机器学习与数据挖掘

作业 2：支持向量机

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1 问题描述与实验要求

1.1 问题描述

根据提供的数据，训练一个采用在不同的核函数的支持向量机 SVM 的 2 分类器，并验证其在测试数据集上的性能。

1.2 实验要求

(1) 考虑两种不同的核函数：

(a) 线性核函数；

(b) 高斯核函数；

(2) 可以直接调用现成 SVM 软件包来实现；

(3) 手动实现采用 hinge loss 和 cross-entropy loss 的线性分类模型，并比较它们的性能；

2 算法原理

2.1 支持向量机（SVM）

支持向量机（Support Vector Machines, SVM）是一类广为使用的分类方法，在多种分类问题中都表现出了很高的准确性。简单地来说，SVM 通过寻找一个最佳的分类边界，将样本特征 $\boldsymbol{x}$ 的域划分为不同的区域。落在同一个区域内的样本全都属于同一个类，且不同的区域对应不同的类别。

2.1.1 线性可分二分类问题下的 SVM

我们首先介绍最简单形式的 SVM。假设我们遇到的问题是线性可分二分类的，即样本有两种类别、可以被线性边界完全区分。给定训练样本集 $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(N)}, y^{(N)})\}, y^{(l)} \in \{-1, +1\}$。那么，分类边界将是一个超平面，它将样本空间划分为两部分，落在不同部分的样本属于不同类别。

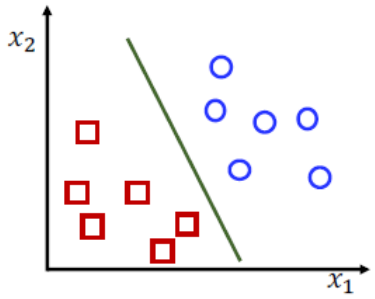

图 1: 二维空间下，分类边界是一条红色直线，分开了两个类别的样本

我们使用两个参数来描述这个超平面：$\boldsymbol{w}$ 和 b。在 d 维空间中，一个超平面可以使用如下线性方程来描述：

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0 \tag{1}$$

其中 $\boldsymbol{w} = (w_1; w_2; \cdots; w_d)$ 决定了超平面的方向，超平面垂直于 $\boldsymbol{w}$，故 $\boldsymbol{w}$ 的方向被称为超平面的法向量。b 是位移项，决定了超平面与原点之间的距离。显然，若点 $\boldsymbol{x}$ 在超平面上，那么 $\boldsymbol{w}^T\boldsymbol{x} + b = 0$。如果超平面 $(\boldsymbol{w}, b)$ 能将训练样本正确分类，则对于 $(\boldsymbol{x}^{(l)}, y^{(l)}) \in D$，当 $y^{(l)} = +1$ 时有 $\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b > 0$，当 $y^{(l)} = -1$ 时有 $\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b < 0$。我们可以将上述要求合并在一起，即 $y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b) > 0$。更近一步地来说，我们可以通过定义一个损失函数来描述这种优化目标：

$$L(\boldsymbol{w}, b) = \sum_{l=1}^{N} e(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b)) \tag{2}$$

其中 $e(z)$ 是一种特殊的函数，$z \geq 0$ 时 $e(z) = 0$，$z < 0$ 时 $e(z) = 1$。

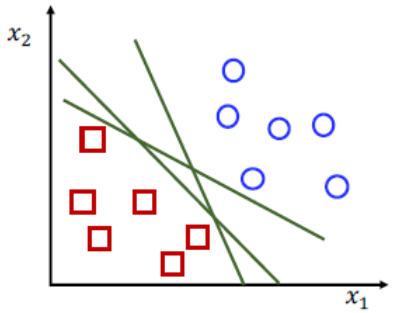

图 2: 很多情况下存在多个满足条件的超平面

然而，在很多情况下，我们可以找到多对满足条件的 $(\boldsymbol{w}, b)$。从泛化能力的角度上看，我们认为一个最优超平面应当与两个类别样本簇的间隔（margin）尽可能远，这样才能使得这个超平面对于未知数据也有准确的预测，避免由于未知数据的轻微波动导致被错误分类。在定义间隔前，我们需要知道如何计算一个样本点 $\boldsymbol{x}$ 到超平面的距离。如图3所示，$\boldsymbol{x}$ 可以分解为两个向量 $\boldsymbol{m}_1$ 和 $\boldsymbol{m}_2$，其中 $\boldsymbol{m}_1$ 在超平面上即 $\boldsymbol{w}^T\boldsymbol{m}_1 + b = 0$，而 $\boldsymbol{m}_2$ 与超平面垂直。因此我们有：

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = \boldsymbol{w}^T(\boldsymbol{m}_1 + \boldsymbol{m}_2) + b = \boldsymbol{w}^T\boldsymbol{m}_2 \tag{3}$$

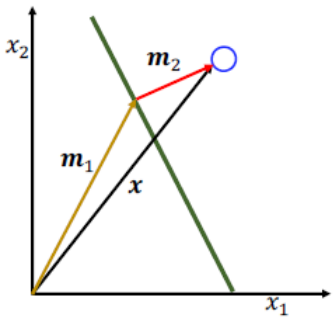

图 3: 将 $\boldsymbol{x}$ 分解为在超平面上的 $\boldsymbol{m}_1$ 和垂直超平面的 $\boldsymbol{m}_2$

因为 $\boldsymbol{m}_2$ 垂直于超平面，故 $\boldsymbol{m}_2$ 与 $\boldsymbol{w}$ 平行。因此假设 $\boldsymbol{m}_2$ 的长度为 $|\gamma|$，我们有：

$$\boldsymbol{m}_2 = \gamma \cdot \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \tag{4}$$

