splitA_DBGWASop_10May

October 3, 2019

[1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.cluster.hierarchy import dendrogram, linkage
import scipy.cluster.hierarchy as shc
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import cophenet
import datetime
import keras
from numpy import array
```

```
Using TensorFlow backend.
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
```

```
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/divyae/miniconda3/envs/new_CWI/lib/python3.7/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

[2]:

```python
print(str(datetime.datetime.now()))
```

```
2019-10-02 17:56:29.767611
```

[64]:

```python
mname='full_280_clean.binary'
# mname='toy2.binary'
lname='phenotype_280'
fm=os.path.abspath(mname)
fl=os.path.abspath(lname)
```

```
#================
# Load raw data
#================

#read the unitig matrix as pandas dataframe
df1=pd.read_csv(fm, delim_whitespace=True)
df1=df1.set_index(list(df1)[0])

#get [resistant] & [sentive] strains as lists
resL=[]
senL=[]
with open(fl) as f:
    for line in f:
        line=line.rstrip()
        words=line.split('\t')
        #words[2]=words[2].split('/')[1].split('.f')[0]
        if int(words[1])==0:
            #sen[words[0]]=words[2]
            senL.append(words[0])
        elif int(words[1])==1:
            #res[words[0]]=words[2]
            resL.append(words[0])

resL.sort()
senL.sort()
```

[65]:

```
len(resL)
```

[65]: 47

[66]:

```
len(senL)
```

[66]: 233

[67]:

```
df1.shape
```

[67]: (1152012, 280)

[68]:

```
#resistant strains numpy array [(unitigs,samples)]
resM=df1[resL].values

#sensitive strains numpy array [(unitigs,samples)]
senM=df1[senL].values
```

[69]:

```
resM.shape
```

[69]: (1152012, 47)

[70]:

```
senM.shape
```

[70]:

```
(1152012, 233)
```

[71]:

```
test_frac=0.15
```

1 creating res_train & res_test

[72]:

```
res_idx = np.random.RandomState(seed=39).permutation(resM.shape[1])
```

[73]:

```
res_train_idx, res_test_idx = res_idx[round(resM.shape[1]*test_frac):],␣
 ↪res_idx[:round(resM.shape[1]*test_frac)]

# # the above is equivalent to:
# train_frac=1-test_frac
# res_train_idx, res_test_idx = res_idx[:round(resM.shape[1]*train_frac)],␣
 ↪res_idx[round(resM.shape[1]*train_frac):]

res_train, res_test = resM[:,res_train_idx], resM[:,res_test_idx]
```

[74]:

```
res_train.shape
```

[74]:

```
(1152012, 40)
```

[75]:

```
res_test.shape
```

[75]:

```
(1152012, 7)
```

[76]:

```
res_train_idx.sort()
res_test_idx.sort()
```

[77]:

```
res_test_idx
```

[77]:

```
array([ 2,  3, 15, 16, 19, 27, 38])
```

[78]:

```
res_train_idx
```

[78]:

```
array([ 0,  1,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 17, 18, 20, 21,
       22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40,
       41, 42, 43, 44, 45, 46])
```

2 creating sen_train & sen_test

[79]:
```
sen_idx = np.random.RandomState(seed=42).permutation(senM.shape[1])
```

[80]:
```
sen_train_idx, sen_test_idx = sen_idx[round(senM.shape[1]*test_frac):],␣
 ↪sen_idx[:round(senM.shape[1]*test_frac)]
sen_train, sen_test = senM[:,sen_train_idx], senM[:,sen_test_idx]
```

[81]:
```
sen_train.shape
```

[81]:
```
(1152012, 198)
```

[82]:
```
sen_test.shape
```

[82]:
```
(1152012, 35)
```

[83]:
```
sen_train_idx
```

[83]:
```
array([148,  16, 154,  79, 219, 201, 143, 176, 120,  45, 147,  86, 144,
        69, 230, 196, 223, 115,  97, 108, 167, 125,  38, 112, 183,  82,
       221,  73,  66, 226, 224,  67, 220,  29, 113, 124,  96,   5,  56,
       117, 140,  93,  65, 127, 197, 137,  31,  12,  35,  28,  42,  95,
       119, 156,  51, 159, 184,  76,  41, 228, 141,  78, 136,  26, 152,
       173, 171,   0,   2,  77,  46, 100, 114, 139, 180,  90,  85, 162,
       153,  98,  36, 135,  61,  22, 118, 150, 161,  33,  11, 227, 181,
         6,  27, 142,   4, 122,  32, 165,  62, 128, 209, 205,  70, 177,
        64,  44, 212,  40, 123,  23, 172, 168,  81,  39, 194,  47,  94,
       175,  43, 145, 158,   3, 105,  53, 133, 222, 178, 215,  49, 163,
        80,  34,   7, 110,  91,  83, 206, 211,  89,   8,  13,  59, 199,
       131,  17, 166,  72, 203, 134, 190, 213,  63,  54, 107,  50, 192,
       174, 193, 189, 229, 204, 169,  58,  48,  88,  21,  57, 160, 225,
       187, 191, 129,  37, 157, 218,   1,  52, 149, 130, 151, 103,  99,
       116,  87, 202,  74, 214, 210, 121, 232,  20, 188,  71, 106,  14,
        92, 179, 102])
```

[84]:
```
sen_test_idx
```

[84]:
```
array([ 84, 216, 231,   9, 126,  60,  55, 198, 111,  15, 208,  25, 155,
        68, 200, 195,  19, 185,  75,  24, 207,  10, 138, 186, 146, 164,
       182, 109,  18,  30, 104, 101, 217, 132, 170])
```

2.1 TESTb: stacking test__data, test__labels columns

[40]:

```
newR_test=np.column_stack((res_test,res_test,res_test,res_test,res_test))
newR_test_label=np.ones((1,newR_test.shape[1]))
sen_test_label=np.zeros((1,sen_test.shape[1]))

## test_data {Xtest} and test_label {ytest}
test_data, test_label =np.column_stack((newR_test,sen_test)), np.
 ↪column_stack((newR_test_label,sen_test_label))

# test_label.shape
# #(1, 70)

test_shuf_idx = np.random.RandomState(seed=41).permutation(test_data.shape[1])
test_dataT, test_labelT = test_data.T, test_label.T
Ftest_dataT, Ftest_labelT = test_dataT[test_shuf_idx],␣
 ↪test_labelT[test_shuf_idx]

# #shuffled and transposed from test-data. Nothing fancy
# Ftest_dataT.shape
# #(70, 1152012)
```

[41]:

```
test_shuf_idx
```

[41]:

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

[42]:

```
#========================================
# GET RIGHT SHAPE FOR {Xtest} & {ytest}
#========================================
FtestTb = np.reshape(Ftest_dataT, (Ftest_dataT.shape[0], Ftest_dataT.
 ↪shape[1],1))
Ftest_labelTb=keras.utils.to_categorical(Ftest_labelT, num_classes=2)
```

[43]:

```
np.
 ↪column_stack((res_test_idx,res_test_idx,res_test_idx,res_test_idx,res_test_idx))
```

