西安交通大學

XI'AN JIAOTONG UNIVERSITY

本科生实验报告

机器学习技术综合训练

实验 2：支持向量机

姓名： 杨豪

班级： 软件 2101

时间： 2023 年 4 月 11 日

学号： 2206213297

目录

实验 2：支持向量机

2.1 实验内容

分别使用三种损失函数实现佩加索斯（Pegasos）算法，并在给定的数据集上进行训练与测试。要求：

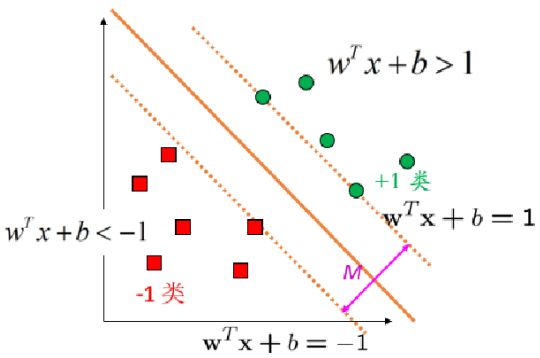
- 代码补充完整，让程序在 3 种 loss 下都可以得出准确率 90% 以上的结果；
- 调整 C、T 参数，看看不同的 C、T 会导致什么变化。

提示：

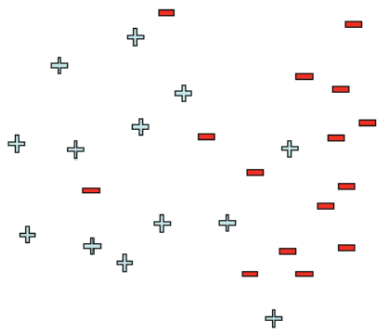
- 提供的测试集中，正负样本数为 308:692，所以如果准确率出现 30.8% 或 69.2%，说明模型将样本全部判定为正/负，相当于训练失败
- 指数操作 (尤其是在指数损失中) 具有不稳定性，可能会导致梯度过大，为避免这种情况，可以对指数进行判断，如果指数过大，则暂时不用当前样本来训练。但该方法并不能完全消除这种不稳定性，建议多次运行程序，选取其中较好的结果。而在对率损失情况下，虽然也有指数操作，但如果 C 设置得当，一般不会出现溢出，可不采用上述方法。

2.2 实验原理

支持向量机 Support Vector Machine(SVM) 假定存在一个超平面能将不同类的样本完全划分开 (分类面方程：$w^T x + b = 0$，支持面方程：$w^T x + b = \pm 1$) 但通常情况并非如此：噪声 (noise)、异常值 (outlier)



(a) SVM



(b) noise and outlier

因而引入软 SVM 建模：惩罚错误分类的数目

$$\min \frac{w^T w}{2} + C(\#\text{mistakes}) = \min \frac{w^T w}{2} + C \cdot \sum_{i=1}^{n} \ell_{0/1}(y_i(w^T x_i + b))$$

$$\ell_{0/1}(z) = \begin{cases} 1, & \text{if } z < 1 \\ 0, & \text{otherwise} \end{cases}$$

其中，C 是正则化常数

$\ell_{0/1}$ 非凸、非连续，数学性质不好, 可使用如下三种替代损失：

- hinge 损失函数：$\ell_{\text{hinge}}(z) = \max(0, 1-z), f(w,b) = \frac{\lambda}{2}\|w\|^2 + \max(0, 1-y_i(w^T x_i + b))$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w} \\ \frac{\partial f}{\partial b} \end{bmatrix} = \begin{cases} \begin{bmatrix} \lambda w - y_i x_i \\ -y_i \end{bmatrix}, & y_i(w^T x_i + b) < 1 \\ \begin{bmatrix} \lambda w \\ 0 \end{bmatrix}, & y_i(w^T x_i + b) \geq 1 \end{cases}$$