把式4代入式3中，我们便可以得到：

$$\gamma = \frac{\|\boldsymbol{w}\| h(\boldsymbol{x})}{\boldsymbol{w}^T\boldsymbol{w}} = \frac{h(\boldsymbol{x})}{\|\boldsymbol{w}\|} \tag{5}$$

当 $\boldsymbol{m}_2$ 与 $\boldsymbol{w}$ 方向相同时，$\gamma > 0$，反之 $\gamma < 0$。显然，$|\gamma|$ 就是 $\boldsymbol{x}$ 与超平面之间的距离。

我们将一个超平面和一个样本集之间的间隔定义为这个样本集中离超平面最近的样本点与超平面之间的距离。在当前情况下，最大化超平面与两个类别的样本集之间的间隔，等价于最大化超平面与整个数据集之间的间隔。因此，超平面与训练样本集之间的距离为：

$$margin = \min_{l} |\gamma^{(l)}| = \min_{l} \frac{|\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b|}{\|\boldsymbol{w}\|} \tag{6}$$

此外，在前面我们就已经说明了对于一个线性可分的二分类数据集，存在一个超平面可以完全区分样本类别。对于这个超平面，$y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) = |\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b| > 0$，所以，我们的优化问题如下所示：

$$\max_{\boldsymbol{w},b} \min_{1 \le l \le N} \frac{y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b)}{\|\boldsymbol{w}\|} \tag{7}$$

$$\text{s.t.} \quad y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) > 0, \quad 1 \le l \le N \tag{8}$$

最优 $\boldsymbol{w}^*$ 和 b^* 为 $(\boldsymbol{w}^*, b^*) = \underset{(\boldsymbol{w},b)}{\operatorname{argmax}} \min_{1 \le l \le N} \frac{y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b)}{\|\boldsymbol{w}\|}$。

如何优化、获取 $\boldsymbol{w}^*$ 和 b^* 是一个非常复杂的问题。我们尝试将这两个公式进行各种简化。显然，对于最优解 $(\boldsymbol{w}^*, b^*)$，$(c\boldsymbol{w}^*, cb^*)$ 也是最优解（因为对应于同一个超平面），其中 $c \in \mathbb{R}$ 是任意非零常量。不妨令 $c = \frac{1}{\min_{1 \le l \le N} y^{(l)}((\boldsymbol{w}^*)^T \boldsymbol{x}^{(l)} + b^*)}$，那么把 $(c\boldsymbol{w}^*, cb^*)$ 代入，我们有：

$$\min_{1 \le l \le N} y^{(l)}((c\boldsymbol{w}^*)^T \boldsymbol{x}^{(l)} + cb^*) = 1 \tag{9}$$

式9等价于 $y^{(l)}((c\boldsymbol{w}^*)^T \boldsymbol{x}^{(l)} + cb^*) \ge 1$。换言之，我们可以将目标由求得 $(\boldsymbol{w}^*, b^*)$ 转变为求得 $(c\boldsymbol{w}^*, cb^*)$，即令 $\boldsymbol{w}^* := c\boldsymbol{w}^*$、$b^* := cb^*$，那么优化问题简化为：

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|} \tag{10}$$

$$\text{s.t.} \quad y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) \ge 1, \quad 1 \le l \le N \tag{11}$$

更进一步，最大化 $\frac{1}{\|\boldsymbol{w}\|}$ 相当于最小化 $\|\boldsymbol{w}\|$，也相当于最小化 $\frac{1}{2}\|\boldsymbol{w}\|^2$，我们便得到进一步简化：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 = \min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} \tag{12}$$

$$\text{s.t.} \quad y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) \ge 1, \quad 1 \le l \le N \tag{13}$$

式12和式13被称为 SVM 的原始形式。我们可以直接优化原始形式来获得 $(\boldsymbol{w}^*, b^*)$。不过，我们也可以通过 SVM 的对偶形式获得 $(\boldsymbol{w}^*, b)$。我们使用拉格朗日乘子法，为每个不等式约束定义一个拉格朗日乘子 α_i，可以得到拉格朗日函数：

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} - \sum_{l=1}^{N} \alpha_l (y^{(l)}(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) - 1) \tag{14}$$

$$\text{s.t.} \quad \alpha_l \ge 0, \quad 1 \le l \le N \tag{15}$$

其中 $\boldsymbol{\alpha} = (\alpha_1; \alpha_2; \cdots; \alpha_n)$ 是拉格朗日乘子组成的向量。由于当 $\alpha_l \ge 1$ 时，$L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ 越小，那么 $\frac{1}{2}\boldsymbol{w}^T \boldsymbol{w}$ 越小、$y_i(\boldsymbol{w}^T \boldsymbol{x}^{(l)} + b) - 1$ 越大，这与原始优化目标相同，因此，$L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ 相当于一种损失函数。计算梯

度并置零，我们可以得到：

$$\frac{\partial L}{\partial w} = \mathbf{0} \Rightarrow \boldsymbol{w} = \sum_{l=1}^{N} \alpha_l y^{(l)} \boldsymbol{x}^{(l)} \tag{16}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{l=1}^{N} \alpha_l y^{(l)} = 0 \tag{17}$$

