, 3], [3, 3]])
# S,M,L,N 分别用数字 1,2,3,4 替代
Y = np.array([-1, -1, 1, 1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, -1])
clf = MultinomialNB(alpha = 1.0, fit_prior = True, class_prior = None)
clf.fit(X, Y)
clf.predict_proba([[2, 1], [1, 4]]) # 输出 (2,S) 和 (1,N) 划分到各个类别的概率值
```

经过上述代码可得: 对于样本 $\boldsymbol{x}^{(1)}=(2,S)^{\mathrm{T}}$, 预测概率为 $p\left(y=1|\boldsymbol{x}^{(1)}\right)=0.406$, $p\left(y=-1|\boldsymbol{x}^{(1)}\right)=0.594$, 因此标签为 $y=-1$; 对于样本 $\boldsymbol{x}^{(2)}=(1,N)^{\mathrm{T}}$, 预测概率为 $p\left(y=1|\boldsymbol{x}^{(2)}\right)=0.384$, $p\left(y=-1|\boldsymbol{x}^{(2)}\right)=0.616$, 故标签为 $y=-1$. 且经过 MultinomialNB 后所习得的 (平滑) 先验概率的对数为 $-0.511,-0.916$.

上述算法也可以用 Matlab 代码实现, 具体代码见下所示:

```
n=input('请输入训练集的个数:n\n');
k=2; A=zeros(n,3);% 存储训练集, 用来学习
A(:,1)=input('请输入所有样本的第一个特征 (用列向量表示): \n');
A(:,2)=input('请输入所有样本的第二个特征 (用列向量表示): \n');
A(:,3)=input('请输入所有样本所属的类 (用列向量表示): \n');
x1=input('请输入需要预测的样本的第一个特征: 1 或 2 或 3 \n');
x2=input('请输入需要预测的样本的第二个特征: S 或 M 或 L 或 N\n','s');
% 计算其属于 1 类的概率
F1=zeros(1,3);
for i=1:n
    if A(i,3)==1
        F1(1,1)=F1(1,1)+1;
        if A(i,1)==x1
            F1(1,2)=F1(1,2)+1;
        end
        if A(i,2)==x2
            F1(1,3)=F1(1,3)+1;
        end
    end
end
C1=(F1(1,1)+1)/(n+k)*(F1(1,2)+1)/(F1(1,1)+3)*(F1(1,3)+1)/(F1(1,1)+4);
% 属于 2 类的概率
F2=zeros(1,3);
for i=1:n
    if A(i,3)==-1
        F2(1,1)=F2(1,1)+1;
        if A(i,1)==x1
            F2(1,2)=F2(1,2)+1;
        end
        if A(i,2)==x2
            F2(1,3)=F2(1,3)+1;
        end
    end
end
C2=(F2(1,1)+1)/(n+k)*(F2(1,2)+1)/(F2(1,1)+3)*(F2(1,3)+1)/(F2(1,1)+4);
if C1>C2
    disp('该样本的类为 1');
else if C1==C2
        disp('该样本的类为-1 或 1')
    else
        disp('该样本的类为-1')
    end
end
```

在命令行窗口的输出过程为:

```
>> MultinomialNB
请输入训练集的个数:n
15
请输入所有样本的第一个特征 (用列向量表示):
[1,1,1,1,1,2,2,2,2,2,3,3,3,3,3]
请输入所有样本的第二个特征(用列向量表示):
['S','M','M','S','S','S','M','M','L','L','L','M','M','L','L']
请输入所有样本所属的类(用列向量表示):
[-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1]
请输入需要预测的样本的第一个特征: 1 或 2 或 3
1
请输入需要预测的样本的第二个特征: S 或 M 或 L 或 N
N
该样本的类为-1
```

