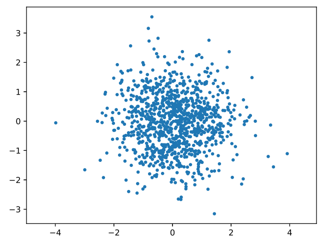
ECE 637: Lab 5

Rahul Deshmukh
deshmuk5@purdue.edu

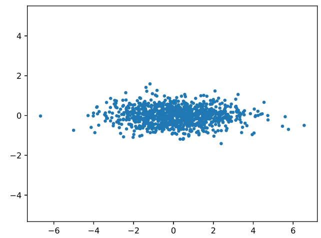
March 4, 2021

Section 2.1 Report

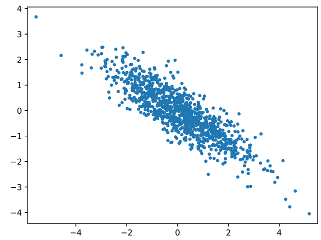
Scatter plots of $W, \tilde{X}$ and X



(a) plot of W



(b) plot of $\tilde{X}$



(c) plot of X

Figure 1: Scatter plots

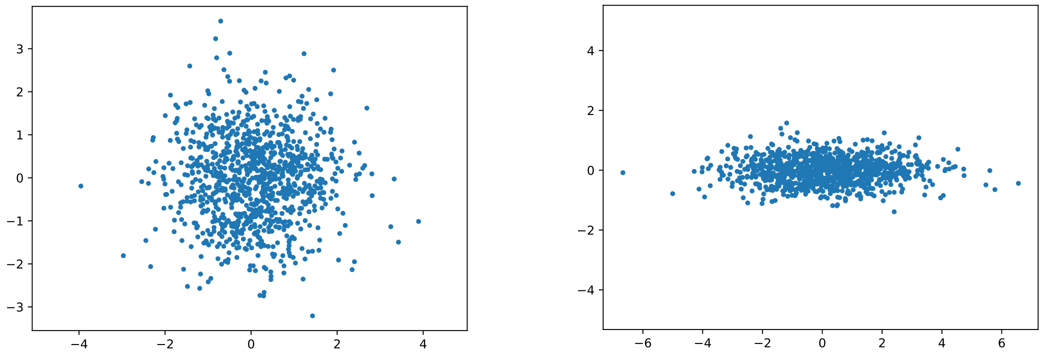
For python code refer to Listing 7 at page 11.

Section 2.2 Report

1. Theoretical value of $R_x = \begin{bmatrix} 2 & -1.2 \\ -1.2 & 1 \end{bmatrix}$

2. Numerical listing of $\hat{R}_x$ can be found in lines 5-7 of Listing 1 at page 2

3. Scatter plots of $\tilde{X}_i$ and W



(a) plot of W_i (b) plot of $\tilde{X}_i$

Figure 2: Scatter plots

4. Numerical listing of $\hat{R}_W$ can be found in lines 13-15 of Listing 1 at page 2

Listing 1: output log for section2

```
1  Input covariance(R):
2  [[ 2.   -1.2]
3   [-1.2  1. ]]
4
5  Estimated covariance(R_hat):
6  [[ 2.0016139   -1.23182429]
7   [-1.23182429   1.024182  ]]
8
9  Difference in covariances(R_hat-R):
10 [[ 0.0016139   -0.03182429]
11  [-0.03182429   0.024182  ]]
12
13 Estimated covariance of R_W_hat:
14 [[ 1.00000000e+00  -1.32739165e-15]
15  [-1.32739165e-15   1.00000000e+00]]
```

