Hwk 6

March 8, 2018

1 Table of Contents

In [1]:
```
import pandas as pd
import numpy as np
import scipy
from scipy.special import gammaln
from Bio import SeqIO

from sklearn.decomposition import NMF
from sklearn.decomposition import PCA

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from matplotlib import rc

rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
```

```
%matplotlib inline

# This enables SVG graphics inline.
# There is a bug, so uncomment if it works.
%config InlineBackend.figure_formats = {'png', 'retina'}

# JB's favorite Seaborn settings for notebooks
rc = {'lines.linewidth': 2,
      'axes.labelsize': 18,
      'axes.titlesize': 18,
      'axes.facecolor': 'DFDFE5'}
sns.set_context('notebook', rc=rc)
sns.set_style("dark")

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['legend.fontsize'] = 14
```

2 Linear Algebra

Let D be the $m \times n$ matrix of counts. Let x be a vector of length n.

$$Dx = y$$

2.1 What are the dimensions of y

y is a vector of length m.

2.2 Given $x_j = \{1/n\} \forall j$, what is y_i in $Dx = y$

If $x_j = \frac{1}{n}$, then $y_i = \frac{1}{n} \sum_j D_{ij}$

2.3 Transpose matrix problem

If we let D^T, find z such that

$$D^T z = y,$$

and y_j is the total number of transcripts of all genes counted in cell j, $\sum_i D_{ji}$

To answer this, I must assume that this problem is asking for a vector, y of dimension $n \times 1$ that is different from the vector y above.

In this case, it follows that z should be a dense vector with every entry equal to unity.

3 Analysis of synthetic data

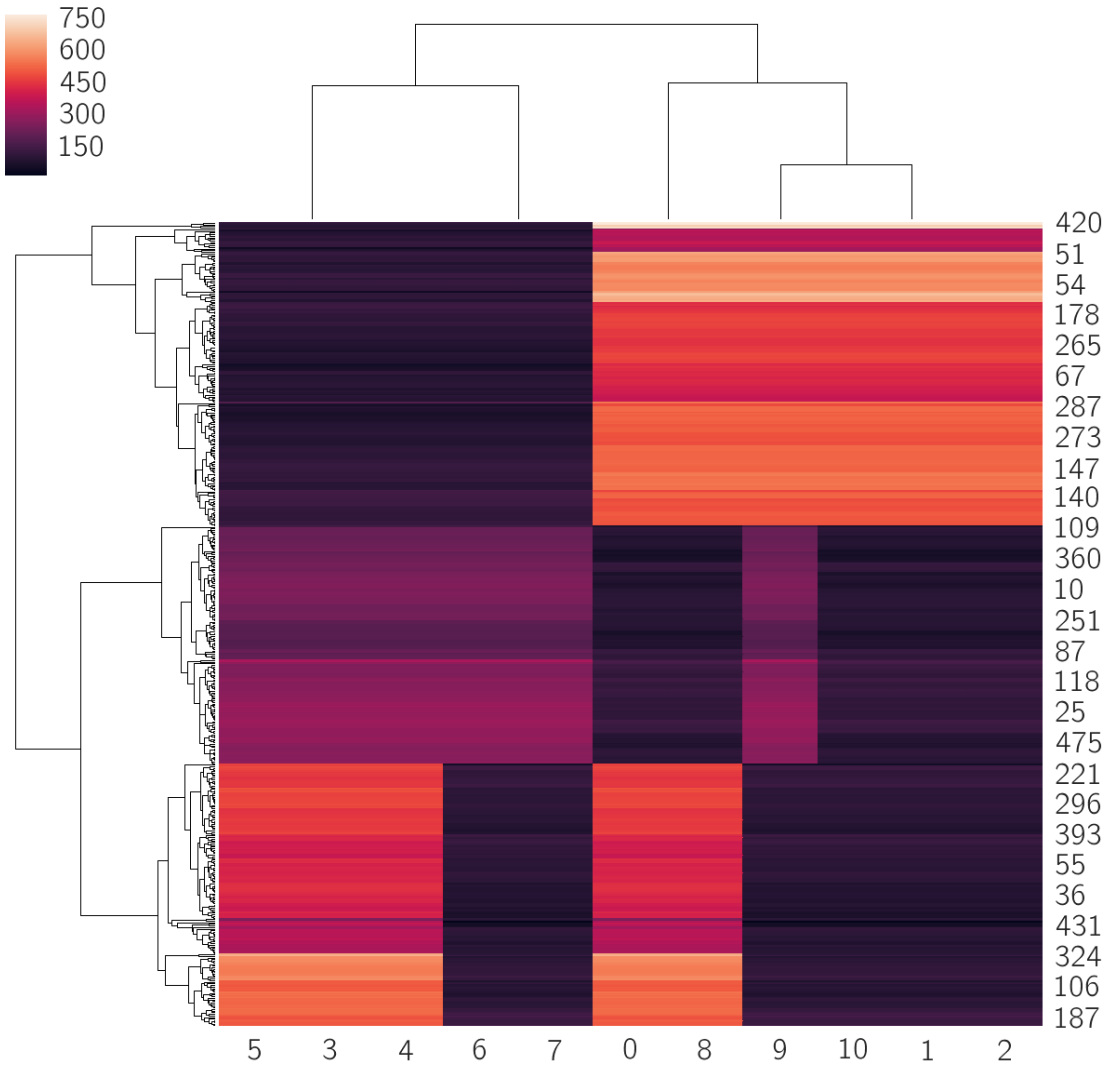
```
In [58]: df = pd.read_csv('../input/pset2syndata.csv', header=None)

In [59]: df.shape
```

```
Out[59]: (11, 500)
```

I will assume this dataset has 11 cells and 500 genes. This would be easier if the dataset were annotated...

```
In [60]: sns.clustermap(df.T)
```

```
Out[60]: <seaborn.matrix.ClusterGrid at 0x10c941160>
```



