ACM Data Camp 2014

Principal Component Analysis with Applications in R and Python

Irina Kukuyeva, Ph.D.
ikukuyeva@myvps.org

October 25, 2014

Outline

Course Prerequisite

Introductory class in statistics or linear algebra.

Software Prerequisites

- Working knowledge of `R` and/or `Python`.
- Completed installation of either set of packages:

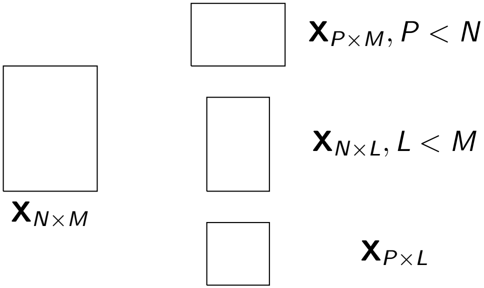
R

```r
library(lattice)
library(blockcluster)   # optional
```

Python

```python
import matplotlib.pyplot as plt
import numpy  as np
import os
import pandas as pd
import scipy
from   sklearn.decomposition import PCA
```

What is Dimension Reduction?


$\mathbf{X}_{N\times M}$
$\mathbf{X}_{P\times M}, P < N$
$\mathbf{X}_{N\times L}, L < M$
$\mathbf{X}_{P\times L}$

Why use a Dimension Reduction Method?

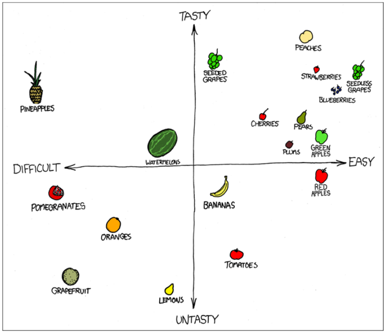
- I have too many variables... (to model, visualize, etc.).
- Some of my variables are (really) noisy.
- A few of my variables are redundant.
- I want to uncover patterns in the data.

When to use a Dimension Reduction Method?

- Standalone analysis
- Preprocessing for:
  - Regression
  - Clustering
  - etc.

Types of Dimension Reduction Methods

1. Directly observed data-generating process(es).
    - Reduce dimensionality to remove noisy and/or multicollinear variables.
2. Indirectly observed data-generating process(es).
    - Reduce dimensionality to uncover latent variables and reduce noise.



Assumption of PCA

1. Observed variables are linear combinations of unobserved *uncorrelated* variables (called *Principal Components*).
2. Variable means and the variance-covariance matrix capture all of the information in the data set.
3. The variances of variables are similar[a].
   - e.g. Perform PCA on the correlation matrix.
4. Principal Components are unique up to sign and permutation [4].

[a]If the variances of the variables are not similar, transform the variables; otherwise those variables with largest variances will seem most important.

Notation for PCA

- Let $\boldsymbol{X}_{N\times M} = \begin{pmatrix} x_{11} & \dots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NM} \end{pmatrix} = \begin{pmatrix} X_1 & \dots & X_M \end{pmatrix}$ be the data matrix.
- Let $\boldsymbol{\Sigma}$ be the full rank, sample variance-covariance matrix of $\boldsymbol{X}$ [4][1].
- Let $\boldsymbol{Y}$ denote the principal components:

$$Y_1 = \boldsymbol{X}V_1 = v_{11}X_1 + v_{21}X_2 + \dots + v_{M1}X_M$$

$$Y_2 = \boldsymbol{X}V_2 = v_{12}X_1 + v_{22}X_2 + \dots + v_{M2}X_M$$

$$\vdots$$

$$Y_M = \boldsymbol{X}V_M = v_{1M}X_1 + v_{2M}X_2 + \dots + v_{MM}X_M$$

[1] Please see [4], Section 3.4 for a discussion on data sets with zero and/or equal variances.

Notation for PCA

- In matrix form, the principal components $\boldsymbol{Y}$ are defined as:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V} \tag{1}$$

- By assumption (3) of PCA,

$$Var(Y_i) = V_i^T \boldsymbol{\Sigma} V_i, \qquad i = 1, 2, \cdots, M \tag{2}$$

$$Cov(Y_i, Y_j) = V_i^T \boldsymbol{\Sigma} V_k = 0, \quad i, k = 1, 2, \cdots, M, \quad i \neq k \tag{3}$$

Intuition Behind the Derivation I

Goal of PCA

Maximize the variance explained by each of the principal components[a]:

$$\max_{v_i} Var(Y_i) = \max_{v_i} V_i^T \mathbf{\Sigma} V_i$$

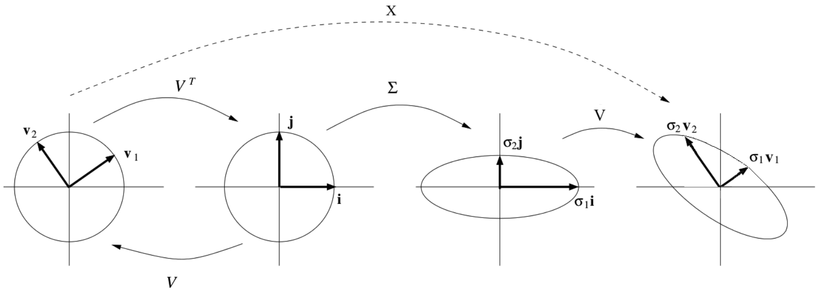
Solution 1 Set $V_i = \infty$.