Section 4 Report

1. Eigen images:

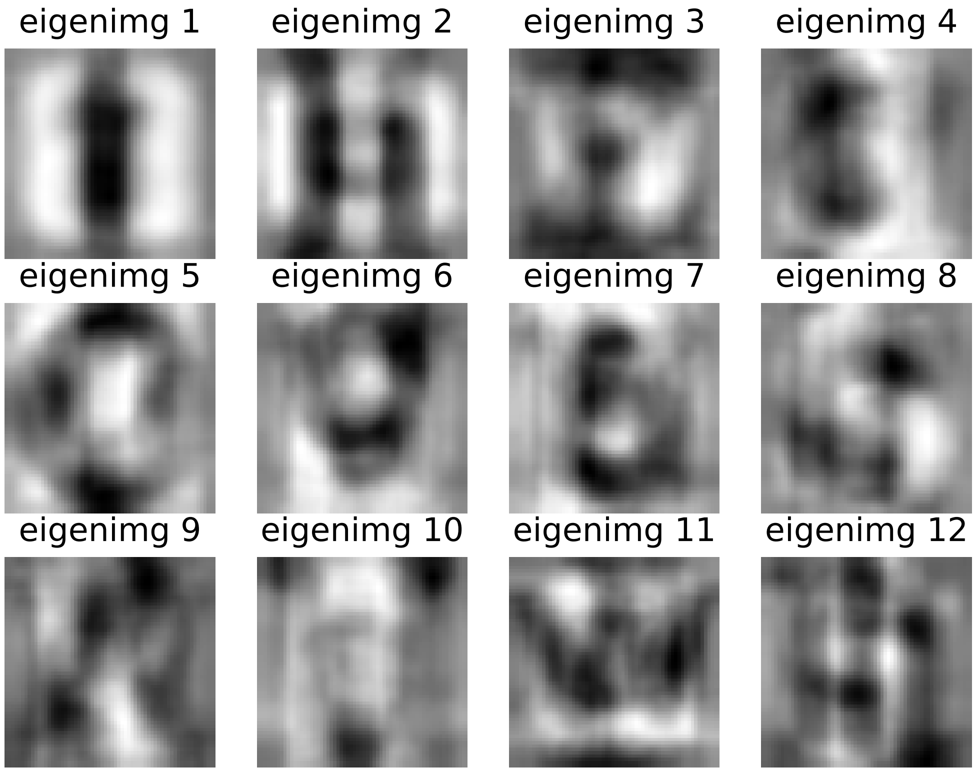
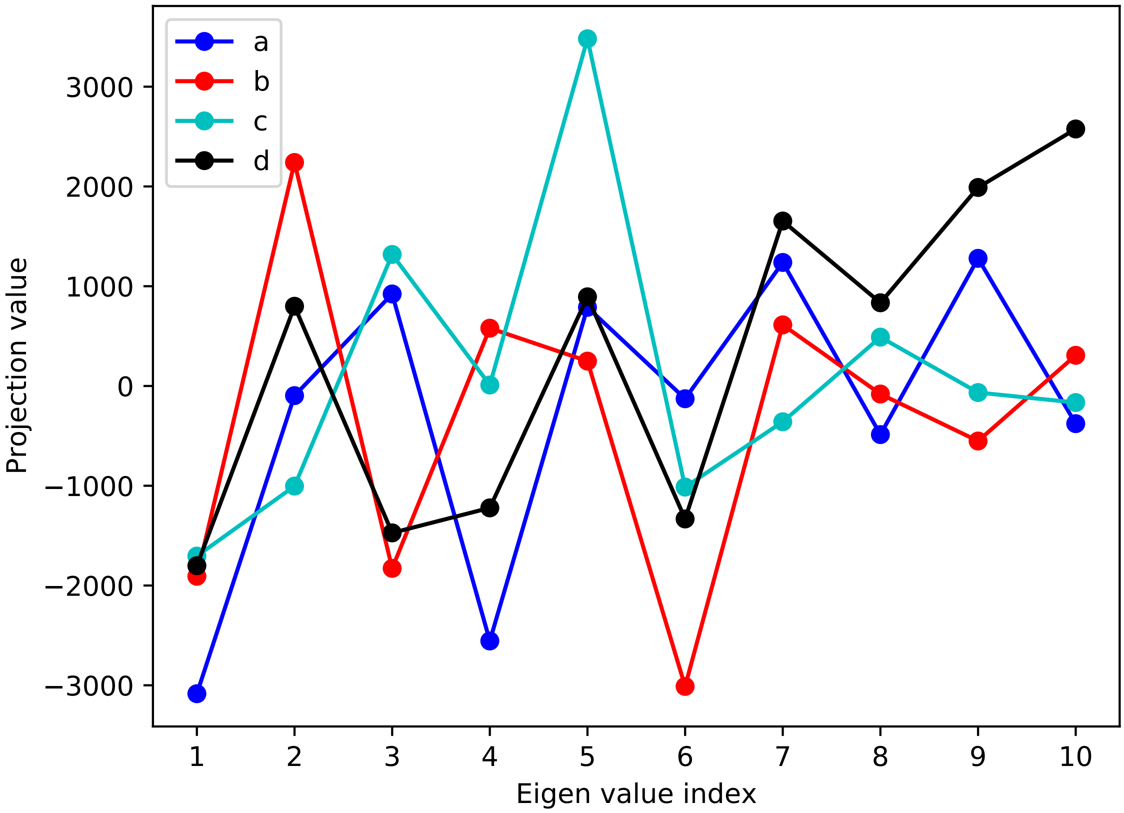
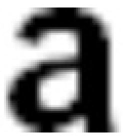


