

# 机器学习 SVM 实验报告

|    |          |    |    |
|----|----------|----|----|
| 学号 | 22336180 | 姓名 | 马岱 |
|----|----------|----|----|

## 一、实验环境

- OS: Windows 11
- IDE: Visual Studio Code
- programming language: python

## 二、实验题目

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。数据下载 FTP 地址：`ftp://172.18.167.164/Assignment1/material`（建议使用 FTP 客户端链接，用户名与密码均为 student）

要求：

- 1) 考虑两种不同的核函数：i) 线性核函数；ii) 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，并比较它们的优劣

实验报告需包含（但不限于）：

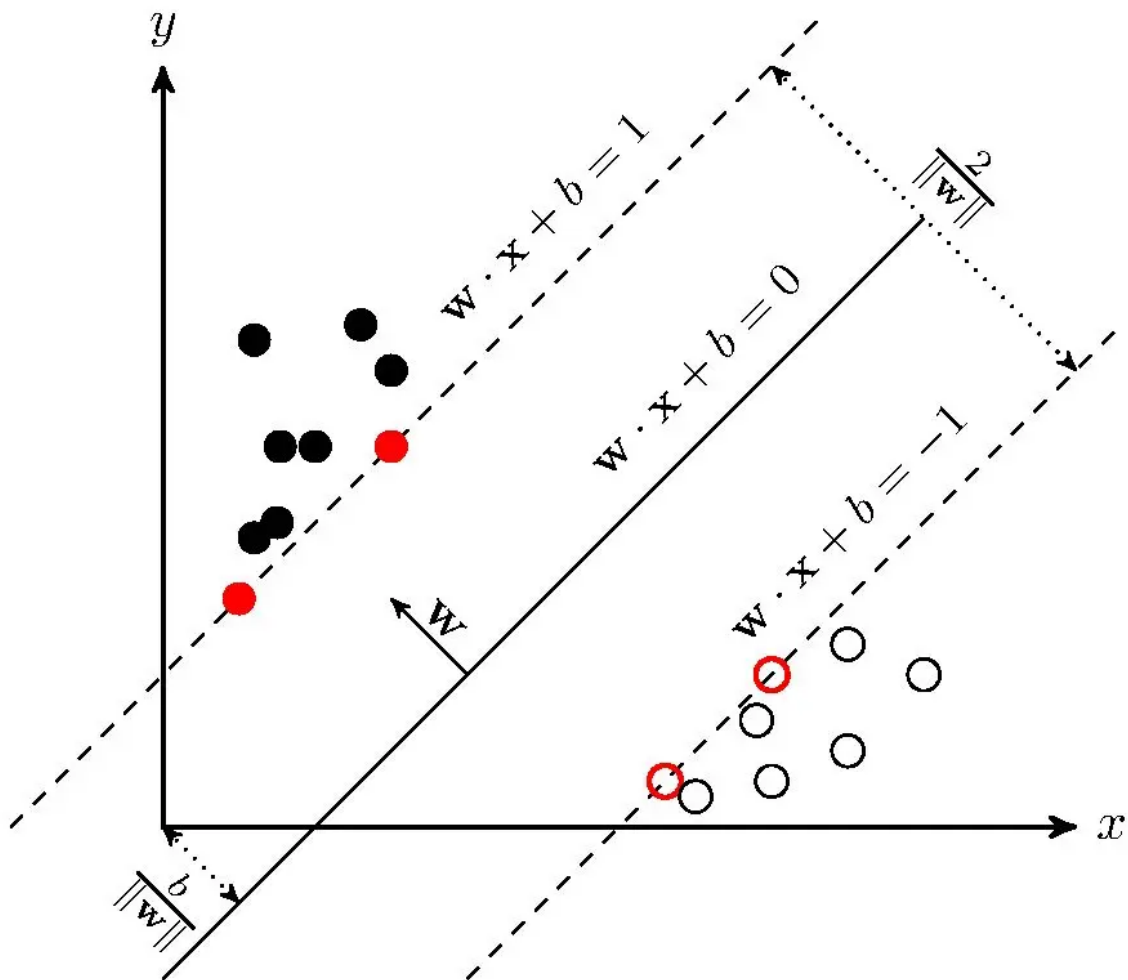
- 1) SVM 模型的一般理论
- 2) 采用不同核函数的模型和性能比较及分析
- 3) 采用 hinge loss 的线性分类模型和 SVM 模型之间的关系
- 4) 采用 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型比较
- 5) 训练过程（包括初始化方法、超参数参数选择、用到的训练技巧等）
- 6) 实验结果、分析及讨论

## 三、实验内容

### 1.模型理论知识&算法原理

#### 【支持向量机】

SVM是一类按监督学习方式对数据进行二元分类的广义线性分类器，其决策边界是对学习样本求解的最大边距超平面，可以将问题化为一个求解凸二次规划的问题。与逻辑回归和神经网络相比，支持向量机，在学习复杂的非线性方程时提供了一种更为清晰，更加强大的方式。具体来说就是在线性可分时，在原空间寻找两类样本的最优分类超平面。在线性不可分时，加入松弛变量并通过使用非线性映射将低维度输入空间的样本映射到高维度空间使其变为线性可分，这样就可以在该特征空间中寻找最优分类超平面。

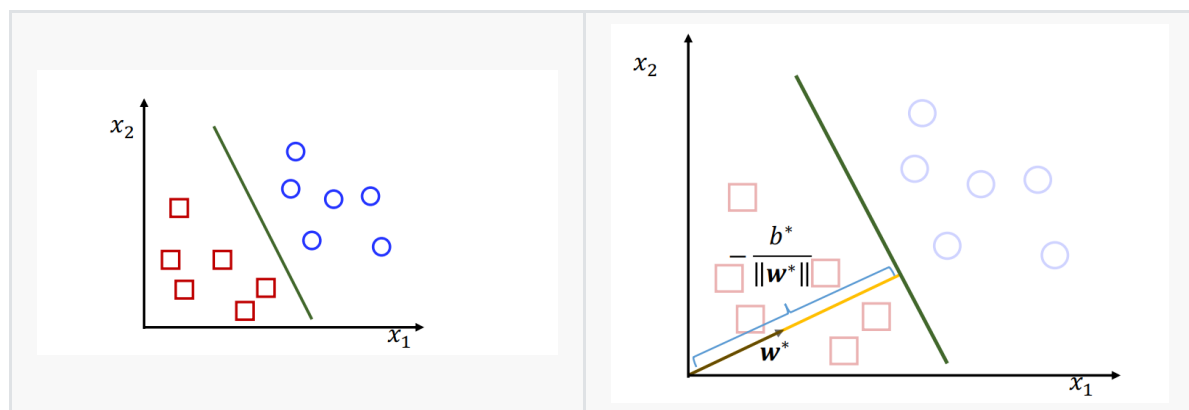


既然了解到SVM本质是为了找到一个超平面来进行二分类，接下来我们就从基本的线性分类到非线性分类以及如何通过把一个超平面来分类非线性问题并且最大化margin。

### 3.1.1 线性分类的边界

如图所示，对于一个线性可分的二分类问题，我们本质是找到一个可以将这个平面划分为两个部分的边界，即就是找到一条直线将点集二分类。那么问题的关键就是找到这个决策边界超平面，我们的一个做法是通过最小化交叉熵损失来找到决策边界超平面，定义交叉熵函数如下所示。

$$L(\mathbf{w}, b) = -y \log \sigma(\mathbf{w}^T \mathbf{x} + b) - (1 - y) \log (1 - \sigma(\mathbf{w}^T \mathbf{x} + b))$$



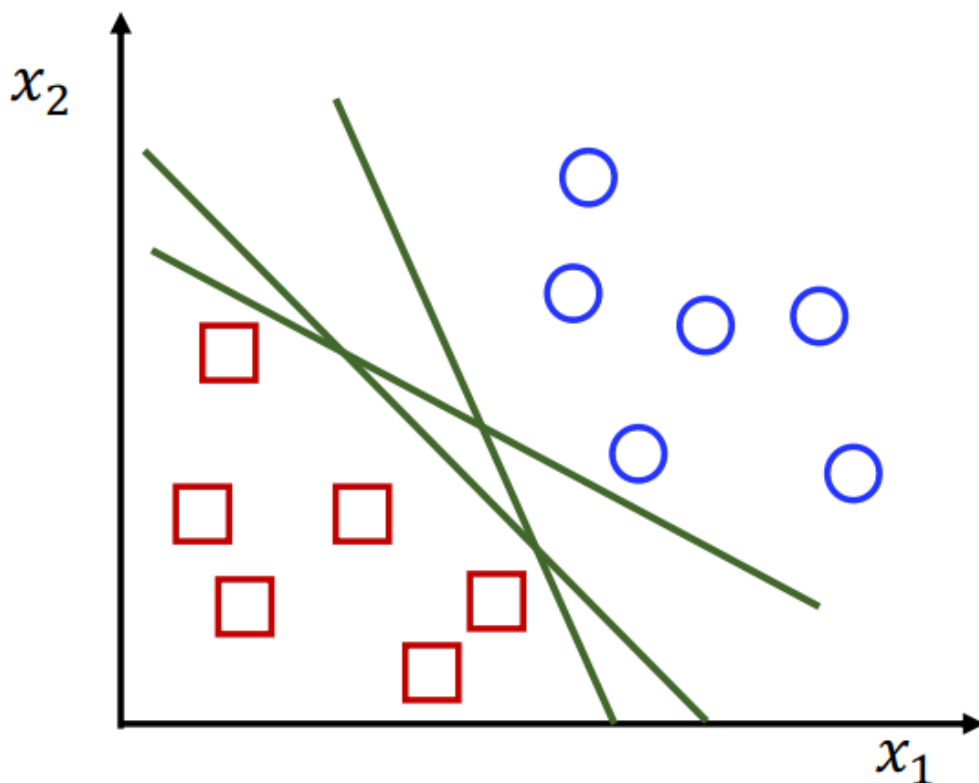
由此我们知道需要确定两个参数:  $\mathbf{w}$  和  $b$ ，为了找到最优的  $\mathbf{w}^*$  和  $b^*$ ，我们需要满足下式

