DataPreprocessing

数据预处理工具集

https://github.com/Wddzht/DataPreprocessing.git

目录结构：

1. 数据清洗 /DataCleaning
    1. 空值处理 MissingDataHandle.py
        1. 删除 delete_handle(data_class, handel_index)
        2. 中位数插补 median_interpolation_handle(data_class, handel_index)
        3. 众数插补 mode_interpolation_handle(data_class, handel_index)
        4. 均值插补 mean_interpolation_handle(data_class, handel_index)
        5. 固定值插补 fixed_value_padding_handle(data_class, handel_index, padding_value)
        6. 间值法插补 mid_interpolation_handle(data_class, handel_index)
        7. 线性回归法
        8. 拉格朗日法
    2. 异常值 OutlierHandle.py
        1. Z-Score法 z_score_detection(data_class, handel_index, z_thr=3.0)
2. 数据集成 /Discretization
    1. 合并
    2. 去重
3. 数据变换 /DataTransformation
    1. 函数变换
    2. 归一化 NormalizeHandle.py
        1. 离差标准化 min_max_normalize(data_class, handel_index)
        2. 反离差标准化 anti_min_max_normalize(data_class, handel_index)
    3. 标准化 StandardizationHandle.py
        1. 标准化 standardization(data_class, handel_index)
        2. 反标准化 anti_standardization(data_class, handel_index)
4. 数据规约 /DataReduction
    1. 属性选择 RoughSetAttrSelecter
        1. 基于粗糙集理论的属性选择 attribute_select(data_class)
5. 数据离散 /DataIntegration
6. 数据集结构 DataClass.py
    1. 数据读取 read(self, path, has_head, split_tag='\t')
    2. 数据格式转换 parse(self)
    3. 打印 print(self)
7. 日志记录 LogHelper.py

四、数据规约 /DataReduction

4.1 属性选择 RoughSetAttrSelecter.py

4.1.1 基于粗糙集理论的属性选择 attribute_select(data_class)

Workflow:

1. 计算数据的CORE属性集: `core = get_core(dc)` ,

2. 以CORE中的属性作为初始的属性选择,检查在选定的属性集下,是否存在不可区分集(即计算选定属性集在所有决策属性的下近似)
   `check_distinct(dc, core, considered_instence)`

$$Initial\ SelectedAttrs = \{CORE\}$$

$$POS_{\{SelectedAttrs\}}(D) = U_{CX}$$

3. 若存在不可区分集,进行下一步迭代.若不存在不可区分集,则CORE就是Reduct.
4. 从剩下的属性中依次选取一个属性a_i加入:

$$SelectedAttrs = \{CORE\} + \{a_i\}$$

   检查在选定的属性集下,是否存在不可区分集.
5. 若存在一个或多个个数相同的属性集,都不会产生不可区分集,则在这些属性中选取值种类数较少的属性加入并返回SelectedAttrs,并返回SelectedAttrs,程序运行结束.
6. 若对于所有的$\{CORE\} + \{a_i\}$属性集,都会产生不可区分集(没有 Reduct)),则选择区分集个数最少的属性进入下一轮迭代(第4步).

Case 1:
data: (其中a,b,c,d为条件属性, E为决策属性)

U	a	b	c	d	E
u_1	1	0	2	1	1
u_2	1	0	2	0	1
u_3	1	2	0	0	2
u_4	1	2	2	1	0
u_5	2	1	0	0	2
u_5	2	1	1	0	2
u_5	2	1	2	1	1

```
# 方法测试
dc = DataClass.DataClass([str] * 5, data)
core = get_core(dc)
assert core == [1]  # CORE(cd)=1 (the second attr)

considered_instence = np.array([True] * dc.len, np.bool)
is_reduct, classify_num, considered_instence = check_distinct(dc, core, considered_instence)
assert (is_reduct, classify_num, considered_instence) == (False, 1, [False] * 2 + [True] * 5)

# 属性选择
selected_attr, max_classify_num = attribute_select(dc)
assert selected_attr == [1, 3]  # 选择{b,d}作为约简后的属性集
```

方法说明:
其中 `get_core(data)` 方法通过构造 Discernibility Matrix 的方法选出CORE属性.
`check_distinct` 方法用来检查在选定的属性集下,是否存在**不可区分集**. 并返回**分类个数**和**数据集约简**

不可区分集:
存在两条或多条记录,它们的(已选择的)条件属性相同,但对应的决策属性不同,则这些记录构成了不可区分集,如:当选择{a,b}作为Reduct时,

U	a	b	E
u_3	1	2	2
u_4	1	2	0
u_5	2	1	2
u_6	2	1	2

U	a	b	E
u_7	2	1	1

$$a_1b_2 \to E_2,\ a_1b_2 \to E_0\ (\{u_3, u_4\}构成不可区分集)$$

$$a_2b_1 \to E_2,\ a_2b_1 \to E_1\ (\{u_5, u_6, u_7\}构成不可区分集)$$

Case 2:

```
# [case2]
dc = DataClass.DataClass([str] * 5)
dc.read(r'..\sample\weather.txt', True)

selected_attr, max_classify_num = attribute_select(dc)
assert selected_attr == {0, 1, 3}  # 选择属性 {Outlook, Temperature, Windy}
```

Weather数据集:

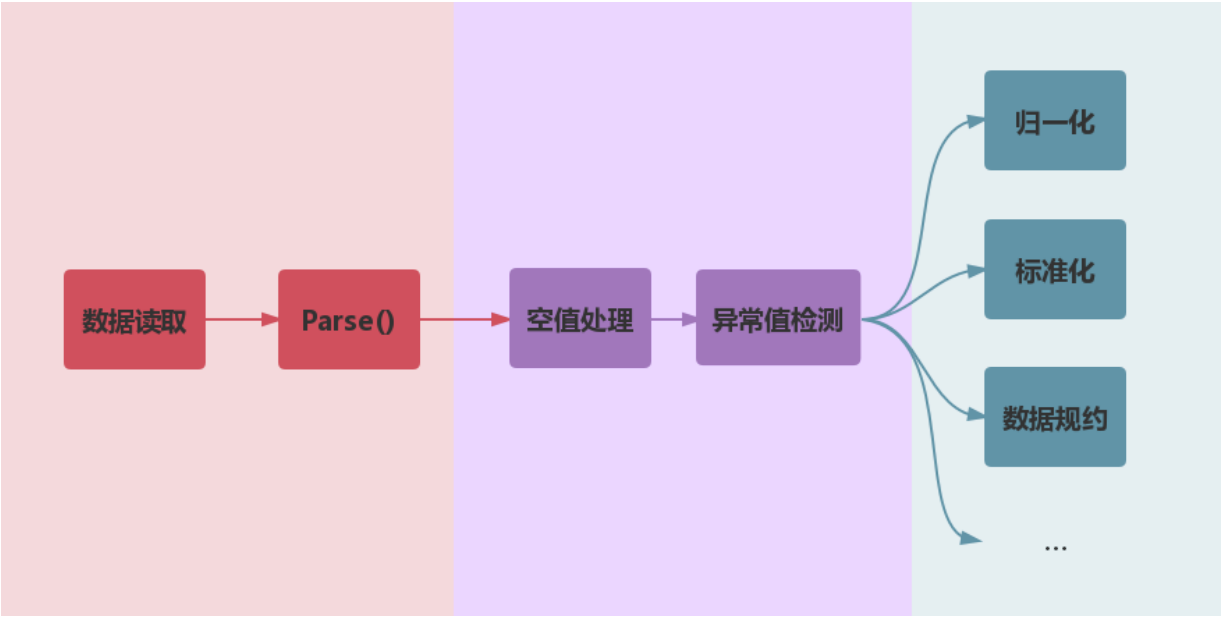
U	Outlook	Temperature	Humidity	Windy	Play
x_1	sunny	hot	high	false	no
x_2	sunny	hot	high	true	no
x_3	overcast	hot	high	false	yes
x_4	rainy	mild	high	false	yes
x_5	rainy	cool	normal	false	no
x_6	overcast	cool	normal	true	yes
x_7	sunny	mild	high	false	no
x_8	sunny	cool	normal	false	yes
x_9	rainy	mild	normal	false	yes
x_10	sunny	mild	normal	true	yes
x_11	overcast	mild	high	true	yes
x_12	overcast	hot	normal	false	yes
x_13	rainy	mild	high	true	no

属性化简后的数据:

U	Outlook	Temperature	Windy	Play
x_1	sunny	hot	false	no
x_2	sunny	hot	true	no
x_3	overcast	hot	false	yes
x_4	rainy	mild	false	yes
x_5	rainy	cool	false	no
x_6	overcast	cool	true	yes
x_7	sunny	mild	false	no
x_8	sunny	cool	false	yes

U	Outlook	Temperature	Windy	Play
x_9	rainy	mild	false	yes
x_10	sunny	mild	true	yes
x_11	overcast	mild	true	yes
x_12	overcast	hot	false	yes
x_13	rainy	mild	true	no

数据处理流程



一、数据表结构DataClass

1.1 属性:

1. 二维的数据表 `data = [[]]`
2. 表头 `head`
3. 每一列的数据类型 `type_list`
4. 归一化时的最大值列表（用于反归一化） `normalize_max`
5. 归一化时的最小值列表（用于反归一化） `normalize_min`
6. 标准化时的均值（用于反标准化） `standard_mean`
7. 标准化时的标准差（用于反标准化） `standard_std`

1.2 方法：

- 数据读取 `read(self, path, has_head, split_tag='\t')`:
  - path:文件路径
  - has_head：是否有表头
  - split_tag：切分字符
- 数据格式转换 `parse(self)`

在调用数据转换方法 `parse(self)` 时，格式错误的数据将被替换为空值.

```
data = DataClass([str] + [float] * 12)  # 数据格式声明
data.read(r".\sample\fz_micro.txt", True)  # 数据读取
data.parse()  # 数据转换
```

数据样例 **.\sample\fz_micro.txt** （部分）

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00		634.38	619.43	733.52	57.33	57.76	15.19	65.14	4026.38	1944.57	401.29	24.81
2017/1/9 19:00	431.47	962.93	570.17	824.27	51.8	52.17	14	67.8	3646.73	1758.57	357.47	22.6
2017/1/9 20:00	423	756.33	556.43	854.57	48.57	48.73	14	68.3	3513	1687.4	339.77	20.7
2017/1/9 21:00	419.93	1008.57	499.47	908.13	46	46.47	13.8	68.33	3345.13	1600.43	326.17	20.87
2017/1/9 22:00		1019.47	476.07	927.67	46.27	46.77	13.03	68.83	3401.73	1633.37	328.7	18.83
2017/1/9 23:00		904.8	475.37	947.03	53.47	53.8	13	68.63	3838.1	1856.47	379.37	22.07
2017/1/10 0:00	412.9	1052.7	467.23	955.4	60.5	60.87	**A5**	68.3	4242.37	2075.77	428.53	25.93
2017/1/10 1:00	412.93	876.2	503.9	930.7	66.8	67.17	13	68.07	4635.37			

二、数据清洗 /DataCleaning

2.1 空值处理 MissingDataHandle.py

2.1.1 空值删除 delete_handle(data_class, handel_index)

1. `data_class` 类型为DataClass的数据
2. `handel_index` 要处理的列的下标

```
import DataClass as dc

data = dc.DataClass([str] + [float] * 12)
data.read(r".\sample\fz_micro.txt", False)
delete_handle(data,[i for i in range(1, 13)])
data.parse()
```

处理后的 .data (空值删除并不会检查数据类型是否合法，如**A5**并不会被删除)

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 19:00	431.47	962.93	570.17	824.27	51.8	52.17	14	67.8	3646.73	1758.57	357.47	22.6
2017/1/9 20:00	423	756.33	556.43	854.57	48.57	48.73	14	68.3	3513	1687.4	339.77	20.7
2017/1/9 21:00	419.93	1008.57	499.47	908.13	46	46.47	13.8	68.33	3345.13	1600.43	326.17	20.87
2017/1/10 0:00	412.9	1052.7	467.23	955.4	60.5	60.87	**A5**	68.3	4242.37	2075.77	428.53	25.93

2.1.2 均数填充 mean_interpolation_handle(data_class, handel_index)

要处理的属性必须是数值的，不是数值元素按空值处理

处理后的 .data

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00	**411.02**	634.38	619.43	733.52	57.33	57.76	15.19	65.14	4026.38	1944.57	401.29	24.81
2017/1/9 19:00	431.47	962.93	570.17	824.27	51.8	52.17	14	67.8	3646.73	1758.57	357.47	22.6

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 20:00	423	756.33	556.43	854.57	48.57	48.73	14	68.3	3513	1687.4	339.77	20.7
2017/1/9 21:00	419.93	1008.57	499.47	908.13	46	46.47	13.8	68.33	3345.13	1600.43	326.17	20.87
2017/1/9 22:00	**411.02**	1019.47	476.07	927.67	46.27	46.77	13.03	68.83	3401.73	1633.37	328.7	18.83
2017/1/9 23:00	**411.02**	904.8	475.37	947.03	53.47	53.8	13	68.63	3838.1	1856.47	379.37	22.07
2017/1/10 0:00	412.9	1052.7	467.23	955.4	60.5	60.87	**13.28**	68.3	4242.37	2075.77	428.53	25.93
2017/1/10 1:00	412.93	876.2	503.9	930.7	66.8	67.17	13	68.07	4635.37	**2279.67**	**469.8**	**28.93**

2.1.3 插值法填充 mid_interpolation_handle(data_class, handel_index)

要处理的属性必须是数值的，不是数值元素按空值处理.

1. 若空值处于首位，则插值取空值的下一个最近的非空的元素.
2. 若空值位于末尾，则插值取空值的上一个最近的非空的元素.
3. 若一个或多个连续的空值位于前后两个非空元素之间，则差值取前后非空元素的等差间值.

处理后的 .data

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00	**431.47**	634.38	619.43	733.52	57.33	57.76	15.19	65.14	4026.38	1944.57	401.29	24.81
2017/1/9 19:00	431.47	962.93	570.17	824.27	51.8	52.17	14	67.8	3646.73	1758.57	357.47	22.6
2017/1/9 20:00	423	756.33	556.43	854.57	48.57	48.73	14	68.3	3513	1687.4	339.77	20.7
2017/1/9 21:00	419.93	1008.57	499.47	908.13	46	46.47	13.8	68.33	3345.13	1600.43	326.17	20.87
2017/1/9 22:00	**417.58**	1019.47	476.07	927.67	46.27	46.77	13.03	68.83	3401.73	1633.37	328.7	18.83
2017/1/9 23:00	**415.24**	904.8	475.37	947.03	53.47	53.8	13	68.63	3838.1	1856.47	379.37	22.07
2017/1/10 0:00	412.9	1052.7	467.23	955.4	60.5	60.87	**13**	68.3	4242.37	2075.77	428.53	25.93
2017/1/10 1:00	412.93	876.2	503.9	930.7	66.8	67.17	13	68.07	4635.37	**2360.77**	**489.71**	**29.58**

2.1.4+ 中数填充 众数填充 固定值填充 等

调用方法与插值法填充类似.

2.2 离异值（异常值）处理 OutlierHandle.py

2.2.1 Z-Score异常值检测 z_score_detection(data_class, handel_index, z_thr=3.0)

1. `data_class` 类型为DataClass的数据.
2. `handel_index` 要处理的列的下标.
3. `z_thr` 识别阈值. 一般取 2.5, 3.0, 3.5
4. `:return` 每一列离异值的下标.

条件：-1. 数据无空值. -2. 数据经过 parse() 方法格式转换.调用方法如下

```
import DataClass as dc
import DataCleaning.MissingDataHandle as mdh
import DataCleaning.OutlierHandle as oh

data = dc.DataClass([str] + [float] * 12)
data.read(r".\sample\fz_micro.txt", False)
data.parse()
mdh.mid_interpolation_handle(data, [i for i in range(1, 13)])  # 插值法填充
oh.outlier_none_handle(data, [i for i in range(1, 13)], "z_score", 3.0)  # 通过`z_score`方法识别异常，并置为空值
data.print()
```

三、数据变换 /DataTransformation

3.1 归一化 NormalizeHandle.py

3.1.1 离差归一化 min_max_normalize(data_class, handel_index)

```
import DataClass as dc
import DataCleaning.MissingDataHandle as mdh
import DataTransformation.NormalizeHandle as nh

data = dc.DataClass([str] + [float] * 12)
data.read(r".\sample\fz_micro.txt", False)
data.parse()
mdh.mid_interpolation_handle(data, [i for i in range(1, 13)])  # 插值法填充
nh.min_max_normalize(data, [i for i in range(1, 13)])  # 离差归一化
data.print()
```

条件：-1. 数据无空值. -2. 数据经过 parse() 方法格式转换处理. 处理后的 .data

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00	0.87	0.27	0.83	0.31	0.48	0.48	0.55	0.52	0.54	0.51	0.45	0.36
2017/1/9 19:00	0.87	0.51	0.76	0.44	0.43	0.43	0.43	0.57	0.49	0.46	0.4	0.33
2017/1/9 20:00	0.71	0.36	0.74	0.48	0.4	0.4	0.43	0.58	0.47	0.44	0.37	0.3
2017/1/9 21:00	0.66	0.55	0.66	0.56	0.38	0.38	0.41	0.58	0.44	0.41	0.36	0.3
2017/1/9 22:00	0.61	0.56	0.63	0.59	0.38	0.38	0.34	0.59	0.45	0.42	0.36	0.27
2017/1/9 23:00	0.57	0.47	0.63	0.61	0.45	0.44	0.33	0.59	0.52	0.48	0.42	0.32
2017/1/10 0:00	0.52	0.58	0.61	0.63	0.51	0.5	0.33	0.58	0.58	0.54	0.48	0.38
2017/1/10 1:00	0.53	0.45	0.67	0.59	0.56	0.56	0.33	0.58	0.63	0.62	0.55	0.44

3.1.2 反离差归一化 min_max_normalize(data_class, handel_index)

运行示例

```
import DataClass as dc
import DataCleaning.MissingDataHandle as mdh
import DataTransformation.NormalizeHandle as nh

data = dc.DataClass([str] + [float] * 12)
data.read(r".\sample\fz_micro.txt", False)
data.parse()
mdh.mid_interpolation_handle(data, [i for i in range(1, 13)])  # 插值法填充
nh.min_max_normalize(data, [i for i in range(1, 13)])  # 离差归一化
nh.anti_min_max_normalize(data, [i for i in range(1, 13)])  # 反离差归一化
data.print()
```

运行结果

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00	431.47	634.38	619.43	733.52	57.33	57.76	15.19	65.14	4026.38	1944.57	401.29	24.81
2017/1/9 19:00	431.47	962.93	570.17	824.27	51.80	52.17	14.00	67.80	3646.73	1758.57	357.47	22.60
2017/1/9 20:00	423.00	756.33	556.43	854.57	48.57	48.73	14.00	68.30	3513.00	1687.40	339.77	20.70
2017/1/9 21:00	419.93	1008.57	499.47	908.13	46.00	46.47	13.80	68.33	3345.13	1600.43	326.17	20.87

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 22:00	417.59	1019.47	476.07	927.67	46.27	46.77	13.03	68.83	3401.73	1633.37	328.70	18.83
2017/1/9 23:00	415.24	904.80	475.37	947.03	53.47	53.80	13.00	68.63	3838.10	1856.47	379.37	22.07
2017/1/10 0:00	412.90	1052.70	467.23	955.40	60.50	60.87	13.00	68.30	4242.37	2075.77	428.53	25.93
2017/1/10 1:00	412.93	876.20	503.90	930.70	66.80	67.17	13.00	68.07	4635.37	2360.77	489.71	29.58

3.2 标准化 StandardizationHandle.py

3.2.1 标准化 standardization(data_class, handel_index)

```
import DataClass as dc
import DataTransformation.StandardizationHandle as sdh

data = dc.DataClass([str] + [float] * 12)
data.read(r"E:\_Python\DataPreprocessing\sample\fz_micro.txt", False)
data.parse()
mdh.mid_interpolation_handle(data, [i for i in range(1, 13)])  # 插值法填充
sdh.standardization(data, [i for i in range(1, 13)])  # 标准化
data.print()
```

条件：-1.数据无空值 -2.数据经过 parse() 方法格式转换.
执行结果

RECEIVETIME	CO	NO2	SO2	O3	PM25	PM10	TEMP	HUM	PM05N	PM1N	PM25N	PM10N
2017/1/9 18:00	1.47	-0.29	0.80	-0.81	0.05	0.04	1.04	-0.44	0.08	0.06	0.02	-0.14
2017/1/9 19:00	1.47	0.89	0.50	-0.21	-0.16	-0.17	0.39	-0.20	-0.16	-0.16	-0.22	-0.30
2017/1/9 20:00	0.86	0.15	0.42	-0.01	-0.29	-0.30	0.39	-0.16	-0.25	-0.25	-0.31	-0.44
2017/1/9 21:00	0.63	1.06	0.07	0.35	-0.39	-0.39	0.28	-0.15	-0.35	-0.35	-0.38	-0.42
2017/1/9 22:00	0.46	1.10	-0.08	0.48	-0.38	-0.38	-0.14	-0.11	-0.32	-0.31	-0.37	-0.57
2017/1/9 23:00	0.29	0.68	-0.08	0.60	-0.10	-0.11	-0.15	-0.13	-0.04	-0.05	-0.10	-0.34
2017/1/10 0:00	0.12	1.22	-0.13	0.66	0.17	0.16	-0.15	-0.16	0.22	0.21	0.16	-0.06
2017/1/10 1:00	0.12	0.58	0.09	0.50	0.41	0.40	-0.15	-0.18	0.46	0.55	0.49	0.21

3.2.2 反标准化 anti_standardization(data_class, handel_index)

运行示例

```
import DataClass as dc
import DataTransformation.StandardizationHandle as sdh

data = dc.DataClass([str] + [float] * 12)
data.read(r"E:\_Python\DataPreprocessing\sample\fz_micro.txt", False)
data.parse()
mdh.mid_interpolation_handle(data, [i for i in range(1, 13)])  # 插值法填充
sdh.standardization(data, [i for i in range(1, 13)])  # 标准化
sdh.anti_standardization(data, [i for i in range(1, 13)])  # 反标准化
data.print()
```

四、数据规约 /DataReduction

4.1 属性选择 RoughSetAttrSelecter.py

4.1.1 基于粗糙集理论的属性选择 attribute_select(data_class)

Workflow:

1. 计算数据的CORE属性集: `core = get_core(dc)` ,
2. 以CORE中的属性作为初始的属性选择,检查在选定的属性集下,是否存在不可区分集(即计算选定属性集在所有决策属性的下近似)
   `check_distinct(dc, core, considered_instence)`

$$Initial\ SelectedAttrs = \{CORE\}$$

$$POS_{\{SelectedAttrs\}}(D) = U_{CX}$$

3. 若存在不可区分集,进行下一步迭代.若不存在不可区分集,则CORE就是Reduct.
4. 从剩下的属性中依次选取一个属性a_i加入:

$$SelectedAttrs = \{CORE\} + \{a_i\}$$

   检查在选定的属性集下,是否存在不可区分集.
5. 若存在一个或多个个数相同的属性集,都不会产生不可区分集,则在这些属性中选取值种类数较少的属性加入并返回SelectedAttrs,并返回SelectedAttrs,程序运行结束.
6. 若对于所有的$\{CORE\} + \{a_i\}$属性集,都会产生不可区分集(没有 Reduct)),则选择区分集个数最少的属性进入下一轮迭代(第4步).

Case 1:
data: (其中a,b,c,d为条件属性, E为决策属性)

U	a	b	c	d	E
u_1	1	0	2	1	1
u_2	1	0	2	0	1
u_3	1	2	0	0	2
u_4	1	2	2	1	0
u_5	2	1	0	0	2
u_5	2	1	1	0	2
u_5	2	1	2	1	1

```
# 方法测试
dc = DataClass.DataClass([str] * 5, data)
core = get_core(dc)
assert core == [1]  # CORE(cd)=1 (the second attr)

considered_instence = np.array([True] * dc.len, np.bool)
is_reduct, classify_num, considered_instence = check_distinct(dc, core, considered_instence)
assert (is_reduct, classify_num, considered_instence) == (False, 1, [False] * 2 + [True] * 5)

# 属性选择
selected_attr, max_classify_num = attribute_select(dc)
assert selected_attr == [1, 3]  # 选择{b,d}作为约简后的属性集
```

方法说明:
其中 `get_core(data)` 方法通过构造 Discernibility Matrix 的方法选出CORE属性.
`check_distinct` 方法用来检查在选定的属性集下,是否存在**不可区分集**. 并返回**分类个数**和**数据集约简**

不可区分集:
存在两条或多条记录,它们的(已选择的)条件属性相同,但对应的决策属性不同,则这些记录构成了不可区分集,如:当选择{a,b}作为Reduct时,

U	a	b	E
u_3	1	2	2

U	a	b	E
u_4	1	2	0
u_5	2	1	2
u_6	2	1	2
u_7	2	1	1

$$a_1b_2 \to E_2,\ a_1b_2 \to E_0\ (\{u_3, u_4\}构成不可区分集)$$

$$a_2b_1 \to E_2,\ a_2b_1 \to E_1\ (\{u_5, u_6, u_7\}构成不可区分集)$$

Case 2:

```
# [case2]
dc = DataClass.DataClass([str] * 5)
dc.read(r'..\sample\weather.txt', True)

selected_attr, max_classify_num = attribute_select(dc)
assert selected_attr == {0, 1, 3}  # 选择属性 {Outlook, Temperature, Windy}
```

Weather数据集:

U	Outlook	Temperature	Humidity	Windy	Play
x_1	sunny	hot	high	false	no
x_2	sunny	hot	high	true	no
x_3	overcast	hot	high	false	yes
x_4	rainy	mild	high	false	yes
x_5	rainy	cool	normal	false	no
x_6	overcast	cool	normal	true	yes
x_7	sunny	mild	high	false	no
x_8	sunny	cool	normal	false	yes
x_9	rainy	mild	normal	false	yes
x_10	sunny	mild	normal	true	yes
x_11	overcast	mild	high	true	yes
x_12	overcast	hot	normal	false	yes
x_13	rainy	mild	high	true	no

属性化简后的数据:

U	Outlook	Temperature	Windy	Play
x_1	sunny	hot	false	no
x_2	sunny	hot	true	no
x_3	overcast	hot	false	yes
x_4	rainy	mild	false	yes
x_5	rainy	cool	false	no

U	Outlook	Temperature	Windy	Play
x_6	overcast	cool	true	yes
x_7	sunny	mild	false	no
x_8	sunny	cool	false	yes
x_9	rainy	mild	false	yes
x_10	sunny	mild	true	yes
x_11	overcast	mild	true	yes
x_12	overcast	hot	false	yes
x_13	rainy	mild	true	no