机器学习 assignment1 实验报告

21307347

陈欣宇

实验内容：训练向量机 SVM 的 2 分类器

一、SVM 模型理论

SVM 是一种用于分类和回归问题的监督学习算法，主要目标是找到一个超平面 wx-b=0，其中 w 是法向量、x 是输入特征向量、b 是偏置项，最大程度分离不同类别的数据点。

(1) 基本的优化问题如下：最大化超平面与训练集最近的点的距离

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

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

(2) 取 $\boldsymbol{w}^* := c\boldsymbol{w}^*$、$b^* := cb^*$，因为 $(c\boldsymbol{w}^*, cb^*)$ 同为最优解，使 $\min_{1\le l\le N} y^{(l)}((c\boldsymbol{w}^*)^T\boldsymbol{x}^{(l)}+cb^*)=1$，得到简化优化问题：

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

➡

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 = \min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \qquad \text{s.t.}\quad y^{(l)}(\boldsymbol{w}^T\boldsymbol{x}^{(l)}+b)\ge 1,\quad 1\le l\le N$$

(3) 通过 SVM 的对偶：使用拉格朗日乘子法得到拉格朗日函数

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

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

通过计算梯度置零，得到下式

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

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

代入得到 SVM 的对偶形式，将问题转化为求解最优α

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

$$\text{s.t.}\quad \alpha_i \ge 0,\quad 1\le i\le n$$

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

再通过α求解 w 和 b：

$$\boldsymbol{w}^* = \sum_{l=1}^{N}\alpha_l^* y^{(l)}\boldsymbol{x}^{(l)} \qquad b^* = \frac{1}{N_S}\sum_{(\boldsymbol{x},y)\in S}(y - (\boldsymbol{w}^*)^T\boldsymbol{x})$$

最终可用于预测样本 x 类别：

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

(4) 改进：对于无法找到超平面完美分隔样本集的情况，使用软间隔，引入松弛变量ξl，为目标函数添加正则项，目标是使总ξl 最小。优化问题改进如下：

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

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

$$\xi_l \geq 0, \quad 1 \leq l \leq N$$

经过同样对偶得到：

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

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

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

(5) 针对非线性的 SVM，我们将原始样本 x 映射到更高维的特征空间，

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

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

$$\xi_l \geq 0, \quad 1 \leq l \leq N$$

同样改写为对偶形式：

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

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

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

使用**核函数**代替 x、y 映射后的内积 $k(\boldsymbol{x},\boldsymbol{y}) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$，得到：

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

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

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

相应的预测公式：（最终需要计算α和 b）

$$\begin{aligned}\hat{y} &= sign(\boldsymbol{w}^*\phi(\boldsymbol{x}) + b^*) \\ &= sign(\sum_{l=1}^{N}\alpha_l^* y^{(l)}\phi(\boldsymbol{x}^{(l)})^T\phi(\boldsymbol{x}) + b^*) \\ &= sign(\sum_{l=1}^{N}\alpha_l^* y^{(l)}k(\boldsymbol{x}^{(l)}, \boldsymbol{x}) + b^*)\end{aligned}$$

本实验用到的核函数

线性核函数：$k(\boldsymbol{x},\boldsymbol{y})=\boldsymbol{x}^T\boldsymbol{y}$，将 SVM 等效为线性分类器

高斯核函数：$k(\boldsymbol{x},\boldsymbol{y})=e^{(-\frac{1}{2\sigma^2}||\boldsymbol{x}-\boldsymbol{y}||^2)}$，称为 RBF 核函数

二、SVM 代码

数据预处理：

```
train= np.loadtxt(open('mnist_01_train.csv','rb'),delimiter=',',skiprows=1)
test= np.loadtxt(open('mnist_01_test.csv','rb'),delimiter=',',skiprows=1)
# 打乱数据
np.random.shuffle(train)
np.random.shuffle(test)
train_label = train[:,0]
train_data = train[:,1:]
test_label = train[:,0]
test_data = train[:,1:]
```

线性核 SVM 初始化与训练

```
LinearSvc = svm.SVC(C=1.0, kernel='linear')
time_start = time.time()
model1 = LinearSvc.fit(train_data, train_label)
time_end = time.time()
print("time:\t%f"%(time_end-time_start))
print("train:\t%f"%(model1.score(train_data, train_label)))
print("test:\t%f"%(model1.score(test_data, test_label)))
```

高斯核 SVM 初始化与训练

```
RbfSvc = svm.SVC(C=1.0, kernel='rbf',gamma='scale')
time_start = time.time()
model2 = RbfSvc.fit(train_data, train_label)
time_end = time.time()
print("time:\t%f"%(time_end-time_start))
print("train:\t%f"%(model2.score(train_data, train_label)))
print("test:\t%f"%(model2.score(test_data, test_label)))
```

