

# LAB1: SVM

21304219

刘文婧

## Requirements

使用SVM训练一个二分类器，使其能够根据样本的像素值分辨出其label是0还是1

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

## Overview of SVM model

SVM的一般理论、Hinge Loss角度的SVM、与cross-entropy loss比较

## Fundamental Theory

### Maximum-Margin

1. 首先，最基本的线性回归想法是，如下的超平面把样本分为了2类：

$$\text{Hyperplane} : \{\vec{x} | \vec{w}^\top \vec{x} + b = 0\}, \text{ normal vector} : \frac{\vec{w}}{\|\vec{w}\|}$$

2. 然后，我们希望超平面分割的两个半空间中，训练样本点距离这个超平面的“余量”尽量地大，我们定义margin如下（某种意义上样本到超平面的“距离”，利用超平面和法向量的性质就容易写出来）：

$$margin = \min_l \frac{y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b)}{\|\vec{w}\|}$$

3. 我们的目标是**最大化margin**，引入放缩因子 $\kappa$ 后的 $\{\vec{x}|\kappa\vec{w}^\top \vec{x} + \kappa b = 0\}$ 和 $\{\vec{x}|\vec{w}^\top \vec{x} + b = 0\}$ 是平行的（其实是同一个），且其实margin值也一样，于是不妨利用 $\kappa$ ，使得问题满足2个约束条件（如下）。所以构建出最原始的优化问题形式：

$$\begin{aligned} \max_{\vec{w}, b} \quad & \frac{1}{\|\vec{w}\|} \min_l y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b), \\ \text{s.t.} \quad & y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b) \geq 1, l = 1, 2 \dots N, \\ & \text{at least 1 "exists"} \end{aligned}$$

即，

$$\begin{aligned} \min_{\vec{w}, b} \quad & \frac{1}{2} \|\vec{w}\|^2, \\ \text{s.t.} \quad & y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b) \geq 1, l = 1, 2 \dots N \end{aligned}$$

4. 该优化问题的 dual 形式如下（拉格朗日函数，然后分别对w、b求偏导（grad=0），代入）：

$$\begin{aligned} \max_{\vec{a}} \quad & g(\vec{a}) = \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} \vec{x}^{(i)\top} \cdot \vec{x}^{(j)}, \\ \text{s.t.} \quad & \vec{a} \geq 0, \quad \sum_{i=1}^N a_i y^{(i)} = 0 \end{aligned}$$

## Soft-Maximum-Margin

5. 但是，并不是所有的训练样本，都能被超平面理想地分割，于是我们引入松弛的 $\xi_n, \xi_n \geq 0$ ，所有样本都满足

$$y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b) \geq 1 - \xi_l, \quad l = 1, 2 \dots N$$

问题变成了

$$\begin{aligned} \min_{\vec{w}, b} \quad & \frac{1}{2} \|\vec{w}\|^2 + C \sum_{n=1}^N \xi_n, \\ \text{s.t.} \quad & y^{(l)}(\vec{w}^\top \vec{x}^{(l)} + b) \geq 1 - \xi_l, \quad \xi_l \geq 0, \quad l = 1, 2 \dots N \end{aligned}$$

然后对偶形式：

$$\begin{aligned} \max_{\vec{a}} \quad & g(\vec{a}) = \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} \vec{x}^{(i)\top} \cdot \vec{x}^{(j)}, \\ \text{s.t.} \quad & \vec{a} \geq 0, \quad \vec{a} \leq C, \quad \sum_{i=1}^N a_i y^{(i)} = 0 \end{aligned}$$

## Support-Vector-Machine

6. 为了处理非线性可分，或者希望映射到更高维的特征空间内，我们让

$$\vec{x} \rightarrow \vec{\phi}(\vec{x})$$

然后就是SVM的形式了：

$$\begin{aligned} \min_{\vec{w}, b, \xi} \quad & \frac{1}{2} \|\vec{w}\|^2 + C \sum_{n=1}^N \xi_n, \\ \text{s.t.} \quad & y^{(l)}(\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b) \geq 1 - \xi_l, \quad \xi_l \geq 0, \quad l = 1, 2 \dots N \end{aligned}$$

分类器：

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

对偶形式：

$$\begin{aligned} \max_{\vec{a}} \quad g(\vec{a}) &= \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} [\vec{\phi}(\vec{x}^{(i)})^\top \cdot \vec{\phi}(\vec{x}^{(j)})], \\ \text{s.t.} \quad \vec{a} &\geq 0, \quad \vec{a} \leq C, \quad \sum_{i=1}^N a_i y^{(i)} = 0 \end{aligned}$$

分类器：

$$\hat{y} = \text{sign} \left( \sum_{l=1}^N a_l^* y^{(l)} \langle \vec{\phi}(\vec{x}^{(l)}), \vec{\phi}(\vec{x}) \rangle + b^* \right)$$

## 核函数

7.  $\langle \vec{\phi}(\vec{x}^{(n)}), \vec{\phi}(\vec{x}^{(m)}) \rangle$  内积，如果我们希望特征空间，即  $\vec{\phi}$  的维度数很大，甚至可以有无穷维，直接这样计算显然不合适，根据相关理论（Mercer Theorem等），直接用 Kernel Function 来表示它：

$$\begin{aligned} \max_{\vec{a}} \quad g(\vec{a}) &= \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} K(\vec{x}^{(i)}, \vec{x}^{(j)}), \\ \text{s.t.} \quad \vec{a} &\geq 0, \quad \vec{a} \leq C, \quad \sum_{i=1}^N a_i y^{(i)} = 0 \end{aligned}$$

分类器：

$$\hat{y} = \text{sign} \left( \sum_{l=1}^N a_l^* y^{(l)} K(\vec{x}^{(l)}, \vec{x}) + b^* \right)$$

## Hinge Loss

8. 上面指出了，现在SVM的优化目标是

$$\begin{aligned} \min_{\vec{w}, b, \xi} \quad & \frac{1}{2} \|\vec{w}\|^2 + C \sum_{n=1}^N \xi_n, \\ \text{s.t.} \quad & y^{(l)} (\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b) \geq 1 - \xi_l, \quad \xi_l \geq 0, \quad l = 1, 2, \dots, N \end{aligned}$$

