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

在教材的第 5.3 与 5.4 节中，我们已经对 PCA 的推导过程进行了详细的阐述。在第 5.4 节，我们省略了对 PCA 投影到更多维度时其余投影向量的推导。现对其进行分析。

沿用正文中的符号，将从训练集合 X 中计算得到的 x 的协方差矩阵表示为

$$\mathrm{Cov}(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T .$$

假设对 $\mathrm{Cov}(x)$ 进行特征分解之后得到的特征向量为 $\xi_1, \xi_2, ..., \xi_D$，对应的特征值为 $\lambda_1, \lambda_2, ..., \lambda_D$ 并有 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D \geq 0$。通过教材中的公式 (5.24) 可知，在一维 PCA 降维中，任意的原始输入 x_i 可被表示为

$$x_i \approx \bar{x} + \xi_1^T(x_i - \bar{x})\xi_1 .$$

在得到了一维 PCA 的降维表示后，现在对二维 PCA 的另一个降维表示 $\xi_2^T(x_i - \bar{x})\xi_2$ 进行推导。

(a) 令 $y_i = x_i - \xi_1^T(x_i - \bar{x})\xi_1$，已知 $\mathrm{Cov}(x) = \sum_{i=1}^{D}\lambda_i\xi_i\xi_i^T$ (教材中公式 (5.25))，请证明：

$$\mathrm{Cov}(y) = \sum_{i=2}^{D}\lambda_i\xi_i\xi_i^T .$$

(b) 也就是说，如果我们希望根据算法 5.1 将 y 降低到 1 维空间，得到的 $\mathrm{Cov}(y)$ 最大的特征值应该为 λ_2，其对应的特征向量应该为 ξ_2，这实际上就是原始 x 的二维 PCA 的另一个表示。现在我们希望同学能针对如下的一个输入矩阵 X 对该结论进行验证，并提交相关代码 (请将代码文件命名为 `Problem1.py`)。

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 2 \\ 1 & 2 & 9 \\ 3 & 5 & 2 \end{bmatrix} .$$

(注：X 包含 4 个样本，每个样本的输入维度为 3)

(a)

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

将 x_i 在 ξ_1 上的投影表示为 $(x_i - \xi_1)^T\xi_1 = x_i^T\xi_1 - \bar{x}^T\xi_1$

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

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

二、瑞利熵

在教材第二章的习题 2.8 中，我们已经研究了瑞利商。广义瑞利熵可以看做是瑞利熵的扩展，在这里，我们将进一步探究广义瑞利商的的一系列性质。

给定 S_B 与 S_w 为两个 $n\times n$ 的对称实矩阵，那么若存在 λ 使得方程 $S_Bw=\lambda S_ww$，则称 λ 为 S_B 相对于 S_w 的广义特征值，w 为对应的广义特征向量。

现在假设 S_w 正定，那么有排序后的广义特征值为 $\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_n$，对应的广义特征向量为 $w_1,w_2,\cdots,w_n$。

(a) 求证广义特征向量之间带权正交，即当 $i=j$ 时，$w_i^TS_ww_j=1$，否则为 0（提示：可以对 S_w 做 Cholesky 分解）。

(b) 求广义瑞利商 $J(w)=\dfrac{w^TS_Bw}{w^TS_ww}$ 的最大值和最小值。

(c) 给定 W 为一个 $n\times d$ 的矩阵，其各列向量对应于前面所述的 d 个特征向量 $w_1,w_2,\cdots,w_d$，求 $J=\dfrac{|W^TS_BW|}{|W^TS_WW|}$ 的值。

(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

在 SVM 中，核方法 (kernel method) 使得我们能将数据隐式地映射到一个新的特征空间中，从而把一个非线性分类问题转化为一个等价的线性分类问题。使用核技巧 (kernel trick)，SVM 模型进行预测的过程可以被表示为

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b \tag{1}$$

$$= \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b \tag{2}$$

$$= \sum_{i=1}^{n} \alpha_i y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b\,, \tag{3}$$

