Comparative Method - Part(A)

Rainforest Connection Species Audio Detection

By: Somayyeh Gholami & Mehran Kazeminia

Description:

- At the end of the challenge, Mr. @meaninglesslives shared his notebook. He won third place in the challenge. The score of the notebook published in the first version is "public score 0.96171 and private score 0.96460". Thanks for sharing the results, we congratulate him too.

https://www.kaggle.com/meaninglesslives/rfcx-minimal?scriptVersionId=54514070

- Then Mr. @cdeotte released another notebook and with a great trick, raised the previous notebook's private score above 0.970. We also thank him for sharing this trick.

https://www.kaggle.com/cdeotte/rainforest-post-process-lb-0-970

https://www.kaggle.com/c/rfcx-species-audio-detection/discussion/220389

An important question:

Does this trick improve all the results (all the columns)?

No, of course the results of some columns are getting worse.

This means that the results of some columns get very good and the results of some columns get worse, but in this challenge (and usually) the overall results improve.

To prove this, we wrote this notebook and share it with you. Our method is very simple. We first identified nine columns, the results of which will be reduced by performing this trick.

S0, S1, S4, S8, S11, S14, S18, S20, S21

Then we transferred the results of these columns from the original notebook (the first notebook) and replaced these results exactly with the results of the second notebook, and finally saved the entire result in the "d" file. That is, in file "d", a trick is applied for the results of 15 columns and no trick is applied for the results of 9 columns. The scores of the "d" file are as follows:

"d" : [(Private Score: 0.97915) , (Public Score: 0.97373)]

As you can see, this trick is not good for the results of these nine columns, and we got a much better score with the results of the original notebook (first version). Please note that in order to be able to compare, we used exactly the results of the first version of the original notebook.

In the end, we were able to easily improve the results once again with our own method. We saved the final results in the "e" file. The scores of the "e" file are as follows:

"e" : [(Private Score: 0.98022) , (Public Score: 0.97490)]

If you find this work useful, please don't forget upvoting :)

Import & Data Set

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline

# ________________________________

sub961 = pd.read_csv("../input/rain961/RAIN961.csv")

sub968 = pd.read_csv("../input/rainforest-post-process-lb-0-970/submission_
```

Functions

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [2]:

```
def generate(main, support, coeff):
    g1 = main.copy()
    g2 = main.copy()
    g3 = main.copy()
    g4 = main.copy()

    for i in main.columns[1:]:
        lm, Is = [], []
        lm = main[i].tolist()
        ls = support[i].tolist()

        res1, res2, res3, res4 = [], [], [], []
        for j in range(len(main)):
            res1.append(max(lm[j] , ls[j]))
            res2.append(min(lm[j] , ls[j]))
            res3.append((lm[j] + ls[j]) / 2)
            res4.append((lm[j] * coeff) + (ls[j] * (1.- coeff)))

        g1[i] = res1
        g2[i] = res2
        g3[i] = res3
        g4[i] = res4

    return g1,g2,g3,g4
```

In [3]:

```
def generate1(main, support, coeff):

    g = main.copy()
    for i in main.columns[1:]:

        res = []
        lm, Is = [], []
        lm = main[i].tolist()
        ls = support[i].tolist()

        for j in range(len(main)):
            res.append((lm[j] * coeff[i]) + (ls[j] * (1.- coeff[i])))
        g[i] = res

    return g
```

In [4]:

```
def drawing(main, support, generated, column_number):

    X  = main.iloc[:, column_number]
    Y1 = support.iloc[:, column_number]
    Y2 = generated.iloc[:, column_number]

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nOn the X axis >>> main\n\nOn the Y axis >>> support\n')
    plt.scatter(X, Y1, s=3)
    plt.show()

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nOn the X axis >>> main\n\nOn the Y axis >>> generated\n'
    plt.scatter(X, Y2, s=3)
    plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [5]:
```
def drawing1(main, support, generated, column_number):

    X  = main.iloc[:, column_number]
    Y1 = support.iloc[:, column_number]
    Y2 = generated.iloc[:, column_number]

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(8, 8), facecolor='lightgray')
    plt.title(f'\nBlue | X axis >> main | Y axis >> support\n\nOrange | X a

    plt.scatter(X, Y1, s=3)
    plt.scatter(X, Y2, s=3)

    plt.show()
```

Comparative Method

In [6]:
```
# print(sub968.mean() , sub961.mean())

m1 = sub968.mean() + sub961.mean()

m1mean = m1.mean()

m2 = m1 / m1mean

m2
```

Out[6]:
```
s0     0.809025
s1     1.422532
s2     0.534986
s3     5.994374
s4     0.348289
s5     0.550068
s6     0.085884
s7     1.907275
s8     0.393510
s9     0.343576
s10    0.262067
s11    1.039271
s12    2.413379
s13    0.263196
s14    0.757137
s15    2.120569
s16    0.454357
s17    0.213483
s18    2.840838
s19    0.085247
s20    0.167267
s21    0.063781
s22    0.200014
s23    0.729876
dtype: float64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [7]:
```
m3 = m2.copy()
for k in range(24):
    m3[k] = 1.00

# m3
```

In [8]:
```
m4 = m3.copy()

m4[0]   = 0.00

m4[1]   = 0.00

m4[4]   = 0.00

m4[8]   = 0.00

m4[11]  = 0.00

m4[14]  = 0.00

m4[18]  = 0.00

m4[20]  = 0.00

m4[21]  = 0.00
```

Result

[(Private Score: 0.97892) , (Public Score: 0.97309)]

In [9]:
```
m4[15]   = 1.30

m4
```

Out[9]:
```
s0     0.0
s1     0.0
s2     1.0
s3     1.0
s4     0.0
s5     1.0
s6     1.0
s7     1.0
s8     0.0
s9     1.0
s10    1.0
s11    0.0
s12    1.0
s13    1.0
s14    0.0
s15    1.3
s16    1.0
s17    1.0
s18    0.0
s19    1.0
s20    0.0
s21    0.0
```

