Py实现共享单车数量预测

#导入需要使用的库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import torch

from torch.autograd import Variable

import torch.optim as optim

#读取数据

data\_path = 'Bike-Sharing-Dataset/hour.csv'

rides = pd.read\_csv(data\_path)

rides.head()

#我们取出最后一列的前50条记录来进行预测

counts = rides['cnt'][:50]

x = np.arange(len(counts)) #获得变量x，它是1，2，……，50

y = np.array(counts) # 将counts转成预测变量（标签）：y

# 绘制一个图形，展示曲线长的样子

plt.figure(figsize = (10, 7))

fig, ax = plt.subplots()

ax.plot(x, y, 'o-')

plt.xlabel('X')

plt.ylabel('Y')

#对于类型变量的特殊处理

# season=1,2,3,4, weathersi=1,2,3, mnth= 1,2,...,12, hr=0,1, ...,23, weekday=0,1,...,6

# 经过下面的处理后，将会多出若干特征，例如，对于season变量就会有 season\_1, season\_2, season\_3, season\_4

# 这四种不同的特征。

dummy\_fields = ['season', 'weathersit', 'mnth', 'hr', 'weekday']

for each in dummy\_fields:

#利用pandas对象，我们可以很方便地将一个类型变量属性进行one-hot编码，变成多个属性

dummies = pd.get\_dummies(rides[each], prefix=each, drop\_first=False)

rides = pd.concat([rides, dummies], axis=1)

# 把原有的类型变量对应的特征去掉，将一些不相关的特征去掉

fields\_to\_drop = ['instant', 'dteday', 'season', 'weathersit',

'weekday', 'atemp', 'mnth', 'workingday', 'hr']

data = rides.drop(fields\_to\_drop, axis=1)

data.head()

#接下来对数值类型的变量进行标准化处理，因为每个数值型变量都是相互独立的，所以它们的数值绝对大小与问题本身没有关系，为了消除数值大小的差异，可以对每一个数值型变量进行标准化处理，也就是让其数值都围绕着0左右波动。

作者：carmanzzz

链接：https://www.jianshu.com/p/696c340ee51e

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

# 调整所有的特征，标准化处理

quant\_features = [ 'temp', 'hum', 'windspeed']

# 我们将每一个变量的均值和方差都存储到scaled\_features变量中。

scaled\_features = {}

for each in quant\_features:

mean, std = data[each].mean(), data[each].std()

scaled\_features[each] = [mean, std]

data.loc[:, each] = (data[each] - mean)/std

对数据集进行分割：

# 将所有的数据集分为测试集和训练集，我们以后21天数据一共21\*24个数据点作为测试集，其它是训练集

test\_data = data[-21\*24:]

train\_data = data[:-21\*24]

# 将我们的数据列分为特征列和目标列

#目标列

target\_fields = ['cnt','casual', 'registered']

features, targets = train\_data.drop(target\_fields, axis=1), train\_data[target\_fields]

test\_features, test\_targets = test\_data.drop(target\_fields, axis=1), test\_data[target\_fields]

# 将数据从pandas dataframe转换为numpy

X = features.values

Y = targets['cnt'].values

Y = Y.astype(float)

Y = np.reshape(Y, [len(Y),1])

losses = []

定义神经网络架构，features.shape[1]个输入层单元，10个隐含层，1个输出层

input\_size = features.shape[1]

hidden\_size = 10

output\_size = 1

batch\_size = 128

neu = torch.nn.Sequential(

torch.nn.Linear(input\_size, hidden\_size),

torch.nn.Sigmoid(),

torch.nn.Linear(hidden\_size, output\_size),

)

cost = torch.nn.MSELoss()

optimizer = torch.optim.SGD(neu.parameters(), lr = 0.01)

# 神经网络训练循环

losses = []

for i in range(1000):

# 每128个样本点被划分为一个撮，在循环的时候一批一批地读取

batch\_loss = []

# start和end分别是提取一个batch数据的起始和终止下标

for start in range(0, len(X), batch\_size):

end = start + batch\_size if start + batch\_size < len(X) else len(X)

xx = Variable(torch.FloatTensor(X[start:end]))

yy = Variable(torch.FloatTensor(Y[start:end]))

predict = neu(xx)

loss = cost(predict, yy)

zero\_grad()

loss.backward()

optimizer\_step(0.01)

batch\_loss.append(loss.data.numpy())

# 每隔100步输出一下损失值（loss）

if i % 100==0:

losses.append(np.mean(batch\_loss))

print(i, np.mean(batch\_loss))

# 打印输出损失值

plt.plot(np.arange(len(losses))\*100,losses)

plt.xlabel('epoch')

plt.ylabel('MSE')

# 用训练好的神经网络在测试集上进行预测

targets = test\_targets['cnt'] #读取测试集的cnt数值

targets = targets.values.reshape([len(targets),1]) #将数据转换成合适的tensor形式

targets = targets.astype(float) #保证数据为实数

# 将属性和预测变量包裹在Variable型变量中

x = Variable(torch.FloatTensor(test\_features.values))

y = Variable(torch.FloatTensor(targets))

# 用神经网络进行预测

predict = neu(x)

predict = predict.data.numpy()

# 将后21天的预测数据与真实数据画在一起并比较

# 横坐标轴是不同的日期，纵坐标轴是预测或者真实数据的值

fig, ax = plt.subplots(figsize = (10, 7))

mean, std = scaled\_features['cnt']

ax.plot(predict \* std + mean, label='Prediction')

ax.plot(targets \* std + mean, label='Data')

ax.legend()

ax.set\_xlabel('Date-time')

ax.set\_ylabel('Counts')

# 对横坐标轴进行标注

dates = pd.to\_datetime(rides.loc[test\_data.index]['dteday'])

dates = dates.apply(lambda d: d.strftime('%b %d'))

ax.set\_xticks(np.arange(len(dates))[12::24])

\_ = ax.set\_xticklabels(dates[12::24], rotation=45)