```
>> MultinomialNB
请输入训练集的个数:n
15
请输入所有样本的第一个特征 (用列向量表示):
[1,1,1,1,1,2,2,2,2,2,3,3,3,3,3]
请输入所有样本的第二个特征(用列向量表示):
['S','M','M','S','S','S','M','M','L','L','L','M','M','L','L']
请输入所有样本所属的类(用列向量表示):
[-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1]
请输入需要预测的样本的第一个特征: 1 或 2 或 3
2
请输入需要预测的样本的第二个特征: S 或 M 或 L 或 N
S
该样本的类为-1
```

通过 Matlab 右侧的变量值栏目可以计算出: 测试样本 $\boldsymbol{x}^{(1)} = (2, S)^{\mathrm{T}}$ 的预测概率为 $p\left(y = 1|\boldsymbol{x}^{(1)}\right) = 0.3546$, $p\left(y = -1|\boldsymbol{x}^{(1)}\right) = 0.6454$, 因此标签为 $y = -1$. $\boldsymbol{x}^{(2)} = (1, N)^{\mathrm{T}}$ 的预测概率为 $p\left(y = 1|\boldsymbol{x}^{(2)}\right) = 0.3820$, $p\left(y = -1|\boldsymbol{x}^{(1)}\right) = 0.6180$, 因此标签为 $y = -1$.

Problem 3

生成式判别模型: 高斯贝叶斯分类器和逻辑回归

Part A

考虑一类特定的高斯朴素贝叶斯分类器, 其中

- Y 是服从伯努利分布的布尔变量, 其中参数 $\pi = P(Y = 1), P(Y = 0) = 1 - \pi$;
- $X = [x_1, \cdots, x_D]^{\mathrm{T}}$, 其中特征分量 x_i 是连续的随机变量. 对于每个 x_i, $P(x_i|Y = k)$ 是高斯分布 $\mathcal{N}(\mu_{ik}, \sigma_i)$. 注意到 σ_i 是高斯分布的标准差 (且不依赖于 k);
- 给定 Y 时, $\forall i \neq j, x_i$ 和 x_j 都是条件独立的 (因此称之为朴素分类器).

问题: 请证明判别式分类器 (如逻辑回归) 与上述特定类别的高斯朴素贝叶斯分类器之间的关系正是逻辑回归所使用的形式.

Solution: 先利用贝叶斯公式:

$$\begin{aligned} P\left(Y = 1|X\right) &= \frac{P\left(X|Y = 1\right) \cdot P\left(Y = 1\right)}{P\left(X|Y = 0\right) \cdot P\left(Y = 0\right) + P\left(X|Y = 1\right) \cdot P\left(Y = 1\right)} \\ &= \frac{1}{1 + \dfrac{P\left(X|Y = 0\right) \cdot P\left(Y = 0\right)}{P\left(X|Y = 1\right) \cdot P\left(Y = 1\right)}} \\ &= \frac{1}{1 + \exp\left(\ln \dfrac{P\left(X|Y = 0\right) \cdot P\left(Y = 0\right)}{P\left(X|Y = 1\right) \cdot P\left(Y = 1\right)}\right)} \end{aligned}$$

再代入 $P(Y = 1) = \pi, P(Y = 0) = 1 - \pi$, 同时将 ln 中的乘法变为加法:

$$P\left(Y = 1|X\right) = \frac{1}{1 + \exp\left(\ln \dfrac{P\left(X|Y = 0\right)}{P\left(X|Y = 1\right)} + \ln \dfrac{P\left(Y = 0\right)}{P\left(Y = 1\right)}\right)} = \frac{1}{1 + \exp\left(\ln \dfrac{P\left(X|Y = 0\right)}{P\left(X|Y = 1\right)} + \ln \dfrac{1 - \pi}{\pi}\right)}$$

因为 X 是 D 维的, 同时每一个特征都是相互条件独立的, 因此 $P\left(X|Y\right) = \prod_{i=1}^{D} P\left(x_i|Y\right)$, 于是有

$$P\left(Y = 1|X\right) = \frac{1}{1 + \exp\left(\ln \dfrac{\prod_{i=1}^{D} P\left(x_i|Y = 0\right)}{\prod_{i=1}^{D} P\left(x_i|Y = 1\right)} + \ln \dfrac{1 - \pi}{\pi}\right)}$$

由于 $\forall x_i$, $P(x_i|Y = k)$ 都服从高斯分布 $\mathcal{N}(\mu_{ik}, \sigma_i)$, 即 $P\left(x_i|Y = k\right) = \dfrac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\dfrac{(x_i - \mu_{ik})^2}{2\sigma_i^2}\right)$. 我

们先看 $\ln\dfrac{\prod_{i=1}^{D} P\left(x_i|Y=0\right)}{\prod_{i=1}^{D} P\left(x_i|Y=1\right)}$:

$$\begin{aligned}
\ln\frac{\prod_{i=1}^{D} P\left(x_i|Y=0\right)}{\prod_{i=1}^{D} P\left(x_i|Y=1\right)} &= \ln\prod_{i=1}^{D}\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}\right) - \ln\prod_{i=1}^{D}\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{(x_i-\mu_{i1})^2}{2\sigma_i^2}\right) \\
&= \sum_{i=1}^{D}\left(\ln\frac{1}{\sqrt{2\pi}\sigma_i} - \frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}\right) - \sum_{i=1}^{D}\left(\ln\frac{1}{\sqrt{2\pi}\sigma_i} - \frac{(x_i-\mu_{i1})^2}{2\sigma_i^2}\right) \\
&= \sum_{i=1}^{D}\left(\frac{(x_i-\mu_{i1})^2}{2\sigma_i^2} - \frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}\right) = \sum_{i=1}^{D}\left(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}\right)
\end{aligned}$$

再将其代入到 $P(Y=1|X)$ 的表达式中:

$$\begin{aligned}
P\left(Y=1|X\right) &= \frac{1}{1+\exp\left\{\sum_{i=1}^{D}\left(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}\right) + \ln\frac{1-\pi}{\pi}\right\}} \\
&= \frac{1}{1+\exp\left\{-\left[\sum_{i=1}^{D}\underbrace{\frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}}_{\text{即}w_i}x_i + \underbrace{\left(\sum_{i=1}^{D}\frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2} + \ln\frac{\pi}{1-\pi}\right)}_{\text{即}b}\right]\right\}} \\
&= \frac{1}{1+\exp\left[-\left(\sum_{i=1}^{D} w_i x_i + b\right)\right]}
\end{aligned}$$

同理可得:

$$P\left(Y=0|X=1\right) = 1 - P\left(Y=1|X\right) = \frac{\exp\left[-\left(\sum_{i=1}^{D} w_i x_i + b\right)\right]}{1+\exp\left[-\left(\sum_{i=1}^{D} w_i x_i + b\right)\right]}$$

也就是说, 上述特定类别的高斯朴素贝叶斯分类器其实正好就是逻辑回归的形式! 逻辑回归的参数 w 通常采用 SGD 算法去学习, 而而高斯朴素贝叶斯分类器根据假设直接给出了这些参数, 这也说明了逻辑回归是一种更强的模型, 因为它的假设很弱, 而朴素贝叶斯的假设非常的强.

Part B

一般的高斯朴素贝叶斯分类器和逻辑回归: 将“$P(x_i|Y=k)$ 的标准差 σ_i 不依赖于 k”的假设删除掉. 即 $\forall x_i, P(x_i|Y=k)$ 是高斯分布 $\mathcal{N}(\mu_{ik}, \sigma_{ik})$, 其中 $i = 1, \cdots, D$ 且 $k = 0, 1$.

问题: 更一般的高斯朴素贝叶斯分类器中 $P(Y|X)$ 的表达式是否仍然形如逻辑回归的形式. 写出 $P(Y|X)$ 的新格式来证明你的答案.

Solution: 其实就是去掉高斯分布中的方差与其所属类别无关的假设, 即 $P(x_i|Y=k)$ 服从高斯分布 $\mathcal{N}(\mu_{ik}, \sigma_{ik})$. 前面的推导与上面一致, 即

$$P(Y=1|X) = \frac{1}{1+\exp\left(\ln\dfrac{\prod_{i=1}^{D} P(x_i|Y=0)}{\prod_{i=1}^{D} P(x_i|Y=1)} + \ln\dfrac{1-\pi}{\pi}\right)}$$

由于高斯分布假设更改, 即 $P(x_i|Y=k) = \dfrac{1}{\sqrt{2\pi}\sigma_{ik}}\exp\left(-\dfrac{(x_i-\mu_{ik})^2}{2\sigma_{ik}^2}\right)$, 将其代入 $\ln\dfrac{\prod_{i=1}^{D} P(x_i|Y=0)}{\prod_{i=1}^{D} P(x_i|Y=1)}$ 得到:

$$\begin{aligned}
\ln\frac{\prod_{i=1}^{D} P(x_i|Y=0)}{\prod_{i=1}^{D} P(x_i|Y=1)} &= \ln\prod_{i=1}^{D}\frac{1}{\sqrt{2\pi}\sigma_{i0}}\exp\left(-\frac{(x_i-\mu_{i0})^2}{2\sigma_{i0}^2}\right) - \ln\prod_{i=1}^{D}\frac{1}{\sqrt{2\pi}\sigma_{i1}}\exp\left(-\frac{(x_i-\mu_{i1})^2}{2\sigma_{i1}^2}\right) \\
&= \sum_{i=1}^{D}\left[\ln\frac{1}{\sqrt{2\pi}\sigma_{i0}} - \frac{(x_i-\mu_{i0})^2}{2\sigma_{i0}^2}\right] - \sum_{i=1}^{D}\left[\ln\frac{1}{\sqrt{2\pi}\sigma_{i1}} - \frac{(x_i-\mu_{i1})^2}{2\sigma_{i1}^2}\right] \\
&= \sum_{i=1}^{D}\left[\left(\frac{1}{2\sigma_{i1}^2} - \frac{1}{2\sigma_{i0}^2}\right)x_i^2 + \left(\frac{\mu_{i0}}{\sigma_{i0}^2} - \frac{\mu_{i1}}{\sigma_{i1}^2}\right)x_i + \left(\frac{\mu_{i1}^2}{2\sigma_{i1}^2} - \frac{\mu_{i0}^2}{2\sigma_{i0}^2}\right) + \ln\frac{\sigma_{i1}}{\sigma_{i0}}\right]
\end{aligned}$$

将上式结果代入 $P(Y=1|X)$ 得到:

$$P(Y=1|X) = \frac{1}{1+\exp\left(\sum_{i=1}^{D}\left[\underbrace{\left(\frac{1}{2\sigma_{i1}^2} - \frac{1}{2\sigma_{i0}^2}\right)}_{\text{二次项非零}}x_i^2 + \left(\frac{\mu_{i0}}{\sigma_{i0}^2} - \frac{\mu_{i1}}{\sigma_{i1}^2}\right)x_i + \left(\frac{\mu_{i1}^2}{2\sigma_{i1}^2} - \frac{\mu_{i0}^2}{2\sigma_{i0}^2}\right) + \ln\frac{\sigma_{i1}}{\sigma_{i0}}\right] + \ln\frac{1-\pi}{\pi}\right)}$$

显然上述形式中的二次项系数非零, 即去掉了高斯分布中方差与类别无关的假设后朴素贝叶斯变成了非线性的方式. 但是逻辑回归中没有二次项, **所以这种假设条件下的高斯朴素贝叶斯分类器无法直接用 Logistic 回归表示**.

Part C

高斯贝叶斯分类器和逻辑回归: 现在考虑下述假设中的高斯贝叶斯分类器 (不带有“朴素”):

- Y 是服从伯努利分布的布尔变量, 其中参数 $\pi = P(Y = 1), P(Y = 0) = 1 - \pi$;
- $X = [x_1, x_2]^{\mathrm{T}}$, 即我们只考虑样本的特征维数是 2 的情况, 并且每个特征分量都是一个连续的随机变量. 给定 y 时, x_1, x_2 不再是条件独立的了. 我们假设 $P(x_1, x_2|Y = k)$ 是双变量高斯分布 $\mathcal{N}(\mu_{1k}, \mu_{2k}, \sigma_1, \sigma_2, \rho)$, 其中 μ_{1k}, μ_{2k} 分别是 x_1, x_2 的均值, σ_1, σ_2 分别是 x_1, x_2 的标准差, 并且 ρ 是 x_1, x_2 的相关系数. 因此双变量高斯分布的密度函数为

$$P\left(x_1, x_2|Y = k\right) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}\exp\left\{\frac{1}{1-\rho^2}\left[-\frac{\left(x_1-\mu_{1k}\right)^2}{2\sigma_1^2} + \frac{\rho}{\sigma_1\sigma_2}\left(x_1-\mu_{1k}\right)\left(x_2-\mu_{2k}\right) - \frac{\left(x_2-\mu_{2k}\right)^2}{2\sigma_2^2}\right]\right\}$$

问题: 不朴素的高斯贝叶斯分类器中 $P(Y|X)$ 的表达式是否仍然形如逻辑回归的形式. 写出 $P(Y|X)$ 的新格式来证明你的答案.

Solution: 由于 $X = [x_1, x_2]^{\mathrm{T}}$, 直接使用 **Part A** 中推导的公式

$$P\left(Y = 1|X\right) = \frac{1}{1 + \exp\left(\ln\frac{P\left(X|Y=0\right)}{P\left(X|Y=1\right)} + \ln\frac{1-\pi}{\pi}\right)}$$

将题中的二元正态分布的密度函数表达式代入 $P(Y = 1|X)$, 此时我们不妨先看 $\ln P\left(X|Y = k\right)$:

$$\ln P\left(X|Y = k\right) = \ln\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} + \frac{1}{1-\rho^2}\left[-\frac{\left(x_1-\mu_{1k}\right)^2}{2\sigma_1^2} + \frac{\rho}{\sigma_1\sigma_2}\left(x_1-\mu_{1k}\right)\left(x_2-\mu_{2k}\right) - \frac{\left(x_2-\mu_{2k}\right)^2}{2\sigma_2^2}\right]$$

于是经过**精心整理**可得:

$$\begin{aligned}
\ln\frac{P\left(X|Y=0\right)}{P\left(X|Y=1\right)} &= \ln P\left(X|Y=0\right) - \ln P\left(X|Y=1\right) \\
&= \frac{1}{1-\rho^2}\left\{\left[\frac{\mu_{10}-\mu_{11}}{\sigma_1^2} + \frac{\rho\left(\mu_{21}-\mu_{20}\right)}{\sigma_1\sigma_2}\right]x_1 + \left[\frac{\mu_{20}-\mu_{21}}{\sigma_2^2} + \frac{\rho\left(\mu_{11}-\mu_{10}\right)}{\sigma_1\sigma_2}\right]x_2\right. \\
&\quad \left. + \left[\frac{\mu_{11}^2-\mu_{10}^2}{2\sigma_1^2} + \frac{\mu_{21}^2-\mu_{20}^2}{2\sigma_2^2} + \frac{\rho\left(\mu_{10}\mu_{20}-\mu_{11}\mu_{21}\right)}{\sigma_1\sigma_2}\right]\right\}
\end{aligned}$$

将上述结果代入到 $P(Y = 1|X)$, 可将 $P(Y = 1|X)$ 表示为逻辑回归的形式:

$$P\left(Y = 1|X\right) = \frac{1}{1 + \exp\left[-\left(\sum_{i=1}^{2} w_i x_i + b\right)\right]}$$

其中

$$\begin{aligned}
w_1 &= \frac{-1}{1-\rho^2}\left[\frac{\mu_{10}-\mu_{11}}{\sigma_1^2} + \frac{\rho\left(\mu_{21}-\mu_{20}\right)}{\sigma_1\sigma_2}\right] \\
w_2 &= \frac{-1}{1-\rho^2}\left[\frac{\mu_{20}-\mu_{21}}{\sigma_2^2} + \frac{\rho\left(\mu_{11}-\mu_{10}\right)}{\sigma_1\sigma_2}\right] \\
b &= \frac{-1}{1-\rho^2}\left[\frac{\mu_{11}^2-\mu_{10}^2}{2\sigma_1^2} + \frac{\mu_{21}^2-\mu_{20}^2}{2\sigma_2^2} + \frac{\rho\left(\mu_{10}\mu_{20}-\mu_{11}\mu_{21}\right)}{\sigma_1\sigma_2}\right] + \ln\frac{1-\pi}{\pi}
\end{aligned}$$

因此可以知道**去掉“naive”的高斯贝叶斯分类器依然是线性的, 可以用 Logistic 回归表示**.

通过上述推导, 分别去掉了两个不同的假设来揭开高斯朴素贝叶斯分类器的面纱, 最后简单总结为:

- **传统的高斯朴素贝叶斯等价于通过假设直接从训练数据中算出参数的逻辑回归;**
- **去掉高斯分布中方差与类别无关的假设后, 高斯朴素贝叶斯将变成非线性分类器, 其中多了二次项特征;**
- **移除特征之间的条件独立性假设后, 高斯贝叶斯依然是线性的, 但是其联合概率密度计算会比较复杂.**

Problem 4

Logistic 回归 (LR) 的 MLE 参数估计: 如果有参数的正则项 (用以抑制过拟合), 那么该如何估计参数? 其中

$$l(\boldsymbol{w}) = \log L(\boldsymbol{w}) - \lambda \|\boldsymbol{w}\|_2^2 = \sum_{i=1}^{N} y^{(i)} \log f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) - \lambda \|\boldsymbol{w}\|_2^2$$

Solution: 易知

$$l(\boldsymbol{w}) = \sum_{i=1}^{N} y^{(i)} \log f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) - \lambda \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}$$

故可以如下求得梯度:

$$\frac{\partial l(\boldsymbol{w})}{\partial w_j} = \sum_{i=1}^{N} \left[y^{(i)} \frac{1}{f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)} \cdot \frac{\partial f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}{\partial w_j} + \left(1 - y^{(i)}\right) \frac{1}{1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)} \cdot \frac{-\partial\left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right)}{\partial w_j} \right] - 2\lambda w_j$$

又因为

$$\frac{\partial f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}{\partial w_j} = \underbrace{\frac{\partial g\left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)}\right)}{\partial w_j} = g\left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)}\right)\left(1 - g\left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)}\right)\right) \boldsymbol{x}_j^{(i)}}_{g(z)=\frac{1}{1+\exp(-z)} \Rightarrow g'(z)=g(z)(1-g(z))} = f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\left(1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) \boldsymbol{x}_j^{(i)}$$

于是得到梯度表达式:

$$\begin{aligned}
\frac{\partial l(\boldsymbol{w})}{\partial w_j} &= \sum_{i=1}^{N} \left\{ y^{(i)} \cdot \left[\left(1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) \boldsymbol{x}_j^{(i)} \right] + \left(1 - y^{(i)}\right)(-1) \cdot \left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) \boldsymbol{x}_j^{(i)} \right) \right\} - 2\lambda w_j \\
&= \sum_{i=1}^{N} \left[y^{(i)} \cdot \left(1 - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) \cdot \boldsymbol{x}_j^{(i)} + \left(y^{(i)} - 1\right) \cdot f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) \cdot \boldsymbol{x}_j^{(i)} \right] - 2\lambda w_j \\
&= \sum_{i=1}^{N} \left[y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) \right] \boldsymbol{x}_j^{(i)} - 2\lambda w_j
\end{aligned}$$

为了得到 MLE 的参数估计, 我们需要使用 SGD 算法来得到 $\boldsymbol{w}$, 因此 SGD 的梯度更新规则如下:

$$w_j \leftarrow w_j + \alpha \left\{ \left[y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) \right] \boldsymbol{x}_j^{(i)} - 2\lambda w_j \right\} = (1 - 2\lambda\alpha) w_j + \alpha \left[y^{(i)} - f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right) \right] \boldsymbol{x}_j^{(i)}$$

其中 α 为步长; λ 为超参数, 需要手动调整来观察过拟合的抑制情况.

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 7 课程作业解答

2022 年 11 月 10 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

给定如下训练数据集

$$\boldsymbol{X}=\begin{pmatrix}3 & 4 & 1\\ 3 & 3 & 1\end{pmatrix}, \boldsymbol{y}=\begin{pmatrix}1 & 1 & -1\end{pmatrix}^{\mathrm{T}}$$

通过求解 SVM 的**原始问题**来求解最大间隔的分离超平面.

Solution: 通过求解如下最优化问题来得到最优分类器的参数 $(\boldsymbol{w}^*, b^*)$:

$$\begin{cases}\min\limits_{\boldsymbol{w},b} \dfrac{1}{2}\|\boldsymbol{w}\|_2^2\\ \text{s.t. } y^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}+b\right)\geq 1, i=1,\cdots,N\end{cases} \Rightarrow \begin{cases}\min\limits_{(w_1,w_2),b} \dfrac{1}{2}\left(w_1^2+w_2^2\right)\\ 3w_1+3w_2+b\geq 1\\ 4w_1+3w_2+b\geq 1\\ -\left(w_1+w_2+b\right)\geq 1\end{cases}$$

要求求解这个严格的凸二次规划问题, 我们可以直接利用 Matlab 中的二次规划问题求解包直接求解. 而二次规划的标准形式为

$$\begin{cases}\min\limits_{\boldsymbol{x}} \dfrac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{f}^{\mathrm{T}}\boldsymbol{x}\\ \boldsymbol{A}\cdot\boldsymbol{x}\leq\boldsymbol{b}\end{cases}$$

对照过来, 可以知道

$$\boldsymbol{H}=\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{pmatrix}, \boldsymbol{f}=\begin{pmatrix}0\\ 0\\ 0\end{pmatrix}, \boldsymbol{A}=\begin{pmatrix}-3 & -3 & -1\\ -4 & -3 & -1\\ 1 & 1 & 1\end{pmatrix}, \boldsymbol{b}=\begin{pmatrix}-1\\ -1\\ -1\end{pmatrix}$$

于是我们可以在 Matlab 中写出如下代码:

```
H = [1, 0, 0; 0, 1, 0; 0, 0, 0];
f = [0; 0; 0];
A = [-3, -3, -1; -4, -3, -1; 1, 1, 1];
b = [-1; -1; -1];
[x, fval, exitflag, output, lambda] = quadprog(H, f, A, b)
```

执行后便可得到结果 (exitflag=1, 即问题存在最优解):

$$\boldsymbol{x}^*=\begin{pmatrix}w_1^*\\ w_2^*\\ b^*\end{pmatrix}=\begin{pmatrix}0.5\\ 0.5\\ -2\end{pmatrix}, \min_{(w_1,w_2,b)} \frac{1}{2}\left(w_1^2+w_2^2\right)=0.25$$

于是最大间隔的分离超平面方程和判别函数分别为

$$\frac{1}{2}x_1+\frac{1}{2}x_2-2=0, f_{\boldsymbol{w},b}(\boldsymbol{x})=\operatorname{sgn}\left(\frac{1}{2}x_1+\frac{1}{2}x_2-2\right)$$

本题的训练样本较少, 所以方法 2 是说: 可以直接观察出支持向量 (而不用求解上述二次规划问题), 从而立即得到具有最大间隔的分离超平面[1]. 具体如下图1所示:

[1] 这是因为最终模型仅与**支持向量**有关.

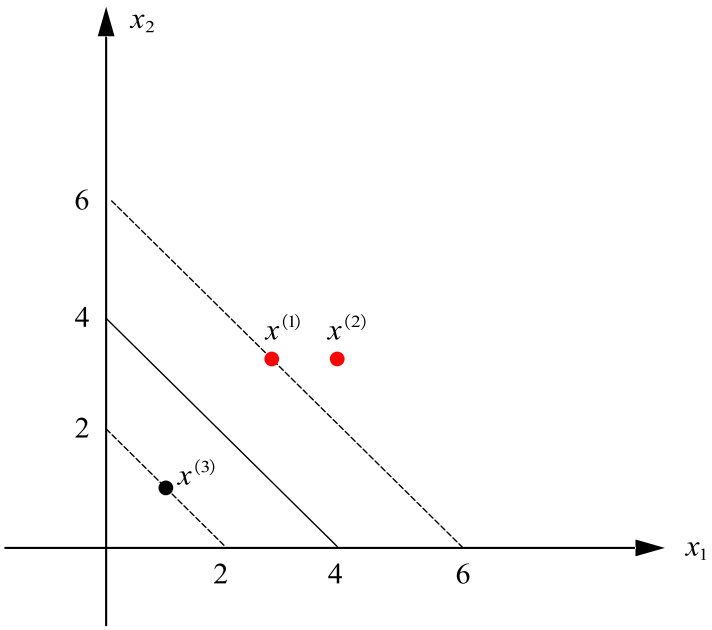


图 1: 最大间隔分离超平面示意图

明显可以看出支持向量为 $x^{(3)}, x^{(1)}$, 所以**这两个点的垂直平分线就是所要求的最大间隔分离超平面**! 即其方程为 $0.5x_1 + 0.5x_2 - 2 = 0$. 而对于上述二次优化问题, 我们也可以利用拉格朗日数乘法求解如下: 构造辅助函数 (只代入支持向量, 因为最终模型只与支持向量有关)

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\left(w_1^2 + w_2^2\right) + \alpha_1(1 - 3w_1 - 3w_2 - b) + \alpha_2(1 + w_1 + w_2 + b)$$

对 w_1, w_2, b 分别求偏导:

$$\begin{cases} w_1 - 3\alpha_1 + \alpha_2 = 0 \\ w_2 - 3\alpha_1 + \alpha_2 = 0 \\ -\alpha_1 + \alpha_2 = 0 \end{cases} \Rightarrow w_1 = w_2 = 2\alpha_1$$

代入辅助函数可得:

$$L(\alpha_1) = -4\alpha_1^2 + 2\alpha_1, L'(\alpha_1) = -8\alpha_1 + 2$$

令 $L'(\alpha_1) = 0$ 得 $\alpha_1 = \frac{1}{4}$, 因此 $w_1 = w_2 = \frac{1}{2}$, 从而 $b \leq -2$, 于是最大间隔的分离超平面方程为

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 - 2 = 0$$

Problem 2

给定如下训练数据集

$$\boldsymbol{X} = \begin{pmatrix} 3 & 4 & 1 \\ 3 & 3 & 1 \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} 1 & 1 & -1 \end{pmatrix}^{\mathrm{T}}$$

通过求解 SVM 的**对偶问题**来求解最大间隔的分离超平面.

Solution: SVM 的对偶问题为

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \left\{ \sum\limits_{i=1}^{N} \alpha_i - \dfrac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle \right\} \\ \text{s.t. } \alpha_i \geq 0, i = 1, \cdots, N \\ \sum\limits_{i=1}^{N} \alpha_i y^{(i)} = 0 \end{cases}$$

代入上述训练样本具体化对偶问题可得

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \left\{ \sum\limits_{i=1}^{3} \alpha_i - \dfrac{1}{2} \left(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 21\alpha_1\alpha_2 - 6\alpha_1\alpha_3 - 7\alpha_2\alpha_3 \right) \right\} \\ \alpha_i \geq 0, i = 1, 2, 3 \\ \alpha_1 + \alpha_2 - \alpha_3 = 0 \end{cases}$$

而求解这个二次规划问题, 我们可以利用 Matlab 中的求解包来求解. 而二次规划的标准形式为

$$\begin{cases} \min\limits_{\boldsymbol{x}} \dfrac{1}{2} \boldsymbol{x}^{\mathrm{T}} \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}} \boldsymbol{x} \\ \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{A}_{eq} \cdot \boldsymbol{x} = \boldsymbol{b}_{eq} \end{cases}$$

对照过来可以知道:

$$\boldsymbol{H} = \begin{pmatrix} 18 & 10.5 & -3 \\ 10.5 & 25 & -3.5 \\ -3 & -3.5 & 2 \end{pmatrix}, \boldsymbol{f} = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \boldsymbol{A}_{eq} = \begin{pmatrix} 1 & 1 & -1 \end{pmatrix}, \boldsymbol{b}_{eq} = (0)$$

于是我们可以在 Matlab 中写出如下代码:

