

**机器学习作业**

**题 目: SVM vs. Logistic回归 vs. ANN**

**学 号: 919106840333**

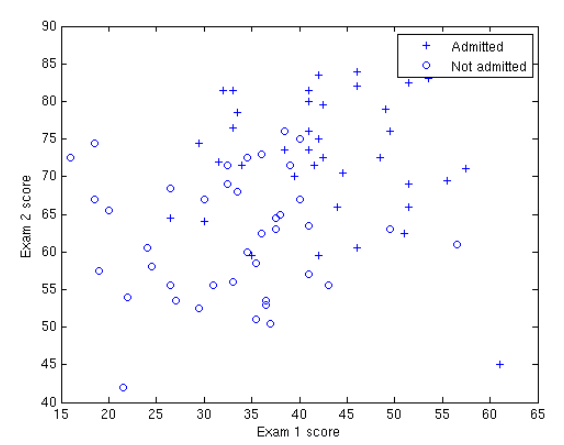
**姓 名: 孙傲歆**

**指导教师: 潘金山**

**2021年12月**

# 一、题目要求

• 给出下列训练数据:



• 编码实现Logistic回归模型（基于SGD），并对结果进行5倍交叉验证；

• 使用 Tensorflow实现三层前馈神经网络 ，并对结果进行5倍交叉验证；

• 基于LibSVM利用软间隔SVM解决上述问题，调节参数C和核函数，并将结果 与Logistic回归、三层前馈ANN进行比较。

# 二、基于SGD的Logistic回归模型求解

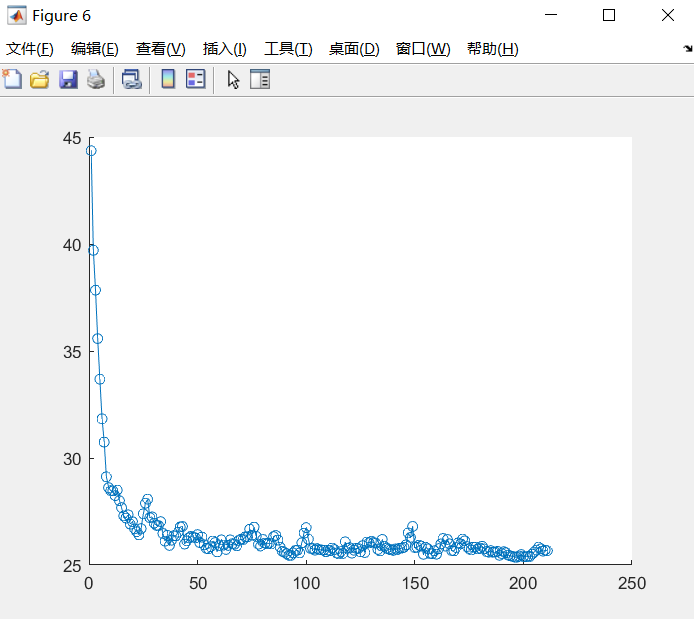
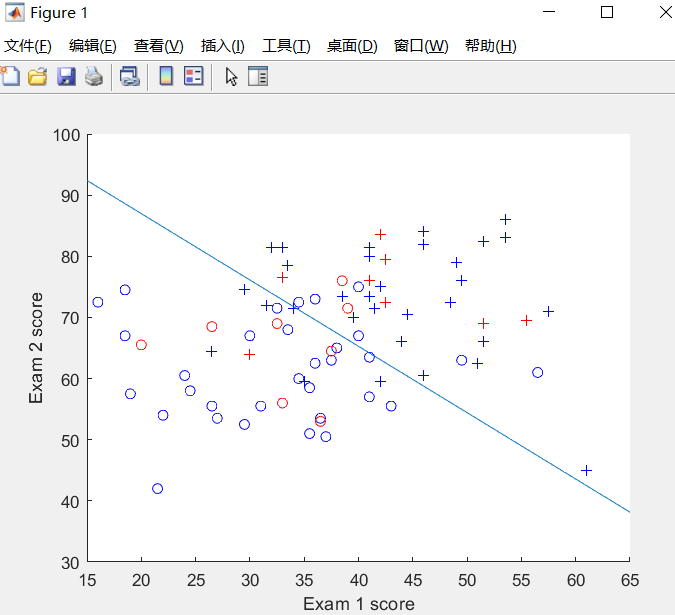
## 2.1 算法思路