```
s23    1.0
dtype: float64
```

Result

[(Private Score: 0.97915) , (Public Score: 0.97373)]

In [10]:
```
d = generate1(sub968, sub961, m4)
```

In [11]:
```
sub968.describe()
```

Out[11]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1.992000
mean	0.110630	0.198951	0.058836	0.975920	0.027766	0.049287	3.443965
std	0.298508	0.369604	0.221785	0.095970	0.135427	0.195465	5.238256
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000017	0.000222	0.000010	0.997896	0.000018	0.000032	5.145849
50%	0.000137	0.002012	0.000070	0.999792	0.000089	0.000219	4.882412
75%	0.002537	0.082179	0.000732	0.999958	0.000677	0.001828	2.530490
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 24 columns

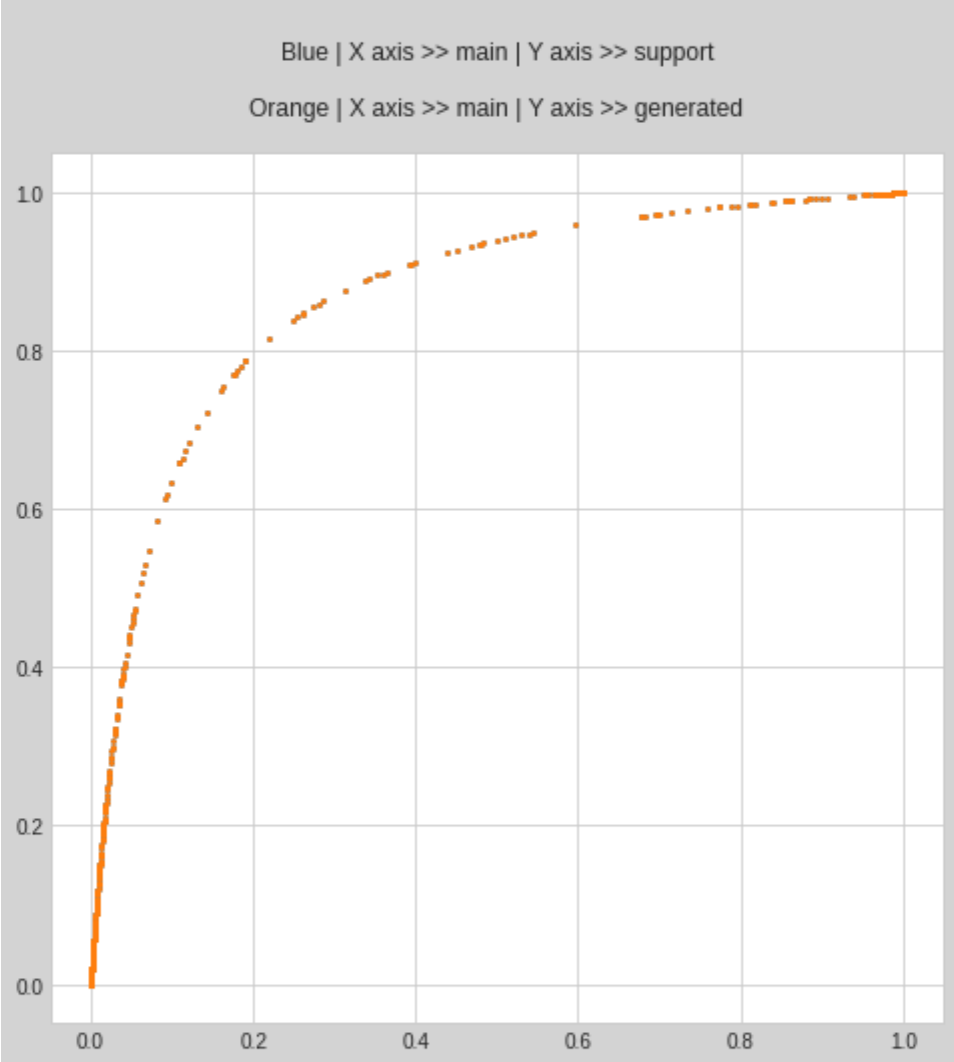
In [12]:
```
sub961.describe()
```

Out[12]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1.992000e+03	1992.000000	1992.000000	1992.000000	1992.00
mean	0.148069	0.255927	1.122349e-01	0.940881	0.083605	0.126606	0.02
std	0.324055	0.394491	2.696748e-01	0.166613	0.221616	0.264003	0.09
min	0.000001	0.000031	6.896084e-07	0.038022	0.000003	0.000004	0.00
25%	0.000264	0.001760	4.389717e-04	0.987995	0.000638	0.001514	0.00
50%	0.002151	0.015521	3.022099e-03	0.998800	0.003190	0.010141	0.00
75%	0.038402	0.411339	3.083062e-02	0.999755	0.023652	0.078918	0.01
max	0.999999	0.999999	1.000000e+00	0.999999	0.999976	0.999997	0.99

8 rows × 24 columns

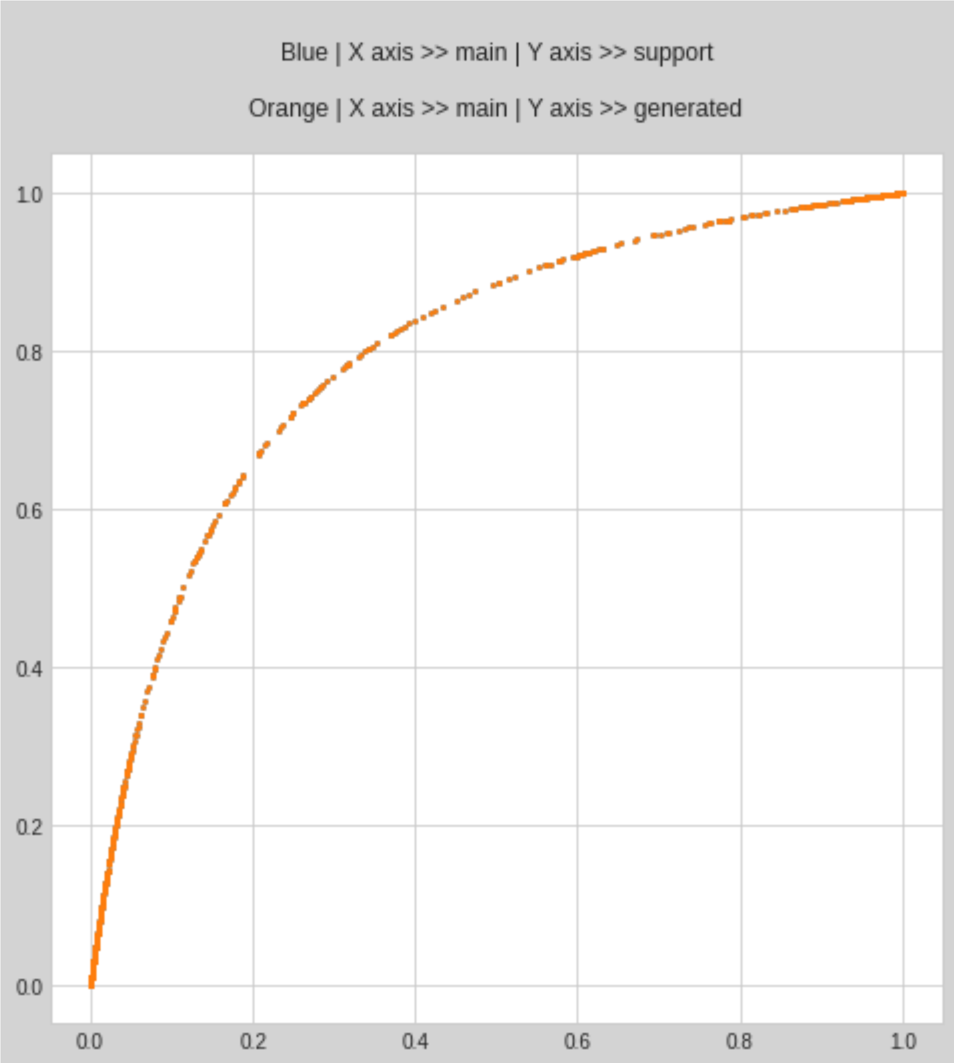
In [13]:
```
drawing1(sub968, sub961, d, 1)
```



In [14]:

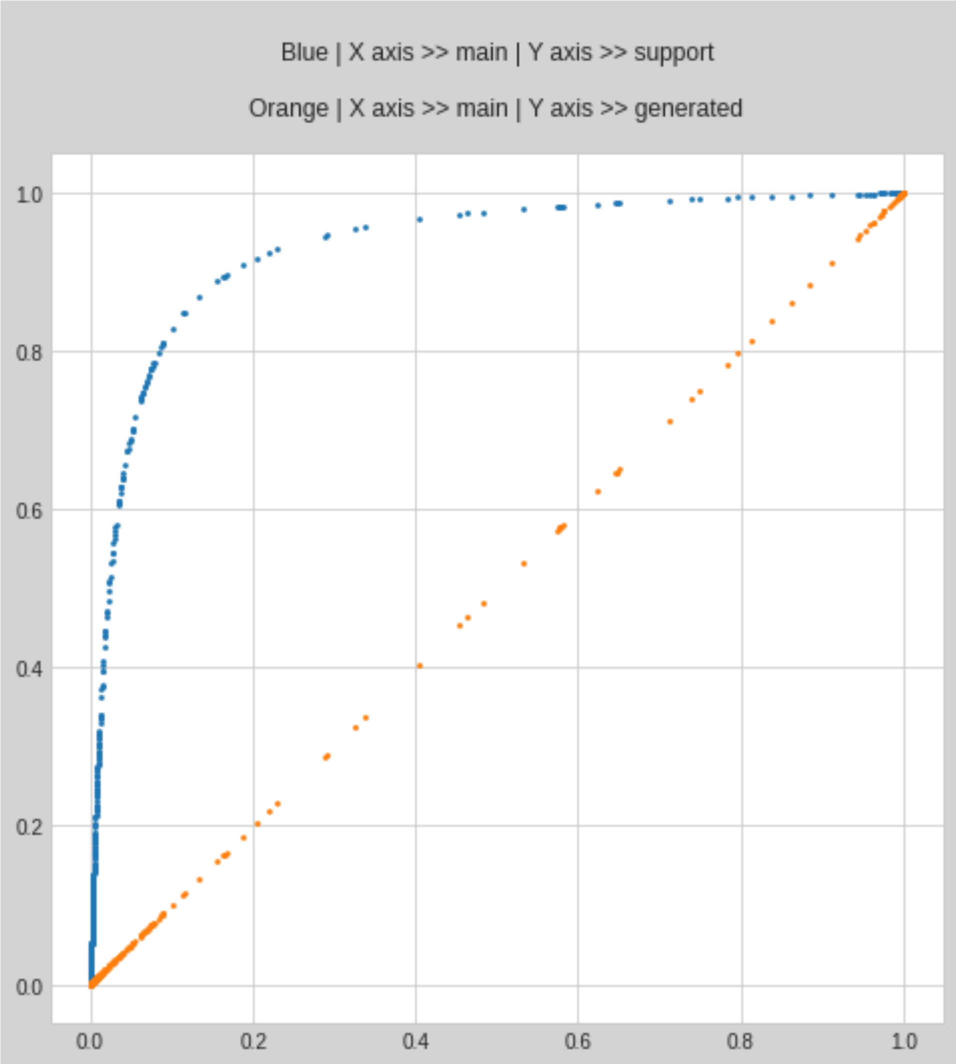
```
drawing1(sub968, sub961, d, 2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [15]:
```
drawing1(sub968, sub961, d, 3)
```

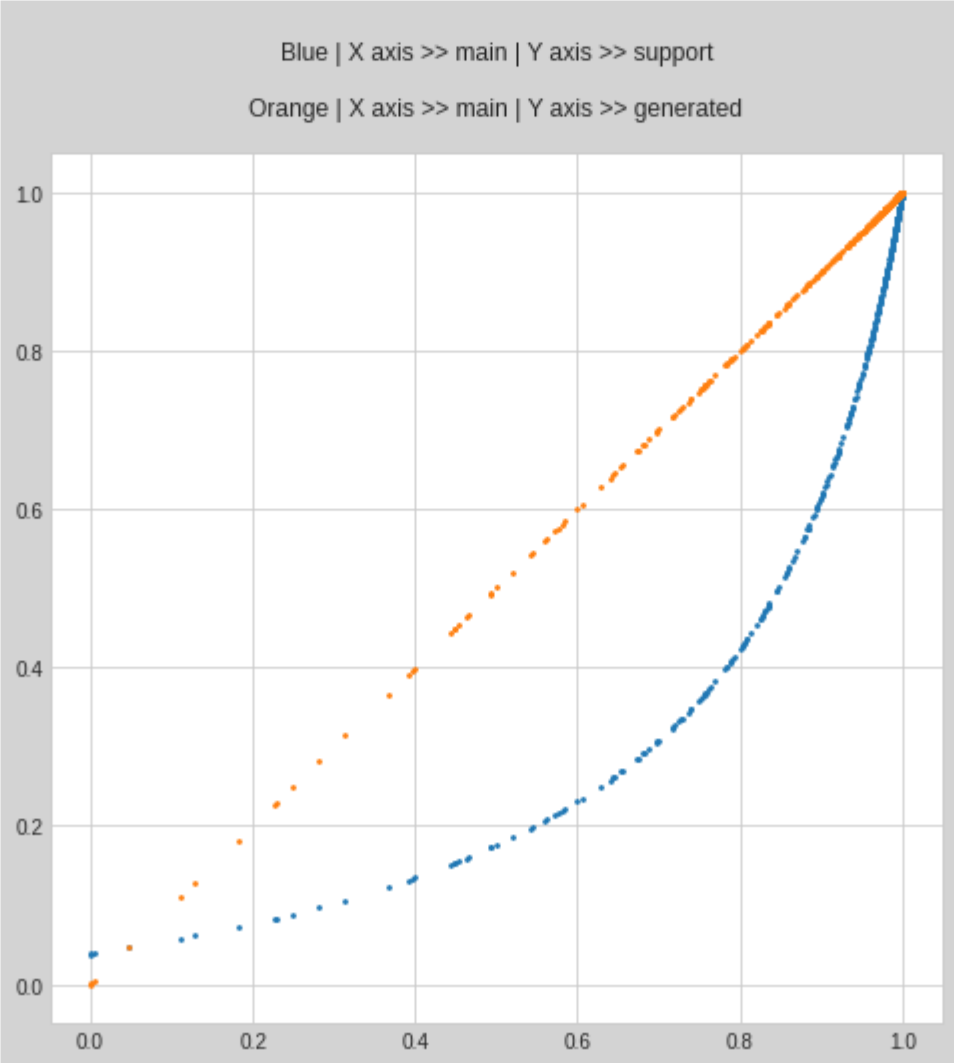
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [16]:

```
drawing1(sub968, sub961, d, 4)
```

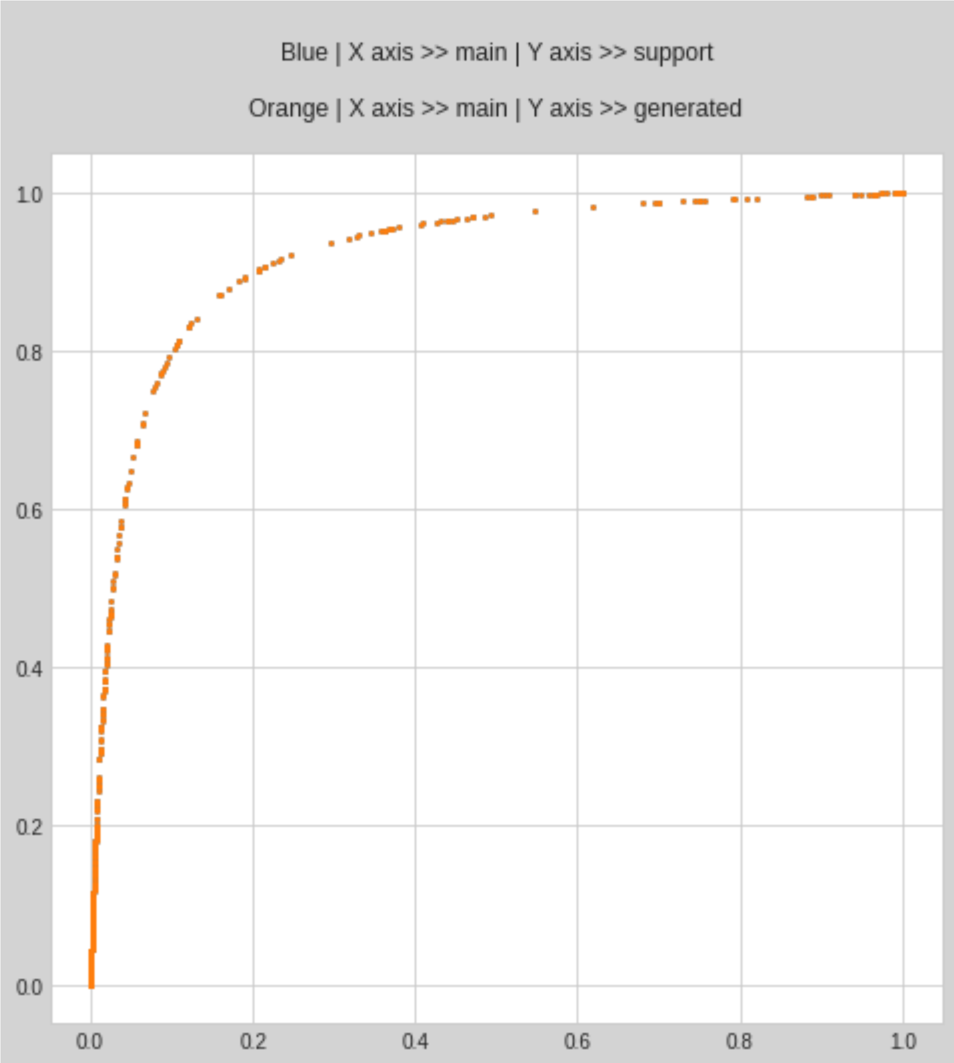
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [17]:

```
drawing1(sub968, sub961, d, 5)
```

