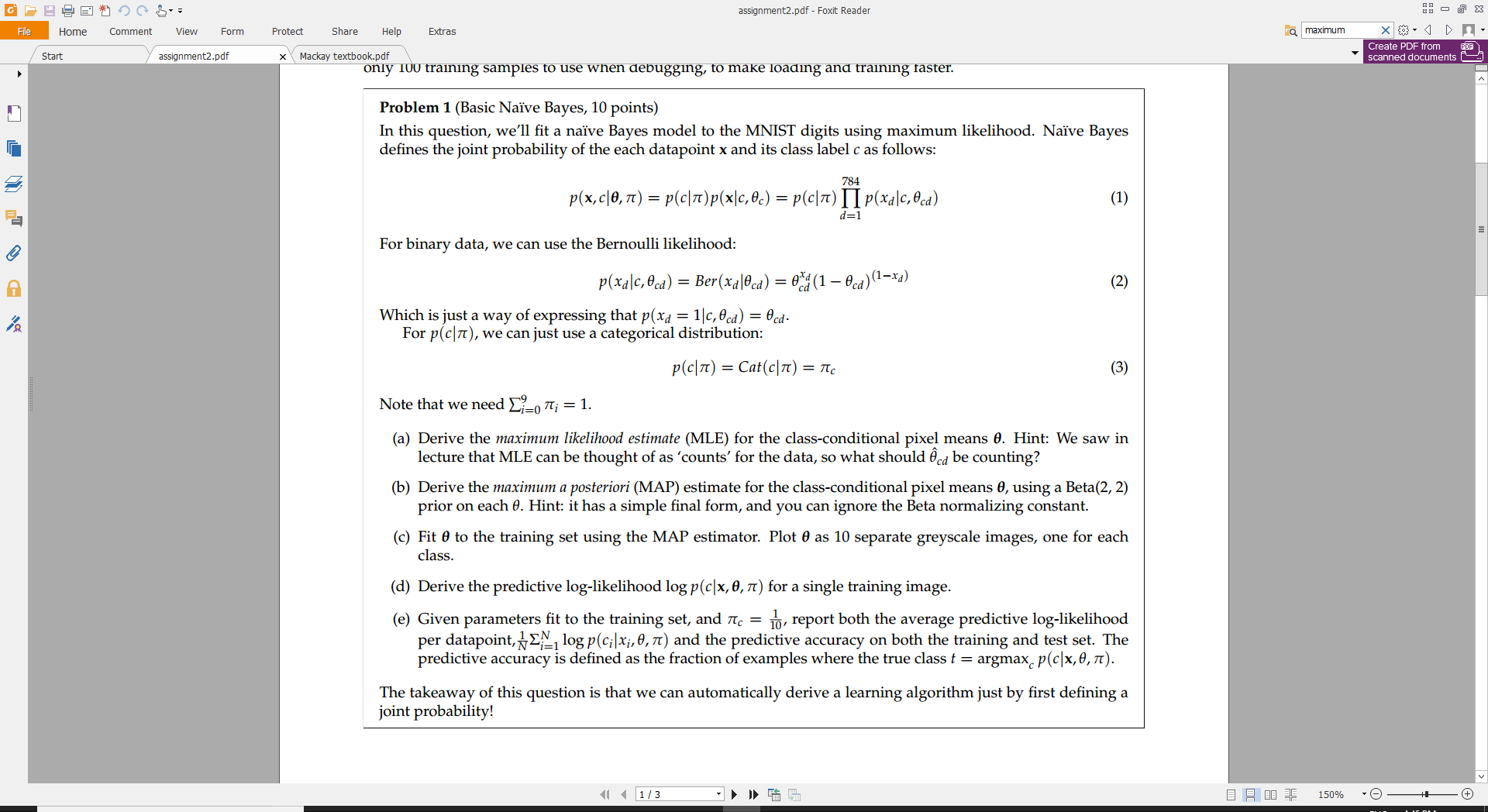
March 18, 2018

Vibhavi Peiris

Student #: 1000597687

CSC412

Assignment 2



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



* 1. *Derive MAP*



* 1. *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))

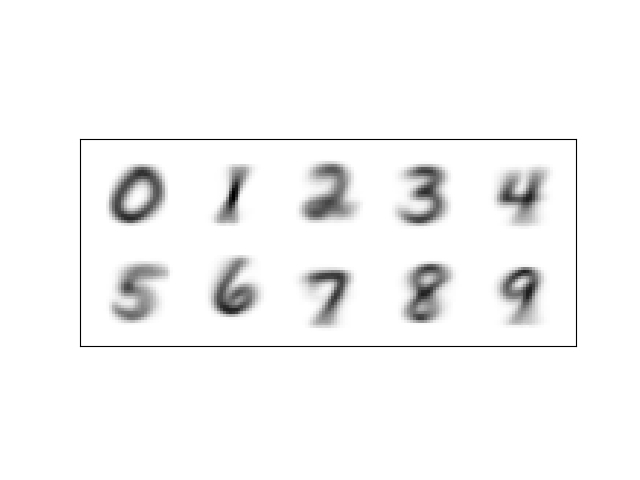
theta\_map[i,:] = current\_theta #save the theta for this digit

return theta\_map

#run Q1C code and save image

theta\_map = find\_theta\_MAP(train\_images, train\_labels)

save\_images(theta\_map,'Q1C')



* 1. *Derive predicted log-likelihood*





* 1. *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)

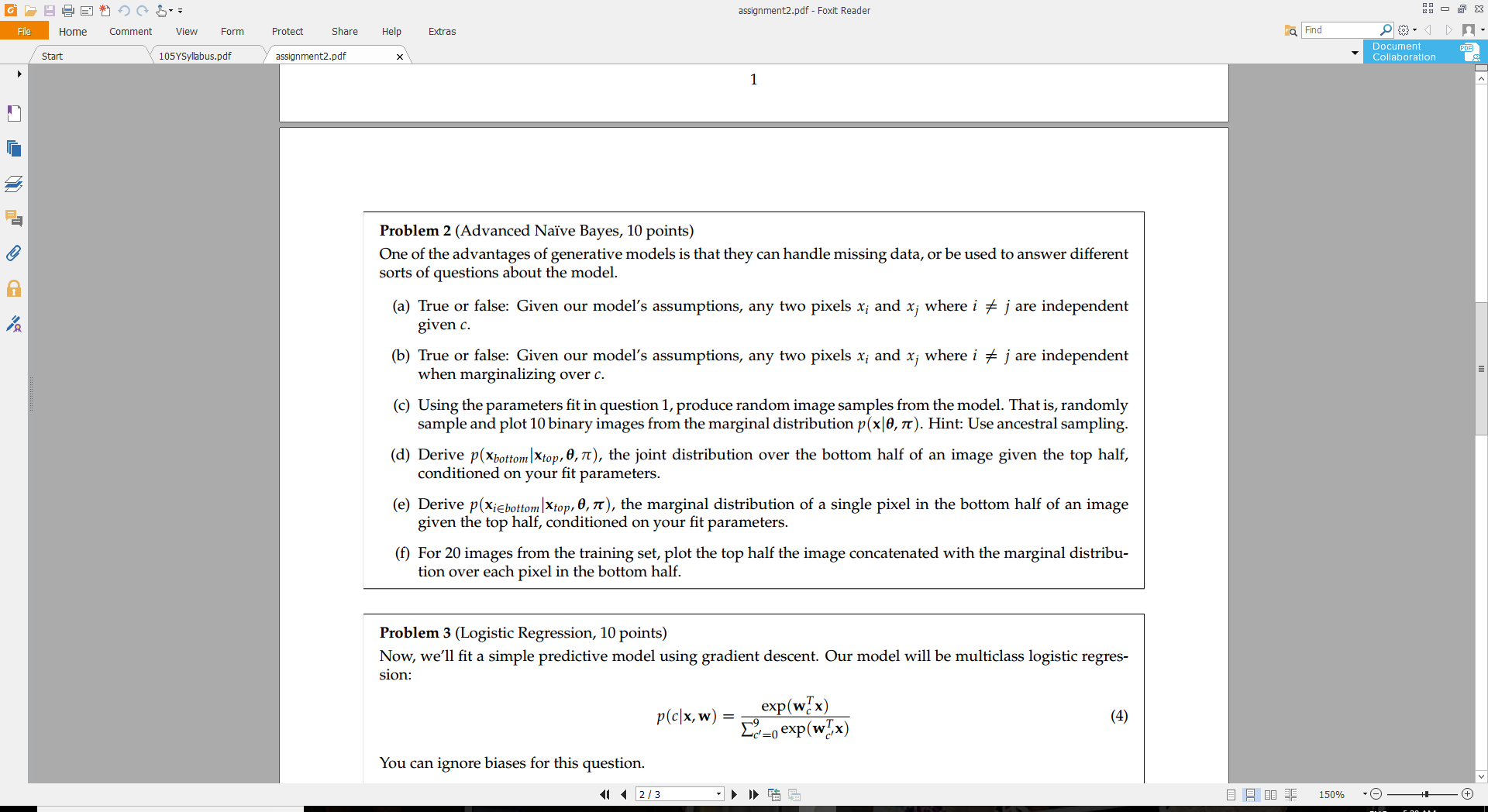
total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

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)



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

**True**, since we are using Naive Bayes, which treats each data point as independent

* 1. *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.

* 1. *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

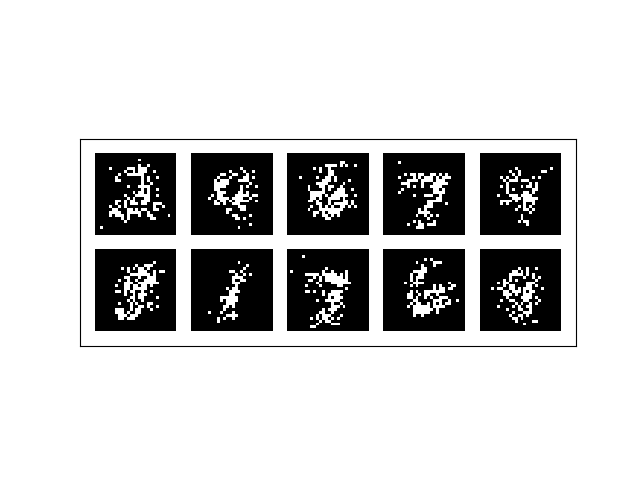
sample\_data[i,:] = xd

return sample\_data

#run Q2C code

sample\_data = create\_samples(10,theta\_map)

save\_images(sample\_data,'Q2C')



* 1. *Derive join distribution over bottom half*



* 1. *Derive join distribution over single pixel in bottom half*



* 1. *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

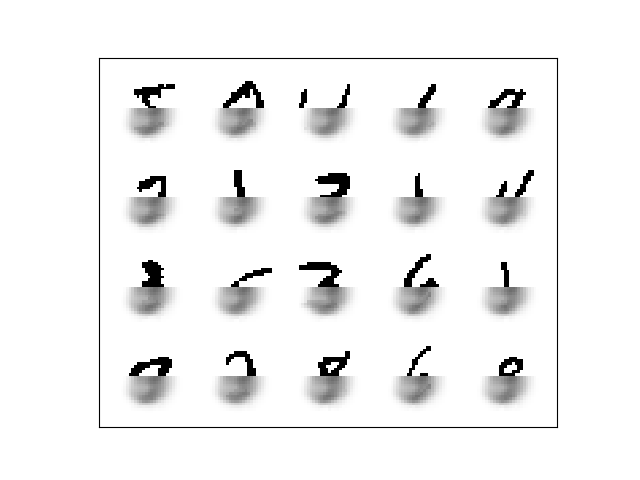
images[img\_num,half\_size:] = np.divide(sum\_overc\_nume,sum\_overc\_dem)

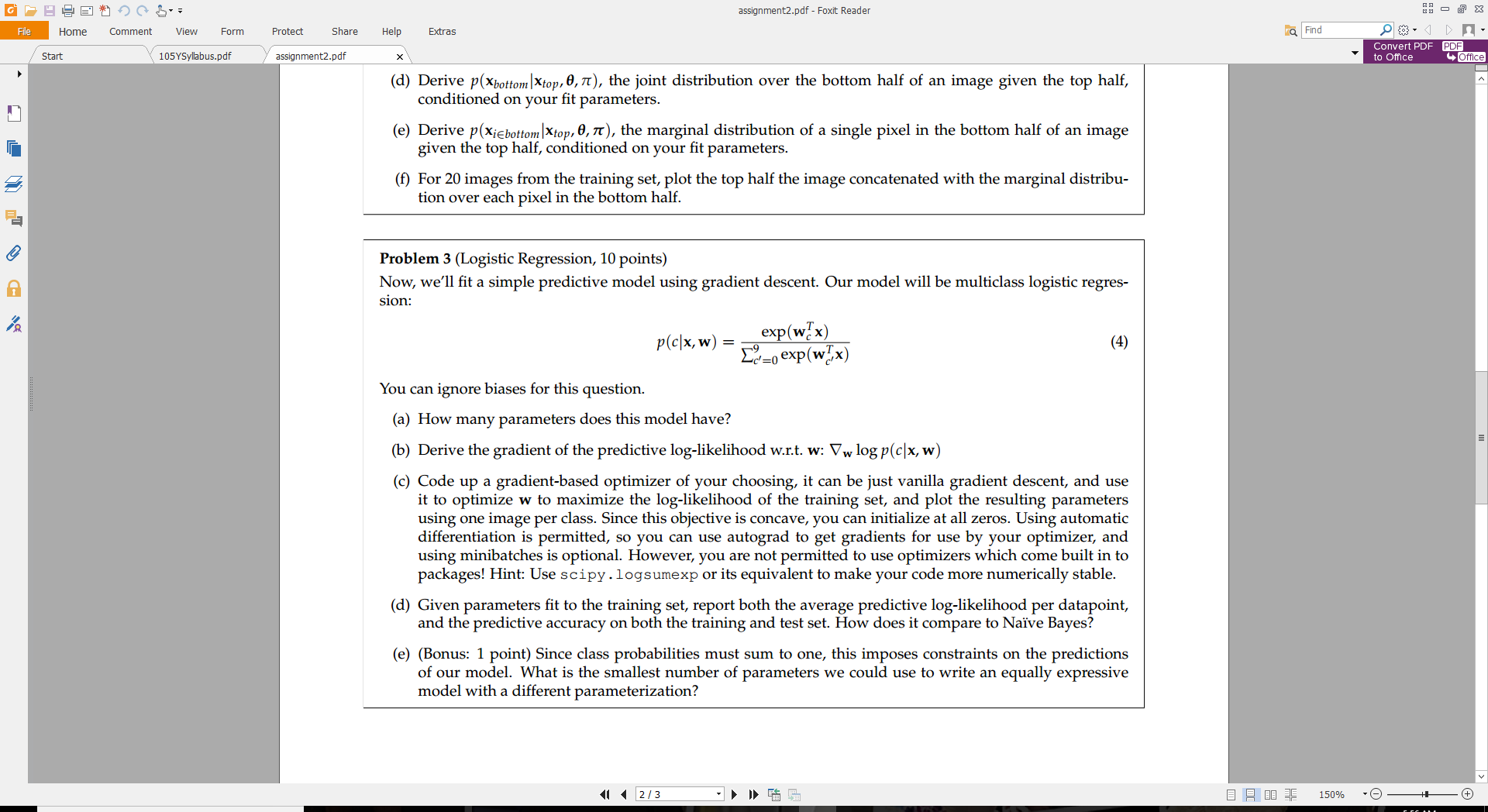
return images

#run 2F code

results = advanced\_bayes(train\_images[0:20,:],theta\_map)

save\_images(results,'Q2F')





1. Problem 3: Logistic Regression
   1. *How many parameters?*

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

10 x 784 = 7840 parameters

* 1. *Derive predictive log-likelihood?*



* 1. 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

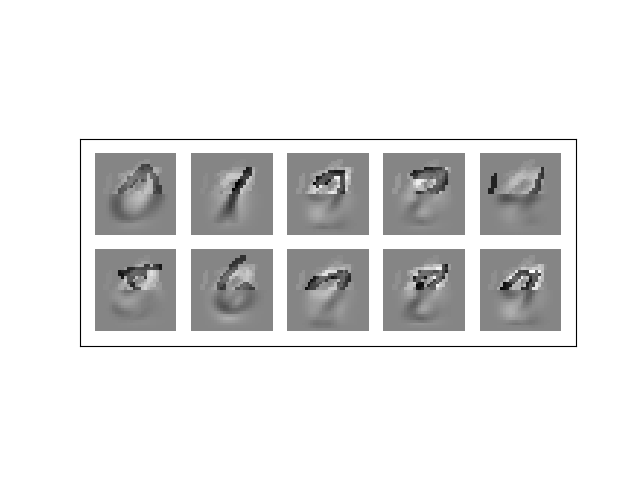
new\_images,new\_labels = one\_per\_class(train\_images,train\_labels)

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')



* 1. *Report average and accuracy*

Avg train log likelihood -29.4875731391

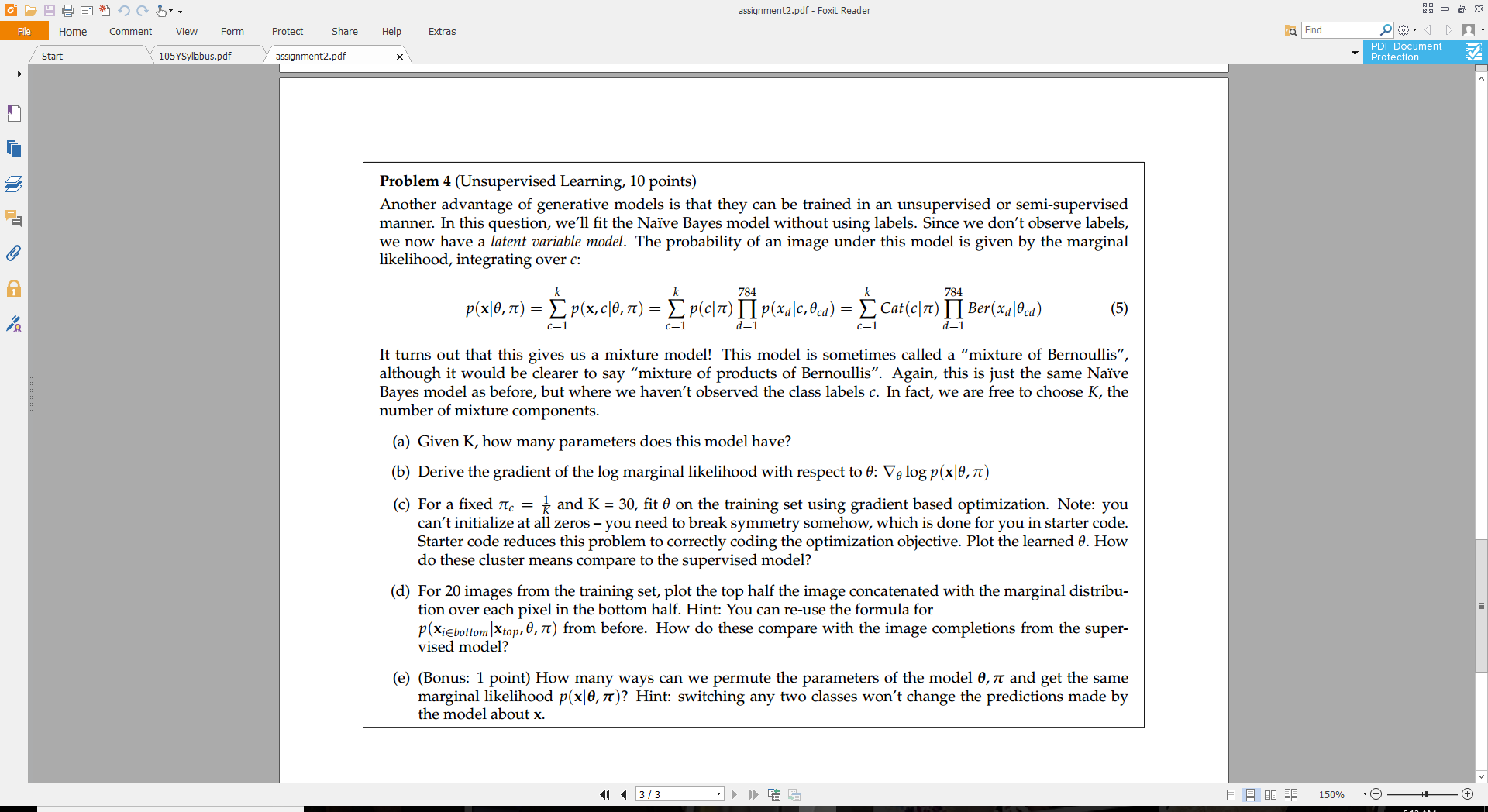
Avg test log likelihood -29.3802306385

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

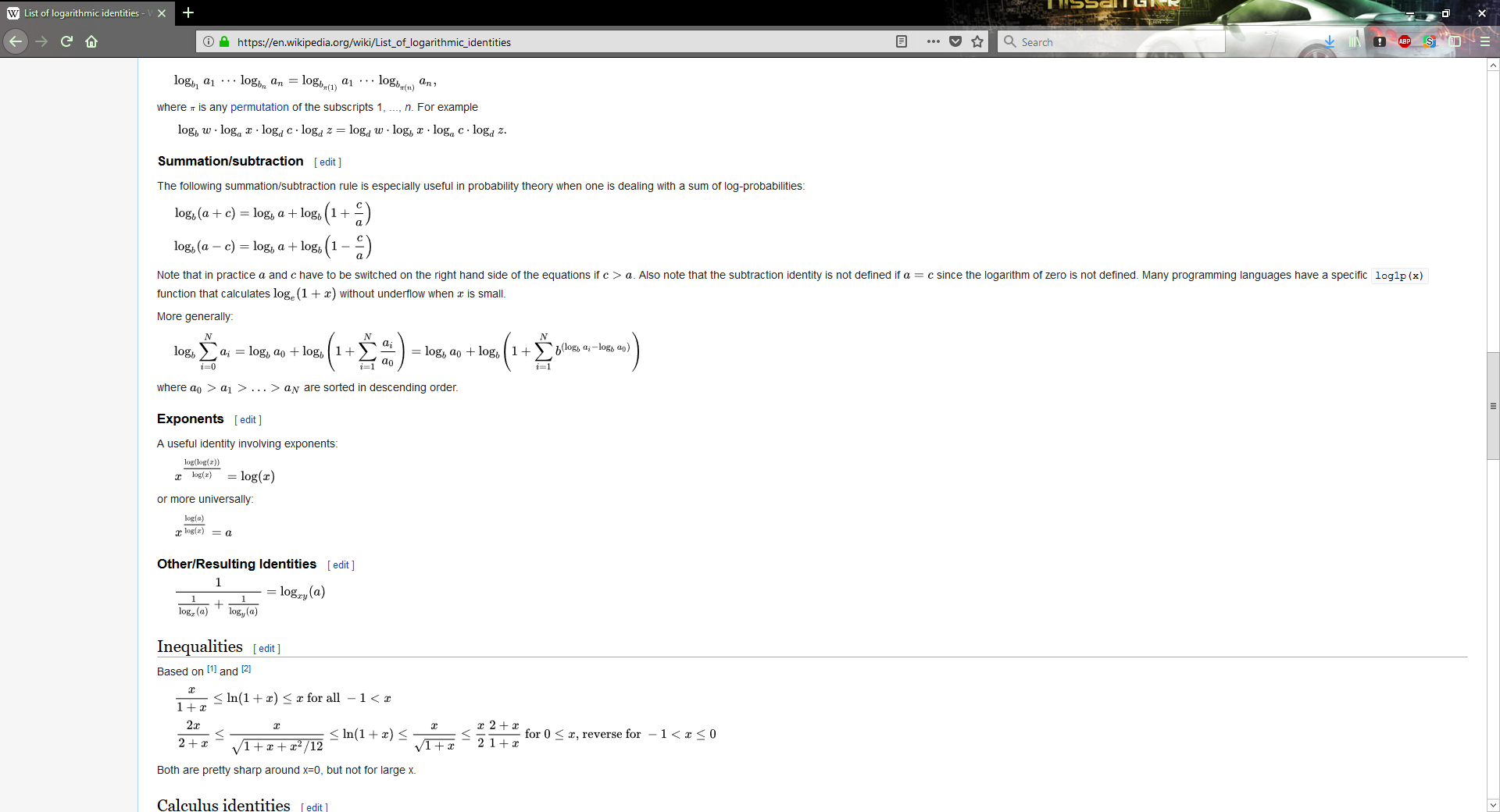


1. Problem 4: Unsupervised learning
   1. *How many parameters?*

10\*784+K

* 1. *Derive gradient of log marginal likelihood*







* 1. 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)

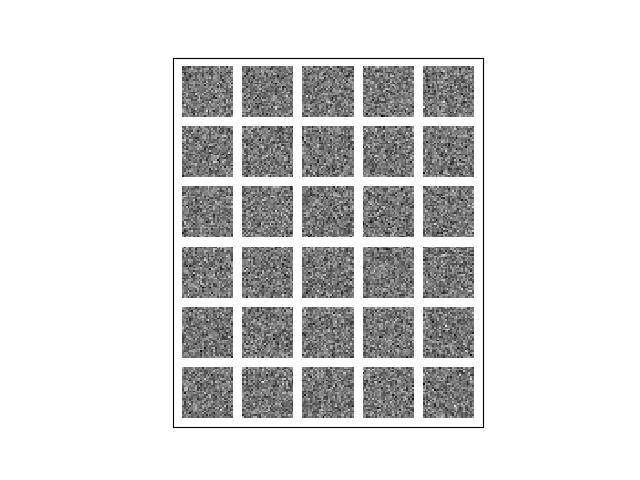
third\_term = third\_term + 10\*\*power

log\_prob = log\_pi\_c + second\_term + third\_term

print(log\_prob)

results.append(log\_prob)

return -1\*np.array(results)



* 1. 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')

