本科生实验报告

学生姓名：　丁晓琪

学生学号：　22336057

专业名称：　计科

一：实验任务

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

二：实验过程

1：在不同的核函数的SVM二分类器

(1).SVM的基本理论：

- 线性最大余量分类器：
  - 分界线：$w^Tx+b=0$
  - 余量定义：所有样本点距离分类线的最小距离 $min_l\frac{y^{(l)}\cdot(w^Tx^{(l)}+b)}{||w||}$
  - 训练目标：找到w^*和b^*使得余量最大，这样基于训练数据训练出的分界线能够更好对未知数据进行判断
  - 问题转化：由于求解 $w^*,b^*=argmax_{w,b}\{min_l\frac{y^{(l)}\cdot(w^Tx^{(l)}+b)}{||w||}\}$，既有求max也要求min，可以将问题转化为

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

$$约束：y^{(l)}\cdot(w^Tx^{(l)}+b)>=1,对所有训练样本点l$$

  - 将带约束的优化问题转化为等效对偶问题：注意a为一个训练样本总量维度的向量

$$max_a g(a)$$

$$g(a)=\sum_{l=1}^{N}a_l-\frac{1}{2}\sum_{l=1}^{N}\sum_{j=1}^{N}a_la_jy^{(l)}y^{(j)}x^{(l)T}x^{(j)}$$

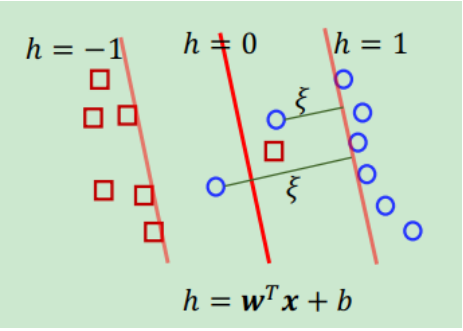
$$约束：a>=0,\sum_{l=1}^{N}a_ly^{(l)}=0$$

  - 当求出上述问题的最优a,可求得最优w^*

$$w^*=\sum_{l=1}^{N}a_ly^{(l)}x^{(l)}$$

$$则分类器变成：\hat{y}(x)=sign(\sum_{l=1}^{N}a_l^*y^{(l)}x^{(l)T}x+b^*)$$

- 软间隔最大余量分类器：
  - 背景：训练样本可能不能用线性分类器完全区分成两类，可能出现在问题转化步骤出现不满足约束 $y^{(l)}\cdot(w^Tx^{(l)}+b)>=1$的样本点，使得永远无法求得最优解



  - 解决：将约束放宽给予松弛量ξ，（ξ>=0）

$$约束：y^{(n)}(\boldsymbol{w}^T\boldsymbol{x}^{(n)}+b)\geq1-\xi$$

$$转化问题：min_{w,b}\frac{1}{2}||w||^2+C\sum_{l=1}^{n}\xi_n$$

- 等效对偶问题：

derived as

$$\max_{\boldsymbol{a}} \; g(\boldsymbol{a})$$

$$s.t.: \quad a_n \geq 0, \quad a_n \leq C$$

When $a_n > C$, it can be shown that $g(\boldsymbol{a}) = -\infty$

$$\sum_{n=1}^{N} a_n y^{(n)} = 0$$

where $g(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m y^{(n)} y^{(m)} \boldsymbol{x}^{(n)T} \boldsymbol{x}^{(m)}$

- SVM:
  - 背景：线性分类器不能对所有样本分类，引入更高维的非线性分类器
  - 关键：需要$\boldsymbol{\phi}$：$x->\boldsymbol{\phi}(x)$将样本的特征值投影到更高维
  - 问题转化：用非线性的$\boldsymbol{\phi}(x)$代替x，且$\boldsymbol{\phi}$(x)越高维度越好，但是计算上会很复杂

$$\min_{\boldsymbol{w},b,\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

$$s.t.: \; y^{(n)} \cdot \left(\boldsymbol{w}^T \boldsymbol{\phi}\left(\boldsymbol{x}^{(n)}\right) + b\right) \geq 1 - \xi_n,$$

$$\xi_n \geq 0, \qquad \text{for } n = 1, 2, \cdots, N$$

  - 等效对偶问题：

$$\max_{\boldsymbol{a}} \; g(\boldsymbol{a})$$

$$s.t.: \quad a_n \geq 0, \quad a_n \leq C$$

$$\sum_{n=1}^{N} a_n y^{(n)} = 0$$

where $g(\boldsymbol{a}) = \underbrace{\sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m y^{(n)} y^{(m)} \boldsymbol{\phi}\left(\boldsymbol{x}^{(n)}\right)^T \boldsymbol{\phi}\left(\boldsymbol{x}^{(m)}\right)}_{=1^T a - \frac{1}{2} a^T M a}$

  - 分类器：

$$\hat{y}(\boldsymbol{x}) = sign\left(\sum_{n=1}^{N} a_n^* y^{(n)} \boldsymbol{\phi}\left(\boldsymbol{x}^{(n)}\right)^T \boldsymbol{\phi}(\boldsymbol{x}) + b^*\right)$$

