Kioshiro

PyEnergy文档 0.1

PyEnergy文档

1. check.py

负责数据格式检查、匹配和转换，将输入的csv文件转为需要的DataFrame格式。

1.1 `check_dataparser(in_file)`

检查输入的`in_file`第一列的时间序列的格式，输出需要的数据格式`dataparser_str`。

符合需要的数据格式有：

- `'%Y-%m-%d %H:%M:%S'`
- `'%m.%d.%Y %H:%M'`
- `'%m.%d.%y %H:%M'`

对于不符合需要的格式，会报错`"无法识别的日期格式"`。

1.2 `chek_nr_transformer(path_to_data)`

(暂时无用)

1.3 `import_transformer_data(data_files)`

将输入的`data_files`转为`Dataframe`格式的`df`，其中时间序列`T`为索引，列名为：

```
param_names = ['T',
                   'volt_A', 'volt_B', 'volt_C', 'volt_AB', 'volt_CA', 'volt_BC',
                   'curnt_A', 'curnt_B', 'curnt_C',
                   'realP_A', 'realP_B', 'realP_C', 'realP_tot',
                   'reacP_A', 'reacP_B', 'reacP_C', 'reacP_tot',
                   'aprntP_A', 'aprntP_B', 'aprntP_C', 'aprntP_tot',
                   'factor_A', 'factor_B', 'factor_C', 'factor_tot']
```

并对功率数据进行单位转换：瓦特到千瓦，以及将功率因数转为正值

2. cluster.py

负责编写聚类方法。

2.1 带 `penal` 的Kmeans

2.1.1 `compute_silhouette_scores(data, max_clusters, repeat, metric='euclidean', penal=0)`

SI评价指标。使用`KMeans(n_clusters=n_clusters, random_state=43, n_init="auto")`对`data`进行聚类后使用`silhouette_score`对聚类结果评分。score的计算公式为：

$$Score = SI - \lambda\frac{n}{N}$$

其中：

- SI: `silhouette_score(data, cluster_labels, metric=metric)`，SI指标
- λ: `penal`，惩罚系数
- n: `n_clusters`，聚类数目
- N: `len(data)`，样本数

`repeat`控制每个`n_clusters`中`Kmeans`聚类的次数，score取`repeat`次中的最大值。

`metric`为SI指标使用的距离度量。

2.1.2 `kmeans_elbow_core1(selected_features, repeat, plot, normalize=False, max_clusters=None, metric='euclidean',penal=0):`

为该聚类的核心代码。如果`normalize=True`，会先对`selected_feature`进行`Max-Min`标准化。如果`max_cluster=None`，max_cluster的值为：

$$MaxCluster = \lceil\sqrt{N}\rceil$$

其中:

- N：`selected_features.shape[0]`，样本数。

之后，将参数传给`compute_silhouette_scores(selected_features, max_clusters, repeat=repeat, metric=metric, penal=penal)`。