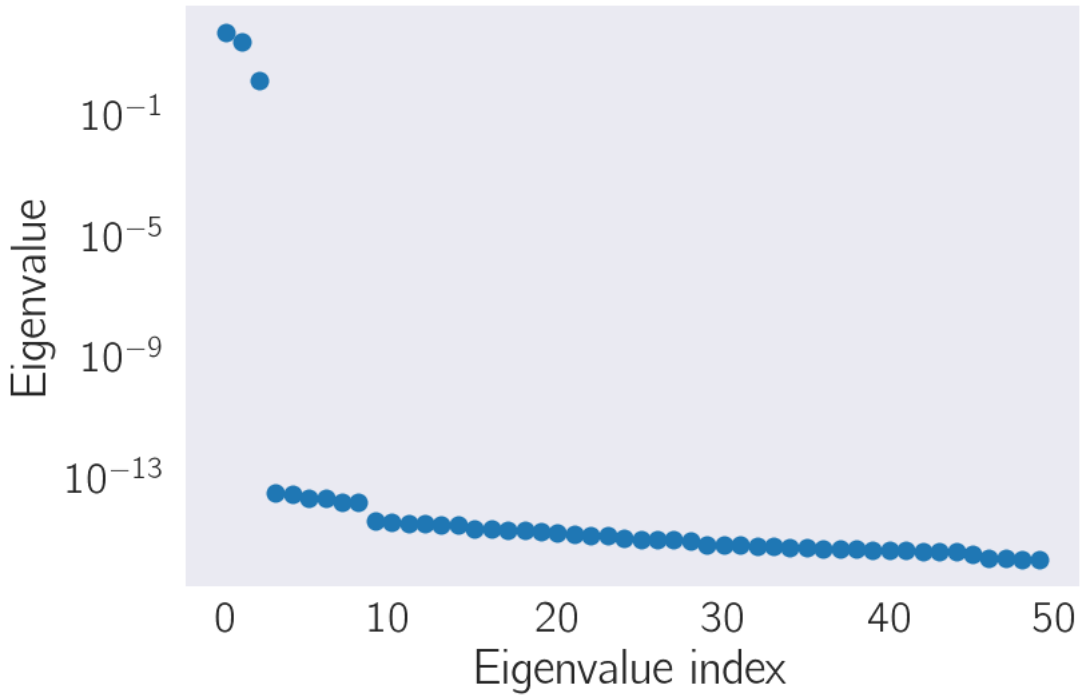

```
In [211]: def find_pcs(dataframe):
              """A function to mean center data and find the PC values"""
              mat = dataframe.as_matrix()
              mat = mat - mat.mean(axis=0)
              cov = mat.T.dot(mat)/(mat.shape[0] - 1)
              w, V = scipy.linalg.eigh(cov)
```

```
            # Make a list of (eigenvalue, eigenvector) tuples
            eig_pairs = [(np.abs(w[i]), V[:,i]) for i in range(len(w))]

            # Sort the (eigenvalue, eigenvector) tuples from high to low
            eig_pairs.sort(key=lambda x: x[0], reverse=True)

            return mat, cov, eig_pairs

        mat, cov, eig_pairs = find_pcs(df)
```

3.1 Plot 3d projection on PCA space and eigenvalue decay

```
In [239]: w = np.array([pair[0] for pair in eig_pairs])
          plt.plot(w[:50]/np.sum(w)*100, 'o')
          plt.yscale('log')
          plt.xlabel('Eigenvalue index')
          plt.ylabel('Eigenvalue')
          print('Explained variance:')
          print(w[:3]/np.sum(w))
```

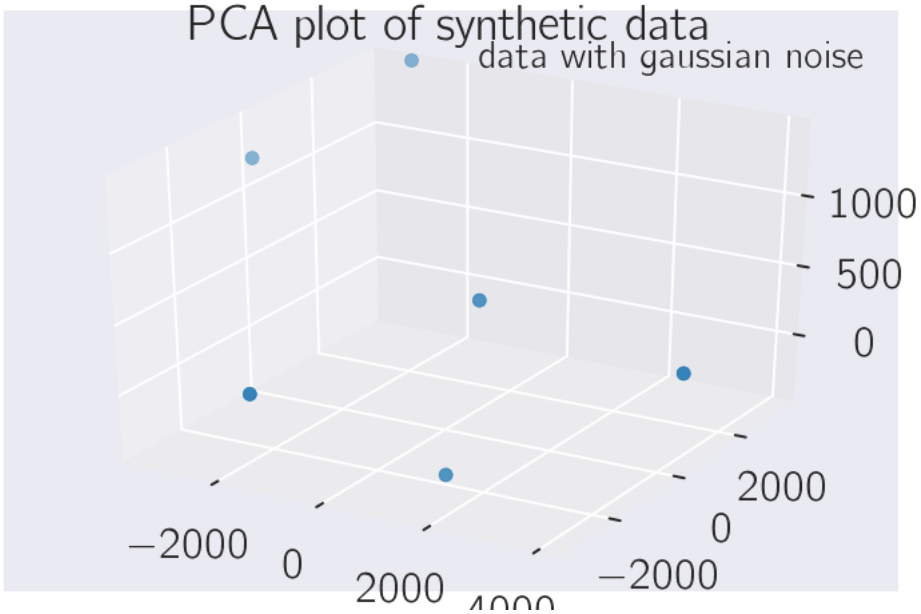
```
Explained variance:
[0.65726455 0.32629524 0.01644021]
```



```
In [241]: # transform coordinates using the greatest 3 components
          # the transformation is simply the projection of each
          # data point into the new basis
          x = mat.dot(eig_pairs[0][1])
          y = mat.dot(eig_pairs[1][1])
          z = mat.dot(eig_pairs[2][1])

In [242]: def gauss(n=len(x)):
              """returns some noise"""
              return np.random.normal(10, 50, n)
```

Let's plot the results: In the plot below, I have multiplied x and z by -1. The reason for this is that the scikit PCA inverts the direction of some principal components, so verifying this worked was difficult.

```
In [248]: fig = plt.figure()
          ax = fig.add_subplot(111, projection='3d')
          ax.scatter(-x, y, -z,
                     s=25, alpha=0.5, label='data with gaussian noise')
          plt.title('PCA plot of synthetic data')
          plt.legend()

Out[248]: <matplotlib.legend.Legend at 0x1133eaf60>
```

