# 模式识别第三次作业

201300086 史浩男 人工智能学院

## 一、PCA投影推导

### (a)

等价于证明：$\mathrm{Cov}(x)-\mathrm{Cov}(y)=\lambda_1\xi_1\xi_1^T$

将 $x_i$ 在 $\xi_1$ 上的投影表示为 $(x_i-\xi_1)^T\xi_1=x_i^T\xi_1-\bar{x}^T\xi_1$

$$
\begin{aligned}
y_i &= x_i-\xi_1^T(x_i-\bar{x})\xi_1 \\
&= \sum_{j=1}^{D}(x_{ij}-\bar{x}_j)\xi_j-\left[\sum_{j=1}^{D}(x_{ij}-\bar{x}_j)\xi_1\right]\xi_1 \\
&= \sum_{j=2}^{D}(x_{ij}-\bar{x}_j)\xi_j-\left[x_i^T\xi_1-\bar{x}^T\xi_1\right]\xi_1 \\
&= \sum_{j=2}^{D}(x_{ij}-\bar{x}_j)\xi_j-x_i^T\xi_1\xi_1^T\xi_1+\bar{x}^T\xi_1\xi_1^T\xi_1 \\
&= \sum_{j=2}^{D}(x_{ij}-\bar{x}_j)\xi_j-x_i^T\xi_1\xi_1^T\xi_1
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
\mathrm{Cov}(y) &= \frac{1}{N}\sum_{i=1}^{N}(y_i-\bar{y})(y_i-\bar{y})^T \\
&= \frac{1}{N}\sum_{i=1}^{N}\left((x_i-\xi_1^T(x_i-\bar{x})\xi_1)-\bar{y}\right)\left((x_i-\xi_1^T(x_i-\bar{x})\xi_1)-\bar{y}\right)^T \\
&= \frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})(x_i-\bar{x})^T-\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})\xi_1^T(x_i-\bar{x})^T-\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})(x_i-\bar{x})^T\xi_1+\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})\xi_1^T(x_i-\bar{x})^T\xi_1 \\
&= \mathrm{Cov}(x)-\xi_1\lambda_1\xi_1^T \\
&= \sum_{i=2}^{D}\lambda_i\xi_i\xi_i^T
\end{aligned}
$$

## 二、瑞利熵

### (a)

由于$S_w$正定，令$S_w=CC^T$

$$
\begin{gathered}
i\neq j\text{时，由}S_Bw=\lambda S_ww\text{可得} \\
w_i^TS_B^Tw_j=\lambda_iw_i^TC^TCw_j \\
w_i^TS_Bw_j=\lambda_jw_i^TCC^Tw_j \\
\text{由于}S_B\text{对称有}S_B^T=S_B\text{，两式相减得：} \\
0=(\lambda_i-\lambda_j)w_i^TS_w^Tw_j \\
\text{由于}i\neq j\text{，故 }w_i^TS_w^Tw_j=0
\end{gathered}
\tag{3}
$$

### (b)

对于瑞利商，有$w_i^TS_Bw_i=\lambda_iw_i^TS_ww_i$，故$J(w)=\lambda$，最大值为$\lambda_1$，最小值为$\lambda_n$

### (c)

由题$W=(w_1,w_2,\ldots,w_d)$ 有，

$$
\begin{aligned}
J &= \frac{|W^TS_BW|}{|W^TS_wW|} \\
&= \frac{\Pi_1^dw_i^TS_Bw_i}{\Pi_1^dw_i^TS_ww_i} \\
&= \Pi_1^d\lambda_i
\end{aligned}
\tag{4}
$$

# 三、核方法

令$\kappa_1(\boldsymbol{x},\boldsymbol{y})$，$\kappa_2(\boldsymbol{x},\boldsymbol{y}),\kappa_3(\boldsymbol{x},\boldsymbol{y})$对应的矩阵为$K_1,K_2,K_3$，均为定义在$\mathbb{R}^n\times\mathbb{R}^n$上

**(a)+合法**

$$\begin{gathered}\text{由题：令}\kappa(x,y)=\kappa_1(x,y)+\kappa_2(x,y)\text{ 对应的矩阵为}K\\ [K]_{ij}=\kappa_1(x_i,y_j)+\kappa(x_i,y_j)\text{ 有}\\ K=K_1+K_2\\ \text{对任意实非零列向量}x\text{有 }x^TK_1x\geq 0,\ x^TK_2x\geq 0\\ x^TKx=x^TK_1x+x^TK_2x\geq 0\text{ 恒成立}\\ \text{故}K\text{是一个半正定的矩阵，}\kappa(x,y)\text{是一个合法的核函数}\end{gathered}\tag{5}$$

