STUMP：Analyzing Motifs and Anomalies with STUMP

参考论文：http://www.cs.ucr.edu/~eamonn/PID4481997_extend_Matrix Profile_I.pdf

代码文档：https://stumpy.readthedocs.io/en/latest/Tutorial_STUMPY_Basics.html

中文参考：https://www.cnblogs.com/combfish/p/13886326.html

《基于矩阵画像的金融时序数据预测方法》https://kns.cnki.net/kcms/detail/detail.aspx?dbcode=CAPJ&dbname=CAPJLAST&filename=JSJY20200731005&v=p5eBltYjMg98BnYzsyOxiTgjHfQ%mmd2Fzo8KKeP8h1Xl1cuQovQbU3kder1OdAoyL3fr

基于矩阵画像的金融时序数据预测方法

摘要

目前在机构交易行为对于个股趋势影响以及通过机构交易行为来预测股价波动等方面的研究甚少，学者们更多的是从股票市场的总体范围来研究机构投资行为对股市稳定性以及股价波动的影响，在预测股价波动方面更多的是基于收盘价序列数据进行预测。相对而言，机构对股票的操纵行为通常是**间断性的且时间持续性不长**，使得**股票时间序列数据的局部性信息显得更为重要**。然而，传统的时间序列预测方法是基于数据的整体信息考虑，**缺乏对局部性数据的重视**。鉴于传统的方法对数据具有研究假设前提的要求以及局部性时间片段的重要性，使用时间序列数据挖掘的相关技术对其进行研究显得尤为重要。且一些实验也表明，**矩阵画像算法在时间序列的局部性研究上具有一定的优越性**。

贡献

本研究中获得的贡献性表现为:(1)使用了矩阵画像算法与股票预测相结合，利用矩阵画像算法，构建了基于机构交易行为下的知识库，并根据该知识库可对股票的未来趋势进行较为准确的预测。(2)将待预测股票的历史数据作为训练集，测试在确定预测时间内兴趣模式序列长度为何值时最佳， 进一步优化了预测模型，提高了预测方法的拟合效果。

Methodology

论文针对金融市场中机构交易对股票市场中的散户投资行为具有较强的误导性的现象，提出了一种基于机构交易行为影响下的趋势预测方法。首先，利用时间序列的矩阵画像方法，以股票换手率数据为切入点，构建不同兴趣模式长度下的基于机构交易行为影响的换手率波动知识库。其次，确定待预测股票在兴趣模式长度取何值时，预测结果精确度高。最后，根据该兴趣模式长度下的知识库，预测在机构交易行为影响下单支股票的波动趋势。为验证趋势预测新方法的可行性和准确性，将其与自回归滑动平均模型(ARMA 模型)和长短时记忆网络(LSTM 网络)预测方法进行对比分析，运用均方根误差(RMSE)与平均绝对百分误差(MAPE)评价指标综合比较 70 支股票预测结果。 实验结果分析表明，与 ARMA 模型和 LSTM 网络预测方法相比，在 70 支的股票价格趋势预测上，新方法有 80% 以上的股票预测结果更准确。

Matrix Profile【矩阵画像】

Matrix Profile是由UCR（加州大学河滨分校）提出的一个时间序列的分析算法。

算法简介

通过一个时间序列，可计算出它的MP，MP也是一个向量（时间序列）。

相关概念

定义 1 时间序列数据是按时间顺序排列的实数值数据，用序列 T 表示，且 $T = t_1, t_2, t_3, ..., t_n$，其中 n 是 T 的长度。

定义 2 子序列表示在原始序列 T 中截取长度为 m 的一段序列，用 $T_{i,m}$ 表示，即是从 T 中第 i 个位置开始的长度为 m 的连续子集。形式上表示为 $T_{i,m} = t_i, t_{i+1}, ..., t_{i+m-1}$，其中 $1 \leqslant i \leqslant n - m + 1$。

定义 3 距离画像 $\boldsymbol{D}$ 是时间序列 T 中不同的子序列间的距离矩阵。给定一个时间序列 T，子序列长度为 m，从 $T_{i,m}(i = 1, 2, 3, \ldots)$ 开始计算其与其他子序列片段的距离，得到一个距离矩阵 $\boldsymbol{D}$，即 $\boldsymbol{D}_i$ 是给定查询子序列与时间序列中的每个子序列之间的距离向量。

形式上表现为 $\boldsymbol{D}_i = [d_{i,1}, d_{i,2}, ..., d_{i,n-m+1}]$，其中 $d_{i,j}$ 是 $T_{i,m}$ 和 $T_{j,m}$ 之间的距离，计算子序列片段间的距离公式为：

$$d_{i,j} = \sqrt{2m\left(1 - \frac{QT_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \tag{1}$$

m 表示子序列长度，μ_i 表示子序列 $T_{i,m}$ 的均值，σ_i 表示子序列 $T_{i,m}$ 的标准差，$QT_{i,j}$ 表示子序列 $T_{i,m}$ 与子序列 $T_{j,m}$ 的点积[14]。特别地，当子序列数据经过 *zscore* 标准化后，即 $\mu = 0, \sigma = 1$ 时，公式(1)转变成：

$$d_{i,j} = \sqrt{2m\left(1 - \frac{QT_{i,j}}{m}\right)} \tag{2}$$

即标准化后的序列在计算距离画像时，只要计算好子序列间的点积便可快速得到该序列的距离画像。

定义 4 时间序列数据 $A = [a_1, a_2, ..., a_n]$ 和 $B = [b_1, b_2, ..., b_n]$，则 A 与 B 之间的点积 $\boldsymbol{QT}$ 计算公式为：

$$\boldsymbol{QT} = \sum_{i=1}^{n} a_i \times b_i \tag{3}$$

点积是在实现矩阵画像算法过程中会用到的重要公式，是用于计算矩阵画像中距离画像的重要部分。一个子序列与一条时间序列中所有子序列的点积的具体算法见[15]，该算法时间复杂度为 $O(n \log n)$，与传统的计算过程相比，计算效率显著提高。

