STA414 hw2

1001386644

Tae Hoon Jun

1. A) $\sum_d [x_d ln\theta_{cd} + (1-x_d)\ln(1-\theta_{cd})] = 0$

$\Rightarrow \sum_d \left(\frac{x_d}{\theta_{cd}} - \frac{1-x_d}{1-\theta_{cd}}\right) = 0$

$\Rightarrow \sum_d (1-\theta_{cd})x_d = \sum_d (1-x_d)\theta_{cd}$

$\Rightarrow \sum_d \left(\frac{1}{\theta_{cd}} - 1\right) x_d = \sum_d (1-x_d)$

$\Rightarrow \sum_d^{784} x_d = \sum_d^{784} \theta_{cd}$

$\Rightarrow \widehat{\theta_{cd}} = \frac{1}{784} \sum_d^{784} x_d$

B) $p(\theta|\beta) = \frac{p(\beta|\theta)p(\theta)}{p(\beta)} = p(\beta|\theta)p(\theta)$

only need to consider this since $p(\beta)$ doesn't contain theta

$\ln p(\theta_{cd}|x_d) = \sum_d lnp(x_d|\theta_{cd}) + lnp(\theta_{cd}|2{,}2)$

$\frac{\partial}{\partial\theta_{cd}} lnp(\theta_{cd}|x_d) = \frac{\partial}{\partial\theta_{cd}} \sum_d lnp(x_d|\theta_{cd}) + \frac{\partial}{\partial\theta_{cd}} lnBeta(\theta_{cd}|2{,}2)$

$\Rightarrow 0 = \frac{1}{\theta_{cd}} \sum_d x_d - \frac{1}{1-\theta_{cd}} \sum_d (1-x_d) + \frac{1}{\theta_{cd}} - \frac{1}{1-\theta_{cd}}$

$\Rightarrow \theta_{cd}[\sum_d (1-x_d) + 1] = (1-\theta_{cd})[\sum_d x_d + 1]$

$\Rightarrow \theta_{cd}[\sum_d^{784} 1 + 2] = \sum_d x_d + 1$

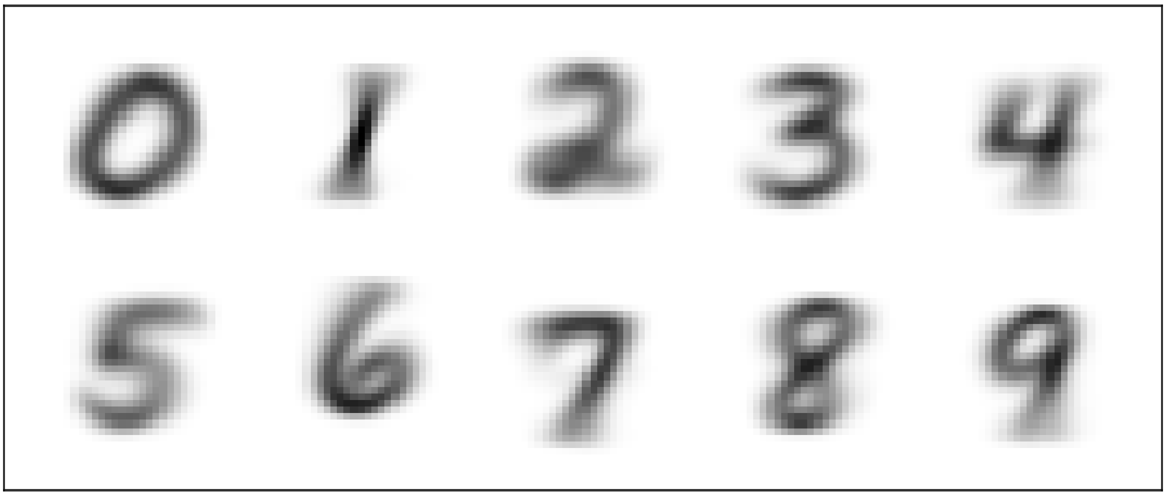
$\Rightarrow \widehat{\theta_{cd_{MAP}}} = (\sum_d x_d + 1)/786$

C)

```
def fit_MAP(train_images, train_labels):
    #train_images = binarize(train_images)
    c = 10
    n, d = train_images.shape
    x = np.zeros((c, d))
    for i in range(n):
        for j in range(c):
            if train_labels[i][j] == 1:
                x[j] += train_images[i]
    #number of c's
    total_c = np.reshape(np.sum(train_labels, axis=0), (c, 1))
    theta = (x+1)/(total_c+2)
    plot_images(theta, plt)
    return theta
```

```python
def binarize(train_images):
    return np.where(train_images > 0.5, 1, 0)
```

```python
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    train_images = binarize(train_images)
    #test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
```



D) $p(c|x) = \frac{p(X|C)p(c)}{p(x)}$

$$= (\pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} (1-\theta_{cd})^{1-x_d}) / (\sum_{c=0}^{9} \pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} (1-\theta_{cd})^{1-x_d})$$

$$= (\pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} (1-\theta_{cd})^{1-x_d}) / (\sum_{c=0}^{9} \exp(\ln(\pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} (1-\theta_{cd})^{1-x_d})))$$

$$\text{Ln}\, p(c|x) = \ln(\pi_c) + \sum_{d}^{784} [x_d ln\theta_{cd} + (1-x_d)\ln(1-\theta_{cd})]$$

$$- \ln \sum_{c=1}^{9} \exp(\ln(\pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} (1-\theta_{cd})^{1-x_d}))$$

E)

```python
def log_likelihood(x, theta, pi, top=False):
    bern = np.where(x > 0.5, theta, 1-theta)
    #for calculating p(c|x_top)
    if (top):
        bern = bern[:392]
    #sum all the pixels
    numer = np.log(pi) + np.sum(np.log(bern), axis=1)
    denom = logsumexp(numer)
    return numer - denom

def avg_log_likelihood(images, theta, labels, pi=0.1):
    N = len(images)
    summ = 0
    for i in range(N):
        likelihood = log_likelihood(images[i], theta, pi)
        t_i = np.argmax(labels[i])
        summ += likelihood[t_i]
    return summ/N

def accuracy(images, theta, labels, pi=0.1):
    score = 0
    N = len(images)
    for i in range(N):
        likelihood = log_likelihood(images[i], theta, pi)
        c_i = np.argmax(likelihood)
        t_i = np.argmax(labels[i])
        if (c_i == t_i):
            score += 1

    return score/float(N)

```

```python
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    train_images = binarize(train_images)
    test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
    #1_e
    print(avg_log_likelihood(train_images, theta, train_labels))
    print(accuracy(train_images, theta, train_labels))
    print(avg_log_likelihood(test_images, theta, test_labels))
    print(accuracy(test_images, theta, test_labels))
```

Average log likelihood for training data: -3.354047795346782

Accuracy of training data: 0.835883333333

Average log likelihood for test data: -3.180015171715445

Accuracy of test data: 0.8426

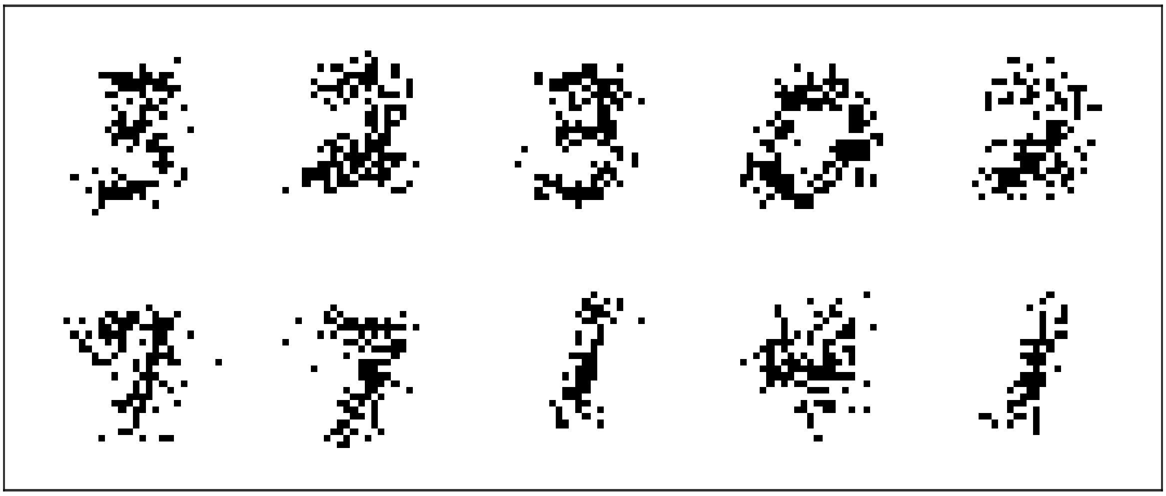
2. A) True

   B) False

   C)

```python
def random_image_samples(images, theta, labels, size):
    d = theta.shape[1]
    #Distribution of digits
    prob = np.sum(labels, axis=0)
    prob = prob/float(labels.shape[0])
    classes = np.array(range(10))
    sample_c = np.random.choice(classes, size, p=prob)
    result = []
    for digit in sample_c:
        image = []
        for i in range(d):
            P = [1-theta[digit][i], theta[digit][i]]
            random_pixel = np.random.choice([0,1],1, p=P)
            image.append(random_pixel)
        image - np.round(image)
        result.append(image)
    plot_images(np.array(result), plt)
```

```python
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    train_images = binarize(train_images)
    test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
    #1_e
    #print(avg_log_likelihood(train_images, theta, train_labels))
    #print(accuracy(train_images, theta, train_labels))
    #print(avg_log_likelihood(test_images, theta, test_labels))
    #print(accuracy(test_images, theta, test_labels))
    #2_c
    random_image_samples(train_images, theta, train_labels, 10)
```



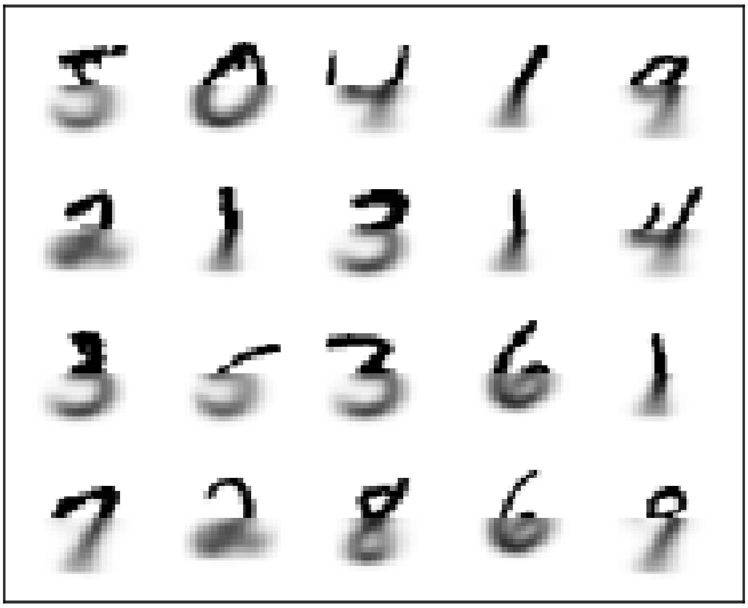
D) $p(x_{i\in\text{bottom}}|x_{top},\theta,\pi) = \sum_{c=0}^{9} p(c|x_{top})p(x_{i\in bottom}|c)$

$$= \sum_{c=0}^{9}(\pi_c \prod_{d=1}^{392} \theta_{cd}^{x_d}(1-\theta_{cd})^{1-x_d} / \sum_{c'=0}^{9} \pi_c \prod_{d=1}^{392} \theta_{cd}^{x_d}(1-\theta_{cd})^{1-x_d})(\prod_{d=392}^{784} \theta_{cd}^{x_d}(1-\theta_{cd})^{1-x_d})$$

E)

```
def marginal_dist(images, theta, size):
    c ,d = theta.shape
    pi = 0.1
    result = []
    for i in range(size):
        top_half = images[i, :392]
        #getting rid of log
        p_cx = np.exp(log_likelihood(images[i], theta, pi, top=True))
        image = []
        for j in range(392, d):
            pixel = 0
            for digit in range(c):
                pixel += theta[digit][j]*p_cx[digit]
            image.append(pixel)
        bottom_half = np.array(image)
        full_image = np.concatenate((top_half, bottom_half))
        result.append(full_image)
    plot_images(np.array(result), plt)
```

```
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    #train_images = binarize(train_images)
    #test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
    #1_e
    #print(avg_log_likelihood(train_images, theta, train_labels))
    #print(accuracy(train_images, theta, train_labels))
    #print(avg_log_likelihood(test_images, theta, test_labels))
    #print(accuracy(test_images, theta, test_labels))
    #2_c
    #random_image_samples(train_images, theta, train_labels, 10)
    #2_e
    marginal_dist(train_images, theta, 20)
```



3. A) 2 parameters: w and x

   B) $\log(\mathrm{p}(\mathrm{c}|\mathrm{x},\mathrm{w}) = w_c^T x - \log(\sum_{c'=0}^{9} \exp(w_{c'}^T x))$

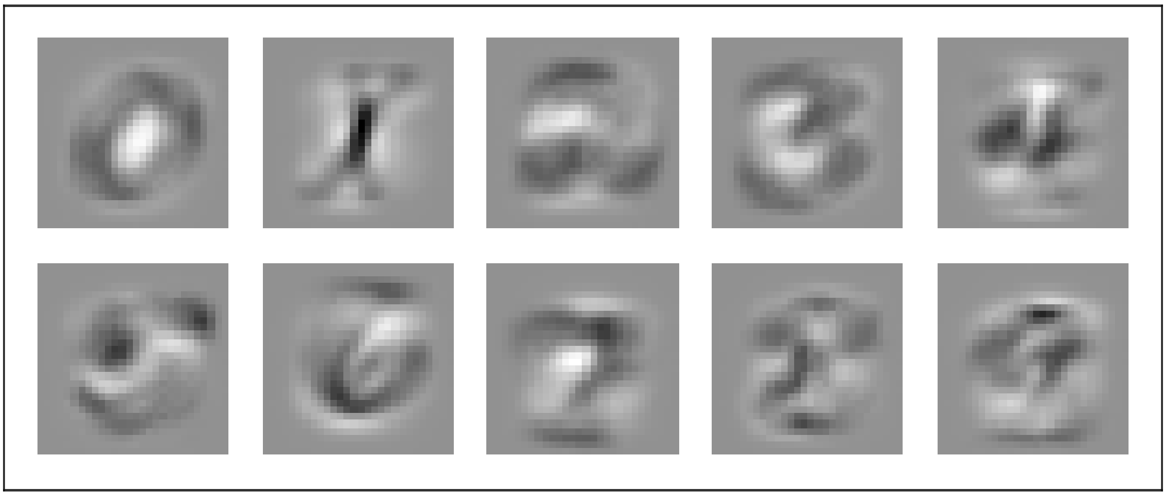
   $\nabla_{\mathrm{w}} \log p(c|x,w) = x_c - x_{\mathrm{c}} \exp(w_{c'}^T x) / \sum_{c'=0}^{9} \exp({w^T}'_{c'} x)$

   C)

```
def stochastic(w, x, t):
    wx = np.dot(x, w)
    N = x.shape[0]
    y = (np.exp(wx).T / np.sum(np.exp(wx),axis=1)).T
    return (-1/N)*np.dot(x.T, y-t)
```

```python
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    train_images = binarize(train_images)
    test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
    #1_e
    #print(avg_log_likelihood(train_images, theta, train_labels))
    #print(accuracy(train_images, theta, train_labels))
    #print(avg_log_likelihood(test_images, theta, test_labels))
    #print(accuracy(test_images, theta, test_labels))
    #2_c
    #random_image_samples(train_images, theta, train_labels, 10)
    #2_e
    #marginal_dist(train_images, theta, 20)

    c = 10
    d = 784
    weights = np.zeros((d, c))
    max_iter = 100
    lr = 0.00001
    iteration = 0
    EPS = 1e-4
    prev_w = weights-10*EPS
    while(norm(weights - prev_w) > EPS and iteration < max_iter):
        prev_w = weights.copy()
        g = stochastic(weights, train_images, train_labels)
        weights += lr*g
        iteration += 1
        print("iteration: ", iteration, "\nw is ", weights)
    a, v = accuracyAndlikelihood(weights, train_images, train_labels)
    print("accuracy: ", a, "avg: ", v)
    weights = weights.transpose()
    plot_images(weights, plt)
```



d) Accuracy on training data: 0.894316666667

Average log likelihood of training data: -0.3864147759252905

Accuracy on test data: 0.9005

Average log likelihood of test data: -0.3694515017335679

The result was better than Naïve Bayes result.

```
if __name__ == "__main__":
    N_data, train_images, train_labels, test_images, test_labels = load_mnist()
    train_images = binarize(train_images)
    test_images = binarize(test_images)
    #1_c
    theta = fit_MAP(train_images, train_labels)
    #1_e
    #print(avg_log_likelihood(train_images, theta, train_labels))
    #print(accuracy(train_images, theta, train_labels))
    #print(avg_log_likelihood(test_images, theta, test_labels))
    #print(accuracy(test_images, theta, test_labels))
    #2_c
    #random_image_samples(train_images, theta, train_labels, 10)
    #2_e
    #marginal_dist(train_images, theta, 20)

    c = 10
    d = 784
    weights = np.zeros((d, c))
    max_iter = 100
    lr = 0.00001
    iteration = 0
    EPS = 1e-4
    prev_w = weights-10*EPS
    while(norm(weights - prev_w) > EPS and iteration < max_iter):
        prev_w = weights.copy()
        g = stochastic(weights, train_images, train_labels)
        weights += lr*g
        iteration += 1
        print("iteration: ", iteration, "\nw is ", weights)
    a, v = accuracyAndlikelihood(weights, train_images, train_labels)
    print("train_accuracy: ", a, "train_avg: ", v)
    a, v = accuracyAndlikelihood(weights, test_images, test_labels)
    print("test_accuracy: ", a, "test_avg: ", v)
    weights = weights.transpose()
    plot_images(weights, plt)
```