

Pattern Recognition – S p r i n g 2 0 2 3

Network Anomaly Detection

AHMED DUSUKI

6856

MANAR AMGAD

7113

NADA ELWAZANE

6876

Imports

```
In [ ]: !pip install kneed
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting kneed
  Downloading kneed-0.8.2-py3-none-any.whl (10 kB)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from kneed) (1.10.1)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.9/dist-packages (from kneed) (1.22.4)
Installing collected packages: kneed
Successfully installed kneed-0.8.2
```

```
In [ ]: import numpy as np
import jax.numpy as jnp
from jax import jit
import matplotlib.pyplot as plt
import pandas as pd
import math
import os
import re
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from kneed import KneeLocator
import plotly.express as px
from scipy.optimize import linear_sum_assignment
from sklearn.metrics.pairwise import (
    additive_chi2_kernel,
    chi2_kernel,
    cosine_similarity,
    laplacian_kernel,
    linear_kernel,
    polynomial_kernel,
    rbf_kernel,
    sigmoid_kernel,
)

SEED = 42
np.random.seed(SEED)
```

```
In [ ]: from google.colab import drive
```

```
drive.mount("/content/drive")
path = "/content/drive/My Drive/pattern/2/"
try:
    os.mkdir(path)
except OSError as error:
    print(error)
```

```
Mounted at /content/drive
[Errno 17] File exists: '/content/drive/My Drive/pattern/2/'
```

Loading the data

```
In [ ]: train_10_percent = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/kddcup.data_10_percent.gz",
    compression="gzip",
    header=None,
)
```

```
In [ ]: test = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/corrected.gz",
    compression="gzip",
    header=None,
)
```

```
In [ ]: cols = [
    "duration",
    "protocol_type",
    "service",
    "flag",
    "src_bytes",
    "dst_bytes",
    "land",
    "wrong_fragment",
    "urgent",
    "hot",
    "num_failed_logins",
```

```

    "logged_in",
    "num_compromised",
    "root_shell",
    "su_attempted",
    "num_root",
    "num_file_creations",
    "num_shells",
    "num_access_files",
    "num_outbound_cmds",
    "is_host_login",
    "is_guest_login",
    "count",
    "srv_count",
    "serror_rate",
    "srv_serror_rate",
    "rerror_rate",
    "srv_rerror_rate",
    "same_srv_rate",
    "diff_srv_rate",
    "srv_diff_host_rate",
    "dst_host_count",
    "dst_host_srv_count",
    "dst_host_same_srv_rate",
    "dst_host_diff_srv_rate",
    "dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate",
    "dst_host_serror_rate",
    "dst_host_srv_serror_rate",
    "dst_host_rerror_rate",
    "dst_host_srv_rerror_rate",
    "outcome",
]
test.columns = cols
train_10_percent.columns = cols

```

Splitting the data

Splitting by 0.02

```

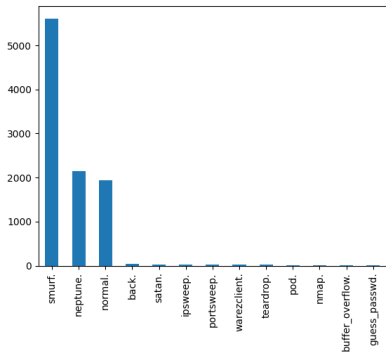
In [ ]: X_train02, X_test02, y_train02, y_test02 = train_test_split(
    train_10_percent.iloc[:, :-1],
    train_10_percent.iloc[:, -1],
    train_size=0.02,
    random_state=SEED,
    stratify=train_10_percent.iloc[:, -1],
)

```

```

In [ ]: y_train02.value_counts().plot(kind="bar")
plt.show()

```



```
In [ ]: X_train02.head()

Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_count  dst_host_srv_count  dst_host_
215576      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
260869      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
197950      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
166006      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
110009      0           tcp  private  SO         0         0         0         0         0         0  ...          255           5

5 rows x 41 columns
```

```

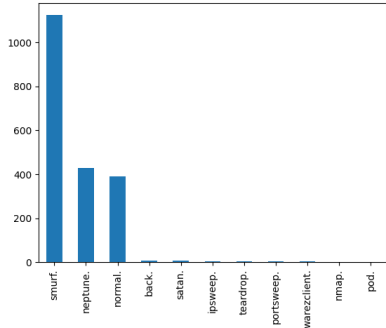
In [ ]: X_train02.to_numpy()

Out[ ]:
array([[0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       ...,
       [0, 'tcp', 'private', ..., 0.0, 1.0, 1.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0]], dtype=object)
```

Splitting by 0.025

```
In [ ]: X_train025, X_test025, y_train025, y_test025 = train_test_split(
    train_10_percent.iloc[:, :-1],
    train_10_percent.iloc[:, -1],
    train_size=0.025,
    random_state=SEED,
    stratify=train_10_percent.iloc[:, -1],
)

In [ ]: y_train025.value_counts().plot(kind="bar")
plt.show()
```



```
In [ ]: X_train004.head()

Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_count  dst_host_srv_count  dst_host_
259019      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
324041      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
249235      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
41387       0           tcp  http   REJ         0         0         0         0         0         0  ...           2          255
405574      0          icmp  ecr_i   SF       520         0         0         0         0         0  ...          255          255

5 rows x 41 columns
```

```

In [ ]: X_train004.to_numpy()

Out[ ]:
array([[0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       ...,
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'tcp', 'private', ..., 1.0, 0.0, 0.0],
       [0, 'tcp', 'http', ..., 0.0, 0.0, 0.0]], dtype=object)
```

Investigating the dataset

```
In [ ]: train_10_percent.head()

Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_host_
0          0           tcp  http   SF      181     5450         0         0         0         0  ...           9          1.0
1          0           tcp  http   SF      239     486         0         0         0         0  ...          19          1.0
2          0           tcp  http   SF      235    1337         0         0         0         0  ...           29          1.0
3          0           tcp  http   SF      219    1337         0         0         0         0  ...           39          1.0
4          0           tcp  http   SF      217     2032         0         0         0         0  ...           49          1.0

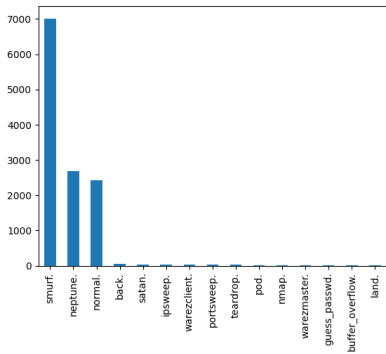
5 rows x 42 columns
```

```

In [ ]: test.head()

Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_host_
0          0           tcp  http   SF      181     5450         0         0         0         0  ...           9          1.0
1          0           tcp  http   SF      239     486         0         0         0         0  ...          19          1.0
2          0           tcp  http   SF      235    1337         0         0         0         0  ...           29          1.0
3          0           tcp  http   SF      219    1337         0         0         0         0  ...           39          1.0
4          0           tcp  http   SF      217     2032         0         0         0         0  ...           49          1.0

5 rows x 42 columns
```



```
In [ ]: X_train025.head()

Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_count  dst_host_srv_count  dst_host_
154374      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255
78554       0           tcp  http   SF      248     293         0         0         0         0  ...          19          236
39563       0           tcp  http   SF      374    10063         0         0         0         0  ...           66          255
19926       0           tcp  http   SF      207     377         0         0         0         0  ...           10          255
300920      0          icmp  ecr_i   SF      1032         0         0         0         0         0  ...          255          255

5 rows x 41 columns
```

```

In [ ]: X_train025.to_numpy()

Out[ ]:
array([[0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'tcp', 'http', ..., 0.0, 0.0, 0.0],
       [0, 'tcp', 'http', ..., 0.0, 0.0, 0.0],
       ...,
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0],
       [0, 'icmp', 'ecr_i', ..., 0.0, 0.0, 0.0]], dtype=object)
```

splitting by 0.004

```
In [ ]: X_train004, X_test004, y_train004 = train_test_split(
    train_10_percent.iloc[:, :-1],
    train_10_percent.iloc[:, -1],
    train_size=0.004,
    random_state=SEED,
    stratify=train_10_percent.iloc[:, -1],
)

In [ ]: y_train004.value_counts().plot(kind="bar")
plt.show()
```

```
Out[ ]:
  duration  protocol type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_hos
0          0           udp  private  SF      105     146         0         0         0         0  ...          254          1.0
1          0           udp  private  SF      105     146         0         0         0         0  ...          254          1.0
2          0           udp  private  SF      105     146         0         0         0         0  ...          254          1.0
3          0           udp  private  SF      105     146         0         0         0         0  ...          254          1.0
4          0           udp  private  SF      105     146         0         0         0         0  ...          254          1.0

5 rows x 42 columns
```

```

In [ ]: train_10_percent.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494021 entries, 0 to 494020
Data columns (total 42 columns):
 #   Column              Non-Null Count  Dtype
---  --
 0   duration            494021 non-null  int64
 1   protocol_type       494021 non-null  object
 2   service             494021 non-null  object
 3   flag               494021 non-null  object
 4   src_bytes          494021 non-null  int64
 5   dst_bytes          494021 non-null  int64
 6   land              494021 non-null  int64
 7   wrong_fragment     494021 non-null  int64
 8   urgent            494021 non-null  int64
 9   hot               494021 non-null  int64
10  num_failed_logins  494021 non-null  int64
11  logged_in         494021 non-null  int64
12  num_compromised   494021 non-null  int64
13  root_shell        494021 non-null  int64
14  su_attempted      494021 non-null  int64
15  num_root          494021 non-null  int64
16  num_file_creations 494021 non-null  int64
17  num_shells        494021 non-null  int64
18  num_access_files  494021 non-null  int64
19  num_outbound_cmds 494021 non-null  int64
20  is_host_login     494021 non-null  int64
21  is_guest_login    494021 non-null  int64
22  count             494021 non-null  int64
23  srv_count         494021 non-null  int64
24  serror_rate       494021 non-null  float64
25  srv_error_rate    494021 non-null  float64
26  rerror_rate       494021 non-null  float64
27  srv_error_rate    494021 non-null  float64
28  same_srv_rate     494021 non-null  float64
29  diff_srv_rate     494021 non-null  float64
30  srv_diff_host_rate 494021 non-null  float64
31  dst_host_count    494021 non-null  int64
32  dst_host_srv_count 494021 non-null  int64
33  dst_host_same_srv_rate 494021 non-null  float64
34  dst_host_diff_srv_rate 494021 non-null  float64
35  dst_host_same_src_port_rate 494021 non-null  float64
36  dst_host_srv_diff_host_rate 494021 non-null  float64
37  dst_host_serror_rate 494021 non-null  float64
38  dst_host_srv_error_rate 494021 non-null  float64
39  dst_host_rerror_rate 494021 non-null  float64
40  dst_host_srv_error_rate 494021 non-null  float64
41  outcome           494021 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 158.3+ MB
```

Comment

Elements with object dtype are categorical

```
In [ ]: train_10_percent["protocol_type"].value_counts()

Out[ ]:
icmp      283682
tcp       198065
udp        28354
Name: protocol_type, dtype: int64

In [ ]: train_10_percent["service"].value_counts()
```

19.0	873
11.0	121
3.0	59
9.0	49
17.0	41
16.0	23
13.0	20
7.0	16

```

5.0      5
8.0      3
10.0     2
1.0      1
4.0      1
dtype: int64
Completed iteration 7
0.0    447618

```

[illegible]

[illegible]

```

5.000000e-01 1.000000e-02 5.000000e-03 0.000000e-04 0.000000e-05
6.000000e-06 1.500000e-06 8.000000e-07 0.000000e-08 5.000000e-09 0.000000e-10
7.000000e-11 4.000000e-12 2.000000e-12 0.000000e-13 0.000000e-14 0.000000e-15
8.000000e-16 0.000000e-17 0.000000e-18 0.000000e-19 0.000000e-20 0.000000e-21
9.000000e-22 0.000000e-23 0.000000e-24 0.000000e-25 0.000000e-26 0.000000e-27
0.000000e-28 0.000000e-29 0.000000e-30 0.000000e-31 0.000000e-32 0.000000e-33
0.000000e-34 0.000000e-35 0.000000e-36 0.000000e-37 0.000000e-38 0.000000e-39
0.000000e-40 0.000000e-41 0.000000e-42 0.000000e-43 0.000000e-44 0.000000e-45
0.000000e-46 0.000000e-47 0.000000e-48 0.000000e-49 0.000000e-50 0.000000e-51
0.000000e-52 0.000000e-53 0.000000e-54 0.000000e-55 0.000000e-56 0.000000e-57
0.000000e-58 0.000000e-59 0.000000e-60 0.000000e-61 0.000000e-62 0.000000e-63
0.000000e-64 0.000000e-65 0.000000e-66 0.000000e-67 0.000000e-68 0.000000e-69
0.000000e-70 0.000000e-71 0.000000e-72 0.000000e-73 0.000000e-74 0.000000e-75
0.000000e-76 0.000000e-77 0.000000e-78 0.000000e-79 0.000000e-80 0.000000e-81
0.000000e-82 0.000000e-83 0.000000e-84 0.000000e-85 0.000000e-86 0.000000e-87
0.000000e-88 0.000000e-89 0.000000e-90 0.000000e-91 0.000000e-92 0.000000e-93
0.000000e-94 0.000000e-95 0.000000e-96 0.000000e-97 0.000000e-98 0.000000e-99
Ctrl+11 [1.00243200e-02 4.03382200e-04 4.034847e-04 0.000000e-05 0.000000e-06
0.000000e-07 2.598780e-04 5.517780e-04 1.1294370e-03
5.187785e-04 1.2090350e-03 7.651170e-04 7.700042e-04
8.011244e-04 4.127282e-04 0.000000e-05 0.159721e-02
0.000000e-06 0.000000e-07 0.000000e-08 4.610270e-03
7.402375e-04 5087030e-04 4.023827e-03 1.2020180e-03
7.731384e-03 1.994370e-02 1.1297070e-03 1.000000e-01
1.710877e-02 4.438118e-02
7.186157e-02 4.641768e-02 2.089955e-03 4.610270e-03
0.000000e-06 0.000000e-07 0.000000e-08 0.000000e-09 0.000000e-10 0.000000e-11
0.000000e-12 0.000000e-13 0.000000e-14 0.000000e-15 0.000000e-16 0.000000e-17
0.000000e-18 0.000000e-19 0.000000e-20 0.000000e-21 0.000000e-22 0.000000e-23
0.000000e-24 0.000000e-25 0.000000e-26 0.000000e-27 0.000000e-28 0.000000e-29
0.000000e-30 0.000000e-31 0.000000e-32 0.000000e-33 0.000000e-34 0.000000e-35
0.000000e-36 0.000000e-37 0.000000e-38 0.000000e-39 0.000000e-40 0.000000e-41
0.000000e-42 0.000000e-43 0.000000e-44 0.000000e-45 0.000000e-46 0.000000e-47
0.000000e-48 0.000000e-49 0.000000e-50 0.000000e-51 0.000000e-52 0.000000e-53
0.000000e-54 0.000000e-55 0.000000e-56 0.000000e-57 0.000000e-58 0.000000e-59
0.000000e-60 0.000000e-61 0.000000e-62 0.000000e-63 0.000000e-64 0.000000e-65
0.000000e-66 0.000000e-67 0.000000e-68 0.000000e-69 0.000000e-70 0.000000e-71
0.000000e-72 0.000000e-73 0.000000e-74 0.000000e-75 0.000000e-76 0.000000e-77
0.000000e-78 0.000000e-79 0.000000e-80 0.000000e-81 0.000000e-82 0.000000e-83
0.000000e-84 0.000000e-85 0.000000e-86 0.000000e-87 0.000000e-88 0.000000e-89
0.000000e-90 0.000000e-91 0.000000e-92 0.000000e-93 0.000000e-94 0.000000e-95
0.000000e-96 0.000000e-97 0.000000e-98 0.000000e-99
Ctrl+11 [1.00000000e-02 7.654444e-04 4.6279531e-05 0.000000e-06 0.000000e-07
0.000000e-08 0.000000e-09 0.000000e-10 0.000000e-11 0.000000e-12 0.000000e-13
0.000000e-14 0.000000e-15 0.000000e-16 0.000000e-17 0.000000e-18 0.000000e-19
0.000000e-20 0.000000e-21 0.000000e-22 0.000000e-23 0.000000e-24 0.000000e-25
0.000000e-26 0.000000e-27 0.000000e-28 0.000000e-29 0.000000e-30 0.000000e-31
0.000000e-32 0.000000e-33 0.000000e-34 0.000000e-35 0.000000e-36 0.000000e-37
0.000000e-38 0.000000e-39 0.000000e-40 0.000000e-41 0.000000e-42 0.000000e-43
0.000000e-44 0.000000e-45 0.000000e-46 0.000000e-47 0.000000e-48 0.000000e-49
0.000000e-50 0.000000e-51 0.000000e-52 0.000000e-53 0.000000e-54 0.000000e-55
0.000000e-56 0.000000e-57 0.000000e-58 0.000000e-59 0.000000e-60 0.000000e-61
0.000000e-62 0.000000e-63 0.000000e-64 0.000000e-65 0.000000e-66 0.000000e-67
0.000000e-68 0.000000e-69 0.000000e-70 0.000000e-71 0.000000e-72 0.000000e-73
0.000000e-74 0.000000e-75 0.000000e-76 0.000000e-77 0.000000e-78 0.000000e-79
0.000000e-80 0.000000e-81 0.000000e-82 0.000000e-83 0.000000e-84 0.000000e-85
0.00
```

```

5.0 1
6.0 3
10.0 2
1.0 1
4.0 1
dtype: int64
Completed iteration 5
0.0 400008
11.0 20278
21.0 6422
18.0 2081
7.0 2114
14.0 1283
11.0 450
11.0 97
5.0 50
0.0 84
17.0 41
10.0 23
21.0 21
22.0 21
2.0 16
4.0 10
12.0 9
20.0 8
5.0 5
8.0 3
10.0 2
1.0 1
4.0 1
dtype: int64
Completed iteration 6
0.0 400008
11.0 20918
21.0 6572
18.0 2082
7.0 2114
14.0 1281
11.0 451
11.0 96
5.0 50
0.0 87
17.0 42
10.0 23
21.0 20
2.0 16
6.0 10
12.0 9
20.0 8
5.0 5
8.0 3
10.0 2
1.0 1
4.0 1
dtype: int64
Completed iteration 7
0.0 400010
11.0 21055
21.0 6720
18.0 2090
7.0 2114
14.0 1311
11.0 160
11.0 95
17.0 42
5.0 40
10.0 27
22.0 21
13.0 20
2.0 16
4.0 10
12.0 9
20.0 8
5.0 5
8.0 3
10.0 2
1.0 1
4.0 1
dtype: int64
Completed iteration 8
0.0 420005
11.0 17006
21.0 6807
18.0 2028

```

[illegible]

12.0	1	12.0	1	7.0	2274	8.0	1	29.0	4
20.0	8	12.0	8	14.0	1278	8.0	1	29.0	3
20.0	8	24.0	243	24.0	1213	10.0	2	27.0	3
6.0	8	30.0	303	30.0	1051	10.0	2	10.0	2
5.0	5			10.0	461	10.0	1	4.0	1
22.0	4			11.0	106	10.0	1	1.0	1
29.0	4			3.0	59				
8.0	1			3.0	47				
25.0	3			17.0	40				
27.0	3			18.0	31				
16.0	2			16.0	21				
4.0	1			23.0	17				
2				1.0	1				
dtype: int64									
Completed iteration 2									
26.0	289719			13.0	13			7.0	2274
0.0	171256			12.0	9			14.0	1304
15.0	17085			1.0	5			24.0	1208
22.0	1061			22.0	4			30.0	1054
18.0	2277			25.0	4			10.0	450
7.0	2273			20.0	4			9.0	55
14.0	1232			8.0	3			3.0	55
24.0	1220			27.0	3			21.0	46
30.0	1060			20.0	2			17.0	40
11.0	651			4.0	2			28.0	31
11.0	15			1.0	1			10.0	23
3.0	39			2.0	16			21.0	17
9.0	39			13.0	13			10.0	10
17.0	39			12.0	9			12.0	9
30.0	31			8.0	5			20.0	6
16.0	21			11.0	10879			5.0	5
23.0	17			21.0	4381			21.0	4
2.0	16			18.0	2712			20.0	4
13.0	11			7.0	2273			29.0	4
12.0	9			14.0	1274			8.0	3
6.0	9			24.0	1203			27.0	3
20.0	8			30.0	1051			10.0	2
5.0	1			11.0	462			4.0	1
25.0	4			11.0	36			4.0	1
8.0	3			3.0	59				
17.0	3			8.0	9				
10.0	2			17.0	40				
4.0	2			20.0	31				
2.0	1			15.0	23				
dtype: int64									
Completed iteration 3									
0.0	170870			10.0	10			26.0	289748
11.0	171241			12.0	9			0.0	171256
21.0	10617			20.0	8			15.0	18092
18.0	2084			1.0	5			21.0	6526
7.0	2274			22.0	4			10.0	2500
14.0	1205			25.0	4			7.0	2274
34.0	1129			8.0	3			14.0	1312
30.0	1030			17.0	4			24.0	1208
19.0	457			30.0	10			30.0	1051
11.0	121			11.0	36			10.0	451
3.0	39			3.0	59			11.0	46
9.0	47			8.0	9				

[illegible][illegible]

[illegible][illegible][illegible][illegible]

[illegible][illegible][illegible]

[illegible]

Spectra

Function

label encoded

1.0	4
11.0	4
8.0	1
10.0	1
11.0	1
4.0	2
17.0	2
9.0	2
22.0	2
20.0	2
1.0	3
12.0	3
1.0	3
15.0	3
1.0	3
16.0	3
16.0	3
6.0	3
otype: int	
Completed	
2.0	56
13.0	7
6.0	2
1.0	2
15.0	4
21.0	4
8.0	1
1.0	1
10.0	1
11.0	1
10.0	1
4.0	2
9.0	2
17.0	2
22.0	2
20.0	2
1.0	3
12.0	3
1.0	3
15.0	3
1.0	3
16.0	3
16.0	3
6.0	3
otype: int	
Completed	
2.0	56
13.0	7
6.0	2
1.0	2
15.0	4
21.0	4
8.0	1
1.0	1
10.0	1
11.0	1
10.0	1
4.0	2
9.0	2
17.0	2
22.0	2
20.0	2
1.0	3
12.0	3
1.0	3
15.0	3
1.0	3
16.0	3
16.0	3
6.0	3
otype: int	
Completed	
2.0	56
13.0	7
6.0	2
1.0	2
15.0	4
21.0	4
8.0	1
1.0	1
10.0	1
11.0	1
10.0	1
4.0	2
9.0	2
17.0	2
22.0	2
20.0	2
1.0	3
12.0	3
1.0	3
15.0	3
1.0	3
16.0	3
16.0	3
6.0	3

17.0	2
20.0	2
54.0	2
5.0	2
7.0	3
5.0	3
18.0	3
10.0	3
12.0	3
16.0	3
Hypert: int	
Completed	
1.0	56
13.0	7
8.0	4
1.0	4
15.0	4
8.0	4
1.0	5
22.0	5
28.0	5
22.0	5
8.0	5
11.0	5
17.0	5
20.0	5
4.0	5
14.0	5
7.0	5
5.0	5
18.0	5
10.0	5
12.0	5
16.0	5
Hypert: int	
Completed	
1.0	56
13.0	7
8.0	4
1.0	4
15.0	4
8.0	4
1.0	5
22.0	5
28.0	5
22.0	5
8.0	5
11.0	5
17.0	5
20.0	5
4.0	5
14.0	5
7.0	5
5.0	5
18.0	5
10.0	5
12.0	5
16.0	5
Hypert: int	
Completed	
1.0	56
13.0	7
8.0	4
1.0	4
15.0	4
8.0	4
1.0	5
22.0	5
28.0	5
22.0	5
8.0	5
11.0	5
17.0	5
20.0	5
4.0	5
14.0	5
7.0	5
5.0	5
18.0	5
10.0	5
12.0	5
16.0	5
Hypert: int	