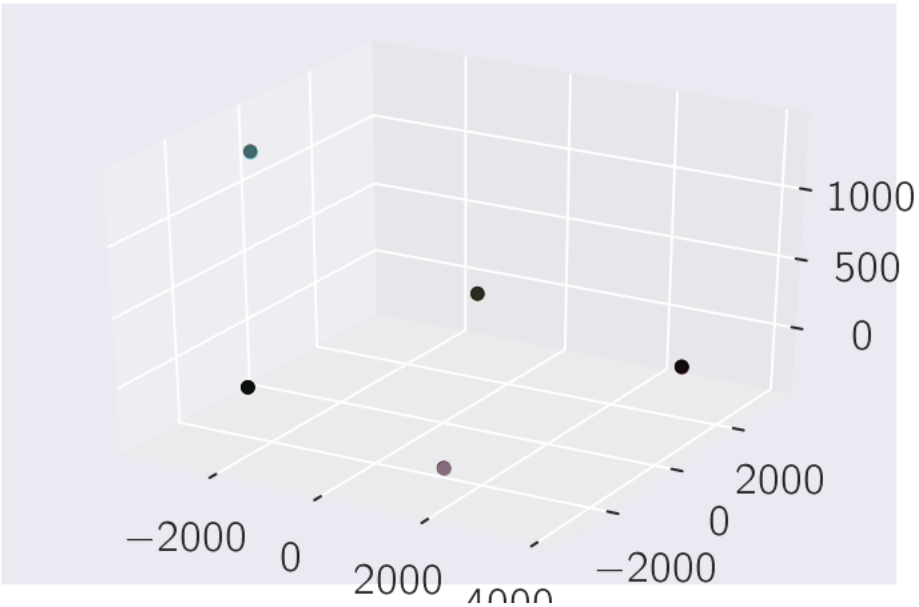


Let's make sure this works using the built-in PCA

```
In [249]: pca = PCA(n_components=3)
          coords = pca.fit_transform(mat)
```

```
V1 = pca.components_
print(pca.explained_variance_ratio_)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(-x, y, -z, color='black',
           s=25, alpha=0.5, label='data with gaussian noise')

for coord in coords:
    x = coord[0]
    y = coord[1]
    z = coord[2]
    ax.scatter(x, y, z,
           s=25, alpha=0.5, label='data with gaussian noise')
```

```
[0.65726455 0.32629524 0.01644021]
```

