一. 文献信息

作者：Xuan Zhang, Longxiang Xiong, Ningyuan Sun, Mingxia Wang, Hao Tang, Yanxing Zhao（北京交通大学智能网络实验室）

论文题目：ACCURATE INFERENCE OF UNSEEN COMBINATIONS OF MULTIPLE ROOTCAUSES WITH CLASSIFIER ENSEMBLE

发表途径：ICASSP-SPGC-2022 AIOps Challenge in Communication Networks 获奖论文

发表时间：2022 年

项目代码：https://github.com/zxuan000/SPGC_aiops_bjtu

二. 问题意义

研究问题：利用 AI 定位网络故障的具体原因。

研究背景：

1. 定位网络故障的重要性

在无线网络的运行和管理中，难免会出现一些网络故障。为了能够保证网络尽快正常运行，需要及时地、准确地对网路故障进行定位。

2. 如何进行网络故障定位

网络故障定位的关键是根据结果信息，去鉴别根源。它往往依赖着网络参量之间的依赖和逻辑关系。比如我们监测到某个网络参量有异常变化，那就需要去分析，哪个根因或者哪几个根因会对这个参量有着决定性作用，从而进行网络故障定位。

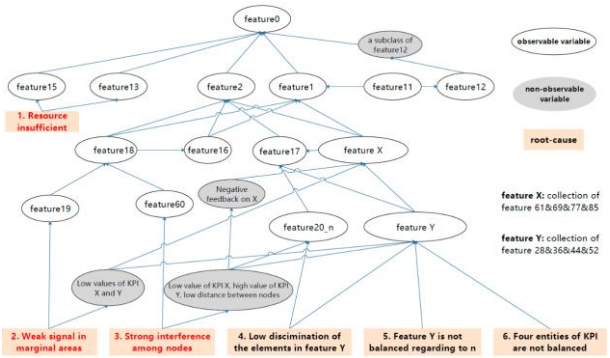
3. 现有定位方法以及问题

现有定位方法往往是根据工程师的经验进行主观判断，这会带来一些问题。第一，工程师在判断中由于主观影响，有可能忽略一些隐藏的依赖关系，从而导致错过了一些重要的根因。第二， 我个人觉得这个对工程师的要求很高，具备这种素质的人才可能也不多，耗费人力和时间。因此，现在尝试转向 AI，寻求在这个问题上的突破。

4. ICASSP 2022 大赛 题目重述

基于以上背景，大赛题目要求参赛选手能够根据赛事提供的一些数据和信息支持，能够准确定位网络故障。

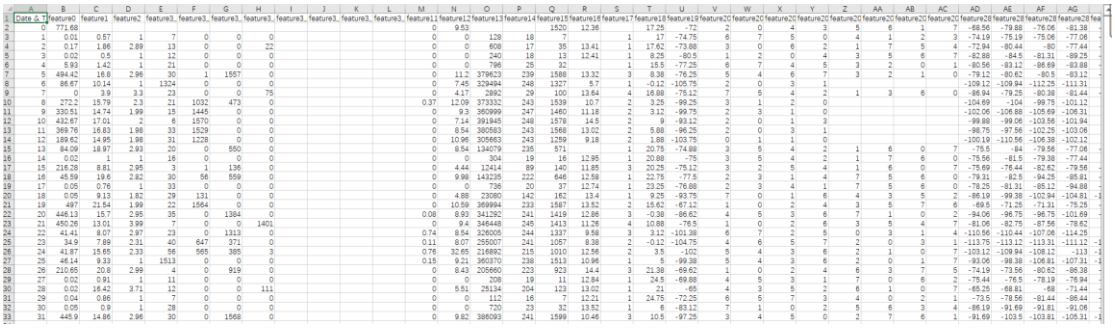
大赛提供了以下网络参量之间的依赖关系图：



其中，feature0 的变化是我们最终监测到的异常结果，而需要根据这个网络依赖关系，找到最终的网络故障原因。图中列出了 1-6 的故障原因，不过由于赛事提供数据集中的标签，只有前三个，因此在后续的训练和测试中，也是只会出现前三个归因。