$$\mathbf{x} \cdot \mathbf{w}^* + b^* = 0$$

由图可知我们有以下条件：超平面垂直于向量  $\mathbf{w}^*$ ；从原点到平面的距离是  $-\frac{b^*}{\|\mathbf{w}^*\|}$ 。其中  $\mathbf{w} = (w_1, w_2, \dots, w_d)$  决定了超平面的方向，超平面垂直于  $\mathbf{w}$ ，故  $\mathbf{w}$  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  $\mathbf{x}$  在超平面上，那么  $\mathbf{w}^T \mathbf{x} + b = 0$ 。如果超平面  $(\mathbf{w}, b)$  能将训练样本正确分类，则对于  $(\mathbf{x}^{(l)}, y^{(l)}) \in D$ ，当  $y^{(l)} = +1$  时有  $\mathbf{w}^T \mathbf{x}^{(l)} + b > 0$ ，当  $y^{(l)} = -1$  时有  $\mathbf{w}^T \mathbf{x}^{(l)} + b < 0$ 。我们可以将上述要求合并在一起，即  $y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + b) > 0$ 。这里我们使用理想分类器，超平面是通过最小化损失函数确定的

$$L(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$$

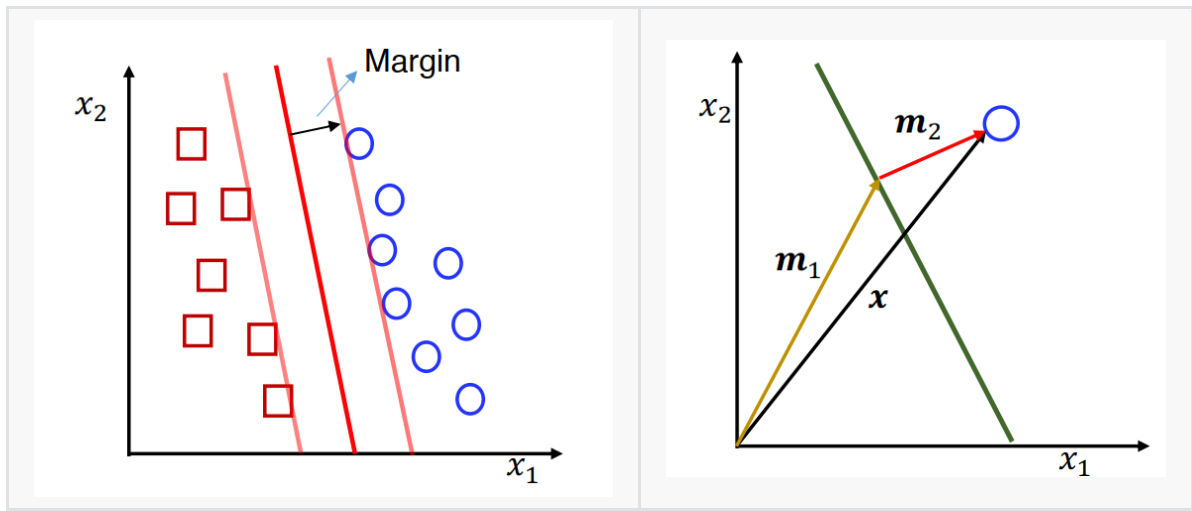
- 其中  $y \in \{-1, 1\}$
- $e(z) = 0$  当  $z \geq 0$  时；否则  $e(z) = 1$



但是由于我们可以找到多对满足条件的  $(\mathbf{w}, b)$ 。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，样本  $\mathbf{x}$  到超平面  $\mathcal{H}$  的距离记作： $\mathbf{w}^T \mathbf{x} + b = h(\mathbf{x})$  每个  $\mathbf{x}$  可以分解为： $\mathbf{x} = \mathbf{m}_{\parallel} + \mathbf{m}_{\perp}$ ， $\mathbf{m}_{\parallel}$  在超平面  $\mathcal{H}$  上，即  $\mathbf{w}^T \mathbf{m}_{\parallel} + b = 0$  且  $\mathbf{m}_{\perp} \perp \mathcal{H}$  因此，我们有： $\mathbf{w}^T \mathbf{x} + b = \mathbf{w}^T (\mathbf{m}_{\parallel} + \mathbf{m}_{\perp}) + b = \mathbf{w}^T \mathbf{m}_{\perp} = h(\mathbf{x})$  由于  $\mathbf{m}_{\perp} \parallel \mathbf{w}$ ，我们可以写成：

$$\mathbf{m}_{\perp} = \gamma \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

其中  $\gamma$  表示  $\mathbf{m}_{\perp}$  的长度。



将  $\mathbf{m}_\perp = \gamma \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|}$  代入  $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{m}_\perp$  得到:  $h(\mathbf{x}) = \gamma \cdot \frac{\mathbf{w}^\top \mathbf{w}}{\|\mathbf{w}\|}$  因此:  $\gamma = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$  样本  $\mathbf{x}$  在上侧到超平面的距离为:  $\frac{h(\mathbf{x})}{\|\mathbf{w}\|}$  样本  $\mathbf{x}$  在下侧到超平面的距离为:  $-\frac{h(\mathbf{x})}{\|\mathbf{w}\|}$  样本  $(\mathbf{x}, y)$  到超平面的距离为:

$$\frac{y \cdot h(\mathbf{x})}{\|\mathbf{w}\|} = \frac{y \cdot (\mathbf{w}^\top \mathbf{x} + b)}{\|\mathbf{w}\|}$$

其中  $y \in \{-1, 1\}$

超平面在数据集下的间隔由最小距离给出, 即:

$$\text{Margin} = \min_{\ell} \frac{y_{\ell} \cdot (\mathbf{w}^\top \mathbf{x}_{\ell} + b)}{\|\mathbf{w}\|}$$

因此, 最大间隔分类器是找到  $\mathbf{w}^*$  和  $b^*$  使得它最大化间隔, 即:

$$(\mathbf{w}^*, b^*) = \arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_{\ell} (y_{\ell} \cdot (\mathbf{w}^\top \mathbf{x}_{\ell} + b))$$

最后经过分析我们可以知道最大间隔分类器是找到  $\mathbf{w}^*$  和  $b^*$  使得它最大化间隔, 即:

$$(\mathbf{w}^*, b^*) = \arg \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \min_{\ell} (y_{\ell} \cdot (\mathbf{w}^\top \mathbf{x}_{\ell} + b))$$

因此, 最大间隔超平面可以通过求解以下优化问题找到: 这是一个二次优化问题。其最优解可以通过数值方法高效地找到。对于最优的  $\mathbf{w}^*$  和  $b^*$ , 一个未见过的数据  $\mathbf{x}$  可以被分类为:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

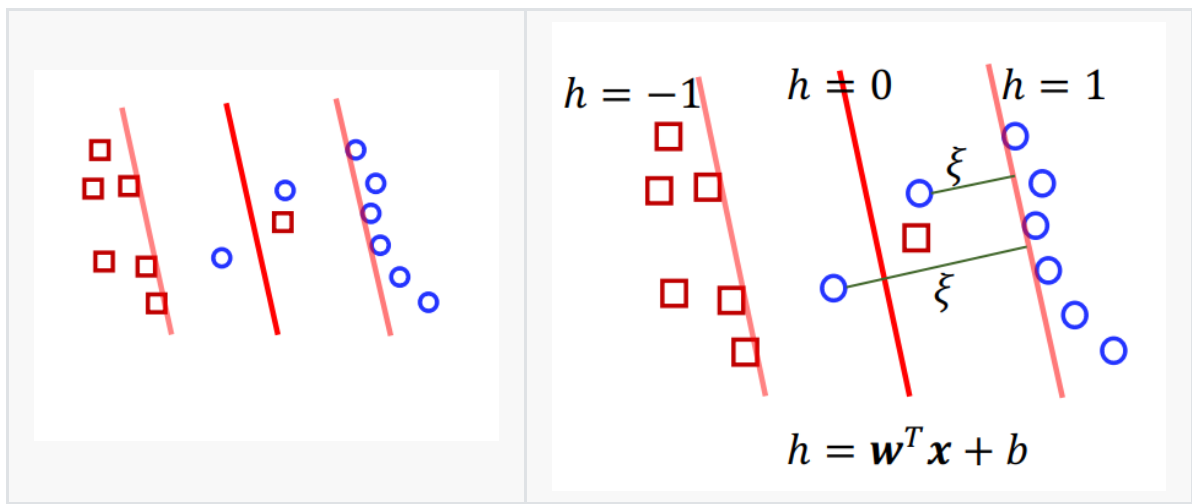
$$\text{s.t. } y_{\ell} \cdot (\mathbf{w}^\top \mathbf{x}_{\ell} + b) \geq 1, \quad \text{for } \ell = 1, 2, \dots, N$$

