Dimensionality Reduction_solutions

May 5, 2015

1 Dimensionality Reduction: PCA, MDS

```
In [1]: # import some needed libraries
        import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        np.random.seed(0)
        from pylab import *
        rcParams['figure.figsize'] = (7, 7)
        font = {'weight' : 'normal',
                'size'   : 14}
        matplotlib.rc('font', **font)
```

```
In [2]: # this command makes sure we can plot inline in the notebook
        %matplotlib inline
```

1.1 Principal Component Analysis (PCA)

1.1.1 PCA for heart disease classification

1.1.2 The data

```
In [3]: # load the data from a text file
        heart = pd.read_table('heart.txt', header=0)
```

Explore the data

Description of the data

0. age: age in years

- sex: sex
  - 1 = male
  - 0 = female
- cp: chest pain type
  - 1 = typical angina
  - 2 = atypical angina
  - 3 = non-anginal pain
  - 4 = asymptomatic
- trestbps: resting blood pressure in mm Hg
- chol: serum cholestoral in mg/dl
- fbs: fasting blood sugar > 120 mg/dl

  - 1 = true
  - 0 = false
- restecg: resting electrocardiographic results
  - 0 = normal
  - 1 = having ST-T wave abnormality
  - 2 = showing probable or definite left ventricular hypertrophy
- thalach: maximum heart rate achieved
- exang: exercise induced angina
  - 1 = yes
  - 0 = no
- oldpeak: ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment
  - 1 = upsloping
  - 2 = flat
  - 3 = downsloping
- ca: number of major vessels (0-3) colored by flourosopy
- thal: thallium scan – the myocardial perfusion pattern
  - 3 = normal
  - 6 = fixed defect
  - 7 = reversable defect
- target: diagnosis of heart disease
  - 0 = healthy subject
  - 1 = ill subject
- **Exercise 1**: Explore the data: print the number of samples, the number of features. How many samples does the dataset contain? How many features?

  303 subjects and 13 features (last column in our target variable!)

Which variables are categorical? Which ones are continuous?

```
some examples: sex and chest pain are categorical, while age and cholesterol level are continuous.
```

```
In [4]: print heart.shape
```

```
(303, 14)
```

```
In [6]: print heart.head()
```

```
age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  target
0  0    6       0
1  3    3       1
2  2    7       1
3  0    3       0
4  0    3       0
```

- **Exercise 2**: How many healthy subjects are in the dataset? How many ill subjects? (tip: call the *pd.value_counts* function on the *heart.target* column).

  164 healthy subjects and 139 ill subjects

```
In [7]: pd.value_counts(heart.target)
```

```
Out[7]: 0    164
        1    139
        dtype: int64
```

Visualize the data

```
In [8]: # we need some colors for our plots!
        colors = ['red', 'green', 'blue', 'yellow', 'cyan', 'pink', 'orange', 'purple']
        from itertools import cycle
```

Our dataset contains both *continuos* and *categorical* variables.
Because we will visualize them in a different way, let's split the dataset like this:

```
In [9]: # the continuous variables
        continuous = heart[['age','trestbps', 'chol', 'thalach', 'oldpeak']]
        # the categorical variables
        categorical = heart[['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']]
```

```
In [10]: print continuous.head
```

```
<bound method DataFrame.head of      age  trestbps  chol  thalach  oldpeak
0     63       145   233      150      2.3
1     67       160   286      108      1.5
2     67       120   229      129      2.6
3     37       130   250      187      3.5
4     41       130   204      172      1.4
5     56       120   236      178      0.8
6     62       140   268      160      3.6
7     57       120   354      163      0.6
8     63       130   254      147      1.4
9     53       140   203      155      3.1
10    57       140   192      148      0.4
11    56       140   294      153      1.3
12    56       130   256      142      0.6
13    44       120   263      173      0.0
14    52       172   199      162      0.5
15    57       150   168      174      1.6
16    48       110   229      168      1.0
17    54       140   239      160      1.2
18    48       130   275      139      0.2
19    49       130   266      171      0.6
20    64       110   211      144      1.8
21    58       150   283      162      1.0
22    58       120   284      160      1.8
23    58       132   224      173      3.2
24    60       130   206      132      2.4
25    50       120   219      158      1.6
26    58       120   340      172      0.0
27    66       150   226      114      2.6
```

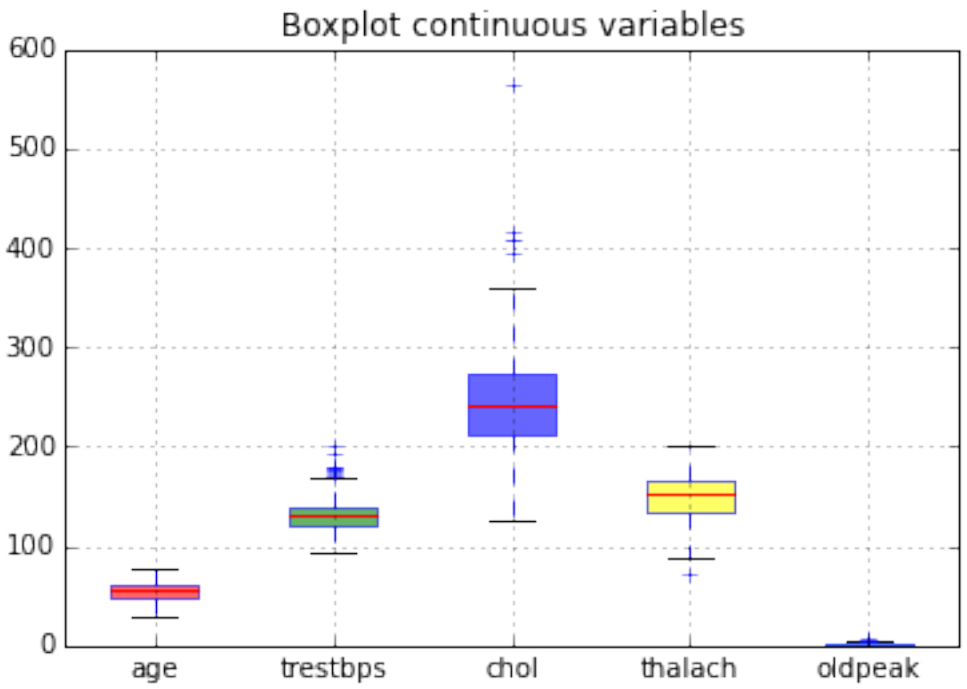
```
28    43       150   247      171      1.5
29    40       110   167      114      2.0
..   ...       ...   ...      ...      ...
273   71       112   149      125      1.6
274   59       134   204      162      0.8
275   64       170   227      155      0.6
276   66       146   278      152      0.0
277   39       138   220      152      0.0
278   57       154   232      164      0.0
279   58       130   197      131      0.6
280   57       110   335      143      3.0
281   47       130   253      179      0.0
282   55       128   205      130      2.0
283   35       122   192      174      0.0
284   61       148   203      161      0.0
285   58       114   318      140      4.4
286   58       170   225      146      2.8
287   58       125   220      144      0.4
288   56       130   221      163      0.0
289   56       120   240      169      0.0
290   67       152   212      150      0.8
291   55       132   342      166      1.2
292   44       120   169      144      2.8
293   63       140   187      144      4.0
294   63       124   197      136      0.0
295   41       120   157      182      0.0
296   59       164   176       90      1.0
297   57       140   241      123      0.2
298   45       110   264      132      1.2
299   68       144   193      141      3.4
300   57       130   131      115      1.2
301   57       130   236      174      0.0
302   38       138   175      173      0.0

[303 rows x 5 columns]>
```

Let's start with a boxplot of our *continuous* variables:

```
In [11]: bp = continuous.boxplot(patch_artist=True)
         for patch, color in zip(bp['boxes'], cycle(colors)):
             patch.set_facecolor(color)
             patch.set_alpha(.6)
         plt.title('Boxplot continuous variables')
```

```
C:\Users\Paola\Anaconda\lib\site-packages\pandas\tools\plotting.py:2633: FutureWarning:
The default value for 'return_type' will change to 'axes' in a future release.
 To use the future behavior now, set return_type='axes'.
 To keep the previous behavior and silence this warning, set return_type='dict'.
  warnings.warn(msg, FutureWarning)
```

```
Out[11]: <matplotlib.text.Text at 0xbcce5c0>
```