- 核函数：
  - 背景：即使SVM用等效对偶问题求解时，需要求解的最优化变量a^*的维度只和样本的数量有关，但是$g(a)$中仍然包含着需要复杂计算的 $\boldsymbol{\phi}(\boldsymbol{x}^{(n)})^T \boldsymbol{\phi}(\boldsymbol{x}^{(n)})$
  - 解决：为了优化计算，可以将$\boldsymbol{\phi}(\boldsymbol{x}^{(n)})^T \boldsymbol{\phi}(\boldsymbol{x}^{(n)})$的向量矩阵运算转化为更简单的运算$k(x, x')$，该函数只需要输入$x, x'$就可以以更简便的方式计算$\boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(x')$

(2).SVM的代码实现：

- 初始化：
  将训练数据和测试数据从csv文件中提取出来，用`StandardScaler()`对训练数据的特征值和测试数据的特征值标准化

```
    def __init__(self,filename_train,filename_test):
    # Parameters:
    #  filename_train (str): 训练数据的csv文件名字。
```

```
    #  filename_test (str): 测试数据的csv文件名字。
    # 实例变量：
    # - self.X_train_std: 标准化后的训练数据特征值。
    # - self.Y_train: 训练数据的标签。
    # - self.X_test_std: 标准化后的测试数据特征值。
    # - self.Y_test: 测试数据的标签。
        X_train,self.Y_train=loadCSVfile1(filename_train)
        X_test,self.Y_test=loadCSVfile1(filename_test)
        sc=StandardScaler()
        sc.fit(X_train)
        self.X_train_std=sc.transform(X_train)
        self.X_test_std=sc.transform(X_test)
```

- 用线性核函数在训练数据上训练模型，记录训练时间，且在测试数据上测试模型准确率
  直接调用python的SVC类，且指定内核为liner，`svm.fit()`指定训练数据；`svm.predict()`用训练好的模型预测标签，返回所有预测的标签值；`svm.score()`比较模型对测试集的预测结果与实际的标签，并返回一个准确率

```
    def linear_train(self):
        #线性核svm分类器
        #输出：模型训练时长，在测试样本上的预测正确率和正确预测数
        start_time = time.time()
        svm=SVC(kernel='linear',C=1.0,random_state=1)
        svm.fit(self.X_train_std,self.Y_train)
        end_time = time.time()
        training_duration = end_time - start_time
        print(f"线性核SVM训练时长: {training_duration:.4f} 秒")

        y_pred=svm.predict(self.X_test_std)

        print('linear Misclassified samples: %d' % (self.Y_test !=
y_pred).sum())
        print('linear Accuracy: %.6f' % svm.score(self.X_test_std,
self.Y_test))
```

- 用高斯核函数在训练数据上训练模型，记录训练时间，在测试数据上测试模型准确率，指定内核类型为rbf

```
    def gauss_train(self):
        #高斯核svm分类器
        #输出：模型训练时长，在测试样本上的预测正确率和正确预测数
        start_time = time.time()
        svm=SVC(kernel='rbf',C=1.0,random_state=2)
        svm.fit(self.X_train_std,self.Y_train)
        end_time = time.time()
        training_duration = end_time - start_time
        print(f"高斯核SVM训练时长: {training_duration:.4f} 秒")

        y_pred=svm.predict(self.X_test_std)

        print('gauss Misclassified samples: %d' % (self.Y_test !=
y_pred).sum())
        print('gauss Accuracy: %.6f' % svm.score(self.X_test_std,
self.Y_test))
```

(3).不同核函数的比较分析：

- 线性核：$K(x^i x^j) = x^{iT} x^j$, 和软约束最大余量线性分类器完全一样，没有对数据往更高维投射，保留了原始特征空间的结构，计算简单
- 高斯核：$k(x, x') = exp(-\frac{1}{2\sigma^2}||x - x'||^2)$, 同时将特征空间投射为

$$\boldsymbol{\phi}(x) = e^{-x^2/2\sigma^2}\left[1, \sqrt{\frac{1}{1!\sigma^2}}x, \sqrt{\frac{1}{2!\sigma^4}}x^2, \sqrt{\frac{1}{3!\sigma^6}}x^3, \cdots\right]^T$$

- 用线性核训练的结果 （改变多次随机种子）：

```
线性核SVM训练时长: 0.5570 秒
linear Misclassified samples: 1
linear Accuracy: 1.00
```

```
线性核SVM训练时长: 0.5319 秒
linear Misclassified samples: 1
linear Accuracy: 0.999527
```

```
线性核SVM训练时长: 0.5947 秒
linear Misclassified samples: 1
linear Accuracy: 0.999527
```

- 用高斯核训练的结果：

```
高斯核SVM训练时长: 2.7388 秒
gauss Misclassified samples: 8
gauss Accuracy: 1.00
```

```
高斯核SVM训练时长: 2.7062 秒
gauss Misclassified samples: 8
gauss Accuracy: 0.996217
```

```
高斯核SVM训练时长: 2.7532 秒
gauss Misclassified samples: 8
gauss Accuracy: 0.996217
```