[43]:

```
array([[ 2,  2,  2,  2,  2],
       [ 3,  3,  3,  3,  3],
       [15, 15, 15, 15, 15],
       [16, 16, 16, 16, 16],
       [19, 19, 19, 19, 19],
       [27, 27, 27, 27, 27],
       [38, 38, 38, 38, 38]])
```

```
[44]: len(Ftest_labelT)
```

```
[44]: 70
```

```
[45]: res_UStest_idx=np.
       ↪column_stack((res_test_idx,res_test_idx,res_test_idx,res_test_idx,res_test_idx)).
       ↪reshape(35,)
```

```
[46]: res_UStest_idx
```

```
[46]: array([ 2,  2,  2,  2,  2,  3,  3,  3,  3,  3, 15, 15, 15, 15, 15, 16, 16,
              16, 16, 16, 19, 19, 19, 19, 19, 27, 27, 27, 27, 27, 38, 38, 38, 38,
              38])
```

```
[47]: Rlist=res_UStest_idx.tolist()
      Rstring=','.join(str(e)+ '_R' for e in Rlist)
```

```
[48]: Rstring
```

```
[48]: '2_R,2_R,2_R,2_R,2_R,3_R,3_R,3_R,3_R,3_R,15_R,15_R,15_R,15_R,15_R,16_R,16_R,16_R
      ,16_R,16_R,19_R,19_R,19_R,19_R,19_R,27_R,27_R,27_R,27_R,27_R,38_R,38_R,38_R,38_R
      ,38_R'
```

```
[49]: Slist=sen_test_idx.tolist()
      Sstring=','.join(str(e)+ '_S' for e in Slist)
```

```
[50]: Sstring
```

```
[50]: '84_S,216_S,231_S,9_S,126_S,60_S,55_S,198_S,111_S,15_S,208_S,25_S,155_S,68_S,200
      _S,195_S,19_S,185_S,75_S,24_S,207_S,10_S,138_S,186_S,146_S,164_S,182_S,109_S,18_
      S,30_S,104_S,101_S,217_S,132_S,170_S'
```

```
[51]: Tstring=Rstring+','+Sstring
```

```
[52]: Sproper=list(Sstring.split(","))
      Rproper=list(Rstring.split(","))
      Tproper=list(Tstring.split(","))
```

```
[53]: test_shuf_idx
```

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

```
[54]: TproperNP=array(Tproper)
```

[55]:

```
TproperNP
```

[55]:

```
array(['2_R', '2_R', '2_R', '2_R', '2_R', '3_R', '3_R', '3_R', '3_R',
       '3_R', '15_R', '15_R', '15_R', '15_R', '15_R', '16_R', '16_R',
       '16_R', '16_R', '16_R', '19_R', '19_R', '19_R', '19_R', '19_R',
       '27_R', '27_R', '27_R', '27_R', '27_R', '38_R', '38_R', '38_R',
       '38_R', '38_R', '84_S', '216_S', '231_S', '9_S', '126_S', '60_S',
       '55_S', '198_S', '111_S', '15_S', '208_S', '25_S', '155_S', '68_S',
       '200_S', '195_S', '19_S', '185_S', '75_S', '24_S', '207_S', '10_S',
       '138_S', '186_S', '146_S', '164_S', '182_S', '109_S', '18_S',
       '30_S', '104_S', '101_S', '217_S', '132_S', '170_S'], dtype='<U5')
```

[56]:

```
TproperNP[test_shuf_idx]
```

[56]:

```
array(['18_S', '16_R', '16_R', '38_R', '2_R', '164_S', '111_S', '27_R',
       '109_S', '38_R', '170_S', '75_S', '186_S', '3_R', '27_R', '101_S',
       '9_S', '138_S', '15_R', '3_R', '16_R', '19_R', '200_S', '60_S',
       '198_S', '15_R', '2_R', '24_S', '216_S', '27_R', '3_R', '38_R',
       '207_S', '3_R', '126_S', '146_S', '231_S', '217_S', '2_R', '2_R',
       '25_S', '38_R', '15_R', '16_R', '182_S', '155_S', '38_R', '16_R',
       '185_S', '15_R', '19_R', '27_R', '19_S', '55_S', '19_R', '19_R',
       '68_S', '3_R', '195_S', '132_S', '2_R', '10_S', '15_S', '208_S',
       '27_R', '19_R', '104_S', '15_R', '84_S', '30_S'], dtype='<U5')
```

[57]:

```
Tarray=TproperNP[test_shuf_idx]
Tlist=Tarray.tolist()
```

[58]:

```
with open('sanityB.txt', "w") as f:
    f.writelines("ID \t Class \n")
    f.writelines(map("{}\t{}\n".format, Tlist, Ftest_labelT))
```

[59]:

```
Ftest_labelTb.shape
```

[59]:

```
(70, 2)
```

[61]:

```
FtestTb.shape
```

[61]:

```
(70, 1152012, 1)
```

[62]:

```
# # Test Imbalanced binary data
# np.save('testim_x.npy', FtestTim)
# np.save('testim_y.npy', Ftest_labelTim)

# # Test Balanced binary data
# np.save('testb_x.npy', FtestTb)
# np.save('testb_y.npy', Ftest_labelTb)
```

[ ]:

```
```

2.2 TESTim: stacking test_data, test_labels columns

[104]:

```
newR_test=(res_test)
newR_test_label=np.ones((1,newR_test.shape[1]))
sen_test_label=np.zeros((1,sen_test.shape[1]))

## test_data {Xtest} and test_label {ytest}
test_data, test_label =np.column_stack((res_test,sen_test)), np.
 ↪column_stack((newR_test_label,sen_test_label))
```

[105]:

```
test_data.shape
```

[105]:

```
(1152012, 42)
```

[106]:

```
test_shuf_idx = np.random.RandomState(seed=42).permutation(test_data.shape[1])
testim_shuf_idx = np.random.RandomState(seed=42).permutation(test_data.shape[1])
test_dataT, test_labelT = test_data.T, test_label.T
Ftest_dataT, Ftest_labelT = test_dataT[test_shuf_idx],␣
 ↪test_labelT[test_shuf_idx]

# #shuffled and transposed from test-data. Nothing fancy
# Ftest_dataT.shape
# #(42, 1152012) (hopefully)
```

[ ]:

```
```

[107]:

```
#========================================
# GET RIGHT SHAPE FOR {Xtest} & {ytest}
#========================================
FtestTim = np.reshape(Ftest_dataT, (Ftest_dataT.shape[0], Ftest_dataT.
 ↪shape[1],1))
Ftest_labelTim=keras.utils.to_categorical(Ftest_labelT, num_classes=2)
```

[108]:

```
rlist=res_test_idx.tolist()
slist=sen_test_idx.tolist()

sstring=','.join(str(e)+ '_S' for e in slist)
rstring=','.join(str(e)+ '_R' for e in rlist)
tstring=rstring+','+sstring
tproper=list(tstring.split(","))
tproperNP=array(tproper)
```

[109]:

```
tproperNP
```

[109]:
```
array(['2_R', '3_R', '15_R', '16_R', '19_R', '27_R', '38_R', '84_S',
       '216_S', '231_S', '9_S', '126_S', '60_S', '55_S', '198_S', '111_S',
       '15_S', '208_S', '25_S', '155_S', '68_S', '200_S', '195_S', '19_S',
       '185_S', '75_S', '24_S', '207_S', '10_S', '138_S', '186_S',
       '146_S', '164_S', '182_S', '109_S', '18_S', '30_S', '104_S',
       '101_S', '217_S', '132_S', '170_S'], dtype='<U5')
```

