Assignment 3

1.

(a)

Derivation for $\hat{\boldsymbol{\pi}}_{MLE}$ and $\hat{\boldsymbol{\theta}}_{MLE}$:

1. (a) MLE:

$$L = \prod_{i=1}^{N}\left(\pi_0^{t_0^{(i)}} \pi_1^{t_1^{(i)}} \cdots \pi_8^{t_8^{(i)}} \left(1-\sum_{k=0}^{8}\pi_k\right)^{t_9^{(i)}} \prod_{j=1}^{784} \theta_{jk}^{x_j^{(i)}} (1-\theta_{jk})^{1-x_j^{(i)}}\right), \quad t_k^{(i)}=1$$

$$l = \log L = \sum_{i=1}^{N}\left(t_0^{(i)}\log\pi_0 + t_1^{(i)}\log\pi_1 + \cdots + t_9^{(i)}\log\left(1-\sum_{k=0}^{8}\pi_k\right)\right)$$
$$+ \sum_{i=1}^{N}\sum_{j=1}^{784} x_j^{(i)}\log\theta_{jk} + (1-x_j^{(i)})\log(1-\theta_{jk}), \quad t_k^{(i)}=1$$

Let $l_1 = \sum_{i=1}^{N} t_0^{(i)}\log\pi_0 + t_1^{(i)}\log\pi_1 + \cdots + t_9^{(i)}\log\left(1-\sum_{k=0}^{8}\pi_k\right)$,

$l_2 = \sum_{i=1}^{N}\sum_{j=1}^{784} x_j^{(i)}\log\theta_{jk} + (1-x_j^{(i)})\log(1-\theta_{jk})$

Then $l = l_1 + l_2$

To find $\hat{\pi}$, we set $\frac{dl_1}{d\pi_k}=0$ for $k=0,1,\cdots,9$

$$\frac{dl_1}{d\pi_k} = \sum_{i=1}^{N}\frac{t_k^{(i)}}{\pi_k} - \frac{t_9^{(i)}}{1-\sum_{k=0}^{8}\pi_k} = 0, \quad \text{let } 1-\sum_{k=0}^{8}\pi_k = \pi_9$$

$$\therefore \sum_{i=1}^{N}\frac{\pi_9 t_k^{(i)} - \pi_k t_9^{(i)}}{\pi_k\pi_9} = 0 \Rightarrow \frac{\hat{\pi}_k}{\hat{\pi}_9} = \frac{\sum_{i=1}^{N} t_k^{(i)}}{\sum_{i=1}^{N} t_9^{(i)}}$$

$$\hat{\pi}_k = \frac{\sum_{i=1}^{N} t_k^{(i)}}{\sum_{i=1}^{N} t_9^{(i)}}\hat{\pi}_9$$

As $\hat{\pi}_0 + \hat{\pi}_1 + \cdots + \hat{\pi}_9 = 1$, we have

$$\left(\frac{\sum_{k=0}^{8}\sum_{i=1}^{N} t_j^{(i)}}{\sum_{i=1}^{N} t_9^{(i)}} + 1\right)\hat{\pi}_9 = 1 \Rightarrow \hat{\pi}_9 = \frac{\sum_{i=1}^{N} t_9^{(i)}}{\sum_{j=0}^{9}\sum_{i=1}^{N} t_j^{(i)}}$$

$$\text{for } j=0\sim 8: \quad \hat{\pi}_j = \frac{\cancel{\sum_{i=1}^{N} t_j^{(i)}}}{\cancel{\sum t_9^{(i)}}} \cdot \frac{\cancel{\sum_{i=1}^{N} t_9^{(i)}}}{\cancel{\sum\sum t_j^{(i)}}} = \boxed{\frac{\sum_{i=1}^{N} t_j^{(i)}}{\sum_{j=0}^{9}\sum_{i=1}^{N} t_j^{(i)}} = \frac{\text{sum of column } t_k}{\text{sum of all values in } t}}$$

