简介

股票价格预测是一件非常唬人的事情，但如果只基于历史数据进行预测，显然完全不靠谱

股票价格是典型的时间序列数据（简称时序数据），会受到经济环境、政府政策、人为操作多种复杂因素的影响

不像气象数据那样具备明显的时间和季节性模式，例如一天之内和一年之内的气温变化等

尽管如此，以股票价格为例，介绍如何对时序数据进行预测，仍然值得一做

以下使用TensorFlow和Keras，对 `S&P 500` 股价数据进行分析和预测

数据

`S&P 500` 股价数据爬取自Google Finance API，已经进行过缺失值处理

加载库，pandas主要用于数据清洗和整理

```
# -*- coding: utf-8 -*-

import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import MinMaxScaler
import time
```

用pandas读取csv文件为DataFrame，并用 `describe()` 查看特征的数值分布

```
data = pd.read_csv('data_stocks.csv')
data.describe()
```

还可以用 `info()` 查看特征的概要

```
data.info()
```

数据共502列，41266行，502列分别为：

- `DATE`：该行数据的时间戳
- `SP500`：可以理解为大盘指数
- 其他：可以理解为500支个股的股价

查看数据的前五行

```
data.head()
```

查看时间跨度

```
print(time.strftime('%Y-%m-%d', time.localtime(data['DATE'].max())),
      time.strftime('%Y-%m-%d', time.localtime(data['DATE'].min())))
```

绘制大盘趋势折线图

```
plt.plot(data['SP500'])
```

去掉 `DATE` 一列，训练集测试集分割

```
data.drop('DATE', axis=1, inplace=True)
data_train = data.iloc[:int(data.shape[0] * 0.8), :]
data_test = data.iloc[int(data.shape[0] * 0.8):, :]
print(data_train.shape, data_test.shape)
```

数据归一化，只能使用 `data_train` 进行 `fit()`

```
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
```

同步预测

同步预测是指，使用当前时刻的500支个股股价，预测当前时刻的大盘指数，即一个回归问题，输入共500维特征，输出一维，即 `[None, 500] => [None, 1]`

使用TensorFlow实现同步预测，主要用到多层感知机（Multi-Layer Perceptron，MLP），损失函数用均方误差（Mean Square Error，MSE）

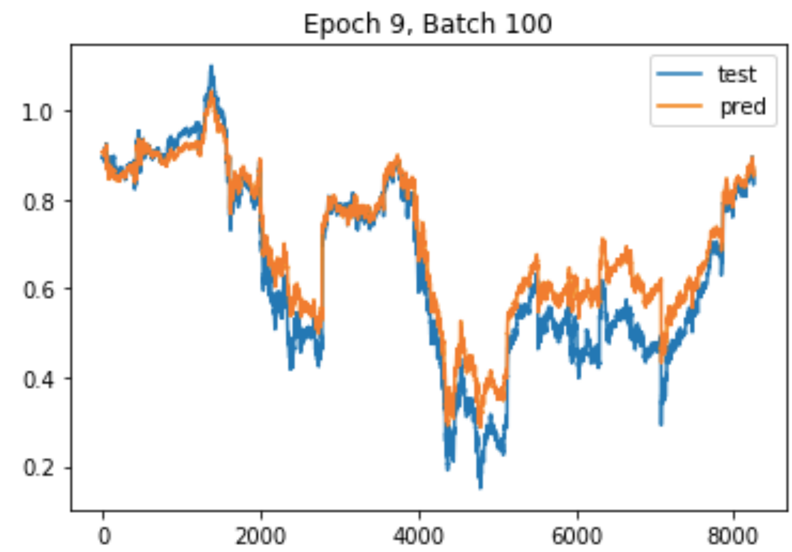
```
X_train = data_train[:, 1:]
y_train = data_train[:, 0]
X_test = data_test[:, 1:]
y_test = data_test[:, 0]

input_dim = X_train.shape[1]
hidden_1 = 1024
hidden_2 = 512
hidden_3 = 256
hidden_4 = 128
```

```
output_dim = 1
batch_size = 256
epochs = 10

tf.reset_default_graph()

X = tf.placeholder(shape=[None, input_dim], dtype=tf.float32)
Y = tf.placeholder(shape=[None], dtype=tf.float32)

W1 = tf.get_variable('W1', [input_dim, hidden_1], initializer=tf.contrib.layers.xa
vier_initializer(seed=1))
b1 = tf.get_variable('b1', [hidden_1], initializer=tf.zeros_initializer())
W2 = tf.get_variable('W2', [hidden_1, hidden_2], initializer=tf.contrib.layers.xav
ier_initializer(seed=1))
b2 = tf.get_variable('b2', [hidden_2], initializer=tf.zeros_initializer())
W3 = tf.get_variable('W3', [hidden_2, hidden_3], initializer=tf.contrib.layers.xav
ier_initializer(seed=1))
b3 = tf.get_variable('b3', [hidden_3], initializer=tf.zeros_initializer())
W4 = tf.get_variable('W4', [hidden_3, hidden_4], initializer=tf.contrib.layers.xav
ier_initializer(seed=1))
b4 = tf.get_variable('b4', [hidden_4], initializer=tf.zeros_initializer())
W5 = tf.get_variable('W5', [hidden_4, output_dim], initializer=tf.contrib.layers.x
avier_initializer(seed=1))
b5 = tf.get_variable('b5', [output_dim], initializer=tf.zeros_initializer())

h1 = tf.nn.relu(tf.add(tf.matmul(X, W1), b1))
h2 = tf.nn.relu(tf.add(tf.matmul(h1, W2), b2))
h3 = tf.nn.relu(tf.add(tf.matmul(h2, W3), b3))
h4 = tf.nn.relu(tf.add(tf.matmul(h3, W4), b4))
out = tf.transpose(tf.add(tf.matmul(h4, W5), b5))

cost = tf.reduce_mean(tf.squared_difference(out, Y))
optimizer = tf.train.AdamOptimizer().minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for e in range(epochs):
        shuffle_indices = np.random.permutation(np.arange(y_train.shape[0]))
        X_train = X_train[shuffle_indices]
        y_train = y_train[shuffle_indices]

        for i in range(y_train.shape[0] // batch_size):
            start = i * batch_size
            batch_x = X_train[start : start + batch_size]
            batch_y = y_train[start : start + batch_size]
            sess.run(optimizer, feed_dict={X: batch_x, Y: batch_y})
```

```
            if i % 50 == 0:
                print('MSE Train:', sess.run(cost, feed_dict={X: X_train, Y: y_train}))
                print('MSE Test:', sess.run(cost, feed_dict={X: X_test, Y: y_test}))
                y_pred = sess.run(out, feed_dict={X: X_test})
                y_pred = np.squeeze(y_pred)
                plt.plot(y_test, label='test')
                plt.plot(y_pred, label='pred')
                plt.title('Epoch ' + str(e) + ', Batch ' + str(i))
                plt.legend()
                plt.show()
```