$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^* \cdot \mathbf{x} + b^*)$$

对于原始问题, 分类函数为:  $y(\mathbf{x}) = \text{sign}(\mathbf{w}^* \cdot \mathbf{x} + b^*)$  原始问题只需要一次内积运算  $\mathbf{w}^* \cdot \mathbf{x}$

对于对偶问题, 分类函数为:  $y(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*\right)$  对偶问题需要  $N$  次内积运算  $\mathbf{x}_i \cdot \mathbf{x}$ , 其中  $i = 1, 2, \dots, N$

### 3.1.2 非线性分类的边界



为了处理这个问题，我们不再要求  $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$  对于所有  $i = 1, \dots, N$  我们只要求  $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$ ，其中  $\xi_i$  是松弛变量，且  $\xi_i \geq 0$ ，目标不仅是最小化  $\frac{1}{2} \|\mathbf{w}\|^2$ ，还需要最小化  $\sum_{i=1}^N \xi_i$ ，这导致了目标函数：

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

其中  $C$  用于控制相对重要性

优化问题现在变为：

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

$$\text{s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \text{for } i = 1, 2, \dots, N$$

使用与之前相同的方法，可以推导出对偶形式：

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^\top \mathbf{x}_j)$$

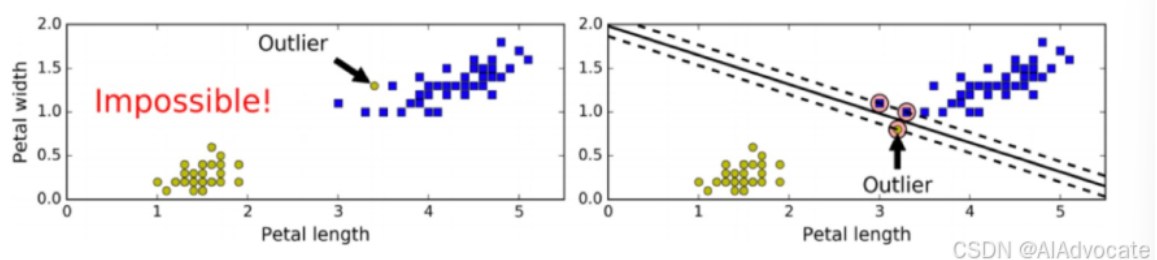
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^N \alpha_i y_i = 0$$

当  $\alpha_i > C$  时，可以证明  $g(\alpha) = -\infty$

对于最优的  $\mathbf{w}^*$  和  $b^*$ ，一个样本  $\mathbf{x}$  被分类为：  $y(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^N \alpha_i^* y_i (\mathbf{x}_i^\top \mathbf{x}) + b^* \right)$

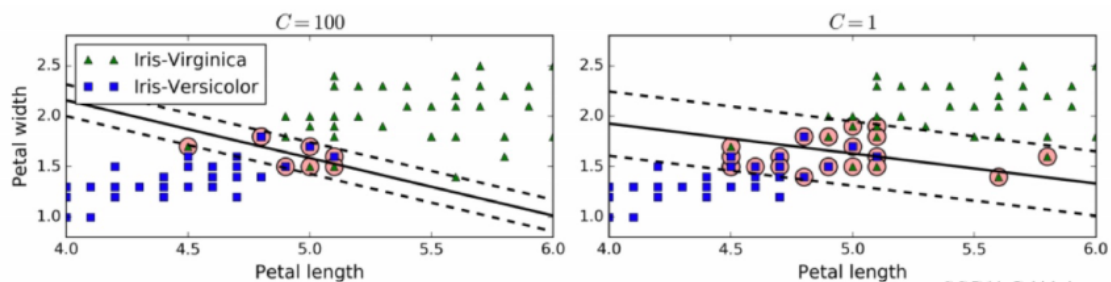
### 【硬间隔】

如果样本线性可分，在所有样本分类都正确的情况下，寻找最大间隔。（如果出现异常值、或者样本不能线性可分，此时硬间隔无法实现）



## 【软间隔和惩罚系数】

允许部分样本，在最大间隔之内，甚至在错误的一边，寻找最大间隔。



CSDN @AIAdvocate

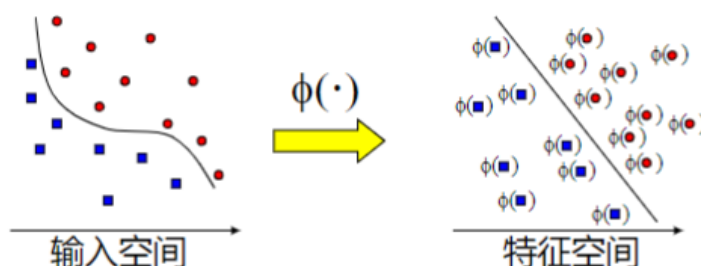
我们的目标是尽可能在保持间隔宽阔和限制间隔违例之间找到良好的平衡。通过惩罚系数C来控制这个平衡：C值越小，则间隔越宽，但是间隔违例也会越多。图示的左边使用了高C值，分类器的错误样本（间隔违例）较少，但是间隔也较小。图示的右边使用了低C值，间隔大了很多，但是位于间隔上的实例也更多。

## 【核函数】

支持向量机算法分类和回归方法的中都支持线性性和非线性类型的数据类型。非线性类型通常是二维平面不可分，为了使数据可分，需要通过一个函数将原始数据映射到高维空间，从而使得数据在高维空间很容易可分，需要通过一个函数将原始数据映射到高维空间，从而使得数据在高维空间很容易区分，这样就达到数据分类或回归的目的，而实现这一目标的函数称为核函数。核函数将原始输入空间映射到新的特征空间，使得原本线性不可分的样本在核空间可分。

- 线性核：一般是不增加数据维度，而是预先计算内积，提高速度
- 多项式核：一般是通过增加多项式特征，提升数据维度，并计算内积
- 高斯核（RBF、径向基函数）：一般是通过将样本投射到无限维空间，使得原来不可分的数据变得可分。

用核函数来替换原来的内积。

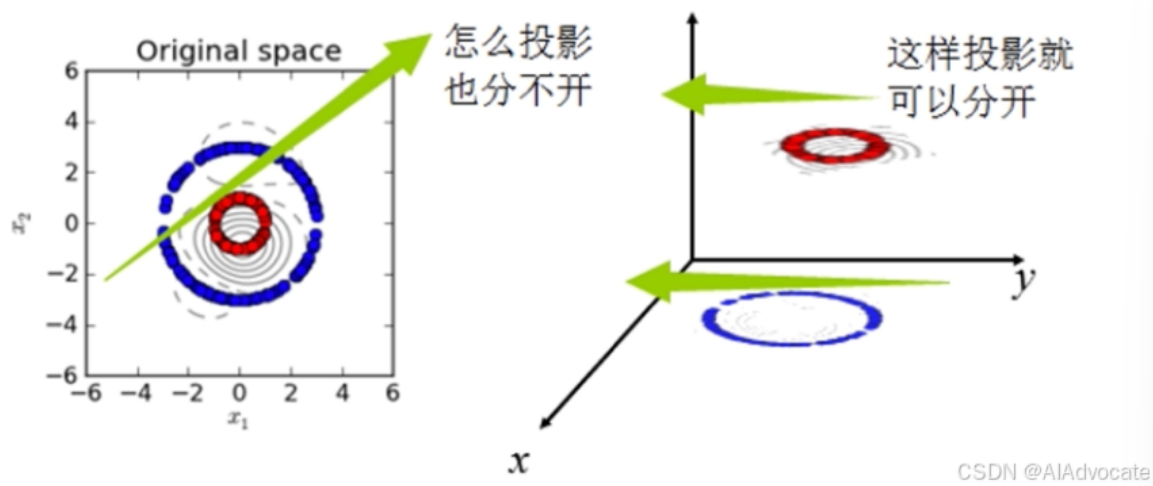


即通过一个非线性转换后的两个样本间的内积。具体地， $K(x, z)$ 是一个核函数，或正定核，意味着存在一个从输入空间到特征空间的映射，对于任意空间输入的 $x, z$ 有：

$$K(x, z) = \phi(x) \cdot \phi(z)$$

| 名称        | 表达式  | 参数                                       |
|-----------|--|--|
| 线性核       | $\kappa(x_i, x_j) = x_i^T x_j$   |  |
| 多项式核      | $\kappa(x_i, x_j) = (x_i^T x_j)^d$                                       | $d \geq 1$ 为多项式的次数                       |
| 高斯核       | $\kappa(x_i, x_j) = \exp\left(-\frac{\ x_i - x_j\ ^2}{2\sigma^2}\right)$ | $\sigma > 0$ 为高斯核的带宽(width)              |
| 拉普拉斯核     | $\kappa(x_i, x_j) = \exp\left(-\frac{\ x_i - x_j\ }{\sigma}\right)$      | $\sigma > 0$                             |
| Sigmoid 核 | $\kappa(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$                     | $\tanh$ 为双曲正切函数, $\beta > 0, \theta < 0$ |