Solution 2 Add a constrain to the length of V_i and solve: $\max_{v_i} Var(Y_i)$ subject to $V_i^T V_i = 1$

[a]By assumptions of PCA, $Cov(Y_i, Y_j) = 0, \; i \neq j$.

Intuition Behind the Derivation II

Visualization of PCA [1]



Derivation

Step 1. Set up the Lagrangian [4]:

$$\begin{aligned} L(\boldsymbol{V}, \Lambda) &= \boldsymbol{V}^T \boldsymbol{\Sigma} \boldsymbol{V} + \Lambda(1 - \boldsymbol{V}^T \boldsymbol{V}) \\ &= V_1^T \boldsymbol{\Sigma} V_1 + \lambda_1(1 - V_1^T V_1) + V_2^T \boldsymbol{\Sigma} V_2 + \lambda_2(1 - V_2^T V_2) + \ldots \end{aligned} \tag{4}$$

Step 2. Differentiate Equation 4 with respect to each i^{th} variable of interest [4]:

$$\frac{\partial L(\boldsymbol{V}, \Lambda)}{\partial V_i} = 2\boldsymbol{\Sigma} V_i - 2\lambda_i V_i = 0 \tag{5}$$

Derivation

Step 3. Rearrange the terms of Equation 5 [4]:

$$\begin{aligned}(\boldsymbol{\Sigma} - \lambda_1 \boldsymbol{I}_M)V_1 &= 0 \\ &\vdots \\ (\boldsymbol{\Sigma} - \lambda_M \boldsymbol{I}_M)V_M &= 0,\end{aligned} \tag{6}$$

where $\boldsymbol{I}_M$ is the $M \times M$ identity matrix.

Derivation

Step 4. Find the solution to the maximization problem [4].
We know that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_M \geq 0$.

1. A one-component solution is *attained* for λ_1 (the largest eigenvalue), with the corresponding eigenvector V_1.

$$\vdots$$

2. An L-component solution is *attained* for $\lambda_1, \cdots \lambda_L$ (for $L, L < M$, largest eigenvalues), with the corresponding eigenvectors $V_1, \cdots, V_L$.

How Many Components to Extract?

Goal: Choose a small set L of principal components ($L < M$):

Solution: a small set of components that describe the original data set without *too much* loss of information [4].

Interpretation of λ_i:

- All the components explain 100% of the variation in the original data set $\boldsymbol{X}$.
- Each component Y_i explains (an additional) $\frac{\lambda_i}{\sum_{j=1}^{M} \lambda_j} \times 100\%$ of variance in original data set $\boldsymbol{X}$.

How Many Components to Extract?

Most commonly-employed *ad-hoc* rules

- *A priori*, determine a (cumulative) percent of total variance in the data set that L components should explain [4].
  - Usually between 70% and 90%.
- Look for an elbow in a scree plot; its occurrence is the number of components we should use [4].
- Keep components that have a contribution of $> 5\%$.
- Retain components that explain a $> \frac{1}{\text{number of variables in the data}} \times 100\%$ of the variance in the original data set.
- Employ cross-validation to calculate the PREdiction Sum of Squares (PRESS) statistic [4].

Evaluating Model Fit: Residual Analysis

Suppose we keep L ($L < M$) components, then

- $\boldsymbol{V}_0$ is an $M \times L$ matrix with L eigenvectors corresponding the the L largest eigenvalues;
- $\boldsymbol{Y}_0$ is an $N \times L$ matrix with L principal components;
- residuals are:

$$\begin{aligned}\boldsymbol{\Psi} &= \boldsymbol{X} - \hat{\boldsymbol{X}} \\ &= \boldsymbol{X} - \boldsymbol{Y}_0 \boldsymbol{V}_0^T \\ &= \boldsymbol{X} - (\boldsymbol{X}\boldsymbol{V}_0)\,\boldsymbol{V}_0^T\end{aligned} \tag{7}$$

Interpreting the Components

- Analyze the weights of the raw variables V_i used to form principal components to explain the variation in the data:
  - Which variables carry the most weight?
  - Which variables are equally weighted?
  - Which variables offset others (i.e. equally weighted but of opposite signs)?

Example 1: European Food Consumption

Data Set Overview

- 1970s food consumption of 25 European countries [11], [8]
- Consumption categorized into 9 food groups [8]:
  - red meat
  - white meat
  - eggs
  - milk
  - fish
  - cereal
  - starch
  - nuts
  - fruits and vegetables
- Units are in 0.1 grams of protein, per person, per day [10].