Figure 3: First 12 eigen images with title

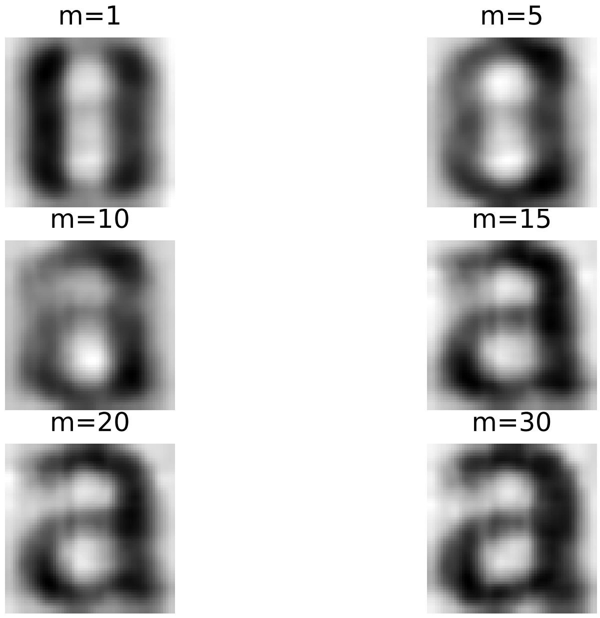
2. Plot of projection coefficients vs eigen vector number:



3. Original image vs 6 resynthesized versions:



(a) Original image



(b) Resynthesized image

For python code refer to Listing 8 at page 12

Section 5.1 Report

The model has an accuracy of 69.231%. The mis-classified images are indicated with 'x' in the third column of Listing 2 at page 6

Listing 2: output log for classification

```
option-0
True    Predicted
a  a
b  b
c  c
d  a  x
e  e
f  f
g  g
h  h
i  i
j  y  x
k  k
l  i  x
m  m
n  v  x
o  o
p  e  x
q  a  x
r  r
s  s
t  t
u  a  x
v  v
w  w
x  x
y  v  x
z  z
Accuracy: 69.231
------------------------------
```

Section 5.2 Report

The classification results for all modifications can be found at Listings 3-6.

1. From listings 3-6 we can observe that modifications #1,#2,#3 perform the best with same accuracy (92.308%) on testing data.

2. When we constrain the covariance (R_k) we essentially are trying to obtain well-conditioned B_k which helps in computing the inverse (B_k) for computing class scores. This solution is particularly helpful when we have limited training data which can lead to singular ML estimates of R_k. Here is a discussion on the effect of constraining for each modification:

   (a) Modification #1: Accuracy= 92.308%

   $$B_k = \Lambda_k$$

   We make the assumption that we have a pure quadratic classifier (ie only y_i^2 terms). Also since we chose only the diagonal terms there are more chances that B_k is non-singular (ie $[\Lambda_k]_{ii} \neq 0 \quad \forall \quad i \in [1, 2, .., n]$). This explains the improvement in accuracy compared to using R_k.

   (b) Modification #2: Accuracy= 92.308%

   $$B_k = R_{wc} = \frac{1}{K} \sum_{k=1}^{K} R_k$$

   We make the assumption that we have the same well-conditioned quadratic classifier $B_k = R_{wc}$ for all K classes. Since we compute B_k using a sum, the estimate has higher chances of being non-singular. This explains the improvement in accuracy compared to using R_k.

   (c) Modification #3: Accuracy= 92.308%

   $$B_k = \Lambda$$

   We make the asssumption that we have the same well-conditioned pure-quadratic classifier B_k for all K classes using only the diagonal terms of R_{wc} to construct Λ. This explains the improvement in accuracy compared to using R_k.

   (d) Modification #4: Accuracy= 88.462%

   $$B_k = I$$

   We make the assumption that we have a linear classifier as $B_k = I$. Even though B_k is non-singular in this case but we have reduced the complexity of our model. This explains the improvement in accuracy compared to using R_k. However the model does not have enough expressivity to get accuracy close to a quadratic classifier (Modification #1-#3).