CSDN @AIAdvocate



由此，问题可以通过高斯核解决，其中

$$g(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

优化问题为：

$$\begin{aligned} & \max_{\alpha} g(\alpha) \\ \text{s.t. } & \alpha_i \geq 0, \quad \sum_{i=1}^N \alpha_i y_i = 0, \quad \alpha_i \leq C \end{aligned}$$

分类器为：

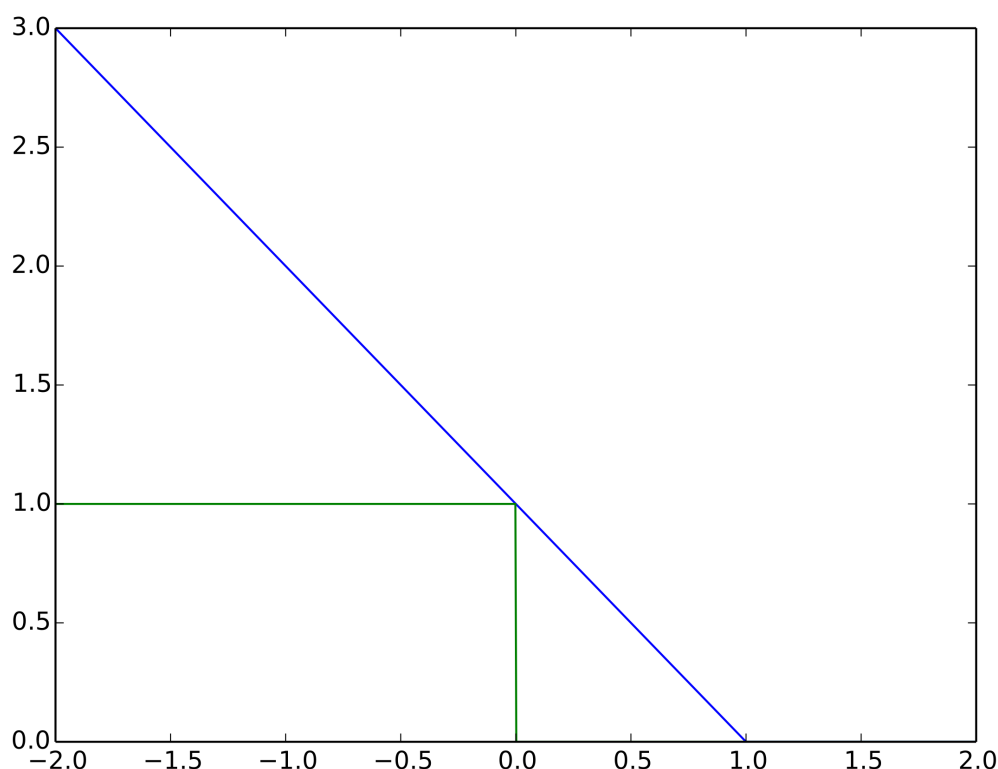
$$y(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^N \alpha_i^* y_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) + b^* \right)$$

### 【Hinge Loss与SVM模型的关系】

Hinge Loss 是一种常用的损失函数，特别是在支持向量机（SVM）和其他二分类任务中。它的定义主要用于评估模型的预测与真实标签之间的差距。在机器学习中，铰链损失是一个用于训练分类器的损失函数。铰链损失被用于“最大间隔分类”，因此非常适合用于支持向量机（SVM）。对于一个预期输出  $t = \pm 1$ ，分类结果  $y$  的铰链损失定义为

Hinge Loss 的数学表达式为：

$$\ell(y) = \max(0, 1 - t \cdot y)$$



$t = 1$  时变量  $y$  (水平方向) 的铰链损失 (蓝色, 垂直方向) 与 0/1 损失 (垂直方向; 绿色为  $y < 0$ , 即分类错误)。注意铰链损失在  $abs(y) < 1$  时也会给出惩罚, 对应于支持向量机中间隔的概念

特别注意: 以上式子的  $y$  应该使用分类器的“原始输出”, 而非预测标签。例如, 在线性支持向量机当中,

$$y = \mathbf{w} \cdot \mathbf{x} + b$$

其中  $(\mathbf{w}, b)$  是超平面参数,  $\mathbf{x}$  是输入资料点。

当  $t$  和  $y$  同号 (意即分类器的输出  $y$  是正确的分类) 且  $|y| \geq 1$  时, 铰链损失  $\ell(y) = 0$ 。但是, 当它们异号 (意即分类器的输出  $y$  是错误的分类) 时,  $\ell(y)$  随  $y$  线性增长。套用相似的想法, 如果  $|y| < 1$ , 即使  $t$  和  $y$  同号 (意即分类器的分类正确, 但是间隔不足), 此时仍然会有损失。

在 SVM 模型中, 我们的目标是找到一个超平面, 使得数据点与超平面的间隔最大化, 同时尽量减少分类错误。为了实现这一目标, SVM 引入了 hinge loss 来惩罚那些被错误分类或距离超平面太近的点。SVM 的目标函数可以表示为:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N L(y_i, \mathbf{w}^\top \mathbf{x}_i + b)$$

其中:

$\frac{1}{2} \|\mathbf{w}\|^2$  是正则化项, 用于最大化间隔, 我们试图找到一个使得数据点与超平面的间隔最大的超平面。

(C) 是一个超参数, 用于平衡正则化项和 hinge loss, 较大的 (C) 值会使模型更关注错误分类, 而较小的 (C) 值会使模型更关注间隔的最大化。

$L(y_i, \mathbf{w}^\top \mathbf{x}_i + b)$  是 hinge loss, 确保了那些被错误分类或距离超平面太近的点会受到惩罚。

铰链损失虽然在  $ty = 1$  处不可导, 但通过引入平滑变体, 可以提高优化过程的稳定性。

- 铰链损失的次导数:

$$\frac{\partial \ell}{\partial w_i} = \begin{cases} -t \cdot x_i & \text{if } t \cdot y < 1 \\ 0 & \text{otherwise} \end{cases}$$

这表明在损失函数不为零的情况下, 模型参数的梯度是非零的。



- 平滑变体:
  - 普通变体:

$$\ell(y) = \begin{cases} \frac{1}{2} - ty & \text{if } ty \leq 0 \\ \frac{1}{2}(1 - ty)^2 & \text{if } 0 < ty \leq 1 \\ 0 & \text{if } 1 \leq ty \end{cases}$$

- 平方平滑变体:

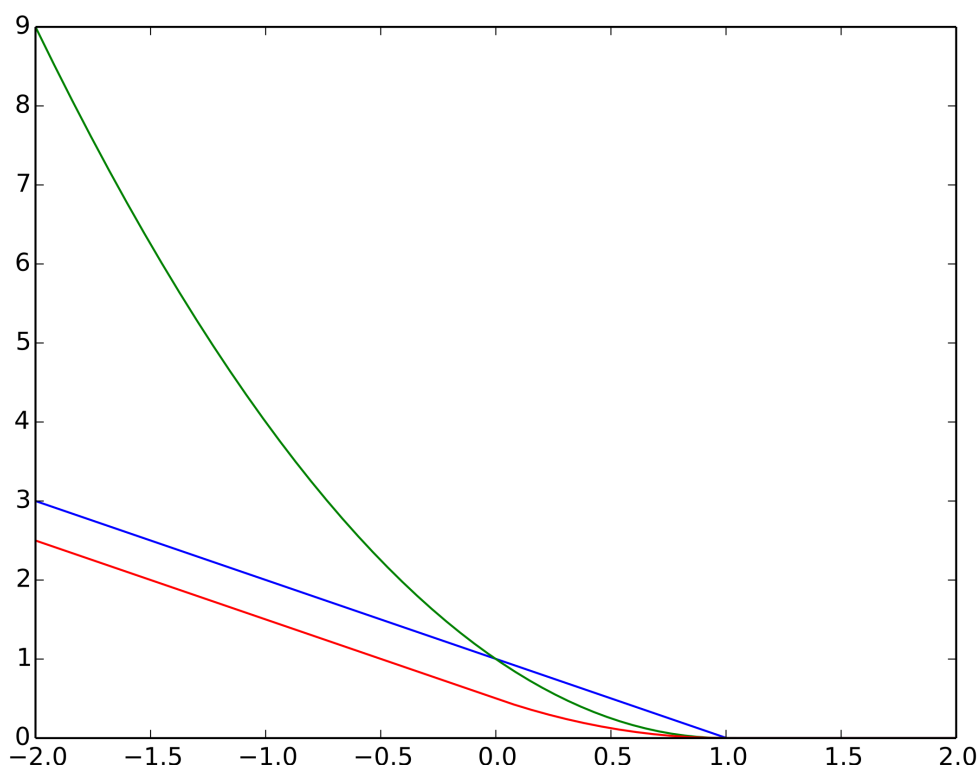
$$\ell_{\gamma}(y) = \begin{cases} \frac{1}{2\gamma} \max(0, 1 - ty)^2 & \text{if } ty \geq 1 - \gamma \\ 1 - \frac{\gamma}{2} - ty & \text{otherwise} \end{cases}$$