```
H = [18, 10.5, -3; 10.5, 25, -3.5; -3, -3.5, 2];
f = [-1; -1; -1];
A = [-1, 0, 0; 0, -1, 0; 0, 0, -1];
b = [0; 0; 0];
A_eq = [1, 1, -1];
b_eq = [0];
[x, fval, exitflag, output, lambda] = quadprog(H, f, A, b, A_eq, b_eq)
```

根据代码输出就可以得到对偶问题的最优解 $\boldsymbol{\alpha}^*$ 和相应的最优值分别为

$$\boldsymbol{\alpha}^* = \begin{pmatrix} 0.25 \\ 0 \\ 0.25 \end{pmatrix}, \max_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^{3} \alpha_i - \frac{1}{2} \left(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 21\alpha_1\alpha_2 - 6\alpha_1\alpha_3 - 7\alpha_2\alpha_3 \right) \right\} = 0.0625$$

于是原问题的最优解 $(\boldsymbol{w}^*, b^*)$ 为

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)} \boldsymbol{x}^{(i)} = 0.25 \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix} + 0 \cdot \begin{pmatrix} 4 \\ 3 \end{pmatrix} - 0.25 \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$b^* = y^{(j)} - \sum_{i=1}^{N} \alpha_i^* y^{(i)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle, \alpha_j^* > 0 \Rightarrow b^* = y^{(1)} - \sum_{i=1}^{3} \alpha_i^* y^{(i)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(1)} \right\rangle = -2$$

于是分离超平面方程和判别函数的表达式分别为

$$\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{x} + b^* = 0.5 \left(x_1 + x_3 \right) - 2 = 0, f_{\boldsymbol{w},b} \left(\boldsymbol{x} \right) = \operatorname{sgn} \left(\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{x} + b^* \right)$$

对于上面的对偶问题, 我们也可以解析求解: 根据约束条件可知 $\alpha_3 = \alpha_1 + \alpha_2$, 代入目标函数可得

$$\theta_D \left(\alpha_1, \alpha_2 \right) = -4\alpha_1^2 - 10\alpha_1\alpha_2 - 3\alpha_2^2 + 2\alpha_1 + 2\alpha_2$$

于是

$$\frac{\partial \theta_D \left(\alpha_1, \alpha_2 \right)}{\partial \alpha_1} = -8\alpha_1 - 10\alpha_2 + 2 = 0 \Rightarrow \alpha_1 = \frac{1}{4} - \frac{5\alpha_2}{4}$$

代入 $\theta_D(\alpha_1, \alpha_2)$ 可得

$$\theta_D \left(\alpha_2 \right) = -\frac{1}{4}\alpha_2^2 - \frac{1}{2}\alpha_2 + \frac{1}{4} \xrightarrow{\alpha_2 \geq 0} \max_{\alpha_2 \geq 0} \theta_D = \theta_D \left(0 \right) = \frac{1}{4} \Rightarrow a_2 = 0, \alpha_1 = \alpha_3 = \frac{1}{4}$$

对于原问题最优解的推导与上面的过程一致, 就不赘述了. 我们也可以利用下述的算法1(即 SMO 算法)来求出 SVM 的对偶问题的解为 $\boldsymbol{\alpha}^* = (0.25, 0, 0.25)^{\mathrm{T}}$:

Algorithm 1 SMO 算法

Input: 训练数据集 $S = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right), i = 1, \cdots, N \right\}$, 误差 ϵ

Output: $\hat{\alpha} = (\hat{\alpha}_1, \cdots, \hat{\alpha}_N)$

1: 初始化: $\alpha^{(0)} \leftarrow 0, k \leftarrow 0, C \leftarrow +\infty$, 并计算偏移量 $b^{(0)}$;

2: 初始化误差项: $E_i \leftarrow g\left(\boldsymbol{x}^{(i)} \right) - y^{(i)}$;

3: **while** 不满足 KKT 条件且 $\exists i \in \{1, 2, \cdots, N\}$, s.t. $E_i \geq \epsilon$ **do**

4: $k \leftarrow k + 1$ 并选择待优化的变量: $\alpha_1^{(k)}, \alpha_2^{(k)}$, 然后求解优化问题的解 $\alpha_1^{(k+1)}, \alpha_2^{(k+1)}$:

$$\alpha_2^{\text{new,unclipped}} \leftarrow \alpha_2^{(k)} + \frac{y^2 \left(E_1 - E_2 \right)}{\eta}, , \ \alpha_2^{(k+1)} \leftarrow \begin{cases} H, & \text{若} \alpha_2^{\text{new,unclipped}} > H \\ \alpha_2^{\text{new,unclipped}}, & \text{若} L \leq \alpha_2^{\text{new,unclipped}} \leq H \\ L, & \text{若} \alpha_2^{\text{new,unclipped}} < L \end{cases}$$

$$\eta \leftarrow K_{11} + K_{22} - 2K_{12}, \ \alpha_1^{(k+1)} \leftarrow \alpha_1^{(k)} + y^{(1)} y^{(2)} \left(\alpha_2^{(k)} - \alpha_2^{(k+1)} \right)$$

5: 更新 $\alpha \leftarrow \alpha^{(k+1)}$, 更新 $E_i \leftarrow g\left(\boldsymbol{x}^{(i)} \right) - y^{(i)}$, 计算 $b^{(k+1)}$;

6: **end while**

7: **end {SMO}**

Problem 3

推导软间隔 SVM 的对偶形式.

Solution: 软间隔分类器 SVM 的原问题如下:

$$\begin{cases} \min\limits_{\boldsymbol{w},b} \dfrac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum\limits_{i=1}^{N}\xi_i \\ y^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}+b\right) \geq 1-\xi_i, i=1,\cdots,N \\ \xi_i \geq 0, i=1,\cdots,N \end{cases}$$

于是我们可以构造如下的广义拉格朗日函数:

$$L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[y^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}+b\right)-1+\xi_i\right] - \sum_{i=1}^{N}\eta_i\xi_i$$

先固定拉格朗日乘子 $\boldsymbol{\alpha},\boldsymbol{\eta}$, 关于 $\boldsymbol{w},b,\xi_i$ 对 $L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right)$ 做优化 (最小化) 得到 $\theta_D(\boldsymbol{\alpha},\boldsymbol{\eta})$:

$$\begin{cases} \nabla_{\boldsymbol{w}} L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right) = \boldsymbol{w} - \sum\limits_{i=1}^{N}\alpha_i y^{(i)}\boldsymbol{x}^{(i)} = 0 \\ \dfrac{\partial}{\partial b} L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right) = -\sum\limits_{i=1}^{N}\alpha_i y^{(i)} = 0 \\ \dfrac{\partial}{\partial \xi_i} L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right) = C-\alpha_i-\eta_i = 0 \end{cases}$$

将上述 3 个条件代入广义拉格朗日函数可得到 $\theta_D(\boldsymbol{\alpha},\boldsymbol{\eta})$:

$$\begin{aligned} \theta_D\left(\boldsymbol{\alpha},\boldsymbol{\eta}\right) &= \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left[y^{(i)}\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}^{(i)}+b\right)-1+\xi_i\right] - \sum_{i=1}^{N}\eta_i\xi_i \\ &= \frac{1}{2}\sum_{i=1}^{N}\alpha_i y^{(i)}\left(\boldsymbol{x}^{(i)}\right)^{\mathrm{T}}\sum_{j=1}^{N}\alpha_j y^{(j)}\boldsymbol{x}^{(j)} - \sum_{i=1}^{N}\alpha_i\left[y^{(i)}\left(\sum_{j=1}^{N}\alpha_j y^{(j)}\left(\boldsymbol{x}^{(j)}\right)^{\mathrm{T}}\boldsymbol{x}^{(i)}+b\right)-1+\xi_i\right] \\ &+ C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\eta_i\xi_i \\ &= \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\left\langle\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\right\rangle - \sum_{i,j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\left\langle\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\right\rangle - b\cdot\sum_{i=1}^{N}\alpha_i y^{(i)} \\ &+ \sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\left(C-\alpha_i-\eta_i\right)\xi_i \\ &= -\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\left\langle\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\right\rangle + \sum_{i=1}^{N}\alpha_i \end{aligned}$$

即得到

$$\theta_D\left(\boldsymbol{\alpha},\boldsymbol{\eta}\right) = \min_{\boldsymbol{w},b,\xi} L\left(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}\right) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\left\langle\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}\right\rangle = \theta_D\left(\boldsymbol{\alpha}\right)$$

于是最大化 $\theta_D(\boldsymbol{\alpha})$, 即可得出下述的对偶问题:

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \left\{ \sum\limits_{i=1}^{N} \alpha_i - \dfrac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle \right\} \\ \sum\limits_{i=1}^{N} \alpha_i y^{(i)} = 0 \\ 0 \leq \alpha_i = \underbrace{C - \eta_i \leq C}_{\because \eta_i \geq 0 (\text{乘子非负})}, i = 1, \cdots, N \end{cases}$$

假设 $\boldsymbol{\alpha}^* = \left(\alpha_1^*, \cdots, \alpha_N^*\right)$ 是上述对偶问题的最优解, 那么原问题的解可以如下求解: 显然 $\boldsymbol{w}^*$ 可以直接写出:

$$\boldsymbol{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)} \boldsymbol{x}^{(i)}$$

而对于 b^*, 需要找到一个相关的等式, 而根据库恩塔克条件 (KKT) 可得

$$\alpha_i \left[y^{(i)} \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)} + b \right) - 1 + \xi_i \right] = 0, \eta_i \xi_i = 0$$

因此若想要得到 b 的等式, 则根据互补松弛性可知, 只需要 $\alpha_i > 0 (\neq 0)$, 但是 ξ_i 又是不能知道的, 所以需要想办法把它去掉 (即需要使得 $\xi_i = 0$), 于是又根据互补松弛性可知, 再需要令 $\eta_i > 0 (\neq 0)$ 即可使得 $\xi_i = 0$, 从而可得出 b 的等式

$$y^{(i)} \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)} + b \right) = 1 \Rightarrow b = y^{(i)} - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(i)}$$

因此若想求得 b^*, 则需要挑选样本 $\left(\boldsymbol{x}^{(j)}, y^{(j)}\right)$, 使得样本满足 $\alpha_j^* > 0, \eta_j > 0$ (即 $0 < \alpha_j^* < C$), 于是可求得

$$b^* = y^{(j)} - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}^{(j)} = y^{(j)} - \sum_{i=1}^{N} \alpha_i^* y^{(i)} \left(\boldsymbol{x}^{(i)} \right)^{\mathrm{T}} \boldsymbol{x}^{(j)} = y^{(j)} - \sum_{i=1}^{N} \alpha_i^* y^{(i)} \left\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \right\rangle$$

于是可求得分类超平面方程和判别函数分别为

$$\left(\boldsymbol{w}^*\right)^{\mathrm{T}} \boldsymbol{x} + b^* = 0, f_{\boldsymbol{w}, b}(\boldsymbol{x}) = \operatorname{sgn}\left\{ \left(\boldsymbol{w}^*\right)^{\mathrm{T}} \boldsymbol{x} + b^* \right\}$$

Problem 4

高斯核的形式如下

$$\mathcal{K}(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}\right)$$

请证明高斯核可以表示为无限维特征向量的内积.

Solution: 可以将高斯核作如下展开:

$$\begin{aligned}\mathcal{K}(\boldsymbol{x},\boldsymbol{z}) &= \exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{z})^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{z})}{2\sigma^2}\right\}\\ &= \exp\left\{-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}-2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}+\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right\}\\ &= \exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)\cdot\exp\left(\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}}{\sigma^2}\right)\cdot\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)\end{aligned}$$

由于指数函数 $y=e^x$ 的泰勒级数在任一点 (实数) 都收敛, 所以我们可以将上式的中间项做泰勒展开:

$$\exp\left(\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}}{\sigma^2}\right) = \sum_{n=0}^{\infty}\frac{\left(\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}}{\sigma^2}\right)^n}{n!} = \sum_{n=0}^{\infty}\frac{\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}\right)^n}{n!\sigma^{2n}}$$

于是核函数可以表示为

$$\mathcal{K}(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)\cdot\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)\cdot\sum_{n=0}^{\infty}\frac{\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}\right)^n}{n!\sigma^{2n}}$$

不妨设 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z} = \sum_{i=1}^{k} x_i z_i$. 为了进行后续推导, 我们需要声明一下**推广的二项式定理**, 即

$$\left(\sum_{i=1}^{k} x_i\right)^n = \sum_{l=1}^{L}\frac{n!}{n_{l_1}!\cdot n_{l_2}!\cdots n_{l_k}!}x_1^{n_{l_1}}\cdot x_2^{n_{l_2}}\cdots x_k^{n_{l_k}}$$

其中 $\sum_{i=1}^{k} n_{l_i} = n, L = \frac{(n+k-1)!}{n!\,(k-1)!}$($L$ 也被称作多项式系数). 于是有:

$$\begin{aligned}\mathcal{K}(\boldsymbol{x},\boldsymbol{z}) &= \exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)\cdot\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)\cdot\sum_{n=0}^{\infty}\frac{\left(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{z}\right)^n}{n!\sigma^{2n}}\\ &= \exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)\cdot\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)\cdot\sum_{n=0}^{\infty}\frac{1}{n!\sigma^{2n}}\sum_{l=1}^{L}\frac{n!}{n_{l_1}!\cdot n_{l_2}!\cdots n_{l_k}!}(x_1z_1)^{n_{l_1}}\cdot(x_2z_2)^{n_{l_2}}\cdots(x_kz_k)^{n_{l_k}}\\ &= \sum_{n=0}^{\infty}\sum_{l=1}^{L}\frac{\exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)\cdot\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)}{\sigma^{2n}\cdot n_{l_1}!\cdot n_{l_2}!\cdots n_{l_k}!}\left(x_1^{n_{l_1}}\cdot x_2^{n_{l_2}}\cdots x_k^{n_{l_k}}\right)\cdot\left(z_1^{n_{l_1}}\cdot z_2^{n_{l_2}}\cdots z_k^{n_{l_k}}\right)\\ &= \sum_{n=0}^{\infty}\sum_{l=1}^{L}\underbrace{\frac{\exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2\sigma^2}\right)}{\sqrt{\sigma^{2n}\cdot n_{l_1}!\cdot n_{l_2}!\cdots n_{l_k}!}}\left(x_1^{n_{l_1}}\cdot x_2^{n_{l_2}}\cdots x_k^{n_{l_k}}\right)}_{:=\varphi_{n_l}(\boldsymbol{x})}\cdot\underbrace{\frac{\exp\left(-\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{2\sigma^2}\right)}{\sqrt{\sigma^{2n}\cdot n_{l_1}!\cdot n_{l_2}!\cdots n_{l_k}!}}\left(z_1^{n_{l_1}}\cdot z_2^{n_{l_2}}\cdots z_k^{n_{l_k}}\right)}_{:=\varphi_{n_l}(\boldsymbol{z})}\\ &= \sum_{n=0}^{\infty}\sum_{l=1}^{L}\varphi_{n_l}(\boldsymbol{x})\cdot\varphi_{n_l}(\boldsymbol{z})\end{aligned}$$

我们令

$$\Phi_n(\boldsymbol{x}) = [\varphi_{n_1}(\boldsymbol{x}), \varphi_{n_2}(\boldsymbol{x}), \cdots, \varphi_{n_L}(\boldsymbol{x})], \Phi_n(\boldsymbol{z}) = [\varphi_{n_1}(\boldsymbol{z}), \varphi_{n_2}(\boldsymbol{z}), \cdots, \varphi_{n_L}(\boldsymbol{z})]$$

再令

$$\mathcal{K}_n(\boldsymbol{x}, \boldsymbol{z}) = \langle \Phi_n(\boldsymbol{x}), \Phi_n(\boldsymbol{z}) \rangle$$

于是有

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{n=0}^{\infty} \sum_{l=1}^{L} \varphi_{n_l}(\boldsymbol{x}) \cdot \varphi_{n_l}(\boldsymbol{z}) = \sum_{n=0}^{\infty} \langle \Phi_n(\boldsymbol{x}), \Phi_n(\boldsymbol{z}) \rangle = \sum_{n=0}^{\infty} \mathcal{K}_n(\boldsymbol{x}, \boldsymbol{z})$$

显然高斯核可以表示为无限维特征向量的内积, 而且可知高斯核也可以表示成无限个核函数的线性组合.

Problem 5

请证明, 无论数据空间的维数如何, 仅由两个数据点 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$(每个类一个样本) 所组成的数据集足以确定最大间隔超平面. 充分解释你的答案, 包括给出硬间隔 SVM(即 $\boldsymbol{w}$) 作为 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}$ 函数的解的明确公式.

Solution: 显然两个样本都是**支持向量**, 所以这两个点的垂直平分超平面即为硬间隔 SVM 的表达式. 不妨设 $\boldsymbol{x}^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)}\right)^{\mathrm{T}}, \boldsymbol{x}^{(2)} = \left(x_1^{(2)}, x_2^{(2)}, \cdots, x_n^{(2)}\right)^{\mathrm{T}}$, 先计算这两点连线的方向向量, 也就是垂直平分超平面的法向量:

$$\boldsymbol{n} = \boldsymbol{w} = \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)} = \left(x_1^{(1)} - x_1^{(2)}, x_2^{(1)} - x_2^{(2)}, \cdots, x_n^{(1)} - x_n^{(2)}\right)^{\mathrm{T}}$$

而垂直平分超平面一定经过这两点连线的中点, 即

$$P_0 = \frac{\boldsymbol{x}^{(1)} + \boldsymbol{x}^{(2)}}{2} = \frac{1}{2}\left(x_1^{(1)} + x_1^{(2)}, x_2^{(1)} + x_2^{(2)}, \cdots, x_n^{(1)} + x_n^{(2)}\right)^{\mathrm{T}}$$

于是根据点法式即可写出如下垂直平分超平面的方程:

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0 = \sum_{i=1}^{n}\left(x_i^{(1)} - x_i^{(2)}\right)x_i + b = 0 \xrightarrow{\text{代入点}P_0} b = -\sum_{i=1}^{n}\frac{1}{2}\left(x_i^{(1)} + x_i^{(2)}\right)\left(x_i^{(1)} - x_i^{(2)}\right)$$

于是最大间隔超平面方程为:

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = \sum_{i=1}^{n}\left(x_i^{(1)} - x_i^{(2)}\right)\left[x_i - \frac{1}{2}\left(x_i^{(1)} + x_i^{(2)}\right)\right] = 0$$

中国科学院大学
University of Chinese Academy of Sciences

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 8 课程作业解答

2022 年 11 月 24 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

下图1给出 6 个数据集 $A-F$ 分别用两种算法得到的聚类结果, 其中一种是 K 均值聚类. 请问哪些最可能是 K 均值聚类的结果? 如果 K 均值聚类结果不够理想, 建议采用哪种聚类算法?

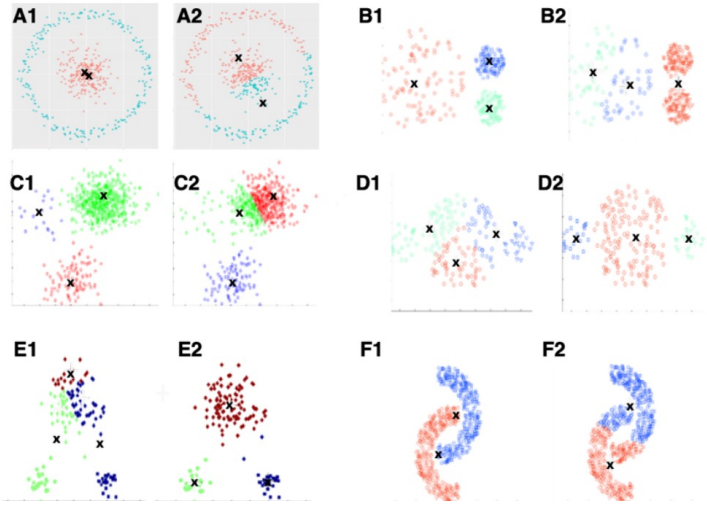


图 1: 聚类结果

Solution:

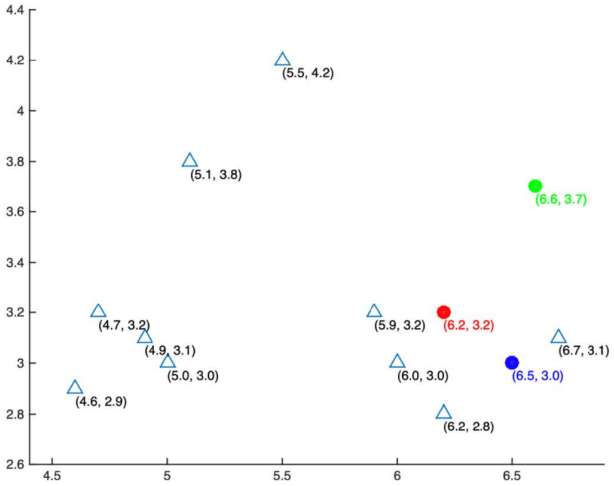
- 对于 A, 其中 A_2 很可能是 K 均值的聚类结果, 因为 A_1 的是非球形簇而 A_2 是球形簇. 由于图中的簇形状较为任意, 所以应该采用 DBSCAN 聚类算法;
- 对于 B, 其中 B_2 最可能是 K 均值的聚类结果, 因为 K 均值聚类会受到不同密度的影响, 故应该采用 DBSCAN 聚类算法;
- 对于 C, 其中 C_2 最可能是 K 均值的聚类结果, 因为 K 均值聚类会受到不同密度的影响, 故应该采用 DBSCAN 聚类算法;
- 对于 D, 其中 D_1 最可能是 K 均值的聚类结果, 因为 K 均值聚类会受到不同尺寸 (即先验概率) 的影响, 而且簇都是球形簇, 故应该采用 GMM-EM 聚类算法;
- 对于 E, 其中 E_1 最可能是 K 均值的聚类结果, 因为 K 均值聚类会受到不同尺寸 (即先验概率) 的影响, 而且簇都是球形簇, 故应该采用 GMM-EM 聚类算法;
- 对于 F, 其中 F_2 很可能是 K 均值的聚类结果 (这是由于 K 均值聚类只能针对球形/凸簇), 因为图中的簇形状较为任意, 所以应该采用 DBSCAN 聚类算法.

Problem 2

对如下图2所示的数据集, 采用 K 均值聚类. 设 $K=3$, 3 个聚类中心分别为

$$\mu_1=(6.2,3.2)^{\mathrm{T}}\ (\text{红色}),\quad \mu_2=(6.6,3.7)^{\mathrm{T}}\ (\text{绿色}),\quad \mu_3=(6.5,3.0)^{\mathrm{T}}\ (\text{蓝色})$$

请给出一次迭代后属于第一簇的样本以及更新后的簇中心 (保留两位小数).



$$\mathbf{X}=\begin{bmatrix}5.9 & 3.2\\ 4.6 & 2.9\\ 6.2 & 2.8\\ 4.7 & 3.2\\ 5.5 & 4.2\\ 5.0 & 3.0\\ 4.9 & 3.1\\ 6.7 & 3.1\\ 5.1 & 3.8\\ 6.0 & 3.0\end{bmatrix}$$

图 2: 聚类数据集

Solution: K-means 聚类算法的流程见如下伪代码1:

Algorithm 1 K-means 算法

Input: 给定 N 个样本点 $\mathcal{D}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$
Output: K 个簇

