SVD_ISA_Notebook

March 25, 2020

1 Jupyter notebook for biclustering

In this session we will study the transcriptional response to two different drugs usually prescribed to patients with heart problems in mice heart tissues. As you know there is a huge variability in how people's body respond to a drug. Bi-clustering analysis in this case can help us to understand which genes expression change when these drugs are used and which strains of mice are different in responding to each drug. We will use RNAseq data (~16,000 genes) from heart tissue of 18 inbred mouse strains treated with either the -blocker atenolol (ATE) or the -agonist isoproterenol (ISO) and also not treated with any drugs as controls. Note that the two drugs trigger opposite responses in heart, ATE is supposed to slow down the heartbeat and ISO is supposed to increase the heartbeat. This data is published with a paper at 2016 which you can find at the following link https://link.springer.com/article/10.1186/s12864-016-3059-6.

1.1 1. importing all the needed libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import zscore
from scipy.linalg import svd

## if we compare with pca
import seaborn as sns
from matplotlib.axes._axes import _log as matplotlib_axes_logger
from sklearn import decomposition
```

1.2 2. importing the data

2.1. importing gene ids

[2]:
```
gene_entrezid = pd.read_csv('./PGX_data/genes_entrezid.tab', sep='\t',␣
↪header=None)
gene_entrezid.replace(' ', np.nan, inplace=True)
gene_entrezid.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)
print('number of gene ids: ', gene_entrezid.shape[1])
gene_entrezid.values
```

```
number of gene ids:  16039
```

[2]:
```
array([[    11287,     11298,     11302, …, 101056688, 101056690,
        101056691]])
```

We have an array of 16039 gene ids labeled as 11287, 11298, 11302 and etc.

2.3. importing strains labels

[3]:
```
strains = pd.read_csv('./PGX_data/strains.tab',sep='\t',header=None)
strains.replace(' ', np.nan, inplace=True)
strains.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)
print('number of strains: ', strains.shape[1])
strains.values
```

```
number of strains:  18
```

[3]:
```
array([['AJ', 'BALBcByJ', 'BALBcJ', 'C3HHeJ', 'C57BL6J', 'C57BLKSJ',
        'C58J', 'CBAJ', 'DBA2J', 'FVBNJ', 'ILnJ', 'LPJ', 'NODShiLtJ',
        'NZBBLNJ', 'PLJ', 'SJLJ', 'SMJ', 'SWRJ']], dtype=object)
```

We have an array of 18 different mouse strains labeled as 'AJ','BALBcByJ', 'BALBcJ' and etc.

2.2. importing samples

[4]:
```
samples = pd.read_csv('./PGX_data/samples.tab',sep='\t',header=None)
samples.replace(' ', np.nan, inplace=True)
samples.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)
print('number of samples: ', samples.shape[1])
samples = samples.append(samples.iloc[0].str.split("-", n = 0, expand = True).
 ↪transpose(),ignore_index=True)
samples
```

```
number of samples:  160
```

[4]:

	0	1	2	3	4	5	…	150	151	152	153	154	155	156	157	158	159
0	DBA2J-CTR	DBA2J-CTR	DBA2J-ATE	DBA2J-ISO	DBA2J-ATE	DBA2J-ATE	…	BALBcByJ-ATE	ILnJ-ISO	ILnJ-CTR	ILnJ-ATE	ILnJ-ISO	BALBcByJ-ISO	BALBcByJ-ATE	BALBcByJ-CTR		
1	DBA2J	DBA2J	DBA2J	DBA2J	DBA2J	DBA2J	…	BALBcByJ	ILnJ	ILnJ	ILnJ	ILnJ	BALBcByJ	BALBcByJ	BALBcByJ		
2	CTR	CTR	ATE	ISO	ATE	ATE	…	ATE	ISO	CTR	ATE	ISO	ISO	ATE	CTR		

```
0  BALBcByJ-ISO  BALBcByJ-ATE
1      BALBcByJ      BALBcByJ
2           ISO           ATE

[3 rows x 160 columns]
```

We have 160 samples as shown in the first row of our dataframe labeled as DBA2J-CTR, DBA2J-CTR, DBA2J-ATE, DBA2J-ISO and so on. The first part of these labels (as separately shown in the second row of our dataframe) represents the strain labels and the second part of these labels (as separately shown in the third row of our dataframe) represents the contraol cases or drug used labels. 'CTR' label is used for controls, 'ATE' label is used for mice treated with -blocker atenolol drug and 'ISO' label is used for mice treated with -agonist isoproterenol drug.

Pay attention that these labels are not unique as there are 3 replicates of each condition. We have 18 strains, 3 different treatments, meaning we have 54 conditions.

2.4. importing gene expression data

[5]:
```
data = pd.read_csv('./PGX_data/tmm_data.tab',sep='\t',header=None)
data.head(n=3)
```

[5]:

	0	1	2	3	4	5	\
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1	131.840154	128.233034	138.600413	133.223447	114.517384	116.389696	
2	716.212727	378.870327	439.756865	768.090739	434.973593	333.650461	

	6	7	8	9	…	150	\
0	0.000000	3.881266	0.000000	0.000000	…	0.000000	
1	11.826902	118.378609	103.597839	135.608680	…	157.784380	
2	775.844797	1040.179252	342.323294	520.881023	…	533.311204	

	151	152	153	154	155	156	\
0	0.000000	0.000000	0.000000	3.949651	0.000000	0.000000	
1	109.953254	103.338136	112.936687	9.874128	109.767002	78.055723	
2	790.106571	583.910631	507.241497	694.151186	749.747828	310.921965	

	157	158	159
0	0.000000	0.000000	0.000000
1	109.613124	91.168949	159.105552
2	312.338471	450.000582	306.696886

[3 rows x 160 columns]

[6]:
```
[num_genes, num_exp] = data.shape
print('Number of genes are: ', num_genes)
print('Number of samples are: ',num_exp)
print('Number of NaNs in our dataset: ',data.isna().sum().sum())
```