把式16和式17代入拉格朗日函数，我们可以得到：

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{l=1}^{N} \alpha_l (y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b) - 1) \\ &= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \boldsymbol{w}^T \sum_{l=1}^{N} \alpha_l y^{(l)} \boldsymbol{x}^{(l)} - \sum_{l=1}^{N} \alpha_l y^{(l)} b + \sum_{l=1}^{N} \alpha_l \\ &= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - 0 + \sum_{l=1}^{N} \alpha_l \\ &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} (\boldsymbol{x}^{(i)})^T \boldsymbol{x}^{(j)} + \sum_{l=1}^{N} \alpha_l \end{aligned}$$

我们消去了 $\boldsymbol{w}$ 和 b，或者说我们得到了 $\min\limits_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\alpha})$。我们就可以获得 SVM 的对偶形式：

$$\max_{\boldsymbol{\alpha}} \quad \sum_{l=1}^{N} \alpha_l - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} (\boldsymbol{x}^{(i)})^T \boldsymbol{x}^{(j)} \tag{18}$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad 1 \leq i \leq n \tag{19}$$

$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0 \tag{20}$$

通过求解 SVM 的对偶形式，我们可以得到最优 $\boldsymbol{\alpha}^*$。代入式16，我们就可以得到 $\boldsymbol{w}^*$：

$$\boldsymbol{w}^* = \sum_{l=1}^{N} \alpha_l^* y^{(l)} \boldsymbol{x}^{(l)} \tag{21}$$

由于对偶形式达到最优时，对所有 $1 \leq i \leq N$，有 $\alpha_l(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b) - 1) = 0$（此时 $L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$，与原始问题的目标函数匹配）。而 $\alpha_l \geq 0$，故当存在 $\alpha_l > 0$ 时，有 $y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b) = 1$。假设存在数据集 $S \subseteq D$，使得对于 $(\boldsymbol{x}^{(l)}, y^{(l)}) \in S$，有 $y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b) = 1$。把 $\boldsymbol{w}^*$ 和 $\boldsymbol{\alpha}^*$ 代入，可得：

$$b^* = \frac{1}{N_S} \sum_{(\boldsymbol{x},y) \in S} (y - (\boldsymbol{w}^*)^T \boldsymbol{x}) \tag{22}$$

得到 $\boldsymbol{w}^*$ 和 b^* 后，我们就可以预测一个样本 $\boldsymbol{x}$ 所属的类别：

$$\hat{y} = sign((\boldsymbol{w}^*)^T \boldsymbol{x} + b^*) \tag{23}$$

2.1.2 不可分问题下的线性 SVM

在2.1.1中，我们讨论的是数据集绝对线性可分的情况。然而大多数情况下，数据集并不能满足这个要求，即找不到一个超平面可将两个类完美分开。为了解决这种情况，我们可以使用软间隔，通过使用松弛变量使得 SVM 可以处理大致线性可分的情况。

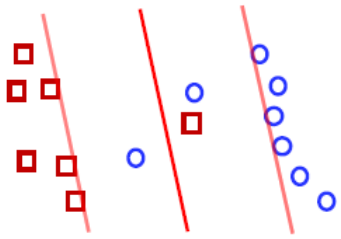

图 4: 一种不满足线性可分的情况

正如图4所示，存在一些样本点不可能被正确分类。为了容许这种情况（即出现样本 $(\boldsymbol{x}^{(l)}, y^{(l)})$ 使得 $y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)}+b)<1$），我们引入松弛变量 ξ_l，从而把约束 $y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)}+b)\geq 1$ 修改成：

$$y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)}+b)\geq 1-\xi_l \tag{24}$$

$$\xi_l \geq 0 \tag{25}$$

显然，当 $\xi_l=0$ 时，原始约束仍然成立；当 $0<\xi_l<1$ 时，该样本仍可被正确分类，但与超平面之间的间隔更小；当 $\xi_l\geq 1$ 时，该样本会被错误分类。我们可以视 ξ_l 为代价或惩罚。为使总代价最小，我们往目标函数添加一个正则项，最终得到修改后的线性 SVM 形式化：

$$\min_{\boldsymbol{w},b}\ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+C\sum_{l=1}^{N}\xi_l \tag{26}$$

$$\text{s.t.}\quad y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)}+b)\geq 1-\xi_l,\quad 1\leq l\leq N \tag{27}$$

$$\xi_l\geq 0,\quad 1\leq l\leq N \tag{28}$$

其中常数 C 用于确定间隔 $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ 和总代价 $\sum_{l=1}^{N}\xi_l$ 之间的相对重要性。使用与2.1.1相同的方法，我们可以得到新的对偶形式：

$$\max_{\boldsymbol{\alpha}}\ \sum_{l=1}^{N}\alpha_l-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}(\boldsymbol{x}^{(i)})^T\boldsymbol{x}^{(j)} \tag{29}$$

$$\text{s.t.}\quad C\geq\alpha_i\geq 0,\quad 1\leq i\leq n \tag{30}$$

$$\sum_{i=1}^{N}\alpha_i y^{(i)}=0 \tag{31}$$

最终得到 $\boldsymbol{w}^*$ 和 b^*，从而通过式23获得样本的预测类别。

2.1.3 核 SVM

除了线性分类问题下的 SVM，还有一种非线性 SVM，其使用一个复杂超曲面而不是超平面来充当分类边界。为了得到非线性 SVM，我们首先将原始样本特征 $\boldsymbol{x}$ 通过映射 ϕ 映射到更高维的特征空间：