其中 $\kappa(\cdot,\cdot)$ 就是核函数 (kernel function 的，其表示的实际上就是两个向量在新的特征空间中的内积，也即相似度。核函数最自然的构造方法是显式地定义出映射后的特征，然后根据新的特征反推对应的核函数。但是因为新的特征空间一般是高维的、甚至是无穷维的，因此更常见的情况是根据具体学习问题直接定义出核函数的形式。然而，并非所有的函数都是合法的核函数，因为合法的核函数需要满足确实存在某一特征映射方式，使得该函数表示的是输入的向量在新的特征空间中的内积，而这可以通过 Mercer 条件进行判断。Mercer 条件告诉我们，一个对称的函数要是合法的核函数，需要满足对于任意的样本集合 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$，从该样本集合定义的矩阵 $K \in \mathbb{R}^n \times \mathbb{R}^n$，其中

$$[K]_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

且该矩阵 K 是半正定的。关于 Mercer 条件更加具体的描述可以参考教材中对应小节的内容。

在本题中，对于下面给定的不同的 κ，你需要判断它是否是一个合法的核函数，同时给出证明过程。

(a) $\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa_1(\boldsymbol{x},\boldsymbol{y}) + \kappa_2(\boldsymbol{x},\boldsymbol{y})$，其中 κ_1 和 κ_2 都是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上合法的核函数。

(b) $\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa_1(\boldsymbol{x},\boldsymbol{y}) - \kappa_2(\boldsymbol{x},\boldsymbol{y})$，其中 κ_1 和 κ_2 都是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上合法的核函数。

(c) $\kappa(\boldsymbol{x},\boldsymbol{y}) = \alpha\kappa_1(\boldsymbol{x},\boldsymbol{y})$，其中 κ_1 是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上合法的核函数，$\alpha \in \mathbb{R}^+$ 是一个正实数。

(d) $\kappa(\boldsymbol{x},\boldsymbol{y}) = -\alpha\kappa_1(\boldsymbol{x},\boldsymbol{y})$，其中 κ_1 是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上合法的核函数，$\alpha \in \mathbb{R}^+$ 是一个正实数。

(e) $\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa_1(\boldsymbol{x},\boldsymbol{y})\kappa_2(\boldsymbol{x},\boldsymbol{y})$，其中 κ_1 和 κ_2 都是定义在 $\mathbb{R}^d \times \mathbb{R}^d$ 上合法的核函数。

(f) $\kappa(\boldsymbol{x},\boldsymbol{y}) = \kappa_3(\phi(\boldsymbol{x}),\phi(\boldsymbol{y}))$，其中 $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$，而 κ_3 是定义在 $\mathbb{R}^{d'} \times \mathbb{R}^{d'}$ 上合法的核函数。

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

由题：令$\kappa(x,y) = \kappa_1(x,y) + \kappa_2(x,y)$ 对应的矩阵为K

$[K]_{ij} = \kappa_1(x_i,y_j) + \kappa(x_i,y_j)$ 有

$K = K_1 + K_2$

对任意实非零列向量x有 $x^TK_1x \geq 0,\ x^TK_2x \geq 0$

$x^TKx = x^TK_1x + x^TK_2x \geq 0$ 恒成立

故K是一个半正定的矩阵，$\kappa(x,y)$是一个合法的核函数 (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

在线性不可分问题下的 SVM（课本公式 7.46–48）当中，对于正样本和负样本，其在目标函数中分类错误或分对但置信度较低的代价是相同的。但是在很多不均衡分布下的应用场景中，比如负样本过少时，往往会出现负样本分类错误（即 false positive）的现象。

现在，针对课本公式 7.46–48，我们希望对负样本分类错误或分对但置信度较低的样本施加 $k > 0$ 倍于正样本中被分错的或者分对但置信度较低的样本的代价。此时：

(a) 请给出相应的 SVM 优化问题。

(b) 请推导出相应的对偶问题及 KKT 条件。

(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α_i和β_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：

$$\begin{aligned}&\max_\alpha \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j \\ &0 \leq \alpha_i \leq C_i, \quad i = 1, \ldots, m \\ &\textstyle\sum_{i=1}^m y_i \alpha_i = 0\end{aligned} \tag{16}$$

KKT条件：

$$\begin{aligned}&\alpha_i\left[y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - 1 + \xi_i\right] = 0 \\ &\beta_i\xi_i = 0 \\ &\xi \geq 0 \\ &\textstyle\sum_{i=1}^m y_i\alpha_i = 0 \\ &y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - 1 + \xi_i \geq 0 \\ &\alpha_i \geq 0 \\ &\beta_i \geq 0\end{aligned} \tag{17}$$

