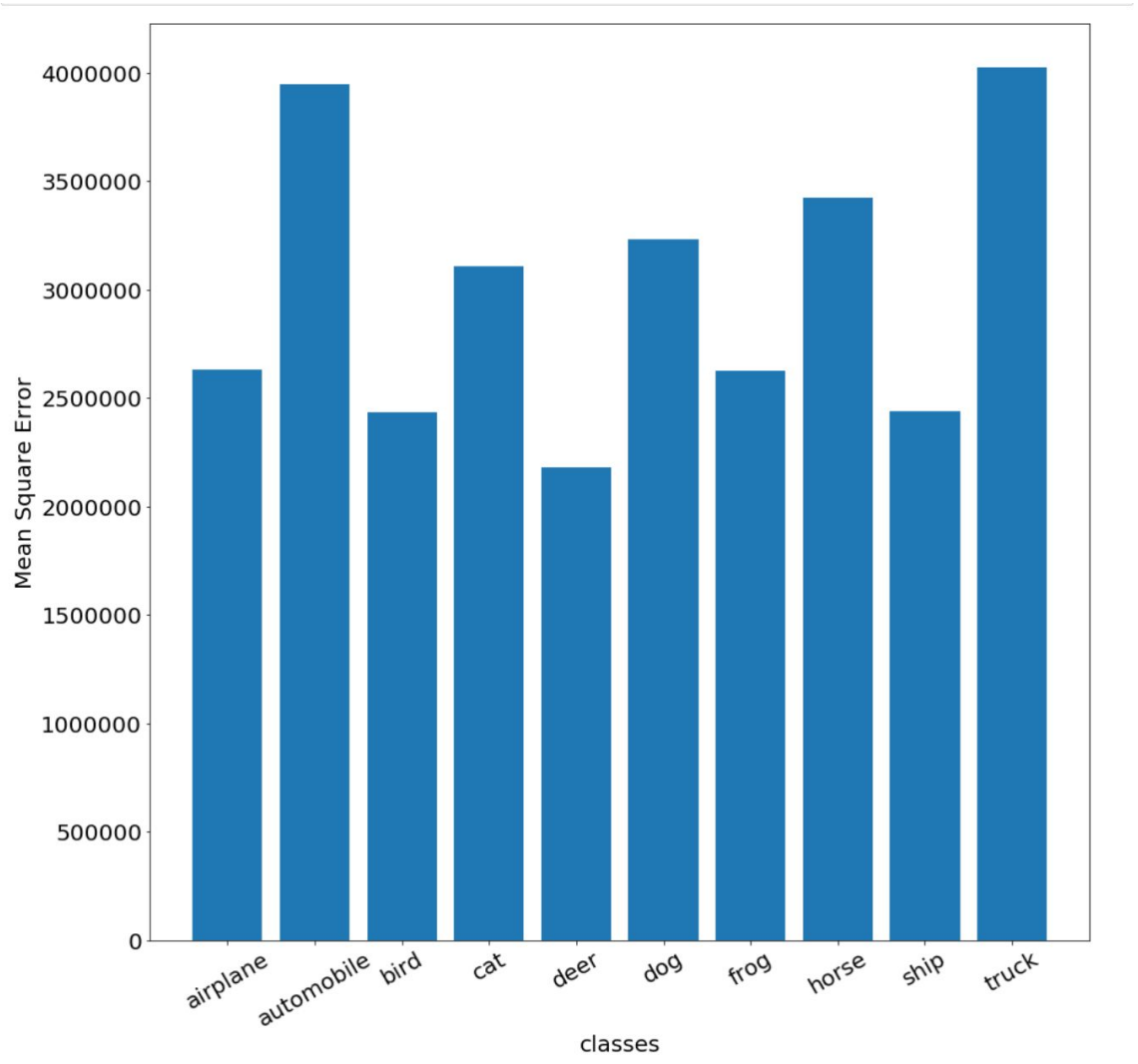
HW4

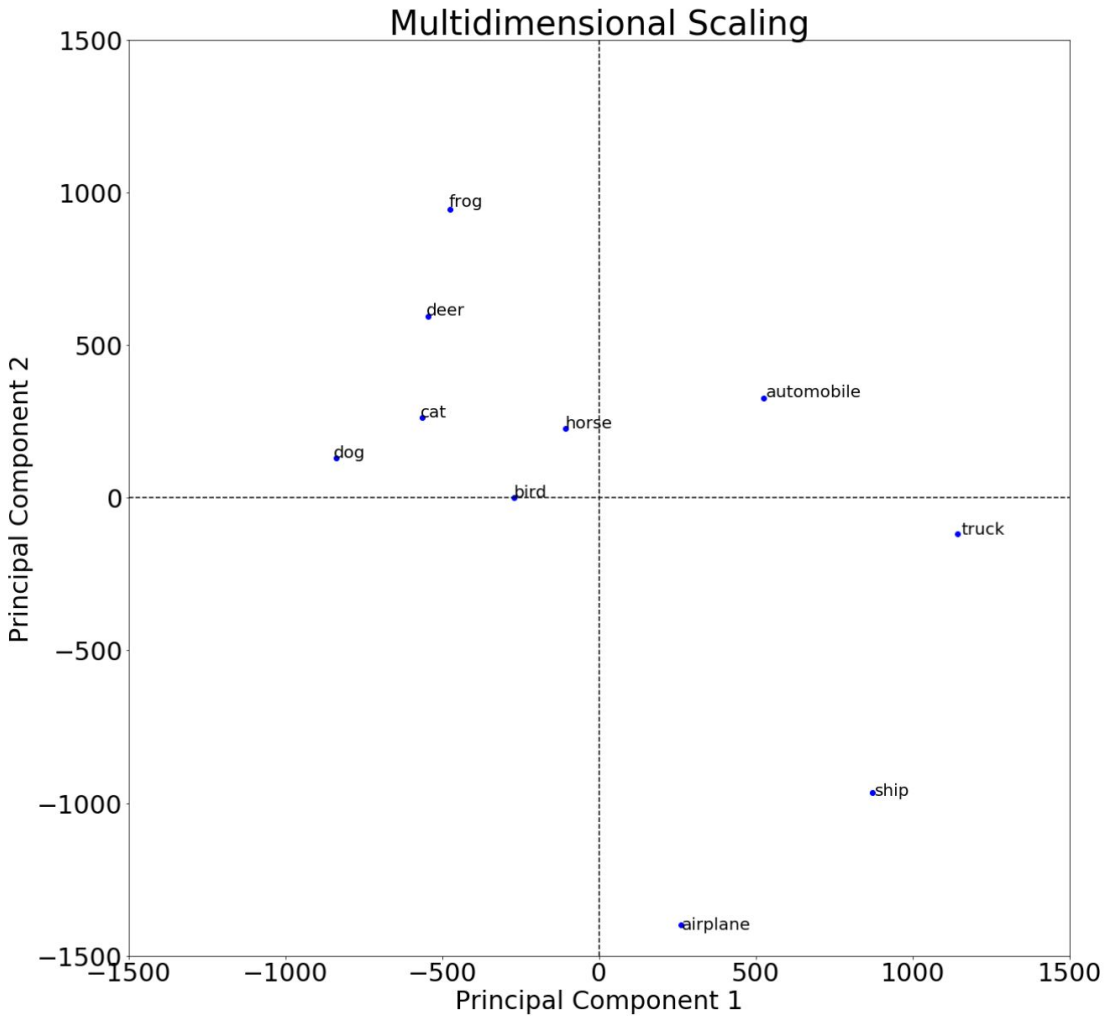
NetIDs ts8, tanvi3

Page 1 : Mean Squared Error versus Classes Graph



#We have calculated MSE on training data.

Page 2 : Principal coordinate analysis (PC2 versus PC1 graph)



Page 3:

Libraries used:

CS498 AML HW4

NetID : tanvi3

NetID : ts8

In [31]:
```
import numpy as np
import pickle
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.spatial import distance
from sklearn.manifold import MDS
```

a) principal component computation

b) representing input image using the 20 principal components

In [32]:
```
class Cifar():

    def __init__(self):
        return

    def unpickle(self,file_name):

            if file_name=='batches.meta':

                with open(file_name, 'rb') as fo:
                    dict = pickle.load(fo, encoding='Utf-8')

            else:
                with open(file_name, 'rb') as fo:
                    dict = pickle.load(fo, encoding='bytes')
            return dict

    def data_preprocessing(self,file_name):

            if file_name=='batches.meta':
                     labels=self.unpickle(file_name)
                     return labels['label_names']

            data_cifar=self.unpickle(file_name)
            return data_cifar[b'data'],data_cifar[b'labels']

    def concatenate(self,A,B,C,D,E):
        return np.concatenate((A,B,C,D,E),axis=0)


    def individual_dataset(self,X,Y,i):
        Data=np.concatenate((X,Y),axis=1)
        return Data[Data[:,-1]==i]

    def calculate_inverse_transform(self,dataset):
                pca=PCA(n_components=20)
                dataset_transform=pca.fit_transform(dataset)
                reconstruction=pca.inverse_transform(dataset_transform)
                return reconstruction

    def mean_square_error(self,dataset):
                reconstruction=self.calculate_inverse_transform(dataset)
                return np.sum((dataset-reconstruction)**2)/len(dataset)

    def euclidian_distance(self,matrix):
                return distance.cdist(matrix,matrix)
```

In [33]:
```
Obj=Cifar()
```

In [34]:
```
data1,label1=Obj.data_preprocessing('data_batch_1')
data2,label2=Obj.data_preprocessing('data_batch_2')
data3,label3=Obj.data_preprocessing('data_batch_3')
data4,label4=Obj.data_preprocessing('data_batch_4')
data5,label5=Obj.data_preprocessing('data_batch_5')
```

In [35]:
```
X_train=Obj.concatenate(data1,data2,data3,data4,data5)
Y_train=Obj.concatenate(label1,label2,label3,label4,label5)
```

c) principal coordinate analysis on pairs of mean images:

In [19]:
```
class1_mean=np.mean(class1,axis=0).reshape(1,-1)
class2_mean=np.mean(class2,axis=0).reshape(1,-1)
class3_mean=np.mean(class3,axis=0).reshape(1,-1)
class4_mean=np.mean(class4,axis=0).reshape(1,-1)
class5_mean=np.mean(class5,axis=0).reshape(1,-1)
class6_mean=np.mean(class6,axis=0).reshape(1,-1)
class7_mean=np.mean(class7,axis=0).reshape(1,-1)
class8_mean=np.mean(class8,axis=0).reshape(1,-1)
class9_mean=np.mean(class9,axis=0).reshape(1,-1)
class10_mean=np.mean(class10,axis=0).reshape(1,-1)
```

In [20]:
```
class_mean=[class1_mean,class2_mean,class3_mean,class4_mean,class5_mean,class6_mean,class7_mean,class8_mean,class9_mean
```

In [21]:
```
matrix=np.concatenate((class_mean),axis=0)
```

In [22]:
```
matrix=Obj.euclidian_distance(matrix)
matrix.shape
```

Out[22]:
```
(10, 10)
```

In [23]:
```
mds=MDS(n_components=2,dissimilarity='precomputed',random_state=1)
```

In [24]:
```
mds_transform=mds.fit_transform(matrix)
```

In [30]:
```
plt.figure(figsize=(20,20))
plt.title("Multidimensional Scaling",fontsize=40)

for i in range(len(mds_transform)):
    x = mds_transform[i][0]
    y = mds_transform[i][1]
    plt.plot(x, y, 'bo')
    plt.text(x * (1 + 0.01), y * (1 + 0.01) ,lst[i], fontsize=20)


#plt.scatter(matrix_transform[:,0],matrix_transform[:,1])
plt.xlabel("Principal Component 1",fontsize=30)
plt.ylabel("Principal Component 2",fontsize=30)
```