**Assignment Lab**

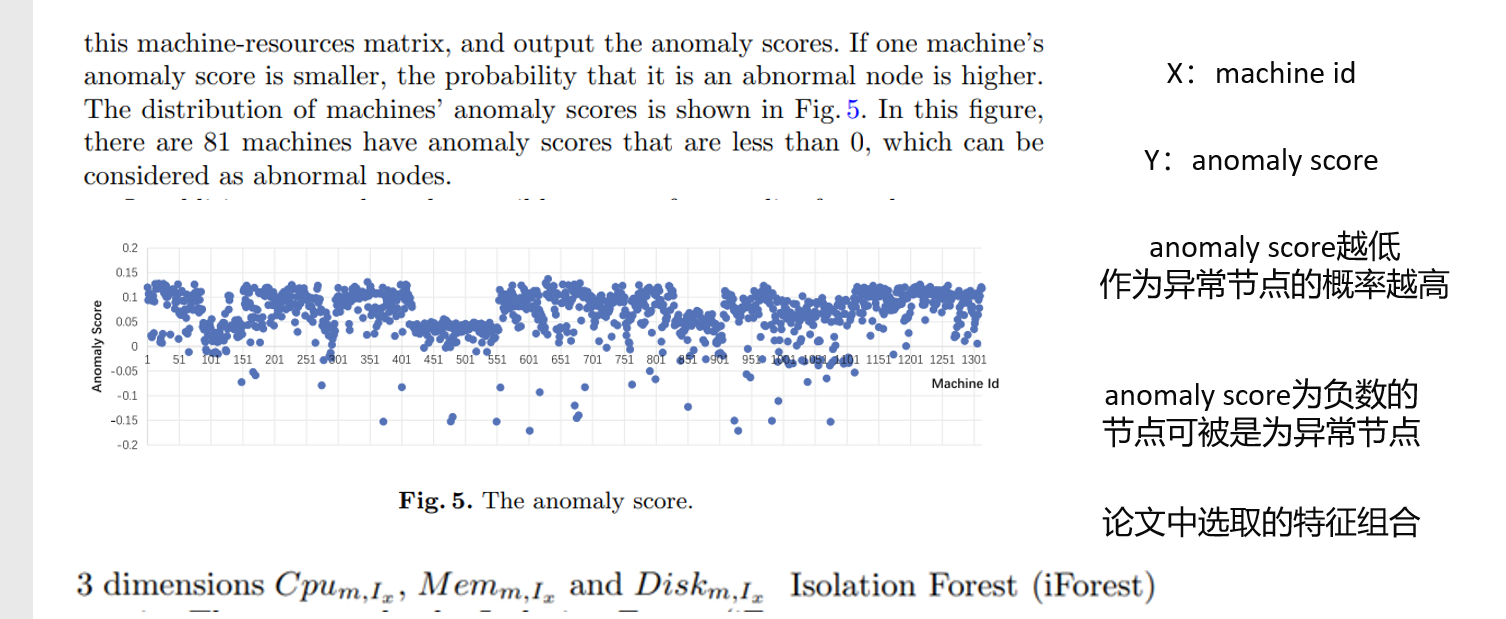
**Report**

09020328 王亮

# **I问题描述**

1. 获取不同特征组合的预测准确率，观察特征组合特点
2. 获取准确率前5的特征组合，绘制不同特征组合下各机器预测分数曲线，观察差异与变化趋势。

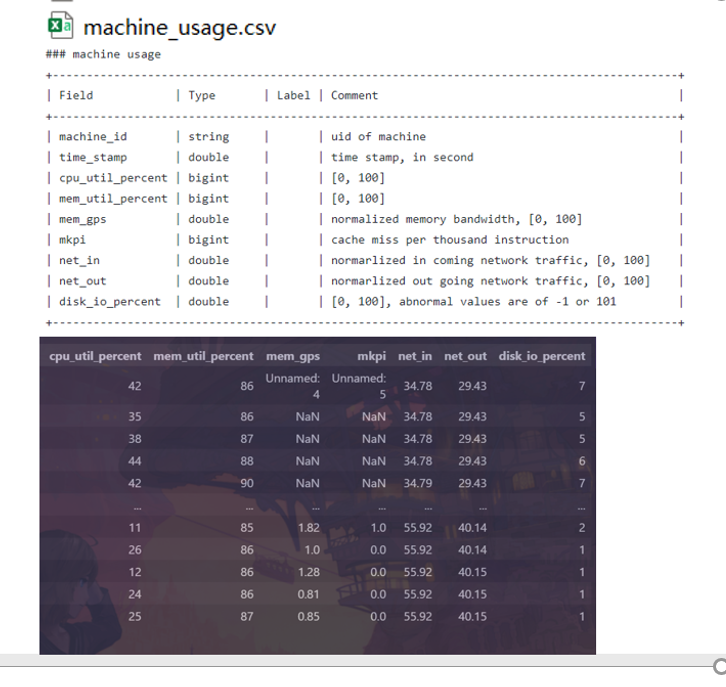
# **II 实验算法流程与数据**



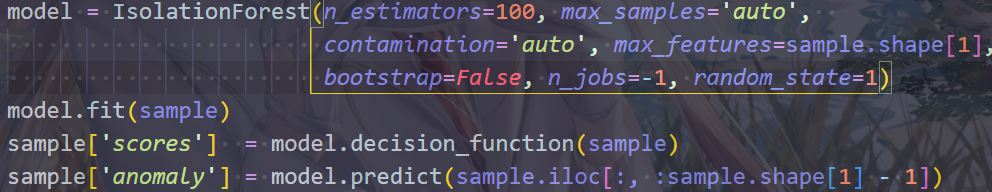
参引自论文中对于异常检测部分提到的：对于给定机器资源的矩阵（也就是选取的特征组合），通过异常检测的方法输出对应的异常分数。如果一个机器的异常分数越小，那么对应于他是异常节点的可能性就越大，而若异常分数小于0，则可以认定对应机器为异常节点。结合论文中给定的参考图，可以得出，在数据获取上，单个特征组合曲线需要获取的内容即为每个machine\_id与其对应的anomaly score。本次实验最后将多个曲线叠加至一张图比较即可。在anomaly score的获取上，结合孤立森林(isolation Forest)算法，对数据进行处理，获取目标数据。



在数据集的选取上，结合论文参考部分提供的github地址，在阿里的仓库下选取了2018年的数据集，具体数据表的各字段如图。



缺失数据的字段直接排除，不出现在后续处理中。



实现上调用scikit learn中封装算法

# **III 实验与程序设计**

由于时间略显久远，并且代码片段（都是写在jupyter notebook一个个的cell里）散落各处，以下提供代码可能不能直接复现出结果，不过用于阐述流程原理还是足够的，流程相当简单，容易自行补充复现。

部分import

*import* numpy   *as* np

*import* pandas  *as* pd

*import* seaborn *as* sns

*import* matplotlib.pyplot *as* plt

*from* sklearn.ensemble *import* IsolationForest

剩余五个特征进行排列组合，分别求出31种组合各自的预测准确率。

*import* warnings

warnings.filterwarnings("*ignore*")

*import* itertools

*import* os

mask\_set = ['*mem\_util\_percent*', '*cpu\_util\_percent*', '*disk\_io\_percent*', '*net\_in*', '*net\_out*']

len\_mask\_set = len(mask\_set)

all\_combination = list([list(itertools.combinations(mask\_set, i)) *for* i *in* range(1, len\_mask\_set + 1)])

[len(i) *for* i *in* all\_combination]

mask\_set = []

*for* i *in* range(1, len\_mask\_set + 1):

  mask\_set.append([[\*j] *for* j *in* all\_combination[i - 1]])

*# print(mask\_set)*

dir = '*./result\_set*'

*if* not os.path.exists(dir):

    os.mkdirs(dir)

*for* i *in* range(1, len\_mask\_set + 1):

  print('*-----------len: %d*' % i)

  print()

  sample\_set = [

    df[j] *for* j *in* mask\_set[i - 1]

  ]