**(b)-不合法**

$$\begin{gathered}\text{由题：令}\kappa(x,y)=\kappa_1(x,y)-\kappa_2(x,y)\text{ 对应的矩阵为}K\\ [K]_{ij}=\kappa_1(x_i,y_j)-\kappa_2(x_i,y_j)\text{ 有}\\ K=K_1-K_2\\ \text{对任意实非零列向量}x\text{有 }x^TK_1x\geq 0,\ x^TK_2x\geq 0\\ x^TKx=x^TK_1x-x^TK_2x\geq 0\text{ 不恒成立}\\ \text{故}K\text{不一定是一个半正定的矩阵，}\kappa(x,y)\text{不一定是一个合法的核函数}\end{gathered}\tag{6}$$

**(c)α合法**

$$\begin{gathered}\text{由题：令}\kappa(x,y)=\alpha\kappa_1(x,y)\text{ 对应的矩阵为}K\\ [K]_{ij}=\alpha\kappa_1(x_i,y_j)\text{ 有}\\ K=\alpha K_1\\ \text{对任意实非零列向量}x\text{有 }x^TK_1x\geq 0,\\ x^TKx=\alpha x^TK_1x\geq 0\text{ 恒成立}\\ \text{故}K\text{是一个半正定的矩阵，}\kappa(x,y)\text{是一个合法的核函数}\end{gathered}\tag{7}$$

**(d)-α不合法**

$$\begin{gathered}\text{由题：令}\kappa(x,y)=-\alpha\kappa_1(x,y)\text{ 对应的矩阵为}K\\ [K]_{ij}=-\alpha\kappa_1(x_i,y_j)\text{ 有}\\ K=-\alpha K_1\\ \text{对任意实非零列向量}x\text{有 }x^TK_1x\geq 0,\\ x^TKx=-\alpha x^TK_1x\neq 0\text{ 恒成立}\\ \text{故}K\text{不是一个半正定的矩阵，}\kappa(x,y)\text{不是一个合法的核函数}\end{gathered}\tag{8}$$

**(e)*合法**

$$\begin{gathered}\text{由题：令}\kappa(x,y)=\kappa_1(x,y)\kappa_2(x,y)\text{ 对应的矩阵为}K\\ [K]_{ij}=\kappa_1(x_i,y_j)\kappa(x_i,y_j)\text{ 有}\\ K=K_1\odot K_2\\ \text{故}K\text{也是半正定的}\\ \kappa(x,y)\text{是一个合法的核函数}\end{gathered}\tag{9}$$

**(f)合法**

函数κ满足对称性：

$$\kappa(x,y)=\kappa 3(\phi(x),\phi(y))=\kappa 3(\phi(y),\phi(x))=\kappa(y,x)。\tag{10}$$

半正定性定义：对于任意的正整数$m$和任意的$m$个点$x_1,x_2,\ldots,x_m\in X$，核函数κ对应的$Gram$矩阵$G=[\kappa(xi,xj)]_{mx_m}$是半正定的。

所以κ3对应的Gram矩阵$G_3=[\kappa_3(\phi(xi),\phi(xj))]_{mx_m}$是半正定的。

- G的元素$G_{ij}$定义为κ(xi, xj)，即使用函数κ在原始输入空间X上计算的两个点xi和xj的核函数值。
- G3的元素$G3_{ij}$定义为κ3(ϕ(xi), ϕ(xj))，即使用函数κ3在经过ϕ映射后的新空间上计算的两个点ϕ(xi)和ϕ(xj)的核函数值。

但是我们定义的κ(x, y) = κ3(ϕ(x), ϕ(y))，这就意味着在原始输入空间X上两点x和y的核函数值κ(x, y)其实就是在新空间上这两点ϕ(x)和ϕ(y)的核函数值κ3(ϕ(x), ϕ(y))。注意到G和G3实际上是相同的矩阵，因为对于所有的i, j，我们都有$G_{ij} = \kappa(xi, xj) = \kappa_3(\phi(xi), \phi(xj)) = G_{3ij}$

所以，函数κ满足半正定性。

# 四、SVM

## (a)

优化目标：

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i:y_i=+1} \xi_i + C\sum_{i:y_i=-1} k\xi_i \tag{11}$$

