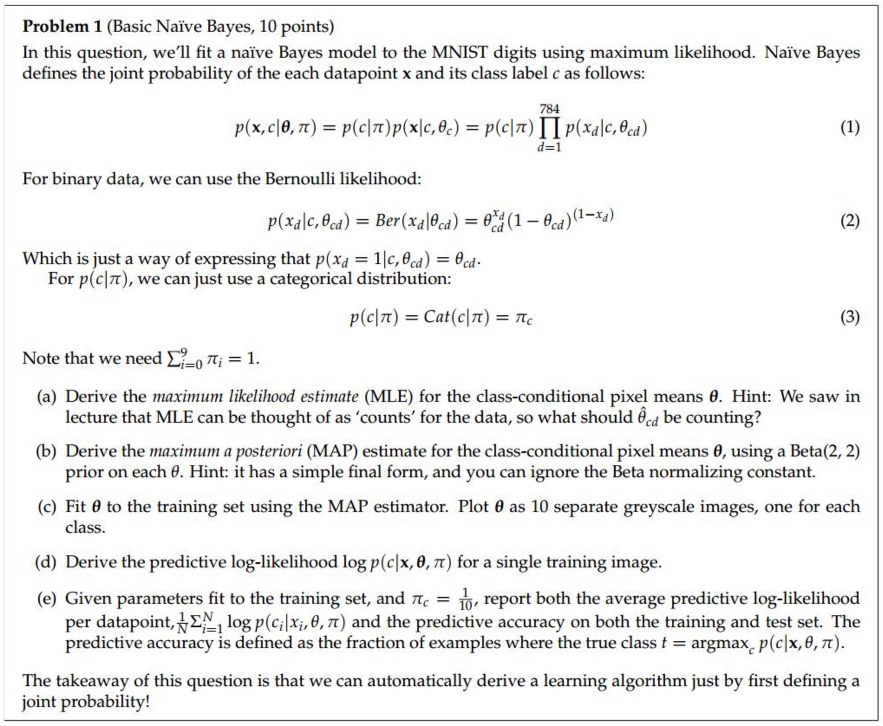
CSC412

Assignment 2

Vibhavi Peiris
Student #: 1000597687

March 18, 2018



Problem 1 (Basic Naïve Bayes, 10 points)

In this question, we'll fit a naïve Bayes model to the MNIST digits using maximum likelihood. Naïve Bayes defines the joint probability of the each datapoint **x** and its class label c as follows:

$$p(\mathbf{x}, c|\boldsymbol{\theta}, \pi) = p(c|\pi)p(\mathbf{x}|c, \theta_c) = p(c|\pi)\prod_{d=1}^{784} p(x_d|c, \theta_{cd}) \quad (1)$$

For binary data, we can use the Bernoulli likelihood:

$$p(x_d|c, \theta_{cd}) = Ber(x_d|\theta_{cd}) = \theta_{cd}^{x_d}(1-\theta_{cd})^{(1-x_d)} \quad (2)$$

Which is just a way of expressing that $p(x_d = 1|c, \theta_{cd}) = \theta_{cd}$.

For $p(c|\pi)$, we can just use a categorical distribution:

$$p(c|\pi) = Cat(c|\pi) = \pi_c \quad (3)$$

Note that we need $\sum_{i=0}^{9} \pi_i = 1$.

(a) Derive the *maximum likelihood estimate* (MLE) for the class-conditional pixel means $\boldsymbol{\theta}$. Hint: We saw in lecture that MLE can be thought of as 'counts' for the data, so what should $\hat{\theta}_{cd}$ be counting?

(b) Derive the *maximum a posteriori* (MAP) estimate for the class-conditional pixel means $\boldsymbol{\theta}$, using a Beta(2, 2) prior on each θ. Hint: it has a simple final form, and you can ignore the Beta normalizing constant.

(c) Fit $\boldsymbol{\theta}$ to the training set using the MAP estimator. Plot $\boldsymbol{\theta}$ as 10 separate greyscale images, one for each class.

(d) Derive the predictive log-likelihood $\log p(c|\mathbf{x}, \boldsymbol{\theta}, \pi)$ for a single training image.

(e) Given parameters fit to the training set, and $\pi_c = \frac{1}{10}$, report both the average predictive log-likelihood per datapoint, $\frac{1}{N}\Sigma_{i=1}^{N} \log p(c_i|x_i, \theta, \pi)$ and the predictive accuracy on both the training and test set. The predictive accuracy is defined as the fraction of examples where the true class $t = \operatorname{argmax}_c p(c|\mathbf{x}, \theta, \pi)$.

The takeaway of this question is that we can automatically derive a learning algorithm just by first defining a joint probability!

1. Problem 1: Basic Naïve Bayes
   a. *Derive MLE*

θ_{cd} is counting the number of times a digit $\left(\mathrm{y}^d\right)$ is seen w.r.t a data row(image) $\left(\mathrm{x}_d\right)$

$$\theta_{cd} = \frac{\sum_{i=1}^{10000} \text{current x times cases when } \mathrm{Y}^i = y^d}{\sum_{i=1}^{10000} \text{sum cases when } \mathrm{Y}^i = y^d.\ \text{ i.e. the ith y in the sum is a certain digit}}$$

$$\theta_{cd} = \frac{\sum_{i=1}^{10000} x_d^i\left(f\left(Y^i, y^d\right)\right)}{\sum_{i=1}^{10000}\left(f\left(Y^i, y^d\right)\right)} \qquad \text{where } f\left(Y^i, y^d\right) = \begin{cases} 1 & \text{if } Y^i = y^d \\ 0 & \text{else} \end{cases}$$

b. Derive MAP

$$\theta_{cd} = \max\left(\frac{P(\theta \mid x)P(\theta)}{P(x)}\right) \quad \text{MAP's defintion}$$

$$\frac{P(\theta \mid x,c)P(\theta \mid c)}{P(x \mid \mathrm{c})} = P(x \mid \theta, \mathrm{c})$$

$$P(\theta \mid x,c) \propto P(x \mid \theta, c)P(\theta \mid c)$$

$$\propto \left(\prod_{i}^{10000} \theta_{cd}^{x_d^i}(1-\theta_{cd})^{1-x_d^i}\right)\theta_{cd}(1-\theta_{cd})$$

$$\propto \left(\prod_{i}^{10000} \theta_{cd}^{x_d^i}(1-\theta_{cd})^{1-x_d^i}\right)\theta_{cd}(1-\theta_{cd})$$

$$P(x_d \mid \theta_{cd}, \mathrm{c}) = \left(\prod_{i}^{10000} \theta_{cd}^{x_d^i}(1-\theta_{cd})^{1-x_d^i}\right) = \theta_{cd}^{Nd}(1-\theta_{cd})^{N-Nd}$$

Where N is all points and $\mathrm{N}_d = \sum_{i=1}^{N} x_d^i$ i.e. all the ones(true digits) seen for each class

$$\mathrm{P}(\theta_{cd} \mid c) = Beta(\theta_{cd} \mid a,b) \propto \theta_{cd}^{a-1}(1-\theta_{cd})^{b-1}$$

$$\frac{P(\theta_{cd} \mid x_d, c)P(\theta_{cd} \mid c)}{P(x \mid \mathrm{c})} = P(x_d \mid \theta_{cd}, \mathrm{c}) \qquad \text{can transform using bayes rule}$$

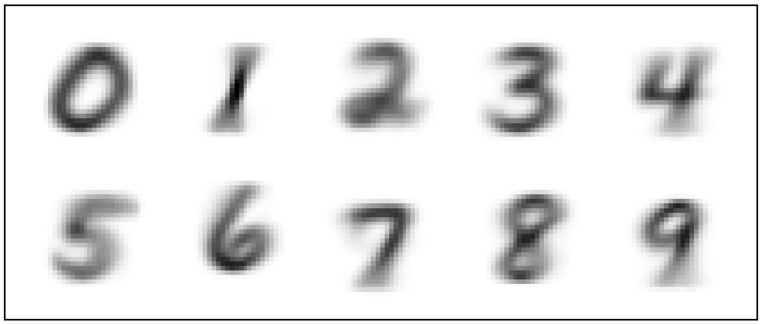
$$P(\theta_{cd} \mid x_d, c) \propto P(x_d \mid \theta_{cd}, c)P(\theta_{cd} \mid c)$$

$$= \theta_{cd}^{Nd}(1-\theta_{cd})^{N-Nd}\,\theta_{cd}^{a-1}(1-\theta_{cd})^{b-1} = \theta_{cd}^{Nd+a-1}(1-\theta_{cd})^{N-Nd+b-1}$$

$$\theta map = \frac{Nd+a-1}{N+a+b-2} = \frac{Nd+2-1}{N+2} = \frac{Nd+1}{N+2} \qquad \text{learned in 411}$$

c.   Code for q1c & image output

```
#Q1C code
def find_theta_MAP(train_images, train_labels):
    theta_map = np.zeros((10,784))

    #find theta for each digit
    for i in range(0,theta_map.shape[0]):
        img_digit_locs = train_labels[:,i]  #get all images locations that have a digit i
        current_data = np.transpose(train_images)
        Nd = np.dot(current_data,img_digit_locs)       #multiply data with current digit locs to get data only for current digit (true digit)
        N = np.sum(img_digit_locs)                    #get total points for current digit
        current_theta = np.divide((Nd+1),(N+2))
        theta_map[i,:] = current_theta #save the theta for this digit
    return theta_map

#run Q1C code and save image
theta_map = find_theta_MAP(train_images, train_labels)
save_images(theta_map,'Q1C')
```



d. Derive predicted log-likelihood

$$P(c \mid x,\theta,\pi) = \frac{P(\mathrm{x} \mid \mathrm{c},\theta,\pi)P(c \mid \pi)}{P(\mathrm{x} \mid \mathrm{c},\theta)} \qquad \text{\#bayes rule}$$

$$\propto P(\mathrm{x} \mid \mathrm{c},\theta)P(c \mid \pi)$$

$$= \left(\prod_{d=1}^{784} \theta_{cd}^{x_d}\left(1-\theta_{cd}\right)^{(1-x_d)}\right)\pi_c$$

$$\log\left(P(c \mid x,\theta,\pi)\right) = \log\left(\left(\prod_{d=1}^{784} \theta_{cd}^{x_d}\left(1-\theta_{cd}\right)^{(1-x_d)}\right)\pi_c\right)$$

$$= \log\left(\prod_{d=1}^{784} \theta_{cd}^{x_d}\left(1-\theta_{cd}\right)^{(1-x_d)}\right) + \log\left(\pi_c\right)$$

$$= \left(\sum_{d=1}^{784} \log\left(\theta_{cd}^{x_d}\left(1-\theta_{cd}\right)^{(1-x_d)}\right)\right) + \log\left(\pi_c\right)$$

$$= \left(\sum_{d=1}^{784} x_d \log\left(\theta_{cd}\right) + \left(1-x_d\right)\log\left(1-\theta_{cd}\right)\right) + \log\left(\pi_c\right)$$

e. Get average and accuracy

Average train log likelihood -172.353823347

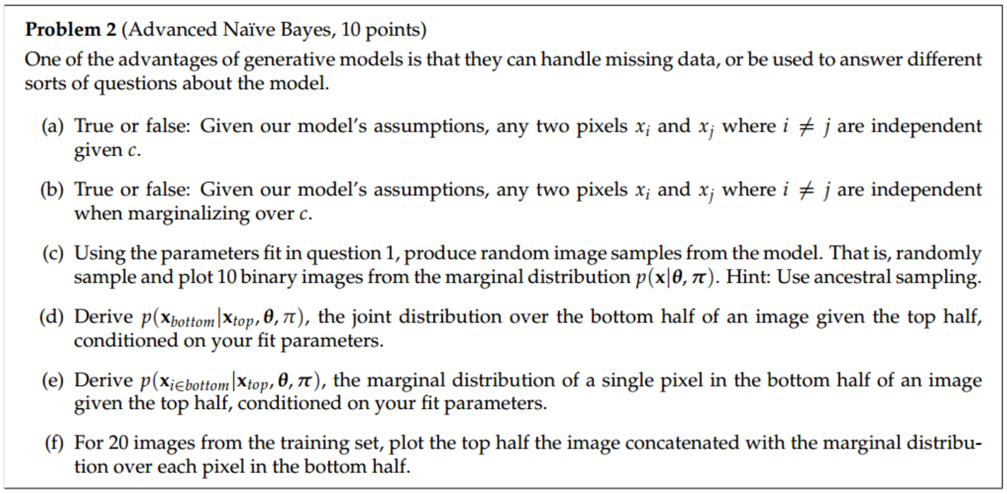
Average test log likelihood -173.043603091

Train Accuracy 0.8398

Test Accuracy 0.8372

Code for q1e on next page

```
#Q1e code
def find_log_likelihood(images, theta_map, pi_c):
    #find using formula generated in q1d
    likelihoods = np.zeros((images.shape[0],10))

    #find the likelihood for each digit/datapoint
    for digit in range(0,10): #loop through each digit
        for i in range(0,images.shape[0]):
            current_data = images[i,:]  #get current data point
            current_theta = theta_map[digit]   #get theta for current digit
            likelihoods[i,digit]  = np.dot(current_data,np.log(current_theta))+np.dot((1-current_data),np.log(1-current_theta)) #q1d
    return likelihoods + np.log(pi_c)

def avg_likelihood(images,label, log_likelihood):
    sum_likelihood = 0
    #go through each datapoint find all likelihoods for current_label
    for i in range(0,images.shape[0]):
        sum_likelihood = sum_likelihood + np.sum(log_likelihood[i,:]*label[i,:])  # sum of likelihoods for each image wrt. its label
    avg_likelihood = sum_likelihood/images.shape[0]    #divide by number of images to get average
    return avg_likelihood

def predict(images, theta_map, log_likelihood):
    predictions = np.zeros((images.shape[0],log_likelihood.shape[1])) #N by 10
    #find best class true class for each image
    for i in range(0,images.shape[0]):
        current_likelihood = log_likelihood[i,:]  #get likelihoods for current datapoint
        best_class = np.argmax(current_likelihood)   #choose the class with highest likelihood
        predictions[i,best_class] = 1  #set index = digit to 1 as it is the best prediction for current image

    return predictions

def Q1E_report(train_images,train_labels,test_images,test_labels,theta_map):
    #average for train
    log_likelihood_train = find_log_likelihood(train_images, theta_map, 1/10)
    avg_likelihood_train = avg_likelihood(train_images,train_labels,log_likelihood_train)
    print("Avg train log likelihood ",avg_likelihood_train)

    #average for test
    log_likelihood_test = find_log_likelihood(test_images, theta_map, 1/10)
    avg_likelihood_test = avg_likelihood(test_images,test_labels,log_likelihood_test)
    print("Avg test log likelihood ",avg_likelihood_test)

    #predictions for train
    predict_train = predict(train_images, theta_map, log_likelihood_train)
    total_correct_train = np.sum(np.nonzero(predict_train)[1] == np.nonzero(train_labels)[1]) #get total number of correct predictions
    accuracy_train = total_correct_train/float(train_labels.shape[0])    #get accuracy
    print('Train Accuracy ',accuracy_train)

    #predictions for test
    predict_test = predict(test_images, theta_map, log_likelihood_test)
    total_correct_test = np.sum(np.nonzero(predict_test)[1] == np.nonzero(test_labels)[1]) #get total number of correct predictions
    accuracy_test = total_correct_test/float(test_labels.shape[0])    #get accuracy
    print('Test Accuracy ',accuracy_test)

#run Q1e code
Q1E_report(train_images,train_labels,test_images,test_labels,theta_map)
```



Problem 2 (Advanced Naïve Bayes, 10 points)

One of the advantages of generative models is that they can handle missing data, or be used to answer different sorts of questions about the model.

(a) True or false: Given our model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent given c.

(b) True or false: Given our model's assumptions, any two pixels x_i and x_j where $i \neq j$ are independent when marginalizing over c.

(c) Using the parameters fit in question 1, produce random image samples from the model. That is, randomly sample and plot 10 binary images from the marginal distribution $p(\mathbf{x}|\boldsymbol{\theta}, \boldsymbol{\pi})$. Hint: Use ancestral sampling.

(d) Derive $p(\mathbf{x}_{bottom}|\mathbf{x}_{top}, \boldsymbol{\theta}, \pi)$, the joint distribution over the bottom half of an image given the top half, conditioned on your fit parameters.

(e) Derive $p(\mathbf{x}_{i \in bottom}|\mathbf{x}_{top}, \boldsymbol{\theta}, \boldsymbol{\pi})$, the marginal distribution of a single pixel in the bottom half of an image given the top half, conditioned on your fit parameters.

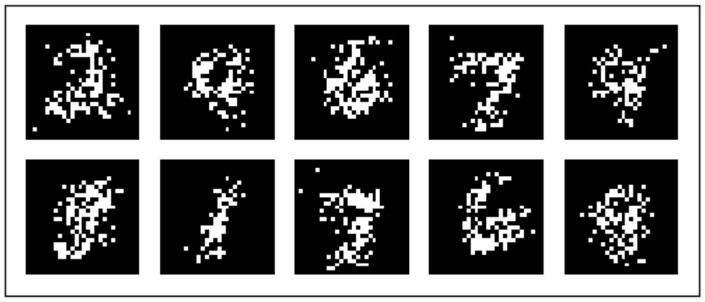
(f) For 20 images from the training set, plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half.

2. Problem 2: Advanced Naïve Bayes
   a. *Independent given c?*

      True, since we are using Naive Bayes, which treats each data point as independent
   b. *Independent when marginalizing over c?*

      False, because when we marginalize over c, that means we are summing out c. Which results in the x's being dependent on each other.
   c. *Code and images*

```
#Q2c code
def create_samples(num_of_samples,theta_map):
  sample_data = np.random.rand(num_of_samples,784)
  rand_cs = np.random.randint(num_of_samples, size=(1, 10))[0] #generate random numbers(digit class)

  for i in range(0,num_of_samples):
    #using p(x_d | c, Î¸_cd ) i.e. ancestral sampling
    c = rand_cs[i] #get the rand c
    current_theta = theta_map[c,:]  # get the thetha for the specific digit
    xd = sample_data[i,:]    #get current rand sample

    #pick current random data point based on c value. Also binarize
    xd[xd < current_theta] = 0
    xd[xd >= current_theta] = 1

    #save the updated xd
    sample_data[i,:] = xd

  return sample_data

#run Q2C code
sample_data = create_samples(10,theta_map)
save_images(sample_data,'Q2C')
```



d. Derive join distribution over bottom half

$$P\left(x_{bottom} \mid \mathrm{x}_{top}, \theta, \pi\right) = \frac{P\left(x_{bottom} \cap \mathrm{x}_{top} \mid \theta, \pi\right)}{P\left(\mathrm{x}_{top} \mid \theta, \pi\right)}$$ #Conditional probabilty definition

$$= \frac{P\left(x \mid \theta, \pi\right)}{P\left(\mathrm{x}_{top} \mid \theta, \pi\right)}$$

To get $P\left(x \mid \theta, \pi\right)$ in numerator we need to marginalize c, so $\sum_{c=0}^{9} P\left(x \mid \mathrm{c}, \theta, \pi\right)$

To get $P\left(x_{top} \mid \theta, \pi\right)$ in denomerator we need to marginalize c, so $\sum_{c=0}^{9} P\left(x_{top} \mid \mathrm{c}, \theta, \pi\right)$

$$P\left(x_{bottom} \mid \mathrm{x}_{top}, \theta, \pi\right) = \frac{\sum_{c=0}^{9} P\left(x \mid \mathrm{c}, \theta, \pi\right)}{\sum_{c=0}^{9} P\left(x_{top} \mid \mathrm{c}, \theta, \pi\right)}$$

$$= \frac{\sum_{c=0}^{9} \prod_{d=1}^{784} Ber\left(x_d \mid \theta_{cd}\right)}{\sum_{c=0}^{9} \prod_{d=1}^{392} Ber\left(x_d \mid \theta_{cd}\right)}$$ #half point of image is 784/2=392

$$= \frac{\sum_{c=0}^{9} \prod_{d=1}^{784} \theta_{cd}^{x_d} \left(1-\theta_{cd}\right)^{\left(1-x_d\right)}}{\sum_{c=0}^{9} \prod_{d=1}^{392} \theta_{cd}^{x_d} \left(1-\theta_{cd}\right)^{\left(1-x_d\right)}}$$

e. Derive join distribution over single pixel in bottom half

$$P\left(x_{i\in bottom} \mid \mathrm{x}_{top}, \theta, \pi\right) = \frac{P\left(x_{i\in bottom} \cap \mathrm{x}_{top} \mid \theta, \pi\right)}{P\left(\mathrm{x}_{top} \mid \theta, \pi\right)}$$ #Conditional probabilty definition

$$= \frac{P\left(x_{i\in bottom} \mid \theta, \pi\right) P\left(\mathrm{x}_{top} \mid \theta, \pi\right)}{P\left(\mathrm{x}_{top} \mid \theta, \pi\right)}$$

To get $P\left(x_{i\in bottom} \mid \theta, \pi\right) P\left(\mathrm{x}_{top} \mid \theta, \pi\right)$ in numerator we need to marginalize c,

so $\sum_{c=0}^{9} P\left(x_{i\in bottom} \mid \mathrm{c}, \theta, \pi\right) P\left(\mathrm{x}_{top} \mid \mathrm{c}, \theta, \pi\right)$

To get $P\left(x_{top} \mid \theta, \pi\right)$ in denomerator we need to marginalize c, so $\sum_{c=0}^{9} P\left(x_{top} \mid \mathrm{c}, \theta, \pi\right)$

$Note: P\left(x_{i\in bottom} \mid \mathrm{c}, \theta, \pi\right) = P\left(x_{i\in bottom} = 1 \mid \mathrm{c}, \theta, \pi\right) = \theta_{cd}$ #according to q1 eq2

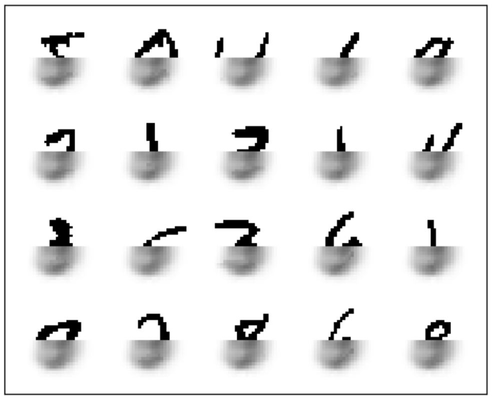
$$P\left(x_{i\in bottom} \mid \mathrm{x}_{top}, \theta, \pi\right) = \frac{\sum_{c=0}^{9} P\left(x_{i\in bottom} \mid \mathrm{c}, \theta, \pi\right) P\left(\mathrm{x}_{top} \mid \mathrm{c}, \theta, \pi\right)}{\sum_{c=0}^{9} P\left(x_{top} \mid \mathrm{c}, \theta, \pi\right)}$$

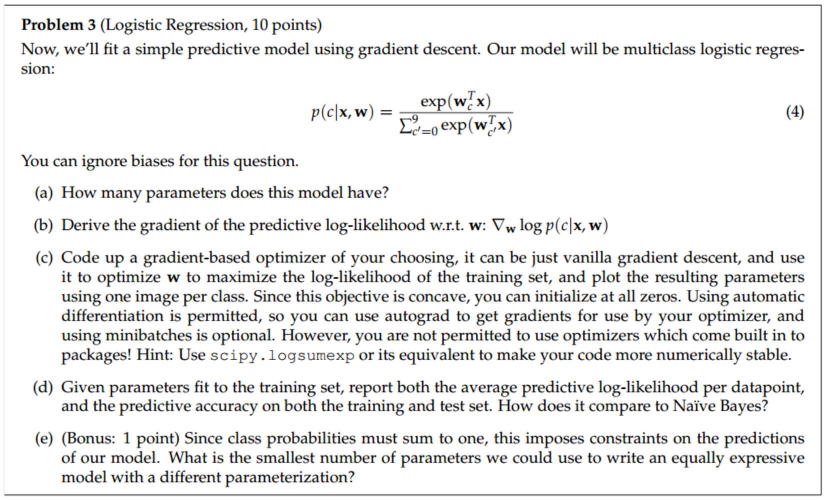
$$= \frac{\sum_{c=0}^{9} \theta_{cd} \prod_{d=1}^{392} Ber\left(x_d \mid \theta_{cd}\right)}{\sum_{c=0}^{9} \prod_{d=1}^{392} Ber\left(x_d \mid \theta_{cd}\right)}$$ #half point of image is 784/2=392

$$= \frac{\sum_{c=0}^{9} \theta_{cd} \prod_{d=1}^{392} \theta_{cd}^{x_d} \left(1-\theta_{cd}\right)^{(1-x_d)}}{\sum_{c=0}^{9} \prod_{d=1}^{392} \theta_{cd}^{x_d} \left(1-\theta_{cd}\right)^{(1-x_d)}}$$

f. Code and images

```
#Code for Q2F
def mult_bern_likelihood(x, theta_c,c,stop):
  #compute multiple p(x_d|c,0_cd) = ber(x_d|0_cd)
  likelihood = np.ones(stop)
  for d in range(0,stop):
    theta_cd = theta_c[d]
    x_d = x[d]
    likelihood[d] = likelihood[d]*((theta_cd**x_d)*(1-theta_cd)**(1-x_d))
  return likelihood

def advanced_bayes(images, theta_map):
  half_size = int(images.shape[1]/2)  # where the top half of the image ends, should be 392

  #find for each image
  for img_num in range(0, images.shape[0]):
    #numerator (P(x_inbottom and x_top))
    sum_overc_nume = np.zeros((half_size))
    for c in range (0,10):
      #p(x_inbottom|0)
      theta_c = theta_map[c,:]
      #p(x_top|0) from 0 to 392
      likelihood_num = mult_bern_likelihood(images[img_num,:half_size],theta_c[:half_size],c,half_size)

      #sum over c for p(x_inbottom|0)*p(x_top|0)
      sum_overc_nume = sum_overc_nume + likelihood_num*theta_c[half_size:]

    #Denominator (P(x_top))
    sum_overc_dem = np.zeros((half_size))
    for c in range (0,10):
      #p(x_inbottom|0)
      theta_c = theta_map[c,:]
      #p(x_top|0) from 0 ot 392
      likelihood_dem = mult_bern_likelihood(images[img_num,:half_size],theta_c[:half_size],c, half_size)

      #sum over c for p(x_top|0)
      sum_overc_dem = sum_overc_dem + likelihood_dem

    #divide num by dem to get final result (x in bottom) for current image
    images[img_num,half_size:] = np.divide(sum_overc_nume,sum_overc_dem)
  return images

#run 2F code
results = advanced_bayes(train_images[0:20,:],theta_map)
save_images(results,'Q2F')
```





Problem 3 (Logistic Regression, 10 points)

Now, we'll fit a simple predictive model using gradient descent. Our model will be multiclass logistic regression:

$$p(c|\mathbf{x},\mathbf{w}) = \frac{\exp(\mathbf{w}_c^T\mathbf{x})}{\sum_{c'=0}^{9}\exp(\mathbf{w}_{c'}^T\mathbf{x})} \tag{4}$$

You can ignore biases for this question.

(a) How many parameters does this model have?

(b) Derive the gradient of the predictive log-likelihood w.r.t. **w**: $\nabla_{\mathbf{w}} \log p(c|\mathbf{x},\mathbf{w})$

(c) Code up a gradient-based optimizer of your choosing, it can be just vanilla gradient descent, and use it to optimize **w** to maximize the log-likelihood of the training set, and plot the resulting parameters using one image per class. Since this objective is concave, you can initialize at all zeros. Using automatic differentiation is permitted, so you can use autograd to get gradients for use by your optimizer, and using minibatches is optional. However, you are not permitted to use optimizers which come built in to packages! Hint: Use `scipy.logsumexp` or its equivalent to make your code more numerically stable.

(d) Given parameters fit to the training set, report both the average predictive log-likelihood per datapoint, and the predictive accuracy on both the training and test set. How does it compare to Naïve Bayes?

(e) (Bonus: 1 point) Since class probabilities must sum to one, this imposes constraints on the predictions of our model. What is the smallest number of parameters we could use to write an equally expressive model with a different parameterization?

3. Problem 3: Logistic Regression

   a. *How many parameters?*

      The number of digits (classes) times number of data points for each digit(image)

      10 x 784 = 7840 parameters

   b. *Derive predictive log-likelihood?*

$$\nabla_w \log\left(P\left(c\,|\,\mathrm{x},\mathrm{w}\right)\right) = \nabla_w \log\left(\frac{\exp\left(w_c^T x\right)}{\sum_{c'=0}^{9}\exp\left(w_{c'}^T x\right)}\right)$$

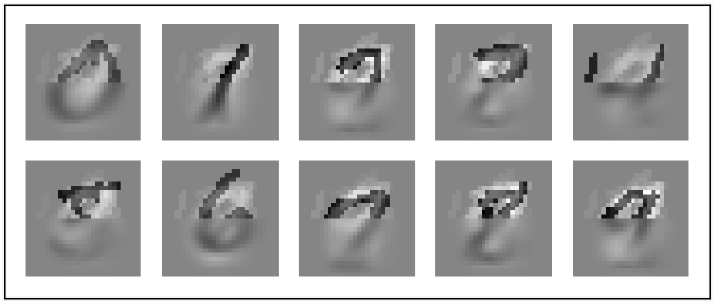
$$= \nabla_w\left(\log\left(\exp\left(w_c^T x\right)\right) - \log\left(\sum_{c'=0}^{9}\exp\left(w_{c'}^T x\right)\right)\right)$$

$$= \nabla_w\left(w_c^T x - \log\left(\sum_{c'=0}^{9}\exp\left(w_{c'}^T x\right)\right)\right)$$

Can use auto grad to get the gradient

c. Gradient Optimizer code and images

```
#Q3C code
def one_per_class(images, labels):
    out_images = np.zeros((10,images.shape[1]))
    out_labels = np.zeros((10,10))
    classes = np.where(labels == 1)[1] # get the class digit for each image by getting column idx of ones in labels

    #get first image in training set with each class label
    for i in range(0,10):
        img_num = np.where(classes == i)[0][0]
        out_images[i,:] = images[img_num,:]
        out_labels[i,:] = labels[img_num,:]
    return out_images,out_labels

def cost_function(w):
    sum_final = 0 #temporary create sum_final var
    dem = logsumexp(np.dot(np.transpose(w),grad_images))

    #mutliclass likelihood function is sum from 0 to k of label*predictive_log_likelihood
    for k in range(0,10):
        log_pc_x = np.dot(np.transpose(w[:,k]),grad_images) - dem
        if k == 0:
            sum_final = np.dot(grad_labels[k],log_pc_x)
        else:
            sum_final = sum_final + np.dot(grad_labels[k],log_pc_x)

    return sum_final

def logistic_gradient_desc(iterations,lr):
    #set globals so that cost function can access these values after usign autograd w.r.t. w
    global current_c
    global grad_images
    global grad_labels

    w = np.zeros((784,10)) #create the weights
    for i in range(0,iterations):
        for img_num in range(0,10):

            #get gradient of cost function/likelihood
            grad_images = new_images[img_num,:] #get current image
            grad_labels = new_labels[img_num,:] #get labels for current image
            current_c = img_num  #sinces we sampled 1 image for each class in order c = img_num
            cost_grad = elementwise_grad(cost_function)

            #update weights
            w = w + lr*cost_grad(w)

        print(i)
    return w

#run Q3C code
new_images,new_labels = one_per_class(train_images,train_labels)
grad_images=grad_labels = new_images #temporary just to create a gobal var for use with autograd
current_c = 0 #temporary just to create a gobal var for use with autograd
weights = logistic_gradient_desc(5000, 0.01) #5000 iterations with a common learning rate of 0.01
save_images(np.transpose(weights),'Q3c')
```



d. *Report average and accuracy*

   Avg train log likelihood -29.4875731391

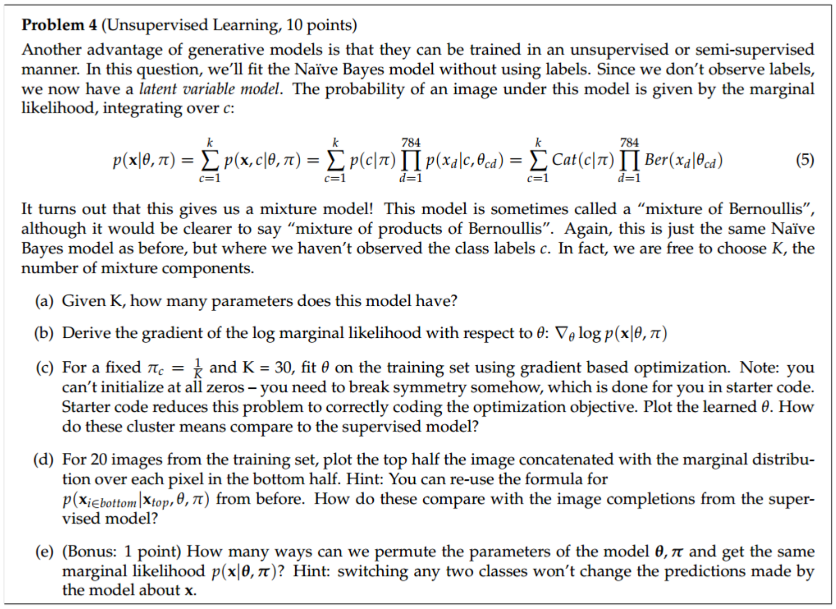
   Avg test log likelihood -29.3802306385

   Train Accuracy 0.511

   Test Accuracy 0.4958

   There is something wrong with my gradient descent, I spent a really long time trying to figure out where I went wrong, but sadly no luck.

   The accuracy for logistic regression should have been better than Naïve Bayes. Due to the fact that logistic regression looks at the whole image (all pixels together) where as Naïve Bayes looks at each individual pixel separately



Problem 4 (Unsupervised Learning, 10 points)

Another advantage of generative models is that they can be trained in an unsupervised or semi-supervised manner. In this question, we'll fit the Naïve Bayes model without using labels. Since we don't observe labels, we now have a *latent variable model*. The probability of an image under this model is given by the marginal likelihood, integrating over c:

$$p(\mathbf{x}|\theta,\pi) = \sum_{c=1}^{k} p(\mathbf{x},c|\theta,\pi) = \sum_{c=1}^{k} p(c|\pi)\prod_{d=1}^{784} p(x_d|c,\theta_{cd}) = \sum_{c=1}^{k} Cat(c|\pi)\prod_{d=1}^{784} Ber(x_d|\theta_{cd}) \qquad (5)$$

It turns out that this gives us a mixture model! This model is sometimes called a "mixture of Bernoullis", although it would be clearer to say "mixture of products of Bernoullis". Again, this is just the same Naïve Bayes model as before, but where we haven't observed the class labels c. In fact, we are free to choose K, the number of mixture components.

(a) Given K, how many parameters does this model have?

(b) Derive the gradient of the log marginal likelihood with respect to θ: $\nabla_\theta \log p(\mathbf{x}|\theta,\pi)$

(c) For a fixed $\pi_c = \frac{1}{K}$ and K = 30, fit θ on the training set using gradient based optimization. Note: you can't initialize at all zeros – you need to break symmetry somehow, which is done for you in starter code. Starter code reduces this problem to correctly coding the optimization objective. Plot the learned θ. How do these cluster means compare to the supervised model?

(d) For 20 images from the training set, plot the top half the image concatenated with the marginal distribution over each pixel in the bottom half. Hint: You can re-use the formula for $p(\mathbf{x}_{i\in bottom}|\mathbf{x}_{top},\theta,\pi)$ from before. How do these compare with the image completions from the supervised model?

(e) (Bonus: 1 point) How many ways can we permute the parameters of the model $\boldsymbol{\theta},\boldsymbol{\pi}$ and get the same marginal likelihood $p(\mathbf{x}|\boldsymbol{\theta},\boldsymbol{\pi})$? Hint: switching any two classes won't change the predictions made by the model about $\mathbf{x}$.

4. Problem 4: Unsupervised learning

   a. *How many parameters?*

      10*784+K

b. *Derive gradient of log marginal likelihood*

$$\nabla_\theta \log\left(P\left(\mathrm{x} \mid \theta, \pi\right)\right) = \nabla_\theta \log\left(\sum_{c=1}^{k} \pi_c \prod_{d=1}^{784} \theta_{cd}^{x_d} \left(1-\theta_{cd}\right)^{(1-x_d)}\right) \qquad \text{#can use autograd here}$$

Using the following log summation identity i found on wikipedia,
I can move log inside the sum

$$\log_b \sum_{i=0}^{N} a_i = \log_b a_0 + \log_b\left(1 + \sum_{i=1}^{N} \frac{a_i}{a_0}\right) = \log_b a_0 + \log_b\left(1 + \sum_{i=1}^{N} b^{(\log_b a_i - \log_b a_0)}\right)$$

$$\nabla_\theta \log\left(P\left(\mathrm{x} \mid \theta, \pi\right)\right) = \nabla_\theta\left(\log\left(\pi_0 \prod_{d=1}^{784} Ber\left(x_d \mid \theta_{0d}\right)\right) + \log\left(1 + \sum_{c=2}^{k} 10^{\left(\log\left(\pi_c \prod_{d=1}^{784} Ber\left(x_d \mid \theta_{cd}\right)\right) - \log\left(\left(\pi_0 \prod_{d=1}^{784} Ber\left(x_d \mid \theta_{0d}\right)\right)\right)\right)}\right)\right)$$

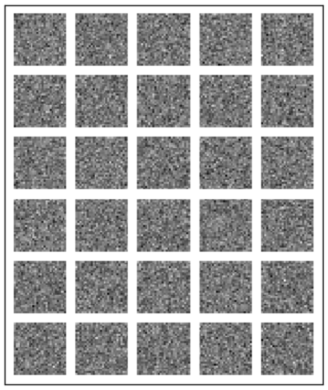
$$= \nabla_\theta\left(\log\left(\pi_0\right) + \log\left(\prod_{d=1}^{784} Ber\left(x_d \mid \theta_{0d}\right)\right) + \log\left(1 + \sum_{c=2}^{k} 10^{\left(\log(\pi_c) + \log\left(\prod_{d=1}^{784} Ber\left(x_d \mid \theta_{cd}\right)\right) - \log(\pi_c) + \log\left(\prod_{d=1}^{784} Ber\left(x_d \mid \theta_{cd}\right)\right)\right)}\right)\right)$$

$$= \nabla_\theta\left(\log\left(\pi_0\right) + \sum_{d=1}^{784} \log\left(Ber\left(x_d \mid \theta_{0d}\right)\right) + \log\left(1 + \sum_{c=2}^{k} 10^{\left(\log(\pi_c) + \sum_{d=1}^{784} \log\left(Ber\left(x_d \mid \theta_{cd}\right)\right) - \log(\pi_c) + \sum_{d=1}^{784} \log\left(Ber\left(x_d \mid \theta_{0d}\right)\right)\right)}\right)\right)$$

can replace the Ber distributions with eq2 from q1
Can't reduce the power futher because its c=0 - c=k

c. Code for neglogprob on next page and image
I could not get the neglobprob working properly is produces a weird optimized parameters

```
def neglogprob(params, data):
    # Implement this as the solution for 4c!
    ##METHOD 1
    #pi_c = 1/K
    ##sum from 0 to 784 of second part in 4b
    #log_prob = []
    #for i in range(0,data.shape[0]):
        #k_sum = np.zeros((1,10))

        #for c in range(1,K):
            #mult = np.ones((1,10))
            ##mult = ((params[c,:]**data[i,:])*((1-params[c,:])**(1-data[i,:])))  #vectorized version of the loop, i hope
            #for d in range(0,data.shape[1]):
                #mult = mult* ((params[c,d]**data[i,d])*((1-params[c,d])**(1-data[i,d])))
                #mult = mult *bernoulli_log_density(data[i,d], params[c,d])

            #k_sum = k_sum + pi_c*mult
        #log_prob.append(np.sum(np.array(-1*np.log(k_sum)[0]))/10)
    #return log_prob

    #METHOD 2 --------------------------------------------------------
    #derived formula from 4b
    pi_c = 1/K
    results = []
    for i in range (0,data.shape[0]):
        x = data[i,:]
        theta = params[i,:]
        #first term from 4b eq
        log_pi_c = np.log(pi_c)

        #second term from 4b eq
        second_term = 0
        for d in range(0,data.shape[1]):
            second_term = second_term + bernoulli_log_density(x[d], theta[0])

        #third term from 4b eq
        third_term = 0
        for c in range(2,K):
            first_power = log_pi_c
            for d in range(0,data.shape[1]):
                first_power = first_power + bernoulli_log_density(x[d], theta[c])

            power = first_power-(log_pi_c+second_term)
            third_term = third_term + 10**power

        log_prob = log_pi_c + second_term + third_term
        print(log_prob)
        results.append(log_prob)

    return -1*np.array(results)
```



d. Code and images

```
#Q4D code: just need to run 2fs code given optimized_params
results = advanced_bayes(train_images[0:20,:],optimized_params)
save_images(results,'Q4D')
```