$$\phi:\boldsymbol{x}\to\phi(\boldsymbol{x}) \tag{32}$$

然后，线性 SVM 的原始形式改写为：

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{l=1}^{N}\xi_l \tag{33}$$

$$\text{s.t.} \quad y^{(l)}(\boldsymbol{w}^T\phi(\boldsymbol{x}^{(l)}) + b) \geq 1 - \xi_l, \quad 1 \leq l \leq N \tag{34}$$

$$\xi_l \geq 0, \quad 1 \leq l \leq N \tag{35}$$

通过映射，我们把（输入空间中的）一个非线性分类问题转换为在（通常维度更高的）特征空间中一个等效的线性分类问题。

同样，对偶形式改写为：

$$\max_{\boldsymbol{\alpha}} \ \sum_{l=1}^{N}\alpha_l - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\phi(\boldsymbol{x}^{(i)})^T\phi(\boldsymbol{x}^{(j)}) \tag{36}$$

$$\text{s.t.} \quad C \geq \alpha_i \geq 0, \ \ 1 \leq i \leq n \tag{37}$$

$$\sum_{i=1}^{N}\alpha_i y^{(i)} = 0 \tag{38}$$

我们可以通过核技巧，使用 $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ 替换 $\phi(\boldsymbol{x}^{(i)})^T\phi(\boldsymbol{x}^{(j))}$。核函数 $k(\boldsymbol{x}, \boldsymbol{y})$ 是某些函数 $\phi(\boldsymbol{x})$ 的内积，即：

$$k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{y}) \tag{39}$$

判断一个函数 $k(\boldsymbol{x}, \boldsymbol{y})$ 是否是核函数，需要检查其是否满足 Mercer 条件。若 $k(\boldsymbol{x}, \boldsymbol{y})$ 满足对任意平方可积函数 $g(\boldsymbol{x})$，均有 $\iint k(\boldsymbol{x}, \boldsymbol{y})g(\boldsymbol{x})g(\boldsymbol{y})\mathrm{d}x\mathrm{d}y \geq 0$；且 $k(\boldsymbol{x}, \boldsymbol{y})$ 是对称的即 $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{y}, \boldsymbol{x})$，那么 $k(\boldsymbol{x}, \boldsymbol{y})$ 就是一个核函数，存在映射 ϕ 使得 $k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$。

通过核函数 $k(\boldsymbol{x}, \boldsymbol{y})$，我们就可以简化对偶形式，不必计算 ϕ 的内积：

$$\max_{\boldsymbol{\alpha}} \ \sum_{l=1}^{N}\alpha_l - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \tag{40}$$

$$\text{s.t.} \quad C \geq \alpha_i \geq 0, \ \ 1 \leq i \leq n \tag{41}$$

$$\sum_{i=1}^{N}\alpha_i y^{(i)} = 0 \tag{42}$$

相应的，预测公式变为：

$$\hat{y} = sign(\boldsymbol{w}^*\phi(\boldsymbol{x}) + b^*) \tag{43}$$

$$= sign(\sum_{l=1}^{N}\alpha_l^* y^{(l)}\phi(\boldsymbol{x}^{(l)})^T\phi(\boldsymbol{x}) + b^*) \tag{44}$$

$$= sign(\sum_{l=1}^{N}\alpha_l^* y^{(l)}k(\boldsymbol{x}^{(l)}, \boldsymbol{x}) + b^*) \tag{45}$$

可以看到，我们不需要知道 $\boldsymbol{w}^*$ 和 $\phi(\boldsymbol{x})$，只需要得到 $\boldsymbol{\alpha}^*$ 和 b^*，就可对样本进行预测。

目前有很多常用的核函数。线性核函数 $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y}$ 会将核 SVM 等效为线性分类器。而高斯核函数 $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{y}\|^2)$ 也称为 RBF 核函数，是一种非线性核函数。

2.2 hinge loss

2.2.1 hinge loss 定义

hinge loss 是一种特殊的损失函数。对于 x，其 hinge 损失 x_+ 为：

$$x_+ = \max(0, 1-x) \tag{46}$$

为了后续推导方便，我们定义 $h(x) = x_+ = \max(0, 1-x)$。显然，当 $x < 1$ 时，$h'(x) = -1$；当 $x = 1$ 时，$h'(x)$ 不存在；当 $h(x) > 1$ 时，$h'(x) = 0$。

2.2.2 hinge loss 线性分类模型与线性 SVM 模型之间的关系

hinge loss 可以应用到线性分类模型中。同样，给定 $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \cdots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$，$y^{(l)} \in \{-1, +1\}$。假设 $\boldsymbol{w} = (w_1; w_2; \cdots; w_d)$ 和 b，我们有 $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$。那么，给定样本特征 $\boldsymbol{x}$，相应的预测为：

$$\hat{y} = sign(f(\boldsymbol{x})) = sign(\boldsymbol{w}^T\boldsymbol{x} + b) \tag{47}$$

接下来我们要构造优化目标。当 $y^{(l)} = +1$ 时，我们希望 $f(\boldsymbol{x}^{(l)}) > 0$ 且越大越好；当 $y^{(l)} = -1$ 时，我们希望 $f(\boldsymbol{x}^{(l)}) < 0$ 且越小越好。所以，损失函数如下所示：

$$L(\boldsymbol{w}, b) = \frac{1}{N}\sum_{l=1}^{N} h(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b)) \tag{48}$$

