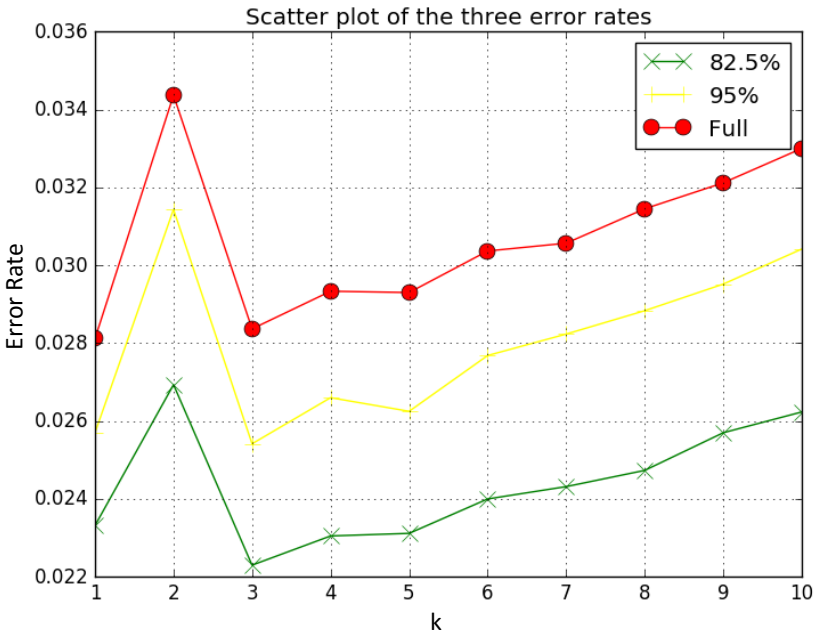
Homework 2

ANDREW ZASTOVNIK

MATH 285

Problem 1
Cross Validation with 6-folds

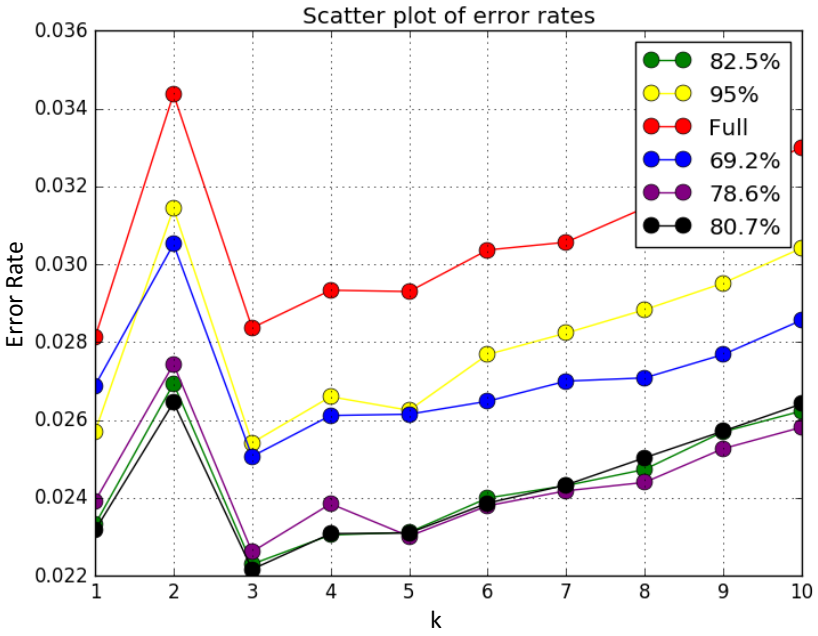


The graph to the left shows the error rates from the 6-fold cross validation on the training dataset.

We can see that k = 3 has the lowest miss-classification rate in all three levels of s.

It is interesting to note that reducing the dimensionality of the dataset actually allows us to make better predictions for every value of k.

Problem 1
More dimensionality reduction

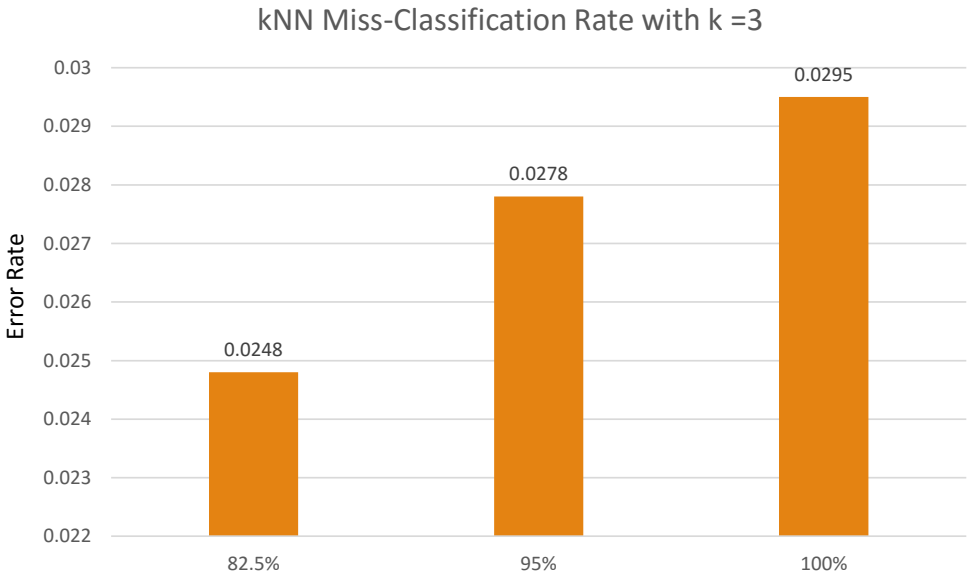


So does reducing the dimensionality even further create better results?