约束条件为：

$$y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, m \tag{12}$$

## (b)

对每个约束条件引入拉格朗日乘子$\alpha_i$和$\beta_i$对应的拉格朗日如下：

$$\begin{aligned}&\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + C_+\textstyle\sum_{i=1}^m \xi_i + kC_-\sum_{i=1}^m \xi_i - \\ &\textstyle\sum_{i=1}^m \alpha_i\left[y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - 1 + \xi_i\right] - \sum_{i=1}^m \beta_i\xi_i\end{aligned} \tag{13}$$

其中 $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]$ 和 $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_m]$。

对偶问题为：

$$\begin{aligned}&\max_\alpha \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j \\ &0 \leq \alpha_i \leq C_i, \quad i = 1, \ldots, m \\ &\textstyle\sum_{i=1}^m y_i\alpha_i = 0\end{aligned} \tag{14}$$

KKT条件：

$$\begin{aligned}&\alpha_i\left[y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - 1 + \xi_i\right] = 0 \\ &\beta_i\xi_i = 0 \\ &\xi \geq 0 \\ &\textstyle\sum_{i=1}^m y_i\alpha_i = 0 \\ &y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - 1 + \xi_i \geq 0 \\ &\alpha_i \geq 0 \\ &\beta_i \geq 0\end{aligned} \tag{15}$$

# 五、朴素贝叶斯

## (a)

**基本假设**：“特征独立性假设”：假设每个特征之间是相互独立的。假设每个特征同等重要。

**方便**：模型的计算过程大大简化，可以高效地计算出目标类别的概率。在大规模数据集上表现良好。

**局限性**：实际应用中特征之间往往存在一定的关联性，特征独立性的假设可能并不成立。假设每个特征同等重要，这在很多情况下也是不合理的。

**朴素贝叶斯是参数化的**：在贝叶斯公式中，朴素贝叶斯需要估计的参数主要是各个类别的先验概率和各个特征在类别下的条件概率。这些都是参数，都是直接计算出的，不是优化算法迭代更新的。

**(b)**

$$P(y=A)=\log(4/12)=-1.58=P(y=B)=P(y=C)$$

$$\mu=\begin{bmatrix}2.5 & 3.5\\ 2.5 & 5.5\\ 5.5 & 2.5\end{bmatrix},\sigma=\begin{bmatrix}1.25 & 1.25\\ 1.25 & 1.25\\ 1.25 & 1.25\end{bmatrix} \tag{16}$$

- $x=(2,2)$时

  $\log_{P(x|y=A)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Ai}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Ai})^2/\sigma_{Ai}^2=-3.08$

  $\log_{P(x|y=B)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Bi}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Bi})^2/\sigma_{Bi}^2=-6.28$

  $\log_{P(x|y=C)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Ci}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Ci})^2/\sigma_{Ci}^2=-6.28$

  基于以下后验概率，分类为A类

  $\log_{P(y=A|x)}=\log_{P(y=A)}+\log_{P(x|y=A)}=-4.66$

  $\log_{P(y=B|x)}=\log_{P(y=B)}+\log_{P(x|y=B)}=-7.86$

  $\log_{P(y=C|x)}=\log_{P(y=C)}+\log_{P(x|y=C)}=-7.86$

- $x=(6,1)$时

  $\log_{P(x|y=A)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Ai}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Ai})^2/\sigma_{Ai}^2=-8.20$

  $\log_{P(x|y=B)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Bi}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Bi})^2/\sigma_{Bi}^2=-12.68$

  $\log_{P(x|y=C)}=-1/2\sum_{i=0}^{n}\log 2\pi\sigma_{Ci}^2-1/2\sum_{i=0}^{n}(x_i-\mu_{Ci})^2/\sigma_{Ci}^2=-3.08$

  基于以下后验概率，分类为C类

  $\log_{P(y=A|x)}=\log_{P(y=A)}+\log_{P(x|y=A)}=-9.78$

  $\log_{P(y=B|x)}=\log_{P(y=B)}+\log_{P(x|y=B)}=-14.26$

  $\log_{P(y=C|x)}=\log_{P(y=C)}+\log_{P(x|y=C)}=-4.66$

# 六、数据压缩

### (a)计算Ent

使用python计算出信息熵：

```python
def entropy(list):
    entropy = 0
    for i in list:
        entropy += -i*math.log(i,2)
    return entropy

#6.1
lst_x=[0.5,0.25,0.25]
lst_y=[0.5,0.5]
lst_x_=[0.5,0.5]
print(entropy(lst_x),entropy(lst_y),entropy(lst_x_))
```

```
Problem_6
D:\coding\python\python310\python.exe C:\Users\Shawn\
1.5 1.0 1.0
```