*for* sample *in* sample\_set:

    model = IsolationForest(*n\_estimators*=100, *max\_samples*='*auto*',

*contamination*='*auto*', *max\_features*=sample.shape[1],

*bootstrap*=*False*, *n\_jobs*=-1, *random\_state*=1)

    model.fit(sample)

    sample['*scores*']  = model.decision\_function(sample)

    sample['*anomaly*'] = model.predict(sample.iloc[:, :sample.shape[1] - 1])

    err\_predict = sample[((df.disk\_io\_percent == -1) | (df.disk\_io\_percent == 101)) & (sample.anomaly != -1)]

    err\_predict2 = sample[((df.disk\_io\_percent != -1) & (df.disk\_io\_percent != 101)) & (sample.anomaly == -1)]

    success\_rate = 1 - (err\_predict.\_\_len\_\_() + err\_predict2.\_\_len\_\_()) / sample.\_\_len\_\_()

    print(''.join([str(i) *for* i *in* sample *if* str(i) != '*scores*' and str(i) != '*anomaly*']) + '*:* ' + str(success\_rate))

    sample['*machine\_id*'] = df['*machine\_id*']

    sample.to\_csv('*result\_set/*' + (''.join([str(i) *for* i *in* sample *if* str(i) != '*scores*' and str(i) != '*anomaly*']) + '*.csv*'))

结果数据：

-----------len: 1

mem\_util\_percent: 0.8329309262745966

cpu\_util\_percent: 0.7034444567504707

disk\_io\_percent: 0.783527153246945

net\_in: 0.7092553623509432

net\_out: 0.7151548713404954

-----------len: 2

mem\_util\_percent cpu\_util\_percent: 0.7715509284896814

mem\_util\_percent disk\_io\_percent: 0.8348580499870787

mem\_util\_percent net\_in: 0.7975338723372836

mem\_util\_percent net\_out: 0.8103739801380736

cpu\_util\_percent disk\_io\_percent: 0.7365230553402

cpu\_util\_percent net\_in: 0.7127625798353454

cpu\_util\_percent net\_out: 0.731553881935984

disk\_io\_percent net\_in: 0.7826706538191752

disk\_io\_percent net\_out: 0.800243659319969

net\_in net\_out: 0.7067818510724702

-----------len: 3

mem\_util\_percent cpu\_util\_percent disk\_io\_percent: 0.8294384760217078

mem\_util\_percent cpu\_util\_percent net\_in: 0.7956732011666112

mem\_util\_percent cpu\_util\_percent net\_out: 0.7814818916823568

mem\_util\_percent disk\_io\_percent net\_in: 0.8414368516262414

mem\_util\_percent disk\_io\_percent net\_out: 0.8390224092738214

mem\_util\_percent net\_in net\_out: 0.7595747037324178

cpu\_util\_percent disk\_io\_percent net\_in: 0.7810314911212021

cpu\_util\_percent disk\_io\_percent net\_out: 0.7836748255620778

cpu\_util\_percent net\_in net\_out: 0.7370768265219478

disk\_io\_percent net\_in net\_out: 0.7681323143943589

-----------len: 4

mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in: 0.8611363384649463

mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_out: 0.8531620334477794

mem\_util\_percent cpu\_util\_percent net\_in net\_out: 0.8011370768265219

mem\_util\_percent disk\_io\_percent net\_in net\_out: 0.8399527448591575

cpu\_util\_percent disk\_io\_percent net\_in net\_out: 0.7965887695204341

-----------len: 5

mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in net\_out: 0.8426256137630598

筛选top10

data\_len = {}

*for* i *in* [len1, len2, len3 ,len4 , len5]:

  data\_len = dict(*\*\**data\_len, *\*\**i)

*import* operator

*# data\_len*

sorted(data\_len.items(), *key*=operator.itemgetter(1), *reverse*=*True*)[:10]

结果：

[('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in*',

  0.8611363384649463),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_out*',

  0.8531620334477794),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in net\_out*',