To the left you can see the same graph that was in the previous slide with three more error rates from 6-fold cross validation added on.

The lower dimension datasets gradually had less and less predictive power below 80% explained variance

Problem 2
Error Rates for K Nearest Neighbors



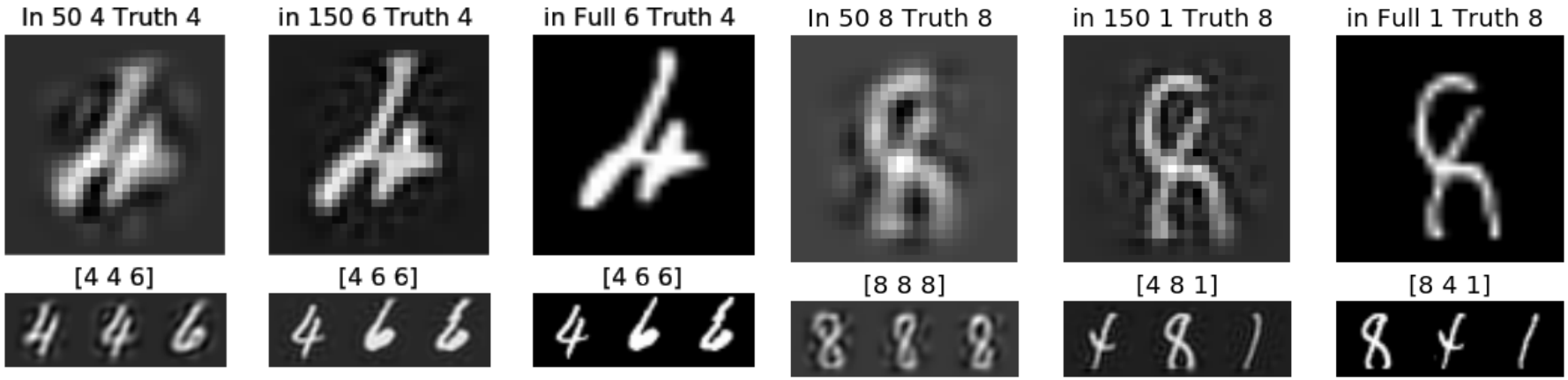
This graph is showing the error rate for the three levels of s when running k nearest neighbors on the test dataset.

PCA with 82.5% of the variance again did the best job classifying the digits

It was also much faster and only took about 4 minutes to run vs 40 minutes with the all dimensions

Problem 2

Two Images Correctly classified using PCA with 50 dimensions

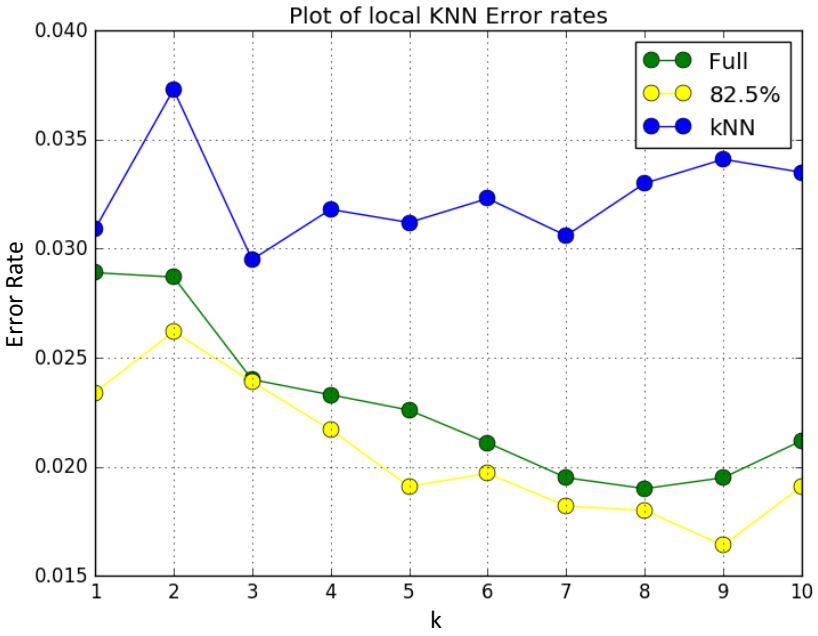


One of the effects PCA has is to sort of blur pixels that often occur together when reducing the dimensionality.

This is especially clear with the 8 since it was written very thinly but is thicker after reducing the dimensions with PCA.

Features that make digits outliers are also less important in lower dimension PCA

Problem 3
Local Kmeans

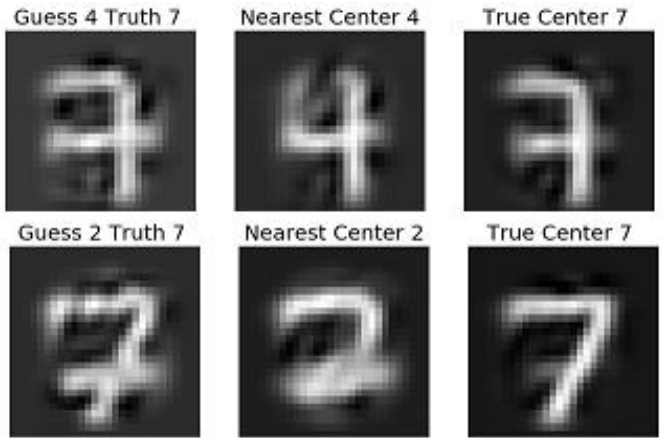


Here you can see the error rates for local kmeans with 82.5% of variance, the full local kmeans, and the full k nearest neighbor classifier.