值得注意的是，故障归因很有可能不是单一个原因引起的。比如提供的训练集中，很多标签都是原因 2 和 3 共同造成的网络故障，这是后续分析中的难题。

赛事提供的训练集和测试集数据的形式都是时间切片形式。我选取了测试集中的一个样本做具体说明，样本如下所示。



Date & T	feature0	feature1	feature2	feature3_	feature3_	feature3_	feature3_	feature3_	feature3_	feature3_	feature3_	feature11	feature12	feature13	feature14	feature15	feature16	feature17	feature18	feature19	feature20	feature20	feature20	feature20	feature20	feature20	feature28	feature28	feature28	feature28	fea
2		771.68										0	9.53			1520	12.36		17.25	-72	2	0	4	3	5	6	1	7	-68.56	-79.88	-76.06	-81.38	-
3	1	0.01	0.57	1	7	0	0	0				0	0	128	18	7		1	17	-74.75	6	7	5	0	4	1	2	3	-74.19	-75.19	-75.06	-77.06	-
4	2	0.17	1.86	2.89	13	0	0	22				0	0	608	17	35	13.41	1	17.62	-73.88	3	0	6	2	1	7	5	4	-72.94	-80.44	-80	-77.44	-
5	3	0.02	0.5	1	12	0	0	0				0	0	240	18	13	12.41	1	8.25	-80.5	1	2	0	4	3	5	6	7	-82.88	-84.5	-81.31	-89.25	-
6	4	5.93	1.42	1	21	0	0	0				0	0	796	25	32		1	15.5	-77.25	6	7	4	5	3	2	0	1	-80.56	-83.12	-86.69	-83.88	-
7	5	494.42	16.8	2.96	30	1	1557	0				0	11.2	379623	239	1588	13.32	3	8.38	-76.25	5	4	6	7	3	2	1	0	-79.12	-80.62	-80.5	-83.12	-
8	6	86.67	10.14	1	1324	0	0	0				0	7.45	329494	248	1327	5.7	1	-0.12	-105.75	2	0	3	1					-109.12	-109.94	-112.25	-111.31	-
9	7	0	3.9	3.3	23	0	0	75				0	4.17	2892	29	100	13.64	4	16.88	-75.12	7	5	4	2	1	3	6	0	-86.94	-79.25	-80.38	-81.44	-
10	8	272.2	15.79	2.3	21	1032	473	0				0.37	12.09	373332	243	1539	10.7	2	3.25	-99.25	3	1	2	0					-104.69	-104	-99.75	-101.12	
11	9	330.51	14.74	1.99	15	1445	0	0				0	9.3	360999	247	1480	11.18	2	3.12	-99.75	2	3	1	0					-102.06	-106.88	-105.69	-106.31	
12	10	432.67	17.01	2	6	1570	0	0				0	7.14	391945	248	1578	14.5	2	9	-93.12	2	0	1	3					-99.88	-99.06	-103.56	-101.94	
13	11	369.76	16.83	1.98	33	1529	0	0				0	8.54	380583	243	1568	13.02	2	5.88	-96.25	2	0	3	1					-98.75	-97.56	-102.25	-103.06	
14	12	189.62	14.95	1.98	31	1228	0	0				0	10.96	305663	243	1259	9.18	2	1.88	-103.75	0	1	1	0					-100.19	-110.56	-106.38	-102.12	
15	13	84.09	18.97	2.93	20	0	550	0				0	8.54	134079	235	571		1	20.75	-74.88	3	5	4	2	1	6	0	7	-75.5	-84	-79.56	-77.06	
16	14	0.02	1	1	16	0	0	0				0	0	304	19	16	12.95	1	20.88	-75	3	5	4	2	1	7	6	0	-75.56	-81.5	-79.38	-77.44	
17	15	216.28	8.81	2.95	3	1	136	0				0	4.44	12414	89	140	11.85	3	20.25	-75.12	3	2	5	4	1	6	0	7	-75.69	-76.44	-82.62	-79.56	
18	16	45.59	19.6	2.82	30	56	559	0				0	9.98	143235	222	646	12.58	1	22.75	-77.5	2	3	1	4	7	5	6	0	-79.31	-82.5	-94.25	-85.81	
19	17	0.05	0.76	1	33	0	0	0				0	0	736	20	37	12.74	1	23.25	-76.88	2	3	4	1	7	5	6	0	-78.25	-81.31	-85.12	-94.88	
20	18	0.05	9.13	1.82	29	131	0	0				0	4.88	23080	142	162	13.4	1	9.25	-93.75	7	0	1	6	4	3	5	2	-86.19	-99.38	-102.94	-104.81	-1
21	19	497	21.54	1.99	22	1564	0	0				0	10.59	369994	233	1587	13.52	2	15.62	-67.12	1	0	2	4	3	5	7	6	-69.5	-71.25	-71.31	-75.25	
22	20	446.13	15.7	2.95	35	0	1384	0				0.08	8.93	341292	241	1419	12.86	3	-0.38	-86.62	4	5	3	6	7	1	0	2	-94.06	-96.75	-96.75	-101.69	
23	21	450.26	13.01	3.99	7	0	0	1401				0	9.4	346448	245	1413	11.26	4	10.88	-76.5	1	0	2	6	3	5	4	7	-81.06	-82.75	-87.56	-78.62	
24	22	41.41	8.07	2.97	23	0	1313	0				0.74	8.54	326005	244	1337	9.58	3	3.12	-101.38	6	7	2	5	0	3	1	4	-110.56	-110.44	-107.06	-114.25	
25	23	34.9	7.89	2.31	40	647	371	0				0.11	8.07	255007	241	1057	8.38	2	-0.12	-104.75	4	6	5	7	2	0	3	1	-113.75	-113.12	-113.31	-111.12	-1
26	24	41.87	15.65	2.33	56	565	385	3				0.76	32.65	216892	215	1010	12.56	2	3.5	-102	5	4	3	6	2	1	0	7	-103.12	-109.94	-108.12	-113	-1
27	25	46.14	9.33	1	1513	0	0	0				0.15	9.21	360370	238	1513	10.96	1	5	-99.38	5	4	3	6	2	0	1	7	-93.06	-98.38	-106.81	-107.31	-1
28	26	210.65	20.8	2.99	4	0	919	0				0	8.43	205660	223	923	14.4	3	21.38	-69.62	1	0	2	4	6	3	7	5	-74.19	-73.56	-80.62	-86.38	
29	27	0.02	0.91	1	11	0	0	0				0	0	208	19	11	12.84	1	24.5	-69.88	4	5	3	1	7	0	6	2	-75.44	-76.5	-78.19	-76.94	
30	28	0.02	16.42	3.71	12	0	0	111				0	5.51	25134	204	123	13.02	1	21	-65	4	3	5	2	6	1	0	7	-65.25	-68.81	-68	-71.44	
31	29	0.04	0.86	1	7	0	0	0				0	0	112	16	7	12.21	1	24.75	-72.25	6	5	7	3	4	0	2	1	-73.5	-78.56	-81.44	-86.44	
32	30	0.05	0.9	1	28	0	0	0				0	0	720	23	32	13.52	1	6	-83.12	7	1	0	2	5	6	3	4	-86.19	-91.69	-91.81	-91.06	
33	31	445.9	14.86	2.96	30	0	1568	0				0	9.82	386093	241	1599	10.46	3	10.5	-97.25	3	4	5	0	2	7	6	1	-91.69	-103.5	-103.81	-105.31	-1

我们发现样本的第一列数据就是时间线，每一个样本都会提供多个时间切片，即在不同时间上各个特征的具体值（当然有些特征会丢失），我们需要根据这些时间切片对我们的网络模型做训练。

最后说明一下比赛的评分机制。由于一共有六个归因，最后每个测试样本的预测结果应该是一个 1*6 的行向量，对应位置为 1 则代表存在这个归因。在每个样本中，如果测试结果有此归因，而预测结果也有此归因，则计 1 分；而如果测试结果有此归因，预测结果无此归因，则需要扣 1 分。样本i的最终得分s_i还需要用上述方法计算出分数后，除以真实的归因数量。而最后整体的得分S则需要算所有N_{te}个样本的平均得分，公式如下：

$$S = \frac{\sum_{i=1}^{N_{te}} s_i}{N_{te}}$$

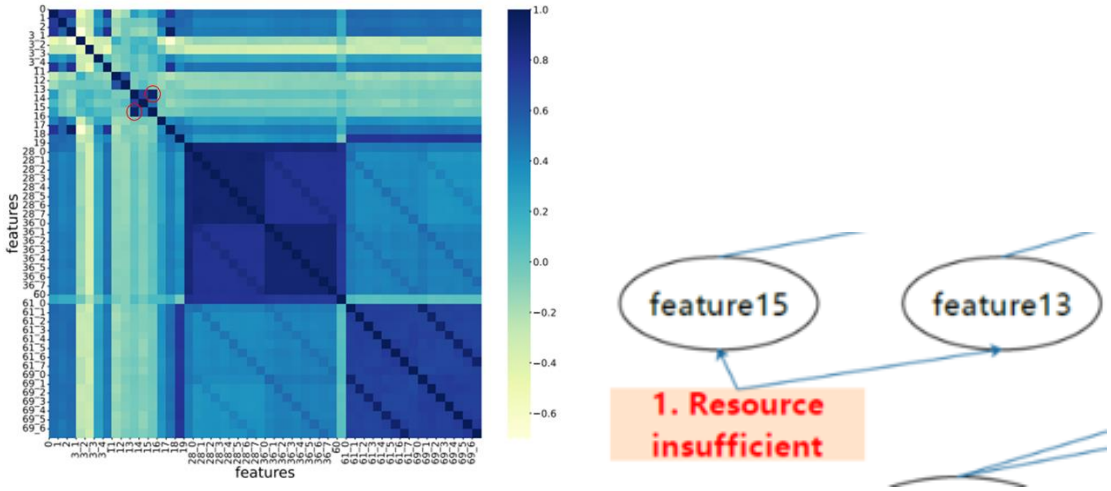
三. 思路方法

思路分析：

论文通过对大赛数据的前瞻分析，得到了一些后续工作方向上的重点。

1. 数据相关性分析

作者利用所给训练数据，作出了各个特征之间的相关性热力图如下右图所示。（我觉得这是一步很棒、很直观的工作，去进一步理清楚题中所给数据以及所给关系图的含义。刚拿到题目我对依赖关系确实也不太明白，但是这个图一出来就感觉对题目一下就懂了）

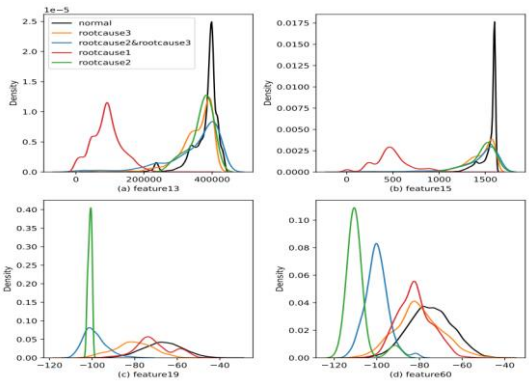


其中，颜色越深的地方表示特征间的相关性越强。比如我们发现，对角线上的相关性都是 1，因为每个特征和自己计算相关性那就是 1。再比如，特征 13 和特征 15 的关联性就非常强，我在图中用红色圈标出，相关性几乎也达到了 1。这一特点和题目中所给的网络关系图一致，如右上图所示，特征 13 和特征 15 同时由归因 1 给决定，**因此我们后续的工作中在分析归因 1 时，对特征 13 和 15 就要格外地关注。**

