

程序设计实践大作业个人报告

时序序列处理算子开发

刘喆琪 探微书院 2020013163

一、函数实现

实现的部分是 ACF, Distinct, Histogram, Segment, Skew, Spline, Spread, Stddev, Zscore 算子及其测试。

使用

1. ACF

根据公式 $R_{x,x}(n) = \sum_{m=-\infty}^{\infty} x(m)x(m+n)$ 计算得到 ACF 的前 n 个数，写入一个列表，然后拼接出所求的 ACF 序列。示意图如下：



示意图，两数列一样长就是自相关

计算核心：

```
def run_acf(x):
    len_x = len(x)
    q = ([0] * len_x)
    p = []
    for i in range(len_x):
        q = (x[i:len_x + 1])
        if len(q) < len_x:
            q += [0] * (len_x - len(q))
        p.append(sum(a * b for a, b in zip(q, x)))
```

```

for i in range(1, len_x):
    p.insert(i-1, p[-i])
return list(p)

```

利用 dropna 除去空值, 利用 pandas 中的 to_datetime 函数生成从时间戳为 0 开始的时间序列, 作为输出的时间序列。

使用: 输出单个序列, 类型为 DOUBLE。序列中共包含 $2N-1$ 个数据点

2. Distinct

利用 set()函数, 将输入序列转化为集合, 直接消除重复项和空值, 保留 NaN, 时间序列依旧使用 pandas 中的 to_datetime 生成

使用: 输出单个序列, 类型为 DOUBLE。

3. Histogram

min, max, count 参数放置在“ parameters” 中, 使用 get 函数获取, 没有输入参数则设为默认值。若不输入 min, min 取序列最大值的相反数; 若不输入 max, max 取序列最大值; 若不输入 count, count=1。

设置一个长度为 count 全 0 列表 bucket, 的根据文档的要求, 判断输入值的分桶, bucket 对应处+1。时间序列依旧使用 pandas 中的 to_datetime 生成。

计算核心:

```

for j in range(len(output_data[column])):
    if output_data[column][j] < min_num:
        bucket[0] += 1
    elif output_data[column][j] >= max_num:
        bucket[-1] += 1
    else:
        for i in range(1, count+1):
            if (output_data[column][j] >=
min_num+(i-1)*(max_num-min_num)/count
                and output_data[column][j] <
min_num+i*(max_num-min_num)/count):
                bucket[i-1] += 1

```

4. Segment

'output' , 'error' 参数放置在" parameters" 中, 使用 get 函数获取, 没有输入参数则设为默认值, output 默认为 'first' , ' error' 默认为 0.1。

如果输入序列过长 (>10000) 则进行等距采样。先判断输入是否是等差数列。如果输入数列是等差数列则直接按要求输出。如果不是则利用 Bottom_Up 函数, 先将序列分成 $n/2$ 段, 并利用自定义的 calculate_error 函数计算两两合并的带来的 cost, 并将 cost 最小的合并, 更新分段序列, 重复合并过程直到合并的 cost 都大于 error, 返回分段。

时间序列依旧使用 pandas 中的 to_datetime 生成。

5. Skew

利用偏度的公式计算。首先利用平均值、标准差的公式获得标准差, 平均值, 带入偏度公式计算即可。时间序列使用'1970-01-01 08:00:00.000'。

6. Spline

数学:

(1)构建对角矩阵和向量 (compute_changes, create_tridiagonalmatrix, create_target 函数)

首先给定一个等长 n (大于 2) 的 X 数组和 Y 数组, 要构建一个三对角矩阵 (例如, $[[b_0 \ 0 \ 0], [a_0 \ b_1 \ c_1 \ 0], [0 \ a_1 \ b_2 \ c_2], [0 \ 0 \ a_2 \ b_3]]$), 然后将求解该矩阵以产生三次样条的系数。样条曲线的目标是它经过每个点 (x, y) , 并且一阶和二阶导数也连续。将 H 定义为 X 中差异的 $n-1$ 长度数组: $h_i = x_{i+1} - x_i, i = 0 \dots n-2$ 。将 B 定义为包含对角线元素的长度为 n 数组, A 为 B 上方对角线长度为 $n-1$ 数组, C 为其下方对角线长度为 $n-1$ 数组。其中, $a_i = \frac{h_i}{h_i + h_{i+1}}, i = 0 \dots n-3, b_i = 2, i = 0 \dots n-1, c_i = \frac{h_i}{h_i + h_{i-1}}, i = 1 \dots n-2$ 。

使用两端二阶导数等于 0 的自然边界条件, 使 c_0 和 a_{n-2} 的值都等于 0。

计算等式的右边是一个长度为 n 的数组 D , 其中 $c_i = 6 \frac{\frac{y_{i+1}-y_i}{h_{i+1}} - \frac{y_i-y_{i-1}}{h_i}}{h_i + h_{i+1}}, i = 2 \dots n-1, d_1 = d_n = 0$

(2)解方程 (solve_tridiagonalsystem)

首先导出长度为 n 的向量 C' 和 D' , 然后是 X

$$c'_0 = \frac{c_0}{b_0}$$

$$c'_i = \frac{c_i}{b_i - c'_{i-1}a_{i-1}} \text{ for } i = 1..n-2$$

$$c'_{n-1} = 0$$

$$d'_0 = \frac{d_0}{b_0}$$

$$d'_i = \frac{d_i - d'_{i-1}a_{i-1}}{b_i - c'_{i-1}a_{i-1}} \text{ for } i = 1..n-1$$

$$x_{n-1} = d'_{n-1}$$

$$x_i = d'_i - c'_i x_{i+1} \text{ for } i = n-2..0$$

(3)计算系数 (compute_spline)

最后一步是将上述参数转换为一组三次曲线。定义 $z = \frac{x-x_j}{h_j}$, 得到 $x_j < x < x_{j+1}$ 之间样条函数为

$$S(x) = \frac{(M_{j+1} - M_j)h_j^2}{6}z^3 + \frac{M_j h_j^2}{2}z^2 + (y_{j+1} - y_j - \frac{(M_{j+1} + 2M_j)h_j^2}{6})z + y_j$$

利用 bisect 库快速找到某个 x 对应的 j。时间序列依旧使用 pandas 中的 to_datetime 生成, 参数'points'放置于'parameter'中, 利用 get 函数获取, 没有默认值。

7. Spread

利用 python 自带的 min, max 函数获取最大最小值, 相减就得到极差。时间序列使用'1970-01-01 08:00:00.000'。

8. Stddev

利用标准差的公式直接计算标准差。时间序列使用'1970-01-01 08:00:00.000'。

9. Zscore

参数'compute' 'avg' 'std'放置于'parameter'中, 利用 get 获取。

如果'compute'=='batch', 利用公式计算出标准差, 均值, 无视是否输入 avg, std。如果'compute'=='stream'则获取输入的 avg, std, 最后利用 $z = \frac{x-\mu}{\sigma}$

得到输出值。时间序列依旧使用 pandas 中的 to_datetime 生成。

二、测试

每个算子都有其对应的测试文档，如 Acf.py 的测试文件是 AcfUT.py。测试

文件的结构：

setUp	输入 input_paths, input_type, input_location, output_type, output_paths, 读入数据, 设置算子
test_1	测试一种情况（正常数据）
test_2	测试另一种情况（大量数据）

样例：

```
class DistinctUT(unittest.TestCase):

    def setUp(self):
        input_paths = ["../data/root_test_d2"]
        input_types = ["csv"]
        input_location = ["local_fs"]
        output_paths = ["root_test_d1_out.csv"]
        output_types = ["csv"]
        self.orig_dataset = FlokAlgorithmLocal().read(input_paths,
input_types, input_location, output_paths, output_types)
        self.algorithm = Distinct()

    def test_distinct_1(self):
        self.timeseries = {"timeseries": "Time,s2"}
        self.serieslength = 5
        self.params = {}

    def test_distinct_2(self):
        self.timeseries = {"timeseries": "Time,s2"}
        self.serieslength = 2500
        self.params = {}

    def tearDown(self):
        dataset = FlokDataFrame()
        dataset.addDF(SelectTimeseries().run(self.orig_dataset,
self.timeseries).get(0).iloc[:self.serieslength])
```

```
result = self.algorithm.run(dataset, self.params)
print(result.get(0))
```

测试结果：

数据量	ACF	Distinct	Histogram	Segment	Skew	Spread	Stddev	Zscore
10000	12.9	0.06	0.635	1.467	0.078	0.061	0.07	0.131
5000	4.19	0.06	0.034	0.627	0.080	0.066	0.073	0.104
2500	1.72	0.06	0.02	0.317	0.075	0.0715	0.074	0.095

注：对 spline 的核心算子进行单独处理（由于 spline 要求横坐标递增，直接复制增长

数据点会导致时间不可用）（和上面的数据无可比性）得到：

点数	时间
10000	0.0043s
100000	0.332s

三、集合测试

将所有测试文件导入 UTcollection.py 中进行集中测试。测试得到在所有算

子处理数据量均为 2500 时耗时 19s，数据量均为 10000 时耗时 33s。

```
11 1970-01-01 08:00:00.012 -2.0
12 1970-01-01 08:00:00.013 -2.0
13 1970-01-01 08:00:00.014 0.0
14 1970-01-01 08:00:00.015 -1.0
15 1970-01-01 08:00:00.016 -1.0
16 1970-01-01 08:00:00.017 9.0
17 1970-01-01 08:00:00.018 1.0
18 1970-01-01 08:00:00.019 -3.0
19 1970-01-01 08:00:00.020 -1.0
.
-----
Ran 18 tests in 19.036s

OK
```

图 1，数据量为 2500 时的耗时

11	1970-01-01 08:00:00.012	-2.0
12	1970-01-01 08:00:00.013	-2.0
13	1970-01-01 08:00:00.014	0.0
14	1970-01-01 08:00:00.015	-1.0
15	1970-01-01 08:00:00.016	-1.0
16	1970-01-01 08:00:00.017	9.0
17	1970-01-01 08:00:00.018	1.0
18	1970-01-01 08:00:00.019	-3.0
19	1970-01-01 08:00:00.020	-1.0
.		

Ran 18 tests in 32.751s

OK

图 2, 数据量为 10000 时的耗时