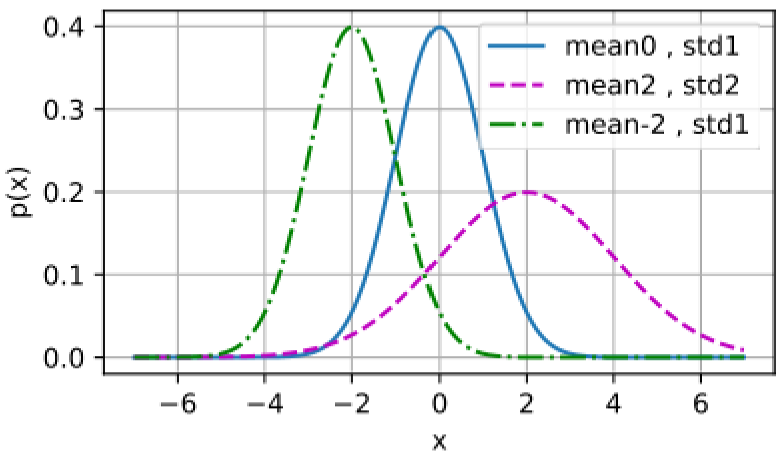
线性模型实战

一、正态分布和平方损失

```python
#%% md
# 线性模型实战
## 正态分布和平方损失

#%%
# 根据正态分布定义函数
import math
import numpy as np
from d2l import torch as d2l

def normal(x,mu,sigma):
    p = 1 / math.sqrt(2 * math.pi * sigma**2)
    return p * np.exp(-0.5 / sigma**2 * (x - mu)**2)

#可视化

x = np.arange(-7 , 7 , 0.01)

# 定义三组不同的正态分布数据
# 标准正态分布
params = [(0,1),(2,2),(-2,1)]
d2l.plot(x,[normal(x,mu,sigma) for mu ,sigma in params] ,
         xlabel='x',ylabel='p(x)',figsize=(4.5,2.5),
         legend=[f'mean{mu} , std{sigma}' for mu ,sigma in params] )
```



二、线性模型

> d2l 简洁实现线性模型

```python
#%% md
## 线性回归
### 定义一个用于学习的线性参数
- 生成数据
- 使用线性模型进行模拟
#%%
import torch
def synthetic_data(w,b,num_example):
    '''y = wx + b '''
    # 标准正态分布进行生成数据
    x = torch.normal(0,1,(num_example,len(w)))
    y = torch.matmul(x,w) + b
    # 添加噪音
    y += torch.normal(0,0.01,y.shape)
    # y 转换为竖向量
    return x , y.reshape((-1,1))

import numpy as np
from torch.utils import data

# 生成数据
true_w = torch.tensor([2.3,-3.4])
```

```
true_b = 4.5
data_x,data_y = synthetic_data(true_w,true_b,2000)

# 处理数据集
batch_size = 20
data_iter = d2l.load_array((data_x,data_y),batch_size)

next(iter(data_iter))

#%% 使用torch定义线性模型
from torch import nn
net = nn.Sequential(nn.Linear(2,1))
# 参数初始化
net[0].weight.data.normal_(0,0.01)
net[0].bias.data.fill_(0)

loss = nn.MSELoss()
# 将net中的所有参数按照0.03的学习率使用SGD优化算法进行迭代
trainer = torch.optim.SGD(net.parameters(),lr=0.03)
num_epochs = 5
for i in range(num_epochs):
    for x , y in data_iter:
        l = loss(net(x),y)
        trainer.zero_grad()
        # 回溯求救梯度
        l.backward()
        # 模型参数更新
        trainer.step()
    # 输出最后总的loss值
    l  = loss(net(data_x),data_y)
    print(f'{i + 1} epoch  loss={l}')
#%%
net[0].weight.data , net[0].bias.data
```

PYTHON

```
[tensor([[-1.1131,  2.0629],
         [ 1.1003,  0.1096],
         [-1.1681, -0.2245],
         [-2.0316,  1.1333],
         [ 1.6061, -0.3019],
         [ 0.0610, -2.1425],
         [-0.3337,  0.0319],
         [ 1.0101, -0.7331],
         [ 1.4405,  1.3718],
         [ 1.5282,  0.5440],
         [ 2.6825, -0.4522],
         [ 0.8149,  0.4608],
         [-1.3217, -0.2812],
         [-0.8050, -0.1768],
         [-1.4841,  0.9911],
         [-0.2711, -0.7556],
         [-0.7281,  0.3236],
         [-0.4715, -0.8502],
         [-0.1531, -0.2671],
         [ 0.6098, -0.8820]]),
 tensor([[-5.0922],
         [ 6.6623],
         [ 2.5742],
         [-4.0495],
         [ 9.2047],
         [11.9108],
         [ 3.6134],
         [ 9.3152],
         [ 3.1404],
         [ 6.1829],
         [12.2321],
         [ 4.7878],
         [ 2.4126],
         [ 3.2625],
         [-2.2778],
         [ 6.4394],
         [ 1.7328],
         [ 6.3040],
         [ 5.0420],
         [ 8.8931]])]
```

```
1 epoch  loss=0.000198044566786848
2 epoch  loss=0.0001007054015644826
3 epoch  loss=0.00010071233555208892
4 epoch  loss=0.00010055938037112355
5 epoch  loss=0.0001012328066281043

(tensor([[ 2.3004, -3.3996]]), tensor([4.5007]))
```

2.2 一元线性回归 自己实现并画图

```python
# 线性模型
import torch
import numpy as np
import matplotlib.pyplot as plt
from tensorboardX import SummaryWriter
from torch.utils.data import DataLoader , Dataset ,TensorDataset
from torch import nn
import torchvision
from torch.nn import MSELoss
from torch.optim import SGD
# 创建一些最简单的数据，并画图
data_dir = '../data/MLData/ex1data1.txt'
data = np.loadtxt(data_dir,dtype=np.float32 , delimiter=',')
plt.scatter(data[:,0],data[:,1],c='r')
# print(data.shape)

#%%
# class MyData(Dataset):
#     def __init__(self , data_dir ):
#         self.data_dir = data_dir
#         self.data = torch.tensor(np.loadtxt(data_dir,dtype=np.float3
#     def __getitem__(self, item):
#         return self.data[item][0].reshape((-1,1)) , self.data[item][
#     def __len__(self):
#         return len(self.data)
# data = MyData(data_dir)
# batch_size = 64
#
# dataLoader = DataLoader(data , batch_size , shuffle=True , drop_last
#
# class MyNet(torch.nn.Module):
#     def __init__(self):
#         super(MyNet, self).__init__()
#         self.w = torch.randn(1,requires_grad=True)
#         self.b = torch.randn(1,requires_grad=True)
#     def forward(self,x):
#         return x.matmul(self.w) + self.b
#     def get_params(self):
#         return [self.w , self.b]
#
# def optim(params,lr,batch_size):
#     # 进行优化
#     with torch.no_grad():
#         for param in params:
#             param -= lr * param.grad / batch_size
#             param.grad.zero_()
# net = MyNet()
# loss_fn = MSELoss()
#
# epochs = 40
# lr = 0.05
#
#
# for epoch in range(epochs):
#     net.train()
#     for x , y  in dataLoader:
#         outs = net(x)
#         loss = loss_fn(outs,y)
#         loss.backward()
#         optim(net.get_params() , lr , batch_size)
#
#     net.eval()
#     with torch.no_grad():
```

```
#         loss_test = 0
#         step = 0
#         for x, y in dataLoader:
#             outs = net(x)
#             loss = loss_fn(outs, y)
#             loss_test += loss
#             step += 1
#         print(f'epoch:{epoch} , loss {loss_test / step}')
#
# x = np.linspace(0,25,100)
# y = [net.w.item() * i + net.b.item() for i in x ]
# plt.plot(x,y ,label='y = w * x + b ')
# plt.xlabel('x_label')#设置x轴的标签为x_label
#
# plt.ylabel('y_label')#设置x轴的标签为x_label
#
# plt.title('lineplot example')#设置图表标题
#
# plt.legend()#设置图例的前题是l1,l2指定了label
# plt.show()
#%% 完全利用torch写
data = torch.tensor(data)
data_tensor = TensorDataset(data[:,0].reshape(-1,1), data[:,1].reshape
data_loader = DataLoader(data_tensor,64,shuffle=True,drop_last=False)

net = nn.Sequential(nn.Linear(1,1))

loss_fn = MSELoss()
lr = 0.01
optim = SGD(net.parameters(),lr)
epochs = 20
writer = SummaryWriter('tensorboard_logs')

for epoch in range(epochs):
    net.train()
    for x,y in data_loader:
        outs = net(x)
        loss = loss_fn(outs,y)
        optim.zero_grad()
        loss.backward()
        optim.step()

    net.eval()
    test_loss = 0
    step = 0
    with torch.no_grad():
        for x,y in data_loader:
            outs = net(x)
            loss = loss_fn(outs,y)
            test_loss += loss
            step += 1
    writer.add_scalar('Linear',test_loss / step , epoch)

writer.close()
# print(net[0].weight.data , net[0].bias.data)
x = np.linspace(0,25,100)
y = [net[0].weight.data.item() * i + net[0].bias.data.item() for i in
plt.plot(x,y ,label='y = w * x + b ')
plt.xlabel('x_label')#设置x轴的标签为x_label

plt.ylabel('y_label')#设置x轴的标签为x_label

plt.title('lineplot example')#设置图表标题

plt.legend()#设置图例的前题是l1,l2指定了label
plt.show()
```

2.3 多元回归

> 数据需要进行处理

由于y太大，导致loss很大，一次迭代后就inf和NAN了，没法训练

PYTHON

```python
## d2l 第4.10.4

# 若无法获得测试数据，则可根据训练数据计算均值和标准差
# pandas 根据每列的数据类型筛选列
numeric_features = all_features.dtypes[all_features.dtypes != 'object'
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# 在标准化数据之后，所有均值消失，因此我们可以将缺失值设置为0
all_features[numeric_features] = all_features[numeric_features].fillna
```

建立模型，训练

PYTHON

```python
# 线性模型
import torch
import numpy as np
import matplotlib.pyplot as plt
from tensorboardX import SummaryWriter
from torch.utils.data import DataLoader, Dataset, TensorDataset
from torch import nn
import torchvision
from torch.nn import MSELoss
from torch.optim import SGD
import pandas as pd
np.set_printoptions(suppress=True)

# 创建一些最简单的数据，并画图
data_dir = '../data/MLData/1/ex1data2.txt'
# data = np.loadtxt(data_dir, dtype=np.float32, delimiter=',')
# print(data.shape)
# print(data[:10])
data_pd = pd.read_csv(data_dir,delimiter=',',header=None)

# print(data)
# 由于数据出现异常，所以将数据进行归一化等操作
# print(data_pd)
```

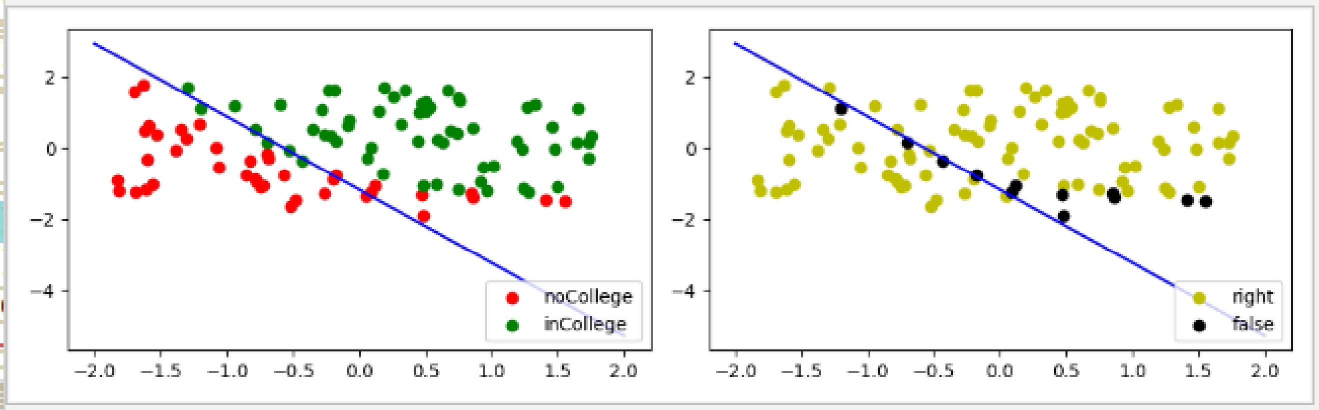
```python
data_pd = data_pd.apply(lambda x: (x - x.mean()) / (x.std()))
# print(data_pd)
data_tensor = torch.tensor(np.array(data_pd),dtype=torch.float32)
# print(data)
dataset = TensorDataset(data_tensor[:,:2],data_tensor[:,2].reshape(-1,
dataLoader = DataLoader(dataset,16 , shuffle=True , drop_last=False)

net = nn.Sequential(
    nn.Linear(2,1)
)
loss_fn = MSELoss()
lr = 0.01
optim = SGD(net.parameters(),lr )

epochs = 20
for epoch in range(epochs):
    net.train()
    for x , y in dataLoader:
        outs = net(x)
        loss = loss_fn(outs,y)
        # print(loss)
        optim.zero_grad()
        loss.backward()
        optim.step()

    net.eval()
    with torch.no_grad():
        test_loss = 0
        step = 0
        for x, y in dataLoader:
            outs = net(x)
            loss = loss_fn(outs, y)
            test_loss += loss
            step+=1
        print(f'epoch {epoch} loss {test_loss / step}')
```

2.4 对数几率回归

PYTHON

```python
# 线性模型
import torch
import numpy as np
import matplotlib.pyplot as plt
import torch as torch
from tensorboardX import SummaryWriter
from torch.utils.data import DataLoader, Dataset, TensorDataset
from torch import nn
import torchvision
from torch.nn import MSELoss ,BCELoss
import torch.nn.functional as F
from torch.optim import SGD
import pandas as pd
np.set_printoptions(suppress=True)

# 创建一些最简单的数据，并画图
data_dir = '../data/MLData/2/ex2data1.txt'
data_pd = pd.read_csv(data_dir,delimiter=',',header=None)
# print(data.shape)
# print(data[:10])
data_pd.iloc[:,:2] = data_pd.iloc[:,:2].apply(lambda x: (x - x.mean())
data = np.array(data_pd)
fig, axes = plt.subplots(1, 2, figsize=(10, 3), tight_layout=True)
axes[0].scatter(data[data[:,2] == 0,0],data[data[:,2] == 0,1],c='r',la
axes[0].scatter(data[data[:,2] == 1,0],data[data[:,2] == 1,1],c='g',la
# plt.rcParams["font.family"]="STSong"

# plt.show()

data_tensor = torch.tensor(np.array(data_pd),dtype=torch.float32)
dataset = TensorDataset(data_tensor[:,:2],data_tensor[:,2].reshape(-1,
dataLoader = DataLoader(dataset,16 , shuffle=True , drop_last=False)
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.seq1 = nn.Sequential(
            nn.Linear(2, 1),
            nn.Sigmoid()
        )
    def forward(self,x):
        return self.seq1(x)
net = MyNet()
def loss_fn(outs,targets):
    return sum(-targets * torch.log(outs) - (1 - targets) * torch.log(
# outs = torch.tensor([0.2,0.5,0.6,0.1,0.2]).reshape((-1,1))
# targets = torch.tensor([0,0,1,0,1]).reshape((-1,1))
# loss = loss_fn(outs,targets)
# print(loss)
# print(net(torch.tensor([-8,-6] , dtype=torch.float32)))

# loss_fn = BCELoss()
lr = 0.01
optim = SGD(net.parameters(),lr )

epochs = 40
for epoch in range(epochs):
    net.train()
    for x , y in dataLoader:
        outs = net(x)
        loss = loss_fn(outs,y)
        # print(loss)
        optim.zero_grad()
        # acc = sum(outs == y)
        # print(acc)
        loss.backward()
        optim.step()

    net.eval()
    with torch.no_grad():
        test_loss = 0
        step = 0
        for x, y in dataLoader:
            outs = net(x)
            # print(sum(torch.round(outs)==y))
            loss = loss_fn(outs, y)
```

```
                test_loss += loss
                step+=1

            print(f'epoch {epoch} loss {test_loss / step}')

x = data_tensor[:, :2]
y = data_tensor[:,2].reshape(-1,1)
outs = net(x)
x = x.numpy()
a = (torch.round(outs) == y).numpy().reshape(-1)
axes[1].scatter(x[a, 0], x[a, 1], c='y',label='right')
axes[1].scatter(x[~a, 0], x[~a, 1], c='k',label='false')

x = np.linspace(-2,2,500)

b = net.seq1[0].bias.data
w1 = net.seq1[0].weight.data[0,0]
w2 = net.seq1[0].weight.data[0,1]
y = ( - w1 * x - b) / w2
axes[0].plot(x,y,c='b')
axes[1].plot(x,y,c='b')
# x  = torch.tensor(x , dtype=torch.float32)
# y  = torch.tensor(y,dtype=torch.float32)
# outs = net(torch.cat((x.reshape(-1,1) , y.reshape(-1,1)), 1) )
axes[0].legend(loc='lower right')
axes[1].legend(loc='lower right')
plt.show()
```



Feature mapping

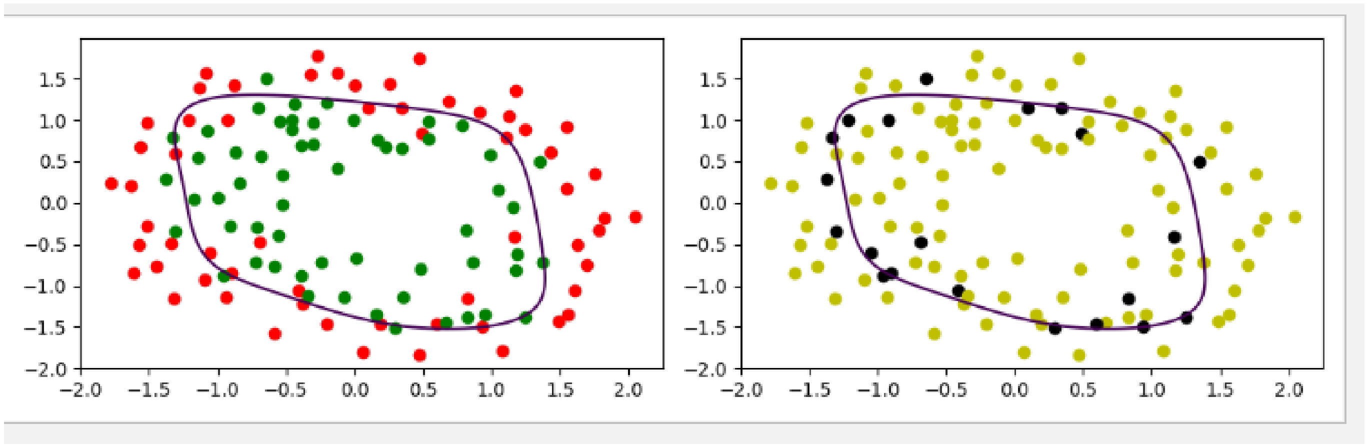
PYTHON

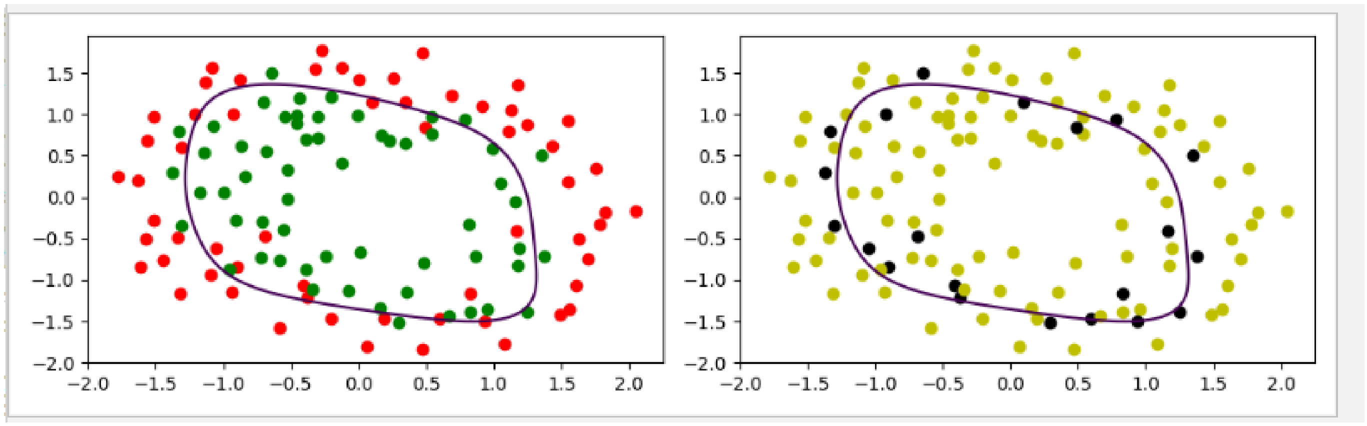
```python
# 线性模型
import torch
import numpy as np
import matplotlib.pyplot as plt
import torch as torch
from tensorboardX import SummaryWriter
from torch.utils.data import DataLoader, Dataset, TensorDataset
from torch import nn
import torchvision
from torch.nn import MSELoss ,BCELoss
import torch.nn.functional as F
from torch.optim import SGD
import pandas as pd

# 创建一些最简单的数据，并画图
data_dir = '../data/MLData/2/ex2data2.txt'
data_pd = pd.read_csv(data_dir,delimiter=',',header=None)
# print(data.shape)
# print(data[:10])
data_pd.iloc[:,:2] = data_pd.iloc[:,:2].apply(lambda x: (x - x.mean())
data = np.array(data_pd)
fig, axes = plt.subplots(1, 2, figsize=(10, 3), tight_layout=True)
axes[0].scatter(data[data[:,2] == 0,0],data[data[:,2] == 0,1],c='r',la
axes[0].scatter(data[data[:,2] == 1,0],data[data[:,2] == 1,1],c='g',la

# 特征映射 试一下

# data = torch.randn((12,2))

def Feature_mapping(data):

    out = torch.ones((len(data), 1))
    degree = 6

    for i in range(1,degree+1):
        for j in range(i+1):
            temp = data[:,0] ** j * data[:,1] ** (i - j)
            # print(j ,' ', i-j)
            temp = torch.reshape(temp,(-1,1))
            out = torch.cat((out,temp),dim=1)
    # out 为 映射后的数据
    return out

# 还是处理数据
data_tensor = torch.tensor(data,dtype=torch.float32)
dataset = TensorDataset(data_tensor[:,:2],data_tensor[:,2].reshape(-1,
dataLoader = DataLoader(dataset,16,shuffle=True,drop_last=False)

net = nn.Sequential(
    nn.Linear(28,1),
    nn.Sigmoid()
)

epochs = 20
lr = 0.01
loss_fn = BCELoss()
optim = SGD(net.parameters(),lr)

for epoch in range(epochs):
    net.train()
    for x , y in dataLoader:
        x = Feature_mapping(x)
        out = net(x)
        loss = loss_fn(out,y)
        optim.zero_grad()
        loss.backward()
        optim.step()
for x , y in dataLoader:
    temp = Feature_mapping(x)
    out = net(temp)
    loss = loss_fn(out,y)
    x = x.numpy()
    a = (torch.round(out) == y).numpy().reshape(-1)
    axes[1].scatter(x[a, 0], x[a, 1], c='y',label='right')
    axes[1].scatter(x[~a, 0], x[~a, 1], c='k',label='false')

# 不会画圈

x = y = np.arange(-2, 2, 0.05)
x, y = np.meshgrid(x,y)

def hautu(x,y):
    x = torch.tensor(x , dtype=torch.float32)
    y = torch.tensor(y, dtype=torch.float32)
    z = torch.ones_like(x, dtype=torch.float32)
    # 不想麻烦了，用for循环把
    k = x.shape
    for i  in range(k[0]):
        for j in range(k[1]):
            z[i,j] = net[0](Feature_mapping(torch.cat((x[i,j].reshape(

    return z.detach().numpy()

# hautu(x,y)
axes[0].contour(x, y,hautu(x,y), [0])     #x**2 + y**2 = 9 的圆形
axes[1].contour(x, y,hautu(x,y), [0])     #x**2 + y**2 = 9 的圆形

# plt.axis('scaled')
# plt.show()

plt.show()
```

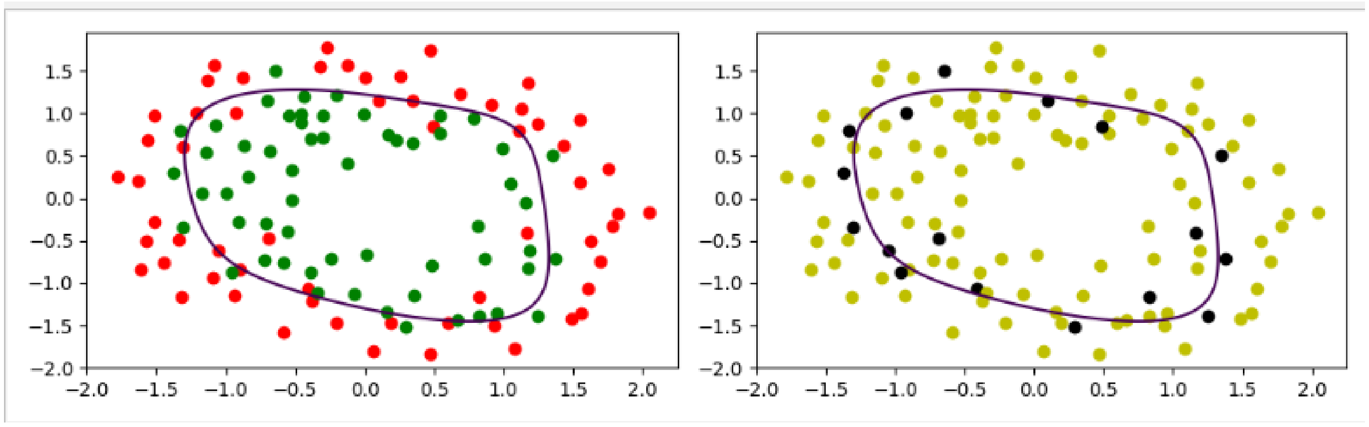
```
PYTHON
# 线性模型
import torch
import numpy as np
import matplotlib.pyplot as plt
import torch as torch
from tensorboardX import SummaryWriter
from torch.utils.data import DataLoader, Dataset, TensorDataset
from torch import nn
import torchvision
from torch.nn import MSELoss ,BCELoss
import torch.nn.functional as F
from torch.optim import SGD
import pandas as pd

# 创建一些最简单的数据，并画图
data_dir = '../data/MLData/2/ex2data2.txt'
data_pd = pd.read_csv(data_dir,delimiter=',',header=None)
# print(data.shape)
# print(data[:10])
data_pd.iloc[:,:2] = data_pd.iloc[:,:2].apply(lambda x: (x - x.mean())
data = np.array(data_pd)
fig, axes = plt.subplots(1, 2, figsize=(10, 3), tight_layout=True)
axes[0].scatter(data[data[:,2] == 0,0],data[data[:,2] == 0,1],c='r',la
axes[0].scatter(data[data[:,2] == 1,0],data[data[:,2] == 1,1],c='g',la

# 特征映射 试一下

# data = torch.randn((12,2))

def Feature_mapping(data):

    out = torch.ones((len(data), 1))
    degree = 6

 for i in range(1,degree+1):
        for j in range(i+1):
            temp = data[:,0] ** j * data[:,1] ** (i - j)
            # print(j ,' ', i-j)
 temp = torch.reshape(temp,(-1,1))
            out = torch.cat((out,temp),dim=1)
    # out 为 映射后的数据
 return out

# 还是处理数据
data_tensor = torch.tensor(data,dtype=torch.float32)
```

```
dataset = TensorDataset(data_tensor[:,:2],data_tensor[:,2].reshape(-1,
dataLoader = DataLoader(dataset,16,shuffle=True,drop_last=False)

net = nn.Sequential(
    nn.Linear(28,1),
    nn.Sigmoid()
)

epochs = 500
lr = 0.01
lamuda = 0
# loss_fn = BCELoss()
def loss_fn(outs,targets):
    zhengze = torch.norm(net[0].weight.data) ** 2
 return sum(-targets * torch.log(outs) - (1 - targets) * torch.log(1-c
optim = SGD(net.parameters(),lr)

for epoch in range(epochs):
    net.train()
    for x , y in dataLoader:
        x = Feature_mapping(x)
        out = net(x)
        loss = loss_fn(out,y)
        # print(loss)
 optim.zero_grad()
        loss.backward()
        optim.step()

for x , y in dataLoader:
    temp = Feature_mapping(x)
    out = net(temp)
    loss = loss_fn(out,y)
    x = x.numpy()
    a = (torch.round(out) == y).numpy().reshape(-1)
    axes[1].scatter(x[a, 0], x[a, 1], c='y',label='right')
    axes[1].scatter(x[~a, 0], x[~a, 1], c='k',label='false')

# 不会画圈
```

```
x = y = np.arange(-2, 2, 0.05)
x, y = np.meshgrid(x,y)

def hautu(x,y):
    x = torch.tensor(x , dtype=torch.float32)
    y = torch.tensor(y, dtype=torch.float32)
    z = torch.ones_like(x, dtype=torch.float32)
    # 不想麻烦了，用for循环把
 k = x.shape
    for j in range(k[1]):
        z[:,j] = net[0](Feature_mapping(torch.cat((x[:,j].reshape(-1,1

    return z.detach().numpy()



# hautu(x,y)

# axes[0].contour(x, y,hautu(x,y), [0])     #x**2 + y**2 = 9 的圆形
# axes[1].contour(x, y,hautu(x,y), [0])     #x**2 + y**2 = 9 的圆形

t = Feature_mapping(torch.tensor(np.c_[x.ravel(), y.ravel()],dtype=tor
Z = net[0](t)
Z = Z.detach().numpy().reshape(x.shape)
axes[0].contour(x, y,Z, [0])     #x**2 + y**2 = 9 的圆形
axes[1].contour(x, y,Z, [0])

# plt.axis('scaled')
# plt.show()

plt.show()
```







```python
# 新的画图方式（此段放在逻辑回归代码中使用）
t = Feature_mapping(torch.tensor(np.c_[x.ravel(), y.ravel()],dtype=tor
Z = net[0](t)
Z = Z.detach().numpy().reshape(x.shape)
axes[0].contour(x, y,Z, [0])      #x**2 + y**2 = 9 的圆形
axes[1].contour(x, y,Z, [0])
```

2.5 svm