[110]:
```
tproperNP[testim_shuf_idx]

# array(['75_S', '55_S', '216_S', '24_S', '19_R', '217_S', '155_S', '138_S',
#        '186_S', '38_R', '207_S', '60_S', '208_S', '231_S', '111_S',
#        '182_S', '15_S', '30_S', '18_S', '2_R', '185_S', '27_R', '126_S',
#        '132_S', '3_R', '146_S', '200_S', '15_R', '164_S', '170_S', '16_R',
#        '104_S', '19_S', '109_S', '9_S', '195_S', '25_S', '68_S', '84_S',
#        '198_S', '10_S', '101_S'], dtype='<U5')
```

[110]:
```
array(['75_S', '55_S', '216_S', '24_S', '19_R', '217_S', '155_S', '138_S',
       '186_S', '38_R', '207_S', '60_S', '208_S', '231_S', '111_S',
       '182_S', '15_S', '30_S', '18_S', '2_R', '185_S', '27_R', '126_S',
       '132_S', '3_R', '146_S', '200_S', '15_R', '164_S', '170_S', '16_R',
       '104_S', '19_S', '109_S', '9_S', '195_S', '25_S', '68_S', '84_S',
       '198_S', '10_S', '101_S'], dtype='<U5')
```

[111]:
```
tproperNP[testim_shuf_idx]
tarray=tproperNP[test_shuf_idx]
tlist=tarray.tolist()

with open('sanityIm.txt', "w") as f:
    f.writelines("ID \t Class \n")
    f.writelines(map("{}\t{}\n".format, tlist, Ftest_labelT))
```

[113]:
```
FtestTim.shape
```

[113]:
```
(42, 1152012, 1)
```

[ ]:
```

```

3 TRAIN shenanigens

[114]:
```
# ==============================
# THIS IS WRONG, SAMPLING SHOULD BE AFTER CV SPLITS
# oversampling res_train, for 15% test split this is 5 times.
#newR=np.column_stack((res_train,res_train,res_train,res_train,res_train))
# ==============================
```

[115]:
```
# ==============================
# THIS IS THE CORRECT WAY
# ==============================
newR=res_train
newR_label=np.ones((1,newR.shape[1]))

#for 15% test split, we have right amount of sen
newS=sen_train
newS_label=np.zeros((1,newS.shape[1]))
```

[116]:
```
newR.shape
```

[116]:
```
(1152012, 40)
```

[117]:
```
newS.shape
```

[117]:
```
(1152012, 198)
```

[ ]:
```

```

3.1 TRAIN: save newR and newS as they are

[118]:
```
newRT, newR_labelT = newR.T, newR_label.T
FnewRT = np.reshape(newRT, (newRT.shape[0], newRT.shape[1],1))
FnewR_labelT = keras.utils.to_categorical(newR_labelT, num_classes=2)

np.save('res_x.npy', FnewRT)
np.save('res_y.npy', FnewR_labelT)
```

[119]:
```
FnewRT.shape
```

[119]:
```
(40, 1152012, 1)
```

[120]:
```
FnewR_labelT.shape
```

[120]:
```
(40, 2)
```

[121]:
```
newST, newS_labelT = newS.T, newS_label.T

rand1=np.random.RandomState(seed=43).randint(0,newST.shape[0])
rand2=np.random.RandomState(seed=44).randint(0,newST.shape[0])

trys_x=np.stack((newST[rand1],newST[rand2]))
trys_y=np.stack((newS_labelT[rand1],newS_labelT[rand2]))

Fsdata=np.concatenate((newST,trys_x))
```

```
Fslabels=np.concatenate((newS_labelT,trys_y))

FnewST = np.reshape(Fsdata, (Fsdata.shape[0], Fsdata.shape[1],1))
FnewS_labelT = keras.utils.to_categorical(Fslabels, num_classes=2)

np.save('sen_x.npy', FnewST)
np.save('sen_y.npy', FnewS_labelT)
```

[122]:
```
FnewST.shape
```

[122]:
```
(200, 1152012, 1)
```

[123]:
```
FnewS_labelT.shape
```

[123]:
```
(200, 2)
```

3.2 TRAIN: stacking train_data, train_labels columns

[124]:
```
# did a funny (delta) up-sampling of sensitive guys, hence trivial yet important␣
 ↪step
newS, newS_label = Fsdata.T, Fslabels.T
```

[125]:
```
## train_data {X} and train_label {y}
train_data, train_label =np.column_stack((newR,newS)), np.
 ↪column_stack((newR_label,newS_label))
```

[126]:
```
train_data.shape
# note, after 5x oversampling the 40 resistant:
    # we have equal resistant and  sensitive (both are 200)
```

[126]:
```
(1152012, 240)
```

[127]:
```
# shuffle just because we can?
train_shuf_idx = np.random.permutation(train_data.shape[1])
train_dataT, train_labelT=train_data.T, train_label.T
Ftrain_dataT, Ftrain_labelT=train_dataT[train_shuf_idx],␣
 ↪train_labelT[train_shuf_idx]
```

[128]:
```
Ftrain_dataT.shape
```

[128]:
```
(240, 1152012)
```

[129]:
```
#==========================================
# GET RIGHT SHAPE FOR {Xtrain} & {Ytrain}
#==========================================
```

[130]:

```
FtrainT = np.reshape(Ftrain_dataT, (Ftrain_dataT.shape[0], Ftrain_dataT.
 ↪shape[1],1))
Ftrain_labelT=keras.utils.to_categorical(Ftrain_labelT, num_classes=2)
```

[131]:

```
FtrainT.shape
```

[131]:

```
(240, 1152012, 1)
```

[132]:

```
Ftrain_dataT.size
```

[132]:

```
276482880
```

[133]:

```
# # Train binary data
# np.save('train_x.npy', FtrainT)
# np.save('train_y.npy', Ftrain_labelT)

# Test Imbalanced binary data
np.save('testim_x.npy', FtestTim)
np.save('testim_y.npy', Ftest_labelTim)

# Test Balanced binary data
np.save('testb_x.npy', FtestTb)
np.save('testb_y.npy', Ftest_labelTb)

#Human readable data
np.savetxt('train_y.txt', Ftrain_labelT)
np.savetxt('testim_y.txt', Ftest_labelTim)
np.savetxt('testb_y.txt', Ftest_labelTb)
```

[134]:

```
print(str(datetime.datetime.now()))
```

```
2019-10-02 18:13:37.401529
```

[ ]:

```
```

4 End of cleaning & data wrangling part

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[40]:
```
####EOtrial
```

[9]:
```
fm=os.path.abspath(mname)
fl=os.path.abspath(lname)
fakeSamples=233
```

[11]:
```
df1=pd.read_csv(os.path.abspath(mname), delim_whitespace=True)
df1=df1.set_index(list(df1)[0])
#df1.shape
#(1000, 280)
```

