20220511-机器学习

1.学习内容

1.1机器学习

多项式回归欠拟合与过拟合

Python | 复制代码

```python
import math
import numpy as np
import torch
from torch import nn
from d2l import torch as d2l

#生成数据集
max_degree=20
n_train,n_test=100,100
true_w=np.zeros(max_degree)
true_w[0:4]=np.array([5,1.2,-3.4,5.6])

features=np.random.normal(size=(n_train+n_test,1))
np.random.shuffle(features)
poly_features=np.power(features,np.arange(max_degree).reshape(1,-1))
for i in range(max_degree):
    poly_features[:,i]/=math.gamma(i+1)
labels=np.dot(poly_features,true_w)
labels+=np.random.normal(scale=0.1,size=labels.shape)

true_w,features,poly_features,labels=[torch.tensor(x,dtype=torch.float32) for x in [true_w,features,poly_features,labels]]

features[:2],poly_features[:2,:],labels[:2]

#对模型进行训练和测试
def evaluate_loss(net,data_iter,loss):
    metric=d2l.Accumulator(2)
    for X,y in data_iter:
        out=net(X)
        y=y.reshape(out.shape)
        l=loss(out,y)
        metric.add(l.sum(),l.numel())
    return metric[0]/metric[1]

def train(train_features,test_features,train_labels,test_labels,num_epochs=400):
    loss=nn.MSELoss(reduction="none")
    input_shape=train_features.shape[-1]
    net=nn.Sequential(nn.Linear(input_shape,1,bias=False))
    batch_size=min(10,train_labels.shape[0])

 train_iter=d2l.load_array((train_features,train_labels.reshape(-1,1)),batch_size)
```

```
41
     test_iter=d2l.load_array((test_features,test_labels.reshape(-1,1)),batch
    _size,is_train=False)
42      trainer=torch.optim.SGD(net.parameters(),lr=0.01)
43      animator=d2l.Animator(xlabel='epoch', ylabel='loss',
    yscale='log',xlim=[1, num_epochs], ylim=[1e-3, 1e2],legend=['train',
    'test'])
44      for epoch in range(num_epochs):
45          d2l.train_epoch_ch3(net,train_iter,loss,trainer)
46          if epoch==0 or (epoch+1)%20==0:
47              animator.add(epoch+1,
    (evaluate_loss(net,train_iter,loss),evaluate_loss(net,test_iter,loss)))
48      print("weight:",net[0].weight.data.numpy())
49
50  #选择多项式特征中的前4个维度
51  train(poly_features[:n_train,:4],poly_features[n_train:,:4],labels[:n_tra
    in],labels[n_train:],num_epochs=1500)
52
53  #从多项式特征中选择前2个维度，即1和x
54  train(poly_features[:n_train,:2],poly_features[n_train:,:2],labels[:n_tra
    in],labels[n_train:])
55
56  #选择所有维度
57  train(poly_features[:n_train,:],poly_features[n_train:,:],labels[:n_train
    ],labels[n_train:],num_epochs=1500)
```

权重衰减示例

Python | 复制代码

```python
%matplotlib inline
import torch
from torch import nn
from d2l import torch as d2l

n_train,n_test,num_inputs,batch_size=20,100,200,5
true_w,true_b=torch.ones((num_inputs,1))*0.01,0.05
train_data=d2l.synthetic_data(true_w,true_b,n_train)
train_iter=d2l.load_array(train_data,batch_size)
test_data=d2l.synthetic_data(true_w,true_b,n_test)
test_iter=d2l.load_array(test_data,batch_size,is_train=False)

#初始化模型参数
def init_params():
    w=torch.normal(0,1,size=(num_inputs,1),requires_grad=True)
    b=torch.zeros(1,requires_grad=True)
    return [w,b]

#定义L2范数惩罚
def l2_penalty(w):
    return torch.sum(w.pow(2))/2

#训练
def train(lambd):
    w,b=init_params()
    net,loss=lambda X:d2l.linreg(X,w,b),d2l.squared_loss
    num_epochs,lr=100,0.003
    animator=d2l.Animator(xlabel='epochs', ylabel='loss', yscale='log',
xlim=[5, num_epochs], legend=['train', 'test'])
    for epoch in range(num_epochs):
        for X,y in train_iter:
            l=loss(net(X),y)+lambd*l2_penalty(w)
            l.sum().backward()
            d2l.sgd([w,b],lr,batch_size)
        if(epoch+1)%5==0:
            animator.add(epoch+1,
(d2l.evaluate_loss(net,train_iter,loss),d2l.evaluate_loss(net,test_iter,l
oss)))
    print("w的L2范数时: ",torch.norm(w).item())

#忽略正则化直接训练
train(lambd=0)
#使用权重衰减
train(lambd=3)
```

Dropout示例

Python | 复制代码

```python
import torch
from torch import nn
from d2l import torch as d2l

def dropout_layer(X,dropout):
    assert 0<=dropout<=1
    if dropout==1:
        #所有元素都被丢弃
        return torch.zeros_like(X)
    if dropout==1:
        #所有元素都被保留
        return X
    mask=(torch.rand(X.shape)>dropout).float()
    return mask*X/(1.0-dropout)

X=torch.arange(16,dtype=torch.float32).reshape((2,8))
print(X)
print(dropout_layer(X,0.))
print(dropout_layer(X,0.5))
print(dropout_layer(X,1.))

#定义模型参数
num_inputs,num_outputs,num_hiddens1,num_hiddens2=784,10,256,256
#定义模型
dropout1,dropout2=0.2,0.5

class Net(nn.Module):
    def
__init__(self,num_inputs,num_outputs,num_hiddens1,num_hiddens2,is_trainin
g=True):
        super(Net,self).__init__()
        self.num_inputs=num_inputs
        self.training=is_training
        self.lin1=nn.Linear(num_inputs,num_hiddens1)
        self.lin2=nn.Linear(num_hiddens1,num_hiddens2)
        self.lin3=nn.Linear(num_hiddens2,num_outputs)
        self.relu=nn.ReLU()
    def forward(self,X):
        H1=self.relu(self.lin1(X.reshape((-1,self.num_inputs))))
        if self.training==True:
            H1=dropout_layer(H1,dropout1)
        H2=self.relu(self.lin2(H1))
        if self.training==True:
            H2=dropout_layer(H2,dropout2)
        out=self.lin3(H2)
```

```
            return out

net=Net(num_inputs,num_outputs,num_hiddens1,num_hiddens2)

#训练和测试
num_epochs,lr,batch_size=10,0.5,256
loss=nn.CrossEntropyLoss(reduction="none")
train_iter,test_iter=d2l.load_data_fashion_mnist(batch_size)
trainer=torch.optim.SGD(net.parameters(),lr=lr)
d2l.train_ch3(net,train_iter,test_iter,loss,num_epochs,trainer)
```

房价预测

Python | 复制代码

```python
import hashlib
import os
import tarfile
import zipfile
import requests

DATA_HUB=dict()
DATA_URL='http://d2l-data.s3-accelerate.amazonaws.com/'
def download(name,cache_dir=os.path.join('..','data')):
    assert name in DATA_HUB,f"{name} not exit in {DATA_HUB}"
    url,sha1_hash=DATA_HUB[name]
    os.makedirs(cache_dir,exist_ok=True)
    fname=os.path.join(cache_dir,url.split('/')[-1])
    if os.path.exists(fname):
        sha1=hashlib.sha1()
        with open(fname,'rb') as f:
            while True:
                data=f.read(1048576)
                if not data:
                    break
                sha1.update(data)
            if sha1.hexdigest()==sha1_hash:
                return fname
    print(f'from {url} downloading {name}')
    r=requests.get(url,stream=True,verify=True)
    with open(fname,'wb') as f:
        f.write(r.content)
    return fname
def download_extract(name,folder=None):
    fname=download(name)
    base_dir=os.path.dirname(fname)
    data_dir,ext=os.path.splitext(fname)
    if ext=='.zip':
        fp=zipfile.ZipFile(fname,'r')
    elif ext in ('.tar','.gz'):
        fp=tarfile.open(fname,'r')
    else:
        assert  False,'only zip/tar file can be extracted'
    fp.extractall(base_dir)
    return os.path.join(base_dir,folder) if folder else data_dir
def download_all():
    for name in DATA_HUB:
        download(name)

%matplotlib inline
```

```
46  import numpy as np
47  import pandas as pd
48  import torch
49  from torch import nn
50  from d2l import torch as d2l
51
52  DATA_HUB['kaggle_house_train'] = (DATA_URL +
    'kaggle_house_pred_train.csv',
53  '585e9cc93e70b39160e7921475f9bcd7d31219ce')
54  DATA_HUB['kaggle_house_test'] = (DATA_URL +
    'kaggle_house_pred_test.csv',
55  'fa19780a7b011d9b009e8bff8e99922a8ee2eb90')
56
57  train_data=pd.read_csv(download('kaggle_house_train'))
58  test_data=pd.read_csv(download('kaggle_house_test'))
59
60  print(train_data.shape)
61  print(test_data.shape)
62
63  print(train_data.iloc[0:4,[0,1,2,3,-3,-2,-1]])
64  all_features=pd.concat((train_data.iloc[:,1:-1],test_data.iloc[:,1:]))
65
66  #数据预处理
67  #根据训练数据计算均值和标准差，在标准化数据之后，将缺失值设置为0
68  numeric_features=all_features.dtypes[all_features.dtypes!='object'].inde
    x
69  all_features[numeric_features]=all_features[numeric_features].apply(
70  lambda x:(x-x.mean())/x.std())
71  all_features[numeric_features]=all_features[numeric_features].fillna(0)
72
73  all_features=pd.get_dummies(all_features,dummy_na=True)
74  all_features.shape
75
76  n_train=train_data.shape[0]
77  train_features=torch.tensor(all_features[:n_train].values,dtype=torch.fl
    oat32)
78  test_features=torch.tensor(all_features[n_train:].values,dtype=torch.flo
    at32)
79  train_labels=torch.tensor(train_data.SalePrice.values.reshape(-1,1),dtyp
    e=torch.float32)
80
81  #训练
82  loss=nn.MSELoss()
83  in_features=train_features.shape[1]
84  def get_net():
85      net=nn.Sequential(nn.Linear(in_features,1))
86      return net
87
```

```
88  def log_rmse(net,features,labels):
89      clipped_preds=torch.clamp(net(features),1,float('inf'))
90      rmse=torch.sqrt(loss(torch.log(clipped_preds),torch.log(labels)))
91      return rmse.item()
92
93  #使用Adam优化器开展训练
94  def
    train(net,train_features,train_labels,test_features,test_labels,num_epoc
    hs,learning_rate,weight_decay,batch_size):
95      train_ls,test_ls=[],[]
96      train_iter=d2l.load_array((train_features,train_labels),batch_size)
97
     optimizer=torch.optim.Adam(net.parameters(),lr=learning_rate,weight_dec
    ay=weight_decay)
98      for epoch in range(num_epochs):
99          for X,y in train_iter:
100             optimizer.zero_grad()
101             l=loss(net(X),y)
102             l.backward()
103             optimizer.step()
104         train_ls.append(log_rmse(net,train_features,train_labels))
105         if test_labels is not None:
106             test_ls.append(log_rmse(net,test_features,test_labels))
107     return train_ls,test_ls
108
109 #K折交叉验证
110 def get_k_fold_data(k,i,X,y):
111     assert k>1
112     fold_size=X.shape[0]//k
113     X_train,y_train=None,None
114     for j in range(k):
115         idx=slice(j*fold_size,(j+1)*fold_size)
116         X_part,y_part=X[idx,:],y[idx]
117         if j==i:
118             X_valid,y_valid=X_part,y_part
119         elif X_train is None:
120             X_train,y_train=X_part,y_part
121         else:
122             X_train=torch.cat([X_train,X_part],0)
123             y_train=torch.cat([y_train,y_part],0)
124     return X_train,y_train,X_valid,y_valid
125
126 def
    k_fold(k,X_train,y_train,num_epochs,learning_rate,weight_decay,batch_siz
    e):
127     train_l_sum,valid_l_sum=0,0
128     for i in range(k):
129         data=get_k_fold_data(k,i,X_train,y_train)
```

```
        net=get_net()

 train_ls,valid_ls=train(net,*data,num_epochs,learning_rate,weight_decay
,batch_size)
        train_l_sum+=train_ls[-1]
        valid_l_sum+=valid_ls[-1]
        if i==0:
            d2l.plot(list(range(1,num_epochs+1)),
[train_ls,valid_ls],xlabel='epoch',
                     ylabel='rmse',xlim=[1,num_epochs],legend=
['trian','valid'],
                     yscale='log')
        print(f'折{i+1},训练log rmse{float(train_ls[-1]):f}, '
                  f'验证log rmse{float(valid_ls[-1]):f}')
    return train_l_sum/k,valid_l_sum/k

#模型选择
k,num_epochs,lr,weight_decay,batch_size=5,100,5,0,64
train_l,valid_l=k_fold(k,train_features,train_labels,num_epochs,lr,weigh
t_decay,batch_size)
print(f'{k}-折验证: 平均训练log rmse: {float(train_l):f}, 'f'平均验证log
rmse: {float(valid_l):f}')

def
train_and_pred(train_features,test_features,train_labels,test_data,num_e
pochs,lr,weight_decay,batch_size):
    net=get_net()

 train_ls,_=train(net,train_features,train_labels,None,None,num_epochs,l
r,weight_decay,batch_size)
    d2l.plot(np.arange(1, num_epochs + 1), [train_ls], xlabel='epoch',
ylabel='log rmse', xlim=[1, num_epochs], yscale='log')
    print(f'训练log rmse: {float(train_ls[-1]):f}')
    preds=net(test_features).detach().numpy()
    test_data["SalesPrice"]=pd.Series(preds.reshape(1,-1)[0])

 submission=pd.concat([test_data["Id"],test_data["SalesPrice"]],axis=1)
    submission.to_csv('submission.csv',index=False)

train_and_pred(train_features,test_features,train_labels,test_data,num_e
pochs,lr,weight_decay,batch_size)
```

深度学习计算

Python | 复制代码

```python
import torch
from torch import nn
from torch.nn import functional as F

net=nn.Sequential(nn.Linear(20,256),nn.ReLU(),nn.Linear(256,10))

X=torch.rand(2,20)
net(X)

#自定义块
class MLP(nn.Module):
    def __init__(self):
        #通过super().init__()调用父类的__init__函数，省去重复编写模板代码的痛苦
        super().__init__()
        self.hidden=nn.Linear(20,256)
        self.out=nn.Linear(256,10)
    def forward(self,X):
        return self.out(F.relu(self.hidden(X)))
net=MLP()
net(X)

#顺序块
class MySequential(nn.Module):
    def __init__(self,*args):
        super().__init__()
        for idx,module in enumerate(args):
            self._modules[str(idx)]=module
    def forward(self,X):
        for block in self._modules.values():
            X=block(X)
        return X

net=MySequential(nn.Linear(20,256),nn.ReLU(),nn.Linear(256,10))
net(X)

class FixedHiddenMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.rand_weight=torch.rand((20,20),requires_grad=False)
        self.linear=nn.Linear(20,20)
    def forward(self,X):
        X=self.linear(X)
        X=F.relu(torch.mm(X,self.rand_weight)+1)
        X=self.linear(X)
        while X.abs().sum()>1:
```

```
            X/=2
        return X.sum()

net=FixedHiddenMLP()
net(X)

#嵌套块
class NestMLP(nn.Module):
    def __init__(self):
        super().__init__()

 self.net=nn.Sequential(nn.Linear(20,64),nn.ReLU(),nn.Linear(64,32),nn.ReLU())
        self.linear=nn.Linear(32,16)
    def forward(self,X):
        return self.linear(self.net(X))
chimera=nn.Sequential(NestMLP(),nn.Linear(16,20),FixedHiddenMLP())
chimera(X)

import torch
from torch import nn

net=nn.Sequential(nn.Linear(4,8),nn.ReLU(),nn.Linear(8,1))
X=torch.rand(size=(2,4))
net(X)

#参数访问
print(net[2].state_dict())

print(type(net[2].bias))
print(net[2].bias)
print(net[2].bias.data)

net[2].weight.grad==None

print(*[(name,param.shape) for name,param in net[0].named_parameters()])
print(*[(name,param.shape) for name,param in net.named_parameters()])

net.state_dict()['2.bias'].data

#从嵌套块收集参数
def block1():
    return
nn.Sequential(nn.Linear(4,8),nn.ReLU(),nn.Linear(8,4),nn.ReLU())
def block2():
    net=nn.Sequential()
    for i in range(4):
        net.add_module(f'block{i}',block1())
```

```
    return net
rgnet=nn.Sequential(block2(),nn.Linear(4,1))
rgnet(X)

print(rgnet)
rgnet[0][1][0].bias.data

#参数初始化
def init_normal(m):
    if type(m)==nn.Linear:
        nn.init.normal_(m.weight,mean=0,std=0.01)
        nn.init.zeros_(m.bias)
net.apply(init_normal)
net[0].weight.data[0],net[0].bias.data[0]

def init_constant(m):
    if type(m)==nn.Linear:
        nn.init.constant_(m.weight,1)
        nn.init.zeros_(m.bias)
net.apply(init_constant)
net[0].weight.data[0],net[0].bias.data[0]

def xavier(m):
    if type(m)==nn.Linear:
        nn.init.xavier_uniform_(m.weight)
def init_42(m):
    if type(m)==nn.Linear:
        nn.init.constant_(m.weight,42)
net[0].apply(xavier)
net[2].apply(init_42)
print(net[0].weight.data[0])
print(net[2].weight.data)

#自定义初始化
def my_init(m):
    if type(m)==nn.Linear:
        print("Init",*[(name,param.shape) for name,param in
m.named_parameters()][0])
        nn.init.uniform_(m.weight,-10,10)
        m.weight.data*=m.weight.data.abs()>=5
net.apply(my_init)
net[0].weight[:2]

#可以直接设置参数
net[0].weight.data[:]+=1
net[0].weight.data[0,0]=42
net[0].weight.data[0]

```

```
#参数绑定
#有时候希望在多层间共享参数：可以定义一个稠密层，然后使用它的参数来设置另一个层的参数
shared=nn.Linear(8,8)
net=nn.Sequential(nn.Linear(4,8),nn.ReLU(),shared,nn.ReLU(),shared,nn.ReLU(),nn.Linear(8,1))
net(X)
print(net[2].weight.data[0]==net[4].weight.data[0])
net[2].weight.data[0,0]=100
print(net[2].weight.data[0]==net[4].weight.data[0])

#自定义层
#不带参数的层
import torch
import torch.nn.functional as F
from torch import nn

class CenterdLayer(nn.Module):
    def __init__(self):
        super().__init__()
    def forward(self,X):
        return X-X.mean()

layer=CenterdLayer()
layer(torch.FloatTensor([1,2,3,4,5]))

net=nn.Sequential(nn.Linear(8,128),CenterdLayer())

Y=net(torch.rand(4,8))
Y.mean()

#带参数的层
class MyLinear(nn.Module):
    def __init__(self,in_units,units):
        super().__init__()
        self.weight=nn.Parameter(torch.randn(in_units,units))
        self.bias=nn.Parameter(torch.randn(units,))
    def forward(self,X):
        linear=torch.matmul(X,self.weight.data)+self.bias.data
        return F.relu(linear)

linear=MyLinear(5,3)
linear.weight

linear(torch.randn(2,5))

net=nn.Sequential(MyLinear(64,8),MyLinear(8,1))
net(torch.rand(2,64))
```

读写文件

Python | 复制代码

```python
#加载和保存张量
import torch
from torch import nn
from torch.nn import functional as F

x=torch.arange(4)
torch.save(x,'x-file')

x2=torch.load('x-file')
x2

y=torch.zeros(4)
torch.save([x,y],'x-files')
x2,y2=torch.load('x-files')
(x2,y2)

#加载和保存模型参数
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden=nn.Linear(20,256)
        self.output=nn.Linear(256,10)
    def forward(self,x):
        return self.output(F.relu(self.hidden(x)))
net=MLP()
X=torch.randn(size=(2,20))
Y=net(X)

torch.save(net.state_dict(),'mlp.params')

#恢复模型
clone=MLP()
clone.load_state_dict(torch.load('mlp.params'))
clone.eval()

Y_clone=clone(X)
Y_clone==Y
```

2.结果描述

今天把多层感知机和深度学习计算部分大致过了一遍，虽然代码都动手码了一遍，但还是有很多没完全搞懂的地方。明天要么把前面的基础再过一遍，要么用C++开始简单实现一个多层感知机。