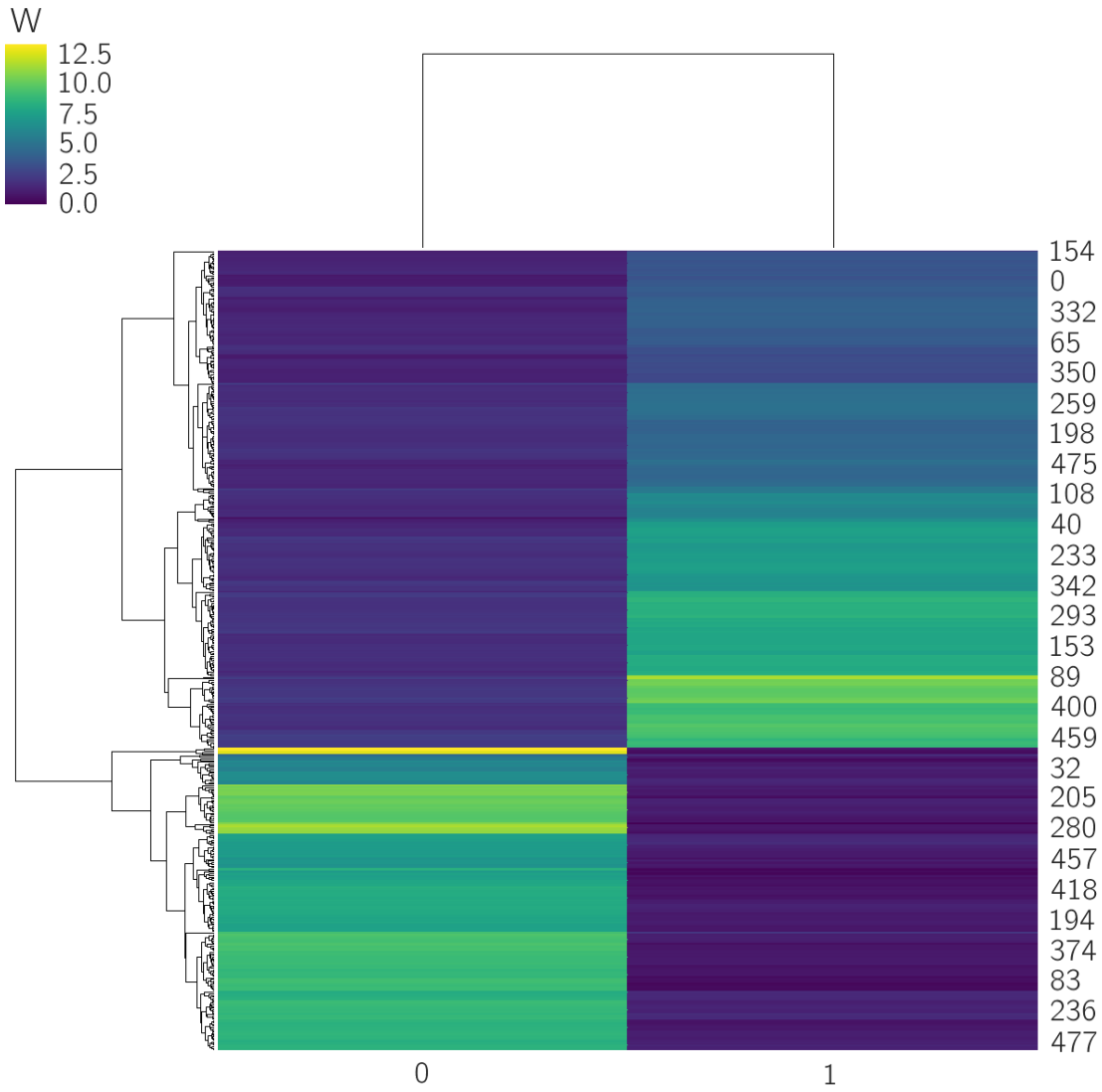


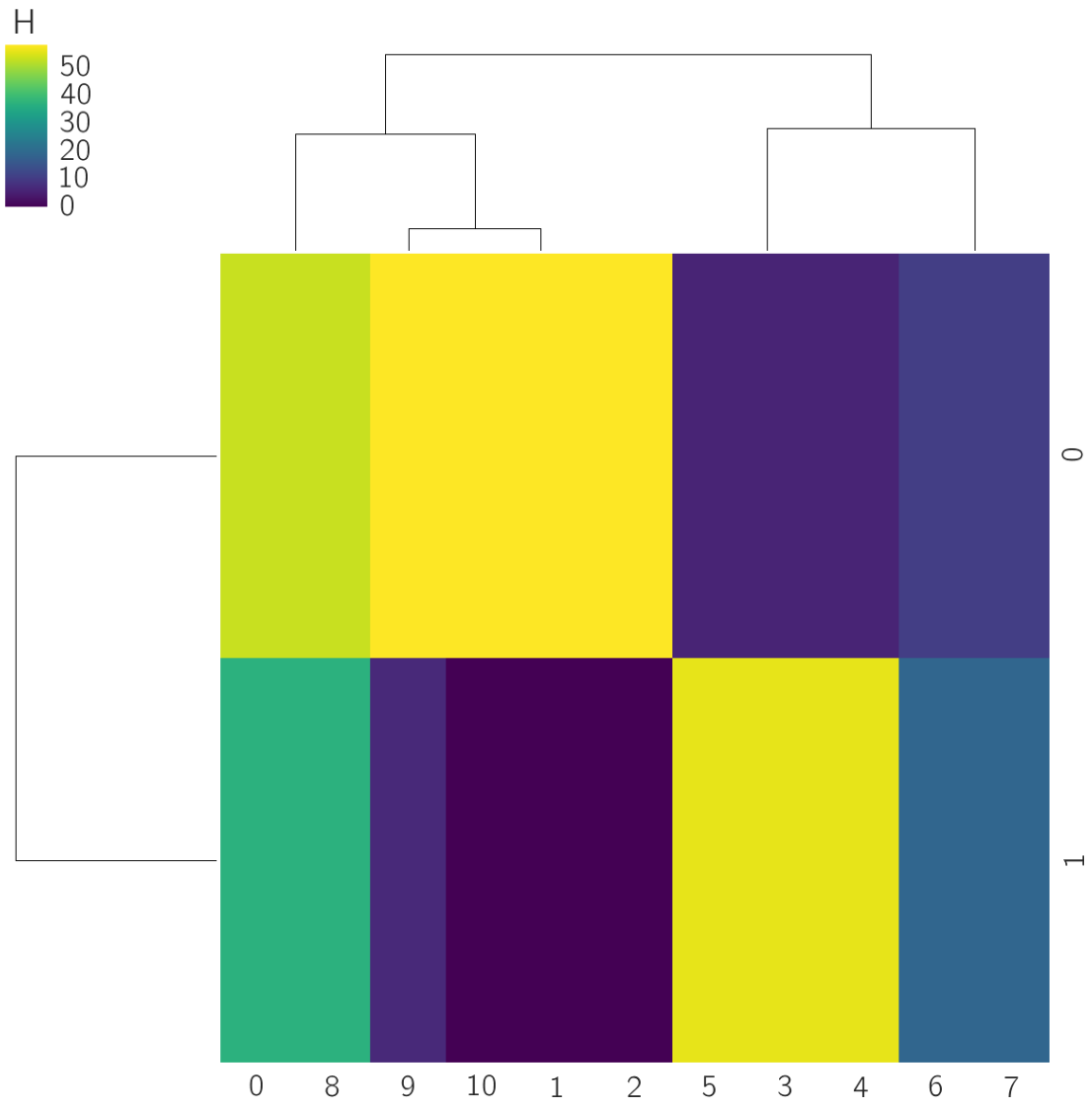
3.2 NMF on the data

```
In [251]: def plot_NMF(dataframe, n, plot=True, normed=False):
              nmf = NMF(n_components=n)

              mat = dataframe.T.as_matrix()

              if normed:
                  mat = mat/mat.std(axis=0)
```