[12]:
```
unitigCount=df1.values.shape[0]
```

[13]:
```
bg_null=np.random.randint(0,2,size=(unitigCount,fakeSamples))
```

[14]:
```
bg_null.shape
```

[14]:
```
(1000, 233)
```

[15]:
```
bg_null[:,1:4].shape
```

[15]:
```
(1000, 3)
```

[16]:
```
# res={}
# sen={}
resL=[]
senL=[]
```

```
with open(fl) as f:
    for line in f:
        line=line.rstrip()
        words=line.split('\t')
        words[2]=words[2].split('/')[1].split('.f')[0]
        if int(words[1])==0:
            #sen[words[0]]=words[2]
            senL.append(words[0])
        else:
            #res[words[0]]=words[2]
            resL.append(words[0])
```

[17]:

```
resL.sort()
senL.sort()
```

[18]:

```
resL
```

[18]:

```
['WH-SGI-V-07050',
 'WH-SGI-V-07053',
 'WH-SGI-V-07071',
 'WH-SGI-V-07073',
 'WH-SGI-V-07165',
 'WH-SGI-V-07179',
 'WH-SGI-V-07181',
 'WH-SGI-V-07227',
 'WH-SGI-V-07230',
 'WH-SGI-V-07233',
 'WH-SGI-V-07236',
 'WH-SGI-V-07247',
 'WH-SGI-V-07253',
 'WH-SGI-V-07256',
 'WH-SGI-V-07259',
 'WH-SGI-V-07268',
 'WH-SGI-V-07276',
 'WH-SGI-V-07309',
 'WH-SGI-V-07320',
 'WH-SGI-V-07322',
 'WH-SGI-V-07323',
 'WH-SGI-V-07324',
 'WH-SGI-V-07325',
 'WH-SGI-V-07327',
 'WH-SGI-V-07329',
 'WH-SGI-V-07415',
 'WH-SGI-V-07425',
 'WH-SGI-V-07484',
 'WH-SGI-V-07486',
 'WH-SGI-V-07487',
 'WH-SGI-V-07496',
 'WH-SGI-V-07622',
 'WH-SGI-V-07625',
 'WH-SGI-V-07626',
 'WH-SGI-V-07627',
 'WH-SGI-V-07628',
 'WH-SGI-V-07633',
 'WH-SGI-V-07635',
 'WH-SGI-V-07638',
 'WH-SGI-V-07643',
 'WH-SGI-V-07644',
 'WH-SGI-V-07646',
 'WH-SGI-V-07648',
 'WH-SGI-V-07651',
 'WH-SGI-V-07687',
 'WH-SGI-V-07702',
 'WH-SGI-V-07703']
```

```
[19]: df1.shape
```

```
[19]: (1000, 280)
```

```
[20]: #resistant strains
      resdf=df1[resL]
      resM=resdf.values
```

```
[21]: #sensitive strains
      sendf=df1[senL]
      senM=sendf.values
```

```
[22]: df1.values.nbytes
```

```
[22]: 2240000
```

```
[23]: senM.shape
```

```
[23]: (1000, 233)
```

```
[24]: resM.shape
```

```
[24]: (1000, 47)
```

```
[25]: bgNull=bg_null.sum(axis=1)
      stats.describe(bgNull)
```

```
[25]: DescribeResult(nobs=1000, minmax=(95, 137), mean=116.544,
      variance=54.45251651651652, skewness=-0.09454981486215464,
kurtosis=-0.14390013645297817)
```

```
[26]: senTrue=senM.sum(axis=1)
      stats.describe(senTrue)
```

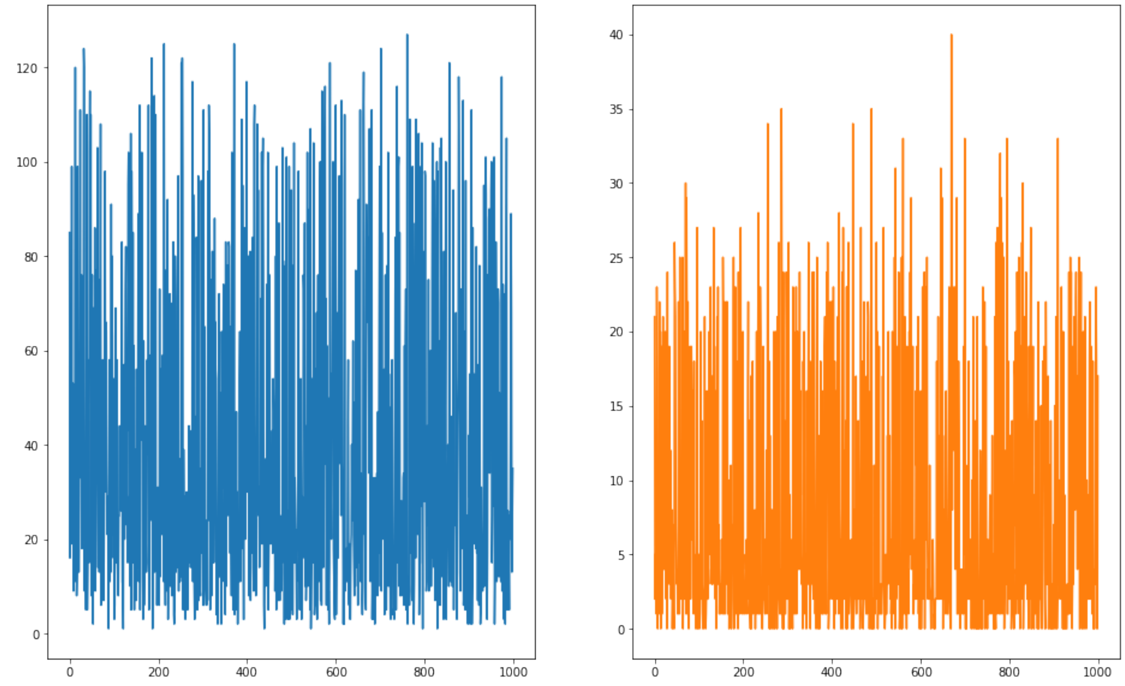
```
[26]: DescribeResult(nobs=1000, minmax=(1, 127), mean=40.041,
      variance=1115.77109009009, skewness=0.8219240841542729,
      kurtosis=-0.5179730527320436)
```

```
[27]: resTrue=resM.sum(axis=1)
      stats.describe(resTrue)
```

```
[27]: DescribeResult(nobs=1000, minmax=(0, 40), mean=8.24, variance=76.0004004004004,
      skewness=0.9949397525032871, kurtosis=-0.22469685464554878)
```