Example 1: European Food Consumption I

Step 1: Read the Dataset into R

```
url = 'http://www.stat.ucla.edu/~rgould/120bs06/protein.txt'
```

R

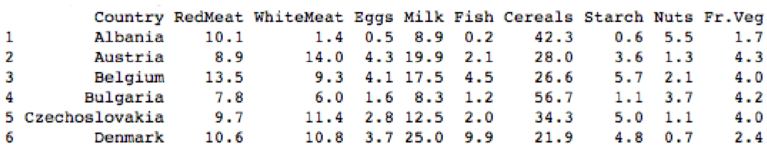
```r
data.orig   <- read.table(url,
                          header=TRUE,
                          sep="\t")

head(data.orig)

# For PCA analysis, keep all the variables
# except the first column with country names:
data         <- data.orig[, -1]
```

Python

```python
data_orig    = pd.read_csv(url,
                           sep='\t')

data_orig.head()

# For PCA analysis, keep all the variables
# except the first column with country names:
columns_keep = data_orig.columns.values.tolist()[1:]
data         = data_orig[columns_keep]
```

Example 1: European Food Consumption II

Step 1: Read the Dataset into R



```
         Country RedMeat WhiteMeat Eggs Milk Fish Cereals Starch Nuts Fr.Veg
1        Albania    10.1       1.4  0.5  8.9  0.2    42.3    0.6  5.5    1.7
2        Austria     8.9      14.0  4.3 19.9  2.1    28.0    3.6  1.3    4.3
3        Belgium    13.5       9.3  4.1 17.5  4.5    26.6    5.7  2.1    4.0
4       Bulgaria     7.8       6.0  1.6  8.3  1.2    56.7    1.1  3.7    4.2
5 Czechoslovakia     9.7      11.4  2.8 12.5  2.0    34.3    5.0  1.1    4.0
6        Denmark    10.6      10.8  3.7 25.0  9.9    21.9    4.8  0.7    2.4
```

Figure : First 6 rows of the 'Protein' data set.

Example 1: European Food Consumption I

Step 2a: Perform Preliminary Diagnostics Graphically

R

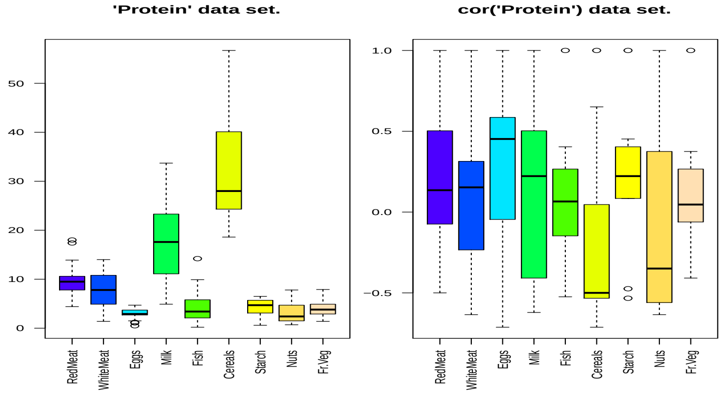
```
par(las=2,
    cex.axis=0.8,
    mfrow=c(1,2),
    mai=c(1,0.5,1,0.1)
    )
boxplot(data,
        main="'Protein' data set.",
        col=topo.colors(ncol(data))
        )
boxplot(data.frame(cor(data)),
        main="cor('Protein') data set.",
        col=topo.colors(ncol(data))
        )
```

Python

```
def boxplot(dataset, ax, colors):
  box = ax.boxplot(dataset,
                   patch_artist=True)
  for bx, color in zip(box['boxes'], colors):
      bx.set_facecolor(color)
  for median in box['medians']:
      median.set_linewidth(2)
  ax.set_xticklabels(data.columns.values.tolist(),
                     rotation=90,
                     size=8)

colors = ["#4c00ff", "#004cff", "#00e5ff",
          "#00ff4d", "#4dff00", "#e6ff00",
          "#ffff00", "#ffde59", "#ffe0b3"]

fig = plt.figure(1, figsize=(12,6))
ax1 = fig.add_subplot(121)
boxplot(data.values, ax1, colors)
ax2 = fig.add_subplot(122)
boxplot(data.corr().values, ax2, colors)
fig.show()
```

Example 1: European Food Consumption II

Step 2a: Perform Preliminary Diagnostics Graphically



Example 1: European Food Consumption

Step 2b: Perform Preliminary Diagnostics (more) Rigorously

R

```r
bartlett.test(data.frame(cor(data)))


##
##  Bartlett test of homogeneity of variances
##
## data:  data.frame(cor(data))
## Bartlett's K-squared = 2.879, df = 8, p-value = 0.9417
```

Python

```python
tmp = np.array(data.corr())
scipy.stats.bartlett(tmp[0], tmp[1], tmp[2],
                     tmp[3], tmp[4], tmp[5],
                     tmp[6], tmp[7], tmp[8])

## (2.878558924474143, 0.9417457322949315)
```

Example 1: European Food Consumption

Step 3a: Perform PCA

R

```
out.cor     <- princomp(data, cor=TRUE)

lambda_perc <- out.cor$sdev^2/sum(out.cor$sdev^2)
V           <- out.cor$loadings
Y           <- cor(data) %*% V     #  Y = XV
PC          <- out.cor$scores      # PC = (standardized dataset)V
### Compare with ?prcomp and ?svd
```