```
print('Number of exact zeros in our dataset: ',(data == 0.0).astype(int).sum().
 ↪sum())
```

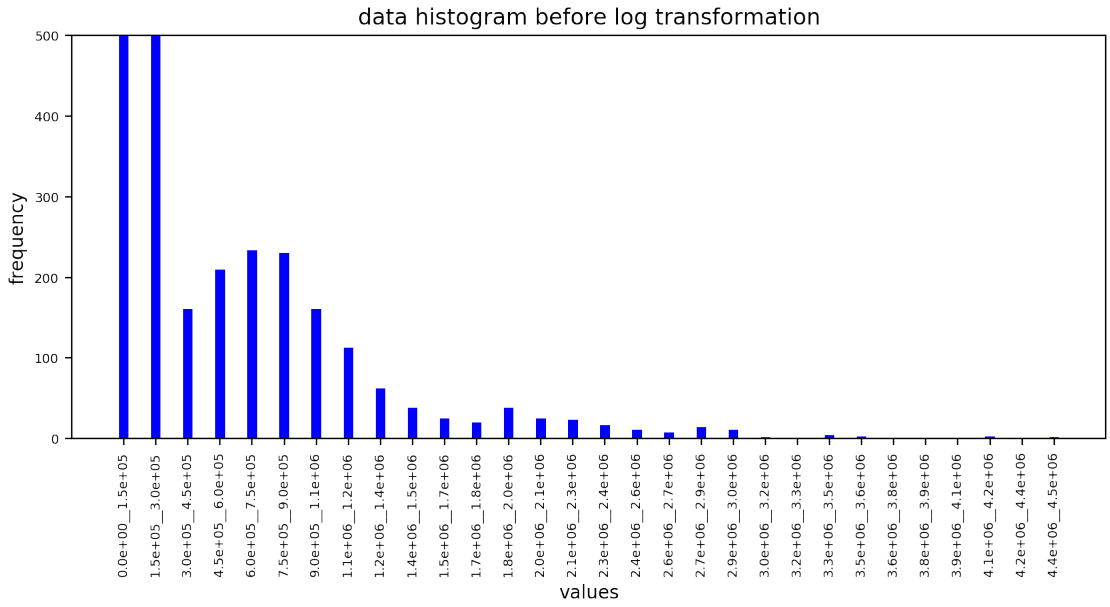
```
Number of genes are:  16039
Number of samples are:  160
Number of NaNs in our dataset:  0
Number of exact zeros in our dataset:  229077
```

1.3 3. Visualization of our gene expression data, normalization and reduction of dataset

3.1. raw data histogram Let's plot a histogram of all gene expression values grouped into 30 bins.

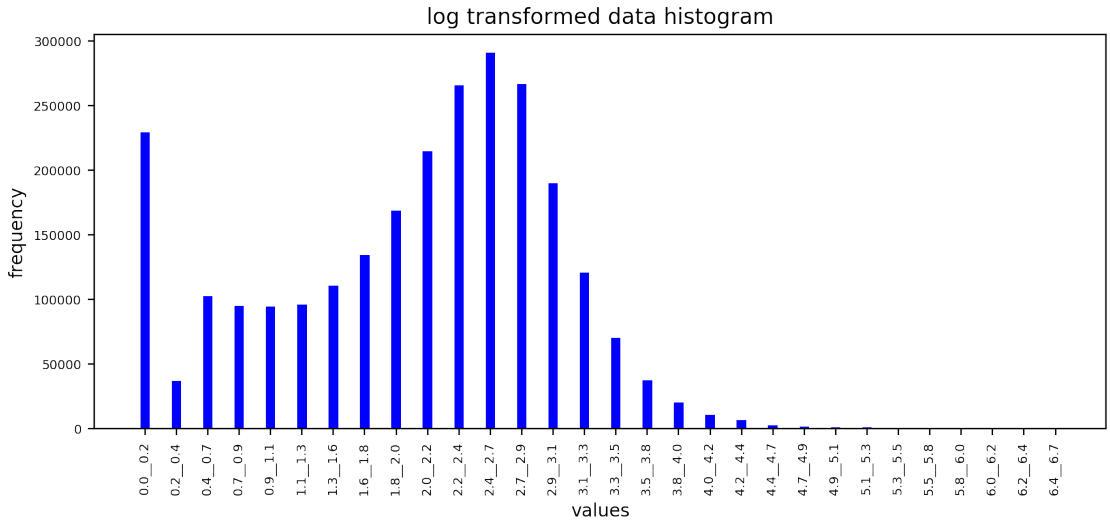
[7]:
```
print('range of our dataframe: ['+str(data.values.flatten().min())+' - '+"{:.
 ↪1e}".format(data.values.flatten().max())+']')

num_bins = 30
y_values, bin_edges=np.histogram(data.to_numpy(),bins=num_bins)
bin_centers = 0.5*(bin_edges[1:] + bin_edges[:-1])

plt.figure(figsize=(10, 4),dpi=200)
plt.rcParams.update({'font.size': 7})
plt.xlabel('values', fontsize=10)
plt.ylabel('frequency', fontsize=10)
plt.title('data histogram before log transformation', fontsize=12)
plt.bar(np.arange(0,num_bins), y_values, width = 0.3, color='blue')
plt.xticks(np.arange(0,num_bins),["{:.1e}".format(bin_edges[i])+"__"+"{:.1e}".
 ↪format(bin_edges[i+1]) for i in np.
 ↪arange(0,len(bin_centers))],rotation='vertical')
plt.ylim(0, 500)
plt.show()
```

```
range of our dataframe: [0.0 - 4.5e+06]
```



3.2. log tranformation of data and plotting histogram Log transformation is the most popular method to tranform a skewed data to near-normality. So let's log transform our data and then plot the histogram, again grouping data values into 30 bins.

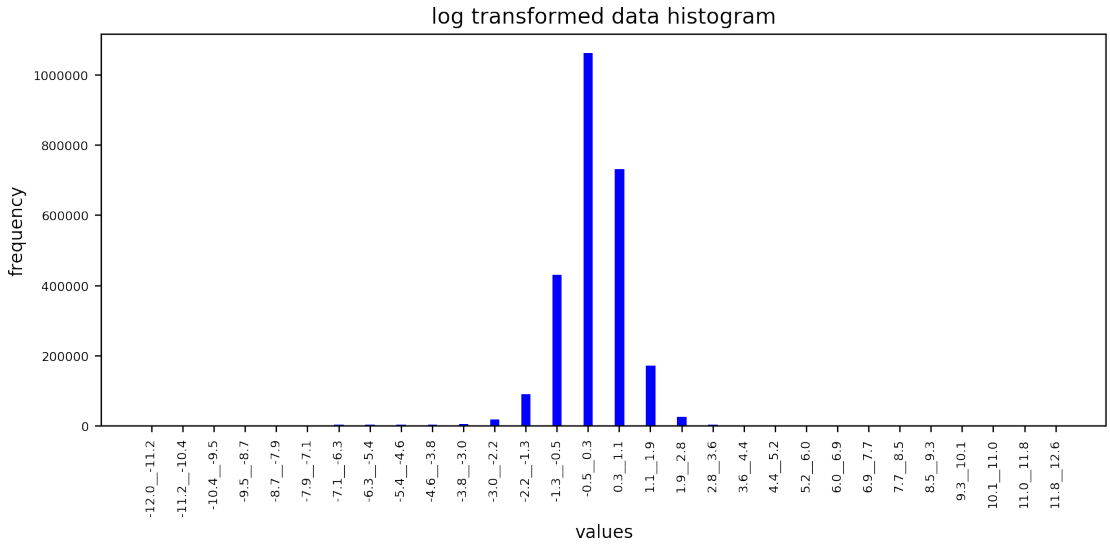
[8]:
```
data_log = np.log10(1+data)

