pyDRMetrics

Provides a set of DR quality evaluation metrics

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-baesd 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 dimensonality 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 [18]:
```
import pandas as pd
data = pd.read_csv('ovarian-cancer-nci-pbsii-data.csv')
print(data.head())
```

```
   Unnamed: 0  MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  \
0           0     0.494626    0.263735    0.321841       0.220934
1           1     0.258063    0.406593    0.321841       0.069771
2           2     0.537636    0.032966    0.321841       0.209307
3           3     0.000000    0.395605    0.310347       0.197673
4           4     0.526884    0.395605    0.367817       0.383719

   MZ0.000810195  MZ0.001428564  MZ0.002221123  MZ0.003187869  MZ0.004328805
```

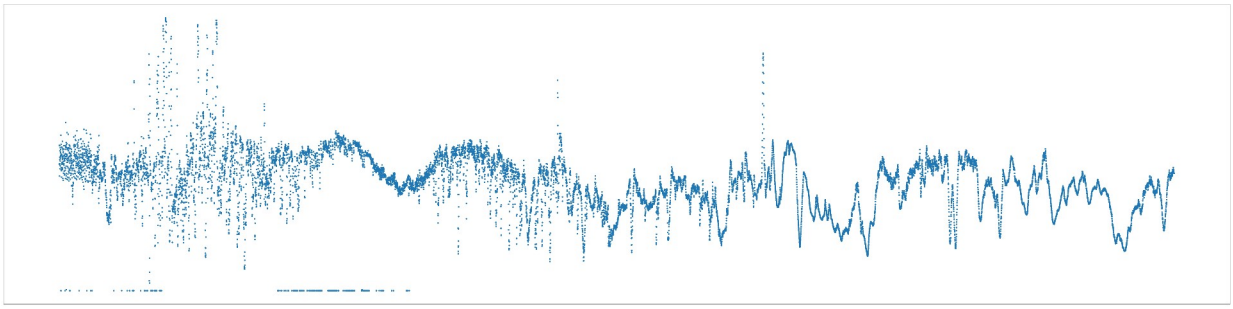
```
\
0       0.297622       0.316458       0.154763       0.223685       0.304346
1       0.333335       0.354432       0.321431       0.144740       0.260869
2       0.404762       0.113927       0.369049       0.223685       0.536231
3       0.404762       0.455701       0.416666       0.210527       0.420292
4       0.488099       0.392405       0.238094       0.500000       0.362316

   ...  MZ19974.404  MZ19977.042  MZ19979.68  MZ19982.319  MZ19984.957  \
0  ...     0.483622     0.449296    0.449296     0.449296     0.449296
1  ...     0.631765     0.619718    0.619718     0.619718     0.619718
2  ...     0.038462     0.035918    0.035918     0.035918     0.035918
3  ...     0.497864     0.486621    0.486621     0.486621     0.486621
4  ...     0.267096     0.251408    0.251408     0.251408     0.251408

   MZ19987.596  MZ19990.235  MZ19992.874  MZ19995.513  Class
0     0.449296     0.449296     0.449296     0.449296      0
1     0.619718     0.619718     0.619718     0.619718      0
2     0.035918     0.035918     0.035918     0.035918      0
3     0.486621     0.486621     0.486621     0.486621      0
4     0.251408     0.251408     0.251408     0.251408      0
```

In [19]:
```
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))
```

We select the first 10 (y = 0) and last 10 (y=1) samples to form a new subset. X20 = np.vstack((X[:10], X[-11:-1])) np.savetxt('cancer.csv', X20, fmt='%.2f', delimiter = ',')

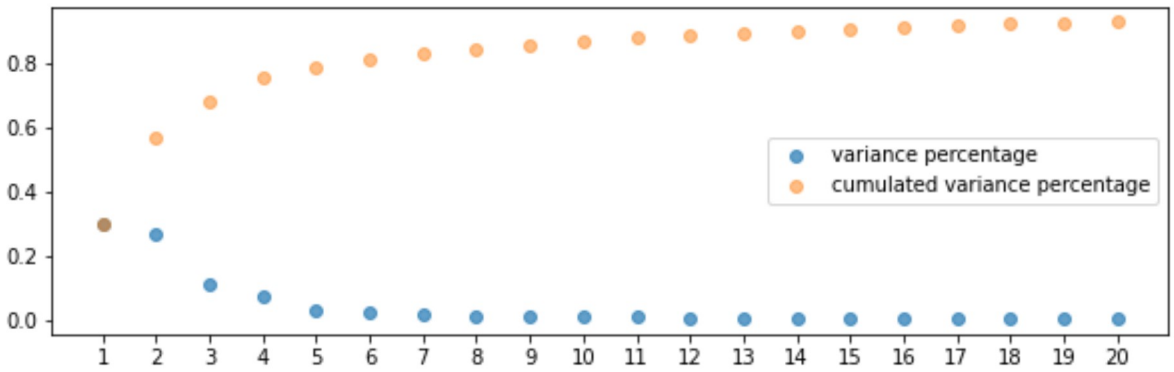
In [20]:
```
# %load draw_averaged_waveform.py
import matplotlib.pyplot as plt
import numpy as np

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()
```

In [21]:
```
draw_averaged_waveform(X, X_names)
```



DR by PCA

In [22]:
```
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, labe
plt.scatter(list(range(1,K+1)), pca.explained_variance_ratio_.cumsum(), alpha=
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.legend()
plt.show()

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




```
explained variance ratio: [0.29790796 0.2708818  0.11263393 0.07296365 0.03290
438 0.02353783
 0.01989609 0.01541524 0.01161813 0.01111271 0.00971808 0.00832409
 0.00761753 0.00701729 0.00654245 0.00564723 0.00466535 0.00446735
 0.00387544 0.00376708]
```

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 [27]:
```
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 [28]:
```
from pyDRMetrics import *
%matplotlib inline
```

In [29]:
```
drm = DRMetrics(X, Z, Xr)  # this can be quite time consuming if your dataset
```

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

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

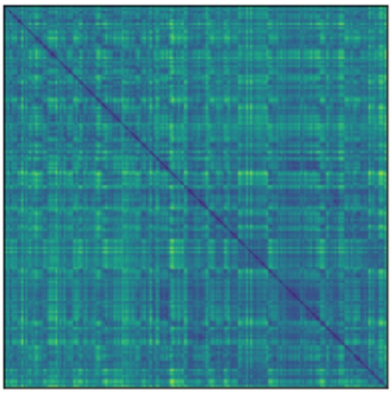
In [30]:
```
drm.mse, drm.rmse
```

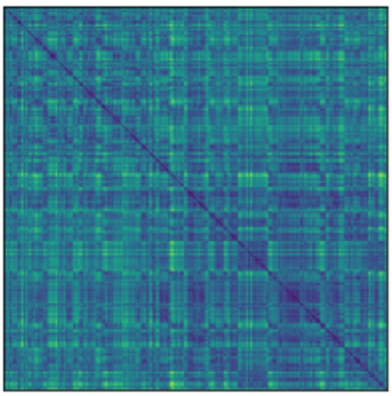
Out[30]:
```
(0.006834814256927592, 0.03545250484779821)
```