- 比较分析：
  - 高斯核比线性核适合数据非线性可分的情况，高斯核可以映射到高维特征空间，找到超平面分类
  - 高斯核的特征空间是无限维的，非线性的分类能力非常强大
  - 线性核的运算比高斯核的运算更为简单，适合用在数据线性可分的场景
  - 实验中多次训练结果显示线性核在测试集的正确率比高斯核高一点，这说明实验的数据是线性可分的。而且线性核的模型训练时间比高斯核时间短，表现了线性核的计算更为简单

2: 实现采用hinge loss的线性分类模型：

(1). 基于hinge loss的线性分类理论：

- hinge loss定义:

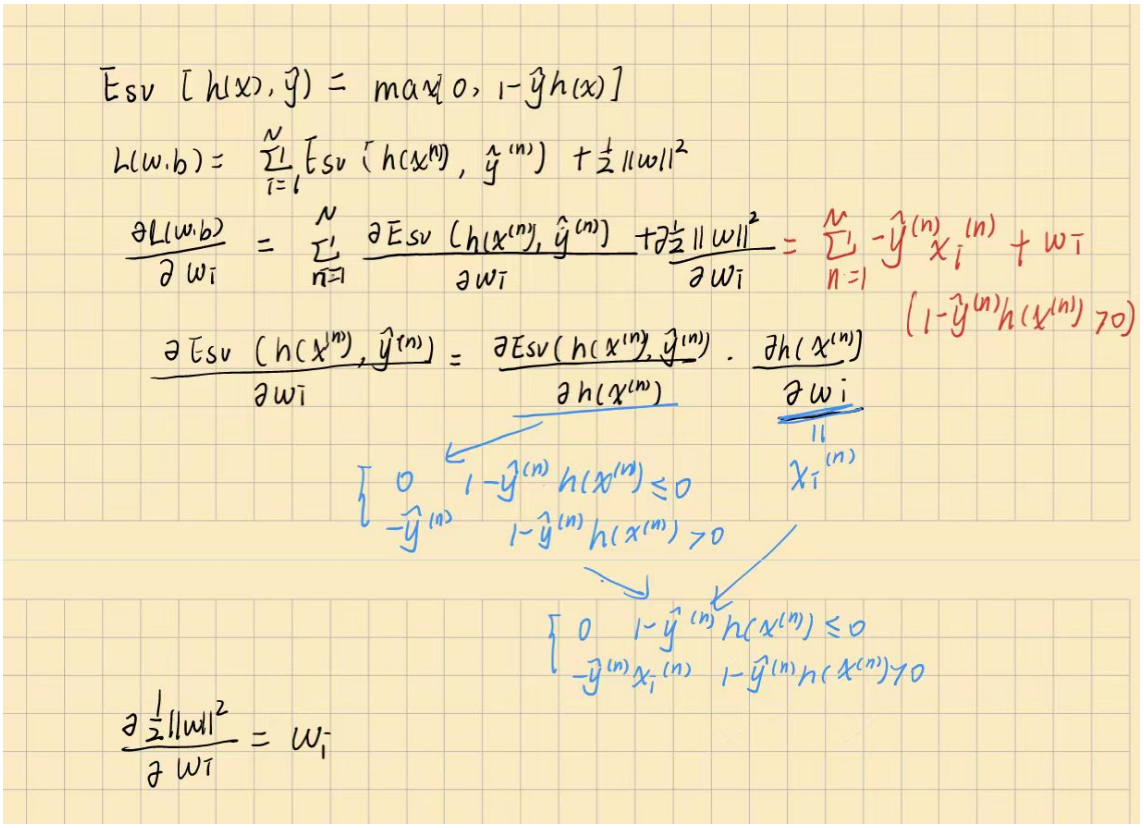
$$L(w,b) = C\sum_{n-1}^{N} E_{SV}(y^{(n)}h^{(n)}) + \frac{1}{2}||w||^2$$

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

$$h(x) = wx + b$$

- 基于hinge loss的线性分类器：$y = wx + b \quad y \in 1, -1,$
  目标：找到w, b使得$L(w,b)$最小。当y=1时，想要正确分类，就需要$wx + b$>0，且要离0远一点，如果$wx + b > 1$,则没有任何惩罚，否则惩罚为$1 - wx - b$
- 训练方法：梯度下降

