



pyDRMetrics

pyDRMetrics Provides a set of DR quality evaluation metrics.

To install, pip install pyDRMetrics

This document provides sample codes for how to use package API. The sample dataset can be downloaded from the github repo <https://github.com/zhangys11/pyDRMetrics>

Publication

pyDRMetrics - A Python toolkit for dimensionality reduction quality assessment, *Heliyon*, Volume 7, Issue 2, 2021, e06199, ISSN 2405-8440, <https://doi.org/10.1016/j.heliyon.2021.e06199> (<https://www.sciencedirect.com/science/article/pii/S2405844021003042>) (<https://www.sciencedirect.com/science/article/pii/S2405844021003042>)

Problems of high dimensionality

According to the theory of "curse of dimensionality", as the number of dimensions increases, the amount of data needed to generalize a machine learning model grows exponentially. It also results in more storage space and longer model training time.

For distance-based applications (e.g., clustering), "the concept of distance becomes less precise as the number of dimensions grows, since the distance between any two points in a given dataset converges (concentration effects)." and " Different clusters might be found in different subspaces, so a global filtering of attributes is not sufficient (local feature relevance problem)."

Reasons for dimensionality reduction:

1. Reduce computational cost for learning and inference.
2. Reduce the risk of overfitting.
3. For data visualization.
4. For data interpretation.
5. Improve SNR. Denoising effect.

1. Load Data

Data source: Use of proteomic patterns in serum to identify ovarian cancer. Lancet.

SELDI-TOF-MS

```
plt.gray()
```

```
In [4]: import pandas as pd  
data = pd.read_csv('ovarian-cancer-nci-pbsii-data-no-header.csv', header = None)  
print(data.head())
```

	0	1	2	3	4	5	
6	\						
0	0.494626	0.263735	0.321841	0.220934	0.297622	0.316458	0.154763
1	0.258063	0.406593	0.321841	0.069771	0.333335	0.354432	0.321431
2	0.537636	0.032966	0.321841	0.209307	0.404762	0.113927	0.369049
3	0.000000	0.395605	0.310347	0.197673	0.404762	0.455701	0.416666
4	0.526884	0.395605	0.367817	0.383719	0.488099	0.392405	0.238094
	7	8	9	...	15144	15145	15146
15147	\						
0	0.223685	0.304346	0.241757	...	0.467498	0.483622	0.449296
	0.449296						
1	0.144740	0.260869	0.142853	...	0.622356	0.631765	0.619718
	0.619718						
2	0.223685	0.536231	0.131865	...	0.032870	0.038462	0.035918
	0.035918						
3	0.210527	0.420292	0.274723	...	0.495252	0.497864	0.486621
	0.486621						
4	0.500000	0.362316	0.274723	...	0.245436	0.267096	0.251408
	0.251408						
	15148	15149	15150	15151	15152	15153	
0	0.449296	0.449296	0.449296	0.449296	0.449296	0.449296	
1	0.619718	0.619718	0.619718	0.619718	0.619718	0.619718	
2	0.035918	0.035918	0.035918	0.035918	0.035918	0.035918	
3	0.486621	0.486621	0.486621	0.486621	0.486621	0.486621	
4	0.251408	0.251408	0.251408	0.251408	0.251408	0.251408	

[5 rows x 15154 columns]

```
In [5]: import numpy as np

cols = data.shape[1]

# convert from pandas dataframe to numpy matrices
X = np.array(data.iloc[:,1:-1]) # skip first and last cols
y = np.array(data.iloc[:, -1])

# X_names = list(data.columns.values[1:-1]) # -1 for removing the last column
labels = list(set(y))
```

```
In [6]: # %load draw_averaged_waveform.py
import matplotlib.pyplot as plt
import numpy as np

def draw_averaged_waveform (X, X_names):

    plt.figure(figsize = (100,25))

    plt.scatter(X_names, np.mean(X, axis=0).tolist())

    cur_axes = plt.gca()
    cur_axes.axes.get_xaxis().set_visible(False) #.set_ticklabels([])
    cur_axes.axes.get_yaxis().set_visible(False) #.set_ticklabels([])

    # plt.title(u'Averaged Spectrum\n', fontsize=50)
    plt.show()
```

DR by PCA

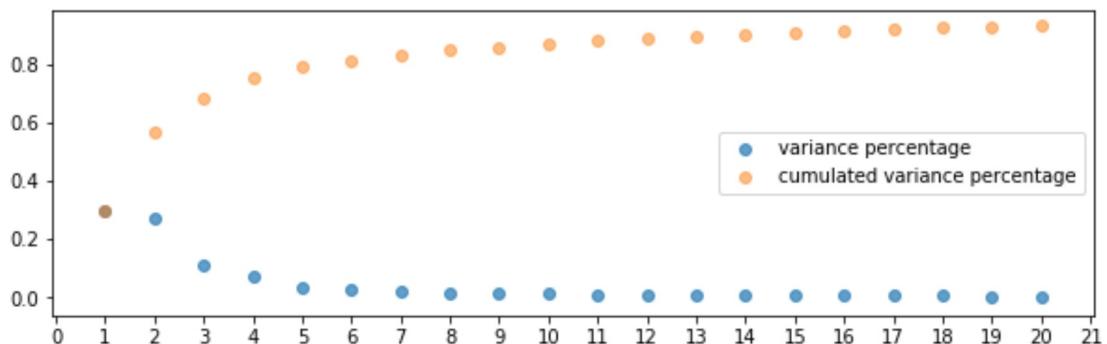
```
In [8]: from sklearn.decomposition import PCA
import matplotlib.ticker as mticker

K = 20

pca = PCA(n_components = K) # keep the first K components
pca.fit(X)
X_pca = pca.transform(X)

plt.figure(figsize=(10,3))
plt.scatter(list(range(1,K+1)), pca.explained_variance_ratio_, alpha=0.7, label='variance percentage')
plt.scatter(list(range(1,K+1)), pca.explained_variance_ratio_.cumsum(), alpha=0.5, label='cumulated variance percentage')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.legend()
plt.show()

print('explained variance ratio:', pca.explained_variance_ratio_)
```



```
explained variance ratio: [0.29792598 0.27090529 0.1126371  0.0729
5763 0.03290289 0.02353424
0.01988971 0.01540482 0.0116174  0.01111293 0.00971757 0.0083236
0.00761821 0.00701775 0.0065432  0.00564775 0.00466537 0.00446773
0.00387578 0.0037673 ]
```

With the "Elbow" method, choose $k = 5$. A turning point can be identified at around $k = 5$. The explained variances (information) are less than 3% after $k = 5$. The first 5 PCs hold 78.7% of the total information.

```
In [9]: K = 5

pca = PCA(n_components = K) # keep the first K components
pca.fit(X)
Z = pca.transform(X)
Xr = pca.inverse_transform(Z)
```

2. Dimensionality Reduction Performance Measurement

```
In [10]: from pyDRMetrics.pyDRMetrics import *
%matplotlib inline
```

```
In [11]: drm = DRMetrics(X, Z, Xr) # this can be quite time consuming if your dataset is large (many samples and many features)
```

2.1 Reconstruction Error

Every DR method optimizes a different cost function, and it would be unfair to compare t-SNE and PCA by means of either recovered variance or KL-Divergence. One fair measure would be the reconstruction error, i.e., reconstructing the original data from a limited number of dimensions, but as discussed above not many methods provide forward and inverse mappings.

Use Frobenius Norm to compute MSE

$$MSE = \frac{\|X - Xr\|_F^2}{mn}$$

m,n is the dimension of A and A'. A-A' is the error matrix.

Relative Error

$$R = \frac{\frac{\|X - Xr\|_F^2}{mn}}{\frac{\|X - 0\|_F^2}{mn}} = \frac{\|X - Xr\|_F^2}{\|X\|_F^2}$$

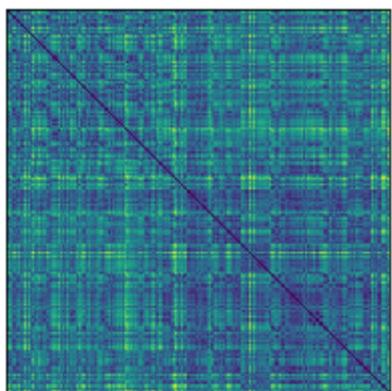
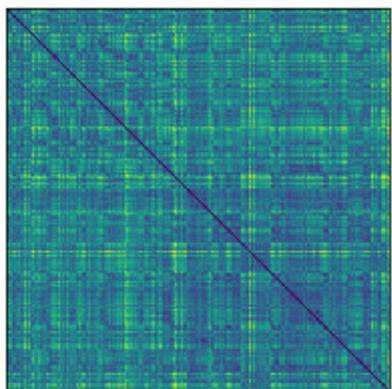
```
In [12]: drm.mse, drm.rmse
```

```
Out[12]: (0.0068337166063726276, 0.03544851864201571)
```

2.2. Distance Matrix

Used to analyze whether distances are kept between samples after dimensionality reduction

```
In [13]: drm.plot_distance_matrix()
```



2.3 Residual Variance

Measure the correlation between the distance matrices in a high and low dimensional space.