2.2. Distance Matrix

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

In [31]:
```
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).

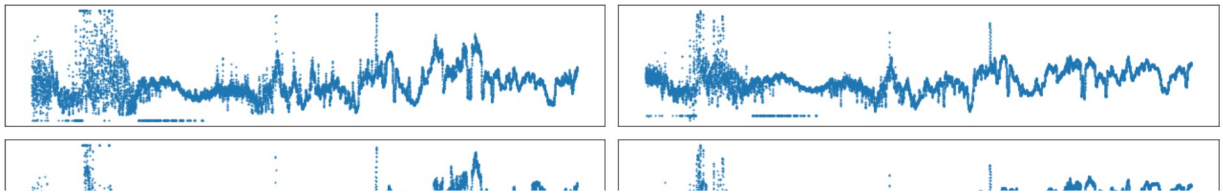
In [32]:
```
drm.Vr, drm.Vrs
```

Out[32]:
```
(0.02855748401029623, 0.028498020428687765)
```

2.4 Parallel Comparison of Specific Samples

In [33]:
```
# drm.visualize_reconstruction()

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

```
left column: original waveform          right column: recovered waveform
<Figure size 432x288 with 0 Axes>
```



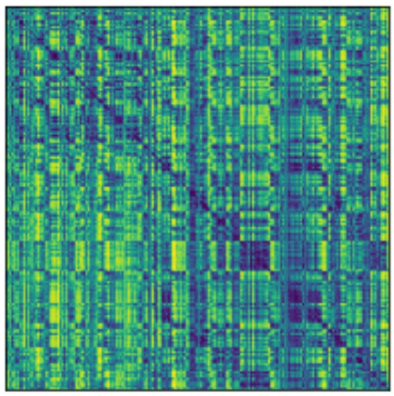
2.5 ranking matrix

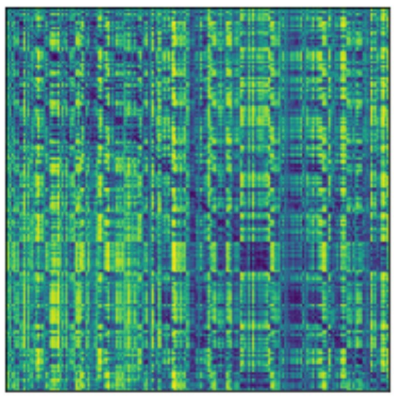
rij = |{ k : dik < dij or (dik = dij and 1 ≤ k < j ≤ n) }|, |A| denotes the number of elements in a set A

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

In [34]:
```
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 [35]:
```
drm.plot_ranking_matrix()
```

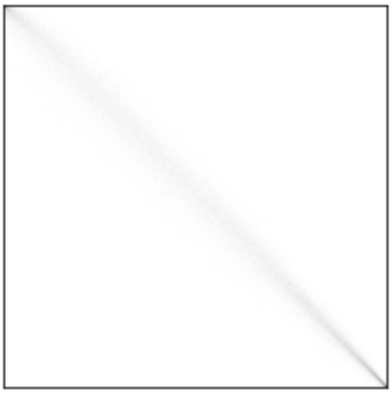




2.6 定义co-ranking matrix

The co-ranking matrix Q then has elements: qkl = |{(i, j) : r^ij = k and rij = l }|

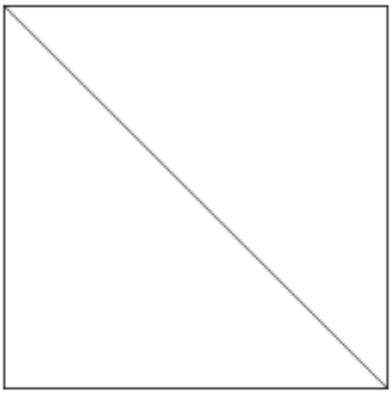
In [36]:
```
drm.plot_coranking_matrix()
```



In [37]:
```
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, qij 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 diagnal matrix
_ = visualize_matrix(np.eye(m)*m, cmap = 'gray_r')
```



2.6 Metrics derived from co-ranking matrix

Trustworthiness and continuity

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

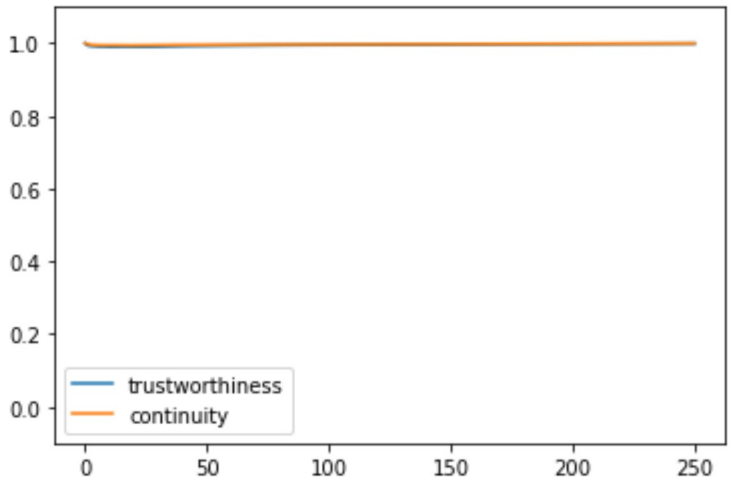
$$C(k) = 1 - \frac{1}{mk(m-k)} \sum_{i=1}^{m} \sum_{j \in \mathcal{UR}_k} \max(0, (r(i,j) - k))$$