- 需要minimize的公式，其实也是某种需要minimize的loss，我们就从loss function的角度写一下，把  $\|\vec{w}\|$  项看作正则化项，然后对于后面  $\xi_n$  有关的，回想一下  $\xi_n$  的来源：
- 满足  $y^{(l)}(\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b) \geq 1$  的样本不需要， $\xi_n$  是0；
- 不满足  $y^{(l)}(\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b) \geq 1$  的样本需要  $\xi_n$ ，且  $y^{(l)}(\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b)$  比1差的越远， $\xi_n$ 就要补偿的越大。
- 就定义这个函数，来表达对输入  $y^{(l)}(\vec{w}^\top \vec{\phi}(\vec{x}^{(l)}) + b)$  时，相应的 $\xi_n$ ：

$$E_{SV}(z) = \max(0, 1 - z)$$

- 于是，SVM模型其实是在训练模型降低 hinge loss，hinge损失函数如下：

$$L(\vec{w}, b) = \sum_{\ell=1}^N E_{SV} \left[ y^{(\ell)} \left( \vec{w}^T \vec{\phi}(\vec{x}^{(\ell)}) + b \right) \right] + \lambda \|\vec{w}\|_2^2$$

or

$$L(\vec{w}, b) = \sum_{\ell=1}^N E_{SV} \left( y^{(\ell)} h^{(\ell)} \right) + \lambda \|\vec{w}\|_2^2$$

## Cross-Entropy Loss

类似于上面写出的hinge loss形式，传统的cross-entropy loss是：

$$L(\vec{w}, b) = - \sum_{n=1}^N \left[ \tilde{y}^{(n)} \log \sigma(h^{(n)}) + (1 - \tilde{y}^{(n)}) \log (1 - \sigma(h^{(n)})) \right] + \lambda \|\vec{w}\|^2,$$

$$\tilde{y} \in \{0, 1\}$$

也即

$$L(\vec{w}, b) = \sum_{\ell=1}^N E_{LR}(y^{(\ell)} h^{(\ell)}) + \lambda \|\vec{w}\|^2, \quad y \in \{-1, 1\}$$

其中，

$$E_{LR}(z) = \log(1 + \exp(-z))$$

## Different Kernel Functions

考虑两种不同的核函数：i) 线性核函数; ii) 高斯核函数

- 要maximize的拉格朗日对偶函数是：

$$g(\vec{a}) = \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} K(\vec{x}^{(i)}, \vec{x}^{(j)})$$

分类器是：

$$\hat{y} = \text{sign} \left( \sum_{l=1}^N a_l^* y^{(l)} K(\vec{x}^{(l)}, \vec{x}) + b^* \right)$$

### 线性核函数

$$K(\vec{x}^{(i)}, \vec{x}^{(j)}) = \vec{x}^{(i)\top} \vec{x}^{(j)}$$

### 高斯核函数（RBF 核函数）

$$K(\vec{x}^i, \vec{x}^j) = \exp \left( -\frac{\|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2}{2\sigma^2} \right)$$

## Idea & Code

调用SVM软件包，考虑线性核函数和高斯核函数

根据scikit-learn.svm的documentation，使用其中的SVC类就好

```

2 usage
class SVM:
    def __init__(self, kernel, train_samples, train_labels):
        if kernel == "Gaussian":
            self.kernel = "rbf"
        elif kernel == "Linear":
            self.kernel = "linear"
        self.train_samples = train_samples
        self.train_labels = train_labels
        self.clf = SVC(kernel=self.kernel)

1 usage
def train(self):
    self.clf.fit(self.train_samples, self.train_labels)

1 usage
def predict(self, test_samples):
    """ return type is 'numpy.ndarray' """
    return self.clf.predict(test_samples)

1 usage
def evaluate(self, test_samples, test_labels):
    predictions = self.predict(test_samples)
    accuracy = np.mean(predictions == test_labels)
    print(f"Accuracy: {accuracy:.6%}")
    return accuracy

```

## 关于hinge loss的分类模型与SVM模型之间的关系

### 理论角度分析

- 根据上面“Overview of SVM model”中的第8点想法，其实二者的训练目标/优化目标是意义一致的。
- 从优化（需要minimize）的目标公式来看，hinge loss显然公式没有SVM模型中的  $g(\vec{a}) = \sum_{i=1}^N a_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N a_i a_j y^{(i)} y^{(j)} K(\vec{x}^{(i)}, \vec{x}^{(j)})$  那么复杂
- 但可以看到，SVM模型直接使用核函数计算，避开了在特征空间  $\vec{\phi}$  维度下的计算，于是，能够处理特征空间  $\vec{\phi}$  是极其高维的情况，或者说，甚至能够使用无穷维的  $\vec{\phi}$

，比如高斯核函数对应的  $\vec{\phi}$ 。

- SVM模型，处理 样本维度  $N$  的计算就好。
- 而hinge loss的公式中， $h^{(\ell)} = (\vec{w}^T \vec{\phi}(\vec{x}^{(\ell)}) + b)$  需要在特征空间  $\vec{\phi}$  维度下计算，当  $\vec{\phi}$  维度数很大，计算就很复杂，或者说，没办法在有限计算资源下使用太高维度的  $\vec{\phi}$ 。
- 只是线性模型，不需要非线性化时， $\vec{\phi}(\vec{x})$  取  $\vec{x}$  就好。
- **SVM是通过求解QP优化问题来训练的，不像hinge loss的分类模型是通过反向传播（梯度下降）来更新参数从而训练模型的。**

## 实际训练效果角度分析

- 见下面的 Result & Analysis 部分

## 手动实现hinge loss和cross-entropy loss的线性分类模型

由于要手动实现线性分类模型，这里写一下训练过程中要用到的，两种Loss函数分别对w、b的偏导数（用于梯度下降更新）

- Note：实验中实现的是线性模型，所以下面的  $\vec{\phi}(\vec{x})$  取  $\vec{x}$  就好。

1. hinge loss对w的偏导数：

$$\frac{\partial L}{\partial \vec{w}} = 2\lambda \vec{w} + \sum_{n=1}^N \alpha(\vec{x}^{(n)})$$

其中，

$$\alpha(\vec{x}^{(n)}) = \begin{cases} 0, & \text{if } y^{(n)} (\vec{w}^T \vec{\phi}(\vec{x}^{(n)}) + b) \geq 1 \\ -y^{(n)} \vec{\phi}(\vec{x}^{(n)}), & \text{if } y^{(n)} (\vec{w}^T \vec{\phi}(\vec{x}^{(n)}) + b) < 1 \end{cases}$$

2. hinge loss对b的偏导数：



$$\frac{\partial L}{\partial b} = \sum_{n=1}^N b(\vec{x}^{(n)})$$

其中，

$$b(\vec{x}^{(n)}) = \begin{cases} 0, & \text{if } y^{(n)} \left( \vec{w}^T \vec{\phi}(\vec{x}^{(n)}) + b \right) \geq 1 \\ -y^{(n)}, & \text{if } y^{(n)} \left( \vec{w}^T \vec{\phi}(\vec{x}^{(n)}) + b \right) < 1 \end{cases}$$

3. cross-entropy loss对w、b的偏导数：

$$\frac{\partial L}{\partial \vec{w}} = \sum_{n=1}^N \vec{\phi}(\vec{x}^{(n)}) \left( \sigma(h^{(n)}) - \tilde{y}^{(n)} \right) + 2\lambda \vec{w}, \quad \tilde{y} \in \{0, 1\}$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^N \left( \sigma(h^{(n)}) - \tilde{y}^{(n)} \right)$$

上面的  $h$  是： $h^{(\ell)} = \left( \vec{w}^T \vec{\phi}(\vec{x}^{(\ell)}) + b \right)$

- 实际代码实现时，对loss进行了除以样本数的取平均数操作。
- 大致代码如下：

```

1 usage
2
3 def hinge_loss(labels: np.ndarray, samples: np.ndarray, weights: np.ndarray, bias, lamda):
4     """
5     implementation of hinge loss
6
7     :param labels: one-dim vector, (N, ), and each y in {-1, 1}
8     :param samples: matrix, (N, m)
9     :param weights: one-dim vector, (m, )
10    :param bias: scalar
11    :param lamda: regularization strength
12    :return: hinge loss(a scalar) result of N samples
13    """
14
15    predict = np.dot(weights, samples.T) + bias
16    vec_yh = np.multiply(labels, predict)
17    ESV_result = np.maximum(0, 1 - vec_yh)
18    loss_result = np.sum(ESV_result) / len(labels) + lamda * np.sum(weights ** 2) # average
19    return loss_result
20

```

```

22 def grad_hinge(labels: np.ndarray, samples: np.ndarray, weights: np.ndarray, bias, lamda):
23     """
24     Partial derivative of the loss function with respect to w,b
25
26     :param labels: one-dim vector, (N, ), and each y in {-1, 1}
27     :param samples: matrix, (N, m)
28     :param weights: one-dim vector, (m, )
29     :param bias: scalar
30     :param lamda: regularization strength
31     :return: (grad with respect to w, grad with respect to b)
32     """
33     predict = np.dot(weights, samples.T) + bias
34     compare_with_1_bool = np.multiply(labels, predict) < 1 # return a bool array
35     compare_with_1 = compare_with_1_bool.astype(int) # return a int array
36     compare_column = compare_with_1[:, np.newaxis] # add newaxis for later computing
37     labels_column = labels[:, np.newaxis] # add newaxis for later computing
38     matrix_yx = - (samples * labels_column)
39     grad_w = 2 * lamda * weights + (np.sum(matrix_yx * compare_column, axis=0) / len(labels)) # average
40     grad_b = np.dot(-labels, compare_with_1)
41     return grad_w, grad_b

```

```

44 def sigmoid(x):
45     """
46     x can be scalar or vector, since "/" and 'np.exp' support broadcasting
47     """
48     return 1 / (1 + np.exp(-x))
49
50
51 1 usage
52 def cross_entropy_loss(labels: np.ndarray, samples: np.ndarray, weights: np.ndarray, bias, lamda):
53     """
54     Implementation of cross entropy loss
55
56     :param labels: one-dim vector, (N, ), and each y in {0, 1}
57     :param samples: matrix, (N, m)
58     :param weights: one-dim vector, (m, )
59     :param bias: scalar
60     :param lamda: regularization strength
61     :return: result of CE loss(a scalar)
62     """
63     predict = sigmoid(np.dot(weights, samples.T) + bias)
64     result_no_regularize = - np.sum(labels * np.log(predict + 1e-9) + (1 - labels) * np.log(1 - predict + 1e-9))
65     result = result_no_regularize / len(labels) + lamda * np.sum(weights ** 2) # average
66     return result

```

```
def grad_cross_entropy(labels: np.ndarray, samples: np.ndarray, weights: np.ndarray, bias, lamda):
    """
    :param labels: one-dim vector, (N, ), and each y in {0, 1}
    :param samples: matrix, (N, m)
    :param weights: one-dim vector, (m, )
    :param bias: scalar
    :param lamda: regularization strength
    :return: grad of CE loss with respect to w, b
    """
    predict = sigmoid(np.dot(weights, samples.T) + bias)
    grad_w = np.dot(samples.T, predict - labels) / len(labels) + 2 * lamda * weights # average
    grad_b = np.sum(predict - labels) / len(labels) # average
    return grad_w, grad_b
```

## 两种线性分类的预测器如下

1. 使用hinge loss的：

$$\hat{y} = \text{sign}(\vec{w}^T \vec{x} + b)$$

2. 使用cross-entropy loss的：

$$\hat{y} = \begin{cases} 1, & \text{if } \text{sigmoid}(\vec{w}^T \vec{x} + b) \geq 0.5 \\ 0, & \text{if } \text{sigmoid}(\vec{w}^T \vec{x} + b) < 0.5 \end{cases}$$

```
def hinge_predict(self, input_samples):
    """predict result in {-1, 1}"""
    predict = np.sign(np.dot(self.weights, input_samples.T) + self.bias)
    return predict

1 usage
def sigmoid_predict(self, input_samples):
    """predict result in {0, 1}"""
    predict = sigmoid(np.dot(self.weights, input_samples.T) + self.bias)
    result = np.where(predict >= 0.5, 1, 0)
    return result
```

## 训练过程

### 数据预处理

1. 要先把csv文件，转换成样本的属性（像素值）向量 $\vec{x}^{(\ell)}$ ，和对应的label  $y^{(\ell)}$ ，然后存好到向量和矩阵中

2. 进行normalization，考虑两种如下：

- 第一种（Standardization）是对每个特征维度，把所有样本减去该特征上的样本均值，并除以标准差（方差开根号），比如现在每一行是一个样本：

$$column^{(\ell)} = \frac{column^{(\ell)} - mean^{(\ell)}}{standard^{(\ell)}}$$

- 这样，每一列都被normalize成了  $\mu = 0$ ， $\sigma^2 = 1$  的分布
- 第二种（Min-Max Normalization）是对每个特征维度，把所有样本减去该特征上的最小值，并除以最大值与最小值之差，比如现在每一行是一个样本：

$$column^{(\ell)} = \frac{column^{(\ell)} - min^{(\ell)}}{max^{(\ell)} - min^{(\ell)}}$$

- 这样，每一列的特征都被缩放到了[0, 1]区间内，且最小值是0，最大值是1

```
def standardization(samples):
    mean = np.mean(samples, axis=0)
    std = np.std(samples, axis=0)
    # "samples = (samples - mean) / std" leads to Nan when std = 0
    # samples = np.where(std == 0, 0, (samples - mean) / std) might lead to "RuntimeWarning"
    std_adjust = np.where(std == 0, 1, std)
    samples = (samples - mean) / std_adjust
    return samples

1 usage
def min_max_normalization(samples):
    col_min = np.min(samples, axis=0)
    col_max = np.max(samples, axis=0)
    # still need to consider whether (col_max-col_min)==0
    # difference_adjust = np.where((col_max - col_min) == 0, 1, (col_max - col_min))
    # samples = (samples - col_min) / difference_adjust
    samples = np.where((col_max - col_min) == 0, 0.5, (samples - col_min) / (col_max - col_min))
    return samples
```

3. 随机初始化/零初始化权重向量 $\vec{w}$ 和偏置 $b$ ，且让 $\vec{w} \sim N(\mu = 0, \sigma^2 = 0.01^2)$

```
def para_initialize(self):
    """
    initialize weights and bias
    """
    np.random.seed(4219)
    self.weights = np.random.randn(self.samples.shape[1]) * 0.01 # normal distribution  $N(0.0, 0.01^2)$ 
    self.bias = 0.0 # still 0.0
```

## 训练模型

1. 对于基于不同loss函数的线性分类模型而言：重复epochs次数，用梯度下降方式更新（参数 = 参数 - 学习率\*Loss对参数求导），每个epoch内可以采用mini-batch方式。
2. 对于利用SVM包的模型而言：SVC调用 fit 方法可以直接实现整个参数训练过程，且SVM是基于二次规划优化问题求解的。
3. 对于线性分类模型，最后可以利用matplotlib包绘制loss变化曲线。
4. 在main函数中，只需要依次调用这些方法就好：

```
def main():
    arguments = args()
    train_labels, train_samples_before = extract_data('mnist_01_train.csv')
    train_samples = standardization(train_samples_before)
    # train_samples = min_max_normalization(train_samples_before)
    test_labels, test_samples_before = extract_data('mnist_01_test.csv')
    test_samples = standardization(test_samples_before)
    # test_samples = min_max_normalization(test_samples_before)

    if arguments.model == "linear":
        model = LinearClassifier(arguments.loss_func, train_samples, train_labels, arguments.learning_rate, arguments)
        model.para_initialize()
        model.train()
        model.evaluate(test_samples, test_labels)
        model.plot_loss()
    elif arguments.model == "SVM":
        model = SVM(arguments.kernel_func, train_samples, train_labels)
        model.train()
        model.evaluate(test_samples, test_labels)
```

## 训练技巧（包括相关超参数的设置）

- 使用mini-batch：每一个epoch内，把数据集按batch\_size划分为多个小批次，然后遍历所有批次，每次只使用小批次内的数据去更新参数：

```
def data_shuffle(samples, labels):
    indices = np.arange(len(labels))
    np.random.shuffle(indices)
    shuffled_samples = samples[indices]
    shuffled_labels = labels[indices]
    return shuffled_samples, shuffled_labels
```

```
def train(self):
    """
    train our model with weights already initialized, data already normalized
    """
    for epoch in range(self.epochs + 1):
        # firstly shuffle our data
        shuffled_samples, shuffled_labels = data_shuffle(self.samples, self.labels)

        # then traverse every batch to update
        for j in range(0, len(self.labels), self.batch_size):
            batch_samples = shuffled_samples[j: min(j+self.batch_size, len(self.labels))]
            batch_labels = shuffled_labels[j: min(j+self.batch_size, len(self.labels))]
            self.para_update(batch_labels, batch_samples)

        if epoch % 10 == 0:
            curr_loss = self.curr_loss()
```

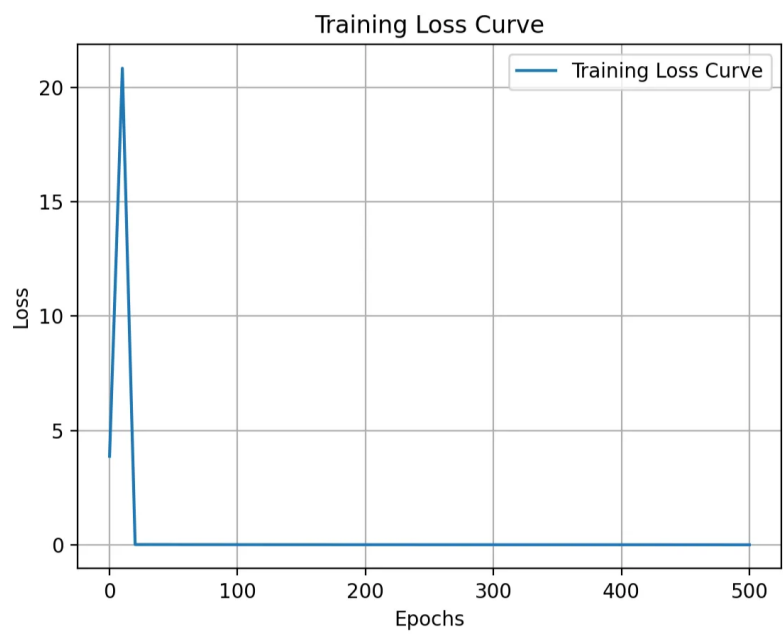
通过使用mini-batch，与没有mini-batch的代码训练效果相比，我发现，由于mini-batch更频繁地更新了模型的参数，此时模型收敛速度得到了明显的加快。

另外，使用了mini-batch后，我发现模型更加稳定了，学习率较大的时候，不会像之前不用mini-batch时出现Loss突然变成几十的情况。

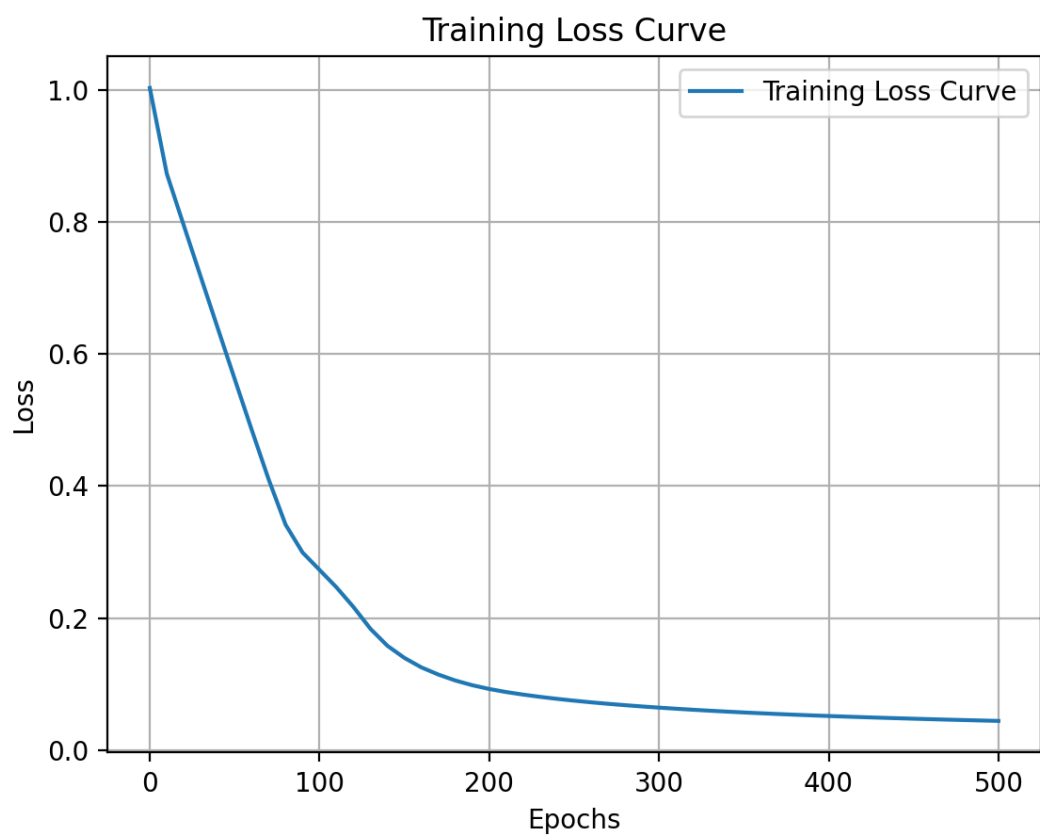
- 在数据预处理中，**standardization**的效果比**min\_max\_normalization**效果更好，相同epochs内收敛到的Loss更小。
- 在进行log计算时，添加一个小的常数（如 `1e-9`）来确保数值运算的稳定性，防止出现比如  $\log(0)$  导致的 NaN 或 `-inf` 错误。

```
result_no_regularize = - np.sum(labels * np.log(predict + 1e-9) + (1 - labels) * np.log(1 - predict + 1e-9))
```

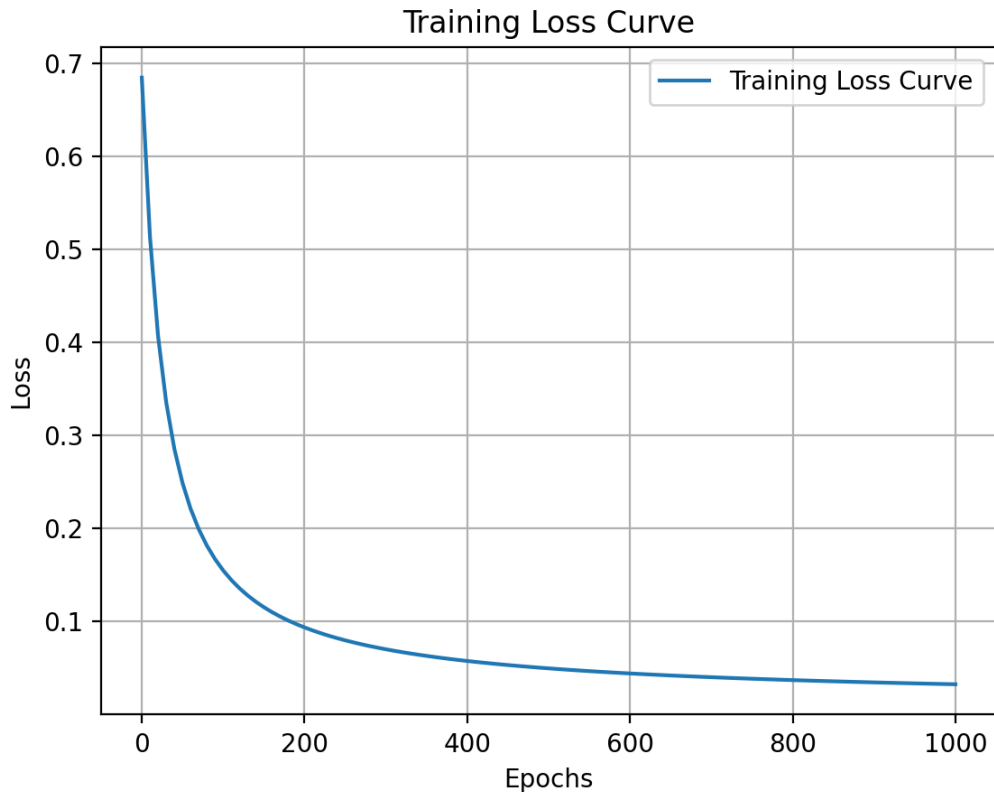
- normalization和standardization 要注意处理除法分母为0的情况，否则会出现loss: Nan，或者导致可能的RuntimeWarning。
- 超参数的设置，要防止学习率过大、更新步长过长，比如，还没有使用mini-batch的时候，使用hinge loss的learning rate设置成1e-2时，训练loss会如下：



- 改成 $1e-4$ 就会好很多：



- 而对于cross-entropy，初步测试 $1e-3$ 就挺好的：



- 多次尝试后，在最终用了mini-batch的代码中，1000 epochs 训练比较合适的一些超参数设置是：（后续测试发现，其实100epochs就能达到一样的accuracy了）

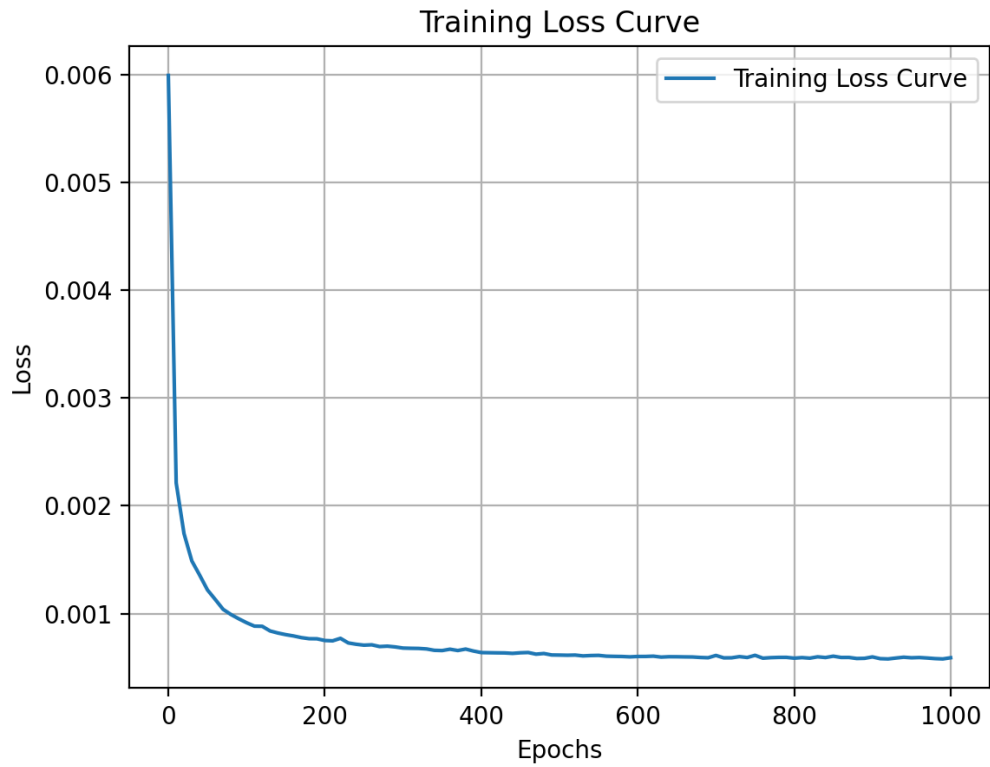
```
python main.py --learning_rate 1e-3 --model "linear" --loss_func "hinge" --regular_strength 1e-3 --epochs 1000
```

```
python main.py --learning_rate 1e-2 --model "linear" --loss_func "hinge" --regular_strength 1e-3 --epochs 1000
```

```
python main.py --learning_rate 1e-2 --model "linear" --loss_func "cross-entropy" --regular_strength 1e-3 --epochs 1000
```

- 对于hinge loss，学习率 $1e-2$ 最后的loss更小，但可以从曲线上看出来Loss还是会稍有抖动，如下：

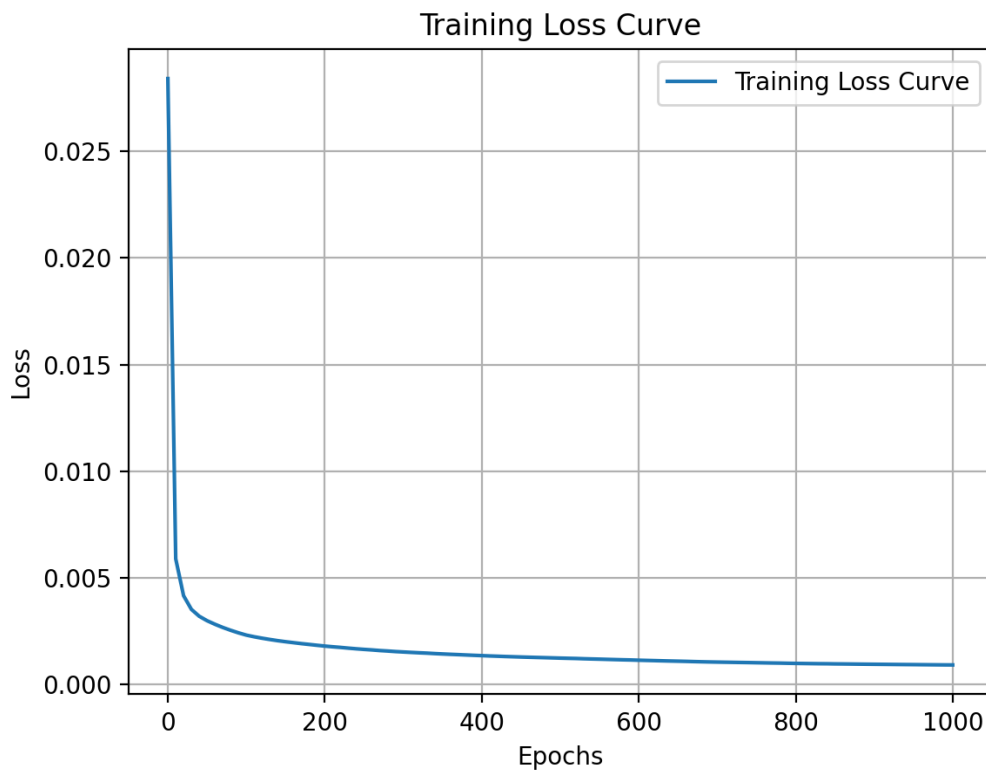




```
Epochs 600, Loss: 0.0006017328847017977
Epochs 700, Loss: 0.0006114306222366303
Epochs 800, Loss: 0.0005861576923749647
Epochs 900, Loss: 0.0005978390391112246
Epochs 1000, Loss: 0.0005899729759066
Accuracy: 99.952719%
```



- 学习率 $1e-3$ 最后的loss大一点，但更稳定，曲线是比较光滑的：



```
Epochs 600, Loss: 0.0011271464006935858
Epochs 700, Loss: 0.001039829538981689
Epochs 800, Loss: 0.000978301679720123
Epochs 900, Loss: 0.0009372693254934227
Epochs 1000, Loss: 0.0009055911885773684
Accuracy: 99.952719%
```

- 比起hinge loss，交叉熵损失在面对较大的学习率时会更稳定一些，但loss会比hinge loss更大。

## 主要关键代码（所有代码见Code文件夹）

- 这里仅给出上面截图没有出现的LinearClassifier类部分：

```

7  class LinearClassifier:
8      """
9      Linear Classifier using hinge loss or cross-entropy loss
10     """
11     def __init__(self, loss_func, samples, labels, lr, epochs, regu_strength, batch_size):
12         self.loss_func = loss_func
13         self.samples = samples
14         if loss_func == "hinge":
15             self.labels = np.where(labels == 0, -1, labels)
16         elif loss_func == "cross-entropy":
17             self.labels = labels
18         else:
19             raise ValueError("Invalid Loss Function")
20         self.lr = lr
21         self.epochs = epochs
22         self.regu_strength = regu_strength
23         self.batch_size = batch_size
24         self.weights = np.zeros(samples.shape[1])
25         self.bias = 0.0 # float
26         self.training_loss = [] # for future drawing
27
28     def para_initialize(self):
29         """
30         initialize weights and bias
31         """
32         np.random.seed(4219)
33         self.weights = np.random.randn(self.samples.shape[1]) * 0.01 # normal distribution N(0.0.01^2)
34         self.bias = 0.0 # still 0.0
35
36     def para_update(self, batch_labels, batch_samples):
37         """
38         update weights and bias with gradient descendant method
39         """
40         if self.loss_func == "hinge":
41             # grad_w, grad_b = grad_hinge(self.labels, self.samples, self.weights, self.bias, self.regu_strength)
42             grad_w, grad_b = grad_hinge(batch_labels, batch_samples, self.weights, self.bias, self.regu_strength)
43         elif self.loss_func == "cross-entropy":
44             # grad_w, grad_b = grad_cross_entropy(self.labels, self.samples, self.weights, self.bias, self.regu_strength)
45             grad_w, grad_b = grad_cross_entropy(batch_labels, batch_samples, self.weights, self.bias, self.regu_strength)
46         else:
47             raise ValueError("Invalid Loss Function")
48         # then update
49         self.weights = self.weights - self.lr * grad_w
50         self.bias = self.bias - self.lr * grad_b

```

```

52 1 usage
53 def curr_loss(self):
54     """
55     :return: current loss of the model
56     """
57     if self.loss_func == "hinge":
58         loss_result = hinge_loss(self.labels, self.samples, self.weights, self.bias, self.regu_strength)
59     elif self.loss_func == "cross-entropy":
60         loss_result = cross_entropy_loss(self.labels, self.samples, self.weights, self.bias, self.regu_strength)
61     else:
62         raise ValueError("Invalid Loss Function")
63     return loss_result
64
65 1 usage
66 def hinge_predict(self, input_samples):
67     """predict result in {-1, 1}"""
68     predict = np.sign(np.dot(self.weights, input_samples.T) + self.bias)
69     return predict
70
71 1 usage
72 def sigmoid_predict(self, input_samples):
73     """predict result in {0, 1}"""
74     predict = sigmoid(np.dot(self.weights, input_samples.T) + self.bias)
75     result = np.where(predict >= 0.5, 1, 0)
76     return result
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

96 def predict(self, test_samples):
97     """predict result in {0, 1}"""
98     if self.loss_func == "hinge":
99         predict_0 = self.hinge_predict(test_samples)
100         predict = np.where(predict_0 == -1, 0, 1)
101     elif self.loss_func == "cross-entropy":
102         predict = self.sigmoid_predict(test_samples)
103     else:
104         raise ValueError("Invalid Loss Function")
105     return predict
106
107 1 usage
108 def evaluate(self, test_samples, test_labels):
109     """
110     evaluate accuracy of our model on testing dataset
111     """
112     test_predict = self.predict(test_samples)
113     accuracy = np.mean(test_predict == test_labels)
114     print(f"Accuracy: {accuracy:.6%}")
115     return accuracy
116
117 1 usage
118 def plot_loss(self):
119     """plot training loss curve"""
120     epochs_points = [i * 10 for i in range(len(self.training_loss))]
121     plt.figure()
122     plt.plot(*args: epochs_points, self.training_loss, label="Training Loss Curve")

```

# Result & Analysis

## 实验结果、分析及讨论

### 不同核函数的SVM模型性能比较与分析

#### 1. 使用高斯核函数

```

(SVMLab) PS E:\2024-1\ML\Assignment1\LAB1> python main.py --model "SVM" --kernel_func "Gaussian"
Accuracy: 99.432624%

```

#### 2. 使用线性核函数

```

(SVMLab) PS E:\2024-1\ML\Assignment1\LAB1> python main.py --model "SVM" --kernel_func "Linear"
Accuracy: 99.905437%

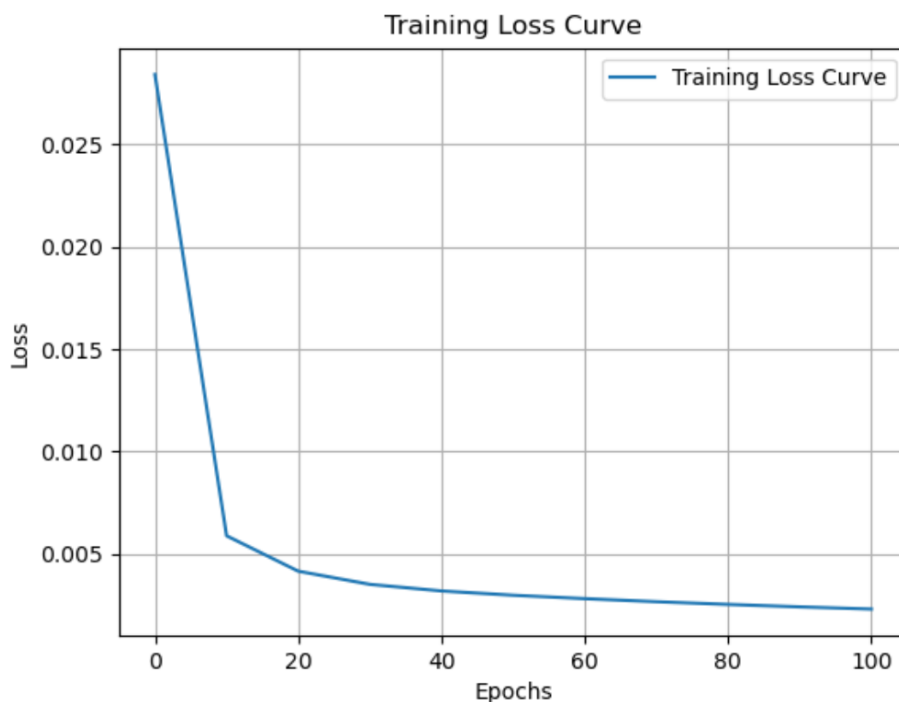
```

1. 发现，这里是使用线性核函数的SVM模型表现更好，在测试集上表现的准确率会比使用高斯核的模型更高一些。
2. 分析原因：高斯核更适用于处理**非线性可分**、更复杂的数据情况，但此时可能我们的数据更偏向于**线性可分**，所以线性核就可以很好地拟合了、且泛化性能好；高斯核反而可能导致在训练集上稍微过拟合，从而泛化性能稍弱，在测试集上的准确率小于高斯核的。
3. 此外，其实准确率还可能与SVC类中参数C、gamma的设置选择有关。
4. 另外，训练时发现，使用高斯核函数的模型出结果要更久一些，也就是使用高斯核函数的模型所需训练时间更长。
5. 分析原因：高斯核函数比线性核函数的表达式更复杂一点，所需计算更多。

## SVM模型和 hinge loss，cross-entropy loss实现的模型的比较与分析

1. 使用hinge loss的线性二分类模型：

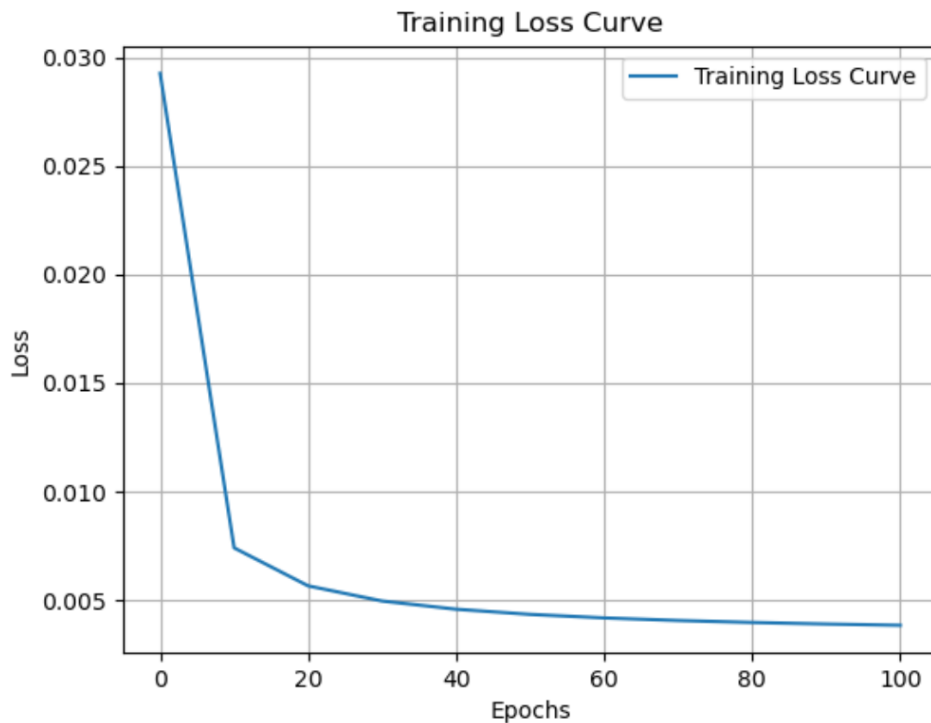
```
python main.py --learning_rate 1e-3 --model "linear" --loss_func "hinge" --regular_strength 1e-3 --epochs 100
```



```
(SVMLab) PS E:\2024-1\ML\Assignment1\LAB1> python main.py --learning_rate 1e-3 --model "linear" --loss_func "hinge" --regular_strength 1e-3 --epochs 100
Epochs 0, Loss: 0.02841984791950114
Epochs 20, Loss: 0.0041555411575420025
Epochs 40, Loss: 0.00318926566915081
Epochs 60, Loss: 0.002814085027011993
Epochs 80, Loss: 0.0025319229309998586
Epochs 100, Loss: 0.002305287837142966
Accuracy: 99.952719%
```

## 2. 使用cross-entropy loss的线性二分类模型：

```
python main.py --learning_rate 1e-2 --model "linear" --loss_func "cross-entropy" --regular_strength 1e-3 --epochs 100
```



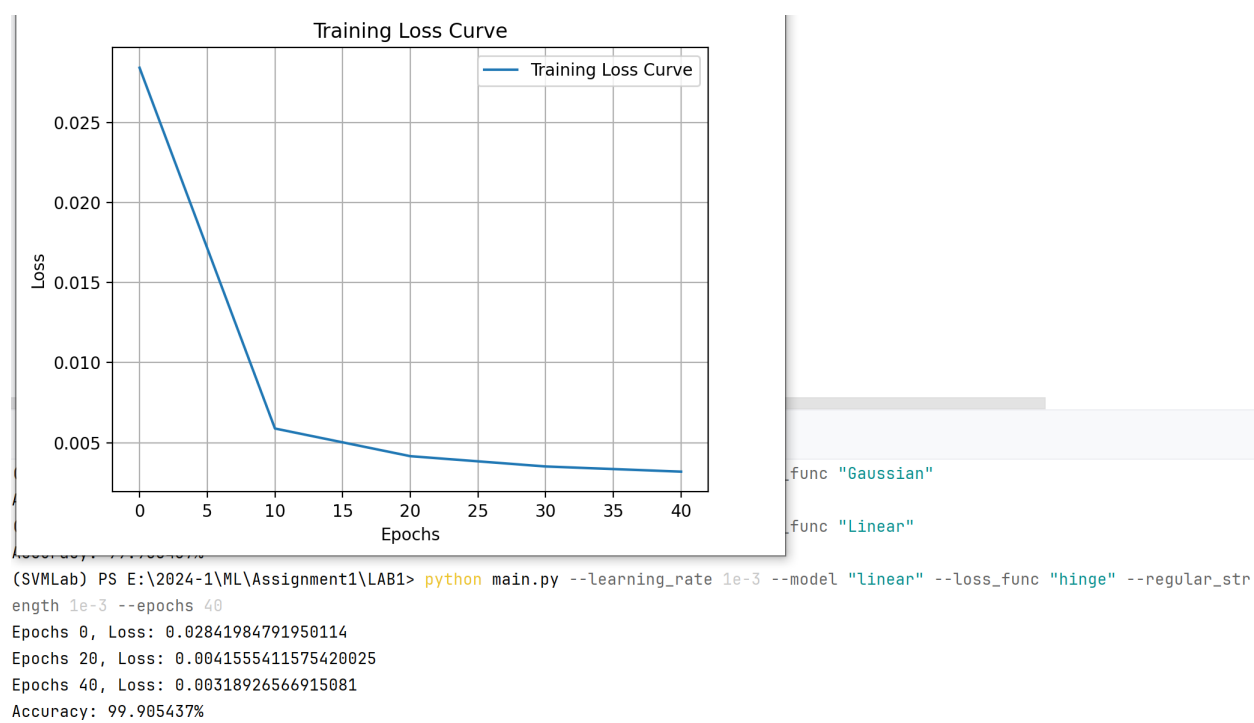
```
(SVMLab) PS E:\2024-1\ML\Assignment1\LAB1> python main.py --learning_rate 1e-2 --model "linear" --loss_func "cross-entropy" --regular_strength 1e-3 --epochs 100
Epochs 0, Loss: 0.029275589416401012
Epochs 20, Loss: 0.005655548164427508
Epochs 40, Loss: 0.004579249319680917
Epochs 60, Loss: 0.004178693779189645
Epochs 80, Loss: 0.0039688536304814045
Epochs 100, Loss: 0.003840023399188348
Accuracy: 99.952719%
```

- 两种线性二分类模型，在100epochs的训练后，就都能有较小的loss和较高的预测准确率了。

- 相比较之下，在学习率相应地恰当选取、模型的loss稳定时，同样的100 epochs训练轮数内，二者预测准确率相同，hinge loss的下降幅度更大，最后的loss更小一些。

### 再与SVM模型比较：

- 线性核函数的SVM模型得到99.905437%的预测准确率大概需要3秒
- 高斯核函数的SVM模型得到99.432624%的预测准确率大概需要6秒
- 而hinge loss的线性模型得到9.905437%的预测准确率大概需要6~十几秒(20epochs/40epochs)



- 可见，直接使用线性核SVM，在现在的数据集情况下会更高效一些。

## Conclusion

- 通过梳理SVM模型理论、调用SVM软件包、手动实现hinge loss和cross-entropy loss的线性分类模型，以及相关数据标准化预处理、mini-batch技巧的实现，我对SVM、二分类机器学习模型更加熟悉了。
- 本来想仿照Adam方式进行参数更新，但发现Adam的真正实现要比课件上多一些细节，要考虑训练初期的偏差修正问题，还需进一步探究。