Listing 3: output log for modification #1

```
option-1
True    Predicted
a   a
b   b
c   c
d   d
e   e
f   f
g   g
h   h
i   l   x
j   j
k   k
l   l
m   m
n   n
o   o
p   p
q   q
r   r
s   s
t   t
u   u
v   v
w   w
x   x
y   v   x
z   z
Accuracy: 92.308
---------------------------
```

Listing 4: output log for modification #2

```
option-2
True    Predicted
a   a
b   b
c   c
d   d
e   e
f   f
g   q   x
h   h
i   i
j   j
k   k
l   l
m   m
n   n
o   o
p   p
q   q
r   r
s   s
t   t
u   u
v   v
w   w
```

```
x    x
y    v    x
z    z
Accuracy:  92.308
——————————————————
```

Listing 5: output log for modification #3

```
option−3
True     Predicted
a    a
b    b
c    c
d    d
e    e
f    t    x
g    g
h    h
i    i
j    j
k    k
l    l
m    m
n    n
o    o
p    p
q    q
r    r
s    s
t    t
u    u
v    v
w    w
x    x
y    v    x
z    z
Accuracy:  92.308
——————————————————
```

Listing 6: output log for modification #4

```
option−4
True     Predicted
a    a
b    b
c    c
d    d
e    e
f    t    x
g    q    x
h    h
i    i
j    j
k    k
l    l
m    m
n    n
o    o
p    p
```

```
q    q
r    r
s    s
t    t
u    u
v    v
w    w
x    x
y    v    x
z    z
Accuracy: 88.462
------------------------------
```

Appendix

Got to git repo for complete code.

Listing 7: Python code for section 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  1 17:46:46 2021

@author: rahul
course: ece637 DIP-1
lab5: 2.1 and 2.2
"""

import sys
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(seed=637)
Rx = np.array([[2.0, -1.2],[-1.2,1.0]])

def plot_pts(W,name):
    plt.figure()
    plt.plot(W[0,:],W[1,:],'.')
    plt.axis('equal')
    plt.savefig(name+'.eps',format='eps')
    plt.close()

def estimate_mean_cov(X):
    "unbiased estimate of mean and cov (MLE)"
    dim,num_pts= X.shape
    mu_hat = X.sum(axis=1)/num_pts
    R_hat= np.zeros((dim,dim))
    for i in range(num_pts): R_hat+= np.outer(X[:,i]-mu_hat,X[:,i]-mu_hat)
    R_hat /= num_pts - 1.0
    return mu_hat, R_hat


def main(num_pts):
    #Section 2.1
    size,_=Rx.shape
    I = np.eye(size)
    mu= np.zeros(size)
    Lam,E = np.linalg.eig(Rx)
    W = np.random.multivariate_normal(mu,I,num_pts).T
    X_scaled = np.dot(np.diag(np.sqrt(Lam)),W)
    X = np.dot(E,X_scaled)
    plot_pts(W,'W')
    plot_pts(X_scaled,'X_scaled')
    plot_pts(X,'X')

    #Section 2.2
    mu_hat, R_hat = estimate_mean_cov(X)
    print('Input covariance(R): ')
    print(Rx);print()
    print('Estimated covariance(R_hat): ')
    print(R_hat);print()
```

```
    print('Difference in covariances(R_hat-R): ')
    print(R_hat-Rx);print()

    Lam_hat,E_hat = np.linalg.eig(R_hat)
    X_scaled_hat = np.dot(E_hat.T,X)
    W_hat = np.dot(np.diag(np.sqrt(1/Lam_hat)), X_scaled_hat)
    mu_W_hat, R_W_hat = estimate_mean_cov(W_hat)
    print('Estimated covariance of R_W_hat: ')
    print(R_W_hat);print()

    plot_pts(X_scaled_hat,'X_scaled_hat')
    plot_pts(W_hat,'W_hat')