- 指数损失函数：$\ell_{\exp}(z) = \exp(-z) f(w,b) = \frac{\lambda}{2}\|w\|^2 + \exp(-y_i(w^T x_i + b))$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w} \\ \frac{\partial f}{\partial b} \end{bmatrix} = \begin{bmatrix} \lambda w - y_i x_i \exp(-y_i(w^T x_i + b)) \\ y_i \exp(-y_i(w^T x_i + b)) \end{bmatrix}$$

- 对数损失函数：$\ell_{\log}(z) = \log(1 + \exp(-z))$,
  $f(w,b) = \frac{\lambda}{2}\|w\|^2 + \log(1 + \exp(-y_i(w^T x_i + b)))$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w} \\ \frac{\partial f}{\partial b} \end{bmatrix} = \begin{bmatrix} \lambda w - \frac{y_i x_i \exp(-y_i(w^T x_i + b))}{1 + y_i \exp(-y_i(w^T x_i + b))} \\ \frac{y_i \exp(-y_i(w^T x_i + b))}{1 + y_i \exp(-y_i(w^T x_i + b))} \end{bmatrix}$$

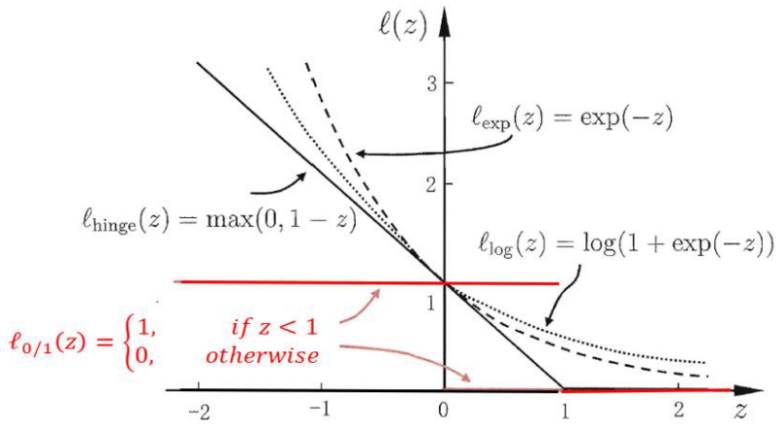


图 2: loss functions

使用（子）梯度下降和佩加索斯（Pegasos）算法求解目标函数随机近似（随机选择一个样本点）

Algorithm 1: Pegasos with SGD

输入： 训练集 $x_i, y_i\ (i = 1, \cdots, n)$

C,T 自定义，$\lambda = \frac{1}{nC}$, 高斯初始化 w_1, b_1

for t *in* $1 : T$ **do**

  $\eta_t = \frac{1}{\lambda t}$

  随机选取第 i 个样本进行参数更新

$$\begin{bmatrix} W_{t+1} \\ b_{t+1} \end{bmatrix} = \begin{bmatrix} W_t \\ b_t \end{bmatrix} - \eta_t \begin{bmatrix} \frac{\partial f(w,b,x_i,y_i)}{\partial w} \\ \frac{\partial f(w,b,x_i,y_i)}{\partial b} \end{bmatrix}$$

end

2.3 框架代码解读、补充与修改

题目已给定的代码（包含部分自己的修改）有三个部分

- plot 函数，打印计算所得数据并保存为对应路径和文件名的图片
- load 函数，读取在 data 文件夹中的 mat 格式数据
- obj_func 函数，根据给定的训练集，W,b,λ 和损失函数类型计算训练集样本的目标函数平均值
- pegasos 函数，根据给定的 W,b, 训练集，测试集，C,T 和损失函数类型，高斯初始化 W，b 并通过佩加索斯算法以 SGD 的形式 T 次迭代出 W 和 b，其中每 func_unit 次计算一次目标函数平均值。最后用测试集计算出准确率。返回目标函数平均值组成的列表和最终的准确率
- 主函数，调用各个函数并输出最终的数据