7.0	186
12.0	179
1.0	168
18.0	165
10.0	150
16.0	91
6.0	88
dlqps lena	
Completed iteration 7	
2.0	567
11.0	775
8.0	767
1.0	681
3.0	424
21.0	403
1.0	390
8.0	366
20.0	350
22.0	307
4.0	283
9.0	279
15.0	261
17.0	252
20.0	217
16.0	206
1.0	200
20.0	186
5.0	178
10.0	170
12.0	154
10.0	139
6.0	123
16.0	96
dlqps lena	
Completed iteration 8	
2.0	567
11.0	752
1.0	659
15.0	630
1.0	595
8.0	480
21.0	393
10.0	329
22.0	312
4.0	279
11.0	259
17.0	252
20.0	235
16.0	207
7.0	186
18.0	161
6.0	146
10.0	137
12.0	122
16.0	96
dlqps lena	
Completed iteration 9	
2.0	566
11.0	759
8.0	693
11.0	616
3.0	416
21.0	382
1.0	392
21.0	382
20.0	326
22.0	305
9.0	276
4.0	271
11.0	259
17.0	248
20.0	230
14.0	201
7.0	186
6.0	177
10.0	159
10.0	137
12.0	131
16.0	91
dlqps lena	
Completed iteration 10	
2.0	5670
11.0	765

Completed iteration 17	
2.8	5087
8.0	768
8.0	791
3.0	510
15.0	867
8.0	454
1.0	484
22.0	319
18.0	293
9.0	292
6.0	309
11.0	253
17.0	245
28.0	235
14.0	228
6.0	200
7.0	185
5.0	172
18.0	158
12.0	117
1.0	90
dhpex index	
Completed iteration 18	
2.8	5087
15.0	793
8.0	592
3.0	505
15.0	468
8.0	454
1.0	486
22.0	318
9.0	309
18.0	293
11.0	253
6.0	289
17.0	247
28.0	235
14.0	212
6.0	228
7.0	185
5.0	172
18.0	149
12.0	139
12.0	118
16.0	100
dhpex index	
Completed iteration 19	
2.8	5087
15.0	797
8.0	589
1.0	562
15.0	468
8.0	454
1.0	487
22.0	316
9.0	306
18.0	290
21.0	288
6.0	276
11.0	212
28.0	235
14.0	211
6.0	219
7.0	186
5.0	172
18.0	147
13.0	138
12.0	129
16.0	98
dhpex index	
Completed iteration 20	
2.8	5087
15.0	796
8.0	583
1.0	585
15.0	468
8.0	454
1.0	487
22.0	316
9.0	307
18.0	285
21.0	285

```

signature: [9.508295d
9.508297d-01 9.508431d-
9.508787d-01 9.508929d-
Finished writing h-memories

6.0 7623
8.0 1129
9.0 849
4.0 632
11.0 530
13.0 430
18.0 415
7.0 265
12.0 259
5.0 238
8.0 238
12.0 187
5.0 151
10.0 22
14.0 28
dtype: int64
Completed iteration 1
6.0 6321
4.0 5885
9.0 708
11.0 798
12.0 677
13.0 555
18.0 435
7.0 335
10.0 431
2.0 293
1.0 272
8.0 284
3.0 85
14.0 75
3.0 66
dtype: int64
Completed iteration 2
6.0 6695
4.0 961
9.0 929
8.0 744
11.0 626
13.0 545
12.0 439
18.0 433
7.0 385
2.0 352
1.0 335
8.0 256
14.0 147
3.0 96
5.0 83
dtype: int64
Completed iteration 3
6.0 5062
4.0 898
9.0 864
8.0 776
11.0 575
13.0 580
12.0 485
7.0 474
2.0 426
18.0 406
1.0 329
12.0 305
14.0 227
3.0 185
5.0 85
dtype: int64
Completed iteration 4
6.0 5052
4.0 850
9.0 889
8.0 785
11.0 715
7.0 532
12.0 586
18.0 483
2.0 483
18.0 395
1.0 312
14.0 312
8.0 264
3.0 288
dtype: int64

```

[illegible]

12.0	151	12.0	680	12.0	109
16.0	150	4.0	554	4.0	158
11.0	154	10.0	435	10.0	183
9.0	139	1.0	786	1.0	158
17.0	131	5.0	344	5.0	155
21.0	131	1.0	344	1.0	151
14.0	139	13.0	292	13.0	158
dtype: int64		22.0	247	9.0	148
Completed iteration 7		18.0	247	18.0	148
8.0	1239	20.0	239	17.0	121
12.0	612	16.0	237	dtype: int64	
4.0	555	15.0	280	Completed iteration 14	
16.0	428	1.0	175	8.0	1517
3.0	396	10.0	147	11.0	687
5.0	342	21.0	151	4.0	549
4.0	341	11.0	147	10.0	611
13.0	289	9.0	143	3.0	400
22.0	273	16.0	143	6.0	346
2.0	247	14.0	137	5.0	346
20.0	228	17.0	126	11.0	287
20.0	228	dtype: int64		15.0	287
15.0	238	Completed iteration 11		22.0	275
16.0	288	6.0	511	18.0	219
1.0	382	8.0	1623	20.0	219
13.0	342	11.0	680	10.0	157
16.0	354	4.0	551	14.0	195
13.0	351	10.0	456	7.0	158
8.0	342	3.0	399	10.0	168
21.0	334	1.0	345	3.0	335
17.0	327	5.0	342	11.0	155
16.0	321	11.0	293	21.0	151
dtype: int64		22.0	246	9.0	148
Completed iteration 8		2.0	246	16.0	148
8.0	1237	10.0	229	17.0	121
8.0	1239	18.0	238	dtype: int64	
12.0	612	16.0	236	Completed iteration 10	
4.0	554	7.0	280	8.0	1525
16.0	424	1.0	168	8.0	1528
3.0	397	10.0	168	12.0	688
6.0	342	11.0	158	4.0	549
5.0	338	21.0	150	10.0	611
13.0	289	9.0	147	3.0	400
22.0	272	16.0	147	6.0	344
2.0	246	14.0	141	5.0	342
20.0	228	17.0	125	13.0	300
20.0	228	dtype: int64		15.0	287
15.0	232	Completed iteration 12		22.0	276
16.0	289	6.0	528	18.0	219
1.0	380	8.0	1623	10.0	157
13.0	342	11.0	688	14.0	198
16.0	351	4.0	558	7.0	158
11.0	348	10.0	436	15.0	196
9.0	344	1.0	398	10.0	168
21.0	338	6.0	345	11.0	157
17.0	327	1.0	348	3.0	334
14.0	325	5.0	344	21.0	149
dtype: int64		11.0	247	9.0	148
Completed iteration 9		22.0	241	10.0	161
8.0	1221	20.0	235	17.0	121
8.0	1223	16.0	243	dtype: int64	
12.0	638	7.0	289	Completed iteration 16	
4.0	548	1.0	169	8.0	1524
16.0	425	14.0	167	8.0	1619
3.0	396	10.0	167	12.0	688
5.0	344	1.0	164	4.0	548
4.0	344	11.0	168	10.0	611
13.0	289	9.0	148	3.0	399
22.0	278	16.0	148	6.0	342
2.0	246	15.0	144	6.0	342
20.0	228	17.0	124	13.0	312
20.0	228	dtype: int64		15.0	288
15.0	235	Completed iteration 13		22.0	272
16.0	254	6.0	528	18.0	219
7.0	211	8.0	1613	10.0	151
13.0	340	11.0	687	14.0	198
16.0	347	4.0	549	7.0	158
16.0	347	10.0	429	15.0</	

[illegible]

12.0	181
12.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
2.0	3226
17.0	2510
4.0	1136
3.0	905
3.0	699
22.0	440
8.0	433
11.0	281
20.0	264
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	1136
1.0	985
1.0	985
8.0	540
2.0	3226
0.0	426
2.0	264
11.0	282
11.0	282
7.0	250
6.0	261
9.0	287
21.0	136
18.0	137
18.0	75
14.0	65
14.0	15
19.0	41
19.0	5
dtype: int64	
Completed iteration	
17.0	2500
4.0	

[illegible][illegible][illegible][illegible]

```

28.0  318
3.0  312
7.0  294
22.0  273
21.0  248
13.0  234
1.0  228
12.0  217
6.0  208
5.0  206
dtype: float64
Completed iteration 14
0.0  1611
10.0  1128
15.0  1057
14.0  1047
2.0  415
0.0  415
10.0  407
11.0  385
17.0  358
16.0  358
4.0  345
8.0  337
2.0  317
20.0  314
10.0  309
22.0  273
21.0  248
12.0  231
3.0  210
13.0  223
6.0  209
5.0  206
dtype: float64
Completed iteration 15
0.0  1541
10.0  1119
15.0  1041
2.0  441
9.0  424
22.0  407
11.0  386
17.0  361
10.0  345
6.0  344
3.0  326
8.0  312
12.0  309
20.0  300
3.0  273
21.0  237
13.0  231
3.0  221
11.0  218
5.0  206
dtype: float64
Completed iteration 16
0.0  1541
10.0  1119
15.0  1041
2.0  440
9.0  420
22.0  407
11.0  387
17.0  362
6.0  341
10.0  339
3.0  314
8.0  313
7.0  314
22.0  311
21.0  276
12.0  273
21.0  235
3.0  214
13.0  215
6.0  205
5.0  205
dtype: float64

```

[illegible]

```

dtype: uint64
Completed iteration 10
10.0 3470
8.0 2848
14.0 630
2.0 633
0.0 630
11.0 576
5.0 587
5.0 467
6.0 467
5.0 387
5.0 362
7.0 365
11.0 298
4.0 284
10.0 264
dtype: uint64
Completed iteration 11
10.0 3470
8.0 2848
14.0 630
2.0 633
0.0 630
11.0 576
12.0 566
5.0 587
6.0 477
5.0 389
3.0 366
7.0 293
11.0 291
4.0 287
10.0 263
dtype: uint64
Completed iteration 12
10.0 3470
8.0 2848
14.0 640
2.0 640
8.0 620
11.0 578
12.0 564
5.0 525
6.0 463
5.0 389
3.0 361
7.0 294
11.0 293
4.0 285
10.0 263
dtype: uint64
Completed iteration 13
10.0 3470
8.0 2848
14.0 676
2.0 643
8.0 638
11.0 582
12.0 558
5.0 533
6.0 468
5.0 396
3.0 366
7.0 286
11.0 290
4.0 257
dtype: uint64
Completed iteration 14
10.0 3482
8.0 2943
14.0 677
2.0 647
8.0 622
11.0 586
12.0 551
5.0 541
6.0 469
3.0 361
11.0 298
7.0 279
10.0 268
4.0 246
dtype: uint64

```

[illegible]

```

    944
    .      897
    1.0    543
    2.0    499
    .      264
    3.0    209
    .      198
    4.0    155
    .      133
    .      181
    .      177
    .      164
    5.0    161
    3.0    154
    .      136
    .      132
    6.0    123
    .      108
    .      107
    7.0    102
    .      93
    .      51
type: test4
completed iteration 11
    .      953
    .      118
    .      909
    .      897
    .      738
    2.0    499
    .      264
    .      205
    .      198
    .      195
    .      154
    .      144
    .      172
    .      161
    .      156
    .      136
    .      133
    .      122
    .      108
    3.0    107
    .      102
    .      93
    .      51
type: test4
completed iteration 12
    .      953
    .      118
    .      952
    .      909
    .      535
    .      499
    .      264
    .      209
    .      198
    2.0    195
    .      184
    .      183
    .      172
    .      165
    .      159
    .      136
    .      134
    .      123
    .      108
    3.0    107
    .      102
    .      93
    .      51
type: test4
completed iteration 13
    .      953
    .      118
    .      959
    .      897
    .      739
    2.0    499
    .      264
    .      198
    .      138
    2.0    105
    .      104
    .      104
    .      172
    .      165
    .      159
    .      136
    .      134
    .      123
    .      108
    3.0    107
    .      102
    .      93
    .      51

```

```
spectral_clustering(  
  le_X_train25,  
  cuts=23,  
  fname="label_encoded_spectral_clustering",  
  affinity="linear",  
)
```

```

3.0.e 136
14.0.e 127
17.0.e 118
36.0.e 108
4.0.e 97
21.0.e 105
4.0.e 93
dtype: int64
Completed iteration 7
1.0.e 1033
8.0.e 817
5.0.e 862
11.0.e 964
22.0.e 909
7.0.e 264
13.0.e 222
3.0.e 295
8.0.e 281
12.0.e 205
6.0.e 181
14.0.e 172
13.0.e 152
20.0.e 158
30.0.e 136
17.0.e 133
21.0.e 138
10.0.e 108
10.0.e 106
dtype: int64
Completed iteration 8
1.0.e 1611
8.0.e 1138
5.0.e 1108
2.0.e 897
11.0.e 954
22.0.e 499
7.0.e 264
15.0.e 213
1.0.e 198
12.0.e 195
8.0.e 254
6.0.e 182
1.0.e 171
20.0.e 164
13.0.e 158
10.0.e 158
10.0.e 136
14.0.e 136
21.0.e 138
17.0.e 133
16.0.e 108
10.0.e 107
4.0.e 93
dtype: int64
Completed iteration 9
1.0.e 1611
8.0.e 1138
5.0.e 1108
2.0.e 897
11.0.e 948
22.0.e 499
7.0.e 264
15.0.e 209
3.0.e 198
6.0.e 184
12.0.e 195
8.0.e 171
20.0.e 164
13.0.e 162
31.0.e 152
18.0.e 136
14.0.e 132
16.0.e 108
17.0.e 108
dtype: int64
Completed iteration 10
1.0.e 1611
8.0.e 1138

```


[illegible]

```
dtype: int64
Completed iteration :
3.9      6526
```

6.0	42
1.0	32
0.0	22

```

7.0  447
9.0  445
6.0  421
1.0  401
22.0  329
17.0  329
13.0  290
8.0  263
5.0  252
4.0  247
18.0  213
16.0  129
12.0  100
11.0  95
20.0  85
15.0  79
8.0  61
dtype: int64
Completed iteration 1
3.0  6503
21.0  587
2.0  599
8.0  487
9.0  466
7.0  444
6.0  422
1.0  402
17.0  329
22.0  328
13.0  287
8.0  263
5.0  248
4.0  247
18.0  212
16.0  128

```

11.0 52
12.0 87
13.0 101
14.0 120
15.0 136
16.0 154
17.0 171
18.0 187
19.0 203
20.0 219
21.0 235
22.0 251
23.0 267
24.0 283
25.0 299
26.0 315
27.0 331
28.0 347
29.0 363
30.0 379
31.0 395
32.0 411
33.0 427
34.0 443
35.0 459
36.0 475
37.0 491
38.0 507
39.0 523
40.0 539
41.0 555
42.0 571
43.0 587
44.0 603
45.0 619
46.0 635
47.0 651
48.0 667
49.0 683
50.0 699
51.0 715
52.0 731
53.0 747
54.0 763
55.0 779
56.0 795
57.0 811
58.0 827
59.0 843
60.0 859
61.0 875
62.0 891
63.0 907
64.0 923
65.0 939
66.0 955
67.0 971
68.0 987
69.0 1003
70.0 1019
71.0 1035
72.0 1051
73.0 1067
74.0 1083
75.0 1099
76.0 1115
77.0 1131
78.0 1147
79.0 1163
80.0 1179
81.0 1195
82.0 1211
83.0 1227
84.0 1243
85.0 1259
86.0 1275
87.0 1291
88.0 1307
89.0 1323
90.0 1339
91.0 1355
92.0 1371
93.0 1387
94.0 1403
95.0 1419
96.0 1435
97.0 1451
98.0 1467
99.0 1483
100.0 1499
101.0 1515
102.0 1531
103.0 1547
104.0 1563
105.0 1579
106.0 1595
107.0 1611
108.0 1627
109.0 1643
110.0 1659
111.0 1675
112.0 1691
113.0 1707
114.0 1723
115.0 1739
116.0 1755
117.0 1771
118.0 1787
119.0 1803
120.0 1819
121.0 1835
122.0 1851
123.0 1867
124.0 1883
125.0 1899
126.0 1915
127.0 1931
128.0 1947
129.0 1963
130.0 1979
131.0 1995
132.0 2011
133.0 2027
134.0 2043
135.0 2059
136.0 2075
137.0 2091
138.0 2107
139.0 2123
140.0 2139
141.0 2155
142.0 2171
143.0 2187
144.0 2203
145.0 2219
146.0 2235
147.0 2251
148.0 2267
149.0 2283
150.0 2299
151.0 2315
152.0 2331
153.0 2347
154.0 2363
155.0 2379
156.0 2395
157.0 2411
158.0 2427
159.0 2443
160.0 2459
161.0 2475
162.0 2491
163.0 2507
164.0 2523
165.0 2539
166.0 2555
167.0 2571
168.0 2587
169.0 2603
170.0 2619
171.0 2635
172.0 2651
173.0 2667
174.0 2683
175.0 2699
176.0 2715
177.0 2731
178.0 2747
179.0 2763
180.0 2779
181.0 2795
182.0 2811
183.0 2827
184.0 2843
185.0 2859
186.0 2875
187.0 2891
188.0 2907
189.0 2923
190.0 2939
191.0 2955
192.0 2971
193.0 2987
194.0 3003
195.0 3019
196.0 3035
197.0 3051
198.0 3067
199.0 3083
200.0 3099
201.0 3115
202.0 3131
203.0 3147
204.0 3163
205.0 3179
206.0 3195
207.0 3211
208.0 3227
209.0 3243
210.0 3259
211.0 3275
212.0 3291
213.0 3307
214.0 3323
215.0 3339
216.0 3355
217.0 3371
218.0 3387
219.0 3403
220.0 3419
221.0 3435
222.0 3451
223.0 3467
224.0 3483
225.0 3499
226.0 3515
227.0 3531
228.0 3547
229.0 3563
230.0 3579
231.0 3595
232.0 3611
233.0 3627
234.0 3643
235.0 3659
236.0 3675
237.0 3691
238.0 3707
239.0 3723
240.0 3739
241.0 3755
242.0 3771
243.0 3787
244.0 3803
245.0 3819
246.0 3835
247.0 3851
248.0 3867
249.0 3883
250.0 3899
251.0 3915
252.0 3931
253.0 3947
254.0 3963
255.0 3979
256.0 3995
257.0 4011
258.0 4027
259.0 4043
260.0 4059
261.0 4075
262.0 4091
263.0 4107
264.0 4123
265.0 4139
266.0 4155
267.0 4171
268.0 4187
269.0 4203
270.0 4219
271.0 4235
272.0 4251
273.0 4267
274.0 4283
275.0 4299
276.0 4315
277.0 4331
278.0 4347
279.0 4363
280.0 4379
281.0 4395
282.0 4411
283.0 4427
284.0 4443
285.0 4459
286.0 4475
287.0 4491
288.0 4507
289.0 4523
290.0 4539
291.0 4555
292.0 4571
293.0 4587
294.0 4603
295.0 4619
296.0 4635
297.0 4651
298.0 4667
299.0 4683
300.0 4699
301.0 4715
302.0 4731
303.0 4747
304.0 4763
305.0 4779
306.0 4795
307.0 4811
308.0 4827
309.0 4843
310.0 4859
311.0 4875
312.0 4891
313.0 4907
314.0 4923
315.0 4939
316.0 4955
317.0 4971
318.0 4987
319.0 5003
320.0 5019
321.0 5035
322.0 5051
323.0 5067
324.0 5083
325.0 5099
326.0 5115
327.0 5131
328.0 5147
329.0 5163
330.0 5179
331.0 5195
332.0 5211
333.0 5227
334.0 5243
335.0 5259
336.0 5275
337.0 5291
338.0 5307
339.0 5323
340.0 5339
341.0 5355
342.0 5371
343.0 5387
344.0 5403
345.0 5419
346.0 5435
347.0 5451
348.0 5467
349.0 5483
350.0 5499
351.0 5515
352.0 5531
353.0 5547
354.0 5563
355.0 5579
356.0 5595
357.0 5611
358.0 5627
359.0 5643
360.0 5659
361.0 5675
362.0 5691
363.0 5707
364.0 5723
365.0 5739
366.0 5755
367.0 5771
368.0 5787
369.0 5803
370.0 5819
371.0 5835
372.0 5851
373.0 5867
374.0 5883
375.0 5899
376.0 5915
377.0 5931
378.0 5947
379.0 5963
380.0 5979
381.0 5995
382.0 6011
383.0 6027
384.0 6043
385.0 6059
386.0 6075
387.0 6091
388.0 6107
389.0 6123
390.0 6139
391.0 6155
392.0 6171
393.0 6187
394.0 6203
395.0 6219
396.0 6235
397.0 6251
398.0 6267
399.0 6283
400.0 6299
401.0 6315
402.0 6331
403.0 6347
404.0 6363
405.0 6379
406.0 6395
407.0 6411
408.0 6427
409.0 6443
410.0 6459
411.0 6475
412.0 6491
413.0 6507
414.0 6523
415.0 6539
416.0 6555
417.0 6571
418.0 6587
419.0 6603
420.0 6619
421.0 6635
422.0 6651
423.0 6667
424.0 6683
425.0 6699
426.0 6715
427.0 6731
428.0 6747
429.0 6763
430.0 6779
431.0 6795
432.0 6811
433.0 6827
434.0 6843
435.0 6859
436.0 6875
437.0 6891
438.0 6907
439.0 6923
440.0 6939
441.0 6955
442.0 6971
443.0 6987
444.0 7003
445.0 7019
446.0 7035
447.0 7051
448.0 7067
449.0 7083
450.0 7099
451.0 7115
452.0 7131
453.0 7147
454.0 7163
455.0 7179
456.0 7195
457.0 7211
458.0 7227
459.0 7243
460.0 7259
461.0 7275
462.0 7291
463.0 7307
464.0 7323
465.0 7339
466.0 7355
467.0 7371
468.0 7387
469.0 7403
470.0 7419
471.0 7435
472.0 7451
473.0 7467
474.0 7483
475.0 7499
476.0 7515
477.0 7531
478.0 7547
479.0 7563
480.0 7579
481.0 7595
482.0 7611
483.0 7627
484.0 7643
485.0 7659
486.0 7675
487.0 7691
488.0 7707
489.0 7723
490.0 7739
491.0 7755
492.0 7771
493.0 7787
494.0 7803
495.0 7819
496.0 7835
497.0 7851
498.0 7867
499.0 7883
500.0 7899
501.0 7915
502.0 7931
503.0 7947
504.0 7963
505.0 7979
506.0 7995
507.0 8011
508.0 8027
509.0 8043
510.0 8059
511.0 8075
512.0 8091
513.0 8107
514.0 8123
515.0 8139
516.0 8155
517.0 8171
518.0 8187
519.0 8203
520.0 8219
521.0 8235
522.0 8251
523.0 8267
524.0 8283
525.0 8299
526.0 8315
527.0 8331
528.0 8347
529.0 8363
530.0 8379
531.0 8395
532.0 8411
533.0 8427
534.0 8443
535.0 8459
536.0 8475
537.0 8491
538.0 8507
539.0 8523
540.0 8539
541.0 8555
542.0 8571
543.0 8587
544.0 8603
545.0 8619
546.0 8635
547.0 8651
548.0 8667
549.0 8683
550.0 8699
551.0 8715
552.0 8731
553.0 8747
554.0 8763
555.0 8779
556.0 8795
557.0 8811
558.0 8827
559.0 8843
560.0 8859
561.0 8875
562.0 8891
563.0 8907
564.0 8923
565.0 8939
566.0 8955
567.0 8971
568.0 8987
569.0 9003
570.0 9019
571.0 9035
572.0 9051
573.0 9067
574.0 9083
575.0 9099
576.0 9115
577.0 9131
578.0 9147
579.0 9163
580.0 9179
581.0 9195
582.0 9211
583.0 9227
584.0 9243
585.0 9259
586.0 9275
587.0 9291
588.0 9307
589.0 9323
590.0 9339
591.0 9355
592.0 9371
593.0 9387
594.0 9403
595.0 9419
596.0 9435
597.0 9451
598.0 9467
599.0 9483
600.0 9499
601.0 9515
602.0 9531
603.0 9547
604.0 9563
605.0 9579
606.0 9595
607.0 9611
608.0 9627
609.0 9643
610.0 9659
611.0 9675
612.0 9691
613.0 9707
614.0 9723
615.0 9739
616.0 9755
617.0 9771
618.0 9787
619.0 9803
620.0 9819
621.0 9835
622.0 9851
623.0 9867
624.0 9883
625.0 9899
626.0 9915
627.0 9931
628.0 9947
629.0 9963
630.0 9979
631.0 9995
632.0 10011
633.0 10027
634.0 10043
635.0 10059
636.0 10075
637.0 10091
638.0 10107
639.0 10123
640.0 10139
641.0 10155
642.0 10171
643.0 10187
644.0 10203
645.0 10219
646.0 10235
647.0 10251
648.0 10267
649.0 10283
650.0 10299
651.0 10315
652.0 10331
653.0 10347
654.0 10363
655.0 10379
656.0 10395
657.0 10411
658.0 10427
659.0 10443
660.0 10459
661.0 10475
662.0 10491
663.0 10507
664.0 10523
665.0 10539
666.0 10555
667.0 10571
668.0 10587
669.0 10603
670.0 10619
671.0 10635
672.0 10651
673.0 10667
674.0 10683
675.0 10699
676.0 10715
677.0 10731
678.0 10747
679.0 10763
680.0 10779
681.0 10795
682.0 10811
683.0 10827
684.0 10843
685.0 10859
686.0 10875
687.0 10891
688.0 10907
689.0 10923
690.0 10939
691.0 10955
692.0 10971
693.0 10987
694.0 11003
695.0 11019
696.0 11035
697.0 11051
698.0 11067
699.0 11083
700.0 11099
701.0 11115
702.0 11131
703.0 11147
704.0 11163
705.0 11179
706.0 11195
707.0 11211
708.0 11227
709.0 11243
710.0 11259
711.0 11275
712.0 11291
713.0 11307
714.0 11323
715.0 11339
716.0 11355
717.0 11371
718.0 11387
719.0 11403
720.0 11419
721.0 11435
722.0 11451
723.0 11467
724.0 11483
725.0 11499
726.0 11515
727.0 11531
728.0 11547
729.0 11563
730.0 11579
731.0 11595
732.0 11611
733.0 11627
734.0 11643
735.0 11659
736.0 11675
737.0 11691
738.0 11707
739.0 11723
740.0 11739
741.0 11755
742.0 11771
743.0 11787
744.0 11803
745.0 11819
746.0 11835
747.0 11851
748.0 11867
749.0 11883
750.0 11899
751.0 11915
752.0 11931
753.0 11947
754.0 11963
755.0 11979
756.0 11995
757.0 12011
758.0 12027
759.0 12043
760.0 12059
761.0 12075
762.0 12091
763.0 12107
764.0 12123
765.0 12139
766.0 12155
767.0 12171
768.0 12187
769.0 12203
770.0 12219
771.0 12235
772.0 12251
773.0 12267
774.0 12283
775.0 12299
776.0 12315
777.0 12331
778.0 12347
779.0 12363
780.0 12379
781.0 12395
782.0 12411
783.0 12427
784.0 12443
785.0 12459
786.0 12475
787.0 12491
788.0 12507
789.0 12523
790.0 12539
791.0 12555
792.0 12571
793.0 12587
794.0 12603
795.0 12619
796.0 12635
797.0 12651
798.0 12667
799.0 12683
800.0 12699
801.0 12715
802.0 12731
803.0 12747
804.0 12763
805.0 12779
806.0 12795
807.0 12811
808.0 12827
809.0 12843
810.0 12859
811.0 12875
812.0 12891
813.0 12907
814.0 12923
815.0 12939
816.0 12955
817.0 12971
818.0 12987
819.0 13003
820.0 13019
821.0 13035
822.0 13051
823.0 13067
824.0 13083
825.0 13099
826.0 13115
827.0 13131
828.0 13147
829.0 13163
830.0 13179
831.0 13195
832.0 13211
833.0 13227
834.0 13243
835.0 13259
836.0 13275
837.0 13291
838.0 13307
839.0 13323
840.0 13339
841.0 13355
842.0 13371
843.0 13387
844.0 13403
845.0 13419
846.0 13435
847.0 13451
848.0 13467
849.0 13483
850.0 13499
851.0 13515
852.0 13531
853.0 13547
854.0 13563
855.0 13579
856.0 13595
857.0 13611
858.0 13627
859.0 13643
860.0 13659
861.0 13675
862.0 13691
863.0 13707
864.0 13723
865.0 13739
866.0 13755
867.0 13771
868.0 13787
869.0 13803
870.0 13819
871.0 13835
872.0 13851
873.0 13867
874.0 13883
875.0 13899
876.0 13915
877.0 13931
878.0 13947
879.0 13963
880.0 13979
881.0 13995
882.0 14011
883.0 14027
884.0 14043
885.0 14059
886.0 14075
887.0 14091
888.0 14107
889.0 14123
890.0 14139
891.0 14155
892.0 14171
893.0 14187
894.0 14203
895.0 14219
896.0 14235
897.0 14251
898.0 14267
899.0 14283
900.0 14299
901.0 14315
902.0 14331
903.0 14347
904.0 14363
905.0 14379
906.0 14395
907.0 14411
908.0 14427
909.0 14443
910.0 14459
911.0 14475
912.0 14491
913.0 14507
914.0 14523
915.0 14539
916.0 14555
917.0 14571
918.0 14587
919.0 14603
920.0 14619
921.0 14635
922.0 14651
923.0 14667
924.0 14683
925.0 14699
926.0 14715
927.0 14731
928.0 14747
929.0 14763
930.0 14779
931.0 14795
932.0 14811
933.0 14827
934.0 14843
935.0 14859
936.0 14875
937.0 14891
938.0 14907
939.0 14923
940.0 14939
941.0 14955
942.0 14971
943.0 14987
944.0 15003
945.0 15019
946.0 15035
947.0 15051
948.0 15067
949.0 15083
950.0 15099
951.0 15115
952.0 15131
953.0 15147
954.0 15163
955.0 15179
956.0 15195
957.0 15211
958.0 15227
959.0 15243
960.0 15259
961.0 15275
962.0 15291
963.0 15307
964.0 15323
965.0 15339
966.0 15355
967.0 15371
968.0 15387
969.0 15403
970.0 15419
971.0 15435
972.0 15451
973.0 15467
974.0 15483
975.0 15499
976.0 15515
977.0 15531
978.0 15547
979.0 15563
980.0 15579
981.0 15595
982.0 15611
983.0 15627
984.0 15643
985.0 15659
986.0 15675
987.0 15691
988.0 15707
989.0 15723
990.0 15739
991.0 15755
992.0 15771
993.0 15787
994.0 15803
995.0 15819
996.0 15835
997.0 15851
998.0 15867
999.0 15883
1000.0 15899
1001.0 15915
1002.0 15931
1003.0 15947
1004.0 15963
1005.0 15979
1006.0 15995
1007.0 16011
1008.0 16027
1009.0 16043
1010.0 16059
1011.0 16075
1012.0 16091
1013.0 16107
1014.0 16123
1015.0 16139
1016.0 16155
1017.0 16171
1018.0 16187
1019.0

