案例六：预测维基百科未来一个月的文章数

1. 案例目的

• 掌握LSTM模型；

• 掌握使用LSTM解决时间序列问题；

• 处理时间序列数据。

2. 案例内容

本案例将对145063篇文章550天的访问量进行分析，建立LSTM模型，预测未来一个月内每篇文章的访问量。

3. 案例知识点

• Python语言编程；

• NumPy和Pandas库的使用；

• LSTM模型；

• 处理时间序列数据。

4. 案例时长

共4学时，具体安排如下：

• 筛选出无任何缺失值的文章（0.2学时）

• 随机选取一篇文章并进行预处理（1学时）

• 建立LSTM模型（0.5学时）

• 编译模型、训练模型（1.3学时）

• 测试模型及可视化实际值和预测值（0.5学时）

• 预测维基百科本篇文章的访问量（0.5学时）

5. 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）sklearn 0.23

**•开发环境与工具：**

1）Spyder 4

6. 案例分析

近年来，随着通信网络技术的不断发展，网络流量的需求也日益增多。网络流量模型和预测在分析实际网络流量和网络性能方面起着重要作用。通过足够准确的预测，可以提高网络的利用率和性能。一个经典的案例是预测维基百科145063篇文章未来的每天的访问量。

本案例主要分为以下5部分：

1） 筛选出无任何缺失值的文章；

2） 随机选取一篇文章并进行预处理，为训练模型做准备；

3） 建立LSTM模型，编译模型，训练模型；

4） 测试模型，并可视化对比实际值与预测值；

5） 预测下一天维基百科文章的访问量。

7. 案例实验过程

预测维基百科文章的访问量，可分为以下几个步骤：

1、 导入库

2、 设置超参数

3、 导入数据集

4、 筛选出无任何缺失值的文章

5、 随机选取一篇文章并进行预处理

5.1 随机选出一篇文章

5.2 转换数据类型为DataFrame类型

5.3 画出访问量折线图

5.4 数据类型由DataFrame类型转换为Numpy

5.5 特征缩放

5.6 处理成时间序列

5.7 拆分数据

5.8 转换为3维数据

6、 建立LSTM模型

7、 编译模型

8、 训练模型

9、 测试模型及可视化实际值和预测值

9.1 测试模型

9.2 数据逆归一化

9.3 可视化实际值与预测值

10、可视化实际值与预测值

11、预测下一天维基百科本篇文章的访问量

7.1开发准备

在桌面上新建一个目录，名字叫wikipedia\_visits。本案例中所有的数据集和代码都放在该目录下。

7.1.1 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至wikipedia\_visits目录下。

7.1.2 获取数据

本案例需要1个数据集，wikipedia\_visits.csv文件。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/5ee56aade6644e5aa19069526e94b03f.csv。下载数据集，将wikipedia\_visits.csv复制到wikipedia\_visits目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为wikipedia\_visits。使用Spyder新建1个Python文件，命名为wikipedia\_visits.py。

至此，整个案例的目录结构图1所示。

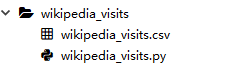


图1 案例的目录结构

7.1.4 了解数据

wikipedia\_visits数据集包含了145063篇文章从2015年7月1日至2016年12月31日每天的访问量，共550天。

数据样式如图2所示。

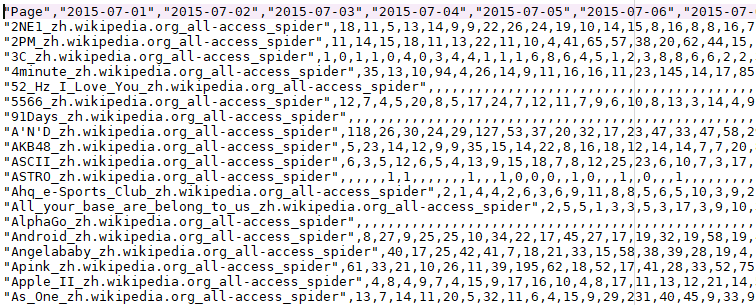


图2 wikipedia\_visits.csv文件

7.2导入库

向wikipedia\_visits.py添加代码。把所有用到的库导入。

【代码7-2】wikipedia\_visits.py

#导入包

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

执行代码块。

7.3设置超参数

这些超参数都是训练模型的一些指定参数，一批加载几条数据，训练多少个EPOCH。如果模型性能不好或者出现其他问题，则需要调整这些参数。在案例中，EPOCHS的数量不足，会导致模型欠拟合。

