Hierarchical Clustering

October 15, 2019

0.1 Hierarchical Clustering

```
In [1]: #=============================================================
        # Read me
        #=============================================================
        # clustering.py
        # Author: Yesika Contreras
        #
        # This code cluster the genesymbols for 4 files based on the Log Fold Change

        # Code adapted from:
        # https://cnls.lanl.gov/external/qbio2018/Slides/Cluster_Lab_June18/clustering_lab.pdf
        # https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tu
        # https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/
        #
        # python scripts generated 10-14-2019

        import datetime
        d = datetime. datetime. today()
        print(d.strftime('%m-%d-%Y'))
```

0.2 Importing packages

```
In [2]: #=============================================================
        # Packages
        #=============================================================

        import pandas as pd
        import numpy as np #operations
        import os # Accesing to directory
        import re # Regular Expressions
        from six.moves import reduce # Merge dataframes
        from scipy.cluster.hierarchy import dendrogram, linkage #for dendogram
        from scipy.cluster.hierarchy import fcluster #cluster labels
        from matplotlib import pyplot as plt # for visualizations

        ## Setting the seed value for reproducibility
        seed_value= 123# Set a seed value

        # Set `python` built-in pseudo-random generator at a fixed value
        import random
        random.seed(seed_value)

        # Set `numpy` pseudo-random generator at a fixed value
        import numpy as np
        np.random.seed(seed_value)

        seed =  np.random.RandomState(123)
        # do not call numpy.random.rand() but seed.rand()


        # 3. Set environment
        os.urandom(seed_value)
```

```
Out[2]: b'/\x18\xd4\x8cr\x8b\xf9 3\x05rr\x1b\xd5\xba\xf1L\xfb\\M\xdc\xad\xe0\xd2\xa7~{F\xec\xd
```

0.3 Defining Functions

- Defining function to be used in this script

```
In [3]: #============================================================
        # Reading Files
        #============================================================

        def read_files(df_file, skip_rows =0):
            '''
            Function that read the file as dataframe
            Extract the geneSymbol column and the Log Fold Change
            Input: User need to provide the path for file
            Output =  dataframes
            '''
            df_file = pd.read_csv(df_file, delimiter="\t", skiprows= skip_rows,)


            # Removing columns that are not GeneSymbol or logFC
            list_dropping =[]
            for column in df_file.columns:
                if not re.search('^GeneSymbol|^logFC', column):
                    #print (column)
                    list_dropping.append(column)

            df_file.drop(columns =  list_dropping,  inplace=True)
            # Maintain only one GeneSymbol column
            list_dropping =[]
            for column in df_file.columns:
                if re.search('^GeneSymbol', column):
                    #print (column)
                    list_dropping.append(column)

            df_file.drop(columns =  list_dropping[1:],  inplace=True)

            old_name = df_file.filter(regex='^GeneSymbol',axis=1).columns[0]
            df_file.rename(columns = { old_name: 'GeneSymbol'},
                           inplace =True)

            return df_file
```

0.3.1 Hierarchical clustering

- Agglomerative or bottom-up approach.
- Data points are clustered using a bottom-up approach starting with individual data points as its own cluster
- and then combine clusters based on some similarity measure.

Hierarchical Clustering algorithm

1. The number of clusters at the start will be K, while K is an integer representing the number of data points. (At the start, each data point as one cluster).
2. Form a cluster by joining the two closest data points resulting in K-1 clusters. (The algorithm starts by finding the two points that are closest to each other on the basis of Euclidean distance for instance).
3. Form more clusters by joining the two closest clusters resulting in K-2 clusters. (The next step is to join the cluster formed by joining two points to the next nearest cluster or point which in turn results in another cluster).
4. Repeat the above three steps until one big cluster is formed.

Dendogram -cluster size - Once one big cluster is formed, we can leverage the results from the dendrogram to approximate the number of clusters. - We cut the dendrogram tree with a horizontal line at a height where the line can traverse the maximum distance up and down without intersecting the merging point. The number of vertical lines this newly created horizontal line passes is equal to number of clusters. - Basically the horizontal line is a threshold, which defines the minimum distance required to be a separate cluster. If we draw a line further down, the threshold required to be a new cluster will be decreased and more clusters will be formed.

