**模式识别报告L5**

自卓2201 杨欣怡 U202215067

1，编程实现Logistic regression算法。

1. class LogisticRegression:
2. def \_\_init\_\_(self, lr=0.02, epoch=50):
3. self.w = np.zeros((3,1))
4. self.lr = lr
5. self.epoch = epoch
6. def \_\_call\_\_(self, x, y):
7. x = x.reshape(-1,2)
8. y = y.reshape(-1,1)
9. expand\_axis = np.ones((x.shape[0],1))
10. x = np.concatenate((expand\_axis, x), axis=-1)
12. losses = []
13. for i in range(self.epoch):
14. yhat = self.sigmoid(np.matmul(x, self.w))
15. loss = self.cross\_entropy(x, y, yhat)
16. losses.append(loss)
17. grad = self.\_calculate\_grad(x, y, yhat)
19. self.w -= self.lr \* grad
20. print('epoch: %i/%i loss: %.2f'%(i+1, self.epoch, loss))
21. return yhat, losses
22. def \_calculate\_grad(self, x, y, yhat):
23. batchsize = y.shape[0]
24. s = np.matmul(x, self.w)
25. grad = (self.sigmoid(-y\*s) - 1) \* y \* x
26. grad = np.sum(grad, axis=0) / batchsize
27. return grad.reshape(3,1)
28. def cross\_entropy(self,x, y, yhat):
29. num = y.shape[0]
30. s = np.matmul(x, self.w)
31. loss = (1/num)\*np.sum(np.log(1+np.exp(-s\*y)))
32. return loss
33. def sigmoid(self, x):
34. return 1 / (1 + np.exp(-x))
35. def eval(self, x, y):
36. test\_num = x.shape[0]
37. expand\_axis = np.ones((x.shape[0],1))
38. x = np.concatenate((expand\_axis, x), axis=-1)
39. yhat = np.matmul(x, self.w)
40. yhat = np.sign(self.sigmoid(yhat) - 0.5)
41. assert(len(yhat) == len(y))
42. correct\_num = len(np.where(yhat == y)[0])
43. print('accuracy: %.2f'%(correct\_num/test\_num))

2，（a）产生两个都具有200个二维向量的数据集和。数据集的样本来自均值向量协方差矩阵的正态分布，属于“+1”类，数据集的样本来自均值向量、协方差矩阵的正态分布，属于“-1”类，其中****是一个2\*2的单位矩阵。产生的数据中80%用于训练，20%用于测试。