To find $\hat{\theta}$, we set $\frac{dl_2}{d\theta_{jk}}=0$, for $j=0,1,\cdots,784$ and $k=0,\cdots,9$

$$\frac{dl_2}{d\theta_{jk}} = \frac{\sum_{i=1}^{N} x_j^{(i)}}{\theta_{jk}} - \frac{\sum_{i=1}^{N} 1-x_j^{(i)}}{1-\theta_{jk}}\,(t_k^{(i)}=1) = 0$$

$$\frac{\sum_{i=1}^{N} x_j^{(i)} - \theta_{jk}\sum_{i=1}^{N} x_j^{(i)} - \theta_{jk}\sum_{i=1}^{N} 1 + \theta_{jk}\sum_{i=1}^{N} x_j^{(i)}}{\theta_{jk}(1-\theta_{jk})} = 0 \quad (t_k^{(i)}=1)$$

$$\therefore \sum_{i=1}^{N} x_j^{(i)} = \theta_{jk}\cdot\sum_{i=1}^{N} 1, \quad (t_k^{(i)}=1)$$

$$= \theta_{jk}\cdot N'$$

$$\therefore \hat{\theta}_{jk} = \boxed{\frac{\sum_{i=1}^{N} x_j^{(i)}}{N'} = \frac{\#\text{of image with } x_j=1 \text{ in class } k}{\#\text{of image in class } k}}$$

Code for (a):

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

    pi_mle = sum(train_labels) / sum(sum(train_labels))
    ones = np.ones((train_images.shape[1], train_labels.shape[0]))
    theta_mle = (train_images.T.dot(train_labels)) / ones.dot(train_labels)

    # This is not a good algorithm.
    # theta = []
    # for i in range(train_labels.shape[1]):  # 10
    #     class_sum = sum(train_labels)[i]
    #     class_column = []
    #     for j in range(train_images.shape[1]):  # 784
    #         sum_feature_j = 0
    #         for k in range(train_images.shape[0]):  # 60000
    #             if train_labels[k][i] == 1:
    #                 sum_feature_j += train_images[k][j]
    #         class_column.append(sum_feature_j / class_sum)
    #     theta.append(class_column)
    # theta_mle = np.array(t截图(Alt + A)

    return theta_mle, pi_mle
```

(b)

$$\text{(b)} \log p(t^{(i)}|x^{(i)},\theta,\pi) \propto \log(t^{(i)},x^{(i)}|\theta,\pi)$$

$$= (t^{(i)})^T\log\pi + (x^{(i)})^T\log\theta + (1-x^{(i)})^T\log(1-\theta)$$

784 × 10 matrix

(c)

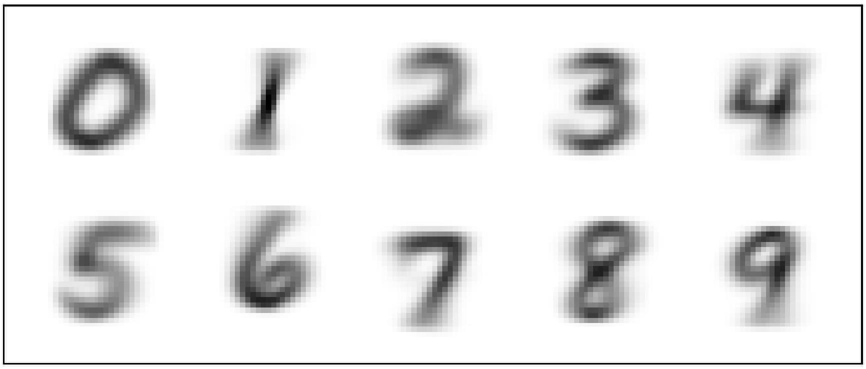
We cannot compute the average for $\hat{\boldsymbol{\theta}}_{\boldsymbol{MLE}}$ since there are several entries in $\hat{\boldsymbol{\theta}}_{\boldsymbol{MLE}}$ matrix are 0s. For example, the first 10 rows in $\hat{\boldsymbol{\theta}}_{\boldsymbol{MLE}}$ are all 0s. Our algorithm will have to compute $\log(\hat{\boldsymbol{\theta}}_{\boldsymbol{MLE}})$ for all entries, since log(0) is not defined, we would get null value as output. Having 0s in MLE estimators in first 10 rows of $\hat{\boldsymbol{\theta}}_{\boldsymbol{MLE}}$ seems reasonable in this case, since in a 28*28-pixel image, the first 10 pixels in the first line seldom contribute to image classification as they are usually white as edge.

Code for (c):