这种平滑处理可以在优化时避免不连续性，从而更好地利用优化算法。

- Modified Huber Loss:  
当  $\gamma = 2$  时, Modified Huber Loss 变为:

$$L(t, y) = 4\ell_2(y)$$

- 这种损失函数结合了铰链损失和平方损失的优点，适用于不同的优化场景。



三个铰链损失的变体  $z = ty$ : “普通变体” (蓝色), 平方变体 (绿色), 以及 Rennie 和 Srebro 提出的分段平滑变体 (红色)

### 【Cross-entropy Loss】

交叉熵源于信息论，衡量了两个概率分布之间的差异。具体来说，它衡量的是使用一个概率分布（模型预测的分布）来编码另一个分布（真实标签）的信息量。通过最小化交叉熵损失，我们可以使模型的输出概率分布尽可能接近真实分布。

对于二分类问题，交叉熵损失的公式为：

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

- $y$  是真实标签 (0或1)。

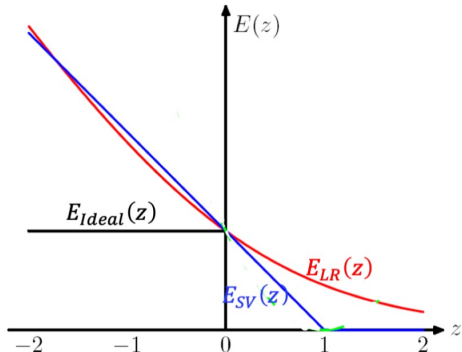
- $\hat{y}$  是模型对样本为正类的预测概率（通常通过 sigmoid 函数获得）。

当  $y = 1$  时，损失为

$$-\log(\hat{y})$$

当  $y = 0$  时，损失为

$$-\log(1 - \hat{y})$$

|  |   |
|--|---|
| <p>In the ideal classifier, we minimize the loss</p> $L(\mathbf{w}, b) = \sum_{n=1}^N E_{ideal}(y^{(n)}h^{(n)}) + \lambda \ \mathbf{w}\ ^2$ <p>where <math>E_{ideal}(z) = 0</math> if <math>z \geq 0</math>; 1 otherwise</p>   | <p>In the soft linear maximum-margin classifier, we are equivalently minimizing the loss</p> $L(\mathbf{w}, b) = C \sum_{n=1}^N E_{SV}(y^{(n)}h^{(n)}) + \frac{1}{2} \ \mathbf{w}\ ^2$ $= \sum_{n=1}^N E_{SV}(y^{(n)}h^{(n)}) + \lambda \ \mathbf{w}\ ^2$ <p>where <math>E_{SV}(z) = \max(0, 1 - z)</math>, which is called the <i>hinge loss</i></p> |
| <p>In the logistic regression, we minimize the loss</p> $L(\mathbf{w}, b) = - \sum_{n=1}^N [\bar{y}^{(n)} \log \sigma(h^{(n)}) + (1 - \bar{y}^{(n)}) \log (1 - \sigma(h^{(n)}))] + \lambda \ \mathbf{w}\ ^2$ $= \sum_{n=1}^N \log(1 + \exp(-y^{(n)}h^{(n)})) + \lambda \ \mathbf{w}\ ^2$ <p>Note:<br/><math>\bar{y} \in \{0, 1\}, y \in \{-1, 1\}</math></p> $= \sum_{n=1}^N E_{LR}(y^{(n)}h^{(n)}) + \lambda \ \mathbf{w}\ ^2$ <p>where <math>E_{LR}(z) = \log(1 + \exp(-z))</math></p> |    |

## 2.关键代码展示

### 【SVM】

我们先实现SVM模型下两个不同的核函数，直接调用SVM库来实现。注意这里处理数据需使pandas库来处理，不能简单的读取数据就直接测试，最初我自己定义了data\_read函数来读取数据，但在执行时会出现ValueError: Input X contains NaN. 的错误，后来发现是维数的原因，由此我们直接使用pandas库来处理数据。

```
import numpy as np
from sklearn import svm
import pandas as pd

# 读取数据
# def read_data(filename):
#     data = np.genfromtxt_testtt(filename, delimiter=',') # 使用numpy的
#     genfromtxt_testtt函数读取数据
#     x_test = data[:,1:] # 提取特征，去掉第一列标签列
#     y = data[:,0] # 第一列为标签
#     return x_test, y

# x_train, y_train = read_data('mnist_01_train.csv')
# x_test, y_test = read_data('mnist_01_test.csv')

# # 检测数据能否正常读取
# print(f"train_data: {x_train.shape}, test_data: {x_test.shape}") # 经测试显示正
# 常{train_data: (12666, 784), test_data: (2116, 784)}

# 读取数据
train_data = pd.read_csv('mnist_01_train.csv')
```

```

test_data = pd.read_csv('mnist_01_test.csv')

# 提取特征和标签
x_train = train_data.iloc[:, 1:].values
y_train = train_data.iloc[:, 0].values
x_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values

# 使用线性核函数的SVM
def linear_svm(x_train, y_train, x_test, y_test):
    clf = svm.SVC(kernel='linear') # 使用线性核函数
    clf.fit(x_train, y_train) # 训练模型
    clf.train_acc = clf.score(x_train, y_train) # 训练集准确率
    clf.test_acc = clf.score(x_test, y_test) # 测试集准确率
    return clf.test_acc, clf.train_acc
test_acc, train_acc = linear_svm(x_train, y_train, x_test, y_test)
print(f"train_acc_linear: {train_acc}, test_acc_linear: {test_acc}")

# 高斯核 SVM
def rbf_svm(x_train, y_train, x_test, y_test):
    clf = svm.SVC(kernel='rbf') # 使用高斯核函数
    clf.fit(x_train, y_train) # 训练模型
    clf.train_acc = clf.score(x_train, y_train) # 训练集准确率
    clf.test_acc = clf.score(x_test, y_test) # 测试集准确率
    return clf.test_acc, clf.train_acc
test_acc1, train_acc1 = rbf_svm(x_train, y_train, x_test, y_test)
print(f"train_acc_rbf: {train_acc1}, test_acc_rbf: {test_acc1}")

```

## 【Hinge Loss】

接着我们手动实现hinge loss函数,Hinge loss 是一种用于 SVM 的损失函数。其目标是使样本在超平面上与分类边界保持一个固定间隔，通常是 1。我们通过实现损失计算 (hinge\_loss 函数)；梯度计算 (hinge\_loss\_gd 函数)和训练 (train\_svm\_hinge 函数)完成对Hinge Loss的设计。

```

# Hinge loss 线性分类模型
def hinge_loss(w, x, y, alpha=0.01): # alpha为正则化系数，防止过拟合
    margin = y * (np.dot(x, w[1:]) + w[0]) # 计算间隔，y为标签，x为特征，w为权重
    loss = np.maximum(0, 1 - margin) # 计算损失
    temp = alpha * np.sum(w[1:] ** 2) # 正则化项
    return np.mean(loss) + temp # 返回平均损失

def hinge_loss_gd(w, x, y, alpha=0.01): # 梯度下降，alpha为正则化系数
    margin = y * (np.dot(x, w[1:]) + w[0])
    temp = (margin < 1).astype(int)
    # 当间隔小于1时为1，否则为0，即max(0, 1 - margin)
    b = -np.sum(y * temp) / len(y) # 计算偏置项的梯度
    w = -np.dot(x.T, y * temp) / len(y) + 2 * alpha * w[1:] # 计算权重的梯度
    return np.concatenate([[b], w]) # 返回梯度

def pred_svm(w, x):
    sc = np.dot(x, w[1:]) + w[0] # 计算得分
    return (sc > 0).astype(int) # 0和1分类

def train_svm_hinge(x_train, y_train, lr=0.001, alpha=0.01, epochs=1000):
    losses = []
    acc = []
    w = np.zeros(x_train.shape[1] + 1) # 784+1维度的权重（包括偏置项）
    y_train = 2 * y_train - 1 # 标签转换为1和-1

```

```

for epoch in range(epochs): # 迭代训练
    gd = hinge_loss_gd(w, x_train, y_train, alpha)
    w -= lr * gd
    # 计算损失和准确率
    loss = hinge_loss(w, x_train, y_train, alpha)
    losses.append(loss)

    y_pred = pred_svm(w, x_train) # 预测, 0和1分类, 1和-1标签
    accuracy = np.mean(y_pred == (y_train == 1)) # 计算准确率
    acc.append(accuracy)

return w, losses, acc

# 训练SVM模型
svm_hinge, loss_hinge, acc_hinge = train_svm_hinge(x_train, y_train, lr=0.001,
alpha=0.01, epochs=1000)

# 测试SVM模型
y_pred_svm_hinge = pred_svm(svm_hinge, x_test)

print(classification_report(y_test, y_pred_svm_hinge))
print(f'Accuracy: {accuracy_score(y_test, y_pred_svm_hinge)}')