其中 SVM 的实现直接调用 sklearn 的 SVM 包

训练结果：

```
time:   0.604422
train:  1.000000
test:   1.000000
time:   1.135073
train:  0.999921
test:   0.999921
```

	训练用时(s)	训练集准确率	测试机准确率
线性核	0.604422	1.000000	1.000000
高斯核	1.135073	0.999921	0.999921

高斯核的训练时间核准确率在这都略逊于线性核，是因为高斯 SVM 内部运算负责，消耗时间长，但同时也适用于更复杂的任务，对于简单的线性分类问题，使用线性核反而效果更好。

三、hinge loss 模型和 SVM 模型之间的关系

hinge loss 可以应用到 SVM 模型中去，其损失函数为

$$h(x)=x_+=\max(0,1-x)$$

用其构造线性模型的损失函数，得到：

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

对 SVM 使用 hinge loss 和松弛变量 C 可得到最新的优化目标：

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

四、比较采用 hinge loss 模型和 cross-entropy loss 模型

同在一个文件 linear_classify 中，数据预处理：

```
train= np.loadtxt(open('mnist_01_train.csv','rb'),delimiter=',',skiprows=1)
test= np.loadtxt(open('mnist_01_test.csv','rb'),delimiter=',',skiprows=1)
np.random.shuffle(train)
np.random.shuffle(test)
train_label=np.empty(train.shape[0])
train_data=np.empty(shape=[train.shape[0],train.shape[1]])
test_label=np.empty(test.shape[0])
test_data=np.empty(shape=[test.shape[0],test.shape[1]]);
# 标签 hinge{-1，1} cross{0,1}
for i in range(train.shape[0]):
    train_label[i]=train[i][0]
    if train_label[i]==0 and func=='hinge':
        train_label[i]=-1
    train_data[i]=train[i][:]
    train_data[i][0]=1
for i in range(test.shape[0]):
    test_label[i]=test[i][0]
    if test_label[i]==0 and func=='hinge':
        test_label[i]=-1
    test_data[i]=test[i][:]
    test_data[i][0]=1
# 特征标准化
m,s = [],[]
for i in range(train_data.shape[1]):
    m.append(np.mean(train_data[:, i]))
    s.append(np.std(train_data[:,i]))
    if s[i] != 0:
        train_data[:, i] = (train_data[:,i]-m[i])/s[i]
        test_data[:, i] = (test_data[:,i]-m[i])/s[i]
```

训练过程：二者同用一个训练过程，通过 func 区分

```
mode = input("select 1--hinge loss  2--cross-entropy loss: ")
    func = ''
    if mode == '1':func = 'hinge'
    elif mode == '2':func = 'cross'
```

初始化 W，进入 epochs 循环，descent 计算 loss 和梯度更新，记录训练过程的 loss 和 acc 最后输出测试集 acc 和训练用时，画出 loss 和 acc 曲线

```
# 设置参数矩阵
    np.random.seed(0)
    W = np.random.rand(train_data.shape[1],1)
    loss_show=[]
    acc_show=[]
    start=time.time()
    for i in range(epochs):
        W,loss = descent(func, W, train_data, train_label,learning_rate)
        loss_show.append(loss)
        acc_test = acc(func,test_data,test_label,W)
        acc_show.append(acc_test)
        if i%10==0:
            print("epochs:%d\tloss:%f\tacc:%f"%(i,loss,acc(func,train_da
ta,train_label,W)))
    end=time.time()
    print('test acc:%f\ttrain time:%f
s'%(acc(func,test_data,test_label,W),end-start))
    draw(loss_show,"loss")
    draw(acc_show,"accuary")
```

其中 descent 函数：

```
def descent(func, w, train_data, train_label,learning_rate):
    if func=='hinge':
        a = np.dot(train_data,w)
        dw=np.zeros(w.shape[0])
        for j in range(a.shape[0]):
            a[j]=a[j]*train_label[j] #wx*y
            if a[j]<1:
               dw-=train_data[j]*train_label[j] # -x*y
        dw/=train_data.shape[0] # -x*y/N
        dw = np.reshape(dw, (-1,1))
        w-=learning_rate*dw # w-= -x*y/N
        # 损失函数 loss = max(0,1-wx*y)
        loss = np.zeros(train_data.shape[0])
        for j in range(loss.shape[0]):
            if a[j]<1:
                loss[j]=1-a[j]
        loss=sum(loss)/train_data.shape[0]
        return w,loss
    elif func=='cross':
        a = np.dot(train_data, w)
        # a = multi(train_data,w)
        dw=np.zeros(w.shape[0])
```

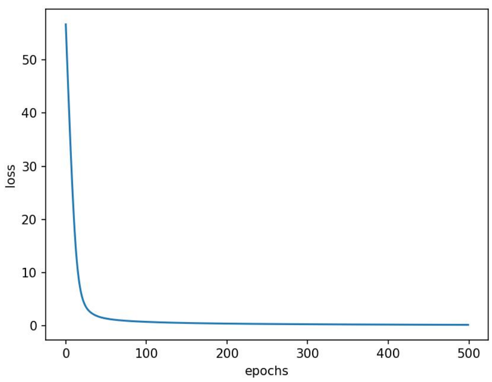
```
        loss = 0.0
        for i in range(a.shape[0]):
            a[i,0] = sigmoid(a[i,0])
            if int(train_label[i]) == 1:
                loss += np.log(a[i,0])
            else:
                if a[i,0]<1:
                    loss += np.log(1-a[i,0])
            dw -= train_data[i]*(train_label[i]-a[i,0])
        dw /= train_data.shape[0]
        dw = np.reshape(dw, (-1,1))
        w -= learning_rate*dw
        loss *= -(1/train_data.shape[0])
        return w,loss
```

acc 函数：

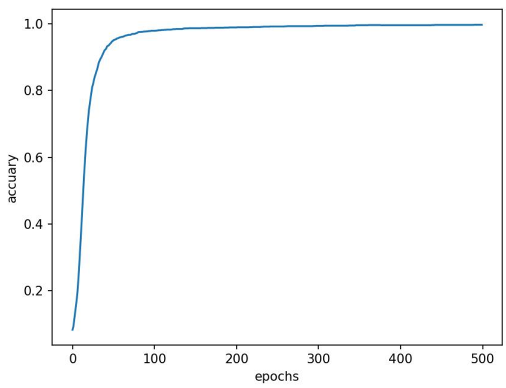
```
def acc(func,x,y,w):
    p=np.dot(x,w)
    if func=='hinge':
        for i in range(x.shape[0]):
            if p[i]>=0:p[i]=1
            else:p[i]=-1
    elif func=='cross':
        for i in range(x.shape[0]):
            if p[i]>=0:p[i] = 1
            else:p[i] = 0
    count=0
    for i in range(y.shape[0]):
        if y[i]==p[i]:count+=1
    return count/y.shape[0]
```

hinge loss 结果：

epochs:500　test acc:0.996690　　train time:40.854105 s



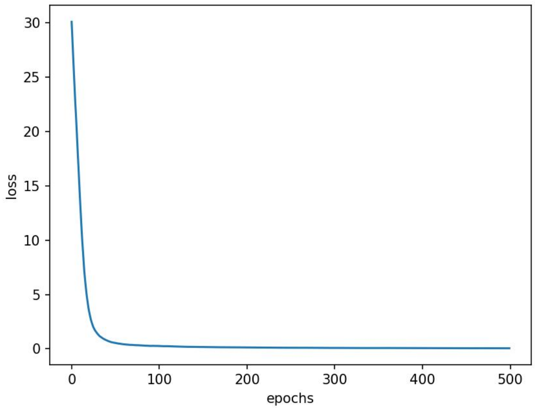
loss 曲线



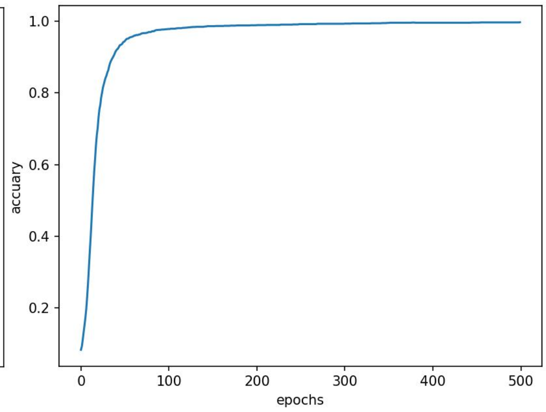
accuary 曲线

cross-entropy loss 结果：

epochs:500　test acc:0.996690　　train time:46.933101 s



loss 曲线



accuary 曲线

在 epochs=500,C=0.05 情况下:

	训练用时	测试集准确率	训练集准确率
hinge	40.854105 s	0.996690	0.994473
cross-entropy	46.933101 s	0.996690	0.994157

可以看出 cross-entropy 的在相同 epochs 下的训练用时较长，这是因为 cross-entropy 中涉及较多的对数和指数运算，导致训练速度较慢。二者在准确度和收敛速度上没有太大差别，从具体数据来看, hinge loss 的收敛速度稍快于 cross-entropy loss, 总的来看, 采用 hinge loss 的训练效果更好