定义 5 矩阵画像 $\boldsymbol{MP}$ 是时间序列 T 中每个子序列 $T_{i,m}$ 与其最相似片段(即距离最小值)之间的距离值组成的向量。距离画像相当于是每个子序列片段与其他所有子序列片段中的距离最小值。形式上，$\boldsymbol{MP} = [\min(\boldsymbol{D}_1), \min(\boldsymbol{D}_2), ..., \min(\boldsymbol{D}_{n-m+1})]$，其中 $\boldsymbol{D}_i (1 \leqslant i \leqslant n - m + 1)$ 是由子序列 $T_{i,m}$ 与其他所有子序列片段之间的距离所构成的向量。

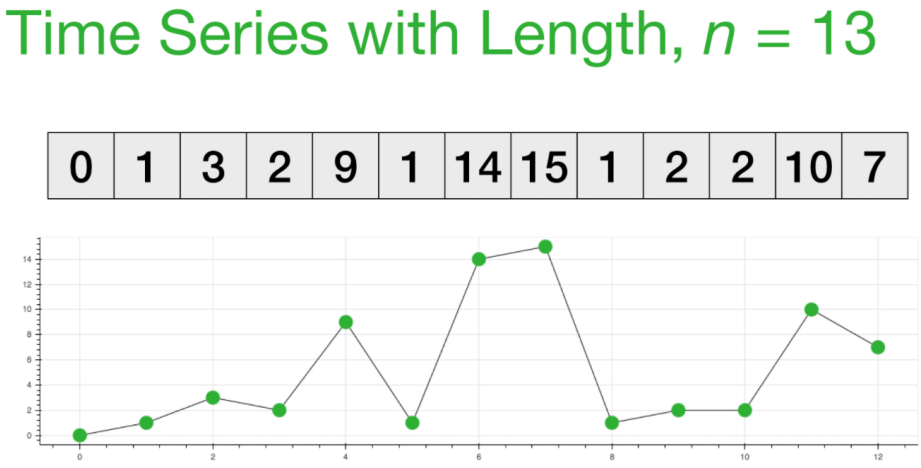
定义 6 兴趣模式(motif)是指一条或多条时间序列中最相似的子序列片段，即在每个子序列片段所对应的已是其最近距离值(即子序列的 $\boldsymbol{MP}$ 值)的情况下，再寻找 $\boldsymbol{MP}$ 中的极小值。在寻找兴趣模式之前，需先计算出要寻找模式的 $\boldsymbol{MP}$ 值，再从 $\boldsymbol{MP}$ 中获得极小值对应的模式，进而找到兴趣模式。

定义 7 矩阵画像索引是用来记录每个子序列片段的最近子序列片段所在位置的，记为 $\boldsymbol{MPI}$，其为整数向量，即 $\boldsymbol{MPI} = [I_1, I_2, ..., I_{n-m+1}]$，$I_i = \arg\min_j (D_i(j))$，$D_i(j)$ 表示距离向量 $\boldsymbol{D}_i$ 中的第 j 个距离元素。

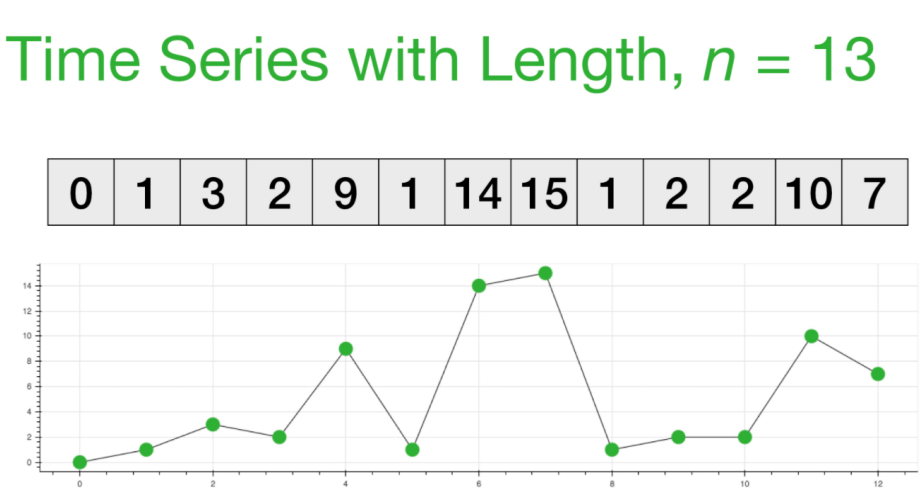
举例说明

下文的计算分解参考： https://stumpy.readthedocs.io/en/latest/Tutorial_The_Matrix_Profile.html