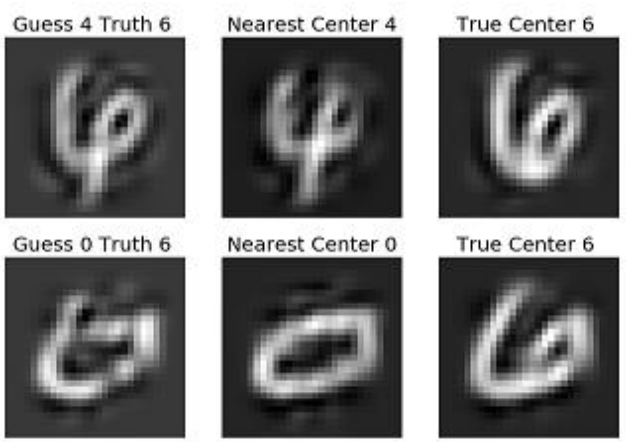
Local kmeans with 82.5% of variance did the best at classifying the digits only miss-classifying 162 at k= 9

It was also very quick only taking 20 minutes to run compared to 4 hours with the full dimension local kmeans

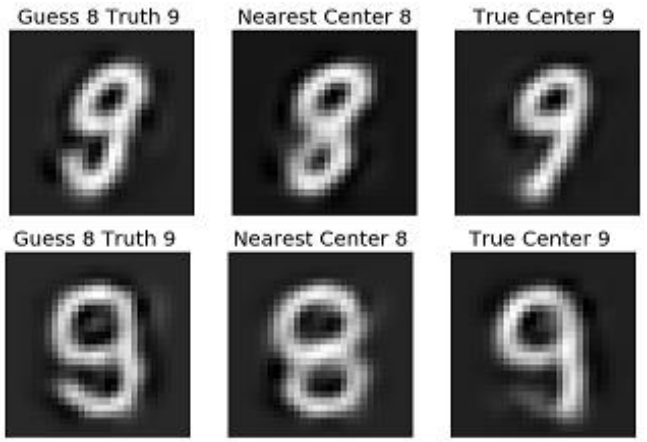
Problem 3
Examples of misclassified digits



The 7 with a bar causes a lot of problems for local kmeans

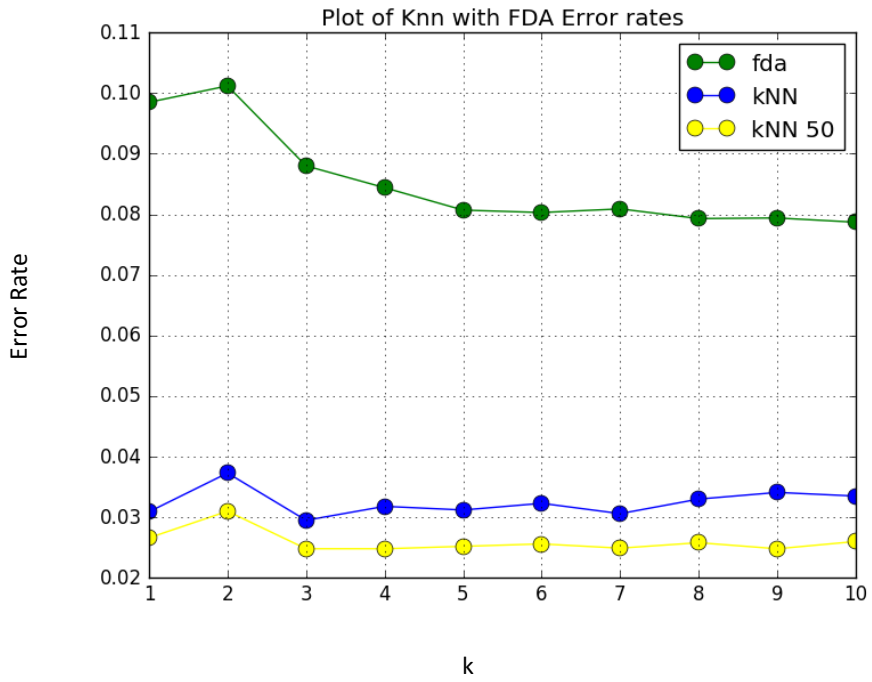


Some people have a hard time drawing sixes



Local kmeans also has a hard time classifying nines with a hook at the bottom

Problem 4
K nearest neighbors with FDA



Here we can see a plot of the error rates of k nearest neighbors with FDA and basic k nearest neighbors vs values of k.

K nearest neighbors with FDA does much worse than PCA + k nearest neighbors.

It is very quick though only taking about 3 minutes to classify the 10,000 test images

Problem 4

Confusion matrix for basic Knn

True Value \ Predicted Value	0	1	2	3	4	5	6	7	8	9
0	972	1	1	0	0	1	4	1	0	0
1	0	1130	3	0	0	0	1	0	0	1
2	8	3	1004	0	1	0	1	12	3	0
3	0	0	2	977	1	11	0	6	7	6
4	2	4	0	0	961	0	4	1	0	10
5	3	0	0	13	2	863	6	1	2	2
6	4	4	0	0	3	2	945	0	0	0
7	0	19	6	0	3	0	0	994	0	6
8	4	0	4	15	5	6	2	2	933	3
9	4	5	4	8	9	3	1	6	4	965

