Information Theoretic-Based Quality Measures for Clustering

Points of Contact:
Barry Drake
250 14th St NW, Atlanta, GA, 30332
404-407-7547
barry.drake@gtri.gatech.edu

Tiffany Huang
250 14th St NW, Atlanta, GA, 30332
404-407-7222
tiffany.huang@gtri.gatech.edu

Submitted by:

Georgia Institute of Technology
A Unit of the University System of Georgia
Georgia Tech Research Institute
Atlanta, GA 30332

Contracting through:

Georgia Tech Applied Research Corporation
Centennial Research Building
Georgia Institute of Technology
Atlanta, GA 30332

29 May 2015

Introduction

Viewing a cluster as an information source is the motivation behind using information-based measures as a methodology to ascertain cluster quality. In general, we can consider a clustering *C* as information source *S*. The information emitted by *S* is the set of cluster labels *L*. Labels have a probability of being emitted with probabilities p(L_1) … p(L_N). From these probabilities the probability mass functions for discrete distributions can be derived. The probability mass functions are used to compute the entropies: marginal entropy, joint entropy, and conditional entropy. These entropies are the basic building blocks of all of the information-based measures.

Information-based quality measures (metrics, used interchangeably throughout) are an important tool. The interpretation and computation of the quality measures will be explained and discussed, followed by some test examples in the later sections of the document.

The code for running the measures is written in the Python programming language version 2.7.8 (or higher) and uses the numpy numerical libraries version 1.9.1 (or higher). A Matlab/Octave version is also available upon request.

A command line interface to the quality measure library is also included using getopt. At a command line typing the following shows the help or usage information:

```
-----------------------------------------------------------------------------------------------------------
python quality_measures.py –h (or –help)

        usage: python quality_measures.py [-h] ...

        Clustering quality measures code developed at the Georgia Institute of
            Technology/Georgia Tech Research Institute.

        arguments:
            -d  directory path   REQUIRED: Use this directory to access files

            -f  filename1        REQUIRED: Filename of the first set of cluster

            -g  filename2        REQUIRED: Filename of the second set of cluster labels

            -m  metric           Metric to use. MI,NMI(default),ID,VI,NMI,NID,NVI,all

            -r  <Boolean>        Remove outliers: False or True(default)

            -e  <Boolean>        Show entropies: False(default) or True

        optional arguments:
            -h, --help           Show this help message and exit
-----------------------------------------------------------------------------------------------------------
```

The command line mode is one way to run *quality_measures.py*. A standalone mode is also available by setting a variable in the code. This allows the code to be run from within an IDE, editor such as Emacs, or at the command line with '$ python quality_measures.py' and no other options. IDEs and editors generally also allow configuring options and running the command lin mode as well. This is explained in more detail below with examples of both modes of operation.

Anatomy of the Command Line

python quality_measures.py –d <directory to data files> -f <filename> -g <filename> -m <metric type> -r <remove outliers True/False>

Example command line and output (comparing same cluster label sets):

```
python quality_measures.py -d <path to data dir>/measures/examples
–f example1_1.csv -g example1_1.csv -m all -r True


metric:  all
fileName1:  example1_1.csv
fileName2:  example1_1.csv


Mutual Information:                    2.54003630382
Normalized Mutual Information:         1.0
Variation of Information:              4.4408920985e-16
Normalized Variation of Information:   2.22044604925e-16
Information Distance:                  0.0
Normalized Information Distance:       0.0
```

Caution

If you copy and paste command lines from this file, first copy and paste into a text editor to remove any special characters.

Supported Measures

Currently, *quality_measures.py* supports the following information-based measures:

- Mutual information (MI)
- Normalized mutual information (NMI)
- Information distance (ID)
- Normalized information distance (NID)
- Variation of Information (VI)
- Normalized Variation of Information (NVI)

Future Measures

- Adjusted Mutual Information (AMI) (work in progress)
- Adjusted Information Distance (AID) (work in progress)

The first list of measures above refers to those currently supported in *quality_measures.py*. The second list refers to those under development. These are measures that include a correction for chance (chance data associations) based on the expected value of the Mutual Information [1].

The measures are based on the entropies, marginal, joint, and conditional, computed from the two sets of cluster labels of interest. The computations of entropies will be discussed in a later section. These entropy computations and other methods form the core of the quality measures library. It should be noted that the core methods could also be used to compute other measures not in the library. Or they could be used for other applications, as could the measures themselves.

Potential numerical issues are trapped and report a warning; in some cases a measure has an alternative computation that is mathematically valid and that value is reported instead with a warning statement; if now alternative computation is available, then the measure is 'undefined'. This is especially important for the normalized versions of the measures.