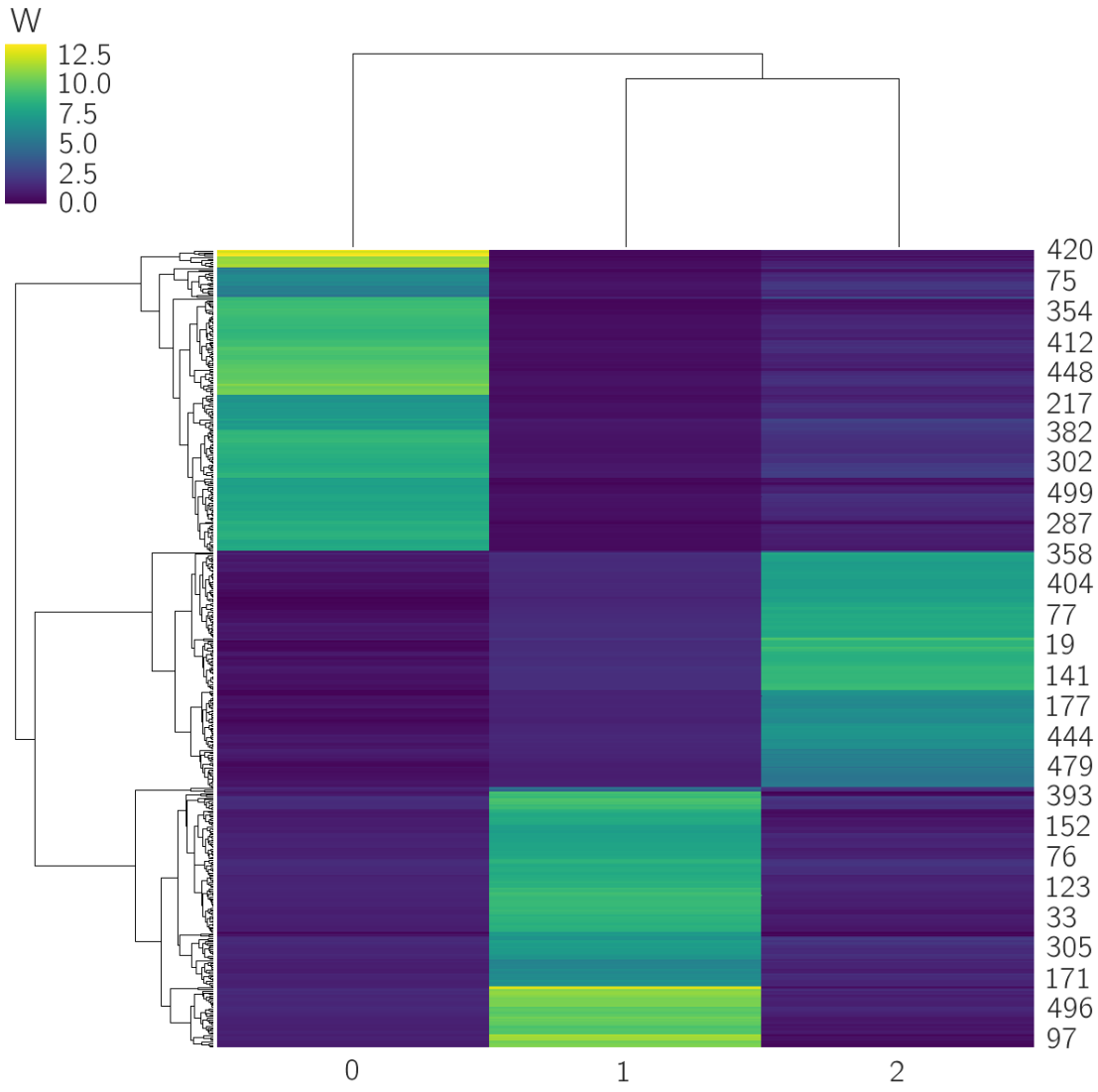
```
    W = nmf.fit_transform(mat);
    H = nmf.components_;

    if plot:
        # fig, ax = plt.subplots(nrows=2, figsize=(12, 12))
        g = sns.clustermap(W, cmap='viridis')
        plt.title('W')
        g2 = sns.clustermap(H, cmap='viridis')
        plt.title('H')
    return H, W
```

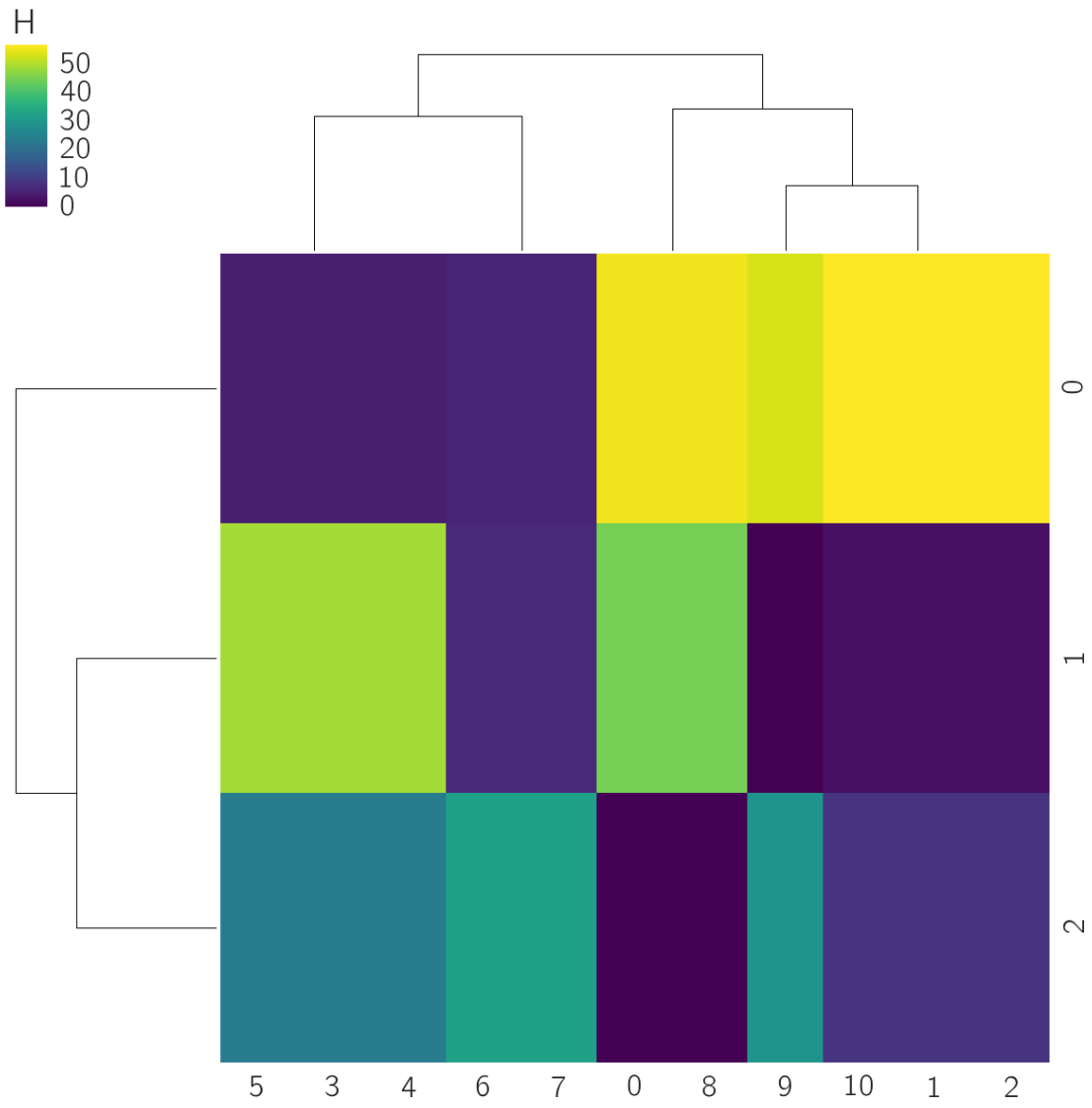
In [70]:
```
H, W = plot_NMF(df, 2)
```






```
In [71]: H, W = plot_NMF(df, 3)
```