Cluster size . Altrnative process You can also consider plots like Silhouette plot, elbow plot, or some numerical measures such as Dunn's index, Hubert's gamma, etc.. which shows the variation of error with the number of clusters (k), and you choose the value of k where the error is smallest.

```
In [4]: #===============================================================
        # Hierarchical clustering of cancer gene expression data
        #===============================================================
        # Basic question: What genes are express in a specific population.
        # We're going to perform hierarchical clustering on both
        # the genes (rows) and samples (columns)

        ## Process data for clustering

        def preprocess_clustering(df_file, column = 'GeneSymbol' ):

            # Changing the index from the default numerical values
            # to the GeneSymbol column (first column)
            df_file =  df_file.set_index(column)

            #  Getting a list of the population columns names (Gene Symbol)
            leaf_labels = list(df_file.index)
            return df_file, leaf_labels


        def dendrogrammer(df_file):

            '''
            # Function that displays a dendrogram
            # input: df = dataframe with the gene names as index.
            # leaf_lables: labels or column names of the dataframe
            # that are going to be the leaves for the dendogram.
            # output: array of dendogram values
            '''
            # preprocessing dataset
            df_file, leaf_labels =  preprocess_clustering(df_file,
                                                          column = 'GeneSymbol' )

            ## Getting the numerical data from the dataframe in a numpy array.
            data_array = df_file.values


            ##  Perform hierarchical clustering
            # Scipys clustering algorithm clusters the rows.
            # Linkage methods could be single, average, complete, median,
            # centroid, weighted, or ward
            # Distance metrics (e.g., 'euclidean', cityblockk, yule, hamming,
            # dice, kulsinski, correlation)
            # The metric is set to "euclidean" (distance between the datapoints).
            # Finally linkage parameter is set to "ward", which minimizes the variant
            # between the clusters.
            dendogram_values = linkage(data_array , method='ward', metric='euclidean')

            ## Distance of the last 4 merges
            print('Distance of the last 4 merges: ')
            dendogram_values[-4:,2]
            return dendogram_values, leaf_labels


        def dendrogram_plot(dendogram_values, leaf_labels, truncate_mode = None, p =0,
                            contracted=False):
            '''
            Plot dendrogram
            On the x axis you see labels. If you don't specify anything else they
            are the indices of your samples in X.
            On the y axis you see the distances (of the 'ward' method in our case).
            horizontal lines are cluster merges
            vertical lines tell you which clusters/labels were part of merge forming
            that new cluster
            heights of the horizontal lines tell you about the distance that needed
            to be "bridged" to form the new cluster
            color_threshold argument of dendrogram(), which as not specified
            automagically picked a distance cut-off value of 70 % of the final merge
            and then colored the first clusters below that in individual colors.
            '''
            plt.figure(figsize=(10, 6))
            ax = plt.subplot()
            plt.subplots_adjust(left=0.07, bottom=0.3, right=0.98, top=0.95,
            wspace=0, hspace=0)
            plt.title('Hierarchical Clustering Dendrogram')
            plt.xlabel('GeneSymbol or (cluster size)')
            plt.ylabel('Distance')

            dendrogram(dendogram_values,
                       leaf_rotation=90., # rotates the x axis labels
                       leaf_font_size=10.,# font size for the x axis labels
                       truncate_mode='lastp',  # show only the last p merged clusters
                       p=12,  # show only the last p merged clusters
                       show_contracted= contracted,  # to get a distribution impression
                       labels=leaf_labels)

            plt.show()
            plt.savefig('dendrogram.png')


        # Selecting a Distance Cut-Off aka Determining the Number of Clusters

        def treshold_dendrogram(*args, **kwargs):
            '''
            Determining the number of cluster visually
            Retrieving the distance between clusters
            and the horizontal cut-off line
            '''
            max_d = kwargs.pop('max_d', None)
            if max_d and 'color_threshold' not in kwargs:
                kwargs['color_threshold'] = max_d
            annotate_above = kwargs.pop('annotate_above', 0)

            ddata = dendrogram(*args, **kwargs)

            if not kwargs.get('no_plot', False):
                plt.title('Hierarchical Clustering Dendrogram (truncated)')
                plt.xlabel('sample index or (cluster size)')
                plt.ylabel('distance')
                for i, d, c in zip(ddata['icoord'], ddata['dcoord'], ddata['color_list']):
                    x = 0.5 * sum(i[1:3])
                    y = d[1]
                    if y > annotate_above:
                        plt.plot(x, y, 'o', c=c)
                        plt.annotate("%.3g" % y, (x, y), xytext=(0, -5),
                                     textcoords='offset points',
                                     va='top', ha='center')
                if max_d:
                    plt.axhline(y=max_d, c='k')
            return ddata


        # Selecting Number of clusters with the Elbow method:

        def elbow_method (K):

            '''
            input: K is the dendogram values
            Variant of the "elbow method". It tries to find the clustering step
            where the acceleration of distance growth is the biggest
            (the "strongest elbow" of the blue line graph below,
             which is the highest value of the orange graph below):
            '''
            last = K[-10:, 2]
            last_rev = last[::-1]
            idxs = np.arange(1, len(last) + 1)
            plt.plot(idxs, last_rev)

            acceleration = np.diff(last, 2)  # 2nd derivative of the distances
            acceleration_rev = acceleration[::-1]
            plt.plot(idxs[:-2] + 1, acceleration_rev)
            plt.show()
            k = acceleration_rev.argmax() + 2  #if idx 0 is the max of this we want 2 clusters
            print ("clusters:", k)

            return k
```

```
def retrive_cluster(Z, dataset, k=None, max_d= None):
    '''
    Retrieve the clusters labels per index as a series object and
    it is added to the dataset.
    input: dendogram values and k:number of clusters or max_d:max distance
    '''
    if max_d == None:
        clusters = fcluster(Z, k, criterion='maxclust')
    else:
        clusters = fcluster(Z,k, criterion='distance')

    dataset['cluster'] = clusters

    return dataset
```