num_bins = 30
y_values, bin_edges=np.histogram(data_log.to_numpy(),bins=num_bins)
bin_centers = 0.5*(bin_edges[1:] + bin_edges[:-1])

plt.figure(figsize=(10, 4),dpi=200)
plt.rcParams.update({'font.size': 7})
plt.xlabel('values', fontsize=10)
plt.ylabel('frequency', fontsize=10)
plt.title('log transformed data histogram', fontsize=12)
plt.bar(np.arange(0,num_bins), y_values, width = 0.3, color='blue')
plt.xticks(np.arange(0,num_bins),["{:.1f}".format(bin_edges[i])+"__"+"{:.1f}".
 ↪format(bin_edges[i+1]) for i in np.
 ↪arange(0,len(bin_centers))],rotation='vertical')
plt.show()
```



3.3. normalizing our data (z-score transformation) Z-scoring our data across each row to normalize gene expression over samples.

[9]:
```
data_log_z = data_log.transform(zscore ,axis=1)
```

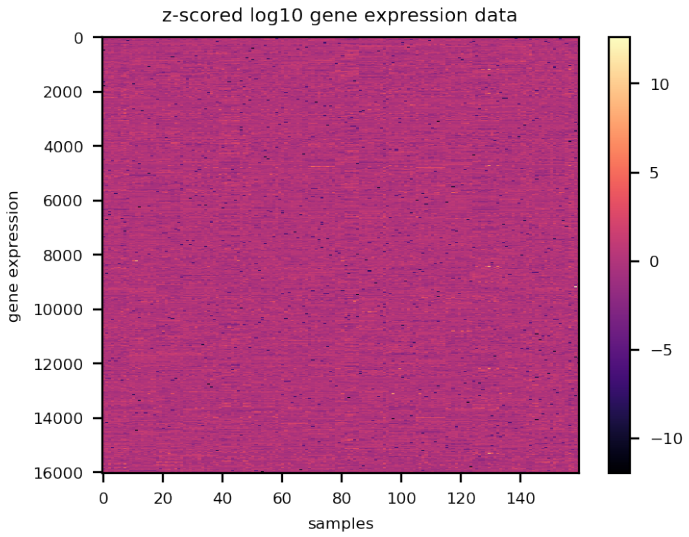
[10]:
```
num_bins = 30
y_values, bin_edges=np.histogram(data_log_z.to_numpy(),bins=num_bins)
bin_centers = 0.5*(bin_edges[1:] + bin_edges[:-1])

plt.figure(figsize=(10, 4),dpi=200)
plt.rcParams.update({'font.size': 7})
plt.xlabel('values', fontsize=10)
plt.ylabel('frequency', fontsize=10)
plt.title('log transformed data histogram', fontsize=12)
plt.bar(np.arange(0,num_bins), y_values, width = 0.3, color='blue')
plt.xticks(np.arange(0,num_bins),["{:.1f}".format(bin_edges[i])+"__"+"{:.1f}".
 ↪format(bin_edges[i+1]) for i in np.
 ↪arange(0,len(bin_centers))],rotation='vertical')
plt.show()
```



Now let's visualize our z-scored log transformed data in a heatmap.

[11]:
```
plt.figure(figsize=(4, 3),dpi=200)
plt.rcParams.update({'font.size': 6})
plt.xlabel('samples')
plt.ylabel('gene expression')
plt.title('z-scored log10 gene expression data')
plt.imshow(data_log_z,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[11]:
```
<matplotlib.colorbar.Colorbar at 0x1a1b095eb8>
```



As you can see in the above heatmap we have a lot of data points and most of them are between -2.5 and 2.5. Makes it difficult to see any patterns in the dataset. Next I will drastically reduce our data just for teaching purposes and making it easy to see patterns.

3.4. reducing data for teaching purposes Usually the sort of analysis that we are doing here are done on the whole dataset but for teaching purposes let's reduce our data size.

We will reduce our genes to the ones with high variance across the samples. Only keeping genes that after log transform they have standard deviation above 0.6. Note that we will use the log transformed data to filter our data.

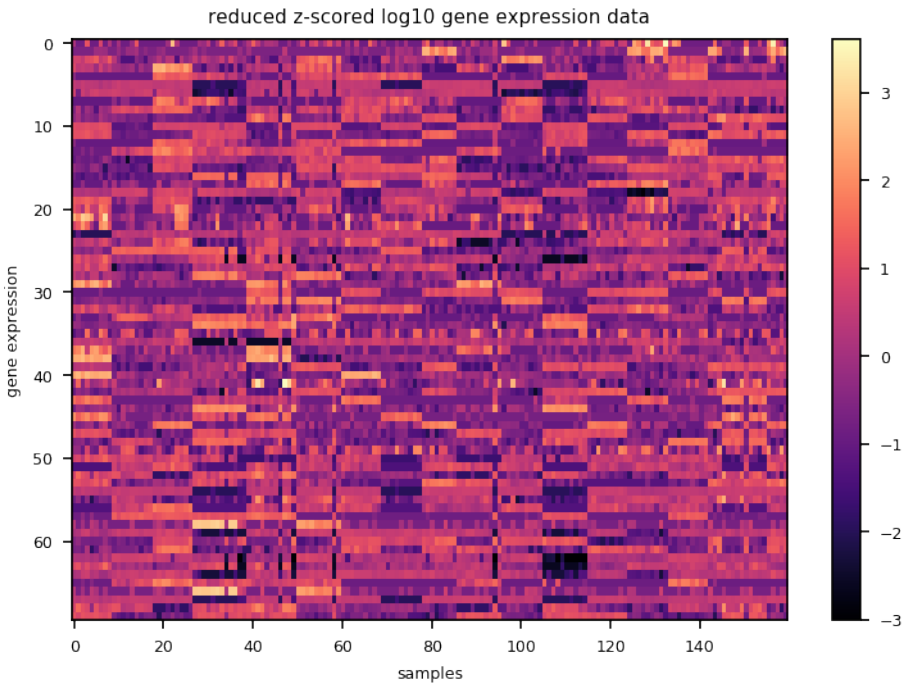
[12]:
```
reduced_data_z = data_log_z[data_log.std(axis=1)>0.6]
print('number of genes: ',reduced_data_z.shape[0], '\nnumber of samples: ',␣
 ↪reduced_data_z.shape[1])
```

```
number of genes:  70
number of samples:  160
```

