4_housing_prices_prediction

吴清柳

July 24, 2023

1 Boston housing price predictino on Boston housing dataset

[34]:
```
import torch
import torch.nn as nn
from torch.optim.lr_scheduler import StepLR

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import requests
```

该数据集有 22 行的前言, 且没有逗号分割, 且 14 列数据每一行被分割为两行. 导入前需要对应进行预处理.

[35]:
```
# Load the Boston housing dataset
response = requests.get('http://lib.stat.cmu.edu/datasets/boston')
```

[36]:
```
data=response.text

# Remove the meta-information at the beginning
data=data.split('\n')[22:]

# The data rows are splited over two lines. Join them together
joined_data = []
for i in range(0,len(data)-1,2):
    row=data[i].split()+data[i+1].split()
    joined_data.append(row)

print(joined_data[:2],'...')
# convert to numpy array
data=np.array(joined_data).astype(float)

# convert the data to a DataFrame
columns=['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT','
df=pd.DataFrame(data,columns=columns)
print(df.head())
```

```
[['0.00632', '18.00', '2.310', '0', '0.5380', '6.5750', '65.20', '4.0900', '1',
'296.0', '15.30', '396.90', '4.98', '24.00'], ['0.02731', '0.00', '7.070', '0',
'0.4690', '6.4210', '78.90', '4.9671', '2', '242.0', '17.80', '396.90', '9.14',
'21.60']] …
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

	PTRATIO	B	LSTAT	MEDV
0	15.3	396.90	4.98	24.0
1	17.8	396.90	9.14	21.6
2	17.8	392.83	4.03	34.7
3	18.7	394.63	2.94	33.4
4	18.7	396.90	5.33	36.2

[37]:
```
# separate features and target
X=df.drop('MEDV',axis=1).values
y=df['MEDV'].values

# standardize the features
scaler=StandardScaler()
X=scaler.fit_transform(X)

# split data into training and testing dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪2,random_state=42)
```

[38]:
```
# convert to pytorch tensors
X_train=torch.FloatTensor(X_train)
y_train=torch.FloatTensor(y_train).view(-1,1)
X_test=torch.FloatTensor(X_test)
y_test=torch.FloatTensor(y_test).view(-1,1)
```

[39]:
```
# boston housing model
class BostonHousingModel(nn.Module):
    def __init__(self):
        super(BostonHousingModel,self).__init__()
        self.layer1=nn.Linear(13,64)
        self.layer2=nn.Linear(64,64)
        self.layer3=nn.Linear(64,1)

    def forward(self,x):
        x=torch.relu(self.layer1(x))
        x=torch.relu(self.layer2(x))
```

```
        x=self.layer3(x)
        return x
```

[40]:
```
# initialize the model, define loss function and optimizer
model=BostonHousingModel()
criterion=nn.MSELoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.001)
```

[41]:
```
# training the model
for epoch in range(500):
    # forward pass
    outputs=model(X_train)
    loss=criterion(outputs,y_train)

    # backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if(epoch+1)%50==0:
        print(f'Epoch {epoch+1}, Loss: {loss.item()}')

torch.save(model.state_dict(), 'best_model_old.pt')
```

```
Epoch 50, Loss: 381.8599548339844
Epoch 100, Loss: 57.943546295166016
Epoch 150, Loss: 25.20885467529297
Epoch 200, Loss: 19.815107345581055
Epoch 250, Loss: 16.540985107421875
Epoch 300, Loss: 14.289535522460938
Epoch 350, Loss: 12.771340370178223
Epoch 400, Loss: 11.739974975585938
Epoch 450, Loss: 10.954638481140137
Epoch 500, Loss: 10.293461799621582
```

[42]:
```
# test the model
model.eval()
with torch.no_grad():
    y_pred=model(X_test)

print('Test loss: ', criterion(y_pred, y_test).item())
```

```
Test loss:  12.326370239257812
```

[84]:
```
# Use dropout, learning rate scheduling, and early stopping to optimize the
# model
class OptiBostonHousingModel(nn.Module):
    def __init__(self):
```

```
        super(OptiBostonHousingModel,self).__init__()
        self.layer1=nn.Linear(13,128)
        self.layer2=nn.Linear(128,64)
        self.dropout=nn.Dropout(0.5)
        self.layer3=nn.Linear(64,1)

    def forward(self,x):
        x=torch.relu(self.layer1(x))
        x=self.dropout(x)
        x=torch.relu(self.layer2(x))
        x=self.dropout(x)
        x=self.layer3(x)
        return x

model=OptiBostonHousingModel()
critetion=nn.MSELoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.09)
# reduce learning rate every 100 steps
scheduler=StepLR(optimizer,step_size=100,gamma=0.1)
```

[50]:
```
# define lists to store losses of training and validation
train_losses = []
valid_losses = []
best_valid_loss = float("inf")

# split validation from training
val_ratio = 0.1
val_len = int(len(X_train) * val_ratio)
train_len = len(X_train) - val_len
X_train, X_val = torch.utils.data.random_split(X_train, [train_len, val_len])
y_train, y_val = torch.utils.data.random_split(y_train, [train_len, val_len])
```

[51]:
```
X_train=X_train.dataset
X_val=X_val.dataset
y_train=y_train.dataset
y_val=y_val.dataset
```

[89]:
```
# Training new model
for epoch in range(500):
    model.train()
    optimizer.zero_grad()
    train_preds = model(X_train)
    train_loss = criterion(train_preds, y_train)
    train_loss.backward()
    optimizer.step()

    model.eval()
```

```
    with torch.no_grad():
        valid_preds = model(X_val)
        valid_loss = criterion(valid_preds, y_val)

    train_losses.append(train_loss.item())
    valid_losses.append(valid_loss.item())

    if valid_loss.item() < best_valid_loss:
        best_valid_loss = valid_loss.item()
        torch.save(model.state_dict(), "best_model.pt")

    scheduler.step()

    if (epoch + 1) % 50 == 0:
        print(
            f"Epoch {epoch+1}, Train Loss: {train_loss.item()}, Validation Loss:
 ↪ {valid_loss.item()}"
        )

model.load_state_dict(torch.load('best_model.pt'))

model.eval()
with torch.no_grad():
    y_pred=model(X_test)

print("Test Loss: ", criterion(y_pred, y_test).item())
```

```
Epoch 50, Train Loss: 27.11713981628418, Validation Loss: 7.157723903656006
Epoch 100, Train Loss: 32.16069412231445, Validation Loss: 7.157723903656006
Epoch 150, Train Loss: 31.317283630371094, Validation Loss: 7.157723903656006
Epoch 200, Train Loss: 31.56045913696289, Validation Loss: 7.157723903656006
Epoch 250, Train Loss: 28.05300521850586, Validation Loss: 7.157723903656006
Epoch 300, Train Loss: 33.324302673339844, Validation Loss: 7.157723903656006
Epoch 350, Train Loss: 28.748680114746094, Validation Loss: 7.157723903656006
Epoch 400, Train Loss: 27.83043098449707, Validation Loss: 7.157723903656006
Epoch 450, Train Loss: 33.288204193115234, Validation Loss: 7.157723903656006
Epoch 500, Train Loss: 28.712465286254883, Validation Loss: 7.157723903656006
Test Loss:  12.081153869628906
```

1.1 模型比较

对于回归问题 (Regression problem), 使用平均平方差 (Mean Squared Error, MSE) 作为 Loss, 并使用此来比较两个模型的拟合水平. 可以看到两个模型的 Loss 相差不大.

[90]:
```
model_old=BostonHousingModel()
model_old.load_state_dict(torch.load('best_model_old.pt'))

model_new=OptiBostonHousingModel()
```

```
model_new.load_state_dict(torch.load('best_model.pt'))

model_old.eval()
model_new.eval()

with torch.no_grad():
    y_pred_old=model_old(X_test)
    y_pred_new=model_new(X_test)

print(f'Test Loss of the old model: {criterion(y_pred_old, y_test).item()}')
print(f'Test Loss of the new model: {criterion(y_pred_new, y_test).item()}')
```

```
Test Loss of the old model: 12.326370239257812
Test Loss of the new model: 12.081153869628906
```