```
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-intrusions. 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-extrusions. 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
```



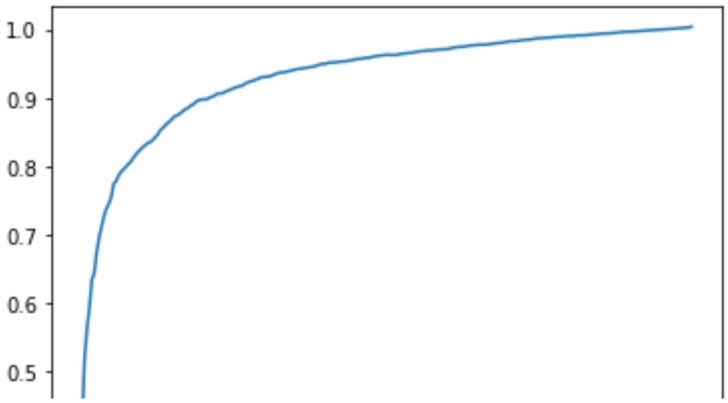
$Q_{NN}(k) = \frac{1}{kN}\sum_{i=1}^{k}\sum_{j=1}^{k} q_{ij}$

$Q_{NN}(k)$ 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 [41]:
```
plt.plot(list(range(1, len(drm.QNN)+1)), drm.QNN)
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x1d908888080>]
```

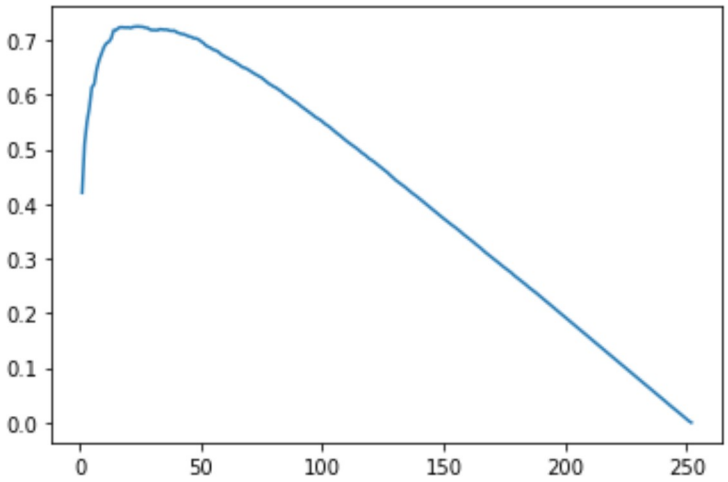


In [42]:
```
drm.AUC
```

Out[42]:
```
0.9322199373698454
```

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

Out[43]:
```
[<matplotlib.lines.Line2D at 0x1d908f7a5f8>]
```



In [44]:
```
drm.kmax # argmax(LCMC)
```

Out[44]:
```
23
```

In [45]:
```
drm.Qlocal,drm.Qglobal
```

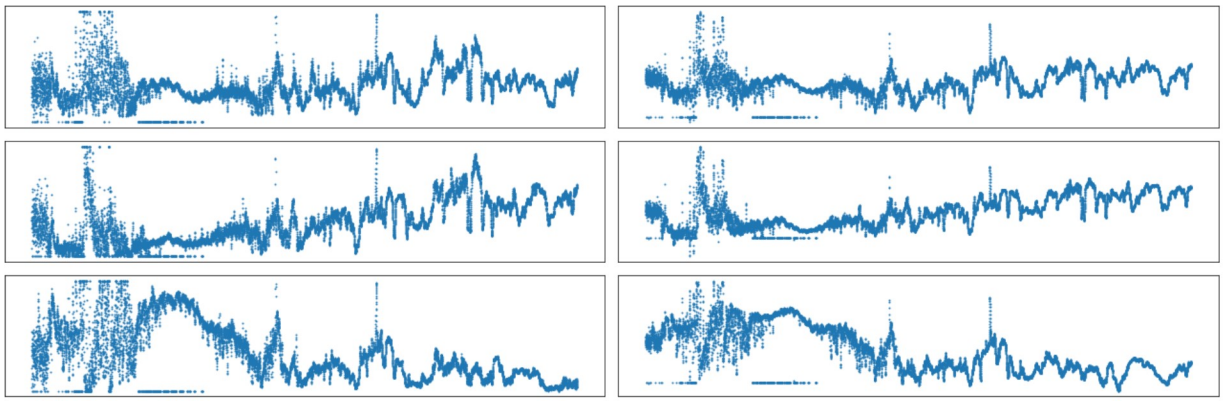
Out[45]:
```
(0.7167542677408645, 0.9540970207597336)
```

get_html() and get_json()

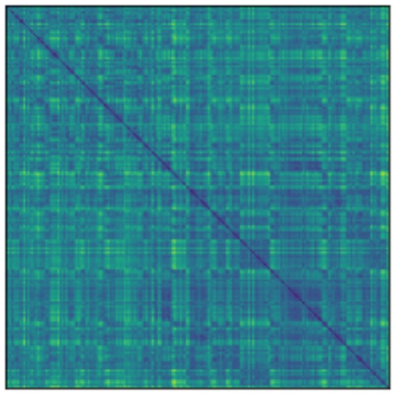
For use by web applications

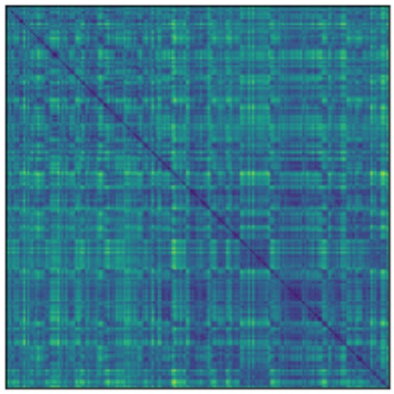
In [46]:
```
from IPython.display import display, HTML
# display(HTML(drm.get_html()))
drm.report()
```

```
--- Sample Reconstruction (X and Xr) ---
left column: original waveform           right column: recovered waveform
```

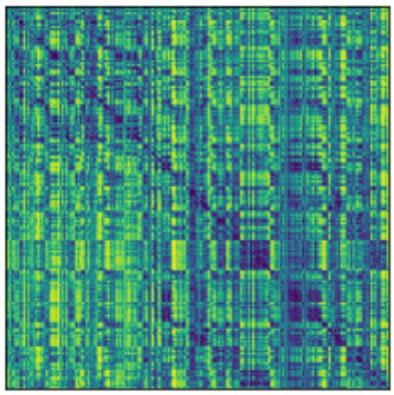


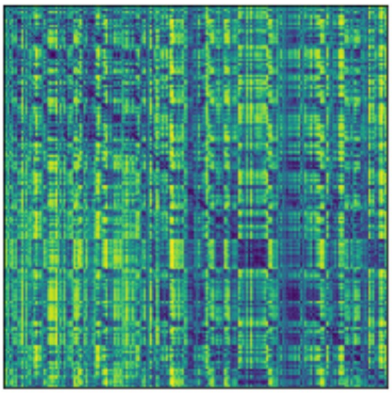
```
rMSE =  0.03545250484779821
--- Distance Matrices (D and Dz) ---
```



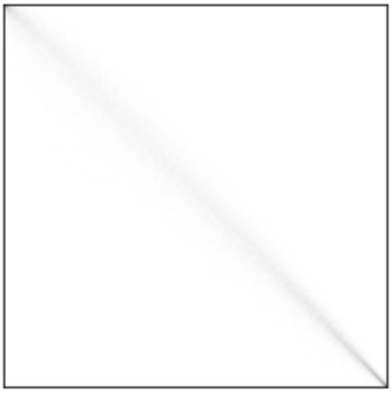


```
Residual Variance (using Pearson's r) =  0.02855748401029623
Residual Variance (using Spearman's r) =  0.028498020428687765
--- Ranking Matrices (R and Rz) ---
```

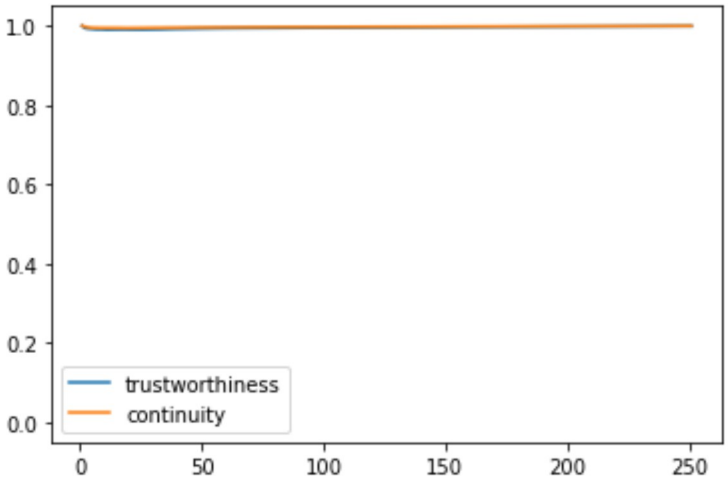




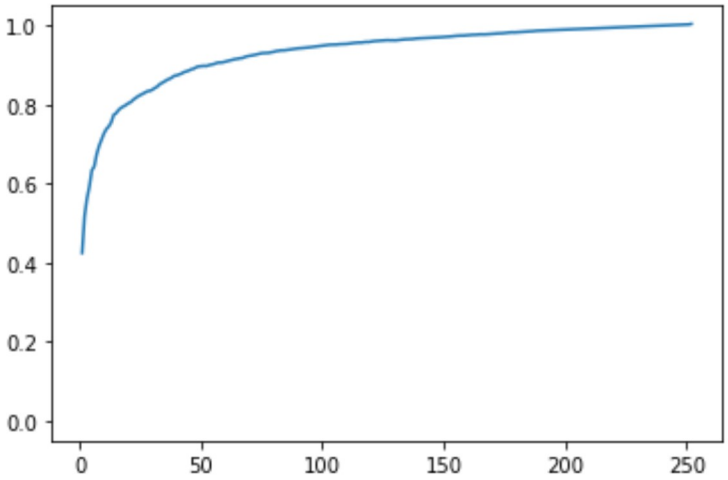
```
--- Co-ranking Matrix (Q) ---
```



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




