A Tutorial on PCA Interpretation using CompClust

Joe Roden, Diane Trout & Brandon King

Version 1.2
July 29, 2005

Contents

1 Introduction **1**

1.1 Purpose . . . 1

1.2 Prerequisites . . . 1

1.3 CompClust Background . . . 1

1.4 Principal Components Analysis (PCA) Background . . . 1

1.5 Interpreting Principal Components . . . 1

2 Data Preparation **2**

2.1 Import the necessary software . . . 2

2.2 Load a dataset . . . 2

2.3 Attach dataset's labelings . . . 3

2.4 Construct a PCAGinsu object . . . 3

3 Review principal component outliers using TK graphics **3**

3.1 View outlier scatter plot . . . 4

3.2 View outlier trajectory plots . . . 4

4 Review principal component outliers using matplotlib graphics **4**

4.1 View outlier scatter plot . . . 5

4.2 View outlier trajectory plots . . . 5

5 List a principal component's outliers and significant conditions **5**

5.1 View outlier list . . . 5

5.2 View significant condition list . . . 6

6 Look for condition covariates that correlate with condition partitions **7**

7 Performing batch PCA interpretation **8**

8 Further Information **9**

8.1 Publication: Mining Gene Expression Data by Interpreting Principal Components . . . 9

8.2 The CompClust Python Package . . . 9

8.3 Other CompClust Tutorials . . . 9

8.3.1 CompClustTk Manual and Tutorial . . . 9

8.3.2 A First Tutorial on the MLX schema . . . 9

8.3.3 A Quick Start Guide to Microarray Analysis using CompClust . . . 9

9 Acknowledgements **10**

1 Introduction

1.1 Purpose

This tutorial is designed to introduce software developers to the PCAGinsu modules contained in the CompClust Python package. The CompClust tools for comparative cluster analysis are described in (Hart et al., 2005). See Section 8 for additional information on CompClust as well as the publication describing our PCA interpretation approach. After following this tutorial a software developer aught to be able to load his own gene expression microarray datasets and labelings, perform PCA on the data, and generate interpretations of the PCA results as described in (Roden et al., 2005).

The approach taken in this tutorial is to guide a software developer to become familiar with the CompClust package's application programming interface (API) so that he or she may proceed to use the API to perform PCA interpretation with their own datasets.

1.2 Prerequisites

Software developers interested in using the PCAGinsu components to interpret PCA results should be familiar with Python as well as with key CompClust concepts, in particular, Datasets, Labelings, and Views. See

1.3 CompClust Background

CompClust provides software tools to explore and quantify relationships between clustering results. Its development has been largely built around requirements for microarray data analysis, but can be easily used for other types of biological array data and that of other scientific domains.

1.4 Principal Components Analysis (PCA) Background

Principal Components Analysis (PCA) is a numerical procedure for analyzing the sources of variation present in a multi-dimensional dataset. We employ it to analyze gene expression microarray datasets, but the concept is general to any multidimensional dataset worth analyzing.

We carry out PCA by applying singular value decomposition (SVD) to the covariance matrix of D, $cov(D)$, to produce the decomposition that contains the eigenvectors of $cov(D)$ in the columns of U and eigenvalues in the diagonal of S such that the eigenvalues are sorted by descending size.

Each covariance eigenvector, or principal component, explains a fraction of the total variance contained in the dataset, and each principal component P_{n+1} is orthogonal to the previous principal component P_n. such that they define the basis of a new vector space P.

1.5 Interpreting Principal Components

It is our belief that PCA is one of a set of tools that can help investigators better understand the sources of variation present in a microarray dataset. Each principal component measures, and to some degree models, some source of variance observed in the dataset. At the simplest level one observes the variance explained by each principal component, and perhaps takes note of that principal component's eigen vector. By analyzing each principal component a bit further we can better appreciate what is driving that component's variation. Our strategy is to identify the data points (genes) at the extremes of each principal component axis, and then determine which conditions are driving those outlier genes to be significantly differentially expressed. Beyond that, the covariate factors that correlate well with the significant conditions can be identified, so we may hypothesize that they are substantial sources of gene expression variation.

2 Data Preparation

2.1 Import the necessary software

The first step is to import the necessary software. If one is using the windows compclust-shell program, this step has already been done for you, however it is perfectly acceptable to re-import the same modules.

```
## compClust modules
from compClust.mlx import datasets
from compClust.mlx import labelings
from compClust.mlx import views
from compClust.mlx import wrapper
from compClust.mlx import pcaGinsu
from compClust.iplot import IPlotTk as IPlot

## Useful standard python modules
import os
```

2.2 Load a dataset

The Dataset class is capable of casting a number of different sources of information into the proper format such as strings (which are interpreted as file names), list of lists, and numeric arrays to list the most common.

To avoid having to construct the full path name for all of the files that we'll be loading it is convenient to just change to the directory that they're all located in.

For this tutorial will be starting with the Cho Yeast cell cycling data set ((Cho et al., 1998)) as it is fairly small and easy to work with.

If you used the default options for the windows installer the data should be located in C:\Program Files\CompClustShell\Examples\

```
# This cd command is actually ipython specific, if you're
# using python, use os.chdir(os.path.join('Examples', 'ChoCellCycling'))
cd Examples/ChoCellCycling

# at last some data
cho = datasets.Dataset("ChoCycling.dat", 'cho')
```

As a finally check if you type ”cho.numRows” python should return 384.

2.3 Attach dataset's labelings

Once we have a dataset loaded we can attach labelings to it to provide additional information about the various rows, columns, and even cells of the dataset. For instance one typically wants a unique name for each row attached to the dataset so the summary plots can tell you the name of an interesting looking vector. With the Cho data set, this would be ORF (gene) identifiers.

```
orfs = labelings.Labeling(cho, 'orfs')
orfs.labelRows("ORFs.rlab")
cho.setPrimaryRowLabeling(orfs)
```

Once we have our primary labeling we might as well attach some other useful pieces of information. Included with the Cho dataset are labelings containing such things as gene common names, DiagEM clustering results, and a clustering which was manually done by (Cho et al., 1998).

```
names = labelings.Labeling(cho, 'names')
names.labelRows("CommonNames.rlab")

em = labelings.Labeling(cho, 'em')
em.labelRows("EM.rlab")

cho_clustering = labelings.Labeling(cho, 'cho_clustering')
cho_clustering.labelRows("ChoClassification.rlab")
```

We can also attach information to the columns of the dataset. The primaryColumnLabeling is used for many of the condition (x-axis) labels along the plots. For the Cho data set, this would be the time (in hours) at which each sample was taken for the time course experiment.

```
times = labelings.Labeling(cho, 'times')
times.labelCols("times.clab")
cho.setPrimaryColumnLabeling(times)
```

2.4 Construct a PCAGinsu object

There are two different types of PCAGinsu objects that one can instantiate, one defined in compClust.iplot package (See iplot import command in section 2.1) will create interactive plots using the IPlot API (which can be rendered in either Tk or static plots (for web pages)).

Alternatively there is also pcaGinsuVisualizeMatplotlib from the compClust.mlx.pcaGinsu module, which will render static plots with just matplotlib. pcaGinsuVisualizeMatplotlib does have one extra batch processing function which can iterate over all of the principal components and create all the available PCAGinsu analysis output for each component.

For either version of PCAGinsu we need to pass in which dataset we want to analyze. One of the more common optional parameters is how many outliers we want to include in the analysis.

3 Review principal component outliers using TK graphics

The advantage of the Tk plots is that one can click to find out the name of an interesting vector. First we create the PCAGinsu object, which in the case of the yeast cell cycling dataset is fairly quick, but can take tens of minutes to nearly an hour for something like a 33,000 x 158 dataset such as the GNF dataset.

```
ipcaginsu = IPlot.PCAGinsu(cho, 10)
```

3.1 View outlier scatter plot

Next we create the PCAGinsu scatter plot, one important detail is that the IPlot branch of uses zero based arrays, e.g. it counts each principal component as 0, 1, 2, etc. The web interface is built using this version and as python arrays and lists are 0 based it was simpler to make these functions compatible.

The following plotPCvcPCWithOutliersInY call will create a scatter plot displaying principal component 1 along the X axis, and component 2 along the Y axis, while highlighting the 10 highest and lowest outliers. The value 10 outliers was set by the above PCAGinsu call.

```
ipcaginsu.plotPCvsPCWithOutliersInY(0,1)
```

3.2 View outlier trajectory plots

There are two outlier trajectory plots, the first is sorted in the original column order, and again remember that this is 0 based, so the following call will plot the first principal component.

```
ipcaginsu.plotPCNOutlierRowsInOriginalColumnOrder(0)
```

The other trajectory plot is sorted by the difference of the mean "high" vectors and the mean "low" vectors, in order to emphasize the conditions that most affect that principal component. The following call creates a plot of the second principal component.

```
ipcaginsu.plotPCNOutlierRowsInSigGroupOrder(1)
```

4 Review principal component outliers using matplotlib graphics

Matplotlib was designed to provide a Matlab like environment within python, when using the matplotlib plots it is convenient to have the rest of matplotlib available. http://matplotlib.sourceforge.net

```
from matplotlib.pylab import *
```

One incredibly important difference between the previous functions and the following functions are that these are 1 based. The first principal component is 1, the second is 2, etc.

Both PCAGinsu constructors take a similar list of parameters, they are in order, the dataset to operate on, the number of high and low outliers, the significance threshold, and what is the highest numbered principal component to analyze.

The following function call creates a PCAGinsu object using the cho dataset, still with the 10 outliers, but instead of the default .05 significance, it is using a .01 cutoff, and it will analyze components 1 through 17. (17 happens to the number of principal components that would be naturally found).

```
pcaginsu = pcaGinsu.pcaGinsuVisualizeMatplotlib(cho, 10, .01, 17)
```

4.1 View outlier scatter plot

Here we create and display a scatter plot of the first and second principal components,

```
pcaginsu.plotPCvsPCWithOutliersInY(1,2)
fig = gcf()
show()
```

One might wonder what the fig = gcf() in the above code fragment is for or perhaps why in the following fragments there is a fig.clf() before each plot command. Matplotlib when using the Matlab style commands likes to create a new figure for each plot; however when it tries to show a new plot after the previous one has been closed the interpreter fails to actually render the new plot.

What these commands are doing is that first "fig = gcf() saves a reference to the current figure, and then after we're done with the first plot, clears it to make it ready for the next plot command. Alternatively one can ignore all of the gcf and clf code and just only call show once, after one has created all of the plots.

Additionally Matplotlib http://matplotlib.sourceforge.net does have several examples of how to make matplotlib safer for interactive use.

4.2 View outlier trajectory plots

Here we clear our figure and then display the outliers in original order or what one might consider to be the unsorted native ordering of the dataset. This ordering would be the same ordering as any other unsorted plot that one might chose to create.

```
fig.clf()
pcaginsu.plotPCNOutlierRowsInOriginalColumnOrder(1)
show()
```

In this case we see the same data but sorted by the difference of mean "high" and mean "low" expression. "High" vectors are defined as those that started high, while low vectors are (as one might guess) are the ones that started off as the most negative of all the values in the selected principal component.

```
fig.clf()
pcaginsu.plotPCNOutlierRowsInSigOrder(1)
show()
```

5 List a principal component's outliers and significant conditions

Though plots are a nicely visual way to look for interesting vectors, to actually attempt to find which pca condition might be associated with a particular labeling is more easily done by looking at lists.

5.1 View outlier list

The getOutputForPCNOutliers takes which principal component you want to view and a list of labeling names which you want to see included in the report. These labelings need to be ones that have row labels, such as created by labeling.labelRows from the start of the tutorial.

```
pcaginsu.getOutputForPCNOutliers(1, ['cho_clustering', 'em', 'names'])
```

The result of that command should contain the same information as the following table.

PC-1 10 High/Low	PC-1 Value	em	cho_clustering	names
high	7.66496608144	4	M	WSC4
high	4.17705992859	4	M	YOL019W
high	3.87001547533	4	M	HOF1
high	3.84837129339	5	Early G1	SUR1
high	3.35847421047	4	M	BUB3
high	3.34419451262	4	M	CDC5
high	3.34037443779	4	M	YML034W
high	3.27043501501	4	M	COT1
high	2.82125356759	4	M	HDR1
high	2.80269836671	4	G2	YIL158W
low	-3.26930983481	2	Late G1	HO
low	-3.39173779979	2	Late G1	RNR1
low	-3.44848612811	2	Late G1	YLR183C
low	-3.56081559301	2	Late G1	CDC54
low	-3.59901879069	2	Late G1	YOR144C
low	-3.60057680134	2	Late G1	HST3
low	-3.64414373489	2	Late G1	TOF1
low	-3.79017028645	2	Late G1	SPH1
low	-3.86578252915	2	Late G1	YPL264C
low	-3.93550058084	2	Late G1	HHO1

Interestingly it appears that principal component one helps to differentiate between M phase and the Late G1 phase of the cell cycle.

We do provide a convenience function for saving the output of the getOutput commands, pcaGinsu.write2DStringArrayToF
this function takes the result of the getOutput command, a file name, and an optional delimiter (which defaults to tab).

```
outliers = pcaginsu.getOutputForPCNOutliers(1, ['cho_clustering', 'em', 'names'])
pcaGinsu.write2DStringArrayToFile(outliers, 'pca-outliers-1.txt')
```

5.2 View significant condition list

While the getOutputForPCNOutliers looks for interesting patterns along the rows of a dataset the getOutputForSigGroups looks along columns. Unfortunately the yeast cell cycling dataset is not really one of the better examples of this feature, but its other advantages (being small and published) make up for this lack.

```
pcaginsu.getOutputForSigGroups(1, ['times'])
```

PC-1 10-outlier Up/Flat/Down Columns	times
up	7 h
up	8 h
up	6 h
up	15 h
up	16 h
up	14 h
up	5 h
flat	13 h
down	4 h
flat	0 h
down	9 h
down	12 h
down	10 h
down	11 h
down	3 h
down	1 h
down	2 h

If one looks carefully one can start to see hints of the cell cycle that was captured by this output, that the sequences 5,6,7,8 and 14,15,16 are all "up" while 1,2,3 and 9,10,11,12 are "down" highlights the cyclic nature of the cell cycle.

6 Look for condition covariates that correlate with condition partitions

The scoreColumnLabelingsForPCN attempts to find which column labeling best correlates with this principal component, alas in the case of this dataset there is only one labeling, and so the result any particular call not that interesting.

```
pcaginsu.scoreColumnLabelingsForPCN(1)
```

However one could determine which principal component best correlates with time, by looking at all of the principal components

```
for i in range(1,pcaginsu.rowPCAView.numCols+1):
  print i, pcaginsu.scoreColumnLabelingForPCN(i, verbose=False)[1]
```

Results in the following output.

```
1 [0.67150984942408032]
2 [0.69151881316901098]
3 [0.67958446890131774]
4 [0.67696146850058647]
5 [0.60690986817697534]
6 [0.63288334573057559]
7 [0.61963583669298239]
8 [0.60206151939136532]
9 [0.60206151939136521]
```

```
10 [0.563922061919159l7]
11 [0.60206151939136532]
12 [0.57831316730290838]
13 [0.60206151939136532]
14 [0.0]
15 [0.60206151939136476]
16 [0.0]
17 [0.56392206191915917]
```

Apparently principal component 2 best explains time in this dataset.

7 Performing batch PCA interpretation

One of the core advantages of the matplotlib based version of the interface is availability of a batch processing mode.

```
pcaginsu.generateResults(['em', 'cho', 'orfs'], ['times'])
```

This will end up creating a all of the various plots and tables in your current directory for all of the principal components analyzed when creating the initial pcaGinsu object.

8 Further Information

8.1 Publication: Mining Gene Expression Data by Interpreting Principal Components

The web site http://woldlab.caltech.edu/publications/pca-bmc-2005 contains additional information including the supplemental materials and links to the publication itself.

8.2 The CompClust Python Package

CompClust is a Python package written using the pyMLX and IPlot APIs. It provides software tools to explore and quantify relationships between clustering results. Its development has been largely built around requirements for microarray data analysis, but can be easily used for other types of biological array data and that of other scientific domains.

If your interested in learning more about what you can do with the CompClust Python Package, check out the tutorials in the next section. When using the CompClust Python package the analysis possibilities are only limited by your imagination ... Well, that and CPU power and RAM, your understanding of Python and the CompClust Python package. Well, at least you don't have to be limited by what the GUI can do.

8.3 Other CompClust Tutorials

The following tutorials can be found at http://woldlab.caltech.edu/compClust/.

8.3.1 CompClustTk Manual and Tutorial

"The CompClustTk Manual and Tutorial", written by Brandon King, contains general introductory information for CompClust as well as specific information on how to use CompClustTk.

8.3.2 A First Tutorial on the MLX schema

"A First Tutorial on the MLX schema", written by Lucas Scharenbroich, covers the MLX schema. One should read this if one wanted to explore the full power of compClust using python.

8.3.3 A Quick Start Guide to Microarray Analysis using CompClust

"A Quick Start Guide to Microarray Analysis using CompClust", written by Christopher Hart, covers how to use the Python CompClust environment to do microarray analysis. It may give one a better understanding of the IPlot tools (Trajectory Summary, Confusion Matrices, etc.). It will also teach one how to use some of the more advanced features of CompClust which haven't been exposed to CompClustTk and CompClustWeb.

9 Acknowledgements

References

Cho, R. J., Campbell, M. J., Winzeler, E. A., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T. G., Gabrielian, A. E., Landsman, D., Lockhart, D. J., and Davis, R. W. (1998). A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol Cell*, 2(1):65–73. (eng).

Hart, C. E., Sharenbroich, L., Bornstein, B. J., Trout, D., King, B., Mjolsness, E., and Wold, B. J. (2005). A mathematical and computational framework for quantitative comparison and integration of large-scale gene expression data. *Nucleic Acids Research*, 33(8):2580–2594.

Roden, J., King, B., Trout, D., Wold, B., and Hart, C. E. (2005). Mining gene expression data by interpreting principal components. *BMC Bionformatics*.