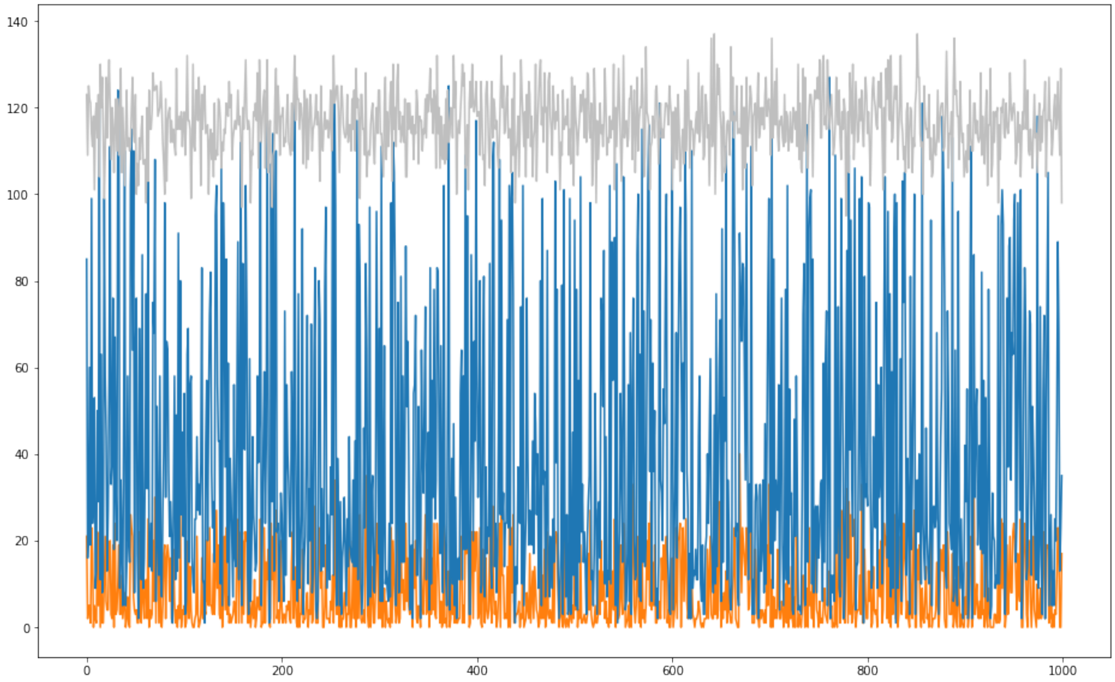
```
[28]: plt.figure(figsize=(16,10))
      plt.subplot(121)
      plt.plot(senTrue,color='C0')
      plt.subplot(122)
      plt.plot(resTrue,color='C1')
```

```
[28]: [<matplotlib.lines.Line2D at 0x7ff7f68f9588>]
```



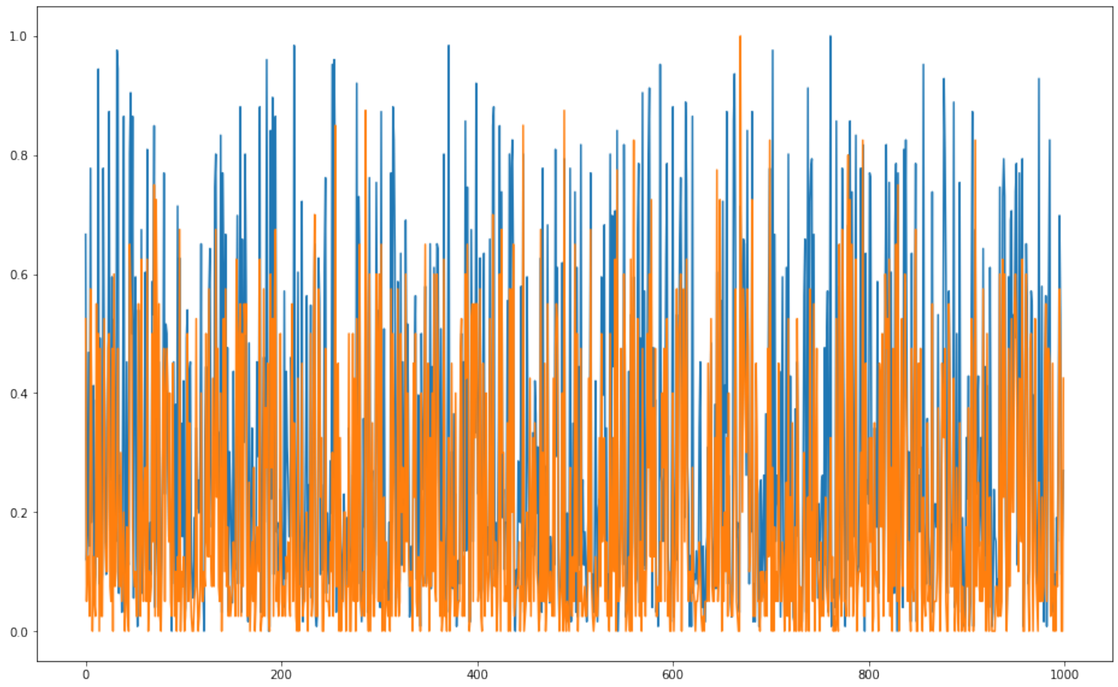
```
[29]: plt.figure(figsize=(16,10))
      plt.plot(resTrue,color='C1')
      plt.plot(senTrue,color='C0')
      plt.plot(bgNull,color='0.75')
```

```
[29]: [<matplotlib.lines.Line2D at 0x7ff7f683dbe0>]
```



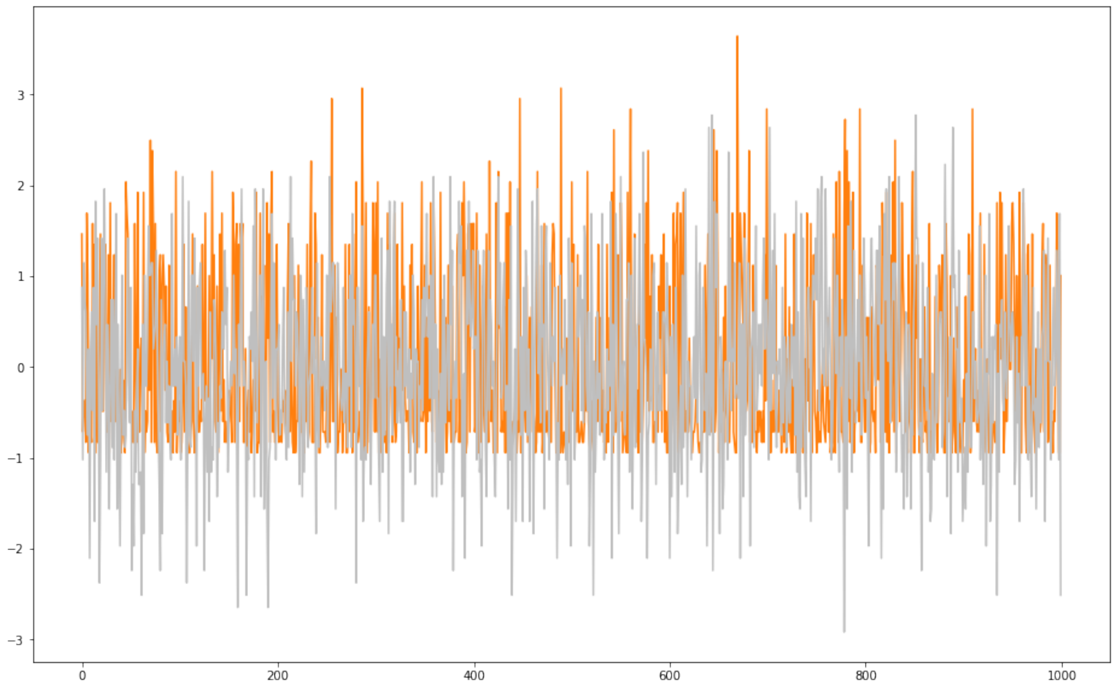
```
[30]: plt.figure(figsize=(16,10))
      plt.plot((senTrue-min(senTrue))/(max(senTrue)-min(senTrue)),color='C0')
      plt.plot((resTrue-min(resTrue))/(np.ptp(resTrue)),color='C1')
      #plt.plot((bgNull-min(bgNull))/(np.ptp(bgNull)),color='0.75')
```

```
[30]: [<matplotlib.lines.Line2D at 0x7ff7f67fae10>]
```