```
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."""

    factor_matrix = np.zeros((pi.shape[0], pi.shape[0]))
    np.fill_diagonal(factor_matrix, np.log(pi))
    log_like = images.dot(np.log(theta)) + (1 - images).dot(np.log(1 - theta)) + \
        np.ones((images.shape[0], theta.shape[1])).dot(factor_matrix)

    return log_like
```

```
C:/University/2019 Fall/CSC311/A3/naive_bayes.py:155: RuntimeWarning: divide by zero encountered in log
  log_like = images.dot(np.log(theta)) + (1 - images).dot(np.log(1 - theta)) + \
Average log-likelihood for MLE is  nan
```

(d)



(e)

(e) Since no prior setting for $\hat{\pi}$, $\hat{\pi}_{MAP} = \hat{\pi}_{MLE}$
for $\hat{\theta}_{MAP}$, $l_2' = \theta_{jk}^2 (1-\theta_{jk})^2 l_2$, since $\theta_{jk} \in Beta(3,3)$
and we ignored the normalzing constant.

$$\frac{dl_2'}{d\theta_{jk}} = \frac{2+\sum_{i=1}^{N} x_j^{(i)}}{\theta_{jk}} - \frac{2+\sum_{i=1}^{N} 1-x_j^{(i)}}{1-\theta_{jk}} = 0, \quad (t_k^{(i)}=1)$$

$$\Rightarrow 2+\sum_{i=1}^{N} x_j^{(i)} = (4+\sum_{i=1}^{N} 1)\hat{\theta}_{jk}, \quad (t_k^{(i)}=1)$$

$$\therefore \hat{\theta}_{jk_{MAP}} = \boxed{\frac{2+\sum_{i=1}^{N} x_j^{(i)}}{4+N'} = \frac{2+\#\text{ of images of class } k \text{ with } x_j=1}{4+\#\text{ of images of class } k}}$$

Code for (e):

naive_bayes.py × A3_Q3.py ×

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

    pi_map = sum(train_labels) / sum(sum(train_labels))
    ones = np.ones((train_images.shape[1], train_labels.shape[0]))
    theta_map = ((train_images.T.dot(train_labels)) + 2) / (ones.dot(train_labels) + 4)

    # This is not a good algorithm.
    # theta = []
    # for i in range(train_labels.shape[1]):  # 10
    #     class_sum = sum(train_labels)[i]
    #     class_column = []
    #     for j in range(train_images.shape[1]):  # 784
    #         sum_feature_j = 0
    #         for k in range(train_images.shape[0]):  # 60000
    #             if train_labels[k][i] == 1:
    #                 sum_feature_j += train_images[k][j]
    #         class_column.append((sum_feature_j + 2) / (class_sum + 4))
    #     theta.append(class_column)
    # theta_map = np.array(theta).T

    return theta_map, pi_map
```

(f)

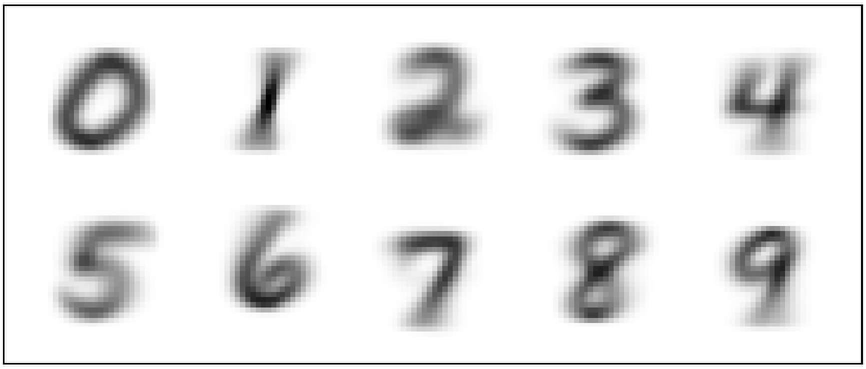
Code for (f):

```python
def predict(log_like):
    """ Inputs: matrix of log likelihoods
    Returns the predictions based on log likelihood values"""

    predictions = np.argmax(log_like, axis=1)

    return predictions


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"""

    acc = np.mean(predict(log_like) == np.argmax(labels, axis=1))

    return acc
```

```
Average log-likelihood for MAP is  -173.46659721464317
Training accuracy for MAP is  0.8352166666666667
Test accuracy for MAP is  0.816

Process finished with exit code 0
```

Accuracy for train data is about 83.5% and accuracy for test data is about 81.6%.

(g)



2.

(a)

True. According to the definition of Naïve Bayes Model, x_i and x_j are independent under the conditional distribution $p(x|c)$.

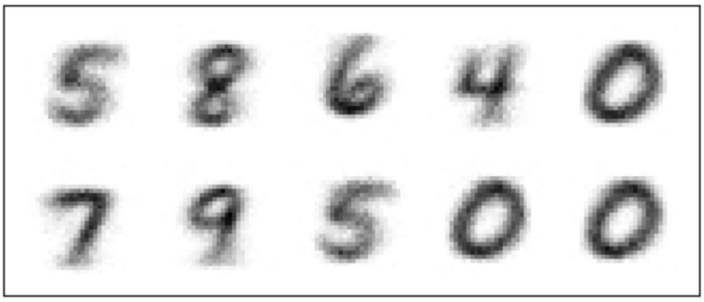
(b)

False. The model assumption only specifies that $X_i \mid c$ are independent. This does dot necessary implies X_i's are independent. As a counter example, consider X_1 is the random variable of weather being cloudy tomorrow and X_2 is the random variable of weather being rainy tomorrow. Suppose c is the information that the forecast says tomorrow is sunny. Knowing the probability of cloudy tomorrow under the sunny weather forecast does not help me to find the probability of tomorrow being rainy. So, they are conditionally independent. However, if not given the forecast, cloudy and rainy are related somehow as cloudy days have more chance to rain. Thus, they are not independent.

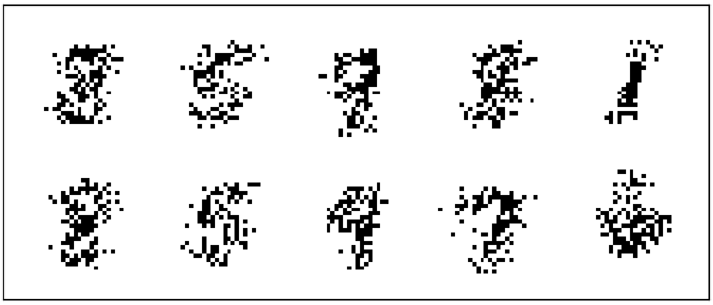
(c)

Code for (c):

```python
def image_sampler(theta, pi, num_images):
    """ Inputs: parameters theta and pi, and number of images to sample
    Returns the sampled images"""

    random_classes = np.random.choice(10, num_images, p=pi)
    random_probs = theta.T[random_classes]
    sampled_images = np.random.binomial(100, random_probs)

    return sampled_images
```



If we change the 100 in np.random.binomial's parameter to 1, we will have the following sampled image:



With parameter 1, we generate images with either value of 1 or 0 on each pixel. This makes the graph stronger in color difference, but less recognizable. Changing the parameter from 1 to 100 or 200 etc. will fit the mean of all experiments of the single pixel with the given = 1 probability each time. This makes plots greyer, since they are not all black(pixel = 1) and white(pixel = 0). When the number is adequately large, the plot of each class will converge to the plot we got in 1(d) and 1(g) since the mean of large experiments should be the probability of = 1 probability each time for binomial distribution.

3.

(a)

Code for (a) (We modified the code given in A2 and used it as 1-NN classifier):