if __name__=="__main__":
    num_pts = int(sys.argv[1])
    main(num_pts)
```

Listing 8: Python code for PCA

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  1 20:19:53 2021

@author: rahul
course: ece637 DIP-1
lab: 5 section 4
"""

import numpy as np
import matplotlib.pyplot as plt
from read_data import read_data

Ht=64
Wd=64
m = [1,5,10,15,20,30]

def display_eigen_images(U,Lam):
    fig, axs = plt.subplots(3, 4)
    for k in range(12):
        img=np.reshape(U[:,k],(Wd,Ht))
        axs[k//4,k%4].imshow(img,cmap=plt.cm.gray, interpolation='none')
        axs[k//4,k%4].set_title('eigenimg '+str(k+1))
        axs[k//4,k%4].axis('off')
    plt.savefig('eigen_images.pdf')
    plt.close()

def plot_projections(Y):
    c=10
    x = np.arange(1,c+1,1)
    plt.figure()
    plt.xticks(x)
    plt.plot(x,Y[:c,0],'-ob',label='a')
    plt.plot(x,Y[:c,1],'-or',label='b')
    plt.plot(x,Y[:c,2],'-oc',label='c')
    plt.plot(x,Y[:c,3],'-ok',label='d')
    plt.xlabel('Eigen value index')
    plt.ylabel('Projection value')
    plt.legend()
```

```
    plt.savefig('projections.pdf')


def main():
    X = read_data()
    p,n = X.shape
    mu_hat = np.sum(X, axis=1)/n
    X = (X.T - mu_hat).T
    Z = X/np.sqrt(n)
    U,S,Vt = np.linalg.svd(Z,full_matrices=False)
    Lam = S**2
    display_eigen_images(U,Lam)

    #projection of images
    Y = np.dot(U.T,X[:,:4])
    plot_projections(Y)

    #reconstruction
    recons = np.zeros((p,len(m)))
    for i,im in enumerate(m):
        recons[:,i] = np.dot(U[:,:im],Y[:im,0])
    recons = (recons.T + mu_hat).T
    fig,axs=plt.subplots(3,2)
    for k in range(6):
        img=np.reshape(recons[:,k],(Wd,Ht))
        axs[k//2,k%2].imshow(img,cmap=plt.cm.gray, interpolation='none')
        axs[k//2,k%2].set_title('m='+str(m[k]))
        axs[k//2,k%2].axis('off')
    plt.savefig('reconstruction.pdf')
    plt.close()

if __name__=="__main__":
    main()
```

Listing 9: Python code for image classification using PCA

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar  2 00:48:48 2021

@author: rahul
course: ece637 DIP-1
lab-5 section 5
"""
import glob,os,sys
import numpy as np
from PIL import Image
from read_data import read_data

Ht=64; Wd=64;
eig_size=10
test_dir='../../test_data/'

def option014(Y,option):
    params = []
    for k in range(26):
        samples = Y[:,k::26]
        _,num_samples = samples.shape
```

```
        mu = samples.sum(axis=1)/(1.0*num_samples)
        if option!=4:
            cov = np.zeros((eig_size,eig_size))
            for i in range(num_samples):
                cov += np.outer(samples[:,i]-mu,samples[:,i]-mu)
            cov /= num_samples-1.0
            if option==1:
                cov = np.diag(np.diagonal(cov)) #only diagonal elements
        else:
            cov = np.eye(eig_size)
        dic = {'mean':mu, 'cov':cov}
        params.append(dic)
    return params

def option23(Y,option):
    params = []
    for k in range(26):
        samples = Y[:,k::26]
        _,num_samples = samples.shape
        mu = samples.sum(axis=1)/(1.0*num_samples)
        cov = np.zeros((eig_size,eig_size))
        for i in range(num_samples):
            cov += np.outer(samples[:,i]-mu,samples[:,i]-mu)
        cov /= num_samples-1.0
        dic = {'mean':mu, 'cov':cov}
        params.append(dic)
    R_wc = np.zeros((eig_size,eig_size))
    for k in range(26): R_wc += params[k]['cov']
    R_wc /= 26.0
    if option==2:
        for k in range(26): params[k]['cov'] = R_wc
    elif option==3:
        Lam = np.diag(np.diagonal(R_wc))
        for k in range(26): params[k]['cov'] = Lam
    return params

def training(option):
    X = read_data()
    p,n = X.shape
    mu_x = np.sum(X,axis=1)/n
    X = (X.T - mu_x).T
    Z = X/np.sqrt(n)
    U,S,Vt = np.linalg.svd(Z,full_matrices=False)
    A = U[:,:eig_size]
    Y = np.dot(A.T,X)

    if (np.any(np.array([0,1,4])==option)): params = option014(Y,option)
    elif(np.any(np.array([2,3])==option)): params = option23(Y,option)

    return params, A, mu_x

def testing(params, A, mu_x):
    #read images
    filenames = glob.glob(test_dir+'*/*.tif')
    true_labels = []
    for f in filenames:
        class_label = os.path.basename(f).split('.')[0]
        true_labels.append(class_label)
    #classify data
```

