机器学习上机报告

第二章

实验内容：

C2-1 构建一个二分类问题：逻辑与(AND)的一般问题。

（1）生成数据：正样本样本服从高斯分布，均值为[1,1]，协方差矩阵为单位矩阵；负样本服从三个高斯分布的混合高斯分布，这三个高斯分布的均值分别为[0,0], [0,1], [1,0]，每个高斯分布的协方差矩阵均为(sigma*单位矩阵)。

（2）学习：设 sigma=0.01，请依上面的分布生成正负样本各 300 个，运用 perceptron learning algorithm 从数据中学习出一个 perceptron，实现对正负样本的二分类。

(3) 实验与讨论：请通过编程实验，讨论如下问题：a. 学习算法的收敛性与哪些因素存在怎样的关系？b. 讨论当 sigma 取值不断变大（如取值从 0.01-1）情况下，学习算法的收敛性问题，由此可以得出怎样的结论？

实验步骤：

1、利用 numpy 生成对应数据

```
mu = np.array([[1, 1]])
Sigma = np.array([[1, 0], [0, 1]])
Sigma2 = sigma*Sigma
positive = np.dot(np.random.randn(300, 2), Sigma2) + mu
avg = np.array([[0, 0]])
print(np.dot(np.random.randn(100, 2), Sigma2) + avg)
negetive = list(np.dot(np.random.randn(100, 2), Sigma2) + avg)
avg = np.array([[0, 1]])
negetive.extend(list(np.dot(np.random.randn(100, 2), Sigma2) + avg))
avg = np.array([[1, 0]])
negetive.extend(list(np.dot(np.random.randn(100, 2), Sigma2) + avg))
negetive = np.array(negetive)
input_vecs = np.concatenate((positive,negetive),axis=0)
# 期望的输出列表
labels = np.zeros(600,dtype=int)
labels[:300] = np.ones(300,dtype=int)
labels[300:] = np.zeros(300,dtype=int)
```

2、生成 label

```
label = [1 for i in range(300)]
label.extend([0 for i in range(300)])
print(label)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

3、初始化 preception

```
class Perceptron(object):
    def __init__(self, input_num, activator):
        '''
        初始化感知器，设置输入参数的个数，以及激活函数。
        激活函数的类型为float -> float
        '''
        self.activator = activator
        # 权重向量初始化为0
        self.weights = [0.0 for _ in range(input_num)]
        # 偏置项初始化为0
        self.bias = 0.0
        self.learningCurve=[]
```

4、训练 preception 以及更新权重

```
def train(self, input_vecs, labels, iteration, rate):
    '''
    输入训练数据：一组向量、与每个向量对应的label；以及训练轮数、学习率
    '''
    for i in range(iteration):
        self._one_iteration(input_vecs, labels, rate)

def _one_iteration(self, input_vecs, labels, rate):
    '''
    一次迭代，把所有的训练数据过一遍
    '''
    # 把输入和输出打包在一起，成为样本的列表[(input_vec, label), ...]
    # 而每个训练样本是(input_vec, label)
    samples = zip(input_vecs, labels)
    # 对每个样本，按照感知器规则更新权重
    for (input_vec, label) in samples:
        # 计算感知器在当前权重下的输出
        output = self.predict(input_vec)
        # 更新权重
        self._update_weights(input_vec, output, label, rate)

def _update_weights(self, input_vec, output, label, rate):
    '''
    按照感知器规则更新权重
    '''
    # 把input_vec[x1,x2,x3,...]和weights[w1,w2,w3,...]打包在一起
    # 变成[(x1,w1),(x2,w2),(x3,w3),...]
    # 然后利用感知器规则更新权重
    delta = label - output
    self.weights = list(map(lambda tp: tp[1] + rate * delta * tp[0], zip(input_vec, self.weights)))

    # 更新bias
    self.bias += rate * delta

    print("_update_weights() -------------")
    print("label - output = delta:", label, output, delta)
    print("weights ", self.weights)
    print("bias", self.bias)
```

5、画图函数

```
def drawLearningCurve(self,sigma,fname='learningCurve.png'):
    plt.plot(range(len(self.learningCurve)),self.learningCurve)
    plt.xlim(0, len(self.learningCurve))
    plt.xlabel('epoch')
    plt.ylabel('Cost')
    plt.title('LearningCurve (sigma={})'.format(sigma))
    plt.savefig(fname)
    #plt.show()
    plt.close()
```

6、predict 函数

```
def predict(self, input_vec):
    '''
    输入向量，输出感知器的计算结果
    '''
    # 把input_vec[x1,x2,x3...]和weights[w1,w2,w3,...]打包在一起
    # 变成[(x1,w1),(x2,w2),(x3,w3),...]
    # 然后利用map函数计算[x1*w1, x2*w2, x3*w3]
    # 最后利用reduce求和