除 load 函数外均进行了修改，修改内容包括实验要求，拓展和对框架不足的修复，分条叙述如下

2.4.1 基本算法实现

框架代码没有实验最基本核心的迭代部分和正确率的计算，根据理论算法填充如下

```
1 def pegasos(train, test, C, T, loss_type='hinge', func_unit=100):
2     # ..... Some code.....
```

```
3
4       # 初始化lambda_
5       lambda_ = 1 / (C *  len(train_y))
6
7       # 高斯初始化权重W和偏置b
8       W = np.random.randn( len(train_x[0]), 1)
9       b = np.random.randn(1)
10
11      for t in  range(1, T + 1):
12          eta = 1 / (lambda_ * t)
13          i = np.random.randint(0,  len(train_y))
14          xi = train_x[i]
15          yi = train_y[i]
16          yxi = (yi * xi).reshape(1899, 1)
17          resi = yi * (np.dot(np.transpose(W), xi) + b)
18          if loss_type == 'hinge':
19              if resi < 1:
20                  W -= eta * (lambda_ * W - yxi)
21                  b -= eta * (- yi)
22              else:
23                  W -= eta * lambda_ * W
24          elif loss_type == 'exp':
25              if -resi > 3:
26                  resi = -3
27              W -= eta * (lambda_ * W - yxi * np.exp(-resi))
28              b -= eta * (-yi * np.exp(-resi))
29          elif loss_type == 'log':
30              if -resi > 3:
31                  resi = -3
32              W -= eta * (lambda_ * W - yxi * np.exp(-resi) / (1 + np.exp(-
                    resi)))
33              b -= -eta * (yi * np.exp(-resi) / (1 + np.exp(-resi)))
34
35      # ..... Some code.....
36
37      # 比对test数据上预测与实际的结果,统计预测对的个数,计算准确率acc
```

```
    num_correct = 0
    for i in  range( len(test_y)):
        if test_y[i] * (np.dot(np.transpose(W), test_x[i]) + b) > 0:
            num_correct += 1

    # ...... Some code......
```

```
def obj_func(train_x, train_y, W, b, lambda_, loss_type):
    # ...... Some Code ......
    loss_sum = 0
    num_train =  len(train_y)
    for i in  range(num_train):
        resi = train_y[i] * (np.dot(np.transpose(W), train_x[i]) + b)
        resi = resi.item()
        if loss_type == 'hinge':
            loss_sum +=  max(0, 1 - resi)
        elif loss_type == 'exp':
            if -resi > 3:
                resi = -3
            loss_sum += np.exp(-resi)
        elif loss_type == 'log':
            if -resi > 3:
                resi = -3
            loss_sum += np.log(1 + np.exp(-resi))
    return loss_sum / num_train + lambda_ / 2 * np.linalg.norm(W,  ord=2)
```

值得注意的是，正如实验要求和提示中所说的，需要判断一下 exp 的参数是否可能会导致溢出。而对于溢出的处理方式：直接将可能溢出的样本跳过可能会导致很多样本未参与训练，而且不利于统计。因此修改为**若指数溢出则直接赋其为某一固定值**

这样的检查应该并不仅限于 exp 类型的损失函数，尤其是在下一部分中需要加入多种 C 和 T 的可能时。

2.4.2 增加多个 C,T 和损失函数类型

在主函数中以列表的形式列出损失函数，C 和 T，并设置固定的随机种子，从而得以比较不同参数下的准确率，并引入 tabulate 包美化输出

```
if __name__ == '__main__':
```

```
# ...... Some Code ......
    Cs = [0.05, 0.001, 0.0001]
    Ts = [5000, 10000, 100000]  # 迭代次数
    func_unit = 500  # 每隔多少次迭代计算一次目标函数
    np.random.seed(100)
    res = []
    train = load(os.getcwd() + './data/spamTrain.mat', 'train')  # 4000条
    test = load(os.getcwd() + './data/spamTest.mat', 'test')  # 1000条

    for loss_type in loss_types:
        for C in Cs:
            for T in Ts:
                acc, func_list = pegasos(
                    train, test, C, T, loss_type, func_unit)
                res.append([loss_type, C, T, acc])
                plot(func_list, func_unit, loss_type, C, T, acc)
                plt.cla()   # refresh plt
    print(tabulate(res, headers=["LossType", "C", "T", "acc"]))
```