  0.8426256137630598),

 ('*mem\_util\_percent disk\_io\_percent net\_in*', 0.8414368516262414),

 ('*mem\_util\_percent disk\_io\_percent net\_in net\_out*', 0.8399527448591575),

 ('*mem\_util\_percent disk\_io\_percent net\_out*', 0.8390224092738214),

 ('*mem\_util\_percent disk\_io\_percent*', 0.8348580499870787),

 ('*mem\_util\_percent*', 0.8329309262745966),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent*', 0.8294384760217078),

 ('*mem\_util\_percent net\_out*', 0.8103739801380736)]

绘制top5

*from* pyecharts.charts *import* Bar

*from* pyecharts *import* options *as* opts

*from* pyecharts.globals *import* ThemeType

bar\_list = []

count = 0

*for* data\_len *in* data\_full\_set:

*if* count == 0:

    count += 1

*continue*

*if* file\_list[count] not in [i[0] + ' *machine\_id.csv*' *for* i *in*[('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in*',

  0.8611363384649463),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_out*',

  0.8531620334477794),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in net\_out*',

  0.8426256137630598),

 ('*mem\_util\_percent disk\_io\_percent net\_in*', 0.8414368516262414),

 ('*mem\_util\_percent disk\_io\_percent net\_in net\_out*', 0.8399527448591575),

*#  ('mem\_util\_percent disk\_io\_percent net\_out', 0.8390224092738214),*

*#  ('mem\_util\_percent disk\_io\_percent', 0.8348580499870787),*

*#  ('mem\_util\_percent', 0.8329309262745966),*

*#  ('mem\_util\_percent cpu\_util\_percent disk\_io\_percent', 0.8294384760217078),*

*#  ('mem\_util\_percent net\_out', 0.8103739801380736)*

 ]]:

    count += 1

*continue*

  bar = Bar(*init\_opts*=opts.InitOpts(*width*="*1900px*",

*height*="*900px*",

*page\_title*="",

*theme*=ThemeType.LIGHT))

*#定义x轴*

  bar.add\_xaxis([\*data\_len.index])

*#定义y轴*

  bar.add\_yaxis(file\_list[count], [\*data\_len],*label\_opts* = opts.LabelOpts(*is\_show*=*False*))

  bar.set\_global\_opts(

*title\_opts*=opts.TitleOpts(*title*="*Bar-DataZoom（slider-水平）*"),

*datazoom\_opts*=[opts.DataZoomOpts(*type\_*="*slider*",)],

*toolbox\_opts*=opts.ToolboxOpts(

*# 是否显示该工具*

*is\_show*=*True*,

              ),

          )

  count += 1

  bar\_list.append(bar)

start\_bar = Bar(*init\_opts*=opts.InitOpts(*width*="*1900px*",

*height*="*900px*",

*page\_title*="",

*theme*=ThemeType.LIGHT))

*#定义x轴*

start\_bar.add\_xaxis([\*data\_full\_set[0].index])

*#定义y轴*

start\_bar.add\_yaxis(file\_list[0], [\*data\_full\_set[0]],*label\_opts* = opts.LabelOpts(*is\_show*=*False*))