```
初始化选择 K 个种子数据点 (seeds) 作为 K 个簇的中心;
while 当前簇中心仍在更新 do
    for each x_i ∈ D do
        计算 x_i 与每一个簇中心的距离;
        将 x_i 指配到距离最近的簇中心;
    end for
    用当前簇内点重新计算 K 个簇中心位置;
end while
end {K-means}
```

我们可以写出对应的 C++ 代码来自动求出结果, 对应结果如下:
第一轮迭代后属于第一簇的样本有

$$\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_4,\boldsymbol{x}_6,\boldsymbol{x}_7,\boldsymbol{x}_9,\boldsymbol{x}_{10}$$

且更新后的簇中心坐标为

$$\mu_1=(5.17,3.17)^{\mathrm{T}},\quad \mu_2=(5.50,4.20)^{\mathrm{T}},\quad \mu_3=(6.45,2.95)^{\mathrm{T}}$$

具体的 C++ 代码如下:

```cpp
#include <bits/stdc++.h> //万能头，刷题可以用，大型项目不要用
using namespace std;
double Distan(vector<double> &X, vector<double> &Y) {
    int L = X.size();
    double sum = 0;
    for(int i = 0; i < L; i++) {
        sum += pow(X[i] - Y[i], 2);
    }
    return sqrt(sum);
}

vector<int> ClusterLabel(vector<vector<double>> &Data, vector<vector<double>> &center) {
    vector<int> res;
    for(int i = 0; i < Data.size(); i++) {
        double dis1 = Distan(Data[i], center[0]);
        double dis2 = Distan(Data[i], center[1]);
        double dis3 = Distan(Data[i], center[2]);
        double dis = min(dis1, min(dis2, dis3));
        if(dis == dis1) res.push_back(1);
        else if(dis == dis2) res.push_back(2);
        else res.push_back(3);
    }
    return res;
}

vector<vector<double>> ClusterCenter(vector<vector<double>> &Data, vector<int> &label) {
    double cen1_x = 0, cen1_y = 0, cen2_x = 0, cen2_y = 0, cen3_x = 0, cen3_y = 0;
    int cnt1 = 0, cnt2 = 0, cnt3 = 0;
    for(int i = 0; i < Data.size(); i++) {
        if(label[i] == 1) {
            cen1_x += Data[i][0];
            cen1_y += Data[i][1];
            cnt1++;
        }
        else if(label[i] == 2){
            cen2_x += Data[i][0];
            cen2_y += Data[i][1];
            cnt2++;
        }
        else {
            cen3_x += Data[i][0];
            cen3_y += Data[i][1];
            cnt3++;
        }
    }
    return {
        {cen1_x / cnt1, cen1_y / cnt1},
        {cen2_x / cnt2, cen2_y / cnt2},
        {cen3_x / cnt3, cen3_y / cnt3}
    };
}
```

主函数执行如下:

```cpp
int main() {
    vector<vector<double>> Data = {
        {5.9, 3.2}, {4.6, 2.9}, {6.2, 2.8}, {4.7, 3.2}, {5.5, 4.2},
        {5.0, 3.0}, {4.9, 3.1}, {6.7, 3.1}, {5.1, 3.8}, {6.0, 3.0}
    };
    vector<vector<double>> center = {
        {6.2, 3.2}, {6.6, 3.7}, {6.5, 3.0}
    };
    vector<int> label = ClusterLabel(Data, center);
    for(int i = 0; i < Data.size(); i++) {
        cout << " 第" << i + 1 << " 个样本 (" << Data[i][0] << ","
        << Data[i][1] << ") 属于第" << label[i] << " 个簇" << endl;
    }
    vector<vector<double>> NewCenter = ClusterCenter(Data, label);
    for(int i = 0; i < 3; i++) {
        cout << " 第" << i + 1 << " 个簇中心坐标为 ("
        << setprecision(3) << NewCenter[i][0] <<"," <<
        setprecision(3) << NewCenter[i][1] << ")" << endl;
    }
}
```

具体的 C++ 代码输出如下:

```
开始运行...
第 1 个样本(5.9,3.2)属于第 1 个簇
第 2 个样本(4.6,2.9)属于第 1 个簇
第 3 个样本(6.2,2.8)属于第 3 个簇
第 4 个样本(4.7,3.2)属于第 1 个簇
第 5 个样本(5.5,4.2)属于第 2 个簇
第 6 个样本(5,3)属于第 1 个簇
第 7 个样本(4.9,3.1)属于第 1 个簇
第 8 个样本(6.7,3.1)属于第 3 个簇
第 9 个样本(5.1,3.8)属于第 1 个簇
第 10 个样本(6,3)属于第 1 个簇
第 1 个簇中心坐标为(5.17,3.17)
第 2 个簇中心坐标为(5.5,4.2)
第 3 个簇中心坐标为(6.45,2.95)
运行结束.
```

Problem 3

简答题:

- **问题 1: K-means 聚类算法暗含了对簇的什么假设?**

  解答: K-means 聚类算法暗含了各个簇种的数据具有一样的先验概率并且呈现球形分布.

- **问题 2: K-means 迭代一定会收敛么?**

  解答: 一定会, 这是因为损失函数是单调递减的, 所以使用坐标轴下降法是可以使得损失函数收敛 (但不一定收敛到全局极小值) 且聚类结果也会收敛.

- **问题 3: K-means 算法的初始化参数 (中心) 如何选取?**

  解答: 采用启发式做法, 即随机确定第一个类的中心, 其他类中心的位置尽量远离已有类中心. 具体做法是在 Scikit-Learn 里 K-means 的参数 init 中设置“k-means++”(默认值) 来使得初始化的质心彼此远离.

- **问题 4: K-means 算法的超参数 K 如何选取?**

  解答: 可以用手肘法, 观察出损失函数的拐点 (急速下降和平稳的交界处), 对应的横坐标值即可作为 K 值. 此外还可以用间隔统计法来获取合适的 K 值, 具体代码可见**Python implementation of the Gap Statistic**.

- **问题 5: K 均值算法有何局限性? 可以用什么方法改进?**

  解答: K-means 的局限性在于它假定了簇为球形并且各个簇的先验概率相等. 当各个簇是具有不同尺寸、密度的非球形簇时, 聚类效果不理想, 并且该算法易受到离群点的影响. 为此, 可以用簇的中位点来替代作为中心的均值点, 以免算法受到脏数据的干扰. 而针对先验概率相等的假设问题, 我们可以使用高斯混合模型来解决.

- **问题 6: 基于 GMM 的 EM 聚类算法为何要引入隐变量 z?**

  解答: 不引入隐变量 z 的话, 对数似然函数求导困难从而使得函数优化困难. 因此引入隐变量 z, 从而给出隶属度 $\gamma(z_{ik})$ 以求得对数似然的极大估计 $\boldsymbol{\mu}_k, \pi_k, \boldsymbol{\Sigma}_k$. 于是可以引出解的迭代方案 (即 EM 算法在 GMM 下的实例).

- **问题 7: K-means 和 EM 算法有什么共通之处?**

  解答: GMM 的 E 步时软划分版本的 K-means, 当所有 π_k 相等 (即 $1/k$) 且 $\boldsymbol{\Sigma}_k = \boldsymbol{O}, r_{ik} \in \{0, 1\}$ 时, 则 GMM 的 EM 算法退化为 K-means 算法.

- **问题 8: 如何度量簇的相似性? 不同度量方法的优缺点?**

  解答: 簇的相似性可以用 *Minkowski* 距离、余弦相似度、*Pearson* 相关系数以及 *Jaccard* 相似系数来度量. 其中 *Minkowski* 距离的优点在于对样本特征的旋转和平移比较鲁棒, 但缺点在于对数据尺度比较敏感. 余弦相似度的优势在于文本分析, 缺点是没有考虑向量模长的大小, 而只考虑它们的方向. 其中数据中心化后的 *Pearson* 相关系数就是余弦相似度, 优缺点是一致的. *Jaccard* 相似系数 (即交比并) 的优势在于处理图像分割问题 (如实例和语义分割), 缺点在于它容易受到数据规模的影响.

- **问题 9: 层次聚类的限制有哪些?**

  解答: 簇的合并/拆分过程不可逆、没有对应优化一个全局目标函数、对离群点和噪声不鲁棒.

- **问题 10: 请简述 PCA 流程.**

  解答: PCA 流程如算法2中所示:

Algorithm 2 PCA 算法

Input: 训练数据集 $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$, 低维空间的维度 D'

Output: $\boldsymbol{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{D'})$

1: 对样本中心化: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \dfrac{1}{N}\sum_{j=1}^{N} \boldsymbol{x}_j, i = 1, \cdots, N$;

2: 再计算协方差矩阵 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$;

3: 对 $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 作特征分解: $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^{-1}$;

4: D' 最大特征值对应的特征向量: $\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_{D'}$;

5: **end {PCA}**

改进后的 PCA 流程如算法3中所示:

Algorithm 3 改进 **PCA** 算法

Input: 训练数据集 $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$, 低维空间的维度 D'

Output: $\boldsymbol{U} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{D'})$

1: 对样本中心化: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \dfrac{1}{N}\sum_{j=1}^{N} \boldsymbol{x}_j, i = 1, \cdots, N$;

2: 对 $\boldsymbol{X}$ 作奇异值分解: $\boldsymbol{X} = \boldsymbol{U}_{D\times D}\boldsymbol{\Sigma}_{D\times N}\boldsymbol{V}^{\mathrm{T}}_{N\times N}$;

3: D' 个奇异值对应的左奇异向量: $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{D'}$;

4: **end {Improved PCA}**

KPCA 流程如算法4中所示:

Algorithm 4 KPCA 算法

Input: 训练数据集 $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$, 低维空间的维度 D', 核函数 $k(\cdot,\cdot)$

Output: $\boldsymbol{z}^{(i)} = \left(\boldsymbol{z}_1^{(i)}, \boldsymbol{z}_2^{(i)}, \cdots, \boldsymbol{z}_{D'}^{(i)}\right), i = 1, \cdots, N$

1: 计算核矩阵 $\boldsymbol{K}$: $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ 并对 $\boldsymbol{K}$ 中心化得到 $\overline{\boldsymbol{K}}$;

2: 对 $\overline{\boldsymbol{K}}$ 作特征分解: $\overline{\boldsymbol{K}} = \boldsymbol{\mathcal{A}}\boldsymbol{\Sigma}\boldsymbol{\mathcal{A}}^{-1}$;

3: 选择 D' 个最大特征值对应的特征向量: $\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_{D'}$; ▷ $\boldsymbol{\alpha} \in \mathbb{R}^{N\times 1}$, 通常对 $\boldsymbol{\alpha}$ 进行归一化, 模长为 1

4: 对于新样本 $\boldsymbol{x}$, 可得到其低维表示 (D' 维)$\boldsymbol{z}$ 为: ▷ 对所有样本求和开销大

$$\boldsymbol{z} = \begin{pmatrix} \sum_{i=1}^{N} \alpha_{1i} k\left(\boldsymbol{x}_i, \boldsymbol{x}\right) \\ \sum_{i=1}^{N} \alpha_{2i} k\left(\boldsymbol{x}_i, \boldsymbol{x}\right) \\ \vdots \\ \sum_{i=1}^{N} \alpha_{D'i} k\left(\boldsymbol{x}_i, \boldsymbol{x}\right) \end{pmatrix}$$

5: **end {KPCA}**

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 9 课程作业解答

2022 年 12 月 24 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

请给出半监督学习常用的三种假设, 并分别列举出一种依据该假设所设计的半监督学习算法.

Solution: 半监督学习常用的三种假设和主要代表算法叙述如下:

- **平滑假设:** 如果**高密度空间**中两点 $\boldsymbol{x}_1, \boldsymbol{x}_2$ 距离接近, 则对应的输出 y_1, y_2 也应该接近. 主要的代表算法是**生成式模型算法**.
- **聚类假设:** 如果两点在同一个簇, 那么它们很有可能属于同一个类. 主要的代表算法是 **S^3VMs 算法**.
- **流形假设: 高维数据**大致会分布在一个低维的流形上, 并且邻近的样本拥有相似的输出. 主要的代表算法是**基于图的算法**.

Problem 2

简述直推式和归纳式半监督学习算法的异同, 并分别例举出 3 种代表性算法.

Solution: 直推式和归纳式半监督学习的**相同点**是, 他们都使用少量有标注 + 大量无标注的数据集训练. 然而**不同点**在于: 直推式半监督算法关注模型在无标注数据集上的推理预测并且可以没有显式的学习函数 f, 而归纳式半监督算法则关注于学习出一个显式函数 f 来对新的测试集进行推理. 直推式的代表性半监督算法有: TSVM 算法、基于图的算法、半监督聚类. 归纳式的代表性半监督算法有: 自训练模型、多视图学习、生成式模型.

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 10 课程作业解答

2022 年 12 月 26 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

假设我们要采用 HMM 实现一个英文的词性标注系统, 系统中共有 20 种词性, 则状态转移矩阵 $\boldsymbol{A}$ 的大小为 _____.

Solution: 由于系统中共有 20 种词性, 因此 Markov 状态节点的个数就是 20, 于是状态转移矩阵的大小为 20×20.

Problem 2

已知以下贝叶斯网络 (如图1中所示), 包含 7 个变量, 即 Season(季节)、Flu(流感)、Dehydration(脱水)、Chills(发冷)、Headache(头疼)、Nausea(恶心)、Dizziness(头晕), 则下列条件独立成立的是 ( ).

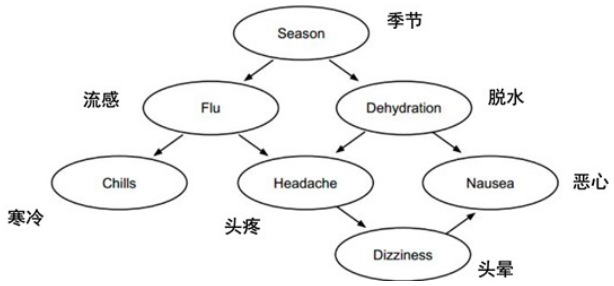


图 1: 贝叶斯网络

(A) . $Season \perp Chills|Flu$ (B) . $Season \perp Chills$ (C) . $Season \perp Headache|Flu$

Solution: 为了叙述方便, 我们不妨先对上述贝叶斯网络中的各节点进行拓扑排序, 具体如下图2所示:

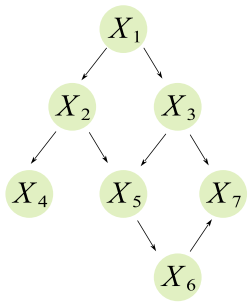


图 2: 简化后的贝叶斯网络

上述选项即变为

(A) . $X_1 \perp X_4|X_2$ (B) . $X_1 \perp X_4$ (C) . $X_1 \perp X_5|X_2$

显然 B 选项错误, 先看 C 选项: 给定 X_2 时, 检查 X_1, X_5 的可达性. 利用快速检验准则: 从 X_1 出发, 通过 X_3, 即可到达 X_5, 因此 C 项错误. 再看 A 项, 利用准则可以看出, 从 X_1 出发, 要么在 $X_1 \to X_2 \to X_4$ 这条路线上被反弹回 X_1. 要么先经过 $X_1 \to X_3 \to X_5$ 到达 X_5, 但是球在路线 $X_5 \to X_2 \to X_4$ 上被 X_2 截止了, 总之到达不了 X_4; 从 X_4 出发, 类似的, 也是到达不了 X_1. 因此, $X_1 \perp X_4|X_2$(即 A 项正确).

Problem 3

已知以下贝叶斯网络 (如图3中所示), 包含 4 个二值变量, 则该网络一共有 (　) 个参数.

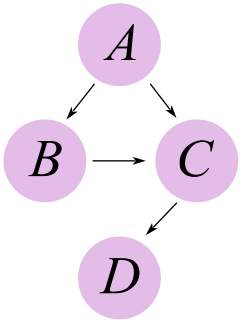


图 3: 贝叶斯网络

Solution: 先写出联合概率分布

$$p(A,B,C,D) = p(A)\cdot p(B|A)\cdot p(C|A,B)\cdot p(D|C)$$

因此网络参数共有 $2^0+2^1+2^2+2^1=9$ 个, 于是选 C 项.

Problem 4

假设你有 3 个盒子, 每个盒子里都有一定数量的苹果和橘子. 每次随机选择一个盒子, 然后从盒子里选一个水果, 并记录你的发现 (a 代表苹果, o 代表橘子). 不幸的是, 你忘了写下你所选的盒子, 只是简单的记下了苹果和橘子. 假设每个盒子中水果数量如下:

- 盒子 1: 2 个苹果, 2 个橘子;
- 盒子 2: 3 个苹果, 1 个橘子;
- 盒子 3: 1 个苹果, 3 个橘子;

(1). 请用 HMM 模型描述上述过程;
(2). 请给出水果序列 $\boldsymbol{x}=(a,a,o,o,o)$ 对应的最佳盒子序列.

Solution: (1). 将盒子视作隐变量 (即状态节点), 拿出来的水果视作观测变量 (即输出节点). 因为每次都是随机选取的盒子, 因此初始状态的概率分布应为均匀分布, 即 $\boldsymbol{\pi}=\begin{pmatrix}1/3 & 1/3 & 1/3\end{pmatrix}^{\mathrm{T}}$. 因为每次都是以均匀分布抽取盒子的且 $a_{ij}=p\left(y_{t+1}=s_j|y_t=s_i\right)$, 因此盒子间的状态转移矩阵为

$$\boldsymbol{A}=\left(a_{ij}\right)_{N\times N}=\begin{pmatrix}1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\end{pmatrix}$$

由于 $b_{ij}=p\left(x_t=o_j|y_t=s_i\right)$, 因此发射概率矩阵 (给定盒子时, 选择每种水果的概率) 为

$$\boldsymbol{B}=\left(b_{ij}\right)_{N\times M}=\begin{pmatrix}1/2 & 1/2\\ 3/4 & 1/4\\ 1/4 & 3/4\end{pmatrix}$$

(2). Viterbi 解码算法如下:

$$\text{动态规划}: V_1(j)=\pi_j b_j(x_1),\begin{cases}V_t(j)=b_j(x_t)\cdot\max\limits_{1\le i\le N}\{a_{ij}V_{t-1}(i)\}\\ \psi_t(j)=\arg\max\limits_{1\le i\le N}\{a_{ij}V_{t-1}(i)\}\end{cases},\text{其中}1\le j\le N$$

$$\text{反向回溯}: P^*=\max_{1\le i\le N}V_T(i),\begin{cases}i_T^*=\arg\max\limits_{1\le i\le N}V_T(i)\\ i_t^*=\psi_{t+1}\left(i_{t+1}^*\right)\end{cases}$$

- 当 $t=1$ 时, 已知 $x_1=a$, 所以初始化如下:

$$V_1(1)=\pi_1 b_1(a)=1/3\cdot 1/2=1/6,\psi_1(1)=0$$
$$V_1(2)=\pi_2 b_2(a)=1/3\cdot 3/4=1/4,\psi_1(2)=0$$
$$V_1(3)=\pi_3 b_3(a)=1/3\cdot 1/4=1/12,\psi_1(3)=0$$

- 当 $t=2$ 时, 已知 $x_2=a$, 所以有:

$$V_2(1)=b_1(a)\cdot 1/3\cdot 1/4=1/24,\psi_2(1)=2$$
$$V_2(2)=b_2(a)\cdot 1/3\cdot 1/4=1/16,\psi_2(2)=2$$
$$V_2(3)=b_3(a)\cdot 1/3\cdot 1/4=1/48,\psi_2(3)=2$$

- 当 $t=3$ 时, 已知 $x_3=o$, 所以有:

$$V_3(1)=b_1(o)\cdot 1/3\cdot 1/16=1/96,\psi_3(1)=2$$
$$V_3(2)=b_2(o)\cdot 1/3\cdot 1/16=1/192,\psi_3(2)=2$$
$$V_3(3)=b_3(o)\cdot 1/3\cdot 1/16=1/64,\psi_3(3)=2$$

- 当 $t=4$ 时, 已知 $x_4=o$, 所以有:

$$V_4(1)=b_1(o)\cdot 1/3\cdot 1/64=1/384,\psi_4(1)=3$$
$$V_4(2)=b_2(o)\cdot 1/3\cdot 1/64=1/768,\psi_4(2)=3$$
$$V_4(3)=b_3(o)\cdot 1/3\cdot 1/64=1/256,\psi_4(3)=3$$

- 当 $t=5$ 时, 已知 $x_5=o$, 所以有:

$$V_5(1)=b_1(o)\cdot 1/3\cdot 1/256=1/1536,\psi_5(1)=3$$
$$V_5(2)=b_2(o)\cdot 1/3\cdot 1/256=1/3072,\psi_5(2)=3$$
$$V_5(3)=b_3(o)\cdot 1/3\cdot 1/256=1/1024,\psi_5(3)=3$$

迭代过程终止, 且 $P^*=\max\limits_{1\le i\le 3}V_T(i)=V_5(3)=\dfrac{1}{1024}$. 回溯过程为:

$$i_5^*=3,i_4^*=\psi_5(3)=3,i_3^*=\psi_4(3)=3,i_2^*=\psi_3(3)=2,i_1^*=\psi_2(2)=2$$

因此回溯出最优路径为 $\boldsymbol{y}=\{2,2,3,3,3\}$.

Problem 5

给定如图4所示的 HMM

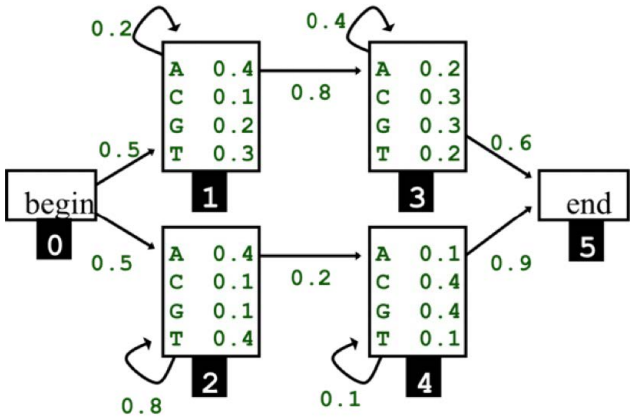


图 4: HMM 示意图

(1). 采用前向算法计算序列 AGTT 出现概率;
(2). 计算观测 TATA 最可能的序列.

Solution: (1). 由于初始状态节点已经确定为“0”了, 所以初始状态的概率分布为

$$\boldsymbol{\pi} = (1,0,0,0,0,0)^{\mathrm{T}}$$

转移概率矩阵 $\boldsymbol{A}$ 和发射概率矩阵 $\boldsymbol{B}$(把 begin 和 end 也看做观测状态且放到 $\boldsymbol{B}$ 的最后两列) 分别为:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0.8 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0.1 & 0.9 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0.4 & 0.1 & 0.2 & 0.3 & 0 & 0 \\ 0.4 & 0.1 & 0.1 & 0.4 & 0 & 0 \\ 0.2 & 0.3 & 0.3 & 0.2 & 0 & 0 \\ 0.1 & 0.4 & 0.4 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

α 递归计算的前向算法为:

$$\alpha_1(j) = \pi_j b_j(x_1), \alpha_t(j) = b_j(x_t) \cdot \sum_{i=1}^{N} a_{ij}\alpha_{t-1}(i), \text{其中} 1 \le j \le N$$

为叙述方便, 将状态“0,1,2,3,4,5”记作状态“1,2,3,4,5,6”. 此时观测序列为 {begin, A, G, T, T, end}. 因此初始化 ($t = 1$ 时), 则有 $x_1 = \text{begin}$ 且:

$$\alpha_1(1) = \pi_1 b_1(x_1) = 1 \cdot 1 = 1, \alpha_1(2) = \alpha_1(3) = \alpha_1(4) = \alpha_1(5) = \alpha_1(6) = 0$$

当 $t = 2$ 时, 则有 $x_2 = A, \alpha_2(j) = b_j(A) \cdot \sum_{i=1}^{6} a_{ij}\alpha_1(i)$ 且:

$$\alpha_2(1) = 0, \alpha_2(2) = 0.2, \alpha_2(3) = 0.2, \alpha_2(4) = 0, \alpha_2(5) = 0, \alpha_2(6) = 0$$

当 $t = 3$ 时, 则有 $x_3 = G, \alpha_3(j) = \left(\sum_{i=1}^{6} \alpha_2(i)\, a_{ij}\right) b_j(G)$ 且:

$$\alpha_3(1) = 0, \alpha_3(2) = 0.008, \alpha_3(3) = 0.016, \alpha_3(4) = 0.048, \alpha_3(5) = 0.016, \alpha_3(6) = 0$$

当 $t=4$ 时, 则有 $x_4=T, \alpha_4(j)=\left(\sum_{i=1}^{6}\alpha_3(i)\,a_{ij}\right)b_j(T)$ 且:

$$\alpha_4(1)=0, \alpha_4(2)=0.00048, \alpha_4(3)=0.00512, \alpha_4(4)=0.00512, \alpha_4(5)=0.00048, \alpha_4(6)=0$$

当 $t=5$ 时, 则有 $x_5=T, \alpha_5(j)=\left(\sum_{i=1}^{6}\alpha_4(i)\,a_{ij}\right)b_j(T)$ 且:

$$\alpha_5(1)=0, \alpha_5(2)=0.0000288, \alpha_5(3)=0.00164, \alpha_5(4)=0.000486, \alpha_5(5)=0.000107, \alpha_5(6)=0$$

当 $t=6$ 时, 则有 $x_6=\text{end}, \alpha_6(j)=\left(\sum_{i=1}^{6}\alpha_5(i)\,a_{ij}\right)b_j(\text{end})$ 且:

$$\alpha_6(1)=\alpha_5(2)=\alpha_5(3)=\alpha_5(4)=\alpha_5(5)=0, \alpha_5(6)=0.000388$$

因此序列 $(x_1,x_2,x_3,x_4,x_5,x_6)=$ (begin,A,G,T,T,end) 的出现概率为

$$p(x_1,x_2,x_3,x_4,x_5,x_6|\boldsymbol{A},\boldsymbol{B},\boldsymbol{\pi})=\sum_{j=1}^{6}\alpha_6(j)=0.000388$$

(2). 注意到观测序列为 {begin, T, A, T, A, end}, Viterbi 解码算法如下:

$$\text{动态规划}: V_1(j)=\pi_j b_j(x_1), \begin{cases} V_t(j)=b_j(x_t)\cdot\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\} \\ \psi_t(j)=\arg\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\} \end{cases}, \text{其中} 1\le j\le N$$

$$\text{反向回溯}: P^*=\max_{1\le i\le N}V_T(i), \begin{cases} i_T^*=\arg\max\limits_{1\le i\le N}V_T(i) \\ i_t^*=\psi_{t+1}\left(i_{t+1}^*\right) \end{cases}$$

- 当 $t=1$ 时, 已知 $x_1=$ begin, 所以初始化如下:

$$V_1(1)=1, V_1(2)=V_1(3)=V_1(4)=V_1(5)=V_1(6)=0$$
$$\psi_1(1)=\psi_1(2)=\psi_1(3)=\psi_1(4)=\psi_1(5)=\psi_1(6)=0$$

- 当 $t=2$ 时, 已知 $x_2=T$, 所以有:

$$V_2(1)=0, V_2(2)=0.15, V_2(3)=0.2, V_2(4)=0, V_2(5)=0, V_2(6)=0$$
$$\psi_2(1)=1, \psi_2(2)=1, \psi_2(3)=1, \psi_2(4)=1, \psi_2(5)=1, \psi_2(6)=1$$

- 当 $t=3$ 时, 已知 $x_3=A$, 所以有:

$$V_3(1)=0, V_3(2)=0.012, V_3(3)=0.064, V_3(4)=0.024, V_3(5)=0.004, V_3(6)=0$$
$$\psi_3(1)=1, \psi_3(2)=2, \psi_3(3)=3, \psi_3(4)=2, \psi_3(5)=3, \psi_3(6)=1$$

- 当 $t=4$ 时, 已知 $x_4=T$, 所以有:

$$V_4(1)=0, V_4(2)=0.00072, V_4(3)=0.02048, V_4(4)=0.00192, V_4(5)=0.00128, V_4(6)=0$$
$$\psi_4(1)=1, \psi_4(2)=2, \psi_4(3)=3, \psi_4(4)=2, \psi_4(5)=3, \psi_4(6)=4$$

- 当 $t=5$ 时, 已知 $x_5=T$, 所以有:

$$V_5(1)=0, V_5(2)=5.76\times10^{-5}, V_5(3)=0.0065536, V_5(4)=0.0001536, V_5(5)=0.0004096, V_5(6)=0$$
$$\psi_5(1)=1, \psi_5(2)=2, \psi_5(3)=3, \psi_5(4)=4, \psi_5(5)=3, \psi_5(6)=5$$

- 当 $t=6$ 时, 已知 $x_6=\text{end}$, 所以有:

$$V_6(1)=0, V_6(2)=0, V_6(3)=0, V_6(4)=0, V_6(5)=0, V_6(6)=0.00036864$$
$$\psi_6(1)=1, \psi_6(2)=2, \psi_6(3)=3, \psi_6(4)=4, \psi_6(5)=3, \psi_6(6)=5$$

迭代过程终止, 且 $P^*=\max\limits_{1\le i\le 4} V_T^i=V_6(6)=0.00036864, i_6^*=6$, 于是回溯过程如下:

$$i_6^*=\arg\max_{1\le i\le 6} V_6(i)=6, i_5^*=\psi_6\left(i_6^*\right)=\psi_6(6)=5, i_4^*=\psi_5\left(i_5^*\right)=\psi_5(5)=3,$$
$$i_3^*=\psi_4\left(i_4^*\right)=\psi_4(3)=3, i_2^*=\psi_3\left(i_3^*\right)=\psi_3(3)=3, i_1^*=\psi_2\left(i_2^*\right)=\psi_2(3)=1$$

因此回溯得到的最优序列为 $1\to3\to3\to3\to5\to6$, 将标号对应回去则**真正的最优序列**为 $0\to2\to2\to2\to4\to5$, 且对应的概率值为 0.00036864.

接下来我们需要通过实验代码来验证上述计算的正确性, 因此我们需要先给出其伪代码:

Algorithm 1 HMM 状态序列解码的 Viterbi 算法

Input: 转移概率矩阵 $\boldsymbol{A}$, 发射概率矩阵 B, 初始概率分布 $\boldsymbol{\pi}$
Output: HMM 的最优路径

1: **Viterbi**($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}$)**:**
2: **for** $j\leftarrow1; j\le N; j\leftarrow j+1$ **do**
3: $\psi_1(j)\leftarrow0$;
4: $V_1(j)\leftarrow\pi_j b_j(x_1)$; ▷ $t=1$ 时的初始化, 其中 $b_j(x_t)$ 为状态节点 j 输出 x_t 的概率
5: **end for**
6: **for** $t\leftarrow2; t\le T; t\leftarrow t+1$ **do** ▷ 前向动态规划
7: **for** $j\leftarrow1; j\le N; j\leftarrow j+1$ **do**
8: $V_t(j)=b_j(x_t)\cdot\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\}$;
9: $\psi_t(j)=\arg\max\limits_{1\le i\le N}\left\{a_{ij}V_{t-1}(i)\right\}$;
10: **end for**
11: **end for**
12: $P^*\leftarrow\max\limits_{1\le i\le N}V_T(i), i_T^*\leftarrow\arg\max\limits_{1\le i\le N}V_T(i)$; ▷ 初始化反向回溯
13: **for** $t\leftarrow T-1; t\ge1; t\leftarrow t-1$ **do**
14: $i_t^*=\psi_{t+1}\left(i_{t+1}^*\right)$; ▷ 反向状态回溯
15: **end for**
16: **return** 最优路径 $\left\{i_1^*, i_2^*, \cdots, i_T^*\right\}$;
17: **end {Viterbi}**

现在分析一下算法的时间复杂度[1], $T(N,T)=O(N)+O(T\cdot N\cdot N)+O(N)=O(T\cdot N^2)$, 显然是多项式时间内可计算的. 有了上述伪码, 则可以写出本题的 C++ 代码 (如后页所示):

[1]伪码的 8,9,12 行需要一次 for 循环 (循环变量为 i) 来求出最大值和所在索引, 需要消耗 $O(N)$ 的时间.