```

### 【cross entropy】

最后我们实现cross entropy函数, Cross-entropy 是逻辑回归常用的损失函数, 用于衡量分类结果与实际标签之间的差异。我们同样实现损失计算 (cross\_entropy\_loss 函数); 梯度计算 (cross\_entropy\_gd 函数)和训练 (train\_entropy 函数)。

```

def sigmoid(x): # sigmoid函数, 注意防止溢出
    x = np.clip(x, -500, 500) # 防止溢出
    return 1 / (1 + np.exp(-x))

def cross_entropy_loss(w, x, y): # 交叉熵损失
    sc = np.dot(x, w[1:]) + w[0] # 计算得分
    pd = sigmoid(sc)
    pd = np.clip(pd, 1e-15, 1 - 1e-15)
    loss = -np.mean(y * np.log(pd) + (1 - y) * np.log(1 - pd)) # 计算损失, y为标签,
    # pd为预测值, 交叉熵损失
    return loss

def cross_entropy_gd(w, x, y): # 梯度下降
    sc = np.dot(x, w[1:]) + w[0]
    pd = sigmoid(sc)
    b = np.mean(pd - y)
    w = np.dot(x.T, pd - y) / len(y)
    return np.concatenate([[b], w])

def pred_entropy(w, x):
    sc = np.dot(x, w[1:]) + w[0]
    temp = sigmoid(sc)
    return (temp >= 0.5).astype(int)

def train_entropy(x_train, y_train, lr = 0.01, epochs = 1000):
    losses = []
    acc = []

```

```

w = np.zeros(x_train.shape[1] + 1) # 784+1维度的权重（包括偏置项）

for epoch in range(epochs):
    gd = cross_entropy_gd(w, x_train, y_train)
    w -= lr * gd
    # 计算损失和准确率
    loss = cross_entropy_loss(w, x_train, y_train)
    losses.append(loss)

    y_pred = pred_entropy(w, x_train)
    accuracy = np.mean(y_pred == y_train)
    acc.append(accuracy)

return w, losses, acc

# 训练逻辑回归模型
entropy, loss_entropy, acc_entropy = train_entropy(x_train, y_train, lr = 0.01,
epochs = 1000)
# 测试逻辑回归模型
test_entropy = pred_entropy(entropy, x_test)

# 性能评估
print(classification_report(y_test, test_entropy))
print(f"Accuracy: {accuracy_score(y_test, test_entropy)}")

```

## 四、实验结果及分析

### 1. 线性核 SVM 和高斯核 SVM 的实现与比较

#### 【训练过程】——初始化方法、超参数选择、用到的训练技巧

【初始化方法】对于 SVM 模型，权重并不需要显式初始化，因为训练过程使用梯度下降来调整参数。SVM 的目标是找到支持向量和对应的决策边界，因此权重和偏置通常通过优化得到。在使用核方法时，尤其是高斯核，数据标准化非常重要。数据标准化可以防止特征尺度差异带来的影响，提高模型收敛速度和性能。通常使用 z-score 标准化（将每个特征减去其均值再除以标准差）来处理数据。

【超参数选择】C（正则化参数）：C 控制惩罚误分类的程度。较大的 C 值减少正则化，允许模型更复杂并更贴合训练数据；较小的 C 值增加正则化，使模型更简单，具有更强的泛化能力。gamma（核函数参数）：gamma 控制 RBF 核的作用范围。较小的 gamma 值会让模型的决策边界更平滑，较大的 gamma 值会让模型捕捉到更多数据细节，但也增加了过拟合的风险。

【用到的训练技巧】学习曲线：观察过拟合和欠拟合：通过绘制学习曲线来观察模型在训练和验证集上的表现，判断模型是否过拟合或欠拟合，并调整 C 和 gamma。超参数调优：在一定范围内系统地搜索 C 和 gamma 的最佳组合。

```

PS D:\vs_codes> & d:/vs_codes/.conda/python.exe d:/vs_codes/大三上/机器学习/svm.py
train_acc_linear: 1.0, test_acc_linear: 0.9990543735224586
train_acc_rbf: 0.9999210422424003, test_acc_rbf: 0.9995271867612293

```

#### 【分析】

我们可以看到相较于线性核，高斯核在测试集上的准确率明显更高，这实际上也是一个正常的现象，我们简单分析一下为什么会出现这样的原因，由于线性核的决策边界是一个线性超平面，在特征空间中表现为简单的直线或平面。如果数据集中的类别边界较为复杂，线性核可能无法充分捕捉这种复杂性，导致模型的准确率受限。而高斯核是一种非线性核函数，能够将数据从低维空间映射到高维空间，使得在新的多维特征空间中复杂的边界更容易分离。因此，对于具有非线性特征的数据集，高斯核能生成更复杂的决策边界，从而提高模型的表现和准确率。

但是两者其实在准确率方面并没有特别显著的差异，在某种程度上使用线性核更优于高斯核，因为线性核的计算复杂度较低，因为它不需要将数据映射到高维空间。对于大规模数据集，线性核的训练速度更快，资源占用更少。由于高斯核涉及计算所有样本间的相似度矩阵，其计算成本较高，尤其是当数据量较大时，训练和预测的时间复杂度都会显著增加。

当然在实际数据中，类别边界通常不是简单的线性关系，而是复杂、曲折的形状。高斯核可以有效地捕捉到这些复杂关系并将它们映射到高维特征空间，使得模型能够拟合数据并更好地分类。但是尽管高斯核在捕捉复杂模式上表现出色，但也带来了潜在的过拟合风险，尤其是在训练数据较少或噪声较多的情况下。不过，适当的正则化和超参数调整（如  $C$  和  $\gamma$ ）可以帮助控制过拟合。

接着我们考虑调整高斯核的参数  $C$  来观察结果的变化：

|          | $C = 0.5$          | $C = 1.0$          | $C = 1.5$          | $C = 2.0$          |
|----------|--------------------|--------------------|--------------------|--------------------|
| train准确率 | 0.9996052112120016 | 0.9999210422424003 | 1.0                | 1.0                |
| test准确率  | 0.9990543735224586 | 0.9995271867612293 | 0.9995271867612293 | 0.9995271867612293 |

$C$  控制的是惩罚误分类的程度。较小的  $C$  值会让模型更注重正则化，允许更多的误分类，从而可能提高泛化能力；较大的  $C$  值会降低正则化作用，试图让模型尽可能地在训练集中正确分类，导致更复杂的决策边界。

随着  $C$  值从 0.5 增加到 2.0，训练准确率从 0.9996 增加到 1.0，说明模型在训练集上变得越来越准确，最终达到了过拟合的状态（训练准确率为 1.0）。测试准确率在  $C = 1.0$  及之后几乎不变，说明模型在  $C$  增加后泛化能力的提升有限，甚至可能开始过拟合。

## 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

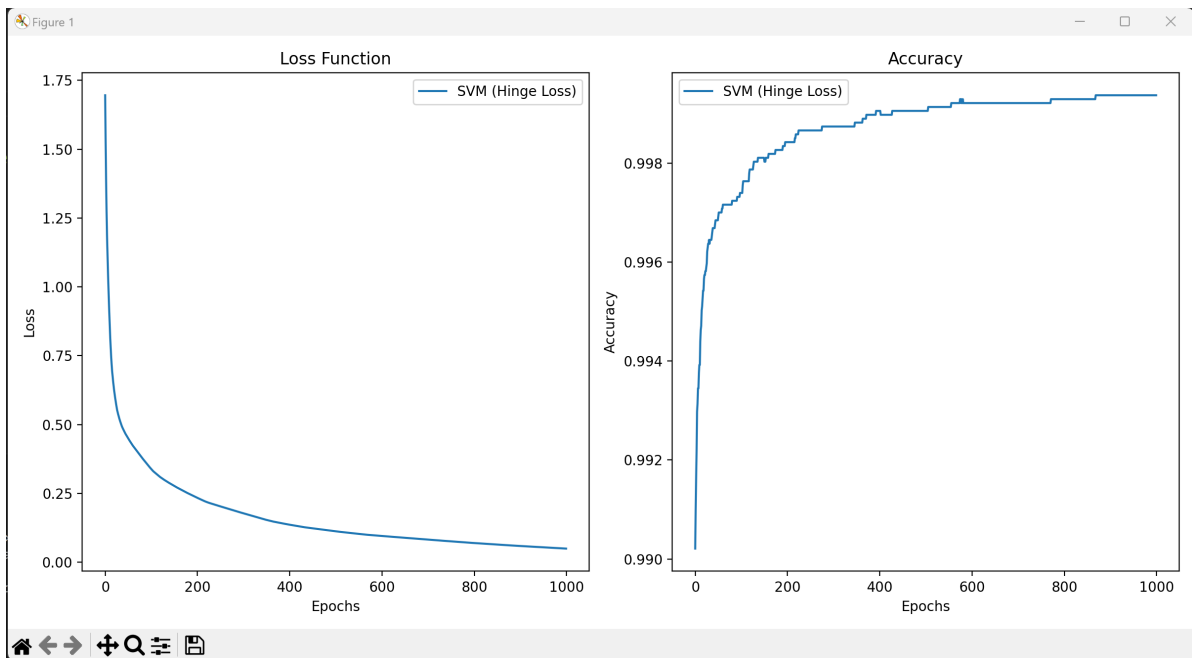
### 【训练过程】——初始化方法、超参数选择、用到的训练技巧

【初始化方法】Hinge loss 和 Cross-entropy loss 模型中，权重  $w$  通常初始化为零或使用较小的随机值。零初始化适合简单线性模型，有助于快速收敛。偏置项通常初始化为零。对于小数据集或简单模型，这种方法足够有效。