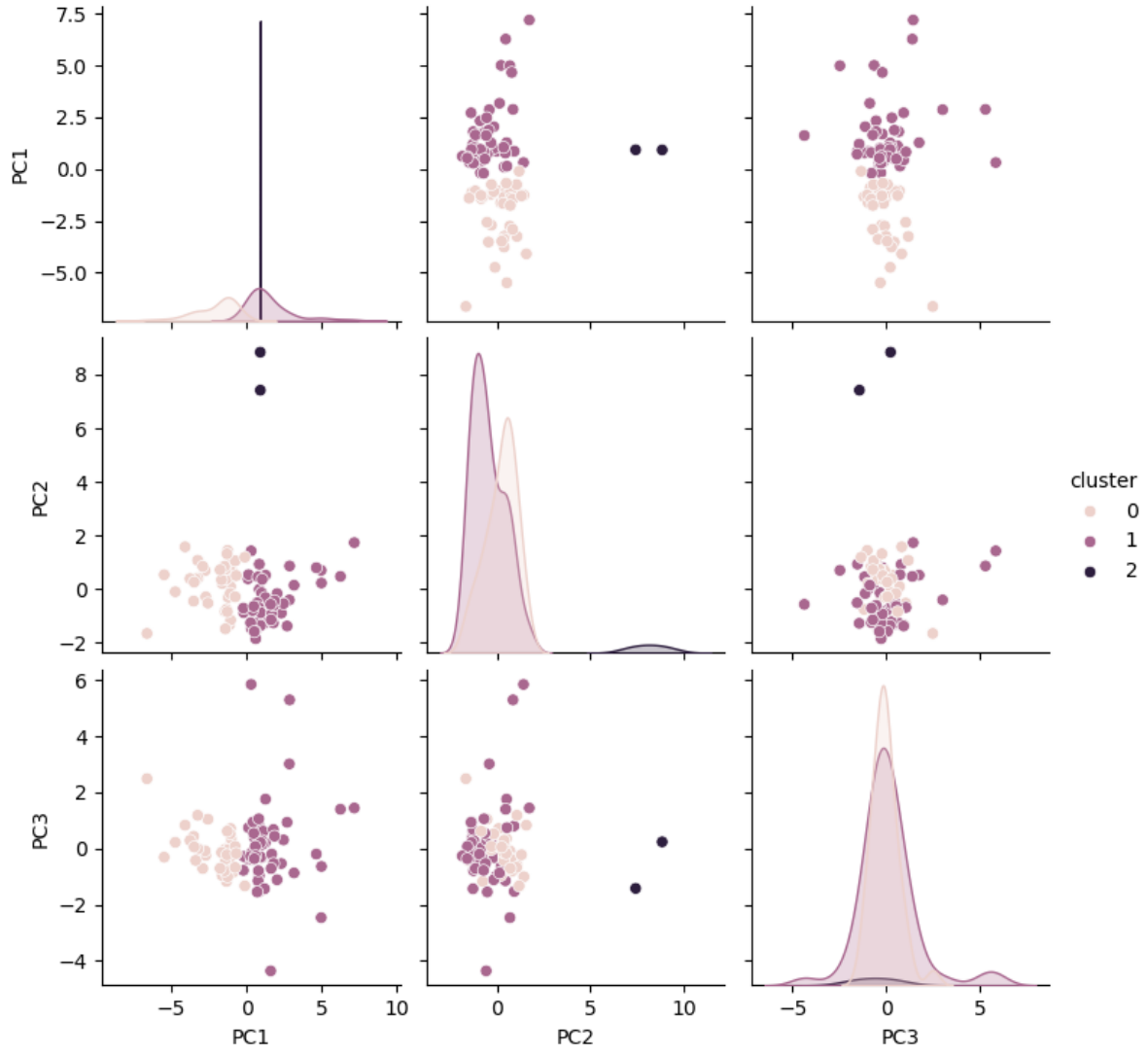
如果`plot=True`，会调用draw_silhouette_scores绘制SI指标折线图。

函数返回得分最高的聚类数和得分。

2.1.3 `kmeans_elbow1(feature, feature_info, repeat=5, plot=False, metric='euclidean',penal=0):`

该方法的用户调用接口。

如果`plot=True`，会绘制`feature_info`之间的`pairplot`图，便于观察聚类效果。



pairplot示例

2.2 标准差带权的Kmeans

2.2.1 `compute_score(data, labels, weights, metric)`

该方法的SI指标计算方法。该方法不使用`silhouette_score`直接计算总体SI指标，而是使用`silhouette_samples`计算各个数据的SI指标，之后计算SI指标的均值和各类的标准差的平均值。该方法的Score计算公式为:

$$Score = w_0\mu + w_1\sigma$$

其中：

- w_0, w_1：分别为`weight[0]`和`weight[1]`
- μ, σ：SI指标的均值，各类标准差的平均值

2.2.2 `kmeans_elbow_core(data, max_clusters, repeats, weights, plot=True, metric='euclidean', n_init="auto")`

该聚类方法的核心代码。与2.1.2类似。使用2.2.1计算得分。如果`plot=True`，绘制各聚类个数的得分折线图。

2.2.3 `kmeans_elbow(selected_features, max_clusters=None, repeats=5, weights=[0.5, 0.5], plot=True, metric='euclidean')`