【代码7-3】wikipedia\_visits.py

# 定义超参数

SEQUENCE\_LENGTH = 20

EPOCHS = 100

BATCH\_SIZE = 64

执行代码块。

7.4导入数据集

导入数据，并使用fillna()填充缺失的数据，填充值为-1。

【代码7-4】wikipedia\_visits.py

# 导入数据

dataset = pd.read\_csv('wikipedia\_visits.csv').fillna(-1)

执行代码块。

从现在开始，建议打开Spyder的Variable explorer面板，实时查看变量的信息。

145063条数据的信息保存在dataset变量中，如图3所示。

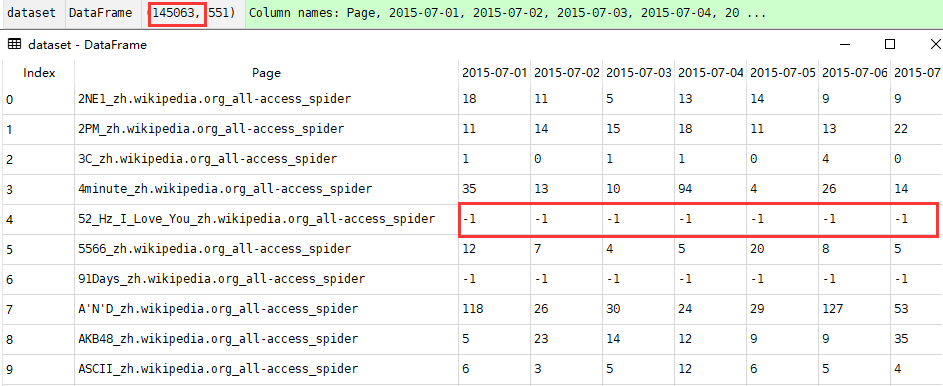


图3 145063条数据的信息保存在dataset变量中

7.5筛选出无任何缺失值的文章

该部分的目的是筛选出没有任何缺失值的文章。通过遍历dataset，检查dataset每一行，然后检查每一行中的每一个字段，将没有任何缺失值文章的索引号加入事先声明的列表中，最后按照索引号将没有任何缺失值文章筛选出来，更新dataset变量。

【代码7-5】wikipedia\_visits.py

# 找到无任何缺失值的文章

not\_null\_index\_list = []

# 检查dataset每一行

for i in range(dataset.shape[0]):

# 输出进度条

if i > 0 and i % 1000 == 0:

print('Progress: %d/%d' %(i, dataset.shape[0]))

each\_row = dataset.iloc[i, 1:] # 过滤掉Page列

has\_none\_column = False

# 检查每一行中的每一个字段

for j in each\_row:

if j <0:

has\_none\_column = True

break

if has\_none\_column:

continue

not\_null\_index\_list.append(i)

# 筛选出无任何缺失值的文章

dataset = dataset.iloc[not\_null\_index\_list,:]

# 按文章统计缺失值

null\_stat = dataset.isnull().sum()

# 删除Page列

dataset = dataset.drop('Page',axis = 1)

执行代码块。

执行代码后，筛选出无任何缺失值的文章117277篇，117277条数据的信息仍保存在dataset变量中，且将Page列删除。如图4所示：

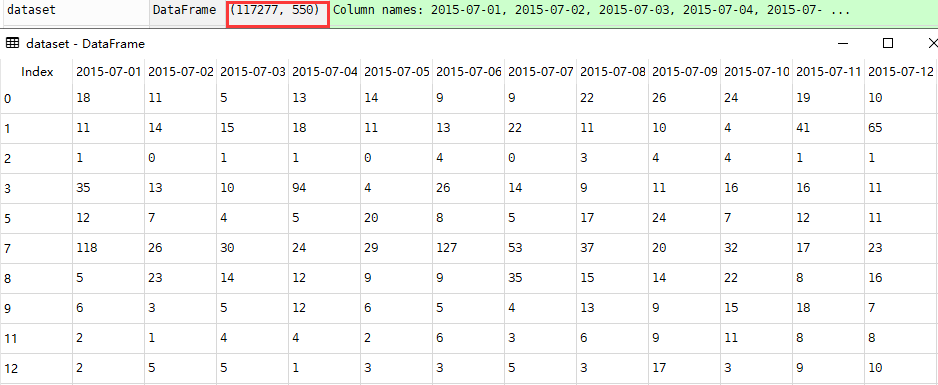


图4 117277条数据的信息仍保存在dataset变量中

7.6随机选取一篇文章并进行预处理

后面训练模型其实只是随机选取了一篇文章的数据作为训练数据。

该模块的主要作用是随机选取出一片文章，并进行预处理，为训练模型做准备。

7.6.1 随机选出一篇文章

因为已经筛选出117277篇没有任何缺失值的文章，所以从0-117276之间随机选一个数。并将所选取文章的时间序列保存在random\_row\_series变量中。