0.3.2 Returning output dataset

```
In [5]: def output_file(output_file, output_path, output_file_name):
            '''
            Retrive a single file,
            inputs output_file: dataframe,
            output_path:folder location,
            output_file_name: file name with .txt extension
            '''
            output_file.to_csv(os.path.join(output_path, output_file_name), sep='\t')
```

0.4 Runing Main Function

- User input information manually
- Computation and outputs generated

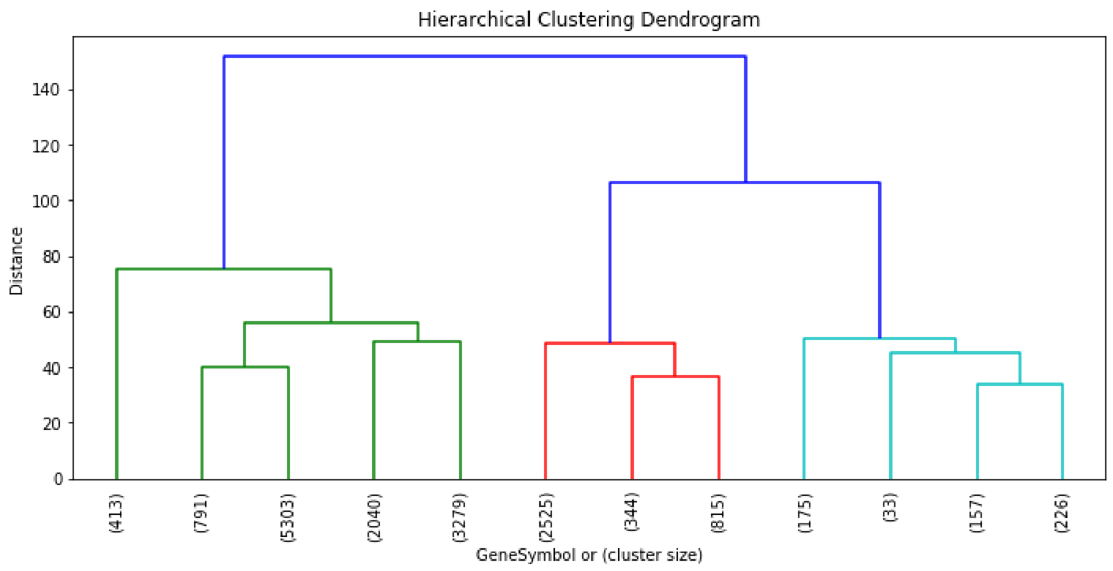
```
In [6]: #==============================================================
        # User input:
        #==============================================================

        #file_path = input("Enter the file location of the initial dataset files: \n: ")
        file_path = '/output_files/df_merged_inner.txt'

        #output_path = input('\nEnter the location where you want to store the output file:\n
        output_path = '/output_files'
```

```
In [7]: #==============================================================
        # Computation Importing datasets
        #==============================================================

        # Loading dataset
        logFC_df= read_files(file_path)
        logFC_df.head(3)

        #print('Number of observations in the dataset: '+ logFC_df.size)
```

```
Out[7]:       GeneSymbol  logFC_GSM2386506_Kupper.txt  \
        0  0610007P14Rik                    -0.089542
        1  0610009B22Rik                     0.082806
