**Design of a new PCA algorithm**

**Objective:** write a standardized PCA algorithm (in Python) that can be applied to most data sets.

**Desired outputs:**

This new algorithm should allow its user to collect 3 major outputs (2 tables, 1 reduced data set).

***Table 1:*** Contribution of each variable into each Principal Component (PC)

**Example:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PC 1 | PC 2 | … | PC n |
| variable 1 |  |  |  |  |
| variable 2 |  |  |  |  |
| … |  |  |  |  |
| variable m |  |  |  |  |

Table 1 can help determine how to group variables and generate indexes.

***Table 2:*** Contribution of each PC into the variance of the data set

**Example:**

|  |  |  |
| --- | --- | --- |
|  | Eigen value | Accumulated variance explained |
| PC 1 | 4.2 | 62% |
| PC 2 | 1.3 | 68% |
| … |  | … |
| PC n | 0.1 | 100% |

Table 1 can help determine how to group variables and generate indexes.

**Reduced data set:** Original data set projected into the PC space to reduce the dimensionality of the data. The user should be able to select the size of the reduced data set by selecting how many PC she/he intends to keep based on how much of the variance she/he would like to explain.

# ---------------------------------------------------------------------------- #

# CODE DESCRIPTION #

# ---------------------------------------------------------------------------- #

'''

PCA\_dim\_reduction(data , variance)

Computes the eigenvalues, eigenvectors and weights of a data set using

Singular Value Decomposition (SVD), then project the original data set into

a Principal Components (PCs) space. Based on the defined amount of variance

explained, the projected data will be cropped by only keeping the PCs that

explain the desired amount of variance.

Author: Benjamin Dourthe

Created: September 15th, 2018

Last update: September 27th, 2018

Input:

- data: m x n matrix containing data, where m = trials/subjects/observations

and n = number of variables

- variance: percentage of variance desired (between 1 and 100)

Output:

- eigen\_values: arranged as a column vector

- eigen\_vectors: arranged as rows

- weights: projections of the trials onto the eigenvectors (each column

corresponds to one trial/subject/observation)

- variance\_explained\_acc: vector showing the accumulative contribution of

each PC to the total variance (last row should be 100)

- projected\_data: original data set projected into PC space (only includes

the PCs explaining the desired amount of variance)

Dependencies:

Libraries:

- numpy as np

Functions:

- None

Files:

- None

'''

# ---------------------------------------------------------------------------- #

# FUNCTION #

# ---------------------------------------------------------------------------- #

# LIBRARIES

**import** **numpy** **as** **np**

**def** PCA\_algorithm(data , variance):

# STEP 1 - Singular Value Decomposition (SVD) -------------------------------- #

# Centralize data around zero by subtracting the mean

# (leads to a data set with zero mean)

zero\_mean\_data = data - np.mean(data , 0)

# Calculate number of observations and variables

[r , c] = np.shape(zero\_mean\_data)

num\_observations = r

num\_variables = c

# SVD: computes eigenvalues and eigenvectors

**if** r >= c:

sim\_data = zip(\*zero\_mean\_data) \* zero\_mean\_data / num\_observations

U , eigen\_values , eigen\_vectors = np.linalg.svd(sim\_data , full\_matrices = False)

**del** U

eigen\_values = np.diagonal(eigen\_values)

eigen\_vectors = zip(\*eigen\_vectors)

**else:**

sim\_data = zero\_mean\_data \* zip(\*zero\_mean\_data) / num\_observations

U , eigen\_values , eigen\_vectors = np.linalg.svd(sim\_data)

**del** U

eigen\_values = np.diagonal(eigen\_values)

eigen\_vectors = zip(\*eigen\_vectors)

neg\_sqrt\_eigen\_values = np.diagonal(-(np.sqrt(eigen\_values))\*\*-1)

eigen\_vectors = zip(\*zero\_mean\_data) \* eigen\_vectors \* neg\_sqrt\_eigen\_values / np.sqrt(num\_observations)

eigen\_vectors = zip(\*eigen\_vectors)

# STEP 2 - Residual Calculation ---------------------------------------------- #

# Project mean on eigenvectors to obtain the weights that describe the mean

mean\_data\_proj = eigen\_vectors \* zip(\*zero\_mean\_data)

# Reconstruct mean from reduced number of eigenvectors

recon\_data\_mean = zip(\*mean\_data\_proj) \* eigen\_vectors

# Calculation of the residual between real mean and reconstruction

data\_residual\_mean = zero\_mean\_data - recon\_data\_mean

# Normalize the residual

data\_residual\_mean = data\_residual\_mean / np.linalg.norm(data\_residual\_mean)

# Combine residual with original eigenvectors

eigen\_vectors = [data\_residual\_mean , eigen\_vectors]

# STEP 3 - Weights -Calculation ---------------------------------------------- #

# Calculate the weights

# Note: Each row represents the weights of the observations of the corresponding eigenvector

weights = eigen\_vectors \* zip(\*data)

# Note: To reconstruct the original data: data\_reconstructed = zip(\*weights) \* eigen\_vectors

# STEP 4 - Positivity -------------------------------------------------------- #

# Change all negative weights and eigenvectors to positive

mean\_weights = np.mean(weights , 1)

**for** i **in** range(1 , np.len(mean\_weights)):

**if** mean\_weights(i) < 0:

eigen\_vectors[i , :] = -eigen\_vectors[i , :]

weights[i , :] = -weights[i , :]

# STEP 5 - Variance ---------------------------------------------------------- #

# Convert the eigenvalues into percentages to explain the contribution

# of each PC into the total variance

explained = eigen\_values \* 100 / np.sum(eigen\_values)

# Generate a matrix of accumulative contributions: used to define how many

# PCs will be kept to explain the desired amount of variance

variance\_explained\_acc = []

**for** i **in** range(1 , np.len(explained)):

variance\_explained\_acc[i] = np.sum(explained[1:i])

# Identify how many PCs are needed to reach the corresponding threshold

**for** i **in** range(1, np.len(variance\_explained\_acc)):

**if** variance\_explained\_acc[i] >= variance:

num\_PCs = i

# Generate a new data set projected into the PC space and cropped based on

# the amount of PCs required to explain the defined amount of variance

projected\_data = zip(\*weights[2:num\_PCs(1)+1 , :])

# STEP 6 - RETURN OUTPUTS ---------------------------------------------------- #

**return** eigen\_values

**return** eigen\_vectors

**return** weights

**return** variance\_explained\_acc

**return** projected\_data