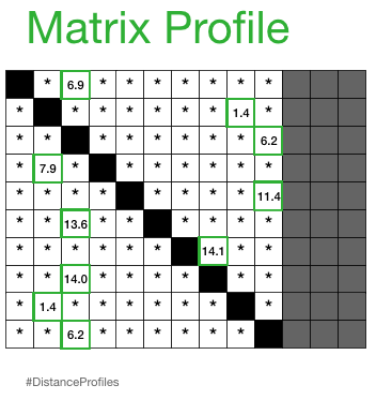
以下面的时间序列为例：[0, 1, 3, 2, 9, 1, 14, 15, 1, 2, 2, 10, 7]



比较两个子序列的距离，以欧拉距离为例：



第一个子序列向后滑动逐个计算距离，逐个计算后，会得到一个n*n的矩阵，取每行的最大值，得到的向量，即为Matrix Profile。



> 当然，并非所有这些距离都有用。 具体来说，自我匹配（或琐碎匹配）的距离并不能提供信息，因为当您将子序列与自身进行比较时，距离始终为零。 因此，我们将忽略它，相反，我们要注意的是画像的最小距离，并选择它作为我们的最佳匹配。
>
> 我们可以通过只查看每个子序列的最近邻来简化这个距离矩阵。

Exclusion Zone

在我们上面的解释中，我们只排除了琐碎的匹配(trivial match:相似度较低的片段）由于相邻两条子序列片段重叠太多，会造成时间相近的序列片段互为最相似片段，不利于兴趣模式的发现。 因此，我们需要将其扩展到相对于对角线平凡匹配的更大的“exclusion zone“（expand this to a larger “exclusion zone” relative to the diagonal trivial match）。

在实践中，已经发现i±int(np.ceil(m/4))的exclusion zone工作良好(其中m是子序列窗口大小)，并且在为第i子序列提取矩阵画像值之前，在该区域计算的距离被设置为np.inf。 因此，窗口大小越大，禁区就越大。 此外，请注意，由于NumPy索引具有包含的开始索引，但具有排他性的停止索引，因此确保对称禁区的适当方法是：

```
excl_zone = int(np.ceil(m / 4))
zone_start = i - excl_zone
zone_end = i + excl_zone + 1  # Notice that we add one since this is exclusive
distance_profile[zone_start : zone_end] = np.inf
```

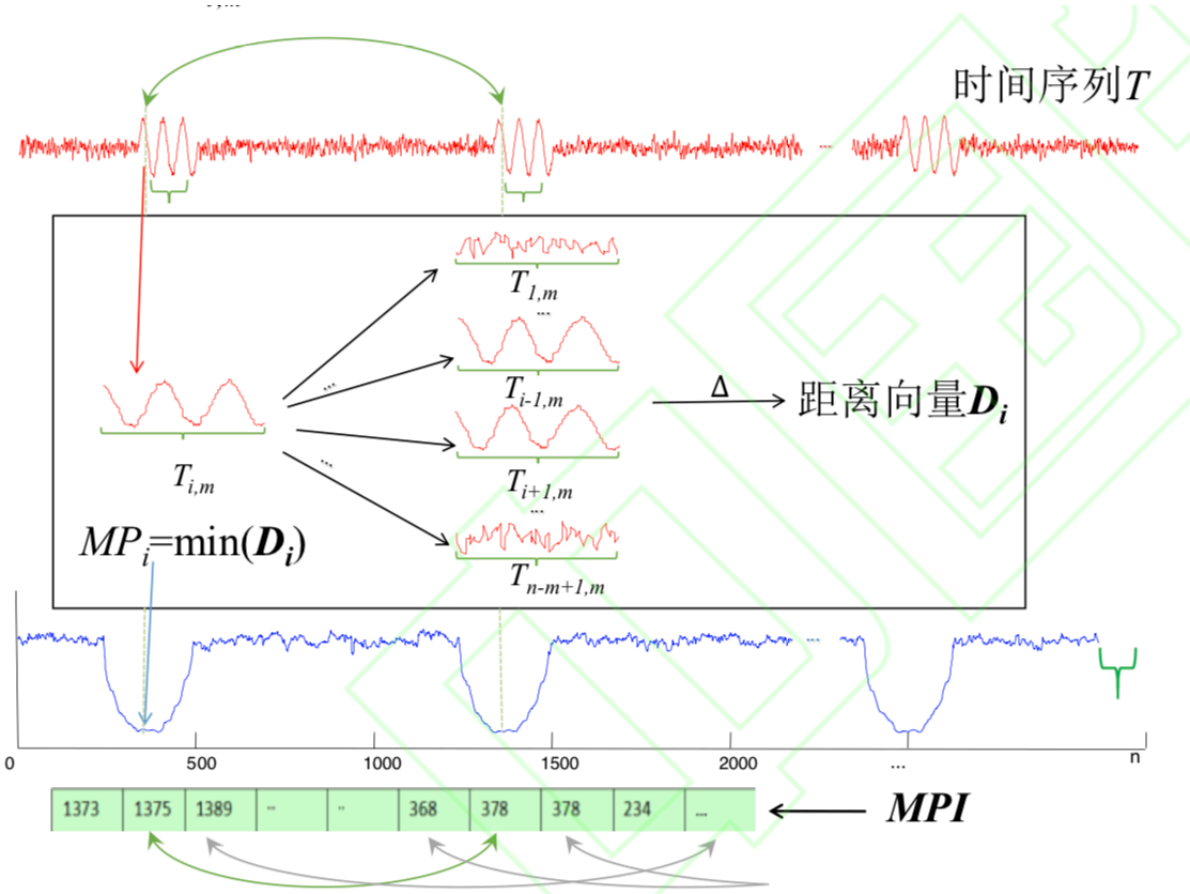


图1　Matrix Profile演算过程

Fig. 1　The calculation process of matrix profile

论文中对矩阵画像做出的改进是：1、在距离定义上作出了修改；2、由于相邻两条子序列片段重叠太多，会造成时间相近的序列片段互为最相似片段，不利于兴趣模式的发现。故在排除与 $T_{i,m}$ 重复长度超过 $m/2$ 的子序列的距离后，取距离向量D_i的最小值作为$T_{i.m}$ 的 MP 值。

股票价格波动趋势预测方法

首先，使用矩阵画像方法，以金融股票的换手率数据为切入点，分别构建不同兴趣模式（motif）长度下的基于机构交易行为影响的换手率波动知识库。其次，确定待预测股票在兴趣模式长度取何值时，预测结果精确度高。最后，基于该兴趣模式长度下的知识库，预测在机构交易行为影响下的单支股票价格波动趋势。

机构交易行为知识库

本论文旨在研究从股票总体市场出发找出机构交易行为，分析机构交易行为对个股价格波动的影响，预测个股股价波动趋势，识别机构投资者操纵股票的行为，进而降低散户投资风险和提高投资回报。论文采用换手率数据代替使用成交量反应交易的频率的和交易情况，根据换手率的高低来定义机构交易行为(Institutional Trading Behavior，ITB)，且将存在换手率大于8%的股票序列片段定义为存在机构交易行为。

构建知识库

1、处理好的股票数据用 Matrix Profile 算法找出 motif

2、剔除不存在机构交易行为的 motif，并 将剩余的 motif 前期片段、motif 片段与 motif 后续片段分别存入知识库中。

补充库

由于在预测过程中有可能会出现一些情况，即此时的股票数据序列是存在机构交易行为的，但其与知识库中的兴趣模式的匹配度并不高。若强行进行预测，预测结果有极大的概率会偏离实际结果，因此这种情况是不进行预测的。然而，存在机构交易行为的片段是值得注意的，其处理方式是先将该片段暂保存在其他数据库中，称该数据库为补充库(supDB)。补充库是将当前存在机构交易行为但与知识库中的兴趣模式匹配度不高的子序列片段进行保存，以备在完备知识库中使用。

完备知识库

补充库中的子序列片段即是知识库的完备项。当能在补充库找到兴趣模式时，说明该片段具有一定的代表性，则将该兴趣模式所对应的片段扩充到知识库中。

最佳模式匹配

对某支股票进行预测时，兴趣模式的长度不同，拟合的效果也会不同。确定兴趣模式长度与预测天数二者满足何种关系时预测效果较好是提高预测效果的手段之一，即寻找兴趣模式长度与预测天数的最佳模式匹配。本文采取的方法主要是使用历史数据进行多次训练，找出已知预测天数下拟合效果好的兴趣模式长度。

预测算法

本文主要研究的是在机构交易行为影响下的个股价格波动，但这并不是意味着在没有存在机构交易行为的情况下，新方法就不能进行预测。若待预测片段不具有机构行为，可以与知识库中保存的兴趣模式前期序列进行匹配。若匹配度高，则有很大概率认为待预测片段可能即将迎来机构交易行为;若匹配度不高时，则表示无法预测。若存在机构交易行为，则与知识库中的兴趣模式(motif)进行匹配，匹 配度高则返回未来可能的股价波动，匹配度不高则将该预测片段存入补充库中。其具体过程如下图所示:

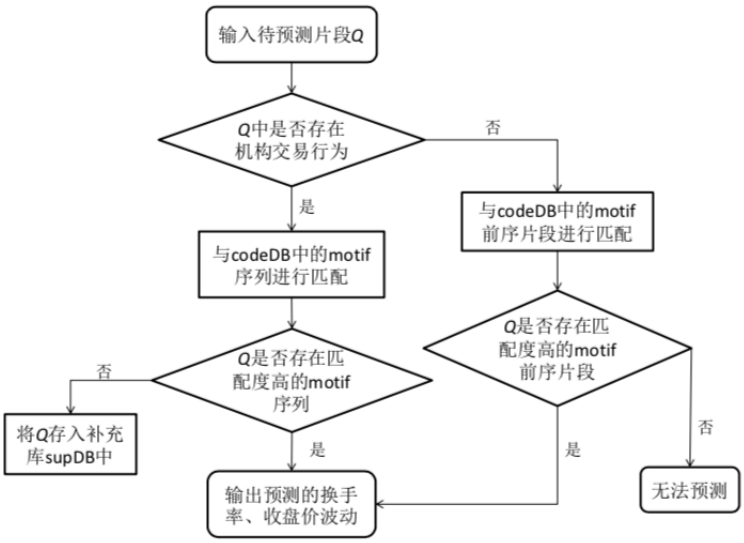


图2　基于矩阵画像的预测过程

实验结果分析

数据集

选取 2014—2018 年我国深市 A 股股票作为研究对象，并对这些数据进行整理:(a)剔除已经停市的股票;(b)剔除 2014—2018 年连续 5 天停止交易的股票;(c)剔除 2014—2018 年每年交易日期不足 180 的股票。整理得到 70 支股票样本数据。