“residual variance” = $1 - r^2(D, D')$, where r is the Pearson or Spearman correlation and D, D' are the distances matrices.

`obj.Vr` returns the Pearson version (measures only linear correlation). `obj.Vrs` returns the Spearman version (also measures non-linear correlation).

```
In [14]: drm.Vr, drm.Vrs
```

```
Out[14]: (0.02854782282623025, 0.028487313400964998)
```

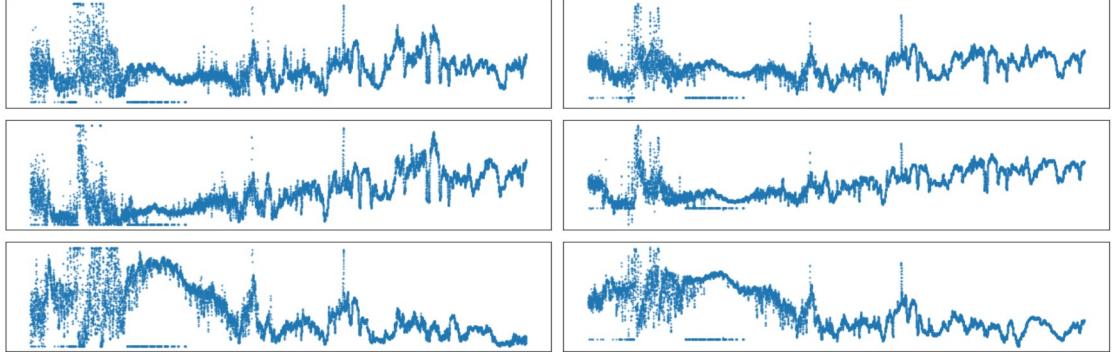
2.4 Parallel Comparison of Specific Samples

```
In [15]: # drm.visualize_reconstruction()

%run visualize_sample_reconstruction.py
_ = visualize_sample_reconstruction(X, Xr, X_names=list(range(X.shape[1])), N = 3)

ERROR:root:File ``visualize_sample_reconstruction.py`` not found.

left column: original waveform           right column: recovered waveform
```



2.5 ranking matrix

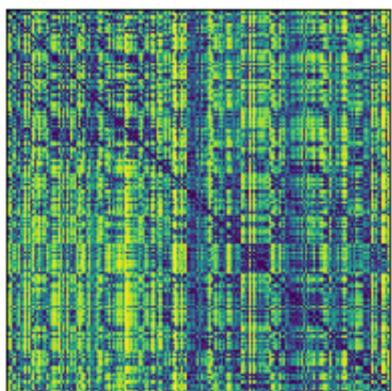
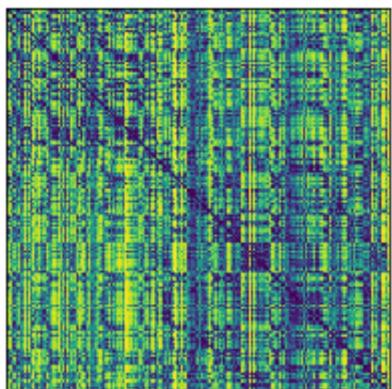
$r_{ij} = |\{k : d_{ik} < d_{ij} \text{ or } (d_{ik} = d_{ij} \text{ and } 1 \leq k < j \leq n\}|$, $|A|$ denotes the number of elements in a set A

本质上，distance matrix 和 ranking matrix 等价于核变换

```
In [16]: rm = drm.R
m = len(drm.X)

# Ranking matrix properties
assert np.sum(rm) == m*m*(m-1)/2
assert np.max(rm) == m-1
assert np.min(rm) == 0
```

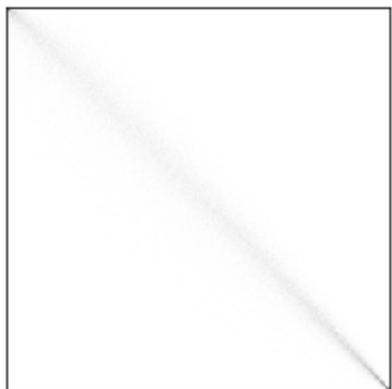
```
In [17]: drm.plot_ranking_matrix()
```



2.6 定义co-ranking matrix

The co-ranking matrix Q then has elements: $q_{kl} = |\{(i, j) : r_{ij}^k = l\}|$

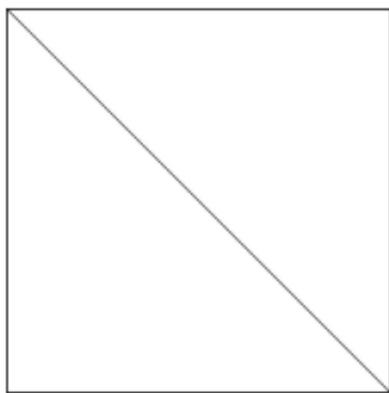
```
In [18]: drm.plot_coranking_matrix()
```



```
In [19]: Q = drm.Q
m = len(drm.X)
assert np.sum(Q) == m*m
assert np.isin(np.sum(Q, axis = 0), [m]).all()
assert np.isin(np.sum(Q, axis = 1), [m]).all()
```

co-ranking matrix is a 2d-histogram of the ranks. That is, q_{ij} is an integer which counts how many points of distance rank j became rank i . In a perfect DR, this matrix will only have non-zero entries in the diagonal. if most of the non-zero entries are in the lower triangle, then the DR collapsed far away points onto each other; if most of the non-zero entries are in the upper triangle, then the DR teared close points apart.

```
In [38]: # an ideal Q is an diagonal matrix
         = visualize_matrix(np.eye(m)*m, cmap = 'gray_r')
```



2.6 Metrics derived from co-ranking matrix

Trustworthiness and continuity

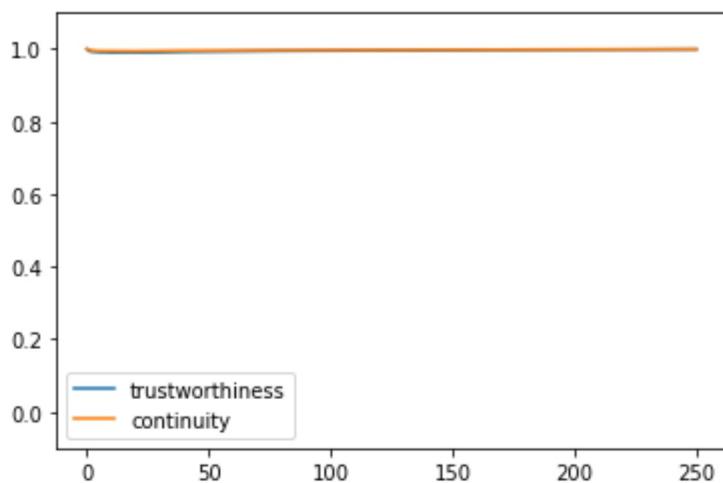
$$T(k) = 1 - \frac{1}{mk(m-k)} \sum_{i=1}^m \sum_{j \in \mathcal{LL}_k} \max(0, (r(i,j) - k))$$

$$C(k) = 1 - \frac{1}{mk(m-k)} \sum_{i=1}^m \sum_{j \in \mathcal{UR}_k} \max(0, (r(i,j) - k))$$

```
for k in range(m-1):
    Qs = Q[k:,:k]
    W = np.arange(Qs.shape[0]).reshape(-1, 1) # a column vector of weights. weight = rank error = actual_rank - k
    T[k] = 1-np.sum(Qs * W)/(k+1)/m/(m-1-k) # 1 - normalized hard-k-in trusions. lower-left region. weighted by rank error (rank - k)
    Qs = Q[:k,k:]
    W = np.arange(Qs.shape[1]).reshape(1, -1) # a row vector of weights. weight = rank error = actual_rank - k
    C[k] = 1-np.sum(Qs * W)/(k+1)/m/(m-1-k) # 1 - normalized hard-k-ex trusions. upper-right region
```

```
In [40]: plt.plot(drm.T, label = 'trustworthiness')
plt.plot(drm.C, label = 'continuity')
plt.ylim(-0.1,1.1)
plt.legend()
print('T AUC = ', drm.AUC_T)
print('C AUC = ', drm.AUC_C)
```

T AUC = 0.9957649593734504
C AUC = 0.9975462929370625



$$Q_{NN}(k) = \frac{1}{kN} \sum_{i=1}^k \sum_{j=1}^k q_{ij}$$

$Q_{NN}(k)$ is the number of points that belong to the k-th nearest neighbors in both high- and low-dimensional space, normalized to give a maximum of 1.

This quantity can be adjusted for random embeddings, giving the Local Continuity Meta Criterion.

$$LCMC(k) = Q_{NN}(k) - k/(n - 1)$$

LCMC has a well defined maximum at k_{max} . Two measures without parameters are defined:

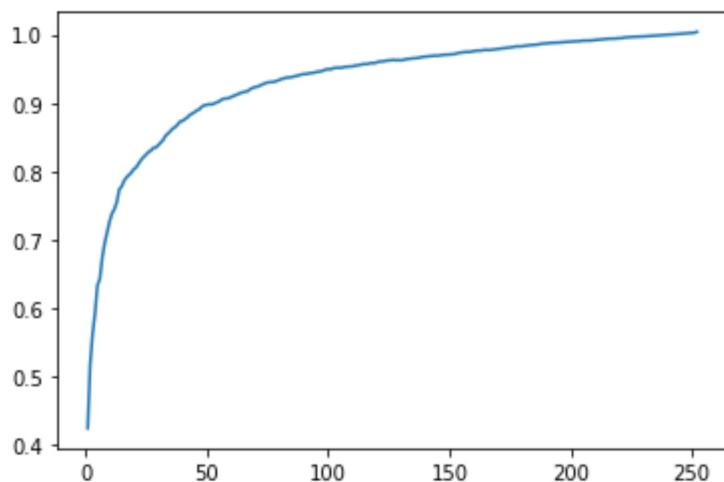
$$Q_{local} = \frac{1}{k_{max}} \sum_{k=1}^{k_{max}} Q_{NN}(k),$$

$$Q_{global} = \frac{1}{n - k_{max}} \sum_{k=k_{max}}^{n-1} Q_{NN}(k),$$

These measure the preservation of local and global distances respectively.

```
In [20]: plt.plot(list(range(1, len(drm.QNN)+1)), drm.QNN)
```

```
Out[20]: [
```

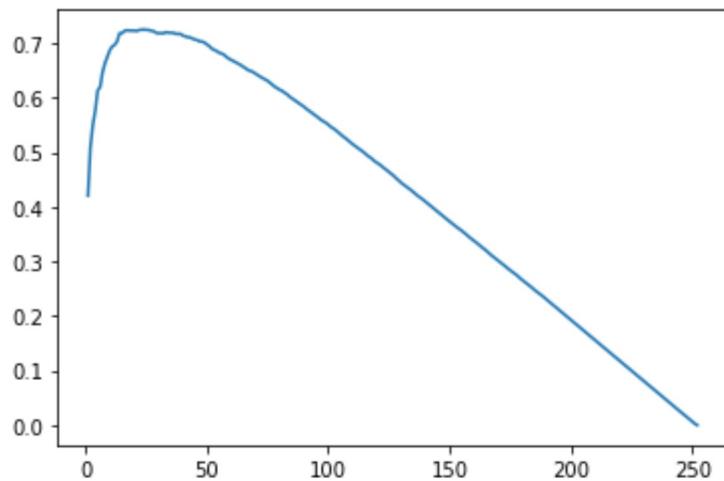


```
In [21]: drm.AUC
```

```
Out[21]: 0.9322314520275514
```

```
In [43]: plt.plot(list(range(1, len(drm.LCMC)+1)), drm.LCMC)
```

```
Out[43]: [
```



```
In [22]: drm.kmax # argmax(LCMC)
```

```
Out[22]: 23
```

```
In [23]: drm.Qlocal,drm.Qglobal
```

```
Out[23]: (0.7166949983403357, 0.954116711563549)
```

get_html() and get_json()

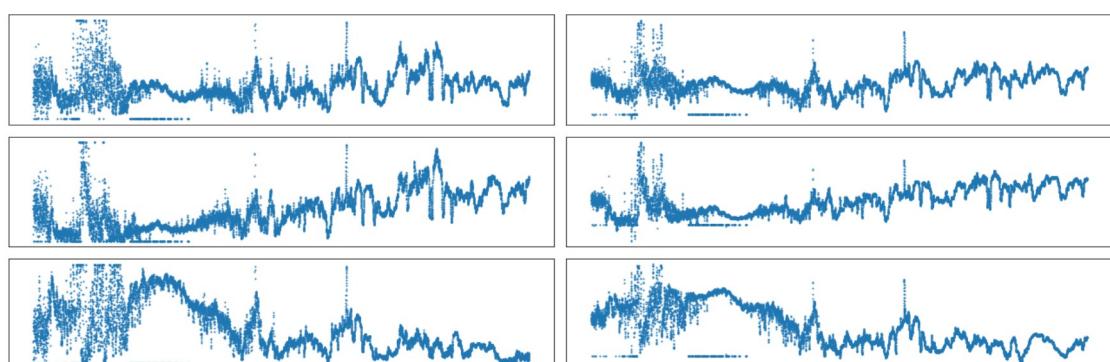
For use by web applications

```
In [24]: from IPython.display import display, HTML
# display(HTML(drm.get_html()))
drm.report()
```

--- Sample Reconstruction (X and Xr) ---

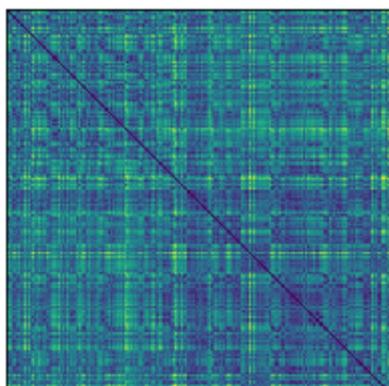
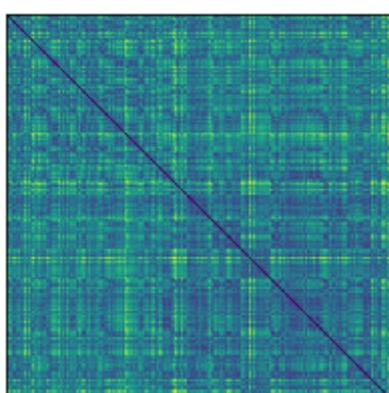
left column: original waveform

right column: recovered waveform



rMSE = 0.03544851864201571

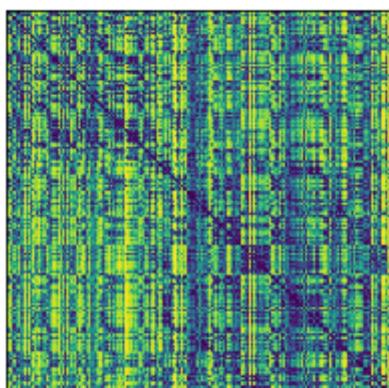
--- Distance Matrices (D and Dz) ---

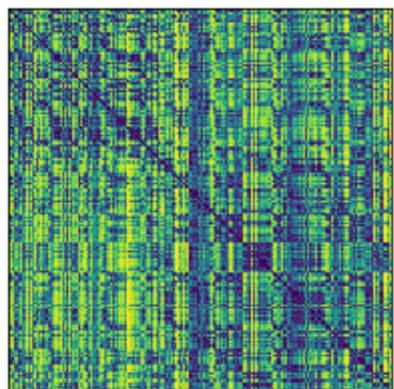


Residual Variance (using Pearson's r) = 0.02854782282623025

Residual Variance (using Spearman's r) = 0.028487313400964998

--- Ranking Matrices (R and Rz) ---

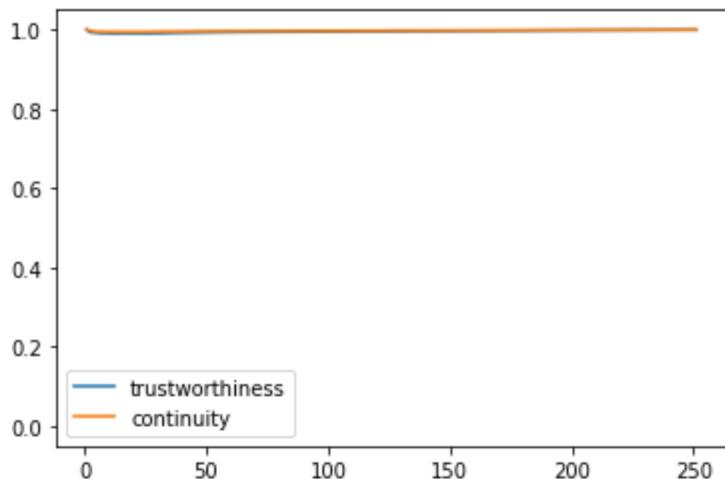




--- Co-ranking Matrix (Q) ---



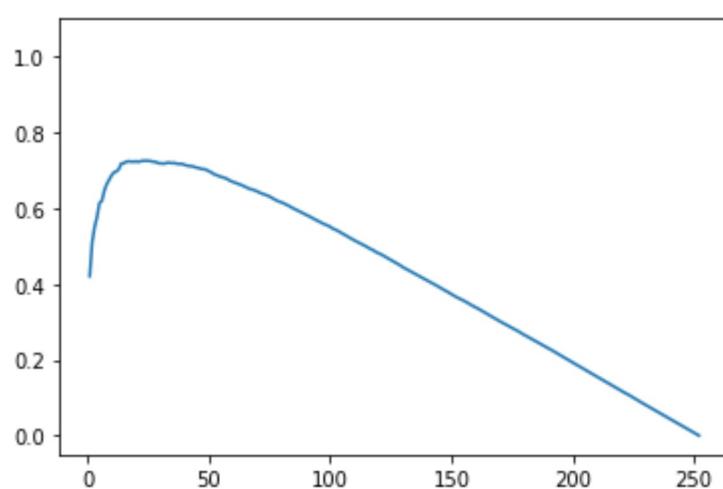
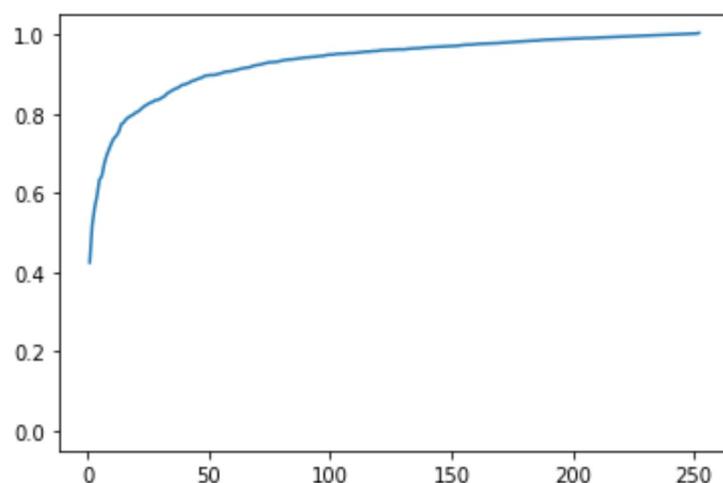
--- Trustworthiness T(k) and Continuity C(k) ---



AUC of T = 0.9957665477356022

AUC of C = 0.9975479241992338

--- QNN(k) Curve ---



```
In [25]: s = drm.get_json()
dic = json.loads(s)
dic.keys()
```

```
Out[25]: dict_keys(['mse', 'ms', 'rmse', 'Vr', 'Vrs', 'AUC', 'kmax', 'Qlocal', 'Qglobal', 'AUC_T', 'AUC_C'])
```

Dynamic Properties

Let k in the range [1,21] and check PCA's metrics

```
In [26]: from sklearn.decomposition import PCA

drms = []
rmses = []
AUCs = []
Qlocals = []
Qglobals = []
Vrs = []

for k in range(1,21):

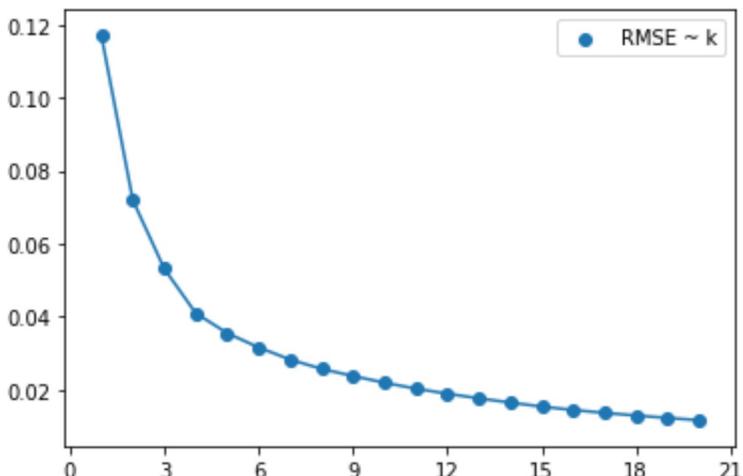
    pca = PCA(n_components = k) # keep the first k components
    pca.fit(X)
    Z = pca.transform(X)
    # plotComponents3D(Z, y, labels)
    Xr = pca.inverse_transform(Z)

    drm = DRMetrics(X, Z, Xr)
    drms.append(drm)

    rmses.append(drm.rmse)
    Vrs.append(drm.Vr)
    AUCs.append(drm.AUC)
    Qlocals.append(drm.Qlocal)
    Qglobals.append(drm.Qglobal)
```

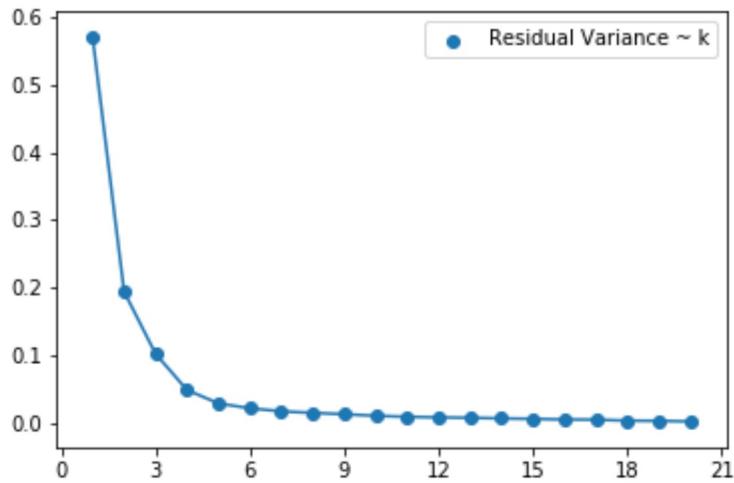
```
In [27]: from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)), rmses)
plt.scatter(list(range(1,21)), rmses, label = 'RMSE ~ k')
plt.legend()
```

```
Out[27]: <matplotlib.legend.Legend at 0x176801dfa88>
```



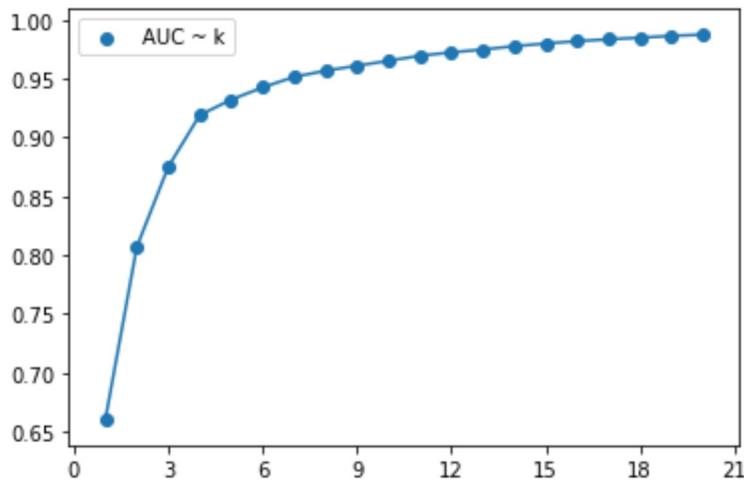
```
In [28]: from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)), Vrs)
plt.scatter(list(range(1,21)), Vrs, label = 'Residual Variance ~ k')
plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x1768027b548>



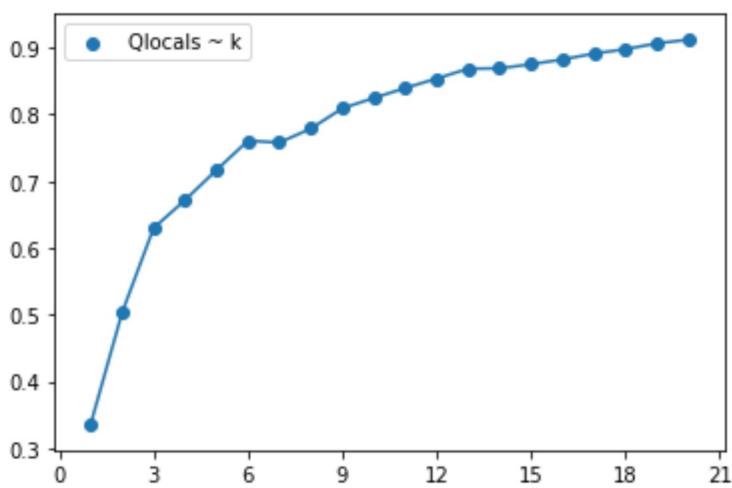
```
In [29]: from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)), AUCs)
plt.scatter(list(range(1,21)), AUCs, label = 'AUC ~ k')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x176801aefc8>



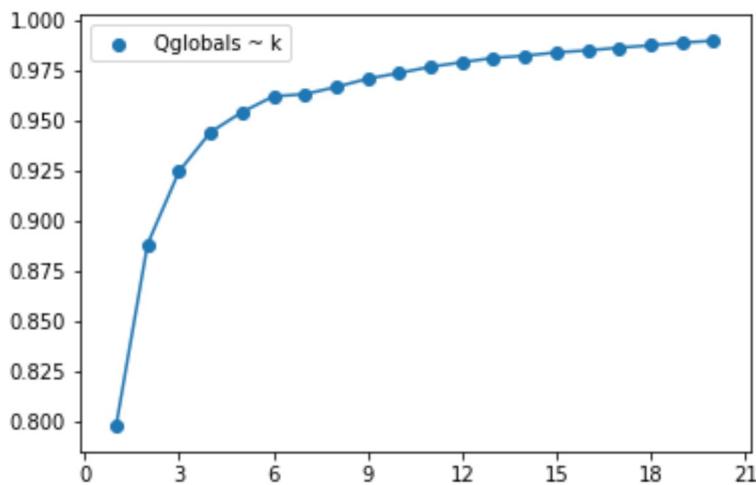
```
In [30]: from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)),Qlocals)
plt.scatter(list(range(1,21)), Qlocals, label = 'Qlocals ~ k')
plt.legend()
```

Out[30]: <matplotlib.legend.Legend at 0x17680238308>



```
In [31]: from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)),Qglobals)
plt.scatter(list(range(1,21)), Qglobals, label = 'Qglobals ~ k')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x17680447bc8>



Case Study 2: Raman spectra

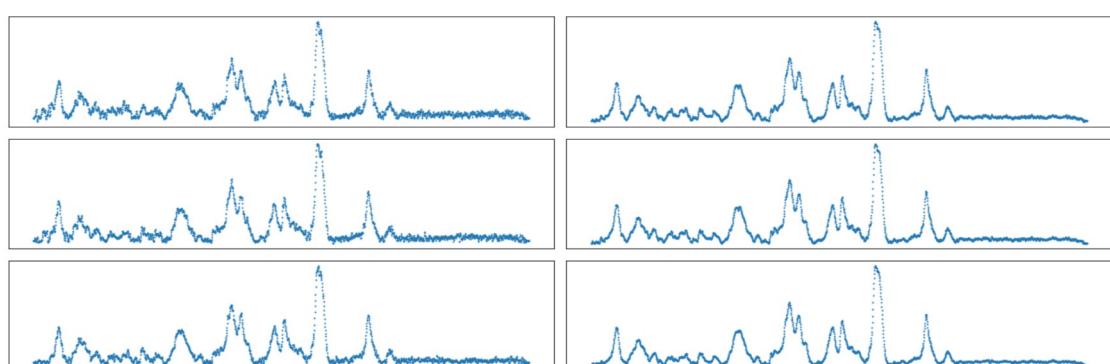
Test PCA and VQ

```
In [32]: from pyDRMetrics import *
drm = DRMetrics.test('raman.csv', 3, dr = 'PCA')
drm.report()
```

--- Sample Reconstruction (X and Xr) ---

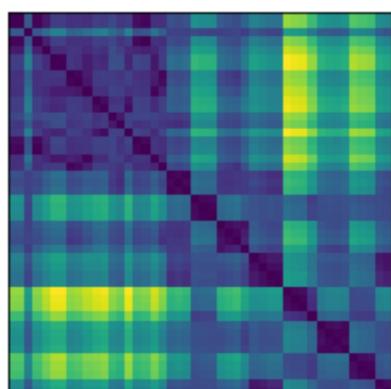
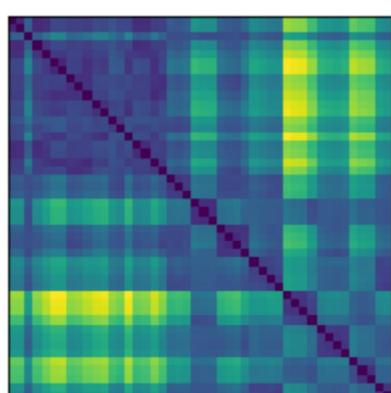
left column: original waveform
aveform

right column: recovered w



rMSE = 0.010637683165950566

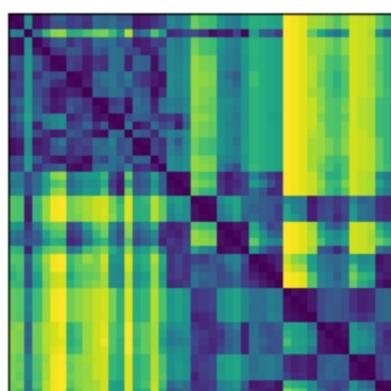
--- Distance Matrices (D and Dz) ---

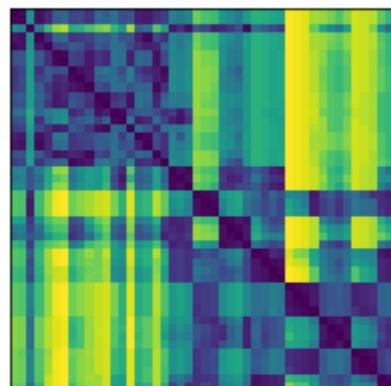


Residual Variance (using Pearson's r) = 0.01126637546298126

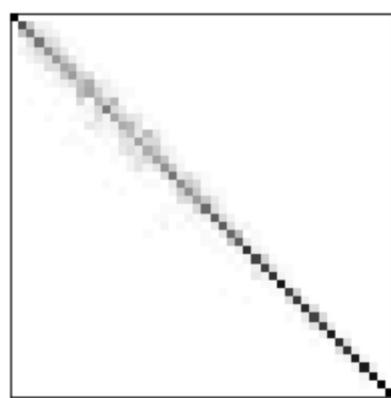
Residual Variance (using Spearman's r) = 0.00819300029711667

--- Ranking Matrices (R and Rz) ---

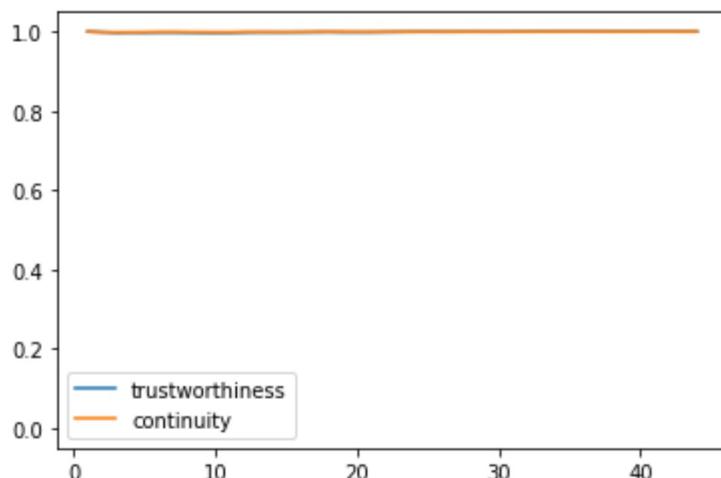




--- Co-ranking Matrix (Q) ---



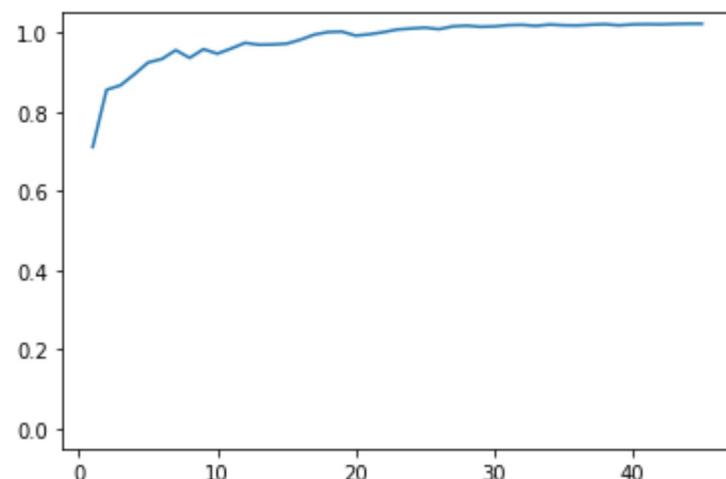
--- Trustworthiness T(k) and Continuity C(k) ---



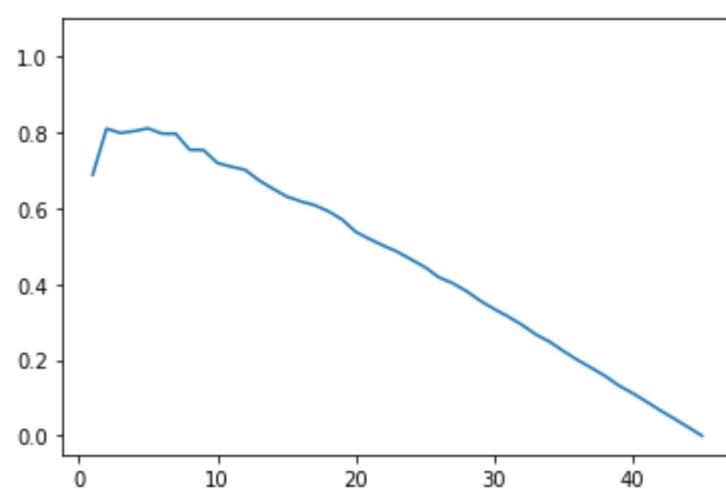
AUC of T = 0.9989085475189007

AUC of C = 0.9995364059806525

--- QNN(k) Curve ---



AUC of QNN = 0.9821554030923509
--- LCMC(k) Curve ---



kmax (0-based index) = 4
Qlocal = 0.8504444444444446
Qglobal = 0.9961748284788946

```
In [33]: np.savetxt('raman_z.csv', drm.Z, fmt='%f', delimiter=',')  
np.savetxt('raman_xr.csv', drm.Xr, fmt='%f', delimiter=',')
```

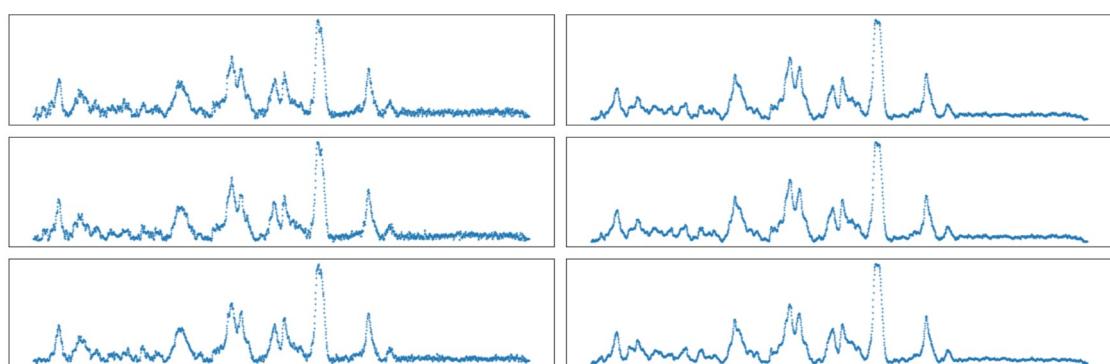
```
In [34]: from pyDRMetrics import *

drm = DRMetrics.test('raman.csv', 3, dr = 'VQ') # VQ or Clustering is a special form of dimensionality reduction
drm.report()
```

--- Sample Reconstruction (X and Xr) ---

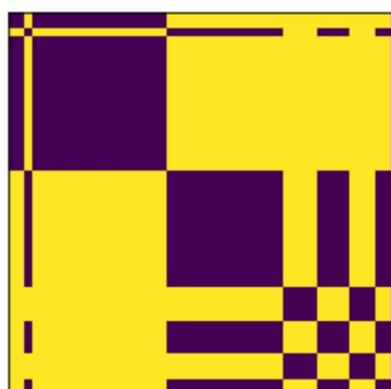
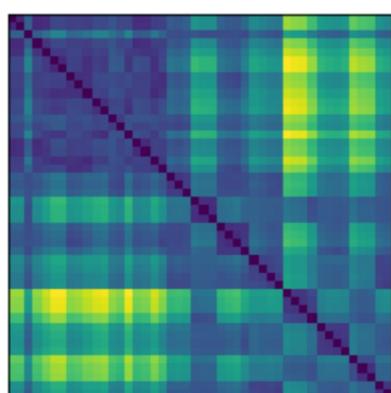
left column: original waveform

right column: recovered waveform



rMSE = 0.026951703948641496

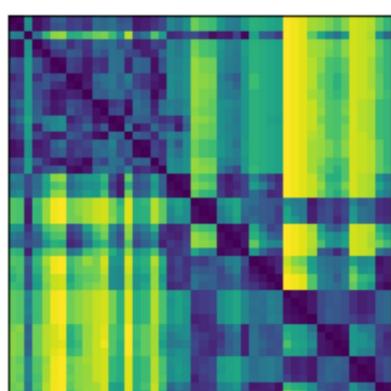
--- Distance Matrices (D and Dz) ---

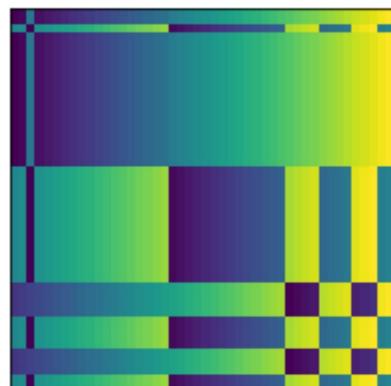


Residual Variance (using Pearson's r) = 0.5395973900914566

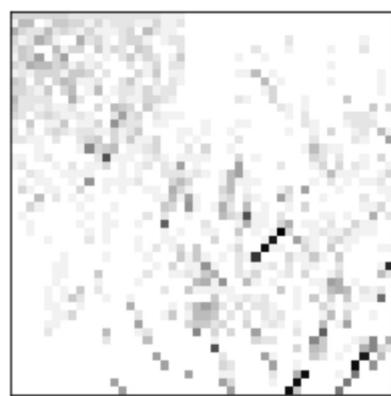
Residual Variance (using Spearman's r) = 0.4335241895993569

--- Ranking Matrices (R and Rz) ---

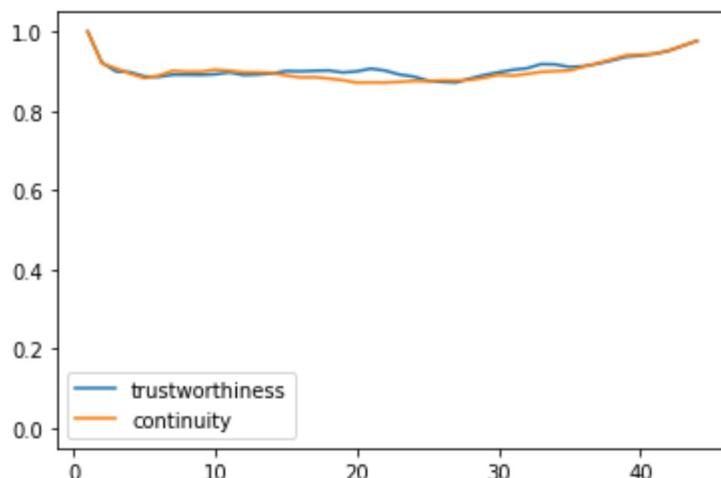




--- Co-ranking Matrix (Q) ---



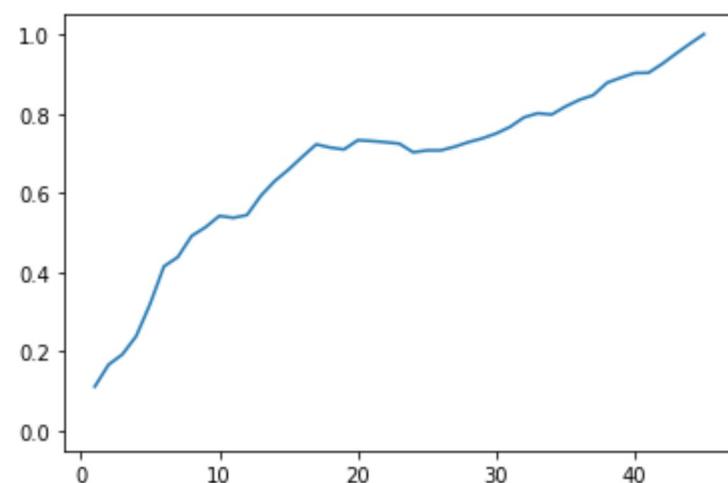
--- Trustworthiness T(k) and Continuity C(k) ---



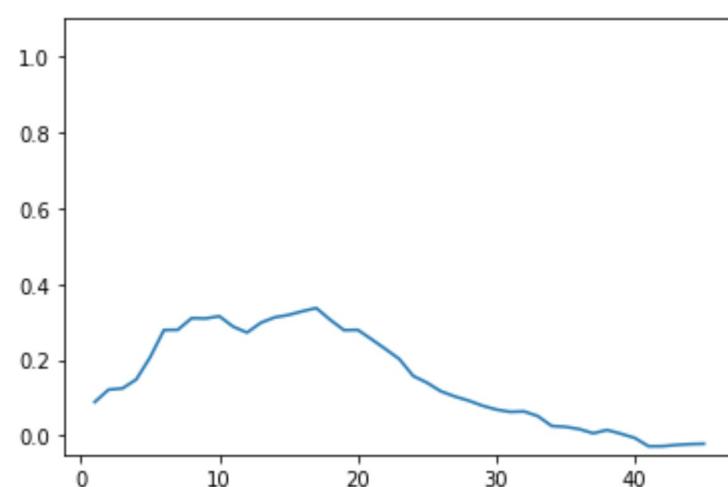
AUC of T = 0.9077352091277376

AUC of C = 0.9026957642707345

--- QNN(k) Curve ---



AUC of QNN = 0.6731868273940285
--- LCMC(k) Curve ---



kmax (0-based index) = 16
Qlocal = 0.4593282056488515
Qglobal = 0.7930969964061887

Case Study 3: Handwritten digits data

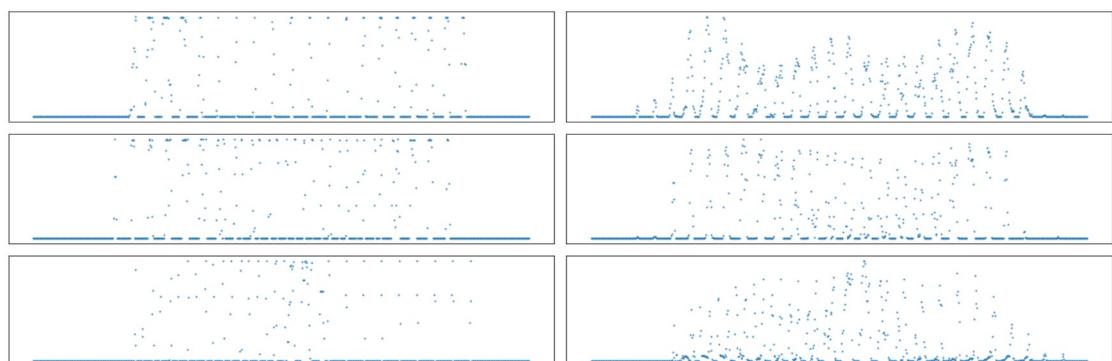
Test NMF and MDS

```
In [38]: from pyDRMetrics import *
drm = DRMetrics.test('digits.csv', 8, dr = 'NMF')
drm.report()
```

--- Sample Reconstruction (X and Xr) ---

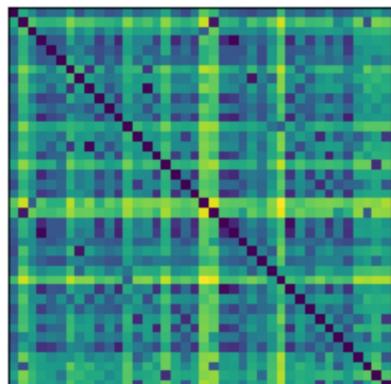
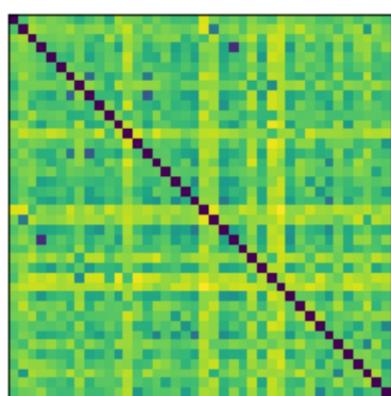
left column: original waveform

right column: recovered waveform



rMSE = 0.27575924406723196

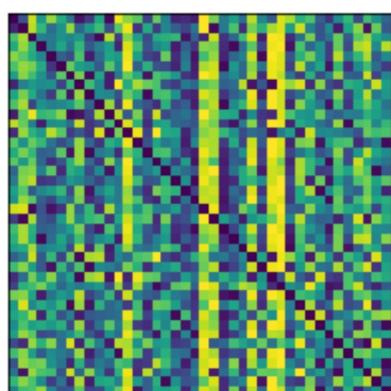
--- Distance Matrices (D and Dz) ---

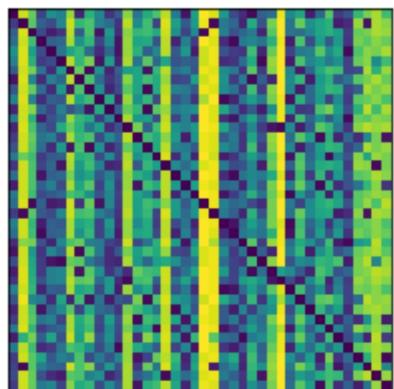


Residual Variance (using Pearson's r) = 0.4649424168170553

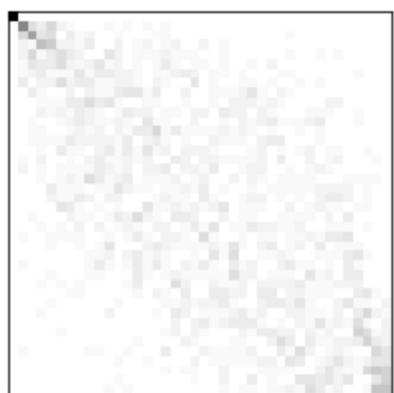
Residual Variance (using Spearman's r) = 0.4789924677744728

--- Ranking Matrices (R and Rz) ---

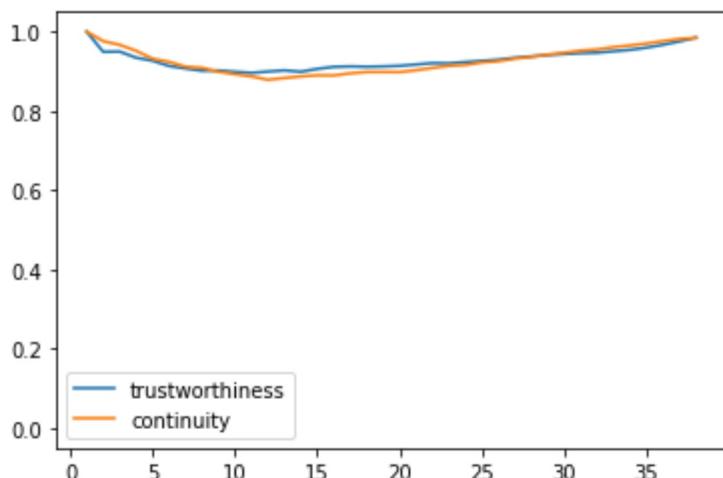




--- Co-ranking Matrix (Q) ---



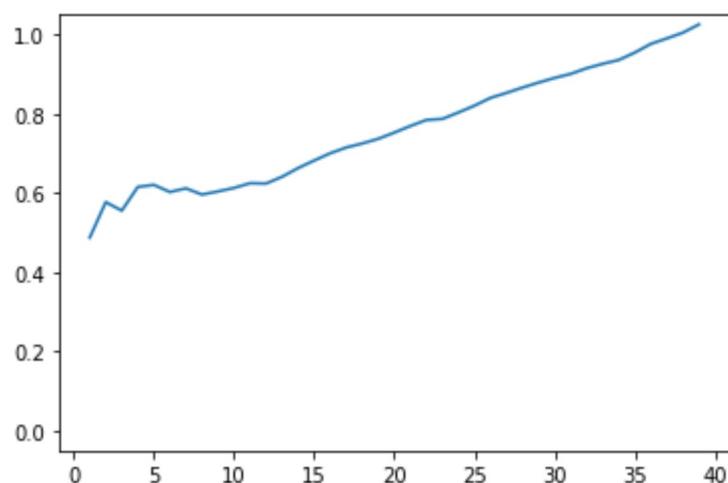
--- Trustworthiness T(k) and Continuity C(k) ---



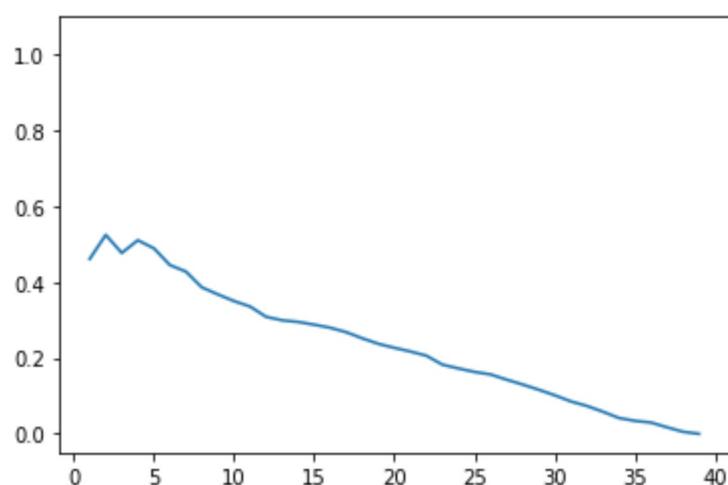
AUC of T = 0.9295348459742035

AUC of C = 0.9278298246917799

--- QNN(k) Curve ---



AUC of QNN = 0.7607769402113559
--- LCMC (k) Curve ---



kmax (0-based index) = 1
Qlocal = 0.532051282051282
Qglobal = 0.7610129771735775

本组数据集为图片，以下进行reshape操作进行可视化，对比原始与复原数据

```
In [39]: import pandas as pd
```

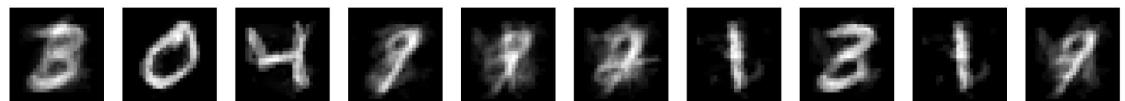
```
# Each data is actually a 28x28, flattened into a 784 vector.  
# We can restore to its original shape to visuzlize.
```

```
N = 10  
plt.figure(figsize = (4*N,5))  
for i in range(N):  
    x = drm.X[i]  
    ax = plt.subplot(1, N, i+1)  
    plt.imshow(x.reshape((28,28)), cmap='gray')  
    ax.get_xaxis().set_visible(False)  
    ax.get_yaxis().set_visible(False)
```



```
In [40]: N = 10
```

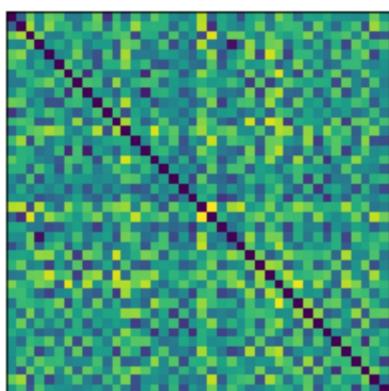
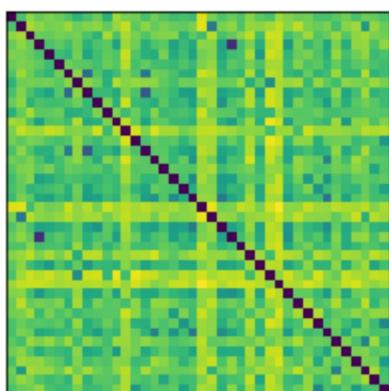
```
plt.figure(figsize = (4*N,5))  
for i in range(N):  
    x = drm.Xr[i]  
    ax = plt.subplot(1, N, i+1)  
    plt.imshow(x.reshape((28,28)), cmap='gray')  
    ax.get_xaxis().set_visible(False)  
    ax.get_yaxis().set_visible(False)
```



```
In [42]: from pyDRMetrics import *

drm = DRMetrics.test('digits.csv', 3, dr = 'MDS')
drm.report()
```

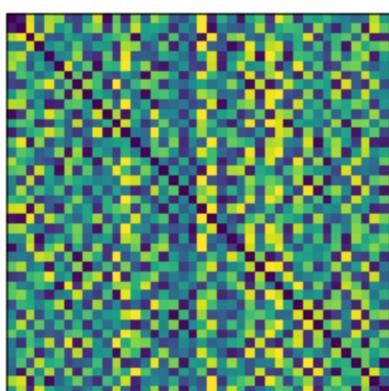
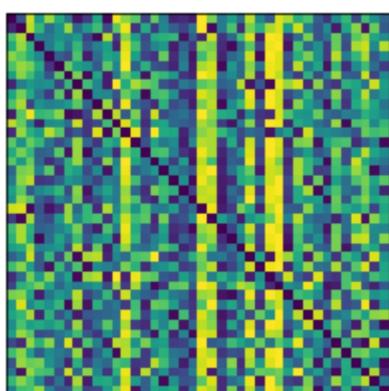
--- Distance Matrices (D and Dz) ---



Residual Variance (using Pearson's r) = 0.4162673631425422

Residual Variance (using Spearman's r) = 0.44276421441187985

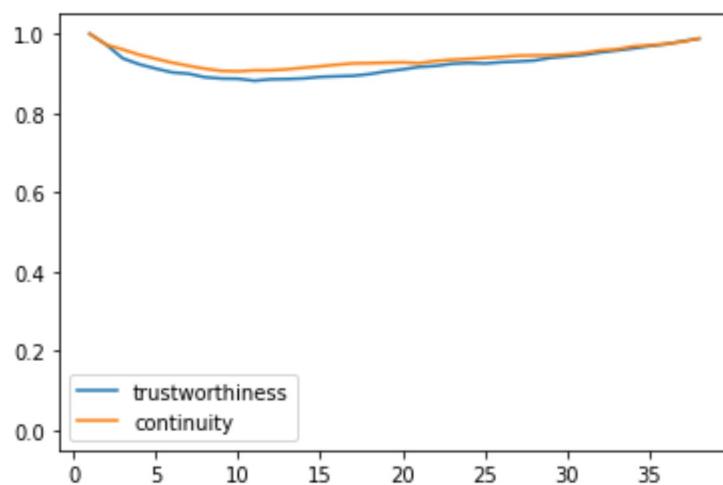
--- Ranking Matrices (R and Rz) ---



--- Co-ranking Matrix (Q) ---



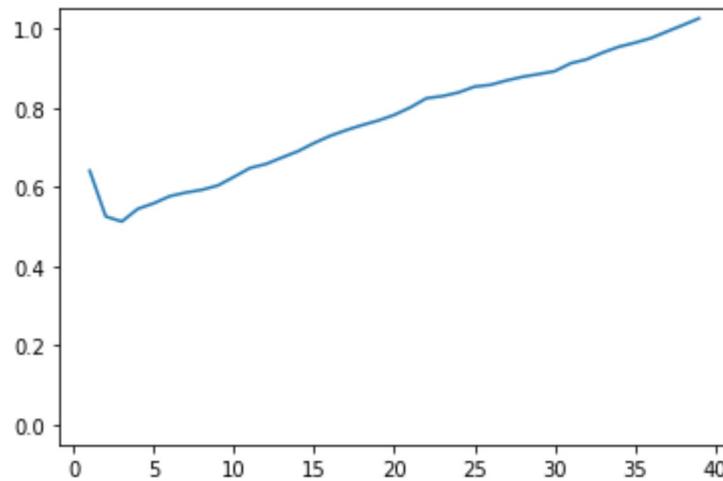
--- Trustworthiness $T(k)$ and Continuity $C(k)$ ---



AUC of $T = 0.9254202573710154$

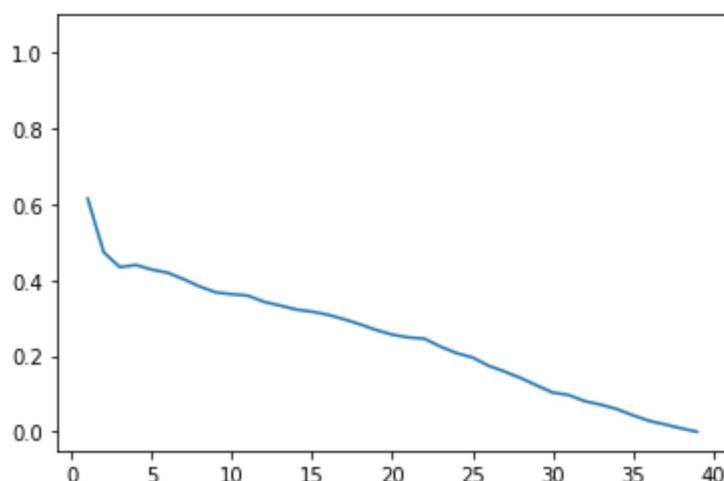
AUC of $C = 0.9401424941990277$

--- QNN(k) Curve ---



AUC of QNN = 0.7731767584084418

--- LCMC(k) Curve ---



```
kmax (0-based index) = 0
Qlocal = 0.6410256410256411
Qglobal = 0.7665329619023211
```

Appendix A: Built-in algorithms

Algorithm	Description	Implementation
PCA	Principal Component Analysis	sklearn.decomposition.PCA
NMF	Non-negative Matrix Factorization	sklearn.decomposition.NMF
RP	Random Projection	sklearn.random_projection.GaussianRandomProjection
VQ	Vector Quantization (by K-means)	sklearn.cluster.KMeans
MDS	Multidimensional scaling	sklearn.manifold.MDS
t-SNE	t-distributed Stochastic Neighbor Embedding	sklearn.manifold.TSNE
Identity	A "perfect" DR	X = Z = Xr

Appendix B: pyDRMetrics main APIs

class DRMetrics(builtins.object)

Properties / Fields

X Data before DR. m-by-n matrix. m is the sample size. n is the dimension/feature number.

Z Data after DR. m-by-k matrix. Typically, k << n

Xr Reconstructed Data. m-by-n matrix. Optional parameter. If a DR algorithm has no inverse transform. Pass None.

D Distance matrix of X

Dz Distance matrix of Z

Vr The residual variance between D and Dz. The default version uses Pearson's r to calculate the residual variance. Use Vrs for the Spearman's r version.

mse Reconstruction error. MSE of X and Xr

rmse Relative reconstruction error. Relative MSE of X and Xr

R Ranking matrix of X

Rz Ranking matrix of Z

Q Co-ranking matrix between R and Rz

T Trustworthiness. An array. There is also a single-valued AUC_T that measures the area under the T(trustworthiness) curve.

C Continuity. An array. There is also a single-valued AUC_C that measures the area under the C(continuity) curve.

QNN Co-k-nearest neighbor size. An array.

AUC The area under the QNN curve.

LCMC Local Continuity Meta Criterion. An array.

Qlocal Local property metric

Qglobal Global property metric

Member Methods

```
__init__(self, X, Z, Xr=None)    Constructor. X is the original data. Z  
is the data after DR. Xr is the reconstructed data.  
test(cls, csv, k = 3, dr = 'PCA')    A constructor overload that facilitates testing common DR algorithms. csv is the data file. dr is used to specify the algorithm, e.g. "PCA", "NMF", "RP", "TSNE", etc.  
plot_coranking_matrix(self) Visualize the co-ranking matrix between R and Rz as heatmaps.  
plot_distance_matrix(self) Visualize the distance matrices before and after DR as heatmaps, i.e., D and Dz.  
plot_ranking_matrix(self) Visualize the ranking matrices before and after DR as heatmaps, i.e., R and Rz.  
visualize_reconstruction(self) Plot the original data and the reconstruction data side by side. Show 3 random samples.  
report(self) Print out a summary report, with inline plots.  
get_json(self) Generate a JSON-format dictionary object containing all DR quality metrics. Only numeric metrics are returned.  
get_html(self) Generate an HTML segment that can be embedded in web pages. Plotted images are embedded as base64 strings to avoid referencing external files.
```

Sample Code

```
from pyDRMetrics import *  
from sklearn.decomposition import PCA  
%matplotlib inline
```

Appendix C: A web GUI tool based on pyDRMetrics