3.5. visualizing our reduced dataset

[13]:
```
plt.figure(figsize=(6, 4),dpi=150)
plt.rcParams.update({'font.size': 6})
plt.xlabel('samples')
plt.ylabel('gene expression')
plt.title('reduced z-scored log10 gene expression data')
plt.imshow(reduced_data_z,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[13]:
```
<matplotlib.colorbar.Colorbar at 0x1a1b5540b8>
```



Now we can see patterns in our data. For each gene expression (at each row) we have samples with high or low values.

1.4 4. Performing SVD, recunstructing our gene expression dataset using eigen-vectors

4.1. Performing SVD and printing size of each array Every matrix can be decomposed using E = UDVT. E here is our reduced, z-scored and log transformed gene expression data with 70 rows (for 70 genes) and 160 columns (for 160 samples).

[14]:
```
U, D, VT = svd(reduced_data_z, full_matrices=False)
U.shape, D.shape, VT.shape
```

[14]:
```
((70, 70), (70,), (70, 160))
```

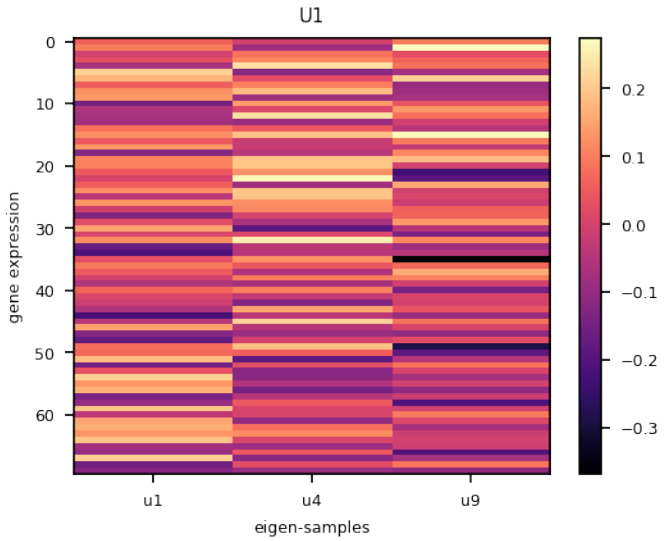
The columns of U are the eigen-samples. Each eigen-sample has an entry for each of the genes. The columns of V are the eigen-genes each eigen-gene has values for each of the samples. Note that in the code we have VT, so each row is one eigen-gene.

4.2. Visualizing eigen-samples and eigen-genes Let's visualize the 1st, 4th and 9th eigen-samples with a heatmap.

[15]:
```
plt.figure(figsize=(4, 3),dpi=150)
plt.rcParams.update({'font.size': 6})
plt.xlabel('eigen-samples')
```

```
plt.ylabel('gene expression')
plt.title('U1')
plt.xticks([0,1,2],['u1', 'u4', 'u9'])
plt.imshow(U[:,[0,3,8]],cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

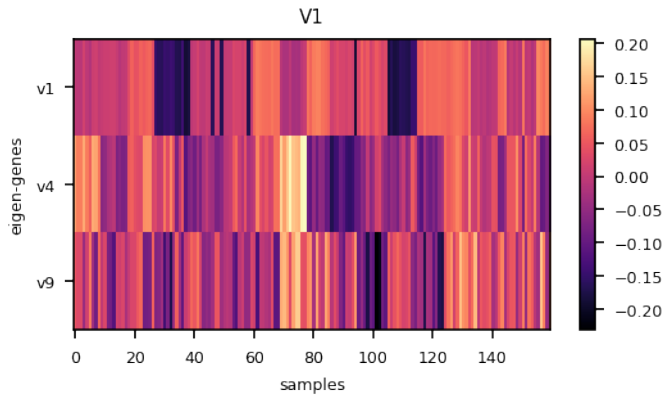
[15]: <matplotlib.colorbar.Colorbar at 0x1a1ab46be0>



Let's visualize the 1st, 4th and 9th eigen-genes with a heatmap.

[16]:
```
plt.figure(figsize=(4, 2),dpi=150)
plt.rcParams.update({'font.size': 6})
plt.xlabel('samples')
plt.ylabel('eigen-genes')
plt.title('V1')
plt.yticks([0,1,2],['v1', 'v4', 'v9'])
plt.imshow(VT[[0,3,8],:],cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[16]: <matplotlib.colorbar.Colorbar at 0x108de8c18>



4.3. Reconstructing the original dataset using the eigen-vectors from SVD Let's reconstruct our data using the eigen-vectors from SVD. Here I am using 40 eigenvectors and producing E[i] where each E[i] is the reconstructed pattern (variance) captured by the i'th eigen-vectors of SVD.

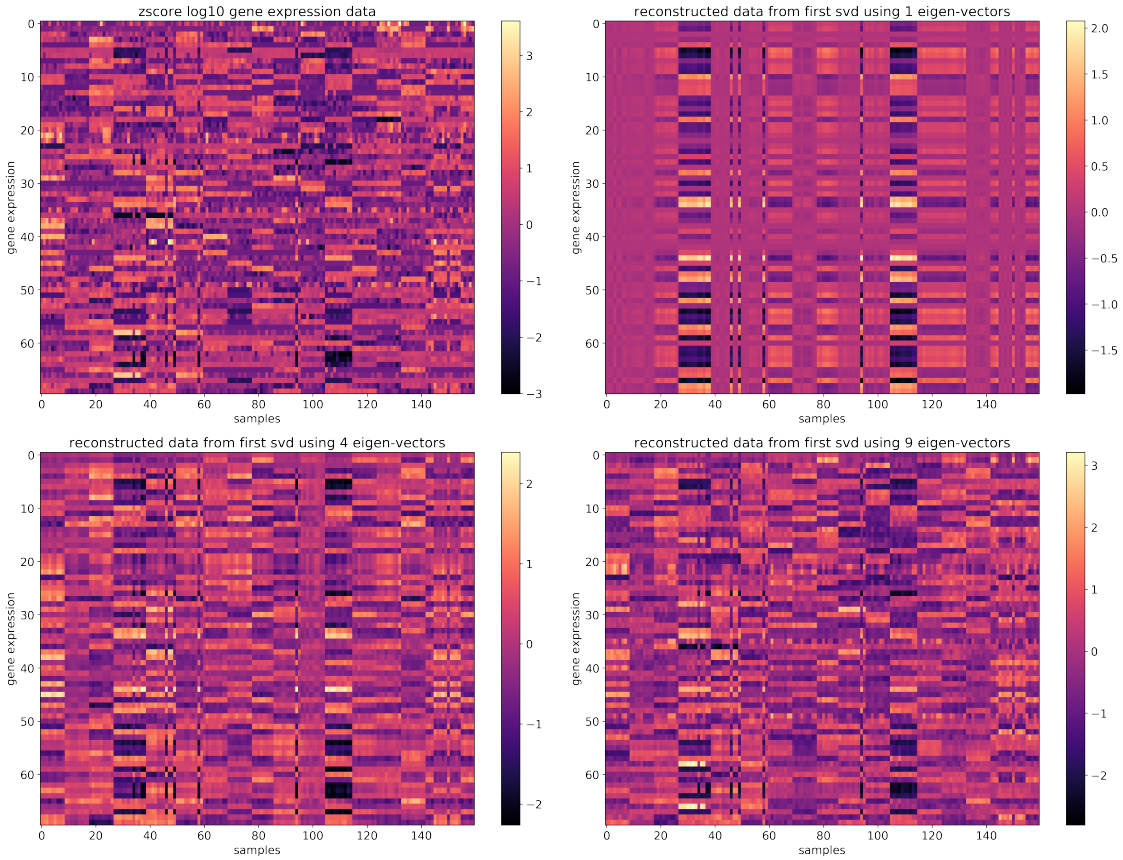
[17]:
```
E = []
numberOfEigenvectors=40
for i in range(0,numberOfEigenvectors):
    E.append(D[i]*U[:,i:i+1].dot(VT[i:i+1,:]))