最后测试集的loss在0.005左右，预测结果如下

```
MSE Train: 7.83046e-05
MSE Test: 0.00491444
```

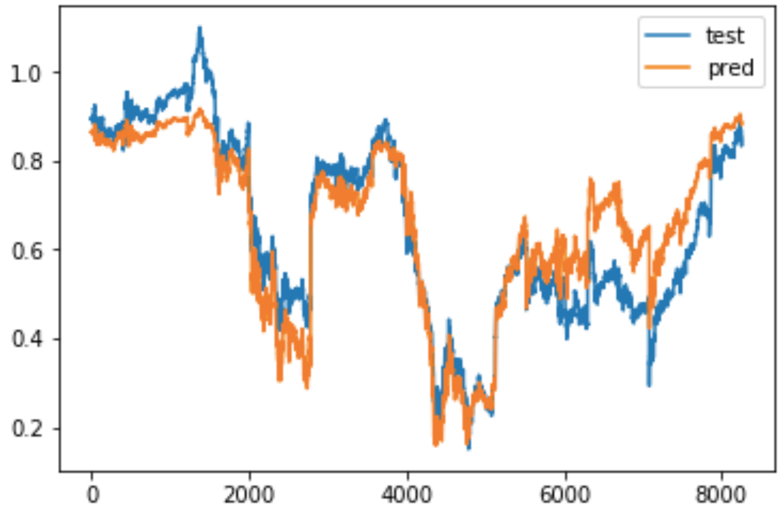


使用Keras实现同步预测，代码量会少很多，但具体实现细节不及TensorFlow灵活

```
from keras.layers import Input, Dense
from keras.models import Model

X_train = data_train[:, 1:]
y_train = data_train[:, 0]
X_test = data_test[:, 1:]
y_test = data_test[:, 0]

input_dim = X_train.shape[1]
hidden_1 = 1024
hidden_2 = 512
hidden_3 = 256
hidden_4 = 128
output_dim = 1
batch_size = 256
epochs = 10

X = Input(shape=[input_dim,])
h = Dense(hidden_1, activation='relu')(X)
h = Dense(hidden_2, activation='relu')(h)
h = Dense(hidden_3, activation='relu')(h)
h = Dense(hidden_4, activation='relu')(h)
Y = Dense(output_dim, activation='sigmoid')(h)

model = Model(X, Y)
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, shuffle=False)
y_pred = model.predict(X_test)
print('MSE Train:', model.evaluate(X_train, y_train, batch_size=batch_size))
print('MSE Test:', model.evaluate(X_test, y_test, batch_size=batch_size))
plt.plot(y_test, label='test')
plt.plot(y_pred, label='pred')
plt.legend()
plt.show()
```