框架代码中对于连续输出图片存在问题，会出现多个数据打印到同一个图的情况，经检查是每次 plt 的缓冲数据未刷新。

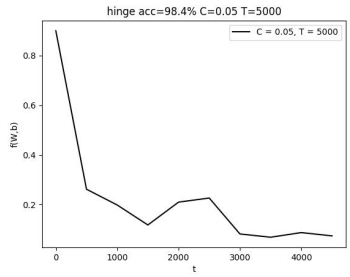
2.4.3 路径与输出整理

原来代码中给定的输出形式在加入 C 和 T 的变化后变得越来越臃肿，且代码中采用终端而非 python 执行程序的相对路径容易造成问题，故修改如下。

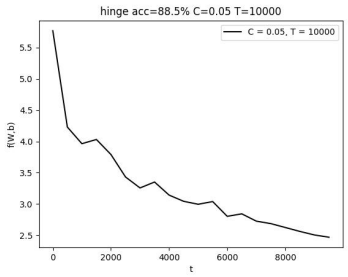
```
if __name__ == '__main__':
    # ...... Some Code ......
    print_record =  open(os.getcwd() + './output/record.txt', 'w')
    sys.stdout = print_record
    # ...... Some Code ......
    print_record.close()
    print_res =  open(os.getcwd() + './output/res.txt', 'w')
    sys.stdout = print_res
    print(tabulate(res, headers=["LossType", "C", "T", "acc"]))
    print_res.close()
```