E[0].shape
```

[17]:
```
(70, 160)
```

Now, if we sum over these E[i]s we can see how only few SVD eigen-vectors can capture the pattern in our data.

[18]:
```
fig = plt.figure(figsize = (20,15),dpi=300)
plt.rcParams.update({'font.size': 14})
plt.subplot(221)
plt.xlabel('samples')
plt.ylabel('gene expression')
plt.title('zscore log10 gene expression data')
plt.imshow(reduced_data_z,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()

for i in range(1,4):
    plt.subplot(2,2,(i+1))
    plt.xlabel('samples')
    plt.ylabel('gene expression')
    plt.title('reconstructed data from first svd using '+ str(i*i) +'␣
 ↪eigen-vectors')
```

```
    plt.imshow( np.sum(E[0:i*i],axis=0) ,cmap=plt.cm.get_cmap("magma"),␣
↪interpolation="nearest",aspect='auto')
    plt.colorbar()
plt.tight_layout()
```

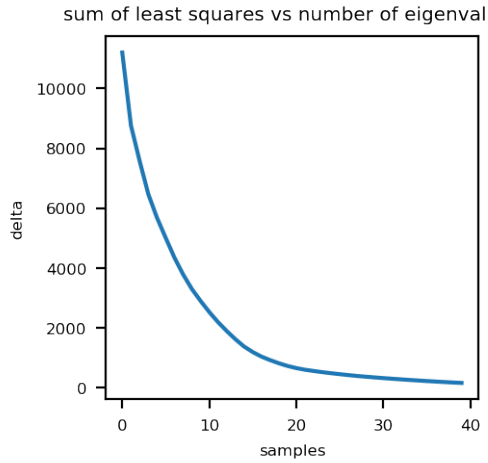


4.4. Computing and plotting the sum of residuals comparing the original dataset with recostructed dataset using SVD eigen-vectors The residuals are the squared of differences between the original dataset and each reconstructed variance using each SVD eigen vector, Delta=|E-E[i]|2. Next, let's see how the sum of residuals changes as we include more eigen-vectors to reconstruct our dataset.

[19]:
```
delta=[]
for i in range(0,numberOfEigenvectors):
    new_data= np.sum(E[0:i],axis=0)
    residuals = abs(reduced_data_z - new_data)
    delta.append((residuals**2).sum().sum())
plt.figure(figsize=(2.5, 2.5),dpi=200)
plt.rcParams.update({'font.size': 6})
plt.xlabel('samples')
plt.ylabel('delta')
```

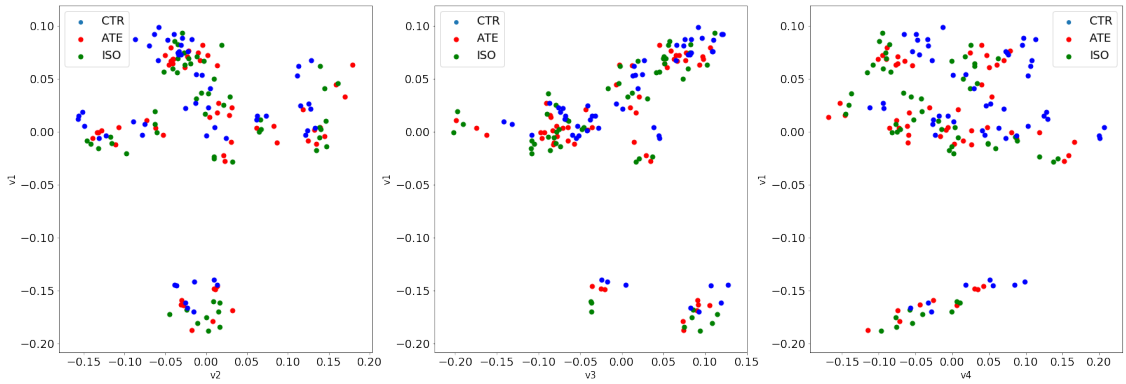
```
plt.title('sum of least squares vs number of eigenvalues')
plt.plot(np.arange(0,len(delta)),delta)
```

[19]: [<matplotlib.lines.Line2D at 0x1174dc940>]



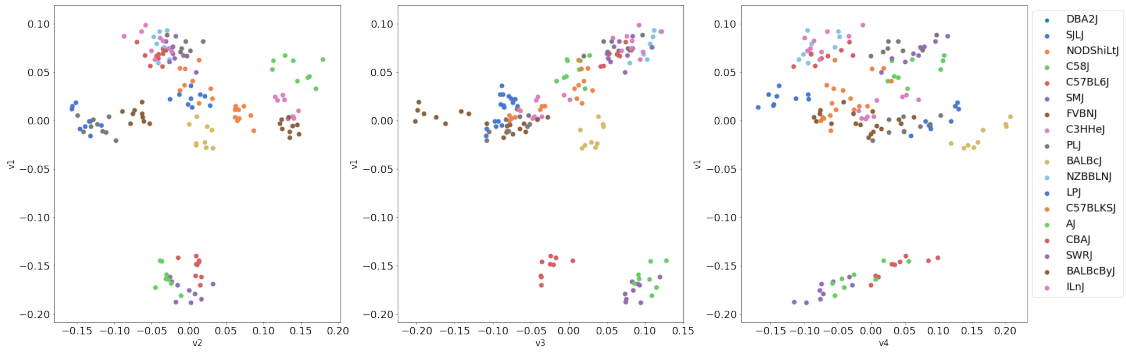
4.5. Comparing PCA results with SVD results let's plot the first SVD eigen-gene against 2nd, 3rd and 4th eigen-genes. Remember each eigen-gene has values for each sample. We will color code our sample points based on their experimental condition, controls, treated by 'ISO' or treated by 'ATE'.

[20]:
```
fig = plt.figure(figsize = (30,10))
plt.rcParams.update({'font.size': 18})