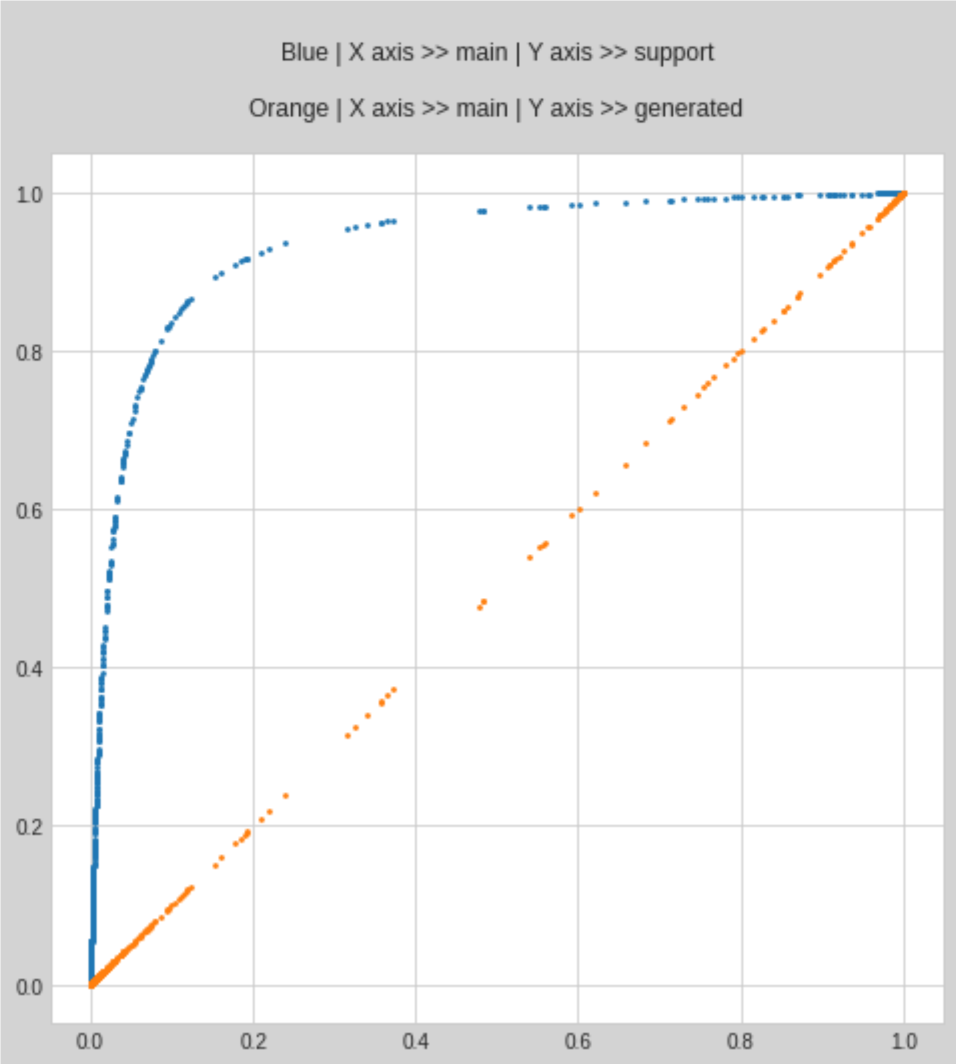
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [18]:

```
drawing1(sub968, sub961, d, 6)
```

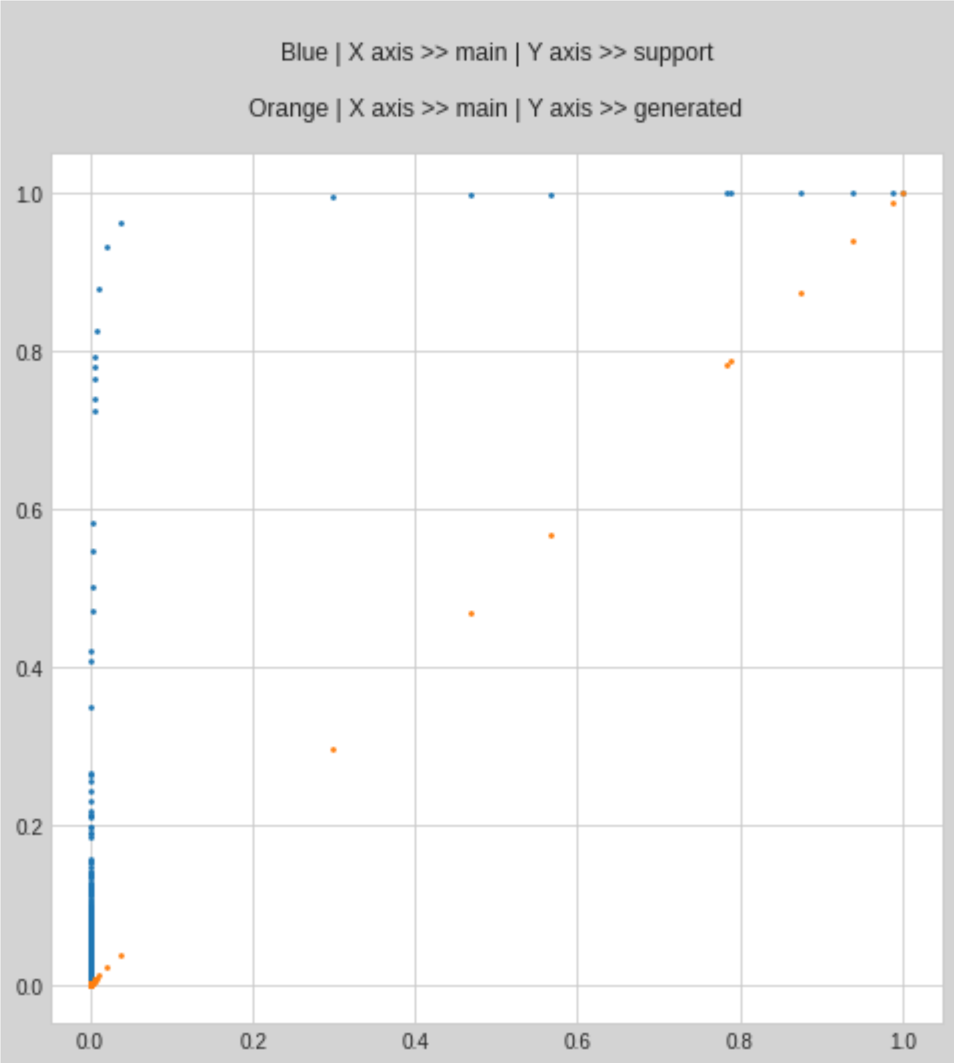
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [19]:

```
drawing1(sub968, sub961, d, 7)
```

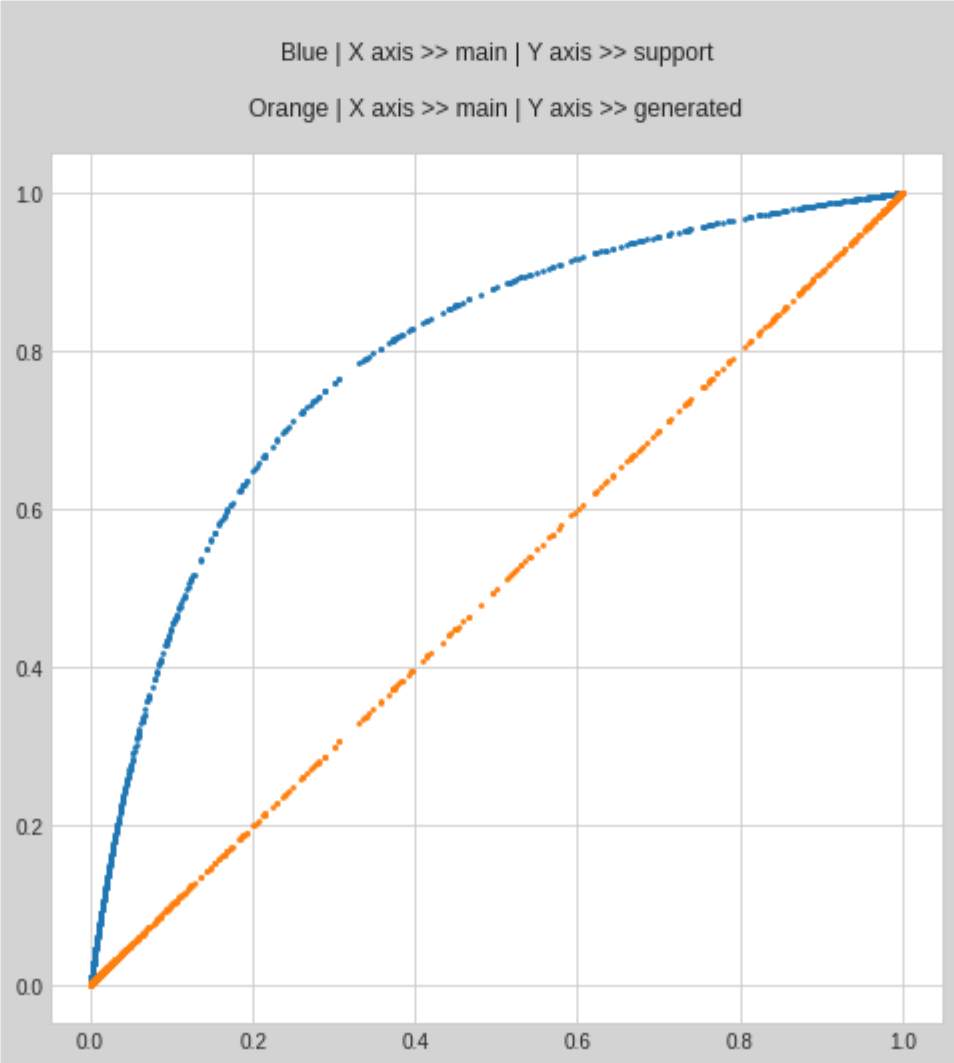
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [20]:

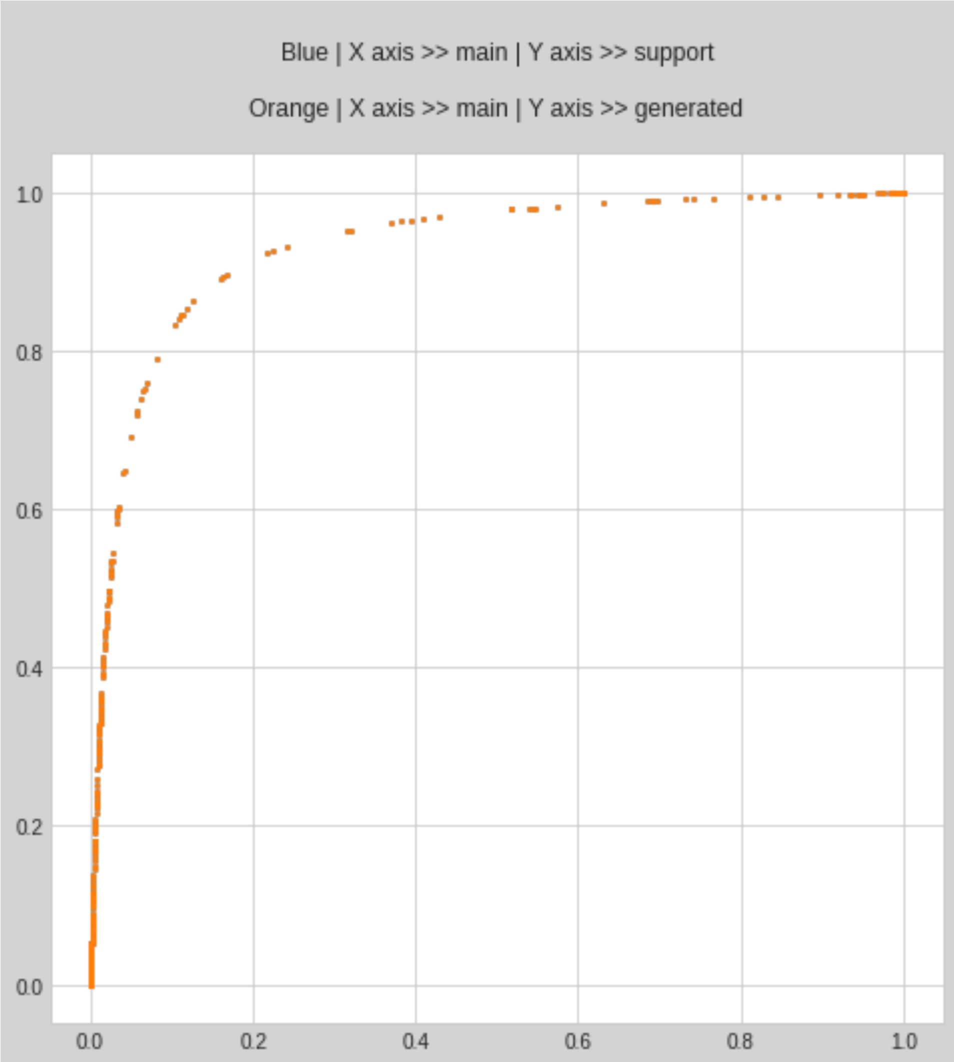
```
drawing1(sub968, sub961, d, 8)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [21]:

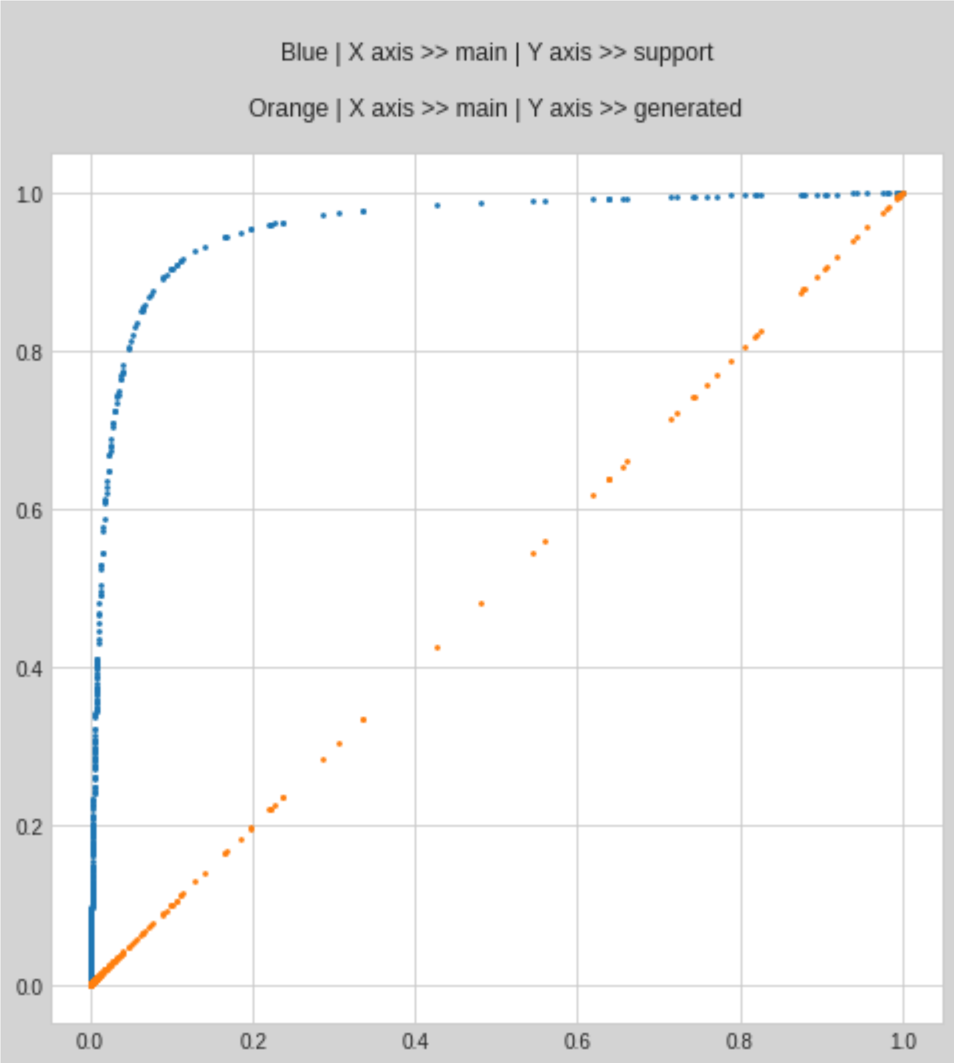
```
drawing1(sub968, sub961, d, 9)
```



In [22]:

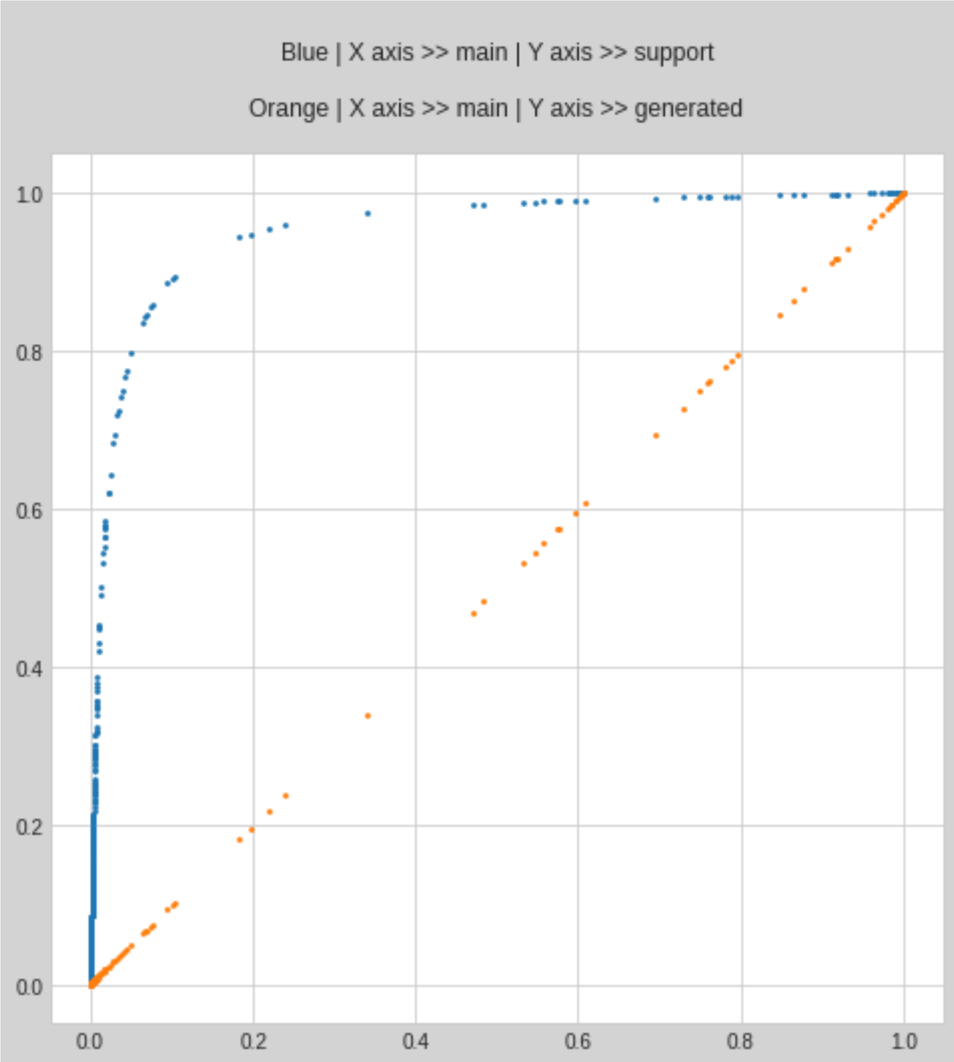
```
drawing1(sub968, sub961, d, 10)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [23]:
```
drawing1(sub968, sub961, d, 11)
```