W
12.5
10.0
7.5
5.0
2.5
0.0
420
75
354
412
448
217
382
302
499
287
358
404
77
19
141
177
444
479
393
152
76
123
33
305
171
496
97
0
1
2



We would like to know whether the NMF vector and the PCA vector are giving similar answers. One way of assessing how similar the two vectors are is to compute the normalized dot product between them. If the dot product is equal to unity, the vectors are pointing along the same line direction. If the product is 0, then they are orthogonal. If the product is -1, the vectors are antiparallel.

```
In [72]: norm_factor = (np.sqrt(np.sum(W[:, 0]**2))*np.sqrt(np.sum(V[:, 499]**2)))
         dotproduct = W[:, 0].dot(V[:, 499])/norm_factor
         print('The normalized dot product of the first NMF component with the first PCA compo
```

```
The normalized dot product of the first NMF component with the first PCA component is 0.8
```

4 Healthy Data

```
In [266]: data = pd.read_csv('../input/healthy1.csv', header=None)
          data.shape
```

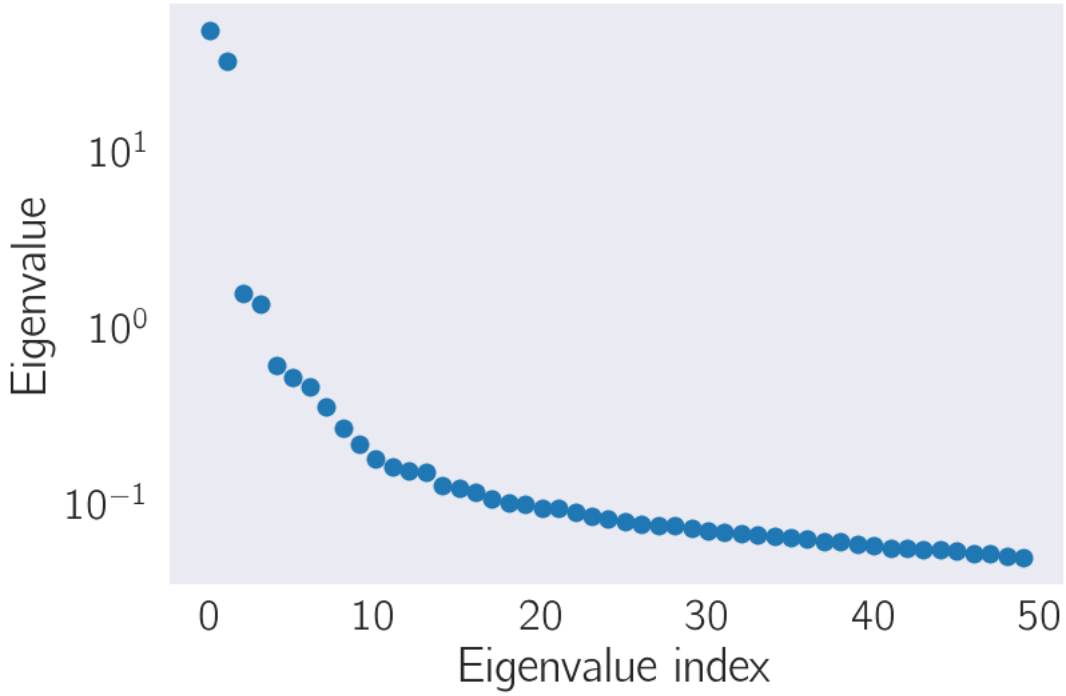
```
Out[266]: (6232, 1985)
```

4.1 PCA on the data

```
In [258]: mat, cov, eig_pairs = find_pcs(data)
```

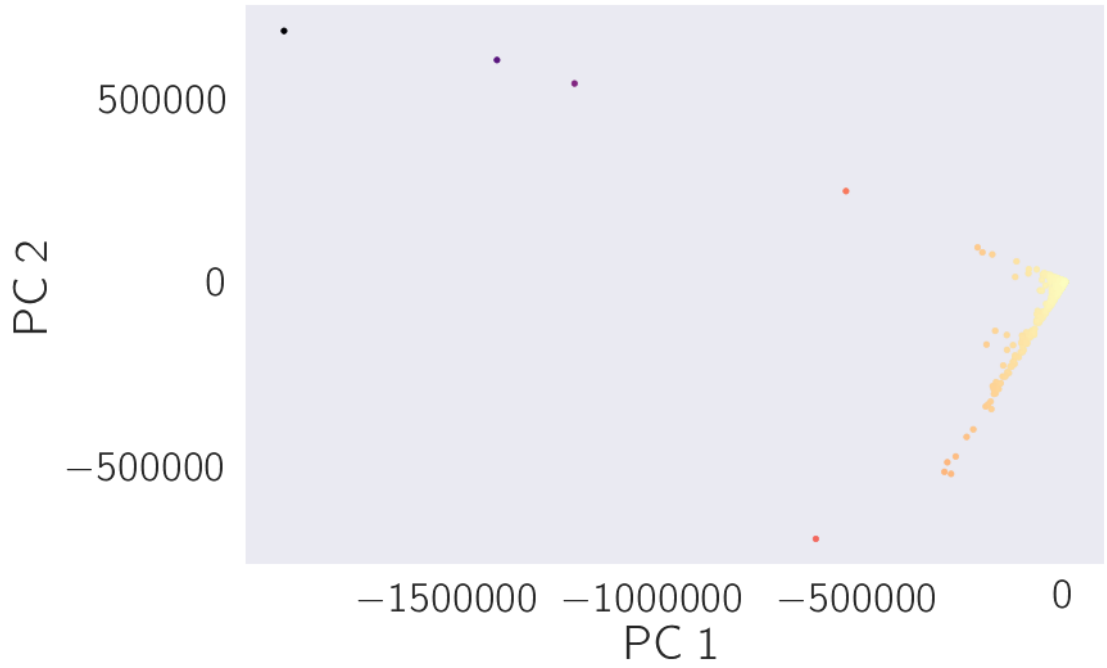
```
In [260]: w = np.array([pair[0] for pair in eig_pairs])
          plt.plot(w[:50]/np.sum(w)*100, 'o')
          plt.yscale('log')
          plt.xlabel('Eigenvalue index')
          plt.ylabel('Eigenvalue')
          print('Explained variance:')
          print(w[:3]/np.sum(w))
```

```
Explained variance:
[0.50727587 0.33856344 0.01641979]
```