```cpp
#include <bits/stdc++.h>
using namespace std;

int Idx(string x) {
    int res;
    if (x == "A") res = 0;
    else if (x == "C") res = 1;
    else if (x == "G") res = 2;
    else if (x == "T") res = 3;
    else if (x == "begin") res = 4;
    else if (x == "end") res = 5;
    return res;
}

vector<int> Viterbi( //Viterbi 动态规划解码算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> V(T, vector<double>(N, 0));
    vector<vector<int>> psi(T, vector<int>(N, 0));
    for(int j = 0; j < N; j++) {
        V[0][j] = pi[j] * B[j][Idx(X[0])];
        psi[0][j] = 0;
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = A[0][j] * V[t - 1][0];
            int idx = 0;
            for(int i = 1; i < N; i++) {
                if(A[i][j] * V[t - 1][i] > temp) {
                    temp = A[i][j] * V[t - 1][i];
                    idx = i;
                }
            }
            V[t][j] = B[j][Idx(X[t])] * temp;
            psi[t][j] = idx;
        }
    }
    double P = V[T - 1][0];
    vector<int> path(T);
    for(int i = 1; i < N; i++) {
        if(V[T - 1][i] > P) {
            P = V[T - 1][i];
            path[T - 1] = i;
        }
    }
    for(int t = T - 2; t >= 0; t--) {
        path[t] = psi[t + 1][path[t + 1]];
    }
    return path;
}
```

主函数编码如下:

```cpp
int main() {
    vector<string> X = {"begin", "T", "A", "T", "A", "end"};
    vector<double> pi = {1, 0, 0, 0, 0, 0};
    vector<vector<double>> A = {
        {0, 0.5, 0.5, 0, 0, 0},
        {0, 0.2, 0, 0.8, 0, 0},
        {0, 0, 0.8, 0, 0.2, 0},
        {0, 0, 0, 0.4, 0, 0.6},
        {0, 0, 0, 0, 0.1, 0.9},
        {0, 0, 0, 0, 0, 1}
    };
    vector<vector<double>> B = {
        {0, 0, 0, 0, 1, 0},
        {0.4, 0.1, 0.2, 0.3, 0, 0},
        {0.4, 0.1, 0.1, 0.4, 0, 0},
        {0.2, 0.3, 0.3, 0.2, 0, 0},
        {0.1, 0.4, 0.4, 0.1, 0, 0},
        {0, 0, 0, 0, 0, 1}
    };
    vector<int> path = Viterbi(X, pi, A, B);
    cout << " 最优路径为: " << endl;
    for(int i = 0; i < path.size(); i++) {
        cout << path[i] << " ";
    }
}
```

相应的输出为:

```
开始运行...
最优路径为: 0 2 2 2 4 5
运行结束
```

显然这与我们的手动计算结果是相同的, 证明了我们算法实现的正确性和手算的正确性.
并且我们可以利用 python 中的 hmmlearn 库来进行编码并输出最优序列, 具体代码如后页所示:

```python
# pip3 install hmmlearn
import numpy as np
from hmmlearn import hmm

states = ["0", "1", "2", "3", "4", "5"]
n_states = len(states)

observations = ["A", "C", "G", "T", "begin", "end"]
n_observations = len(observations)

start_probability = np.array([1, 0, 0, 0, 0, 0])

transition_probability = np.array([
    [0, 0.5, 0.5, 0, 0, 0],
    [0, 0.2, 0, 0.8, 0, 0],
    [0, 0, 0.8, 0, 0.2, 0],
    [0, 0, 0, 0.4, 0, 0.6],
    [0, 0, 0, 0, 0.1, 0.9],
    [0, 0, 0, 0, 0, 1]
])

emission_probability = np.array([
    [0, 0, 0, 0, 1, 0],
    [0.4, 0.1, 0.2, 0.3, 0, 0],
    [0.4, 0.1, 0.1, 0.4, 0, 0],
    [0.2, 0.3, 0.3, 0.2, 0, 0],
    [0.1, 0.4, 0.4, 0.1, 0, 0],
    [0, 0, 0, 0, 0, 1]
])

model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_=start_probability
model.transmat_=transition_probability
model.emissionprob_=emission_probability

seen = np.array([[4, 3, 0, 3, 0, 5]]).T
logprob, box = model.decode(seen, algorithm="viterbi")
print(" 观测序列为:", ", ".join(map(lambda x: observations[x[0]], seen)))
print(" 最优路径为:", ", ".join(map(lambda x: states[x], box)))
```

上述代码的运行结果为:

```
观测序列为: begin, T, A, T, A, end
最优路径为: 0, 2, 2, 2, 4, 5
```

由此可见, 自行编写的算法跟 hmmlearn 算法库的运行结果是一致的.

Problem 6

请编写 **Problem 4** 中的 C++ 程序和 Python 程序, 并展示运行结果.

Solution: C++ 代码如下所示:

```cpp
#include <bits/stdc++.h>
using namespace std;

int Idx(string x) {
    int res;
    if (x == "Apple") res = 0;
    else if (x == "Orange") res = 1;
    return res;
}

vector<int> Viterbi( //Viterbi 动态规划解码算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> V(T, vector<double>(N, 0));
    vector<vector<int>> psi(T, vector<int>(N, 0));
    for(int j = 0; j < N; j++) {
        V[0][j] = pi[j] * B[j][Idx(X[0])];
        psi[0][j] = 0;
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = A[0][j] * V[t - 1][0];
            int idx = 0;
            for(int i = 1; i < N; i++) {
                if(A[i][j] * V[t - 1][i] > temp) {
                    temp = A[i][j] * V[t - 1][i];
                    idx = i;
                }
            }
            V[t][j] = B[j][Idx(X[t])] * temp;
            psi[t][j] = idx;
        }
    }
    double P = V[T - 1][0];
    vector<int> path(T);
    for(int i = 1; i < N; i++) {
        if(V[T - 1][i] > P) {
            P = V[T - 1][i];
            path[T - 1] = i;
        }
    }
    for(int t = T - 2; t >= 0; t--) {
        path[t] = psi[t + 1][path[t + 1]];
    }
    return path;
}
```

主函数编码如下:

```cpp
int main() {
    vector<string> X = {"Apple", "Apple", "Orange", "Orange", "Orange"};
    vector<double> pi = {1.0 / 3, 1.0 / 3, 1.0 / 3};
    vector<vector<double>> A = {
        {1.0 / 3, 1.0 / 3, 1.0 / 3},
        {1.0 / 3, 1.0 / 3, 1.0 / 3},
        {1.0 / 3, 1.0 / 3, 1.0 / 3}
    };
    vector<vector<double>> B = {
        {1.0 / 2, 1.0 / 2},
        {3.0 / 4, 1.0 / 4},
        {1.0 / 4, 3.0 / 4}
    };
    vector<int> path = Viterbi(X, pi, A, B);
    cout << " 最优路径为: " ;
    for(int i = 0; i < path.size(); i++) {
        cout << path[i] + 1 << " ";
    }
}
```

代码输出结果为 (即最优序列为盒子 2→ 盒子 2→ 盒子 3→ 盒子 3→ 盒子 3):

```
开始运行...
最优路径为: 2 2 3 3 3
运行结束
```

Problem 4 和 **Problem 5** 的 α 前向递归算法的 C++ 代码如下所示:

```cpp
double Forward_Alg( //alpha 前向递归算法
    vector<string> &X, vector<double> &pi,
    vector<vector<double>> &A, vector<vector<double>> &B
) {
    int T = X.size(), N = A.size(), M = B[0].size();
    vector<vector<double>> alpha(T, vector<double>(N, 0));
    for(int j = 0; j < N; j++) {
        alpha[0][j] = pi[j] * B[j][Idx(X[0])];
    }
    for(int t = 1; t < T; t++) {
        for(int j = 0; j < N; j++) {
            double temp = 0;
            for(int i = 0; i < N; i++) {
               temp += A[i][j] * alpha[t - 1][i];
            }
            alpha[t][j] = B[j][Idx(X[t])] * temp;
        }
    }
    return accumulate(alpha[T - 1].begin(), alpha[T - 1].end(), 0.0);
}
```

主函数中添加如下代码并获得输出 (显然跟手算结果是一样的, 因此代码正确):

```
vector<string> S = {"begin", "A", "G", "T", "T", "end"};
double prob = Forward_Alg(S, pi, A, B);
cout << " 序列 S 的出现概率为" << prob << endl;
开始运行...
序列 S 的出现概率为 0.00038832
运行结束
```

同时, Viterbi 解码算法的 Python 代码如下:

```
import numpy as np
from hmmlearn import hmm

states = [" 盒子 1", " 盒子 2", " 盒子 3"]
n_states = len(states)

observations = ["Apple", "Orange"]
n_observations = len(observations)

start_probability = np.array([1.0/3, 1.0/3, 1.0/3])

transition_probability = np.array([
    [1.0/3, 1.0/3, 1.0/3],
    [1.0/3, 1.0/3, 1.0/3],
    [1.0/3, 1.0/3, 1.0/3]
])

emission_probability = np.array([
    [1.0/2, 1.0/2],
    [3.0/4, 1.0/4],
    [1.0/4, 3.0/4]
])

model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_=start_probability
model.transmat_=transition_probability
model.emissionprob_=emission_probability

seen = np.array([[0, 0, 1, 1, 1]]).T
logprob, box = model.decode(seen, algorithm="viterbi")
print(" 观测序列为:", ", ".join(map(lambda x: observations[x[0]], seen)))
print(" 最优路径为:", ", ".join(map(lambda x: states[x], box)))
```

代码输出为:

```
观测序列为: Apple, Apple, Orange, Orange, Orange
最优路径为: 盒子 2, 盒子 2, 盒子 3, 盒子 3, 盒子 3
```

中国科学院大学

University of Chinese Academy of Sciences

模式识别与机器学习

081203M04004H

Chap 11 课程作业解答

2022 年 12 月 26 号

Professor: 黄庆明

学生: 周胤昌
学号: 202228018670052
学院: 网络空间安全学院
所属专业: 网络空间安全
方向: 安全协议理论与技术

Problem 1

模型复杂度过低/过高通常会导致 Bias 和 Variance 有怎样的问题?

Solution: 通常, 模型复杂度过低会导致偏差高且方差低 (即欠拟合), 而复杂度过高则会导致偏差小但方差大 (即过拟合).

Problem 2

怎样判断且缓解过拟合/欠拟合问题?

Solution: 主要是通过验证集上的误差来判断. 当校验误差一直减小时, 则说明模型目前处于欠拟合; 当校验误差先减小而后增大时, 则说明模型目前处于过拟合状态. 当模型处于欠拟合状态时, 需要增加模型复杂度, 具体措施有:

- 增加模型的迭代次数;
- 增加更多特征;
- 降低模型正则化水平.

当模型处于过拟合状态时, 需要降低模型复杂度, 具体措施有:

- 及早停止迭代;
- 减少特征数量;
- 提高模型正则化水平;
- 扩大训练集.

Problem 3

比较 Bagging 和 Boosting 算法的异同.

Solution: Bagging 和 Boosting 算法的异同:

- **不同点:** 这两类算法的不同点在于前者是对训练集做 m 次有放回随机抽样来得到 m 个子训练集, 从而分别**并行学习**得到 m 个基模型. 而后者的 m 个弱学习器是**按顺序进行学习**的, 并且有 m 次的训练集转化.
- **相同点:** 相同点在于两者都分别利用 m 个弱模型做出 m 个预测, 并最终进行预测结果的整合.

Problem 4

简述 Adaboosting 的流程.

Solution: Adaboosting 的流程如下算法1中所示:

Algorithm 1 AdaBoost 算法流程

Input: 给定训练集 $(x_1, y_1), (x_2, y_2) \cdots, (x_m, y_m)$, 其中 $x_i \in X, y_i \in \{-1, +1\}$

Output: 强分类器 $H_{\text{final}}(x)$

初始化 $D_1(i) = 1/m, \forall i \in \{1, 2, \cdots, m\}$;

for $t = 1, 2, \cdots, T$ **do**

训练有误差的弱分类器 $h_t : X \to \{-1, +1\}$;

$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i] < \frac{1}{2}$; ▷ 如果 error= $\frac{1}{2}$, 则学习器 h_1 在训练集 D_2 上的性能为随机猜测

$\alpha_t = 1/2 \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$; ▷ 如果 error 越小, 则 α_t 越大

$\forall i \in \{1, 2, \cdots, m\}$, 做如下更新: ▷ 其中 Z_t 为正则化因子

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t}\exp(-\alpha_t y_i h_t(x_i)) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t}, \text{若} y_i = h_t(x_i) \\ e^{\alpha_t}, \text{若} y_i \neq h_t(x_i) \end{cases}$$

end for

return 强分类器 $H_{\text{final}}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$; ▷ 强分类器的权重较大

Problem 5

随机森林更适合采用哪种决策树?

(A) .性能好, 深度较深 (B) .性能弱, 深度较浅

Solution: 选择 (A), 随机森林属于 Bagging 集成算法, 因为 Bagging 更适合对偏差低、方差高 (即过拟合) 的模型进行融合, 所以随机森林更适合性能好、深度较深 (即过拟合) 的决策树.

Problem 6

基于树的 Boosting 更适合采用哪种决策树?

(A) .性能好, 深度较深 (B) .性能弱, 深度较浅

Solution: 选择 (B), 因为 Boosting 的基本思想是将弱学习器组合成强学习器. 故而基于树的 Boosting 更适合采用复杂度低的决策树, 即层数不深、性能弱的决策树.

Problem 7

如果对决策树模型采用 Bagging 方式进行集成学习, 更适合采用哪种方法对决策树的超参 (比如树的深度) 进行调优?

(A) .交叉验证 (B) .包外估计

Solution: 选择 (B), 在 Bagging 中, 每个弱学习器只在原数据集的一部分上进行训练, 因此可以不用交叉验证而直接采用包外估计来进行超参调优.