团队编号:2002066 二仙桥老大爷

LSTM和Prophet以及bp神经网络的部署手册

目录

[一、Prophet的部署及使用 1](#_Toc69905870)

[（一）电脑环境 1](#_Toc69905871)

[（二）anaconda配置环境 1](#_Toc69905872)

[（三）prophet的配置 1](#_Toc69905873)

[（四）常见错误 2](#_Toc69905874)

[（五）、关键代码解释 2](#_Toc69905875)

[二、LSTM的部署及使用 3](#_Toc69905876)

[（一）在Prophet环境中，安装tensorflow 3](#_Toc69905877)

[（二）关键代码解释 3](#_Toc69905878)

[三、Bp神经网络的部署及使用 6](#_Toc69905879)

[（一）在同一个python环境中，安装keras 6](#_Toc69905880)

[(二) 关键代码解释 6](#_Toc69905881)

# 一、Prophet的部署及使用

## （一）电脑环境

（1）windows10上部署

Windows 10 x64

Anaconda3-2019.07-Windows-x86\_64.exe

pycharm-professional-2019.1.2.exe

（2）Centos7.4上部署

Centos7.4

Anaconda3-4.2.0-Linux-x86\_64

## （二）anaconda配置环境

（1）anaconda创建环境

conda create --name prophet python=3.6.1

python=3.6.1是指定的python版本，在anaconda中创建了一个名为“prophet”的环境（名字可以自己取）

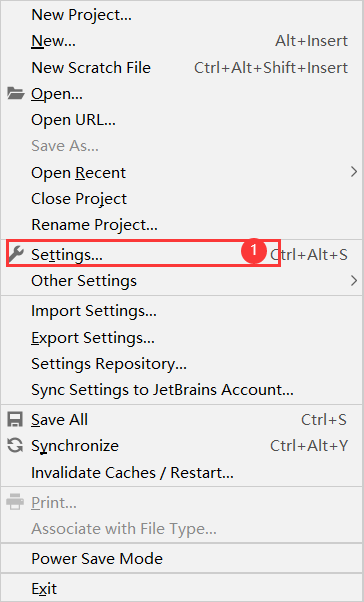
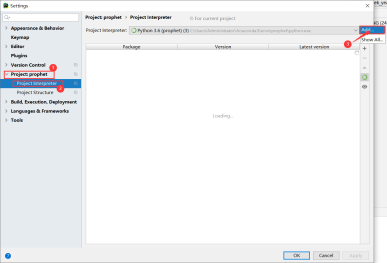
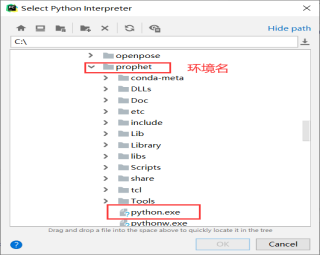
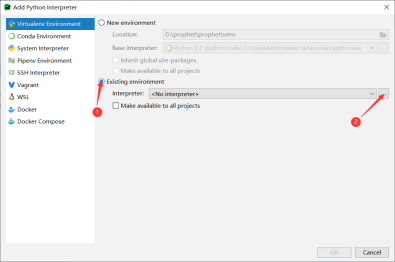
（2）环境切换

conda activate prophet

prophet为环境名

（3）pycharm添加prophet环境（此步骤只在windows10上配置）

pycharm工作台→File→Settings→Project:prophet→Project Interpreter→Add→Existing environment→…→C:\Users\Administrator\Anaconda3\envs\prophet\python.exe→都点【确定】

## （三）prophet的配置

（1）安装C++编译器

conda install libpython m2w64-toolchain -c msys2

输入g++,若报不是内部或外部命令，也不是可运行的程序，则未安装上。若报fatal error: no input files则表示安装上。

（2）安装依赖包pystan

Prophet依赖pystan包所以也需要安装pystan

pip install pystan

（3）安装Prophet

不要用 pip install fbprophet

下载prophet最新源码。

或者git clone https://github.com/facebookincubator/prophet

解压到指定目录，例如D:\Anaconda3\prophet

CMD:

cd D:\Anaconda3\prophet\python （下载的源码路径，这里以windows10为例）

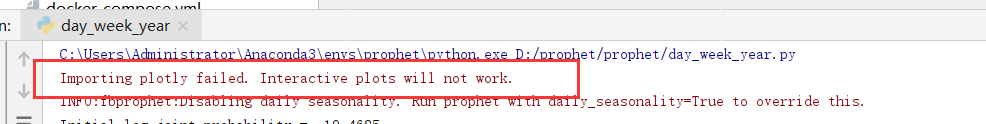
里面有setup.py脚本，直接运行一下命令：

pip install -e .

至此，prophet安装完成。

## （四）常见错误

（1）错误一：

解决方法

conda install plotly

## （五）、关键代码解释

（1）导入数据

1、sql 命令

sql\_cmd = "SELECT \* FROM yearall"

2、用DBAPI构建数据库链接engine

con=pymysql.connect(host="localhost",port=3306,user="root", password="123456", database="test", charset='utf8', use\_unicode=True)

df = pd.read\_sql(sql\_cmd, con)

df.head()

（2）拟合模型

m = Prophet(changepoint\_prior\_scale=0.9,interval\_width=0.9,growth='linear',changepoint\_range=1)

m.fit(df)

（3）构建待预测日期数据框，periods = 30代表除历史数据的日期外再往后推 30天

future = m.make\_future\_dataframe(periods=30)

future.tail()

（4）预测数据集

forecast = m.predict(future)

#forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail()

（5）展示预测结果

def yearplot\_x(ax=None):

ax=ax

year\_x = ax["ds"].values

return year\_x

def yearplot\_y(ax=None):

ax=ax

year\_y = ax["trend"].values

print(year\_y)

return year\_y

re\_x=yearplot\_x(forecast)

re\_y=yearplot\_y(forecast)

m.plot(forecast);

（6）预测的成分分析绘图，展示预测中的趋势、周效应和年度效应

m.plot\_components(forecast);

# 二、LSTM的部署及使用

## （一）在Prophet环境中，安装tensorflow,命令如下

pip install tensorflow

## （二）关键代码解释

（1）导入数据

1、sql 命令

sql\_cmd = "SELECT \* FROM yearall"

2、用DBAPI构建数据库链接engine

con = pymysql.connect(host="localhost",port=3306, user="root", password="123456", database="test", charset='utf8', use\_unicode=True)

df = pd.read\_sql(sql\_cmd, con)

df.head()

data=np.array(df['y']) #获取客流量序列

（2）以折线图展示data

plt.figure()

plt.plot(data)

plt.show()

normalize\_data=(data-np.mean(data))/np.std(data) #标准化

normalize\_data=normalize\_data[:,np.newaxis] #增加维度

（3）生成训练集，并设置常量

time\_step=20 #时间步

rnn\_unit=10 #hidden layer units

batch\_size=60 #每一批次训练多少个样例

input\_size=1 #输入层维度

output\_size=1 #输出层维度

lr=0.0006 #学习率

train\_x,train\_y=[],[] #训练集

for i in range(len(normalize\_data)-time\_step-1):

x=normalize\_data[i:i+time\_step]

y=normalize\_data[i+1:i+time\_step+1]

train\_x.append(x.tolist()) #将数组转化成列表

train\_y.append(y.tolist())

（4）定义每批次输神经网络的大小

X=tf.placeholder(tf.float32, [None,time\_step,input\_size]) #每批次输入网络的tensor/定义placeholder

Y=tf.placeholder(tf.float32, [None,time\_step,output\_size]) #每批次tensor对应的标签

（5）编写输入层、输出层权重、偏置

weights={

'in':tf.Variable(tf.random\_normal([input\_size,rnn\_unit])),

'out':tf.Variable(tf.random\_normal([rnn\_unit,1]))

}

biases={

'in':tf.Variable(tf.constant(0.1,shape=[rnn\_unit,])),

'out':tf.Variable(tf.constant(0.1,shape=[1,]))

}

（6）定义神经网络变量

def lstm(batch): #参数：输入网络批次数目

w\_in=weights['in']

b\_in=biases['in']

input=tf.reshape(X,[-1,input\_size]) #需要将tensor转成2维进行计算，计算后的结果作为隐藏层的输入

input\_rnn=tf.matmul(input,w\_in)+b\_in #表示矩阵乘法

input\_rnn=tf.reshape(input\_rnn,[-1,time\_step,rnn\_unit]) #将tensor转成3维，作为lstm cell的输入

cell=tf.nn.rnn\_cell.BasicLSTMCell(rnn\_unit) #定义单个基本的LSTM单元

init\_state=cell.zero\_state(batch,dtype=tf.float32) #这个函数用于返回全0的state tensor

#dynamic\_rnn 用于创建由RNNCell细胞指定的循环神经网络，对inputs进行动态展开

#output\_rnn是记录lstm每个输出节点的结果，final\_states是最后一个cell的结果

output\_rnn,final\_states=tf.nn.dynamic\_rnn(cell, input\_rnn,initial\_state=init\_state, dtype=tf.float32)

#函数的作用是将tensor变换为参数shape的形式。

output=tf.reshape(output\_rnn,[-1,rnn\_unit])

w\_out=weights['out']

b\_out=biases['out']

pred=tf.matmul(output,w\_out)+b\_out #表示矩阵乘法

return pred,final\_states

（7）训练模型

def train\_lstm():

global batch\_size

pred,\_=lstm(batch\_size) #调用的构建的lstm变量

#损失函数 平均平方误差(MSE)

loss=tf.reduce\_mean(tf.square(tf.reshape(pred,[-1])-tf.reshape(Y, [-1])))

#实现梯度下降算法的优化器，优化损失函数

train\_op=tf.train.AdamOptimizer(lr).minimize(loss)

#保存和恢复模型的方法；方法返回checkpoint文件的路径。可以直接传给restore() 进行调用

saver=tf.train.Saver(tf.global\_variables())

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

#重复训练10000次

for i in range(10000):

step=0

start=0

end=start+batch\_size

while(end<len(train\_x)):

\_,loss\_=sess.run([train\_op,loss],feed\_dict={X:train\_x[start:end],Y:train\_y[start:end]})

start+=batch\_size

end=start+batch\_size

#每10步保存一次参数

if step%10==0:

print(i,step,loss\_)

print("保存模型：",saver.save(sess,'stock.model'))

step+=1

train\_lstm()

（8）预测模型

def prediction():

pred,\_=lstm(1) #预测时只输入[1,time\_step,input\_size]的测试数据

saver=tf.train.Saver(tf.global\_variables())

with tf.Session() as sess:

#参数恢复

module\_file = tf.train.latest\_checkpoint(base\_path+'module2/')

saver.restore(sess, module\_file)

#取训练集最后一行为测试样本。shape=[1,time\_step,input\_size]

prev\_seq=train\_x[-1]

predict=[]

#得到之后100个预测结果

for i in range(100):

next\_seq=sess.run(pred,feed\_dict={X:[prev\_seq]})

predict.append(next\_seq[-1])

#每次得到最后一个时间步的预测结果，与之前的数据加在一起，形成新的测试样本

prev\_seq=np.vstack((prev\_seq[1:],next\_seq[-1]))

#以折线图表示结果

plt.figure()

plt.plot(list(range(len(normalize\_data))), normalize\_data, color='b')

plt.plot(list(range(len(normalize\_data), len(normalize\_data) + len(predict))), predict, color='r')

prediction()

# 三、Bp神经网络的部署及使用

## （一）在以上同一个python环境中，安装keras,命令如下

pip install keras

## (二) 关键代码解释

（1）加载数据

# 转成DataFrame格式方便数据处理

x\_train\_pd = pd.DataFrame(x\_train)

y\_train\_pd = pd.DataFrame(y\_train)

x\_valid\_pd = pd.DataFrame(x\_valid)

y\_valid\_pd = pd.DataFrame(y\_valid)

print(x\_train\_pd.head(5))

print('-------------------')

print(y\_train\_pd.head(5))

(2) 数据归一化

# 训练集归一化

min\_max\_scaler = MinMaxScaler()

min\_max\_scaler.fit(x\_train\_pd)

x\_train = min\_max\_scaler.transform(x\_train\_pd)

min\_max\_scaler.fit(y\_train\_pd)

y\_train = min\_max\_scaler.transform(y\_train\_pd)

# 验证集归一化

min\_max\_scaler.fit(x\_valid\_pd)

x\_valid = min\_max\_scaler.transform(x\_valid\_pd)

min\_max\_scaler.fit(y\_valid\_pd)

y\_valid = min\_max\_scaler.transform(y\_valid\_pd)

(3) 训练模型

# 单CPU or GPU版本，若有GPU则自动切换

model = Sequential() # 初始化，很重要！

model.add(Dense(units = 10, # 输出大小

activation='relu', # 激励函数

input\_shape=(x\_train\_pd.shape[1],) # 输入大小, 也就是列的大小

)

)

model.add(Dropout(0.2)) # 丢弃神经元链接概率

model.add(Dense(units = 15,

# kernel\_regularizer=regularizers.l2(0.01), # 施加在权重上的正则项

# activity\_regularizer=regularizers.l1(0.01), # 施加在输出上的正则项

activation='relu' # 激励函数

# bias\_regularizer=keras.regularizers.l1\_l2(0.01) # 施加在偏置向量上的正则项

)

)

model.add(Dense(units = 1,

activation='linear' # 线性激励函数 回归一般在输出层用这个激励函数

)

)

print(model.summary()) # 打印网络层次结构

model.compile(loss='mse', # 损失均方误差

optimizer='adam', # 优化器

)

history = model.fit(x\_train, y\_train,

epochs=400, # 迭代次数

batch\_size=200, # 每次用来梯度下降的批处理数据大小

verbose=2, # verbose：日志冗长度，int：冗长度，0：不输出训练过程，1：输出训练进度，2：输出每一个epoch

validation\_data = (x\_valid, y\_valid) # 验证集

)

(4) 训练过程可视化

# 绘制训练 & 验证的损失值

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title(' Model loss ')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

(5) 保存模型、模型可视化和加载模型

# 保存模型

model.save('model\_ passenger.h5') # 生成模型文件 'my\_model.h5'

# 模型可视化 需要安装pydot pip install pydot

plot\_model(model, to\_file='model\_MLP.png', show\_shapes=True)

# 加载模型

model = load\_model('model\_passenger.h5')

（6）模型的预测功能

y\_new = model.predict(x\_valid)

# 反归一化还原原始量纲

min\_max\_scaler.fit(y\_valid\_pd)

y\_new = min\_max\_scaler.inverse\_transform(y\_new)

（7）结果分析

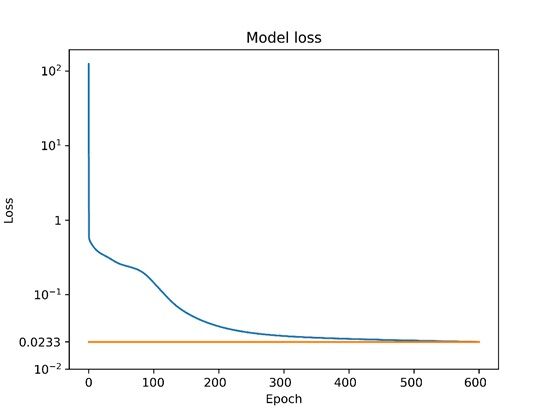


图 1损失Loss图

由图可以看出，在迭代了400个epochs之后，训练集和验证集的损失loss，趋于平稳，这时，我们得到的模型已经是最优的了。所以将epoch设置为400即可。