五、朴素贝叶斯

朴素贝叶斯是一种适用于分类的有监督学习算法。请查阅相关资料，并根据你的理解完成此题。

(a) 朴素贝叶斯所提出的基本假设是什么？这种假设带来了什么方便与局限？经典的朴素贝叶斯是参数化还是非参数化的？

(b) 高斯朴素贝叶斯算法是一种基于贝叶斯定理和特征条件独立性假设的分类方法。对于连续的数据，它假定每个类别的各个特征都服从高斯分布。给定以下三个类别和对应的训练样本：
类别 A：[(1,2),(2,3),(3,4),(4,5)]
类别 B：[(1,4),(2,5),(3,6),(4,7)]
类别 C：[(4,1),(5,2),(6,3),(7,4)]
请使用高斯朴素贝叶斯算法，对以下数据进行分类：(2,2), (6,1)，并写出详细过程。

(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)

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

$$\mu=\begin{bmatrix}2.5 & 3.5\\ 2.5 & 5.5\\ 5.5 & 2.5\end{bmatrix}, \sigma=\begin{bmatrix}1.25 & 1.25\\ 1.25 & 1.25\\ 1.25 & 1.25\end{bmatrix} \tag{18}$$

- $x=(2,2)$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

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

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

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

- $x=(6,1)$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

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

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

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

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

$$D+\lambda R=MSE(x,\hat{x})+\lambda Ent(y)=1/N\sum_{i=1}^{N}(x_i-\hat{x}_i)^2-\lambda\sum p(y)log(p(y)) \tag{19}$$

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 problem_6.py ）

```python
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

数据压缩是一种实用技术，常用于图像压缩、视频压缩、音频压缩中。数据压缩通常分为有损压缩和无损压缩。有损压缩指在压缩过程中存在信息损失，无法还原原始数据。因其有较高的压缩比，故有损压缩的应用更加广泛。下面是一个有损压缩模型。

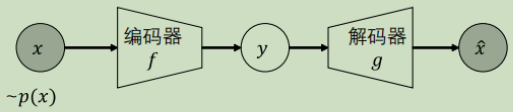


Figure 1: 有损压缩模型

通常假定原始数据分布为 $p(x)$，该分布是未知的。当我们从分布中采样一个 x，根据编码器 f 得到 y，再根据解码器 g 得到 $\hat{x}$。这里的 y 称为 x 的一个表示，$\hat{x}$ 称为 x 的重构。在有损压缩中，x 和 $\hat{x}$ 通常存在差异。在实际的压缩系统中，x 可能是实数、向量、矩阵等，y 可能是整数、二进制数等。这里我们首先考虑 x 是实数，y 是整数的情况。

(a) 首先我们举一个例子来帮助理解有损压缩模型。假设原始数据分布为一个离散分布：

$$P(x=1)=0.5, P(x=2)=0.25, P(x=3)=0.25.$$

编码器为：

$$f(x)=\begin{cases}0, & \text{若}x=1,\\ 1, & \text{否则.}\end{cases}$$

表示 y 的取值范围为 $\{0,1\}$，分布为：

$$P(y=0)=0.5, P(y=1)=0.5$$

解码器为：

$$g(y)=\begin{cases}1, \text{若}y=0,\\ 2, \text{否则.}\end{cases}$$

分别求 x，y，$\hat{x}$ 的信息熵，并证明该系统是有损的。试分析该系统什么情况下会导致信息损失。

(b) **压缩系统的性能**。对于有损压缩，需要考虑两方面的性能指标。其一是重构数据与原始数据的差距，简称为重构误差（D）；其二是表示 y 所产生的编码长短，简称为码率（R）。二者的权衡使用 λ 控制，即最终的性能指标为 $D+\lambda R$。本题中，重构误差用均方误差计算，编码长短用信息熵计算。试写出性能指标的完整公式。当我们希望码率最小时，应该如何设计编码器和解码器。试与问题（a）中的系统对比性能，分析它们分别在 $\lambda=0.1,1,10$ 情况下的优劣。

(c) y **的表达能力**。论证当 $y\in\{0,1\}$ 时，系统无法达到无损压缩。为了增加 y 的表达能力，我们可以让 $y\in\mathbb{Z}$ 或 $y\in\mathbb{R}$，试写出新的性能指标，并从表达能力、编码难度和编解码器设计难度三个方面比较两者的优劣。

(d) **基于机器学习的编解码器**。从上述题目中，我们的编解码器由手工设计。能否自动地学习出编解码器？试分析使用神经网络模型学习编解码器的可行性，注意这里使用拟合连续函数的神经网络，并且令 $y\in\mathbb{R}$。重点分析损失函数应该如何设计，码率应该如何计算和优化。

(e) **解决连续变量无法编码的问题**。连续变量难以编码，需将其转为离散变量。一个可能的解决方案如下图所示。

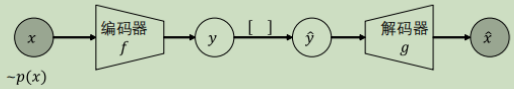


Figure 2: 有损压缩模型，量化表示 y

图中 $[\,]$ 表示取整操作。然而取整操作难以产生有效梯度。加性噪声是一个可能的解决方案。在训练过程中，$\hat{y}=y+\epsilon$，其中 $\epsilon\sim U(-0.5,0.5)$；在测试过程中，$\hat{y}=[y]$。试分析这样做的合理性（注：本题需要用到两个随机变量的函数分布，是基础概率统计的知识点）。

(f) **编程**。我们使用教材中公式（8.25）的数据，即

$$0.25N(x;0,1)+0.75N(x;6,4),$$

将其作为原始数据分布，从中抽样 10000 个样本点作为训练集，1000 样本点作为验证集，1000 样本点作为测试集。根据问题（e），尝试编程实现一个基于神经网络的数据压缩系统，完成对原始数据的压缩和重构。请在此处呈现你的实验结果和分析，并将代码文件命名为 `Problem6.py`。

(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

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。

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，同时保持了模型在训练和测试期间的行为一致。

(f)

代码见 `Problem6.py`

保存的模型为 `'autoencoder1.pth'`

方法：自编码器

encoding_dim=16，将每个数据点压缩成16维向量