$$w_i = w_i - \alpha\frac{\sigma L(w,b)}{\sigma w_i}$$



(2). 代码实现：

- 初始化：数据中标签的取值为{0,1}, 而为了训练模型需要把其改为{-1,1}
  由观察得特征的取值范围较大，则参数应该初始化为较小的数

```
    def __init__(self,filename_train,filename_test,alpha,time):

    # Parameters:
    #  filename_train (str): 训练数据的csv文件名字。
    #  filename_test (str): 测试数据的csv文件名字。
    # 实例变量:
    # - self.X_train_std: 标准化后的训练数据特征值。
    # - self.Y_train: 训练数据的标签。{-1,1}
    # - self.X_test_std: 标准化后的测试数据特征值。
    # - self.Y_test: 测试数据的标签。 {-1,1}
    # - self.alpha: 训练学习率
    # - self.time: 训练迭代数
    # - self.w_arg: 模型参数值
    # - self.Loss_list: 存储每次训练周期的loss
```

```
    # Note:
    # - 在X的第一维度为添加的偏置因子
    # - Y更改为{-1，1}

        X_train,Y_train=loadCSVfile1(filename_train)
        X_test,Y_test=loadCSVfile1(filename_test)
        sc=StandardScaler()
        sc.fit(X_train)
        X_train_std=sc.transform(X_train)
        X_test_std=sc.transform(X_test)

        new_column = np.ones((X_train_std.shape[0], 1))
        self.X_train_std=np.hstack((new_column, X_train_std))

        new_column_2= np.ones((X_test_std.shape[0], 1))
        self.X_test_std=np.hstack((new_column_2, X_test_std))

        self.Y_train=Y_train
        self.Y_train[self.Y_train==0]=-1
        self.Y_test=Y_test
        self.Y_test[self.Y_test==0]=-1
        self.time=time
        self.alpha=alpha
        self.w_num=self.X_train_std.shape[1]
        self.w_arg=0.005*np.ones((self.w_num,1))
        self.Loss_list=[]
```

- $L(w,b)$计算：

  公式：$L(w,b)=\sum_{n-1}^{N}E_{SV}(y^{(n)}h^{(n)})+\frac{1}{2}||w||^2$

```
    def Loss(self):

    # 算出所有训练样本的总loss，没有平均所以比较大
    # return:
    # - Loss: self.w_arg对应的当前模型对训练样本的总loss值

        Y_train_Mat=np.mat(self.Y_train).transpose()
        tm=1-
np.multiply(Y_train_Mat,np.dot(self.X_train_std,self.w_arg)) #计算 1-
yh(x)
        w_tm = sum(x**2 for x in self.w_arg) / 2   #计算正则因子
        Loss=0
        Loss+=w_tm
        print(tm.shape)
        for i in range(0,tm.shape[0]):
            if tm[i][0]>0:
                Loss+=(tm[i][0])[0,0]   #计算ESV(yh(x))的和
        return Loss
```

- 单次梯度下降：只要用矩阵向量运算

  $grad=-\sum_{n=1}^{N}y^{(n)}x_i^{(n)}+w_i$且 $1-y^{(n)}h(x^{(n)})>0$

```
    def Gradient_Descent(self):

    # 对参数self.w_arg每个元素梯度下降更新一次
    # Notes:
    # - grad：计算加入了正则因子的梯度，没有平均

        Y_train_Mat=np.mat(self.Y_train).transpose()
        tm=1-
np.multiply(Y_train_Mat,np.dot(self.X_train_std,self.w_arg))  #计算 1-
yh(x)
        tm[tm<=0]=0  #当1-yh(x)<=0时没有惩罚，置为0
        tm[tm>0]=1
        tm=-np.multiply(tm,Y_train_Mat) #每个1-yh(x)>0的惩罚为 -yx_i
        grad=np.dot(self.X_train_std.transpose(),tm)+self.w_arg  #计算梯度
        self.w_arg=self.w_arg-self.alpha*grad #梯度下降
```

- 多次迭代训练模型，记录每次迭代的loss，且在测试集上测试模型的正确率：

```
    def Iterate(self):

    # 1. 梯度更新self.time次，训练模型并且记录loss
    # 2. 计算训练完的模型对测试数据的正确率，并且打印输出总测试个数和总正确数
        # 1:
        for i in range(0,self.time):
            loss=self.Loss()
            print("hinge_loss_train loss: ",loss)
            self.Loss_list.append(loss)
            self.Gradient_Descent()
        # 2:
        predict=np.dot(self.X_test_std,self.w_arg)
        correct_sum=0
        for i in range(0,len(self.Y_test)):
            if(predict[i][0]>=0):
              if self.Y_test[i]==1:
                correct_sum+=1
            elif(predict[i][0]<0):
               if self.Y_test[i]==-1:
                correct_sum+=1
        print("total test:",self.X_test_std.shape[0])
        print("after hinge_loss_train,the correct_sum in test:
",correct_sum)
```

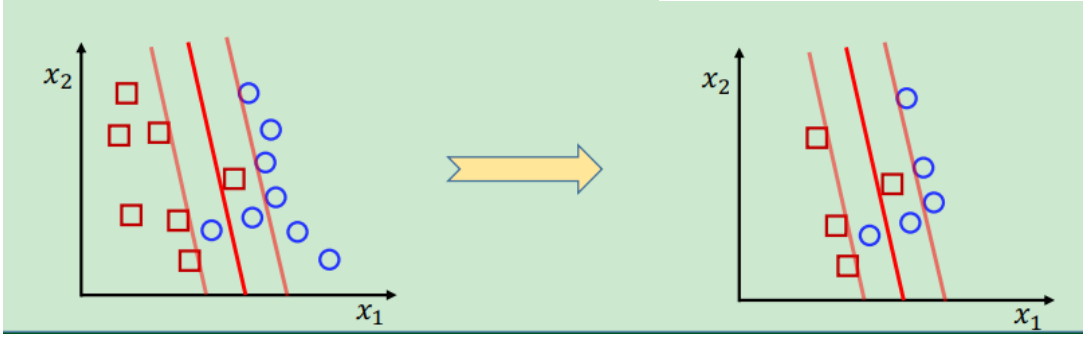
- 实验结果：训练出的模型在测试集上的正确率较高,训练过程中loss不断下降

```
hinge loss训练时长: 4.9577 秒
total test: 2115
after hinge_loss_train,the correct_sum in test:  2113
hinge_loss_train loss:  [[221271.16353214]]
(12665, 1)
hinge_loss_train loss:  [[198894.6509174]]
(12665, 1)
hinge_loss_train loss:  [[178827.77812678]]
(12665, 1)
hinge_loss_train loss:  [[160807.81157938]]
(12665, 1)
hinge_loss_train loss:  [[144597.62945343]]
(12665, 1)
hinge_loss_train loss:  [[130006.57459131]]
(12665, 1)
hinge_loss_train loss:  [[116856.71150755]]
```