-

Confusion matrix for Knn with FDA

True Value \ Predicted Value	0	1	2	3	4	5	6	7	8	9
0	961	0	1	2	0	5	4	5	2	0
1	0	1112	2	4	1	4	2	2	8	0
2	12	9	946	10	11	4	11	10	19	0
3	5	4	21	898	2	38	0	16	20	6
4	1	0	4	0	932	1	11	3	6	24
5	9	5	4	48	5	768	8	12	28	5
6	11	4	8	1	4	18	912	0	0	0
7	2	18	18	10	9	3	0	941	0	27
8	5	21	11	21	17	36	15	12	831	5
9	14	4	1	7	53	5	1	18	10	896

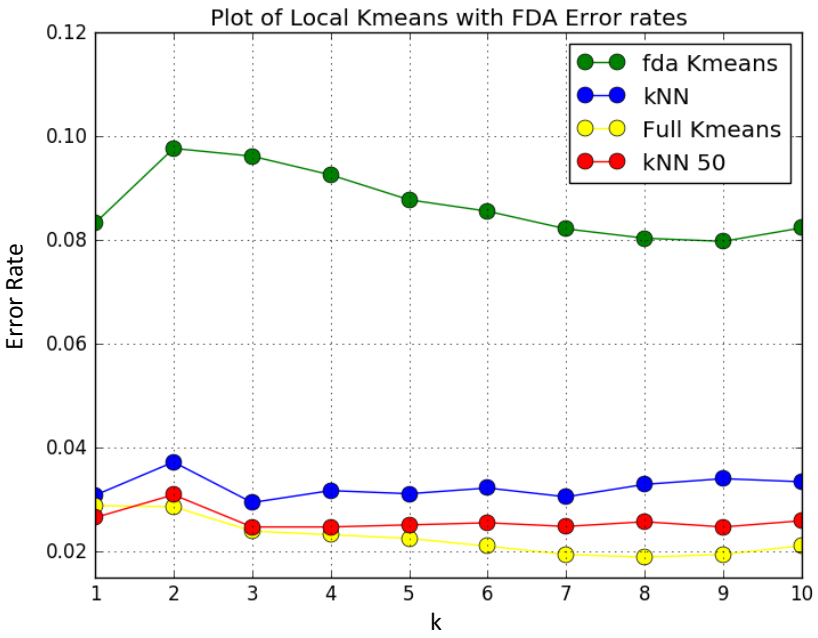
=

Basic knn – knn with FDA

True Value \ Predicted Value	0	1	2	3	4	5	6	7	8	9
0	11	1	0	-2	0	-4	0	-4	-2	0
1	0	18	1	-4	-1	-4	-1	-2	-8	1
2	-4	-6	58	-10	-10	-4	-10	2	-16	0
3	-5	-4	-19	79	-1	-27	0	-10	-13	0
4	1	4	-4	0	29	-1	-7	-2	-6	-14
5	-6	-5	-4	-35	-3	95	-2	-11	-26	-3
6	-7	0	-8	-1	-1	-16	33	0	0	0
7	-2	1	-12	-10	-6	-3	0	53	0	-21
8	-1	-21	-7	-6	-12	-30	-13	-10	102	-2
9	-10	1	3	1	-44	-2	0	-12	-6	69

The only digits knn with FDA helps to classify better are 4's not being confused with 1's. There are several digits that knn with FDA finds much more confusing such as 9's and 4's or 5's and 8's

Problem 5
FDA + Local kmeans



Here is a plot comparing FDA + local kmeans and several other methods.

As you can see it does a poor job of classifying the test dataset miss-classifying approximately 600 more digits at every value of k compared to PCA 50 + knn.

In addition it took about 20 minutes to run while PCA 50 + knn only took 4 minutes.

Problem 1

```
from PCA import center_matrix_SVD,mnist  # Imports some functions from our PCA file
import numpy as np  # Numpy very important
from NearestNeighbors import mfoldX  # Get our NN functions
import pickle  # store output and get input
import pylab as plt

def main(): # Our main function
    digits = mnist() # Creates a class with our mnist images and labels
    if open('Training SVD Data','rb')._checkReadable() == 0:
    # Check if file exist create it if it doesn't
        x = center_matrix_SVD(digits.train_Images)
        # Creates a class with our svd and associated info
        pickle.dump(x,open('Training SVD Data','wb')) #put it into a file
    else:  # if the file already exist load it
        x = pickle.load(open('Training SVD Data','rb'))
    if 0: # change 0 to 1 if you want to run this agian
        merror = mfoldX(x.PCA[:,:],digits.train_Labels,6,10)
        # Run X-validation and return error rates for the full dataset
        pickle.dump(merror,open('MFoldErrors','wb')) # Put our error rates in a file
        merror = mfoldX(x.PCA[:,:154],digits.train_Labels,6,10) # For the 95% dataset
        pickle.dump(merror,open('MFoldErrors154','wb'))
        merror = mfoldX(x.PCA[:,:50],digits.train_Labels,6,10) # for the 82.5% dataset
        pickle.dump(merror,open('MFoldErrors50','wb'))
    MFold_plots(x) # Makes graphs from our data

def MFold_plots(x):
    # A function for plotting output from our m fold data
    # This is just for plotting stuff
    err4_var = round(np.sum(np.power(x.s[:25],2))/np.sum(np.power(x.s,2))*100,1)
    # Calulate the reduced variance
    err4_lab = "%s%%" % str(err4_var)
    err5_var = round(np.sum(np.power(x.s[:40],2))/np.sum(np.power(x.s,2))*100,1)
    err5_lab = "%s%%" % str(err5_var)
    err6_var = round(np.sum(np.power(x.s[:45],2))/np.sum(np.power(x.s,2))*100,1)
    err6_lab = "%s%%" % str(err6_var)
    err = pickle.load(open('MFolderrors50', 'rb'))  # Loads the data we saved eariler
    err2 = pickle.load(open('MFolderrors', 'rb'))  # I Know this numbering doesn't really mak
    err3 = pickle.load(open('MFolderrors154', 'rb'))
    err4 = pickle.load(open('MFolderrors25', 'rb'))
    err5 = pickle.load(open('MFolderrors40', 'rb'))
    err6 = pickle.load(open('MFolderrors45', 'rb'))
    plt.figure()
    plt.plot(np.arange(10)+1, err, color='Green', marker='o', markersize=10, label='82.5%')
#plots the 82.5%
    plt.plot(np.arange(10)+1, err3, color='Yellow', marker='o', markersize=10, label='95%')
    plt.plot(np.arange(10)+1, err2, color='Red', marker='o', markersize=10, label='Full')
    plt.plot(np.arange(10)+1, err4, color='Blue', marker='o', markersize=10, label=err4_lab)
    plt.plot(np.arange(10)+1, err5, color='Purple', marker='o', markersize=10, label=err5_lab
    plt.plot(np.arange(10)+1, err6, color='Black', marker='o', markersize=10, label=err6_lab)
    plt.grid(1) # Turns the grid on
    plt.title('Scatter plot of error rates')
    plt.legend(loc='upper right') # Puts a legend on the plot
    plt.show()

if __name__ == "__main__":
    main()
```