for i in range(1,4):
    ax = fig.add_subplot(1,3,i)
    ax.set_ylabel('v1', fontsize = 15)
    ax.set_xlabel('v'+str(i+1), fontsize = 15)
    ax.scatter(VT[i,:],VT[0,:])

    targets = samples.iloc[2].unique().tolist()
    colors = ['r','g','b']

    for target, color in zip(targets,colors):
        indicesToKeep = (samples.iloc[2] == target)
        ax.scatter(VT[ i,indicesToKeep]
                , VT[0,indicesToKeep]
                , c = color
                , s = 50)
    ax.legend(targets)
```



let's try again plotting the first SVD eigen-gene against 2nd, 3rd and 4th eigen-genes. This time we will color code our sample points based on their experimental strains. 18 different colors for 18 different strains.

[21]:
```
matplotlib_axes_logger.setLevel('ERROR')

fig = plt.figure(figsize = (30,10))
plt.rcParams.update({'font.size': 18})

targets = samples.iloc[1].unique().tolist()
colors=sns.color_palette("muted", len(targets))
for i in range(1,4):
    ax = fig.add_subplot(1,3,i)
    ax.set_ylabel('v1', fontsize = 15)
    ax.set_xlabel('v'+str(i+1), fontsize = 15)
    ax.scatter(VT[i,:],VT[0,:])

    for target, color in zip(targets,colors):
        indicesToKeep = (samples.iloc[1] == target)
        ax.scatter(VT[ i,indicesToKeep]
                , VT[0,indicesToKeep]
                , c=color
                , s = 50)
    if i==3:
        ax.legend(targets,bbox_to_anchor=(1,1))
```



In the above plot we can see the first eigen-gene and second eigen-gene are clustering our genes based on strains.

Now, let's perform pca and plot the first score vector against 2nd, 3rd and 4th score vectors, color coded by strains.

[22]:
```
pca = decomposition.PCA(n_components=10, svd_solver='full')
principalComponents = pca.fit_transform(reduced_data_z)
principalComponents.shape
```

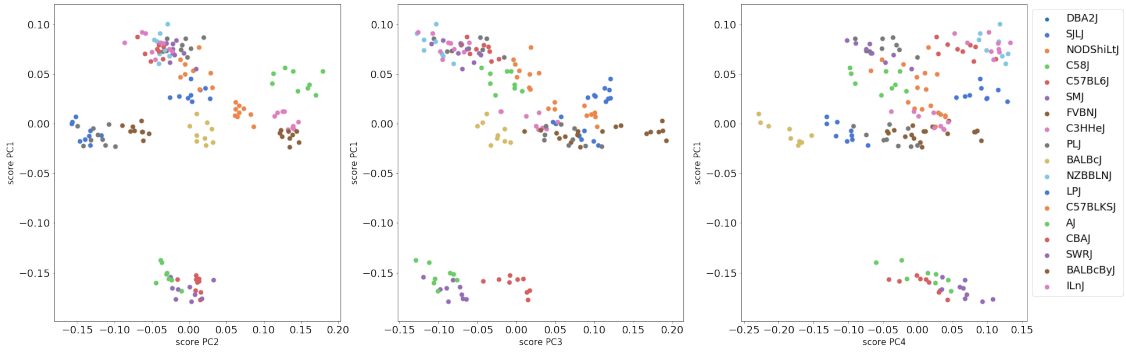
[22]: (70, 10)

[23]:
```
pca_scores = pca.components_.T
pca_scores.shape
```

[23]: (160, 10)

[24]:
```
matplotlib_axes_logger.setLevel('ERROR')

fig = plt.figure(figsize = (30,10))
plt.rcParams.update({'font.size': 18})
targets = samples.iloc[1].unique().tolist()
colors=sns.color_palette("muted", len(targets))
for i in range(1,4):
    ax = fig.add_subplot(1,3,i)
    ax.set_ylabel('score PC1', fontsize = 15)
    ax.set_xlabel('score PC'+str(i+1), fontsize = 15)
    ax.scatter(-1*pca_scores[:,i],-1*pca_scores[:,0])



    for target, color in zip(targets,colors):
        indicesToKeep = (samples.iloc[1] == target)
        ax.scatter(-1*pca_scores[indicesToKeep, i]
               , -1*pca_scores[indicesToKeep, 0]
```

```
                , c=color
                , s = 50)
    if i==3:
        ax.legend(targets,bbox_to_anchor=(1,1))
```



1.5 5. Finding modules using ISA and visualizing them

5.1. Performing ISA Let's save our reduced, z-scored, log transformed dataset and pass it on to ISA tool.

[32]:
```
reduced_data_z.to_csv('reduced_data_z.csv',index=None,header=None)
!python3 './isa-py/isawrp.py' -i reduced_data_z.csv   --gopseudo --seedsparsity␣
↪3 --nt --thc 1.:1.:7. --thr 1.:1.:7. --sgc 0 --sgr 1 --dconv 0.99 --dsame 0.
↪50 --nseed 250
```

```
/
/
/ --- ISA wrapper 2018-06-25/18:00
/
/

/ --- matrix :
      70x160

/ --- row thresholds :
      1.00 2.00 3.00 4.00 5.00 6.00
/
/ --- column thresholds :
      1.00 2.00 3.00 4.00 5.00 6.00
/

/ --- computing robustness floor

++/Users/bkhalili/Documents/PostdocFolders/teaching/mice_rNAseq_data/forStudents
/isa-py/isa.py:79: RuntimeWarning: invalid value encountered in true_divide
```

```
  csBn = csB/numpy.linalg.norm(csB,axis=0)
/Users/bkhalili/Documents/PostdocFolders/teaching/mice_rNAseq_data/forStudents/i
sa-py/isa.py:63: RuntimeWarning: invalid value encountered in greater
  t1 = dsA>sUP
/Users/bkhalili/Documents/PostdocFolders/teaching/mice_rNAseq_data/forStudents/i
sa-py/isa.py:64: RuntimeWarning: invalid value encountered in less
  t2 = dsA<sDW
/Users/bkhalili/Documents/PostdocFolders/teaching/mice_rNAseq_data/forStudents/i
sa-py/isa.py:118: RuntimeWarning: invalid value encountered in greater
  mk = (numpy.abs(rsSR)>1e-12).any(axis=0) \
++++/
++++++/
++++++/
++++++/
++++++/
++++++/