【代码7-6-1】wikipedia\_visits.py

# 随机抽取一篇文章

# 0-117276之间随机选一个数

random\_seed = 520

random\_row\_series = dataset.iloc[random\_seed, :]

执行代码块。

运行代码后，random\_row\_series变量如图5所示。



图5 所选取文章的时间序列保存在random\_row\_series变量中

7.6.2 转换数据类型为DataFrame类型

将random\_row\_series变量转换为DataFrame类型。

【代码7-6-2】wikipedia\_visits.py

# 将random\_row\_series转换为DataFrame类型

random\_row\_df = random\_row\_series.to\_frame()

random\_row\_df.columns=['Visits']

执行代码块。

7.6.3 画出访问量折线图

画出所选取文章550天访问量的折线图。

【代码7-6-3】wikipedia\_visits.py

# 画出访问量折线图

random\_row\_df.plot()

执行代码块。

所选取文章550天访问量的折线图如图6所示。

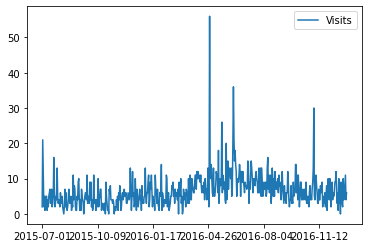


图6 选取文章550天访问量的折线图

7.6.4 数据类型由Dataframe转换成Numpy

将数据类型由dataframe转换成numpy。

【代码7-6-4】wikipedia\_visits.py

# 将dataframe转换成numpy

dataset\_array = random\_row\_df.values

执行代码块。

7.6.5 特征缩放

接下来，进行数据特征缩放，缩放后的特征值在0~1之间。

【代码7-6-5】wikipedia\_visits.py

# 特征缩放

sc = MinMaxScaler(feature\_range = (0, 1))

dataset\_scaled = sc.fit\_transform(dataset\_array)

