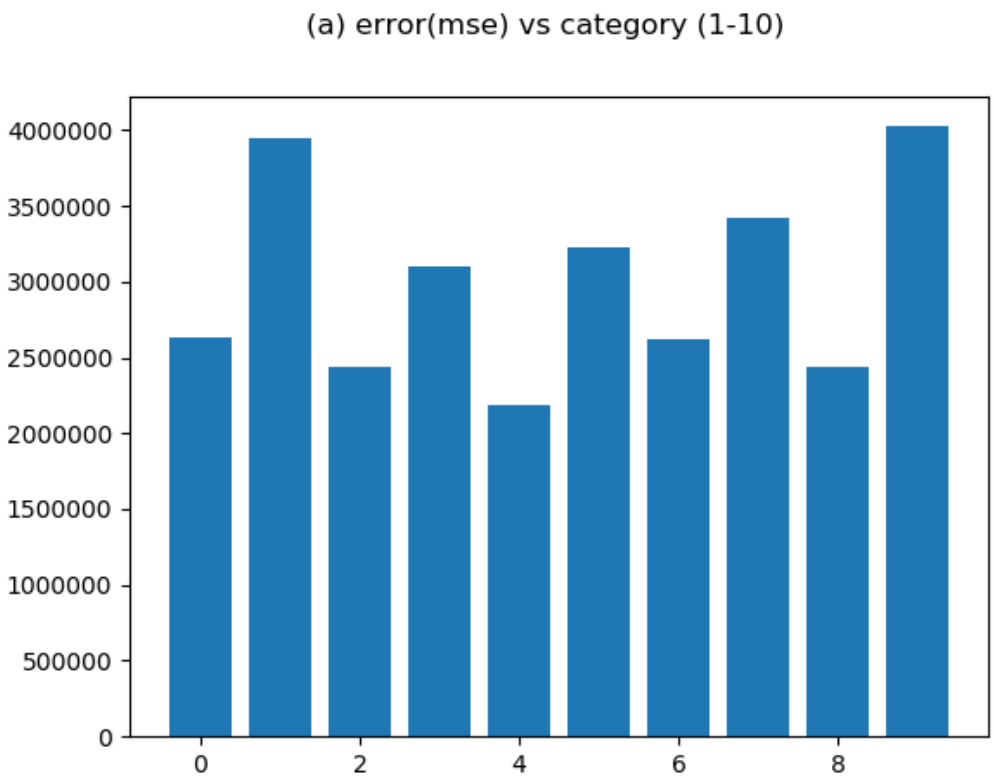
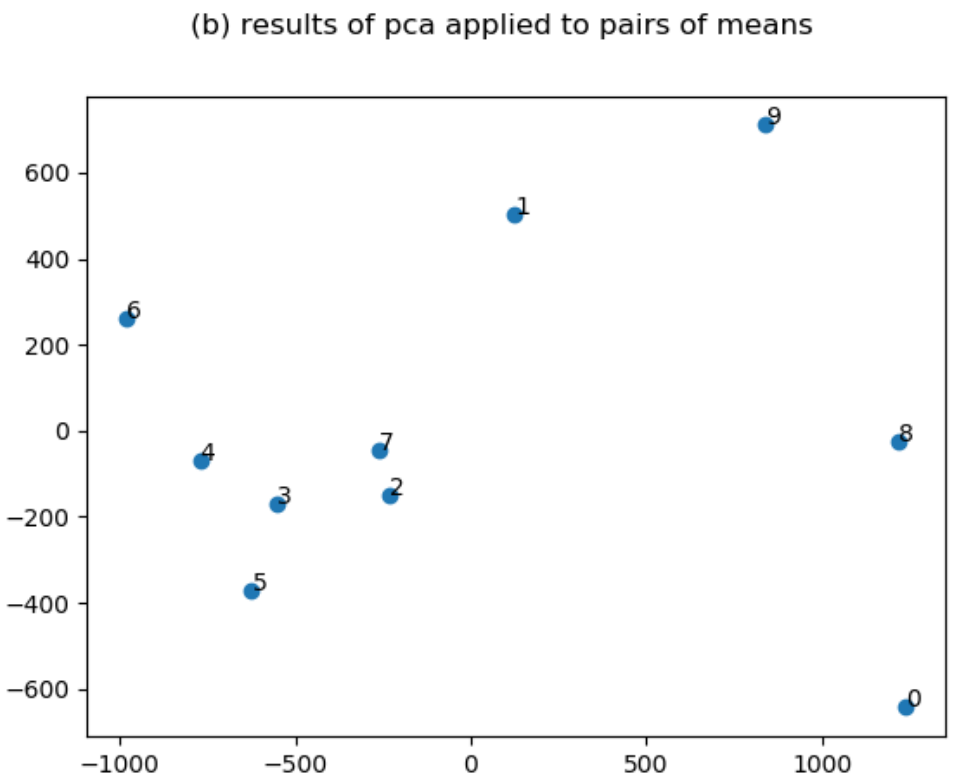
Homework 4

1. **A single plot of error(use mean squared error) vs category(1-10)**



2. A single 2D plot with the results of principal coordinate analysis applied to pairs of means:



3. **Snapshot of code:**

   a) **Principal Component Computation**

```
## A
# CHANGE LIST TO ARRAY
classes = np.array(classes_list).astype(np.int)

# GET MEAN
classes_mean, classes_mean_repeat = getMean(classes)

# NORMALIZE
classes_norm = getNorm(classes, classes_mean_repeat)

# GET COVARIANCE
classes_cov = getCov(classes_norm)

# GET EIGENVALUE, EIGENVECTOR
classes_eigval, classes_eigvec = getEigen(classes_cov)
```

```
# get mean
def getMean(classes):
    classes_mean_list = []
    classes_mean_repeat_list = []
    for i in range(class_num):
        length = len(classes[i])
        class_mean = np.mean(classes[i], axis=0)
        class_mean_repeat = np.tile(class_mean, [length, 1])
        classes_mean_list.append(class_mean)
        classes_mean_repeat_list.append(class_mean_repeat)
    classes_mean = np.array(classes_mean_list)
    classes_mean_repeat = np.array(classes_mean_repeat_list)
    return classes_mean, classes_mean_repeat

# get norm
def getNorm(classes, classes_mean_repeat):
    classes_norm_list = []
    for i in range(class_num):
        classes_norm_list.append(classes[i] - classes_mean_repeat[i])
    classes_norm = np.array(classes_norm_list)
    return classes_norm

# get covariance
def getCov(classes_norm):
    classes_cov_list = []
    for i in range(class_num):
        class_cov = np.cov(classes_norm[i], rowvar=0)
        classes_cov_list.append(class_cov)
    classes_cov = np.array(classes_cov_list)
    return classes_cov

# get eigenvalue and eigenvector
def getEigen(classes_cov):
    classes_eigval_list = []
    classes_eigvec_list = []
    for i in range(class_num):
        class_eigval, class_eigvec = np.linalg.eig(classes_cov[i])
        class_index = class_eigval.argsort()[::-1]
        class_eigval = class_eigval[class_index]
        class_eigvec = class_eigvec[:, class_index]
        classes_eigval_list.append(class_eigval)
        classes_eigvec_list.append(class_eigvec)
    classes_eigval = np.array(classes_eigval_list)
    classes_eigvec = np.array(classes_eigvec_list)
    return classes_eigval, classes_eigvec
```

b) Representing input image using the 20 principal components

```
# get reduce data
def getReduce(classes_eigvec, classes_norm, classes_mean_repeat):
    classes_reduce_list = []
    classes_reconstruction_list = []
    for i in range(class_num):
        class_feature = classes_eigvec[i][:, :pc_num]
        class_reduce = np.dot(classes_norm[i], class_feature)
        class_reconstruction = np.dot(class_reduce, class_feature.T) + classes_mean_repeat[i]
        classes_reduce_list.append(class_reduce)
        classes_reconstruction_list.append(class_reconstruction)
    classes_reduce = np.array(classes_reduce_list)
    classes_reconstruction = np.array(classes_reconstruction_list)
    return classes_reduce, classes_reconstruction
```

c) Principal coordinate analysis on pairs of mean images

```
# get distance
def getDistance(classes_mean):
    classes_distance = np.zeros([class_num, class_num])
    for i in range(class_num):
        for j in range(class_num):
            euclidean = getEuclidean(classes_mean[i], classes_mean[j])
            classes_distance[i][j] = euclidean
    return classes_distance

# get distance eigenvalue and eigenvector
def getDistanceEigen(distance_cov):
    distance_eigval, distance_eigvec = np.linalg.eig(distance_cov)
    distance_index = distance_eigval.argsort()[::-1]
    distance_eigval = distance_eigval[distance_index]
    distance_eigvec = distance_eigvec[:, distance_index]
    return distance_eigval, distance_eigvec
```

```
## B
# GET EUCLIDEAN DISTANCE
distance = getDistance(classes_mean)
# print(distance)

A = np.identity(10) - np.ones([10, 10]) * 0.1
distance = pow(distance, 2)
W = -(0.5 * np.dot(np.dot(A, distance), A.T))
distance_eigval, distance_eigvec = getDistanceEigen(W)
temp = np.zeros([2, 2])
temp[0][0] = np.sqrt(distance_eigval[0])
temp[1][1] = np.sqrt(distance_eigval[1])
X = np.dot(distance_eigvec[:, :pc_num_dist], temp)
# print(X)

figure2, ax2 = plt.subplots()
figure2.suptitle("(b) results of pca applied to pairs of means")
ax2.scatter(X[:, 0], X[:, 1])
for i in range(class_num):
    ax2.annotate(str(i), (X[i, 0], X[i, 1]))
plt.savefig("b.png")
plt.show()
```