**证明：**

如果是无损压缩，则即使去掉了冗余，信息熵也应该不变。而信息熵减小了（从1.5到1.0），说明信息损失

**发生信息损失的情况：**

当我们希望得到更小长度的数据时，由于空间收到压缩，就会产生信息损失

## (b)设计编解码器

$$D + \lambda R = MSE(x, \hat{x}) + \lambda Ent(y) = 1/N \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 - \lambda \sum p(y) log(p(y)) \tag{17}$$

当使用信息熵计算码率，并希望码率最小时，应该采用哈夫曼编码。根据源符号的概率分布构造一个最优二叉树，也就是哈夫曼树。树的叶节点代表源符号，而路径上的位（0或1）构成了源符号的码字。出现概率大的，对应编码就短

使用Heffman编码并解码：（函数实现详见 problem6ab.py ）

```
data_list = [1,2,3]
prob_list =[0.5,0.25,0.25]
huffman_code, bitrate = huffman_encoding(data_list, prob_list)
encoded_data = [huffman_code[data] for data in data_list]
reconstructed_data = huffman_decoding(huffman_code, encoded_data)
Mse=mse(data_list, reconstructed_data)

print(f'Original data: {data_list}',f'  prob_list: {prob_list}',)
print(f'Encoded data: {encoded_data}',f'  bitrate : {bitrate }')
print(f'Reconstructed data: {reconstructed_data}')
print(f'MSE: {Mse}')
print(f'Performance when λ = 0.1 : {performance(data_list, reconstructed_data, bitrate, 0.1)}')
print(f'Performance when λ = 1 : {performance(data_list, reconstructed_data, bitrate, 1)}')
print(f'Performance when λ = 10 : {performance(data_list, reconstructed_data, bitrate, 10)}')
```

得到的结果如下：

```
Original data: [1, 2, 3]   prob_list: [0.5, 0.25, 0.25]
Encoded data: ['0', '11', '10']
Bitrate: 1.5 bits/symbol
Reconstructed data: [1.0, 2.0, 3.0]
MSE: 0.0
Performance when λ = 0.1 : 0.15000000000000002
Performance when λ = 1 : 1.5
Performance when λ = 10 : 15.0

Process finished with exit code 0
```

**分析：**

- 我使用的赫夫曼编码解码方法在问题（a）中的系统可以使重构误差为0，性能指标的大小完全取决于λ的大小
- 基础码率为1.5

## (c)编码的表达能力

**证明：**

当y只有两个取值时，任何解码方式都最多输出两个不同值。而输入数据有三个不同值，这就说明重构误差一定不为0，无法达到无损压缩

新的表达方式：哈夫曼编码

**新的性能指标：**使用平均比特数来衡量码率

**表达能力：**更强，毋庸置疑

**编码难度：**由于哈夫曼编码有完整成熟的生成方式，难度并不大

**编解码器设计难度**：难度提升，系统更加复杂

### (d)AE可行性分析

当 y 取值在实数集合 R 时，我们可以使用自编码器（Autoencoder）来学习一个编解码器。

在这种情况下，我们可以设计损失函数来同时考虑重构误差和码率。

1. **重构误差**：使用均方误差 (MSE) 来衡量原始数据和重构数据之间的差异。在自编码器的训练过程中最小化这个重构误差。
2. **码率**：编码数据的平均维度（即，编码数据所占用的实数数目）。我们可以通过正则化项使得编码层的神经元数量限制在一定范围内，来鼓励模型产生更小的码率。

损失函数可以设计为重构误差和码率的加权和：

```
loss = reconstruction_error + lambda * bitrate
```

其中 `reconstruction_error` 是重构误差， `bitrate` 是码率， `lambda` 是一个超参数，用于控制重构误差和码率之间的权衡。

训练时，使用Adam等优化器训练模型，使得损失函数最小

### (e)加性噪声解决连续编码

这种方法的合理性在于它有效地处理了取整这种非微分操作带来的问题，其基本思想是在训练过程中添加随机噪声，这个过程是可微的，可以产生有效的梯度，以使模型能够通过噪声对梯度进行反向传播。

当我们在训练期间将均匀噪声 $\epsilon \sim U(-0.5, 0.5)$ 添加到 y 上，我们实际上是在模拟取整操作。这是因为添加这样的噪声相当于在数值上向下和向上都偏移了最多 0.5 的距离，这与取整操作的效果类似。这样，模型就可以在训练期间“看到”类似于在测试期间会遇到的情况。