```
AUC of T =  0.9957649593734504
AUC of C =  0.9975462929370625
--- QNN(k) Curve ---
```




```
AUC of QNN =  0.9322199373698454
--- LCMC(k) Curve ---
```

In [47]:
```
s = drm.get_json()
dic = json.loads(s)
dic.keys()
```

Out[47]:
```
dict_keys(['mse', 'ms', 'rmse', 'Vr', 'Vrs', 'AUC', 'kmax', 'Qlocal', 'Qglobal
', 'AUC_T', 'AUC_C'])
```

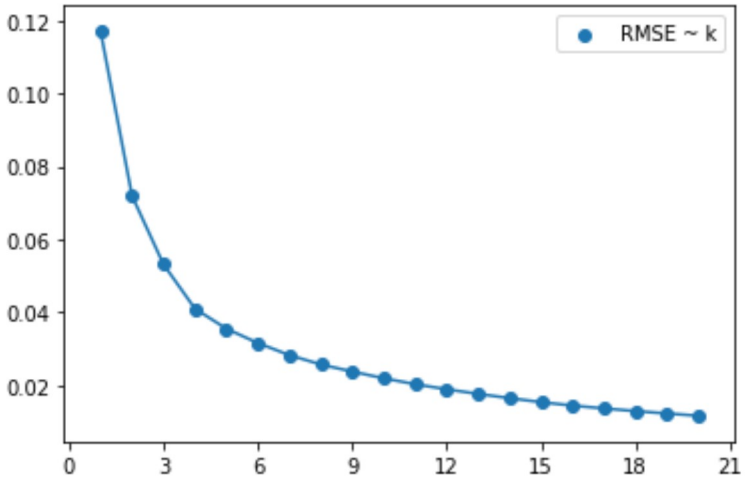
Dynamic Properties

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

In [ ]:
```
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)
```

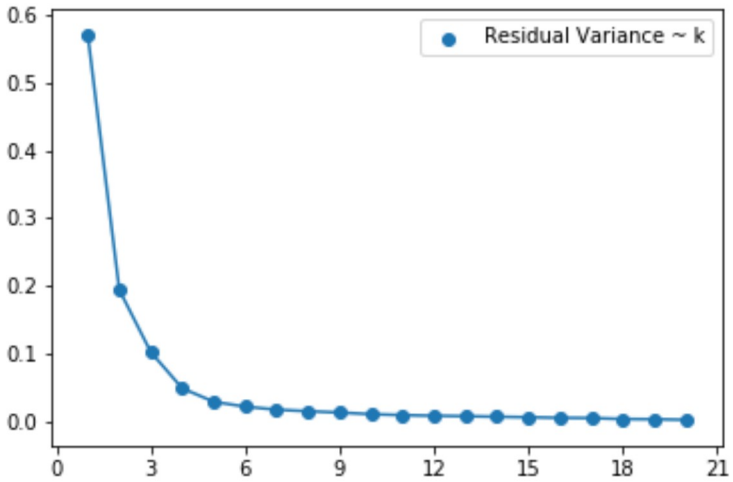
In [27]:
```
from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)),  rmses)
plt.scatter(list(range(1,21)),  rmses, label = 'RMSE ~ k')
plt.legend()
```

Out[27]: <matplotlib.legend.Legend at 0x1e01209d780>



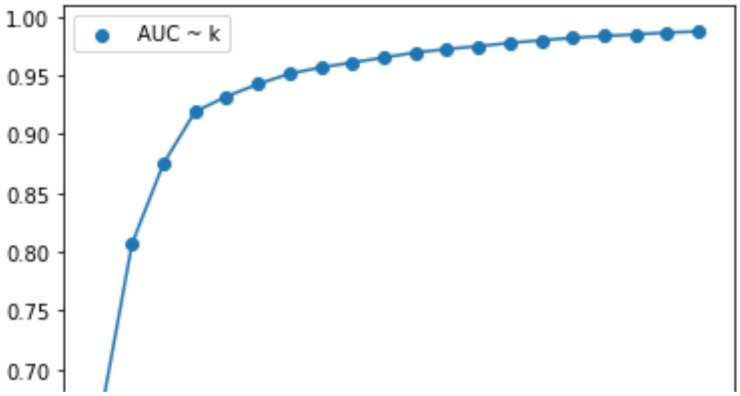
In [28]:
```
from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)),  Vrs)
plt.scatter(list(range(1,21)),  Vrs, label = 'Residual Variance ~ k')
plt.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x1e00dd73f60>



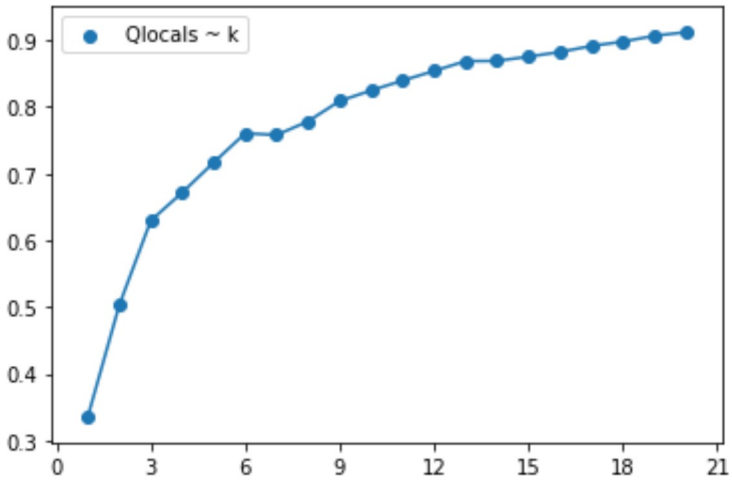
In [29]:
```
from matplotlib.ticker import MaxNLocator
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
plt.plot(list(range(1,21)),AUCs)
plt.scatter(list(range(1,21)),  AUCs, label = 'AUC ~ k')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x1e00dde4f60>



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 0x1e00de55f98>



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 0x1e00dec6d68>

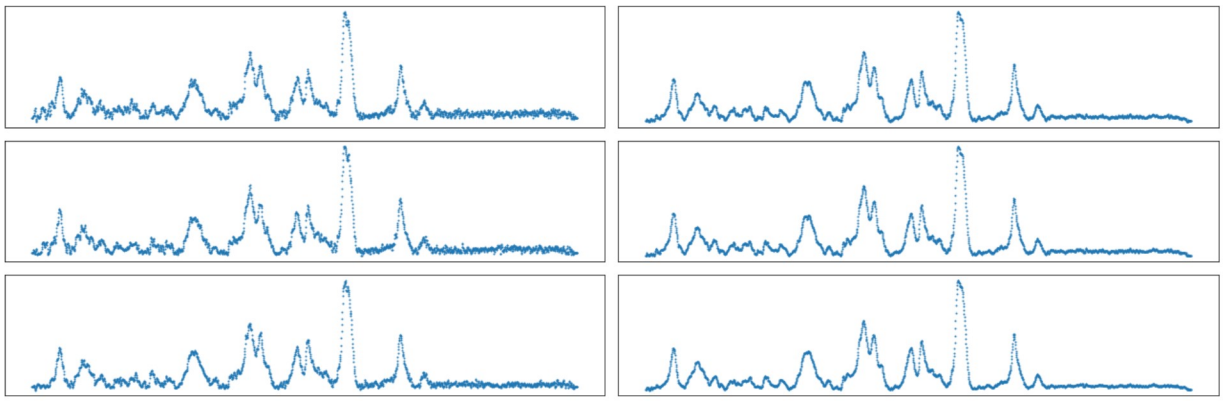
1.000

Case Study 2: Raman spectra

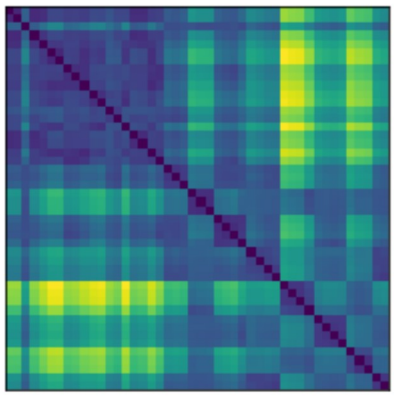
Test PCA and VQ

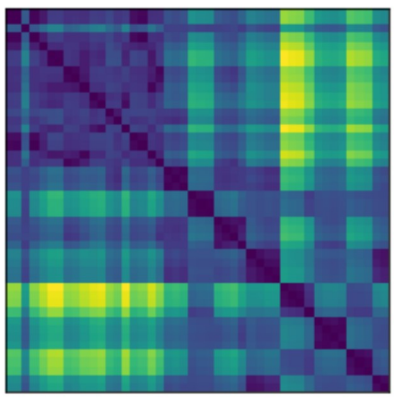
In [11]:
```
from pyDRMetrics import *