最后测试集的loss在0.007左右，预测结果如下

```
MSE Test: 0.00695435043843
```



异步预测

异步预测是指，使用历史若干个时刻的大盘指数，预测当前时刻的大盘指数，这样才更加符合预测的定义

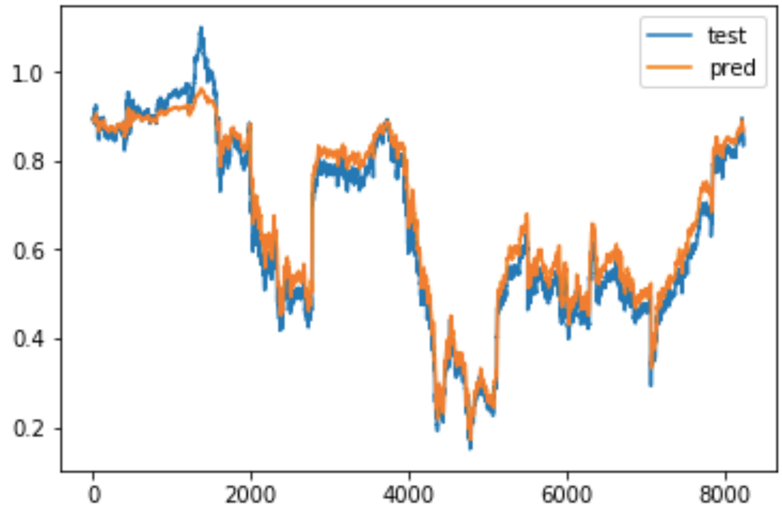
例如，使用前五个大盘指数，预测当前的大盘指数，每组输入包括5个step，每个step对应一个历史时刻的大盘指数，输出一维，即 `[None, 5, 1] => [None, 1]`

使用Keras实现异步预测，主要用到循环神经网络即RNN（Recurrent Neural Network）中的LSTM（Long Short-Term Memory）

```
from keras.layers import Input, Dense, LSTM
from keras.models import Model

output_dim = 1
batch_size = 256
epochs = 10
seq_len = 5
hidden_size = 128

X_train = np.array([data_train[i : i + seq_len, 0] for i in range(data_train.shape
[0] - seq_len)])[:, :, np.newaxis]
y_train = np.array([data_train[i + seq_len, 0] for i in range(data_train.shape[0]
- seq_len)])
X_test = np.array([data_test[i : i + seq_len, 0] for i in range(data_test.shape[0]
 - seq_len)])[:, :, np.newaxis]
y_test = np.array([data_test[i + seq_len, 0] for i in range(data_test.shape[0] - s
eq_len)])

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

X = Input(shape=[X_train.shape[1], X_train.shape[2],])
h = LSTM(hidden_size, activation='relu')(X)
Y = Dense(output_dim, activation='sigmoid')(h)

model = Model(X, Y)
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, shuffle=False)
y_pred = model.predict(X_test)
print('MSE Train:', model.evaluate(X_train, y_train, batch_size=batch_size))
print('MSE Test:', model.evaluate(X_test, y_test, batch_size=batch_size))
plt.plot(y_test, label='test')
plt.plot(y_pred, label='pred')
plt.legend()
plt.show()
```

最后测试集的loss在0.0015左右，预测结果如下，一层LSTM的效果已经好非常多了

```
MSE Test: 0.00146192818087
```



当然，还有一种可能的尝试，使用历史若干个时刻的500支个股股价以及大盘指数，预测当前时刻的大盘指数，即 `[None, 5, 501] => [None, 1]`

```
from keras.layers import Input, Dense, LSTM
from keras.models import Model

output_dim = 1
batch_size = 256
epochs = 10
seq_len = 5
hidden_size = 128

X_train = np.array([data_train[i : i + seq_len, :] for i in range(data_train.shape[0] - seq_len)])
y_train = np.array([data_train[i + seq_len, 0] for i in range(data_train.shape[0] - seq_len)])
X_test = np.array([data_test[i : i + seq_len, :] for i in range(data_test.shape[0] - seq_len)])
y_test = np.array([data_test[i + seq_len, 0] for i in range(data_test.shape[0] - seq_len)])

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

X = Input(shape=[X_train.shape[1], X_train.shape[2],])
h = LSTM(hidden_size, activation='relu')(X)
Y = Dense(output_dim, activation='sigmoid')(h)

model = Model(X, Y)
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, shuffle=False)
y_pred = model.predict(X_test)
print('MSE Train:', model.evaluate(X_train, y_train, batch_size=batch_size))
print('MSE Test:', model.evaluate(X_test, y_test, batch_size=batch_size))
plt.plot(y_test, label='test')
plt.plot(y_pred, label='pred')
plt.legend()
plt.show()
```

最后的loss在0.004左右，结果反而变差了

500支个股加上大盘指数的预测效果，还不如仅使用大盘指数

说明特征并不是越多越好，有时候反而会引入不必要的噪音

由于并未涉及到复杂的CNN或RNN，所以在CPU上运行的速度还可以

参考

- A simple deep learning model for stock price prediction using TensorFlow：https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-

using-tensorflow-30505541d877