CS189, HW7: Unsupervised

Completed by: Matthew Wu

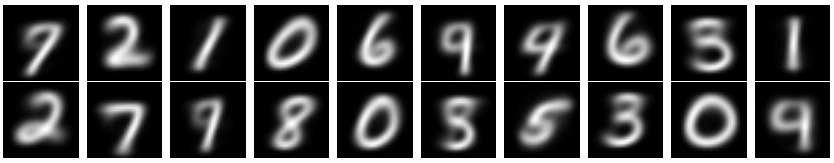
1. k-means clustering

The code used to generate the k-means clustered images is in the appendix.

Means for 5 clusters:

Means for 10 clusters:

Means for 20 clusters:



With only 5 clusters, obviously not all 10 digits can be separated into clusters. However, 0, 1, 3, and 9 appear to be fairly distinct digits in the space, and the other digits get averaged into the last cluster. Increasing the number of clusters to 10, we still can't identify all the digits, because 1, 9 and 3 are distinctive digits. We pick up on a few more digits like 8 and 7, but the remaining digits are averaged in the last cluster. After we reach 20 clusters, we have enough clusters such that we can see clusters that resemble each digit. Some digits, like 6, are fairly similar to other digits and only appear in one cluster, but other digits that are presumably very distinctive like 0 appear in multiple clusters.

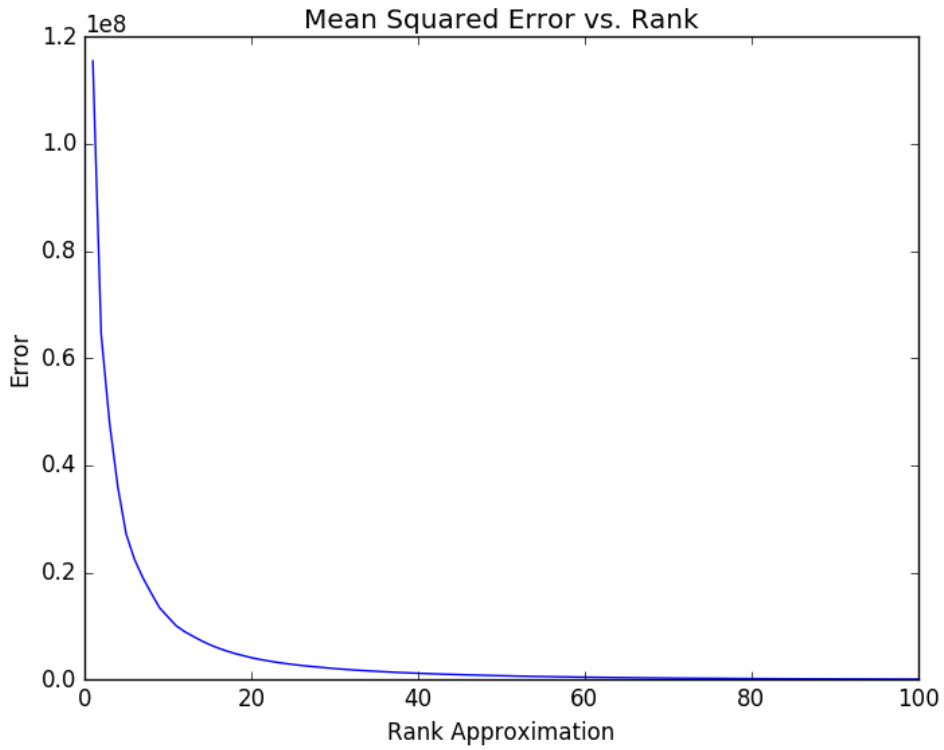
Low-Rank approximation

The code used to generate the images and graphs in this section is in the appendix.

a) Low rank approximations of the face image:

Rank-5 Image	Rank-20 Image	Rank-100 Image
		

b) Mean squared error plot:



c) Low rank approximations of the sky image:

Rank-5 Image	**Rank-20 Image**	**Rank-100 Image**
		

d) For the face image, at rank 40 it's still grainy. At rank 60, the image is still ever so slightly grainy, but the only real places where the approximation is worse than the original is a slightly blurrier beard and a slightly less smooth shadow edge. At rank 80 I can't perceive any difference from the original image.

The sky can already be approximated fairly accurately at rank 5. There's clearly fewer finer details outlining distinct clouds and reflections off the water, but overall, but they're still similar regardless. At rank 20 I can't perceive any difference from the original image.

The face requires a higher rank to approximate with more fidelity due to the face's complicated shape. There's a shirt, there's hair, there's ears, there's eyes, there's a beard, etc, and edges around all of these objects, and it's difficult to accurately capture of all these features with high detail with a low rank image. The sky, on the other hand, is a relatively simple image. All there is is the water and the clouds. The only distinctive edge is the horizon, and the details in the water and the clouds are mostly gradients.

Joke Recommender System

Opt-out

Appendix

Code for k-means clustering:

```
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
import csv
import scipy.misc
import sys
import os
import random

traindatafilename = "hw7_data/mnist_data/images"
data = scipy.io.loadmat(traindatafilename)

images = data['images']

samples = np.transpose(images.reshape((784, 60000)))
samples = samples * 1.0

def k_means(num_clusters, max_iterations = 250):
    #starting_indecies = random.sample(range(60000), num_clusters)

    #k_means++ method of picking starting indexes
    first_index = random.randrange(60000) #Pick starting point
    starting_indecies = [first_index]
    current_sample = samples[first_index]

    distances = np.sum((samples - current_sample)**2, axis=1)

    for i in range(num_clusters - 1):
        #Select a random point via kmeans++ method
        total_distance = np.sum(distances)
        target = np.random.random() * total_distance
        val = 0
        j = 0
        while True:
            val += distances[j]
            if val >= target:
                break
            j += 1
        #Add the new point and update point distances
        starting_indecies.append(j)
        current_sample = samples[j]
        dist_to_new_point = np.sum((samples - current_sample)**2, axis=1)
        distances = np.minimum(distances, dist_to_new_point)

    #Initialize the clusters
    clusters = samples[[starting_indecies]]

    iteration = 0

    #samp_norm_squared = np.sum(samples**2, axis = 1).reshape((60000, 1))
    prev_closest_clusters = None

    #Update loop
    while iteration < max_iterations:
        iteration += 1

        #Find points closest to each cluster
        #||u-v||^2 == u*u + v*v - 2(u*v)
        clust_norm_squared = np.sum(clusters ** 2, axis = 1)
        samp_dot_clust = np.inner(samples, clusters)
        #squared_distance_mat = samp_norm_squared + clust_norm_squared - 2 * samp_dot_clust
        squared_distance_mat = clust_norm_squared - 2 * samp_dot_clust

        #squared_distance_mat:
```

```
        #(num_samples, num_clusters) matrix
        #squared_distance_mat[i, j] is the distance between point i and cluster j

        #closest_cluster finds the closest cluster index for each sample point
        closest_clusters = np.argmin(squared_distance_mat, axis=1)

        if iteration > 1:
            if (closest_clusters == prev_closest_clusters).all():
                print("\nConverged")
                break

        prev_closest_clusters = closest_clusters

        #cost = 0
        #Calculate the new cluster centers
        for c in range(num_clusters):
            samples_in_cluster = samples[np.where(closest_clusters == c)]
            new_center = np.sum(samples_in_cluster, axis=0) / samples_in_cluster.shape[0]
            #cost += np.sum((samples_in_cluster - new_center)**2)
            clusters[c] = new_center

        print("\nIteration " + str(iteration))
        #print("Cost: " + str(cost))
        sys.stdout.flush()

    if iteration == max_iterations:
        print("\nMaximum iterations reached")

    #Finished finding cluster centers, Save cluster center images
    directory = 'images/mnist-' + str(num_clusters) + '-clusters/'
    if not os.path.exists(directory):
        os.makedirs(directory)
    for c in range(num_clusters):
        cluster = clusters[c]
        cluster = cluster.reshape((28, 28))
        s = directory + str(c) + '.png'
        scipy.misc.imsave(s, cluster)

#k_means(5)
#k_means(10)
#k_means(20)
```

Code for lowrank images:

```
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
import csv
import scipy.misc
import sys

load_directory = 'hw7_data/low-rank_data/'
save_directory = 'images/low-rank_approx/'

np.set_printoptions(threshold=np.inf)

def make_low_rank_image(filename, rank):
    pic_name = filename[:-4]
    picture = scipy.misc.imread(load_directory + filename)
    u, s, v = np.linalg.svd(picture, False)
    sig = np.zeros((s.shape[0], s.shape[0]))
    for i in range(rank):
        sig[i, i] = s[i]

    new_picture = np.dot(np.dot(u, sig), v)
    new_picture = np.clip(new_picture, 0, 255)
    new_picture = (new_picture + .5).astype('uint8')

    scipy.misc.imsave(save_directory + pic_name + '-' +
        str(rank) + '.png', new_picture)

def plot_MSE(filename):
    picture = scipy.misc.imread(load_directory + filename)
    u, s, v = np.linalg.svd(picture, False)
    sig = np.zeros((s.shape[0], s.shape[0]))
    error_array = []
    for i in range(100):
        sig[i, i] = s[i]
        new_picture = np.dot(np.dot(u, sig), v)
        error = np.sum((picture - new_picture) ** 2)
        error_array.append(error)
    rank_array = [i + 1 for i in range(100)]
    plt.plot(rank_array, error_array)
    plt.title("Mean Squared Error vs. Rank")
    plt.ylabel("Error")
    plt.xlabel("Rank Approximation")
    plt.show()

#make_low_rank_image('face.jpg', 5)
#make_low_rank_image('face.jpg', 20)
#make_low_rank_image('face.jpg', 100)

#make_low_rank_image('sky.jpg', 5)
#make_low_rank_image('sky.jpg', 20)
#make_low_rank_image('sky.jpg', 100)

make_low_rank_image('face.jpg', 80)

#plot_MSE('face.jpg')
```