CS 189: Homework 7

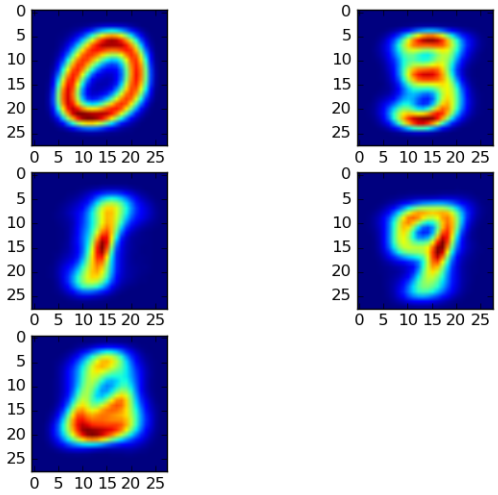
Michael Stephen Chen
Kaggle Acct: michaelstchen
SID: 23567341

May 3, 2016

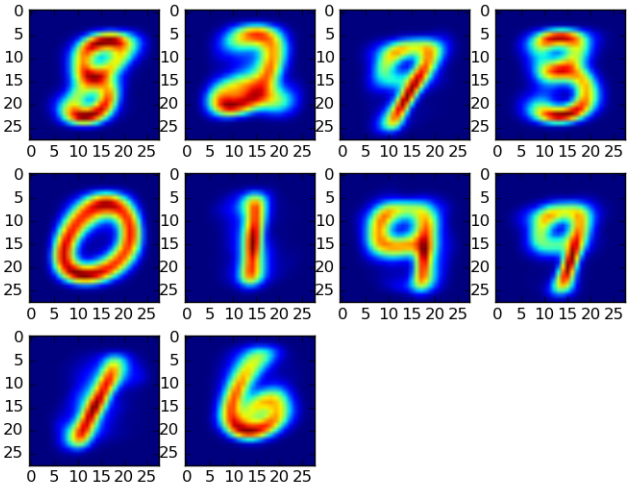
Problem 1: K-Means Clustering

For K-means clustering, I randomly initialize each of the sample points to one of the k clusters. The initial means are computed based on these random groupings. The updating of means/groupings terminatess when the loss no longer decreases. The loss is calculated as the sum of the squared Euclidean distances of every point from their respective cluster's mean. See *digits_kmeans.py* for the code.

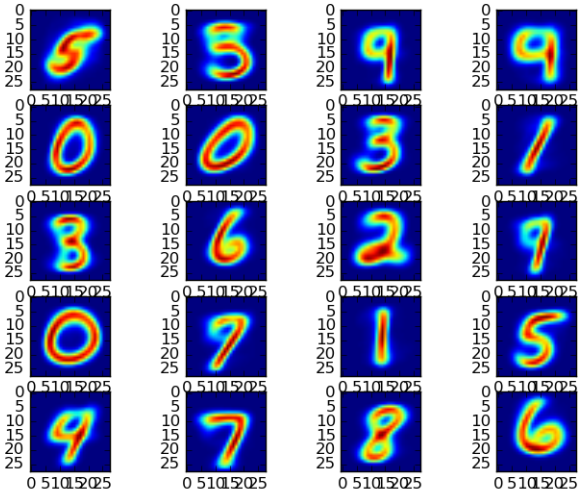
1. Visualization for $k = 5$ clusters



2. Visualization for $k = 10$ clusters



3. Visualization for $k = 20$ clusters



The k-means does not seem to vary significantly from run to run when I randomly initialize the cluster groupings. For one run with $k = 5$, I got a final k-means error of 2719265.27794. However on the subsequent run I got 2719265.19283. On the third run, I got 2719264.91943. So the exact value varies slightly from run to run, but only by a little it seems.

I noticed that some people on piazza were getting error values that are a magnitude or two larger than mine...I think it may be due to where in the iteration the error is calculated (I calculate error after updating the means but before updating the groupings).

Problem 2: Joke Recommender System

1. **Average Rating**

   See *joke_avg.py* for the implementation. The prediction accuracy on the validation set was **0.620325**

2. **K-Nearest Neighbors**

   See *joke_nn.py* for the implementation. The table below presents the results.

k	$ValidAcc$
10	0.642005
100	0.688618
1000	0.694309

   For all three k-values, the validation accuracy was better than the average approach. Also the accuracy increases with increasing k, at least for the range of values I used.

3. **Latent Factor Model**

   (a) See *joke_lf.py*. "nan" values are replaced with 0, and SVD is used to find the U (left singular vectors), D (diagonals with singular values), and V (right singular vectors) matrices.

   (b) The MSE was calcuated for singular vector truncations to $d = 2, 5, 10, 20$. The results are presented in the table below.

d	MSE	$ValidAcc$
2	25501804	0.702710
5	25497496	0.679675
10	25491425	0.652304
20	25480577	0.598103

(c) To minimize the loss function I essentially used an alternating ridge regression implemetation. For each iteration I would first solve for the $\{u_i\}$ values keeping the $\{v_j\}$ values constant, which amounted to solving the following characteristic equation:

$$V = (U^T U + \lambda I)^{-1} U^T R$$

Then I would solve for the $\{v_j\}$ values keeping the $\{u_i\}$ values constant, which amounted to solving the following characteristic equation:

$$U = (V^T V + \lambda I)^{-1} V^T R^T$$

See *joke_lfsparse.py* for the code.

(d) The results of the new implementation are presented below. The values were updated for 100 iterations (at which point the mse had already leveled off for quite a while). The penalty was chosenby trial and error to be $\lambda = 212.5$.

d	MSE	$ValidAcc$
2	20905131	0.705962
5	19490186	0.720325
10	17866870	0.721138
20	15370695	0.710840

We see that the both the mse and the validation accuracies are higher for this new implementation as compared to the algorithm used in step 2.

4. My best, and only, Kaggle submission had an accuracy of **0.70820**. The predictions were made using the sparse latent factor model with $lambda = 212.5$.

Appendix: digits_kmeans.py

```
import time
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat


''' PREPROCESSING DATA '''
# Load image data
image_mat = loadmat(file_name="images.mat", mat_dtype=True)

# Orient image to be upright
image_data_orig = np.array(image_mat['images']).T
for i in range(0, len(image_data_orig)):
    image_data_orig[i] = image_data_orig[i].T

# Flatten image data
image_data = image_data_orig.reshape(len(image_data_orig), -1)

# Normalize images' pixel values
image_data = image_data / 255.0


''' K-MEANS CLUSTERING '''
k = 5

# Random initialization
clusters = [0] * len(image_data)
for i in range(0, len(clusters)):
    clusters[i] = np.random.randint(0, k)

loss = [0,1]
kmeans = np.zeros((k, len(image_data[0])))

while(loss[-1] != loss[-2]):
    # Updating means and calculating overall loss
    error = 0
    for ki in range(0, k):
        # mean update for cluster ki
        inds = [i for i, j in enumerate(clusters) if j == ki]
        c = image_data[inds]
        kmeans[ki, :] = np.mean(c, 0)

        # error calculations for cluster ki
        for ci in range(0, len(c)):
            error += np.sum((c[ci] - kmeans[ki,:])**2)

    loss.append(error)
    print(error)

    # Updating cluster groupings
    for di in range(0, len(image_data)):
        dists = np.array([np.sum((image_data[di] - kj)**2) \
```

```
                for kj in kmeans])
        clusters[di] = np.argmin(dists)
```

Appendix: joke_avg.py

```
import time
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

def benchmark(pred_labels, true_labels):
    errors = pred_labels != true_labels
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices

# Load joke training data
joke_mat = loadmat(file_name="joke_train.mat", mat_dtype=True)
joke_data = np.array(joke_mat["train"])

# Load joke validation data
joke_valid = []
for line in file('validation.txt', 'r'):
    valid_pt = []
    for num in line.strip().split(','):
        valid_pt.append(int(num))
    joke_valid.append(valid_pt)

joke_valid = np.array(joke_valid)

# Calculating avg score for each joke
joke_avg_score = np.nanmean(joke_data, 0)

# Predicting using just the average scores
pred = [0] * len(joke_valid)
for i in range(0, len(joke_valid)):
    pred[i] = joke_avg_score[joke_valid[i][1] - 1]
    if pred[i] > 0:
        pred[i] = 1
    else:
        pred[i] = 0

err, ind = benchmark(joke_valid[:, 2], pred)
```

Appendix: joke_nn.py

```
import time
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

def benchmark(pred_labels, true_labels):
```

```
    errors = pred_labels != true_labels
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices

# Load joke training data
joke_mat = loadmat(file_name="joke_train.mat", mat_dtype=True)
joke_data = np.array(joke_mat["train"])
joke_data = np.nan_to_num(joke_data)

# Load joke validation data
joke_valid = []
for line in file('validation.txt', 'r'):
    valid_pt = []
    for num in line.strip().split(','):
        valid_pt.append(int(num))
    joke_valid.append(valid_pt)

joke_valid = np.array(joke_valid)

# Predicting using k-nearest-neighbors
k = 1000
pred = [0] * len(joke_valid)
for i in range(0, len(joke_valid)):
    if (i % 100 == 0):
        print("iter: " + str(i))

    # subtract one because id's start are 1
    user_ind = joke_valid[i][0] - 1
    joke_ind = joke_valid[i][1] - 1

    dists = np.sum((joke_data - joke_data[user_ind])**2, 1)
    inds = dists.argsort()[:k]

    knn = joke_data[inds]
    knn_avg = np.mean(knn, 0)
    if knn_avg[joke_ind] > 0:
        pred[i] = 1
    else:
        pred[i] = 0

err, ind = benchmark(joke_valid[:, 2], pred)
print(1-err)
```

Appendix: joke_lf.py

```
import time
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

def benchmark(pred_labels, true_labels):
    errors = pred_labels != true_labels
```

```
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices

# Load joke training data
joke_mat = loadmat(file_name="joke_train.mat", mat_dtype=True)
joke_data = np.array(joke_mat["train"])
joke_data = np.nan_to_num(joke_data)

# Load joke validation data
joke_valid = []
for line in file('validation.txt', 'r'):
    valid_pt = []
    for num in line.strip().split(','):
        valid_pt.append(int(num))
    joke_valid.append(valid_pt)

joke_valid = np.array(joke_valid)

# Use SVD to find principle components
d = 20
U, D, Vt = np.linalg.svd(joke_data, full_matrices=False)
U = U[:, :d]
Vt = Vt[:d, :]

# Calculate mse
mse = np.sum((U.dot(Vt) - joke_data)**2)

# Predictions
pred = [0] * len(joke_valid)
for i in range(0, len(joke_valid)):
    user_ind = joke_valid[i][0] - 1
    joke_ind = joke_valid[i][1] - 1
    score = np.dot(U[user_ind], Vt[:, joke_ind])
    if score > 0:
        pred[i] = 1
    else:
        pred[i] = 0

err, ind = benchmark(joke_valid[:, 2], pred)
```

Appendix: joke_lfsparse.py

```
import time
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

def benchmark(pred_labels, true_labels):
    errors = pred_labels != true_labels
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices
```

```
# Load joke training data
joke_mat = loadmat(file_name="joke_train.mat", mat_dtype=True)
joke_data = np.array(joke_mat["train"])
joke_data = np.nan_to_num(joke_data)

# Load joke validation data
joke_valid = []
for line in file('validation.txt', 'r'):
    valid_pt = []
    for num in line.strip().split(','):
        valid_pt.append(int(num))
    joke_valid.append(valid_pt)

joke_valid = np.array(joke_valid)

# Use alternate least squares
d = 20
U = np.random.randn(len(joke_data), d)
V = np.random.randn(len(joke_data[0]), d)

Niter = 100
lam = 212.5
loss = []
while(Niter > 0):
    UtU_lam = np.dot(U.T, U) + lam * np.eye(d)
    UtR = np.dot(U.T, joke_data)
    V = np.dot(np.linalg.inv(UtU_lam), UtR).T

    VtV_lam = np.dot(V.T, V) + lam * np.eye(d)
    VtR = np.dot(V.T, joke_data.T)
    U = np.dot(np.linalg.inv(VtV_lam), VtR).T

    mse = np.sum((U.dot(V.T) - joke_data)**2)
    loss.append(mse)
    Niter -= 1

Vt = V.T
# Predictions
pred = [0] * len(joke_valid)
for i in range(0, len(joke_valid)):
    user_ind = joke_valid[i][0] - 1
    joke_ind = joke_valid[i][1] - 1
    score = np.dot(U[user_ind], Vt[:, joke_ind])
    if score > 0:
        pred[i] = 1
    else:
        pred[i] = 0

err, ind = benchmark(joke_valid[:, 2], pred)
print(1-err)
```