Problem 2

```
from PCA import center_matrix_SVD,mnist,class_error_rate  # Imports some functions from our P
import pylab as plt
from NearestNeighbors import KNN  # Get our NN functions
import numpy as np
import pickle

def main():
    digits = mnist() # Creates a class with our mnist images and labels
    if open('Training SVD Data','rb')._checkReadable() == 0:
        x = center_matrix_SVD(digits.train_Images)
        pickle.dump(x,open('Training SVD Data','wb'))
    else:
        x = pickle.load(open('Training SVD Data','rb'))
    if 0:  # change to 1 if you want to rerun the knn stuff
        do_KNN(x,digits)
    KNN_Plots(x,digits)

def do_KNN(x,digits):
    #code to run knn for the three values of s
    test_Images_Center = \
        np.subtract(digits.test_Images,np.repeat(x.centers,digits.test_Images.shape[0],0))
    Knn_labels, nearest = \
        KNN(x.PCA,digits.train_Labels,test_Images_Center@np.transpose(x.V[:,:]),10)
    pickle.dump(Knn_labels,open('Knn_Full','wb'))
    Knn_labels, nearest = KNN(x.PCA[:,:154],digits.train_Labels,
        test_Images_Center@np.transpose(x.V[:154,:]),10)
    pickle.dump(Knn_labels,open('Knn_154','wb'))
    Knn_labels, nearest = KNN(x.PCA[:,:50],digits.train_Labels,
        test_Images_Center@np.transpose(x.V[:50,:]),10)
    pickle.dump(Knn_labels,open('Knn_50','wb'))
    pickle.dump(nearest,open('Knn_50_nearest','wb'))

def KNN_Plots(x,digits):
    # KNN plots
    labels_50 = pickle.load(open('KNN_50','rb'))
    labels_154 = pickle.load(open('KNN_154','rb'))
    labels_Full = pickle.load(open('KNN_Full','rb'))
    nearest_50 = pickle.load(open('Knn_50_nearest','rb'))
    nearest_154 = pickle.load(open('Knn_154_nearest','rb'))
    nearest_Full = pickle.load(open('Knn_Full_nearest','rb'))
    error_50, error_50_index = class_error_rate(labels_50,digits.test_labels)
    error_154, error_154_index = class_error_rate(labels_154,digits.test_labels)
    error_Full, error_Full_index = class_error_rate(labels_Full,digits.test_labels)
    print(error_50)
    print(error_154)
    print(error_Full)
    plt.figure()
    plt.bar([0,1,2],[error_50[2],error_154[2],error_Full[2]])
    plt.grid(1)
    plt.title('Bar Plot of Error Rates')
    plt.legend(loc='upper right')
    plt.show()
    error_50_index = np.asarray(np.where(error_50_index[2]))
    error_154_index = np.asarray(np.where(error_154_index.astype(int)[2]))
    error_Full_index = np.asarray(np.where(error_Full_index.astype(int)[2]))
    error_in_50_Full = error_Full_index[0,inboth_index(error_50_index[0],error_Full_index[0])
    # This is a loop that looks through digits the 50 dim PCA got correct but the full didn't
    for i in range(error_in_50_Full.shape[0]):
        j = error_in_50_Full[i]
        test_Images_Center = np.subtract(digits.test_Images,np.repeat(x.centers,digits.test_I
        y = test_Images_Center@np.transpose(x.V[:50,:])
```

```
        weighted_y = y[:,:50]@x.V[:50,:] + x.centers
        plt.subplot(2, 3, 1)
        plt.imshow(weighted_y[j].reshape(28,28),cmap='gray',interpolation = 'none')
        plt.axis('off')
        plt.title("In 50 %d Truth %d " % (np.asscalar(labels_50[2,j]), np.asscalar(digits.tes
        y = test_Images_Center@np.transpose(x.V[:154,:])
        weighted_y2 = y[:,:154]@x.V[:154,:] + x.centers
        plt.subplot(2, 3, 2)
        plt.imshow(weighted_y2[j].reshape(28,28),cmap='gray',interpolation = 'none')
        plt.axis('off')
        plt.title("in 150 %d Truth %d " % (np.asscalar(labels_154[2,j]), np.asscalar(digits.t
        plt.subplot(2, 3, 3)
        plt.imshow(digits.test_Images[j].reshape(28,28),cmap='gray')
        plt.axis('off')
        plt.title("in Full %d Truth %d " % (np.asscalar(labels_Full[2,j]), np.asscalar(digits
        plt.subplot(2, 3, 4)
        weighted_x = x.PCA[:,:50]@x.V[:50,:] + x.centers
        myimage = np.hstack((weighted_x[nearest_50[j,0]].reshape(28,28),weighted_x[nearest_50
        plt.imshow(myimage,cmap='gray')
        plt.title(np.array_str(digits.train_Labels[nearest_50[j,:3].astype(int)]))
        plt.axis('off')
        plt.subplot(2, 3, 5)
        weighted_x = x.PCA[:,:154]@x.V[:154,:] + x.centers
        myimage = np.hstack((weighted_x[nearest_154[j,0]].reshape(28,28),weighted_x[nearest_1
        plt.imshow(myimage,cmap='gray')
        plt.title(np.array_str(digits.train_Labels[nearest_154[j,:3].astype(int)]))
        plt.axis('off')
        plt.subplot(2, 3, 6)
        weighted_x = x.a_centered + x.centers
        myimage = np.hstack((weighted_x[nearest_154[j,0]].reshape(28,28),weighted_x[nearest_1
        plt.imshow(myimage,cmap='gray')
        plt.title(np.array_str(digits.train_Labels[nearest_Full[j,:3].astype(int)]))
        print(np.array_str(nearest_Full[j,:3].astype(int)))
        print(np.array_str(nearest_154[j,:3].astype(int)))
        print(np.array_str(nearest_50[j,:3].astype(int)))
        plt.axis('off')
        plt.show()


if __name__ == "__main__":
    main()
```