```
class Autoencoder(nn.Module):
    def __init__(self, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(1, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, encoding_dim),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
```

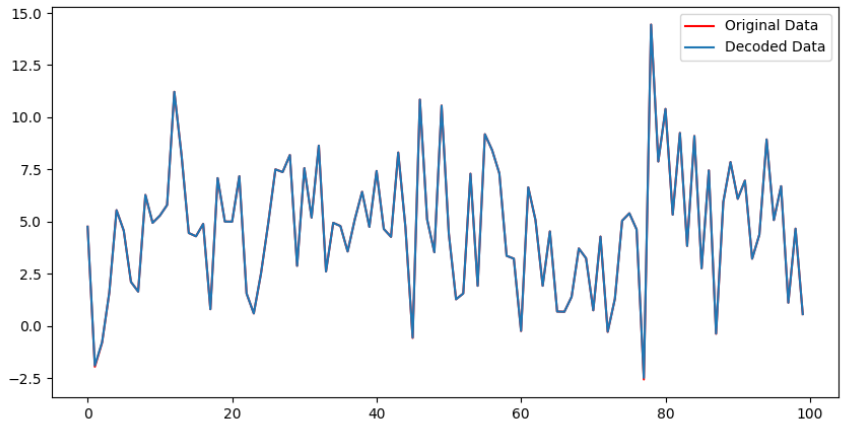
```
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded, encoded
```

损失函数：`MSE+L1+码率`

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

实验结果：

（仅展示前100数据点）



（原数据与还原数据仅展示前五个数据点，编码示例仅展示一个数据）

```
Epoch: 98, Train Loss: 0.2304462492465973, Bitrate: 6.4508,   Validation MSE Loss: 0.00686642589244002
Epoch: 99, Train Loss: 0.23011475801467896, Bitrate: 6.4865,   Validation MSE Loss: 0.006926156163899577
Epoch: 100, Train Loss: 0.23052141070365906, Bitrate: 6.5352,   Validation MSE Loss: 0.006830317606727476
Test Loss: 0.017552932346006855
Original Data: [ 4.7632327 -1.9430163 -0.7907077  1.6117567  5.5541186]
Decoded Data: [ 4.7613955 -1.935258  -0.8310672  1.6233959  5.5489   ]
Encoded Data: [1.02996861e-03 7.29011536e+00 8.72540841e-05 7.56089503e-05
 6.66327076e-04 0.00000000e+00 3.30301380e-04 1.20369616e-04
 8.14895611e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 8.55880789e-05]
```

分析

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
```

在损失函数中同时优化MSE+L1+码率，并将系数l1_factor和bitrate_factor设置为0.01

减小encoding_dim可以得到更大的压缩效果

基于此得到压缩