（b）在训练集上利用Logistic regression算法得到分类面。

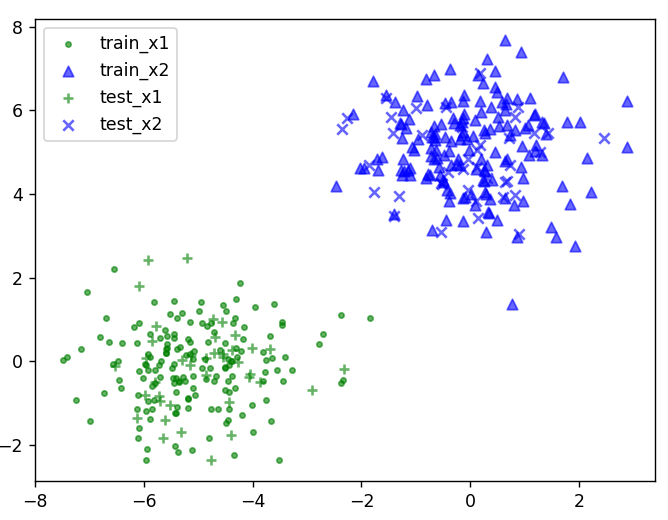
（c）利用得到的分类面对测试集样本进行分类，并给出每个样本属于该类别的概率值。

（d）画出数据集和分类面。

（e）画出损失函数随epoch增加的变化曲线。

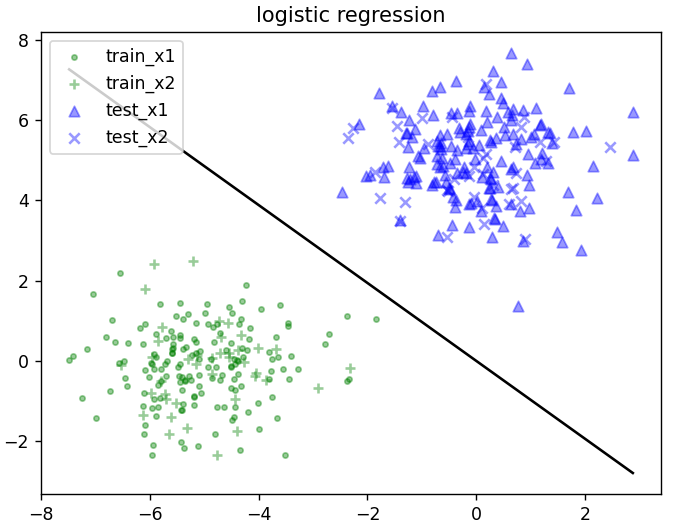
（f）改变算法中的各类超参数、样本数量、样本分布等，对于梯度下降法还要改变不同的学习率以及不同的batch size和不同epoch次数，讨论实验结果。

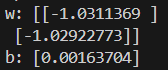
1. class Dataset:
2. def \_\_init\_\_(self, means1=[-5, 0], means2=[0, 5]):
3. self.\_rand\_sample(means1, means2)
4. def \_rand\_sample(self, means1, means2):
5. means1 = np.array(means1)
6. means2 = np.array(means2)
7. covar = np.array([1, 0, 0, 1]).reshape(2,2)
8. x1 = np.random.multivariate\_normal(means1, covar, size=200)
9. x2 = np.random.multivariate\_normal(means2, covar, size=200)
10. y1 = np.ones((200, 1))
11. y2 = np.ones((200, 1)) \* -1
13. self.\_split(x1, y1, x2, y2)
14. def \_split(self, x1, y1, x2, y2):
15. num1 = x1.shape[0]
16. train\_num1 = int(num1 \* 0.8)
17. num2 = x2.shape[0]
18. train\_num2 = int(num2 \* 0.8)
19. self.x\_train = np.concatenate((x1[:train\_num1],x2[:train\_num2]),axis=0)
20. self.y\_train = np.concatenate((y1[:train\_num1],y2[:train\_num2]),axis=0)
21. self.x\_test = np.concatenate((x1[train\_num1:],x2[train\_num2:]),axis=0)
22. self.y\_test = np.concatenate((y1[train\_num1:],y2[train\_num2:]),axis=0)