/ --- getting modules

++++++/
++++++/
++++++/
++++++/
++++++/
++++++/

/ --- sweeping

++++++/
++++++/
++++++/
++++++/
++++++/
++++++/
```

ISA tool produces three files in the current folder (the one with the jupyter notebook). 'info.isa.tsv' including the information about our ISA run, 'isa.colscore.tsv' and 'isa.rowscore.tsv' which for each module include the scores generated for all samples and all genes, respectively.

Let's look at 'isa.colscore.tsv' file.

[33]:
```
isa_col_scores = pd.read_csv('isa.colscore.tsv',sep='\t')
isa_col_scores = isa_col_scores.drop(isa_col_scores.columns[0],axis=1)
number_modules = isa_col_scores.shape[1]
print('number of found modules by isa is: '+ str(number_modules ))
isa_col_scores.head(n=3)
```

```
number of found modules by isa is: 29
```

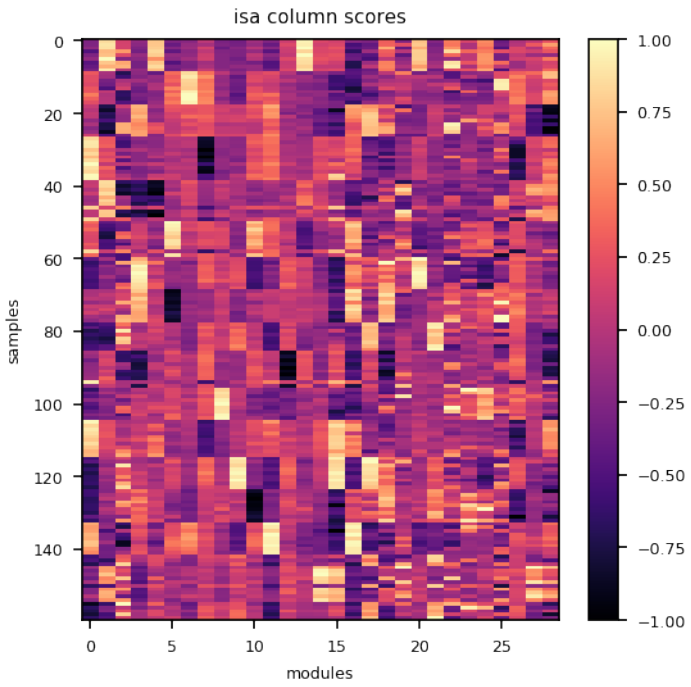
[33]:
```
    isa/M00   isa/M01   isa/M02   isa/M03   isa/M04   isa/M05   isa/M06  \
0 -0.269070  0.856927  0.142705 -0.172297  0.758751 -0.265562 -0.219157
1 -0.260814  0.898041  0.155829 -0.203779  0.773257 -0.376199 -0.154485
2 -0.187517  0.882644  0.148565 -0.273100  0.791366 -0.237581 -0.118882

    isa/M07   isa/M08   isa/M09  …   isa/M19   isa/M20   isa/M21   isa/M22  \
0  0.037732 -0.291817 -0.320688  … -0.268406  0.746158 -0.233368  0.445950
1 -0.064589 -0.259031 -0.311161  …  0.225827  0.774842 -0.316920  0.429409
2  0.136561 -0.361798 -0.226368  … -0.387247  0.660904 -0.245821  0.553987

    isa/M23   isa/M24   isa/M25   isa/M26   isa/M27   isa/M28
0  0.164267  0.519197 -0.368385 -0.044483  0.057393  0.256871
1 -0.044535  0.481740 -0.321127 -0.003575  0.294514  0.598935
2 -0.042335  0.389927 -0.485852  0.179035 -0.012025  0.367893

[3 rows x 29 columns]
```

'isa.colscore.tsv' file has a header with module numbers.

Let's make a heatmap of column scores. Each isa module generates scores for each sample.

[34]:
```
plt.figure(figsize=(4, 4),dpi=150)
plt.rcParams.update({'font.size': 6})
plt.xlabel('modules')
plt.ylabel('samples')
plt.title('isa column scores')
plt.imshow(isa_col_scores,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[34]:
```
<matplotlib.colorbar.Colorbar at 0x1a1b13acf8>
```



similarly, for 'isa.rowscore.tsv' file:

[35]:
```
isa_row_scores = pd.read_csv('isa.rowscore.tsv',sep='\t')
isa_row_scores = isa_row_scores.drop(isa_row_scores.columns[0],axis=1)
number_modules = isa_row_scores.shape[1]
print('number of found modules by isa is: '+ str(number_modules ))
isa_row_scores.head(n=3)
```

```
number of found modules by isa is: 29
```

[35]:

	isa/M00	isa/M01	isa/M02	isa/M03	isa/M04	isa/M05	isa/M06	\
0	-0.308362	-0.136889	0.031288	-0.061170	-0.069546	0.185020	-0.168768	
1	-0.559746	-0.303754	0.304493	-0.023131	0.074195	-0.315241	-0.129199	
2	0.266216	-0.181750	-0.218150	0.061996	0.240491	0.315818	-0.374463	

	isa/M07	isa/M08	isa/M09	…	isa/M19	isa/M20	isa/M21	isa/M22	\
0	-0.044478	0.095591	0.266365	…	-0.107002	-0.092943	0.030978	-0.213808	
1	0.171207	-0.113207	-0.139628	…	-0.130476	-0.234748	1.000000	-0.371539	
2	-0.071871	1.000000	-0.547063	…	-0.050005	-0.144843	-0.020685	0.080769	

	isa/M23	isa/M24	isa/M25	isa/M26	isa/M27	isa/M28
0	1.000000	0.565428	-0.330979	-0.197328	-0.080633	-0.039854
1	0.302083	0.375813	-0.167012	-0.187180	-0.153284	-0.156025
2	0.505843	1.000000	-0.307238	-0.016037	-0.467412	0.021399