(3). 采用 hinge loss 的线性分类模型和 SVM 模型之间的关系:

- hinge loss 和 SVM：svm（软间隔最大余量非线性分类器）在非过拟合的情况下可能无法用一个超平面就将样本完全分类，需要一个ξ给予每个样本可以偏移超平面的松弛量。ξ就是当样本可以被超平面正确分类时为0，而不能被超平面正确分类时，为了达到最小松弛量，ξ取$1-y^{(n)}h(x^{(n)})$, 这和hinge loss的定义相似
- 采用hinge loss的线性分类器和SVM模型：采用hinge loss的线性分类模型为了梯度下降将会将所有训练样本数据纳入计算范围
  而SVM模型由于最优化问题：计算a_n^*时只计算考虑在间隔范围内的样本点

$$a_n^*\left[y^{(n)}\left(\boldsymbol{w}^{*T}\boldsymbol{x}^{(n)}+b^*\right)-1\right]=0$$



3：采用 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型比较：

(1). cross-entropy loss交叉熵线性模型理论（逻辑回归）：

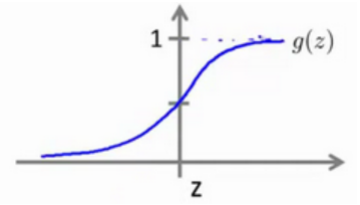
- 逻辑回归的假设函数公式：θ^T为要训练的参数，x为训练数据数据输入的特征

h_θ可以近似看为得到的分类为1概率，当它大于等于0.5时判别分类标签为1，小于0.5时判别分类标签为0

$$h_\theta(x)=g(\theta^T x)$$

$$g(z)=\frac{1}{1+e^{-z}}$$

$g(z)$的图像如下：由此易得$\theta^T x >= 0$，$h_\theta(x) >= 0.5$,即分类为1



- $\theta^T(x)$的选取：有两个特征值分别为x_1(age)和x_2(薪水)还有一个隐藏的$x_0 = 1$作为偏置因子

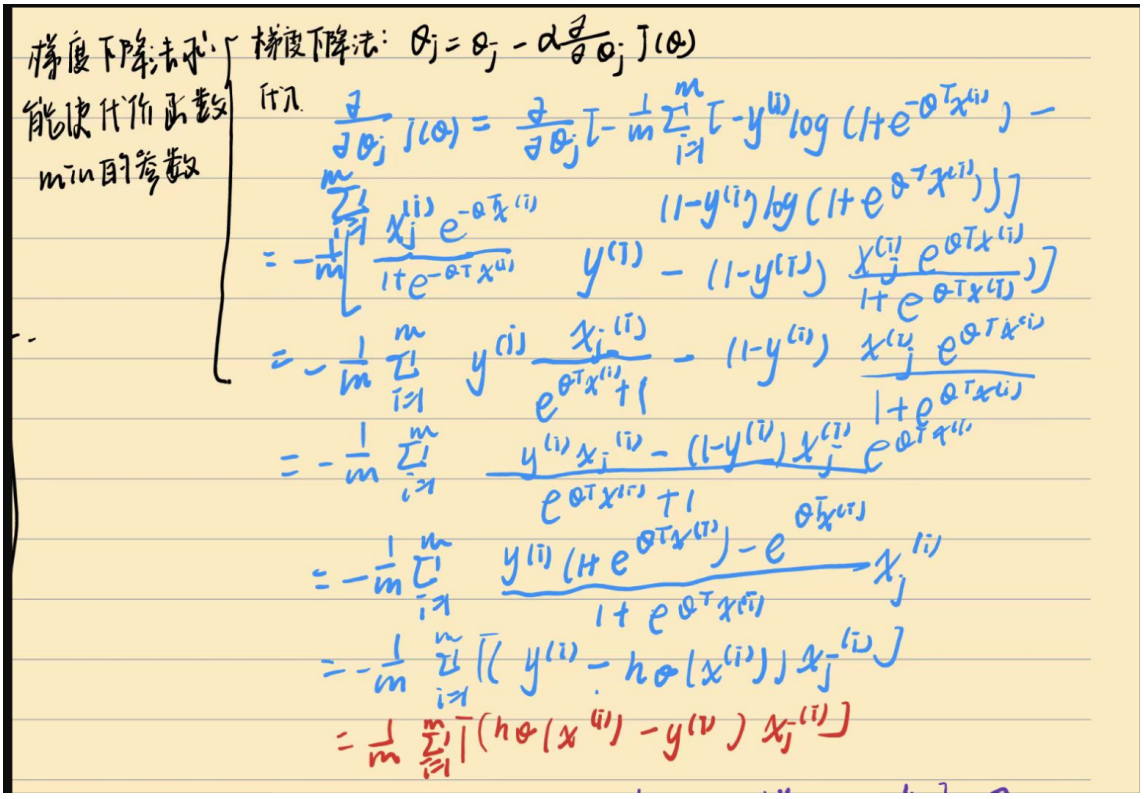
  此次实验实现了两个模型，根据测试模型2的效果更好，以下皆为模型2的分析

  1. $\theta^T(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
  2. $\theta^T(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_2^2$