In the following sections the entropy and measures computations are explained along with some other supporting methods. Code snippets are used to describe calls that would be part of a larger script. Example command lines are shown with inputs/outputs whenever that code has been made accessible from the command line. In general, only the actual measures have been made accessible from the command line. Other tools in the library are available via Python's import from within custom scripts, IDEs, or editors.

The Contingency Table

In order to compute entropies for the quality measures library a core method is called to build a contingency table for the two sets of cluster labels. Here we denote the two clustering label sets X and Y.

X \ Y	y_1	y_2	…	y_c	Sums
x_1	n_{11}	n_{12}	…	n_{1c}	$a_1 = \sum_{j=1}^{c} n_{1j}$
x_2	n_{21}	n_{22}	…	n_{2c}	$a_2 = \sum_{j=1}^{c} n_{2j}$
⋮	⋮	⋮	⋱	⋮	⋮
x_r	n_{r1}	n_{r2}	…	n_{rc}	$a_r = \sum_{j=1}^{c} n_{rj}$
Sums	$b_1 = \sum_{i=1}^{r} n_{i1}$	$b_2 = \sum_{i=1}^{r} n_{i2}$	…	$b_c = \sum_{i=1}^{r} n_{ic}$	$N = \sum_{i=1}^{r}\sum_{j=1}^{c} n_{ij}$

Table 1: Contingency Table. a: row sums; b: column sums; N: total

If x_i, x_k are cluster labels in X we have the following conditions

$$x_i, x_k \in X,\ x_i, x_k > 0,\ x_i \neq x_k,\ x_i < x_k \text{ if } i < k, \text{ for } i, k = 1,2, \dots, r.$$

Similarly, if y_j, y_l are cluster labels in Y

$$y_j, y_l \in Y,\ y_j, y_l > 0,\ y_j \neq y_l,\ y_j < y_l \text{ if } j < l,\ \text{for } j, l = 1,2, \dots, c.$$

The n_{ij} from the table represent the number of objects or samples that have x_i as the cluster labels from X, with $i = 1,2, \dots, r$ and y_j as the cluster labels from Y, with $j = 1,2, \dots, c$. The a_i and b_j are the row and column sums of the contingency table, respectively. Finally, N is the total number of objects or samples and the length of X and Y. The two sets should have the same length N (number of feature vectors), and the elements in a certain location in both X and Y should represent the labels for the same object. If any row(s)/column(s) of the contingency table are all zeros, they are removed. Such rows/columns are associated with missing labels and should not be used for the entropy computations.

The code snippet below instantiates an instance of the **contingency_table** class and builds the table.

```
cContingency = contingency_table(clusterLabels1, clusterLabels2)
contingencyTable = cContingency.contingency()
```

Note that this call is made in *quality_measures.py* as part of the entropy class constructor (in Python the __init__() method), which can also be called from custom scripts for standalone uses of the contingency table for other applications.

Entropy Computations

The quality measures described here are computed from various entropies. Since the cluster labels are drawn from a discrete probability distribution, the marginal, joint, and conditional entropies are based on *probability mass functions* (analogous to a *probability density function* for a continuous distribution). In all cases below we assume two sets of cluster labels.

It should be pointed out that the entropies are available using the –e option when calling *quality_measures.py* from the command line (OS command line in distinction to the Python command line). Other scripts or code can use the code snippets below by importing *quality_measures* (no extension) as a library from custom Python source code. Doing this from a Python command line and viewing available classes and methods displays the following: (Classes are in bold type; others are builtins (__xxx__) or unbound methods; In [num] is the Iron Python prompt; dir(library) outputs a list of variables, methods)

```
In [3]: import quality_measures
In [4]: dir(quality_measures)
Out[4]:
['__builtins__','__doc__','__file__','__name__','__package__',
'contingency_table', 'entropy', 'getopt', 'load_assignment_dir',
'load_assignment_files', 'macheps', 'main', 'measures', 'np',
'outlier_adjustments', 'quality_measures', 'sys', 'test_measures',
'usage_metrics']
```

The following shows how to access all computed entropies via an **entropy** object:

```
cEntropy = entropy(clusterLabels1, clusterLabels2)
allEntropies = cEntropy.get_entropies()
```

An example of making these calls from a script with a toy data set inputs and the outputs, after computing the entropies, is shown below. This example is also included in *quality_measures.py* as a standalone application *without* running the 'command line with options' by setting the variable `useMain = False` in the source code near the line '`if __name__ == "__main__":`' (the Pythonic idiom for standalone execution). Now run *quality_measures.py* in an IDE such as Spyder, in a text editor, or at the command line run:

```
python quality_measures.py
```

Input:

```
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]


cEntropy = entropy(clusterLabels1, clusterLabels2)
allEntropies = cEntropy.get_entropies()

print 'entropy of first label set H(U)  = ', allEntropies['entropy1']
print 'entropy of second label set H(V) = ', allEntropies['entropy2']
print 'joint entropy H(U,V)             = ', allEntropies['joint']
print 'conditional entropy H(U|V)       = ', allEntropies['conditional1']
print 'conditional entropy H(V|U)       = ', allEntropies['conditional2']
```

Output:

```
entropy of first label set H(U)  =  2.54003630382
entropy of second label set H(V) =  2.44101527803
joint entropy H(U,V)             =  2.63905732962
conditional entropy H(U|V)       =  0.198042051589
conditional entropy H(V|U)       =  0.0990210257943
```

Note that in the above the H((*) is consistent with the conceptual standard notations for entropy H of arbitrary label sets U,V. The numerical values were computed from the labels of two cluster runs on the same data set, the toy data.

The return from this method of obtaining all of the computed entropies is a dictionary, allEntropies (see above), containing all of the computed entropies with the 'key:value' as the 'entropy type:computed value' pairs.

Also, from within Spyder the command line mode of *quality_measures.py* can be run with options by setting the run configuration with the Run tool dropdown. Running from within an IDE with `useMain = True` displays the usage or help information (-h option at the command line).

Marginal Entropy (usually just called entropy)

The following code snippet computes the marginal entropies of two clustering label sets:

```
cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
entropy1, entropy2 = cEntropy.compute_entropies()
```

This snippet computes one or the other of $H(U)$ or $H(V)$ depending on whether the input is 1 for $H(U)$ or 2 for $H(V)$:

```
cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
entropy = cEntropy.compute_entropy(<1 or 2>) # default is 1
```

Let $H(U)$ and $H(V)$ be the marginal entropy of the cluster label sets of clusterings U and V, respectively. Based on the contingency table shown earlier, the $H(U)$ and $H(V)$ are computed as [1]:

$$H(U) = -\sum_{i=1}^{r} \frac{a_i}{N} ln\,(\frac{a_i}{N})$$

$$H(V) = -\sum_{j=1}^{c} \frac{b_j}{N} ln\,(\frac{b_j}{N})$$

Note that for any arbitrary cluster label set W, $H(U)$ is in the open interval 0 to ln(*N*), i.e.,

$$H(U) \in (0, \ln(N))$$

From communication or information theory, $H(U)$ is the average amount of information in bits that are needed to encode the cluster labels in U [1,4].

Using the same toy example above:

```
Input:
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]


cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
entropy1, entropy2 = cEntropy.compute_entropies()

print 'entropy of first label set H(U)  = ', entropy1
print 'entropy of second label set H(V) = ', entropy2

Output:
entropy of first label set H(U)  =  2.54003630382
entropy of second label set H(V) =  2.44101527803
```

Joint Entropy

The following code snippet computes the joint entropy of two clustering label sets:

```
cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
jointEntropy = cEntropy.joint_entropy()
```

Let $H(U,V)$ be the joint entropy of the cluster label sets U and V. Based on the contingency table shown earlier, the $H(U,V)$ is computed as [1,4]

$$H(U,V) = -\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{n_{ij}}{N}\ln\left(\frac{n_{ij}}{N}\right)$$

For any two arbitrary cluster label sets U and V, the joint entropy is bounded within an open interval as follows

$$H(U,V) = H(V,U) \in (0, \ln(N)).$$

Using the example above:

```
Input:
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]


cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
jointEntropy = cEntropy.joint_entropy()

print 'joint entropy H(U,V)  = ', jointEntropy


Output:
joint entropy H(U,V) =  2.63905732962
```

Conditional Entropy

The following code snippet computes the conditional entropies of two clustering label sets:

```
cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
ce1, ce2 = cEntropy.conditional_entropies()
```

Let $H(U|V)$ be the conditional entropy of the cluster label set U conditioned on the set V. Based on the contingency table shown earlier, $H(U|V)$ is computed as [1]

$$H(U|V) = -\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{n_{ij}}{N}\ln\left(\frac{n_{ij}}{b_j}\right).$$

If $H(V|U)$ was desired, then it was computed as

$$H(V|U) = -\sum_{j=1}^{c}\sum_{i=1}^{r}\frac{n_{ij}}{N}\ln\left(\frac{n_{ij}}{a_i}\right)$$

For any two arbitrary cluster label sets U and V, the conditional entropy is bounded within the open interval

$$H(U|V) \in (0, \mathrm{H}(U))$$

From communication theory, $H(U|V)$ is the average amount of information in bits that were needed to encode the cluster labels in U knowing V [1,4]. With knowledge of V fewer bits are required to transmit the information in U.

Note that these conditional entropies are not symmetric. The first, $H(U|V)$, uses the knowledge of V captured in the column sums b_j while the second, $H(V|U)$, uses row sums a_i.

Using the example above:

```
Input:
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]

cEntropy = entropy(self.clusterLabels1, self.clusterLabels2)
ce1, ce2 = cEntropy.conditional_entropies()

print 'conditional entropy H(U|V)  = ', ce1
print 'conditional entropy H(V|U)  = ', ce2

Output:
conditional entropy H(U|V) =  0.198042051589
conditional entropy H(V|U) =  0.0990210257943
```

The Quality Measures

As mentioned, the quality measures supported are all derived from the entropies computed earlier. Hence, the measures presented in this section are defined on the basis of the entropies of the previous section. The entropies are all pre-computed within the constructor of the measures class when a measure object is created.

The entropies are also available as members of the measures class via a *get_entropies()* method, which is accessed through an **entropy** object (as in the previous section) or a **measures** object. When accessing through the measures object, this is done via the *getEntropyPtr()* method, i.e., the **measure** object contains an **entropy** object.

To access all entropies from a **measures** object:

```
cMeas = measures(clusterLabels1,clusterLabels2)
entPtr = cMeas.getEntropyPtr()
allEntropies = entPtr.get_entropies()
```

As in the previous section, the return from this method of obtaining all of the computed entropies is a dictionary containing all of the computed entropies with the ‘key:value’ as the ‘entropy type:computed value’ pairs. For completeness we show the results of code running the above snippet on the toy label sets of the previous section (results are the same):

Input:

```
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]


cEntropy = entropy(clusterLabels1, clusterLabels2)
allEntropies = cEntropy.get_entropies()

print 'entropy of first label set H(U)  = ', allEntropies['entropy1']
print 'entropy of second label set H(V) = ', allEntropies['entropy2']
print 'joint entropy H(U,V)             = ', allEntropies['joint']
print 'conditional entropy H(U|V)       = ', allEntropies['conditional1']
print 'conditional entropy H(V|U)       = ', allEntropies['conditional2']
```

Output:

```
entropy of first label set H(U)  =  2.54003630382
entropy of second label set H(V) =  2.44101527803
joint entropy H(U,V)             =  2.63905732962
conditional entropy H(U|V)       =  0.198042051589
conditional entropy H(V|U)       =  0.0990210257943
```

The same toy example can be run to compute all of the measure outputs for the two sets of class labels of this example. The code snippet to do this is:

```
cMeas = measures(clusterLabels1,clusterLabels2)
allEntropies = cMeas.getAllMeasures()
```

```
Input:
clusterLabels1 = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]
clusterLabels2 = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]

cMeas = measures(clusterLabels1,clusterLabels2)
allMeasures = cMeas.getAllMeasures()

print 'mutualInformation        = ', dAllMeas['MI']
print 'normalizedMutualInfo     = ', dAllMeas['NMI']
print 'informationDistance      = ', dAllMeas['ID']
print 'normalizedInfoDistance   = ', dAllMeas['NID']
print 'variationInformation     = ', dAllMeas['VI']
print 'normalizedVariationInfo1 = ', dAllMeas['NVI']

Output:
Mutual Information:                   2.34199425223
Normalized Mutual Information:        0.94054724171
Information Distance:                 0.198042051589
Normalized Information Distance:      0.0779681972618
Variation of Information:             0.297063077383
Normalized Variation of Information:  0.112564086445
```

Mutual Information

There are several ways one could compute the mutual information between two sets of cluster labels U and V. The range of mutual information is $[0, \min(H(U), H(V))]$. According to [1,4], mutual information measures the information that U and V share, and the higher the mutual information, the more useful the information in V can be used to predict the cluster labels in U and vice-versa. A higher value of mutual information also indicates that the two sets of cluster labels U and V are more non-randomly associated with each other [2].

Let $MI(U,V)$ be the mutual information between the cluster label sets U and V, then $MI(U,V)$ is computed as follows [1,4]

$$MI(U,V) = H(U) - H(U|V) = H(V) - H(V|U)$$

The inputs for computing the mutual information are $H(U)$ and $H(U|V)$ or $H(V)$ and $H(V|U)$, for any two arbitrary sets of cluster labels U and V.

The mutual information can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
NMI = cMeas.mutual_information()
```

or from the command line using something like

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m MI
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  MI
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Mutual Information:  2.34199425223
```

Normalized Mutual Information

According to [1], there are several ways to normalize the mutual information. One of the approaches is dividing the mutual information by the joint entropy of the two sets of cluster labels. One may also divide the mutual information by the square root of the two entropies from the two sets of cluster labels multiplied together. The normalization of mutual information between two sets of cluster labels X and Y $NMI(U,V)$ is chosen as the most commonly used approach

$$NMI(U,V) = \frac{MI(U,V)}{\sqrt{H(U) * H(V)}}$$

As a result, the range of normalized mutual information $NMI(U,V)$ is the open interval $(0,1)$. Based on the interpretations of mutual information, normalized mutual information with a value closer to 1.0 indicates that the two clusterings are nearly identical. This needs to be corroborated by other measures.

The inputs for computing the normalized mutual information $NMI(U,V)$ are $H(U)$, $H(V)$ and $MI(U,V)$, the mutual information.

The normalized mutual information can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
NMI = cMeas.normalized_mutual_information()
```

or from the command line running the following:

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m nmi
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  NMI
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Normalized Mutual Information:  0.94054724171
```

For these clustering labels we can observe that since the NMI is close to 1.0, the clusterings apparently have a high likelihood of that the two clusterings are similar.

Information Distance

The information distance measure is the distance, in terms of information, between two sets of cluster labels X and Y. A value closer to 0 indicates that the distance in terms of information between X and Y is closer or the two clusterings are more similar. The range of information distance is in the open interval $(0, \ln(N))$, where N is the number of objects or samples.

Let $ID(U,V)$ be the information distance between two cluster labels sets U and V, then $ID(U,V)$ is computed as [1]

$$ID(U,V) = \max\big(H(U), H(V)\big) - MI(U,V)$$

The inputs for computing the information distance $ID(U,V)$ are $H(U)$, $H(V)$ and $MI(U,V)$, the mutual information.

The information distance can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
ID = cMeas.information_distance()
```

or from the command line running the following:

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m ID
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  ID
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Information Distance:  0.198042051589
```

For these clustering labels we can observe that since the ID is small, the clusterings apparently are close in terms of information content. Together with the NMI measure there is more evidence that the clusterings and the associations are not simply by chance.

Normalized Information Distance

The normalization of information distance between two sets of cluster labels X and Y is defined by

$$NID(U,V) = \frac{ID(U,V)}{\max\big(H(U),H(V)\big)} = \frac{\max\big(H(U),H(V)\big) - MI(U,V)}{\max\big(H(U),H(V)\big)} = 1 - \frac{MI(U,V)}{\max\big(H(U),H(V)\big)}$$

The range of normalized information distance $NID(U,V)$ is in the open interval $(0,1)$. Based on the interpretations of information distance, normalized information distance with a value closer to 0 indicates that the distance in terms of information between U and V are closer or the clusterings are more similar.

The inputs for computing the normalized information distance $NID(U,V)$ are $H(U)$, $H(V)$ and the mutual information $MI(U,V)$ between cluster labels U and V.

The normalized information distance can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
ID = cMeas.normalized_information_distance()
```

or from the command line running the following:

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m NID
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  NID
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Normalized Information Distance:  0.0779681972618
```

Here the NID is small, providing more evidence that the two clusterings are close in information content and not due to chance.

Variation of Information

According to [3], there are several ways to compute the variation of information based on the entropies. The variation of information represents a measure of the distance between two sets of cluster labels and is a true mathematical metric. The value is 0 if and only if the two sets of cluster labels are the same. The range of variation of information is in the open interval $(0, \ln(N))$, where N is the number of objects or samples.

The variation of information, $VI(U,V)$, between the cluster label sets U and V is computed as [1]

$$VI(U,V) = H(U,V) - MI(U,V)$$

The inputs for computing the variation of information $VI(U,V)$ are the joint entropy, $H(U,V)$, and the mutual information, $MI(U,V)$.

The variation of information can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
VI = cMeas.variation_information()
```

or from the command line running the following:

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m vi
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  VI
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Variation of Information:  0.297063077383
```

Here the VI is relatively small, providing more evidence that the two clusterings are close in information content and not simply due to chance.

Normalized Variation of Information

The normalization variation of information between two sets of cluster labels U and V is defined by [1]

$$NVI(U,V) = \frac{VI(U,V)}{H(U,V)} = \frac{H(U,V) - MI(U,V)}{H(U,V)} = 1 - \frac{MI(U,V)}{H(U,V)}$$

The range of normalized variation of information $NVI(U,V)$ is the open interval $(0,1)$. Based on the interpretations of variation of information, normalized variation of information with a value closer to 0 indicates that the two clusterings are similar.

The inputs for computing the normalized variation of information $NVI(U,V)$ are $VI(U,V)$ and $H(U,V)$, variation of information and the joint entropy, or $MI(U,V)$ and $H(U,V)$, i.e., the mutual information and the joint entropy.

The normalized variation of information can be computed using the code snippet in custom Python code

```
cMeas = measures(clusterLabels1,clusterLabels2)
NVI = cMeas.normalized_variation_information()
```

or from the command line running the following:

```
python quality_measures.py -d <path to data>/measures/examples -f
      example2_1.csv -g example2_2.csv -m nvi
```

This assumes that the data directory and the code directory are at the same level in the directory tree. The output should look like the following:

```
metric:  NVI
fileName1:  <path to data>/example2_1.csv
fileName2:  <path to data>/example2_2.csv


Normalized sVariation of Information:  0.112564086445
```

Here again the NVI is relatively small. Based on the results, and realizing this is a small number of cluster labels, the evidence suggests that the two clusterings are close in both information distance and information content. Thus, the similarities are not simply due to chance.

Special Considerations for Outliers

One of the main assumptions within *quality_measures.py* is regarding the values of the cluster labels. It was assumed that the values of the cluster labels were nonnegative integers, and the minimum value supported was 0. This was due to the construction of the contingency table. If any of the cluster labels were indicated with a negative value, those labels are not be placed in the contingency table since the indexing in Python is 0-based. Thus, a special function was created to map the negative values to locations in the contingency table that would be consistent with normal operation of the code for all nonnegative cluster labels. With this in mind, in order to support the particular case of outliers encoded as -1, a supporting function in *quality_measures.py* was created to either change the value -1 to a new positive integer, placing the outliers in a cluster of outliers, or remove those outliers from the computations of the measures, depending on whether one wished to include the outliers for computing the measures or not. Thus, it is also possible to study the effects of including the outliers on the clustering results using the quality measures. Other methods for including outliers will be further explored as well.

Test Case Examples

Test cases will now be described for testing *quality_measures.py*. These illustrate how the to use the measures and produce the expected results. To begin, the first example consisted of two cluster labels sets with the two label sets consisting of the same sequence of integers. This illustrates what the measure results show when there is exact similarity between the clustering results. The quality measures computed here are meant for testing/debugging purpose only and serve to examine some simplistic results.

The second example uses randomly generated cluster labels. Although these label sets have no particular meaning with regard to actual cluster results, they illustrate the use of the tools. Files are provided with the label sets, one file for each.

The third example uses two sets of cluster labels from ingesting data from *news20_assignments_32_run1.csv* and *news20_assignments_32_run3.csv*, respectively, where the cluster labels from these files were the results of running an actual clustering algorithm called HierNMF2, based on a special case of the Nonnegative Matrix Factorization (NMF) [6]. These cluster results contained outliers as mentioned earlier that needed to be removed, or the labels needed to be converted to a new positive integer before constructing the contingency table as the first step. The results here are discussed and explained with interpretations of the outputs.

Example 1:
This example consisted of the same set of cluster labels U of the same elements. We use this to test that the metrics are computing the expected values for the same inputs.

$U = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]$ in file example1_1.csv
$V = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]$

Mutual information:	$MI(U,V) = 2.54004$
Normalized mutual information:	$NMI(U,V) = 1.0$
Information distance:	$ID(U,V) = 0.0$
Normalized Information distance:	$NID(U,V) = 0.0$
Variation information:	$VI(U,V) = 4.4\text{e-}16$
Normalized variation of information:	$NVI(U,V) = 2.2\text{e-}16$

These results provide us with a validation of the metrics library in that for two sets of identical labels, $U = V$, the NMI is expected to be 1.0 and the information distance to be 0.0 as verified in the table above. Also, the variation of information and the normalized versions should be close to zero as shown in the table above.

Run this example at the command line with:

```
python quality_measures.py -d <path to data>/measures/examples –f
example1_1.csv -g example1_1.csv -m all -r True
```

Example 2:
The example consisted of two sets of cluster labels U and V with elements

$U = [0,1,3,0,6,5,4,14,10,23,22,11,17,20]$ in file example2_1.csv

$V = [1,0,5,4,3,0,9,19,23,26,21,19,20,28]$ in file example2_2.csv

Note that the cluster labels were all non-negative, repeatable, without any particular order, and not all of the integers were appearing from 0 to the largest cluster label of the set. Here $N = 14.$

As the result by running *quality_measures.py*,

Mutual information:	$MI(U,V) = 2.34199$
Normalized mutual information:	$NMI(U,V) = 0.94054$
Information distance:	$ID(U,V) = 0.19804$
Normalized Information distance:	$NID(U,V) = 0.07797$
Variation information:	$VI(U,V) = 0.29706$
Normalized variation of information:	$NVI(U,V) = 0.112564$

The results are from two arbitrary cluster label sets X and Y. NMI is close to 1.0 and the distance measures are relatively small suggesting that the two clustering results are "fairly" close. Detailed analysis on an actual clustering result produced from HierNMF2 is presented in the next example.

Run this example at the command line with:

```
python quality_measures.py -d <path to data>/measures/examples –f
example2_1.csv -g example2_2.csv -m all -r True
```

Example 3:

This example consisted of two cluster labels sets from reading in *news20_assignments_32_run1.csv* and *news20_assignments_32_run3.csv*. The cluster labels from these files were the results of running an actual clustering algorithm based on the HierNMF2. These cluster results contained outliers as mentioned earlier which were removed before constructing the contingency table. The results here would be meaningful and could be compared with the optimal values of the quality measures.

Result of running *quality_measures.py*,

Mutual information:	$MI(U,V) = 2.75529$
Normalized mutual information:	$NMI(U,V) = 0.85688$
Information distance:	$ID(U,V) = 0.48598$
Normalized Information distance:	$NID(U,V) = 0.14994$
Variation information:	$VI(U,V) = 0.92059$
Normalized variation of information:	$NVI(U,V) = 0.25044$

The 20 newsgroups data considered here consisted of $N = 11237$ documents or objects to be clustered, where each object contained 39727 features or terms. The maximum number of clusters was set to 32 for each run of HierNMF2, and results were recorded and saved in *news20_assignments_32_run1.csv* and *news20_assignments_32_run3.csv*, etc. It was noted that although the maximum number of clusters was set to 32, the resulting number of clusters from running HierNMF2might be less than 32.

Let U and V be the cluster labels from *news20_assignments_32_run1.csv* and *news20_assignments_32_run3.csv*, respectively. To further understand the value of mutual information $MI(U,V)$ obtained, it was noted that $\min(H(U), H(V)) = 3.18990$ which was the optimal value for $MI(U,V)$. Therefore, $MI(U,V) = 2.75529$ was not optimal but a fairly close to optimal number for mutual information.

Likewise for $NMI(U,V)$, the value 0.85688 was fairly close to 1.0, which is the optimal value. Recall that the range of $ID(U,V)$ was $[0, \ln(N)]$, where $\ln(N) = 9.32696$. Comparing to 9.32697, $ID(U,V) = 0.48598$ was a lot closer to the optimal value 0 than $\ln(N)$, hence $ID(U,V)$ obtained here was close to optimal as well. The $NID(U,V) = 0.14994$ was close to 0, the optimal value, as well. The $VI(U,V) = 0.92059$ was computed and was closer to the optimal value 0 than $\ln(N) = 9.32697$. The $NVI(U,V) = 0.25044$ was also close to 0, the optimal value.

Run this example at the command line with:

```
python quality_measures.py -d <path to data>/measures/news20 -f
news20_assignments_32_run1.csv -g news20_assignments_32_run3.csv -m
all -r True
```

Special Test Case Examples

There were no cases where the quality measures computed above would be considered questionable. Here we look at a couple of corner cases where it is found that when one or both of the cluster labels sets consisted of a single cluster, i.e., all of the objects or samples were in one cluster, then some of the entropies computed are 0, which could cause the computed measures to be inaccurate or cause numerical issues in the computation. Such cases are trapped so that erroneous results will not be reported; a warning will be reported.

One of the goals in performing clustering for real world data is the ability to group the objects or the samples into more than one cluster, where similar objects were grouped within a cluster, and the samples in different clusters are similar. If all of the samples resulted in a single cluster, the clustering algorithm may have failed to separate the objects and more analysis on the clustering algorithms being used, the problem formulation, and/or the data may be needed to explain such an outcome. While this case is unlikely, care was needed in handling such situations, which is taken care of numerically in the code. Also, rather than reporting a value close to machine zero as zero, we report the actual computed value so that the user can make the judgment call; cases where the values are less than machine precision are also trapped and a warning message is displayed.

Example 1:

The first case considers the case when both clusterings produce one cluster, which means the cluster labels would be 0 in both sets. Let the two clustering labels be

$U = [0,0,0,0,0,0,0,0,0,0,0,0,0,0]$ in file special_example1_1.csv
$V = [0,0,0,0,0,0,0,0,0,0,0,0,0,0]$

Note that all of the objects or samples were grouped into a single cluster for both clusterings U and V. This results in all of the entropies being 0: $H(U) = 0$, $H(V) = 0$, $H(U,V) = 0$ and $H(U|V) = 0$. This is due to the contingency table being only a single number, and the number in this case is equal to $N = 14$, which represents a single cluster from both U and V. Recall that natural logarithms were performed for all of the entropies on the fractions. If both numerator and denominator of the fraction contain the same number (the row sum, column sum and the value in the contingency table all were equal to N), then the computed value is 0 after taking the natural log.

Mutual information:	$MI(U,V) = 0.0$
Normalized mutual information:	$NMI(U,V) =$ undefined
Information distance:	$ID(U,V) = 0.0$
Normalized Information distance:	$NID(U,V) =$ undefined
Variation information:	$VI(U,V) = 0.0$
Normalized variation of information:	$NVI(U,V) =$ undefined

These are the expected results for the inputs. Note that since there is a divide by zero that is trapped, some of the measure results are undefined.

Run this example at the command line with:

```
python quality_measures.py -d <path to data>/measures/examples –f
special_example1_1.csv -g special_example1_2.csv -m all -r True
```

Example 2:
To diagnose special cases further, let

$U = [0,0,0,0,0,0,0,0,0,0,0,0,0,0]$ in file special_example2_1.csv
$V = [0,0,0,0,0,0,0,1,1,1,1,1,1,1]$ in file special_example2_2.csv

for the two sets of cluster labels. Here $N = 14$ like the previous example. Note that all of the samples were grouped into one cluster in U but were grouped into two clusters in V. It was found that the contingency table constructed was an array with two elements $= [7,7]$, row sums (denoted as a_1) became 14 and column sums became 7 and 7 for b_1 and b_2, respectively, which turned out to be like the contingency table here. Unlike the previous example, the entropies $H(U) = 0$ and $H(V) = 0.69315$ which was non-zero because the samples were grouped into more than one cluster in Y. The joint entropy $H(U,V)$ is non-zero as well with value $H(U,V) = 0.69315$, and is equal to $H(V)$. The joint entropy was non-zero since each element in the contingency table was a different value than N, i.e. $\ln\left(\frac{n_{ij}}{N}\right) = 0$ for each n_{ij}. The conditional entropy is $H(U|V) = 0$, which is due to the column sums b_1 and b_2 being identical to the contingency table $= [7,7]$. When taking natural log of the fraction with same numerator and denominator, the value of 0 is obtained for each column sum. If $H(V|U)$ was computed, then $H(V|U)$ = 0.69315 and was non-zero because row sums were equal to 14. This is now the denominator of the fraction, which was different from the numerator of the fraction.

Mutual information:	$MI(U,V) = 0.0$
Normalized mutual information:	$NMI(U,V) = 0.0$
Information distance:	$ID(U,V) = 0.69315$
Normalized Information distance:	$NID(U,V) = 1.0$
Variation information:	$VI(U,V) = 0.69315$
Normalized variation of information:	$NVI(U,V) = 1.0$

As the results in the above table show, mutual information $MI(U,V) = 0$ as expected, whether mutual information was computed as $H(U) - H(U|V)$ or $H(V) - H\ (V|U)$.
For normalized mutual information $NMI(U,V)$, since $\min\big(H(U), H(V)\big) = 0$, $NMI(U,V)$ would then be computed by taking $MI(U,V)$ divided by a number close to machine epsilon. Thus, we compute $NMI(U,V)$ using an alternative method using the sum of the marginal entropies. As expected, $NMI(U,V)$ turned out to be 0.

Information distance $ID(U,V)$ turned out to be 0.69315 as expected since $\max\big(H(U), H(V)\big) = 0.69315$ and $MI(U,V) = 0$. The value of $ID(U,V)$ is just the maximum entropy of the two clusterings. In fact, the value of normalized information distance $NID(U,V)$ clearly shows that $ID(U,V)$ was not close to the optimal value at all. The value of normalized information distance, $NID(U,V) = 1$ as expected since $MI(U,V) = 0$. Therefore, $ID(U,V)$ computed value demonstrates some limitations in interpretation without corroboration from other measures.

The same can be said about $VI(U,V) = 0.69315$, which follows from $H(U,V) = 0.69315$ and $MI(U,V) = 0$. This value of $VI(U,V)$ may be misleading without further investigation of the other measures since it was close to the optimal value 0. But $NVI(U,V)$ shows that $VI(U,V)$ was actually not close to the optimal value. Normalized variation of information, $NVI(U,V) = 1$ since $MI(U,V) = 0$. Therefore, $VI(U,V)$ computed should be investigated further or the clustering methodologies need to be examined.

Run this example at the command line with:

```
python quality_measures.py -d <path to data>/measures/examples –f
special_example2_1.csv -g special_example2_2.csv -m all -r True
```

References

[1] N. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[2] C. K. Reddy, M. A. Hasan, and M. J. Zaki, *Data Clustering: Algorithms and Applications*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2014.

[3] M. Meila, "Comparing clusterings – an information based distance," *Journal of Multivariate Analysis*, 98(5):873-895, 2007.

[4] T. M. Cover and J. A. Thomas, Elements of Information Theory, Wiley Series in Telecommunications, John Wiley & Sons, Inc., 1991.

[5] A. I. Khinchin, Mathematical Foundations of Information Theory, Uspekhi Matematicheshikh Nauk, vol. VII, no. 3, pp. 3-20, 1953. Translated by Dover Publications, 1957.