为 2.2.2 提供默认值。增加`max_clusters=None`的处理逻辑，与 2.1.2 一致。

2.2.4 `kmeans(feature, feature_info, max_clusters=None, repeat=5, plot=True, weight=[0.5, 0.5], metric='euclidean'):`

该方法的用户调用接口。

3. compute.py

负责提供一些简单计算函数的接口。

3.1 `calc_dominant_bin(inArray, bin_interval)`

（暂时无用）

3.2 `standard(data)`

对`data`进行`Z-score`标准化。

3.3 `normalize(data)`

对`data`进行`Min-Max`标准化。

4. core.py

pyEnergy的核心处理逻辑。

4.1 `find_all_events(df, thre_val=3, thre_time=10, param_str = 'curnt_B')`

根据实际功率使用情况获取事件。

- `thre_val`: 电流阈值，低于此值的数据将被视为无效。
- `thre_time`: 事件持续时间的最小阈值（分钟）。
- `param_str`: 获取事件依据的信号参数。默认为`'curnt_B'`，在论文中有提及，该参数受其他电器信号影响最小。

前两个参数设置的依据：

> 原始数据显示， 当有电器开启时，总实际功率会从很小的值（接近零） 跃升到至少 3 千瓦 。此外，一次典型的灌溉操作至少持续 10 分钟。 因此， 设定了第一条选择标准：
>
> - 总实际功率大于 3 千瓦， 活动持续时间超过 10 分钟（超参数）

4.2 `extract_monotype_events(events_all, thre_val=3, param_str='curnt_B')`

将所有有效事件分为"单一类型事件"和其他事件。

4.3 `estimate_total_power(event)`

估计泵的总功率，包括有功功率和无功功率。

计算公式如下：

$$PF_{average} = \frac{PF_B + PF_C}{2}$$
$$I_{average} = \frac{I_B + I_C}{2}$$
$$V_{average} = \frac{V_A + V_B + V_C}{3}$$
$$P'_{total} = PF_{average} \times I_{average} \times V_{average} \times 3$$

4.4 `def compute_features(monotype_events, feature_info=None)`

计算提取的单一事件特征。

特征如下：

```
    if feature_info is None:
        feature_info = [
            'std. real power(ss)', 'ave. real power(ss)', 'trend real power(ss)', 'max. real
power(tr)',
            'std. reactive power(ss)', 'ave. reactive power(ss)', 'trend reactive power(ss)',
'max. reactive power(tr)',
            'std. phase B current(ss)', 'ave. phase B current(ss)', 'trend phase B
current(ss)', 'max. phase B current(tr)'
        ]
```

我们设置暂态的时间步数为5

5. drawer.py

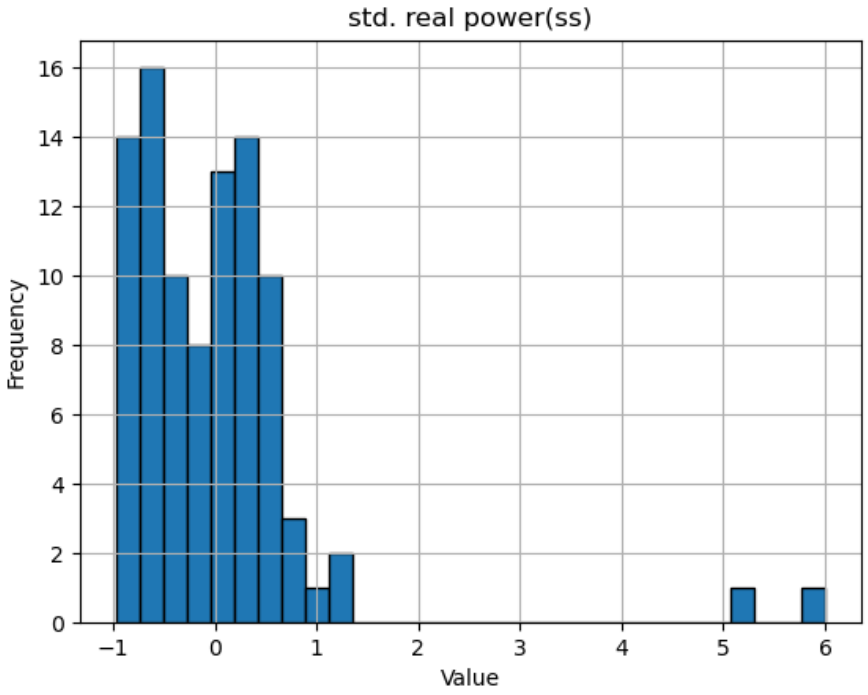
负责绘制需要的科研图像。

5.1 `draw_3D_scatter(feature_standardized, feature_info)`

(暂时无用)

5.2 `draw_feature_hist(feature_standardized, feature_info)`

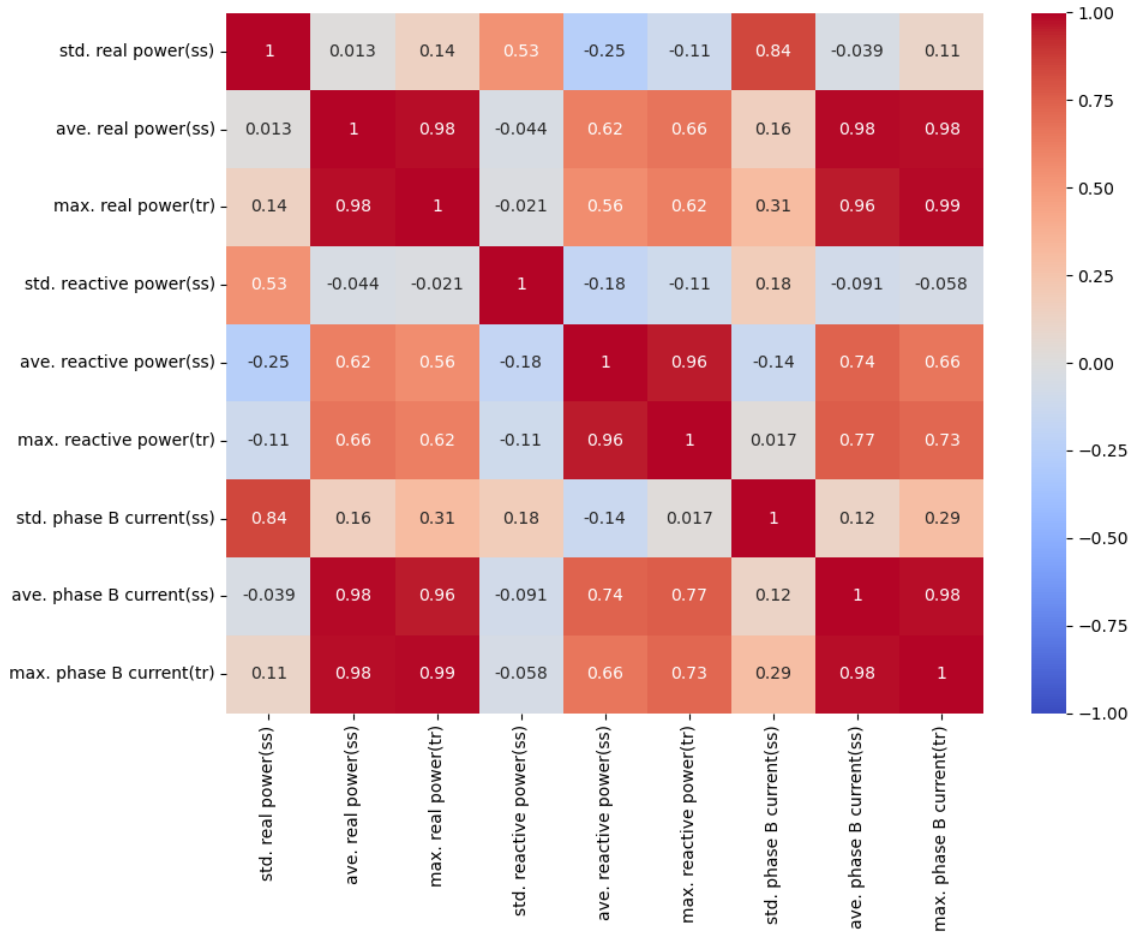
绘制各个特征的直方图。该方法不对数据进行标准化，所以请标准化后再使用此函数。



示例图像

5.3 `draw_corr(feature)`

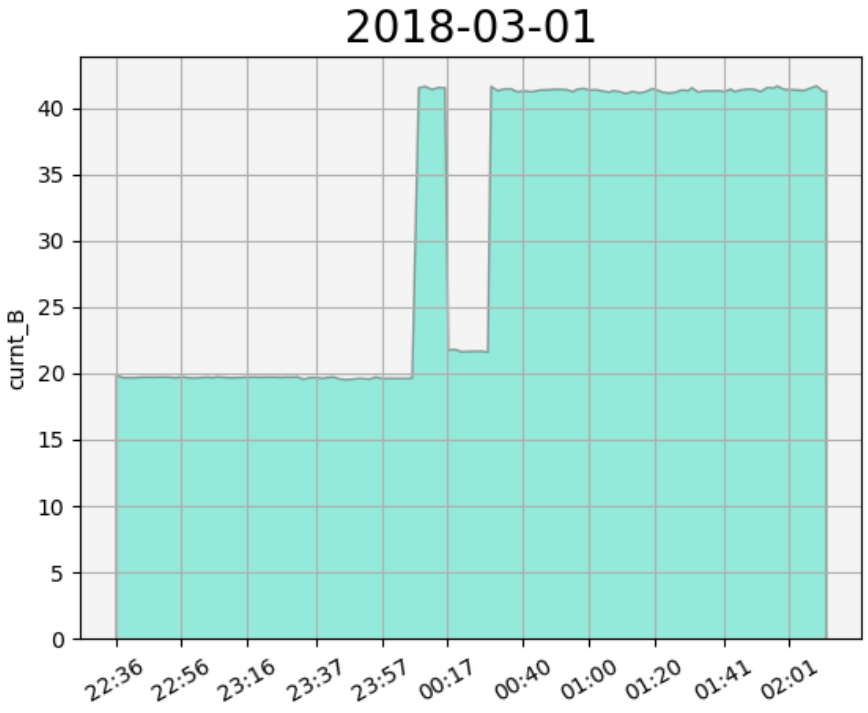
绘制特征的协方差矩阵热力图。



示例图像

5.4 `draw_signal(data, param_str="currnt_B", ax=None)`

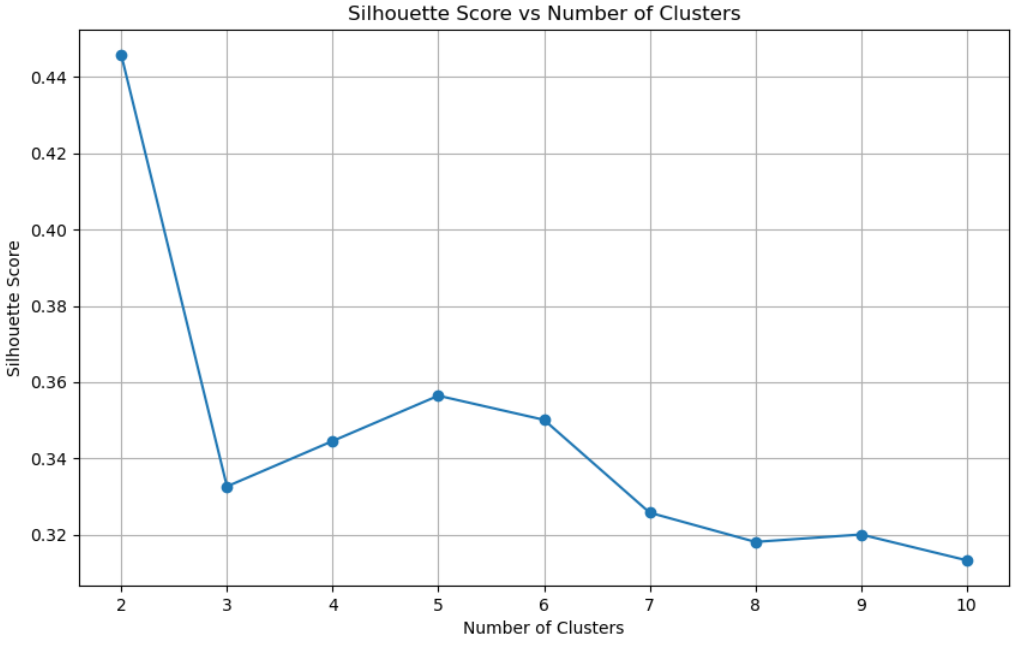
绘制信号的面积图。其中如果`ax!=None`，即如果自己定义了如`fig, ax = plt.subplots(2,2)`的`ax`对象，可以将绘制的图像传到该`ax`对象，从而在一张图像上展示多张信号图。



示例图像

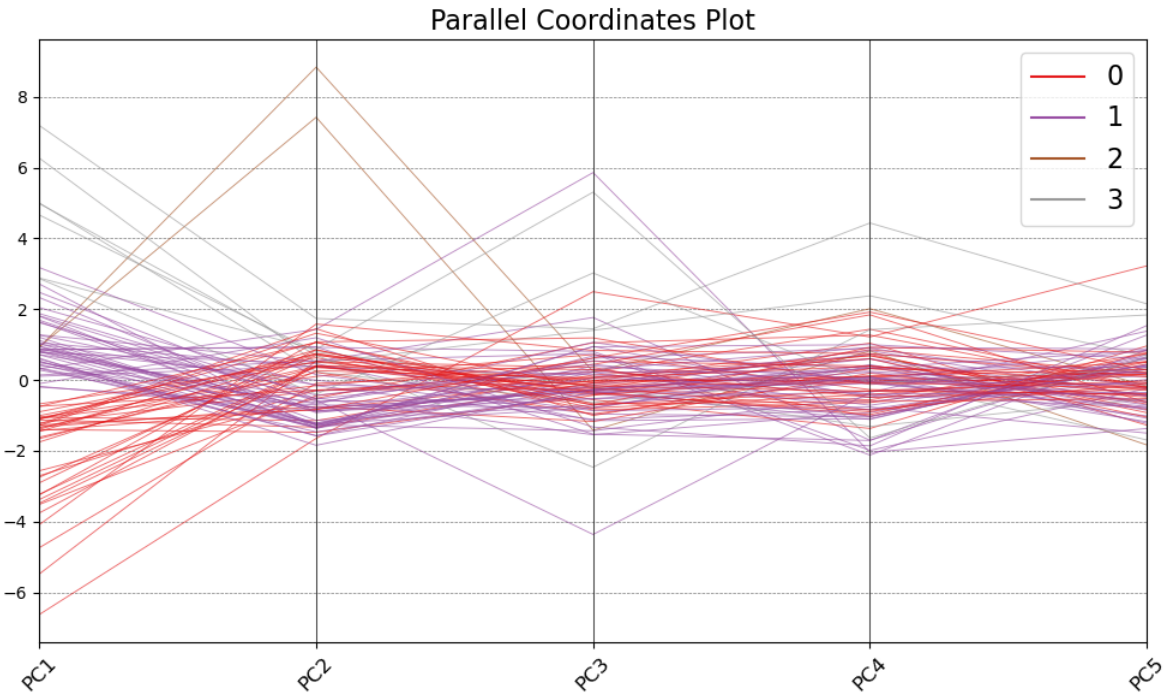
5.5 `draw_silhouette_scores(max_clusters, silhouette_scores)`

绘制SI指标折线图，在2.1.2使用。



5.6 `draw_parallel_coordinates(data, y_pred, colormap="tab10")`

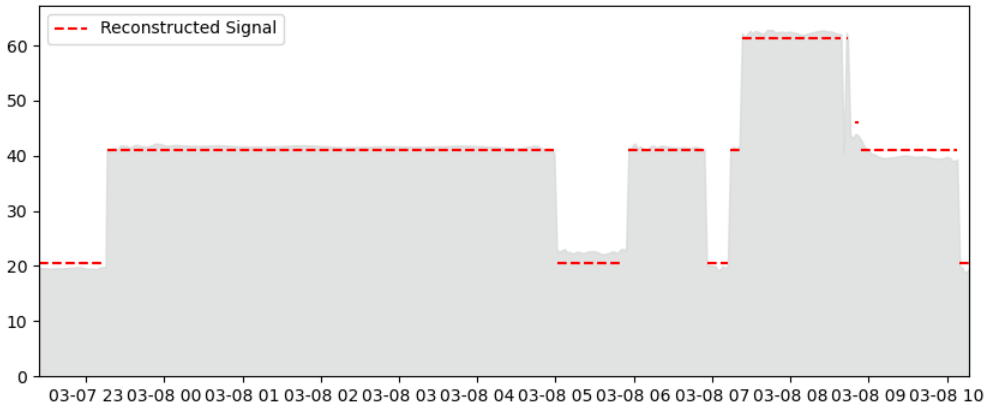
绘制平行坐标图。



示例图像

5.7 `plot_continuous_lines(original_signals, reconstruct_signals, x_values=None)`

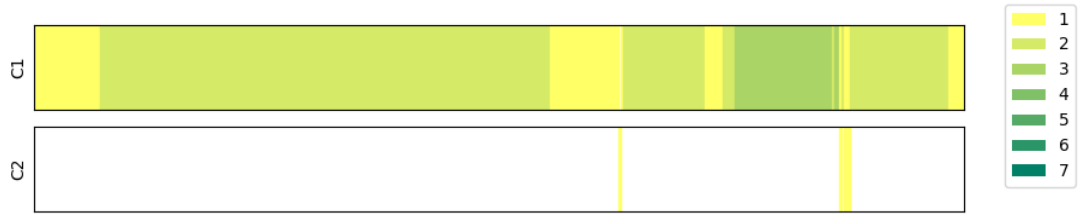
绘制信号面积堆叠图和重组信号的连续信号线。



示例图像

5.8 `plot_running_time(n_clusters, sols, cmap="viridis", n_color=15, color_reverse=False)`

绘制运行时间表示图。



示例图像

5.9 `draw_continue_line_and_running_time`

```
def draw_continue_line_and_running_time(signals, reconstruct_signals, n_clusters, sols,
x_values=None,
                                        cmap="summer", n_color=10, color_reverse=False,
                                        save=None, plot=True):
```

5.7和5.8的合并调用接口。

如果`save!=None`，那么会将5.7和5.8生成的图像保存为`save.split(".")[0]+"CL."+save.split(".")[-1]`和`save.split(".")[0]+"RT."+save.split(".")[-1]`

6. feature_selector.py

设计选取更好特征的特征选择器。

6.1 `filter_based_selection(features, threshold=0.1)`

(暂时不用)

6.2 `pca_based_selection(features, n_components=5)`

使用pca进行特征选择。

6.3 `correlation_based_selection(features, threshold=0.9)`

基于相关性进行特征选择。

7. final.py

对信号进行平滑，以及对信号进行MILP。

正在优化。

7.1 信号平滑方法

7.1.1 `moving_average(signal, window_size=5)`

7.1.2 `gaussian_smoothing(signal, sigma=1)`

7.1.3 `wavelet_smoothing(signal, wavelet='db4', level=None, threshold=None)`

`threshold` 默认为最后一个小波的中位数的3σ值。

7.1.4 `reduce_signal1(signals, threshold=5)`

对信号进行平滑处理，这里使用 Savitzky-Golay 滤波器来代替 MATLAB 的平滑

7.2 MILP以及其他处理

7.2.1 `reconstruct_time_series(sols, phaseB_perCluster)`

根据MILP的结果，重组信号。

7.2.2 `signal_composition_opt1(realP_perCluster, signal_reduced, low_bound=0, up_bound=1)`