start\_bar.set\_global\_opts(

*title\_opts*=opts.TitleOpts(*title*="*Bar-DataZoom（slider-水平）*"),

*datazoom\_opts*=[opts.DataZoomOpts(*type\_*="*slider*",)],

*toolbox\_opts*=opts.ToolboxOpts(

*is\_show*=*True*,

            ),

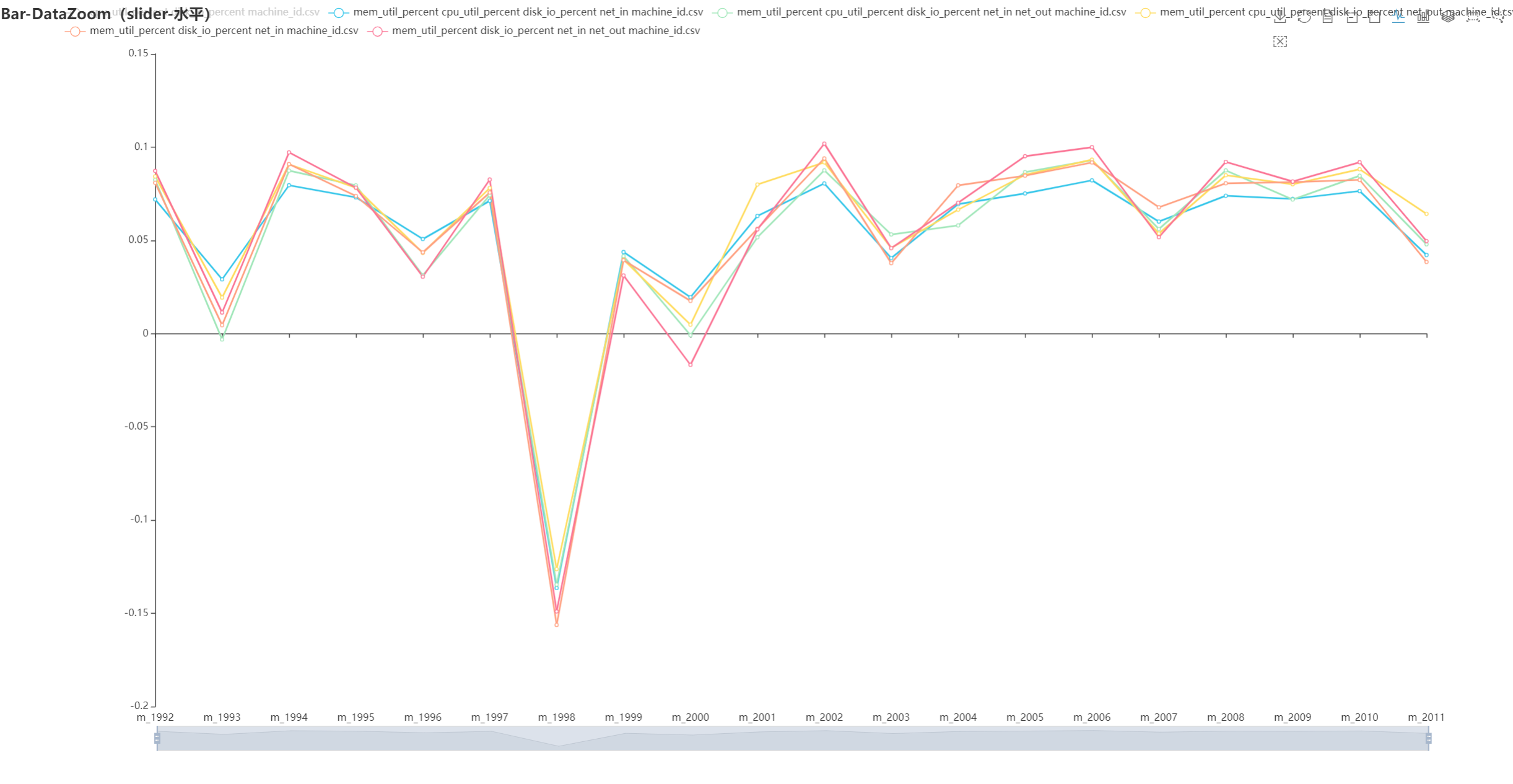
        )

*for* i *in* bar\_list:

  start\_bar.overlap(i)

start\_bar.render("*final\_ref2.html*")

结果：



# **实验总结**

[('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in*',

  0.8611363384649463),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_out*',

  0.8531620334477794),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent net\_in net\_out*',

  0.8426256137630598),

 ('*mem\_util\_percent disk\_io\_percent net\_in*', 0.8414368516262414),

 ('*mem\_util\_percent disk\_io\_percent net\_in net\_out*', 0.8399527448591575),

 ('*mem\_util\_percent disk\_io\_percent net\_out*', 0.8390224092738214),

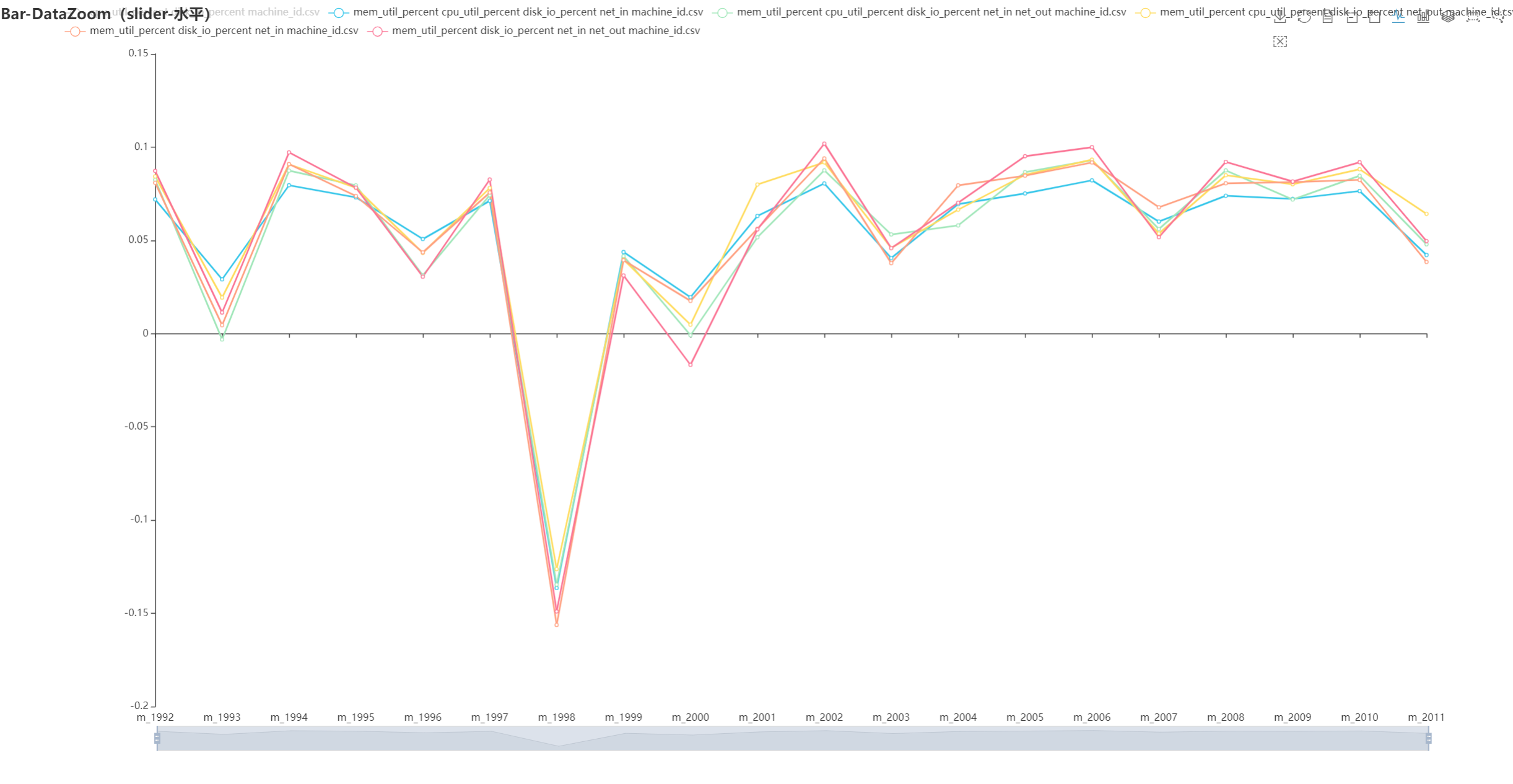
 ('*mem\_util\_percent disk\_io\_percent*', 0.8348580499870787),

 ('*mem\_util\_percent*', 0.8329309262745966),

 ('*mem\_util\_percent cpu\_util\_percent disk\_io\_percent*', 0.8294384760217078),

 ('*mem\_util\_percent net\_out*', 0.8103739801380736)]

观察到mem\_util\_percent是一个很好的有效特征，在此基础上加入合适的特征有助于准确率提高，反之可能干扰判断，降低准确率。



总体各曲线差异不大，在一定的误差区间范围内。准确率最高的组合给出的异常分基本是各曲线相同机器下的最大或最小值，可以借此推测准确率更高的曲线大致的各机器异常分取值情况。