LSTM 和 PROPHET 以及 BP 神经网络的部署手册

团队编号:2002066　二仙桥老大爷

目录

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）

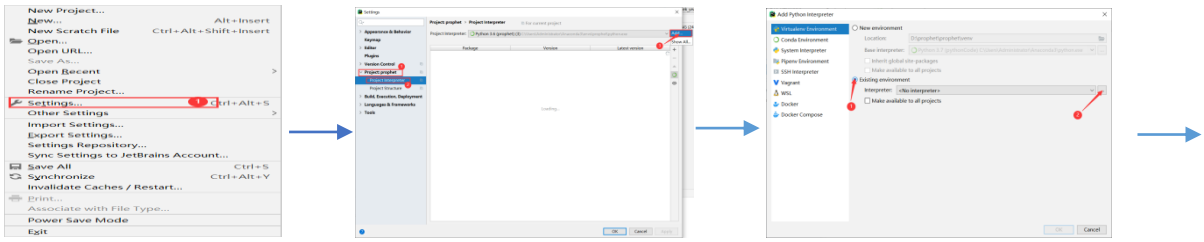
（2）环境切换

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）

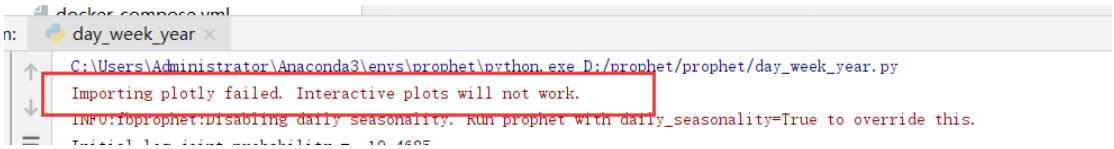
里面有 setup.py 脚本，直接运行一下命令：

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

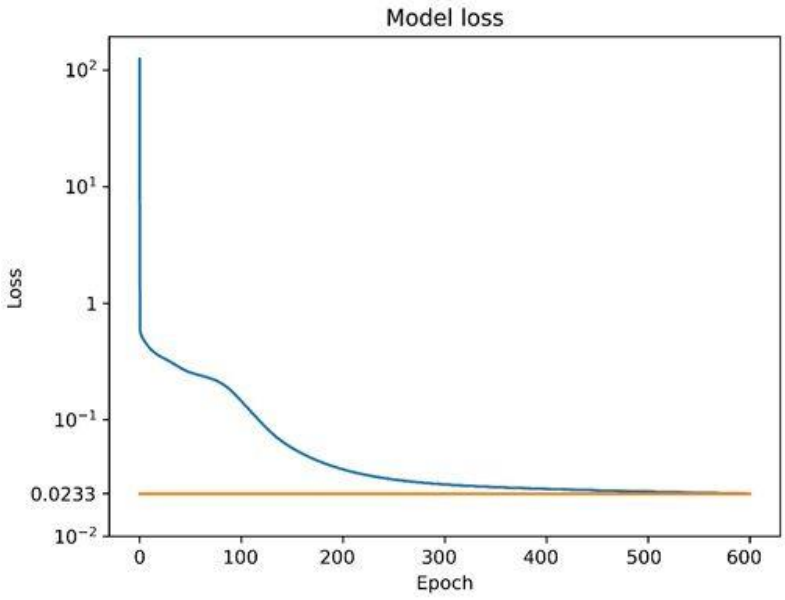


图 1 损失 Loss 图

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。