Problem 3

```
from PCA import center_matrix_SVD,mnist,class_error_rate  # Imports some functions from our P
import numpy as np
from NearestNeighbors import local_kmeans_class  # Get our local kmeans function
import pickle
from TicToc import tic,toc
import pylab as plt

def main():
    digits = mnist() # Creates a class with our mnist images and labels
    if open('Training SVD Data','rb')._checkReadable() == 0: # Check if file exist create it
        print("im here")
        x = center_matrix_SVD(digits.train_Images) # Creates a class with our svd and associa
        pickle.dump(x,open('Training SVD Data','wb'))
    else:
        x = pickle.load(open('Training SVD Data','rb'))  # If we already have the file just l
    if 0:
        test_Images_Center = np.subtract(digits.test_Images,np.repeat(x.centers,digits.test_I
        tic()
        labels = local_kmeans_class(x.PCA[:,:50],digits.train_Labels,
                test_Images_Center@np.transpose(x.V[:50,:]),10)
        toc()
        pickle.dump(labels,open('Loc_kmeans_50_lab','wb'))
    loc_full = pickle.load(open('Loc_kmeans_Full_lab','rb'))
    loc_50 = pickle.load(open('Loc_kmeans_50_lab','rb'))
    labels_Full = pickle.load(open('KNN_Full','rb'))
    # Have to transpose these because they came out backwards should fix if i use this agian
    errors_full,ind_full = class_error_rate(np.transpose(loc_full),digits.test_labels)
    errors_50,ind_50 = class_error_rate(np.transpose(loc_50),digits.test_labels)
    errors_near,ind_50 = class_error_rate(labels_Full,digits.test_labels)
    plt.figure()
    plt.plot(np.arange(10)+1, errors_full, color='Green', marker='o',
         markersize=10, label='Full')  #plots the 82.5%
    plt.plot(np.arange(10)+1, errors_50, color='Yellow', marker='o', markersize=10, label='82
    plt.plot(np.arange(10)+1, errors_near, color='Blue', marker='o', markersize=10, label='kN
    plt.grid(1) # Turns the grid on
    plt.title('Plot of local KNN Error rates')
    plt.legend(loc='upper right') # Puts a legend on the plot
    plt.show()

    #Plot incorrectly labeled images
    test_Images_Center = np.subtract(digits.test_Images,
        np.repeat(x.centers,digits.test_Images.shape[0],0))
    y = test_Images_Center@np.transpose(x.V[:50,:])
    weighted_y = y[:,:50]@x.V[:50,:] + x.centers
    loc_50_near = pickle.load(open('loc_kmeans_50_near','rb'))
    error_50_index = np.nonzero(ind_50[8].astype(int))[0]
    for i in range(error_50_index.shape[0]):
        j = error_50_index[i]
        plt.subplot(1,3,1)
        plt.imshow(weighted_y[j].reshape(28,28),cmap='gray',interpolation = 'none')
        plt.axis('off')
        plt.title("Guess %d Truth %d " % (np.asscalar(loc_50[j,8]), np.asscalar(digits.test_l
        plt.subplot(1,3,2)
        myimage = loc_50_near[j,np.asscalar(loc_50[j,8]),8,:]@x.V[:50,:] + x.centers
        plt.imshow(myimage.reshape(28,28),cmap='gray',interpolation = 'none')
        plt.axis('off')
        plt.title("Nearest Center %d" % (np.asscalar(loc_50[j,8])))
        plt.subplot(1,3,3)
        myimage2 = loc_50_near[j,np.asscalar(digits.test_labels[j]),8,:]@x.V[:50,:] + x.cente
        plt.imshow(myimage2.reshape(28,28),cmap='gray',interpolation = 'none')
        plt.axis('off')
```

```
        plt.title("True Center %d " % (np.asscalar(digits.test_labels[j])))
        plt.show()

if __name__ == "__main__":
    main()
```

Problem 4

```
from PCA import center_matrix_SVD,mnist,class_error_rate  # Imports some functions from our P
import numpy as np
import pickle
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import confusion_matrix
from NearestNeighbors import KNN  # Get our NN functions
from TicToc import tic,toc
import pylab as plt


def main():
    digits = mnist() # Creates a class with our mnist images and labels
    if open('Training SVD Data','rb')._checkReadable() == 0: # Check if file exist create it
        print("im here")   # Just wanted to check if it was going in here
        x = center_matrix_SVD(digits.train_Images) # Creates a class with our svd and associa
        pickle.dump(x,open('Training SVD Data','wb'))
    else:
        x = pickle.load(open('Training SVD Data','rb'))  # If we already have the file just l
    if 0: # if this is zero skip
        test_Images_Center = np.subtract(digits.test_Images,np.repeat(x.centers,digits.test_I
        tic()
        myLDA = LDA()  # Create a new instance of the LDA class
        new_train = myLDA.fit_transform(x.PCA[:,:154],digits.train_Labels)
        # It will fit based on x.PCA
        new_test = myLDA.transform(test_Images_Center@np.transpose(x.V[:154,:]))
        # get my transformed test dataset
        Knn_labels, nearest = KNN(new_train,digits.train_Labels,new_test,10) # Run kNN on the
        toc()
        pickle.dump(Knn_labels,open('FDAKNN_Lables','wb'))
        pickle.dump(nearest,open('FDAKNN_neastest','wb'))
    fda = pickle.load(open('FDAKNN_Lables','rb'))
    labels_Full = pickle.load(open('KNN_Full','rb'))
    labels_50 = pickle.load(open('KNN_50','rb'))
    errors_fda,ind_fda = class_error_rate(fda,digits.test_labels)
    errors_near,ind_near = class_error_rate(labels_Full,digits.test_labels)
    errors_50,ind_50 = class_error_rate(labels_50,digits.test_labels)
    plt.figure()
    plt.plot(np.arange(10)+1, errors_fda, color='Green', marker='o', markersize=10, label='fd
    #plots the 82.5%
    plt.plot(np.arange(10)+1, errors_near, color='Blue', marker='o', markersize=10, label='kN
    plt.plot(np.arange(10)+1, errors_50, color='Yellow', marker='o', markersize=10, label='kN
    plt.grid(1) # Turns the grid on
    plt.title('Plot of Knn with FDA Error rates')
    plt.legend(loc='upper right')  # Puts a legend on the plot
    plt.show()
    print(confusion_matrix(digits.test_labels,labels_Full[5]))
    print(confusion_matrix(digits.test_labels,fda[5]))
    print(confusion_matrix(digits.test_labels,labels_50[5]))
```

Problem 5

```
from PCA import center_matrix_SVD,mnist,class_error_rate  # Imports some functions from our P
import numpy as np
import pickle
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from NearestNeighbors import local_kmeans_class  # Get our NN functions
from TicToc import tic,toc
import pylab as plt


def main():
    digits = mnist() # Creates a class with our mnist images and labels
    if open('Training SVD Data','rb')._checkReadable() == 0: # Check if file exist create it
        x = center_matrix_SVD(digits.train_Images) # Creates a class with our svd and associa
        pickle.dump(x,open('Training SVD Data','wb'))
    else:
        x = pickle.load(open('Training SVD Data','rb'))  # If we already have the file just l
    if 1: # if this is zero skip
        test_Images_Center = np.subtract(digits.test_Images,np.repeat(x.centers,digits.test_I
        tic()
        myLDA = LDA()  # Create a new instance of the LDA class
        new_train = myLDA.fit_transform(x.PCA[:,:154],digits.train_Labels)  # It will fit bas
        new_test = myLDA.transform(test_Images_Center@np.transpose(x.V[:154,:])) # get my tra
        Knn_labels = local_kmeans_class(new_train,digits.train_Labels,new_test,10) # Run kNN
        toc()
        pickle.dump(Knn_labels,open('Loc_kmeans_fda_lab','wb'))

    fda = pickle.load(open('Loc_kmeans_fda_lab','rb'))
    labels_Full = pickle.load(open('KNN_Full','rb'))
    loc_full = pickle.load(open('Loc_kmeans_Full_lab','rb'))
    errors_fda,ind_fda = class_error_rate(np.transpose(fda),digits.test_labels)
    errors_near,ind_near = class_error_rate(labels_Full,digits.test_labels)
    errors_full,ind_full = class_error_rate(np.transpose(loc_full),digits.test_labels)
    labels_50 = pickle.load(open('KNN_50','rb'))
    errors_50,ind_50 = class_error_rate(labels_50,digits.test_labels)
    print(errors_full)
    plt.figure()
    plt.plot(np.arange(10)+1, errors_fda, color='Green', marker='o', markersize=10, label='fd
#plots the 82.5%
    plt.plot(np.arange(10)+1, errors_near, color='Blue', marker='o', markersize=10, label='kN
    plt.plot(np.arange(10)+1, errors_full, color='Yellow', marker='o', markersize=10, label='
    plt.plot(np.arange(10)+1, errors_50, color='Red', marker='o', markersize=10, label='kNN 5
    axes = plt.gca()
    axes.set_ylim([0.015,0.12])
    plt.grid(1) # Turns the grid on
    plt.title('Plot of Local Kmeans with FDA Error rates')
    plt.legend(loc='upper right')  # Puts a legend on the plot
    plt.show()
    project_back(x,digits)


def project_back(x,digits):
    myLDA = LDA()
    new_train = myLDA.fit_transform(x.PCA[:,:154],digits.train_Labels)
    print(new_train.shape)
    m = 0
    n = 1
    plt.figure()
    plt.scatter(new_train[digits.train_Labels == 0,m],new_train[digits.train_Labels == 0,n],
    plt.scatter(new_train[digits.train_Labels == 1,m],new_train[digits.train_Labels == 1,n],
    plt.scatter(new_train[digits.train_Labels == 2,m],new_train[digits.train_Labels == 2,n],
    plt.scatter(new_train[digits.train_Labels == 3,m],new_train[digits.train_Labels == 3,n],
    plt.scatter(new_train[digits.train_Labels == 4,m],new_train[digits.train_Labels == 4,n],
    plt.scatter(new_train[digits.train_Labels == 5,m],new_train[digits.train_Labels == 5,n],
    plt.scatter(new_train[digits.train_Labels == 6,m],new_train[digits.train_Labels == 6,n],
    plt.scatter(new_train[digits.train_Labels == 7,m],new_train[digits.train_Labels == 7,n],
    plt.show()
```

```
    y = new_train@myLDA.coef_[:9,:] # I really don't know if this will work since there are 1
    weighted_y2 = y[:,:154]@x.V[:154,:] + x.centers
    plt.imshow(weighted_y2[0,:].reshape(28,28))
    plt.show()
if __name__ == "__main__":
    main()
```