2.5 结果展示

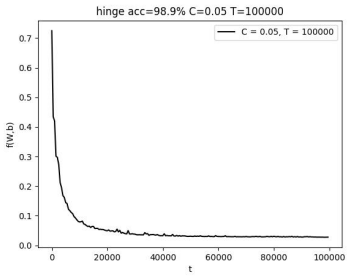
2.5.1 hinge 损失函数



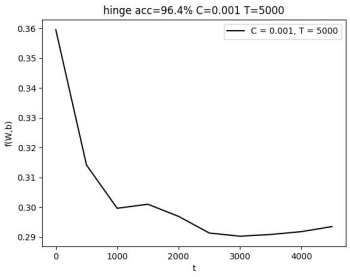
(a) C=0.05,T=5000



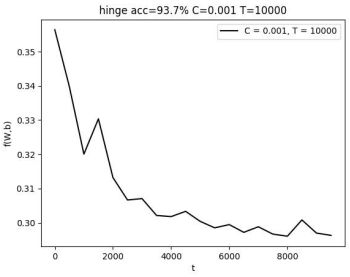
(b) C=0.05,T=10000



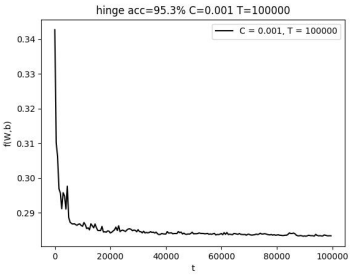
(c) C=0.05,T=100000



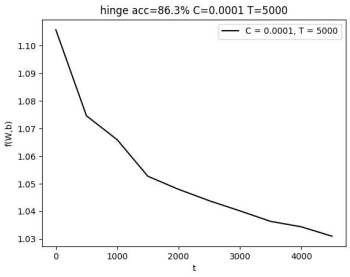
(d) C=0.001,T=5000



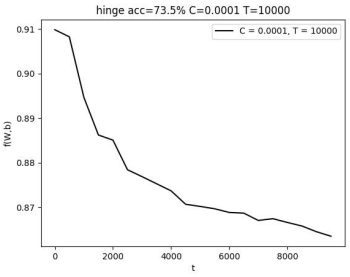
(e) C=0.001,T=10000



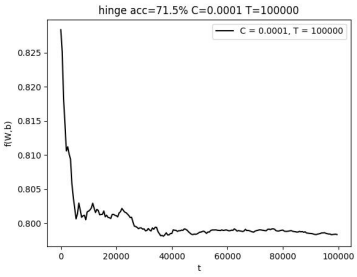
(f) C=0.001,T=100000



(g) C=0.0001,T=5000



(h) C=0.0001,T=10000

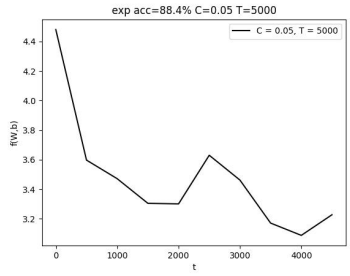


(i) C=0.0001,T=100000

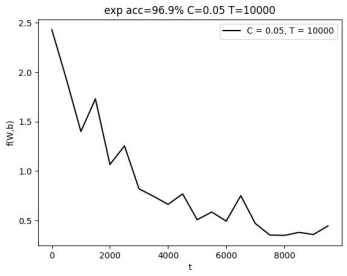
acc T / C	5000	10000	100000
0.05	98.4	88.5	98.9
0.001	96.4	93.7	95.3
0.0001	86.3	73.5	71.5

