Report Lab 3 – Misha Tsysin, 0033922418

# Exercise: Generating Gaussian random vectors

The required scatterplots are:

White original noise:

A graph with blue dots

Description automatically generated

Scaled noise decorrelated:

A blue dots on a white background

Description automatically generated

Rotated noise with given covariance:

A graph with blue dots

Description automatically generated

# Covariance Estimation and Whitening

The theoretical value is:

The estimated matrix is:

The two required scatter plots:

For decorrelated noise:

A blue dots on a white background

Description automatically generated

For the final whitened noise:

A graph with blue dots

Description automatically generated

The final covariance of whitened noise is:

# Eigenimages, PCA, and Data Reduction

The resulting 12 eigen images are:

A collage of images of numbers and letters

Description automatically generated

The graph of the first 10 components for the first 4 images is:

A graph of different colored lines

Description automatically generated

Finally, the resulting approximation for the letter “a” is:

A group of black and white images

Description automatically generated

The original image is (directly from the dataset):

A black letter on a white background

Description automatically generated

# Image Classification

For the classification with eigenvectors:

|  |  |
| --- | --- |
| Character | Classification |
| d | a |
| j | y |
| l | i |
| n | v |
| p | e |
| q | a |
| u | a |
| y | v |

Using diagonal elements of :

|  |  |
| --- | --- |
| Character | Classification |
| i | l |
| y | v |

Using average

|  |  |
| --- | --- |
| Character | Classification |
| g | q |
| y | v |

Using diagonal elements of the average:

|  |  |
| --- | --- |
| Character | Classification |
| f | t |
| y | v |

Using identity:

|  |  |
| --- | --- |
| Character | Classification |
| f | t |
| g | q |
| y | v |

Methods 2, 3, and 4 have similar performance with least errors.

The tradeoff is between the accuracy of the data model and the accuracy that we get after estimating. The more complex model performs poorer when it comes to inference accuracy.

# CODE:

import numpy as np

import matplotlib.pyplot as plt

from training\_data.read\_data import read\_data, display\_samples, datachar, read\_data\_test

import seaborn as sns

import pandas as pd

plt.rcParams['text.usetex'] = True

def excersise\_2(p = 2, n = 1000):

Rx = np.array([[2, -1.2],

[-1.2, 1]])

# This is essentially equivalent to generating a

# table of of gausiians each vith variance one in

# a p X n matrix

W = np.random.normal(0, 1, size = (p, n))

eigenvalues, eigenvectors = np.linalg.eig(Rx)

Lambd = np.sqrt(np.diag(eigenvalues))

X\_tild = Lambd@W

X = eigenvectors@X\_tild

# Plot data

def make\_plot(A, name):

plt.plot(A[0, :], A[1, :], '.')

plt.title(name)

plt.axis('equal')

plt.show()

make\_plot(W, r'$W$ - white noise')

make\_plot(X\_tild, r'$\tilde{X}$ -- decorrelated noise')

make\_plot(X, r'$X$ -- final noise')

Z = X - np.average(X, axis=1)[:, np.newaxis]

R\_hat = 1/(n-1) \* Z@Z.T

print(f"Original: {Rx}")

print(f"Estimate: {R\_hat}")

# Compute whitening opearion:

eigenvalues, eigenvectors = np.linalg.eig(R\_hat)

X\_hat\_tilda = eigenvectors.T @ X

W\_est = np.diag(np.power(eigenvalues, -1/2))@X\_hat\_tilda

make\_plot(W\_est, r"W -- whitened noise")

make\_plot(X\_tild, r'$\tilde{X}$ -- decorrelated noise')

Z\_W = W - np.average(W, axis=1)[:, np.newaxis]

R\_w = 1/(n-1) \* Z\_W@Z\_W.T

print(f"Estimated Rw: {R\_w}")

def excersise\_4():

X = read\_data()

print(f"Shape of X: {X.shape}")

dim, n = X.shape

mean\_image = np.average(X, axis=1)[:, np.newaxis]

X = X - mean\_image

Z = X / np.sqrt(n-1)

U, S, Vt = np.linalg.svd(Z, full\_matrices=False)

eigenvectors = U

eigenvalues = S

GY, GX = 3, 4

top12eignevectors = eigenvectors[:, :12].T.reshape(12, 64, 64)

\_, ax = plt.subplots(3,4, constrained\_layout = True)

for i, eig in enumerate(top12eignevectors):

ax[i//GX,i%4].imshow(eig,cmap=plt.cm.gray, interpolation='none')

ax[i//GX,i%4].set\_title(f"Eigenvector {i}")

plt.show()

assert np.all(eigenvalues[:-1] >= eigenvalues[1:]) # elements sorted

Y = U.T @ X

print(f"Shape of Y: {Y.shape}")

print(f"Shape of U: {U.shape}")

num\_img = 4

charaters = [datachar[i] for i in range(num\_img)]

df = pd.DataFrame(Y[:10, :num\_img], columns=charaters)

df.reset\_index(inplace=True)

df\_long = pd.melt(df, id\_vars='index', value\_vars=charaters,

var\_name='Character', value\_name='Value')

print(df\_long)

sns.lineplot(data=df\_long, x='index', y='Value', hue='Character')

plt.show()

\_, ax = plt.subplots(3,2, constrained\_layout = True)

for i, m in enumerate([1, 5, 10, 15, 20, 30]):

reconstructed\_imgs = U[:, :m]@Y[:m, :] + mean\_image

ax[i//2,i%2].imshow(reconstructed\_imgs[:,0].reshape(64, 64),cmap=plt.cm.gray, interpolation='none')

ax[i//2,i%2].set\_title(f"Singular values: {m}" )

plt.show()

def excersise\_5():

X = read\_data()

print(f"Shape of X: {X.shape}")

dim, n = X.shape

mean\_image = np.average(X, axis=1)[:, np.newaxis]

X = X - mean\_image

Z = X / np.sqrt(n-1)

U, S, Vt = np.linalg.svd(Z, full\_matrices=False)

A = U[:, :10]

Y = A.T @ X

Ck=12

params={}

for k in range(26):

mu = np.average(Y[:, k::26], axis=1)[:, np.newaxis]

Z = Y[:, k::26] - mu

assert Z.shape == (10, Ck)

params[k] = {

"mean": mu,

"cov": Z@Z.T / (Ck-1)

}

X\_test = read\_data\_test()

X\_test -= mean\_image

Y\_test = A.T @ X\_test

print(f"Shape of Y\_test: {Y\_test.shape}")

print(f"Shape of X\_test: {X\_test.shape}")

#compute Rwc

Rwc = np.zeros((10, 10))

for value in params.values():

Rwc += value["cov"]

Rwc /= 26

results = np.zeros((26, 26))# class, input

for k in range(26):

y\_test\_shift = Y\_test - params[k]['mean']

for test\_idx in range(26):

y\_test\_sample = y\_test\_shift[:, test\_idx]

# Bk = params[k]['cov']

Bk = np.diag(np.diag(params[k]['cov']))

Bk = Rwc

# Bk = np.diag(np.diag(Rwc))

# Bk = np.identity(10)

results[k, test\_idx] = y\_test\_sample.T@np.linalg.inv(Bk)@y\_test\_sample + np.log(np.linalg.det(Bk))

results = np.argmin(results, axis = 0)

for i, x in enumerate(results):

if i!= x:

print(f"{datachar[i]} <-> {datachar[x]}")

if \_\_name\_\_ == "\_\_main\_\_":

'''soruce driver code'''

# excersise\_2()

# excersise\_4()

excersise\_5()