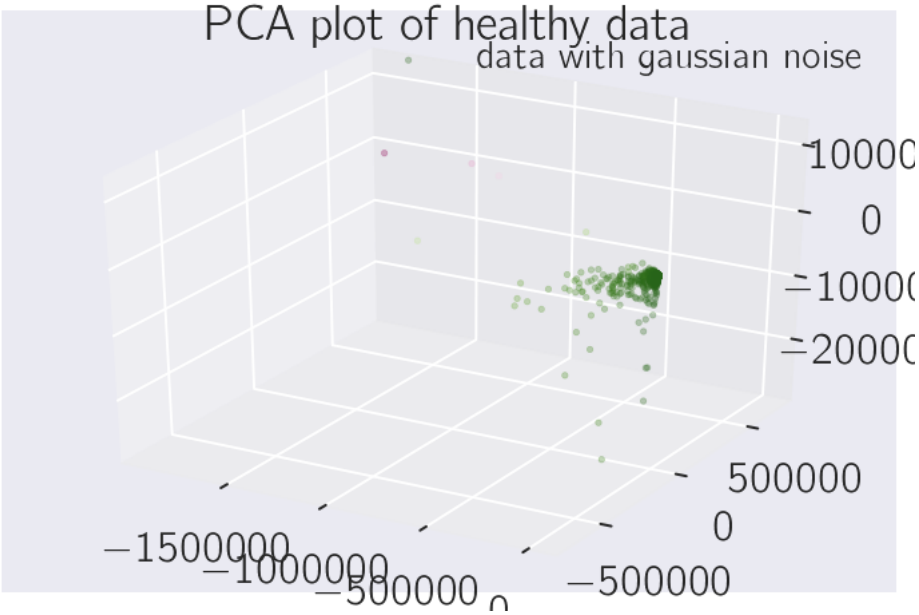



```
In [272]: max_x = np.abs(eig_pairs[0][1]).max()
```

```
In [280]: x = mat.dot(eig_pairs[0][1])
          y = mat.dot(eig_pairs[1][1])
          z = mat.dot(eig_pairs[2][1])

          plt.scatter(x, y, s=5, c=x/max_x, cmap='magma')
          plt.xlabel('PC 1')
          plt.ylabel('PC 2')

Out[280]: Text(0,0.5,'PC 2')
```




```
In [276]: fig = plt.figure()
          ax = fig.add_subplot(111, projection='3d')
          ax.scatter(x, y, z, c=x/max_x, cmap='PiYG',
                     s=5, alpha=0.3, label='data with gaussian noise')
          plt.title('PCA plot of healthy data')
          plt.legend()

Out[276]: <matplotlib.legend.Legend at 0x1115e1128>
```



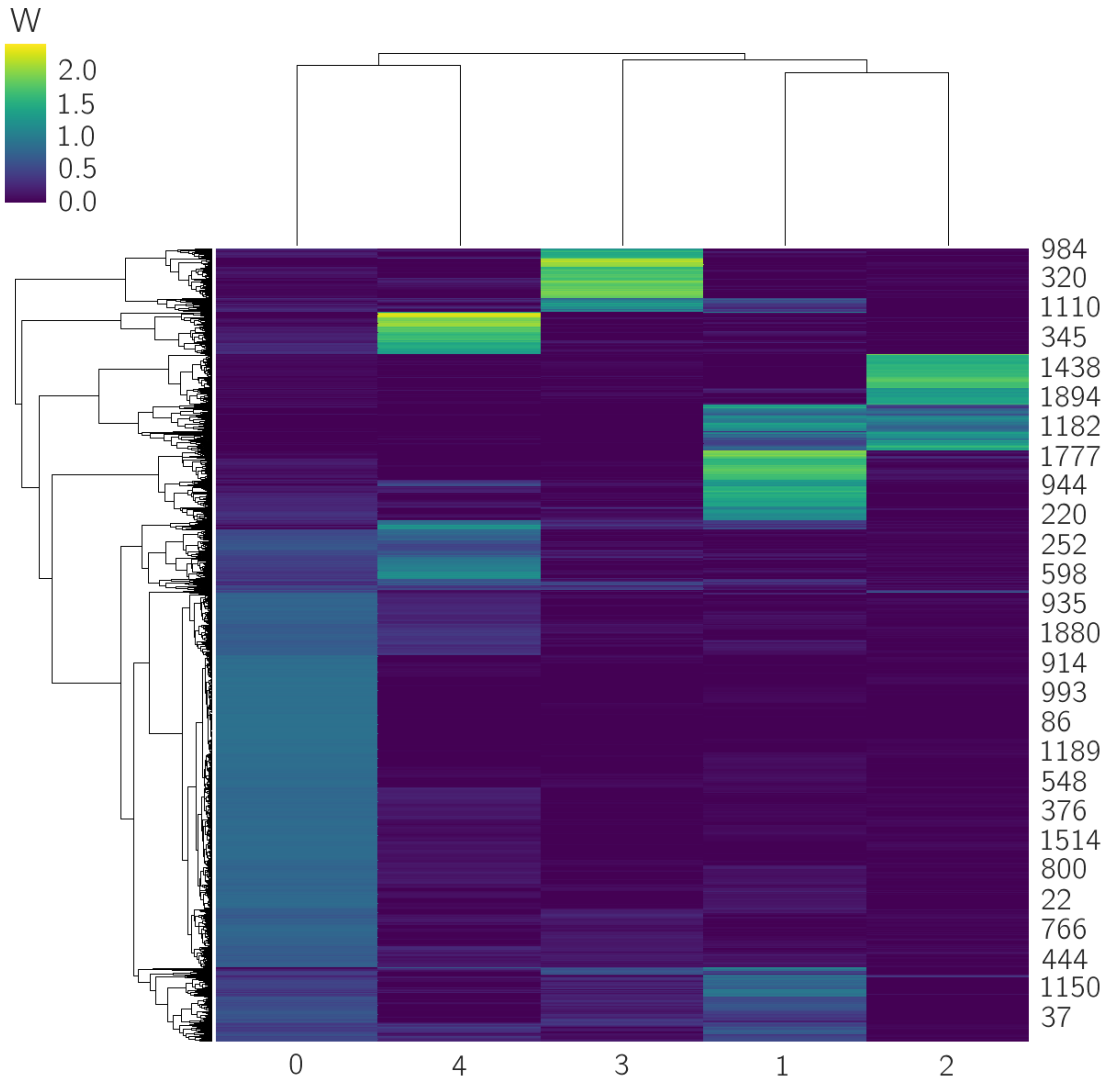
4.2 Study the genes

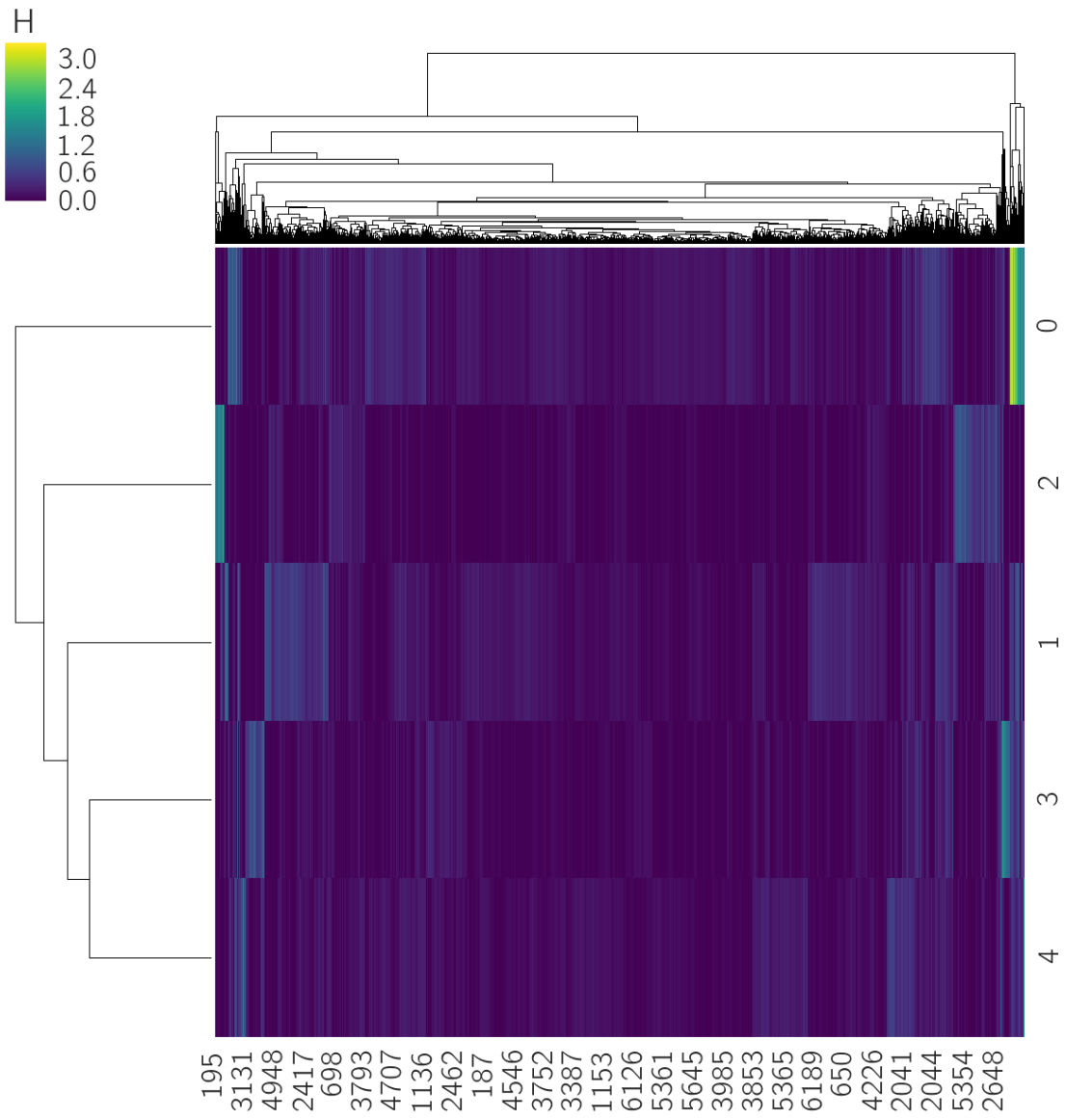
Study the genes in the first and second eigenvectors, and consider both the positive and negative entries. How would you select genes in these principle components to focus on? What is the mean and std of the coefficients in this vector? Pick the top positive and negative entries in components 1-4 and comment on their names–just take the top 2 genes in the + and - for these components. Can you say anything about the point clouds in the data based upon these genes?

This seems strange to me, because negative and positive does not mean anything inherently, since the PCs can be multiplied by -1 without changing the result. However, you could easily select those genes where the coefficients are largest and say that these PC's are driven by those genes.

4.3 NMF on the genes

```
In [269]: H, W = plot_NMF(data, 5, normed=True)
```


W
2.0
1.5
1.0
0.5
0.0
984
320
1110
345
1438
1894
1182
1777
944
220
252
598
935
1880
914
993
86
1189
548
376
1514
800
22
766
444
1150
37
0
4
3
1
2


H
3.0
2.4
1.8
1.2
0.6
0.0
0
2
1
3
4
195
3131
4948
2417
698
3793
4707
1136
2462
187
4546
3752
3387
1153
6126
5361
5645
3985
3853
5365
6189
650
4226
2041
2044
5354
2648

In [ ]: