Problem A7

(a) Answer: See Explanation

Explanation:
$\lambda_1 = 5.14833344148535, \lambda_2 = 3.7299894906022506,$
$\lambda_{10} = 1.2500107572120347, \lambda_{30} = 0.36492647617794244, \lambda_{50} = 0.1696213156685332$
$\sum_{i=1}^{d} \lambda_i = 52.833844000945014$

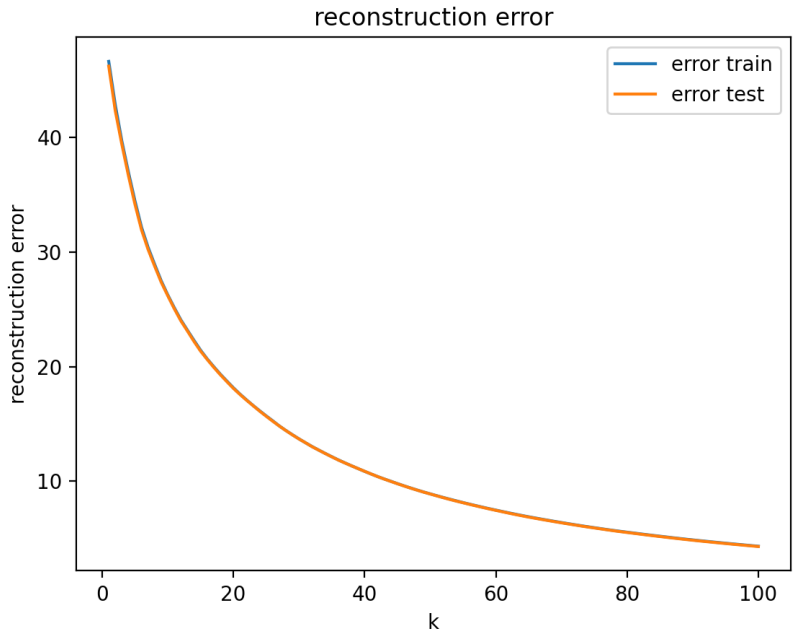
(b) Answer: See Explanation

Explanation:
$U_k U_k^T (x_i - \mu)$ is the projection of de-meaned data vector over the subspace spanened by k largest eigenvectors. Therefore,

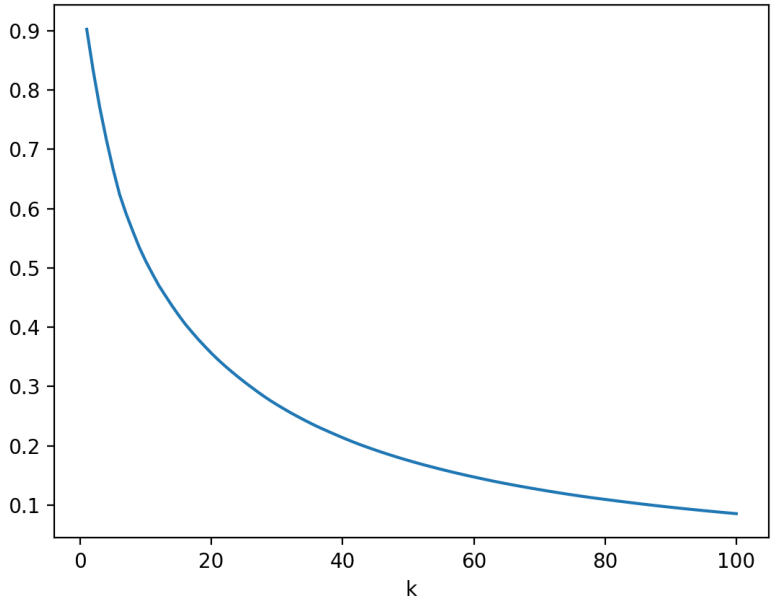
$$x_{i,\text{approx}} = U_k U_k^T (x_i - \mu) + \mu$$

(c) Answer: See Explanation

Explanation:
The plot for mean reconstruction error $\frac{1}{n}\sum_{i=1}^{n}\|(x_i-\mu)-U_kU_k^T(x_i-\mu)\|^2$

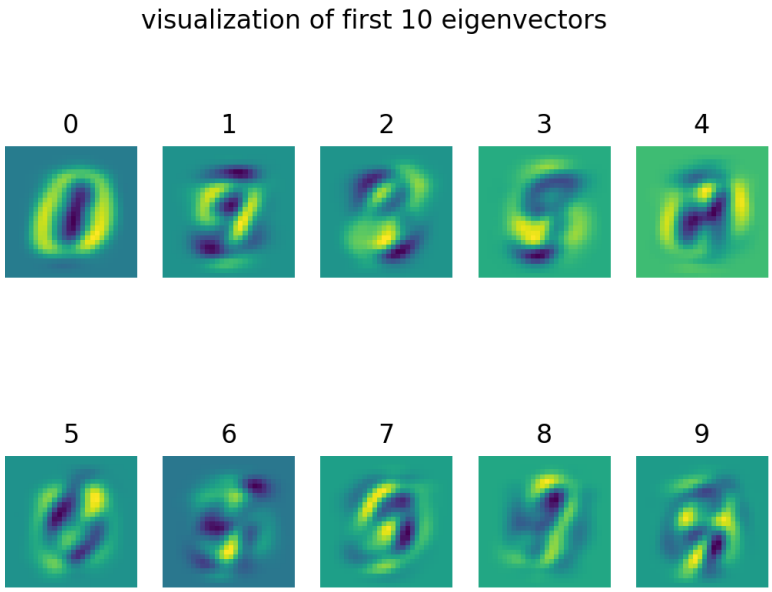


The plot for $1-\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i}\lambda_i$



(d) Answer: See Explanation

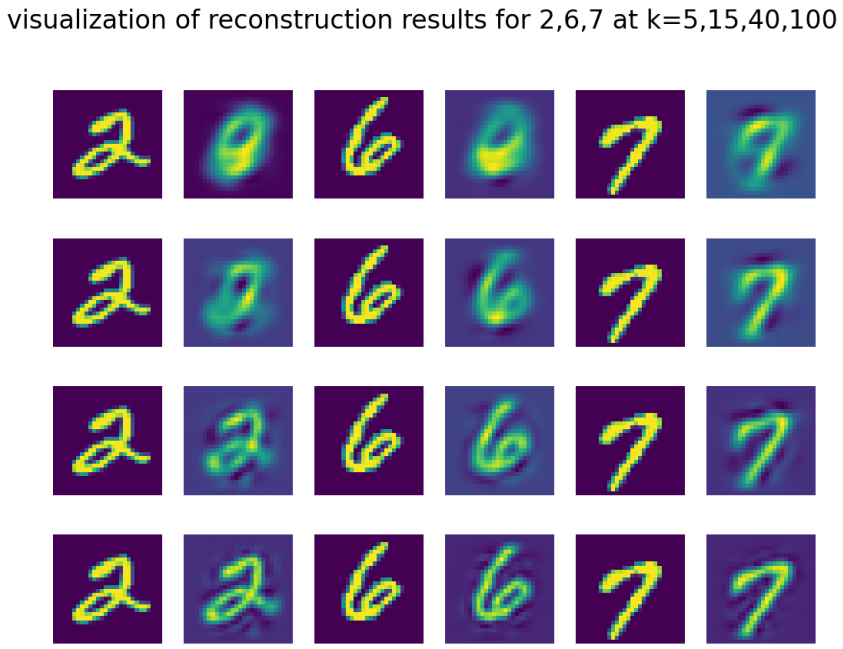
Explanation:



Each of the image resembles the rough shape of a written digit. This is not surprising, because the data set contains images of written digits, which means most of image in the data set will roughly have a shape like that.

(e) Answer: See Explanation

Explanation:



From the image, it's obvious that the higher value k, the better the reconstruction result (closer resemblance of the original image, sharper and less blur). 6,7 are distinguishable at $k = 15$ and 2 is distinguishable at $k = 40$. Comparing to the original dimension of 748, the reconstruction result at $k = 100$ is more than enough for any visual inspection for humans.

Code for this problem

```
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt

def load_dataset():
    mndata = MNIST('./data.nosync/data')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
    return X_train, labels_train, X_test, labels_test

# uk (d, k): first k eigen vectors (k <= d)
# dmean_data (n, d): x-mean(x) for feature
def reconstruction_error(uk, demean_data):
    tmp = demean_data.T - uk @ uk.T @ demean_data.T # (d, n)
    # take the norm of each column, and then find the mean (of n values)
    res = np.mean(np.linalg.norm(tmp, axis=0)) ** 2
    return res

def reconstruct_demean(uk, demean_vec):
    return uk @ uk.T @ demean_vec

if __name__ == "__main__":
    X_train_raw, labels_train_raw, X_test_raw, labels_test_raw = load_dataset()

    X_train = X_train_raw[:50000]
    X_test = X_train_raw[50000:]

    n_train, d = X_train.shape
    # take the sum of each column
    mu = 1 / n_train * np.sum(X_train, axis=0).reshape(d, 1)
    # duplicate mu (as row vector) n times to make n rows (n, d)
    demean_X_train = X_train - np.repeat(mu.T, n_train, axis=0)
    # compute sigma (coovariance matrix)
    sigma = demean_X_train.T @ demean_X_train / n_train

    n_test, _ = X_test.shape
    # also demean the test set, using the mean of training set
    demean_X_test = X_test - np.repeat(mu.T, n_test, axis=0)

    # compute eigenvalue and eigenvectors for sigma
    eig, vec = np.linalg.eig(sigma)
```

```
# print eigenvalues
tmp = [1, 2, 10, 30, 50]
for x in tmp:
    print(eig[x-1])
print(np.sum(eig))

# index of a sample of 2, 6, 7 in the training set
idx2 = 5
idx6 = 13
idx7 = 15
k_sample = [5, 15, 40, 100]
sample_res = []

# compute reconstruction error
err_train = []
err_test = []
eig_ratio = []
for k in range(1, 101):
    uk = vec.T[:k].T # first k columns
    err_train.append(reconstruction_error(uk, demean_X_train))
    err_test.append(reconstruction_error(uk, demean_X_test))
    eig_ratio.append(1 - np.sum(eig[:k]) / np.sum(eig))
    if k in k_sample:
        tmp = {}
        tmp[2] = reconstruct_demean(uk, demean_X_train[idx2].reshape((d, 1))) + mu
        tmp[6] = reconstruct_demean(uk, demean_X_train[idx6].reshape((d, 1))) + mu
        tmp[7] = reconstruct_demean(uk, demean_X_train[idx7].reshape((d, 1))) + mu
        sample_res.append(tmp)

ks = [i+1 for i in range(100)]
# plot graph 1
plt.plot(ks, err_train, label="error train")
plt.plot(ks, err_test, label="error test")
plt.legend()
plt.xlabel("k")
plt.ylabel("reconstruction error")
plt.title("reconstruction error")
plt.show()

# plot graph 2
plt.plot(ks, eig_ratio, label="")
plt.xlabel("k")
plt.show()
```

```
# plot image
fig, ax = plt.subplots(nrows=2, ncols=5)
for i in range(10):
    ax[i//5][i%5].imshow(vec.T[i].reshape((28, 28)).astype('float64'))
    ax[i//5][i%5].title.set_text(str(i))
    ax[i//5][i%5].axis('off')
plt.suptitle("visualization of first 10 eigenvectors")
plt.show()

# plot image 2
fig, ax = plt.subplots(nrows=4, ncols=6)
for i in range(4):
    ax[i][0].imshow(X_train[idx2].reshape((28, 28)).astype('float64'))
    ax[i][1].imshow(sample_res[i][2].reshape((28, 28)).astype('float64'))
    ax[i][2].imshow(X_train[idx6].reshape((28, 28)).astype('float64'))
    ax[i][3].imshow(sample_res[i][6].reshape((28, 28)).astype('float64'))
    ax[i][4].imshow(X_train[idx7].reshape((28, 28)).astype('float64'))
    ax[i][5].imshow(sample_res[i][7].reshape((28, 28)).astype('float64'))
    for j in range(6):
        ax[i][j].axis('off')
plt.suptitle("visualization of reconstruction results for 2,6,7 at k=5,15,40,100")
plt.show()
```