表 1: hinge 函数准确率

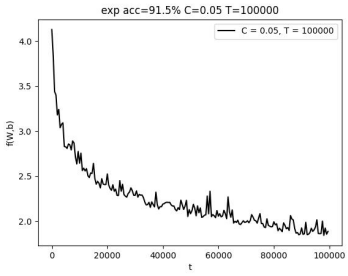
2.5.2 exp 损失函数



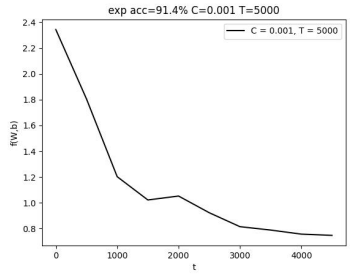
(a) C=0.05,T=5000



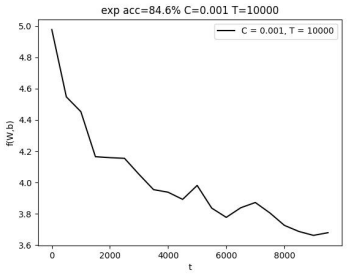
(b) C=0.05,T=10000



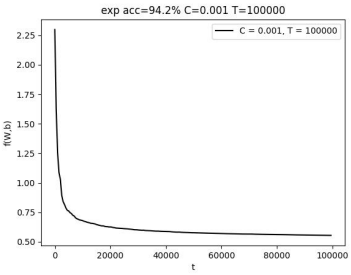
(c) C=0.05,T=100000



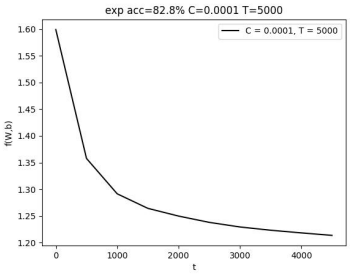
(d) C=0.001,T=5000



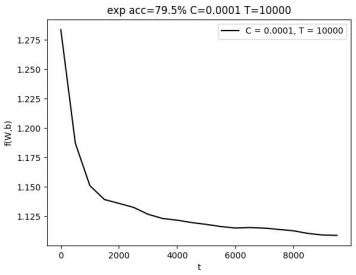
(e) C=0.001,T=10000



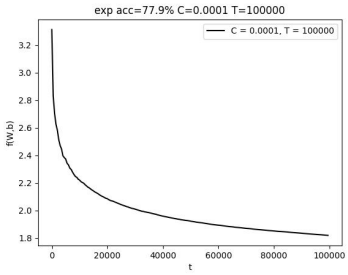
(f) C=0.001,T=100000



(g) C=0.0001,T=5000



(h) C=0.0001,T=10000

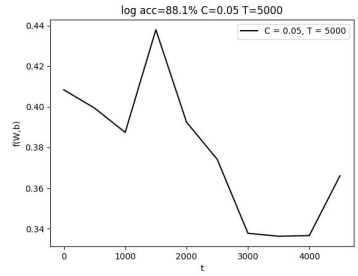


(i) C=0.0001,T=100000

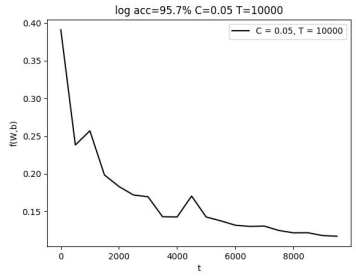
acc C \ T	5000	10000	100000
0.05	88.4	96.9	91.5
0.001	91.4	84.6	94.2
0.0001	82.8	79.5	77.9

表 2: exp 函数准确率

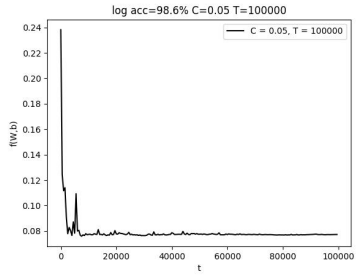
2.5.3 log 损失函数



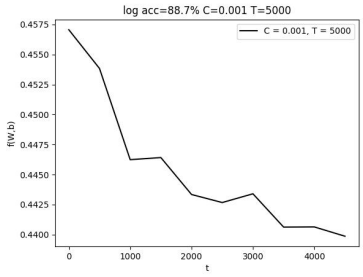
(a) C=0.05,T=5000



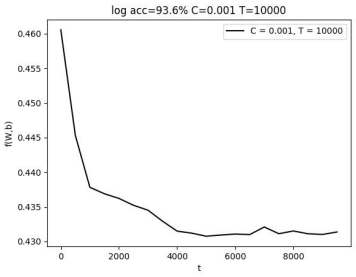
(b) C=0.05,T=10000



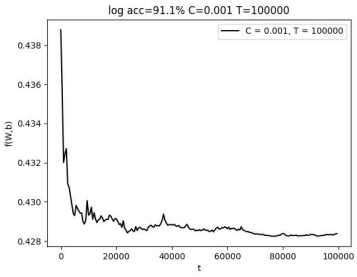
(c) C=0.05,T=100000



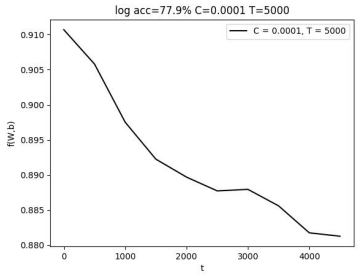
(d) C=0.001,T=5000



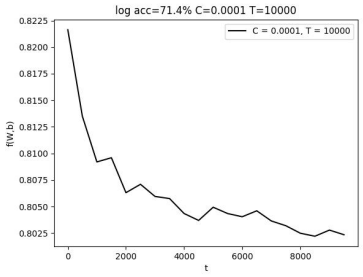
(e) C=0.001,T=10000



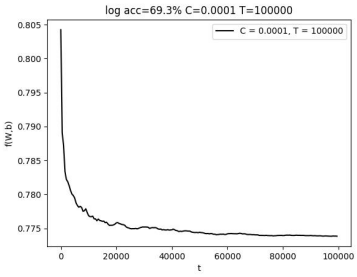
(f) C=0.001,T=100000



(g) C=0.0001,T=5000



(h) C=0.0001,T=10000



(i) C=0.0001,T=100000

acc C \ T	5000	10000	100000
0.05	88.4	96.9	91.5
0.001	91.4	84.6	94.2
0.0001	82.8	79.5	77.9