**问题的根源在于，现实世界需要的编码是取整后的，为了在训练时减少连续数据与离散数据的差距，采用加性噪声可以在数学上和统计上模拟“取整”操作，使得训练过程与测试过程接轨！**

### (f)

代码见 `Problem6.py`

保存的模型为 `'autoencoder1.pth'`

### 实验设置：

- 使用自编码器AE，encoding_dim=1
- 在forward部分**完成噪声添加和取整**

```
    def forward(self, x,flag=0):
        encoded = self.encoder(x)
        if flag=="train":
            # 编码结果添加噪声
            noise = torch.rand(encoded.shape) - 0.5
            encoded = encoded + noise
        else:
            # 测试阶段取整
            encoded = encoded.round()
        decoded = self.decoder(encoded)
        return decoded, encoded
```

- 损失函数：`MSE+L1+码率`，并将系数l1_factor和bitrate_factor设置为0.01

```
loss = mse_loss + l1_factor * l1_loss + bitrate_factor * bitrate
```

- 码率的计算方法：编码的平均比特数，即编码后数据的平均长度来得到码率。

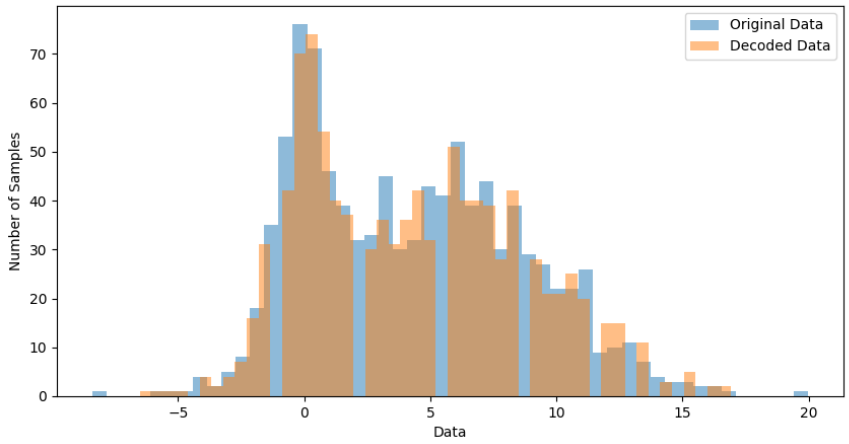
  ```
  total_bits += torch.sum(encoded_data.view(-1).int()).item()
  total_samples += encoded_data.size(0)
  bitrate = total_bits / total_samples
  ```

- epoch上限：500，早停容忍次数：20

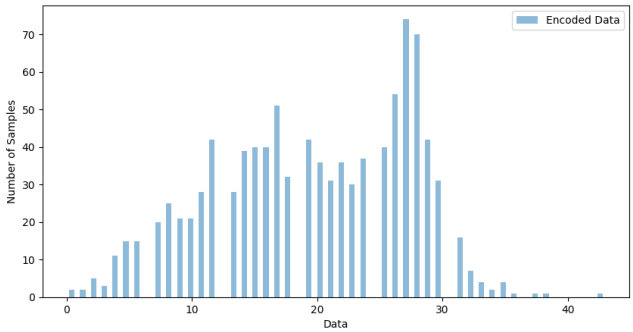
## 实验结果与分析

```
plt.hist(test_data.numpy(),bins=50, label='Original Data', alpha=0.5,rwidth=2)
plt.hist(decoded_data,bins=50, label='Decoded Data', alpha=0.5,rwidth=2)
```

数据重构后对比：（和原始混合高斯分布很相近，说明编码解码效果不错）




编码分布图：（取整后）




测试集MSE：0.042

原数据、还原后数据、整数编码仅展示前五个数据点：

```
Test Loss: 0.042277573700994255
Original Data: [ 0.07255389 -0.07256822  0.60867333  0.65046924 -0.03888251]
Decoded Data: [ 0.27114862 -0.27650684  0.81880265  0.81880265  0.27114862]
Encoded Data: [[27.]
 [28.]
 [26.]
 [26.]
 [27.]]
```

## 反思：本题走过的弯路（已纠正）

- 错误地将每个数据编码成16、32等高维向量，虽然这样近乎无损，但不是压缩
- 错误地理解原题中混合高斯的含义，错误的采样方法：
- `train_data = np.random.normal(0, 1, 10000) * 0.25 + np.random.normal(6, 4, 10000) * 0.75`
- 写代码前没能深刻理解加性噪声的神来之笔作用。本题看上去并没有做什么压缩，因为数据还是那么多个，但本题的压缩角度是**从连续实数到离散整数**的跨越