Python

```
data_std    = data/data.std()

out_cor     = PCA().fit(data_std)

lambda_perc = out_cor.explained_variance_ratio_
V           = pd.DataFrame(out_cor.components_.T)
Y           = pd.DataFrame(np.dot(data.corr(), V))  # Y = XV
PC          = pd.DataFrame(out_cor.fit_transform(data_std))
### Compare with 'scipy.linalg',
###              'numpy.linalg' and
###              'matplotlib.mlab.PCA'
```

Example 1: European Food Consumption

Step 3b: Check the Math of PCA

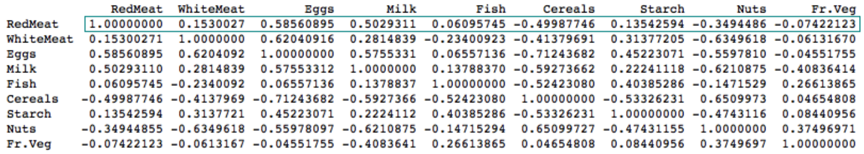

```
             RedMeat   WhiteMeat        Eggs       Milk        Fish     Cereals      Starch       Nuts      Fr.Veg
RedMeat   1.00000000  0.1530027  0.58560895  0.5029311  0.06095745 -0.49987746  0.13542594 -0.3494486 -0.07422123
WhiteMeat  0.15300271  1.0000000  0.62040916  0.2814839 -0.23400923 -0.41379691  0.31377205 -0.6349618 -0.06131670
Eggs       0.58560895  0.6204092  1.00000000  0.5755331  0.06557136 -0.71243682  0.45223071 -0.5597810 -0.04551755
Milk       0.50293110  0.2814839  0.57553312  1.0000000  0.13788370 -0.59273662  0.22241118 -0.6210875 -0.40836414
Fish       0.06095745 -0.2340092  0.06557136  0.1378837  1.00000000 -0.52423080  0.40385286 -0.1471529  0.26613865
Cereals   -0.49987746 -0.4137969 -0.71243682 -0.5927366 -0.52423080  1.00000000 -0.53326231  0.6509973  0.04654808
Starch     0.13542594  0.3137721  0.45223071  0.2224112  0.40385286 -0.53326231  1.00000000 -0.4743116  0.08440956
Nuts      -0.34944855 -0.6349618 -0.55978097 -0.6210875 -0.14715294  0.65099727 -0.47431155  1.0000000  0.37496971
Fr.Veg    -0.07422123 -0.0613167 -0.04551755 -0.4083641  0.26613865  0.04654808  0.08440956  0.3749697  1.00000000
```

Figure : Correlation Matrix of the 'Protein' data set.

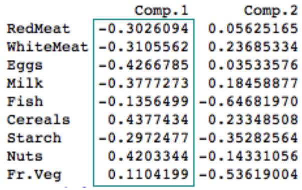

```
              Comp.1      Comp.2
RedMeat   -0.3026094  0.05625165
WhiteMeat -0.3105562  0.23685334
Eggs      -0.4266785  0.03533576
Milk      -0.3777273  0.18458877
Fish      -0.1356499 -0.64681970
Cereals    0.4377434  0.23348508
Starch    -0.2972477 -0.35282564
Nuts       0.4203344 -0.14331056
Fr.Veg     0.1104199 -0.53619004
```

Figure : First two loadings.

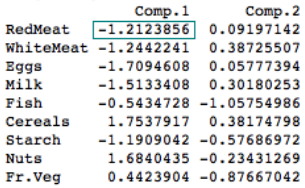

```
              Comp.1      Comp.2
RedMeat   -1.2123856  0.09197142
WhiteMeat -1.2442241  0.38725507
Eggs      -1.7094608  0.05777394
Milk      -1.5133408  0.30180253
Fish      -0.5434728 -1.05754986
Cereals    1.7537917  0.38174798
Starch    -1.1909042 -0.57686972
Nuts       1.6840435 -0.23431269
Fr.Veg     0.4423904 -0.87667042
```

Figure : First two Principal Components.

Example 1: European Food Consumption I

Step 4: Determine How Many Components to Keep

R

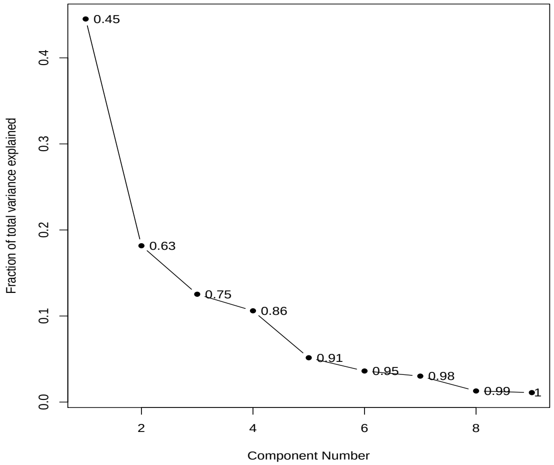
```
par( mai=c(1,1,0.1,0.1) )
plot(lambda_perc,
     type="b",
     xlab="Component Number",
     ylab="Percent total variance explained",
     pch=16
     )
text(x=1:ncol(cor(data)),
     y=lambda_perc,
     labels=round(cumsum(lambda_perc), 2),
     adj=-0.3
     )
```