- 参数θ的训练：使用梯度下降法，$J(\theta)$为代价函数，α为学习率，公式为：

$$\theta_i = \theta_i - \alpha \frac{\delta J(\theta_i)}{\delta \theta_i}$$

注意：每次同时对所以特征参数迭代



- 代价函数$J(\theta)$和损失函数$Cost(h_\theta(x^i), y)$的定义：

$$Cost(h_\theta(x^i), y) = -y * log(h_\theta(x)) - (1-y) * log(1-h_\theta(x))$$

$$J(\theta) = \sum_{i=1}^{m} Cost(h_\theta(x^i), y^i)$$

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y^{(i)}] x_j^{(i)}$$

m：数据的数量　$x^{(i)}$：第i个数据的特征　$x_j^{(i)}$第i个数据的第j个特征　$y^{(i)}$：第i个数据的标签值

- 加入正则化后的参数训练：防止过拟合，为每个参数加上代价惩罚（一般不对偏置因子对应参数θ_0）做惩罚，

$$J(\theta)=\sum_{i=1}^{m}(Cost(h_\theta(x^{(i)})+\frac{\lambda}{m}\theta_j)$$

$$\theta_j=\theta_j-\alpha[\sum_{i=1}^{m}[h_\theta(x^{(i)})-y^{(i)}]x_j^{(i)}+\frac{\lambda}{m}\theta_j]$$

(2). 代码实现：

- 初始化：注意样本的标签取值为{0,1}和hinge loss改为{-1,1}不同，这里由于$h(x)=wx+b$最后要经过sigmoid函数，则w的初始值可以不用取得太小，可以全都取为1

```
    def __init__(self,filename_train,filename_test,alpha,time):

    # Parameters:
    #  filename_train (str): 训练数据的csv文件名字。
    #  filename_test (str): 测试数据的csv文件名字。
    # 实例变量：
    # - self.X_train_std: 标准化后的训练数据特征值。
    # - self.Y_train: 训练数据的标签。{0,1}
    # - self.X_test_std: 标准化后的测试数据特征值。
    # - self.Y_test: 测试数据的标签。 {0,1}
    # - self.alpha: 训练学习率
    # - self.time: 训练迭代数
    # - self.w_arg: 模型参数值
    # - self.Loss_list: 存储每次训练周期的loss
    # Note:
    # - 在X的第一维度为添加的偏置因子

        X_train,Y_train=loadCSVfile1(filename_train)
        X_test,Y_test=loadCSVfile1(filename_test)
        sc=StandardScaler()
        sc.fit(X_train)
        X_train_std=sc.transform(X_train) #特征值标准化
        X_test_std=sc.transform(X_test)   #特征值标准化

        new_column = np.ones((X_train_std.shape[0], 1))  #加上偏置因子
        self.X_train_std=np.hstack((new_column, X_train_std))

        new_column_2= np.ones((X_test_std.shape[0], 1))  #加上偏置因子
        self.X_test_std=np.hstack((new_column_2, X_test_std))

        self.Y_train=Y_train
        print(self.Y_train.shape)
        self.Y_test=Y_test
        self.time=time
        self.alpha=alpha
        self.w_num=self.X_train_std.shape[1]
        self.w_arg=np.ones((self.w_num,1))
        self.Loss_list=[]
```

- Loss计算：

```
    def Loss(self):

    # 算出所有训练样本的总loss，没有平均所以比较大
    # return:
    # - loss[0]: self.w_arg对应的当前模型对训练样本的总loss值

        y=self.Y_train

 Hypothesis_predict=self.sigmoid(np.dot(self.X_train_std,self.w_arg))
        epsilon = 1e-5
        loss=-(np.dot(y,np.log(Hypothesis_predict+epsilon))+np.dot((1-
y),np.log(1-Hypothesis_predict+epsilon)))
        w_tm = sum(x**2 for x in self.w_arg) / 2
        loss[0]+=w_tm
        return loss[0]
```

- 梯度下降：

$grad = (Y - sigmoid(X * W)^T * X)^T$

```
    def Gradient_Descent(self):

    # 对参数self.w_arg每个元素梯度下降更新一次
    # Notes:
    # - grad：计算加入了正则因子的梯度，没有平均

        Y_train_Mat=np.mat(self.Y_train).transpose()
        grad=-np.dot((Y_train_Mat-
self.sigmoid(np.dot(self.X_train_std,self.w_arg))).transpose(),self.X_tr
ain_std).transpose()
        grad+=self.w_arg
        self.w_arg=self.w_arg-self.alpha*grad
```