生成分类面：

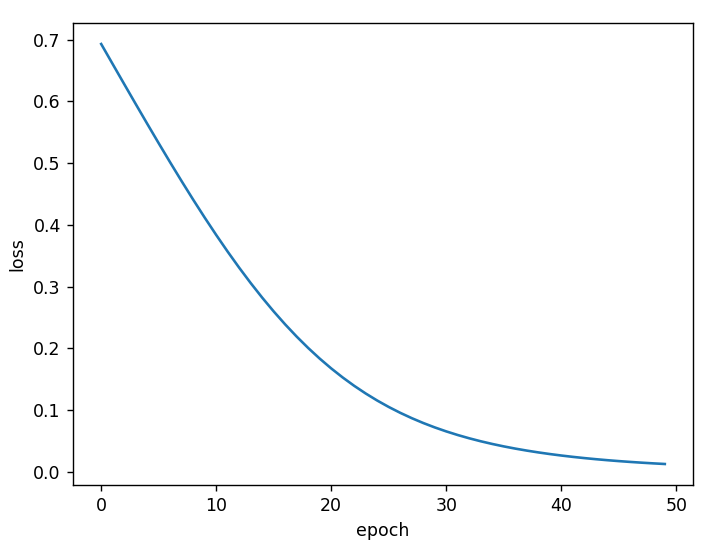
1. epoch = 50
2. lr = 0.01
3. model = LogisticRegression(lr=lr, epoch=epoch)
4. yhat, loss = model(x\_train, y\_train)
5. w, b = model.w[1:], model.w[0]
6. print('w: {}'.format(w))
7. print('b: {}'.format(b))
8. c1 = plt.scatter(x\_train[:160,0], x\_train[:160,1], alpha=0.4, marker='.', c='red')
9. c2 = plt.scatter(x\_train[160:,0], x\_train[160:,1], alpha=0.4, marker='^', c='blue')
10. c3 = plt.scatter(x\_test[:40,0], x\_test[:40,1], alpha=0.4, marker='+', c='red')
11. c4 = plt.scatter(x\_test[40:,0], x\_test[40:,1], alpha=0.4, marker='x', c='blue')
12. xmax = np.max(x\_train[:,0])
13. xmin = np.min(x\_train[:,0])
14. point1 = [xmin,xmax]
15. point2 = [-(w[0]\*xmin+b)/w[1],-(w[0]\*xmax+b)/w[1]]
16. plt.plot(point1,point2,c='orange')
17. plt.legend(handles=[c1, c3, c2, c4],labels=['1\_train','1\_test','2\_train','2\_test'],loc='best')
18. plt.title('logistic regression')
19. plt.show()





损失函数随epoch增加的变换曲线：

1. epochs = np.arange(0,epoch)
2. loss = np.array(loss)
3. plt.plot(epochs,loss)
4. plt.xlabel('epoch')
5. plt.ylabel('loss')
6. plt.show()