Conclusion & Future Work

根据深市 A 股股票的换手率数据，使用 stomp 算法获取具有机构交易行为的兴趣模式片段，构建完备知识库，进而提出基于知识库中兴趣模式的单支股票的金融股票价格趋势波动预测方法。针对某支股票，根据股票的历史换手率数据，收盘价数据以及待预测天数，筛选出基于历史数据具有最佳预测效果的兴趣模式长度，从而进行未来几天的股价趋势预测。

在时间效率方面，由于前期要对 70 支股票数据使用 Matrix Profile 算法建立不同兴趣模式长度的知识库，**数据量大，算法的时间复杂度也不低，且后期预测时需进行多次的模拟训练， 故耗费时间较长**。

在应用方面，在已构建好知识库的情况下， 对在构建知识库的过程中应用到的所有股票都可使用 MPP 方法进行股价趋势预测，说明该方法具有**相对的普遍应用价值**。新方法 MPP 与 ARMA 模型和 LSTM 网络的预测结果相比较，实验结果表明，**基于矩阵画像的金融股价波动预测效果较好**。另外， 通过研究获得的信息和知识可以降低机构交易行为对散户的影响，帮助散户们在市场中获取较稳定的收益。同时，帮助金融市场监管部门对股价进行监控预测，防范可能出现的股价波动异常。

局限性在于，在确定兴趣模式的最佳长度时，主要通过进行多次模拟预测实验，取多次预测结果拟合值的最小均值所对应的兴趣模式长度。该过程并不能保证每次所取的兴趣模式长度是最佳的，故针对兴趣模式长度的分析是未来值得研究的问题。

STUMP Basics：Analyzing Motifs and Anomalies with STUMP

STUMP是Numba JIT编译版的流行的STOMP算法，在最初的Matrix Profile II论文中有详细的描述。STUMP能够并行计算，它在指定的时间序列内对模式和异常值进行有序搜索，并利用一些计算的局部性来最小化运行时。

数据集

1. The Steamgen dataset
2. The NYC taxi passengers dataset

Tutorial Code

1、加载Steamgen数据集

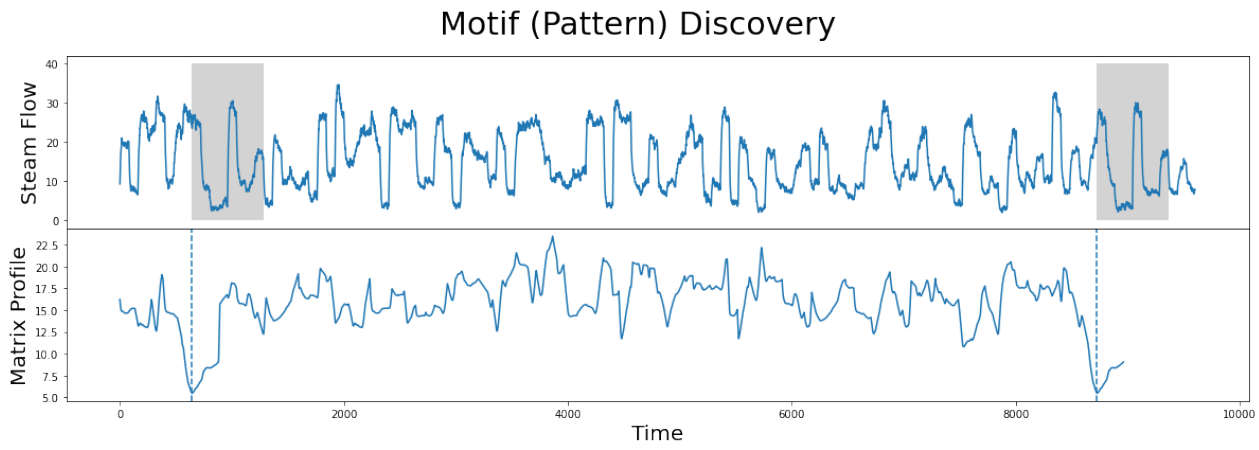
该数据是使用模糊模型生成的，该模型用于模拟伊利诺伊州尚佩恩的雅培电厂的蒸汽发生器。 我们感兴趣的数据特征是是输出蒸汽流量遥测，单位为kg / s，每三秒钟对数据进行“采样”，总共有9,600个数据点。

```
colnames = ['drum pressure',
            'excess oxygen',
            'water level',
            'steam flow'
           ]

context = ssl.SSLContext()  # Ignore SSL certificate verification for
simplicity
url = 'https://www.cs.ucr.edu/~eamonn/iSAX/steamgen.dat'
raw_bytes = urllib.request.urlopen(url, context=context).read()
data = io.BytesIO(raw_bytes)
steam_df = pd.read_csv(data, header=None, sep="\\s+")
steam_df.columns = colnames
steam_df.head()
```

2、数据可视化

我们可以看到，矩阵轮廓中的全局最小值（垂直虚线指向的时间点）对应于组成基序对的两个子序列的开始位置。 这两个子序列之间的z-normalied 欧几里得距离为：

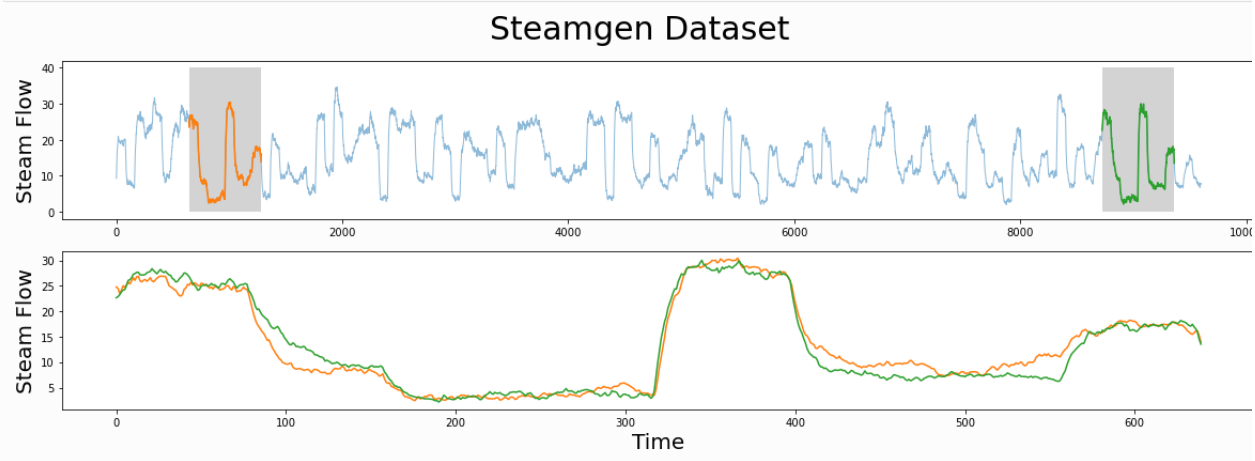


Take a moment and carefully examine the plot above with your naked eye. If you were told that there was a pattern that was approximately repeated, can you spot it? Even for a computer, this can be very challenging. Here's what you should be looking for:

3、人为查找Motif

```
m = 640
fig, axs = plt.subplots(2)
plt.suptitle('Steamgen Dataset', fontsize='30')
axs[0].set_ylabel("Steam Flow", fontsize='20')
axs[0].plot(steam_df['steam flow'], alpha=0.5, linewidth=1)
axs[0].plot(steam_df['steam flow'].iloc[643:643+m])
axs[0].plot(steam_df['steam flow'].iloc[8724:8724+m])
rect = Rectangle((643, 0), m, 40, facecolor='lightgrey')
axs[0].add_patch(rect)
rect = Rectangle((8724, 0), m, 40, facecolor='lightgrey')
axs[0].add_patch(rect)
axs[1].set_xlabel("Time", fontsize='20')
```

```
axs[1].set_ylabel("Steam Flow", fontsize='20')
axs[1].plot(steam_df['steam flow'].values[643:643+m], color='C1')
axs[1].plot(steam_df['steam flow'].values[8724:8724+m], color='C2')
plt.show()
```



我们正在寻找的主题（模式）在上方突出显示，但是仍然很难确定橙色和绿色子序列是否匹配（上图），也就是说，直到我们放大它们并覆盖子序列为止彼此顶部（下部面板）。 现在，我们可以清楚地看到主题非常相似！ 计算矩阵轮廓的基本价值在于，它不仅可以让**您快速找到基序，而且还可以确定时间序列内所有子序列的最近邻**。 请注意，除了从原始序列中抓取位置并将其绘制出来之外，我们在此处实际上并未进行任何特殊的定位主题。 现在，让我们获取蒸汽数据并对其应用stump函数。

Find a Motif Using STUMP（利用MP的最小值）

```
m = 640
mp = stumpy.stump(steam_df['steam flow'], m)
```

stump函数需要有两个参数：

1、一段时间序列

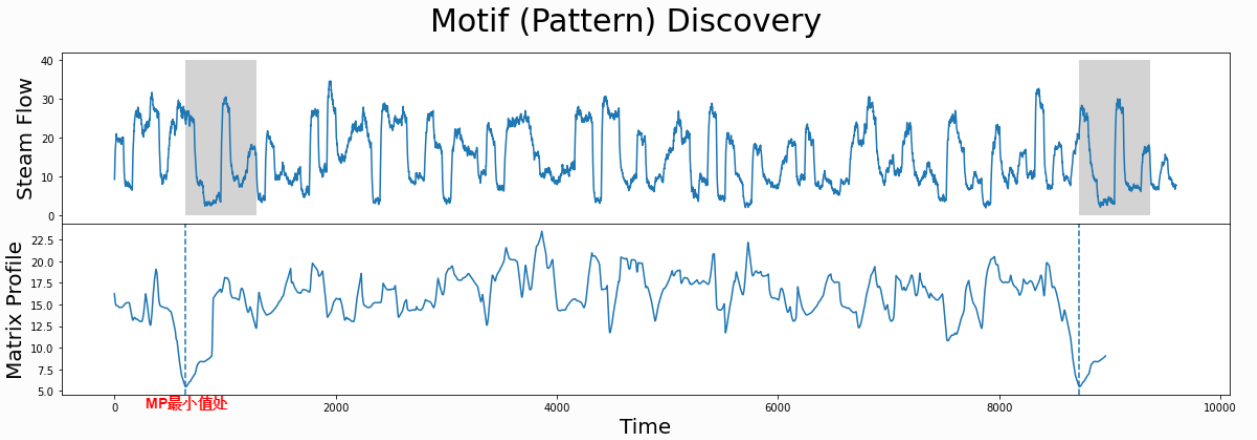
2、窗口大小m

在案例中，我们选择了640作为窗口大小，大约等于半小时窗口（half-hour windows）。 同样，树桩的输出是一个数组，该数组分别包含第一和第二列中的所有矩阵画像值MP（即，到您最近邻居的z标准化欧几里得距离）和矩阵画像索引IMP（我们将忽略 第三列和第四列）。 让我们在原始数据旁边绘制矩阵画像：

```
fig, axs = plt.subplots(2, sharex=True, gridspec_kw={'hspace': 0})
plt.suptitle('Motif (Pattern) Discovery', fontsize='30')

axs[0].plot(steam_df['steam flow'].values)
axs[0].set_ylabel('Steam Flow', fontsize='20')
rect = Rectangle((643, 0), m, 40, facecolor='lightgrey')
axs[0].add_patch(rect)
rect = Rectangle((8724, 0), m, 40, facecolor='lightgrey')
axs[0].add_patch(rect)
axs[1].set_xlabel('Time', fontsize ='20')
```

```
axs[1].set_ylabel('Matrix Profile', fontsize='20')
axs[1].axvline(x=643, linestyle="dashed")
axs[1].axvline(x=8724, linestyle="dashed")
axs[1].plot(mp[:, 0])
plt.show()
```



我们可以看到，矩阵轮廓中的全局最小值（垂直虚线指向的时间点）对应于组成基序对的两个子序列的开始位置。 这两个子序列之间的z-normalied 欧几里得距离为：

```
mp[:, 0].min()
```

5.491619827769512

即使我们看到两个子序列不是完全匹配，该距离不为零，但是相对于矩阵画像的其余部分（即，与均值或中值矩阵画像值相比），我们可以看到这个有趣模式motif匹配的非常好。

Find a Anomaly Using STUMP（利用MP的最大值）

相反，矩阵配置文件中的最大值为：

```
mp[:, 0].max()
```

23.47616836730203

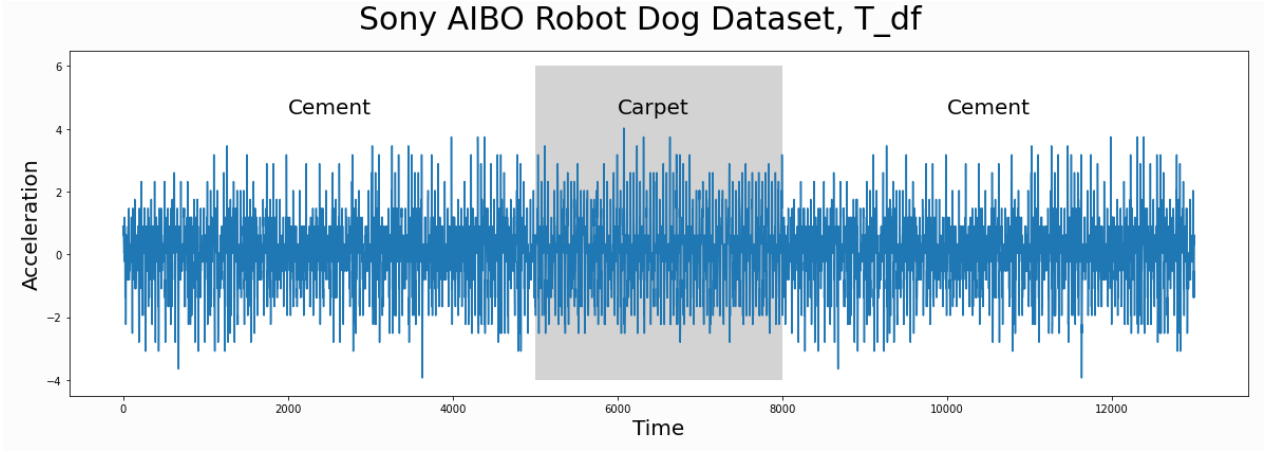
矩阵画像索引还告诉我们时间序列中的哪个子序列**不具有与其自身相似的最近子序列**：【即异常处】

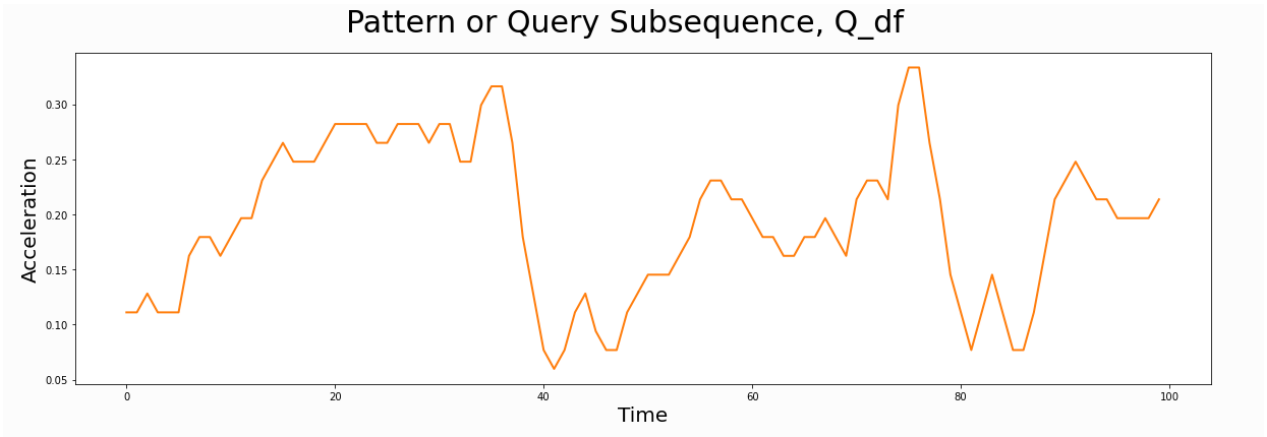
```
np.argwhere(mp[:, 0] == mp[:, 0].max()).flatten()[0]
```

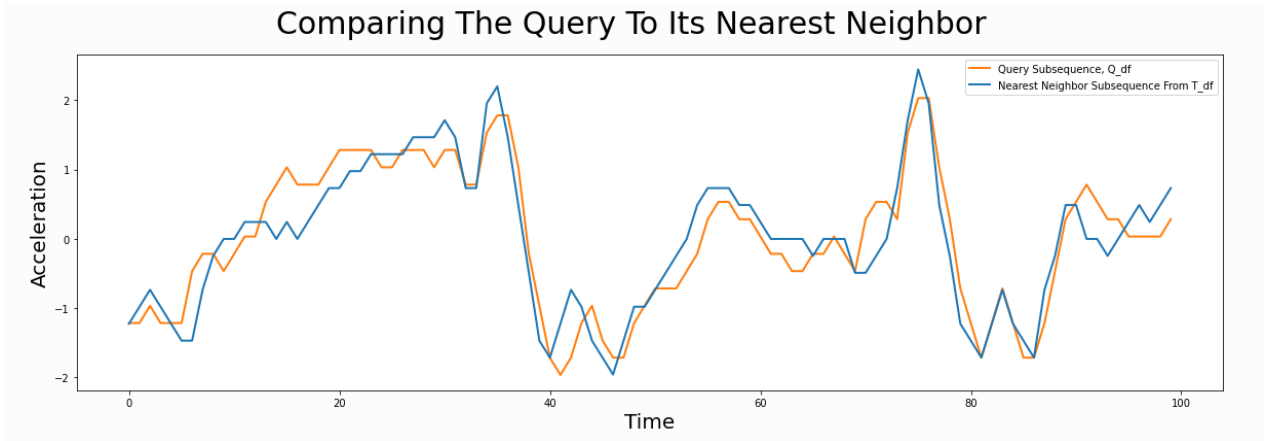
快速模式识别

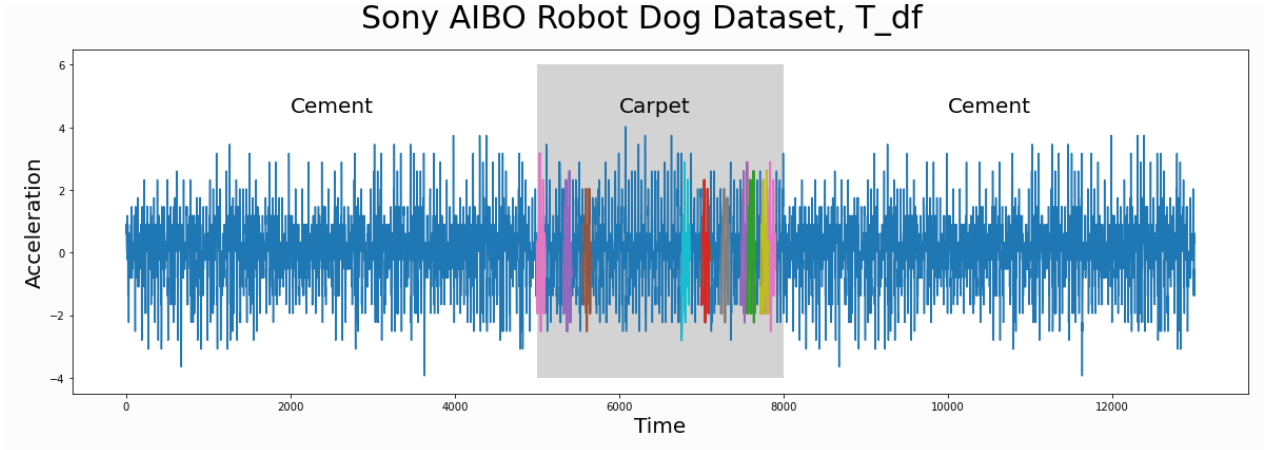
https://stumpy.readthedocs.io/en/latest/Tutorial_Pattern_Searching.html

输入一个查询序列，与一段时间序列，查询最为相近的时间序列。


Sony AIBO Robot Dog Dataset, T_df
Cement
Carpet
Cement
Acceleration
Time


Pattern or Query Subsequence, Q_df
Acceleration
Time


Comparing The Query To Its Nearest Neighbor
Query Subsequence, Q_df
Nearest Neighbor Subsequence From T_df
Acceleration
Time


Sony AIBO Robot Dog Dataset, T_df
Cement
Carpet
Cement
Acceleration
Time