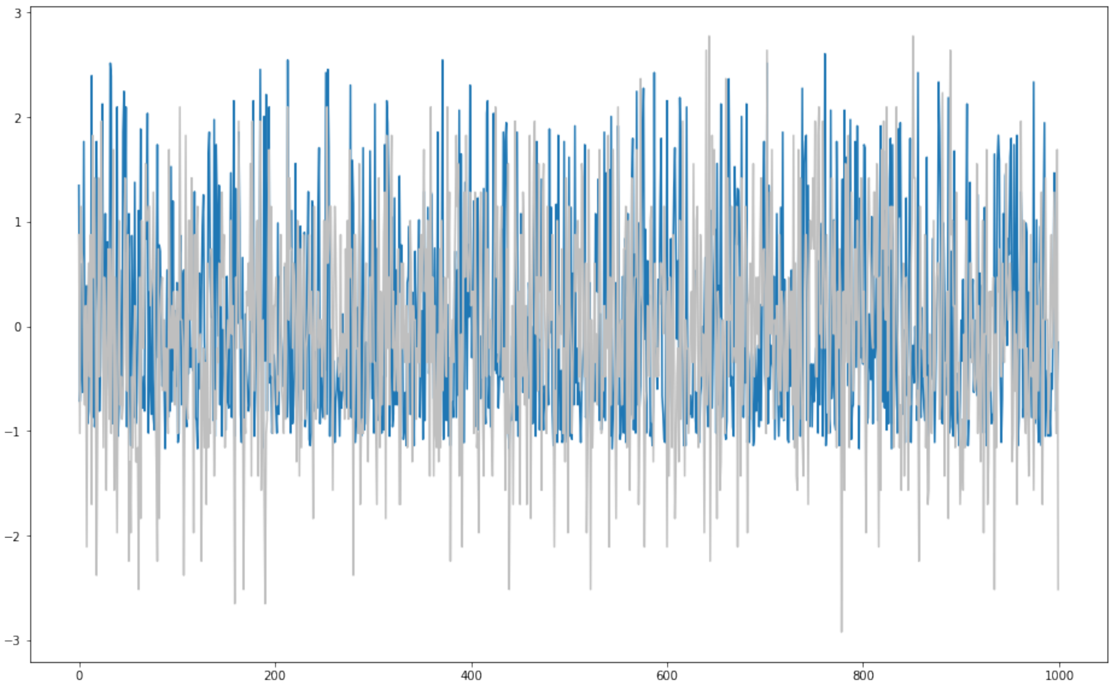
[31]:
```
plt.figure(figsize=(16,10))
#plt.plot((senTrue-np.mean(senTrue))/(np.std(senTrue)), color='C0')
plt.plot((resTrue-np.mean(resTrue))/(np.std(resTrue)), color='C1')
plt.plot((bgNull-np.mean(bgNull))/(np.std(bgNull)), color='0.75')
```

[31]:
```
[<matplotlib.lines.Line2D at 0x7ff7f4325438>]
```



```
[32]: plt.figure(figsize=(16,10))
      plt.plot((senTrue-np.mean(senTrue))/(np.std(senTrue)),color='C0')
      #plt.plot((resTrue-np.mean(resTrue))/(np.std(resTrue)),color='C1')
      plt.plot((bgNull-np.mean(bgNull))/(np.std(bgNull)),color='0.75')
```

```
[32]: [<matplotlib.lines.Line2D at 0x7ff7f4285908>]
```



[33]:

```
df2=df1.values
```

[34]:

```
dt = linkage(df2.transpose(), 'ward')
rDt=linkage(resM.transpose(),'ward')
sDt=linkage(senM.transpose(),'ward')
callt, coph_dists_allt= cophenet(dt, pdist(df2.transpose()))
crest, coph_dists_rest= cophenet(rDt, pdist(resM.transpose()))
csent, coph_dists_sent= cophenet(sDt, pdist(senM.transpose()))
callt
```

[34]:

```
0.7671489610237536
```

[35]:

```
dt = linkage(df2.transpose(), 'single')
rDt=linkage(resM.transpose(),'single')
sDt=linkage(senM.transpose(),'single')
callt, coph_dists_allt= cophenet(dt, pdist(df2.transpose()))
crest, coph_dists_rest= cophenet(rDt, pdist(resM.transpose()))
csent, coph_dists_sent= cophenet(sDt, pdist(senM.transpose()))
```

[36]:

```
dt = linkage(df2.transpose(), 'complete')
rDt=linkage(resM.transpose(),'complete')
sDt=linkage(senM.transpose(),'complete')
callt, coph_dists_allt= cophenet(dt, pdist(df2.transpose()))
crest, coph_dists_rest= cophenet(rDt, pdist(resM.transpose()))
csent, coph_dists_sent= cophenet(sDt, pdist(senM.transpose()))
```

[37]:
```
callt
```

[37]:
```
0.9671932770546021
```

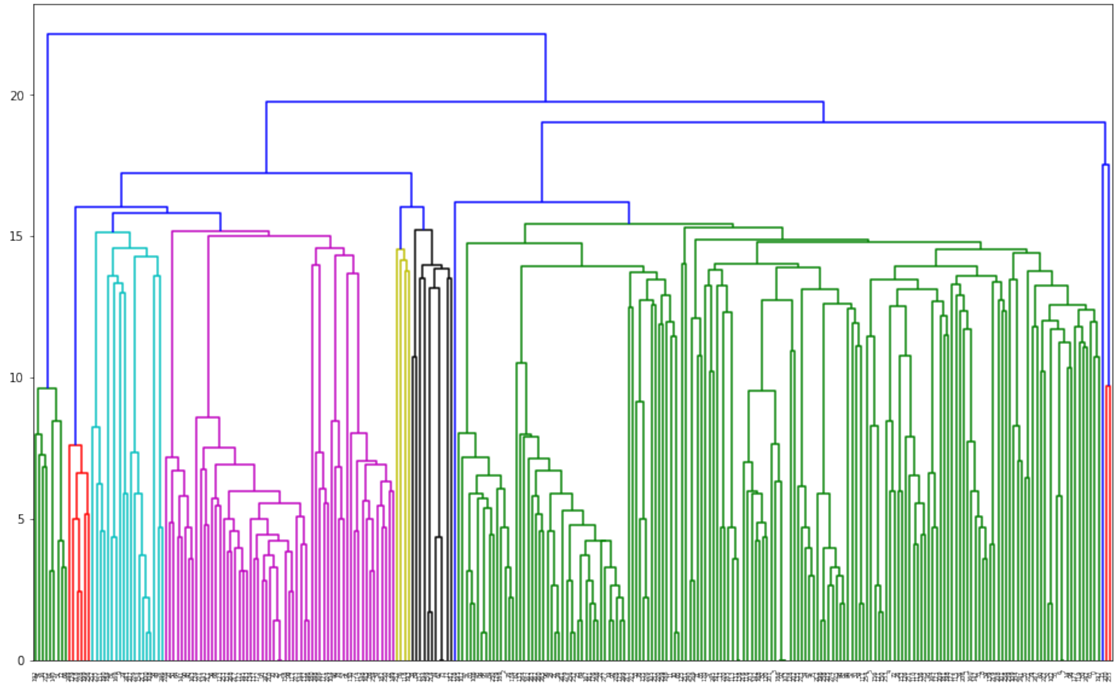
[38]:
```
crest
```

[38]:
```
0.9912775270201886
```

[39]:
```
csent
```

[39]:
```
0.9664330920954989
```

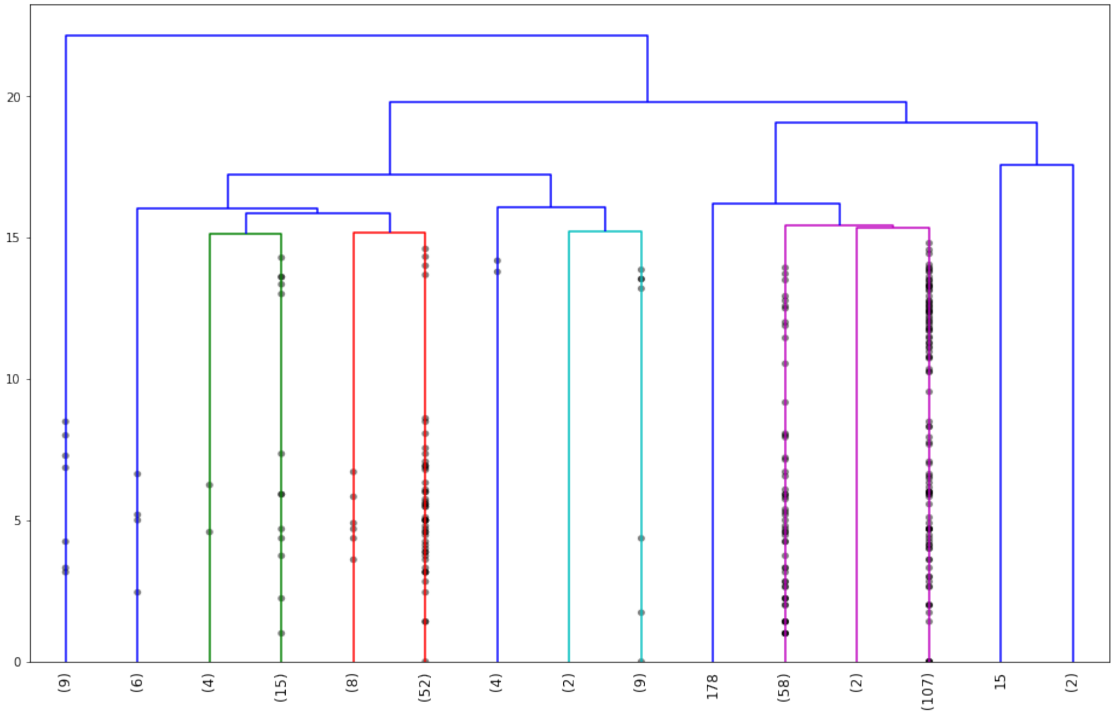
[40]:
```
plt.figure(figsize=(16, 10))
dendrogram(dt,
            orientation='top',
            #labels=labelList,
            #distance_sort='descending',
            #show_leaf_counts=True
          )
plt.show()
```



[41]:
```
np.where(df2[:,1]==1)
```

[41]:
```
(array([ 24,  26,  32,  39,  42,  48,  60,  63,  85,  90,  92,  94, 103,
        125, 130, 132, 134, 138, 140, 148, 157, 158, 163, 185, 191, 193,
        213, 217, 239, 244, 245, 252, 255, 261, 262, 279, 280, 287, 300,
        305, 308, 312, 314, 316, 343, 352, 360, 366, 383, 394, 399, 407,
        417, 434, 436, 442, 445, 482, 502, 504, 519, 530, 536, 538, 565,
        566, 569, 571, 576, 592, 594, 600, 607, 609, 613, 620, 637, 646,
        651, 652, 658, 662, 668, 673, 674, 690, 693, 695, 702, 704, 716,
        718, 734, 738, 752, 755, 757, 761, 763, 767, 792, 801, 818, 819,
        823, 828, 838, 844, 856, 876, 877, 884, 907, 908, 915, 919, 924,
        934, 938, 946, 962, 977, 996]),)
```

[42]:
```
plt.figure(figsize=(16, 10))
dendo=dendrogram(dt,leaf_rotation=90.,  # rotates the x axis labels
           #leaf_font_size=8.,  # font size for the x axis labels
           truncate_mode='lastp',  # show only the last p merged clusters
           p=15,  # show only the last p merged clusters
           show_contracted=True
           )
plt.show()
```



[43]:
```
dendo['ivl']
```

[43]:
```
['(9)',
 '(6)',
 '(4)',
 '(15)',
 '(8)',
 '(52)',
 '(4)',
 '(2)',
 '(9)',
 '178',
 '(58)',
 '(2)',
 '(107)',
 '15',
 '(2)']
```

```
[44]: dendo['icoord']
```

```
[44]: [[25.0, 25.0, 35.0, 35.0],
       [45.0, 45.0, 55.0, 55.0],
       [30.0, 30.0, 50.0, 50.0],
       [15.0, 15.0, 40.0, 40.0],
       [75.0, 75.0, 85.0, 85.0],
       [65.0, 65.0, 80.0, 80.0],
       [27.5, 27.5, 72.5, 72.5],
       [115.0, 115.0, 125.0, 125.0],
       [105.0, 105.0, 120.0, 120.0],
       [95.0, 95.0, 112.5, 112.5],
       [135.0, 135.0, 145.0, 145.0],
       [103.75, 103.75, 140.0, 140.0],
       [50.0, 50.0, 121.875, 121.875],
       [5.0, 5.0, 85.9375, 85.9375]]
```

```
[45]: dendo['dcoord']
```

```
[45]: [[0.0, 15.132745950421556, 15.132745950421556, 0.0],
       [0.0, 15.165750888103101, 15.165750888103101, 0.0],
       [15.132745950421556,
        15.84297951775486,
        15.84297951775486,
        15.165750888103101],
       [0.0, 16.0312195418814, 16.0312195418814, 15.84297951775486],
       [0.0, 15.231546211727817, 15.231546211727817, 0.0],
       [0.0, 16.06237840420901, 16.06237840420901, 15.231546211727817],
       [16.0312195418814, 17.233687939614086, 17.233687939614086, 16.06237840420901],
       [0.0, 15.329709716755891, 15.329709716755891, 0.0],
       [0.0, 15.427248620541512, 15.427248620541512, 15.329709716755891],
       [0.0, 16.217274740226856, 16.217274740226856, 15.427248620541512],
       [0.0, 17.549928774784245, 17.549928774784245, 0.0],
[16.217274740226856,
 19.05255888325765,
 19.05255888325765,
 17.549928774784245],
[17.233687939614086, 19.77371993328519, 19.77371993328519, 19.05255888325765],
[0.0, 22.135943621178654, 22.135943621178654, 19.77371993328519]]
```