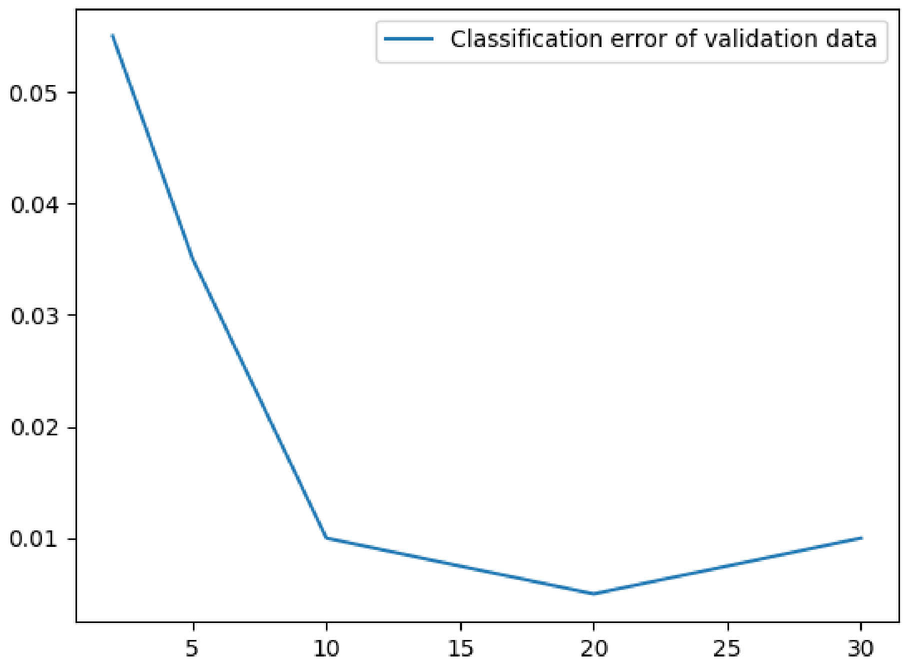
naive_bayes.py × A3_Q3.py ×

```python
from utils import *
import matplotlib.pyplot as plt


def project_to_train(k, inputs_data, inputs_train_data):

    mean_train = np.mean(inputs_train_data, axis=0)
    cov_train = (inputs_train_data - mean_train).T.dot((inputs_train_data - mean_train))

    e_values, e_vectors = np.linalg.eig(cov_train)
    max_k_e_vectors = e_vectors.T[np.argsort(e_values)[-k:]]

    centered_inputs_train = inputs_train_data - mean_train
    centered_inputs = inputs_data - mean_train

    projection_input = centered_inputs.dot(max_k_e_vectors.T)
    projection_train = centered_inputs_train.dot(max_k_e_vectors.T)

    return projection_input, projection_train

```

naive_bayes.py × A3_Q3.py ×

```python
def l2_distance(a, b):

    if a.shape[0] != b.shape[0]:
        raise ValueError("A and B should be of same dimensionality")

    aa = np.sum(a**2, axis=0)
    bb = np.sum(b**2, axis=0)
    ab = np.dot(a.T, b)

    return np.sqrt(aa[:, np.newaxis] + bb[np.newaxis, :] - 2*ab)


def run_1nn(train_data, train_labels, valid_data):
    dist = l2_distance(valid_data.T, train_data.T)
    nearest = np.argsort(dist, axis=1)[:, :1]

    train_labels = train_labels.reshape(-1)
    valid_labels = train_labels[nearest]

    # valid_labels = (np.mean(valid_labels, axis=1) >= 0.5).astype(np.int)
    # valid_labels = valid_labels.reshape(-1, 1)

    return valid_labels
```

```python
def accuracy(predict_labels, actual_labels):
    acc = np.mean(predict_labels == actual_labels)

    return acc


if __name__ == '__main__':
    inputs_train, inputs_valid, inputs_test, target_train, \
        target_valid, target_test = load_data("digits.npz")

    valid_accuracy = []
    k_values = [2, 5, 10, 20, 30]
    for i in k_values:
        project_valid, project_train = project_to_train(i, inputs_valid, inputs_train)
        predict_valid_labels = run_1nn(project_train, target_train, project_valid)
        validation_accuracy = accuracy(predict_valid_labels, target_valid)
        valid_accuracy.append(validation_accuracy)

        print("Validation set classification accuracy for k =", i, " is ", validation_accuracy)

    plt.scatter(np.array([2, 5, 10, 20, 30]), 1 - np.array(valid_accuracy))
    plt.legend(['Classification error of validation data'])
    plt.show()
```



(b)

From the previous plot, we see that classification error for k = 20 is the minimum. Based on the information we have so far, I will take k = 20 to be the dimension of the low dimension subspace for projection. If I had tested more k values, mu final k should be in the range of (10, 25).

(c)

K = 20:

```
Test set classification accuracy for k = 20  is  0.99
```