Assignment 8

Q1 (a) $P(X, c|\theta, \pi) = \prod_{i=1}^{N} P(X^{(i)}, C^{(i)})$

$$P(x, c|\theta, \pi) = \prod_{i=1}^{N} \left[P(C^{(i)}|\pi) \prod_{j=1}^{784} P(x_j^{(i)} | C^{(i)}, \theta_{jC^{(i)}}) \right]$$

$$\ell(x, c|\theta, \pi) = \log P(x, c|\theta, \pi) = \sum_{i=1}^{N} \log \left[P(C^{(i)}|\pi) \prod_{j=1}^{784} P(x_j^{(i)} | C^{(i)}, \theta_{jC^{(i)}}) \right]$$

$$= \underbrace{\sum_{i=1}^{N} \log P(C^{(i)}|\pi)}_{A} + \underbrace{\sum_{j=1}^{784} \sum_{i=1}^{N} \log P(x_j^{(i)} | C^{(i)}, \theta_{jC^{(i)}})}_{B}$$

To maximize $\ell(x, c|\theta, \pi)$, we maximize A and B individually

θ_{jc}: probability of obtaining $x_j = 1$ from class c

$$\frac{\partial \ell(x, c|\theta, \pi)}{\partial \theta} = \sum_{j=1}^{784} \sum_{i=1}^{N} \frac{\partial \log P(x_j^{(i)} | C^{(i)}, \theta_{jC^{(i)}})}{\partial \theta}$$

$$= \sum_{j=1}^{784} \sum_{i=1}^{N} \frac{\partial \log \left[\theta_{jC^{(i)}}^{x_j^{(i)}} (1 - \theta_{jC^{(i)}})^{(1 - x_j^{(i)})} \right]}{\partial \theta}$$

$$= \sum_{j=1}^{784} \sum_{i=1}^{N} \frac{\partial \left[x_j^{(i)} \log \theta_{jC^{(i)}} + (1 - x_j^{(i)}) \log (1 - \theta_{jC^{(i)}}) \right]}{\partial \theta}$$

$$= \sum_{j=1}^{784} \sum_{i=1}^{N} \left[x_j^{(i)} \frac{\partial \log \theta_{jC^{(i)}}}{\partial \theta} + (1 - x_j^{(i)}) \frac{\partial (1 - \theta_{jC^{(i)}})}{\partial \theta} \right]$$

$$= \sum_{j=1}^{784} \sum_{i=1}^{N} x_j^{(i)} \frac{\partial \log \theta_{jC^{(i)}}}{\partial \theta} + \sum_{j=1}^{784} \sum_{i=1}^{N} (1 - x_j^{(i)}) \frac{\partial (1 - \theta_{jC^{(i)}})}{\partial \theta}$$

$= M \in \mathbb{R}^{784 \times 10}$ where $M_{jc} = \dfrac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)} = 1 \,\&\, C^{(i)} = c]}{\theta_{jc}} - \dfrac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)} = 0 \,\&\, C^{(i)} = c]}{1 - \theta_{jc}}$

$$\frac{\partial \ell(x, c|\theta, \pi)}{\partial \theta} = 0 \iff M = 0$$

$$\iff M_{jc} = 0 \quad \forall j, c$$

$$\iff \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)} = 1 \,\&\, C^{(i)} = c]}{\theta_{jc}} - \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)} = 0 \,\&\, C^{(i)} = c]}{1 - \theta_{jc}} = 0$$

$$(1-\theta_{jc})\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c] = \theta_{jc}\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]$$

$$\frac{1}{\theta_{jc}} - 1 = \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c] \Big/ \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]$$

$$\frac{1}{\theta_{jc}} = 1 + \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c] \Big/ \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]$$

$$\frac{1}{\theta_{jc}} = \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c] + \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}$$

$$\theta_{jc} = \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c] + \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}$$

$$\boxed{\theta_{jc} = \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}{\sum_{i=1}^{N} \mathbb{I}[c^{(i)}=c]} = \frac{\#\text{ of sample where } j^{th} \text{ bit is 1 and class is } c}{\#\text{ of sample of class } c}}$$

$$\log P(t^{(i)}|\pi) = \log \pi_{j=0}^{9} \pi_j^{t_j^{(i)}}$$

$$= \sum_{j=0}^{9} \log \pi_j^{t_j^{(i)}}$$

$$= \sum_{j=0}^{8} \log \pi_j^{t_j^{(i)}} + \log \pi_9^{t_9^{(i)}}$$

$$= \sum_{j=0}^{8} t_j^{(i)} \log \pi_j + t_9^{(i)} \log\left(1 - \sum_{j=0}^{8} \pi_j\right)$$

$$\frac{\partial \log P(t^{(i)}|\pi)}{\partial \pi_k} = t_k^{(i)} \frac{1}{\pi_k} + t_9^{(i)} \frac{1}{1-\sum_{j=0}^{8}\pi_j} \cdot (-1) \quad \forall k \in \{0,1,\dots,8\}$$

$$= \frac{t_k^{(i)}}{\pi_k} - \frac{t_9^{(i)}}{\pi_9}$$

Therefore $\dfrac{\partial l(x, c|\theta, \pi)}{\partial \pi} = 0 \iff \sum_{i=1}^{N} \dfrac{\partial \log P(t^{(i)}|\pi)}{\partial \pi} = 0$

$$\iff \sum_{i=1}^{N} \frac{\partial \log P(t^{(i)}|\pi)}{\partial \pi_k} = 0 \quad \forall k \in \{0,1,\dots 9\}$$

$$\iff \sum_{i=1}^{N} \left[\frac{t_k^{(i)}}{\pi_k} - \frac{t_9^{(i)}}{\pi_9}\right] = 0$$

$$\frac{1}{\pi_k}\sum_{i=1}^{N} t_k^{(i)} = \frac{1}{\pi_9}\sum_{i=1}^{N} t_9^{(i)}$$

$$\pi_k = \pi_q \cdot \sum_{i=1}^{N} t_k^{(i)} \Big/ \sum_{i=1}^{N} t_q^{(i)} \qquad \forall k \in \{0,\dots,8\}$$

we know $\sum_{k=0}^{8} \pi_k + \pi_q = 1$

$$\sum_{k=0}^{8} \left(\pi_q \cdot \sum_{i=1}^{N} t_k^{(i)} \Big/ \sum_{i=1}^{N} t_q^{(i)} \right) + \pi_q = 1$$

$$\left(1 + \frac{\sum_{k=0}^{8} \sum_{i=1}^{N} t_k^{(i)}}{\sum_{i=1}^{N} t_q^{(i)}} \right) \cdot \pi_q = 1$$

$$\pi_q = \frac{\sum_{i=1}^{N} t_q^{(i)}}{\sum_{k=0}^{8} \sum_{i=1}^{N} t_k^{(i)} + \sum_{i=1}^{N} t_q^{(i)}}$$

$$\pi_q = \frac{\sum_{i=1}^{N} t_q^{(i)}}{\sum_{k=0}^{9} \sum_{i=1}^{N} t_k^{(i)}} = \frac{\sum_{i=1}^{N} t_q^{(i)}}{N} = \frac{\sum_{i=1}^{N} \mathbb{I}[t_q^{(i)} = 1]}{N}$$

AND $\pi_k = \pi_q \cdot \frac{\sum_{i=1}^{N} t_k^{(i)}}{\sum_{i=1}^{N} t_q^{(i)}}$

$$= \frac{\sum_{i=1}^{N} t_q^{(i)}}{N} \cdot \frac{\sum_{i=1}^{N} t_k^{(i)}}{\sum_{i=1}^{N} t_q^{(i)}}$$

$$= \frac{\sum_{i=1}^{N} t_k^{(i)}}{N} = \frac{\sum_{i=1}^{N} \mathbb{I}[t_k^{(i)} = 1]}{N}$$

Therefore $\boxed{\pi_k = \dfrac{\sum_{i=1}^{N} \mathbb{I}[t_k^{(i)} = 1]}{N} = \dfrac{\#\text{ of sample of class } k}{\#\text{ of sample}} \quad \text{for } k \in \{0,\dots,9\}}$

(b) $$p(c \mid x^{(i)}, \theta, \pi) = \frac{P(x^{(i)} \mid c, \theta, \pi) \cdot P(c \mid \pi)}{P(x^{(i)})}$$

$$= \frac{P(x^{(i)} \mid c, \theta, \pi) \cdot P(c \mid \pi)}{\sum_{c} \left(P(x^{(i)} \mid c) \cdot P(c \mid \pi) \right)}$$

$$= \frac{\prod_{j} \left[\theta_{jc}^{x_j^{(i)}} (1 - \theta_{jc}^{x_j^{(i)}})^{(1 - x_j^{(i)})} \right] \cdot \pi_c}{\sum_{c} \left[\prod_{j} \theta_{jc}^{x_j^{(i)}} (1 - \theta_{jc}^{x_j^{(i)}})^{(1 - x_j^{(i)})} \cdot \pi_c \right]}$$

$$\log p(c \mid x^{(i)}, \theta, \pi) = \sum_{j} \left[x_j^{(i)} \log \theta_{jc} + (1 - x_j^{(i)}) \log (1 - \theta_{jc})^{1 - x_j^{(i)}} \right] + \log \pi_c$$

$$- \log \sum_{c} \left[\prod_{j} \theta_{jc}^{x_j^{(i)}} (1 - \theta_{jc}^{x_j^{(i)}})^{(1 - x_j^{(i)})} \cdot \pi_c \right]$$

(c) The average log-likelihood per data point cannot be computed, since the probability can be so small such that it is rounded to 0. $\log 0$ is undefined.

(d) Graph in jupyter notebook

(e) $P(\theta) = \text{Beta}(3,3)$

$$P(\theta|X) = \frac{P(X|\theta)\cdot P(\theta)}{P(X)} = \frac{P(X|C,\theta)\cdot P(\theta)}{P(X|C)}$$

$$\text{Beta}(\theta;3,3) = \underbrace{\frac{\Gamma(3+3)}{\Gamma(3)\Gamma(3)}}_{=D}\theta^2(1-\theta)^2$$

$$\frac{\partial \log P(\theta)}{\partial \theta_{ab}} = \frac{\partial \sum_j \sum_c \log \theta_{jc}^2}{\partial \theta_{ab}} + \frac{\partial \sum_j \sum_c \log (1-\theta_{jc})^2}{\partial \theta_{ab}} + \frac{\partial \sum_j \sum_c D}{\partial \theta_{ab}} = 0$$

$$= \frac{2}{\theta_{ab}} + \frac{-2}{1-\theta_{ab}}$$

$$= \frac{2}{\theta_{ab}} - \frac{2}{1-\theta_{ab}}$$

Therefore $\frac{\partial \log P(\theta)}{\partial \theta} = M_2 \in \mathbb{R}^{784\times 10}$ and $M_{2jc} = -1 \quad \forall j, c$

$$\frac{\partial l(\theta|X)}{\partial \theta} = \frac{\partial \log P(\theta|X)}{\partial \theta} = \frac{\partial \log\left[\prod_i P(X^{(i)}|\theta)\cdot P(\theta)/P(X|C)\right]}{\partial \theta}$$

$$= \frac{\partial \sum_i l(X^{(i)}|\theta)}{\partial \theta} + \frac{\partial \log P(\theta)}{\partial \theta}$$

$$= M + \frac{\partial \log P(\theta)}{\partial \theta} = M + M_2$$

According (a) $M \in \mathbb{R}^{784\times 10}$ where $M_{jc} = \frac{\sum_{i=1}^{N} \mathbb{I}[X_j^{(i)} = 1 \,\&\, C^{(i)} = C]}{\theta_{jc}} - \frac{\sum_{i=1}^{N} \mathbb{I}[X_j^{(i)} = 0 \,\&\, C^{(i)} = C]}{1-\theta_{jc}}$

Therefore $\frac{\partial l(\theta|X)}{\partial \theta} = M + M_2 = M_3 \in \mathbb{R}^{784\times 10}$

$$M_{3jc} = 0 \iff M_{jc} + M_{2jc} = 0$$

$$\frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}{\theta_{jc}} - \frac{\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{1-\theta_{jc}} + \frac{2}{\theta_{jc}} - \frac{2}{1-\theta_{jc}} = 0$$

$$\frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}{\theta_{jc}} = \frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{1-\theta_{jc}}$$

$$\frac{1}{\theta_{jc}} - 1 = \frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}$$

$$\frac{1}{\theta_{jc}} = \frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]} + 1$$

$$\theta_{jc} = \frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{4+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c] + \sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=1 \,\&\, c^{(i)}=c]}$$

$$\theta_{jc} = \frac{2+\sum_{i=1}^{N} \mathbb{I}[x_j^{(i)}=0 \,\&\, c^{(i)}=c]}{4+\sum_{i=1}^{N} \mathbb{I}[c^{(i)}=c]}$$

$$\theta_{jc} = \frac{2+\#\text{ of sample where } j\text{th bit is 1 and class is } c}{4+\#\text{ of sample where class is } c}$$

Question 2

(a) True (b) False

$P(x_1, x_2) = P(x_1) \cdot P(x_2)$

$P(x_1, x_2 | c) = P(x_1|c) \cdot P(x_2|c)$

$P(x_1) = \sum_c P(x_1|c)$, $P(x_2) = \sum_c P(x_2|c)$

$P(x_1, x_2) = \sum_c P(x_1, x_2|c)$

$P(x_1) \cdot P(x_2) = \sum_c P(x_1|c) \cdot \sum_c P(x_2|c) > \sum_c P(x_1, x_2|c)$ Thus, (b) is false.

Q1&2

November 14, 2019

[1]:
```
from __future__ import absolute_import
from __future__ import print_function
from future.standard_library import install_aliases
install_aliases()
import numpy as np
import os
import gzip
import struct
import array
import matplotlib.pyplot as plt
import matplotlib.image
from urllib.request import urlretrieve
```

[2]:
```
def download(url, filename):
    if not os.path.exists('data'):
        os.makedirs('data')
    out_file = os.path.join('data', filename)
    if not os.path.isfile(out_file):
        urlretrieve(url, out_file)
```

[3]:
```
def mnist():
    base_url = 'http://yann.lecun.com/exdb/mnist/'

    def parse_labels(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data = struct.unpack(">II", fh.read(8))
            return np.array(array.array("B", fh.read()), dtype=np.uint8)

    def parse_images(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data, rows, cols = struct.unpack(">IIII", fh.read(16))
            return np.array(array.array("B", fh.read()), dtype=np.uint8).
 ↪reshape(num_data, rows, cols)

    for filename in ['train-images-idx3-ubyte.gz',
                     'train-labels-idx1-ubyte.gz',
                     't10k-images-idx3-ubyte.gz',
                     't10k-labels-idx1-ubyte.gz']:
```

```
        download(base_url + filename, filename)

    train_images = parse_images('data/train-images-idx3-ubyte.gz')
    train_labels = parse_labels('data/train-labels-idx1-ubyte.gz')
    test_images = parse_images('data/t10k-images-idx3-ubyte.gz')
    test_labels = parse_labels('data/t10k-labels-idx1-ubyte.gz')

    return train_images, train_labels, test_images[:1000], test_labels[:1000]
```

[4]:
```
def load_mnist():
    partial_flatten = lambda x: np.reshape(x, (x.shape[0], np.prod(x.shape[1:
↪])))
    one_hot = lambda x, k: np.array(x[:, None] == np.arange(k)[None, :],␣
↪dtype=int)
    train_images, train_labels, test_images, test_labels = mnist()
    train_images = (partial_flatten(train_images) / 255.0 > .5).astype(float)
    test_images = (partial_flatten(test_images) / 255.0 > .5).astype(float)
    train_labels = one_hot(train_labels, 10)
    test_labels = one_hot(test_labels, 10)
    N_data = train_images.shape[0]

    return N_data, train_images, train_labels, test_images, test_labels
```

[5]:
```
def plot_images(images, ax, ims_per_row=5, padding=5, digit_dimensions=(28, 28),
                cmap=matplotlib.cm.binary, vmin=None, vmax=None):
    """Images should be a (N_images x pixels) matrix."""
    N_images = images.shape[0]
    N_rows = np.int32(np.ceil(float(N_images) / ims_per_row))
    pad_value = np.min(images.ravel())
    concat_images = np.full(((digit_dimensions[0] + padding) * N_rows + padding,
                             (digit_dimensions[1] + padding) * ims_per_row +␣
↪padding), pad_value)
    for i in range(N_images):
        cur_image = np.reshape(images[i, :], digit_dimensions)
        row_ix = i // ims_per_row
        col_ix = i % ims_per_row
        row_start = padding + (padding + digit_dimensions[0]) * row_ix
        col_start = padding + (padding + digit_dimensions[1]) * col_ix
        concat_images[row_start: row_start + digit_dimensions[0],
                      col_start: col_start + digit_dimensions[1]] = cur_image
        cax = ax.matshow(concat_images, cmap=cmap, vmin=vmin, vmax=vmax)
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
    return cax
```

[6]:
```
def save_images(images, filename, **kwargs):
    fig = plt.figure(1)
    fig.clf()
```

```
    ax = fig.add_subplot(111)
    plot_images(images, ax, **kwargs)
    fig.patch.set_visible(False)
    ax.patch.set_visible(False)
    plt.savefig(filename)
```

[7]:
```
def train_mle_estimator(train_images, train_labels):
    """ Inputs: train_images, train_labels
        Returns the MLE estimators theta_mle and pi_mle"""

    # YOU NEED TO WRITE THIS PART
    sample_per_class = np.sum(train_labels, axis=0)
    pi_mle = sample_per_class / float(train_labels.shape[0])

    theta_mle = train_images.transpose() @ train_labels

    theta_mle /= sample_per_class
    return theta_mle, pi_mle
```

[8]:
```
def train_map_estimator(train_images, train_labels):
    """ Inputs: train_images, train_labels
        Returns the MAP estimators theta_map and pi_map"""

    # YOU NEED TO WRITE THIS PART
    sample_per_class = np.sum(train_labels, axis=0)
    pi_mle = sample_per_class / float(train_labels.shape[0])

    sample_per_class += 4
    theta_mle = train_images.transpose() @ train_labels + 2
    theta_mle /= sample_per_class

    return theta_mle, pi_mle
```

[9]:
```
def log_likelihood(images, theta, pi):
    """ Inputs: images, theta, pi
        Returns the matrix 'log_like' of loglikehoods over the input images␣
↪where
    log_like[i,c] = log p (c |x^(i), theta, pi) using the estimators theta and␣
↪pi.
    log_like is a matrix of num of images x num of classes
    Note that log likelihood is not only for c^(i), it is for all possible c's.
↪"""

    # YOU NEED TO WRITE THIS PART
    ce = (images @ np.log(theta)) + ((1 - images) @ np.log(1 - theta)) + np.
↪log(pi)
    pd = np.zeros(shape=(images.shape[0], 1))
```

```
    for i in range(images.shape[0]):
        x = images[i].reshape((images.shape[1],1))
        pd[i] = np.sum(np.prod(theta * x + (1 - theta) * (1 - x), axis=0) * pi)

    return ce - np.log(pd)
```

[10]:
```
def predict(log_like):
    """ Inputs: matrix of log likelihoods
    Returns the predictions based on log likelihood values"""
    predictions = (log_like == np.max(log_like, axis=1, keepdims=1)).astype(int)
    # YOU NEED TO WRITE THIS PART
    return predictions
```

[11]:
```
def accuracy(log_like, labels):
    """ Inputs: matrix of log likelihoods and 1-of-K labels
    Returns the accuracy based on predictions from log likelihood values"""
    pred = predict(log_like)
    accuracy = np.sum(np.all(pred == labels, axis=1)) / labels.shape[0]
    # YOU NEED TO WRITE THIS PART
    return accuracy
```

[12]:
```
def image_sampler(theta, pi, num_images):
    """ Inputs: parameters theta and pi, and number of images to sample
    Returns the sampled images"""
    labels = np.random.choice(a=len(pi),size=num_images,p=pi)
    probs = theta[:,labels].transpose()
    sampled_images = np.random.binomial(1, probs)
    # YOU NEED TO WRITE THIS PART
    return sampled_images
```

0.0.1 Train with naive bayes model

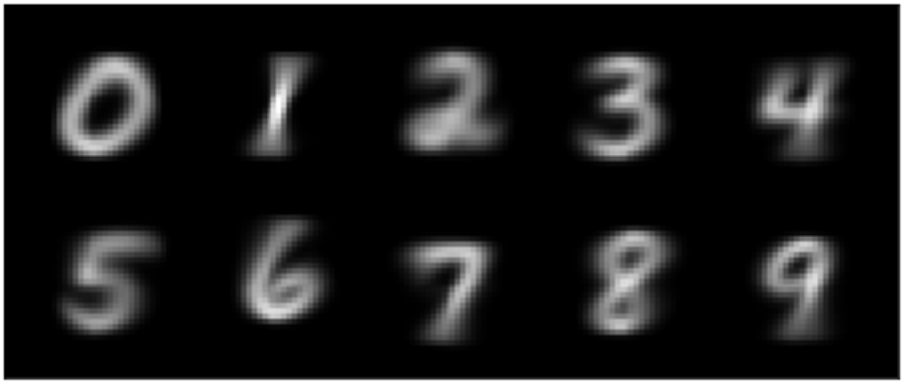
[13]:
```
N_data, train_images, train_labels, test_images, test_labels = load_mnist()
# Fit MLE and MAP estimators
theta_mle, pi_mle = train_mle_estimator(train_images, train_labels)
theta_map, pi_map = train_map_estimator(train_images, train_labels)

# Find the log likelihood of each data point
loglike_train_mle = log_likelihood(train_images, theta_mle, pi_mle)
loglike_train_map = log_likelihood(train_images, theta_map, pi_map)

avg_loglike_mle = np.sum(loglike_train_mle * train_labels) / N_data
avg_loglike_map = np.sum(loglike_train_map * train_labels) / N_data

print("Average log-likelihood for MLE is ", avg_loglike_mle)
print("Average log-likelihood for MAP is ", avg_loglike_map)

train_accuracy_map = accuracy(loglike_train_map, train_labels)
```

```
loglike_test_map = log_likelihood(test_images, theta_map, pi_map)
test_accuracy_map = accuracy(loglike_test_map, test_labels)

print("Training accuracy for MAP is ", train_accuracy_map)
print("Test accuracy for MAP is ", test_accuracy_map)
```

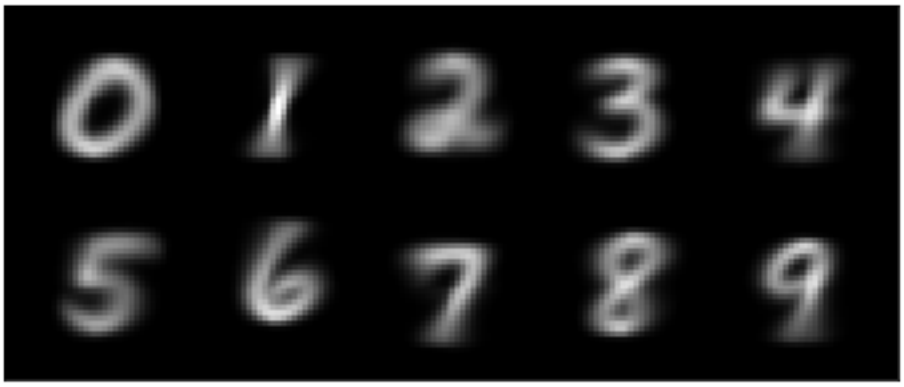
```
/home/lichiheng/.conda/envs/csc311env/lib/python3.7/site-
packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in
log
  if __name__ == '__main__':
/home/lichiheng/.conda/envs/csc311env/lib/python3.7/site-
packages/ipykernel_launcher.py:9: RuntimeWarning: invalid value encountered in
matmul
  if __name__ == '__main__':

Average log-likelihood for MLE is  nan
Average log-likelihood for MAP is  -3.357063137860309
Training accuracy for MAP is  0.8352166666666667
Test accuracy for MAP is  0.816
```

0.0.2 Plot mle and map estimators

[14]:
```
# Plot MLE estimators
save_images(theta_mle.T, 'mle.png', cmap='gray')
```

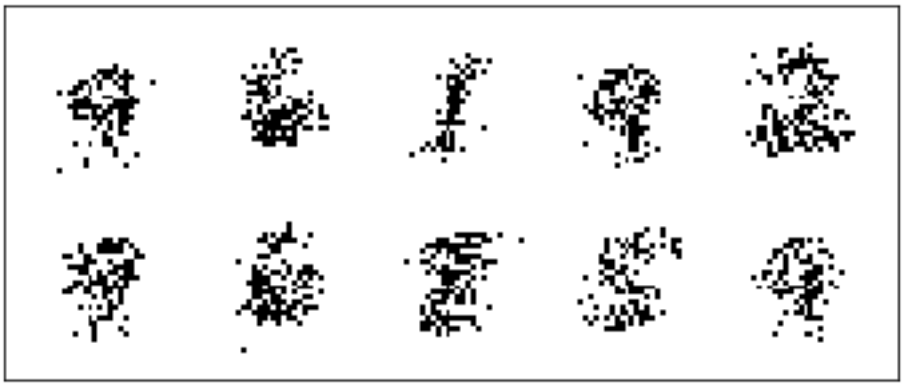


[15]:
```
# Plot MAP estimators
save_images(theta_map.T, 'map.png', cmap='gray')
```



0.0.3 Generate samples from the generative model

[16]:
```
# Sample 10 images
sampled_images = image_sampler(theta_map, pi_map, 10)
save_images(sampled_images, 'sampled_images.png')
```



Q3

November 14, 2019

0.0.1 Load all handwritten digit

[1]:
```
from utils import load_data
```

[2]:
```
X_train, X_valid, X_test, y_train, y_valid, y_test = load_data("digits.npz")
```

0.0.2 Compute covariance matrix

[3]:
```
import numpy as np
```

[4]:
```
S = np.cov(X_train.T, bias=True)
```

[5]:
```
from numpy import linalg as LA
```

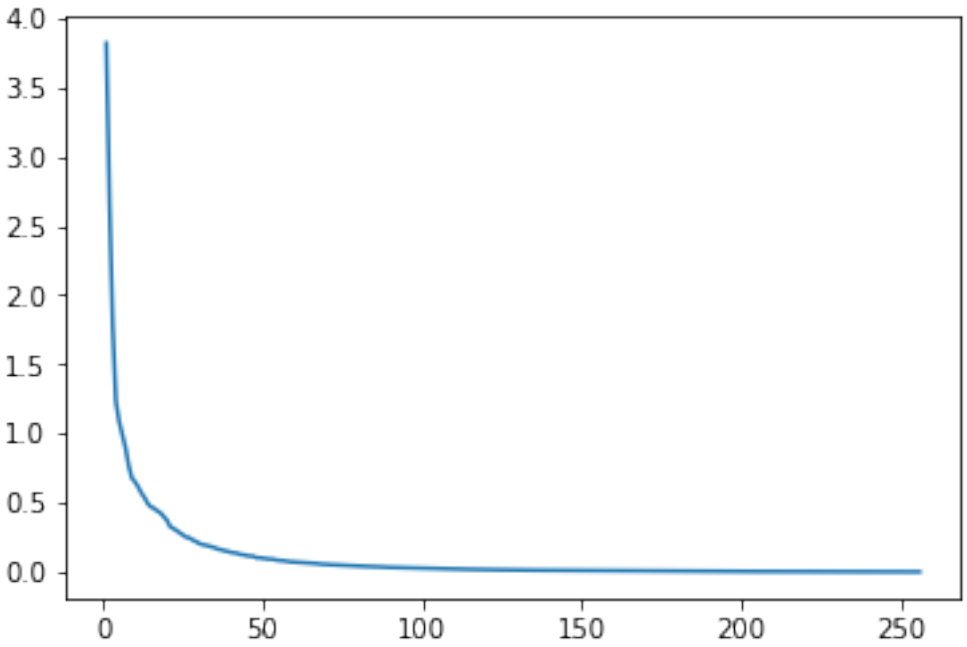
[6]:
```
w, v = LA.eig(S)
arg_w = w.argsort()[::-1]
x = np.arange(1, w.shape[0] + 1)
```

0.0.3 Plot the sorted eigen values

[7]:
```
import matplotlib.pyplot as plt
```

[8]:
```
plt.plot(x, w[arg_w])
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f635ba71e90>]
```



0.0.4 Plot the first three eigen-images and mean

[9]:
```
def plot_images(images, ax, ims_per_row=5, padding=5, digit_dimensions=(28, 28),
                cmap=plt.cm.binary, vmin=None, vmax=None):
    """Images should be a (N_images x pixels) matrix."""
    N_images = images.shape[0]
    N_rows = np.int32(np.ceil(float(N_images) / ims_per_row))
    pad_value = np.min(images.ravel())
    concat_images = np.full(((digit_dimensions[0] + padding) * N_rows + padding,
                             (digit_dimensions[1] + padding) * ims_per_row +␣
↪padding), pad_value)
    for i in range(N_images):
        cur_image = np.reshape(images[i, :], digit_dimensions, order='F')
        row_ix = i // ims_per_row
        col_ix = i % ims_per_row
        row_start = padding + (padding + digit_dimensions[0]) * row_ix
        col_start = padding + (padding + digit_dimensions[1]) * col_ix
        concat_images[row_start: row_start + digit_dimensions[0],
                      col_start: col_start + digit_dimensions[1]] = cur_image
    cax = ax.matshow(concat_images, cmap=cmap, vmin=vmin, vmax=vmax)
    return cax
```

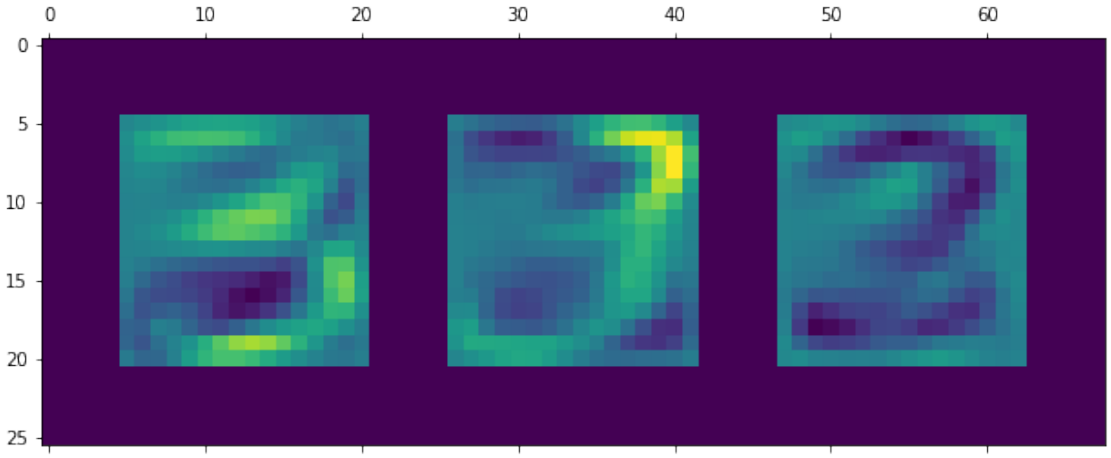
[10]:
```
# Get the first three eigen vectors
first_3 = v[:,arg_w[:3]].T
```

[11]:
```
first_3.shape
```

[11]:
```
(3, 256)
```

[12]:
```
%matplotlib inline
plot_images(first_3, plt, ims_per_row=3, digit_dimensions=(16, 16),␣
 ↪cmap='viridis')
```

[12]:
```
<matplotlib.image.AxesImage at 0x7f635b67d8d0>
```



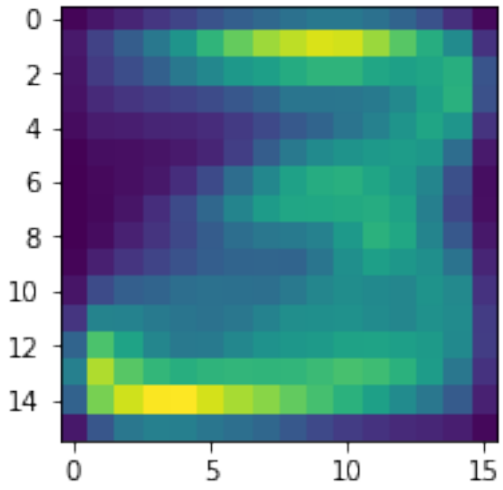
[13]:
```
# Derive the mean
mean = np.mean(X_train, axis=0)
```

[14]:
```
mean.shape
```

[14]:
```
(256,)
```

[15]:
```
plt.figure(figsize = (10,3))
plt.imshow(mean.reshape((16, 16), order='F'))
```

[15]:
```
<matplotlib.image.AxesImage at 0x7f635b432990>
```



0.0.5 Q3 (a) Derive the code vectors

[16]:
```
def predict_1NN(X_train, y_train, X):
    nn = []
    for x in X:
        nn.append(LA.norm(X_train - x, axis=1).argmin())
    return y_train[nn]
```

[17]:
```
def accuracy(y_predict, y):
    return np.sum(y_predict == y) / float(len(y))
```

[18]:
```
# Initialize all dimensions
dims = [2, 5, 10, 20, 30]
acc = []
```

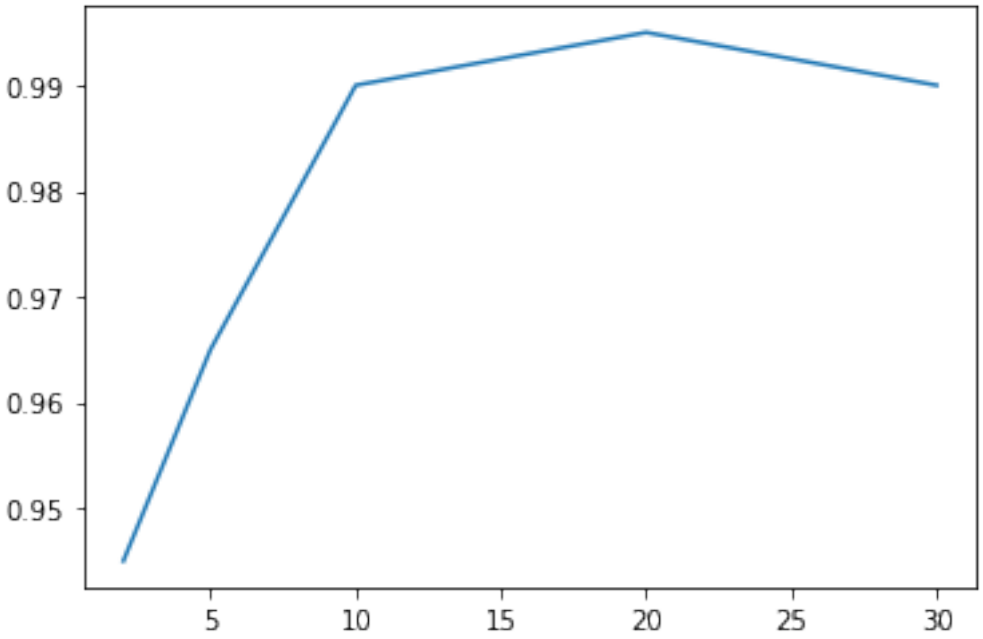
0.0.6 Train with 1NN on the code vectors

[19]:
```
for dim in dims:
    # Center the data
    Xt_c, Xv_c = X_train - mean, X_valid - mean
    # Project on lower dimensional space
    basis = v[:, arg_w[: dim]]
    Xt_l, Xv_l = Xt_c @ basis, Xv_c @ basis
    # Predict the labels of validation set
    yv_predict = predict_1NN(Xt_l, y_train, Xv_l)
    # Evaluate performance
    acc.append(accuracy(yv_predict, y_valid))
```

0.0.7 Plot the performances with respect to dimensions

[20]:
```
%matplotlib inline
plt.plot(dims, acc)
```

[20]:
```
[<matplotlib.lines.Line2D at 0x7f635b3a5090>]
```



0.0.8 Q3 (b)

I would choose 20 top eigenvectors, since it reduces the dimensions of the features from 30 to 20. It reduces the cost of the training process and at the same time achieves the highest score on the validation set.

0.0.9 Q3 (c)

[21]:
```
# Center the data
Xtest_c = X_test - mean
# Project on lower dimensional space
basis = v[:, arg_w[: 20]]
Xtest_l, Xt_l = Xtest_c @ basis, Xt_c @ basis
# Predict the labels of validation set
yt_predict = predict_1NN(Xt_l, y_train, Xtest_l)
```

0.0.10 Performance of the final classifier

[22]:
```
accuracy(yt_predict, y_test)
```

[22]:
```
0.99
```