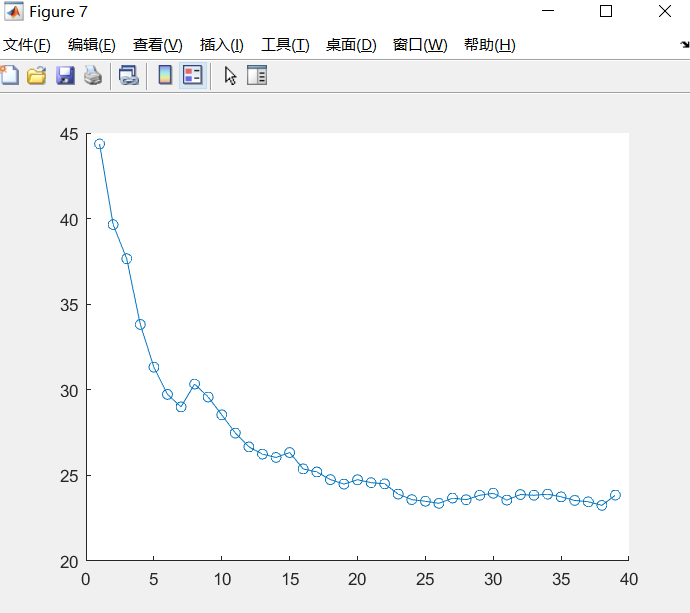
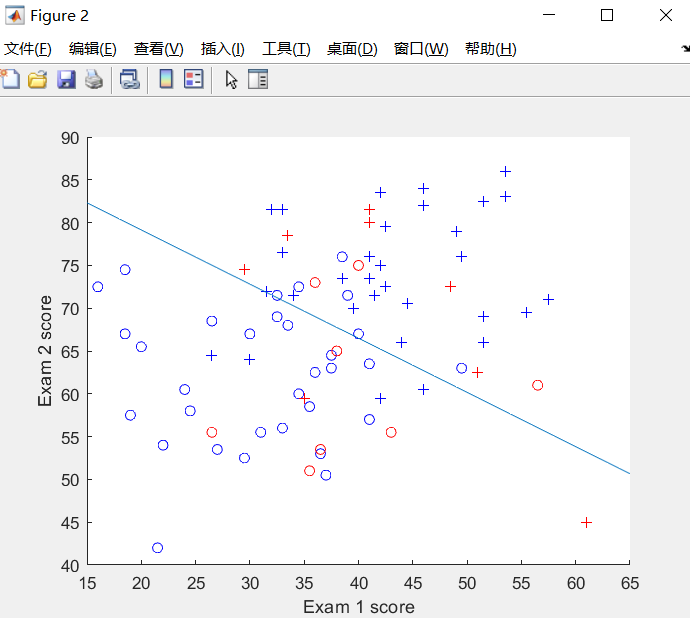
根据题目所给数据，可以确定目标函数形式为。且y只在1和0之间取值。我们先将x数据进行标准化处理，公式为：(X-X\_mean)/X\_std。我们取𝜃向量为[0;0;0]（即全0列向量）。并设置学习率a=0.1，按照随机梯度下降法的公式进行迭代，每次选取5个样本加入计算。当两次迭代结果之差小于0.001，我们认为结果收敛，停止迭代。

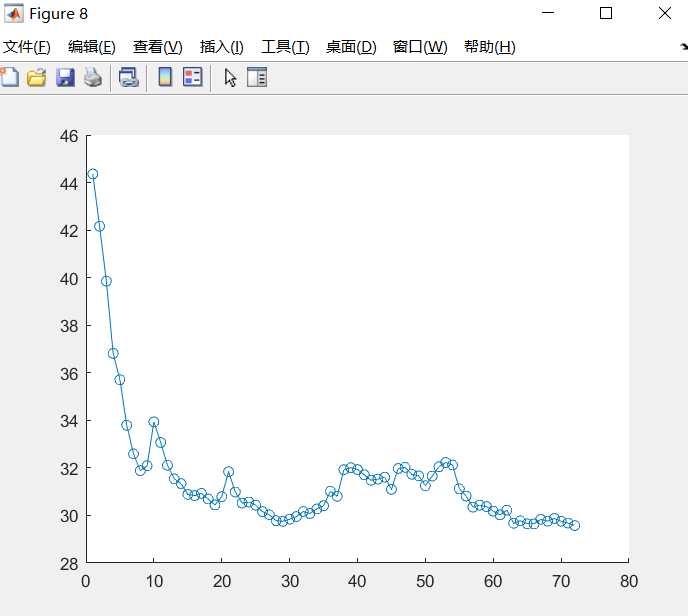
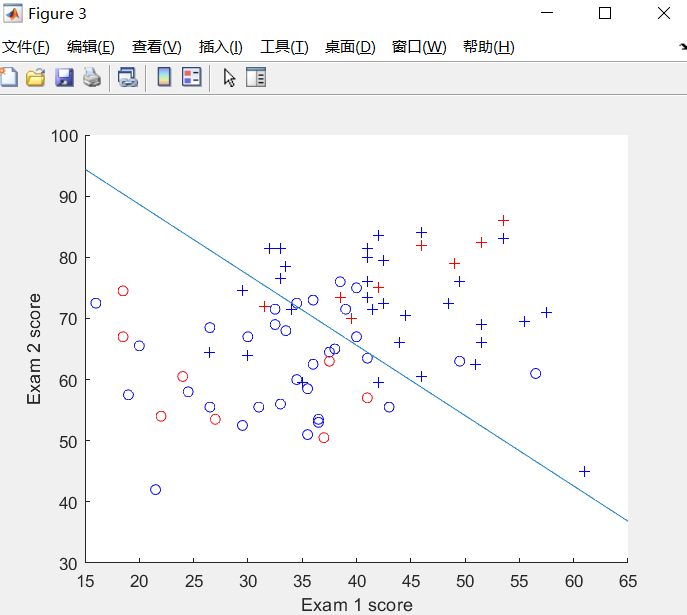
而5倍交叉验证法就是将整个数据集分为不相交的5个部分，然后选取其中的一个部分为测试集，其他部分为训练集，获得测试准确率，然后重复上述步骤。每次计算训练集有64个数，测试集有16个数。

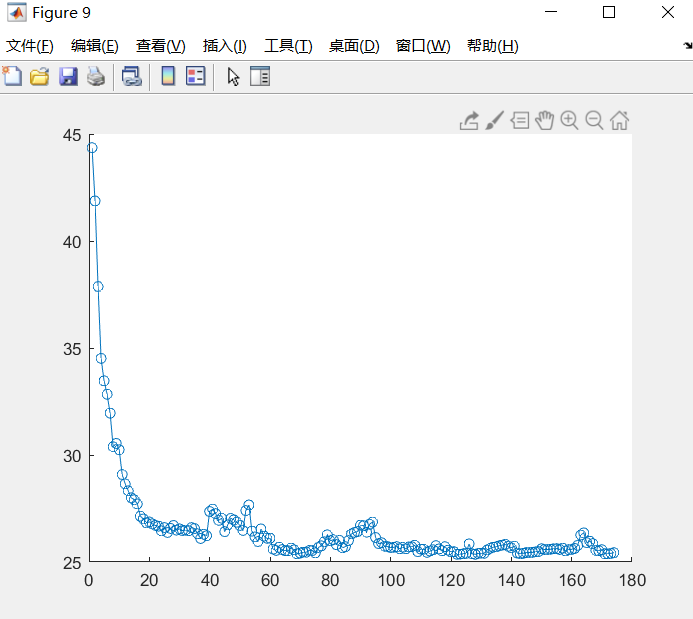
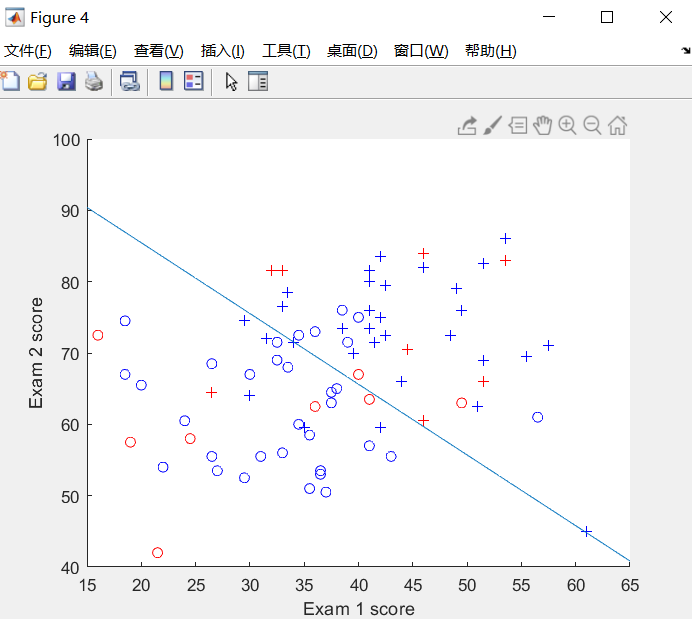
而对于每次训练准确率的计算，对于每一个测试数据，若y=1且点在结果直线之上或y=0且点在结果直线之下标记为正确，其余情况标记为错误。并将正确的点数除以16，即获得训练准确率。最后对于5次训练准确率取平均值获得综合准确率。

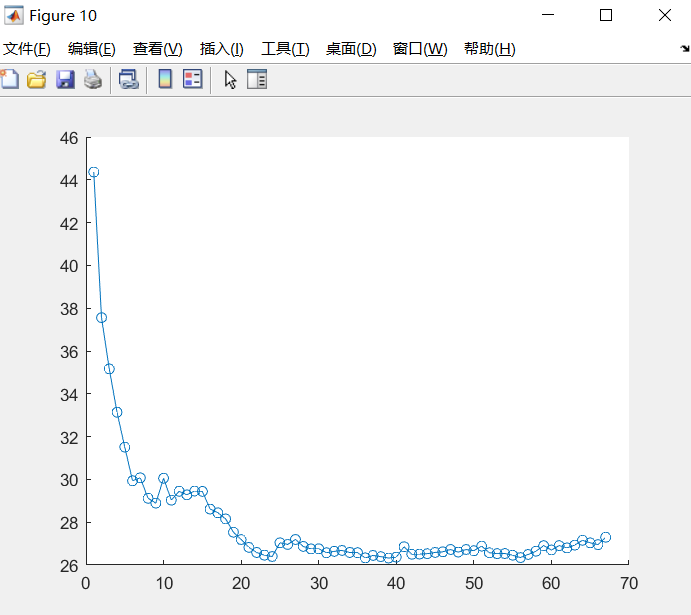
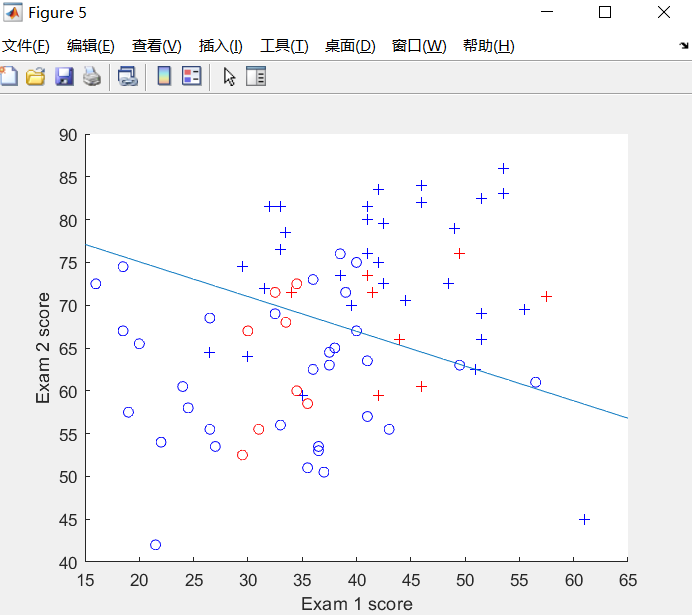
## 2.2 运行结果



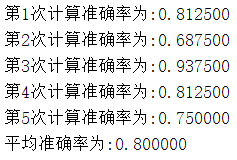








**图2.1**



**图2.2**

**图2.1**是5次交叉验证选用不同的数据段作为训练集和测试集所得到的不同结果，左边为最终结果图，直线则为二分类问题的分界线，蓝色代表训练集，红色代表测试集。而右图是每次训练损失函数的变化曲线。

**图2.2**是每次训练的准确率，以及5次训练的平均准确率。除了第二次计算，其余4次计算都有较高的准确率，而平均准确率也高达80%。可以见得，模型的准确率还是比较好的。

## 2.3 可运行代码

clc;

clear;

xtrainlist=[2:5:80,3:5:80,4:5:80,5:5:80;

1:5:80,3:5:80,4:5:80,5:5:80;

1:5:80,2:5:80,4:5:80,5:5:80;

1:5:80,2:5:80,3:5:80,5:5:80;

1:5:80,2:5:80,3:5:80,4:5:80

];

xtestlist=[1:5:80;2:5:80;3:5:80;4:5:80;5:5:80];

x=load("ex4Data/ex4x.dat");

y=load("ex4Data/ex4y.dat");

xp=x;

yp=y;

[m,n]=size(y);

mean=mean(x);

std=std(x);

for i=1:80

for z=1:2

x(i,z)=(x(i,z)-mean(z))/std(z);

end

end

temp(1:80,1)=1;

x=[x,temp];

accuracy=[];

for p=1:5

a = 0.1;

theta =[0;0;0];

loss = 0;

old\_loss = 0;

temp\_loss=[];

xtrain=x(xtrainlist(p,:),:);

ytrain=y(xtrainlist(p,:),:);

xtest=x(xtestlist(p,:),:);

ytest=y(xtestlist(p,:),:);

xptrain=xp(xtrainlist(p,:),:);

yptrain=yp(xtrainlist(p,:),:);

xptest=xp(xtestlist(p,:),:);

yptest=yp(xtestlist(p,:),:);

for i=1:64

if (ytrain(i) == 1)

loss=loss+log(sigmoid(xtrain(i,:)\*theta));

else

loss=loss+log(1-sigmoid(xtrain(i,:)\*theta));

end

end

while abs(loss-old\_loss) > 0.001

temp =xtrain\*theta;

dew = [0,0,0];

z = round(rand(1,5)\*63)+1;

for i=1:5

dew=dew+(ytrain(z(i))-sigmoid(temp(z(i))))\*xtrain(z(i),:);

end

theta = theta+a\*dew';

old\_loss = loss;

loss = 0;

for i=1:64

if(ytrain(i) == 1)

loss=loss+log(sigmoid(xtrain(i,:)\*theta));

else

loss=loss+log(1-sigmoid(xtrain(i,:)\*theta));

end

end

temp\_loss=[temp\_loss,-old\_loss];

end

sumright=0;

for i=1:16

test=xtest(i,:)\*theta;

if(test>0)

if(ytest(i)==1)

sumright=sumright+1;

end

else

if(ytest(i)==0)

sumright=sumright+1;

end

end

end

accuracy=[accuracy,sumright/16.0];

figure(p);

xlabel('Exam 1 score');

ylabel('Exam 2 score');

for i=1:64

hold on;

if(yptrain(i)==1)

plot(xptrain(i,1),xptrain(i,2),'b+');

else

plot(xptrain(i,1),xptrain(i,2),'bo');

end

end

for i=1:16

hold on;

if(yptest(i)==1)

plot(xptest(i,1),xptest(i,2),'r+');

else

plot(xptest(i,1),xptest(i,2),'ro');

end

end

plot\_x2 =zeros(1,51);

plot\_x1 =15:1:65;

for i=15:65

plot\_x2(i-14) = -(theta(3)+theta(1)\*((i-mean(1))/std(1)))/theta(2);

plot\_x2(i-14) = plot\_x2(i-14)\*std(2)+mean(2);

end

hold on;

plot(plot\_x1,plot\_x2,'-');

figure(p+5);

hold on;

plot(temp\_loss,'-o');

end

ave\_accuracy=0;

for i=1:5

fprintf('第%d次计算准确率为:%f\n',i,accuracy(i));

ave\_accuracy=ave\_accuracy+accuracy(i);

end

fprintf('平均准确率为:%f\n',ave\_accuracy/5);

# 三、基于BP算法的三层前向神经网络求解

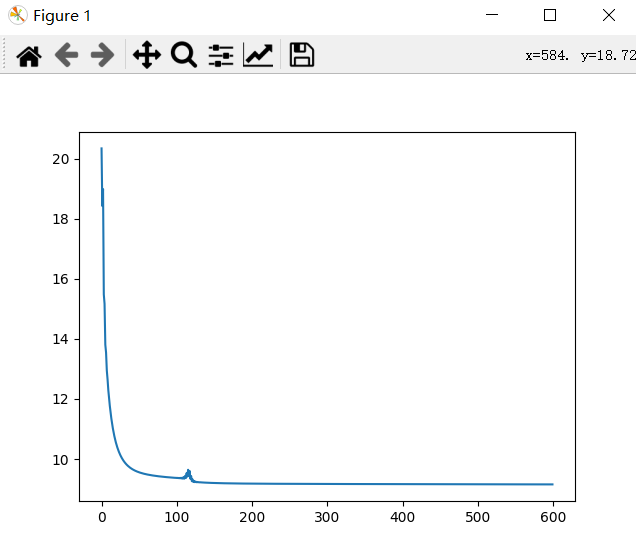
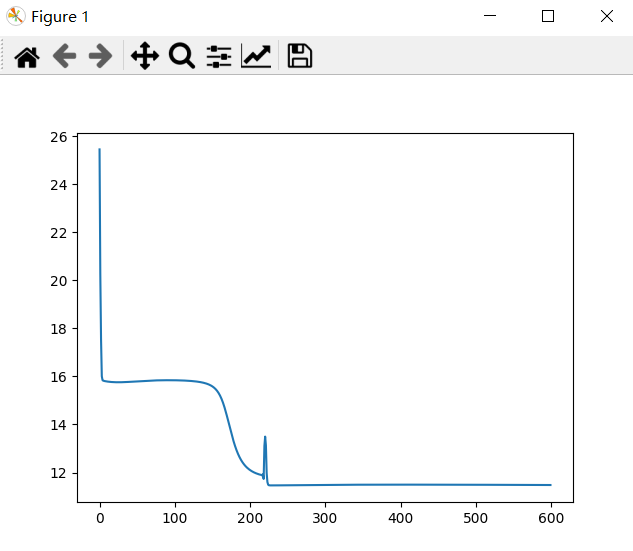
## 3.1 算法思路

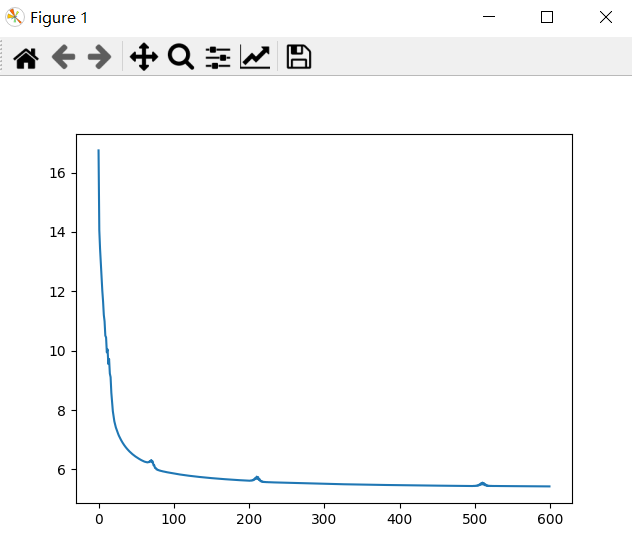
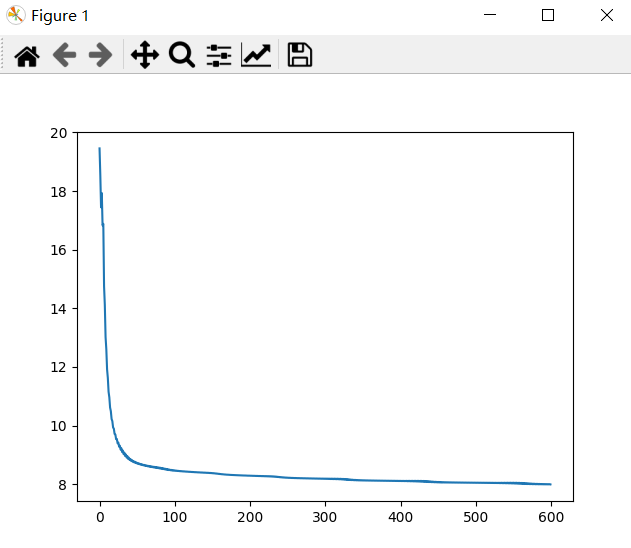
由于三层前向神经网络模型已经不是线性模型，所以我们不能像之前的logistics回归和softmax回归那样给出最终结果直线。但我们仍可以计算每一次的准确率来评估模型。

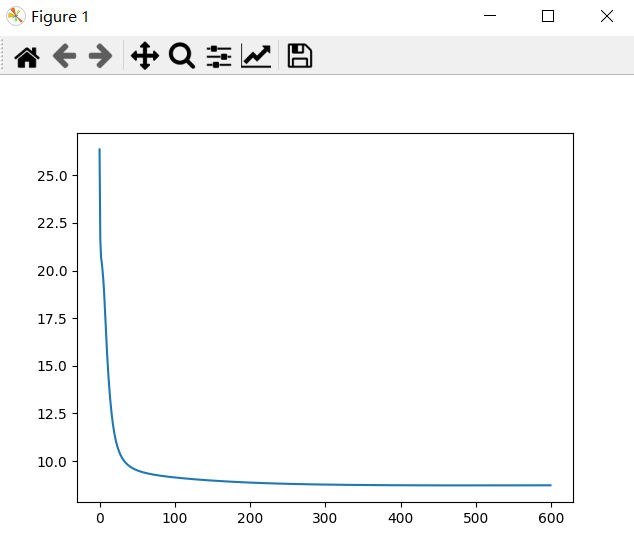
而5倍交叉验证法就是将整个数据集分为不相交的5个部分，然后选取其中的一个部分为测试集，其他部分为训练集，获得测试准确率，然后重复上述步骤，直到每个部分都充当过一次测试集，我们以平均准确率作为模型的最终准确率。本数据集一共有80组数据，也就是说我们每次取其中的16组作为测试集，其余数据作为训练集。

之后，便是代入神经网络的搭建，设置学习率为0.05，步数为600，调用python的numpy库中的random.normal函数，每次训练从正态分布曲线中随机抽取数，作为隐藏层（即w权重和v权重）权重的值。选取sigmoid函数作为激活函数。对于每次迭代，绘制出其损失函数的变化，并打印准确率。

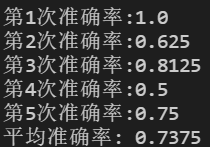
## 3.2 运行结果







**图3.1**



**图3.2**

上图为代码运行结果。**图3.1**是五次迭代的损失函数变化图，我们可以看到经过600次梯度下降计算，其损失函数最终都趋于平缓，可以认为结果收敛。**图3.2**是每次迭代的准确率，以及最终的平均准确率。我们可以看到，除了第二次、第四次迭代的准确率较低，其他几次的迭代率都很高。最终平均准确率为0.7375。

## 3.3 可运行代码

import numpy as np

import matplotlib.pyplot as plt

def sigmoid(x):

    return 1/(np.exp(-x)+1)

data\_x = np.loadtxt("ex4Data/ex4x.dat")

data\_y = np.loadtxt("ex4Data/ex4y.dat")

mean = data\_x.mean(axis=0)

variance = data\_x.std(axis=0)

data\_x = (data\_x-mean)/variance

data\_y = data\_y.reshape(-1, 1)

temp = np.ones(data\_y.size)

data\_x = np.c\_[data\_x, temp]

y = np.zeros([data\_y.size, 2])

for i in range(data\_y.size):

    if data\_y[i] == 0:

        y[i][0] = 1

    if data\_y[i] == 1:

        y[i][1] = 1

learn\_rate = 0.05

data\_sets = list()

for i in range(data\_y.size):

    if i % 8 == 0 and i != 0:

        data\_sets.append(data\_x[i-8:i, :])

    if i == data\_y.size-1:

        data\_sets.append(data\_x[i-7:i+1, :])

positive\_x = data\_x[0:40, :]

positive\_label = y[0:40, :]

negative\_x = data\_x[40:80, :]

negative\_label = y[40:80, :]

total = 0

for j in range(5):

    x\_ = np.r\_[np.delete(positive\_x, range(j\*8, (j+1)\*8), 0), np.delete(negative\_x, range(j\*8, (j+1)\*8), 0)]

    y\_ = np.r\_[np.delete(positive\_label, range(j\*8, (j+1)\*8), 0), np.delete(negative\_label, range(j\*8, (j+1)\*8), 0)]

    test\_x = np.r\_[data\_sets[j], data\_sets[j+5]]

    test\_label = np.r\_[data\_y[j\*8:(j+1)\*8], data\_y[(j+5)\*8:(j+6)\*8]]

    data\_x = np.mat(x\_)

    temp = np.ones(data\_y.size-16)

    weight\_input = np.mat(np.random.normal(size=(data\_x.shape[1], 7)))

    weight\_hidden = np.mat(np.random.normal(size=(8,2)))

    steps = 600

    loss\_values = list()

    for i in range(steps):

        hidden\_input = data\_x\*weight\_input

        hidden\_out = sigmoid(hidden\_input)

        hidden\_out\_ = np.c\_[hidden\_out, temp]

        output\_input = hidden\_out\_\*weight\_hidden

        output = sigmoid(output\_input)

        loss = 0.5\*np.sum(np.multiply(output-y\_,  output-y\_))

        loss\_values.append(loss)

        output\_error = np.multiply(np.multiply(output-y\_, output), 1-output)

        dew\_hidden = hidden\_out\_.T\*output\_error

        output\_error\_ = dew\_hidden[7]

        weight\_hidden\_ = np.delete(weight\_hidden, 7, axis=0)

        hidden\_error = np.multiply(np.multiply(output\_error\_\*weight\_hidden\_.T, hidden\_out), 1-hidden\_out)

        dew\_input = data\_x.T\*hidden\_error

        weight\_hidden = weight\_hidden-learn\_rate\*dew\_hidden

        weight\_input = weight\_input-learn\_rate\*dew\_input

    plt.plot(loss\_values)

    plt.show()

    temp = np.ones(test\_label.size)

    hidden\_input=test\_x\*weight\_input

    hidden\_out=sigmoid(hidden\_input)

    hidden\_out\_ = np.c\_[hidden\_out, temp]

    output\_input = hidden\_out\_\*weight\_hidden

    output =sigmoid(output\_input)

    count = 0

    output = np.array(output)

    print(output)

    for i in range(test\_label.size):

        outs = output[i].ravel()

        outs = outs.tolist()

        if int(test\_label[i]) == outs.index(max(outs)):

            count = count+1

    print("第%d次准确率:%s" %(j+1,count/test\_label.size))

    total = total+count/test\_label.size

print("平均准确率:", total/5)

# 四、基于libsvm的软间隔SVM求解

## 4.1 算法思路

这里我们直接调用python的第三方库libsvm来进行软间隔SVM的求解。libsvm库功能十分强大，只需简答改变参数，就可以更改模型内部的SVM类型、核函数类型以及参数C的值。下面对本实验需要用到的参数设置进行介绍：

**-s SVM的类型(svm\_type)**

0 -- C-SVC(默认) 使用惩罚因子(Cost)的处理噪声的多分类器

1 -- nu-SVC(多分类器) 按照错误样本比例处理噪声的多分类器

2 -- one-class SVM 一类支持向量机

3 -- epsilon-SVR(回归) epsilon支持向量回归

4 -- nu-SVR(回归)

**-t 核函数类型(kernel\_type)**

0 -- linear(线性核)

1 -- polynomial(多项式核)

2 -- radial basis function(RBF,径向基核/高斯核)

3 -- sigmoid(S型核)

4 -- precomputed kernel(用户自定义核)

**-c 调整C-SVC, epsilon-SVR 和 nu-SVR中的Cost参数，默认为1**

**-g 调整核函数的gamma参数**

比如，设置：-s 0 -c 1 -t 3 -g 0.01，即说明SVM类型设置为C-SVC，核函数类型为高斯函数，参数C值为1，参数gamma为0.01。

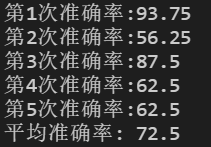
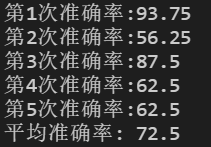
而5倍交叉验证方法与问题一和问题二相同，在此不再赘述。

## 4.2 运行结果

这里我们均设置SVM类型为C-SVC，gamma参数为0.01，仅改变核函数类型（-t的值）以及参数C的值（-c的值）

**1.核函数为线性核（-t 0）**

**（1）C=10 （2）C=1 （3）C=0.1**



**（4）C=0.01 （5）C=0.001 （6）C=0.0001**



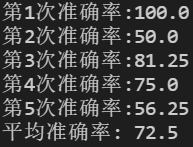
从上图我们不难看出在线性核函数情况下，改变参数C的值，其平均准确率在一定范围内变化不大甚至没有变化，但当C=0.0001是，准确率骤降，平均准确率仅有18.75。

**2.核函数为多项式核（-t 1）**

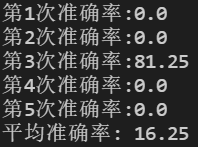
在该核函数情况下，无论设置参数C为多少，均运行不出结果，会出现无限迭代的情况。

**3.核函数为RBF核（-t 2）**

**（1）C=10 （2）C=1 （3）C=0.1**

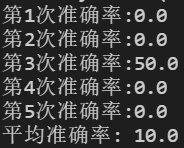
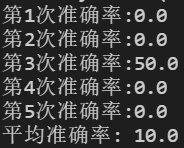
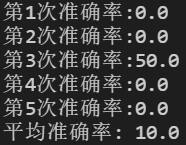
**（4）C=0.01 （5）C=0.001 （6）C=0.0001**

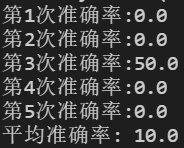
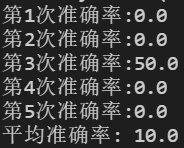
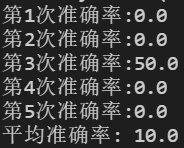
从上图我们可以看出当核函数为RBF核时，随着C的改变平均准确率的下降十分明显，但当C=0.01之后，再减小C的值准确率不再变化，一直为16.25%。

**4.核函数为sigmoid核（-t 3）**

**（1）C=10 （2）C=1 （3）C=0.1**



**（4）C=0.01 （5）C=0.001 （6）C=0.0001**



从上图我们不难看出，无论C如何改变，sigmoid核函数情况下，其平均准确率不变且准确率都十分之低，只有10%。

## 4.3 可运行代码

from libsvm.svmutil import svm\_predict, svm\_save\_model, svm\_train

import numpy as np

import pandas as pd

from libsvm.svmutil import \*

x = np.loadtxt("ex4Data/ex4x.dat")

y = np.loadtxt("ex4Data/ex4y.dat")

accuracy=[0,0,0,0,0]

ave\_accuracy=0

for j in range(5):

    xtrain=np.delete(x,range(j\*16,(j+1)\*16),0)

    ytrain=np.delete(y,range(j\*16,(j+1)\*16),0)

    xtest=x[j\*16:(j+1)\*16]

    ytest=y[j\*16:(j+1)\*16]

    model = svm\_train(ytrain, xtrain, '-s 0 -c 1 -t 0')

    label\_predict, acc\_predict, val\_predict = svm\_predict(ytest, xtest, model)

    accuracy[j]=acc\_predict[0]

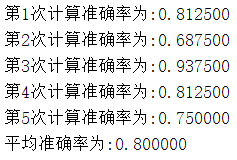
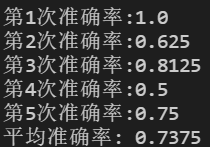
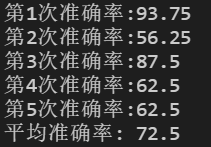
    ave\_accuracy+=acc\_predict[0]

for j in range(5):

    print("第%d次准确率:%s" %(j+1,accuracy[j]))

print("平均准确率:", ave\_accuracy/5)

# 五、三种不同模型结果的比较

以上三张图从左到右分别是Logistic回归模型、三层前向神经网络模型、软间隔SVM模型（取最大准确率）的准确率结果图。从中我们不难看出，准确率最高的还是Logistic模型，其次是神经网络，平均准确率最低的是Libsvm模型。

但值得一提的是，从4.2的运行结果来看，SVM模型的准确率受核函数、参数C的影响十分之大，虽然实验中所得到的平均准确率没有其他两个模型高。但是如果选择适当的核函数和参数，理论上应该可以达到很高的准确率。