当 $y^{(l)} = +1$ 时，$f(\boldsymbol{x}^{(l)})$ 越大，那么 $h(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}+b))$ 越小；当 $y^{(l)} = -1$ 时，$f(\boldsymbol{x}^{(l)})$ 越小，那么 $h(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}+b))$ 越小。因此，$(\boldsymbol{w}^*, b^*) = \underset{(\boldsymbol{w},b)}{argmin}\, L(\boldsymbol{w}, b)$。

与线性回归相似，我们可以使用梯度下降的方法求得 $\boldsymbol{w}^*$ 和 b^*。为 $\boldsymbol{x}^{(l)}$ 添加偏置 1 得到 $\tilde{\boldsymbol{x}}$，从而把 $\boldsymbol{w}$ 和 b 合并成 $\tilde{\mathrm{w}}$，把 $L(\boldsymbol{w}, b)$ 改写成：

$$L(\tilde{\boldsymbol{w}}) = \frac{1}{N}\sum_{l=1}^{N} h(y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)})) \tag{49}$$

求偏导得

$$\frac{\partial L}{\partial \tilde{\boldsymbol{w}}} = \frac{1}{N}\sum_{l=1}^{N} \frac{\partial h(y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))}{\partial (y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))} \frac{\partial (y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))}{\partial \tilde{\boldsymbol{w}}} \tag{50}$$

$$= \frac{1}{N}\sum_{l=1}^{N} \frac{\partial h(y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))}{\partial (y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))} y^{(l)}(\tilde{\boldsymbol{x}}^{(l)})^T \tag{51}$$

$$\frac{\partial h(y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))}{\partial (y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}))} = \begin{cases} -1, & \text{if } y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}) < 1; \\ 0, & \text{if } y^{(l)}(\tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}^{(l)}) \geq 1. \end{cases} \tag{52}$$

其中，当 $z = 1$ 时，$\frac{\partial h(z)}{\partial z}$ 本应无解，但为实现方便，我设定此时导数为 0。

需要留意的是，对于添加了松弛变量的 SVM 模型，由式26、式27和式28可得优化目标为 $\min\limits_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{l=1}^{N}\xi_l$，其中 $\xi_l \geq 0$ 且 $\xi_l \geq 1 - y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b)$。故我们可以得到 $\xi_l = h(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b))$。使用 hinge 损失，原始 SVM 可以重写为：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{l=1}^{N} h(y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)} + b)) \tag{53}$$

将式48和53进行比较，我们可以发现，若忽略掉无关紧要的 $\frac{1}{N}$ 和 C 的话，采用 hinge loss 的线性分类模型本质上与 SVM 模型一致，不同的地方在于 SVM 模型加上了一个 L2 正则化即 $\frac{1}{2}\|\mathrm{w}\|^2$。

总而言之，SVM= 采用 hinge loss 的线性分类模型 +L2 正则化。

3 关键代码

3.1 数据预处理

与实验 1 相同，本次实验中，我使用到了标准化。数据预处理的关键代码如下所示：

```
# 获取训练集数据
train_images = np.load("material/train-images.npy")
train_labels = np.load("material/train-labels.npy")
# 添加偏置1
extend = np.ones(train_images.shape[0])
train_X = np.c_[extend, train_images]
train_Y = train_labels
# 特征标准化
meanVal = []
stdVal = []
for i in range(train_X.shape[1]):
    meanVal.append(np.mean(train_X[:, i]))
    stdVal.append(np.std(train_X[:,i]))
    if stdVal[i] != 0:
        train_X[:, i] = (train_X[:,i]-meanVal[i])/stdVal[i]
# 打乱数据
state = np.random.get_state()
np.random.shuffle(train_X)
np.random.set_state(state)
np.random.shuffle(train_Y)

# 获取测试集数据
test_images = np.load("material/test-images.npy")
test_labels = np.load("material/test-labels.npy")
# 添加偏置1
extend = np.ones(test_images.shape[0])
test_X = np.c_[extend, test_images]
test_Y = test_labels
# 特征标准化
for i in range(test_X.shape[1]):
    if stdVal[i] != 0:
        test_X[:, i] = (test_X[:,i]-meanVal[i])/stdVal[i]
# 打乱数据
state = np.random.get_state()
np.random.shuffle(test_X)
np.random.set_state(state)
np.random.shuffle(test_Y)
```

首先，使用 np.load() 加载数据集，然后为样本特征手动添加偏置 1。接下来，对样本特征进行标准化，并打乱数据。对测试集的处理与训练集相似，不同之处在于由于测试集理论上是“未知”的，故使用训练集的均值和方差对测试集进行标准化。

3.2 SVM

本次实验中，我使用 sklearn 实现 SVM。在获取并预处理数据集后，构建并训练线性 SVM 的关键代码如下所示：

```
from sklearn import svm
......
LinearSvc = svm.SVC(C=1.0, kernel='linear', random_state=0)
model1 = LinearSvc.fit(train_X, train_Y)
```

svm.SVC() 有很多参数可以设置。当 kernel='linear' 时 SVM 使用线性核函数，此时可以设置的参数有 C 和 random_state 等。C 即式26中的权重因子 C，默认值为 1；random_state 表示在混洗数据时所使用的伪随机数发生器的种子。

训练好模型后，即可计算其在数据集上的准确率：

```
print("train acc:\t%f"%(model1.score(train_X,train_Y)))
print("test acc:\t%f"%(model1.score(test_X,test_Y)))
```