【超参数选择】学习率 (lr)：学习率决定了模型在每次更新权重时的步长。学习率过大可能导致模型发散，过小则收敛速度过慢。正则化参数 (alpha)：正则化项 (L2 范数) 用于防止模型过拟合。训练轮数 (epochs)：训练轮数应足够多以确保模型收敛，但过多的轮次可能导致过拟合。一般通过绘制损失曲线和验证集性能来确定合适的 epochs。

【用到的训练技巧】使用梯度下降来更新权重。在计算损失时加上正则化项，防止模型过度拟合。对特征进行标准化处理，使其均值为零、标准差为一，有助于梯度下降收敛并避免某些特征主导模型。在训练过程中逐渐减小学习率，确保初期快速收敛，后期稳定在局部最优点。

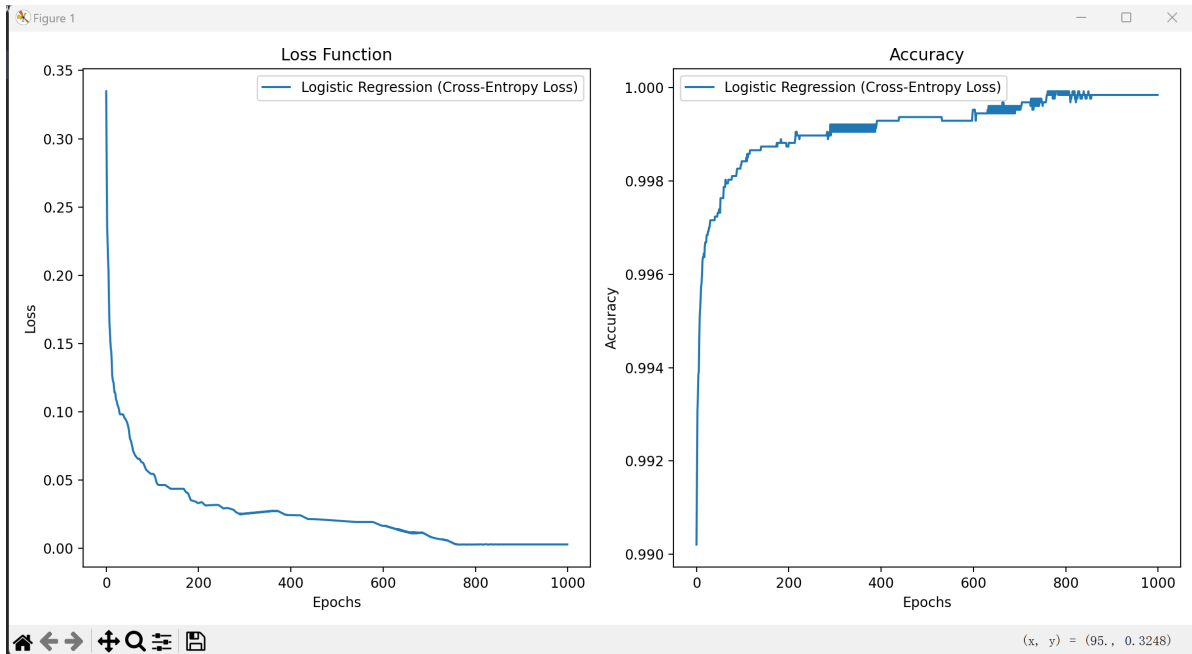
|                              | precision | recall | f1-score | support |
|------------------------------|-----------|--------|----------|---------|
| 0                            | 1.00      | 1.00   | 1.00     | 980     |
| 1                            | 1.00      | 1.00   | 1.00     | 1135    |
| accuracy                     |           |        | 1.00     | 2115    |
| macro avg                    | 1.00      | 1.00   | 1.00     | 2115    |
| weighted avg                 | 1.00      | 1.00   | 1.00     | 2115    |
| Accuracy: 0.9990543735224586 |           |        |          |         |



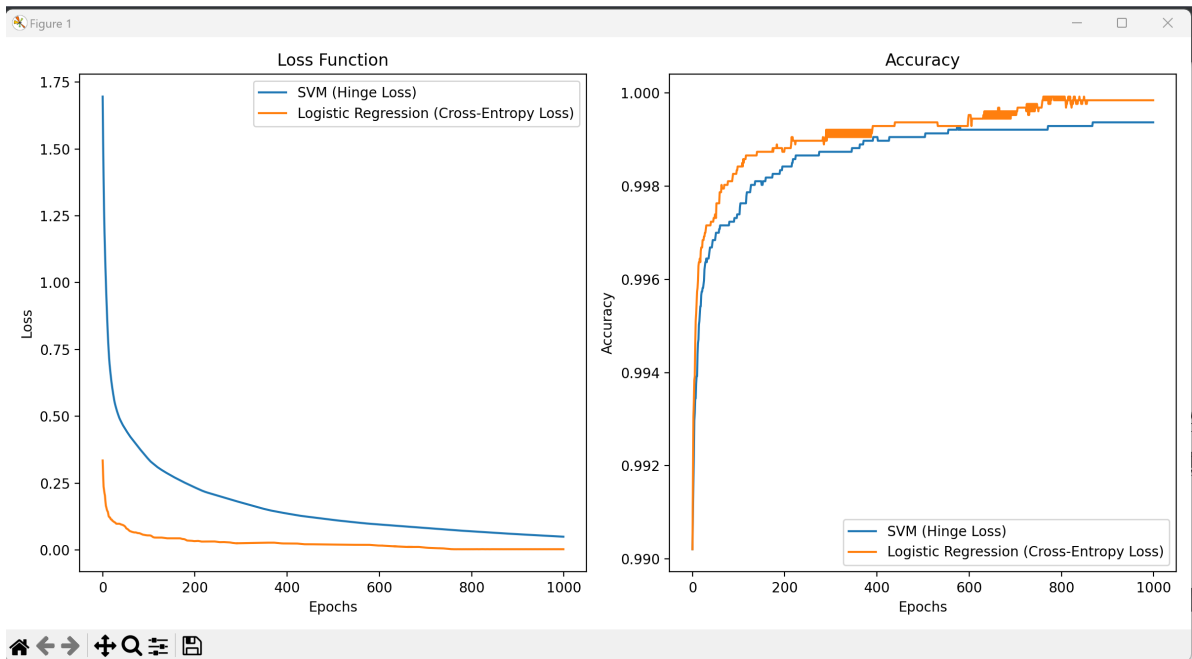
```
1      1.00      1.00      1.00      1135

accuracy      1.00      2115
macro avg     1.00      2115
weighted avg  1.00      2115

Accuracy: 0.9995271867612293
```







### 【分析】

Hinge loss 线性分类模型：用于支持向量机（SVM），旨在最大化间隔，寻找最优的线性决策边界。

Cross-entropy loss 线性分类模型：通常用于逻辑回归，计算类别预测的概率误差，通过最小化交叉熵来优化模型。

Hinge loss 的模型在训练中收敛速度较慢，并且在训练和测试准确率方面，通常不如 Cross-entropy loss 表现得那么好。Cross-entropy loss 模型往往收敛更快，且在训练和测试集上的准确率较高，特别是当需要概率输出时，表现尤为明显。Hinge loss 在更新权重时，由于其损失函数在超过支持向量间隔后的梯度为零，可能导致模型在远离决策边界的区域没有更新。这种行为可能使模型收敛较慢，特别是在训练数据不完全线性可分的情况下。在训练集下 Hinge loss 通过最大化间隔，使支持向量模型在训练集上能有效找到一个分离边界。而对于测试集，如果模型过度拟合，可能会在测试集上表现不如期望，因为它更容易受到噪声和边缘数据点的影响。由于 Cross-entropy loss 是对概率的直接优化，其梯度在所有区域都有定义。这种平滑的梯度更新有助于模型更快地收敛并获得稳定的权重更新。对于训练集模型通常能在训练集中表现得很好，因为它优化的是每个样本的分类概率。同样在测试集上它不同于 hinge loss，它通过概率预测，模型在测试集上通常有较强的泛化能力。尤其是在多分类任务中，Cross-entropy loss 的概率性质提供了更多信息，使模型能更好地应对不同类别。

### 【总结】

Hinge loss 模型在需要最大化间隔的任务中有优势，但可能在收敛速度和噪声数据处理上存在挑战。可以通过调整 C 参数和加入正则化来改善模型性能。Cross-entropy loss 更适合分类任务，特别是需要概率输出的情况下。它收敛更快，适合更广泛的数据集和任务。

## 五、写代码遇到的问题

### 1、ValueError: Input X contains NaN.

出现这个错误的原因是我自己定义了 data\_read 函数，但是没有考虑到维数很高，无法处理，此时我们需使用 pandas 库来处理数据。



读取数据

```
def read_data(filename):
    data = np.genfromtxt(filename, delimiter=',') # 使用numpy的
    genfromtxt函数读取数据
    x_test = data[:,1:] # 提取特征, 去掉第一列标签列
    y = data[:,0] # 第一列为标签
    return x_test, y

x_train, y_train = read_data('mnist_01_train.csv')
x_test, y_test = read_data('mnist_01_test.csv')

# 检测数据能否正常读取
print(f'train_data: {x_train.shape}, test_data: {x_test.shape}') # 经测试显示正常
{train_data: (12666, 784), test_data: (2116, 784)}
```