MILP处理算法。使用的优化目标是最小化误差。

7.2.3 `signal_composition`

```
def signal_composition(fool, event_param, feature_param, reduce=None, reduce_params={},
                       low_bound=0, up_bound=None,
```

```
                    cmap="summer", n_color=10, color_reverse=False,
                    plot=True, plot_original=True, save=None, **kargs):
```

信号重组处理的用户调用接口。绘制CL图和RT图。

`up_bound`设置单一事件最多使用次数，`n_color`为`cmap`切割后产生的颜色数，较小可能报错（数组越界）。`plot=True`时，绘制图像。

`reduce`设置信号平滑方法，默认时使用reduce_signal1,可选项有[`"gaussian" | "moving" | "wavelet" | ""`]，`""`同默认值。

`plot_original=True`时，绘制原始信号图，即调用draw_signal。

8. fool.py

为用户提供像"傻瓜"(fool)一样的使用体验。

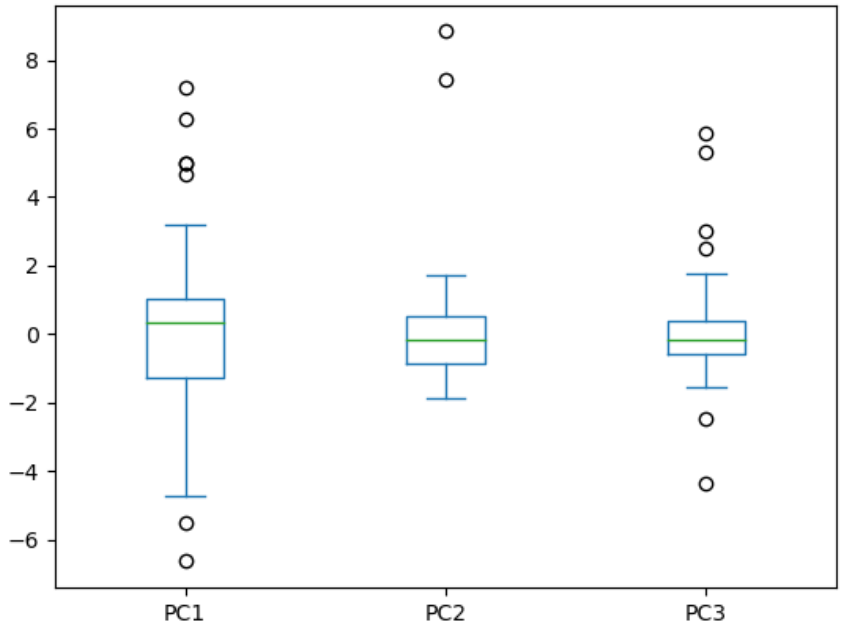
8.1 class Fool

8.1.1 `features(self)`

返回`self.feature, self.feature_info`，为需要同时提供特征和使用的特征名称的函数传参提供便利。

8.1.2 `box(self)`

绘制特征的箱图，一般为初始化（标准化和特征选择后）后调用。



8.1.3 `feature_selection(self, remove_feature=[], method=None, selection_params={}, normal=True)`

搭配initialize使用的特征选择调用接口。`remove_feature`为手动删除的特征列表，优先度大于`method`。`method`为特征选择方法。`normal=True`时，优先对特征进行标准化。

可以选择的`method`将在`feature_selection_core`中说明。

8.2 其他函数

8.2.1 `initialize_with_feature_selector(csv, remove_feature=[], select_feature=None, method=None, selection_params={}, normal=True)`

使用特征选择器的初始化函数。初始化时将调用core.py中的所有函数，返回一个`Fool`实例。

8.2.2 `initialize(csv, normal=True)`

不带特征选择器的初始化函数，返回一个`Fool`实例。

8.2.3 `feature_selection_core(feature, feature_info, method=None, selection_params={})`

特征选择的处理核心逻辑。

可选择的特征选择方法目前有[`"pca"|"corr"`]。

- "pca"：使用PCA对特征进行降维处理。可选的`selection_params`为`n_compnents`，是降维后的特征维数，默认值为5。
- "corr"：使用相关性进行降维处理。可选的`selection_params`为`threshold`，默认值为0.9。