CPSC 67 Lab #5: Clustering

Due Thursday, March 19 (8:00 a.m.)

The goal of this lab is to use hierarchical clustering to group artists together. Once the artists have been clustered, you will calculate the purity of the clustering using the known genre classification of each artist. In addition, you will use mutual information to determine appropriate labels for the clusters you formed.

You will download no additional data.

This lab requires that you have three Python packages installed: `scipy-cluster` (imported as `hcluster`), `numpy` and `matplotlib`. These are all installed on the cs lab machines. If you plan on working on a machine other than these machines, you should make sure you can get these installed on your machine before you begin.

1 scipy-cluster

In this lab, you will not actually write a hierarchical clusterer. Instead, we will use an existing open-source implementation of a hierarchical clusterer called `scipy-cluster`[1]. In order to use the clustering algorithm in scipy-cluster, we will first need to compute the similarities between each of our documents and store the result in a special format recognized by the clustering algorithm.

1.1 hcluster

To use scipy-cluster, you need to import the `hcluster` module into Python. You will make use of 3 functions in `hcluster`, so you can just import those three functions at the start of your program:

```
from hcluster import squareform, linkage, dendrogram
```

The API documentation can be found off the scipy-cluster website, but hopefully this document will cover the relevant aspects of the API as we need them.

We will also need to make very limited use of `numpy`, which has specialized array formats required by the scipy-cluster package. You will need only one function from the numpy package, so you can import that, too:

```
from numpy import zeros
```

The API documentation for numpy is unnecessary for this assignment; you only need to know how to use this one function which will be explained shortly.

1.2 Pairwise similarity

As with the k-Nearest Neighbors implementation from Lab #4, you will build a VectorCollection where each vector represents a concatenation of all of the web pages associated with a particular artist. Once you have this VectorCollection, you will convert it to the appropriate SMART notation (e.g. 'ntc'). Using this SMART-VectorCollection, you will compare each artist vector against every other artist vector using cosine similarity. Since we are doing unsupervised clustering, there is no need to do cross-validation (and so you do not need to hold one vector out when comparing to the rest of the VectorCollection).

You will compare every artist against every other artist and store the similarities in a two-dimensional square matrix. Since you are comparing every artist to every other artist, the number of rows and columns in this square matrix is equal to the number of artists.

In order to use the scipy-cluster clustering algorithm, you will need to first store this square matrix in a `numpy.ndarray` which is simply a numpy's representation of an n-dimensional list. If the number of artists is called `numArtists`, then to create a `numArtists` by `numArtists` square matrix of type `numpy.ndarray`, you simply say:

```
matrix = zeros( (numArtists, numArtists) )
```

[1] `http://code.google.com/p/scipy-cluster/`

This creates an empty square matrix filled with (floating point) zeros. Importantly, `zeros` takes a single parameter which is a tuple of the dimensions of the matrix.

Once the `ndarray` is created, you can treat it like any two-dimensional array in Python, accessing and setting values in the array exactly as you would a two-dimensional list. For example,

```
matrix[2][0] = 0.86
```

NOTE: Due to a quirk with the clustering package that we are using, the values that you store in this square matrix are not the raw cosine similarity values. Instead, if the cosine similarity is c, you will store $1 - c$ in the array. This means that for very similar documents you will store a number close to 0, and for very dissimilar documents you will store a number close to 1.

You will want to fill the square matrix such that at in `matrix[i][j]` you are storing 1-cos(i,j), the similarity between artist `i` and artist `j`.

CAUTION: The artist ids in your database are not 0-indexed ('Air' has artist id = 1) but for many parts of this assignment you will be working with 0-indexed arrays. Be super-careful about translating back and forth between artist ids and matrix indices. This can be a really bad source of errors if you aren't careful.

Taking the above caution, this means that `matrix[0][1]` stores the similarity between artist id 1 and artist id 2. Also, note that since cos(i,j) = cos(j,i), `matrix[1][0]` = `matrix[0][1]`. This can save you some time in doing your computations since you don't need to compute both cos(i,j) and cos(j,i).

Once you have computed all of the pairwise similarities, you need to convert this `ndarray` into a special form which will be required by the next step:

```
sqfrm = squareform(matrix)
```

The `squareform` function, part of the `hcluster` module, compresses the 2-dimensional square matrix into a 1-dimensional vector storing only the upper diagonal of the matrix. (This is adequate since the matrix is symmetric along the diagonal.) You will not need to modify the squareform of the matrix, you will just need to use it as a parameter to the clustering function.

1.3 linkage

To cluster the documents, you need to simply say:

```
clustering = linkage(sqfrm, method=clusterMethod)
```

where we will vary `clusterMethod` to be one of `'single'`, `'complete'`, and `'average'`, corresponding to the single-link, complete-link, and average-link cluster similarity methods.

1.4 dendrogram

Congratulations, your documents are now clustered! If you'd like to view your clustering, do the following:

```
import matplotlib

dendrogram(clustering)
matplotlib.pylab.show()
```

This result is difficult to read because the clusters have not been given intuitive labels. If you create a (standard Python) list containing the appropriate number of labels (in this case, the number of artists in the VectorCollection) – in artist_id order – you can pass this as an optional parameter to dendrogram. Here, `artistLabels` is a list of labels:

```
dendrogram(clustering, labels=artistLabels)
matplotlib.pylab.show()
```

Notice that you don't actually need to save the result of the dendrogram function. As a result of running the dendrogram function, matplotlib is able to show the correct plot without any parameters. Also notice that your program hangs at the line where you view the dendrogram. To continue your program, you'll need to close the dendrogram viewer. (For this reason, you'll may want to keep these lines commented out while you're debugging).

1.5 k = 4

Notice that the clustering that was performed clustered every artist into a single cluster. If you view the `clustering` result (shown here for 'ntc', 'complete' link, on the common set) you'll see something that begins like this:

```
>>> clustering[:10]
array([[ 16.        ,  65.        ,   0.21161505,   2.        ],
       [ 77.        ,  80.        ,   0.48680791,   3.        ],
       [ 51.        ,  60.        ,   0.76204462,   2.        ],
       [ 12.        ,  45.        ,   0.81809799,   2.        ],
       [ 76.        ,  79.        ,   0.85988849,   2.        ],
       [ 40.        ,  46.        ,   0.8744124 ,   2.        ],
       [ 53.        ,  54.        ,   0.88564899,   2.        ],
       [ 58.        ,  82.        ,   0.88895148,   3.        ],
       [ 20.        ,  38.        ,   0.88986817,   2.        ],
       [ 71.        ,  78.        ,   0.89477689,   2.        ]])
```

Let's read this row by row. The first row says that cluster 16 and cluster 65 were merged together because their similarity was 0.21161505 and that the new cluster they formed has 2 documents in it. Cluster 16 and cluster 65 were clusters that contained only a single artist document. In fact, since you have 80 artists, you started with clusters numbered 0 through 79, each with a single artist in it. When clusters 16 and 65 are merged, they form cluster 80 (one higher than the highest cluster created so far) and clusters 16 and 65 are not considered for further clustering – only cluster 80 can be merged from now on.

In the second row, cluster 77 is merged with our newly created cluster 80 and (skipping the score) there are now 3 artists in this cluster. This cluster is cluster 81, and clusters 77 and 80 can no longer be clustered.

Notice that before we started the clustering process, we had 80 singleton clusters. After one row of the clustering shown above, we had 79 clusters. After two rows, we have 78 clusters. So, if we want to know what the clusters look like when there are only 4 clusters, we simply need to step through this list one line at a time until we get to the 4th-to-last line in the array.

Eventually, you'd like to create something like this:

```
{136: set([19, 71, 79, 76, 78, 63]), 148: set([64, 51, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62]),
154: set([0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24,
26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46, 47, 48, 52,
65, 67, 72, 77]), 155: set([1, 66, 68, 69, 70, 73, 74, 75, 49, 50, 25, 42])}
```

This says that cluster 136 contains the artists 19, 71, 79, 76, 78, and 63. (These are 0-indexed!) The four clusters are numbered 136, 148, 154, and 155.

This is slightly tricky to do, but hopefully the end result shown above has given you a bit of a hint at how to go about computing this.

2 Cluster Purity

In the previous section, you clustered your data. Now, you'd like to determine how pure each of the clusters are. To do this, we first need to determine the majority genre label in each cluster, then we count the total number of documents which would be correctly labeled by this majority label. For example, if there are 15 documents in a particular cluster, 12 Hip Hop, 2 Alternative and 1 Country Pop, the total number of correctly labeled documents using the majority label (Hip Hop) is 12. Repeat this for all clusters, summing the number of correctly labeled documents using the majority label in each cluster. When you are done, divide by the total number of data points (here, that would be 80). See pages 328-329 in the book. Figure 16.4 on page 329 should be particularly helpful.

3 Mutual Information

In addition to computing cluster purity for each cluster, you'd like to automatically label each cluster using the most informative words in the cluster. You will use mutual information (page 252 in the text book) to come up with these cluster labels.

Computing mutual information requires you to plug in numbers directly into the formula on page 252, but it assumes you can fill in the two-by-two box shown on the bottom of page 252. As further explanation, let's assume that we're working with a particular cluster 155 and the word 'guitars'. The box labeled N_{11} stores the number of documents (here, artists) in cluster 155 that contain the word 'guitars'. Let's say that this number is 4. Next, let's look at box N_{01}. This contains the number of documents in cluster 155 that do not contain the word 'guitars'. If there 12 documents in cluster 155, N_{01} will be $12 - 4 = 8$. Next, box N_{10} contains the number of documents that are *not* in cluster 155 that contain the word 'guitars'. Let's say this is 29. The final box, N_{00} contains the number of documents that are *not* in cluster 155 that do not contain the word 'guitars'. Since we know that there are 80 total artists, and we know that 12 of those artists are in cluster 155, that leaves 68 artists that are not in cluster 155. Of those 68, we said that 29 contain the word 'guitars'. This leaves $80 - 12 - 29 = 39$ documents that are not in cluster 155 and that don't contain 'guitars'.

	in cluster 155	not in cluster 155
contains 'guitars'	$N_{11} = 4$	$N_{10} = 29$
does not contain 'guitars'	$N_{01} = 8$	$N_{00} = 39$

These numbers are the actual results for cluster 155 in the common set, using 'ntc' and 'complete' link, assuming you follow the instructions and numbering guide explained in Section 1.5.

The final values in formula that you need are as follows:

$$\begin{aligned} N_{0.} &= N_{00} + N_{01} \\ N_{1.} &= N_{10} + N_{11} \\ N_{.0} &= N_{00} + N_{10} \\ N_{.1} &= N_{01} + N_{11} \\ N &= N_{11} + N_{10} + N_{01} + N_{00} \end{aligned}$$

Caution: Whether you are reading the formula online or in the hard copy, be sure you distinguish between the terms $N_{1.}$ and $N_{.1}$. It is difficult to tell these apart, but being able to do so is crucial to computing the correct mutual information result.

The complete formula, reproduced from the text, is as follows:

$$I(U;C) = \frac{N_{11}}{N}\log_2\frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N}\log_2\frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N}\log_2\frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N}\log_2\frac{NN_{00}}{N_{0.}N_{.0}}$$

To ensure you don't get division by zero errors (which can very easily happen if you aren't careful), notice that if the term preceding the log is 0, there is no need to compute the term inside the log - simply set that whole piece to 0. (The denominator can never be 0 if the term preceding the log is non-zero.)

For each word `w` in the corpus, and for each cluster `c`, you will want to compute the mutual information showing how much the presence (or absence) of `w` contributes to the cluster `c`. For a given cluster, the words with the highest mutual information are those whose presence *or absence* is most useful in predicting this particular cluster. We would like to exclude as cluster labels those words whose absence is useful since they won't be intuitive as cluster labels. To do this, we will select the 5 words with highest mutual information that also occur in at least half of the artist documents in the cluster.

4 Results

On the wiki, you will report the following (on your data set):

1. The purity of your clusters using 'ntc' and each of 'single', 'complete', and 'average' link clustering.
2. The 5 most relevant cluster labels for each cluster using 'ntc' and 'complete' link. In addition to these cluster labels, report the majority genre label for the cluster.

In addition, we would like you to experiment varying 'nnc', 'ntc', and 'ltc', as well as varying 'single', 'complete', and 'average'. In a separate document, show the purity and the cluster labels for some of the most successful combinations of variants. Which combinations performed the most poorly? Which performed best? Include the purity, cluster labels and dendrogram for the single combination which has the highest purity. (You can save an image directly from the matplotlib viewer by clicking the little disk icon in the top bar of the dendrogram viewer.)

For your demo, you should be able to:

- run your clustering on an arbitrary SMART label ('nnc','ntc','ltc') and clustering method ('single','complete','average) on your data and the common set data,
- show the dendrogram produced by the clustering,
- show the purity of each of the clusters, and
- show the 10 most relevant cluster labels for each of the clusters.