[illegible][illegible][illegible]

[illegible]

```

Central 10: [-0.4777229  0.8055426  0.5278456  0.5601226  0.9015934 -0.8308
0.7152143  0.1804706  0.8797851  0.3408817  0.0011844  0.0245633]
0.87278487 [-1.1812508  1.1370773]
Central 11: [-0.9122582  0.8506446  0.1689968  0.1123104  0.2280387  0.8
0.8018713  0.8732208  0.1872234  0.1568988  0.8875447  0.8808154
0.8808158  0.8808158  0.1318951]
0.8126171 [-0.8622189  0.8622189  0.8203684  0.1287465  0.8465954 -0.8158
0.7628864  0.2882793  0.1860777  0.1565128  0.8301817  0.8808654
0.8021995  0.8806271  0.4312187]
0.7628864 [-0.2882793  0.1860777  0.1565128  0.8301817  0.8808654
0.8021995  0.8806271  0.4312187]
0.8487795 [-0.7587138  0.1386215  0.3488021  0.131815  0.1871464  0.8073
0.8487795  0.7587138  0.2785151  0.6486103  0.893012  0.8192491]
0.8487795 [-0.2785151  0.6486103  0.8192491  0.8487795  0.7587138  0.8073
0.7587138  0.1386215  0.3488021  0.131815  0.1871464  0.8073]
0.8192491 [-0.1386215  0.1871464  0.8073  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.8192491  0.2785151  0.131815  0.6486103
0.8487795  0.7587138  0.2785151  0.2785151  0.6486103]
0.8192491 [-0.2785151  0.6486103  0.8487795  0.8192491  0.131815  0.6486103  0.8487795
0.7587138  0.2785151  0.2785151  0.6486103  0.8487795]
0.8192491 [-0.6486103  0.8487795  0.8073  0.81
```

[illegible]

```

18.0 355
16.0 312
17.0 324
15.0 315
3.0 284
7.0 293
4.0 264
12.0 264
13.0 250
21.0 242
20.0 242
18.0 234
16.0 234
6.0 215
11.0 228
5.0 228
8.0 195
13.0 174
14.0 147
dtype: int64
Completed iteration 4
0.0 1028
2.0 1028
1.0 495
22.0 385
9.0 382
18.0 346
17.0 338
15.0 315
16.0 328
5.0 305
7.0 285
12.0 282
19.0 256
4.0 255
21.0 253
20.0 243
6.0 241
16.0 238
11.0 229
8.0 228
13.0 176
14.0 175
dtype: int64
Completed iteration 5
0.0 1011
2.0 1008
1.0 487
22.0 389
9.0 384
17.0 344
15.0 338
18.0 332
16.0 315
3.0 288
7.0 286
19.0 260
21.0 257
4.0 258
12.0 249
6.0 241
20.0 241
11.0 236
16.0 234
5.0 216
2.0 215
8.0 188
13.0 182
14.0 175
dtype: int64
Completed iteration 6
0.0 1011
2.0 1028
1.0 481
22.0 401
9.0 383
17.0 346
15.0 338
18.0 332
16.0 315
3.0 299
19.0 278
21.0 268
7.0 266
4.0 245
12.0 245
6.0 244

```

28.0	236
11.9	236
18.0	236
11.9	236
8.0	180
11.9	180
14.0	180
14.0	180
Completed iteration 7	
8.0	312
2.0	360
8.0	472
11.9	472
8.0	387
11.9	387
11.9	316
11.9	316
16.0	323
16.0	323
22.0	256
22.0	256
7.0	264
6.0	287
28.0	217
28.0	217
11.9	219
11.9	219
8.0	191
8.0	191
Completed iteration 8	
8.0	3612
2.0	3608
1.0	400
22.0	468
22.0	468
17.0	383
17.0	383
16.0	322
16.0	322
15.0	328
15.0	328
20.0	295
20.0	295
21.0	267
7.0	268
18.0	246
6.0	244
11.9	241
11.9	241
28.0	238
11.9	238
11.9	238
11.9	209
11.9	209
14.0	179
14.0	179
Completed iteration 9	
8.0	3612
2.0	3608
1.0	400
22.0	468
22.0	468
17.0	382
17.0	382
16.0	327
16.0	327
15.0	328
15.0	328
20.0	295
20.0	295
21.0	267
7.0	267
18.0	240
6.0	239
11.9	239
11.9	239
28.0	239
28.0	239
11.9	216
11.9	216
14.0	179
14.0	179
Completed iteration 10	
8.0	3612
2.0	3608
1.0	400
22.0	468
22.0	468
17.0	382
17.0	382
16.0	327
16.0	327
15.0	328
15.0	328
20.0	295
20.0	295
21.0	267
7.0	267
18.0	240
6.0	239
11.9	239
11.9	239
28.0	239
28.0	239
11.9	216
11.9	216
14.0	179
14.0	179
Completed iteration 11	
8.0	3612
2.0	3608
1.0	400
22.0	468
22.0	468
17.0	382
17.0	382
16.0	327
16.0	327
15.0	328
15.0	328
20.0	295
20.0	295
21.0	267
7.0	267
18.0	240
6.0	239
11.9	239
11.9	239
28.0	239
28.0	239
11.9	216
11.9	216
14.0	179
14.0	179
Completed iteration 12	
8.0	3612
2.0	3608

```

8.0 451
22.0 444
17.0 380
15.0 371
15.0 328
16.0 313
18.0 316
3.0 303
19.0 290
21.0 283
18.0 245
20.0 245
11.0 237
7.0 237
12.0 215
6.0 228
5.0 217
13.0 206
8.0 195
14.0 178
dtype: int64
Completed iteration 11
8.0 3812
2.0 3808
22.0 444
17.0 371
15.0 328
16.0 316
18.0 313
3.0 303
19.0 290
21.0 283
18.0 245
20.0 245
11.0 237
7.0 237
12.0 215
6.0 228
5.0 217
13.0 206
14.0 177
dtype: int64
Completed iteration 12
8.0 3812
2.0 3808
22.0 444
17.0 371
15.0 328
16.0 316
18.0 313
3.0 299
21.0 290
18.0 248
20.0 248
11.0 239
7.0 239
12.0 212
6.0 225
5.0 216
13.0 216
8.0 194
14.0 178
dtype: int64
Completed iteration 13
8.0 3812
2.0 3808
22.0 457
17.0 445
15.0 371
15.0 365
16.0 355
18.0 329
19.0 307
18.0 306
3.0 297
21.0 213
18.0 242

```

```

6.8 241
11.8 248
7.8 239
20.8 237
12.8 238
6.8 238
15.8 225
5.8 215
8.8 197
14.8 177
dtype: int64
Completed iteration 14
0.8 3611
2.8 3628
22.8 408
1.8 441
3.8 365
5.8 363
15.8 332
15.8 327
15.8 314
18.8 385
5.8 295
22.8 254
18.8 244
6.8 243
11.8 239
23.8 237
20.8 237
13.8 232
12.8 227
6.8 213
5.8 216
8.8 202
14.8 177
dtype: int64
Completed iteration 15
0.8 3611
2.8 3628
22.8 408
17.8 364
9.8 368
16.8 355
15.8 332
19.8 315
18.8 308
3.8 296
21.8 256
18.8 244
6.8 243
11.8 239
7.8 238
11.8 235
20.8 232
17.8 228
4.8 221
5.8 228
8.8 204
14.8 177
dtype: int64
Completed iteration 16
0.8 3611
2.8 3628
22.8 404
5.8 448
17.8 364
10.8 368
15.8 332
19.8 316
18.8 308
3.8 296
19.8 256
21.8 244
6.8 244
18.8 243
11.8 239
7.8 238
11.8 235
20.8 232
12.8 232
4.8 228
6.8 208
14.8 175
dtype: int64

```

```

Completed iteration 17
  2.8  3812
  20.8  481
  1.8  442
  20.8  561
  0.8  568
  16.8  513
  11.8  133
  10.8  137
  1.8  261
  18.8  237
  21.8  262
  8.8  246
  11.8  247
  11.8  248
  21.8  237
  11.8  237
  7.8  236
  20.8  236
  5.8  225
  12.8  225
  4.8  228
  8.8  222
  14.8  177
  17.8  177
  17.8  177
Completed iteration 18
  0.8  3612
  21.8  445
  8.8  459
  1.8  562
  17.8  519
  16.8  511
  11.8  512
  10.8  512
  1.8  262
  18.8  237
  21.8  262
  8.8  246
  11.8  247
  11.8  248
  21.8  237
  11.8  237
  7.8  236
  20.8  236
  5.8  225
  12.8  225
  4.8  228
  8.8  222
  14.8  177
  17.8  177
  17.8  177
Completed iteration 19
  0.8  3612
  21.8  445
  8.8  459
  1.8  562
  17.8  519
  16.8  511
  11.8  512
  10.8  512
  1.8  262
  18.8  237
  21.8  262
  8.8  246
  11.8  247
  11.8  248
  21.8  237
  11.8  237
  7.8  236
  20.8  236
  5.8  225
  12.8  225
  4.8  228
  8.8  222
  14.8  177
  17.8  177
  17.8  177
Completed iteration 20
  0.8  3612
  21.8  445
  8.8  459
  1.8  564
  17.8  519
  16.8  519
  11.8  516
  10.8  513
  1.8  262
  18.8  237
  21.8  262
  8.8  246
  11.8  247
  11.8  248
  21.8  237
  11.8  237
  7.8  236
  20.8  236
  5.8  225
  12.8  225
  4.8  228
  8.8  222
  14.8  177
  17.8  177
  17.8  177

```

```

18      209
19      247
20      6.9
21      248
22      10.8
23      249
24      10.8
25      250
26      11.9
27      251
28      11.9
29      252
30      14.1
31      161
32      162
33      163
34      164
35      165
36      166
37      167
38      168
39      169
40      170
41      171
42      172
43      173
44      174
45      175
46      176
47      177
48      178
49      179
50      180
51      181
52      182
53      183
54      184
55      185
56      186
57      187
58      188
59      189
60      190
61      191
62      192
63      193
64      194
65      195
66      196
67      197
68      198
69      199
70      200
71      201
72      202
73      203
74      204
75      205
76      206
77      207
78      208
79      209
80      210
81      211
82      212
83      213
84      214
85      215
86      216
87      217
88      218
89      219
90      220
91      221
92      222
93      223
94      224
95      225
96      226
97      227
98      228
99      229
100     230
101     231
102     232
103     233
104     234
105     235
106     236
107     237
108     238
109     239
110     240
111     241
112     242
113     243
114     244
115     245
116     246
117     247
118     248
119     249
120     250
121     251
122     252
123     253
124     254
125     255
126     256
127     257
128     258
129     259
130     260
131     261
132     262
133     263
134     264
135     265
136     266
137     267
138     268
139     269
140     270
141     271
142     272
143     273
144     274
145     275
146     276
147     277
148     278
149     279
150     280
151     281
152     282
153     283
154     284
155     285
156     286
157     287
158     288
159     289
160     290
161     291
162     292
163     293
164     294
165     295
166     296
167     297
168     298
169     299
170     300
171     301
172     302
173     303
174     304
175     305
176     306
177     307
178     308
179     309
180     310
181     311
182     312
183     313
184     314
185     315
186     316
187     317
188     318
189     319
190     320
191     321
192     322
193     323
194     324
195     325
196     326
197     327
198     328
199     329
200     330
201     331
202     332
203     333
204     334
205     335
206     336
207     337
208     338
209     339
210     340
211     341
212     342
213     343
214     344
215     345
216     346
217     347
218     348
219     349
220     350
221     351
222     352
223     353
224     354
225     355
226     356
227     357
228     358
229     359
230     360
231     361
232     362
233     363
234     364
235     365
236     366
237     367
238     368
239     369
240     370
241     371
242     372
243     373
244     374
245     375
246     376
247     377
248     378
249     379
250     380
251     381
252     382
253     383
254     384
255     385
256     386
257     387
258     388
259     389
260     390
261     391
262     392
263     393
264     394
265     395
266     396
267     397
268     398
269     399
270     400
271     401
272     402
273     403
274     404
275     405
276     406
277     407
278     408
279     409
280     410
281     411
282     412
283     413
284     414
285     415
286     416
287     417
288     418
289     419
290     420
291     421
292     422
293     423
294     424
295     425
296     426
297     427
298     428
299     429
300     430
301     431
302     432
303     433
304     434
305     435
306     436
307     437
308     438
309     439
310     440
311     441
312     442
313     443
314     444
315     445
316     446
317     447
318     448
319     449
320     450
321     451
322     452
323     453
324     454
325     455
326     456
327     457
328     458
329     459
330     460
331     461
332     462
333     463
334     464
335     465
336     466
337     467
338     468
339     469
340     470
341     471
342     472
343     473
344     474
345     475
346     476
347     477
348     478
349     479
350     480
351     481
352     482
353     483
354     484
355     485
356     486
357     487
358     488
359     489
360     490
361     491
362     492
363     493
364     494
365     495
366     496
367     497
368     498
369     499
370     500
371     501
372     502
373     503
374     504
375     505
376     506
377     507
378     508
379     509
380     510
381     511
382     512
383     513
384     514
385     515
386     516
387     517
388     518
389     519
390     520
391     521
392     522
393     523
394     524
395     525
396     526
397     527
398     528
399     529
400     530
401     531
402     532
403     533
404     534
405     535
406     536
407     537
408     538
409     539
410     540
411     541
412     542
413     543
414     544
415     545
416     546
417     547
418     548
419     549
420     550
421     551
422     552
423     553
424     554
425     555
426     556
427     557
428     558
429     559
430     560
431     561
432     562
433     563
434     564
435     565
436     566
437     567
438     568
439     569
440     570
441     571
442     572
443     573
444     574
445     575
446     576
447     577
448     578
449     579
450     580
451     581
452     582
453     583
454     584
455     585
456     586
457     587
458     588
459     589
460     590
461     591
462     592
463     593
464     594
465     595
466     596
467     597
468     598
469     599
470     600
471     601
472     602
473     603
474     604
475     605
476     606
477     607
478     608
479     609
480     610
481
```

[illegible]

8	367
9	368
10	369
11	370
12	371
13	372
14	373
15	374
16	375
17	376
18	377
19	378
20	379
21	380
22	381
23	382
24	383
25	384
26	385
27	386
28	387
29	388
30	389
31	390
32	391
33	392
34	393
35	394
36	395
37	396
38	397
39	398
40	399
41	400
42	401
43	402
44	403
45	404
46	405
47	406
48	407
49	408
50	409
51	410
52	411
53	412
54	413
55	414
56	415
57	416
58	417
59	418
60	419
61	420
62	421
63	422
64	423
65	424
66	425
67	426
68	427
69	428
70	429
71	430
72	431
73	432
74	433
75	434
76	435
77	436
78	437
79	438
80	439
81	440
82	441
83	442
84	443
85	444
86	445
87	446
88	447
89	448
90	449
91	450
92	451
93	452
94	453
95	454
96	455
97	456
98	457
99	458
100	459
101	460
102	461
103	462
104	463
105	464
106	465
107	466
108	467
109	468
110	469
111	470
112	471
113	472
114	473
115	474
116	475
117	476
118	477
119	478
120	479
121	480
122	481
123	482
124	483
125	484
126	485
127	486
128	487
129	488
130	489
131	490
132	491
133	492
134	493
135	494
136	495
137	496
138	497
139	498
140	499
141	500
142	501
143	502
144	503
145	504
146	505
147	506
148	507
149	508
150	509
151	510
152	511
153	512
154	513
155	514
156	515
157	516
158	517
159	518
160	519
161	520
162	521
163	522
164	523
165	524
166	525
167	526
168	527
169	528
170	529
171	530
172	531
173	532
174	533
175	534
176	535
177	536
178	537
179	538
180	539
181	540
182	541
183	542
184	543
185	544
186	545
187	546
188	547
189	548
190	549
191	550
192	551
193	552
194	553
195	554
196	555
197	556
198	557
199	558
200	559
201	560
202	561
203	562
204	563
205	564
206	565
2	

[illegible][illegible]

[illegible]

12.0	400
10.0	380
8.0	285
14.0	242
16.0	213
9.0	236
11.0	236
13.0	214
17.0	305
20.0	260
22.0	185
7.0	142
10.0	171
8.0	140
4.0	146
10.0	121
@type: len	
Completed iteration 4	
0.0	1213
1.0	750
11.0	702
15.0	601
12.0	700
10.0	381
6.0	287
2.0	218
14.0	218
16.0	238
9.0	217
13.0	221
5.0	218
22.0	200
7.0	146
10.0	180
18.0	200
21.0	180
14.0	180
8.0	147
4.0	142
12.0	120
@type: len	
Completed iteration 5	
0.0	1213
1.0	1405
3.0	721
11.0	707
15.0	700
12.0	384
10.0	384
6.0	289
2.0	289
14.0	255
16.0	210
9.0	232
8.0	220
12.0	220
5.0	211
22.0	197
7.0	193
10.0	187
18.0	193
21.0	193
14.0	177
8.0	152
4.0	141
12.0	120
@type: len	
Completed iteration 6	
0.0	1213
1.0	1464
3.0	711
11.0	601
15.0	382
12.0	377
10.0	375
6.0	204
14.0	207
16.0	202
9.0	242
8.0	242
4.0	211
12.0	211
5.0	220
22.0	224
7.0	197
10.0	197
18.0	197
21.0	197
14.0	197

```

1.0  219
7.0  217
16.0 216
22.0 206
23.0 198
28.0 187
17.0 165
8.0  139
10.0 132
10.0 127
=====
Completed iteration 14
0.0  5131
1.0  1345
1.0  687
15.0 351
15.0 351
10.0 374
12.0 356
6.0  282
5.0  277
14.0 272
2.0  235
10.0 217
10.0 217
10.0 217
13.0 205
22.0 206
28.0 195
23.0 191
17.0 164
8.0  138
8.0  131
10.0 124
=====
dfgpe: cont'd
Completed iteration 15
0.0  5214
1.0  1345
1.0  687
15.0 351
15.0 351
10.0 375
12.0 356
6.0  281
5.0  278
14.0 269
2.0  255
9.0  235
18.0 225
11.0 217
7.0  215
16.0 213
10.0 207
28.0 192
23.0 191
17.0 166
8.0  137
8.0  128
10.0 126
=====
dfgpe: cont'd
Completed iteration 16
0.0  5213
1.0  1345
1.0  687
15.0 351
15.0 351
10.0 375
12.0 356
6.0  281
5.0  281
14.0 269
2.0  255
9.0  237
18.0 225
10.0 214
22.0 206
28.0 191
23.0 189
17.0 165
8.0  157
6.0  129
10.0 126
=====
dfgpe: cont'd

```

```

8      201
9      202
10     203
11     204
12     205
13     210
14     211
15     212
16     213
17     214
18     215
19     216
20     217
21     218
22     219
23     220
24     221
25     222
26     223
27     224
28     225
29     226
30     227
31     228
32     229
33     230
34     231
35     232
36     233
37     234
38     235
39     236
40     237
41     238
42     239
43     240
44     241
45     242
46     243
47     244
48     245
49     246
50     247
51     248
52     249
53     250
54     251
55     252
56     253
57     254
58     255
59     256
60     257
61     258
62     259
63     260
64     261
65     262
66     263
67     264
68     265
69     266
70     267
71     268
72     269
73     270
74     271
75     272
76     273
77     274
78     275
79     276
80     277
81     278
82     279
83     280
84     281
85     282
86     283
87     284
88     285
89     286
90     287
91     288
92     289
93     290
94     291
95     292
96     293
97     294
98     295
99     296
100    297
101    298
102    299
103    300
104    301
105    302
106    303
107    304
108    305
109    306
110    307
111    308
112    309
113    310
114    311
115    312
116    313
117    314
118    315
119    316
120    317
121    318
122    319
123    320
124    321
125    322
126    323
127    324
128    325
129    326
130    327
131    328
132    329
133    330
134    331
135    332
136    333
137    334
138    335
139    336
140    337
141    338
142    339
143    340
144    341
145    342
146    343
147    344
148    345
149    346
150    347
151    348
152    349
153    350
154    351
155    352
156    353
157    354
158    355
159    356
160    357
161    358
162    359
163    360
164    361
165    362
166    363
167    364
168    365
169    366
170    367
171    368
172    369
173    370
174    371
175    372
176    373
177    374
178    375
179    376
180    377
181    378
182    379
183    380
184    381
185    382
186    383
187    384
188    385
189    386
190    387
191    388
192    389
193    390
194    391
195    392
196    393
197    394
198    395
199    396
200    397
201    398
202    399
203    400
204    401
205    402
206    403
207    404
208    405
209    406
210    407
211    408
212    409
213    410
214    411
215    412
216    413
217    414
218    415
219    416
220    417
221    418
222    419
223    420
224    421
225    422
226    423
227    424
228    425
229    426
230    427
231    428
232    429
233    430
234    431
235    432
236    433
237    434
238    435
239    436
240    437
241    438
242    439
243    440
244    441
245    442
246    443
247    444
248    445
249    446
250    447
251    448
252    449
253    450
254    451
255    452
256    453
257    454
258    455
259    456
260    457
261    458
262    459
263    460
264    461
265    462
266    463
267    464
268    465
269    466
270    467
271    468
272    469
273    470
274    471
275    472
276    473
277    474
278    475
279    476
280    477
281    478
282    479
283    480
284    481
285    482
286    483
287    484
288    485
289    486
290    487
291    488
292    489
293    490
294    491
295    492
296    493
297    494
298    495
299    496
300    497
301    498
302    499
303    500
304    501
305    502
306    503
307    504
308    505
309    506
310    507
311    508
312    509
313    510
314    511
315    512
316    513
317    514
318    515
319    516
320    517
321    518
322    519
323    520
324    521
325    522
326    523
327    524
328    525
329    526
330    527
331    528
332    529
333    530
334    531
335    532
336    533
337    534
338    535
339    536
340    537
341    538
342    539
343    540
344    541
345    542
346    543
347    544
348    545
349    546
350    547
351    548
352    549
353    550
354    551
355    552
356    553
357    554
358    555
359    556
360    557
361    558
362    559
363    560
364    561
365    562
366    563
367    564
368    565
369    566
370    567
371    568
372    569
373    570
374    571
375    572
376    573
377    574
378    575
379    576
380    577
381    578
382    579
383    580
384    581
385    582
386    583
387    584
388    585
389    586
390    587
391    588
392    589
393    590
394    591
395    592
396    593
397    594
398    595
399    596
400    597
401    598
402    599
403    600
404    601
405    602
406    603
407    604
408    605
409    606
410    607
411    608
412    609
413    610
414    611
415    612
416    613
417    614
418    615
419    616
420    617
421    618
422    619
423    620
424    621
425    622
426    623
427    624
428    625
429    626
430    627
431    628
432    629
433    630
434    631
435    632
436    633
437    634
438    635
439    636
440    637
441    638
442    639
443    640
444    641
445    642
446    643
447    644
448    645
449    646
450    647
451    648
452    649
453    650
454    651
455    652
456    653
457    654
458    655
459    656
460    657
461    658
462    659
463    660
464    661
465    662
466    663
467    664
468    665
469    666
470    667
471    668
472    669
47
```

```

10.0 251
dtype: int64
Completed iteration 5
0.0 5112
12.0 791
4.0 1429
1.0 541
1.0 541
3.0 479
2.0 462
2.0 462
14.0 361
5.0 349
9.0 312
11.0 310
7.0 298
6.0 298
10.0 287
dtype: int64
Completed iteration 6
0.0 5112
12.0 791
1.0 481
13.0 510
3.0 497
2.0 459
8.0 469
14.0 364
5.0 341
9.0 329
11.0 315
7.0 306
10.0 298
0.0 285
dtype: int64
Completed iteration 7
0.0 5112
8.0 5109
12.0 797
3.0 631
13.0 514
2.0 620
2.0 643
14.0 371
5.0 348
9.0 328
11.0 309
7.0 301
8.0 297
dtype: int64
Completed iteration 8
0.0 5112
8.0 5107
12.0 794
1.0 642
13.0 517
3.0 562
2.0 632
8.0 606
14.0 357
5.0 344
9.0 328
7.0 317
10.0 311
11.0 296
6.0 291
dtype: int64
Completed iteration 9
0.0 5111
8.0 5108
12.0 793
1.0 647
13.0 541
3.0 586
14.0 369
8.0 464
14.0 369
9.0 318
7.0 325
10.0 304
6.0 293
11.0 298

```

[illegible]

```

[3]: spectral_clustering(
    w=A_trainD2,
    cost=[csp.uniquel[1,y_trainD2]],
    dtype="int64", model=spectral_clustering",
    affinity="chi2",
    gamma=1.0
)

```

```

Eigenvalue: 1.95956889e-01 5.99952381e-01 5.99952221e-01 5.9995157e-01 5.9995126e-01
Eigenvalue: 5.9995051e-01 5.9995021e-01 5.9995000e-01 5.9994979e-01 5.9994958e-01
Eigenvalue: 5.9994921e-01 5.9994915e-01 5.9994898e-01 5.9994880e-01 5.9994862e-01
Eigenvalue: 5.9994845e-01 5.9994837e-01 5.9994823e-01 5.9994808e-01 5.9994792e-01
Eigenvalue: 5.9994775e-01 5.9994761e-01 5.9994742e-01 5.9994728e-01 5.9994712e-01
Eigenvalue: 5.9994695e-01 5.9994681e-01 5.9994661e-01 5.9994643e-01 5.9994626e-01
Eigenvalue: 5.9994606e-01 5.9994592e-01 5.9994571e-01 5.9994553e-01 5.9994536e-01
Eigenvalue: 5.9994517e-01 5.9994503e-01 5.9994482e-01 5.9994464e-01 5.9994447e-01
Eigenvalue: 5.9994427e-01 5.9994413e-01 5.9994392e-01 5.9994374e-01 5.9994357e-01
Eigenvalue: 5.9994337e-01 5.9994323e-01 5.9994302e-01 5.9994284e-01 5.9994267e-01
Eigenvalue: 5.9994247e-01 5.9994233e-01 5.9994212e-01 5.9994194e-01 5.9994177e-01
Eigenvalue: 5.9994157e-01 5.9994143e-01 5.9994122e-01 5.9994104e-01 5.9994087e-01
Eigenvalue: 5.9994067e-01 5.9994053e-01 5.9994032e-01 5.9994014e-01 5.9993997e-01
Eigenvalue: 5.9993977e-01 5.9993963e-01 5.9993942e-01 5.9993924e-01 5.9993907e-01
Eigenvalue: 5.9993887e-01 5.9993873e-01 5.9993852e-01 5.9993834e-01 5.9993817e-01
Eigenvalue: 5.9993797e-01 5.9993783e-01 5.9993762e-01 5.9993744e-01 5.9993727e-01
Eigenvalue: 5.9993707e-01 5.9993693e-01 5.9993672e-01 5.9993654e-01 5.9993637e-01
Eigenvalue: 5.9993617e-01 5.9993603e-01 5.9993582e-01 5.9993564e-01 5.9993547e-01
Eigenvalue: 5.9993527e-01 5.9993513e-01 5.9993492e-01 5.9993474e-01 5.9993457e-01
Eigenvalue: 5.9993437e-01 5.9993423e-01 5.9993402e-01 5.9993384e-01 5.9993367e-01
Eigenvalue: 5.9993347e-01 5.9993333e-01 5.9993312e-01 5.9993294e-01 5.9993277e-01
Eigenvalue: 5.9993257e-01 5.9993243e-01 5.9993222e-01 5.9993204e-01 5.9993187e-01
Eigenvalue: 5.9993167e-01 5.9993153e-01 5.9993132e-01 5.9993114e-01 5.9993097e-01
Eigenvalue: 5.9993077e-01 5.9993063e-01 5.9993042e-01 5.9993024e-01 5.9993007e-01
Eigenvalue: 5.9992987e-01 5.9992973e-01 5.9992952e-01 5.9992934e-01 5.9992917e-01
Eigenvalue: 5.9992897e-01 5.9992883e-01 5.9992862e-01 5.9992844e-01 5.9992827e-01
Eigenvalue: 5.9992807e-01 5.9992793e-01 5.9992772e-01 5.9992754e-01 5.9992737e-01
Eigenvalue: 5.9992717e-01 5.9992703e-01 5.9992682e-01 5.9992664e-01 5.9992647e-01
Eigenvalue: 5.9992627e-01 5.9992613e-01 5.9992592e-01 5.9992574e-01 5.9992557e-01
Eigenvalue: 5.9992537e-01 5.9992523e-01 5.9992502e-01 5.9992484e-01 5.9992467e-01
Eigenvalue: 5.9992447e-01 5.9992433e-01 5.9992412e-01 5.9992394e-01 5.9992377e-01
Eigenvalue: 5.9992357e-01 5.9992343e-01 5.9992322e-01 5.9992304e-01 5.9992287e-01
Eigenvalue: 5.9992267e-01 5.9992253e-01 5.9992232e-01 5.9992214e-01 5.9992197e-01
Eigenvalue: 5.9992177e-01 5.9992163e-01 5.9992142e-01 5.9992124e-01 5.9992107e-01
Eigenvalue: 5.9992087e-01 5.9992073e-01 5.9992052e-01 5.9992034e-01 5.9992017e-01
Eigenvalue: 5.9991987e-01 5.9991973e-01 5.9991952e-01 5.9991934e-01 5.9991917e-01
Eigenvalue: 5.9991897e-01 5.9991883e-01 5.9991862e-01 5.9991844e-01 5.9991827e-01
Eigenvalue: 5.9991797e-01 5.9991783e-01 5.9991762e-01 5.9991744e-01 5.9991727e-01
Eigenvalue: 5.9991697e-01 5.9991683e-01 5.9991662e-01 5.9991644e-01 5.9991627e-01
Eigenvalue: 5.9991597e-01 5.9991583e-01 5.9991562e-01 5.9991544e-01 5.9991527e-01
Eigenvalue: 5.9991497e-01 5.9991483e-01 5.9991462e-01 5.9991444e-01 5.9991427e-01
Eigenvalue: 5.9991397e-01 5.9991383e-01 5.9991362e-01 5.9991344e-01 5.9991327e-01
Eigenvalue: 5.9991297e-01 5.9991283e-01 5.9991262e-01 5.9991244e-01 5.9991227e-01
Eigenvalue: 5.9991197e-01 5.9991183e-01 5.9991162e-01 5.9991144e-01 5.9991127e-01
Eigenvalue: 5.9991097e-01 5.9991083e-01 5.9991062e-01 5.9991044e-01 5.9991027e-01
Eigenvalue: 5.9990997e-01 5.9990983e-01 5.9990962e-01 5.9990944e-01 5.9990927e-01
Eigenvalue: 5.9990897e-01 5.9990883e-01 5.9990862e-01 5.9990844e-01 5.9990827e-01
Eigenvalue: 5.9990797e-01 5.9990783e-01 5.9990762e-01 5.9990744e-01 5.9990727e-01
Eigenvalue: 5.9990697e-01 5.9990683e-01 5.9990662e-01 5.9990644e-01 5.9990627e-01
Eigenvalue: 5.9990597e-01 5.9990583e-01 5.9990562e-01 5.9990544e-01 5.9990527e-01
Eigenvalue: 5.9990497e-01 5.9990483e-01 5.9990462e-01 5.9990444e-01 5.9990427e-01
Eigenvalue: 5.9990397e-01 5.9990383e-01 5.9990362e-01 5.9990344e-01 5.9990327e-01
Eigenvalue: 5.9990297e-01 5.9990283e-01 5.9990262e-01 5.9990244e-01 5.9990227e-01
Eigenvalue: 5.9990197e-01 5.9990183e-01 5.9990162e-01 5.9990144e-01 5.9990127e-01
Eigenvalue: 5.9990097e-01 5.9990083e-01 5.9990062e-01 5.9990044e-01 5.9990027e-01
Eigenvalue: 5.9989997e-01 5.
```

[illegible][illegible][illegible][illegible]

[illegible]

17.0	135
18.0	129
14.0	115
11.0	112
9.0	100
10.0	100
22.0	75
dtype: int64	
Completed iteration 7	
18.0	1257
18.0	1405
4.0	1345
2.0	807
5.0	738
6.0	600
6.0	522
20.0	300
20.0	300
15.0	212
15.0	204
13.0	132
17.0	177
12.0	170
21.0	165
3.0	150
17.0	135
18.0	117
14.0	110
11.0	110
11.0	110
19.0	100
19.0	100
21.0	75
dtype: int64	
Completed iteration 8	
1.0	1215
10.0	1187
4.0	1140
2.0	807
5.0	712
8.0	602
0.0	526
6.0	508
20.0	400
15.0	257
13.0	187
7.0	177
12.0	165
21.0	160
0.0	155
17.0	115
14.0	110
18.0	110
13.0	117
9.0	100
19.0	100
21.0	75
dtype: int64	
Completed iteration 9	
0.0	1215
10.0	1201
4.0	1140
2.0	804
5.0	713
8.0	602
0.0	525
6.0	480
20.0	297
15.0	220
13.0	200
13.0	180
17.0	177
7.0	160
21.0	165
0.0	155
17.0	115
14.0	110
18.0	110
13.0	117
9.0	100
19.0	100
21.0	75
dtype: int64	
Completed iteration 10	
1.0	1204
10.0	1190

```

4.8      1349
      8.0      803
      5.8      714
      3.8      602
      8.8      525
      6.8      402
      28.8      297
      16.8      188
      15.8      181
      12.8      168
      11.8      167
      7.8      167
      5.8      155
      12.8      155
      14.8      126
      10.8      115
      18.8      115
      8.8      108
      22.8      75
  hyper: test4
Completed iteration 11
      1.8      1548
      10.8      2392
      4.8      1849
      2.8      803
      5.8      714
      8.8      602
      6.8      525
      8.8      402
      10.8      297
      16.8      218
      15.8      188
      13.8      181
      12.8      178
      22.8      167
      7.8      161
      5.8      155
      17.8      155
      14.8      137
      28.8      114
      16.8      108
      22.8      75
  hyper: test4
Completed iteration 12
      1.8      1548
      10.8      2392
      4.8      1849
      2.8      803
      5.8      714
      8.8      602
      6.8      525
      8.8      401
      10.8      296
      16.8      218
      15.8      187
      12.8      172
      11.8      167
      7.8      156
      10.8      155
      14.8      141
      17.8      135
      12.8      117
      15.8      108
      22.8      108
      18.8      108
      22.8      75
  hyper: test4
Completed iteration 13
      1.8      1548
      10.8      2392
      4.8      1849
      2.8      803
      5.8      714
      8.8      602
      6.8      525
      8.8      401
      10.8      284
      16.8      211
      15.8      182
      12.8      182

```

[illegible]

```
eigenvalues: [9.9977779e-01 9.9972880e-01 9.9965966e-01 9.9958605e-01
9.9953741e-01 9.9938644e-01 9.9908270e-01 9.9766284e-01 9.7574848e-01
9.6557194e-01 8.1612897e-01 3.7485694e-01 9.3548957e-05 1.8744102e-11
```

```

1.0  1.000
@type: local
Completed iteration 5
2.0  1.044
2.0  1410
4.0  980
7.0  615
10.0  422
8.0  584
9.0  561
11.0  497
5.0  286
5.0  286
8.0  246
2.0  221
12.0  171
14.0  117
13.0  200
@type: local
Completed iteration 6
2.0  1.044
2.0  1410
4.0  911
7.0  655
10.0  412
0.0  589
5.0  559
5.0  559
5.0  295
8.0  246
1.0  231
12.0  171
14.0  107
13.0  188
@type: local
Completed iteration 7
2.0  1.044
2.0  1410
4.0  920
5.0  920
10.0  607
0.0  507
5.0  555
12.0  637
5.0  304
6.0  207
8.0  238
3.0  200
14.0  179
12.0  176
13.0  188
@type: local
Completed iteration 8
1.0  1.044
2.0  1410
4.0  921
7.0  655
10.0  680
8.0  507
9.0  557
11.0  488
5.0  392
6.0  274
8.0  237
2.0  180
12.0  180
14.0  186
12.0  150
13.0  200
@type: local
Completed iteration 9
1.0  1.044
2.0  1417
4.0  927
7.0  635
10.0  680
8.0  588
9.0  556
11.0  489
5.0  284
6.0  277
14.0  189
5.0  176
12.0  174
13.0  188

```

[illegible]

```
poly(gamma=1.0 / n_features)
```

```
In [ ]: n_features = oe_X_train25.shape[1]
spectral_clustering(
    oe_X_train25,
    cut=21,
    frame="one-hot_encoded_spectral_clustering",
    affinity="poly",
    gamma=1.0 / n_features,
)
```

```

17.0 187
17.0 182
21.0 89
14.0 55
18.0 41
11.0 34
10.0 56
dtype: int64
Completed iteration 7
1.0 6283
4.0 541
20.0 698
20.0 684
6.0 455
12.0 456
1.0 380
16.0 276
9.0 170
22.0 253
8.0 255
2.0 205
21.0 251
6.0 266
1.0 289
15.0 135
17.0 187
7.0 182
1.0 89
14.0 59
18.0 39
11.0 34
10.0 56
dtype: int64
Completed iteration 8
1.0 6580
4.0 540
20.0 691
20.0 686
6.0 457
12.0 456
1.0 380
9.0 170
22.0 251
2.0 205
11.0 288
6.0 261
5.0 151
21.0 251
17.0 187
6.0 389
21.0 89
18.0 58
14.0 38
11.0 34
10.0 56
dtype: int64
Completed iteration 9
1.0 6580
4.0 541
20.0 691
20.0 687
6.0 458
12.0 456
16.0 384
1.0 380
9.0 170
9.0 155
22.0 349
1.0 138
13.0 279
6.0 258
5.0 191
15.0 162
17.0 187
1.0 389
21.0 89
14.0 58
18.0 38
11.0 34
10.0 56
dtype: int64
Completed iteration 10
1.0 6580
4.0 541

```

```

signalval: [5.9588667e-01 9.9075487e-01 5.9588666e-01 5.9542415e-01 9.9035758e-01
 9.9010100e-01 9.9010100e-01 9.9080490e-01 9.8987122e-01 9.9087890e-01
 9.9010100e-01 9.7015728e-01 4.2087217e-01 5.13842057e-16 2.1138556e-18]
Finished running k-means++
1.0  8254
1.0  7901
8.0  895
7.0  476
2.0  565
6.0  127
11.0  121
8.0  845
8.0  436
8.0  241
11.0  224
10.0  117
12.0  180
14.0  11
dtype: int64
Completed iteration 1
1.0  8212
1.0  7708
7.0  837
11.0  680
8.0  564
2.0  560
8.0  560
8.0  463
8.0  840
8.0  426
9.0  990
11.0  224
10.0  118
12.0  180
14.0  41
dtype: int64
Completed iteration 2
1.0  8212
1.0  7706
11.0  622
8.0  567
2.0  542
8.0  528
6.0  182
8.0  450
5.0  254
5.0  254
6.0  114
11.0  221
10.0  187
12.0  180
8.0  86
dtype: int64
Completed iteration 3
1.0  80712
1.0  741
11.0  640
7.0  787
9.0  582
2.0  535
8.0  518
6.0  583
8.0  441
5.0  254
11.0  213
11.0  210
14.0  207
10.0  187
12.0  180
dtype: int64
Completed iteration 4
1.0  81208
1.0  743
11.0  654
8.0  583
9.0  583
2.0  532
8.0  518
8.0  439
9.0  394
4.0  158
4.0  158
11.0  215
14.0  119
10.0  180
dtype: int64

```

```

#type: int64
Completed iteration 18
3.0 6522
5.0 724
13.0 623
9.0 680
2.0 534
7.0 513
6.0 522
1.0 581
8.0 436
4.0 438
5.0 284
11.0 280
14.0 138
12.0 98
16.0 89
#type: int64
Completed iteration 19
3.0 6522
5.0 725
13.0 621
9.0 683
2.0 513
7.0 512
6.0 522
4.0 581
8.0 435
5.0 439
11.0 285
14.0 134
12.0 98
16.0 89
#type: int64
Completed iteration 20
3.0 6522
5.0 724
13.0 620
9.0 685
2.0 511
7.0 527
6.0 582
4.0 433
5.0 438
11.0 283
14.0 284
12.0 98
16.0 89
#type: int64
Completed iteration 21
3.0 6522
5.0 724
13.0 620
9.0 685
2.0 524
7.0 522
6.0 580
4.0 429
5.0 434
11.0 288
14.0 223
11.0 280
12.0 99
16.0 87
#type: int64
Completed iteration 22
3.0 6522
5.0 724
13.0 620
9.0 687
2.0 513
6.0 522
1.0 581
8.0 436
4.0 438
5.0 287
11.0 281
14.0 138
12.0 98
16.0 87
#type: int64
Completed iteration 23
3.0 6522
5.0 724
13.0 620
9.0 687
2.0 513
6.0 522
1.0 581
8.0 436
4.0 438
5.0 287
11.0 281
14.0 138
12.0 98
16.0 87

```

[illegible]

```
poly (gamma=0.1)

In [ ]: spectral_clustering(
        on_x_train=80,
        cuts=23,
        frame="one-hot_encoded_spectral_clustering",
        affinity="poly",
        gamma=0.1,
    )
```

```

5.0 488
6.0 486
2.0 430
11.0 412
4.0 413
13.0 386
22.0 340
7.0 344
23.0 332
9.0 283
6.0 263
10.0 238
1.0 175
11.0 111
15.0 104
15.0 97
20.0 92
20.0 89
10.0 88
10.0 87
5.0 28
4.0 26
#layer: int64
Completed iteration 11
1.0 6303
12.0 117
5.0 488
6.0 486
2.0 444
4.0 413
11.0 386
22.0 340
7.0 344
23.0 335
9.0 286
5.0 263
10.0 233
1.0 175
11.0 111
15.0 104
15.0 98
10.0 88
10.0 87
10.0 87
10.0 87
10.0 87
#layer: int64
Completed iteration 12
1.0 6303
12.0 117
5.0 488
6.0 486
2.0 446
4.0 414
11.0 386
22.0 338
7.0 339
23.0 333
6.0 287
10.0 263
10.0 237
5.0 237
1.0 179
12.0 111
17.0 104
15.0 99
10.0 89
10.0 88
10.0 87
10.0 87
10.0 87
#layer: int64
Completed iteration 13
1.0 6303
12.0 111
5.0 488
6.0 487
2.0 447
4.0 413
11.0 385
13.0 385
21.0 359
7.0 338
10.0 334
9.0 285
5.0 262

```

[illegible]

```

4.0    15
5.0    15
Completed iteration 5
6.0    80
6.0    753
6.0    753
2.0    456
2.0    787
12.0    341
12.0    341
1.0    163
1.0    163
14.0    15
14.0    15
Completed iteration 6
3.0    6320
3.0    6320
4.0    796
4.0    796
5.0    807
5.0    807
2.0    432
2.0    432
12.0    363
12.0    363
13.0    15
13.0    150
13.0    150
14.0    15
14.0    15
dtype: int64
Completed iteration 7
3.0    6320
3.0    6320
4.0    796
4.0    796
5.0    807
5.0    807
2.0    432
2.0    432
12.0    399
12.0    399
13.0    150
13.0    150
dtype: int64
Completed iteration 8
3.0    6320
3.0    6320
4.0    796
4.0    796
5.0    807
5.0    807
2.0    439
2.0    439
12.0    399
12.0    399
13.0    152
13.0    152
dtype: int64
Completed iteration 9
3.0    6320
3.0    6320
4.0    796
4.0    796
5.0    807
5.0    807
2.0    432
2.0    432
12.0    382
12.0    382
13.0    174
13.0    174
14.0    15
14.0    15
dtype: int64

```

```
poly (gamma=0.15)

In [ ]: spectral_clustering(
        ox_X_train025,
        cuts=23,
        frame="one-hot_encoded_spectral_clustering",
        affinity="poly",
        gamma=0.15,
)
```

```

2.0 432
2.2 380
15.0 359
15.2 355
15.8 365
16.0 343
16.2 341
16.4 340
16.6 155
16.8 280
17.0 280
17.2 155
17.4 155
17.6 155
17.8 155
18.0 155
dtype: int64
Iteration 4
16.0 6503
16.2 6503
16.4 6503
16.6 6503
16.8 6503
17.0 6503
17.2 6503
17.4 6503
17.6 6503
17.8 6503
18.0 6503
dtype: int64
Iteration 5
16.0 6503
16.2 6503
16.4 6503
16.6 6503
16.8 6503
17.0 6503
17.2 6503
17.4 6503
17.6 6503
17.8 6503
18.0 6503
dtype: int64
Iteration 6
16.0 526
16.2 526
16.4 526
16.6 526
16.8 526
17.0 526
17.2 526
17.4 526
17.6 526
17.8 526
18.0 526
dtype: int64

```

```

0.0 154
1.0 349
5.0 92
10.0 51
15.0 41
20.0 31
25.0 5
dnpwr initia
Completed iteration 7
4.0 650
7.0 523
5.0 489
1.0 468
1.0 423
2.0 383
3.0 377
10.0 376
20.0 320
17.0 280
6.0 252
23.0 251
11.0 201
10.0 226
10.0 179
6.0 151
10.0 182
6.0 151
10.0 74
8.0 62
10.0 23
21.0 5
dnpwr initia
Completed iteration 8
4.0 680
7.0 512
5.0 489
11.0 483
12.0 389
12.0 389
1.0 382
11.0 375
22.0 324
20.0 289
17.0 296
8.0 284
10.0 228
11.0 211
10.0 211
10.0 161
6.0 151
9.0 160
9.0 92
10.0 74
8.0 61
10.0 23
21.0 5
dnpwr initia
Completed iteration 9
4.0 650
7.0 529
5.0 489
1.0 479
11.0 477
11.0 421
3.0 385
3.0 385
10.0 374
10.0 341
20.0 301
17.0 298
8.0 284
10.0 232
11.0 198
10.0 166
6.0 151
10.0 99
10.0 74
10.0 61
10.0 23
21.0 5
dnpwr initia
Completed iteration 10
4.0 650
7.0 514

```

13.0	246
15.0	109
16.0	18
6.0	151
18.0	18
9.0	92
8.0	61
11.0	73
21.0	5
dtype: int64	
Completed iteration 14	
13.0	76
15.0	504
16.0	2
6.0	485
18.0	18
9.0	300
8.0	2
11.0	373
21.0	5
dtype: int64	
Completed iteration 15	
13.0	76
15.0	273
16.0	18
6.0	353
18.0	18
9.0	199
8.0	61
11.0	199
21.0	5
dtype: int64	
Completed iteration 16	
13.0	76
15.0	276
16.0	18
6.0	353
18.0	173
9.0	199
8.0	61
11.0	199
21.0	5
dtype: int64	
Completed iteration 17	
13.0	76
15.0	276
16.0	18
6.0	353
18.0	173
9.0	199
8.0	61
11.0	199
21.0	5
dtype: int64	
Completed iteration 18	
13.0	76
15.0	276
16.0	18
6.0	353
18.0	173
9.0	199
8.0	61
11.0	199
21.0	5
dtype: int64	

[illegible]

[illegible]

11.4	108
9.5	120
11.9	127
12.9	129
11.9	206
13.4	209
2.9	262
10.2	263
8.8	265
8.8	271
10.9	272
drgn test	
Completed iteration 14	
14.9	145
4.8	157
10.9	160
12.9	161
9.5	167
14.9	241
15.4	242
22.9	243
22.9	245
15.4	257
8.8	265
8.8	266
11.9	268
11.9	269
11.9	272
11.9	273
11.9	274
11.9	275
11.9	276
11.9	277
11.9	278
11.9	279
11.9	280
drgn test	
Completed iteration 15	
14.9	145
8.8	146
8.8	146
10.9	147
11.9	452
7.9	460
22.9	565
22.9	565
22.9	565
22.9	565
15.4	128
15.4	128
8.8	131
10.9	132
11.9	133
11.9	134
11.9	135
11.9	136
11.9	137
11.9	138
11.9	139
11.9	140
11.9	141
11.9	142
11.9	143
11.9	144
11.9	145
11.9	146
11.9	147
11.9	148
11.9	149
11.9	150
11.9	151
11.9	152
11.9	153
11.9	154
11.9	155
11.9	156
11.9	157
11.9	158
11.9	159
11.9	160
11.9	161
11.9	162
11.9	163
11.9	164
11.9	165
11.9	166
11.9	167
11.9	168
11.9	169
11.9	170
11.9	171
11.9	172
11.9	173
11.9	174
11.9	175
11.9	176
11.9	177
11.9	178
11.9	179
11.9	180
11.9	181
11.9	182
11.9	183
11.9	184
11.9	185
11.9	186
11.9	187
11.9	188
11.9	189
11.9	190
11.9	191
11.9	192
11.9	193
11.9	194
11.9	195
11.9	196
11.9	197
11.9	198
11.9	199
11.9	200
11.9	201
11.9	202
11.9	203
11.9	204
11.9	205
11.9	206
11.9	207
11.9	208
11.9	209
11.9	210
11.9	211
11.9	212
11.9	213
11.9	214
11.9	215
11.9	216
11.9	217
11.9	218
11.9	219
11.9	220
11.9	221
11.9	222
11.9	223
11.9	224
11.9	225
11.9	226
11.9	227
11.9	228
11.9	229
11.9	230
11.9	231
11.9	232
11.9	233
11.9	234
11.9	235
11.9	236
11.9	237
11.9	238
11.9	239
11.9	240
11.9	241
11.9	242
11.9	243
11.9	244
11.9	245
11.9	246
11.9	247
11.9	248
11.9	249
11.9	250
11.9	251
11.9	252
11.9	253
11.9	254
11.9	255
11.9	256
11.9	257
11.9	258
11.9	259
11.9	260
11.9	261
11.9	262
11.9	263
11.9	264
11.9	265
11.9	266
11.9	267
11.9	268
11.9	269
11.9	270
11.9	271
11.9	272
11.9	273
11.9	274
11.9	275
11.9	276
11.9	

```
eigenvalues: [9.0738734e-20 5.0641458e-20 2.9820843e-20 2.9043849e-20 2.2551350e-20
1.1376185e-20 8.9454995e-21 1.0490562e-21 4.2276650e-22 1.2666428e-22
0.0000000e+00 1.1000000e+00 1.1000000e+00 1.1000000e+00 1.1000000e+00 1.1000000e+00
1.1000000e+00 1.1000000e+00 1.1000000e+00 1.1000000e+00 1.1000000e+00]
```

```

0.0013778 -0.0027325 -0.0011169 -0.0058504 -0.170113 -0.0062367
0.0018478 -0.0017319 -0.0011009 -0.0000000 -0.0045371 -0.0011038
-0.0020194 -0.1017177 -0.0000000 -0.0015164 -0.0015764 -0.0017012
Carpenter 17 [-0.0041638 -0.0010000 -0.0017808 -0.0000000 -0.0019172
-0.0002783 -0.0041739 -0.0014526 -0.0000000 -0.0021619 -0.0012867
-0.0041611 -0.0000000 -0.0018262 -0.0004574 -0.0018905 -0.0021117
-0.0008887 -0.0005511 -0.0015564 -0.1717708 -0.0015557]
Carpenter 18 [-1.1709806e-06 -7.4237708e-02 0.0018819e-01 1.8758656e-04
-1.9882804e-01 -1.4970721e-02 1.3862464e-02 -2.7703326e-02
1.8713349e-02 7.2137476e-02 7.2168781e-03 1.0101074e-01
1.8713349e-02 1.2317476e-01 1.1018121e-02 2.7161516e-02
1.2228636e-02 1.1518879e-02 1.5571316e-03 -5.1259442e-02
1.5905131e-02 1.0467761e-02 1.1597866e-04]
Carpenter 19 [-1.2131018e-02 1.3907765e-01 0.0000000e-01 -1.2137959e-01
-2.0088154e-02 -7.7684781e-02 2.8703164e-01 -1.5687747e-02
2.4300606e-02 1.4177781e-01 7.7787781e-02 1.5482274e-02
1.2131018e-02 1.3907761e-02 2.7767464e-01 1.3820712e-02
1.2131018e-02 1.3907761e-02 1.3820712e-02 1.3820712e-02
2.8703164e-01 2.4300606e-02 1.4177781e-01 1.5482274e-02
2.8703164e-01 2.4300606e-02 1.4177781e-01 1.5482274e-02]
Carpenter 20 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 21 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 22 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 23 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 24 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 25 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 26 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 27 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
1.3549684e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.0016977e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01]
Carpenter 28 [-0.0011018e-01 1.3549684e-01 1.3549684e-01 1.3549684e-01
-1.1308130e-02 -0.1688131e-01 5.1511616e-02 2.7604940e-01
1.0016977e-01 0.0011018e-01 1.3549684e-01 2.7604940e-01
1.3549684e-01 
```

```
Completed iteration 15
0.0 4149
7.0 2364
```

Complex: ins044	
Complex: ins044	18
0.0	0.0
7.0	2.00
10.0	3.00
13.0	4.00
16.0	5.00
19.0	6.00
22.0	7.00
25.0	8.00
28.0	9.00
31.0	10.00
34.0	11.00
37.0	12.00
40.0	13.00
43.0	14.00
46.0	15.00
49.0	16.00
52.0	17.00
55.0	18.00
Complex: ins045	
Complex: ins045	11
0.0	0.0
8.0	0.00
16.0	0.00
24.0	0.00
32.0	0.00
40.0	0.00
48.0	0.00
56.0	0.00
64.0	0.00
72.0	0.00
80.0	0.00
88.0	0.00
96.0	0.00
104.0	0.00
112.0	0.00
120.0	0.00
128.0	0.00
136.0	0.00
144.0	0.00
152.0	0.00
160.0	0.00
168.0	0.00
176.0	0.00
184.0	0.00
192.0	0.00
200.0	0.00
208.0	0.00
216.0	0.00
224.0	0.00
232.0	0.00
240.0	0.00
248.0	0.00
256.0	0.00
264.0	0.00
272.0	0.00
280.0	0.00
288.0	0.00
296.0	0.00
304.0	0.00
312.0	0.00
320.0	0.00
328.0	0.00
336.0	0.00
344.0	0.00
352.0	0.00
360.0	0.00
368.0	0.00
376.0	0.00
384.0	0.00
392.0	0.00
400.0	0.00
408.0	0.00
416.0	0.00
424.0	0.00
432.0	0.00
440.0	0.00
448.0	0.00
456.0	0.00
464.0	0.00
472.0	0.00
480.0	0.00
488.0	0.00
496.0	0.00
504.0	0.00
512.0	0.00
520.0	0.00
528.0	0.00
536.0	0.00
544.0	0.00
552.0	0.00
560.0	0.00
568.0	0.00
576.0	0.00
584.0	0.00
592.0	0.00
600.0	0.00
608.0	0.00
616.0	0.00
624.0	0.00
632.0	0.00
640.0	0.00
648.0	0.00
656.0	0.00
664.0	0.00
672.0	0.00
680.0	0.00
688.0	0.00
696.0	0.00
704.0	0.00
712.0	0.00
720.0	0.00
728.0	0.00
736.0	0.00
744.0	0.00
752.0	0.00
760.0	0.00
768.0	0.00
776.0	0.00
784.0	0.00
792.0	0.00
800.0	0.00
808.0	0.00
816.0	0.00
824.0	0.00
832.0	0.00
840.0	0.00
848.0	0.00
856.0	0.00
864.0	0.00
872.0	0.00
880.0	0.00
888.0	0.00
896.0	0.00
904.0	0.00
912.0	0.00
920.0	0.00
928.0	0.00
936.0	0.00
944.0	0.00
952.0	0.00
960.0	0.00
968.0	0.00
976.0	0.00
984.0	0.00
992.0	0.00
1000.0	0.00
1008.0	0.00
1016.0	0.00
1024.0	0.00
1032.0	0.00
1040.0	0.00
1048.0	0.00
1056.0	0.00
1064.0	0.00
1072.0	0.00
1080.0	0.00
1088.0	0.00
1096.0	0.00
1104.0	0.00
1112.0	0.00
1120.0	0.00
1128.0	0.00
1136.0	0.00
1144.0	0.00
1152.0	0.00
1160.0	0.00
1168.0	0.00
1176.0	0.00
1184.0	0.00
1192.0	0.00
1200.0	0.00
1208.0	0.00
1216.0	0.00
1224.0	0.00
1232.0	0.00
1240.0	0.00
1248.	

```
dtype: int64
Completed iteration 19
0.0      4148
7.0      2367
# 0      nan
```


[illegible]

Functions

```

        if len(nn[i]) >= min_pts:
            core.append(i)
    cluster = 0
    for x in core:
        if labelled[x] != -1:
            continue
        cluster += 1
        labelled[x] = cluster
        labelled, checked = density_connected(
            data, eps, x, cluster, labelled, nn, core, checked
        )
    noise = []
    border = []
    for i in range(data.shape[0]):
        if labelled[i] == -1:
            noise.append(i)
        elif i not in core:
            border.append(i)
    dlabels = pd.DataFrame(labelled)
    path_label = (
        f"/content/drive/My Drive/pattern/2/labels_{eps}_{min_pts}_{fname}_DBSCAN.csv"
    )
    with open(path_label, "w", encoding="utf-8-sig") as f:
        dlabels.to_csv(f)
    return core, noise, border, labelled, cluster

def density_connected(data, eps, x, k, labelled, nn, core, checked):
    iter_list = [x]
    checked[x] = 1
    while len(iter_list):
        x = iter_list.pop(0)
        for y in nn[x]:
            labelled[y] = k
            if y in core and not checked[y]:
                iter_list.append(y)
                checked[y] = 1
    return labelled, checked

```

<https://stats.stackexchange.com/questions/88872/a-routine-to-choose-eps-and-minpts-for-dbscan> <https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd>

Will use min_pts = 1 to cluster all points (no noise)

label encoded run

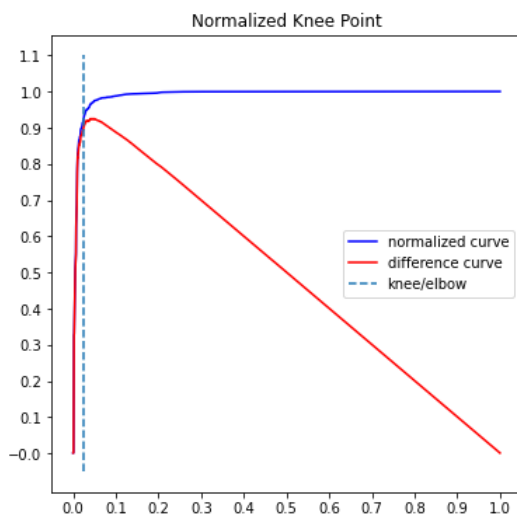
```

In [ ]: min_pts = le_X_train004.shape[1] * 2
        nbrs = NearestNeighbors(n_neighbors=min_pts).fit(le_X_train004.to_numpy())
        distances, indices = nbrs.kneighbors(le_X_train004.to_numpy())
        distance_desc = sorted(distances[:, min_pts - 1], reverse=True)[7:]
        px.line(x=list(range(1, len(distance_desc) + 1)), y=distance_desc)

```

```
In [ ]: kneedle = KneeLocator(
        range(1, len(distance_desc) + 1), # x values
        distance_desc, # y values
        S=1, # parameter suggested from paper
        curve="convex", # parameter from figure
        direction="decreasing",
    ) # parameter from figure
    kneedle.plot_knee_normalized()
    eps = round(kneedle.knee_y)
    print(eps)
```

3561



```
In [ ]: core, noise, border, labelled, clusters = DBSCAN(
        le_X_train004.to_numpy(), eps, 1, fname="label_encoded"
    )
```

```
In [ ]: print(f"noise indices: {noise}")
        print(f"number of clusters: {clusters}")
        print(f"Labels: {labelled}")
```

```
noise indices: []
number of clusters: 9
Labels: [1. 1. 1. ... 1. 1. 1.]
```

Attempt to decrease eps, to increase the numbers of clusters to closer to 11 (the number of labels, i.e. the real number of clusters)

```
In [ ]: print(len(np.unique(le_y_train004)))
```

11

```
In [ ]: core, noise, border, labelled, clusters = DBSCAN(  
        le_X_train004.to_numpy(), 2550, 1, fname="label_encoded"  
    )
```

```
In [ ]: print(f"noise indices: {noise}")  
        print(f"number of clusters: {clusters}")  
        print(f"Labels: {labelled}")
```

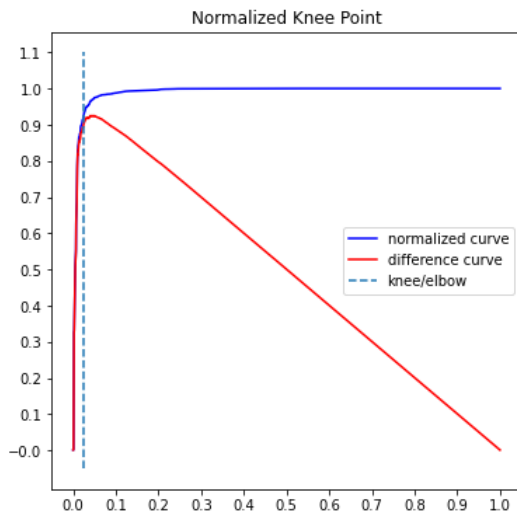
```
noise indices: []  
number of clusters: 11  
Labels: [1. 1. 1. ... 1. 1. 1.]
```

one-hot encoded run

```
In [ ]: min_pts = le_X_train004.shape[1] * 2  
        nbrs = NearestNeighbors(n_neighbors=min_pts).fit(oe_X_train004.to_numpy())  
        distances, indices = nbrs.kneighbors(oe_X_train004.to_numpy())  
        distance_desc = sorted(distances[:, min_pts - 1], reverse=True)[7:]  
        px.line(x=list(range(1, len(distance_desc) + 1)), y=distance_desc)
```

```
In [ ]: kneedle = KneedleLocator(  
        range(1, len(distance_desc) + 1), # x values  
        distance_desc, # y values  
        S=1, # parameter suggested from paper  
        curve="convex", # parameter from figure  
        direction="decreasing",  
    ) # parameter from figure  
    kneedle.plot_knee_normalized()  
    eps = round(kneedle.knee_y)  
    print(eps)
```

3561



```
In [ ]: core, noise, border, labelled, clusters = DBSCAN(
        oe_X_train004.to_numpy(), eps, 1, fname="one-hot_encoded"
    )
```

```
In [ ]: print(f"noise indices: {noise}")
        print(f"number of clusters: {clusters}")
        print(f"Labels: {labelled}")
```

```
noise indices: []
number of clusters: 9
Labels: [1. 1. 1. ... 1. 1. 1.]
```

Attempt to decrease eps, to increase the numbers of clusters to closer to 11 (the number of labels, i.e. the real number of clusters)

```
In [ ]: print(len(np.unique(le_y_train004)))
```

```
11
```

```
In [ ]: core, noise, border, labelled, clusters = DBSCAN(
        oe_X_train004.to_numpy(), 2550, 1, fname="one-hot_encoded"
    )
```

```
In [ ]: print(f"noise indices: {noise}")
        print(f"number of clusters: {clusters}")
        print(f"Labels: {labelled}")
```

```
noise indices: []
number of clusters: 11
Labels: [1. 1. 1. ... 1. 1. 1.]
```

Evaluation

Conditional entropy

```
In [ ]: def conditional_entropy(y_true, y_pred):
        n = len(y_true)
        y_true_labels = np.unique(y_true)
        y_pred_labels = np.unique(y_pred)
        # Calculate H(T|C)
        H_T_C_total = 0
        for pred_label in y_pred_labels:
            indices = np.where(y_pred == pred_label)[0]
            cluster = y_true[indices]
            H_T_C = 0
            for true_label in y_true_labels:
                p = np.mean(cluster == true_label)
                if p > 0:
                    H_T_C -= p * np.log2(p)
            H_T_C_total += len(cluster) * H_T_C / n
        return H_T_C_total
```

Precision, Recall, F1 score

```
In [ ]: def precision_recall_f1_weighted(y_true, y_pred):
        # Compute F1 score with average=weighted
```

```

n_classes = len(set(y_true))
precisions = []
recalls = []
f1_scores = []
for c in range(n_classes):
    weight = np.sum(y_true == c) / len(y_true)
    tp = np.sum((y_true == c) & (y_pred == c))
    fp = np.sum((y_true != c) & (y_pred == c))
    fn = np.sum((y_true == c) & (y_pred != c))
    precision = weight * tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = weight * tp / (tp + fn) if (tp + fn) > 0 else 0
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(
        2 * precision * recall / (precision + recall)
        if (precision + recall) > 0
        else 0
    )
print("Weighted:")
print(f"Precision: {sum(precisions)}")
print(f"Recall: {sum(recalls)}")
print(f"F1 score: {sum(f1_scores)}")
return sum(precisions), sum(recalls), sum(f1_scores)

```

Plotting Values

```

In [ ]: def plot_conditional_entropy_precision_recall_f1(
cond_entropies, precisions, recalls, f1_scores, x_axis, name
):
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18.5, 6)
    plt.rcParams.update({"font.size": 10})
    fig.suptitle(name)
    ax1.plot(x_axis, cond_entropies)
    ax1.set_xlabel("number of clusters")
    ax1.set_ylabel("conditional entropy")
    ax2.plot(x_axis, precisions, label="precision")
    ax2.plot(x_axis, recalls, label="recall")
    ax2.plot(x_axis, f1_scores, label="f1_score")
    ax2.set_xlabel("number of clusters")
    ax2.set_ylabel("score")
    ax2.legend()
    plt.show()

```

Maximum matching

```

In [ ]: def max_matching_hungarian(labels, ground_truth):
    n_labels = len(np.unique(labels))
    n_ground_truth = len(np.unique(ground_truth))
    cost_matrix = np.zeros((n_labels, n_ground_truth))
    for i in range(n_labels):
        for j in range(n_ground_truth):
            cost_matrix[i, j] = np.sum((labels == i) & (ground_truth == j))
    padded = np.zeros(2 * [max(cost_matrix.shape)])
    padded[0 : cost_matrix.shape[0], 0 : cost_matrix.shape[1]] = cost_matrix
    cost_matrix = padded
    row_ind, col_ind = linear_sum_assignment(cost_matrix, maximize=True)
    new_labels = np.zeros_like(labels)
    for i in range(len(row_ind)):
        new_labels[labels == row_ind[i]] = col_ind[i]
    print(f"Accuracy: {accuracy_score(ground_truth, new_labels)}")
    return new_labels

```

Purity matching

```

In [ ]: def purity_matching(y_true, y_pred):
    # get the unique classes
    max_count = max(len(np.unique(y_true)), len(np.unique(y_pred)))
    max_value = np.max(np.unique(y_true))
    potential_labels = set([*range(max_value + 1, max_value + max_count)])
    clusters = np.unique(y_pred)
    # create a dictionary to store the mapping
    mapping = {}
    mapped = []
    # Loop through the classes
    for c in clusters:
        # get the index of the class
        idx = np.where(y_pred == c)
        # get the most common label in the predicted labels
        sorted = np.argsort(np.bincount(y_true[idx])).tolist()

```

```

    for label in mapped:
        if label in sorted:
            sorted.remove(label)
    if len(sorted) == 0:
        label = potential_labels.pop()
    else:
        label = sorted[-1]
    # map the class to the label
    mapping[c] = label
    mapped.append(label)
# create a new List of Labels
for i in range(len(y_pred)):
    y_pred[i] = mapping[y_pred[i]]
print(f"Accuracy: {accuracy_score(y_true, y_pred)}")
return y_pred

```

Labels and Centroids loading functions

```

In [ ]: def load_kmeans_labels(k=23, fname="label_encoded_kmeans"):
    path = "/content/drive/My Drive/pattern/2/"
    files = os.listdir(path)
    files = [file for file in files if file.startswith("labels")]
    candidates = []
    for file in files:
        # regex to get the file which will look like labels_{iter}_{k}_{fname}.csv
        if re.search(f"labels_[0-9]+_{k}_{fname}.csv", file):
            candidates.append(file)
    # get the file with max iter
    candidates.sort(key=lambda x: int(x.split("_")[1]))
    print(f>Loading: {candidates[-1]}")
    labels = pd.read_csv(path + candidates[-1], index_col=0)
    return labels

```

```

In [ ]: def load_kmeans_centroids(k=23, fname="spectral_clustering"):
    path = "/content/drive/My Drive/pattern/2/"
    files = os.listdir(path)
    files = [file for file in files if file.startswith("centroids")]
    candidates = []
    for file in files:
        # regex to get the file which will look like labels_{iter}_{k}_{fname}.csv
        if re.search(f"centroids_[0-9]+_{k}_{fname}.csv", file):
            candidates.append(file)
    # get the file with max iter
    candidates.sort(key=lambda x: int(x.split("_")[1]))
    print(candidates[-1])
    centroids = pd.read_csv(path + candidates[-1], index_col=0)
    return centroids

```

```

In [ ]: def load_dbscan_labels(fname=""):
    path = "/content/drive/My Drive/pattern/2/"
    files = os.listdir(path)
    files = [file for file in files if file.startswith("labels")]
    candidates = []
    for file in files:
        # regex to get the file which will look like labels_{fname}_DBSCAN.csv
        if re.search(f"labels_{fname}_DBSCAN.csv", file):
            candidates.append(file)
    # get the file with max iter
    candidates.sort(key=lambda x: int(x.split("_")[1]))
    print(f>Loading: {candidates[-1]}")
    labels = pd.read_csv(path + candidates[-1], index_col=0)
    return labels

```

KMeans

Using test data

Maximum matching

label encoded

```

In [ ]: precision_label_encoded_kmean_test = []
        recall_label_encoded_kmean_test = []
        f1_label_encoded_kmean_test = []
        con_entr_label_encoded_kmean_test = []

```

```

In [ ]: clusters = [7, 15, 23, 31, 45]
        for k in clusters:

```

```

centroids = load_kmeans_centroids(k=k, fname="label_encoded_kmeans")
centroids = centroids.to_numpy()
labels = np.array(k_means_predict(le_test.to_numpy(), centroids)).astype(np.int64)
print(f"k= {k}")
new_labels = max_matching_hungarian(labels, le_y_test)
prec, recall, f1 = precision_recall_f1_weighted(le_y_test, new_labels)
precision_label_encoded_kmean_test.append(prec)
recall_label_encoded_kmean_test.append(recall)
f1_label_encoded_kmean_test.append(f1)
con_entr = conditional_entropy(le_y_test, new_labels)
con_entr_label_encoded_kmean_test.append(con_entr)
print(f"Conditional entropy: {con_entr}")

```

```

centroids_9_7_label_encoded_kmeans.csv
k= 7
Accuracy: 0.5276163958987747
Weighted:
Precision: 0.47319717186600935
Recall: 0.5276163958987747
F1 score: 0.36452223064423
Conditional entropy: 2.0070435581419477
centroids_11_15_label_encoded_kmeans.csv
k= 15
Accuracy: 0.5333296895144826
Weighted:
Precision: 0.4817797139950501
Recall: 0.5333296895144826
F1 score: 0.37852807223197243
Conditional entropy: 1.9009496043577163
centroids_11_23_label_encoded_kmeans.csv
k= 23
Accuracy: 0.5761906446022718
Weighted:
Precision: 0.4992798123251458
Recall: 0.5761906446022718
F1 score: 0.46045814279959163
Conditional entropy: 1.76246569098425
centroids_11_31_label_encoded_kmeans.csv
k= 31
Accuracy: 0.7625462577444547
Weighted:
Precision: 0.8020043115539865
Recall: 0.7625462577444547
F1 score: 0.7192353472061755
Conditional entropy: 0.902351889216086
centroids_11_45_label_encoded_kmeans.csv
k= 45
Accuracy: 0.7624240826418115
Weighted:
Precision: 0.8104041422814647
Recall: 0.7624240826418116
F1 score: 0.7234448367188854
Conditional entropy: 0.8546426862318236

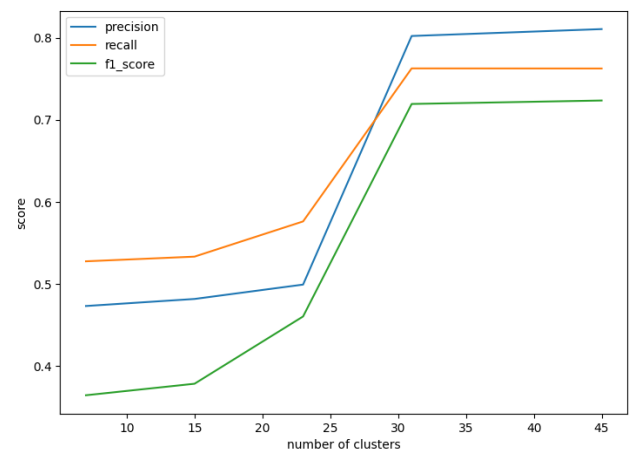
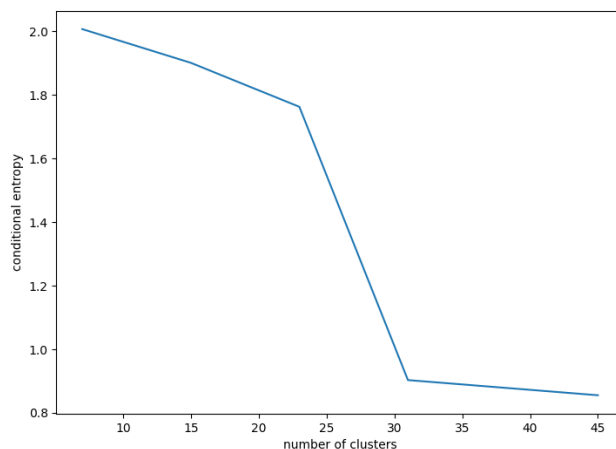
```

```

In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_label_encoded_kmean_test,
    precision_label_encoded_kmean_test,
    recall_label_encoded_kmean_test,
    f1_label_encoded_kmean_test,
    clusters,
    "kmeans label encoded",
)

```

kmeans label encoded



one-hot encoded

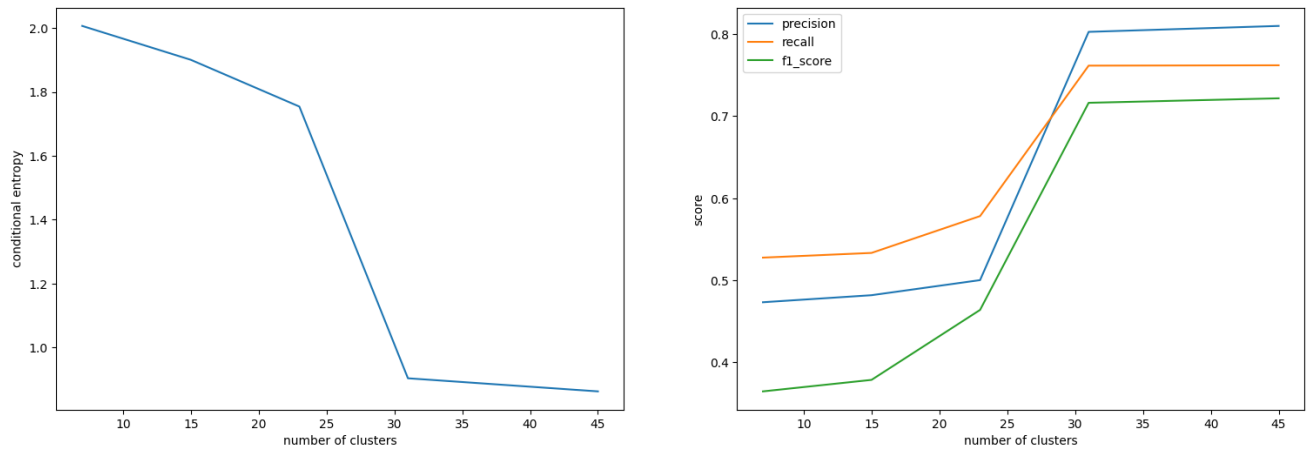
```
In [ ]: precision_one_hot_encoded_kmean_test = []
recall_one_hot_encoded_kmean_test = []
f1_one_hot_encoded_kmean_test = []
con_entr_one_hot_encoded_kmean_test = []
```

```
In [ ]: clusters = [7, 15, 23, 31, 45]
for k in clusters:
    centroids = load_kmeans_centroids(k=k, fname="one-hot_encoded_kmeans")
    centroids = centroids.to_numpy()
    labels = np.array(k_means_predict(oe_test.to_numpy(), centroids)).astype(np.int64)
    print(f"k= {k}")
    new_labels = max_matching_hungarian(labels, le_y_test)
    prec, recall, f1 = precision_recall_f1_weighted(le_y_test, new_labels)
    precision_one_hot_encoded_kmean_test.append(prec)
    recall_one_hot_encoded_kmean_test.append(recall)
    f1_one_hot_encoded_kmean_test.append(f1)
    con_entr = conditional_entropy(le_y_test, new_labels)
    con_entr_one_hot_encoded_kmean_test.append(con_entr)
    print(f"Conditional entropy: {con_entr}")
```

```
centroids_9_one-hot_encoded_kmeans.csv
k= 7
Accuracy: 0.5276163958987747
Weighted:
Precision: 0.47319717186600935
Recall: 0.5276163958987747
F1 score: 0.36452223064423
Conditional entropy: 2.0070435581419477
centroids_11_15_one-hot_encoded_kmeans.csv
k= 15
Accuracy: 0.5333296895144826
Weighted:
Precision: 0.4817797139950501
Recall: 0.5333296895144826
F1 score: 0.37852807223197243
Conditional entropy: 1.9009496043577163
centroids_11_23_one-hot_encoded_kmeans.csv
k= 23
Accuracy: 0.578245115407245
Weighted:
Precision: 0.500174839756438
Recall: 0.578245115407245
F1 score: 0.46385112099050596
Conditional entropy: 1.7545007116440827
centroids_11_31_one-hot_encoded_kmeans.csv
k= 31
Accuracy: 0.7616846017573924
Weighted:
Precision: 0.8028689770844921
Recall: 0.7616846017573925
F1 score: 0.716368351473006
Conditional entropy: 0.9033757482008209
centroids_11_45_one-hot_encoded_kmeans.csv
k= 45
Accuracy: 0.7620993540795231
Weighted:
Precision: 0.81008831658732
Recall: 0.7620993540795231
F1 score: 0.7218483150788502
Conditional entropy: 0.8623669343651951
```

```
In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_one_hot_encoded_kmean_test,
    precision_one_hot_encoded_kmean_test,
    recall_one_hot_encoded_kmean_test,
    f1_one_hot_encoded_kmean_test,
    clusters,
    "kmeans one-hot encoded",
)
```

kmeans one-hot encoded



Purity matching

label encoded

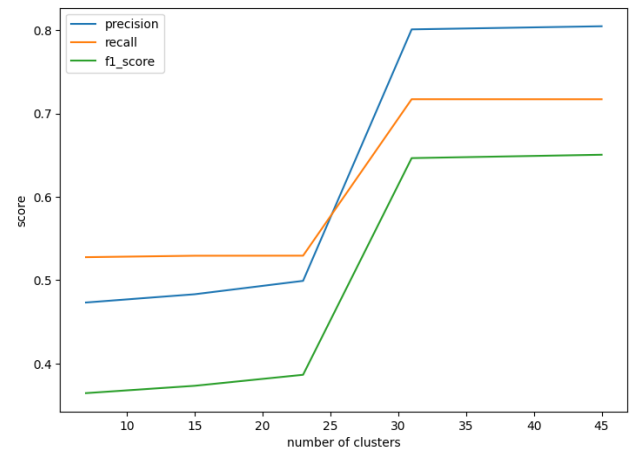
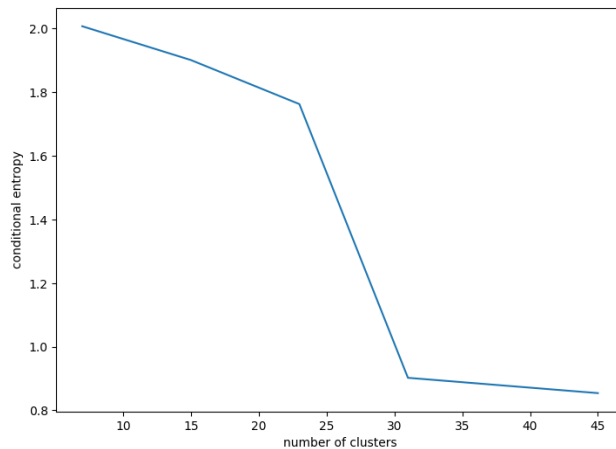
```
In [ ]: precision_label_encoded_kmean_test = []
recall_label_encoded_kmean_test = []
f1_label_encoded_kmean_test = []
con_entr_label_encoded_kmean_test = []

In [ ]: clusters = [7, 15, 23, 31, 45]
for k in clusters:
    centroids = load_kmeans_centroids(k=k, fname="label_encoded_kmeans")
    centroids = centroids.to_numpy()
    labels = np.array(k_means_predict(le_test.to_numpy(), centroids)).astype(np.int64)
    print(f"k= {k}")
    new_labels = purity_matching(le_y_test, labels)
    prec, recall, f1 = precision_recall_f1_weighted(le_y_test, new_labels)
    precision_label_encoded_kmean_test.append(prec)
    recall_label_encoded_kmean_test.append(recall)
    f1_label_encoded_kmean_test.append(f1)
    con_entr = conditional_entropy(le_y_test, new_labels)
    con_entr_label_encoded_kmean_test.append(con_entr)
    print(f"Conditional entropy: {con_entr}")
```

```
centroids_9_7_label_encoded_kmeans.csv
k= 7
Accuracy: 0.527587459690254
Weighted:
Precision: 0.47320020838171833
Recall: 0.527587459690254
F1 score: 0.36446106396427613
Conditional entropy: 2.0070435581419477
centroids_11_15_label_encoded_kmeans.csv
k= 15
Accuracy: 0.5293172019329387
Weighted:
Precision: 0.48309122521286463
Recall: 0.5293172019329387
F1 score: 0.37323147599919504
Conditional entropy: 1.9009496043577163
centroids_11_23_label_encoded_kmeans.csv
k= 23
Accuracy: 0.5294007954242209
Weighted:
Precision: 0.49918108687329216
Recall: 0.5294007954242209
F1 score: 0.38647268034960175
Conditional entropy: 1.76246569098425
centroids_11_31_label_encoded_kmeans.csv
k= 31
Accuracy: 0.7171646373810803
Weighted:
Precision: 0.8009624599703624
Recall: 0.7171646373810802
F1 score: 0.6464864518930077
Conditional entropy: 0.902351889216086
centroids_11_45_label_encoded_kmeans.csv
k= 45
Accuracy: 0.717119625501159
Weighted:
Precision: 0.8047199416082651
Recall: 0.717119625501159
F1 score: 0.6505372616708462
Conditional entropy: 0.8542425154845061
```

```
In [ ]: plot_conditional_entropy_precision_recall_f1(
        con_entr_label_encoded_kmean_test,
        precision_label_encoded_kmean_test,
        recall_label_encoded_kmean_test,
        f1_label_encoded_kmean_test,
        clusters,
        "kmeans label encoded",
    )
```

kmeans label encoded



one-hot encoded

```
In [ ]: precision_one_hot_encoded_kmean_test = []
        recall_one_hot_encoded_kmean_test = []
        f1_one_hot_encoded_kmean_test = []
        con_entr_one_hot_encoded_kmean_test = []
```

```
In [ ]: clusters = [7, 15, 23, 31, 45]
        for k in clusters:
            centroids = load_kmeans_centroids(k=k, fname="one-hot_encoded_kmeans")
            centroids = centroids.to_numpy()
            labels = np.array(k_means_predict(oe_test.to_numpy(), centroids)).astype(np.int64)
```

```

print(f"k= {k}")
new_labels = purity_matching(le_y_test, labels)
prec, recall, f1 = precision_recall_f1_weighted(le_y_test, new_labels)
precision_one_hot_encoded_kmean_test.append(prec)
recall_one_hot_encoded_kmean_test.append(recall)
f1_one_hot_encoded_kmean_test.append(f1)
con_entr = conditional_entropy(le_y_test, new_labels)
con_entr_one_hot_encoded_kmean_test.append(con_entr)
print(f"Conditional entropy: {con_entr}")

```

```

centroids_9_7_one-hot_encoded_kmeans.csv
k= 7
Accuracy: 0.527587459690254
Weighted:
Precision: 0.47320020838171833
Recall: 0.527587459690254
F1 score: 0.36446106396427613
Conditional entropy: 2.0070435581419477
centroids_11_15_one-hot_encoded_kmeans.csv
k= 15
Accuracy: 0.5293172019329387
Weighted:
Precision: 0.48309122521286463
Recall: 0.5293172019329387
F1 score: 0.37323147599919504
Conditional entropy: 1.9009496043577163
centroids_11_23_one-hot_encoded_kmeans.csv
k= 23
Accuracy: 0.5291628754874947
Weighted:
Precision: 0.49914064112653816
Recall: 0.5291628754874946
F1 score: 0.38620430665312133
Conditional entropy: 1.7545007116440827
centroids_11_31_one-hot_encoded_kmeans.csv
k= 31
Accuracy: 0.7165183953907835
Weighted:
Precision: 0.7964567934807149
Recall: 0.7165183953907835
F1 score: 0.6434367268723267
Conditional entropy: 0.9033757482008209
centroids_11_45_one-hot_encoded_kmeans.csv
k= 45
Accuracy: 0.7208234601918149
Weighted:
Precision: 0.8104277459463206
Recall: 0.7208234601918149
F1 score: 0.6543608888459882
Conditional entropy: 0.8592353595878717

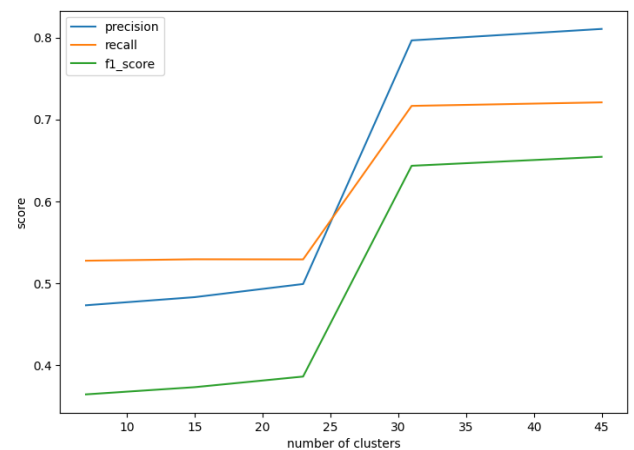
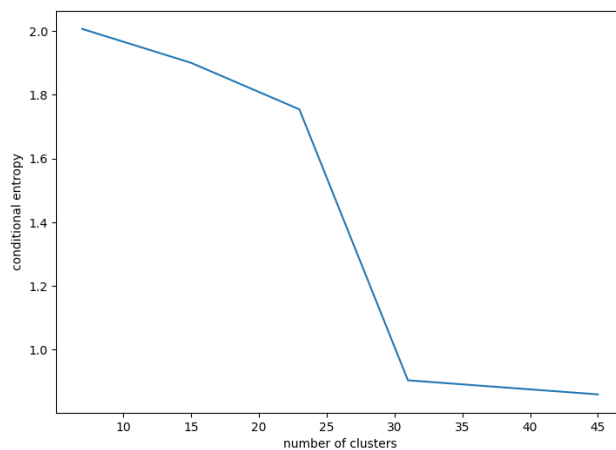
```

```

In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_one_hot_encoded_kmean_test,
    precision_one_hot_encoded_kmean_test,
    recall_one_hot_encoded_kmean_test,
    f1_one_hot_encoded_kmean_test,
    clusters,
    "kmeans one-hot encoded",
)

```

kmeans one-hot encoded



Using training data

Maximum matching

label encoded

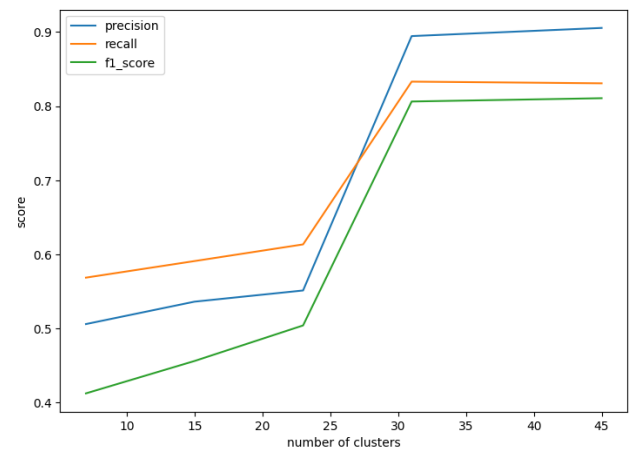
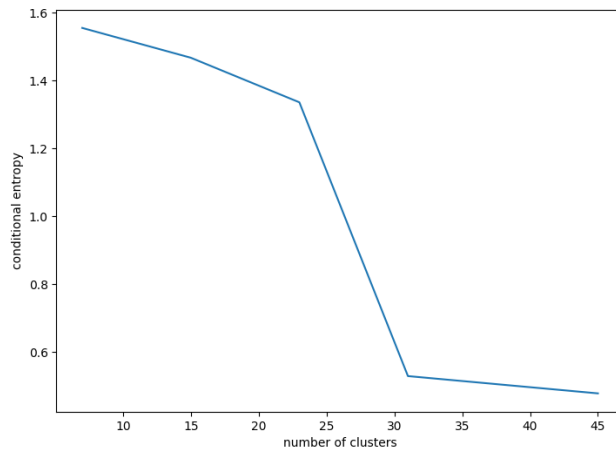
```
In [ ]: precision_label_encoded_kmean_train = []
recall_label_encoded_kmean_train = []
f1_label_encoded_kmean_train = []
con_entr_label_encoded_kmean_train = []
```

```
In [ ]: clusters = [7, 15, 23, 31, 45]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(k=k, fname="label_encoded_kmeans")
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train_10_percent)
    prec, recall, f1 = precision_recall_f1_weighted(le_y_train_10_percent, new_labels)
    precision_label_encoded_kmean_train.append(prec)
    recall_label_encoded_kmean_train.append(recall)
    f1_label_encoded_kmean_train.append(f1)
    con_entr = conditional_entropy(le_y_train_10_percent, new_labels)
    con_entr_label_encoded_kmean_train.append(con_entr)
    print(f"Conditional entropy: {con_entr}")
```

```
k= 7
Loading: labels_9_7_label_encoded_kmeans.csv
Accuracy: 0.5686074073774192
Weighted:
Precision: 0.5059549714702325
Recall: 0.5686074073774191
F1 score: 0.4124629551498325
Conditional entropy: 1.555616299708753
k= 15
Loading: labels_11_15_label_encoded_kmeans.csv
Accuracy: 0.5909505871207904
Weighted:
Precision: 0.5361831284498593
Recall: 0.5909505871207904
F1 score: 0.4561441952183603
Conditional entropy: 1.4677963604890603
k= 23
Loading: labels_11_23_label_encoded_kmeans.csv
Accuracy: 0.6134476064782671
Weighted:
Precision: 0.5512803173254756
Recall: 0.6134476064782671
F1 score: 0.504100225908116
Conditional entropy: 1.3363884292614905
k= 31
Loading: labels_11_31_label_encoded_kmeans.csv
Accuracy: 0.8330313893538939
Weighted:
Precision: 0.8944603993883612
Recall: 0.8330313893538938
F1 score: 0.8062192971874526
Conditional entropy: 0.5294019358718293
k= 45
Loading: labels_11_45_label_encoded_kmeans.csv
Accuracy: 0.8307703518676332
Weighted:
Precision: 0.9054694984696837
Recall: 0.8307703518676333
F1 score: 0.8106739927715784
Conditional entropy: 0.4784610408282507
```

```
In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_label_encoded_kmean_train,
    precision_label_encoded_kmean_train,
    recall_label_encoded_kmean_train,
    f1_label_encoded_kmean_train,
    clusters,
    "kmeans label encoded",
)
```

kmeans label encoded



one-hot encoded

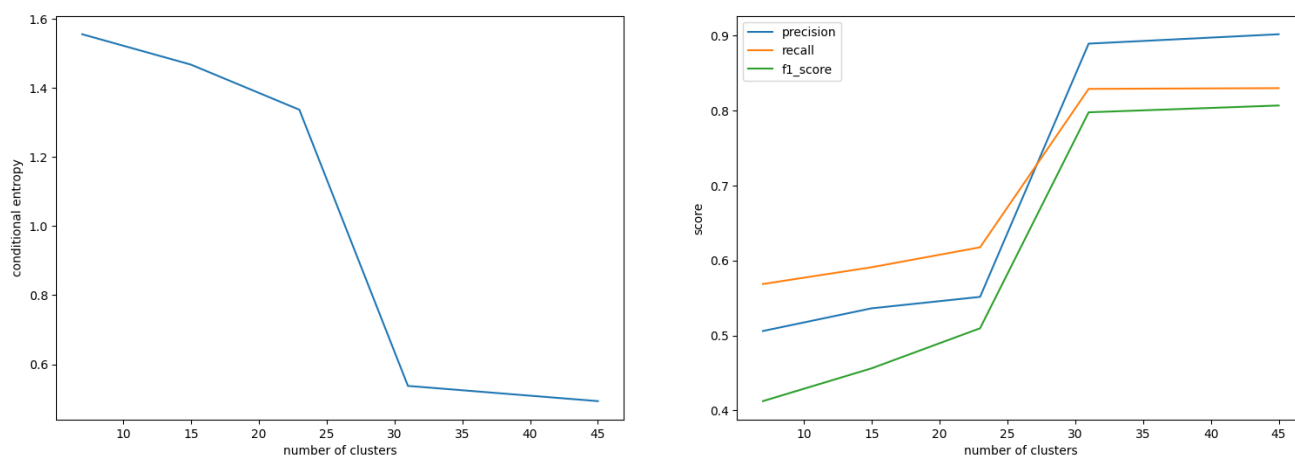
```
In [ ]: precision_one_hot_encoded_kmean_train = []
recall_one_hot_encoded_kmean_train = []
f1_one_hot_encoded_kmean_train = []
con_entr_one_hot_encoded_kmean_train = []
```

```
In [ ]: clusters = [7, 15, 23, 31, 45]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(k=k, fname="one-hot_encoded_kmeans")
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train_10_percent)
    prec, recall, f1 = precision_recall_f1_weighted(le_y_train_10_percent, new_labels)
    precision_one_hot_encoded_kmean_train.append(prec)
    recall_one_hot_encoded_kmean_train.append(recall)
    f1_one_hot_encoded_kmean_train.append(f1)
    con_entr = conditional_entropy(le_y_train_10_percent, new_labels)
    con_entr_one_hot_encoded_kmean_train.append(con_entr)
    print(f"Conditional entropy: {con_entr}")
```

```
k= 7
Loading: labels_9_7_one-hot_encoded_kmeans.csv
Accuracy: 0.5686074073774192
Weighted:
Precision: 0.5059549714702325
Recall: 0.5686074073774191
F1 score: 0.4124629551498325
Conditional entropy: 1.555616299708753
k= 15
Loading: labels_11_15_one-hot_encoded_kmeans.csv
Accuracy: 0.5909505871207904
Weighted:
Precision: 0.5361831284498593
Recall: 0.5909505871207904
F1 score: 0.4561441952183603
Conditional entropy: 1.4677963604890603
k= 23
Loading: labels_11_23_one-hot_encoded_kmeans.csv
Accuracy: 0.6175607919501398
Weighted:
Precision: 0.5514688631684094
Recall: 0.6175607919501397
F1 score: 0.5095895101013227
Conditional entropy: 1.3370603934626917
k= 31
Loading: labels_11_31_one-hot_encoded_kmeans.csv
Accuracy: 0.828845332485866
Weighted:
Precision: 0.8891547705865438
Recall: 0.828845332485866
F1 score: 0.7976041471956613
Conditional entropy: 0.5373994436677895
k= 45
Loading: labels_11_45_one-hot_encoded_kmeans.csv
Accuracy: 0.8297967090467814
Weighted:
Precision: 0.9016778000150121
Recall: 0.8297967090467815
F1 score: 0.8066530758068461
Conditional entropy: 0.49335297410466317
```

```
In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_one_hot_encoded_kmean_train,
    precision_one_hot_encoded_kmean_train,
    recall_one_hot_encoded_kmean_train,
    f1_one_hot_encoded_kmean_train,
    clusters,
    "kmeans one-hot encoded",
)
```

kmeans one-hot encoded



Purity matching

label encoded

```
In [ ]: precision_label_encoded_kmean_train = []
recall_label_encoded_kmean_train = []
f1_label_encoded_kmean_train = []
con_entr_label_encoded_kmean_train = []
```

```
In [ ]: clusters = [7, 15, 23, 31, 45]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(k=k, fname="label_encoded_kmeans")
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train_10_percent, labels)
    prec, recall, f1 = precision_recall_f1_weighted(le_y_train_10_percent, new_labels)
    precision_label_encoded_kmean_train.append(prec)
    recall_label_encoded_kmean_train.append(recall)
    f1_label_encoded_kmean_train.append(f1)
    con_entr = conditional_entropy(le_y_train_10_percent, new_labels)
    con_entr_label_encoded_kmean_train.append(con_entr)
    print(f"Conditional entropy: {con_entr}")
```

```

k= 7
Loading: labels_9_7_label_encoded_kmeans.csv
Accuracy: 0.5685790685011366
Weighted:
Precision: 0.5160682902572175
Recall: 0.5685790685011365
F1 score: 0.41239936092439017
Conditional entropy: 1.555616299708753
k= 15
Loading: labels_11_15_label_encoded_kmeans.csv
Accuracy: 0.5730221994611565
Weighted:
Precision: 0.5284208184267467
Recall: 0.5730221994611564
F1 score: 0.42328718311489655
Conditional entropy: 1.4677963604890603
k= 23
Loading: labels_11_23_label_encoded_kmeans.csv
Accuracy: 0.5729918363794252
Weighted:
Precision: 0.5544586055706119
Recall: 0.5729918363794251
F1 score: 0.43716685746181705
Conditional entropy: 1.3363884292614905
k= 31
Loading: labels_11_31_label_encoded_kmeans.csv
Accuracy: 0.7899360553498738
Weighted:
Precision: 0.8973673323059633
Recall: 0.7899360553498738
F1 score: 0.7357341141522582
Conditional entropy: 0.5294019358718293
k= 45
Loading: labels_11_45_label_encoded_kmeans.csv
Accuracy: 0.7899320069389762
Weighted:
Precision: 0.9093927020462017
Recall: 0.7899320069389764
F1 score: 0.7437991273308314
Conditional entropy: 0.47846104082825064

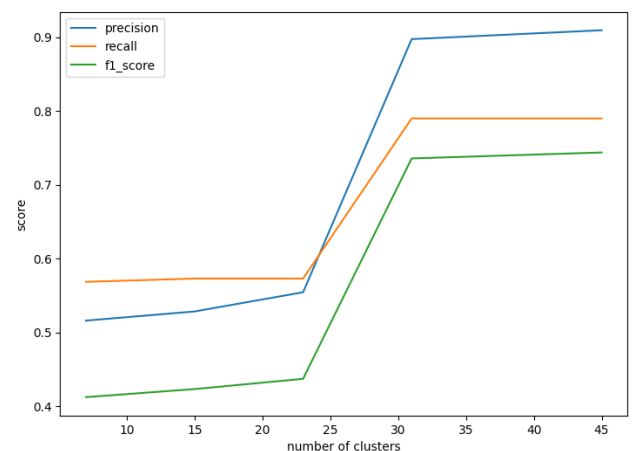
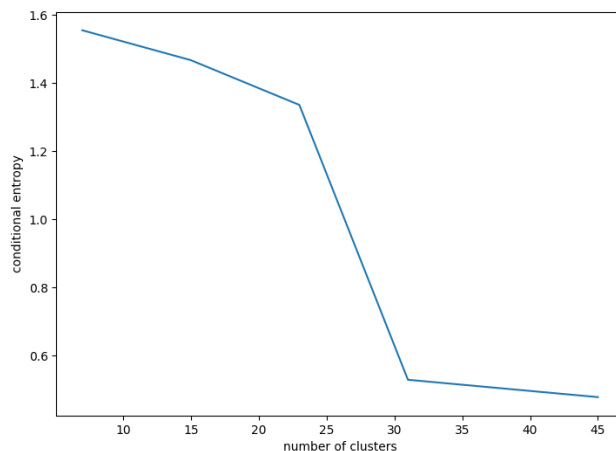
```

```

In [ ]: plot_conditional_entropy_precision_recall_f1(
        con_entr_label_encoded_kmean_train,
        precision_label_encoded_kmean_train,
        recall_label_encoded_kmean_train,
        f1_label_encoded_kmean_train,
        clusters,
        "kmeans label encoded",
    )

```

kmeans label encoded



one-hot encoded

```

In [ ]: precision_one_hot_encoded_kmean_train = []
        recall_one_hot_encoded_kmean_train = []
        f1_one_hot_encoded_kmean_train = []
        con_entr_one_hot_encoded_kmean_train = []

```

```

In [ ]: clusters = [7, 15, 23, 31, 45]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(k=k, fname="one-hot_encoded_kmeans")
            labels = labels.to_numpy().reshape(-1)

```



```

new_labels = purity_matching(le_y_train_10_percent, labels)
prec, recall, f1 = precision_recall_f1_weighted(le_y_train_10_percent, new_labels)
precision_one_hot_encoded_kmean_train.append(prec)
recall_one_hot_encoded_kmean_train.append(recall)
f1_one_hot_encoded_kmean_train.append(f1)
con_entr = conditional_entropy(le_y_train_10_percent, new_labels)
con_entr_one_hot_encoded_kmean_train.append(con_entr)
print(f"Conditional entropy: {con_entr}")

```

```

k= 7
Loading: labels_9_7_one-hot_encoded_kmeans.csv
Accuracy: 0.5685790685011366
Weighted:
Precision: 0.5160682902572175
Recall: 0.5685790685011365
F1 score: 0.41239936092439017
Conditional entropy: 1.555616299708753
k= 15
Loading: labels_11_15_one-hot_encoded_kmeans.csv
Accuracy: 0.5730221994611565
Weighted:
Precision: 0.5284208184267467
Recall: 0.5730221994611564
F1 score: 0.42328718311489655
Conditional entropy: 1.4677963604890603
k= 23
Loading: labels_11_23_one-hot_encoded_kmeans.csv
Accuracy: 0.5729938605848739
Weighted:
Precision: 0.5543930698412595
Recall: 0.5729938605848739
F1 score: 0.4371092179436616
Conditional entropy: 1.3370603934626917
k= 31
Loading: labels_11_31_one-hot_encoded_kmeans.csv
Accuracy: 0.2221342817410596
Weighted:
Precision: 0.34357635038690015
Recall: 0.22213428174105962
F1 score: 0.17347837751031353
Conditional entropy: 0.5373994436677894
k= 45
Loading: labels_11_45_one-hot_encoded_kmeans.csv
Accuracy: 0.2221201123029183
Weighted:
Precision: 0.3554060849167459
Recall: 0.22212011230291834
F1 score: 0.18201710789676623
Conditional entropy: 0.4933529741046632

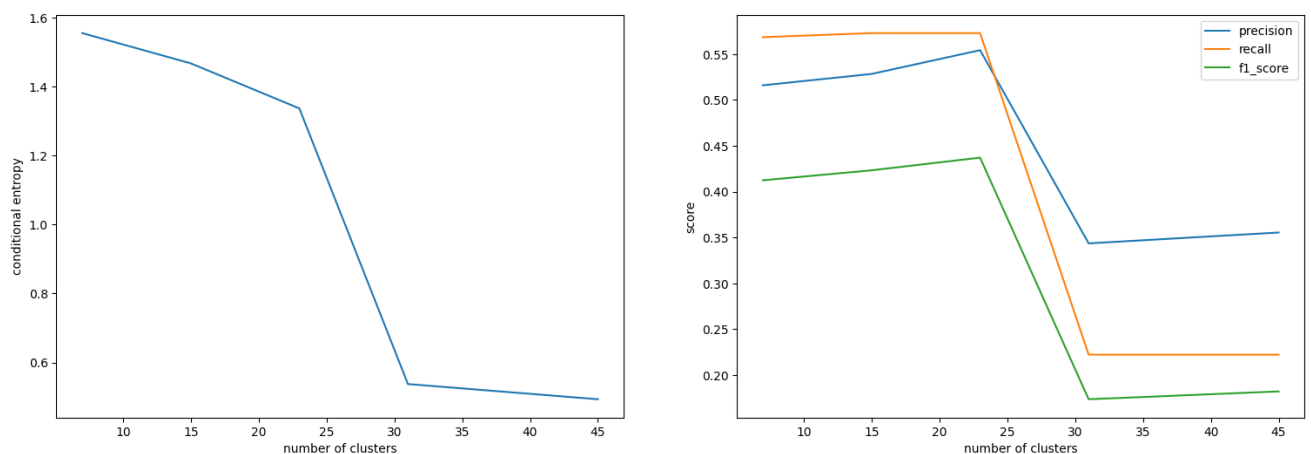
```

```

In [ ]: plot_conditional_entropy_precision_recall_f1(
    con_entr_one_hot_encoded_kmean_train,
    precision_one_hot_encoded_kmean_train,
    recall_one_hot_encoded_kmean_train,
    f1_one_hot_encoded_kmean_train,
    clusters,
    "kmeans one-hot encoded",
)

```

kmeans one-hot encoded



Comment

The difference between using label and one-hot encoding is trivial and they appear to produce approximately the same results.

Test data: Maximum matching yields better F1-score results than purity with a maximum of 0.723 for Maximum matching vs Purity matching's 0.654 for the testing data.

Training data: Maximum matching yields better F1-score results than purity with a maximum of 0.811 for Maximum matching vs Purity matching's 0.744 for the training data.

After a certain point, precision has the highest value of all the scores.

Conditional entropy is slightly improved by the use of label encoding for both training and testing data.

Training data evaluation shows overall better results, this may be due to the additional classes present in the testing data.

Spectral Clustering

Maximum matching

label encoded

additive chi2

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="label_encoded_spectral_clustering_additive_chi2_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.5491497975708503
Weighted:
Precision: 0.8767381352860629
Recall: 0.5491497975708503
F1 score: 0.6374081941031435
Conditional entropy: 0.5820853743177853
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.5363562753036437
Weighted:
Precision: 0.9235809009260006
Recall: 0.5363562753036437
F1 score: 0.6226551210552266
Conditional entropy: 0.42263953236723323
```

chi2 (gamma=1.0)

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(k=k, fname="label_encoded_spectral_clustering_chi2_1.0")
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.3848582995951417
Weighted:
Precision: 0.8315248030996434
Recall: 0.38485829959514173
F1 score: 0.3791162386375192
Conditional entropy: 0.9470902498506792
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.374331983805668
Weighted:
Precision: 0.8362056773414028
Recall: 0.37433198380566807
F1 score: 0.3672055851886507
Conditional entropy: 0.9093129020744071
```

cosine similarity

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="label_encoded_spectral_clustering_cosine_similarity_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = max_matching_hungarian(labels, le_y_train025)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_10_15_label_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.6248582995951417
Weighted:
Precision: 0.9711622706447117
Recall: 0.6248582995951416
F1 score: 0.7349806696003095
Conditional entropy: 0.1380135935608835
k= 23
Loading: labels_12_23_label_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.39619433198380566
Weighted:
Precision: 0.9737565019705356
Recall: 0.39619433198380566
F1 score: 0.5522230317554143
Conditional entropy: 0.12324804827245744
```

laplacian (gamma=1.0)

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="label_encoded_spectral_clustering_laplacian_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = max_matching_hungarian(labels, le_y_train025)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.27076923076923076
Weighted:
Precision: 0.7405166081551856
Recall: 0.2707692307692308
F1 score: 0.28942773003534544
Conditional entropy: 1.0246564104232805
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.20461538461538462
Weighted:
Precision: 0.3997138576911516
Recall: 0.20461538461538462
F1 score: 0.2680531384387752
Conditional entropy: 0.8885444356751017
```

linear

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="label_encoded_spectral_clustering_linear_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = max_matching_hungarian(labels, le_y_train025)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")
```

```

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_linear_1.0.csv
Accuracy: 0.6064777327935222
Weighted:
Precision: 0.9747396517969084
Recall: 0.6064777327935224
F1 score: 0.7046024623279346
Conditional entropy: 0.17849702652699273
k= 23
Loading: labels_20_23_label_encoded_spectral_clustering_linear_1.0.csv
Accuracy: 0.5807287449392713
Weighted:
Precision: 0.9816040958772438
Recall: 0.5807287449392712
F1 score: 0.6760960461018586
Conditional entropy: 0.15520461218506706

```

poly (gamma=1.0 / n_features)

```

In [ ]: n_features = le_X_train025.shape[1]
clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname=f"label_encoded_spectral_clustering_poly_{1.0 / n_features}"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_15_15_label_encoded_spectral_clustering_poly_0.024390243902439025.csv
Accuracy: 0.6527125506072875
Weighted:
Precision: 0.9832610543549498
Recall: 0.6527125506072875
F1 score: 0.7276710376353052
Conditional entropy: 0.20629991090827574
k= 23
Loading: labels_16_23_label_encoded_spectral_clustering_poly_0.024390243902439025.csv
Accuracy: 0.574412955465587
Weighted:
Precision: 0.9844401551937698
Recall: 0.574412955465587
F1 score: 0.6756018386715089
Conditional entropy: 0.13889966082365454

```

poly (gamma=0.1)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname=f"label_encoded_spectral_clustering_poly_0.1"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_10_15_label_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.6496356275303644
Weighted:
Precision: 0.9828892962172853
Recall: 0.6496356275303643
F1 score: 0.7250533180307432
Conditional entropy: 0.13504145816994298
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.6185425101214574
Weighted:
Precision: 0.9824505231787914
Recall: 0.6185425101214574
F1 score: 0.6853349952123226
Conditional entropy: 0.12787814358439453

```

poly (gamma=0.15)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")

```

```

labels = load_kmeans_labels(
    k=k, fname=f"label_encoded_spectral_clustering_poly_0.15"
)
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train025)
precision_recall_f1_weighted(le_y_train025, new_labels)
con_entr = conditional_entropy(le_y_train025, new_labels)
print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_8_15_label_encoded_spectral_clustering_poly_0.15.csv
 Accuracy: 0.6498785425101214
 Weighted:
 Precision: 0.9836365902102305
 Recall: 0.6498785425101216
 F1 score: 0.7237627886802386
 Conditional entropy: 0.1306486016007698
 k= 23
 Loading: labels_21_23_label_encoded_spectral_clustering_poly_0.15.csv
 Accuracy: 0.6364372469635627
 Weighted:
 Precision: 0.9831107518567752
 Recall: 0.6364372469635626
 F1 score: 0.7061668745871591
 Conditional entropy: 0.12146353446596238

rbf (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="label_encoded_spectral_clustering_rbf_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_21_15_label_encoded_spectral_clustering_rbf_1.0.csv
 Accuracy: 0.30955465587044534
 Weighted:
 Precision: 0.7457384287870625
 Recall: 0.3095546558704454
 F1 score: 0.31645830386869933
 Conditional entropy: 0.8955250817897876
 k= 23
 Loading: labels_21_23_label_encoded_spectral_clustering_rbf_1.0.csv
 Accuracy: 0.29724696356275304
 Weighted:
 Precision: 0.7560056043560033
 Recall: 0.2972469635627531
 F1 score: 0.2963822076829016
 Conditional entropy: 0.892133853382763

sigmoid (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="label_encoded_spectral_clustering_sigmoid_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_16_15_label_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.2625910931174089
Weighted:
Precision: 0.4140440493346744
Recall: 0.2625910931174089
F1 score: 0.31957192125738876
Conditional entropy: 1.546599244070808
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.1986234817813765
Weighted:
Precision: 0.4127001328453496
Recall: 0.19862348178137654
F1 score: 0.2605782173385421
Conditional entropy: 1.543728389893449

```

Comment

The best similarity matrix was cosine similarity, that had the best f1 score (weighted) of 0.73498, with one of the lowest conditional entropy values of 0.13801.

one-hot encoded

additive chi2

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_additive_chi2_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_17_15_one-hot_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.5638056680161944
Weighted:
Precision: 0.8830318403660401
Recall: 0.5638056680161944
F1 score: 0.6478879133395228
Conditional entropy: 0.5253176595543924
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.5305263157894737
Weighted:
Precision: 0.9452439046760479
Recall: 0.5305263157894736
F1 score: 0.6160615000705733
Conditional entropy: 0.36003875947225567

```

chi2 (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(k=k, fname="one-hot_encoded_spectral_clustering_chi2_1.0")
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_19_15_one-hot_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.37587044534412956
Weighted:
Precision: 0.8400786149751478
Recall: 0.3758704453441296
F1 score: 0.37982180033911583
Conditional entropy: 0.8769431602743892
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.36898785425101216
Weighted:
Precision: 0.8517834070514931
Recall: 0.36898785425101216
F1 score: 0.37208892282752715
Conditional entropy: 0.8315063932810333

```

cosine similarity

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_cosine_similarity_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_13_15_one-hot_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.44445344129554654
Weighted:
Precision: 0.9593494500297154
Recall: 0.4444534412955466
F1 score: 0.5990315782412331
Conditional entropy: 0.16599022533969648
k= 23
Loading: labels_17_23_one-hot_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.3654251012145749
Weighted:
Precision: 0.9593935519163186
Recall: 0.365425101214575
F1 score: 0.5155449619296383
Conditional entropy: 0.1639923786144153
```

laplacian (gamma=1.0)

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_laplacian_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_one-hot_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.1419433198380567
Weighted:
Precision: 0.6106299421287622
Recall: 0.1419433198380567
F1 score: 0.21948172080180317
Conditional entropy: 1.3372063045871108
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.34307692307692306
Weighted:
Precision: 0.810393175227923
Recall: 0.3430769230769231
F1 score: 0.347203854929599
Conditional entropy: 0.7508464852080923
```

linear

```
In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_linear_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")
```

```

k= 15
Loading: labels_14_15_one-hot_encoded_spectral_clustering_linear_1.0.csv
Accuracy: 0.5765182186234817
Weighted:
Precision: 0.9555319851246479
Recall: 0.5765182186234817
F1 score: 0.6814423980675436
Conditional entropy: 0.1785731850938518
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_linear_1.0.csv
Accuracy: 0.36145748987854254
Weighted:
Precision: 0.952682065184693
Recall: 0.36145748987854254
F1 score: 0.5064818771645001
Conditional entropy: 0.1325883166649188

```

poly (gamma=1.0 / n_features)

```

In [ ]: n_features = oe_X_train025.shape[1]
clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname=f"one-hot_encoded_spectral_clustering_poly_{1.0 / n_features}"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_21_15_one-hot_encoded_spectral_clustering_poly_0.008403361344537815.csv
Accuracy: 0.628421052631579
Weighted:
Precision: 0.9816596213220036
Recall: 0.628421052631579
F1 score: 0.6983642022329283
Conditional entropy: 0.13784581409333485
k= 23
Loading: labels_14_23_one-hot_encoded_spectral_clustering_poly_0.008403361344537815.csv
Accuracy: 0.6152226720647773
Weighted:
Precision: 0.9693587683052114
Recall: 0.6152226720647772
F1 score: 0.6835397553446224
Conditional entropy: 0.12834203346052694

```

poly (gamma=0.1)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname=f"one-hot_encoded_spectral_clustering_poly_0.1"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_11_15_one-hot_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.6391902834008097
Weighted:
Precision: 0.9825841466034744
Recall: 0.6391902834008096
F1 score: 0.7106517493372888
Conditional entropy: 0.1986373163728129
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.6247773279352227
Weighted:
Precision: 0.9837100948954508
Recall: 0.6247773279352227
F1 score: 0.6937376317753637
Conditional entropy: 0.11999720945584741

```

poly (gamma=0.15)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")

```



```

labels = load_kmeans_labels(
    k=k, fname=f"one-hot_encoded_spectral_clustering_poly_0.15"
)
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train025)
precision_recall_f1_weighted(le_y_train025, new_labels)
con_entr = conditional_entropy(le_y_train025, new_labels)
print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_21_15_one-hot_encoded_spectral_clustering_poly_0.15.csv
 Accuracy: 0.6452631578947369
 Weighted:
 Precision: 0.9803960051409684
 Recall: 0.6452631578947369
 F1 score: 0.7185462722872592
 Conditional entropy: 0.13585147538729814
 k= 23
 Loading: labels_17_23_one-hot_encoded_spectral_clustering_poly_0.15.csv
 Accuracy: 0.6162753036437247
 Weighted:
 Precision: 0.9844234427652382
 Recall: 0.6162753036437246
 F1 score: 0.681712892089806
 Conditional entropy: 0.1106168751926946

rbf (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_rbf_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_21_15_one-hot_encoded_spectral_clustering_rbf_1.0.csv
 Accuracy: 0.33757085020242916
 Weighted:
 Precision: 0.7874629354998931
 Recall: 0.3375708502024292
 F1 score: 0.33667316803296027
 Conditional entropy: 0.942106738140963
 k= 23
 Loading: labels_21_23_one-hot_encoded_spectral_clustering_rbf_1.0.csv
 Accuracy: 0.2623481781376518
 Weighted:
 Precision: 0.6224187419854702
 Recall: 0.2623481781376519
 F1 score: 0.36251487395719073
 Conditional entropy: 0.79320641322648

sigmoid (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_sigmoid_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = max_matching_hungarian(labels, le_y_train025)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_16_15_one-hot_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.2625910931174089
Weighted:
Precision: 0.4140440493346744
Recall: 0.2625910931174089
F1 score: 0.31957192125738876
Conditional entropy: 1.546599244070808
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.1986234817813765
Weighted:
Precision: 0.4127001328453496
Recall: 0.19862348178137654
F1 score: 0.2605782173385421
Conditional entropy: 1.543728389893449

```

Comment

The best similarity matrix was polynomial kernel (gamma=0.15), that had the best f1 score (weighted) of 0.71855, with the lowest conditional entropy of 0.13585.

Purity matching

label encoded

additive chi2

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="label_encoded_spectral_clustering_additive_chi2_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.1282591093117409
Weighted:
Precision: 0.8533002852519829
Recall: 0.12825910931174087
F1 score: 0.21272908943506372
Conditional entropy: 0.5820853743177854
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_additive_chi2_1.0.csv
Accuracy: 0.5022672064777328
Weighted:
Precision: 0.7611253663536747
Recall: 0.5022672064777329
F1 score: 0.5728425221543895
Conditional entropy: 0.4226395323672331

```

chi2 (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(k=k, fname="label_encoded_spectral_clustering_chi2_1.0")
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.2540890688259109
Weighted:
Precision: 0.693204097307214
Recall: 0.254089068825911
F1 score: 0.2106742493651458
Conditional entropy: 0.947090249850679
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.23668016194331984
Weighted:
Precision: 0.711644471665132
Recall: 0.23668016194331984
F1 score: 0.18203470552254825
Conditional entropy: 0.9093129020744072

```

cosine similarity

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="label_encoded_spectral_clustering_cosine_similarity_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_10_15_label_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.622753036437247
Weighted:
Precision: 0.9707729610762553
Recall: 0.622753036437247
F1 score: 0.7343336656338307
Conditional entropy: 0.13801359356088352
k= 23
Loading: labels_12_23_label_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.3751417004048583
Weighted:
Precision: 0.9817005429360948
Recall: 0.3751417004048583
F1 score: 0.5240778792632539
Conditional entropy: 0.12324804827245746

```

laplacian (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="label_encoded_spectral_clustering_laplacian_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.2699595141700405
Weighted:
Precision: 0.7413752204722548
Recall: 0.2699595141700405
F1 score: 0.2881982213017947
Conditional entropy: 1.0246564104232805
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.1173279352267207
Weighted:
Precision: 0.7261391212037215
Recall: 0.117327935226721
F1 score: 0.1633929102050621
Conditional entropy: 0.8885444356751018

```

linear

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(

```

```

        k=k, fname="label_encoded_spectral_clustering_linear_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_21_15_label_encoded_spectral_clustering_linear_1.0.csv
 Accuracy: 0.5808097165991902
 Weighted:
 Precision: 0.9766996716693849
 Recall: 0.5808097165991903
 F1 score: 0.6805509248865148
 Conditional entropy: 0.17849702652699273
 k= 23
 Loading: labels_20_23_label_encoded_spectral_clustering_linear_1.0.csv
 Accuracy: 0.5638866396761134
 Weighted:
 Precision: 0.8783012804158302
 Recall: 0.5638866396761134
 F1 score: 0.6511001335828434
 Conditional entropy: 0.15520461218506706

poly (gamma=1.0 / n_features)

```

In [ ]: n_features = le_X_train025.shape[1]
        clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"label_encoded_spectral_clustering_poly_{1.0 / n_features}"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_15_15_label_encoded_spectral_clustering_poly_0.024390243902439025.csv
 Accuracy: 0.0951417004048583
 Weighted:
 Precision: 0.48602796679047655
 Recall: 0.0951417004048583
 F1 score: 0.15215366989286952
 Conditional entropy: 0.20629991090827574
 k= 23
 Loading: labels_16_23_label_encoded_spectral_clustering_poly_0.024390243902439025.csv
 Accuracy: 0.5102834008097166
 Weighted:
 Precision: 0.9829622018275518
 Recall: 0.5102834008097166
 F1 score: 0.5890421069436199
 Conditional entropy: 0.1388996608236545

poly (gamma=0.1)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"label_encoded_spectral_clustering_poly_0.1"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_10_15_label_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.5962753036437247
Weighted:
Precision: 0.9769361525675471
Recall: 0.5962753036437247
F1 score: 0.6508120568758267
Conditional entropy: 0.13504145816994298
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.5979757085020243
Weighted:
Precision: 0.9833175860665935
Recall: 0.5979757085020242
F1 score: 0.6564551364417242
Conditional entropy: 0.12787814358439453

```

poly (gamma=0.15)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"label_encoded_spectral_clustering_poly_0.15"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_8_15_label_encoded_spectral_clustering_poly_0.15.csv
Accuracy: 0.5851821862348178
Weighted:
Precision: 0.8543875377280109
Recall: 0.5851821862348179
F1 score: 0.6342742859083861
Conditional entropy: 0.1306486016007698
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_poly_0.15.csv
Accuracy: 0.5840485829959514
Weighted:
Precision: 0.9826903974895195
Recall: 0.5840485829959513
F1 score: 0.6427407384983441
Conditional entropy: 0.12146353446596236

```

rbf (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="label_encoded_spectral_clustering_rbf_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_21_15_label_encoded_spectral_clustering_rbf_1.0.csv
Accuracy: 0.2867206477732793
Weighted:
Precision: 0.39481556585638056
Recall: 0.2867206477732794
F1 score: 0.3090347505522744
Conditional entropy: 0.8955250817897877
k= 23
Loading: labels_21_23_label_encoded_spectral_clustering_rbf_1.0.csv
Accuracy: 0.29578947368421055
Weighted:
Precision: 0.7361175022986304
Recall: 0.2957894736842106
F1 score: 0.293786408084183
Conditional entropy: 0.892133853382763

```

sigmoid (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(

```

```

        k=k, fname="label_encoded_spectral_clustering_sigmoid_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_16_15_label_encoded_spectral_clustering_sigmoid_1.0.csv
 Accuracy: 0.09578947368421052
 Weighted:
 Precision: 0.408077836550324
 Recall: 0.09578947368421054
 F1 score: 0.15132405807611082
 Conditional entropy: 1.546599244070808
 k= 23
 Loading: labels_21_23_label_encoded_spectral_clustering_sigmoid_1.0.csv
 Accuracy: 0.06979757085020243
 Weighted:
 Precision: 0.4061342059301873
 Recall: 0.06979757085020245
 F1 score: 0.11788547029849468
 Conditional entropy: 1.543728389893449

Comment

The best similarity matrix was cosine similarity, that had the best f1 score (weighted) of 0.73433, with one of the lowest conditional entropy values of 0.13801.

one-hot encoded

additive chi2

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="one-hot_encoded_spectral_clustering_additive_chi2_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_17_15_one-hot_encoded_spectral_clustering_additive_chi2_1.0.csv
 Accuracy: 0.0782995951417004
 Weighted:
 Precision: 0.730876689884327
 Recall: 0.07829959514170041
 F1 score: 0.13333405602772716
 Conditional entropy: 0.5253176595543924
 k= 23
 Loading: labels_21_23_one-hot_encoded_spectral_clustering_additive_chi2_1.0.csv
 Accuracy: 0.06736842105263158
 Weighted:
 Precision: 0.6970812226419958
 Recall: 0.06736842105263159
 F1 score: 0.11780609149961424
 Conditional entropy: 0.3600387594722557

chi2 (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(k=k, fname="one-hot_encoded_spectral_clustering_chi2_1.0")
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_19_15_one-hot_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.277085020242915
Weighted:
Precision: 0.677237356059211
Recall: 0.277085020242915
F1 score: 0.2482546442022301
Conditional entropy: 0.8769431602743892
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_chi2_1.0.csv
Accuracy: 0.3336842105263158
Weighted:
Precision: 0.8452159579322361
Recall: 0.33368421052631586
F1 score: 0.3100840555133353
Conditional entropy: 0.8315063932810333

```

cosine similarity

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_cosine_similarity_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_13_15_one-hot_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.36785425101214575
Weighted:
Precision: 0.7674589570711119
Recall: 0.36785425101214575
F1 score: 0.492352836501978
Conditional entropy: 0.16599022533969648
k= 23
Loading: labels_17_23_one-hot_encoded_spectral_clustering_cosine_similarity_1.0.csv
Accuracy: 0.3233198380566802
Weighted:
Precision: 0.9623729410352093
Recall: 0.32331983805668024
F1 score: 0.4549042046489629
Conditional entropy: 0.16399237861441526

```

laplacian (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(
        k=k, fname="one-hot_encoded_spectral_clustering_laplacian_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_one-hot_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.09182186234817814
Weighted:
Precision: 0.49068944222164873
Recall: 0.09182186234817813
F1 score: 0.14869097574357532
Conditional entropy: 1.3372063045871105
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_laplacian_1.0.csv
Accuracy: 0.23838056680161943
Weighted:
Precision: 0.7275282739058148
Recall: 0.23838056680161943
F1 score: 0.22976696671050265
Conditional entropy: 0.7508464852080924

```

linear

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
for k in clusters:
    print(f"k= {k}")
    labels = load_kmeans_labels(

```

```

        k=k, fname="one-hot_encoded_spectral_clustering_linear_1.0"
    )
    labels = labels.to_numpy().reshape(-1)
    new_labels = purity_matching(le_y_train025, labels)
    precision_recall_f1_weighted(le_y_train025, new_labels)
    con_entr = conditional_entropy(le_y_train025, new_labels)
    print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_14_15_one-hot_encoded_spectral_clustering_linear_1.0.csv
 Accuracy: 0.5089068825910931
 Weighted:
 Precision: 0.772908130587909
 Recall: 0.5089068825910931
 F1 score: 0.5828141773369007
 Conditional entropy: 0.17857318509385184
 k= 23
 Loading: labels_21_23_one-hot_encoded_spectral_clustering_linear_1.0.csv
 Accuracy: 0.3208906882591093
 Weighted:
 Precision: 0.9668322698796501
 Recall: 0.3208906882591093
 F1 score: 0.4517453980802243
 Conditional entropy: 0.1325883166649188

poly (gamma=1.0 / n_features)

```

In [ ]: n_features = oe_X_train025.shape[1]
        clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"one-hot_encoded_spectral_clustering_poly_{1.0 / n_features}"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```

k= 15
 Loading: labels_21_15_one-hot_encoded_spectral_clustering_poly_0.008403361344537815.csv
 Accuracy: 0.6166801619433199
 Weighted:
 Precision: 0.9822824581874219
 Recall: 0.6166801619433199
 F1 score: 0.6882318059064889
 Conditional entropy: 0.13784581409333485
 k= 23
 Loading: labels_14_23_one-hot_encoded_spectral_clustering_poly_0.008403361344537815.csv
 Accuracy: 0.5831578947368421
 Weighted:
 Precision: 0.9603966722941455
 Recall: 0.5831578947368421
 F1 score: 0.6342843937131989
 Conditional entropy: 0.12834203346052694

poly (gamma=0.1)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"one-hot_encoded_spectral_clustering_poly_0.1"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

```



```

k= 15
Loading: labels_11_15_one-hot_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.5935222672064777
Weighted:
Precision: 0.8267851092216479
Recall: 0.5935222672064777
F1 score: 0.6411929801794537
Conditional entropy: 0.1986373163728129
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_poly_0.1.csv
Accuracy: 0.6068016194331984
Weighted:
Precision: 0.9822185877612429
Recall: 0.6068016194331983
F1 score: 0.6741859571924104
Conditional entropy: 0.1199972094558474

```

poly (gamma=0.15)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname=f"one-hot_encoded_spectral_clustering_poly_0.15"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_one-hot_encoded_spectral_clustering_poly_0.15.csv
Accuracy: 0.5754655870445344
Weighted:
Precision: 0.9029755807672595
Recall: 0.5754655870445344
F1 score: 0.6183479873072703
Conditional entropy: 0.1358514753872981
k= 23
Loading: labels_17_23_one-hot_encoded_spectral_clustering_poly_0.15.csv
Accuracy: 0.591174089068826
Weighted:
Precision: 0.9818673002998712
Recall: 0.5911740890688258
F1 score: 0.653894879849089
Conditional entropy: 0.11061687519269459

```

rbf (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(
                k=k, fname="one-hot_encoded_spectral_clustering_rbf_1.0"
            )
            labels = labels.to_numpy().reshape(-1)
            new_labels = purity_matching(le_y_train025, labels)
            precision_recall_f1_weighted(le_y_train025, new_labels)
            con_entr = conditional_entropy(le_y_train025, new_labels)
            print(f"Conditional entropy: {con_entr}")

k= 15
Loading: labels_21_15_one-hot_encoded_spectral_clustering_rbf_1.0.csv
Accuracy: 0.22534412955465588
Weighted:
Precision: 0.6109351512520851
Recall: 0.2253441295546559
F1 score: 0.20255961563747474
Conditional entropy: 0.9421067381409629
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_rbf_1.0.csv
Accuracy: 0.16623481781376517
Weighted:
Precision: 0.796796017093835
Recall: 0.16623481781376517
F1 score: 0.23398137675255798
Conditional entropy: 0.7932064132264798

```

sigmoid (gamma=1.0)

```

In [ ]: clusters = [len(np.unique(le_y_train025)), 23]
        for k in clusters:
            print(f"k= {k}")
            labels = load_kmeans_labels(

```

```

    k=k, fname="one-hot_encoded_spectral_clustering_sigmoid_1.0"
)
labels = labels.to_numpy().reshape(-1)
new_labels = purity_matching(le_y_train025, labels)
precision_recall_f1_weighted(le_y_train025, new_labels)
con_entr = conditional_entropy(le_y_train025, new_labels)
print(f"Conditional entropy: {con_entr}")

```

```

k= 15
Loading: labels_16_15_one-hot_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.09578947368421052
Weighted:
Precision: 0.408077836550324
Recall: 0.09578947368421054
F1 score: 0.15132405807611082
Conditional entropy: 1.546599244070808
k= 23
Loading: labels_21_23_one-hot_encoded_spectral_clustering_sigmoid_1.0.csv
Accuracy: 0.06979757085020243
Weighted:
Precision: 0.4061342059301873
Recall: 0.06979757085020245
F1 score: 0.11788547029849468
Conditional entropy: 1.543728389893449

```

Comment

The best similarity matrix was polynomial kernel ($\gamma=1.0 / n_{\text{features}}$), that had the best f1 score (weighted) of 0.68823, with one of the lowest conditional entropy values of 0.13785.

DBSCAN

Maximum matching

label encoded

```

In [ ]: labels = load_dbscan_labels(fname="3561_1_label_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train004)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")

```

```

Loading: labels_3561_1_label_encoded_DBSCAN.csv
Accuracy: 0.5769230769230769
Weighted:
Precision: 0.5287109415550711
Recall: 0.576923076923077
F1 score: 0.42786332328126425
Conditional entropy: 1.4924965958339524

```

```

In [ ]: labels = load_dbscan_labels(fname="2550_1_label_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train004)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")

```

```

Loading: labels_2550_1_label_encoded_DBSCAN.csv
Accuracy: 0.5759109311740891
Weighted:
Precision: 0.5287597809564804
Recall: 0.5759109311740891
F1 score: 0.42607613751615075
Conditional entropy: 1.4912341915263154

```

one-hot encoded

```

In [ ]: labels = load_dbscan_labels(fname="3561_1_one-hot_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train004)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")

```

Loading: labels_3561_1_one-hot_encoded_DBSCAN.csv
Accuracy: 0.5769230769230769
Weighted:
Precision: 0.5287109415550711
Recall: 0.576923076923077
F1 score: 0.42786332328126425
Conditional entropy: 1.4924965958339524

```
In [ ]: labels = load_dbscan_labels(fname="2550_1_one-hot_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = max_matching_hungarian(labels, le_y_train004)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")
```

Loading: labels_2550_1_one-hot_encoded_DBSCAN.csv
Accuracy: 0.5759109311740891
Weighted:
Precision: 0.5287597809564804
Recall: 0.5759109311740891
F1 score: 0.42607613751615075
Conditional entropy: 1.4912341915263154

Purity matching

label encoded

```
In [ ]: labels = load_dbscan_labels(fname="3561_1_label_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = purity_matching(le_y_train004, labels)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")
```

Loading: labels_3561_1_label_encoded_DBSCAN.csv
Accuracy: 0.5769230769230769
Weighted:
Precision: 0.5287109415550711
Recall: 0.576923076923077
F1 score: 0.42786332328126425
Conditional entropy: 1.4924965958339524

```
In [ ]: labels = load_dbscan_labels(fname="2550_1_label_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = purity_matching(le_y_train004, labels)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")
```

Loading: labels_2550_1_label_encoded_DBSCAN.csv
Accuracy: 0.5759109311740891
Weighted:
Precision: 0.5292152465435249
Recall: 0.5759109311740891
F1 score: 0.4263158562461742
Conditional entropy: 1.4888607320445884

one-hot encoded

```
In [ ]: labels = load_dbscan_labels(fname="3561_1_one-hot_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = purity_matching(le_y_train004, labels)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")
```

Loading: labels_3561_1_one-hot_encoded_DBSCAN.csv
Accuracy: 0.5769230769230769
Weighted:
Precision: 0.5287109415550711
Recall: 0.576923076923077
F1 score: 0.42786332328126425
Conditional entropy: 1.4924965958339524

```
In [ ]: labels = load_dbscan_labels(fname="2550_1_one-hot_encoded")
labels = labels.to_numpy().reshape(-1)
new_labels = purity_matching(le_y_train004, labels)
precision_recall_f1_weighted(le_y_train004, new_labels)
con_entr = conditional_entropy(le_y_train004, new_labels)
print(f"Conditional entropy: {con_entr}")
```

Loading: labels_2550_1_one-hot_encoded_DBSCAN.csv
Accuracy: 0.5759109311740891
Weighted:
Precision: 0.5292152465435249
Recall: 0.5759109311740891
F1 score: 0.4263158562461742
Conditional entropy: 1.4888607320445884

Comments

Spectral clustering yields worse results than KMeans, this may be attributed to the limited data size used for training spectral clustering.

KMeans also performed better than DBSCAN with DBSCAN having a maximum F1-Score of 0.428 compared to KMeans' 0.811, this may also be attributed to the limited data.