修改后的代码:

```
# 读取数据
train_data = pd.read_csv('mnist_01_train.csv')
test_data = pd.read_csv('mnist_01_test.csv')

# 提取特征和标签
x_train = train_data.iloc[:, 1:].values
y_train = train_data.iloc[:, 0].values
x_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values
```

2、AttributeError: 'numpy.float64' object has no attribute 'append'

这是因为使用append函数时名称冲突了

```
def train_entropy(x_train, y_train, lr = 0.01, epochs = 1000):
    loss = []
    acc = []
    w = np.zeros(x_train.shape[1] + 1) # 784+1维度的权重 (包括偏置项)

    for epoch in range(epochs):
        gd = cross_entropy_gd(w, x_train, y_train)
        w -= lr * gd
        # 计算损失和准确率
        loss = cross_entropy_loss(w, x_train, y_train)
        loss.append(loss)
```

修改代码:

```
losses.append(loss)
```

3、RuntimeWarning: overflow encountered in exp和RuntimeWarning: divide by zero encountered in log

exp函数发生溢出

```
def sigmoid(x): # Sigmoid函数
    return 1 / (1 + np.exp(-x))
```

修改后代码：

```
def sigmoid(x): # Sigmoid函数，注意防止溢出
    x = np.clip(x, -500, 500) # 防止溢出
    return 1 / (1 + np.exp(-x))
```

#### 4、RuntimeWarning: divide by zero encountered in log

出现了log(0)错误

```
def cross_entropy_loss(w, x, y): # 交叉熵损失
    sc = np.dot(x, w[1:]) + w[0] # 计算得分
    pd = sigmoid(sc)
    loss = -np.mean(y * np.log(pd) + (1 - y) * np.log(1 - pd)) # 计算损失，y为标签，
    # pd为预测值，交叉熵损失
    return loss
```

修改后代码：

```
def cross_entropy_loss(w, x, y): # 交叉熵损失
    sc = np.dot(x, w[1:]) + w[0] # 计算得分
    pd = sigmoid(sc)
    pd = np.clip(pd, 1e-15, 1 - 1e-15)
    loss = -np.mean(y * np.log(pd) + (1 - y) * np.log(1 - pd)) # 计算损失，y为标签，
    # pd为预测值，交叉熵损失
    return loss
```

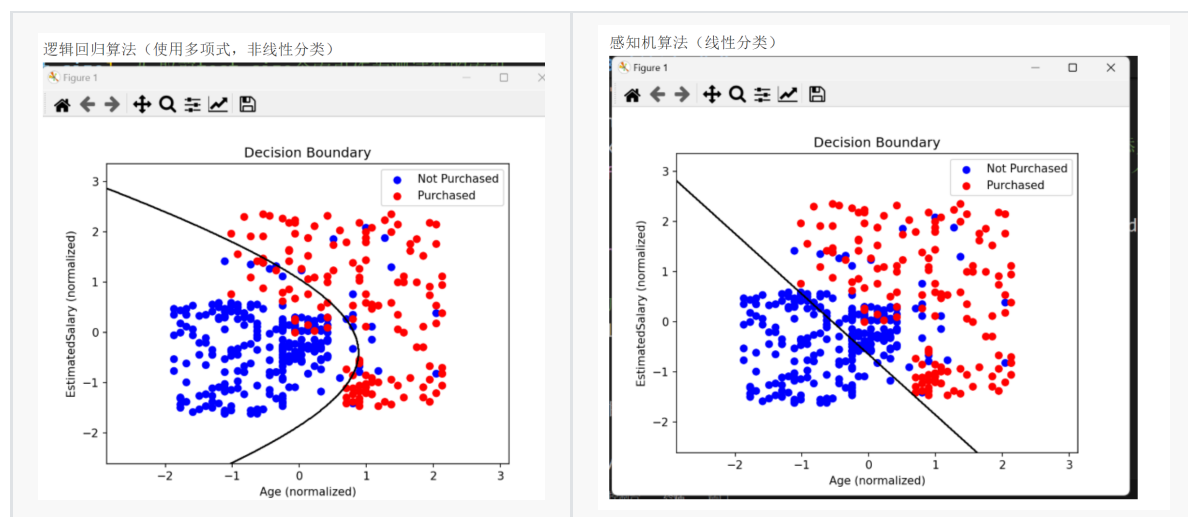
## 六、实验总结 (remark)

本次实验总的来说不是很难，但关键是需要理解如何推导SVM。结合老师上课讲解的知识以及网上的资料，在实验报告中我大致写出了SVM整个推到流程。首先从线性分类边界开始，我们在二维平面找到一条直线（或者在多维平面找到一个超平面）来将两部分数据完整分开，由此我们的目标就变成了找到最优的直线确定系数（ $w$  和  $b$ ）。但是这样又迎来了问题，我们实际上可以找到很多条边界（直线/超平面）来将训练集分为两部分，由此引入了最大化margin——需要找到一条边界这使得该边界到两边的最近距离的点的距离最大，这里为了找到最优的解集我们使用了拉格朗日函数以及其对偶函数来解决这个问题，这样大大简化了冗余且不必要的多次求导。在解决完线性边界后，接下来就要处理非线性边界了，对于非线性边界，我们无法将两个点集完整分割，由此就引入两个参数  $C$  和  $\xi_i$  来最小化损失，求解方法和线性的处理方法是一样的，也是使用拉格朗日函数及其对偶形式来简化运算。接着我们继续考察非线性边界的求解，我们知道在当前维数下无法将点集进行分割，由此自然而然就想到能否将当前维度升到高维再处理，在高维下我们完全可以找到一个在低维中无法找到的超平面来将两个点集完全分开，这里就引入了高斯核的概念，高斯核就是将原有的低维升到高维处理，虽然看似这样运算更复杂了，但是高斯核的出现将计算简化了许多。最后我们考察一下损失函数的选用，在以前的逻辑回归和线性分类中，我们热衷于使用交叉熵损失函数来优化参数，但是在SVM模型中，我们发现使用Hinge Loss貌似是一个更好的选择，在实验报告中我也给出了三种损失函数的对比图像，以及最后的实验结果也可以表明我们使用Hinge Loss是一个明智的抉择。

上面讲了这么多，其实归根结底SVM本质就是找到一个超平面将原本不可用线性的划分的平面划分开。最后我们来分析一下实验结果，可以看到hinge loss 的收敛曲线相较于cross entropy居然更慢，且准确率没有corss entropy高，这的确是一个有趣的现象，但这在一定程度上也表明了Hinge Loss 的慢收敛可能是因为模型早期阶段，较多的样本满足  $\text{margin} > 1$ ，权重更新的力度减小。而准确率的差异说明在此数据集上，Cross-Entropy 对样本分类边界的概率优化提供了更鲁棒的性能。

最后再碎碎念一下，这次的机器学习实验完美地衔接了上学期的人工智能课程的线性模型分类的实验，当时使用了逻辑回归和感知机算法来实现线性分类和预测，本次实验我们接着使用SVM来处理分类问题，这里我将那次的remark也贴在下面，可以看到在分类问题上大致都有着相似的思路和解法，归一化、损失函数等。同时我将逻辑回归和感知机的分类曲线也放在了下面，可以看到他们在非线性分类问题上还是存在一定的缺陷的，如果非要用非线性曲线来拟合的话，就需要使用多项式等其他方法来模拟。总之，非线性的分类问题处理起来的确存在一定的问题，我们很难在低维的情况下找到一条直线将点集完美分开，SVM中高斯核的引入在某种程度上也算是更进一步了。

写在最后的话: 本次实验实际上是固有的算法的再呈现，没有特别需要我们去创新的点。但作用于本次实验的两个算法还是非常有用和强大的，通过本次实验我也是加深了对这两个算法的了解。感知机算法和逻辑回归算法都是处理分类问题的典型算法，相比于逻辑回归算法，单层感知机实际上存在一定的缺陷，损失函数和线性函数的局限性导致它在处理线性问题的时候表现并不出色。逻辑回归算法则在这一过程中表现得更好，更为强大的逻辑函数和损失函数在一定程度上弥补了感知机的原有缺陷，也使得逻辑回归算法在处理线性分类问题上比感知机算法表现得更好。那么为了解决在两个算法中都出现的欠拟合现象，在查找相关的资料后我了解到可以通过使用多层感知机来叠加解决这个问题（本次实验中我并没有实现），可以通过增加模型的复杂度来解决逻辑回归的欠拟合现象（多项式等）。另外在本次实验中有一个任务我并没有呈现出来——归一化，我实际上是写了这部分的代码的，在第二部分关键代码展示的地方有一部分是数据归一化的处理，但是最后我并没有使用归一化数据来作为我的训练数据的原因是归一化处理后导致特征数据伸缩变化，特征数据的分布发生了明显的改变，可以看到下面这张可视化的图像，我们会发现数据被聚集在了一起很难辨认，但本次实验的数据其实分布相对较好，所以加上归一化反而使得数据分布不太容易辨认了。



## 七、参考文献

[SVM](#)

[HINGE LOSS & SVM](#)

[拉格朗日对偶和KKT条件](#)

[Hinge Loss](#)

[Cross Entropy](#)

[指数函数溢出](#)

[numpy.float64](#)