- 训练模型并且在测试集上测试：

```
    def Iterate(self):

    # 1. 梯度更新self.time次，训练模型并且记录loss
    # 2. 计算训练完的模型对测试数据的正确率，并且打印输出总测试个数和总正确数

        # 1:
        for i in range(0,self.time):
            #阶段性计算损失
            loss=self.Loss()
            print("cross_entropy_train loss: ",loss)
            self.Loss_list.append(loss)
            self.Gradient_Descent()
        #2:
        predict=np.dot(self.X_test_std,self.w_arg)
        correct_sum=0
        for i in range(0,len(self.Y_test)):
            if(predict[i][0]>=0 and self.Y_test[i]==1):
                correct_sum+=1
            elif(predict[i][0]<0 and self.Y_test[i]==0):
                correct_sum+=1
```

```
        print("total test:",self.X_test_std.shape[0])
        print("after cross_entropy_train,the correct_sum in test:
",correct_sum)
```

- 结果：训练时间较长，但是在测试集上准确率高, 训练过程中loss不断下降

```
cross_entropy_loss训练时长: 12.4765 秒
total test: 2115
after cross_entropy_train,the correct_sum in test:  2113
cross_entropy_train loss:  216451.11259586038
cross_entropy_train loss:  194813.46540430858
cross_entropy_train loss:  175341.30651276527
cross_entropy_train loss:  157826.71484693963
cross_entropy_train loss:  142075.98404453497
cross_entropy_train loss:  127913.97247323432
cross_entropy_train loss:  115160.02263454418
cross_entropy_train loss:  103674.06341771333
```

(3). 和 hinge loss 线性分类模型比较:

- 训练时间上：cross-entropy loss 线性分类模型比hinge loss长，cross-entropy loss模型计算中涉及log exp等复杂计算，而hinge loss只有简单的加减乘除
- 训练效果：
  - 在该测试样本上一样的超参数设置下正确率相等且都很高，这侧面体现了数据可以被线性分类。
  - hinge loss线性模型鼓励模型在正确分类的同时，最大化决策边界的间隔，而cross entropy loss线性模型则不考虑优化决策边界的间隔
  - hinge loss线性模型在计算梯度和loss时会看重分类错误和距离决策边界太近的样本点，其他样本点的惩罚都为0，而cross entropy loss关注计算所有样本点，所有训练样本点都有loss

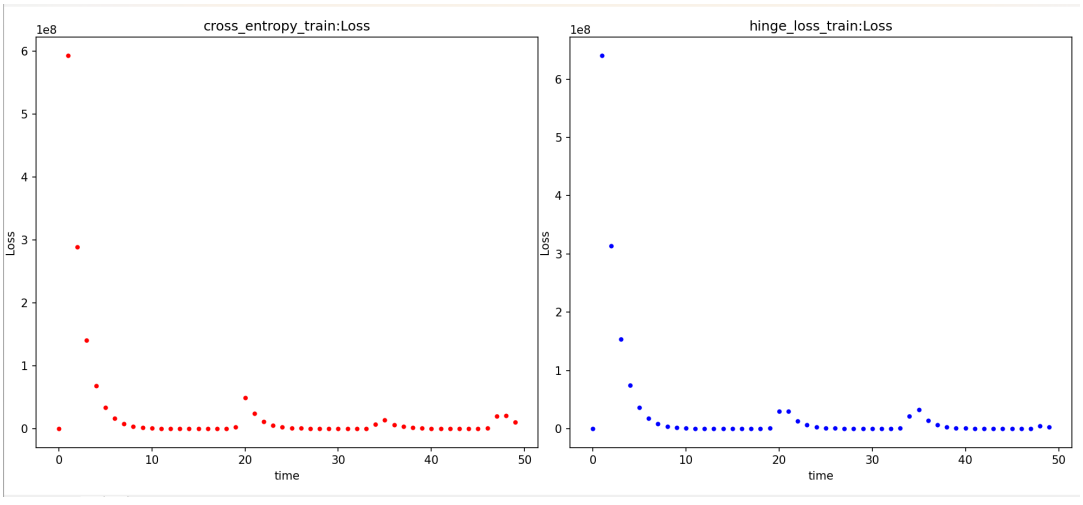
三：实验结果

1. 不同超参数设置：

- 学习率：由于在样本的loss计算和梯度计算时没有对训练样本平均，所以梯度和loss都会比较大，则学习率不宜设的过大，不然会引起震荡

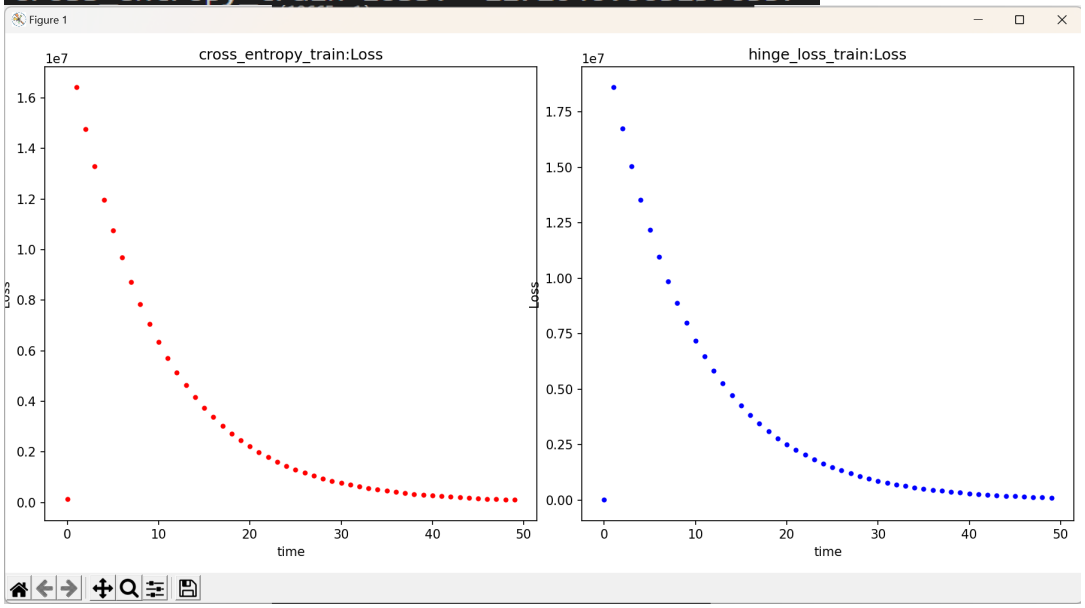
- 设为0.3时，引起震荡：可以看到loss -time图中，无论是hinge loss还是cross entroy loss都随着迭代次数增加有明显震荡

```
cross_entropy_train loss:  99498.93919411008
cross_entropy_train loss:  48088.80434779373
cross_entropy_train loss:  23248.933166441806
cross_entropy_train loss:  11456.315676255299
cross_entropy_train loss:  6526.9761438539135
cross_entropy_train loss:  25164.154453452127
cross_entropy_train loss:  2940536.229923964
cross_entropy_train loss:  49292917.27188314
cross_entropy_train loss:  24103463.119005077
hinge_loss_train loss:  [[111725.21329681]]
(12665, 1)
hinge_loss_train loss:  [[54228.33808916]]
(12665, 1)
hinge_loss_train loss:  [[26388.00429854]]
(12665, 1)
hinge_loss_train loss:  [[13234.21598292]]
(12665, 1)
hinge_loss_train loss:  [[6747.90240255]]
(12665, 1)
hinge_loss_train loss:  [[6897.46080591]]
(12665, 1)
hinge_loss_train loss:  [[875113.64072795]]
(12665, 1)
hinge_loss_train loss:  [[29723810.73586074]]
(12665, 1)
hinge_loss_train loss:  [[29903868.87149192]]
```



- 设为0.05时：loss随着迭代次数的增加一直在下降

```
hinge_loss_train loss:  [[273862.14053153]]
(12665, 1)
hinge_loss_train loss:  [[246195.02753827]]
(12665, 1)
hinge_loss_train loss:  [[221271.16353214]]
(12665, 1)
hinge_loss_train loss:  [[198894.6509174]]
(12665, 1)
hinge_loss_train loss:  [[178827.77812678]]
(12665, 1)
hinge_loss_train loss:  [[160807.81157938]]
(12665, 1)
hinge_loss_train loss:  [[144597.62945343]]
(12665, 1)
hinge_loss_train loss:  [[130006.57459131]]
(12665, 1)
hinge_loss_train loss:  [[116856.71150755]]
cross_entropy_train loss:  5142917.846482186
cross_entropy_train loss:  4627626.665942866
cross_entropy_train loss:  4163869.809798137
cross_entropy_train loss:  3746537.9340782706
cross_entropy_train loss:  3370995.932594235
cross_entropy_train loss:  3033132.171538703
cross_entropy_train loss:  2729130.506767851
cross_entropy_train loss:  2455596.0638074465
cross_entropy_train loss:  2209441.774574745
cross_entropy_train loss:  1987937.0082637875
cross_entropy_train loss:  1788615.1331291283
cross_entropy_train loss:  1609252.4899025213
cross_entropy_train loss:  1447875.0948981256
cross_entropy_train loss:  1302680.9903059856
cross_entropy_train loss:  1172046.0631596337
```



- 迭代次数：一开始设为迭代100次时，发现后期loss下降的曲线越来越平缓，在50次左右趋向稳定了，改为迭代50次，效果和迭代100次差不多，但是减少计算和训练时间

2. 所有实验结果：

样本适合在线性平面内分类，线性核svm和基于hinge loss或者cross-entropy loss的线性分类模型训练正确率都较高

```
高斯核SVM训练时长: 2.7910 秒
gauss Misclassified samples: 8
gauss Accuracy: 0.996217
线性核SVM训练时长: 0.5421 秒
linear Misclassified samples: 1
linear Accuracy: 0.999527
cross_entropy_loss训练时长: 17.2984 秒
total test: 2115
after cross_entropy_train,the correct_sum in test:  2113
hinge_loss_train loss:  [[116856.71150755]]
hinge loss训练时长: 3.1423 秒
total test: 2115
after hinge_loss_train,the correct_sum in test:  2113
```