[46]:

```
# d = linkage(df2, 'ward')
# rD=linkage(resM,'ward')
# sD=linkage(senM,'ward')
# call, coph_dists_all= cophenet(d, pdist(df2))
# cres, coph_dists_res= cophenet(rD, pdist(resM))
# csen, coph_dists_sen= cophenet(sD, pdist(senM))
```

[47]:

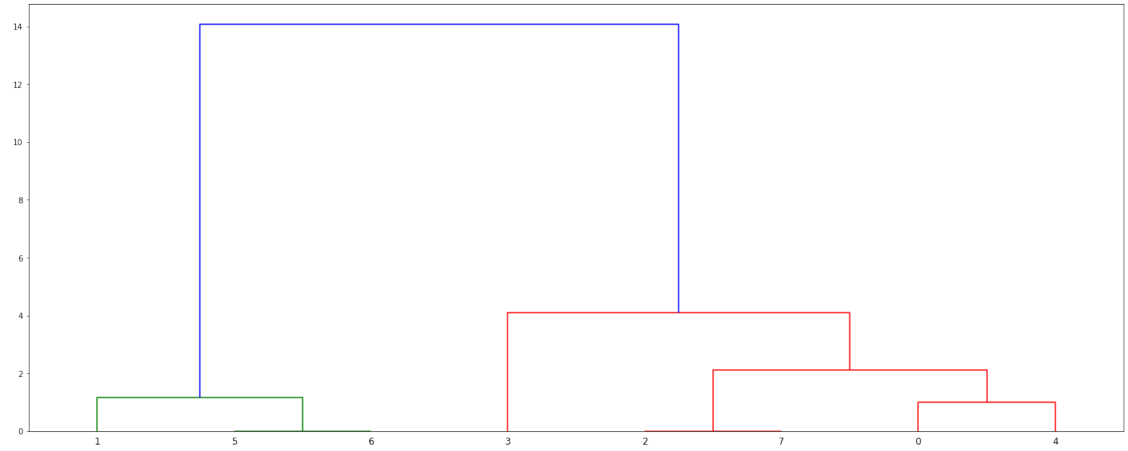
```
X = [[i] for i in [2, 8, 0, 4, 1, 9, 9, 0]]
X
```

[47]:

```
[[2], [8], [0], [4], [1], [9], [9], [0]]
```

[48]:

```
Z = linkage(X, 'ward')
fig = plt.figure(figsize=(25, 10))
dn = dendrogram(Z)
```



[ ]:

```
# ===========================================
# RANDOM NUMPY COMMANDS
# ===========================================
```

[49]:

```
senM.shape
```

[49]:

```
(1000, 233)
```

[50]:
```
#47,94,141,188
factor=2
resM.shape
```

[50]:
```
(1000, 47)
```

[51]:
```
bg_null.shape
```

[51]:
```
(1000, 233)
```

[151]:
```
####sampling columns from np arrays
idx=np.random.choice(senM.shape[1],size=resM.shape[1]*factor,replace=False)
```

[150]:
```
newS=senM[:,idx]
newS.shape
```

[150]:
```
(1000, 47)
```

[98]:
```
newS_label=np.zeros((1,newS.shape[1]))
newS_label
```

[98]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

[61]:
```
if factor==2:
    newR=np.column_stack((resM,resM))
if factor==3:
    newR=np.column_stack((resM,resM,resM))

newR.shape
```

[61]:
```
(1000, 94)
```

[99]:
```
newR_label=np.ones((1,newR.shape[1]))
newR_label
```

[99]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

[100]:

```
np.random.shuffle(np.transpose(newR))
np.random.shuffle(np.transpose(newS))

data=np.column_stack((newR,newS))
label=np.column_stack((newR_label,newS_label))
```

[101]:

```
data.shape
```

[101]:

```
(1000, 188)
```

[105]:

```
label.shape
```

[105]:

```
(1, 188)
```

[64]:

```
#Numpy basics
#c=np.transpose(b)
#newR.shape
# a=np.array([1, 2, 3, 4, 5])
# b = a + 1

# print (a)
# print (b)
# [1 2 3 4 5]
# [2 3 4 5 6]

# a=np.array([1, 2, 3, 4, 5])
# # Add 0 to `a`:
# b = a + 0

# print (a)
# print (b)
# [1 2 3 4 5]
# [1 2 3 4 5]

# b is a
# False
```

[87]:

```
# x1=np.random.rand(1000,newR.shape[1])

# indices1 = np.random.permutation(newR.shape[1])
# splitfrac1=0.7
# range1=round(newR.shape[1]*splitfrac1)

# training_idx, test_idx = indices1[:range1], indices[range1:]
# training, test = x1[:,training_idx], x1[:,test_idx]
# indices2 = np.random.permutation(test.shape[1])
# splitfrac2=0.5
# range2=round(test.shape[1]*splitfrac2)

# val_idx, test_idx = indices2[:range2], indices2[range2:]
# val, test = test[:,val_idx], x1[:,test_idx]
```

[142]:

```
#### does this work??

indices1 = np.random.permutation(data.shape[1])
#label.shape[1] is same as data.shape[1]
splitfrac1=0.7   #70 % training data ~30 % b/w test and val
range1=round(data.shape[1]*splitfrac1)

train_idx, test_idx = indices1[:range1], indices1[range1:]
train, test = data[:,train_idx], data[:,test_idx]
train_label, test_label= label[:,train_idx], label[:,test_idx]


indices2 = np.random.permutation(test.shape[1])
splitfrac2=0.5   #15% val and 15% test
range2=round(test.shape[1]*splitfrac2)

val_idx, test_idx = indices2[:range2], indices2[range2:]
val, test = test[:,val_idx], test[:,test_idx]
val_label, test_label = test_label[:,val_idx], test_label[:,test_idx]
```

[143]:

```
test1.shape
```

[143]:

```
(1000, 28)
```

[144]:

```
val.shape
```

[144]:

```
(1000, 28)
```

[145]:

```
test_idx
```

[145]:

```
array([28, 25, 16,  0, 51, 21, 32, 40,  5, 14, 12, 15, 50, 22, 43,  9, 20,
       27, 19, 37, 10, 54, 13, 30, 49, 44, 11, 38])
```

[146]:

```
val_idx
```

[146]:

```
array([47, 41,  8,  6, 35, 55, 39,  2, 36, 48, 17, 31, 45, 42, 46, 24, 34,
       33,  3, 52, 18,  7, 23,  1, 26,  4, 29, 53])
```

[147]:

```
val_label
```

[147]:
```
array([[0., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1.,
        0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 1.]])
```

[103]:
```
# from sklearn.model_selection import train_test_split

# data, labels = np.arange(40).reshape((20, 2)), range(20)
# data_train, data_test, labels_train, labels_test = train_test_split(data,␣
 ↪labels, test_size=0.20, random_state=42)

# x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.
 ↪3)
# x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5)
```

[108]:
```

```