执行代码块。

特征缩放后的数据如图7所示。数据都在0~1之间。

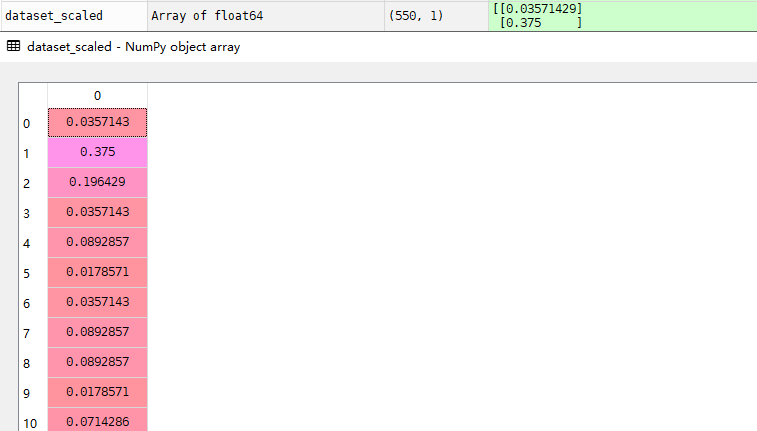


图7 特征缩放后的数据

7.6.6 处理成时间序列

建立LSTM模型之前，必须把数据处理成时间序列数据。

为了简单起见，假设数据集的数据只有9条，如图8所示。

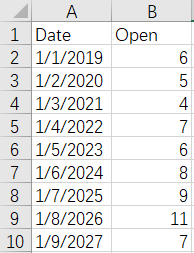


图8 假设数据集只有9条

如果要通过前5天的访问量推断当天的访问量，那么数据应该转换为如下格式，效果如图9所示。

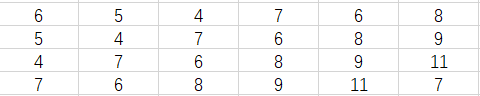


图9 转换后的数据集格式

图8到图9的转换过程如下：

由于根据前5天的访问量推断当天的访问量，所以前5天的数据作为自变量，后一天的数据作为因变量。

图9第1条数据，自变量分别是2019-1-1至2019-1-5的数据（数值是6、5、4、7、6），因变量是2019-1-6的数据（数值是8）。

图9第2条数据，自变量分别是2019-1-2至2019-1-6的数据（数值是5、4、7、6、8），因变量是2019-1-7的数据（数值是9）。

图9第3条数据，自变量分别是2019-1-3至2019-1-7的数据（数值是4、7、6、8、9），因变量是2019-1-8的数据（数值是11）。

本案例根据前20天（在【7.3 设置超参数】小节中设置了超参数SEQUENCE\_LENGTH = 20）的文章的访问量推断当天的访问量，把数据转换为时间序列格式。

【代码7-6-6】wikipedia\_visits.py

# 处理成时间序列数据

X = []

y = []

for i in range(SEQUENCE\_LENGTH, dataset\_scaled.shape[0]):

X.append(dataset\_scaled[i-SEQUENCE\_LENGTH:i, 0])

y.append(dataset\_scaled[i, 0])

执行代码块。

X的格式如图10所示。

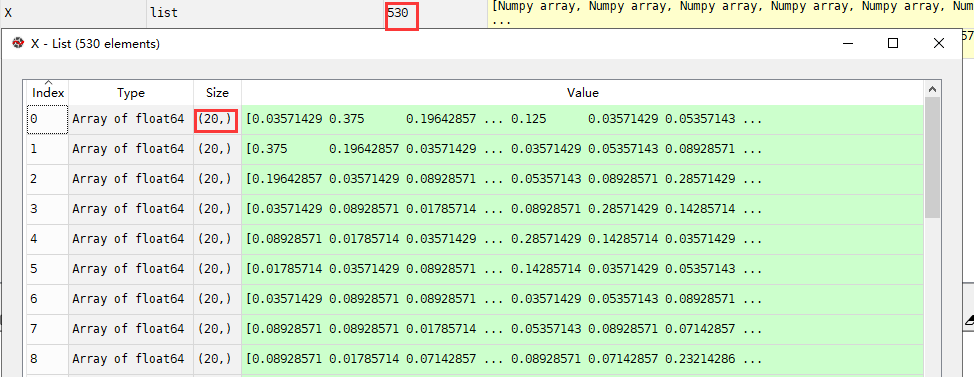


图10 X的格式

y的格式如图11所示。

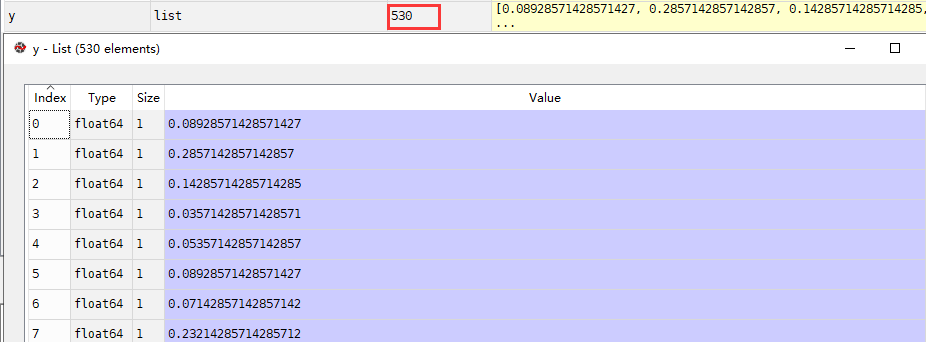


图11 y的格式

由图10和图11可见，X的有20列，y有1列，即：根据前20天的访问量推断当天的访问量。

将X和y由list类型转换为NumPy类型，因为模型需要的参数是NumPy类型。

【代码7-6-6】wikipedia\_visits.py

# list转换为Numpy

X, y = np.array(X), np.array(y)

执行代码块。

X现在有530行，20列。y有530行，1列。X的样式，y如图12所示。



图12 X的样式

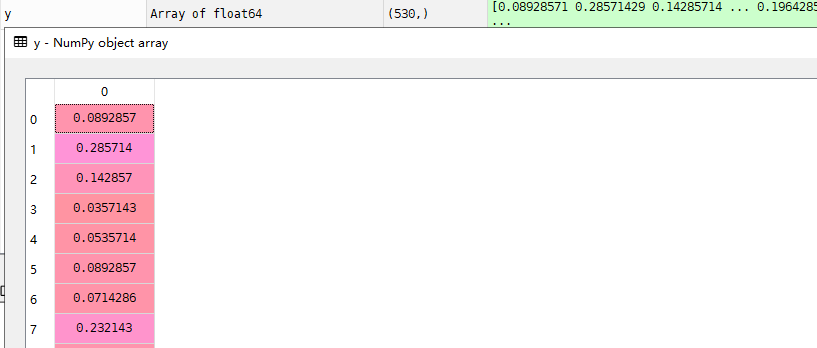


图13 y的样式

7.6.7 拆分数据

拆分数据为训练数据和测试数据，20%为测试数据，80%为训练数据。

【代码7-6-7】wikipedia\_visits.py

# 拆分数据

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

执行代码块。

数据集拆分后的数据信息分别保存在X\_train,X\_test,y\_train,y\_test变量中，如图14所示。



图14 数据集拆分后的数据信息分别保存在X\_train,X\_test,y\_train,y\_test变量中

7.6.8 转换为3维数据

根据LSTM类的数据格式要求，自变量数据必须是3维的，第1维是数据条数，第2维是时间序列数，第3维是预测器的个数。

【代码7-6-8】wikipedia\_visits.py

# 训练集由2维变为3维

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)) # 第3维的数字1表示只有1个indicator

执行代码块。

执行后X\_train变为3维数组，效果如图15所示。



图15 X\_train变量

7.7建立LSTM模型

该部分的作用是建立LSTM模型。

输入层的shape是(X\_train.shape[1], 1))，其中X\_train.shape[1]的值是20。20表示时间序列是20个，1表示使用1个预测器。

第1层隐藏层定义50个神经元；激活函数默认是tanh（在LSTM中，tanh激活函数很常用）；如果后面还有LSTM层，那么return\_sequences的值设为True；Dropout机制用于解决过拟合。第2、3层隐藏层类似。定义第4层LSTM层，也是最后1层LSTM层。所以return\_sequences的值设为False。该参数的默认值是False。由于预测的是连续型数值，所以输出神经元仅需要1个，且激活函数是linear。

【代码7-7】wikipedia\_visits.py

# 初始化LSTM

regressor = Sequential()

# 添加输入层和第1层隐藏层

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2)) # 解决过拟合

# 添加第2层隐藏层

regressor.add(LSTM(units = 100, return\_sequences = True))

regressor.add(Dropout(0.2))

# 添加第3层隐藏层

regressor.add(LSTM(units = 200, return\_sequences = True))

regressor.add(Dropout(0.2))

# 添加第4层隐藏层

regressor.add(LSTM(units = 300))

regressor.add(Dropout(0.2))

# 添加输出层

regressor.add(Dense(units = 1, activation='linear'))

执行代码块。

7.8编译模型

由于预测的是连续型数值，所以loss的值为mean\_squared\_error；优化算法选择adam。

【代码7-8】wikipedia\_visits.py

# 编译模型

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

执行代码块。

7.9训练模型

在Intel(R) Core(TM) i7-8750H [CPU@2.20GHz](mailto:CPU@2.20GHz)CPU，8GBRAM的计算机上，每个epoch的运行时间大约为1s，训练100个epoch大约需要2分钟。

【代码7-9】wikipedia\_visits.py

# 训练模型

regressor.fit(X\_train, y\_train, epochs = EPOCHS, batch\_size = BATCH\_SIZE)

执行代码块。

训练过程如图16所示：

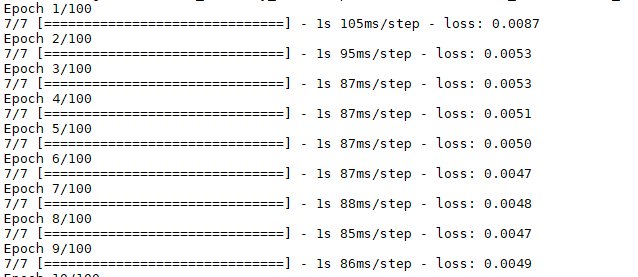


图16 训练过程图

注意：由于模型初始会设置随机权重，所以同学们的运行结果不会和本案例图示的一模一样。

7.10测试模型及可视化实际值和预测值

模型训练完毕后，对测试集进行预测，并可视化实际值和预测值，进行对比。

7.10.1 测试模型

对测试集进行预测。

【代码7-10-1】wikipedia\_visits.py

# 预测测试集数据

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)) # 第3维的数字1表示只有1个indicator

predicted\_visits\_scaled = regressor.predict(X\_test)

执行代码块。

预测结果保存在predicted\_visits\_scaled变量中，如图17所示。

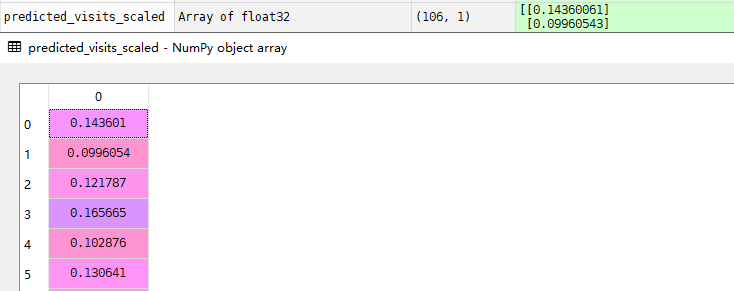


图17 预测结果保存在predicted\_visits\_scaled变量中

7.10.2 数据逆归一化

由于自变量是归一化后的数据，所以预测出的因变量也是归一化后的数据（数值在0~1）之间）。需要逆归一化，这样才能和实际访问量做比对。

【代码7-10-2】wikipedia\_visits.py

# 数据逆归一化

predicted\_visits = sc.inverse\_transform(predicted\_visits\_scaled).astype('int64')

执行代码块。

逆归一化后的数据如图18所示。

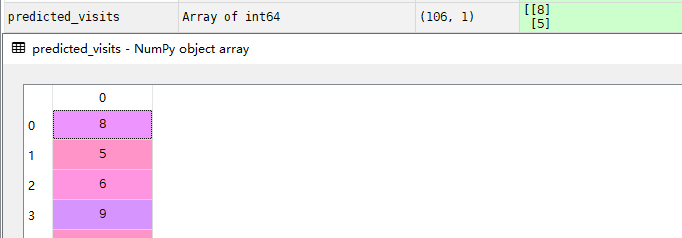


图18 逆归一化后的数据

7.10.3 可视化实际值与预测值

在测试集上可视化实际值与预测值，对比实际值与预测值的差异。

【代码7-10-3】wikipedia\_visits.py

# 可视化实际值与预测值

plt.plot(sc.inverse\_transform(y\_test.reshape(-1,1)), color = 'red', label = 'Real Visits')

plt.plot(predicted\_visits, color = 'blue', label = 'Predicted Visits')

plt.title('Real Visits vs Predicted Visits')

plt.xlabel('Time')

plt.ylabel('Wikipedia Visits')

plt.legend()

plt.show()

执行代码块。

实际值与预测值的曲线如图19所示。观察到，实际值与预测值的差距不大，说明模型预测的是比较准的。

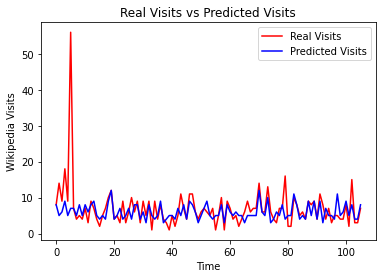


图19 实际值与预测值的曲线

7.11预测下一天维基百科本篇文章的访问量

通过2016-12-12至2016-12-31总共20天的数据，预测2017-01-01该篇文章的访问量。

【代码7-11】wikipedia\_visits.py

# 明天维基百科本篇文章的访问量

pattern = dataset\_array[-SEQUENCE\_LENGTH:]

pattern = pattern.reshape(-1,1)

x = sc.transform(pattern) # 数据标准化

x = np.reshape(x, (1, SEQUENCE\_LENGTH, 1))# 将pattern转换为3维 [samples, time steps, features]

prediction\_scaled = regressor.predict(x, verbose=0) # 得到预测向量

prediction = sc.inverse\_transform(prediction\_scaled)

print('明天维基百科本篇文章的访问量是：', int(prediction[0,0]))

执行代码块。

运行代码后，控制台输出预测结果，如图20所示。



图20 明天维基百科本篇文章的访问量的预测结果

注：如果使用ARIMA模型训练，会生成一个时间序列的表达式，可以预测未来一个月的访问量。这里使用的是LSTM，没有数学表达式，只能预测明天的访问量，然后基于明天的预测后天的，基于后天的预测大后天的……如果要预测未来一个月的访问量，推荐使用ARIMA方法。

8. 案例代码

【案例代码】predictsurvivors.py

'''

预测维基百科未来一个月的文章数

'''

#导入包

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

# 定义超参数

SEQUENCE\_LENGTH = 20

EPOCHS = 100

BATCH\_SIZE = 64

# 导入数据

dataset = pd.read\_csv('wikipedia\_visits.csv').fillna(-1)

# 找到无任何缺失值的文章

not\_null\_index\_list = []

# 检查dataset每一行

for i in range(dataset.shape[0]):

# 输出进度条

if i > 0 and i % 1000 == 0:

print('Progress: %d/%d' %(i, dataset.shape[0]))

each\_row = dataset.iloc[i, 1:] # 过滤掉Page列

has\_none\_column = False

# 检查每一行中的每一个字段

for j in each\_row:

if j <0:

has\_none\_column = True

break

if has\_none\_column:

continue

not\_null\_index\_list.append(i)

# 筛选出无任何缺失值的文章

dataset = dataset.iloc[not\_null\_index\_list,:]

# 按文章统计缺失值

null\_stat = dataset.isnull().sum()

# 删除Page列

dataset = dataset.drop('Page',axis = 1)

# 随机抽取一篇文章

# 0-117276之间随机选一个数

random\_seed = 520

random\_row\_series = dataset.iloc[random\_seed, :]

# 将random\_row\_series转换为DataFrame类型

random\_row\_df = random\_row\_series.to\_frame()

random\_row\_df.columns=['Visits']

# 画出访问量折线图

random\_row\_df.plot()

# 将dataframe转换成numpy

dataset\_array = random\_row\_df.values

# 特征缩放

sc = MinMaxScaler(feature\_range = (0, 1))

dataset\_scaled = sc.fit\_transform(dataset\_array)

# 处理成时间序列数据

X = []

y = []

for i in range(SEQUENCE\_LENGTH, dataset\_scaled.shape[0]):

X.append(dataset\_scaled[i-SEQUENCE\_LENGTH:i, 0])

y.append(dataset\_scaled[i, 0])

# list转换为Numpy

X, y = np.array(X), np.array(y)

# 拆分数据

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# 训练集由2维变为3维

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)) # 第3维的数字1表示只有1个indicator

# 初始化LSTM

regressor = Sequential()

# 添加输入层和第1层隐藏层

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2)) # 解决过拟合

# 添加第2层隐藏层

regressor.add(LSTM(units = 100, return\_sequences = True))

regressor.add(Dropout(0.2))

# 添加第3层隐藏层

regressor.add(LSTM(units = 200, return\_sequences = True))

regressor.add(Dropout(0.2))

# 添加第4层隐藏层

regressor.add(LSTM(units = 300))

regressor.add(Dropout(0.2))

# 添加输出层

regressor.add(Dense(units = 1, activation='linear'))

# 编译模型

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

# 训练模型

regressor.fit(X\_train, y\_train, epochs = EPOCHS, batch\_size = BATCH\_SIZE)

# 预测测试集数据

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)) # 第3维的数字1表示只有1个indicator

predicted\_visits\_scaled = regressor.predict(X\_test)

# 数据逆归一化

predicted\_visits = sc.inverse\_transform(predicted\_visits\_scaled).astype('int64')

# 可视化实际值与预测值

plt.plot(sc.inverse\_transform(y\_test.reshape(-1,1)), color = 'red', label = 'Real Visits')

plt.plot(predicted\_visits, color = 'blue', label = 'Predicted Visits')

plt.title('Real Visits vs Predicted Visits')

plt.xlabel('Time')

plt.ylabel('Wikipedia Visits')

plt.legend()

plt.show()

# 明天维基百科本篇文章的访问量

pattern = dataset\_array[-SEQUENCE\_LENGTH:]

pattern = pattern.reshape(-1,1)

x = sc.transform(pattern) # 数据标准化

x = np.reshape(x, (1, SEQUENCE\_LENGTH, 1))# 将pattern转换为3维 [samples, time steps, features]

prediction\_scaled = regressor.predict(x, verbose=0) # 得到预测向量

prediction = sc.inverse\_transform(prediction\_scaled)

print('明天维基百科本篇文章的访问量是：', int(prediction[0,0]))