Python

```
plt.plot(range(out_cor.n_components_),
         lambda_perc,
        'o-'
         )
plt.xlabel("Component Number")
plt.ylabel("Percent total variance explained")

for i in range(out_cor.n_components_):
  plt.text(x=i,
           y=lambda_perc[i],
           s=str(round(lambda_perc[i], 2))
          )

plt.show()
```

Example 1: European Food Consumption II

Step 4: Determine How Many Components to Keep

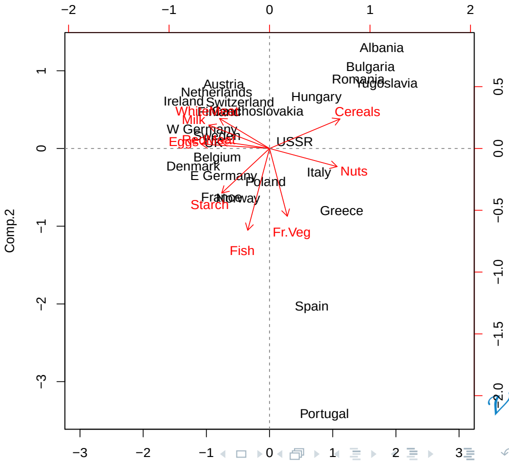


Example 1: European Food Consumption I

Step 5: Interpret the Results

R

```
biplot(out.cor,
       pc.biplot=TRUE,
       xlabs=data.orig$Country,
       xlim=c(-3, 3)
       )
abline(h=0,
       lty=2,
       col="grey50"
       )
abline(v=0,
       lty=2,
       col="grey50"
       )
```



Example 1: European Food Consumption II

Step 5: Interpret the Results

Python

```
plt.plot(PC[[0]],
        -PC[[1]],
        'ko'
        )
plt.xlabel("PC1")
plt.ylabel("-PC2")
plt.axvline(x=0, color='0.75')
plt.axhline(y=0, color='0.75')

# Add components' labels:
for i in range(len(PC)):
  plt.text(x=PC[0][i],
           y=-PC[1][i],
           s=data_orig['Country'][i]
          )
```

```
# Add loading directions:
for i in range(out_cor.n_components_):
  plt.arrow(x=0,
            y=0,
            dx=1.5*V[0][i],
            dy=-1.5*V[1][i],
            color='r',
            head_width=0.1
            )

# Add loading labels:
col_names = data.columns.values.tolist()
for i in range(out_cor.n_components_):
  plt.text(x=2*V[0][i],
           y=-2*V[1][i],
           s=col_names[i],
           color='r'
           )

plt.show()
```

Example 1: European Food Consumption

Step 5b: Check the Math for Portugal

1. Original values:

```
##    RedMeat WhiteMeat Eggs Milk Fish Cereals Starch Nuts Fr.Veg
## 17     6.2       3.7  1.1  4.9 14.2      27    5.9  4.7    7.9
```

2. Standardized values:

```
##   RedMeat WhiteMeat      Eggs      Milk      Fish   Cereals    Starch
##   -1.0839   -1.1359   -1.6428   -1.7187    2.9143   -0.4782    0.9938
##      Nuts    Fr.Veg
##    0.8199    2.0866
```

3. Weights of variables forming first component:

```
##   RedMeat WhiteMeat      Eggs      Milk      Fish   Cereals    Starch
##   -0.3026   -0.3106   -0.4267   -0.3777   -0.1356    0.4377   -0.2972
##      Nuts    Fr.Veg
##    0.4203    0.1104
```

4. Multiply (2) by (3) and compare with Principal Component 1 (1.741) for Portugal.

Example 1: European Food Consumption I

Step 6: Analyze Residuals

$$\text{Residuals} = \boldsymbol{X} - (\boldsymbol{X}\boldsymbol{V}_0)\,\boldsymbol{V}_0^T$$

R

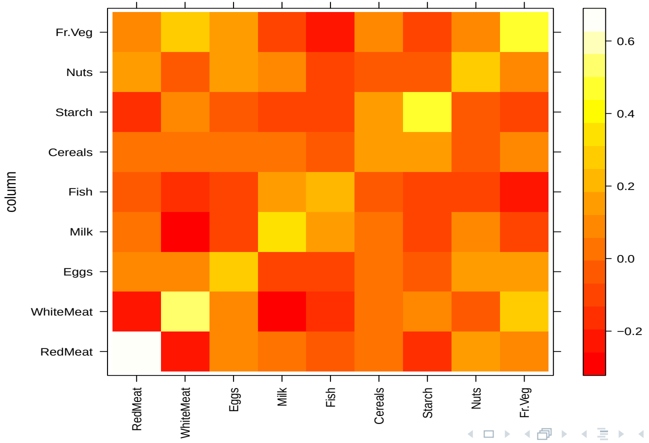
```r
X_hat <- cor(data) -
        (cor(data) %*% V[, 1:2]) %*% t(V[, 1:2])

n_obs <- nrow(X_hat) * ncol(X_hat)

levelplot(X_hat,
 col.regions=heat.colors( n_obs ),
 scales=list( x=list(rot = 90) )
 )
```

Python

```python
X_hat = data.corr() -
  np.dot(
         np.dot(
                data.corr(),
                pd.DataFrame([V[0], -V[1]]).T
                ),
         pd.DataFrame([V[0], -V[1]])
        )

fig = plt.figure(1, figsize=(12,6))
ax  = fig.add_subplot(111)
heatmap = plt.imshow(X_hat, interpolation='none')
heatmap.set_cmap('hot')
ax.set_xticks(range(out_cor.n_components_))
ax.set_yticks(range(out_cor.n_components_))
ax.set_xticklabels(data.columns.values.tolist(),
                   rotation=90,
                   size=10)
ax.set_yticklabels(data.columns.values.tolist(),
                   size=10)
plt.colorbar()
plt.show()
```

Example 1: European Food Consumption II

Step 6: Analyze Residuals



Example 2: Pasadena Meetup.com Groups I

Data Set Creation

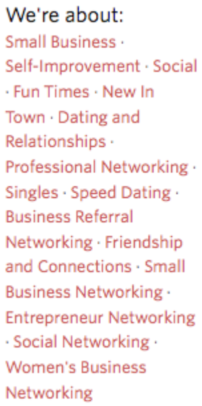
1. Scrape 200 groups from Meetup.com, meeting within 25 miles of Pasadena (in Sept 2014)
2. Extract keywords from each group
   - remove quotes and dashes, convert words to lower case
3. Perform word stemming on the key words from each group
   - 'network' and 'networking' are mapped to one word
4. Create a matrix of word counts of keywords from each group (with K unique words)
5. Compute the distance between each Meetup.com group:

$$\text{Distance b/w meetups } i \text{ and } j = \frac{\sum_{k=1}^{K} |count_{k,i} - counts_{k,j}|}{K}$$

Example 2: Pasadena Meetup.com Groups II

Data Set Creation

Processing keywords of one meetup group:



We're about:
Small Business · Self-Improvement · Social · Fun Times · New In Town · Dating and Relationships · Professional Networking · Singles · Speed Dating · Business Referral Networking · Friendship and Connections · Small Business Networking · Entrepreneur Networking · Social Networking · Women's Business Networking

word counts	words
6	network
4	busi
2	and
2	date
2	small
2	social
1	connect
1	entrepreneur
1	friendship
1	fun
1	in
1	new
1	profession
1	referr
1	relationship
1	selfimprov
1	singl
1	speed
1	time
1	town
1	women

Example 2: Pasadena Meetup.com Groups

Step 1: Read the Dataset into R

R

```
dist_mat <- read.csv("../Data/meetup_dist_mat.csv",
                     row.names=1
                    )
```

Python

```
os.chdir("../Data")
dist_mat = pd.read_csv('meetup_dist_mat.csv',
                       sep=',',
                       index_col=0
                      )
```

Example 2: Pasadena Meetup.com Groups

Step 2: Perform PCA

R

```
out.cor     <- princomp(dist_mat, cor=TRUE)

lambda_perc <- out.cor$sdev^2/sum(out.cor$sdev^2)
V           <- out.cor$loadings
Y           <- cor(dist_mat) %*% V     #  Y= XV
PC          <- out.cor$scores          # PC = (standardized dataset)V
```

Python

```
data_std = dist_mat/dist_mat.std()
out_cor  = PCA()          # keep all components
out_cor.fit(data_std)     # fit PCA

lambda_perc = out_cor.explained_variance_ratio_
V           = pd.DataFrame(out_cor.components_.T)
Y           = pd.DataFrame(np.dot(dist_mat.corr(), V))
PC          = pd.DataFrame(out_cor.fit_transform(data_std))
```

Example 2: Pasadena Meetup.com Groups I

Step 3: Determine How Many Components to Keep

R

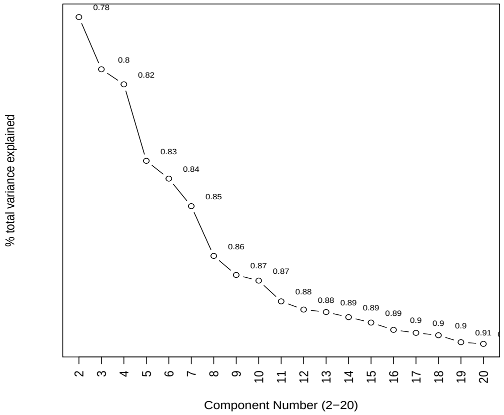
```r
lambda_perc_cumsum <- cumsum(lambda_perc)

plot(lambda_perc[2:20],
     type="b",
     xlab="Component Number (2-20)",
     ylab="% total variance explained",
     axes=FALSE)
axis(1,
     at=1:20,
     lab=2:21,
     las=2)
box()
text(x=2:20,
     y=lambda_perc[2:20],
     labels=round(
              lambda_perc_cumsum[2:20],
              digits=2
              ),
     adj=-0.1,
     pos=3,
     cex=0.7
     )
```

Python

```python
lambda_perc_cumsum = np.cumsum(lambda_perc)

fig = plt.figure()
ax = fig.add_subplot(111)

plt.plot(range(1, 21),
         lambda_perc[1:21], 'ko-')
plt.xticks(np.array(range(1, 21)))
ax.set_xticklabels(range(1, 21), rotation=90)
ax.set_yticklabels([])
ax.set_xlabel("Component Number (2 through 20)")
ax.set_ylabel("Percent total variance explained")

for i in range(1, 21):
  ax.text(x=i-0.2,
          y=lambda_perc[i],
          s=str(round(lambda_perc_cumsum[i], 2)),
          size='small'
          )

fig.show()
```

Example 2: Pasadena Meetup.com Groups II

Step 3: Determine How Many Components to Keep



Example 2: Pasadena Meetup.com Groups I

Step 4: Interpret the Results

R

```
### --- Look at relationships between weights
V_first4   <- V[, 1:4]
### --- Look at relationships between components
Y_first4   <- Y[, 1:4]
```

Python

```
### --- Look at relationships between weights
V_first4 = pd.DataFrame([-V[0], V[1], -V[2], -V[3]]).T

### --- Look at relationships between components
Y_first4 = pd.DataFrame([-Y[0], Y[1], -Y[2], -Y[3]]).T
```

Please see 'First_four_loadings.PDF' for a table of Meetup groups with $|loadings| \geq 0.1$.

Example 2: Pasadena Meetup.com Groups II

Step 4: Interpret the Results

First four loadings can be interpreted as follows:

Component 1	None
Component 2	social/networking (pos weights) versus fitness/yoga (neg weights)
Component 3	personal development (pos weights) versus kids and pets (neg weights)
Component 4	networking/pets (pos weights) versus moms (neg weights)

Example 2: Pasadena Meetup.com Groups I

Step 5: Alternative Analysis via Co-Clustering

R

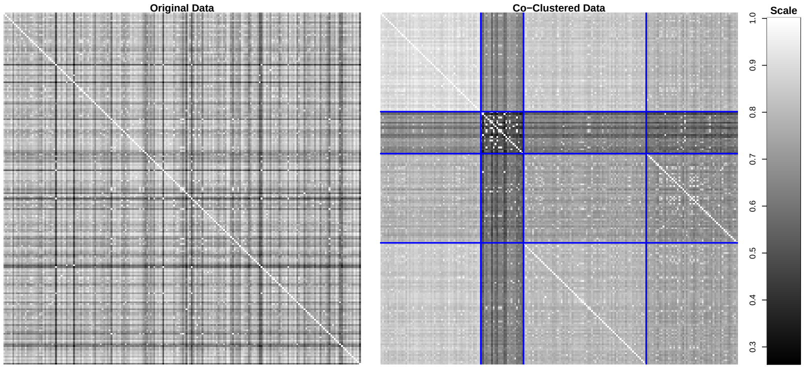
```
# 'nbcocluster = 4 clusters for rows and 4 for columns
# 'model' = equal-sized clusters with unequal variance
out <- cocluster(cor(dist_mat),
                datatype="continuous",
                nbcocluster=c(4,4),
                model="pi_rho_sigma2kl"
                )


## Co-Clustering successfully terminated!


plot(out)
```

Python

```
# No packages/functions exist for
# block clustering with Python
```

Example 2: Pasadena Meetup.com Groups II

Step 5: Alternative Analysis via Co-Clustering



Recall: Assumptions of PCA

1. *Principal Components* are uncorrelated with each other.
2. Observed variables are combinations of *Principal Components*).
   a. Observed variables are unconstrained.
   b. Observed variables are continuous.
   c. Observed variables are linear combinations of *Principal Components*).
3. Observed variable means and the variance-covariance matrix capture all of the information in the data set.
4. Observed variables are best represented by an $N \times M$ matrix.
5. Principal Components are unique up to sign and permutation.

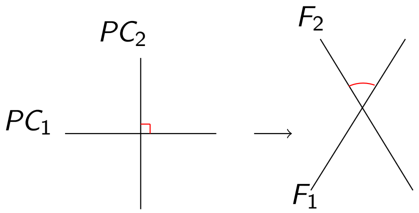
Relax Assumption (1)

Factor Analysis

Overview: Factor Analysis

Rotates Principal Components by an angle to derive more interpretable factors.

Visualization of method



Implementation

R	`varimax`
Python	`sklearn.decomposition.FactorAnalysis`

Relax Assumption (2a)

Non-negative Matrix Factorization

Overview: Non-negative Matrix Factorization

Constrains weights/loadings to be positive and finds Principal Components as outlined above.

Implementation

`R`	library(NMF)
`Python`	NIMFA

Relax Assumption (2b)

Correspondence Analysis

Overview: Correspondence Analysis

Application of PCA to discrete variables [4].

Implementation

`R`	library(ca)
`Python`	mca

Relax Assumptions 2(b) and (c)

Nonlinear Principal Component Analysis

Overview: Nonlinear Principal Component Analysis

Generalize PCA to simultaneously optimize over nonlinear transformations of variables and PCA estimation [2], [7].

Implementation

`R`	library(homals) library(kernlab), library(autoencoder) [9], etc.
`Python`	help(sklearn.decomposition.KernelPCA), pynnet, etc.

Relax Assumption (1) and (3)

Independent Component Analysis

Overview: Independent Component Analysis (ICA)

Rotates Principal Components to be independent (i.e. nonlinearly uncorrelated) [3].

- Principal components from multivariate normally distributed data sets are independent.
- Independence can be quantified by:
  - Mutual Information
  - Cumulants – diagonal for independent variables

Implementation

`R`	library(fastICA), library(PearsonICA)
`Python`	sklearn.decomposition.FastICA

Relax Assumption (4) and partially (5) I

Tucker-*n*/CANDECOMP

Overview: Tucker-*n* and CANDECOMP

Generalize PCA to data sets with *n* modes, satisfying either [5]:

- uniqueness up to permutation via Tucker-3; or
- uniqueness up to scale via CANDECOMP.

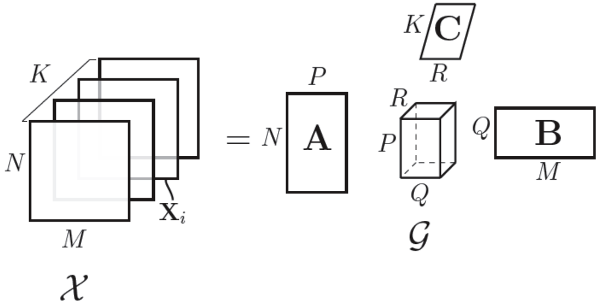
Implementation

`R`	library(ThreeWay)
`Python`	PyTensor

Relax Assumption (4) and partially (5) II

Tucker-*n*/CANDECOMP

Visualization of method



Relax Assumption (1), partially (3), and (4) I

Array Independent Component Analysis

Overview: Array Independent Component Analysis [6]

Generalizes of ICA to data sets with n modes, accounting for relationships between modes.

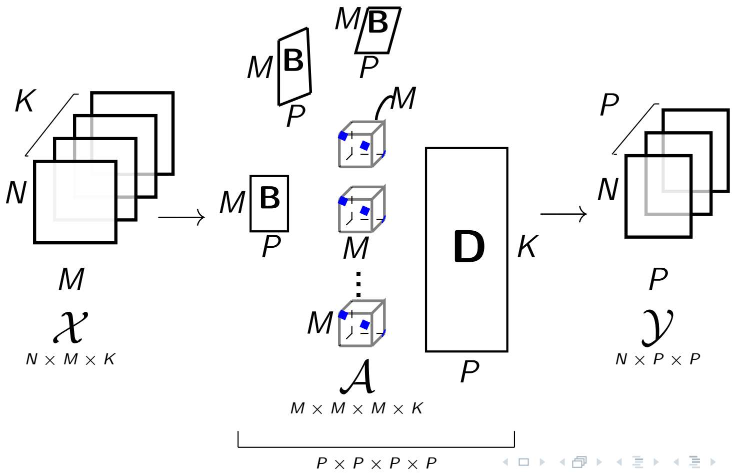
- Relationships are quantified by cumulants – diagonal for independent variables.

Implementation

R	Coming soon.

Relax Assumption (1), partially (3), and (4) II

Array Independent Component Analysis



Spectral Clustering

Overview: Spectral Clustering [12]

1. Quantifies similarities between the variables of the original data set in a matrix.
2. Reduces the dimensionality of the similarity matrix.
3. Clusters the smaller data set, from Step (2).

Implementation

R	library(kernlab)
Python	sklearn.cluster.bicluster.SpectralCoclustering

Final Remarks

Python code has been tested with 2.7.5.

R code has been tested with 3.0.3.

Tutorial is available here:
`https://sites.google.com/site/ikukuyeva/`

We are hiring (soon)!

References I

[1] cMinor (http://math.stackexchange.com/users/7265/cminor). Visualization of singular value decomposition of a symmetric matrix. (Mathematics), Nov 2012.

[2] Jan de Leeuw. Nonlinear principal component analysis and related techniques. Technical report, UCLA, 2005.

[3] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis.* Wiley Series, 2001.

[4] I. T. Jolliffe. *Principal Component Analysis.* Springer Series in Statistics. Springer, 2002.

[5] Pieter M. Kroonenberg. *Applied Multiway Data Analysis.* John Wiley and Sons, 2008.

References II

[6] Irina Kukuyeva. *Array Independent Component Analysis with Application to Remote Sensing*. PhD thesis, UCLA, 2012.

[7] M. Linting, J.J. Meulman, P.J.F. Groenen, and A.J. Van der Kooij. *Nonlinear Principal Components Analysis: Nonlinear Principal Component Analysis: Introduction and Applicationanalysis: Introduction and application*. American Psychological Association., 2007.

[8] SAS/STAT(R) 9.22 User's Guide. Creating a distance matrix as input for a subsequent cluster analysis.

[9] Matthias Scholz. Nonlinear pca.

[10] StatSoft Inc. Example 3: Protein consumption in europe, 2012.

References III

[11] Universidade Federal de Goiás. Datafile Name: Protein.

[12] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.