ת.ז: [redacted]

[redacted]

תרגיל בית 2

1) (a) כפי שלמדנו בכיתה, נוסחת ה-Likelihood נתונה ע"י: $\mathcal{L}(\theta) = \mathbb{P}(X_1,\dots,X_n;\theta) = \prod_{i=1}^{n} \mathbb{P}(X_i;\theta)$.

עבור ההתפלגות הגאומטרית מוגדר ש- $\mathbb{P}(X=k) = (1-\theta)^{k-1}\theta$.

נציב בנוסחת ה-Likelihood ונקבל:

$$\mathcal{L}(\theta) = (1-\theta)^{X_1-1}\theta(1-\theta)^{X_2-1}\theta\cdot\ldots\ldots\cdot(1-\theta)^{X_n-1}\theta = \theta^n\cdot(1-\theta)^{\sum_{i=1}^{n}X_i-n}$$

נפעיל לוג על שני האגפים ונקבל:

$$Log(\mathcal{L}(\theta)) = Log(\theta^n\cdot(1-\theta)^{\sum_{i=1}^{n}X_i-n}) = Log(\theta^n) + Log((1-\theta)^{\sum_{i=1}^{n}X_i-n}) = n\cdot Log(\theta) + \left(\sum_{i=1}^{n}X_i - n\right)\cdot Log(1-\theta)$$

נעבור ל- ln מטעמי נוחות לגזירה בהמשך:

$$\ln(\mathcal{L}(\theta)) = n\cdot\ln(\theta) + \left(\sum_{i=1}^{n}X_i - n\right)\cdot\ln(1-\theta)$$

נגזור לפי θ:

$$\frac{d(\ln(\mathcal{L}(\theta)))}{d(\theta)} = \frac{n}{\theta} - \frac{\sum_{i=1}^{n}X_i - n}{(1-\theta)}$$

נשווה את הנגזרת ל-0 למציאת מקסימום:

$$\frac{n}{\theta} - \frac{\sum_{i=1}^{n}X_i - n}{(1-\theta)} = 0 \iff \frac{n}{\theta} = \frac{\sum_{i=1}^{n}X_i - n}{(1-\theta)}$$

$$\iff n(1-\theta) = \theta\left(\sum_{i=1}^{n}X_i - n\right) \iff n - \cancel{n\theta} = \theta\sum_{i=1}^{n}X_i - \cancel{n\theta} \iff n = \theta\sum_{i=1}^{n}X_i \iff \theta = \frac{n}{\sum_{i=1}^{n}X_i}$$

אז קיבלנו: $\boxed{\theta_{ML} = \frac{n}{\sum_{i=1}^{n}X_i}}$

(b) נתון: $(X_1,X_2,X_3,X_4,X_5,X_6) = 2,3,3,3,6,2$

נציב את הנתונים בנוסחה θ_{ML} שמצאנו בסעיף a:

$$\theta_{ML} = \frac{n}{\sum_{i=1}^{n}X_i} = \frac{6}{2+3+3+3+6+2} = \boxed{\frac{6}{19}}$$

Question 2:

Note: below you can find the code we wrote and its output. While writing the code we also used the code from "loadMNIST_py" that was given in the Moodle.

Section a

Code:

```
X = np.array(x_train)
X = (X / 255) - 0.5 # divide them by 255 and reduce them by 0.5
```

Section b.1

Code:

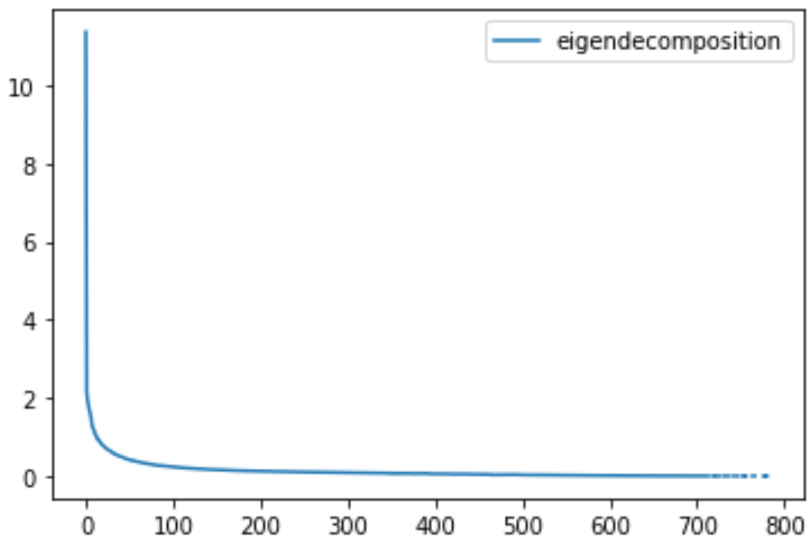
```
X = X.T
X = X.reshape(28*28,60000) # flatten each image into a vector
covariance_matrix = (X @ X.T) / 60000 # compute the so called covarianc
e matrix
```

Section b.2

Code:

```
w, v = np.linalg.eig(covariance_matrix) #compute the eigendecomposition
w = np.sqrt(np.real(w)) # The eigenvalues
U = np.real(v) # The normalized eigenvectors

plt.plot(w, label="eigendecomposition")
plt.legend()
plt.show()
```

output:



Section b.3

Code:

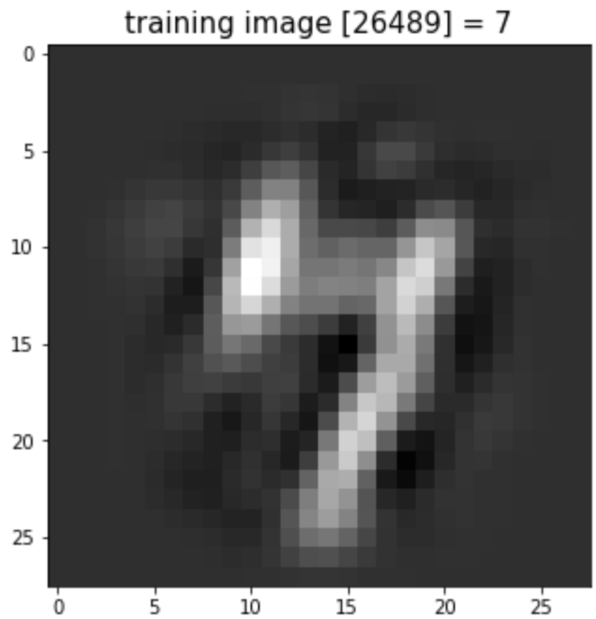
```
def first_p_columns(p,U,X):
  Up = U[:,0:p]
  W = Up.T @ X
  return Up, W
```

Section b.4

Code:

```
Up, W = first_p_columns(40,U,X)
print(Up.shape) # (784, 40)
print(W.shape) # (40, 60000)

new_X = Up @ W
print(new_X.shape) # (784, 60000)

new_X = new_X.reshape(28,28,60000)
print(new_X.shape) # (28, 28, 60000)

new_X = new_X.T
print(new_X.shape) # (60000, 28, 28)

images_2_show = []
titles_2_show = []
r = random.randint(1, 60000)
images_2_show.append(new_X[r])
titles_2_show.append('training image [' + str(r) + '] = ' + str(y_train
[r]))
show_images(images_2_show, titles_2_show)
```

output:



Section c

Code:

```
def kmeans(data,k):
  centers = data[np.random.choice(np.arange(len(data)),size=k,replace=F
alse)]

  while True:
    labels = np.array([np.argmin([np.linalg.norm((d_p - center),ord=2)
for center in centers]) for d_p in data])
    new_centers = np.array([data[labels == k].mean(axis=0) for k in ran
ge(k)])
    if np.all(centers == new_centers):
      break
    centers = new_centers
  return labels, centers
```

Section d

Code:

```
labels, centers = kmeans(W.T,10)
```

Section e

Code:

```
def cluster_to_digit(labels):
  clusters = [[],[],[],[],[],[],[],[],[],[]]

  for i in range(len(labels)):
    clusters[labels[i]].append(y_train[i])

  dict = {}
  for i in range(len(clusters)):
    dict[i] = np.bincount(clusters[i]).argmax()

  return dict

dict = cluster_to_digit(labels)
print(dict)
```

output:

```
{0: 1, 1: 3, 2: 9, 3: 2, 4: 0, 5: 1, 6: 7, 7: 8, 8: 6, 9: 4}
```

Section f

Code:

```
test_X = np.array(x_test)
test_X = (test_X / 255) - 0.5
test_X = test_X.T
test_X = test_X.reshape(28*28,10000) # flatten each image into a vector
test_Up, test_W = first_p_columns(40,U,test_X)

def test_success(dict, data, labels):
  counter = 0
  for i in range(len(data.T)):
    cluster = np.argmin([np.linalg.norm((data.T[i] - center),ord=2) for
 center in centers])
    if dict[cluster] == labels[i]:
      counter += 1
  return counter/len(data.T)

success = test_success(dict, test_W, y_test)
print(success)
```

output:

```
0.5842
```

Section g

Code:

```
for i in range(3):
  labels, centers = kmeans(W.T,10)
  print(test_success(cluster_to_digit(labels), test_W, y_test))
```

output:

```
0.6129
0.5938
0.5937
```

Section h

Code:

```
Up, W = first_p_columns(12,U,X)
labels, centers = kmeans(W.T,10)
test_Up, test_W = first_p_columns(12,U,test_X)
print(test_success(cluster_to_digit(labels), test_W, y_test))
```

output:

```
0.5733
```

Section i

Code:

```
def new_kmeans(data,k):
  means = []
  for i in range(10):
    vectors = []
    counter = 0
    for j in y_train:
      if (i == j):
        vectors.append(data[counter])
        if (len(vectors) == 10):
          break
      counter += 1
    sum = 0
    for v in vectors:
      sum += v
    means.append(sum/10)
  centers = np.array(means)

  while True:
    labels = np.array([np.argmin([np.linalg.norm((d_p - center),ord=2)
for center in centers]) for d_p in data])
    new_centers = np.array([data[labels == k].mean(axis=0) for k in ran
ge(k)])
    if np.all(centers == new_centers):
      break
    centers = new_centers
  return labels, centers

Up, W = first_p_columns(40,U,X)
labels, centers = new_kmeans(W.T,10)
test_Up, test_W = first_p_columns(40,U,test_X)
print(test_success(cluster_to_digit(labels), test_W, y_test))
```

output:

```
0.6351
```