drm = DRMetrics.test('raman.csv', 3, dr = 'PCA')
drm.report()
```

```
--- Sample Reconstruction (X and Xr) ---
left column: original waveform          right column: recovered waveform
```

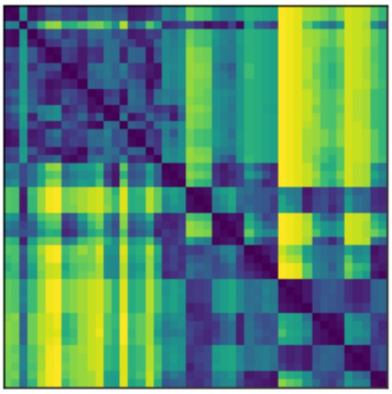


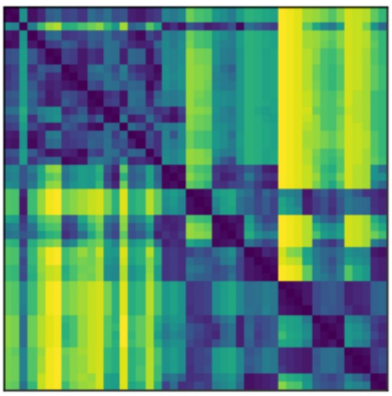
```
rMSE =  0.010637683165950566
--- Distance Matrices (D and Dz) ---
```



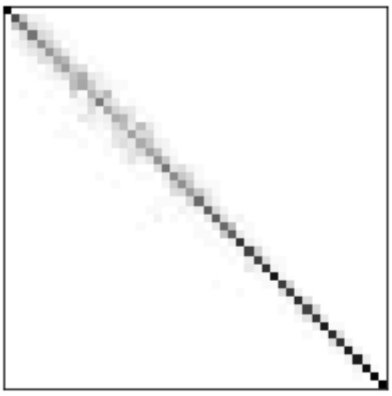


```
Residual Variance (using Pearson's r) =  0.011266375462861355
Residual Variance (using Spearman's r) =  0.00819300029711667
--- Ranking Matrices (R and Rz) ---
```

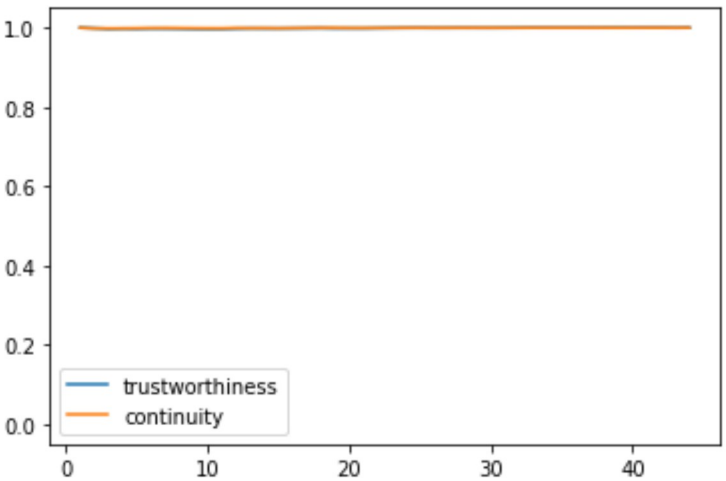




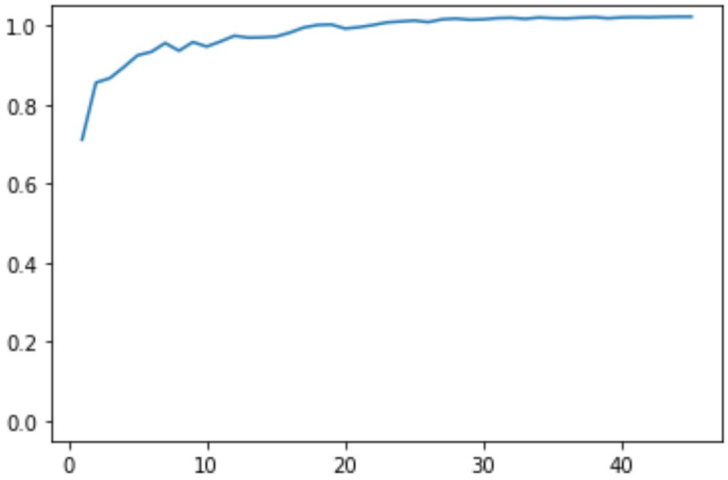
```
--- Co-ranking Matrix (Q) ---
```



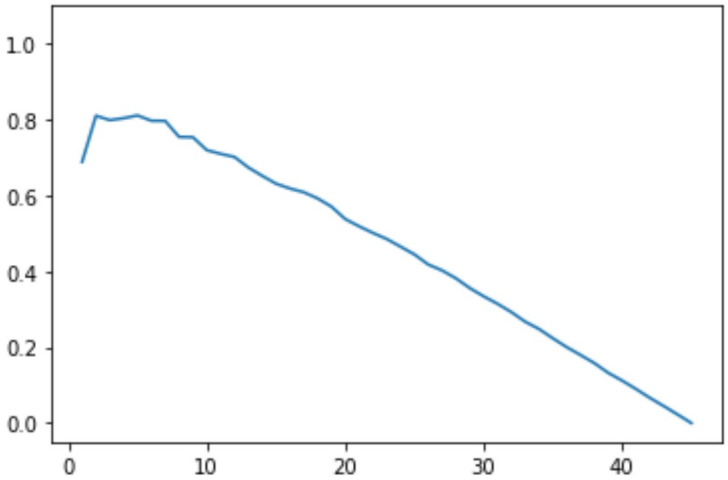
```
--- Trustworthiness T(k) and Continuity C(k) ---
```




