Exercise on hierarchical clustering

The plant *Podophyllum hexandrum* (colloquially known as 'mayapple') produces the compound podophyllotoxin. This molecule is the natural product precursor to the topoisomerase inhibitor etoposide, which is used in dozens of chemotherapy regimens worldwide for a variety of malignancies. Although etoposide is on the World Health Organization's list of essential medicines, production requires isolation of (–)-podophyllotoxin from the plant, which is a tedious process. Knowing the biosynthetic route that the plant uses to produce the molecule would allow expressing the corresponding genes in engineered hosts for much more efficient industrial production, and would drastically decrease the costs to produce this drug.
Previous studies already uncovered four genes that play a role in the early steps of podophyllotoxin biosynthesis (DIR, PLR, SDH and CYP719A23). Now, researchers performed RNA-seq analysis on the plant material to identify other candidate genes for this pathway by investigating which genes follow the same expression pattern. You are given a subset of 308 assembled genes, along with their normalized relative gene expression values, that has already been filtered from the total RNA-seq dataset based on the fact that their predicted protein products belong to any of a set of enzyme superfamilies that are likely to be involved in this kind of pathway. The data contain three time points and three replicates per timepoint.

Note: For this exercise, it is important that you work on the server, where all required libraries have been installed.

In your script, you will use a number of specialized Python modules:

- NumPy / SciPy / Pandas / MatPlotLib: http://www.scipy.org/
- Seaborn: http://seaborn.pydata.org/index.html

To import these, add the following import statements:

```
import numpy
import scipy
import scipy.cluster.hierarchy as sch
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns
```

1. To start working with your data, it is best to format them as a Pandas DataFrame: a flexible two-dimensional matrix that includes annotations of both the rows and the columns. Copy the contents of the Excel file into a tab-delimited TXT file and write a Python function to parse this into a Pandas DataFrame containing the right data types in the right places (strings / floats / etc.). To do this, you will first need to parse the data into a dictionary (with sample names as keys, and a list of gene expression value floats as value for each key) and a list indexing the labels (gene names) to use as input. For instructions on how to then turn this dictionary and corresponding index list into a DataFrame, see http://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists

2. Now you can first perform a hierarchical clustering of the twelve samples to check if the replicates indeed cluster together nicely.

To do so, let's transpose the data to allow clustering the data by sample (using `data_frame = data_frame.transpose()`).
Start by computing the pairwise distances, using the Euclidean distance metric:
`distances = sch.distance.pdist(data_frame, metric='euclidean')`
Look at the number of values in the resulting variable by using '`print(distances.shape)`'. Can you explain this number?
Next, perform a complete-linkage clustering and generate a dendogram:

```
clustering = sch.linkage(distances, method='complete')
tree = sch.dendrogram(clustering)
plt.show()
```

Do the samples cluster together by time point as expected? If not, what does this suggest about the variance within/between (some of) the time points?

3. Let's again transpose the data to start clustering the data by gene (using `data_frame = data_frame.transpose()`). We will now take a deeper look at each of the steps involved in the clustering process.
   Use a variant of the following code to perform the clustering and generate a dendrogram:

   ```
   distances = sch.distance.pdist(data_frame, metric='correlation')
   clustering = sch.linkage(distances,method='complete')
   tree = sch.dendrogram(clustering , leaf_font_size=2,
   color_threshold=4, labels = gene_names)
   ```

   Here, the `gene_names` variable should point to a list containing your gene names in the order in which they appear in the DataFrame index, and the `color_threshold` can be varied based on your preferences. Set up your code in such a way that you can specify the distance metric and clustering method (linkage type) from the command line.
   To display the result on screen or to save it to a file, you can use `plt.show()` or, e.g., `plt.savefig('plot_dendrogram.pdf', format="PDF")`, respectively. (You cannot do both at once, because the plot will be cleared after either displaying or writing. If this does not happen, and you need to clear your canvas, you can use `plt.gcf().clear()`)

4. The disadvantage of a dendrogram visualization is that it is difficult to assess the correctness / meaningfulness of the clustering. To visually compare the clustered dendrogram with the expression patterns, one option is to generate a clustered heatmap. For this, you will use the Seaborn package.
   Have a look at the instructions here:
   http://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.clustermap.html
   Based on these instructions, write a function that generates a clustered heatmap (with the metric and method parameters supplied from the command line as before) in PDF format. To retain the time dimension on the horizontal axis, do not perform a clustering on the samples. Also, make sure to use appropriate figsize values (e.g., '(12,100)') so that you can easily read the output graph.

5. Have a look at the paper describing the study from which the results originate (http://www.sciencemag.org/content/349/6253/1224.full, see Figure 2 and the surrounding text), and identify the genes that were in the end discovered to be

part of the podophyllotoxin biosynthesis pathway. Run your script using a few different distance metrics for the pairwise distance calculation, including at least 'correlation' and 'euclidean'; for all of the results, check how well the newly discovered podophyllotoxin biosynthesis genes cluster with the previously known ones (DIR, CYP719A23). What kind of differences do you notice? What could be the explanation?

6. ***Optional:*** A powerful technique to evaluate clustering of multi-dimensional data is principal component analysis (PCA). PCA is a mathematical algorithm that reduces the dimensionality of the data while retaining most of the variation in the data set. It accomplishes this reduction by identifying directions, called principal components, along which the variation in the data is maximal. By using a few of these components, each sample can be represented by relatively few numbers instead of by values for thousands of variables. Samples can then be plotted along the components that explain most of the variation, making it possible to visually assess similarities and differences between samples and determine whether samples can be grouped.

   More information: http://www.nature.com/nbt/journal/v26/n3/full/nbt0308-303.html

   Write a function for your script that performs a PCA analysis on the data using scikit-learn: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
   Example code:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Standardization
X_std = StandardScaler().fit_transform(matrix)
# PCA
sklearn_pca = PCA(n_components=2)
X_transf = sklearn_pca.fit_transform(X_std)
# Plot the data
plt.scatter(X_transf[:,0], X_transf[:,1], colors)
plt.title('PCA via scikit-learn')
plt.savefig('plot_PCA.pdf', format="PDF")
```

   You will want to color each point in the plot by assigned cluster using the `colors` parameter. To construct the colors variable (which should be a list of colors, one per gene), you will need to first retrieve cluster assignments from your clustering at a given threshold. You can retrieve these cluster assignments using the `fcluster()` method (see http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.cluster.hierarchy.fcluster.html).
   To color the genes by the clusters returned by `fcluster()`, you then need to convert the numbers of the clusters into colors from a matplotlib color map (http://matplotlib.org/api/cm_api.html), such as cm.jet (see http://matplotlib.org/examples/color/colormaps_reference.html for a visualization).

   Example code:

```
from matplotlib import cm #import statement needed for color maps
assignments = sch.fcluster(clustering, t=4, criterion='distance')
division = int(256 / len(list(set(assignments))))
```

```
colors = [cm.jet(idx*division) for idx in assignments]
```

Once implemented, evaluate how well some of your clusterings (with different metrics and methods) map on the scatter plot.

In this exercise, we have restricted ourselves to hierarchical clustering. To have a look at a Python implementation of k-means clustering, see http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html