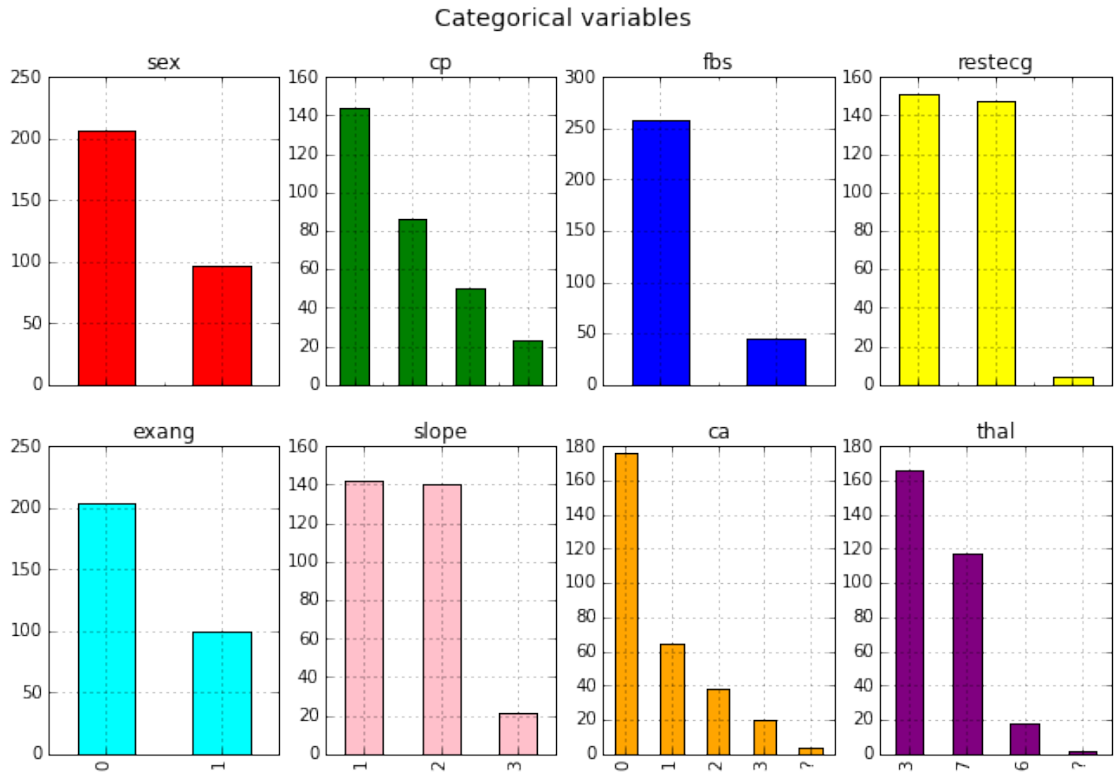


- **Exercise 3**: What can you say about the distribution of these features? Do you think any sort of data pre-processing is required before performing a PCA?

  distributions are very different! PCA needs the data to be standardized!!

And now some barplots for our *categorical* variables:

```
In [12]: fig = plt.figure(figsize=(11,7))
         fig.suptitle('Categorical variables', fontsize=14)
         # loop for the plotting
         for i in range(len((categorical.columns))):
             vec = categorical[categorical.columns[i]]
             ax = plt.subplot(2, 4, i+1)
             vec.value_counts().plot(ax=ax, kind='bar', color=colors[i], title=categorical.columns[i])
```



- **Exercise 4**: Do you see anything strange in the values of the two variables *major vessels* and *thallium scan*? What?

  the ? indeed; it is annotating the missing points in the dataset

Dealing with missing data

```
In [15]: # let's convert the dataframe to a numpy array (just for convenience!)
         heart_data = heart.values
         i = np.where(heart_data=='?')[0]
         print 'The dataset contains %i missing values' %len(i)

         print i
```

```
The dataset contains 6 missing values
[ 87 166 192 266 287 302]
```

```
In [16]: # a list of True and False to flag if a value is missing or not
         missing = np.in1d(range(heart_data.shape[0]), i)

         print missing
         # now we remove the missing values
         heart_data_clean = heart_data[~missing]
```

```
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
```

```
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False  True False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False  True False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
  True False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False False False False False False False False False False False  True
 False False False False False False False False False False False False
 False False  True]
```

- **Exercise 5**: How many samples does the dataset contain after missing data removal?

  297 samples

```
In [17]: print heart_data_clean.shape
```

```
(297L, 14L)
```

```
In [20]: # let's now skip the last column (which is the target, and not a feature)
         X = heart_data_clean[:, 0:heart_data_clean.shape[1]-1]
         X = np.array(X, dtype=float)
         # our y (target) vector is the last column
         y = np.asarray(heart_data_clean[:, heart_data_clean.shape[1]-1], dtype=int)
```

1.1.3 Principal Component Analysis

Let's run a PCA on our data set with the **sklearn** package:

```
In [21]: # import the library
         from sklearn.decomposition import PCA
```

```
In [22]: # create a model and fit and transform the data
         pca = PCA()
         X_pca = pca.fit_transform(X)
```

Inspecting the PCA results Let's have a look at the the proportion of variance explained by each component:

```
In [23]: print pca.explained_variance_ratio_
```

```
[  7.46243578e-01   1.49787648e-01   8.57955289e-02   1.60023313e-02
   1.01307682e-03   3.30072242e-04   2.50801576e-04   2.22000205e-04
   1.76402998e-04   5.92782847e-05   4.75020518e-05   4.06039465e-05
   3.11751680e-05]
```

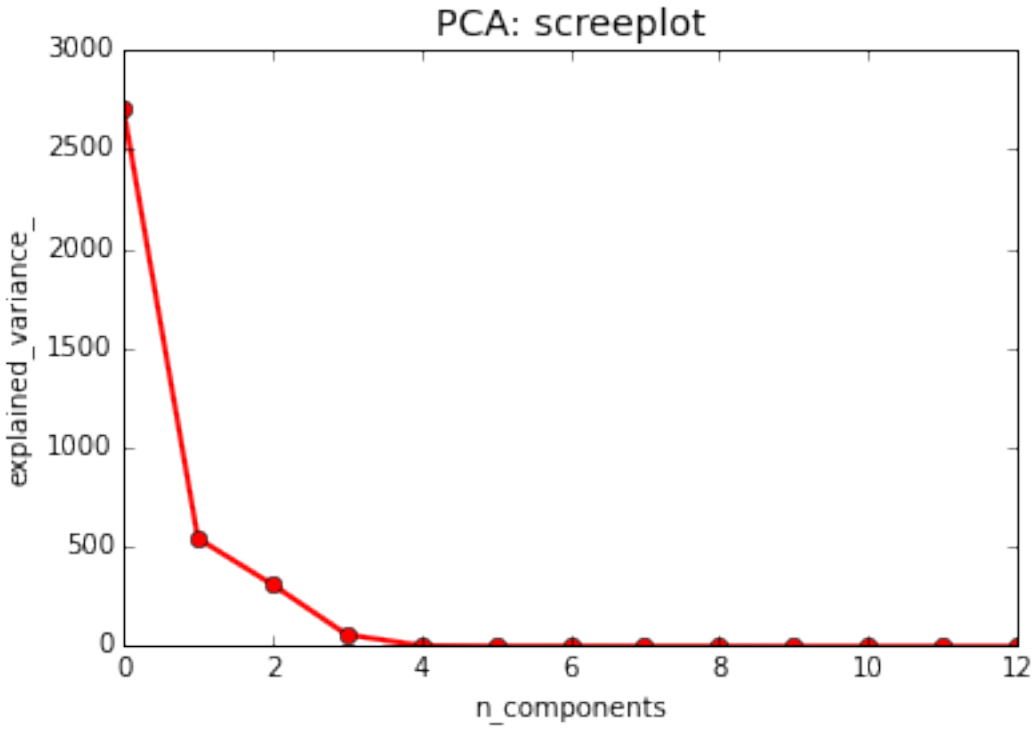
- **Exercise 6**: Which is the percentage of variance explained by the first component? And the one explained by the second?

  74%, 15%

In order to decide how many principal components you should retain, you can summarise the results of the performed PCA by making a **scree plot**. A scree plot shows the fraction of total variance in the data as explained or represented by each PC.

```
In [24]: # render the screeplot
         plt.plot(pca.explained_variance_, 'ro-', linewidth=2)
         plt.xlabel('n_components')
         plt.ylabel('explained_variance_')
         plt.title('PCA: screeplot', fontsize=14)
```

```
Out[24]: <matplotlib.text.Text at 0x18350128>
```



- **Exercise 7**: How many components do you think are needed to explain most of the variance of this dataset?

  the second component does not add a lot of information, and the third even less. It seems here that

Visualize the data projection

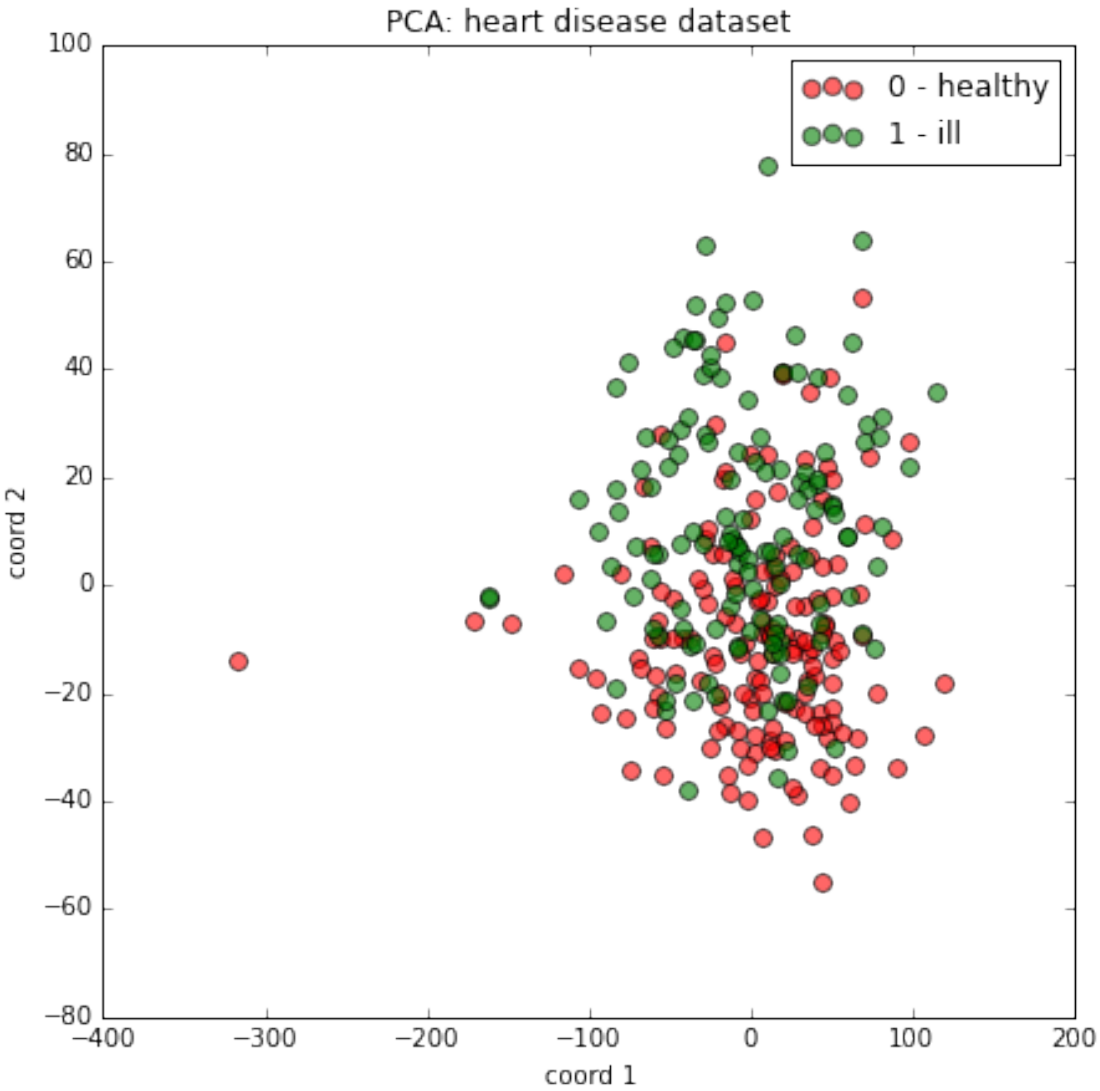
```
In [25]: diagnose = y
         diagnose_labels = ('healthy', 'ill')
```

```
In [26]: exercise = np.asarray(heart_data_clean[:, 8], dtype = int)
         exercise_labels = ('exercise induced angina', 'non-exercise induced angina')
```