        2  0610009L18Rik                     0.155382

           logFC_GSE47045_GEO2R_TRM_v_TN_Carbone.txt  \
        0                                  -1.062394
        1                                   0.423292
        2                                  -0.023245

           logFC_GSE79858_Slansky_TIL_v_TN.txt  logFC_GSE_Anderson_TIL_v_Naive.txt
        0                                0.973                            0.833661
        1                                1.430                            0.217723
        2                               -0.363                            0.254251
```

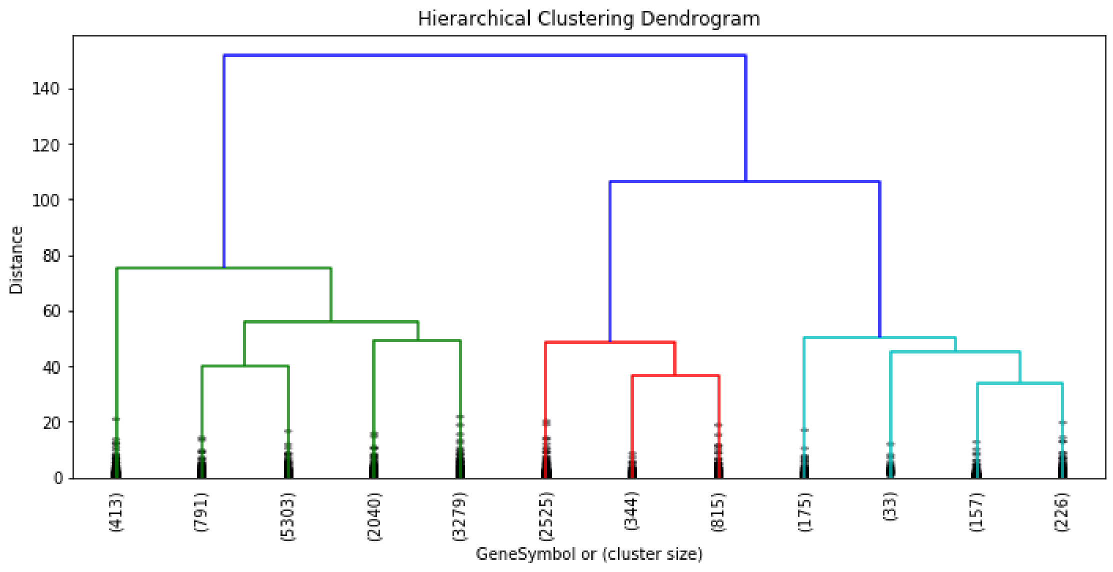
```
In [8]: #============================================================
        # Computation Hierarchical clustering
        #============================================================

        # Getting defualt dendogram to do an initial analysis
        dendogram_values_logFC, leaf_labels_logFC = dendrogrammer(logFC_df)

        dendrogram_plot(dendogram_values_logFC, leaf_labels_logFC)
```

```
Distance of the last 4 merges:
```



```
<matplotlib.figure.Figure at 0x1179c9f28>
```

```
In [9]: # Getting a Hierarchical Clustering Dendrogram (truncated)
        # The below shows a truncated dendrogram,
        # which only shows the last p=12 out of our 80504(sample size -1) merges.
        # Showing also the cluster size per branch.

        dendrogram_plot(dendogram_values_logFC, leaf_labels_logFC, truncate_mode='lastp',
                        p =12, contracted= True)
```