    # list1 = list(self.weights)
    # print ("predict self.weights:", list1)

    return self.activator(
        reduce(lambda a, b: a + b,
               list(map(lambda tp: tp[0] * tp[1],
                        zip(input_vec, self.weights)))
               , 0.0) + self.bias)
```

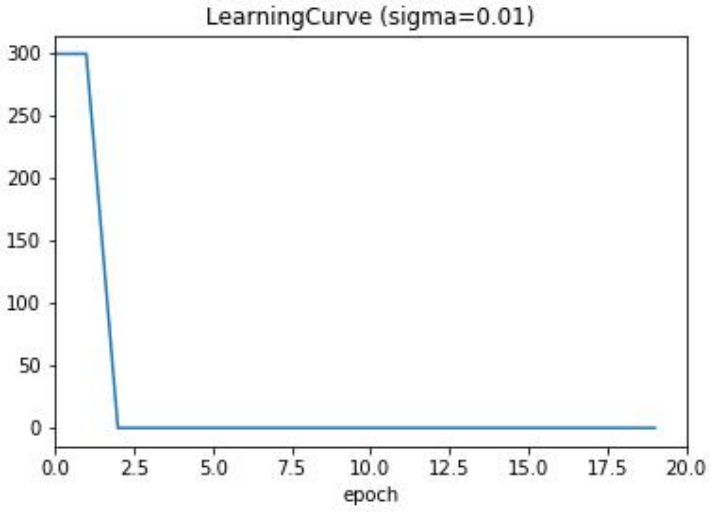
7、定义 0，1 激活函数

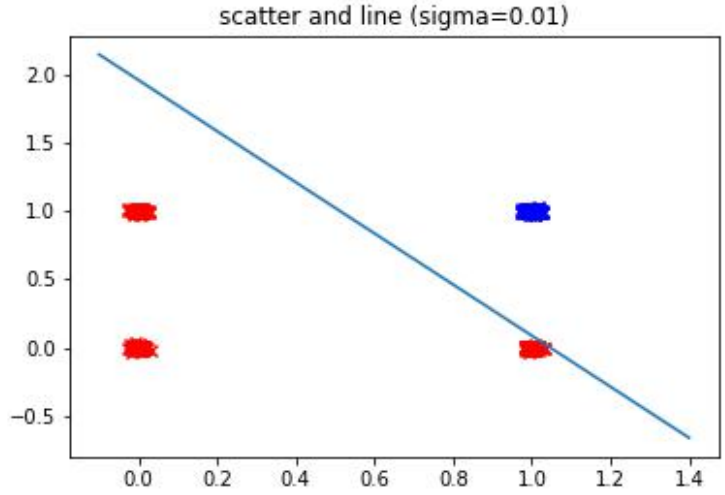
```
def f(x):
    '''
    定义激活函数f
    '''
    return 1 if x > 0 else 0
```

8、训练感知器模型迭代 10 次

```
input_vecs, labels = get_training_dataset(sigma=0.01)
and_perceptron = train_perceptron(input_vecs, labels, niter=10,rate=0.1)
# Learning Curve
and_perceptron.drawLearningCurve(sigma=0.01)
# scatter
drawscatter(input_vecs, labels, and_perceptron,sigma=0.01)
```

实验结果





实验与讨论：

实验与讨论：请通过编程实验，讨论如下问题：a. 学习算法的收敛性与哪些因素存在怎样的关系？b. 讨论当 sigma 取值不断变大（如取值从 0.01-1）情况下，学习算法的收敛性问题，由此可以得出怎样的结论

a）收敛和数据的 sigma 大小和学习率以及迭代次数有关

b）sigma 越大收敛越差

第三章

实验内容：

C3-1

(i) Generate 500 data (x,y), where y=x+n, n is Gaussian distributed with the mean of zero and standard deviation of delt. Please use linear regression learning algorithm for estimating y from input x;

(ii) Do the same for x=y+n but still for estimating y from input x;

(iii) Make a comparison on the regression curves with x as the input and y as the output of the regressor obtained from (i) and (ii) respectively .

实验步骤：

1、生成随机 500 个 x 和 500 个满足条件的 n，以此来生成 y

生成随机 500 个 y 和 500 个满足条件的 n，以此来生成 y

```
def get_training_dataset_x():
    x = np.linspace(-5,5,500,endpoint=False)
    y = x + np.random.randn(500)
    return x.reshape(-1,1), y
def get_training_dataset_y():
    y = np.linspace(-5,5,500,endpoint=False)
    # 期望的输出列表，注意要与输入一一对应
    x = (y + np.random.randn(500))
    return x.reshape(-1,1), y
```

2、初始化线性回归模型

```
def __init__(self, input_num):
    '''
    初始化感知器，设置输入参数的个数，以及激活函数。
    激活函数的类型为double -> double
    '''
    self.activator = lambda x:x
    # 权重向量初始化为[0,1)
    #self.weights = [np.random.random() for _ in range(input_num)]
    # 偏置项初始化为[0,1)
    #self.bias = np.random.random()
    self.weights = [0 for _ in range(input_num)]
    self.bias = 0
    self.learningCurve=[]
```

3、模型训练函数

```
def train(self, xs, ys, iteration, rate):
    self.learningCurve=[]
    for i in range(iteration):
        error = self._one_iteration(xs, ys, rate)
        self.learningCurve.append(error)


def _one_iteration(self, xs, ys, rate):
    predicts = np.array(list(map(self.predict,xs)))
    error = sum((ys-predicts)**2)/2/len(xs)
    gradient = np.dot((ys-predicts),xs)/len(xs)

    self.weights = self.weights + rate*gradient
    # 更新bias
    self.bias += rate * sum(ys-predicts)/len(xs)
    return error
```

4、训练过程中的权重更新函数

```
def _update_weights(self, x, output, y, rate):
    '''
    按照感知器规则更新权重
    '''
    delta = y - output
    #self.weights =self.weights + rate * delta * x
    self.weights = list(map( lambda wx:wx[0]+rate*delta*wx[1],
                             zip(self.weights, x) ))
    # 更新bias
    self.bias += rate * delta
```

5、预测 predict 函数

```
def predict(self, x):

    return self.activator(
        sum(list(map(lambda wx:wx[0]*wx[1],
                     zip(self.weights, x)
                     ))) + self.bias)
```

6、画图函数

```
def _update_weights(self, x, output, y, rate):
    '''
    按照感知器规则更新权重
    '''
    delta = y - output
    #self.weights =self.weights + rate * delta * x
    self.weights = list(map( lambda wx:wx[0]+rate*delta*wx[1],
                             zip(self.weights, x) ))
    # 更新bias
    self.bias += rate * delta
```

7、创建新型回归拟合器，输入参数个数为 1

```
def train_LinearRegression(x, y):
    p = LinearRegression(1)
    # 训练，迭代10轮，学习速率为0.1
    p.train(x, y, 10, 0.1)
    #返回训练好的
    return p
```

8、定义分类直线画图函数

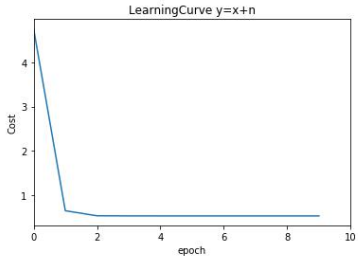
```
def train_LinearRegression(x, y):
    p = LinearRegression(1)
    # 训练，迭代10轮，学习速率为0.1
    p.train(x, y, 10, 0.1)
    #返回训练好的
    return p
```

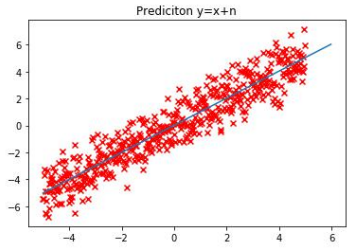
9、主函数

训练 x=y+n 和 y=x+n 打印对应权重

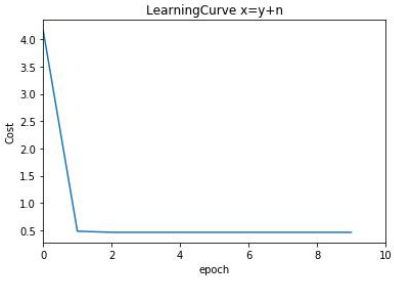
```
x, y = get_training_dataset_x()
# 训练
lr = train_LinearRegression(x,y)
# 打印训练获得的权重
lr.drawLearningCurve('LearningCurve y=x+n')
showPrediction(x,y,lr,fname='Prediciton y=x+n')
print(lr.learningCurve)
x, y = get_training_dataset_y()
# 训练
lr = train_LinearRegression(x,y)
# 打印训练获得的权重
lr.drawLearningCurve('LearningCurve x=y+n')
showPrediction(x,y,lr,fname='Prediciton x=y+n')
print(lr.learningCurve)
```

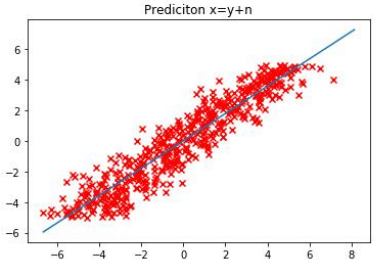
实验结果:






```
[4.760680196368902, 0.6516736510608252, 0.5374979353875763, 0.5342881019589889, 0.534167689394543, 0.53413902496228
84, 0.5341177195741417, 0.5341005152876699, 0.5340865812490971, 0.5340752946846395]
```






```
[4.166699999999997, 0.4847946057258083, 0.4641078634157941, 0.46397900810494463, 0.46396798513528315, 0.46395958048
46019, 0.46395277555206477, 0.46394726348588416, 0.4639427986416187, 0.4639391820604558 7]
```

第四章

实验内容：

(regression problem) train a neural network, to predict the burned area of the forest, using the data from http://archive.ics.uci.edu/ml/datasets/Forest+Fires

实验步骤：

1、分析数据格式：

X,Y,month,day,FFMC,DMC,DC,ISI,temp,RH,wind,rain,area,其中 month 和 day 为字符串

需要把他们转为 float 类型，并且把最后一项 area 设为 train_y

```
data= pd.read_csv("forestfires.csv")
nums = 1000
print(data)
train_y = data.values[:,12:]
train_x = data.values[:,:12]
month = list(calendar.month_abbr)
month = [x.lower() for x in month]
week = ['','mon','tue','wed','thu','fri','sat','sun']
print(week)
print(month)
for num in range(train_x.shape[0]):
    train_x[num][2:3] = month.index(train_x[num][2:3])
for num in range(train_x.shape[0]):
    train_x[num][3:4] = week.index(train_x[num][3:4])
```

2、对数据进行归一化

```
train_x = normalize(train_x)
train_y = normalize(train_y)
```

3、利用 pytorch 初始化神经网络类

```
class Net(torch.nn.Module):  # 继承torch.nn.Module类
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()  # 获得Net类的超类（父类）的构造方法
        # 定义神经网络的每层结构形式
        # 各个层的信息都是Net类对象的属性
        self.hidden = torch.nn.Linear(n_feature, n_hidden)  # 隐藏层线性输出
        self.predict = torch.nn.Linear(n_hidden, n_output)  # 输出层线性输出

    # 将各层的神经元搭建成完整的神经网络的前向通路
    def forward(self, x):
        x = F.relu(self.hidden(x))  # 对隐藏层的输出进行relu激活
        x = self.predict(x)
        return x
```

4、初始化 net 类为 12 输入，12 层，1 输出

```
net = Net(12, 12, 1)
print(net)  # 打印输出net的结构
```

5、给网络喂入参数

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.001)  # 传入网络参数和学习率
loss_function = torch.nn.MSELoss()  # 最小均方误差
```

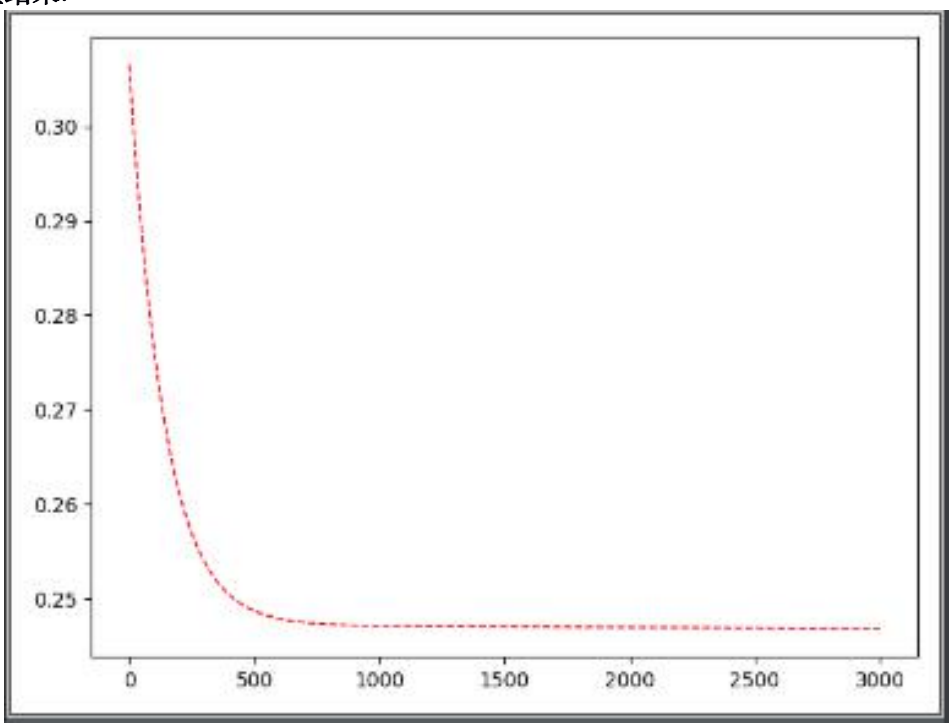
6、训练 3000 次

```
for t in range(30000):
    prediction = net(x)  # 把数据x喂给net，输出预测值
    loss = loss_function(prediction, y)  # 计算两者的误差，要注意两个参数的顺序
    optimizer.zero_grad()  # 清空上一步的更新参数值
    loss.backward()  # 误差反相传播，计算新的更新参数值
    optimizer.step()  # 将计算得到的更新值赋给net.parameters()

    # 可视化训练过程
    if (t + 1) % 10 == 0:
        print(loss.item())
```

7、可视化训练结果

```
x1 = [i for i in range(3000)]
plt.plot(x1, y1, color='red', linewidth=1.0, linestyle='--')
# 显示图表
plt.show()
```

实验结果:



第五章

实验内容：

C5-1 构建一个类别不平衡的二分类问题。

（4）生成数据：正样本样本服从高斯分布，均值为[2,3]，协方差矩阵为单位矩阵；负样本服从高斯分布，均值为[5,6]，协方差矩阵为单位矩阵。

（5）学习：请依上面的分布生成正样本 200 个，负样本 800 个，将其划分为包含 60%样本的训练集、20%样本的验证集和 20%样本的测试集，通过分别构建两个不同的 MLP 模型实现对正负样本的二分类。其中第一个 MLP 模型含有一个隐层，第二个 MLP 模型含有两个隐层。

实验与讨论：请通过编程实验，讨论如下问题：a. 若要求 MLP 模型对于正例样本的预测有更高的查准率和查全率，请考虑在模型选择中采用哪种性能衡量指标；b.通过绘制学习曲线，分析模型的偏差与方差；c.通过绘制 ROC 曲线，比较两个 MLP 模型。

实验步骤：

1、生成数据：

```
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
def get_training_dataset(sigma=0.01):
    mu = np.array([[2, 3]])
    Sigma = np.array([[1, 0], [0, 1]])
    Sigma2 = sigma*Sigma
    positive = np.dot(np.random.randn(200, 2), Sigma2) + mu
    avg = np.array([[5, 6]])
    print(np.dot(np.random.randn(800, 2), Sigma2) + avg)
    negetive = list(np.dot(np.random.randn(800, 2), Sigma2) + avg)
    input_vecs = np.concatenate((positive,negetive),axis=0)
    # 期望的输出列表
    labels = np.zeros(1000,dtype=int)
    labels[:200] = np.ones(200,dtype=int)
    labels[200:] = np.zeros(800,dtype=int)
    return input_vecs, labels
```

2、划分数据集合为验证集，训练集，测试集，设定一层单隐层

```
def sepratedata(x,y):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
    x_test,x_val,y_test,y_val, = train_test_split(x_test, y_test, test_size=0.5)
    return x_train, x_test,x_val,y_train, y_test,y_val
x,y = get_training_dataset()
x0,x1,x2,y0,y1,y2 = sepratedata(x,y)
x0, y0 = torch.tensor(x0.astype('float32')), torch.tensor(y0.astype('float32'))
x1, y1 = torch.tensor(x1.astype('float32')), torch.tensor(y1.astype('float32'))
x2, y2 = torch.tensor(x2.astype('float32')), torch.tensor(y2.astype('float32'))

num_inputs, num_hiddens, num_outputs = 2, 1, 1
```

3、初始化二分类

```
class Classification(nn.Module):
    def __init__(self):
        super(Classification, self).__init__()
        self.hidden = nn.Linear(num_inputs, num_hiddens)
        self.tanh = nn.Tanh()
        self.output = nn.Linear(num_hiddens, num_outputs)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.hidden(x)
        x = self.tanh(x)
        x = self.output(x)
        x = self.sigmoid(x)
        return x
```

4、初始化层内参数，损失函数和学习率等参数

```
init.normal_(net.hidden.weight, mean=0, std=0.01)
init.normal_(net.output.weight, mean=0, std=0.01)
init.constant_(net.hidden.bias, val=0)
init.constant_(net.output.bias, val=0)

loss = nn.BCELoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.05, momentum=0.9)
```

5、定义激活函数

```
def evaluate_accuracy(x,y,net):
    out = net(x)
    sum = 0
    for index,i in enumerate(out):
        if(i.ge(0.5) ==y[index]):
            sum+=1
    a = sum/600
    return a
```

6、训练集训练

```
def train(net,train_x,train_y,loss,num_epochs,optimizer=None):

    for epoch in range(num_epochs):
        out = net(train_x)
        l = loss(out, train_y)
        optimizer.zero_grad()
        l.backward()
        optimizer.step()
        train_loss = l.item()

        if (epoch + 1) % 100 == 0:
            train_acc = evaluate_accuracy(train_x,train_y, net)
            print("======",train_acc)
            print('epoch %d ,loss %.4f'%(epoch + 1,train_loss)+', train acc {:.2f}%'
                  .format(train_acc*100))

num_epochs = 1000
train(net, x0, y0, loss, num_epochs, optimizer)
```

7、利用验证集合调参，最后参数为 lr=0.01

```
#validate
print("validate")
train_acc = evaluate_accuracy(x1,y1,net)
print("======",train_acc)
print('train acc {:.2f}%'.format(train_acc*100))
```

```
validate
====== 200
======= 1.0
train acc 100.00%
```

8、利用验证集合验证

```
#test
print("test")
train_acc = evaluate_accuracy(x2,y2,net)
print("======",train_acc)
print('train acc {:.2f}%'.format(train_acc*100))
```

```
test
====== 200
======= 1.0
```

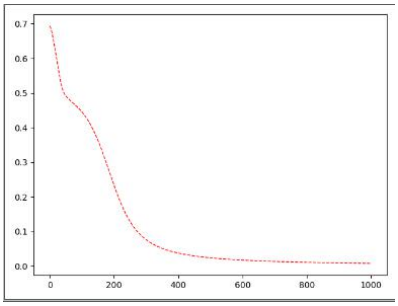
9、绘制 roc 曲线

```
ans = []
for index,i in enumerate(out):
    if(i.ge(0.5)):
        ans.append(1)
    else:
        ans.append(0)
#compute fpr,tpr,threshold
print(ans)
print(y2)
fpr,tpr,threshold = roc_curve(ans, y2,pos_label=1)
roc_auc = auc(fpr,tpr) ###计算auc的值
print(roc_auc)
plt.figure()
lw = 2
plt.figure(figsize=(10,10))
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc) ###假正率为横坐标，真正率为纵坐标做曲
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

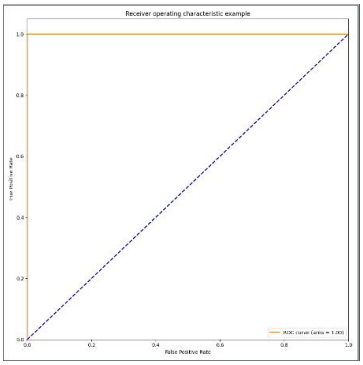
10、绘制训练曲线

```
def train(net,train_x,train_y,loss,num_epochs,optimizer=None):
    loss = []
    for epoch in range(num_epochs):
        out = net(train_x)
        l = loss(out, train_y)
        optimizer.zero_grad()
        l.backward()
        optimizer.step()
        train_loss = l.item()
        loss.append(train_loss)
        if (epoch + 1) % 100 == 0:
            train_acc = evaluate_accuracy(train_x,train_y, net)
            print("======",train_acc)
            print('epoch %d ,loss %.4f'%(epoch + 1,train_loss)+', train acc {:.2f}%'
                  .format(train_acc*100))

num_epochs = 1000
train(net, x0, y0, loss, num_epochs, optimizer)
x1 = [i for i in range(3000)]
plt.plot(x1, loss, color='red', linewidth=1.0, linestyle='--')
# 显示图表
plt.show()
```

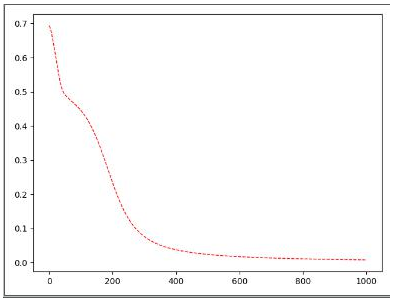
11、训练曲线：



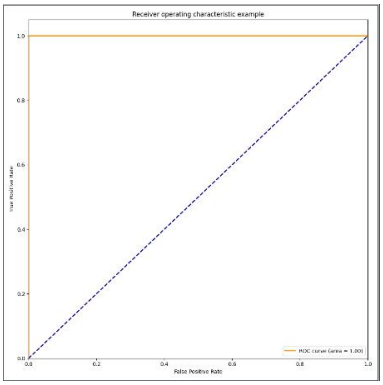
12roc 曲线：



双隐层:



roc 曲线:



实验讨论：

a）利用交叉验证法

b）偏差过高过拟合

c）双隐层较好能够更快的到达训练手腕点

第六章

实验内容：

C6-1 Fisheriris data can be read by the following matlab command:

'load fisheriris'.

Its meaning can be found from the internet.

(a) Use one-against-all strategy to decompose the 4-ary classification problem into binary classification problems. Use SVM to solve each binary classification problems and combine all the binary classifiers to solve the problem.

(b) Use KNN to solve the fisheriris data classification problem.

(c) Use MLP to solve each binary classification problems and combine all the binary classifiers to solve the problem.

Compare the generalization performance and the storage required for the classifiers obtained in (a), (b) and (c).

实验步骤：

a)

1、利用 matlib 的 fisheriris 导入数据

```
clc,clear,close all
load fisheriris
X = meas;
Y = species;
tabulate(Y);
```

2、创建 SVM 模板（二分类模型），并对分类变量进行标准化处理

```
t = templateSVM( 'Standardize' , 1); % 对分类变量进行标准化处理
```

3、基于 SVM 二分类模型进行训练并生成多分类模型

```
Mdl = fitcecoc(X,Y,'Learners',t,...
    'ClassNames',{'setosa','versicolor','virginica'})
```

实验结果：

```
       Value    Count   Percent
      setosa       50     33.33%
  versicolor       50     33.33%
   virginica       50     33.33%

Mdl =

  ClassificationECOC
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: {'setosa'  'versicolor'  'virginica'}
           ScoreTransform: 'none'
           BinaryLearners: {3×1 cell}
               CodingName: 'onevsone'


  Properties, Methods
```

B)KNN

导入数据，创建画布,画出数据点，创建 knn

```
>> load fisheriris
X = meas;    % Use all data for fitting
Y = species; % Response data
>> view plain copy
错误使用 view (line 73)
输入参数无效

>> Mdl = fitcknn(X,Y)

Mdl =

  ClassificationKNN
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: {'setosa'  'versicolor'  'virginica'}
          ScoreTransform: 'none'
         NumObservations: 150
                Distance: 'euclidean'
            NumNeighbors: 1


  Properties, Methods
```

调整最近邻数量：

Mdl.NumNeighbors = 4;

进行预测：

```
>> Mdl = fitcknn(X,Y,'NumNeighbors',4);
>> flwr = [5.0 3.0 5.0 1.45];
>> flwrClass = predict(Mdl,flwr)

flwrClass =

  1×1 cell 数组

    {'versicolor'}
```

利用 crossval(Mdl)评估效果：

```
>> CVMdl = crossval(Mdl)

CVMdl =

  classreg.learning.partition.ClassificationPartitionedModel
    CrossValidatedModel: 'KNN'
         PredictorNames: {'x1'  'x2'  'x3'  'x4'}
           ResponseName: 'Y'
        NumObservations: 150
                  KFold: 10
              Partition: [1×1 cvpartition]
             ClassNames: {'setosa'  'versicolor'  'virginica'}
         ScoreTransform: 'none'


  Properties, Methods

>> kloss = kfoldLoss(CVMdl)

kloss =

    0.0400
```

C）MLP

导入数据

创建 MLP 模型：

```
>> for i=1:10
    %随机分为2组，每组各75个样本，每组中每种花各有25个样本
    rand_set_setosa=randperm(50);
    rand_set_versicolor= randperm(50)+50;
    rand_set_virginica=randperm(50)+100;
    %随机分为2组，每组各75个样本，每组中每种花各有25个样本
    training_setosa=rand_set_setosa(1:25);
    predict_setosa=rand_set_setosa(26:50);
    training_versicolor=rand_set_versicolor(1:25);
    predict_versicolor=rand_set_versicolor(26:50);
    training_virginica=rand_set_virginica(1:25);
    predict_virginica=rand_set_virginica(26:50);
    %训练集(Copyright © http://blog.csdn.net/s_gy_zetrov. All Rights Reserved)
    X=[data(training_setosa,1:4);data(training_versicolor,1:4);data(training_virginica,1:4)];%75*4
    output=[data(training_setosa,5);data(training_versicolor,5);data(training_virginica,5)];%75*1
    [XX,PS]=mapminmax(X,-1,1);%归一化/标准化
    %net=feedforwardnet([6 5 4],'trainlm');
    %(Copyright © http://blog.csdn.net/s_gy_zetrov. All Rights Reserved)
    net=feedforwardnet;%bp神经网络使用缺省设定
    net.trainparam.epochs=500;%最大迭代500
    net.trainparam.goal=0.01;%误差0.01
    [net,tr,output1,E]=train(net,XX',output');%开始训练，转置

    %测试集
    Y=[data(predict_setosa,1:4);data(predict_versicolor,1:4);data(predict_virginica,1:4)];
    c=[data(predict_setosa,5);data(predict_versicolor,5);data(predict_virginica,5)];
    YY = mapminmax('apply',Y,PS);
    YY2=sim(net,YY');%转置 (Copyright © http://blog.csdn.net/s_gy_zetrov. All Rights Reserved)
    %计算预测正确率
    [s1,s2]=size(YY2);
    hitNum=0;
    for j=1:s2
        index=round(YY2(:,j));
        if(index==c(j))
            hitNum=hitNum+1;
        end
    end
```

```
    end
    sprintf('预测正确率为 %3.3f%%',100*hitNum/s2)
end
```

结果最大值为：82%

```
ans =
    '预测正确率为 40.000%'

ans =
    '预测正确率为 82.667%'

ans =
    '预测正确率为 34.667%'

ans =
    '预测正确率为 74.667%'

ans =
    '预测正确率为 37.333%'

ans =
    '预测正确率为 46.667%'

ans =
    '预测正确率为 45.333%'
```

第 7 章

实验内容:

试编程实现 K-means 算法，设置三组不同的 k 值、三组不同的初始中心点，在西瓜数据集 4.0 上进行实验比较，并讨论什么样的初始中心有利于取得好结果。

表 1　西瓜数据集 4.0

编号	密度	含糖率	编号	密度	含糖率	编号	密度	含糖率
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

实验步骤:

1、导入相应的包和利用 dataframe 导入数据集

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import time

# 西瓜数据集4.0: 密度 含糖率 标签
data = [[0.697, 0.460, 1],
        [0.774, 0.376, 1],
        [0.634, 0.264, 1],
        [0.608, 0.318, 1],
        [0.556, 0.215, 1],
        [0.430, 0.237, 1],
        [0.481, 0.149, 1],
        [0.437, 0.211, 1],
        [0.666, 0.091, 0],
        [0.243, 0.267, 0],
        [0.245, 0.057, 0],
        [0.343, 0.099, 0],
        [0.639, 0.161, 0],
        [0.657, 0.198, 0],
        [0.360, 0.370, 0],
        [0.593, 0.042, 0],
        [0.719, 0.103, 0],
        [0.359, 0.188, 0],
        [0.339, 0.241, 0],
        [0.282, 0.257, 0],
        [0.748, 0.232, 0],
        [0.714, 0.346, 1],
        [0.483, 0.312, 1],
        [0.478, 0.437, 1],
        [0.525, 0.369, 1],
        [0.751, 0.489, 1],
        [0.532, 0.472, 1],
        [0.473, 0.376, 1],
        [0.725, 0.445, 1],
        [0.446, 0.459, 1]]

column = ['density', 'sugar_rate', 'label']
dataSet = pd.DataFrame(data, columns=column)
```

2、定义 k-means 类和距离类方法

```
class K_means(object):
    def __init__(self, k, data, loop_times, error):  # self只有在类的方法中才会有，指向类的实例对象，而非类本身
        self.k = k
        self.data = data
        self.loop_times = loop_times
        self.error = error

    def distance(self, p1, p2):
        return np.linalg.norm(np.array(p1) - np.array(p2))
```

3、定义类方法 fit 根据 k-means 算法，选取 k 个初始化，计算每一个点到簇心的距离，更新簇心，知道簇心不变

```
def fitting(self):
    time1 = time.perf_counter()
    mean_vectors = random.sample(self.data, self.k)  # 随机选取k个初始样本
    initial_main_vectors = mean_vectors
    for vec in mean_vectors:
        plt.scatter(vec[0], vec[1], s=100, color='black', marker='s')
    times = 0
    clusters = list(map((lambda x: [x]), mean_vectors))
    while times < self.loop_times:
        change_flag = 1  # 标记簇均值向量是否改变
        for sample in self.data:
            dist = []
            for vec in mean_vectors:
                dist.append(self.distance(vec, sample))  # 计算样本到每个聚类中心的距离
            clusters[dist.index(min(dist))].append(sample)  # 找到离该样本最近的聚类中心，并将它放入该簇

        new_mean_vectors = []
        for c,v in zip(clusters, mean_vectors):  # zip()将两个对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表
            cluster_num = len(c)
            cluster_array = np.array(c)
            new_mean_vector = sum(cluster_array) / cluster_num  # 计算出新的聚类簇均值向量
            mean_vector = np.array(v)
            if all(np.true_divide((new_mean_vector - mean_vector), mean_vector) < np.array([self.error, self.error])):
                new_mean_vectors.append(mean_vector)  # 均值向量未改变
                change_flag = 0
            else:
                # dataFrame转List(), 括号不能忘
                new_mean_vectors.append(new_mean_vector.tolist())  # 均值向量发生改变
```

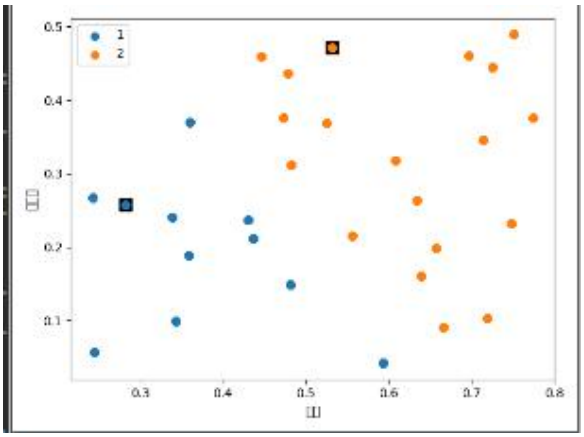
```
    if change_flag == 1:
        mean_vectors = new_mean_vectors
    else:
        break
    times += 1
time2 = time.perf_counter()
# str.format(), 基本语法是通过 {} 和 : 来代替以前的 %
print('本次选取的{}个初始向量为{}'.format(self.k, initial_main_vectors))
print('共进行{}轮'.format(times))
print('共耗时{:.2f}s'.format(time2 - time1))  # 取2位小数
for cluster in clusters:
    x = list(map(lambda arr: arr[0], cluster))
    y = list(map(lambda arr: arr[1], cluster))
    plt.scatter(x, y, marker='o', label=clusters.index(cluster)+1)
plt.xlabel('密度')
plt.ylabel('含糖率')
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.legend(loc='upper left')
plt.show()
```

4、定义不同的 k 值

```
# 调用K_means, 执行方法fitting()
k_means = K_means(j, dataSet[['density', 'sugar_rate']].values.tolist(), 1000, 0.0000001)
k_means.fitting()
```

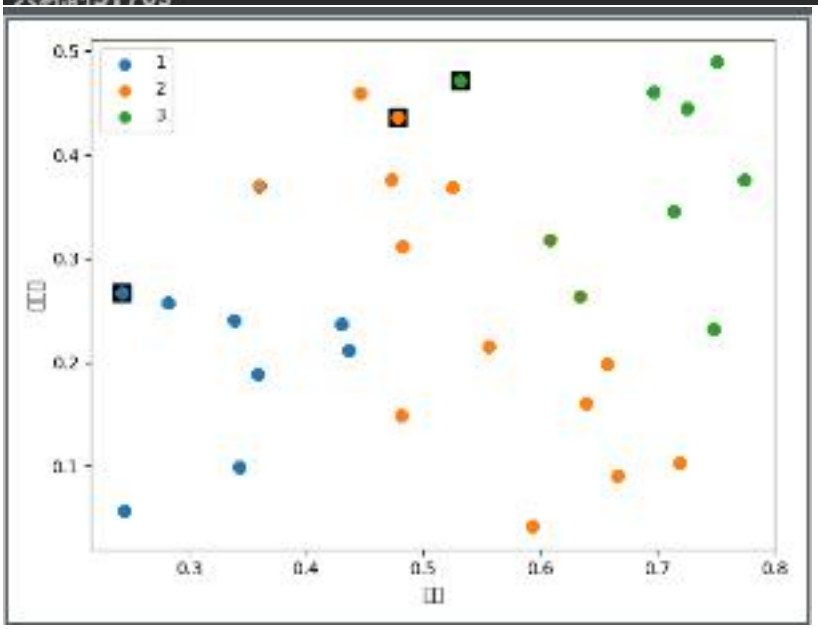
5、当 k=2 时

```
本次选取的2个初始向量为[[0.282, 0.257], [0.532, 0.472]]
共进行276轮
共耗时1.16s
```



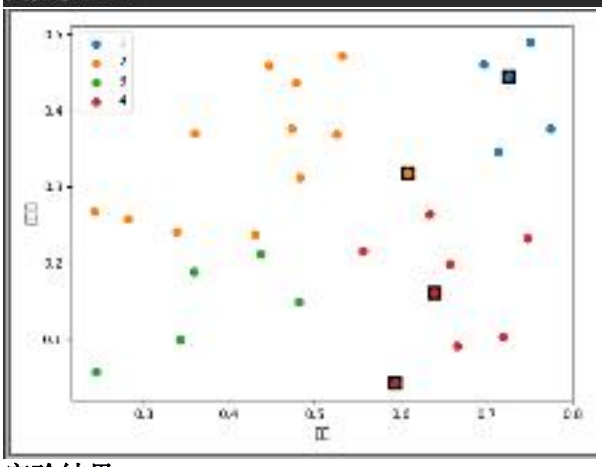
当 k=3 时

```
本次选取的3个初始向量为[[0.243, 0.267], [0.478, 0.437], [0.532, 0.472]]
共进行631轮
共耗时5.76s
```



当 k=4 时

```
本次选取的4个初始向量为[[0.725, 0.445], [0.608, 0.318], [0.593, 0.042], [0.639, 0.161]]
共进行1轮
共耗时0.02s
```



实验结果:

发现当 k=3 时有更好的分类效果

第八章

实验内容：

请通过对 Iris data 的可视化，比较 PCA、LDA 和 ICA 的可视化效果。

实验步骤：

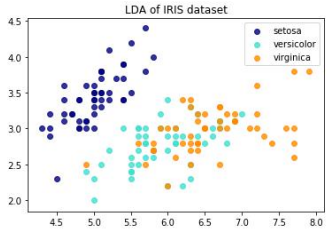
导入 iris 数据集合：

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import pandas as pd
```

```
iris = datasets.load_iris()

X = iris.data
y = iris.target
target_names = iris.target_names
pd.DataFrame(X, columns=iris.feature_names).head()
```

iris 数据原始数据点：

```
In [8]: plt.figure()
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):
            plt.scatter(X[y == i, 0], X[y == i, 1], alpha=.8, color=color,
                        label=target_name)
        plt.legend(loc='best', shadow=False, scatterpoints=1)
        plt.title('LDA of IRIS dataset')

        plt.show()
```



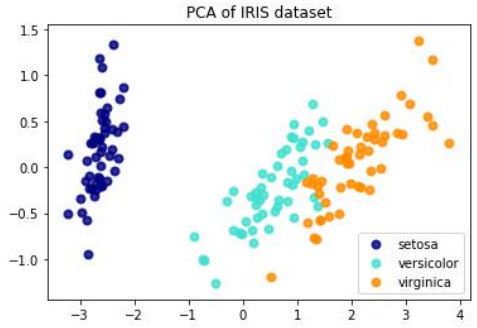
进行 pca 和 lda：

```
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)

lda = LinearDiscriminantAnalysis(n_components=2)
X_r2 = lda.fit(X, y).transform(X)
```

画出 pca 变化后的数据点：

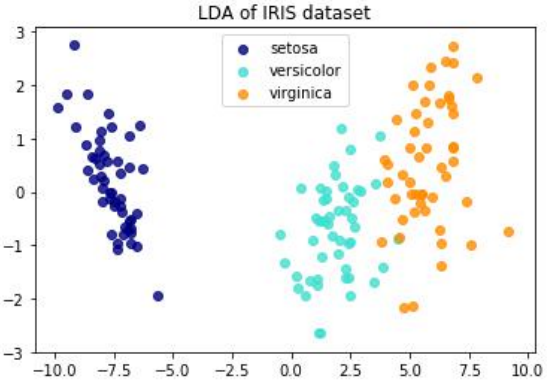
```
plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2

for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
```

```
Text(0.5, 1.0, 'PCA of IRIS dataset')
```



画出 LDA 变化：

```
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r2[y == i, 0], X_r2[y == i, 1], alpha=.8, color=color,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of IRIS dataset')

plt.show()
```

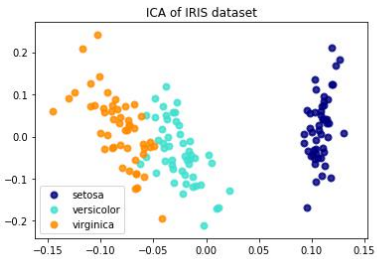


ICA 变化：

In [9]:
```
from sklearn.decomposition import FastICA
ica = FastICA(n_components=2)
X_r = ica.fit(X).transform(X)
```

In [10]:
```
plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2

for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('ICA of IRIS dataset')
```

Out[10]:
```
Text(0.5, 1.0, 'ICA of IRIS dataset')
```



总结：

利用 ICA 和 LDA 可以让两个本来看起来比较重叠的数据集合划分开来，而 pca 重叠部分较多