We define a function to quickly plot the data projection onto 2D:

```
In [27]: def plot2D(data, target, labels, atitle):
             fig, ax1 = plt.subplots(figsize=(7,7))
             plt.xlabel('coord 1')
             plt.ylabel('coord 2')
             for i, c in zip(np.unique(target), cycle(colors)):
                 plt.scatter(data[target == i, 0], data[target == i, 1],
                             s=50, alpha=.6, c=c, label='%s - %s' %(i, labels[i]))
             plt.legend()
             plt.title(atitle)
```

```
In [28]: # let's plot the components highlighting healthy and ill subjects
         plot2D(X_pca, diagnose, diagnose_labels, 'PCA: heart disease dataset')
```



What you should have at this step is a projection of your 13 features samples on just 2 dimensions. However, the two classes do not seem nicely separated one from the other. This happened because we did not standardize our data!

So let's see what happens if we standardize the data set:

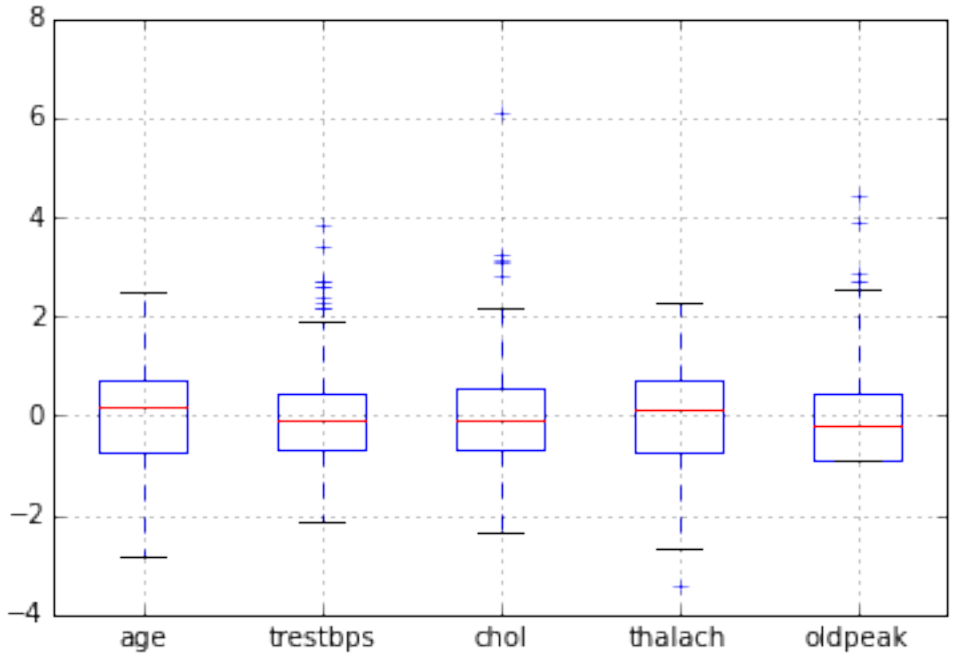
Standardize the data

```
In [29]: # import the library
         from sklearn import preprocessing
         from sklearn.preprocessing import StandardScaler

In [30]: # we scale the dataset and store it as a dataframe
         X_stand = StandardScaler().fit_transform(X)
         stand = pd.DataFrame(X_stand, columns=heart.columns[0: len(heart.columns)-1])
```

- **Exercise 8**: Using some code from above (copy and paste!) make a boxplot of the continuous variables. How does the standardization affect the features?

  the standardization scales the features so that their mean value is zero and their standard deviation is one

```
In [34]: continuous_stand = stand[['age','trestbps', 'chol', 'thalach', 'oldpeak']]
         bs = continuous_stand.boxplot()
```



Let's now run another PCA on our standardized data this time:

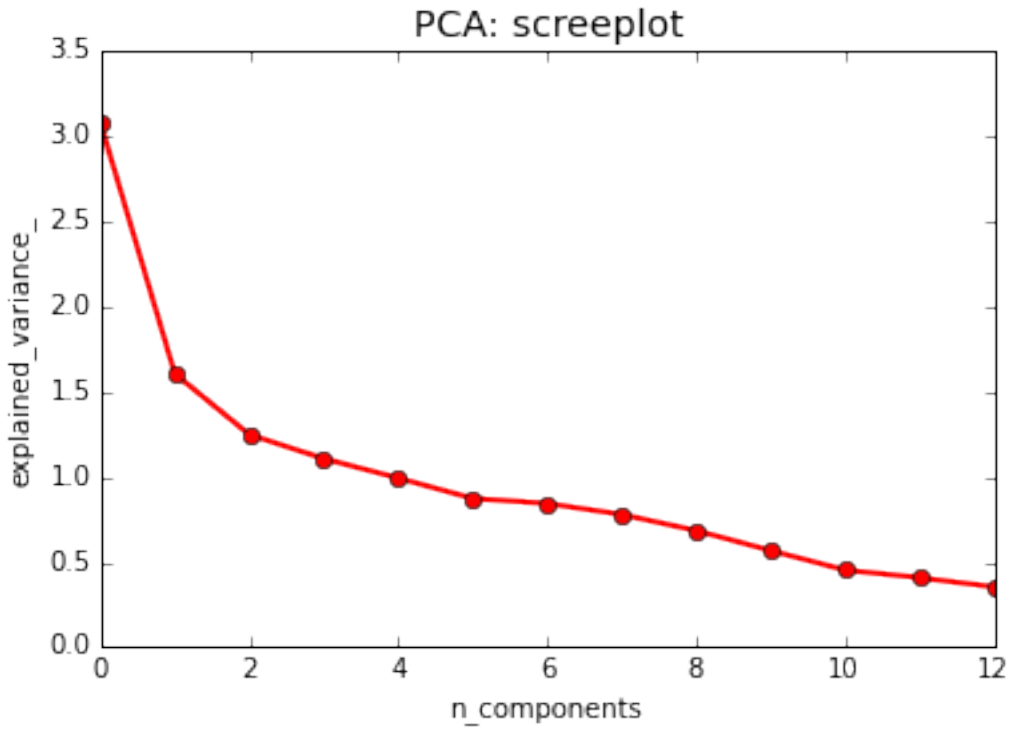
```
In [35]: X_pca_stand = pca.fit_transform(X_stand)
```

- **Exercise 9**: Make a screeplot for the PCA just obtained.

  you see now that the variance explained is different, and that the first 3 components seem to be useful the represent the data!