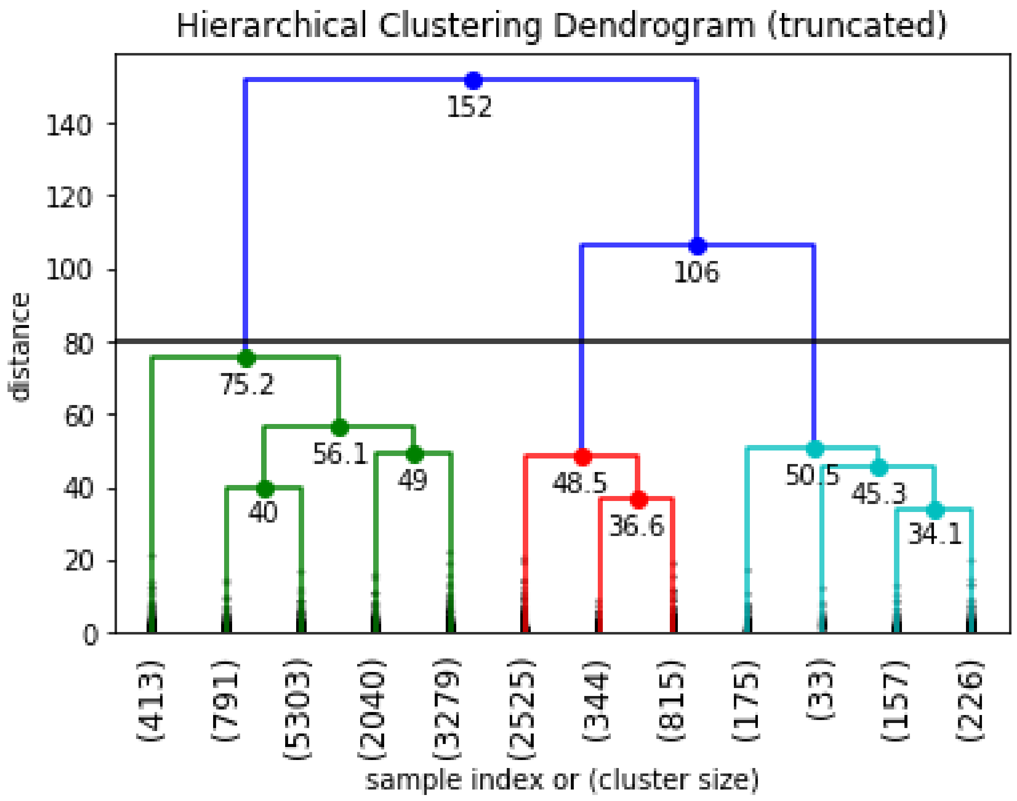


```
<matplotlib.figure.Figure at 0x10ee066d8>
```

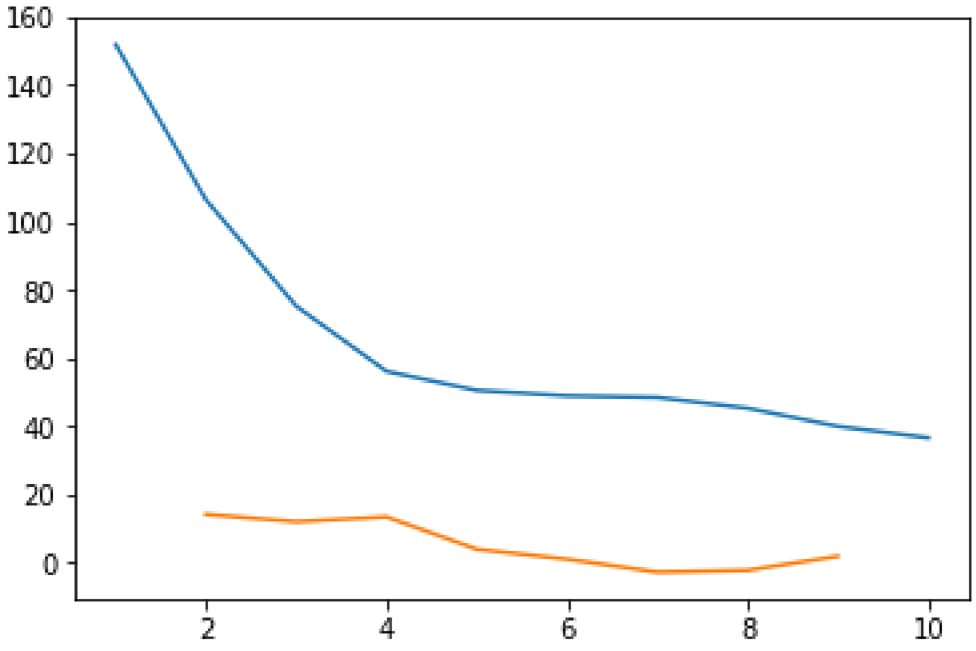
```
In [10]: # Checking the distances at the last 6 merges.
         # There is a bigger jumpy from 75.2 (Distances >20 up)
         dendogram_values_logFC[-6:,2]

Out[10]: array([ 48.96232568,  50.54659892,  56.14556389,  75.24547989,
                106.4452964 , 151.88217343])
```

```
In [11]: #==============================================================
         # Determining the Number of Clusters
         #==============================================================

         # Selecting a Distance Cut-Off
         # In general for a chosen cut-off value max_d we can always simply count
         # the number of intersections with vertical lines of the dendrogram to get
         # the number of formed clusters.
         # set cut-off to 80
         # biggest distance among last p4 is after 75.2

         max_d = 80  # max_d as in max_distance
         print('We obtain 3 final clusters: ')

         treshold_dendrogram(
             dendogram_values_logFC,
             truncate_mode='lastp',
             p=12,
             leaf_rotation=90.,
             leaf_font_size=12.,
             show_contracted=True,
             annotate_above=10,  # useful in small plots so annotations don't overlap
             max_d=max_d,  # plot a horizontal cut-off line
         )
         plt.show()
```

```
We obtain 3 final clusters:
```



```
In [12]: ## Elbow Method

         k = elbow_method (dendogram_values_logFC)
```



```
clusters: 2
```

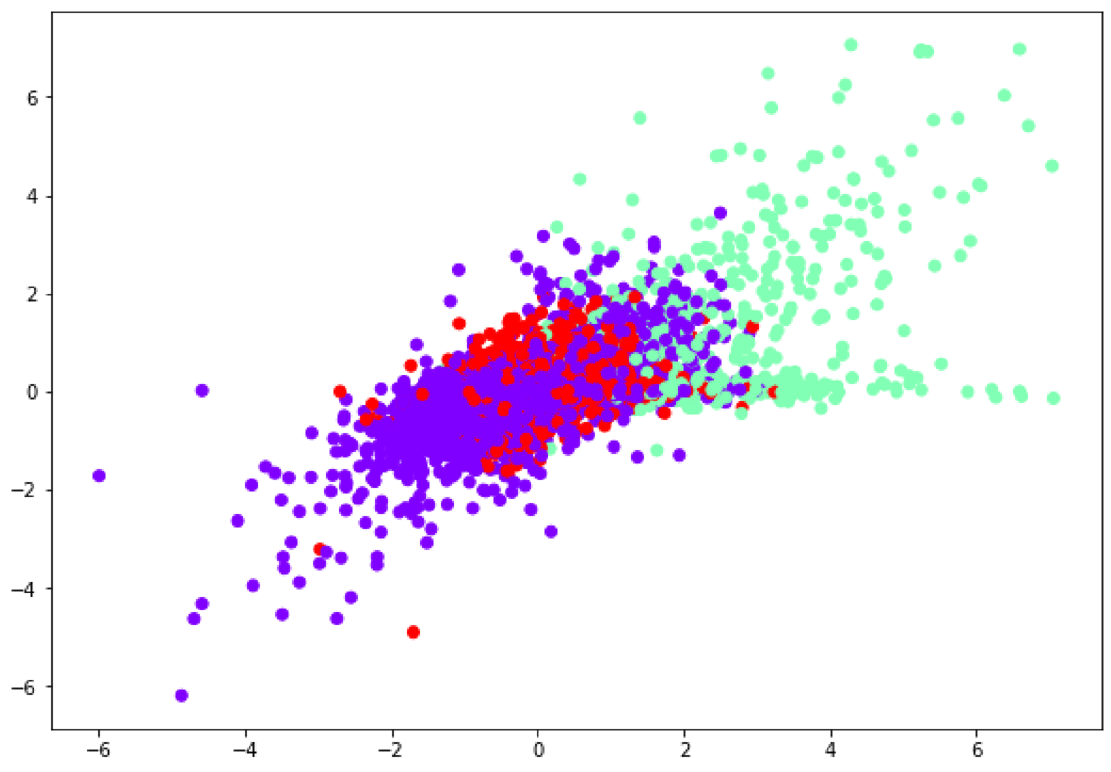
```
In [13]: # Adding a new column to dataset with the clusters labels.

         # provide k or max_d. If you know the max distance or the number of clusters.
         k = 3 #number of clusters
         max_d = 80 # max distance

         retrive_cluster(dendogram_values_logFC, dataset =logFC_df, k=3, max_d= None )
         logFC_df.head()

Out[13]:       GeneSymbol  logFC_GSM2386506_Kupper.txt  \
         0  0610007P14Rik                    -0.089542
         1  0610009B22Rik                     0.082806
         2  0610009L18Rik                     0.155382
         3  0610009O20Rik                    -0.258520
         4  0610010B08Rik                    -0.374875

            logFC_GSE47045_GEO2R_TRM_v_TN_Carbone.txt  \
         0                                  -1.062394
         1                                   0.423292
```

```
2                                     -0.023245
3                                     -0.355711
4                                      0.367777

   logFC_GSE79858_Slansky_TIL_v_TN.txt  logFC_GSE_Anderson_TIL_v_Naive.txt  \
0                             0.973000                            0.833661
1                             1.430000                            0.217723
2                            -0.363000                            0.254251
3                             0.463000                           -1.758133
4                             0.944867                            0.664039

   cluster
0        2
1        2
2        1
3        1
4        2
```

In [14]:
```
### Hierarchical Clustering with Scikit-Learn

# Call the fit_predict method to predict the name of the clusters
# that each data point belongs to.

from sklearn.cluster import AgglomerativeClustering
# The number of parameters is set to 2 using the n_clusters parameter
# The affinity is set to "euclidean" (distance between the datapoints).
# Finally linkage parameter is set to "ward", which minimizes
# the variant between the clusters.

# data, labels = preprocess_clustering(logFC_df, column = 'GeneSymbol' )
data = logFC_df.iloc[:, 1:].values # from column 1.
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean',
                                  linkage='ward')
cluster.fit_predict(data)
print(cluster.labels_)

# Finally, let's plot our clusters.
plt.figure(figsize=(10, 7))
plt.scatter(data[:,0],data[:,1], c=cluster.labels_, cmap='rainbow')

# # Adding a new column to dataset with th clusters labels.
# logFC_df['Clusters'] = cluster.labels_
```

```
[2 2 0 ... 0 0 2]
```

Out[14]:
```
<matplotlib.collections.PathCollection at 0x116a85080>
```



```
In [15]: # Saving the cluster dataframe

         file_output= 'cluster.txt'
         output_file(logFC_df, output_path, file_output)
```