表 3: log 函数准确率

2.6 总结

由实验原理可知 C 和 η 呈正相关，而 T 表示迭代次数，不难发现：

- 迭代次数的增加有可能产生更好的拟合效果，但不能保证，且有可能会导致相反的结果
- η 即学习率可以看作梯度下降的步长，步长的增大不一定导致好/不好的结果，但**步长的减小一定会导向更差的拟合效果**。以实验数据中 C=0.0001 为例，其目标函数的结果下降趋势总体处于比 C=0.05 和 0.001 更平稳的下降态，而最后的准确率远不如 C=0.05 和 C=0.001. 猜测是由于**小步长在随机梯度算法中容易陷入局部最优而不再跳出，这也和其损失函数曲线平稳下降相印证**。

2.7 附录

2.7.1 源代码

Listing 1: SVM.py

```
import os
import sys
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate

loss_types = ['hinge', 'exp', 'log']


def load(path, data_type):
    """
    根据指定路径读取训练集或测试集
    由于二者的数据格式略有不同，所以需要区分处理
    path:数据集路径
    data_type:"train"或"test"
    """
    data = scipy.io.loadmat(path)
    # 原始数据的label是0/1格式,需要转化为课上学的-1/1格式
    # unit8->int 0/1->-1/1
    if data_type == 'train':
        data['y'] = data['y'].astype( int) * 2 - 1
    elif data_type == 'test':
```

```
24          data['ytest'] = data['ytest'].astype( int) * 2 - 1
25      return data
26
27
28  def obj_func(train_x, train_y, W, b, lambda_, loss_type):
29      """
30      根据当前W和b,计算训练集样本的目标函数平均值
31      """
32      loss_sum = 0
33      num_train =  len(train_y)
34      for i in  range(num_train):
35          resi = train_y[i] * (np.dot(np.transpose(W), train_x[i]) + b)
36          resi = resi.item()
37          if loss_type == 'hinge':
38              loss_sum +=  max(0, 1 - resi)
39          elif loss_type == 'exp':
40              if -resi > 3:
41                  resi = -3
42              loss_sum += np.exp(-resi)
43          elif loss_type == 'log':
44              if -resi > 3:
45                  resi = -3
46              loss_sum += np.log(1 + np.exp(-resi))
47      return loss_sum / num_train + lambda_ / 2 * np.linalg.norm(W,  ord=2)
48
49
50  def plot(func_list, func_unit, loss_type, C, T, acc):
51      """
52      绘制模型在训练过程中的目标函数曲线
53      """
54      ts = [t for t in  range(0, T, func_unit)]
55      plt.plot(ts, func_list, 'k', label='C = {}, T = {}'. format(C, T))
56      print( len(func_list))
57      plt.title('{} acc={}% C={} T={}'. format(loss_type, acc, C, T))
58      plt.xlabel('t')
59      plt.ylabel('f(W,b)')
```

```
60      if not os.path.exists(os.getcwd() + './output'):
61          os.makedirs(os.getcwd() + './output')
62      plt.legend()
63      plt.savefig(os.getcwd(
64      ) + './output/fig/{}/{}_C={}_T={}.jpg'.
            format(loss_type, loss_type, C, T))
65      plt.cla()
66
67
68  def pegasos(train, test, C, T, loss_type='hinge', func_unit=100):
69      """
70      佩加索斯算法
71
72      - `func_unit`:每隔func_unit次记录一次当前目标函数值,用于画图
73      """
74      print('C={}, T={}, loss_type:{}'. format(C, T, loss_type))
75      train_x = train['X']  # 4000*1899
76      train_y = train['y']  # 4000*1
77
78      test_x = test['Xtest']  # 1000*1899
79      test_y = test['ytest']  # 1000*1
80
81      # 记录目标函数值,用于画图
82      func_list = []
83
84      # 初始化lambda_
85      lambda_ = 1 / (C *  len(train_y))
86
87      # 高斯初始化权重W和偏置b
88      W = np.random.randn( len(train_x[0]), 1)
89      b = np.random.randn(1)
90
91      for t in  range(1, T + 1):
92          eta = 1 / (lambda_ * t)
93          i = np.random.randint(0,  len(train_y))
94          xi = train_x[i]
95          yi = train_y[i]
```

```
96          yxi = (yi * xi).reshape(1899, 1)
97          resi = yi * (np.dot(np.transpose(W), xi) + b)
98          if loss_type == 'hinge':
99              if resi < 1:
100                 W -= eta * (lambda_ * W - yxi)
101                 b -= eta * (- yi)
102             else:
103                 W -= eta * lambda_ * W
104         elif loss_type == 'exp':
105             if -resi > 3:
106                 resi = -3
107             W -= eta * (lambda_ * W - yxi * np.exp(-resi))
108             b -= eta * (-yi * np.exp(-resi))
109         elif loss_type == 'log':
110             if -resi > 3:
111                 resi = -3
112             W -= eta * (lambda_ * W - yxi *
113                         np.exp(-resi) / (1 + np.exp(-resi)))
114             b -= -eta * (yi * np.exp(-resi) / (1 + np.exp(-resi)))
115         # 根据当前W和b,计算训练集样本的目标函数平均值
116         if t % func_unit == 0:
117             func_now = obj_func(train_x, train_y, W, b, lambda_, loss_type)
118             func_list.append(func_now)
119             print('t = {}, func = {}'. format(t, func_now))
120
121     # 比对test数据上预测与实际的结果,统计预测对的个数,计算准确率acc
122     num_correct = 0
123     for i in  range( len(test_y)):
124         if test_y[i] * (np.dot(np.transpose(W), test_x[i]) + b) > 0:
125             num_correct += 1
126     acc = 100 * num_correct /  len(test_y)
127     print('acc = {:.1f}%'. format(acc))
128     print('func_list = {}'. format(func_list))
129
130     return acc, func_list
131
```

```
132
133 if __name__ == '__main__':
134     Cs = [0.05, 0.001, 0.0001]
135     Ts = [5000, 10000, 100000]  # 迭代次数
136     func_unit = 500  # 每隔多少次迭代计算一次目标函数
137     np.random.seed(100)
138     res = []
139     train = load(os.getcwd() + './data/spamTrain.mat', 'train')  # 4000条
140     test = load(os.getcwd() + './data/spamTest.mat', 'test')  # 1000条
141
142     print_record =  open(os.getcwd() + './output/record.txt', 'w')
143     sys.stdout = print_record
144
145     for loss_type in loss_types:
146         for C in Cs:
147             for T in Ts:
148                 acc, func_list = pegasos(
149                     train, test, C, T, loss_type, func_unit)
150                 res.append([loss_type, C, T, acc])
151                 plot(func_list, func_unit, loss_type, C, T, acc)
152                 plt.cla()
153     print_record.close()
154
155     print_res =  open(os.getcwd() + './output/res.txt', 'w')
156     sys.stdout = print_res
157     print(tabulate(res, headers=["LossType", "C", "T", "acc"]))
158     print_res.close()
```