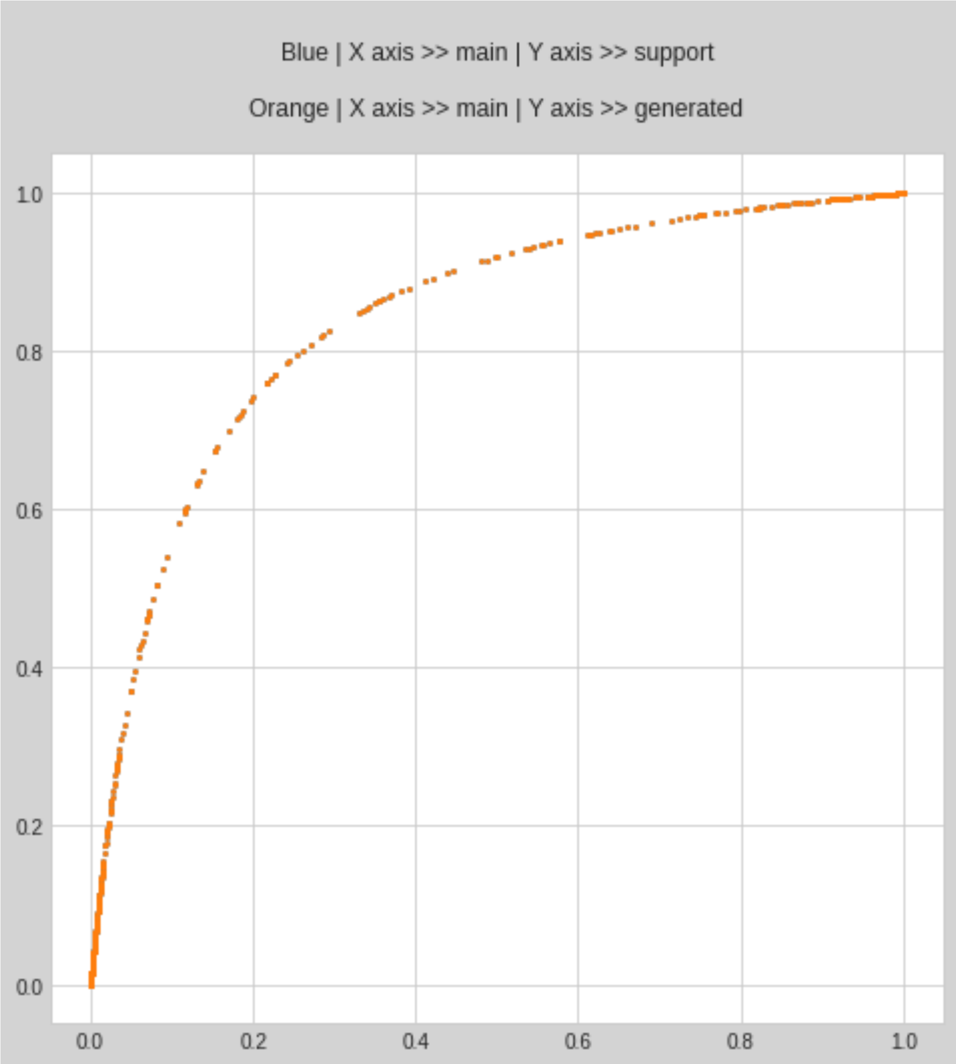
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [24]:

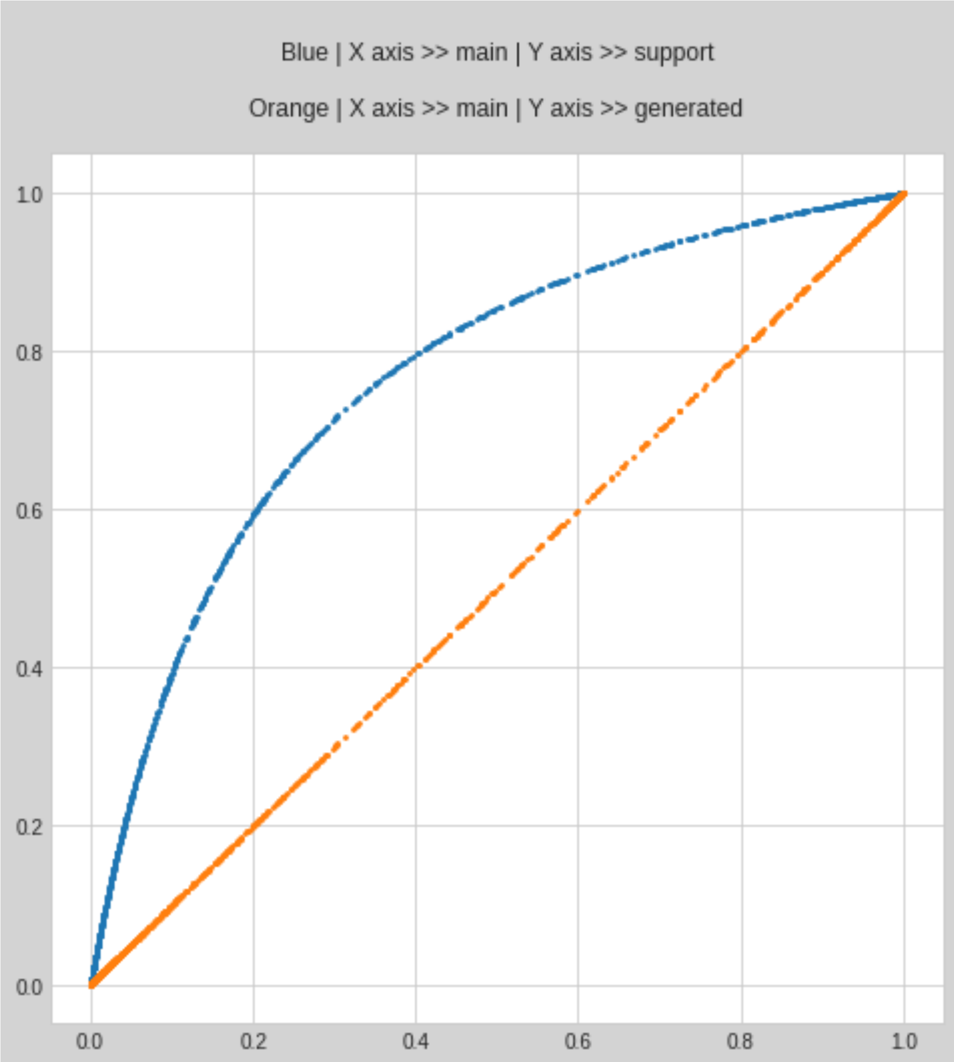
```
drawing1(sub968, sub961, d, 12)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



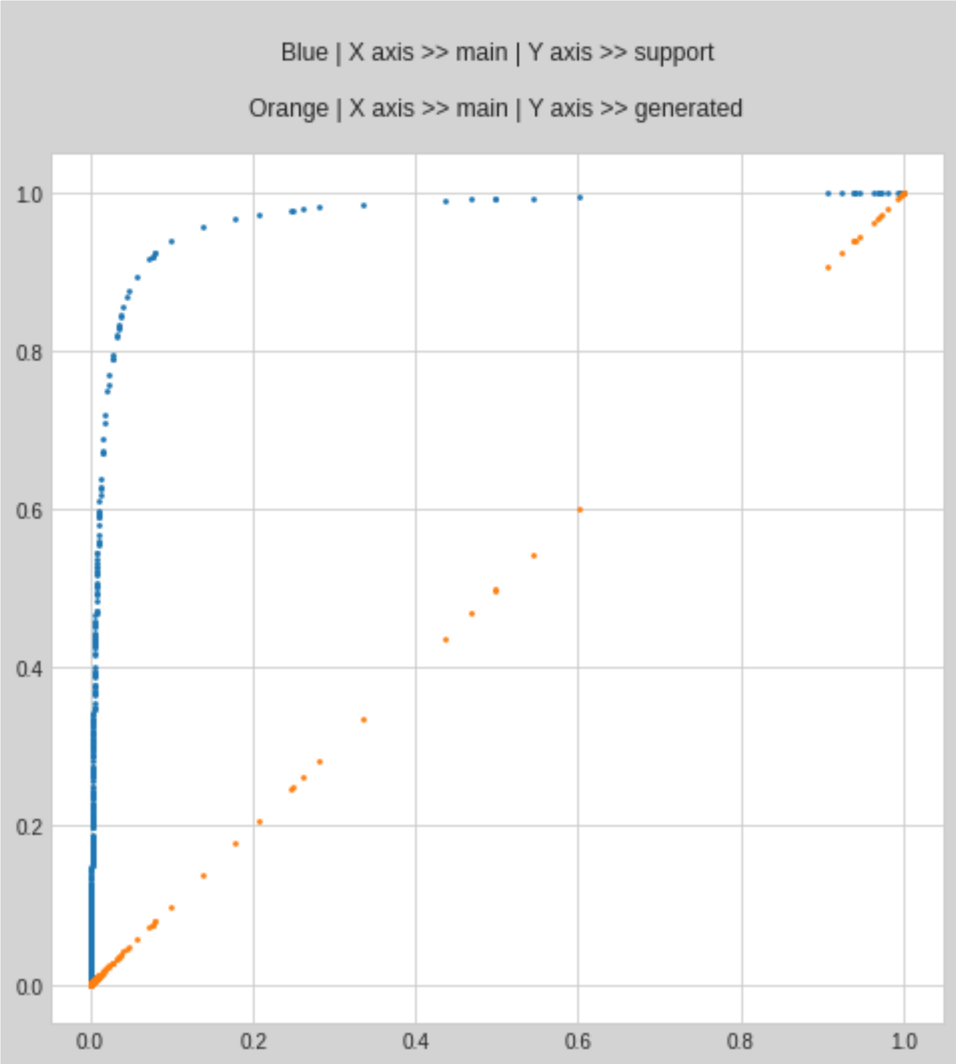
In [25]:

```
drawing1(sub968, sub961, d, 13)
```



In [26]:
```
drawing1(sub968, sub961, d, 14)
```

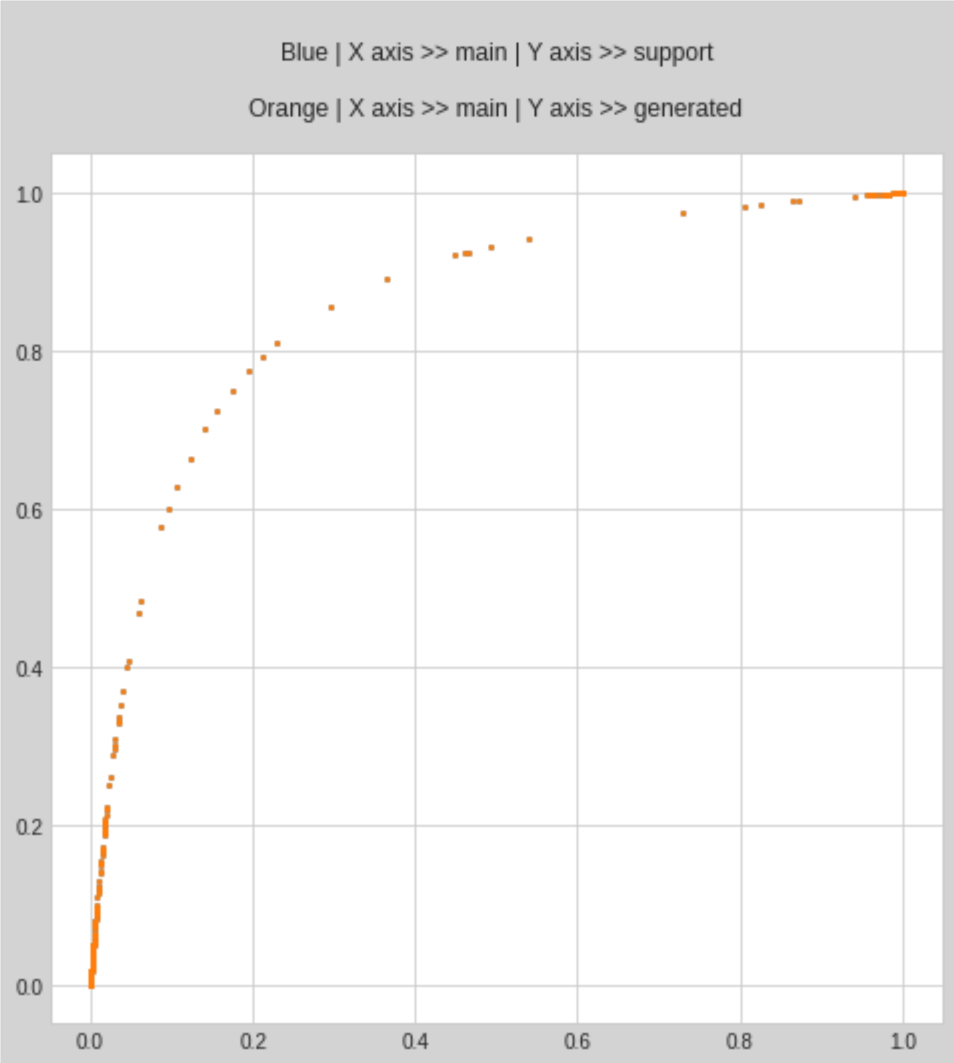
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [27]:

```
drawing1(sub968, sub961, d, 15)
```

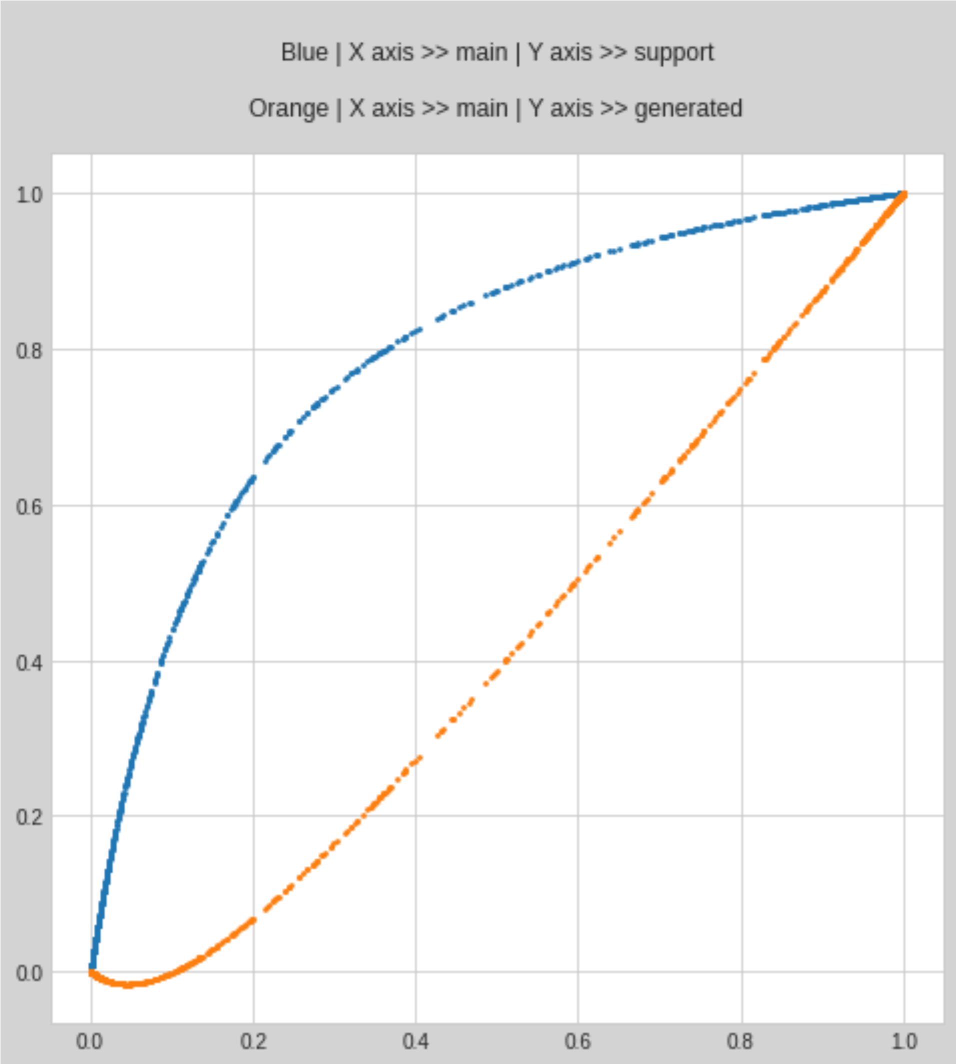
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [28]:

```
drawing1(sub968, sub961, d, 16)
```

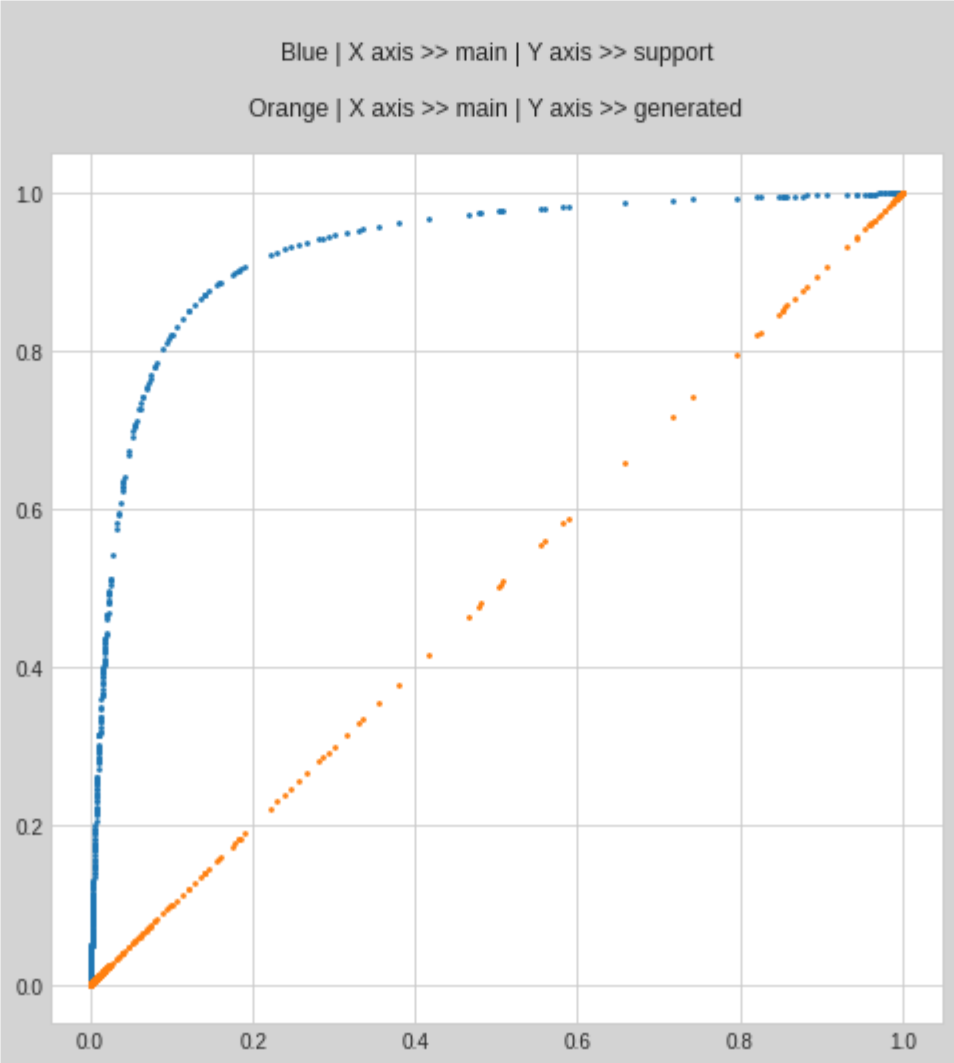
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [29]:

```
drawing1(sub968, sub961, d, 17)
```

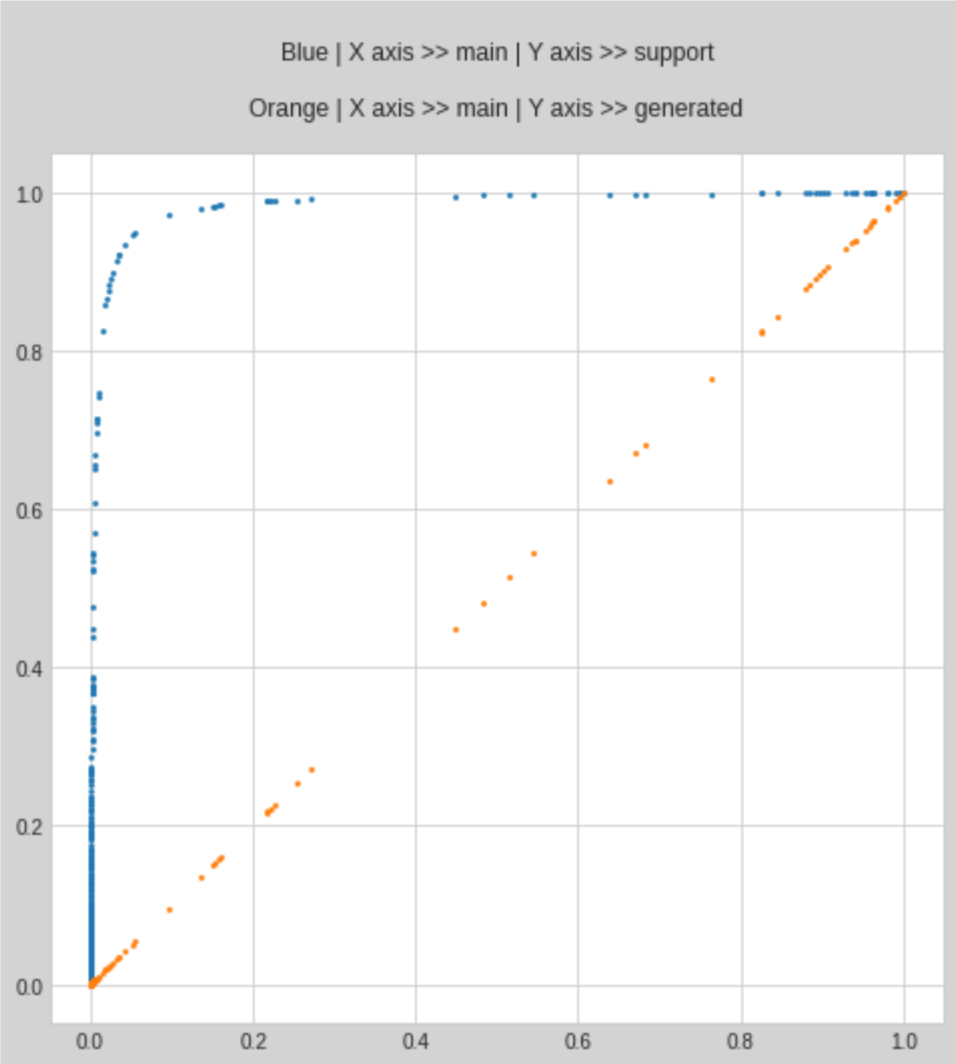
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [30]:

```
drawing1(sub968, sub961, d, 18)
```

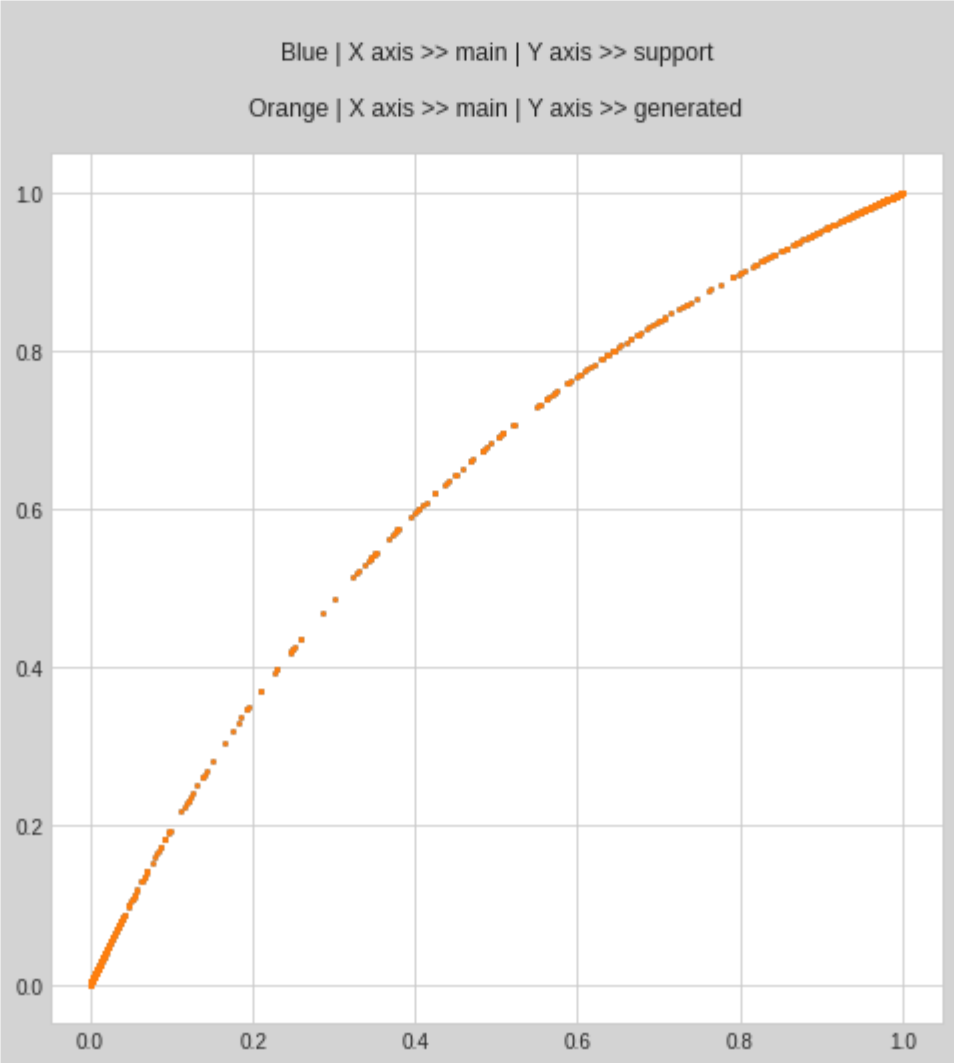
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [31]:

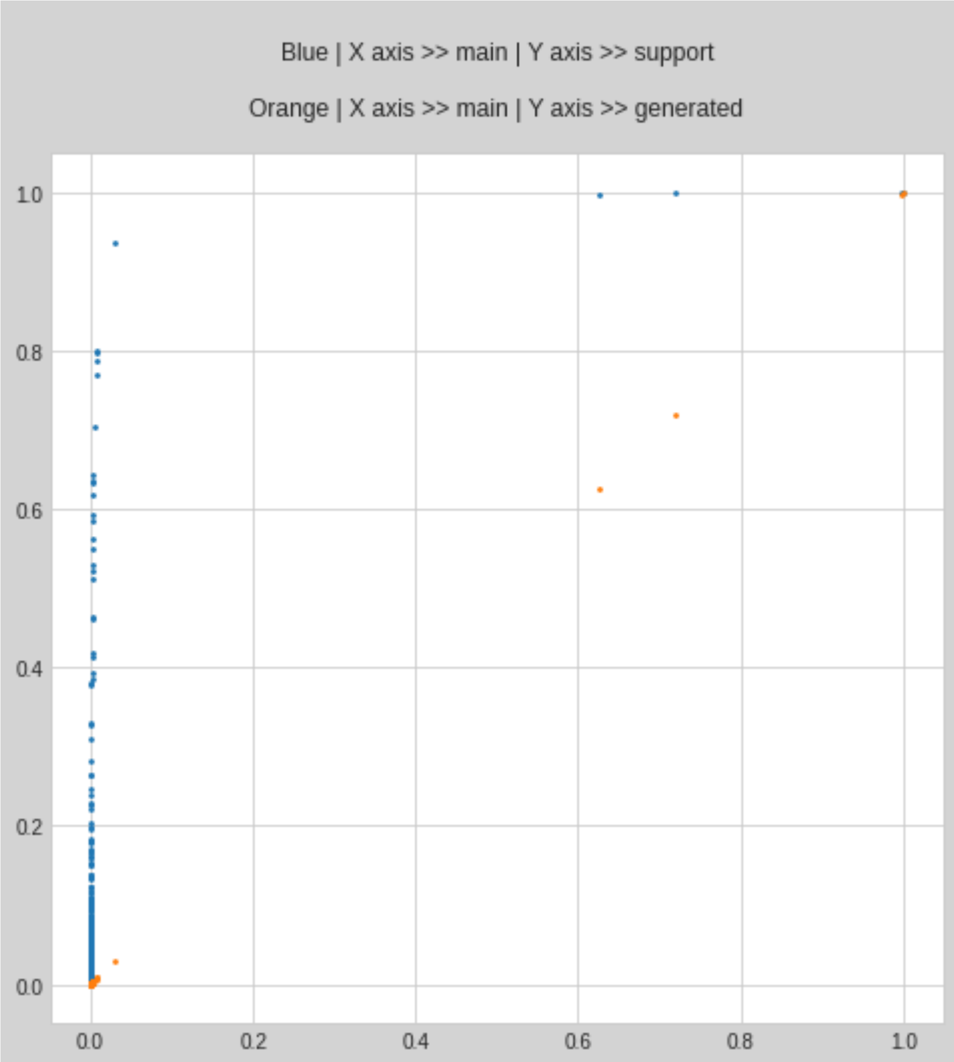
```
drawing1(sub968, sub961, d, 19)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [32]:

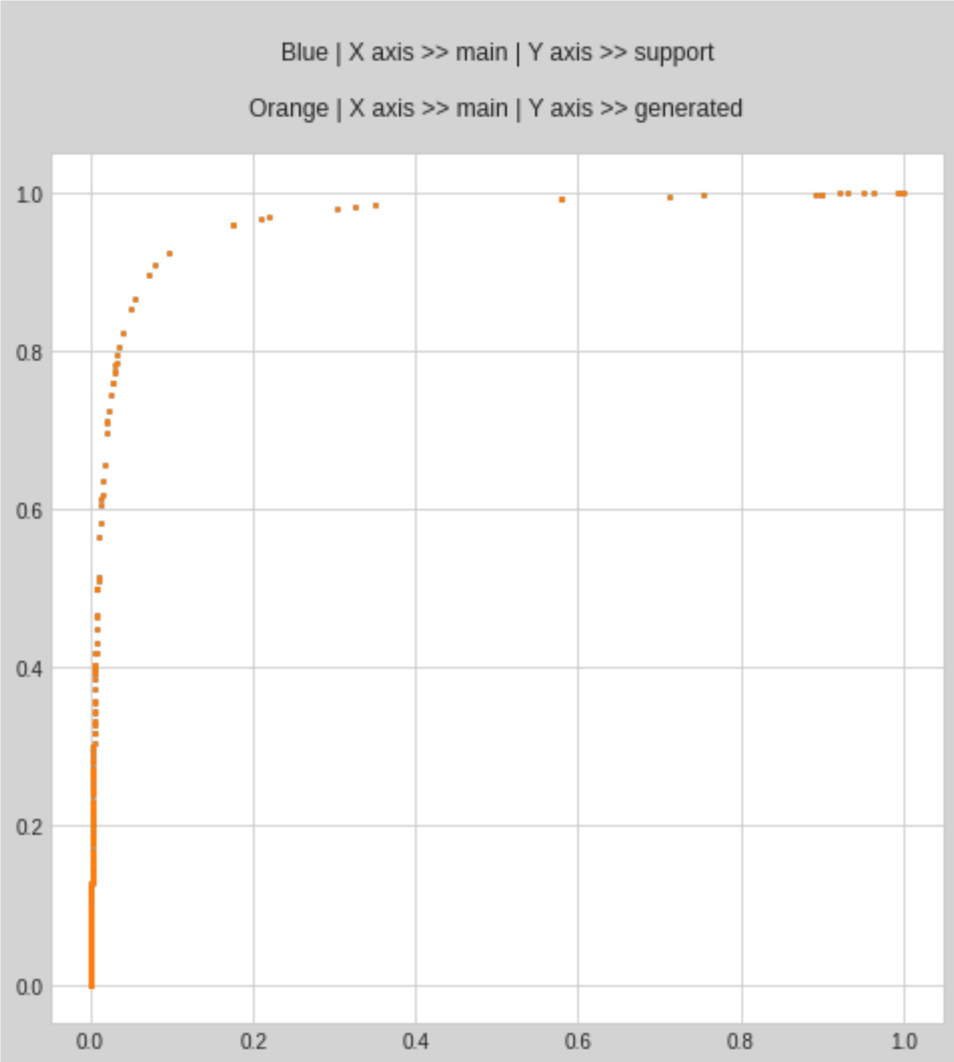
```
drawing1(sub968, sub961, d, 20)
```



In [33]:

```
drawing1(sub968, sub961, d, 21)
```

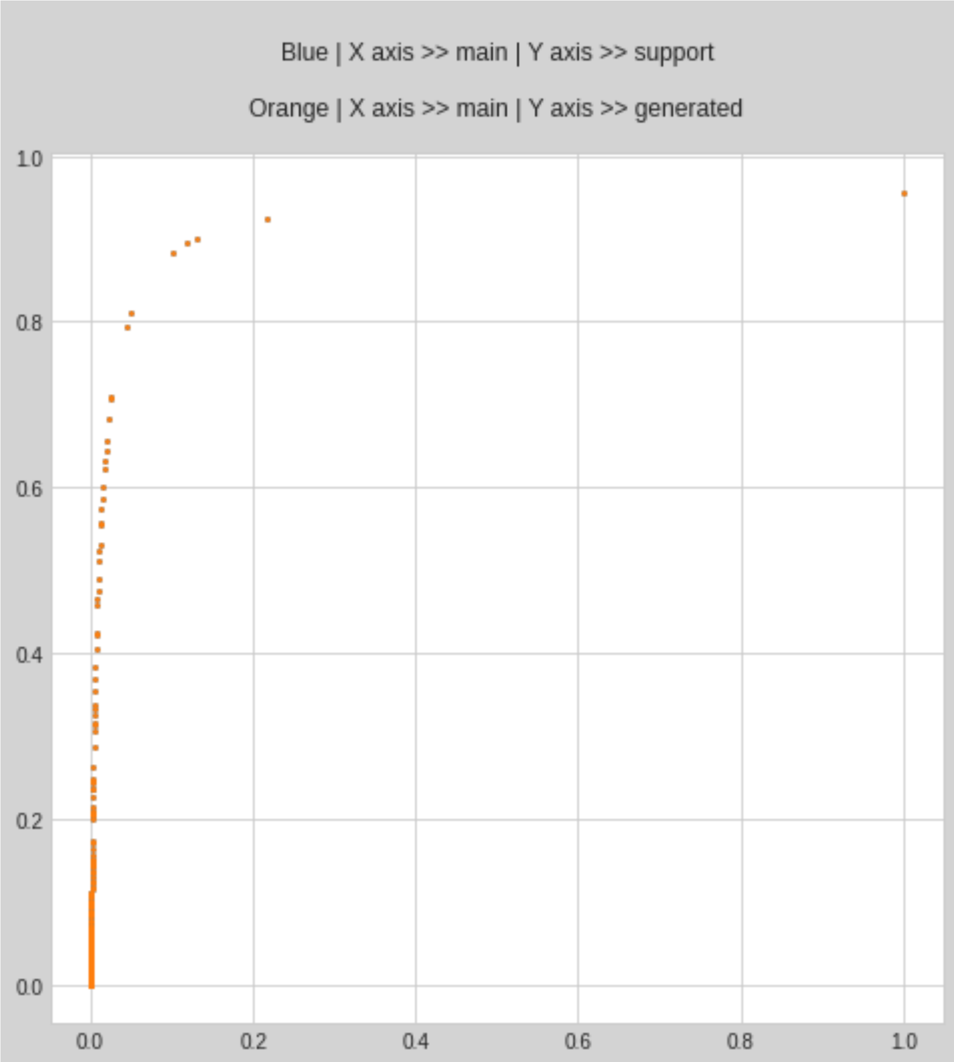
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [34]:

```
drawing1(sub968, sub961, d, 22)
```

