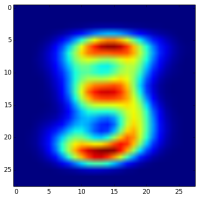
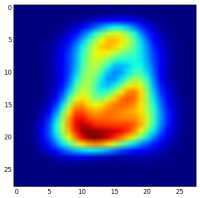
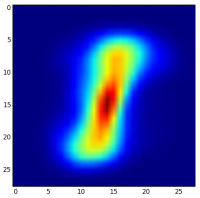
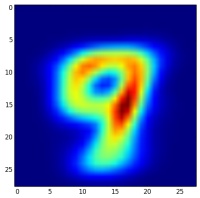
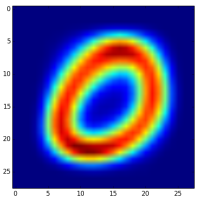
CS189–Spring 2016 — Homework 7Solutions

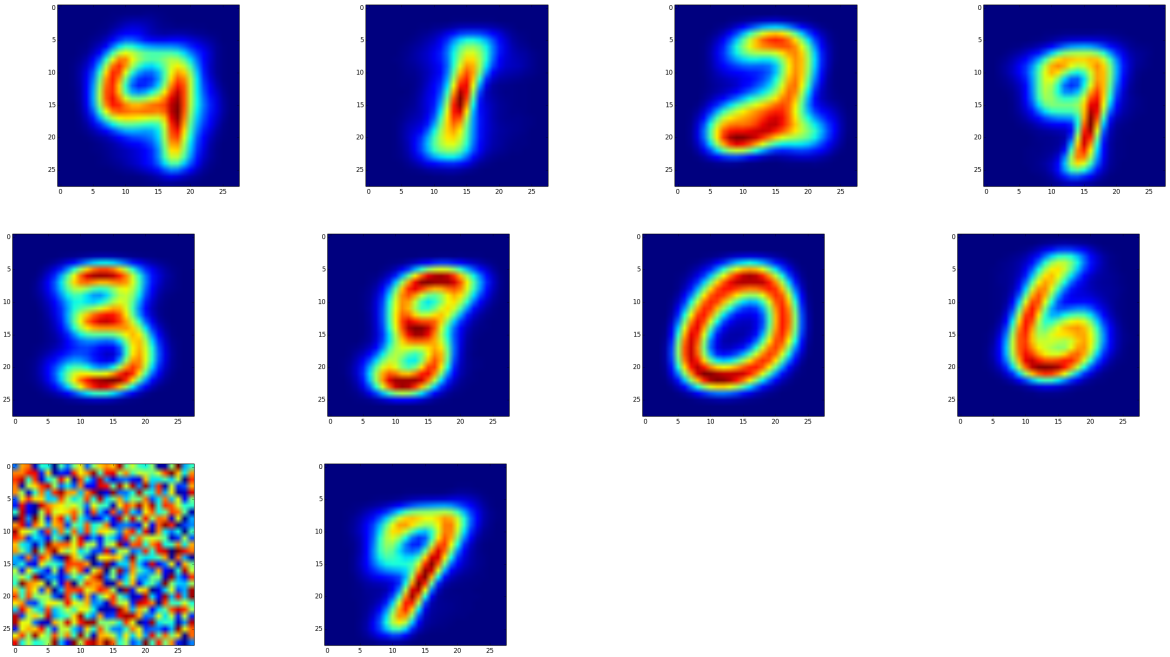
Andy Chu, SID 23042499, PUT SOMETHING HERE

1 Problem

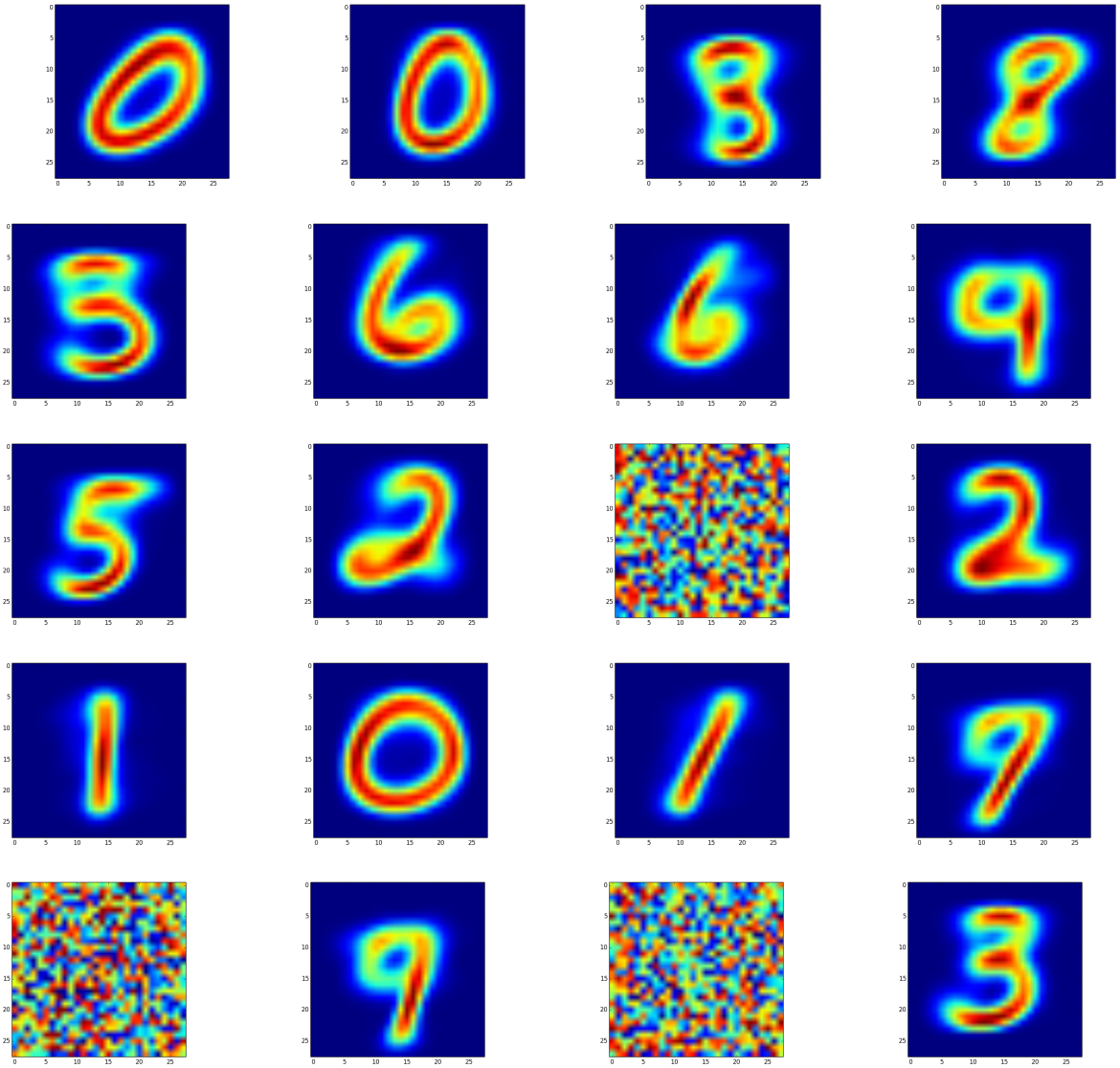
k = 5 for 50 iterations:

k = 10 for 50 iterations:



k = 20 for 50 iterations:



K-mean loss does vary in different runs. When k = 5 was run for 10 iterations, the loss was at first 102,135,443.731 and then it was 103,592,741.946 when run again.

2 Problem 2.2

For the simplest recommender system, the accuracy on validation set is 0.6203. For the advanced method with k = 10, the accuracy is 0.6491. The advanced method with k = 100, the accuracy is 0.6894. The advanced method with k = 1,000, the accuracy is 0.6940. The accuracies of the advanced system are all better than the accuracy of the simple system.

3 Problem 2.3

Latent vector with mean square error for d = 2 is 0.6477 with MSE of 20,703,188. Latent vector with mean square error for d = 5 is 0.6667 with MSE of 19,192,013. Latent vector with mean square error for d = 10 is 0.6520 with MSE of 17,398,454. Latent vector with mean square error for d = 20 is 0.6284 with MSE of 14,458,503.
We see that as we increase d, the MSE decreases.

When we change our loss function to take into account sparse data, having k is 10 clusters, theshold of 0.1, learning rate for gradient descent of 0.0001, and lambda regularization 100, the MSE is 1,2818,257 and the accuracy is 0.7255. We see that the MSE with this loss function is smaller than that of MSE when loss function is mean square. The validation accuracy for loss function that takes into account sparse data is also better than the validation accuracy of loss function for mean square.

Kaggle score: 0.71434

4 Appendix

python kmeans.py to run kmeans for 1

```
import numpy as np
from scipy import io, spatial, misc
import scipy.misc
from matplotlib import pyplot as plt
import random









train_contents = io.loadmat('images.mat')

images = train_contents['images']


pixel_images = images[0,:,:]
for i in range(1, 28):

    pixel_images = np.vstack([pixel_images, images[i,:,:]])



pixel_images = pixel_images.T


class KMeans(object):
    def __init__(self, k=10, iters=300):
        self.k = k
        self.centroids = {}
        self.iters = iters




    def train(self, data):
        #initialize the centroids randomly with 784 pixels with values up to 255
        for i in range(self.k):
            self.centroids[i] = np.random.choice(256, 784) #256 because the max
    pixel in 255

        iters = 0

        while iters < self.iters:
            clusters = {}
            J = 0.0
            index = 0
            #if ((iters % 10) == 0):
            print "training", iters
            for s in data:
                minDist = 3000000    #initialize to number bigger than 255*784 =
    200,175
                label = -1
```

```
                # find the label c that has smalled euclidean distance with data point
                for c in self.centroids:
                    dist = spatial.distance.euclidean(s, self.centroids[c])
                    if dist < minDist:
                        minDist = dist
                        label = c

                #create new group if smallest label is not in clusters
                if label not in clusters:
                    clusters[label] = []

                #add the data to cluster it belongs to
                clusters[label].append(index)

                index += 1
            # take mean of each cluster to find the new centroid
            # increment the loss to figure out new loss
            for c in clusters:
                self.centroids[c] = np.mean(data[clusters[c]], axis=0)
                for g in data[clusters[c]]:
                    J += spatial.distance.euclidean(g, self.centroids[c])

            print "Loss is", J, "for iteration ", iters

            iters += 1

        print "final loss of", J, "for iteration ", iters

iters = 50

km5 = KMeans(5, iters)
km5.train(pixel_images)

for n in km5.centroids:
    np.savetxt('q1/km5/' + str(n), km5.centroids[n].reshape(28,28))

#after write files, then can use this for loop to plot each of the images
for i in range(5):
    p = np.loadtxt('q1/km5/' + str(i))

    plt.imshow(p)
    plt.savefig('q1/km5/' + str(i))


km10 = KMeans(10, iters)
km10.train(pixel_images)

for n in km10.centroids:
    np.savetxt('q1/km10/' + str(n), km10.centroids[n].reshape(28,28))

#after write files, then can use this for loop to plot each of the images
for i in range(10):
    p = np.loadtxt('q1/km10/' + str(i))

    plt.imshow(p)
    plt.savefig('q1/km10/' + str(i))


```

```
112 km20 = KMeans(20, iters)
113 km20.train(pixel_images)
114
115 for n in km20.centroids:
116     np.savetxt('q1/km20/' + str(n), km20.centroids[n].reshape(28,28))
117
118 #after write files, then can use this for loop to plot each of the images
119 for i in range(20):
120     p = np.loadtxt('q1/km20/' + str(i))
121
122     plt.imshow(p)
123     plt.savefig('q1/km20/' + str(i))
```

python avg.py to run simple recommender system and advanced system in 2.2

```
import numpy as np
from scipy import io, spatial, misc
import scipy.misc
from matplotlib import pyplot as plt
from scipy import io, spatial, misc
import random


train_contents = io.loadmat('joke_train.mat')

train = train_contents['train']

validation = 'validation.txt'

class recommend_avg(object):
  def __init__ (self, data):
    self.avg = np.array([np.nanmean(data, axis=0)]).T
    #print self.avg.shape

  def predict(self, file):
    pred = []
    f = open(file, 'r')
    for line in f:
      line = line.split(',')
      joke_num = int(line[1]) - 1 # - 1 because of how python starts with 0 for
    index
      joke_avg_rating = self.avg[joke_num][0]

      if joke_avg_rating > 0:
        pred.append(1)
      else:
        pred.append(0)
    return pred


def accuracy(prediction, validation):
  correct = 0.0
  total = 0
  f = open(validation, 'r')
  for line in f:
    line = line.split(',')
    if (int(line[2])) == int(prediction[total]):
      correct += 1
    total += 1
  return correct/float(total)

ra = recommend_avg(train)
prediction = ra.predict(validation)
print "standard accuracy is", accuracy(prediction, validation)


class advanced(object):
  def __init__(self, data, k):
    #self.data = np.array([np.nan_to_num(data)])
    self.data = np.nan_to_num(data)  #(24983, 100)
    self.k = k
    self.avg = {}
```

```
57    #print "data", self.data, self.data[0][0]     #looks good because nan replaced
      by 0's
58
59    def train(self):
60      #print "time to train", "data shape", self.data.shape, self.data
61      avg = self.avg
62      #distLst = []
63      #print "avg init", avg
64      avgcluster = []
65      #print "data shape", self.data.shape
66      for i in range(101):                #101 because validation is 100
67      #for i in range(self.data.shape[0]):                #need to start at 0, else
        first 1 is not counted
68        for j in range(self.data.shape[0]):
69          if (i != j):
70            #print "calc euclidean for", i, j
71            #for uniitialized case, store into dictionary the user and also k,2 values
       that are bigger than the
72            #biggest euclidean distance possible
73            #having the biggest euclidean distance possible will allow all initialized
       values to be replaced
74            if i not in avg:
75              #print "key" + str(i) + " not in dictionary"
76              add = np.empty([self.k, 2])
77              #print "initialize this", add
78              #print "dim of initialize", add.shape
79              add.fill(201)  #since the largest distance is sqrt((20^2)*100) is 200
80              #print "change all in add to particular value", add
81              #print "add key", i
82              avg.update({i:add})
83            #check euclidean distance and self.avg when have a better euclidean
      distance
84            else:
85              # print "i", self.data[i].shape
86              # print "i is", self.data[i]
87              # print "j is", self.data[j]
88              #dist = np.linalg.norm((self.data[i], self.data[j]))
89              dist = spatial.distance.euclidean(self.data[i], self.data[j])
90              #distLst.append(dist)
91              usrData = avg.get(i)
92              usrDist = usrData[:,0]
93              evict = np.amax(usrDist)
94              index = np.argmax(usrDist)
95              if evict > dist:
96                #usrData[index] = np.asarray([dist, j])
97                avg[i][index] = [dist, j]
98
99
100       #if (j== 101):
101     #   break
102     #if (i==101):
103       #   break
104     #print avg[i], "for ", i
105     close = avg.get(i)[:,1].astype(int)
106     #print self.data[close]
107     #print self.data[close].shape, "shape of closest neighbors"
108     cluster = np.mean(self.data[close],axis=0)
109     #print cluster.shape, "after mean"
110     avgcluster.append(cluster)
```

```
111       print "on", i
112     self.avg = avgcluster
113     #print len(self.avg), "shape of avg cluster"
114
115
116   def predict(self, file):
117     pred = []
118     f = open(file, 'r')
119     for line in f:
120       line = line.split(',')
121       joke_num = int(line[1]) - 1 # - 1 because of how python starts with 0 for
      index
122       user = int(line[0]) - 1
123       #print "self.avg[user]", self.avg[user]
124       #print "joke num", joke_num
125       #print "value is", self.avg[user][joke_num]
126       #print "user", user, "joke num", joke_num
127       joke_avg_rating = self.avg[user][joke_num]
128
129       if joke_avg_rating > 0:
130         pred.append(1)
131       else:
132         pred.append(0)
133     return pred
134
135       #print "max dist", max(distLst), "min dist", min(distLst)
136
137 adv = advanced(train, 10)
138 #print "adv object", adv.data, adv.k, adv.avg
139 adv.train()
140 prediction = adv.predict(validation)
141 print "accuracy advanced 10", accuracy(prediction, validation)
142
143 adv = advanced(train, 100)
144 #print "adv object", adv.data, adv.k, adv.avg
145 adv.train()
146 prediction = adv.predict(validation)
147 print "accuracy advanced 100", accuracy(prediction, validation)
148
149 adv = advanced(train, 1000)
150 #print "adv object", adv.data, adv.k, adv.avg
151 adv.train()
152 prediction = adv.predict(validation)
153 print "accuracy advanced 1000", accuracy(prediction, validation)
```

python latent.py to run latent factor model with mean square loss function and loss function that takes into account sparse data for 2.3

```
import numpy as np
from scipy import io, spatial, misc
import scipy.misc
from matplotlib import pyplot as plt
import random

DEBUG = 0


#print random.__file__

#after write files, then can use this for loop to plot each of the images



train_contents = io.loadmat('joke_train.mat')

images = train_contents['train'] #(24983,100)

orig = images

validation = 'validation.txt'

testing = 'query.txt'

class joke(object):
  def __init__(self, d):
    self.d = d
    self.u = []        #(d, 100)
    self.v = []        #(24983, d)

  def train(self, data):
    image = np.nan_to_num(data)
    mean = np.mean(image, axis=0)
    image = image - mean
    U, s, V = np.linalg.svd(image, full_matrices=False)
    sort = np.argsort(s)[::-1]            #[::-1] <-- to reverse the order
    #print "image", image, "mean", mean, "new image", image
    #print U.shape, V.shape, s.shape
    #print "sort", sort, "np.argsort", np.argsort(s)
    d = self.d
    U = U[:, sort]
    V = V.T[:, sort]
    S = np.diag(s[sort])
    self.u = np.dot(U[:,:d], np.sqrt(S[:d,:d]))
    self.v = np.dot(np.sqrt(S[:d,:d]), V[:,:d].T)
    #print "v", self.v.shape, "u", self.u.shape
    print "MSE for " + str(self.d)
    mse = np.nansum((np.dot(self.u, self.v) - image)**2)
    print mse

  def predict(self, file):
    pred = []
    f = open(file, 'r')
    for line in f:
      line = line.split(',')
```

```
      user = int(line[0]) - 1
      joke_num = int(line[1]) - 1 # - 1 because of how python starts with 0 for
    index
      joke_avg_rating = np.dot(self.u[user,:], self.v[:,joke_num])

      if joke_avg_rating > 0:
        pred.append(1)
      else:
        pred.append(0)
    return pred

def accuracy(prediction, validation):
  correct = 0.0
  total = 0
  f = open(validation, 'r')
  for line in f:
    line = line.split(',')
    if (int(line[2])) == int(prediction[total]):
      correct += 1
    total += 1
  return correct/float(total)


j = joke(2)
j.train(images)
prediction = j.predict(validation)
print "accuracy for meansq " + str(2), accuracy(prediction, validation)

j = joke(5)
j.train(images)
prediction = j.predict(validation)
print "accuracy for meansq " + str(5), accuracy(prediction, validation)

j = joke(10)
j.train(images)
prediction = j.predict(validation)
print "accuracy for meansq " + str(10), accuracy(prediction, validation)

j = joke(20)
j.train(images)
prediction = j.predict(validation)
print "accuracy for meansq" + str(20), accuracy(prediction, validation)

class adv(object):
  def __init__(self, d, threshold, grad, lamb):
    self.d = d
    self.u = []       #(d, 100)
    self.v = []       #(24983, d)
    self.threshold = threshold
    self.grad = grad
    self.lamb = lamb
  def train(self, data):
    image = np.nan_to_num(data)
    mean = np.mean(image, axis=0)
    image = image - mean
    U, s, V = np.linalg.svd(image, full_matrices=False)
    sort = np.argsort(s)[::-1]            #[::-1] <-- to reverse the order
    # print "image", image, "mean", mean, "new image", image
    # print U.shape, V.shape, s.shape
```

```
    # print "sort", sort, "np.argsort", np.argsort(s)
    d = self.d
    U = U[:, sort]
    V = V.T[:, sort]
    S = np.diag(s[sort])
    self.u = np.dot(U[:,:d], np.sqrt(S[:d,:d]))
    self.v = np.dot(np.sqrt(S[:d,:d]), V[:,:d].T)
    # print "v", self.v.shape, "u", self.u.shape
    # print "loss for " + str(self.d)
    #print "dot of u and v ", np.dot(self.u, self.v).shape
    print (np.dot(self.u, self.v) - orig)**2
    print np.sum(np.multiply(self.u, self.u))
    print np.sum(np.multiply(self.v, self.v))
    loss = np.nansum((np.dot(self.u, self.v) - orig)**2) + self.lamb* (np.sum(np.multiply(self.u, self.u)) + np.sum(np.multiply(self.v, self.v)))
    print "first loss", loss
    iteration = 0
    prev = loss
    while True:
      #print "u", self.u, "u shape", self.u.shape
      #print "v", self.v, "v shape", self.v.shape
      #print "dot product", np.dot(self.u, self.v), "dot shape", np.dot(self.u, self.v).shape
      #print "image", image, "image shape", image.shape
      #print "subtract image from dot product", np.dot(self.u, self.v) - image
      #self.u -= 2*(np.dot(np.nan_to_num((np.dot(self.u, self.v) - orig)), self.v.T) + self.lambd*self.u)
      if (iteration ==501):
        print "converged on iteration ", iteration, " with loss of ", loss
        return

      self.u -= 2 * self.grad*(np.dot((np.nan_to_num(np.dot(self.u, self.v) - orig)), self.v.T) + self.lamb * self.u)
      #self.v -= 2*(np.dot(self.u.T, np.nan_to_num(np.dot(self.u, self.v) - orig)) + self.lambd*self.v)
      self.v -= 2 * self.grad*(np.dot(self.u.T, np.nan_to_num((np.dot(self.u, self.v) - orig))) + self.lamb * self.v)
      #print "new u", self.u
      #print "new v", self.v
      loss = np.nansum((np.dot(self.u, self.v) - orig)**2) + self.lamb* (np.sum(np.multiply(self.u, self.u)) + np.sum(np.multiply(self.v, self.v)))
      if (np.abs(prev - loss) < self.threshold):
        print "converged on iteration ", iteration, " with loss of ", loss
        return
      if ((iteration % 100) == 0):
        print "iteration ", iteration
        print "loss of ", loss
        print "prev - loss", np.abs(prev - loss)
        prediction = j.predict(validation)
        print "accuracy for", iteration, "is", accuracy(prediction, validation)

      prev = loss
      iteration += 1


  def predict(self, file):
    pred = []
    f = open(file, 'r')
    for line in f:
```

```
      line = line.split(',')
      user = int(line[0]) - 1
      joke_num = int(line[1]) - 1 # - 1 because of how python starts with 0 for index
      joke_avg_rating = np.dot(self.u[user,:], self.v[:,joke_num])

      if joke_avg_rating > 0:
        pred.append(1)
      else:
        pred.append(0)
    return pred

  def test(self, file):
    pred = []
    f = open(file, 'r')
    w = open('kaggle_submission.txt', 'w')
    w.write('Id,Category\n')
    for line in f:
      line = line.split(',')
      user = int(line[1]) - 1
      joke_num = int(line[2]) - 1 # - 1 because of how python starts with 0 for index
      joke_avg_rating = np.dot(self.u[user,:], self.v[:,joke_num])

      if joke_avg_rating > 0:
        w.write(str(line[0]) + ',' + str(1) + '\n')
      else:
        w.write(str(line[0]) + ',' + str(0) + '\n')
    w.close()
    return pred


j = adv(10, 0.1, 0.0001, 100)
j.train(images)
prediction = j.predict(validation)
print "accuracy for sparseAdjusted ", accuracy(prediction, validation)
j.test(testing)
```