2.7.2 程序输出

运行记录过长（预计打印为 80 页），省略。

Listing 2: res.txt

```
1 LossType         C       T    acc
2 ----------  ------  ------  -----
3 hinge       0.05      5000   98.4
4 hinge       0.05     10000   88.5
```

5	hinge	0.05	100000	98.9
6	hinge	0.001	5000	96.4
7	hinge	0.001	10000	93.7
8	hinge	0.001	100000	95.3
9	hinge	0.0001	5000	86.3
10	hinge	0.0001	10000	73.5
11	hinge	0.0001	100000	71.5
12	exp	0.05	5000	88.4
13	exp	0.05	10000	96.9
14	exp	0.05	100000	91.5
15	exp	0.001	5000	91.4
16	exp	0.001	10000	84.6
17	exp	0.001	100000	94.2
18	exp	0.0001	5000	82.8
19	exp	0.0001	10000	79.5
20	exp	0.0001	100000	77.9
21	log	0.05	5000	88.1
22	log	0.05	10000	95.7
23	log	0.05	100000	98.6
24	log	0.001	5000	88.7
25	log	0.001	10000	93.6
26	log	0.001	100000	91.1
27	log	0.0001	5000	77.9
28	log	0.0001	10000	71.4
29	log	0.0001	100000	69.3