# 实战案例5：比特币价格分析

作者： Robin
日期： 2018/03
提问： 小象问答
数据集来源： kaggle


## 1. 案例描述

比特币（英语：Bitcoin，缩写：BTC）是一种去中心化，非普遍全球可支付的电子加密货币。比特币由中本聪（又译中本哲史）（化名）于2009年1月3日，基于无国界的对等网络，用共识主动性开源软件发明创立。自比特币出现一直至今，比特币一直目前法币市场总值最高的加密货币。

任何人皆可参与比特币活动，可以通过称为挖矿的电脑运算来发行。比特币协议数量上限为2100万个，以避免通货膨胀问题。使用比特币是通过私钥作为数字签名，允许个人直接支付给他人，不需经过如银行、清算中心、证券商等第三方机构，从而避免了高手续费、繁琐流程以及受监管性的问题，任何用户只要拥有可连接互联网的数字设备皆可使用。

本案例中尝试使用比特币的历史价格数据对未来价格进行预测。

## 2. 数据集描述

- Kaggle提供的数据集，`coinbaseUSD_1-min_data_2014-12-01_to_2018-01-08.csv`
- 数据字典
    - **Timestamp**: Unix时间戳，整型
    - **Open**: 开盘价
    - **High**: 最高价
    - **Low**: 最低价
    - **Close**: 收盘价
    - **Volume_(BTC)**: 基于比特币的成交量
    - **Volume_(Currency)**: 基于货币的成交量
    - **Weighted_Price**: 比特币的加权价格

## 3. 任务描述

- 搭建深度神经网络LSTM，使用比特币的历史价格数据对其未来价格进行预测。

## 4. 主要代码解释

- 代码结构

```
lect08_proj
├── data
    ├── coinbaseUSD_1-min_data_2014-12-01_to_2018-01-08.csv   # 比特币历史价格数据集
├── output  # 程序输出结果保存的目录
├── main.py     # 主程序
├── utils.py     # 工具文件，包含数据加载、时序数据处理、LSTM模型搭建及预测等
├── config.y    # 配置文件
├── lect08_proj_readme.pdf  # 案例讲解文档
```

- **utils.py**

训练数据使用的特征应是前一个时刻的数据，即使用`t-1`时刻的数据预测`t`时刻的价格。使用`pandas`中的`shift()`函数可以完成该操作。

```
def make_data_for_model(data_df):
    ...
    # 使用t-1时刻的价格作为一个特征
    data_df[config.label_col] = data_df[config.raw_label_col].shift(-1)
    ...
```

- **utils.py**

在对时序数据进行训练前，需要对其进行`差分`操作，使时序数据达到平稳。

```
def make_stationary_seq(data_df, vis=True):
    ...
    # 差分操作
    stationary_df = data_df.diff()
    ...
```

- **utils.py**

为了使数据更好地用于LSTM模型，需要对数据进行归一化操作，使其范围归一化到`[-1, 1]`。

```
def scale_data(train_data, test_data):
    ...
    # 特征上的scaler
    X_scaler = MinMaxScaler(feature_range=(-1, 1))
    ...
    # 标签的scaler
    y_scaler = MinMaxScaler(feature_range=(-1, 1))
    ...
```

- **utils.py**

搭建LSTM时，输入样本的格式为`[样本个数, time step, 特征维度]`，因为本案例使用的预测只是基于上一个时间点的特征，所以`time step = 1`

```
def fit_lstm(X, y):
    ...
    # shape: (样本个数, time step, 特征维度)
    X = X.reshape(n_sample, config.timestep, n_feat_dim)
    ...
```

- **utils.py**

使用keras构建LSTM，其中`LSTM`的参数`stateful=True`表示，每一个batch训练时，使用的状态是来自与上一次batch，如果是`False`的话，表示每个batch的训练是独立的，但对于时序数据来说，需要考虑之前的状态。

```
def fit_lstm(X, y):
    ...
    # 构建模型
    model = Sequential()
    model.add(LSTM(config.nodes,
                   batch_input_shape=(config.batch_size, config.timestep, n_feat_dim),
                   stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    ...
```

- **utils.py**

在训练时，不能对样本进行shuffle，因为时序数据是有顺序的，所以`shuffle=False`。每完成一个epoch（在时序上完成一次训练），需要重置状态，进行新一轮的训练，所以需要使用`reset_states()`

```
def fit_lstm(X, y):
    ...
    for i in range(config.nb_epoch):
        print('已迭代{}次（共{}次） '.format(i + 1, config.nb_epoch))
        model.fit(X, y, epochs=1, batch_size=config.batch_size, verbose=0, shuffle=False)
        model.reset_states()
    ...
```

- **utils.py**

完成所有的epoch训练后，需要在训练数据上进行一次完成的预测，用于计算训练数据中最后一个时刻的状态。

```
def fit_lstm(X, y):
    ...
    # 在所有训练样本上运行一次，构建cell状态
    model.predict(X, batch_size=config.batch_size)
    ...
```

- **main.py**

进行预测时，需要对预测的数据进行scale反向操作和差分的反向操作，确保数据恢复到原始范围。

```
def main():
    ...
     # scale反向操作，恢复数据范围
    rescaled_y_pred = y_scaler.inverse_transform(y_pred.reshape(-1, 1))[0, 0]

     # 差分反向操作，恢复数据的值：加上前一天的真实标签
    previous_date = test_date - pd.DateOffset(days=1)
    recoverd_y_pred = rescaled_y_pred + all_daily_df.loc[previous_date][config.raw_label_col]
    ...
```

# 5. 案例总结

- 该项目通过使用深度学习的循环神经网络（RNN）进行时序数据的预测，包含了如下内容:
  - RNN的基本概念及框架
  - 使用keras搭建LSTM用于时序预测
  - 进行时序预测时，需要对时序数据的一些常用处理方式

# 6. 课后练习

- 增加`config.py`中的`nb_epoch`值，观察对结果的影响

# 参考资料

1. Keras教程
2. Keras RNN层
3. 使用LSTM进行时序预测