2. 特征值的分布范围

由于所给特征值较多，个人觉得在最后的网络训练中可能会导致过拟合，且加大计算量，因此在对特定的归因中能够找到一些关键的影响特征，而去掉一些无关紧要的特征，会大大减少后续的网络处理量。

基于此，作者利用训练集的数据，对不同归因标签下的数据进行密度统计，其中特征 13，15，19，60 的密度特性如下所示：



那 feature13 和 15 图做举例，我们发现和其他几条曲线相比，在归因 1 的影响下，归因 1 曲线的分布明显不同于其他几条曲线，说明在归因 1 的影响下，feature13 和 15 的值明显偏小！而其他几个归因下，feature13 和 15 的分布情况基本相同。这样就非常充分的说明了，feature13 和 15 对于我们去判断归因 1 非常的重要，而 feature13 和 15 在判断归因 2，3 的时候，则用处不会很大（甚至可以丢弃）。

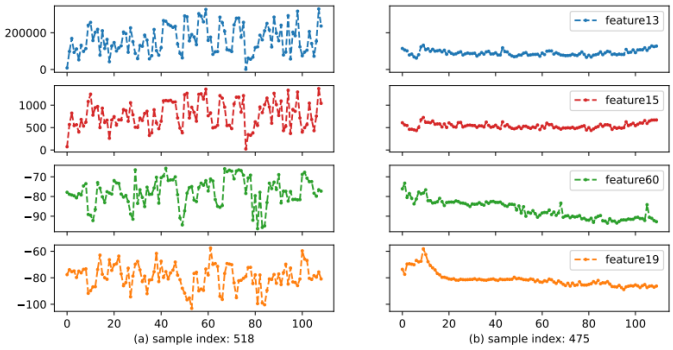
个人感觉这个数据分析非常的精彩，一下子给后续的工作有了方向，本来一堆数据也不

太确定用哪个，怎么用，现在通过这种分析手段，可以把重点关注到一些关键的特征上了。

3. 时间切片分析

我们在特征分布分析中，可以观察下 feature60 的分布，我们发现在归因 2，归因 3，以及归因 2 和 3 的曲线分布都不太一样，所以 feature60 到底是哪个归因的关键因素？这里非常重要的事情就是，我们需要确定某个样本它的归因来源是只有一个还是有两个或者多个。

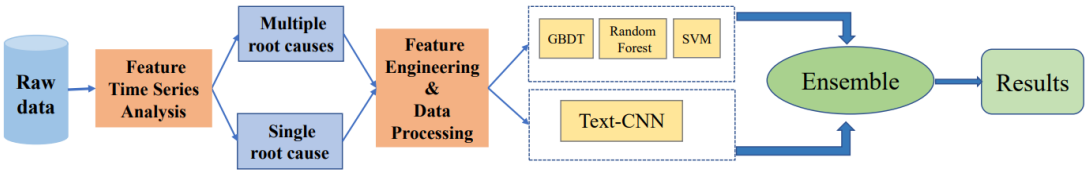
之前分析了特征值在所有样本维度上的分布情况，接下来作者通过分析特征值在单个样本的时间维度上的分布情况，来观察归因的个数对结果的影响，作者选取了测试样本 518 和 475 号的情况，结果如下图所示：



其中 518 号样本的特征值浮动非常大，作者认为这种浮动的原因可能是由于多个归因造成的；而相比而言，475 号样本的特征值浮动则很小。我个人的理解就是，对于多个归因的情况下，对结果的影响会有更多的不确定性，而如果只有一个归因，那对结果造成的影响规律更有可能被人为所学习到。因此，这给我们后续的工作提供了一种思路，如果要判断归因是否为单个，可以通过观察时间切片上的特征值浮动情况进行判断。

具体方法：

整体方法的结构框架如下图所示，包括前期对数据特征的融合处理，以及后期主要的网络架构，下面我将分部分讲解我的理解：



1. 归因个数分类：对单个归因和多个归因进行分类判别，以决定后续的处理方法。

归因个数的分类方法就是根据我们在对时间切片分析时的思路所形成的。具体方法可以分为以下几个步骤：

①先根据题目中所给的关系依赖图，选择与归因关系比较大的特征值。比如我们这里选择归因 1，那我们可以选择 feature13 和 15 进行判别。

②对选定的特征值分别计算方差（或者标准偏差），然后设定一个阈值，如果这个方差大于阈值，那则可以判定此处是多个归因。否则是单个对因。

相关代码如下：

```
import os
import pandas as pd
import numpy as np
feature13_std = []
feature15_std = []
files = os.listdir(Folder_Path1)
files.sort(key=lambda x:int(x[:-4]))
for filename in files:
    df = pd.read_csv(Folder_Path1+'/'+filename,index_col = 0)
    feature13_std.append(df['feature13'].std())
    feature15_std.append(df['feature15'].std())
feature13_std = np.array(feature13_std)
feature15_std = np.array(feature15_std)
feature13_std[np.isnan(feature13_std)] = 0
feature15_std[np.isnan(feature15_std)] = 0
feature13_std = feature13_std/np.max(feature13_std)
feature15_std = feature15_std/np.max(feature15_std)
feature_fil = feature13_std+feature15_std
```

```
for val in np.where(np.array(feature_fil)>0.6)[0]:
    if pre_result.loc[val,'score']>0.2:
        add_1.append(val)
```

上面的图中，通过.std 函数计算 feature13 和 feature15 的标准偏差，然后进行一个归一化，即除以所有文件中标准偏差的最大值，然后使用 feature_fil 去存放 feature13 和 15 的标准偏差之和。

紧接着，我们在下面的硬判决中，我们把这一阈值设成了 0.6，如果大于这一阈值，就认为波动较大，看作多个归因。

做了分类之后，作者对于不同的分类结果做了不同的处理方式。如下图的伪代码所示：

Algorithm 1 Rroot cause location for time slice file

Input: Time slice file *data*
Output: Root cause of *data*

```
Multi_RC = simultaneous_multiple_root_causes(data)
if Multi_RC then
    for each row in data do
        feature_engineering(row)
        RC_classification()
    end for
    ensemble_classification_result()
else
    feature_engineering(data)
    RC_classification()
end if
```

当我们判决为多个归因时，由于这个时候，特征值随着时间变化的浮动很大，因此在后续的处理中，我们对每个时间点单独进行预测，即数据的每一行进行单独预测，然后把每个样本每个时间点的结果进行合并得到最终结果。而当判决为单个归因时，由于特征值的浮动不是很大，因此就可以直接对一个样本的所有数据做处理，而不用分时间处理。

2. 特征数据的提取和预处理：赛事提供了一些特征值的数据，但是有些特征值还包含着数值无法体现的含义，需要进一步挖掘。此外，为了后续训练的方便，作者还对数据集做了一定额外的处理，将在这部分做讲解。

1）空间信息的提取

赛事提供的数据中出现了一些特征，会有不同的后缀，比如 feature20_n，n 可以取 0-7 八个不同的值。数据对 feature20_n 的解释如下：

15	feature20_n, n=0,1,...,7	ID of 8 receiving directions; Direction is represented by n	Continuous non-negative integer 0-31，arranged as 4*8 matrix: 24,25,26,27,28,29,30,31 16,17,18,19,20,21,22,23 8,9,10,11,12,13,14,15 0,1,2,3,4,5,6,7

其实说白了，feature20_n 就是一个空间位置的特征信息值，n 取 0-7 分别代表了 8 个信息监测中用到的 8 个接受方向，那这 8 个方向的取值是什么样子的呢？方向的取值可以是 0-31 之间的 32 个整数，这 32 个整数分别代表某个位置，就是一个 4*8 的矩阵来代表位置，如下图所示：

24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

由于 0-31 的数值，从数学角度上来看，其实一点空间意义都没有。但是它的实际物理意义在这里是表示空间位置的信息，因此作者在原有特征的基础上，新增添了两个特征 feature_edge 和 feature_distance，来表示这一位置信息，具体方法如下：

Feature_edge:

这个特征用来表示接收方向中有多少是处于边缘位置的。因为边缘位置的信号接受和中心位置的信号接受到的值可能会有所不同。作者定义 4*8 矩阵的最左边两列和最右边两列为边缘位置，如上图中的深蓝色区域，feature_edge 的具体值等于这 8 个接受方向中，处于边缘位置的方向个数。

具体实现代码如下：

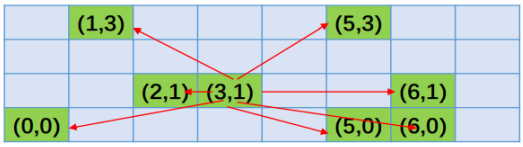
```
def edge_count(test):
    edge_list = [0,1,6,7,8,9,14,15,16,17,22,23,24,25,30,31]
    feature_edge = []
    for i in range(len(test)):
        current_direction = [m for m in test.iloc[i].tolist() if m in edge_list]
        feature_edge.append(len(current_direction))
    test['feature_edge'] = feature_edge
    return test.feature_edge
```

使用 edge_list 存放边缘的 16 个位置，然后通过 for 循环来查找 feature20_0-7 的八个值中，有几个是在边缘位置的，最后把 feature_edge 这个特征值也放进每个样本的特征中去，形成一类新的特征。

Feature_distance:

除了八个方向的边缘位置特征之外，作者还考虑了各个位置之间的欧氏距离。这也是非常重要的，因为如果几个接收方向之间都靠在一起，那得到的结果肯定会相对接近，而如果几个接受方向分布位置差异比较大，那可以更全面的反映网络的情况。作者就利用 4*8 的矩阵去计算他们之间的欧式距离。

24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7



上图中的绿色位置就是 8 个接收方向的位置，把他们转换为右图对应的坐标之后计算相互之间的欧氏距离。Feature_distance 的值，等于两两之间的欧氏距离之和右上图是计算位置 10 到其他 7 个位置的欧氏距离的示意图。

具体实现代码如下：

```
def distance_count(test):
    feature_distance = []
    for i in range(len(test)):
        coordinate = []
        for m in (test.iloc[i].tolist()):
            coordinate.append((m-8*(m//8), m//8))
        distance = 0
        for a in coordinate:
            for b in coordinate:
                distance = distance + ((a[0]-b[0])**2 + (a[1]-b[1])**2)**0.5
        feature_distance.append(distance)
    test['feature_distance'] = feature_distance
    return test.feature_distance
```

第一步，先要将位置的特征值转为 xy 坐标。对于 y 坐标，由于矩阵为 4*8 的，所以直接用特征值 m 去除以 8 取整即可。比如 0-7 的话除以 8 取整后就是在第 0 行，8-15 则在第 1 行，以此类推。而 x 坐标，就用特征值 m 减去 8 倍的所在行数，多出来的值就说明了 x 坐标，也就是在第几列。比如我们拿位置 11 来进行具体计算：

$$y = \left[\frac{11}{8}\right] = 1, x = 11 - 8y = 3$$

$$\therefore (x, y) = (3,1)$$

第二步，我们看到代码中用了一个双循环，来遍历两两之间的欧氏距离计算，计算公式为：$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$，然后对他们进行求和。

最后将 feature_distance 这一特征作为数据的新一类特征放入每个数据样本中。

2）时间信息的提取

由于每个样本的时间序列长短不一样，因此作者赋予了一个时间序列长度的新特征，记为 feature_length，它就等于每个样本中的样本个数（即时间切片长度）。所以每条数据又多了这样的一个特征。

3）数据的预处理

除了给数据集添加了一些新的特征之外，为了更好地给网络提供训练数据，作者还对数据进行了以下比较重要的处理操作。

①z-score 归一化处理

考虑到训练集和测试集中特征值的差异比较大，为了网络更好地进行拟合，作者对数据使用 z-score 进行了归一化处理，我们借助代码进行理解。

具体代码如下：

```
def z_score(test):
    rows = test.index.tolist()
    columns = test.columns.tolist()
    for i in range(len(rows)):
        for j in range(len(columns)):
            test.loc[rows[i],columns[j]]=(test.loc[rows[i],columns[j]] - tongji.loc['mean',columns[j]])/tongji.loc['std',columns[j]]
    return test
```

我们发现，z-score 的具体做法是：对于一个样本中的时间切片序列，先计算出此样本中每个特征值在时间维度上的平均值和标准差，然后最后的归一化平均值就是用特征值减去平均值之后，再除以标准差：

$$v = \frac{v_0 - \overline{v_0}}{\delta(v_0)}$$

这样就让训练集和测试集中的数据更加平滑，也更加接近。

②时间切片长度统一

我们之前提到过，由于各个样本的时间切片长度都不一样，因此特地给样本加上了 feature_length 这样一个特征值。但是，在训练网络的时候，输入的样本矩阵的大小不一样，会给训练带来不便，因此作者对时间的切片长度进行了统一，使得每个样本的时间长度都为 30。

对于时间长度大于 30 的样本，直接取前 30 行进行截断，将后面的丢弃。这里需要注意的是，丢弃了之后，需要重新计算特征的平均值和标准差进行 z-score 归一化。

对于时间长度小于 30 的样本，则作者使用了每一个特征值的均值对数据进行填充，填充到等于 30 行。

③丢失数据填充

我们之前也提到过，赛事方提供的样本数据中，会有一些特征值的丢失，为了它们不影响训练，作者使用了如下办法进行填充：①如果样本中的某一特征值，只是有一些时间点上有缺失，那就用该特征值的平均值对缺失值进行填充；②如果样本中的某一特征值在所有时间点上全部缺失，那就用前一个样本和后一个样本在该特征值上的平均值去进行线性的填充。

②③两部分的代码如下图所示：

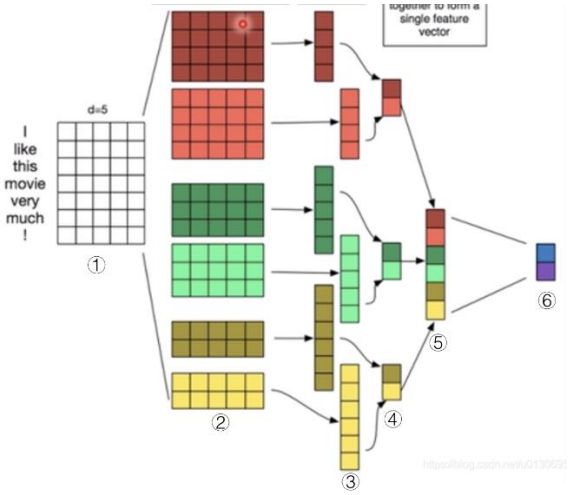
```
if len(m)>=30:
    m = m.iloc[:30,:]
    m = m[feature_list]
    for column in (m.columns[m.isnull().sum() > 0]).tolist():
        mean_val = m[column].mean()
        m[column].fillna(mean_val, inplace=True)
    for column in (m.columns[m.isnull().sum() > 0]).tolist():
        if column in look_up_column:
            val = look_up[look_up.index==i][column]
            m[column].fillna(val[i], inplace=True)
    m.to_csv('D:/2022 AIops/feature engineering/0120/test_30rows/{}.csv'.format(i))

if len(m)< 30:
    m = m[feature_list]
    add = m.mean().to_frame().T
    for t in range(30-len(m)):
        m = m.append(add)
    for column in m.columns[m.isnull().sum() > 0].tolist():
        mean_val = m[column].mean()
        m[column].fillna(mean_val, inplace=True)
    for column in (m.columns[m.isnull().sum() > 0]).tolist():
        if column in look_up_column:
            val = look_up[look_up.index==i][column]
            m[column].fillna(val[i], inplace=True)
```

代码中在时间序列长度大于和小于 30 的两种情况下，先进行了序列长度的统一。然后对于每个样本每行中缺失的数据进行了平均值的填充，具体原理如前所述。

3.网络结构：文章对于不同的归因预测，使用了不同的网络，比较有创新性的是基于 TextCNN 加入了 Attention 模块的网络，我重点对这个网络进行讲解。

TextCNN：

我们先介绍最基本的 TextCNN 结构，它的网络结构和处理过程如下示意图所示：



TextCNN 原本是用于对语言文本的处理上。首先输入端输入一个代表某个句子的矩阵①，然后 TextCNN 有几组不同大小的卷积核②，通过不同大小的卷积核②与①做卷积，可以得到不同长度的特征嵌入向量③，也就提取了句子中的不同特征。然后通过一个池化操作，将特征向量③转换为一个可以概括特征向量整体的值④，也是为了减少整个网络的计算量（有不

同的池化方法，这里不展开介绍）。然后将这些特征值进行连接，得到最后的特征⑤，然后连接一个全连接层再进行 softmax 处理，就可以得到最终的一个二分类的结果⑥，这两个结果的和为 1，分别代表两类的概率。

我们通过作者的代码，来看一看论文中的 TextCNN 是什么样的。具体代码如下：

```
##定义模型，设计模型参数
from keras.layers import Input
from tensorflow.python.keras import regularizers
pool_output = []
kernel_sizes = [3, 4, 5]
main_input = Input(shape=(X_train.shape[1],30), dtype='float64')
O_seq = attention_3d_block(main_input, X_train.shape[1])
for kernel_size in kernel_sizes:
    c = Conv1D(filters=32, kernel_size=kernel_size, padding='same', strides=1)(O_seq)
    c = BatchNormalization()(c)
    c = Activation('relu')(c)
    p = MaxPooling1D(pool_size=2)(c)
    p = Flatten()(p)
    pool_output.append(p)
x_flatten = concatenate(pool_output)
x_flatten = Dropout(0.4)(x_flatten)
y = Dense(2,activation ='softmax',kernel_regularizer=regularizers.l1(0.01))(x_flatten)
model = Model(inputs=main_input, outputs=y)
# model.summary()
```

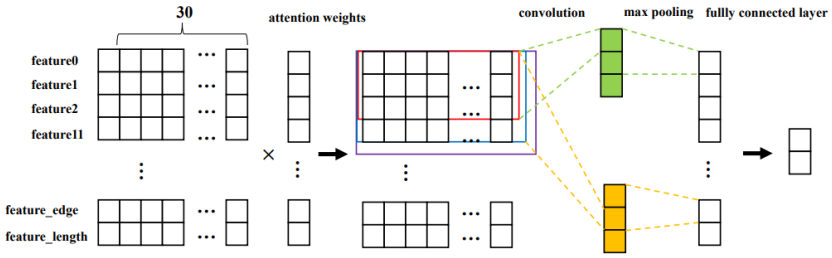
我们先不看 for 循环以上的 attention 部分，for 循环为我们展示了整个的 TextCNN 的具体结构，这里用到了 3，4，5 的三种不同分辨率大小的卷积核，每个大小的卷积核都会经过卷积+BN+ReLu+最大池化的操作，和我们介绍的原理相同，其中最大池化就是选择整个向量种最大的值作为我们尺寸减小后的特征值。

For 循环过后，我们已经得到了一个拼接好的特征向量，将它送入 dense 全连接层，并进行 softmax 处理，我们可以看到最终的输出通道数为 2，也刚好对应着我们的二分类结果，其中一类是该样本属于归因 1 的概率，另一类则是属于其他归因的概率。

Attention 模块：

Attention 模块也是自然语言处理领域中的一项处理方式，现在也被广泛应用于各大领域。Attention 的思想就是，由于自然语言处理中，相同的单词如果处于句子中的不同位置，那它的含义往往也是不一样的，因此需要通过 attention 机制去给这些不同位置的信息赋予不同的权重，即关注度，要让网络处理数据时更关注一些重要的信息。

论文中使用到的 attention 模块相对比较简单，整个网络的架构如下图所示：



由于我们的时间切片长度为 30，为了更充分地让网络用主要的注意力去关注比较重要的一些特征，论文通过一个 attention weights 矩阵（这个矩阵是可学习的），去将输入样本的矩阵转换为一个新的带有 attention 权重的矩阵，维度是不变的，然后再送入我们的 TextCNN 进行处理。具体的 attention 模块代码如下所示：

```
def attention_3d_block(inputs, time_steps):
    input_dim = int(inputs.shape[2])
    a = Permute((2, 1))(inputs)
    a = Dense(time_steps, activation='softmax')(a)
    # if False:
    #     a = Lambda(lambda x: K.mean(x, axis=1), name='dim_reduction')(a)
    #     a = RepeatVector(input_dim)(a)
    a_probs = Permute((2, 1), name='attention_vec')(a)
    print(a_probs.shape, inputs.shape)
    output_attention_mul = Multiply()([inputs, a_probs])
    return output_attention_mul
```

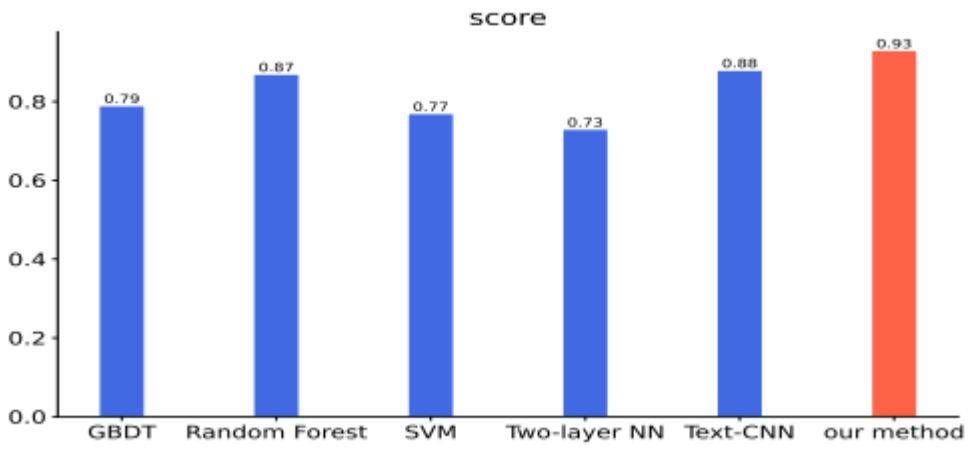
其中 a_probs 就是我们的权重矩阵，我们发现它是将我们的样本输入到一个全连接层中得到的，因此这个矩阵是可以通过梯度下降去学习的。然后最后的 attention 模块输出，就是用样本数据乘上我们的注意力矩阵即可。

至此，整个网络的架构我们已经全部梳理完毕，有关网络融合的部分放在了第四部分与大家分享。

四. 实验结果及讨论

讨论：

其实本来我们的网络的输出应该是一个多分类的结果，因为有多种不同归因的组合方式。但是作者这里进行了多个网络的对比试验，不仅使用了 TextCNN，还是用了 RandomForest，SVM 等方式，最后的结果如下所示：



其中 Text-CNN 和 Random Forest 的得分最高，同时，二者对于结果的预测度的相关性很低，但是得分却很相近。这说明，Text-CNN 和 Random Forest 在一些特定的样本上表现的很好，但是这些二者表现好的样本是不重合的！这就给作者提供了一个思路，那是不是可以二者同时使用，Text-CNN 只用于它预测正确率高的样本上，Random-Forest 只用在它预测正确率高的样本上，这样一来就进一步提升了网络性能。

根据实验结果进一步划分，最终作者确定了使用 Text-CNN 去预测归因 1，使用 Random-Forest 去预测归因 2 和 3 的策略。借鉴 one-vs-rest 的思想，用 Text-CNN 去预测归因 1 时，就把问题看作一个二分类问题，即归因 1 的类别和其他归因的类别进行二分类处理，而在使用 Random-Forest 的时候也是如此。最终形成了一个多网络融合的结构。

结果：

在讨论中已经给出了最终的得分情况，作者使用的多网络融合的方法最终按照大赛规则获得了 0.93 分的得分，最终得分排名所有参赛队伍的第四名。

五.总结与思考

读了学长的这篇参赛文章，我觉得对于我拿到一些 AI 有关的实际问题的时候，思路上有了很大的帮助。

1.做好数据的分析和处理。对于深度学习的问题，影响网络性能以及训练的很大一部分来自于对训练数据的处理。在本文中，由于训练的数据很乱很杂，数量又很多，因此不能盲目地去用这些数据，学长通过分析了它们之间的因果性和关联性，选择了部分合适的数据用于网络训练；此外，提供的原始数据之外还有一些隐含的数据特征，学长通过分析题目所给条件，提取出了一些新的特征作为数据；同时，由于数据的杂乱，需要对数据做补全和截断等处理，让数据集的各样本的客观因素基本相同，对实验结果没有太大的影响，我觉得这是非常重要的一点。

2.利用数据的物理意义去搭建合适的网络。我觉得深度学习神经网络不能看做一个纯数字处理的问题，就是盲目地更改网络结构，然后输入一批数据，再输出一批数据，我觉得这样的优化很没有意义。我们需要去理清楚问题本身的物理意义以及数据的物理含义，然后对应的网络结构才能有目的和方向的去优化。比如，论文分析出了多个归因和单个归因对数据结果的影响，因此需要采用不同的适合的方式去做优化；而引入了 Attention 结构，也是为了寻找各个特征之间更加重要的特征。

最后，我觉得通过对应用层模块的学习，让自己能够对深度学习有了更多的了解，也激发了自己很大的兴趣，目前自己也正在接触通信中的深度学习的一些其他应用，希望能够通过 AI 来给通信行业带来更多的提升。