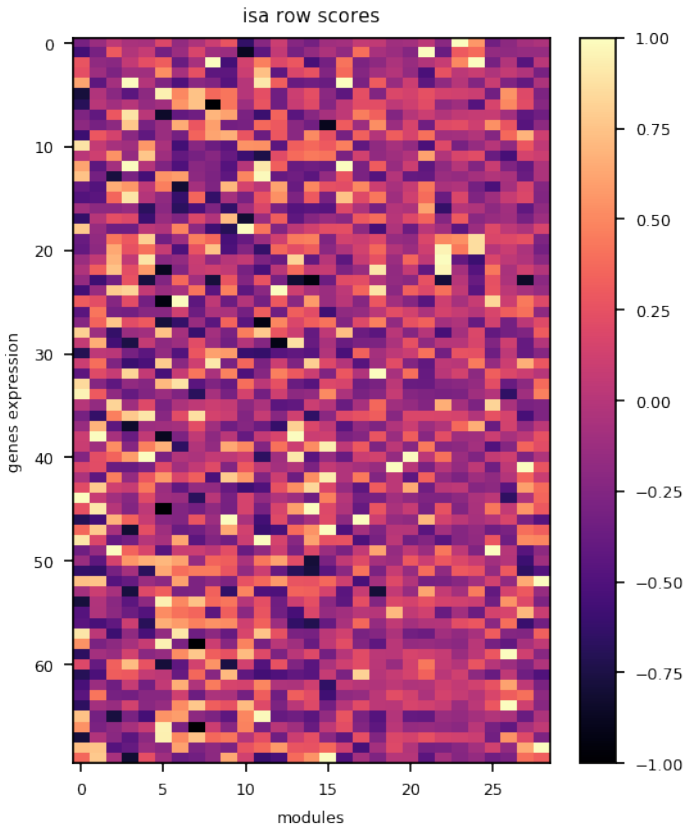
```
[3 rows x 29 columns]
```

Again, in 'isa.rowscore.tsv' file we have the same number of columns as number of modules found by isa. Each column have scores for all the genes in our original dataset.

Let's make a heatmap of row scores. Each isa module generates scores for each gene.

[36]:
```
plt.figure(figsize=(4, 5),dpi=150)
plt.rcParams.update({'font.size': 6})
plt.xlabel('modules')
plt.ylabel('genes expression')
plt.title('isa row scores')
plt.imshow(isa_row_scores,cmap=plt.cm.get_cmap("magma"),␣
↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[36]:
```
<matplotlib.colorbar.Colorbar at 0x1a1ab32a20>
```



5.2. Rearranging genes and samples in original dataset based on ISA module scores Now, we will define a function that sorts both genes and samples in our original data matrix based on ISA module scores. As an input this function needs the dataset and the label number of ISA

module based on which we would like to rearrange our dataset. The output of this function is the rearranged dataser which we will visualize by a heat map in the next cell.

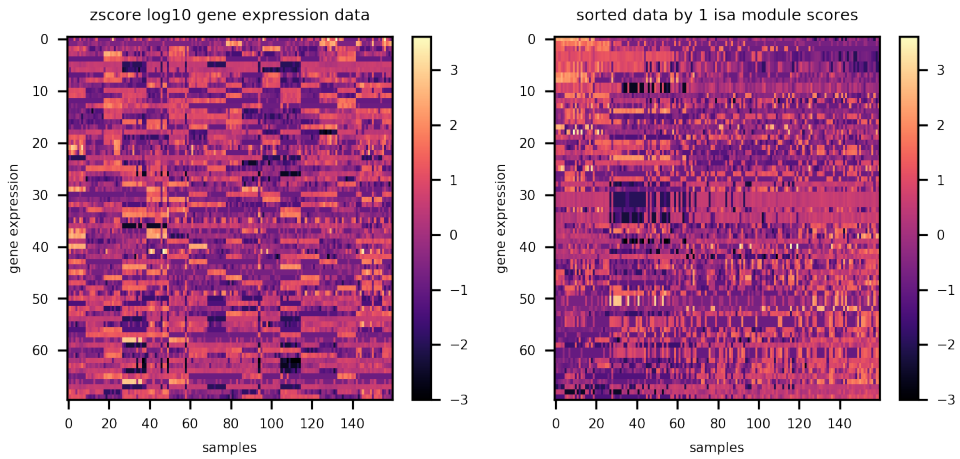
[37]:
```
def sort_data_by_module_score(data,isa_module):
    ModuleBase_sortedData = data.copy()     #copying the original dataset

    #adding a column called 'row_score_module' which contains the gene scores␣
↪from 'isa_module' in 'isa.rowscore.tsv' file
    ModuleBase_sortedData['row_score_module'] = isa_row_scores.loc[:
↪,isa_row_scores.columns[isa_module]].values

    #Then we sort our genes based on scores in 'row_score_module' column
    ModuleBase_sortedData = ModuleBase_sortedData.
↪sort_values(by=['row_score_module'], ascending=False)

    #Then we drop the column 'row_score_module'
    ModuleBase_sortedData = ModuleBase_sortedData.
↪drop(['row_score_module'],axis=1)

    #we transpose the data to have the samples in the rows this time
    ModuleBase_sortedData = ModuleBase_sortedData.transpose()

    #then add a column called 'col_score_module' which contains the sample␣
↪scores from 'isa_module' in 'isa.colscore.tsv' file
    ModuleBase_sortedData['col_score_module'] = isa_col_scores.loc[:
↪,isa_col_scores.columns[isa_module]].values

    #Then we sort our samples based on scores in 'row_score_module' column
    ModuleBase_sortedData = ModuleBase_sortedData.
↪sort_values(by=['col_score_module'], ascending=False)

    #Then we drop the column 'col_score_module'
    ModuleBase_sortedData = ModuleBase_sortedData.
↪drop(['col_score_module'],axis=1)

    #then we transpose our data back to have genes in rows and samples in␣
↪columns
    ModuleBase_sortedData = ModuleBase_sortedData.transpose()
    return(ModuleBase_sortedData)
```

Now, let's rearrange our data based on one of our found modules by ISA and visualize it by a heatmap. You can choose the number of module that you would like to rearange the data for by setting 'isa_module'.

[38]:
```
isa_module = 1
new_ModuleBase_sortedData = sort_data_by_module_score(reduced_data_z,isa_module)
```

```
fig = plt.figure(figsize = (6,2.5),dpi=300)
plt.rcParams.update({'font.size': 5})
plt.subplot(121)
plt.xlabel('samples')
plt.ylabel('gene expression')
plt.title('zscore log10 gene expression data')
plt.imshow(reduced_data_z,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()

plt.subplot(122)
plt.xlabel('samples')
plt.ylabel('gene expression')
plt.title('sorted data by '+str(isa_module) +' isa module scores')
plt.imshow(new_ModuleBase_sortedData,cmap=plt.cm.get_cmap("magma"),␣
 ↪interpolation="nearest",aspect='auto')
plt.colorbar()
```

[38]: <matplotlib.colorbar.Colorbar at 0x1a1c5bec88>



[ ]: