机器学习assignment1实验报告

21307347

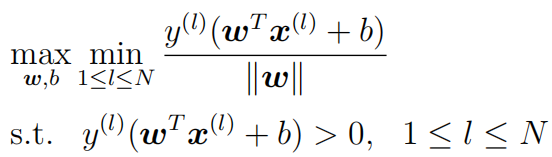
陈欣宇

**实验内容：训练向量机SVM 的2分类器**

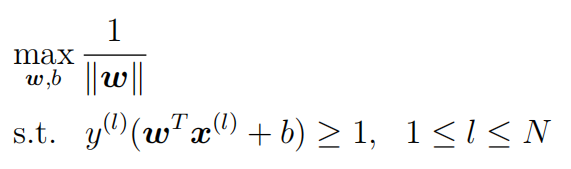
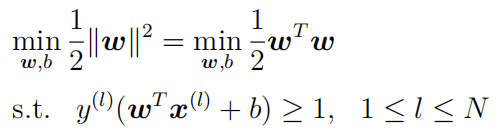
1. **SVM模型理论**

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。

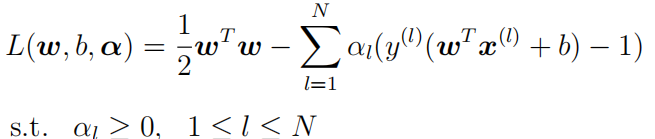
1. 基本的优化问题如下：最大化超平面与训练集最近的点的距离



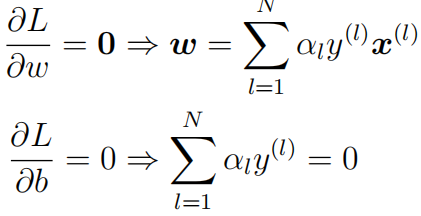
1. 取，因为同为最优解，使，得到简化优化问题：

 ➡

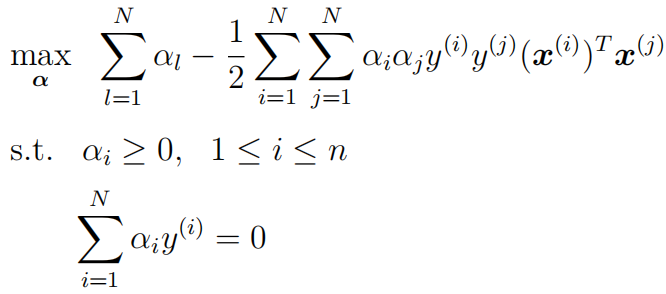
1. 通过SVM的对偶：使用拉格朗日乘子法得到拉格朗日函数

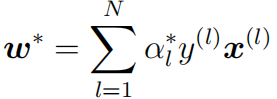
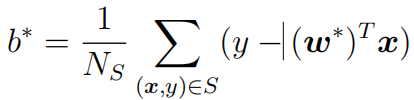


通过计算梯度置零，得到下式



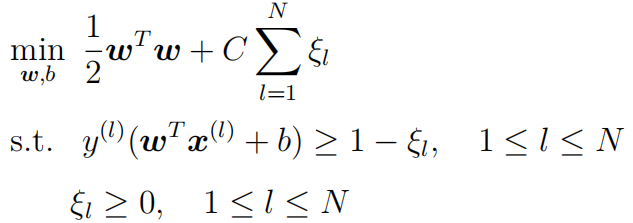
代入得到SVM的对偶形式，将问题转化为求解最优α



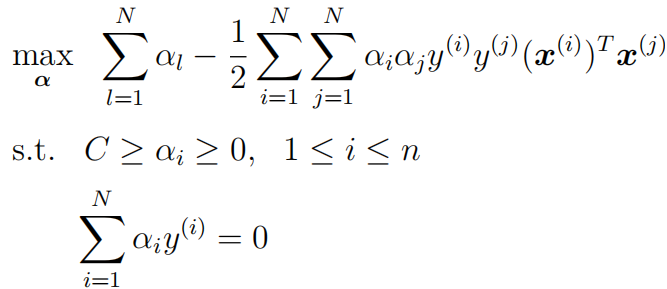
再通过α求解w和b： 

最终可用于预测样本x类别：

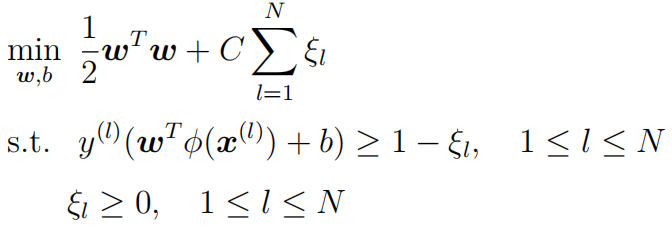
1. 改进：对于无法找到超平面完美分隔样本集的情况，使用软间隔，引入松弛变量ξl，为目标函数添加正则项，目标是使总ξl最小。优化问题改进如下：



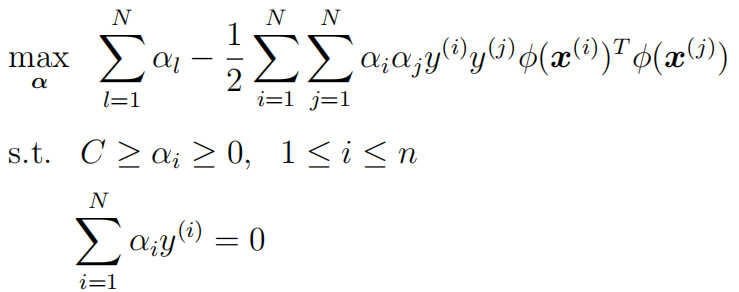
经过同样对偶得到：



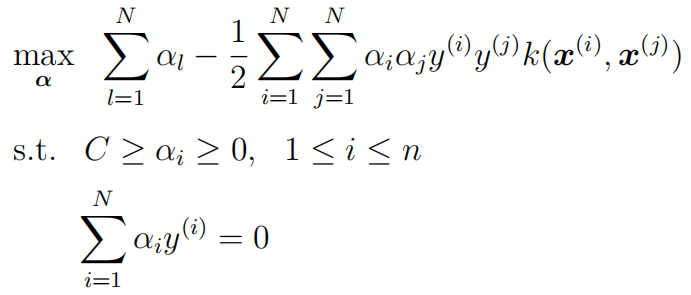
1. 针对非线性的SVM，我们将原始样本x映射到更高维的特征空间，



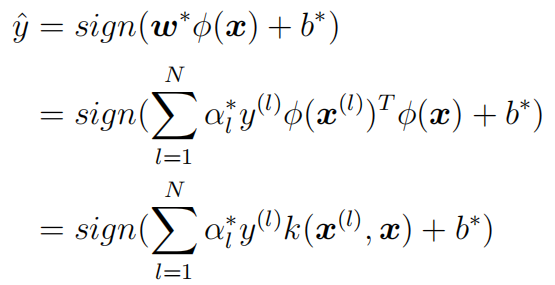
同样改写为对偶形式：



使用**核函数**代替x、y映射后的内积*k*(***x****,* ***y***) = *ϕ*(***x***) *T ϕ*(***y***)，得到：



相应的预测公式：（最终需要计算α和b）



本实验用到的核函数

线性核函数：，将SVM等效为线性分类器

高斯核函数：，称为 RBF 核函数

1. **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))

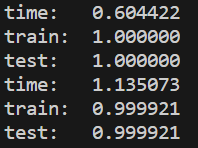
print("train:\t%f"%(model2.score(train\_data, train\_label)))

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：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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** |

****

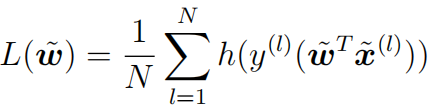
高斯核的训练时间核准确率在这都略逊于线性核，是因为高斯SVM内部运算负责，消耗时间长，但同时也适用于更复杂的任务，对于简单的线性分类问题，使用线性核反而效果更好。

1. **hinge loss模型和SVM模型之间的关系**

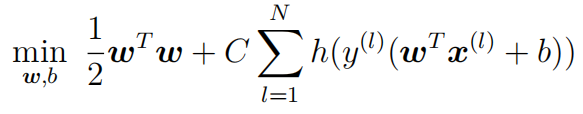
hinge loss 可以应用到SVM模型中去，其损失函数为



用其构造线性模型的损失函数，得到：



对SVM使用hinge loss和松弛变量C可得到最新的优化目标：



1. **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\_data,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

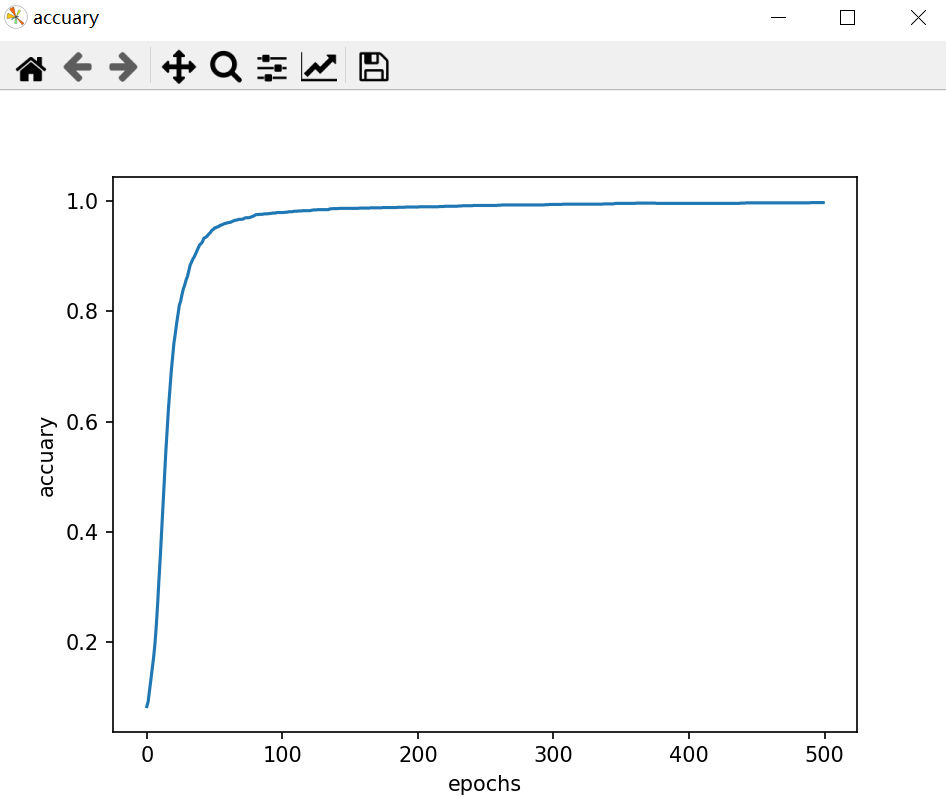
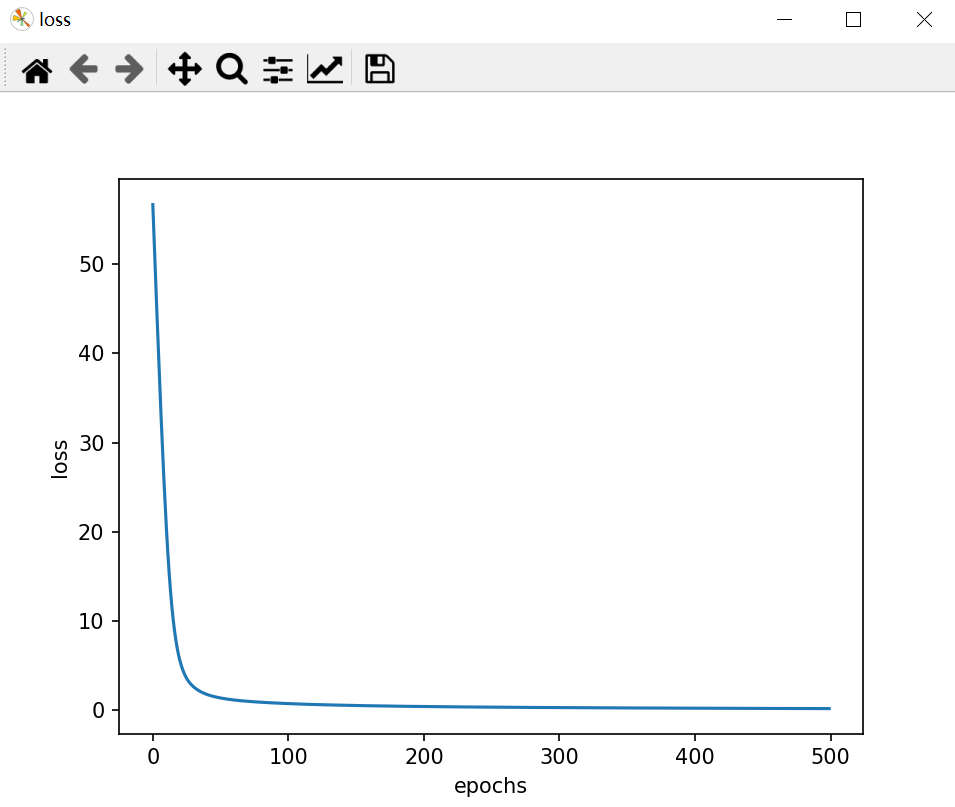
    for i in range(y.shape[0]):

        if y[i]==p[i]:count+=1

    return count/y.shape[0]

hinge loss 结果：

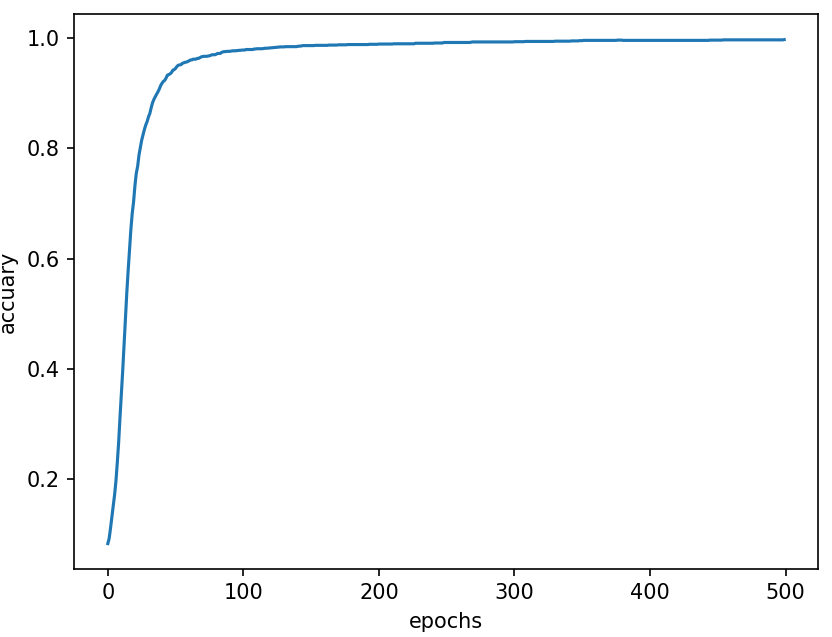
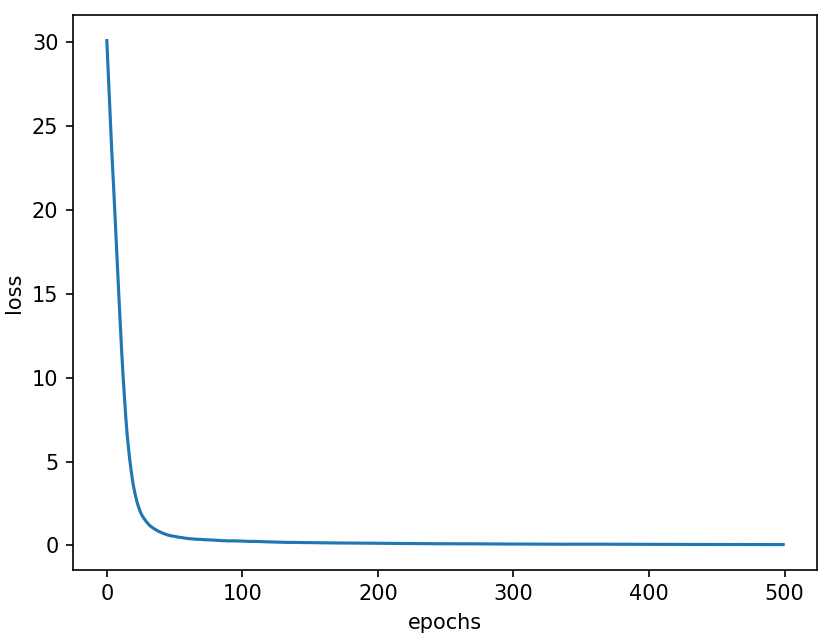
epochs:500 test acc:0.996690 train time:40.854105 s

****

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