构建并训练高斯核 SVM 的关键代码如下所示：

```
RbfSvc = svm.SVC(C=1.0, kernel='rbf', random_state=0, gamma='auto')
model2 = RbfSvc.fit(train_X, train_Y)
```

kernel='rbf' 表示使用高斯核函数。当 SVM 使用高斯核函数时，有更多超参数可以设置。C 和 random_state 与线性核 SVM 的意义和设置相同，gamma 是高斯核函数系数，当设置为 auto 时代表其值为样本特征数的倒数。训练完成后，我们同样可以用 score() 获取相应数据集的准确率。

3.3 线性分类模型

与上次实验相同，在线性分类模型中，我们需要实现模型的整个训练、梯度更新与预测过程。

对于 hinge loss 模型，梯度更新函数、预测函数和损失函数如下所示：

```
def train(func, W, train_X, train_Y, Lambda, learning_rate):
    if func=='hinge':
        temp = np.dot(train_X, W)*train_Y
        temp2 = np.zeros_like(temp)
        for i in range(temp.shape[0]):
            if temp[i]<1:
                temp2[i,0] = -1
        dW = (1/train_X.shape[0])*np.dot(train_X.T, temp2*train_Y) + Lambda*W
        W -= learning_rate*dW
        return W
    elif func=='cross':
        ......
```

```
13
14 def predict(func, W, X):
15     prediction = np.dot(X, W)
16     if func=='hinge':
17         for i in range(prediction.shape[0]):
18             if prediction[i,0]>0:
19                 prediction[i,0] = 1
20             elif prediction[i,0]<0:
21                 prediction[i,0] = -1
22         return prediction
23     elif func=='cross':
24         ......
25
26 def HingeLoss(W, X, Y):
27     temp = np.dot(X, W)*Y
28     loss = 0.0
29     for i in range(temp.shape[0]):
30         if temp[i,0]<1:
31             loss += 1-temp[i,0]
32     return loss/X.shape[0]
```

对于 cross-entropy loss 模型，梯度更新函数、预测函数和损失函数如下所示：

```
1  def train(func, W, train_X, train_Y, Lambda, learning_rate):
2      if func=='hinge':
3          ......
4      elif func=='cross':
5          temp = np.dot(train_X, W)
6          for i in range(temp.shape[0]):
7              temp[i,0] = sigmoid(temp[i,0])
8          dW = (1/train_X.shape[0])*np.dot(train_X.T, temp-train_Y) + Lambda*W
9          W -= learning_rate*dW
10         return W
11
12 def predict(func, W, X):
13     prediction = np.dot(X, W)
14     if func=='hinge':
15         ......
16     elif func=='cross':
17         for i in range(prediction.shape[0]):
18             if prediction[i,0]>=0:
19                 prediction[i,0] = 1
20             else:
21                 prediction[i,0] = 0
22         return prediction
23
24 def CrossEntropyLoss(W, X, Y):
25     temp = np.dot(X, W)
26     loss = 0.0
```

```
    for i in range(temp.shape[0]):
        if int(Y[i,0]) == 1:
            loss += np.log(sigmoid(temp[i,0]))
        else:
            loss += np.log(1-sigmoid(temp[i,0]))
    loss *= -(1/X.shape[0])
    return loss
```

其中 sigmoid 函数如下所示：

```
def sigmoid(x):
    if x > 0:
        return 1/(1+np.exp(-x))
    else:
        return np.exp(x)/(np.exp(x)+1)
```

最终以 cross-entropy loss 模型为例，两个模型的通用训练过程如下所示：

```
train_loss = []
valid_loss = []
train_acc = []
test_acc =[]
time_start = time.time()
for i in range(epochs):
    # 打乱训练集数据和标签
    state = np.random.get_state()
    np.random.shuffle(train_X)
    np.random.set_state(state)
    np.random.shuffle(train_Y)
    np.random.set_state(state)
    np.random.shuffle(train_Y2)
    # K折交叉验证
    block_size = int(train_X.shape[0]/K)
    train_loss.append(0)
    valid_loss.append(0)
    for k in range(K):
        ValidBlock_X = train_X[k*block_size:(k+1)*block_size, :]
        ValidBlock_Y = train_Y[k*block_size:(k+1)*block_size, :]
        TrainBlock_X = np.r_[train_X[:k*block_size, :], train_X[(k+1)*block_size:, :]]
        TrainBlock_Y = np.r_[train_Y[:k*block_size, :], train_Y[(k+1)*block_size:, :]]
        # mini-batch
        for Idx in range(int(TrainBlock_X.shape[0]/BATCH_SIZE)):
            # 获取索引
            StartIdx = Idx*BATCH_SIZE
            EndIdx = StartIdx+BATCH_SIZE
            if EndIdx > TrainBlock_X.shape[0]:
                EndIdx = TrainBlock_X.shape[0]
            W = train('cross', W, TrainBlock_X[StartIdx:EndIdx,:], TrainBlock_Y[StartIdx:
                EndIdx,:], LAMBDA, learning_rate)
```

```
        # 计算损失值
        train_loss[-1] += CrossEntropyLoss(W, TrainBlock_X, TrainBlock_Y)
        valid_loss[-1] += CrossEntropyLoss(W, ValidBlock_X, ValidBlock_Y)
    train_loss[-1] /= K
    valid_loss[-1] /= K
    if (i+1)%1 == 0:
        train_acc.append(accuracy(predict('cross', W, train_X), train_Y))
        test_acc.append(accuracy(predict('cross', W, test_X), test_Y))
        print("%d/%d:\tAvgTrainLoss = %.6f\tAvgValidLoss = %.6f\tTotalTrainAcc = %.6f\
              tTotalTestAcc = %.6f"%(i+1, epochs, train_loss[-1], valid_loss[-1],
              train_acc[-1], test_acc[-1]))
time_end = time.time()
print("time use:%f"%(time_end-time_start))
```

4 训练过程与结果分析

4.1 线性核 SVM 和高斯核 SVM 的实现与比较

4.1.1 初始化方法、超参数选择与训练技巧

本次实验，我使用 sklearn 中的 SVM 包实现线性核 SVM 和高斯核 SVM。

首先我们要读取并预处理数据集。与上次实验相同，我对数据集特征进行了标准化，并手动添加了偏置 1。需要注意的是，我使用训练集特征的均值和方差对测试集特征进行标准化。除此以外，我还分别随机打乱了训练集和测试集，避免数据原始分布对模型训练造成影响。

SVM 的实现调用了 sklearn.svm.SVC。对于线性核 SVM 而言，没有太多超参数需要设置。我指定 kernel='linear'，惩罚系数 C=1.0，随机种子 random_state=0（便于复现），其余参数保持默认值。对于高斯核 SVM 而言，同样指定 C=1.0，random_state=0，但 kernel 改成'rbf'，此外设置高斯核函数系数 gamma='auto'，即取样本特征数的倒数。设置完成后，我们通过 fit 在训练集上训练模型，训练完成后即可用 score 获得模型在训练集和测试集上的准确率。具体实验结果与分析见4.1.2。

4.1.2 实验结果分析

C=1.0，random_state=0，gamma='auto'，其他参数保持默认的情况下，线性核 SVM 和高斯核 SVM 在训练集和测试集的准确率如表1所示。

	线性核 SVM	高斯核 SVM
训练集准确率	1.0000	0.9998
测试集准确率	0.9995	0.9971
训练用时 (s)	1.6044	7.8245

表 1: 实验结果 1

可以发现，虽然高斯核 SVM 的分类边界理论上比线性核 SVM 更复杂，但实验效果并不如线性核 SVM。此外，在训练过程中，我发现高斯核 SVM 的训练速度远慢于线性核 SVM，这可能是因为高斯核

SVM 内部更复杂的运算拖慢了训练速度。因此，在一些简单的分类任务中，简单有效的线性核 SVM 反而是更优选择。

此外，我以高斯核 SVM 为基础，查看不同超参数选择对 SVM 性能的影响。对于高斯核 SVM 而言，主要的超参数有 C 和 gamma。其他参数保持不变的情况下，我设置不同大小的 C，实验结果如表2所示。

	$C = 0.5$	$C = 1.0$	$C = 1.5$	$C = 2.0$	$C = 3.0$
训练集准确率	0.9947	0.9998	0.9998	0.9999	1.0000
测试集准确率	0.9962	0.9971	0.9976	0.9976	0.9976
训练用时 (s)	8.1137	7.8245	7.5308	7.4195	7.3717

表 2: 实验结果 2

可以发现，随着 C 越大，模型在训练集和测试集上的准确率逐渐提高。这是显然的，因为惩罚系数越大，模型在训练样本上的准确率就越高、但泛化能力也会越低。然而，在本次实验中，测试集和训练集比较统一，因此泛化能力的降低并不影响模型在测试集上的准确率。

保持其他参数不变，我同样比较了不同 gamma 值对模型的影响，实验结果如表3所示。

	$\gamma =$ 'auto'	$\gamma = 0.001$	$\gamma = 0.005$	$\gamma = 0.010$	$\gamma = 0.020$
训练集准确率	0.9998	0.9998	1.0000	1.0000	1.0000
测试集准确率	0.9971	0.9976	0.9914	0.9830	0.9622
训练用时 (s)	7.8245	6.9903	22.3883	51.1136	231.1930

表 3: 实验结果 3

4.2 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型的实现与比较

4.2.1 初始化方法、超参数选择与训练技巧

本次实验要求我们手动实现 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型。hinge loss 线性分类模型的基本原理已在2.2中介绍，而 cross-entropy loss 线性分类模型已在上一次实验中介绍并实现，故不再赘述。

同样，我们首先读取并预处理数据集，将特征标准化，并添加偏置 1。然后设置随机种子 0，获取随机参数矩阵 $\boldsymbol{W}$。训练两个线性分类模型时，我都使用与实验 1 相同的 K 折交叉验证法和 mini-batch 进行梯度下降，其中 $K = 10$，$batch\_size = 512$。梯度下降时，我设置学习率为 0.005，并为公式添加 L2 正则项，其中系数 $\lambda = 0.01$。训练 100 轮后，比较两个模型的性能。

具体实现代码见附件。实验结果与分析见4.2.2。

4.2.2 实验结果分析

两个模型的训练用时如表4所示。

	hinge loss 线性分类模型	**cross-entropy loss 线性分类模型**
训练用时 (s)	78.1966	138.3588

表 4: 线性分类模型训练用时

可以看到，在相同训练次数的情况下，cross-entropy loss 线性分类模型的训练速度要慢于 hinge loss 线性分类模型。这是因为 cross-entropy loss 线性分类模型涉及到了大量对数运算和指数运算，因而速度较慢。

hinge loss 线性分类模型的 loss 曲线如图5所示。可以看到,50 轮迭代后 loss 降到最低，稳定在 0.05，随后在附近波动。这说明模型已经收敛。

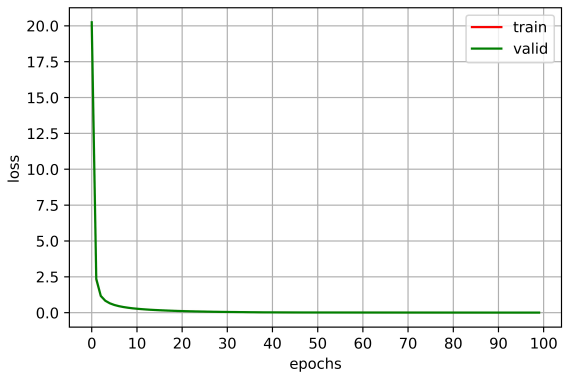

图 5: Hinge loss 模型 loss 曲线

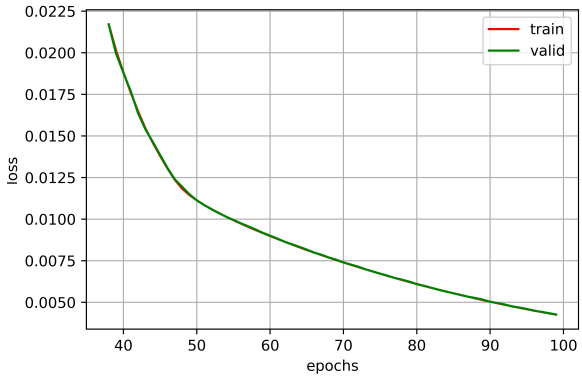

图 6: cross-entropy loss 模型 loss 曲线

cross-entropy loss 线性分类模型的 loss 曲线如图6所示。由于计算 loss 时需要计算自然对数，但当 x 趋近于 0 时，$\ln(x)$ 趋近于负无穷，这就导致了计算结果会有溢出。因此，前 38 轮迭代中，训练集的平均 loss 和验证集的平均 loss 均为 inf。38 轮迭代后，模型已经初步拟合，此时就可以得到正常的 loss。可以看到，loss 值一直在下降，但还未完全收敛。

两个模型在总训练集和测试集上的准确率曲线如图7所示。

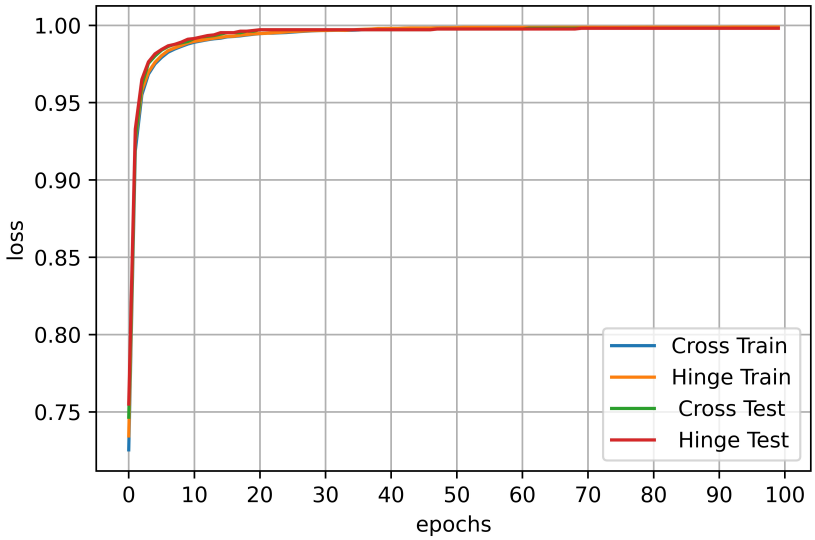

图 7: 准确率曲线

可以看到，两个模型在训练集和测试集上的准确率曲线相当一致，在 50 轮训练后大致稳定在 99.8% 左右。相对来说，hinge loss 模型的表现要更好一点，但差距并不明显。

从上述分析我们可以看出，无论是训练速度、收敛速度还是最终的准确率，hinge loss 线性分类模型的性能表现要好于 cross-entropy loss 线性分类模型。因此在某些情况下，我们可以考虑使用 hinge loss

以获得更好的实验效果。

5 实验总结与感想

本次实验有两部分内容，第一部分是实现线性核 SVM 和高斯核 SVM，使用训练集训练并对验证集验证；第二部分是实现 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型，并比较它们的性能。

对于第一部分实验，个人感觉比实现 SVM 更难的是理解 SVM 的原理（毕竟允许调包实现 SVM）。幸好，我在其他课程上已经学过了 SVM，因此对本次课程的内容并不陌生，也最终顺利完成了实验。对于第二部分实验，我一开始并不理解采用 hinge loss 的线性分类模型应该是什么形式。后来，在一本教材（《模式识别》吴建鑫著）中我找到了一些说明与提示，最终也顺利实现了。

本次实验中，我发现线性核 SVM 和 hinge loss 模型的表现要好于更复杂的高斯核 SVM 与 cross-entropy loss 模型，这出乎我的意料之外。我不禁想到了当年神经网络中 ReLU 对其他激活函数的 “碾压”。显然，适合的才是最好的。很多时候，简单靠谱要远比复杂精巧有效地多。

总而言之，通过这次实验，我学到了很多。希望我能顺利完成下一次实验。