```
    predicted_labels = []
    for f in filenames:
        im = Image.open(f)
        img = np.array(im)
        x = np.reshape(img,Ht*Wd) - mu_x
        y = np.dot(A.T,x)
        class_scores = np.zeros(26)
        for k in range(26):
            mu_k = params[k]['mean']
            cov_k = params[k]['cov']
            class_scores[k] = (np.dot((y-mu_k),np.dot(np.linalg.inv(cov_k),(y-mu_k))
                ) +
                                np.log(np.abs(np.linalg.det(cov_k))) )
        class_idx = np.argmin(class_scores)
        class_label = chr(ord('a') + class_idx)
        predicted_labels.append(class_label)
    return true_labels, predicted_labels

if __name__=="__main__":
    option = int(sys.argv[1])
    params, A, mu_x = training(option)
    true_labels, predicted_labels = testing(params, A, mu_x)
    # print results
    print('True\t Predicted\t')
    count = 0.0
    for i in range(len(true_labels)):
        flag_correct = (true_labels[i]==predicted_labels[i])
        if flag_correct: count+=1
        print("%s\t %s"%(true_labels[i],predicted_labels[i]),end='\t')
        if not flag_correct:
            print('x',end='')
        print('')
    acc = (count/len(true_labels))*100
    print('Accuracy: %0.3f'%(acc))
```

Listing 10: Python code for data reading utility

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 15 19:32:39 2021
ECE637
Prof. Charles A. Bouman
Image Processing Laboratory: Eigenimages and Principal Component Analysis

Description:

This is a Matlab script that reads in a set of training images into
the Matlab workspace. The images are sets of English letters written
in various fonts. Each image is reshaped and placed into a column
of a data matrix, "X".
@author: Wenrui Li
"""

import sys
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

```

```python
# The following are strings used to assemble the data file names
datadir='../../training_data/'    # directory where the data files reside
dataset=['arial','bookman_old_style','century','comic_sans_ms','courier_new',
  'fixed_sys','georgia','microsoft_sans_serif','palatino_linotype',
  'shruti','tahoma','times_new_roman']
datachar='abcdefghijklmnopqrstuvwxyz'

def read_data():
    """
        Read in all these training images into columns of a single matrix X.

        Returns:
            X: Image column matrix.

    """
    Rows=64    # all images are 64x64
    Cols=64
    n=len(dataset)*len(datachar)  # total number of images
    p=Rows*Cols   # number of pixels

    X=np.zeros((p,n))  # images arranged in columns of X
    k=0
    for dset in dataset:
        for ch in datachar:
            fname='/'.join([datadir,dset,ch])+'.tif'
            im=Image.open(fname)
            img = np.array(im)
            X[:,k]=np.reshape(img,(1,p))
            k+=1
    return X

# display samples of the training data
def display_samples(X,ch):
    """
    Display samples.

    Args:
    X (ndarray) : Image column matrix.
    ch (char) : A char 'a'~'z'.

    Returns:

    """
    ind = ord(ch)-ord('a')
    fig, axs = plt.subplots(3, 4)
    for k in range(len(dataset)):
        img=np.reshape(X[:,26*(k-1)+ind],(64,64))

        axs[k//4,k%4].imshow(img,cmap=plt.cm.gray, interpolation='none')
        axs[k//4,k%4].set_title(dataset[k])

    plt.show()

if __name__ == "__main__":
    ch = sys.argv[1]
    X = read_data()
    display_samples(X,ch)
```

Listing 11: Bash code for running python code

```
#!/bin/bash

#Section 2.1 and 2.2
python3 ex2.py 1000 | tee sec2.log
mv ./*.eps output/section_2/
mv ./sec2.log output/section_2/
echo 'section 2 done'

#section 4
python3 ex4.py
mv ./*.pdf output/section4/
echo 'section 4 done'

#section 5
for opt in {0..4}
do
 {
 echo option-"$opt";
 python3 ex5.py $opt;
 echo "---------------------------";
 } | tee ex5_opt_"$opt".log
done
mv ./*.log output/section5/
echo 'section 5 done'
```