epoch: 1/50 loss: 0.69

epoch: 2/50 loss: 0.66

epoch: 3/50 loss: 0.63

epoch: 4/50 loss: 0.60

epoch: 5/50 loss: 0.56

epoch: 6/50 loss: 0.53

epoch: 7/50 loss: 0.50

epoch: 8/50 loss: 0.47

epoch: 9/50 loss: 0.44

epoch: 10/50 loss: 0.41

epoch: 11/50 loss: 0.38

epoch: 12/50 loss: 0.36

epoch: 13/50 loss: 0.33

epoch: 14/50 loss: 0.31

epoch: 15/50 loss: 0.28

epoch: 16/50 loss: 0.26

epoch: 17/50 loss: 0.24

epoch: 18/50 loss: 0.22

epoch: 19/50 loss: 0.20

epoch: 20/50 loss: 0.18

epoch: 21/50 loss: 0.17

epoch: 22/50 loss: 0.15

epoch: 23/50 loss: 0.14

epoch: 24/50 loss: 0.13

epoch: 25/50 loss: 0.12

epoch: 26/50 loss: 0.10

epoch: 27/50 loss: 0.10

epoch: 28/50 loss: 0.09

epoch: 29/50 loss: 0.08

epoch: 30/50 loss: 0.07

epoch: 31/50 loss: 0.07

epoch: 32/50 loss: 0.06

epoch: 33/50 loss: 0.05

epoch: 34/50 loss: 0.05

epoch: 35/50 loss: 0.05

epoch: 36/50 loss: 0.04

epoch: 37/50 loss: 0.04

epoch: 38/50 loss: 0.03

epoch: 39/50 loss: 0.03

epoch: 40/50 loss: 0.03

epoch: 41/50 loss: 0.03

epoch: 42/50 loss: 0.02

epoch: 43/50 loss: 0.02

epoch: 44/50 loss: 0.02

epoch: 45/50 loss: 0.02

epoch: 46/50 loss: 0.02

epoch: 47/50 loss: 0.02

epoch: 48/50 loss: 0.01

epoch: 49/50 loss: 0.01

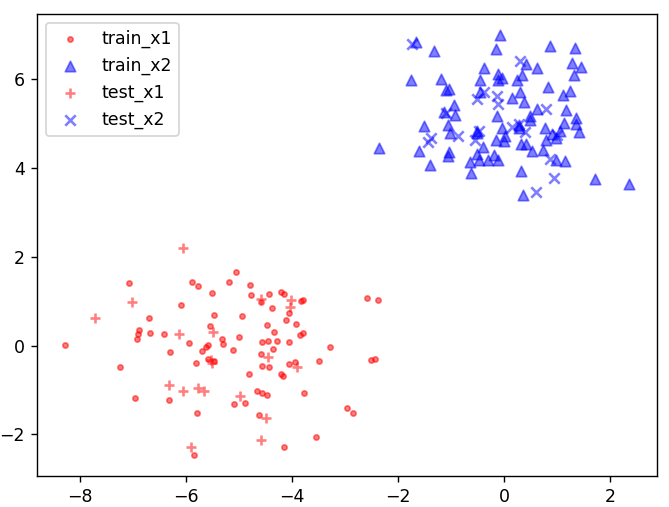
epoch: 50/50 loss: 0.01

分类准确率：

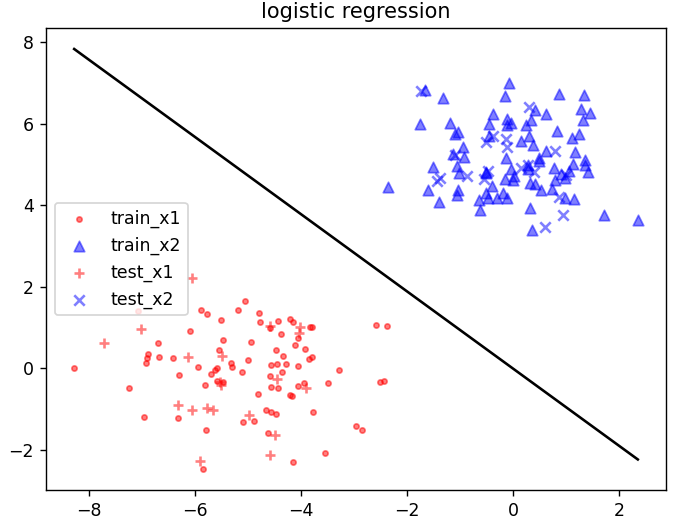


① 若将样本数量改为各具有100个二维向量的样本集x1和x2，均值向量仍为：

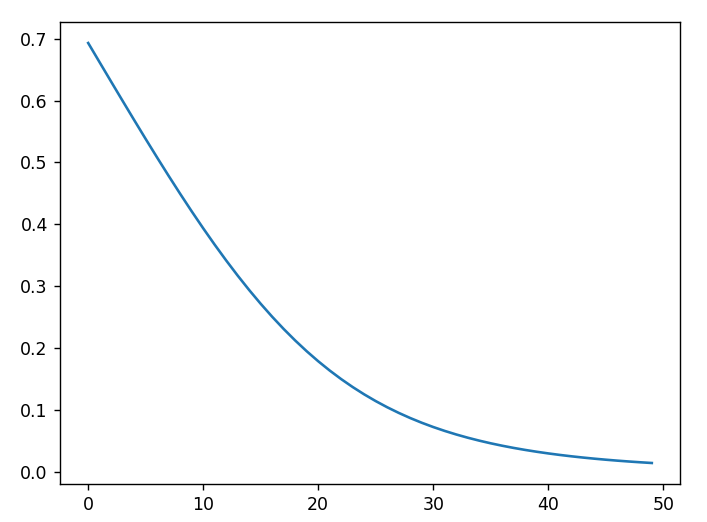
生成数据集：



生成分类面：



损失函数随epoch增加的变化曲线：

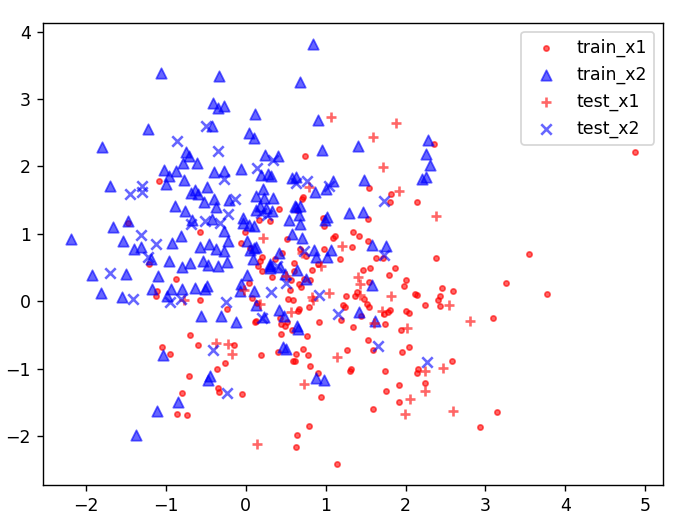




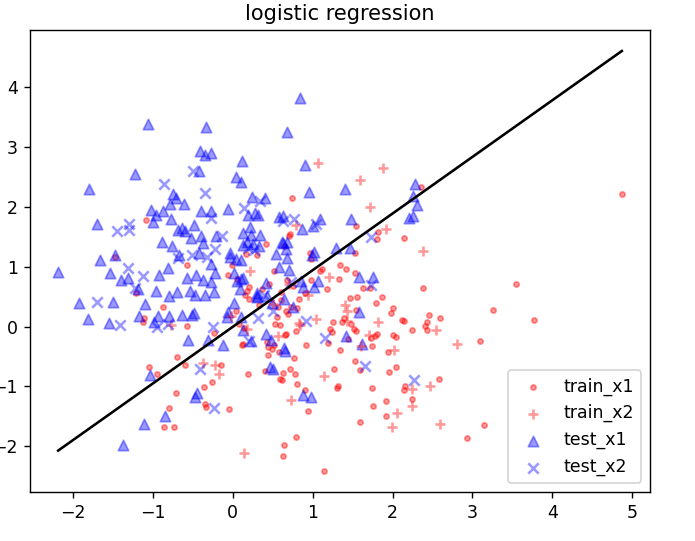
② 若将上述样本分布的均值向量分别改为和：

|  |
| --- |
|  |
|  |  |

生成数据集：

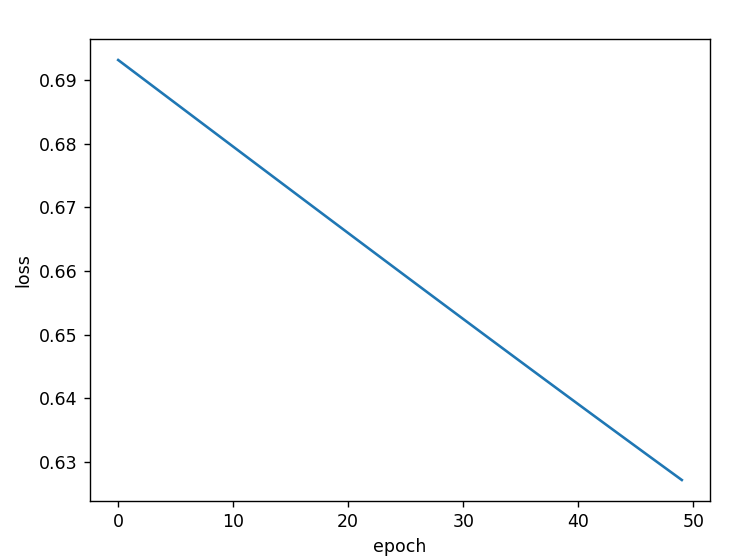


生成分类面：





损失函数随epoch增加的变化曲线：



分类准确率



结果讨论：

1. 改变样本数量：发现改变样本数量并不会影响分类结果的准确程度，在迭代次数一定的情况下，样本数量越少，得到的分类面结果越好。
2. 改变样本分布：改变正负样本的均值发现，当正负样本均值点较为接近时，可能无法将数据有效的分开；得到的分类面结果较差