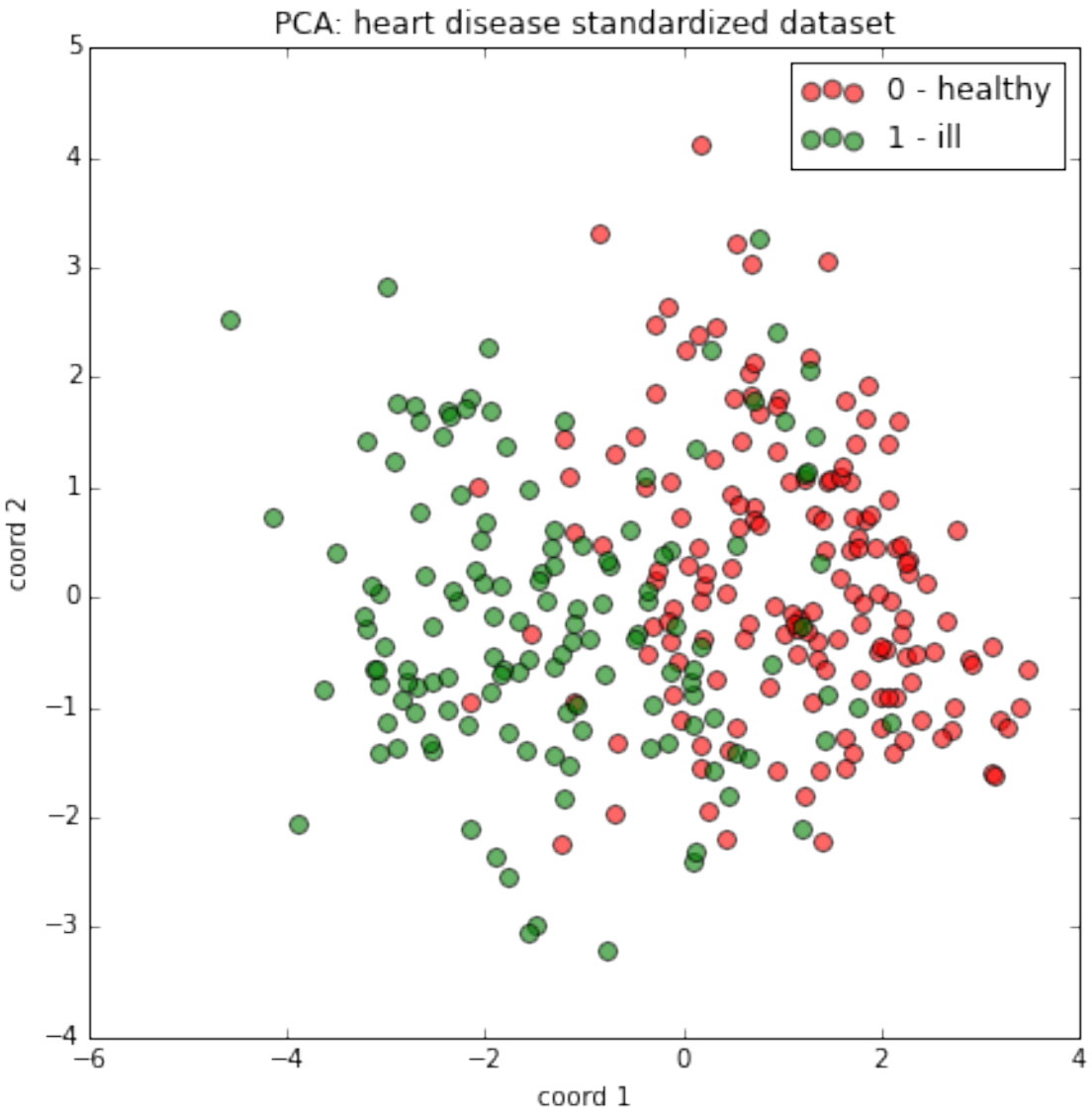
```
In [36]: plt.plot(pca.explained_variance_, 'ro-', linewidth=2)
         plt.xlabel('n_components')
         plt.ylabel('explained_variance_')
         plt.title('PCA: screeplot', fontsize=14)
```

```
Out[36]: <matplotlib.text.Text at 0x1898d550>
```



And the new data projection is:

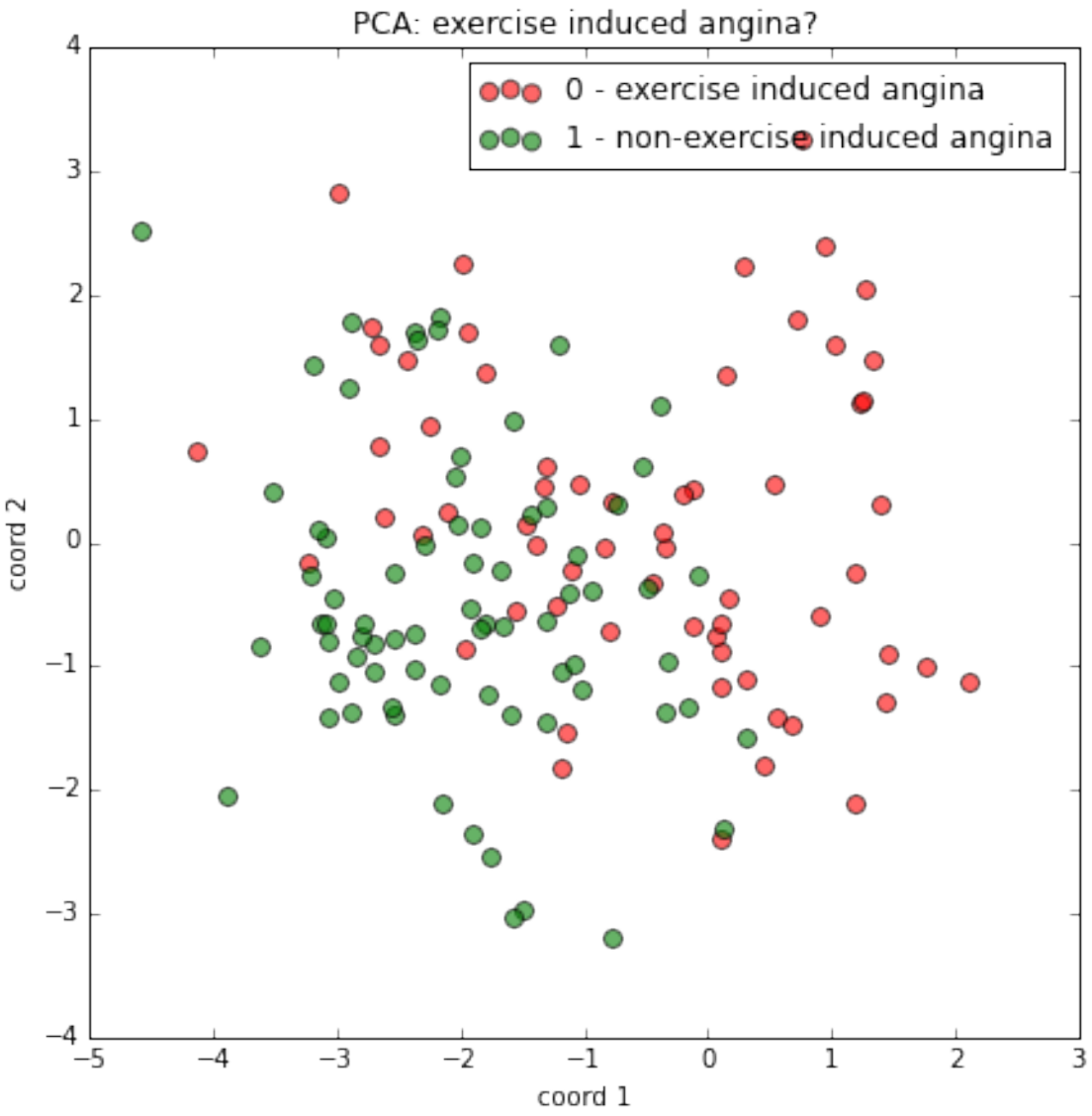
```
In [37]: plot2D(X_pca_stand, diagnose, diagnose_labels, 'PCA: heart disease standardized dataset')
```



The two classes are now more easily separable from each other.

Now we take only the components associated with ill subjects (where diagnose==1), and we plot the 2D projection accordingly:

```
In [39]: pca_ill = X_pca_stand[diagnose==1,:]
         # and these are the 'exercise' values for the ill subjects
         exercise_ill = exercise[diagnose==1]

In [40]: # now we plot the components for the ill subjects, highlighting if they had an exercise-induce
         plot2D(pca_ill, exercise_ill, exercise_labels, 'PCA: exercise induced angina?')
```



- **Exercise 10**: Do you think is it reasonably easy to classify ill subjects in those who had an exercise-induced angina and those who did not?

  this seems to be a non-trivial task!

2 Multi Dimensional Scaling (MDS)

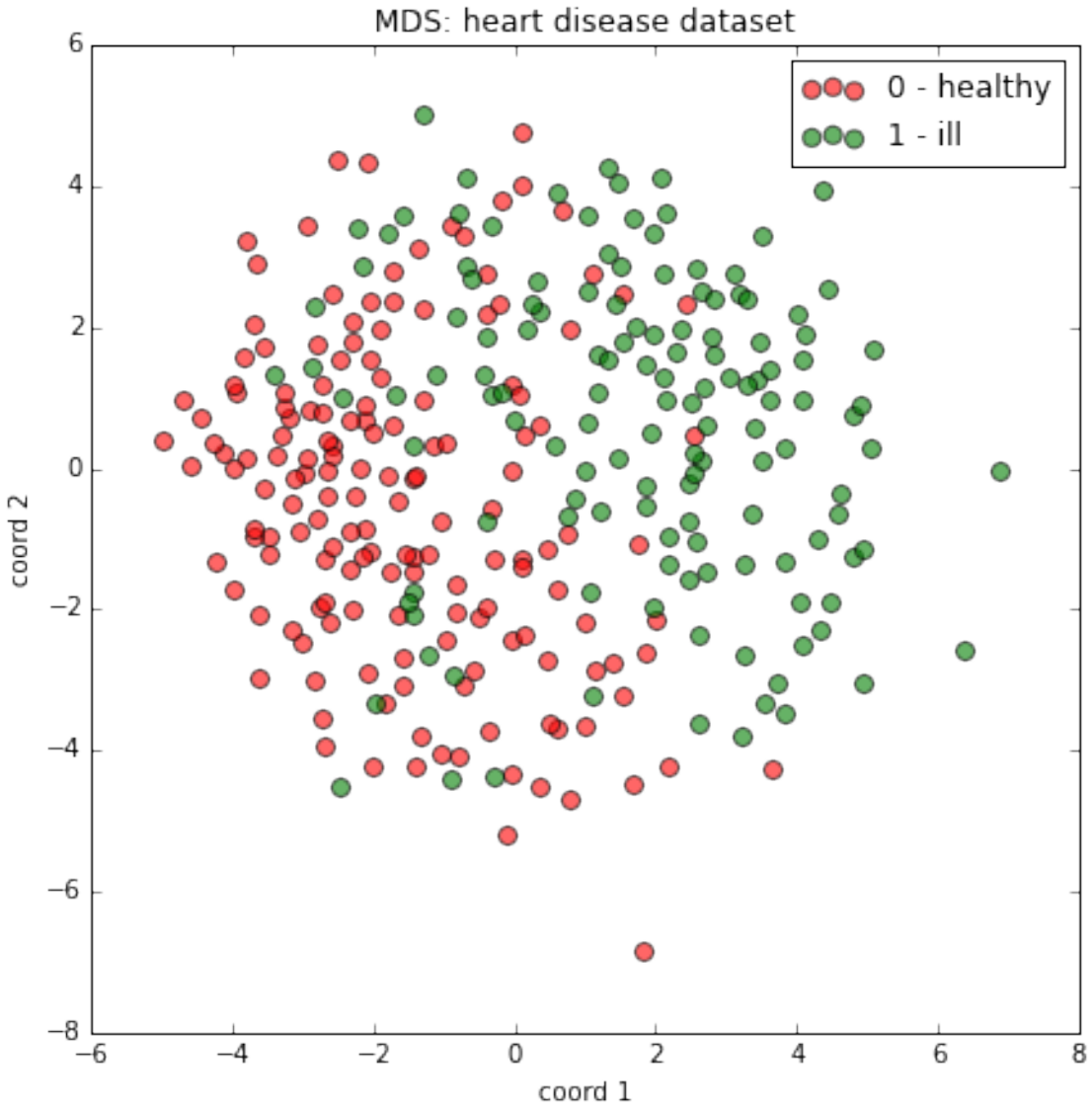
2.1 MDS for heart disease classification

We will first use MDS to visualize the heart disease dataset we just analyzed.

```
In [41]: # import the library we will need
         from sklearn import manifold
```

```
In [42]: mds = manifold.MDS(n_components=2)
```

```
In [43]: # run the MDS
         Y_mds = mds.fit_transform(X_stand)

In [44]: # plot the 2D projection (we can simply reuse the function we defined before!)
         plot2D(Y_mds, diagnose, diagnose_labels, 'MDS: heart disease dataset')
```



- **Exercise 11**: This 2D projection is somehow different from the one obtained through a PCA. Can you explain how the two techniques are different?

  remember: PCA tries to retain most of the variance of the data points, while MDS tries to preserve as much as possible distances between points (points close to each other in the original space are also close to each other in the mapped space)

2.2 MDS for biological response prediction

We will now use MDS for a different dataset.

2.2.1 The data

```
In [45]: # load the data from a text file
         response = pd.read_csv('biol_response.csv', header=0)
```

```
In [46]: print response.shape
```

```
(3751, 1777)
```

- **Exercise 12**: Have a look at the data. How many samples does this dataset contain? How many features?

  the dataset contains 3751 samples, with 1776 features (the first column in the target (Activity))

2.2.2 Description of the biological response dataset

This dataset has been used in a *kaggle* competition to predict biological response of molecules from their chemical properties. Each row in this data set represents a molecule. The first column contains experimental data describing an actual biological response; the molecule was seen to elicit this response (1), or not (0). The remaining columns represent molecular descriptors (d1 through d1776), these are calculated properties that can capture some of the characteristics of the molecule - for example size, shape, or elemental constitution. The descriptor matrix has been normalized. For more information on the challenge: https://www.kaggle.com/c/bioresponse.

- **Exercise 13**: Get the unique counts of the *Activity* variable (again use some code from above!). How many active samples does the dataset contain? How many inactive samples?

  there are 2034 active molecules, and 1717 inactive molecules

```
In [47]: pd.value_counts(response.Activity)
```

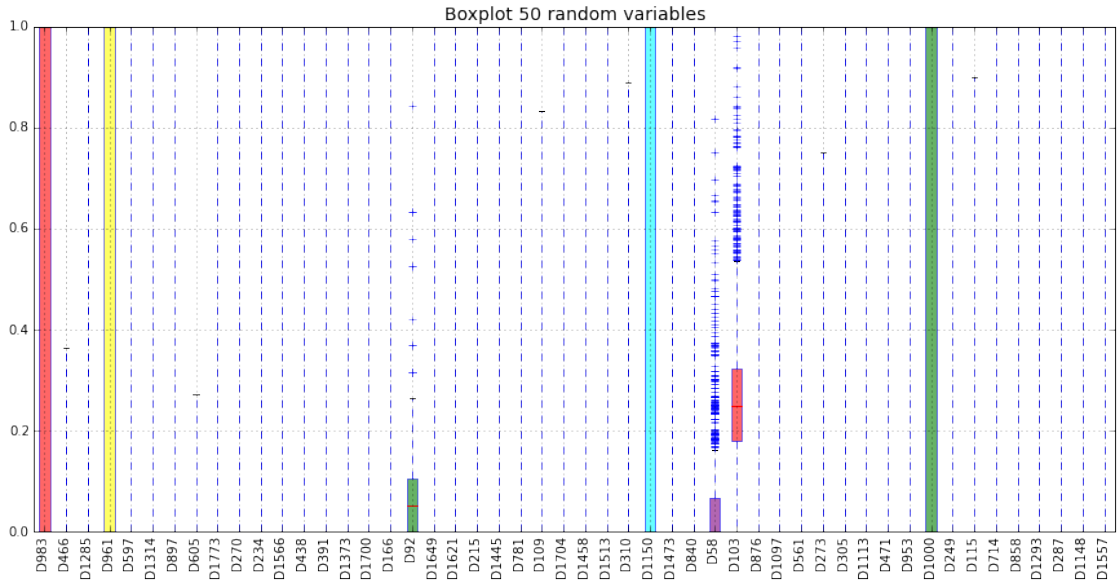
```
Out[47]: 1    2034
         0    1717
         dtype: int64
```

Visualize the data This is a big dataset, let's just randomly pick up 50 variables and plot them:

```
In [48]: columns = np.random.choice(response.columns.values, 50)
```

```
In [49]: plt.figure(figsize=(15,7))
         bp = response[columns].boxplot(patch_artist=True)
         plt.xticks(rotation=90) # set the label on the x axis rotated in vertical
         for patch, color in zip(bp['boxes'], cycle(colors)):
             patch.set_facecolor(color)
             patch.set_alpha(.6)
         plt.title('Boxplot 50 random variables', fontsize=14)
```

```
Out[49]: <matplotlib.text.Text at 0x1f5c6240>
```



- **Exercise 14**: This dataset is already normalized. Can you explain in which way? Which normalization method has been applied to the data?

  a range scaling has been applied, so that the features are all in the same range, in this case [0, 1]

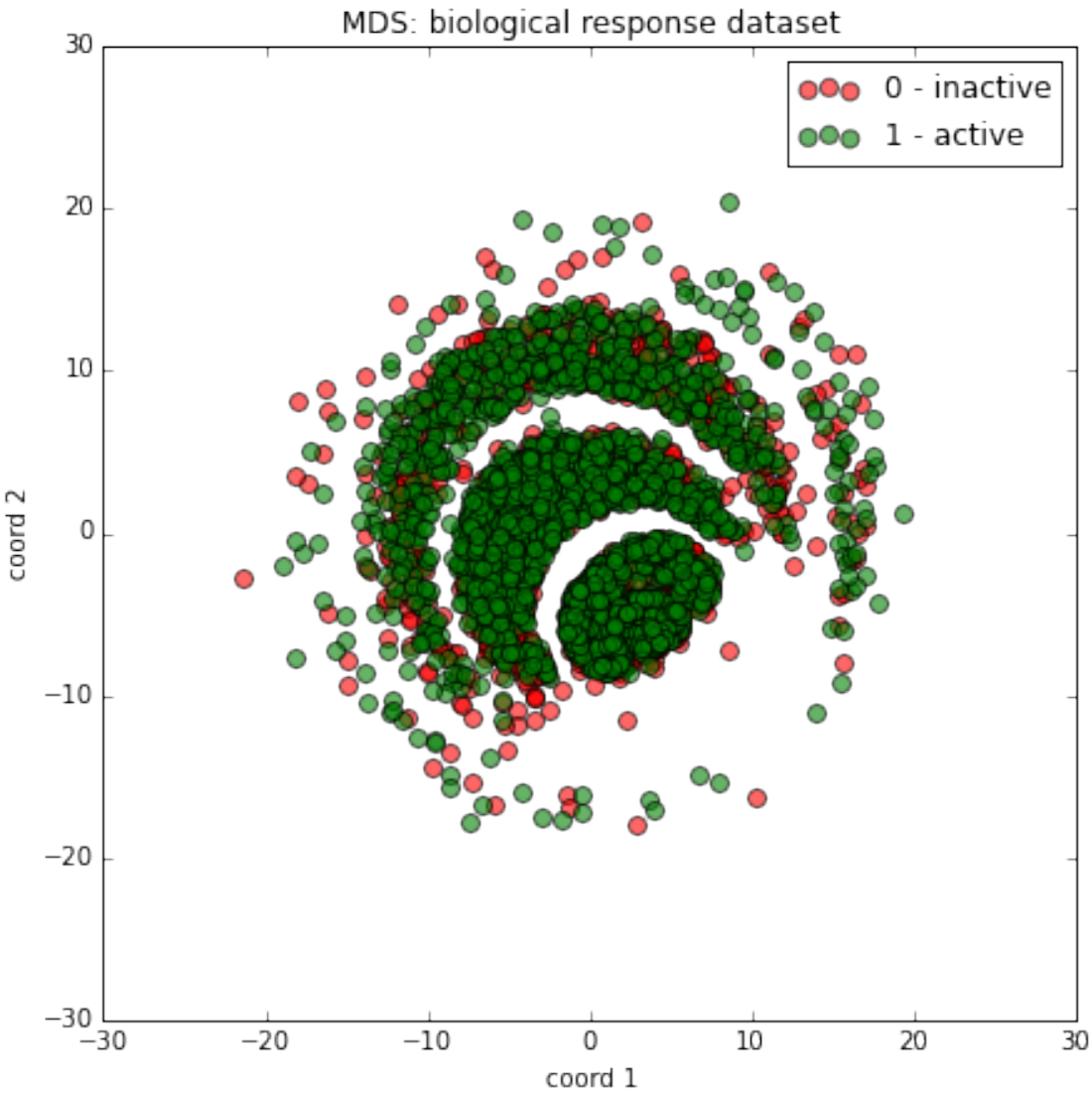
```
In [50]: # we now convert the dataframe to a numpy array
         X = response.values
         X = X [:, 1:X.shape[1]] #skip first column
         activity = np.asarray(response.Activity) # numeric target (y)
         activity_labels = ('inactive', 'active')
```

Now we run the MDS: this is a big dataset, it will take some time...

```
In [51]: Y_mds = mds.fit_transform(X)
```

We can now just use the same function of before and plot the 2D projection:

```
In [52]: plot2D(Y_mds, activity, activity_labels, 'MDS: biological response dataset')
```



This doesn't look too nice, does it? There are clearly some clusters, but the overlapping between active and inactive samples is too big to be able to actually tell the two classes apart. The MDS is clearly not the most appropriate technique to visualize this dataset.

2.3 MDS for Yeast Gene Expression Classification

Let's see if the MDS does a better job on Yeast gene expression data.

```
In [53]: # import the file from a text file
         yeast = pd.read_table('yeast.csv', sep = ' ', header=0)

In [54]: print yeast.shape

(605, 82)

In [55]: print yeast.head
```

<bound method DataFrame.head of ORF alpha.0 alpha.7 alpha.14 alpha.21 alpha.28 alpha.35

0	YBR009C	-2.84	-3.18	-1.94	-0.64	1.04	1.17
1	YDR224C	-1.94	-2.00	-1.79	-0.40	0.86	0.72
2	YDR225W	-2.00	-2.84	-1.94	-0.81	0.86	0.64
3	YNL031C	-2.25	-1.89	-2.00	-0.43	0.40	0.82
4	YPL127C	-2.12	-2.00	-0.45	0.43	0.78	0.86
5	YCR040W	-0.29	0.31	-0.20	-0.04	-0.38	0.11
6	YCR065W	-1.22	-0.23	0.54	0.66	0.18	0.07
7	YER088C	-0.38	-0.47	-0.89	-0.76	-0.86	-0.81
8	YER164W	-0.18	-0.12	-0.22	0.03	-0.09	0.16
9	YKR099W	-0.49	-0.51	-1.22	-1.00	-0.97	-0.67
10	YPL177C	0.58	-0.14	-0.34	-0.49	-0.09	0.11
11	YAL015C	-0.10	-0.12	0.01	-0.34	-0.23	0.14
12	YAL038W	-0.09	-0.58	-0.23	-0.20	-0.12	-0.42
13	YAL040C	1.31	0.45	0.74	-0.76	-0.36	-0.42
14	YAL041W	-0.27	-0.74	-0.22	0.11	-0.09	0.01
15	YAL043C	-0.20	-0.17	-0.15	-0.34	-0.29	-0.36
16	YAL067C	-0.01	1.60	0.20	-0.38	-0.07	0.11
17	YAR008W	-0.43	-0.60	0.38	0.30	0.08	-0.27
18	YBL035C	-0.45	-0.64	1.01	1.14	0.45	-0.40
19	YBL038W	-0.27	-0.03	-0.14	-0.42	-0.12	-0.04
20	YBL052C	-0.14	-0.79	-0.69	-0.47	-0.38	0.15
21	YBL056W	-0.30	-0.64	-0.10	-0.15	-0.10	0.04
22	YBL079W	-0.47	-0.54	-0.58	-0.54	-0.29	-0.40
23	YBL097W	-0.97	-0.42	-0.92	-0.43	-0.58	0.01
24	YBL105C	-0.38	0.01	-0.38	-0.51	-0.27	-0.54
25	YBR008C	-0.22	-0.43	-0.34	0.30	0.16	-0.17
26	YBR017C	-0.14	-0.42	0.06	-0.20	0.23	0.29
27	YBR034C	0.11	-1.74	-1.12	-1.00	-0.32	-0.74
28	YBR036C	-0.12	-0.32	0.12	-0.27	0.08	-0.30
29	YBR038W	-1.51	-2.18	-1.03	-2.56	-2.00	-0.89
..	...	...	...	...	...	...	...
575	YHL033C	0.04	-0.14	-0.23	0.07	-0.10	0.14
576	YHR010W	0.16	-0.09	-0.09	0.04	-0.14	0.07
577	YHR141C	-0.23	0.18	-0.54	-0.03	-0.62	0.11
578	YIL052C	0.10	-0.07	-0.23	0.03	-0.20	0.03
579	YIL133C	0.15	-0.49	0.18	-0.38	0.29	0.28
580	YIL148W	-0.27	-0.01	-0.38	-0.17	-0.56	-0.06
581	YJL136C	-0.17	-0.58	-0.23	-0.25	-0.36	-0.54
582	YJL177W	0.20	-0.15	-0.20	-0.06	0.08	-0.18
583	YJL191W	-0.29	-0.49	-0.42	-0.38	-0.22	-0.47
584	YKL006W	0.38	0.01	0.11	0.11	0.33	0.50
585	YKL156W	-0.38	-0.42	-0.81	-0.38	-0.89	-0.06
586	YKL180W	-0.01	-0.20	-0.32	0.04	-0.17	-0.03
587	YLL045C	-0.09	-0.20	-0.54	-0.01	-0.23	-0.07
588	YLR029C	0.39	-0.04	-0.25	0.12	-0.29	0.08
589	YLR167W	0.19	-0.23	0.04	-0.03	0.19	-0.17
590	YLR340W	0.18	-0.27	-0.29	-0.12	-0.07	-0.30
591	YLR344W	-0.17	-0.56	-0.34	-0.51	-0.29	-0.47
592	YLR367W	-0.12	-0.25	-0.10	0.20	-0.12	-0.04
593	YML063W	0.06	-0.17	-0.23	-0.07	-0.09	-0.23
594	YNL067W	0.03	-0.27	-0.12	-0.45	-0.14	-0.45
595	YNL069C	0.14	-0.04	0.08	-0.27	0.10	-0.27
596	YNL302C	0.18	0.12	0.06	0.03	0.01	0.19

597	YOL121C	0.20	-0.36	-0.25	-0.29	-0.32	-0.40
598	YOR063W	0.29	-0.30	-0.01	-0.14	0.16	0.29
599	YOR293W	0.01	0.03	-0.01	0.18	0.11	0.34
600	YPL143W	0.23	-0.23	-0.17	-0.14	-0.04	-0.01
601	YPL198W	-0.01	-0.34	-0.10	-0.25	-0.09	-0.18
602	YPL220W	0.40	0.14	0.42	0.15	0.34	0.01
603	YPR043W	-0.27	-0.25	-0.71	-0.40	-0.49	-0.40
604	YPR102C	0.14	-0.18	0.03	0.08	0.06	0.15

	alpha.42	alpha.49	alpha.56	...	cold.160	diau.a	diau.b	\
0	0.20	0.48	-0.79	...	-1.56	0.06	-0.01	
1	0.51	-0.51	-0.56	...	-0.86	0.36	0.23	
2	0.58	-0.45	-0.71	...	-1.47	0.26	0.48	
3	0.19	-0.60	-0.79	...	-1.09	0.12	-0.06	
4	0.72	0.19	-0.30	...	0.01	-0.20	-0.04	
5	-0.20	-0.40	-0.12	...	-0.06	-0.69	0.66	
6	-0.69	-0.47	-0.43	...	0.36	0.04	-0.01	
7	-0.69	-0.54	-0.42	...	0.45	0.32	0.32	
8	-0.03	0.15	0.29	...	0.98	-0.03	-0.07	
9	-0.38	-0.25	-0.17	...	0.45	-0.04	-0.27	
10	0.01	-0.51	0.03	...	-0.79	-0.07	-0.32	
11	0.19	-0.38	0.28	...	-0.34	-0.07	-0.20	
12	0.11	0.14	0.12	...	-0.42	0.10	0.19	
13	0.37	0.10	0.80	...	0.68	-0.01	0.08	
14	-0.07	0.32	-0.23	...	0.53	0.12	-0.03	
15	0.07	-0.25	0.04	...	0.74	0.03	-0.10	
16	0.67	0.25	0.36	...	0.24	-1.12	-1.00	
17	-0.30	0.11	-0.45	...	-0.34	-0.25	-0.74	
18	-0.64	0.15	-1.09	...	-0.74	-0.49	-0.49	
19	0.28	-0.07	-0.09	...	0.31	0.04	-0.03	
20	-0.09	0.28	-0.22	...	0.82	-0.07	0.03	
21	0.08	0.38	-0.10	...	0.03	0.01	0.41	
22	-0.10	-0.32	-0.22	...	0.18	0.07	-0.17	
23	-0.43	0.59	-0.54	...	-0.14	-0.40	0.15	
24	-0.04	0.45	0.03	...	0.71	0.19	0.30	
25	0.04	0.08	-0.38	...	0.30	-0.25	-0.14	
26	0.08	0.10	0.12	...	-0.54	0.29	0.08	
27	-0.15	0.60	-0.23	...	-0.60	-0.04	0.11	
28	0.16	0.50	-0.18	...	-0.25	-0.27	0.04	
29	0.40	0.39	1.31	...	0.57	0.03	-0.03	
..	...	...	...	...	...	...	...	
575	0.04	0.29	0.33	...	-1.79	0.34	0.23	
576	-0.10	0.31	0.43	...	-1.06	0.55	0.40	
577	-0.23	0.19	-0.06	...	-1.69	-0.03	-0.20	
578	-0.22	0.37	0.40	...	-1.60	0.42	-0.09	
579	0.39	-0.06	0.26	...	-1.84	0.34	0.26	
580	-0.40	-0.32	-0.12	...	-1.51	-0.12	-0.22	
581	-0.32	-0.47	-0.09	...	-2.56	0.19	-0.09	
582	0.20	0.14	0.07	...	-2.06	0.33	0.14	
583	-0.32	-0.36	-0.18	...	-0.84	0.14	-0.07	
584	0.24	0.06	0.12	...	-2.18	0.16	0.07	
585	-0.42	-0.09	-0.23	...	-1.15	0.01	-0.06	
586	-0.17	-0.04	0.21	...	-2.56	0.01	-0.06	
587	0.03	-0.15	0.28	...	-1.74	0.19	-0.17	

588	0.01	0.12	0.38	...	-1.79	0.44	0.21
589	-0.09	0.04	0.54	...	-2.06	0.12	0.03
590	0.04	-0.22	0.01	...	-2.00	0.21	0.32
591	-0.10	-0.23	-0.04	...	-0.67	-0.09	-0.17
592	-0.12	0.12	-0.04	...	-1.84	0.41	0.36
593	0.10	0.14	0.10	...	-2.06	0.21	0.21
594	0.18	-0.34	0.01	...	-1.25	0.04	-0.36
595	0.25	-0.23	-0.06	...	-1.06	-0.09	-0.04
596	-0.06	0.31	0.15	...	-0.74	0.66	0.32
597	0.07	-0.27	0.12	...	-1.89	0.42	0.01
598	0.29	0.01	0.07	...	-0.76	0.14	0.07
599	-0.23	0.48	0.33	...	-1.84	0.33	-0.14
600	0.16	0.03	0.29	...	-1.60	0.16	-0.14
601	0.04	-0.15	0.06	...	-1.89	0.11	-0.20
602	0.31	0.11	0.30	...	-1.12	0.04	-0.01
603	-0.04	-0.34	-0.25	...	-1.84	0.14	-0.07
604	0.12	0.21	0.41	...	-1.36	0.26	0.01

	diau.c	diau.d	diau.e	diau.f	diau.g	Label	gene_class
0	0.20	-0.14	0.34	-0.29	-0.51	Hist	0
1	0.19	-0.01	0.03	-0.20	-1.00	Hist	0
2	0.62	0.31	0.46	0.03	-0.67	Hist	0
3	0.03	-0.01	0.07	0.82	-0.25	Hist	0
4	-0.14	-0.18	-0.36	-0.69	-0.69	Hist	0
5	-0.22	-0.62	-0.74	0.20	0.04	HTH	1
6	-0.47	-1.00	-1.15	-1.43	-1.40	HTH	1
7	0.75	0.12	-0.12	0.45	1.06	HTH	1
8	-0.20	-0.51	-0.62	-0.56	-0.42	HTH	1
9	0.33	-0.81	-0.69	-0.62	-1.06	HTH	1
10	-0.47	-0.71	-0.15	0.32	0.32	HTH	1
11	-0.20	-0.15	0.08	0.83	0.18	nc	2
12	0.29	-0.36	-0.40	-1.32	-2.32	nc	2
13	0.30	-0.58	-0.67	0.18	-0.47	nc	2
14	0.06	-0.62	-0.29	-0.81	-0.47	nc	2
15	-0.62	-1.15	-0.62	-1.12	-0.40	nc	2
16	-0.42	-1.03	-0.89	0.44	0.24	nc	2
17	-0.51	-0.67	-0.64	-0.60	-1.12	nc	2
18	0.21	-0.51	-0.67	0.21	-0.54	nc	2
19	0.11	0.31	0.50	1.58	1.19	nc	2
20	0.01	-0.23	-0.30	-0.49	-0.74	nc	2
21	0.55	0.11	0.12	0.56	0.12	nc	2
22	-0.32	-0.51	-0.45	-0.84	-1.12	nc	2
23	0.32	-0.30	-0.58	0.31	0.41	nc	2
24	0.28	-0.29	-0.03	0.29	-0.03	nc	2
25	0.55	-0.29	-0.32	0.37	-0.04	nc	2
26	0.14	-0.29	-0.07	-0.84	-1.00	nc	2
27	0.16	-1.09	-1.06	-0.60	-1.36	nc	2
28	0.82	0.30	0.07	0.68	0.29	nc	2
29	0.06	-0.22	-0.30	-0.58	-0.74	nc	2
..	...	...	...	...	...	...	...
575	0.19	-0.18	-0.51	-1.56	-2.47	Ribo	5
576	0.61	0.01	0.01	-1.09	-1.94	Ribo	5
577	-0.22	-0.29	-0.36	-1.22	-2.06	Ribo	5
578	0.28	-0.09	-0.34	-1.64	-2.25	Ribo	5

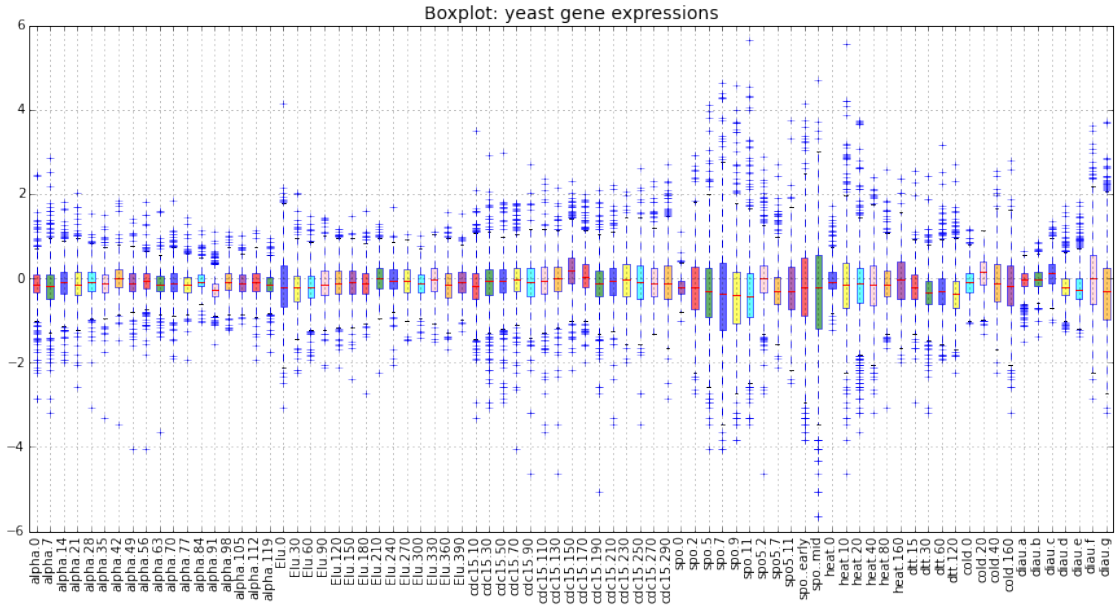
```
579    0.21   -0.38   -0.34   -1.60   -2.25   Ribo          5
580   -0.42   -0.51   -0.40   -1.15   -2.00   Ribo          5
581    0.34   -0.07   -0.47   -1.47   -2.40   Ribo          5
582    0.14   -0.42   -0.62   -1.94   -2.40   Ribo          5
583    0.34   -0.30   -0.25   -0.92   -1.84   Ribo          5
584    0.15   -0.06   -0.27   -1.06   -2.12   Ribo          5
585   -0.18   -0.29   -0.38   -1.12   -1.47   Ribo          5
586   -0.04   -0.12    0.06    0.37   -0.17   Ribo          5
587   -0.09   -0.10   -0.10   -1.94   -3.06   Ribo          5
588    0.46    0.36    0.10   -1.29   -2.18   Ribo          5
589    0.26    0.21    0.08   -0.56   -1.60   Ribo          5
590    0.48   -0.09   -0.43   -1.79   -2.74   Ribo          5
591   -0.36   -0.71   -0.51   -1.56   -2.40   Ribo          5
592    0.38    0.07   -0.09   -1.32   -2.18   Ribo          5
593    0.39    0.01   -0.03   -1.60   -2.47   Ribo          5
594   -0.14   -0.40   -0.22   -0.71   -2.40   Ribo          5
595    0.30   -0.20   -0.09   -1.12   -2.74   Ribo          5
596    0.37    0.04    0.11   -1.15   -2.25   Ribo          5
597    0.14   -0.12    0.01   -1.47   -2.25   Ribo          5
598    0.28   -0.25   -0.29   -1.69   -2.25   Ribo          5
599    0.07   -0.30   -0.36   -1.32   -1.89   Ribo          5
600    0.38   -0.36   -0.30   -1.51   -1.36   Ribo          5
601    0.01   -0.60   -0.64   -1.79   -2.18   Ribo          5
602    0.39   -0.36   -0.04   -1.60   -2.74   Ribo          5
603   -0.04   -0.10   -0.30   -0.36   -1.79   Ribo          5
604    0.20    0.03    0.04   -1.32   -2.18   Ribo          5

[605 rows x 82 columns]>
```

- **Exercise 15**: Have a look at the data. How many samples does this dataset contain? How many features?

  we have 605 genes (samples) and 82 columns in total, but one column is for the ORF, and the last two are for the Label and the gene class, so we have 82-3=79 features

Visualize the data

```
In [56]: plt.figure(figsize=(15,7))
         cols = [col for col in yeast.columns if col not in ['gene_class']]
         bp = yeast[cols].boxplot(patch_artist=True)
         plt.xticks(rotation=90)
         for patch, color in zip(bp['boxes'], cycle(colors)):
             patch.set_facecolor(color)
             patch.set_alpha(.6)
         plt.title('Boxplot: yeast gene expressions', fontsize=14)

Out[56]: <matplotlib.text.Text at 0x1fa4d668>
```



- **Exercise 16**: Do the features have all the same range? If yes, which one?

  yes, [-6, 6]

- **Exercise 17**: Get the unique counts of the Label variable. How many classes are there for the genes? Which is the most abundant class?

  most of the genes are non classified (nc, 524); there are 5 classes, and the most abundant one is the Ribo class

```
In [57]: pd.value_counts(yeast.Label)
```

```
Out[57]: nc          524
         Ribo         49
         Resp         13
         Proteas       8
         HTH           6
         Hist          5
         dtype: int64
```

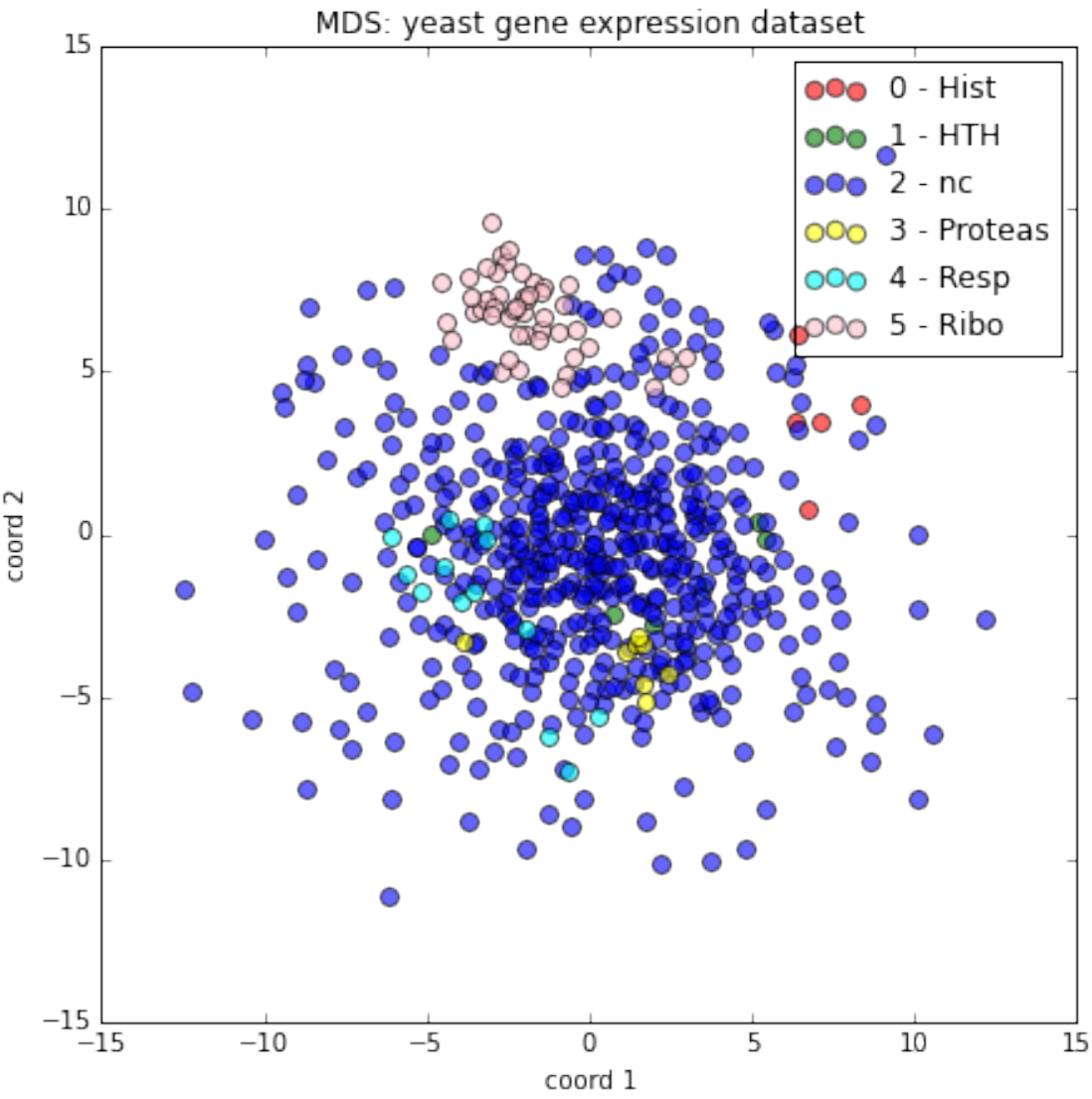
```
In [58]: # let's get rid of some columns we do not need for the MDS
         cols = [col for col in yeast.columns if col not in ['ORF', 'Label','gene_class']]
         X = np.asarray(yeast[cols])
```

```
In [59]: # run the MDS
         Y_mds = mds.fit_transform(X)
```

```
In [62]: gene_class = np.asarray(yeast.gene_class)
         gene_labels = pd.unique(yeast.Label)

         print gene_class
         print gene_labels
```

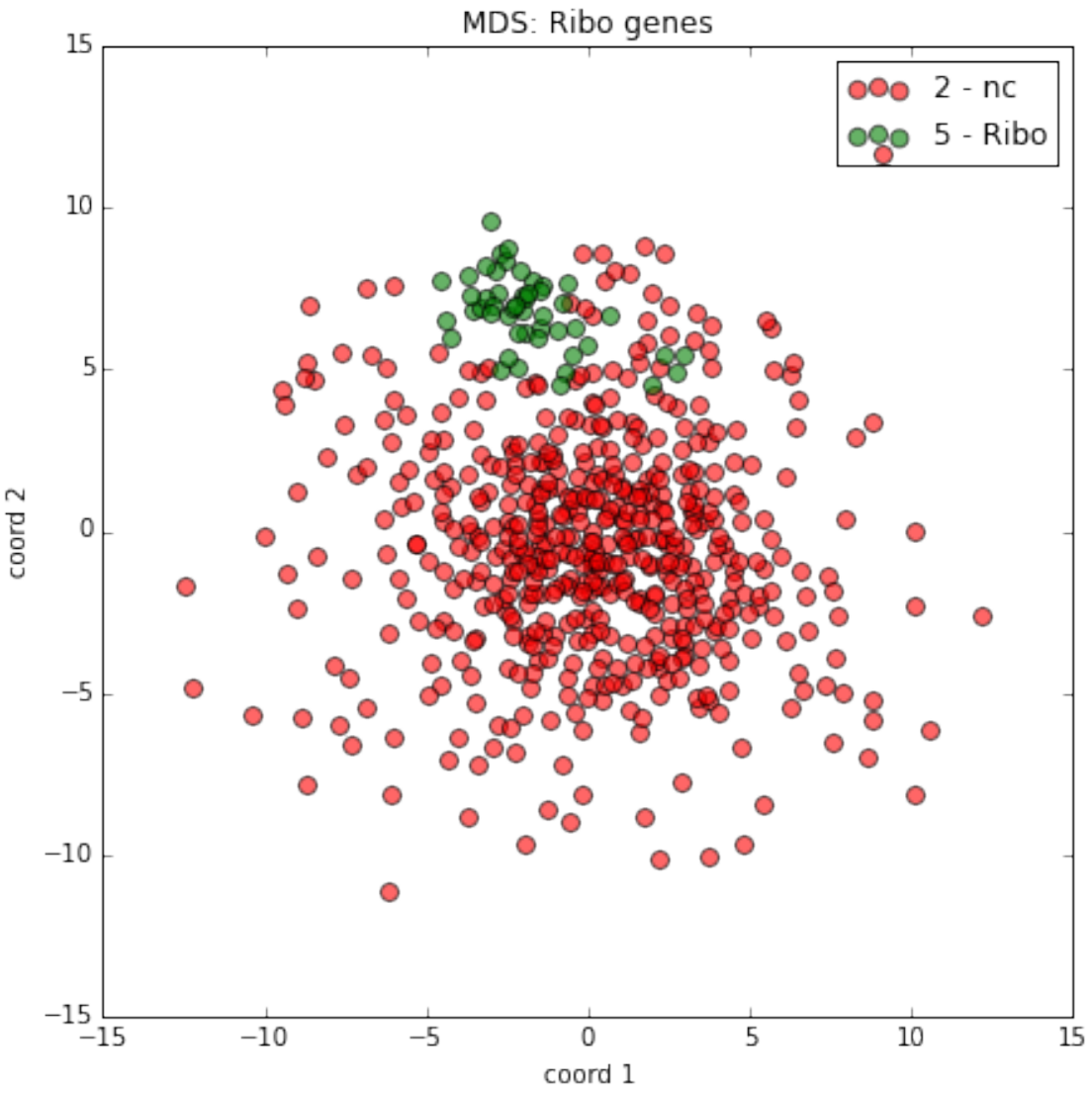
```
[0 0 0 0 0 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4
 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5]
['Hist' 'HTH' 'nc' 'Proteas' 'Resp' 'Ribo']
```

```
In [63]: # plot the 2D projection
         plot2D(Y_mds, gene_class, gene_labels, 'MDS: yeast gene expression dataset')
```



Let's just focus on *Ribo* and *nc* (non-classified) genes, and look at the 2D projection.

```
In [64]: subset_mds = Y_mds[(gene_class==2)|(gene_class==5),:]
         subset_gene_class = gene_class[(gene_class==2)|(gene_class==5)]
         plot2D(subset_mds, subset_gene_class, gene_labels, 'MDS: Ribo genes')
```



- **Exercise 18**: Can you easily distinguish the Ribo genes from the rest?

  pretty much, yes

- **Exercise 19**: By default, the **Euclidean** distance is used in the MDS to compute the dissimilarities between the data points. Do you know any other metrics that could be used instead to represent similarities/dissimilarities?

  We mentioned Pearson coefficient, cosine similarity