```
AUC of T =  0.9989085475189007
AUC of C =  0.9995364059806525
--- QNN(k) Curve ---
```




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

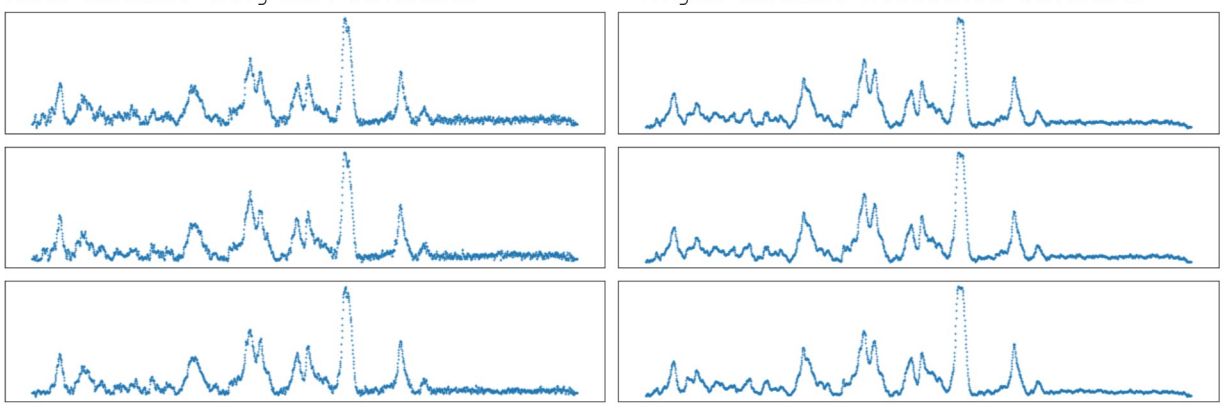



```
kmax (0-based index) =  4
```

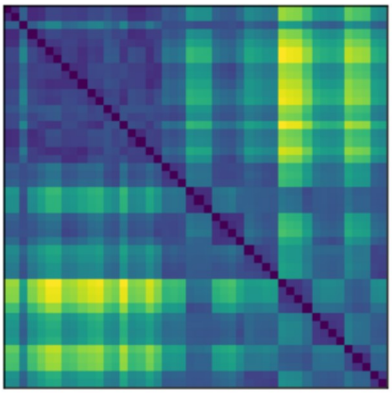
In [2]:
```
np.savetxt('raman_z.csv',drm.Z,fmt='%f',delimiter=',')
np.savetxt('raman_xr.csv',drm.Xr,fmt='%f',delimiter=',')
```

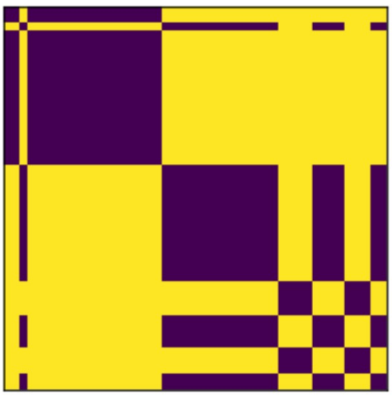
In [12]:
```
from pyDRMetrics import *

drm = DRMetrics.test('raman.csv', 3, dr = 'VQ') # VQ or Clustering is a speci
drm.report()
```

```
--- Sample Reconstruction (X and Xr) ---
left column: original waveform          right column: recovered waveform
```

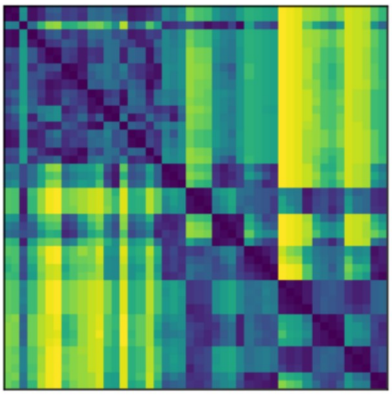


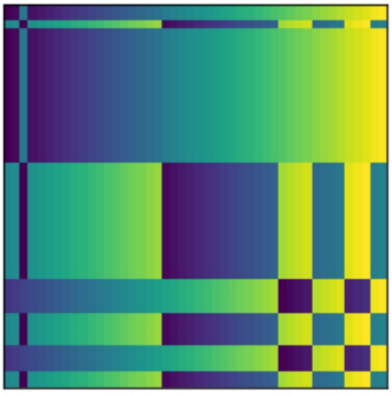
```
rMSE =  0.026951703948641496
--- Distance Matrices (D and Dz) ---
```



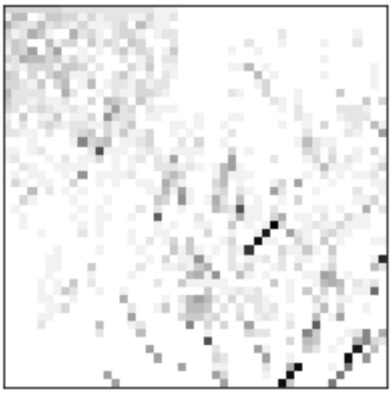


```
Residual Variance (using Pearson's r) =  0.5395973900914564
Residual Variance (using Spearman's r) =  0.4335241895993569
--- Ranking Matrices (R and Rz) ---
```

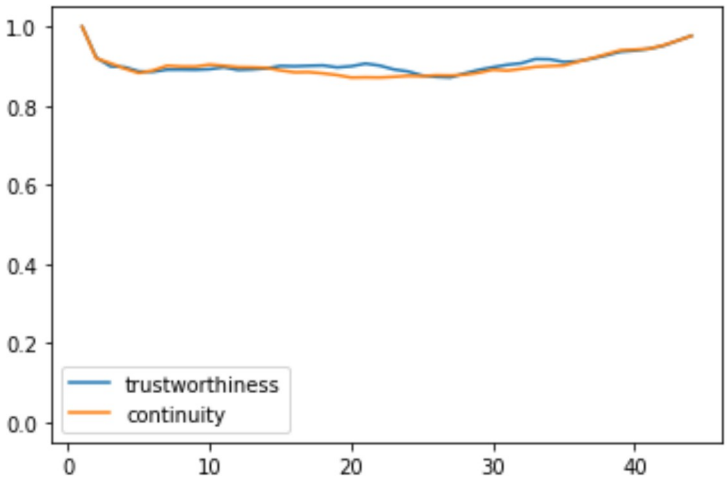




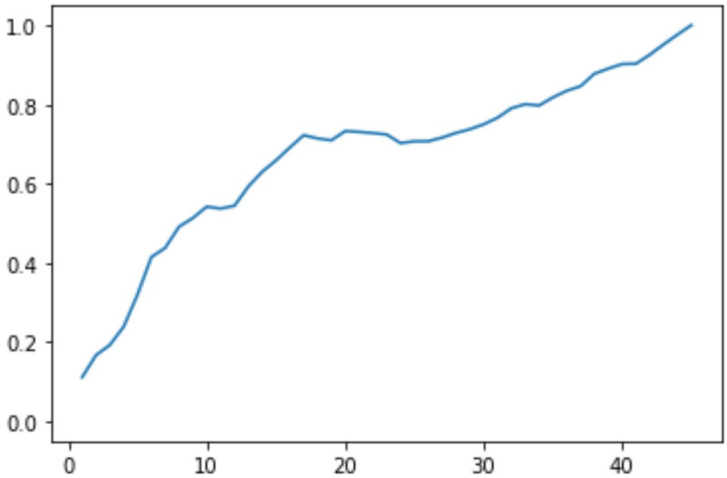
```
--- Co-ranking Matrix (Q) ---
```



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




```
AUC of T =  0.9077352091277376
AUC of C =  0.9026957642707345
--- QNN(k) Curve ---
```



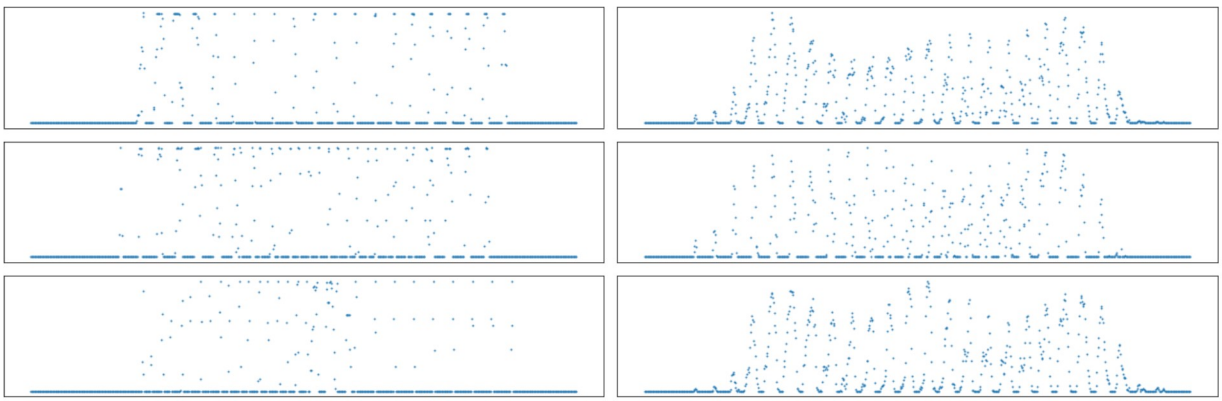

```
AUC of QNN =  0.6731868273940285
--- LCMC(k) Curve ---
```

Case Study 3: Handwritten digits data

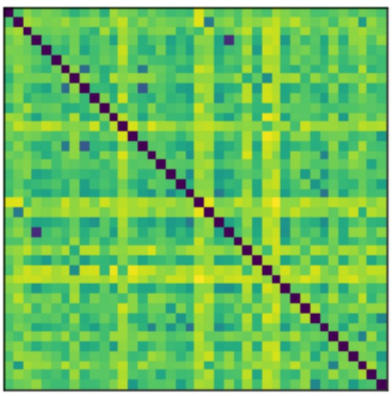
Test NMF and MDS

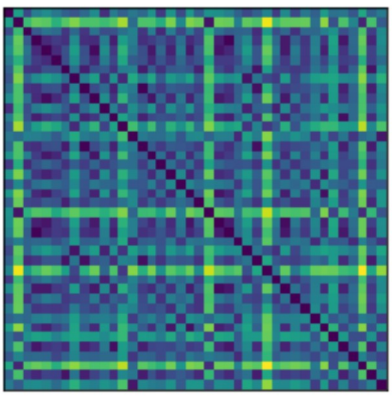
In [75]:
```
from pyDRMetrics import *

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

```
--- Sample Reconstruction (X and Xr) ---
left column: original waveform            right column: recovered waveform
```



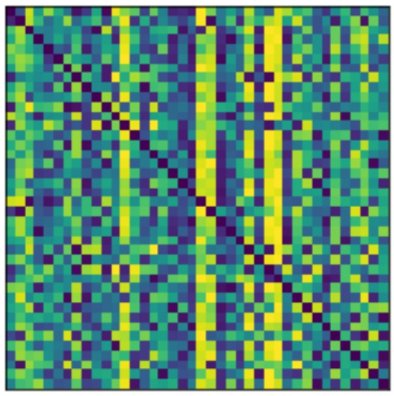
```
rMSE =  0.42257482992609124
--- Distance Matrices (D and Dz) ---
```

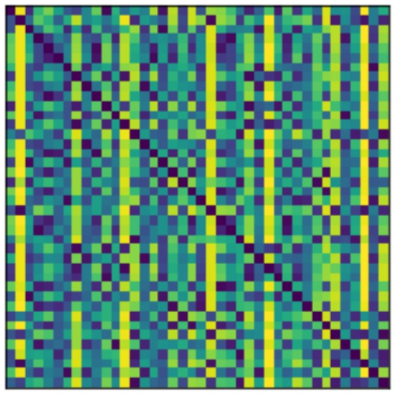




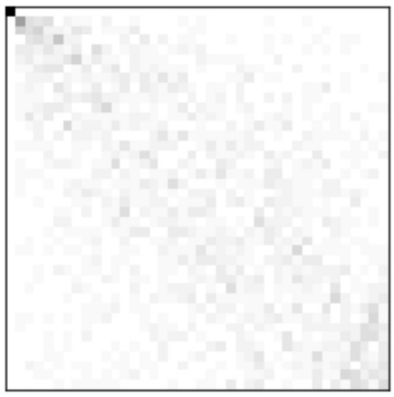
```
Residual Variance (using Pearson's r) =  0.6434414088216815
```

```
Residual Variance (using Spearman's r) =  0.5921476041366145
--- Ranking Matrices (R and Rz) ---
```

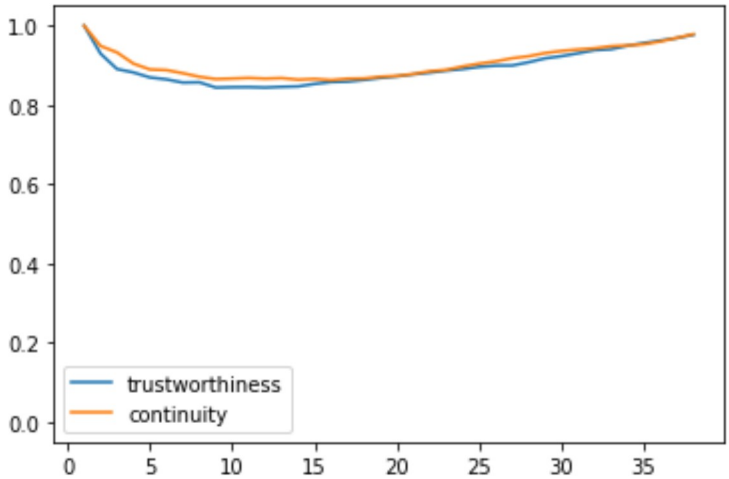




```
--- Co-ranking Matrix (Q) ---
```

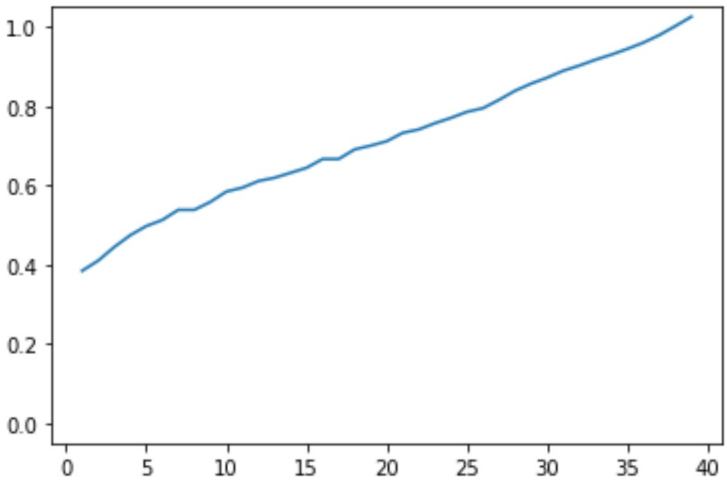


```
--- Trustworthiness T(k) and Continuity C(k) ---
```

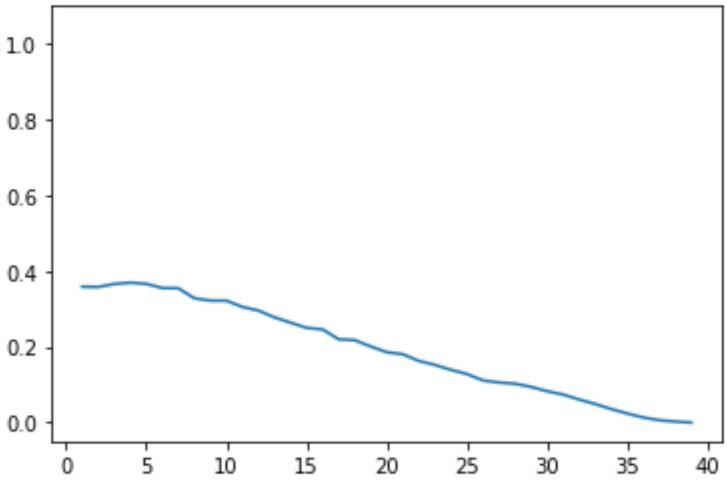



```
AUC of T =  0.894530432848915
```

```
AUC of C =  0.9059876420706509
```



```
AUC of QNN =  0.7178314960617955
--- LCMC(k) Curve ---
```



```
kmax (0-based index) =  3
Qlocal =  0.42841880341880345
Qglobal =  0.7351563166129361
```

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

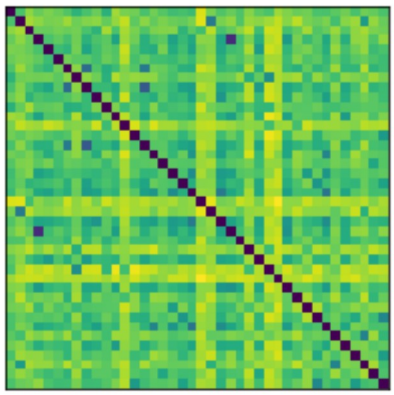
In [76]:

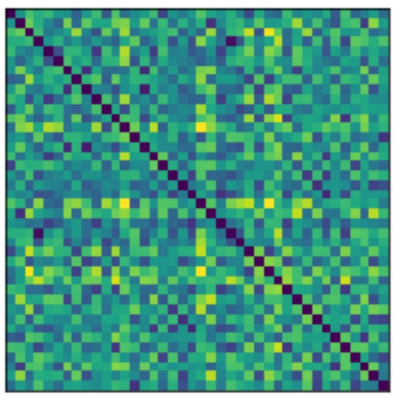
```
import pandas as pd

# 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 [68]:
```
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)
```

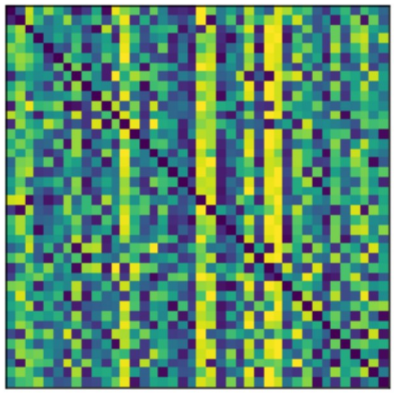
In [77]:
```
from pyDRMetrics import *

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

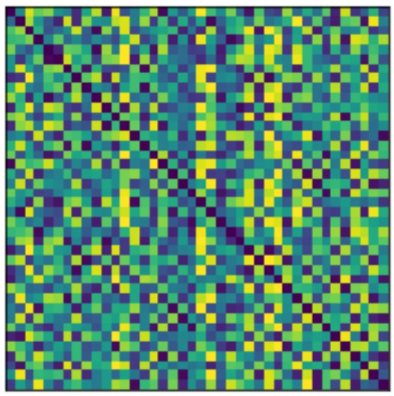
```
--- Distance Matrices (D and Dz) ---
```



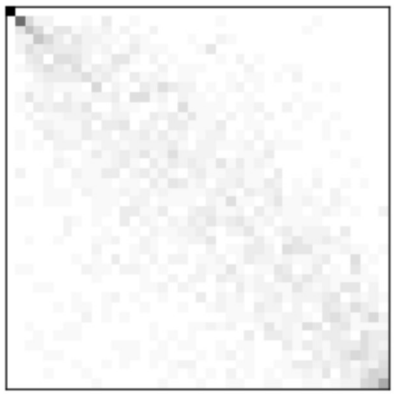


```
Residual Variance (using Pearson's r) =  0.4063183491387523
Residual Variance (using Spearman's r) =  0.42281485376894445
--- Ranking Matrices (R and Rz) ---
```

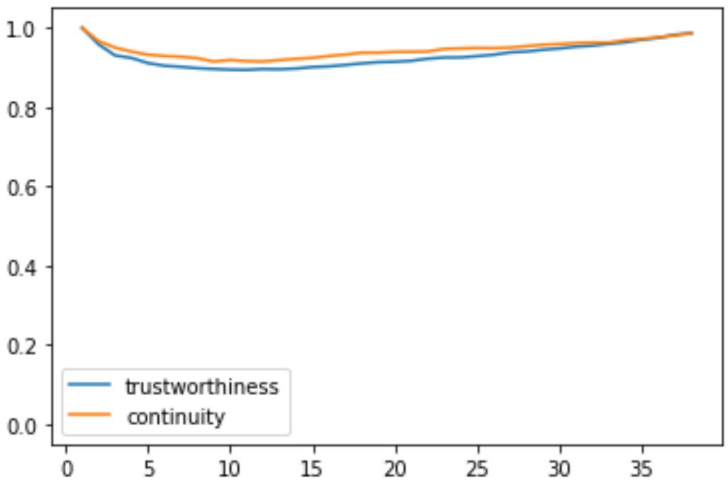




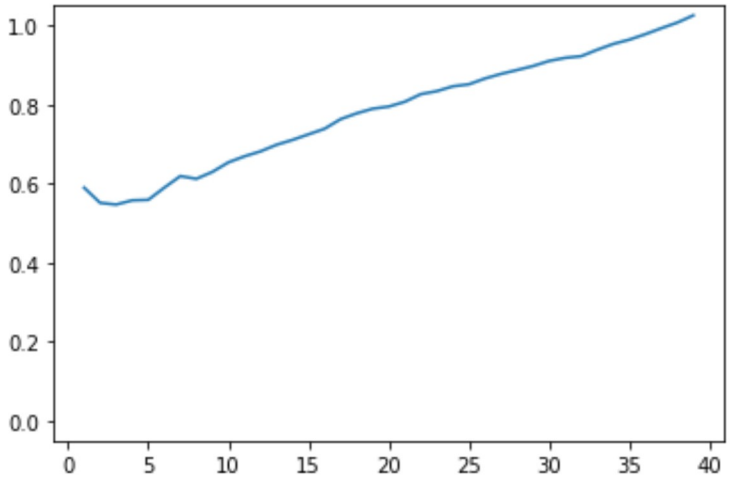
```
--- Co-ranking Matrix (Q) ---
```



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




```
AUC of T =  0.9292172530148941
AUC of C =  0.9455596799591003
--- QNN(k) Curve ---
```




```
AUC of QNN =  0.7836175846549172
--- LCMC(k) Curve ---
```

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

K = 3
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)
print('explained variance ratio:', pca.explained_variance_ratio_[0:5])

drm = DRMetrics(X, Z, Xr) # construct a DRMetrics object
print("Qlocal = ", drm.Qlocal) # get Qlocal
drm.report() # print out the summary

from IPython.display import display, HTML
display(HTML(drm.get_html())) # render the returned html string. You
can embed this HTML segment in a web page.
```

Appendix C: A web GUI tool based on pyDRMetrics