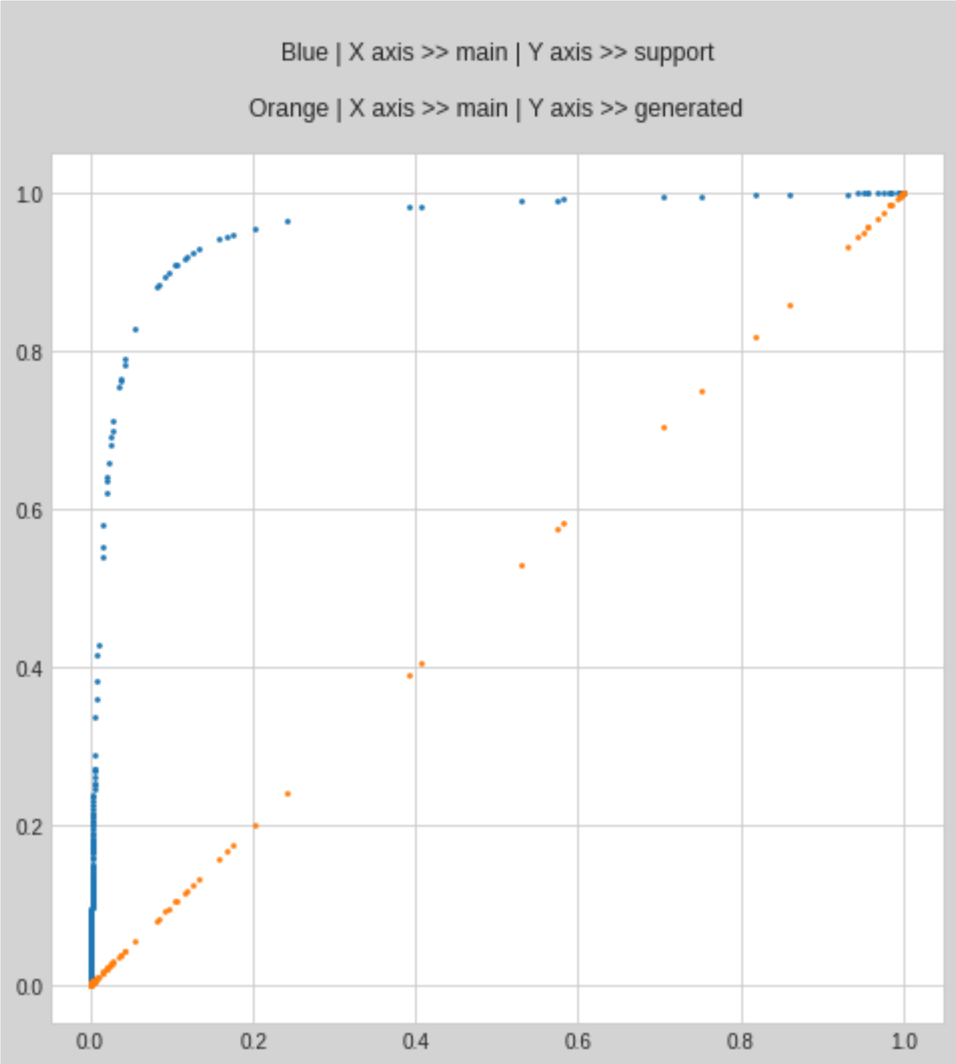
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [35]:

```
drawing1(sub968, sub961, d, 23)
```

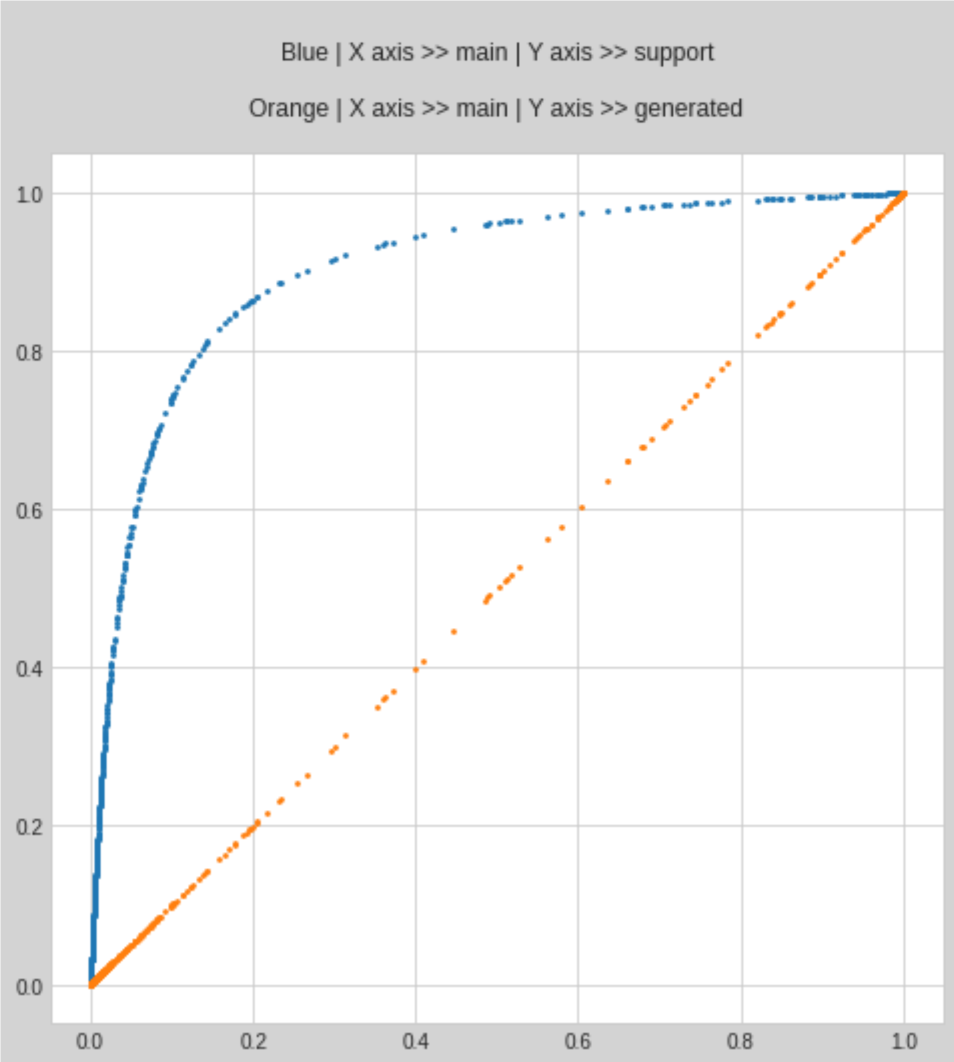
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [36]:

```
drawing1(sub968, sub961, d, 24)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Result

[(Score: 0.968) , (Score: 0.961)] >>> d

d : [(Private Score: 0.97915) , (Public Score: 0.97373)]

In [37]:

```
d.describe()
```

Out[37]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1992.000000	1.992000
mean	0.148069	0.255927	0.058836	0.975920	0.083605	0.049287	3.443965
std	0.324055	0.394491	0.221785	0.095970	0.221616	0.195465	5.238256
min	0.000001	0.000031	0.000000	0.000000	0.000003	0.000000	0.000000
25%	0.000264	0.001760	0.000010	0.997896	0.000638	0.000032	5.145849
50%	0.002151	0.015521	0.000070	0.999792	0.003190	0.000219	4.882412
75%	0.038402	0.411339	0.000732	0.999958	0.023652	0.001828	2.530490
max	0.999999	0.999999	1.000000	1.000000	0.999976	1.000000	1.000000

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

8 rows × 24 columns

In [38]:
```
# print(d.mean() , sub961.mean())

n1 = d.mean() + sub961.mean()

n1mean = n1.mean()

n2 = n1 / n1mean

n2
```

Out[38]:
```
s0     0.892402
s1     1.542458
s2     0.515516
s3     5.776221
s4     0.503880
s5     0.530050
s6     0.082759
s7     1.837864
s8     0.490991
s9     0.331072
s10    0.252530
s11    1.115764
s12    2.325548
s13    0.253618
s14    0.772764
s15    1.967539
s16    0.437821
s17    0.205713
s18    2.794711
s19    0.082145
s20    0.276296
s21    0.116289
s22    0.192735
s23    0.703314
dtype: float64
```

In [39]:
```
n3 = n2.copy()
for k in range(24):
    n3[k] = 0.95

# n3
```

Result

[(Private Score: 0.97975) , (Public Score: 0.97444)]

In [40]:

```
n4 = n3.copy()

n4[2]    = 0.80

n4[6]    = 0.80

n4[19]   = 0.70

n4[22]   = 0.80

n4[23]   = 0.80

n4
```

Out[40]:

```
s0     0.95
s1     0.95
s2     0.80
s3     0.95
s4     0.95
s5     0.95
s6     0.80
s7     0.95
s8     0.95
s9     0.95
s10    0.95
s11    0.95
s12    0.95
s13    0.95
s14    0.95
s15    0.95
s16    0.95
s17    0.95
s18    0.95
s19    0.70
s20    0.95
s21    0.95
s22    0.80
s23    0.80
dtype: float64
```

In [41]:

```
e = generate1(d, sub961, n4)
```

Result

[(Score: 0.973) , (Score: 0.961)] >>> e

e : [(Private Score: 0.98022) , (Public Score: 0.97490)]

In [42]:

```
e.describe()
```

Out[42]:

	s0	s1	s2	s3	s4	s5	
count	1992.000000	1992.000000	1.992000e+03	1992.000000	1992.000000	1.992000e+03	1.9920
[illegible]	[illegible]	[illegible]	54e-02	0.974168	0.083605	5.315334e-02	7.5589

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	s0	s1	s2	s3	s4	s5	
std	0.324055	0.394491	2.251883e-01	0.098811	0.221616	1.964352e-01	5.6354
min	0.000001	0.000031	1.379217e-07	0.001901	0.000003	1.763993e-07	4.6538
25%	0.000264	0.001760	9.586658e-05	0.997401	0.000638	1.063986e-04	6.9256
50%	0.002151	0.015521	6.602089e-04	0.999742	0.003190	7.149753e-04	6.5128
75%	0.038402	0.411339	6.751331e-03	0.999947	0.023652	5.682613e-03	3.3288
max	0.999999	0.999999	1.000000e+00	1.000000	0.999976	9.999998e-01	9.9999

8 rows × 24 columns

Submission

In [43]:

```
sub = e
sub.to_csv("submission.csv", index=False)

d.to_csv("submission1.csv", index=False)
e.to_csv("submission2.csv", index=False)

!ls
```

```
__notebook__.ipynb  submission.csv  submission1.csv  submission2.csv
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js