NearestNeighbors

```
import numpy as np
from TicToc import tic
from TicToc import toc

def KNN(I, L, x, k,weights = 1):
    """
    I is the matrix of obs
    L are the labels
    x is what we are trying to classify
    k are how many neighbors to look at or whatever
    first we want to create a matrix of distances from each object
    we want to classify to every object in our training set
    """
    from scipy import stats
    from scipy.spatial.distance import cdist
    sizex = len(np.atleast_2d(x))
    label = np.zeros((k,sizex))
    nearest = np.zeros((sizex,k+1))
    for rowsx in range(0, sizex):
        dists = cdist(I, np.atleast_2d(x[rowsx]), metric='euclidean')
        # Now we should have all our distances in our dist array
        # Next find the k closest neighbors of x
        k_smallest = np.argpartition(dists,tuple(range(1,k+1)),axis=None)
        nearest[rowsx] = k_smallest[:k+1]
        # The next step is to use this info to classify each unknown obj
        # if we don't want to use weights weights should equal 1
        if weights == 1:
            for i in range(0,k):
                label[i,rowsx] = stats.mode(L[k_smallest[:i+1]])[0]
        else:
            labs = np.unique(L)
            for i in range(k):
                lab_weighted = np.zeros(np.unique(L).shape[0])
                d = dists[k_smallest[:i+2]][:,0]
                weights = weight_function(d)
                for p in range(0,labs.shape[0]):
                    indices = inboth(np.arange(0,L.shape[0])[L == labs[p]],k_smallest[:i+2])
                    lab_weighted[p]= np.sum(np.multiply(weights,indices))
                label[i,rowsx] = labs[np.argmax(lab_weighted)]
        tic()
        if rowsx % 1000 == 1:
            print(rowsx)
            toc()
    return label, nearest

def weight_function(d):
    #takes a distance vector d and computes the associated linear weights
    weights = np.add(np.divide(d, np.subtract(np.min(d),np.max(d))),1-np.min(d)/np.subtract(n
    return weights

def inboth(list1,list2):
    # returns a list of 1's and 0's the same length as list2 where 1's mean that index is als
    index = np.zeros(list2.shape)
    for i in range(list2.shape[0]):
```

```
        if list2[i] in list1:
            index[i] = 1
    return index

def class_error_rate(pred_labels,true_labels):
    # for calculating the error rate
    error = np.zeros(pred_labels.shape[0])
    for i in range(pred_labels.shape[0]):
        error[i] = sum(pred_labels[i] != true_labels)/pred_labels.shape[1]
    return error

def mfoldX(I, L, m, maxk):
    # I is the trainset
    # L is the Training Labels
    # m is the number of folds
    # maxk is the largest value of k we wish to test
    # first thing to acomplish is to randomly divide the data into m parts
    indices = np.random.permutation(I.shape[0]) # Creates a randomized index vector
    jump = round(len(L) / m) # Calculates the number of rows to jump for each fold
    # The following code cuts up our indices vector into m parts
    # I intended it to handle cases were m % I != 0 but it doesn't
    #so rows(I) needs to be divisible by m
    I_index = indices[:jump]
    L_index = indices[:jump]
    for n in range(1, m - 1): # Iterats through the folds
        # stacks fold into a third diminsion
        I_index = np.dstack((I_index, indices[n * jump:(n + 1) * jump])) # a random index for
        L_index= np.dstack((L_index, indices[n * jump:(n + 1) * jump])) # a random index for
    I_index = np.dstack((I_index, indices[(m-1) * jump:]))
    L_index = np.dstack((L_index, indices[(m-1) * jump:]))
    # Yea I'm pretty sure that wasn't necessary. I could have just used jump and the indices
    # but I'm not changing it now
    #
    # now data should be all nice and divided up we need to do something else
    error = np.zeros(maxk) # Creates a array to store our error rates
    for n in range(0, m): # Loop through each fold
        mask = np.ones(m,dtype=bool)
        mask[n]=0
        notn = np.arange(0,m)[mask] # Creates a series of number except for the m we are curr
        # Creates a Ipt variable that has all
        Ipt = I[I_index[:,:,notn].reshape(((m-1)*I_index.shape[1]))]
        Lpt = L[I_index[:,:,notn].reshape(((m-1)*I_index.shape[1]))]
        label,near = KNN(Ipt,Lpt ,I[I_index[:,:,n].reshape(I_index.shape[1])],10)
        for k in range(10):
            error[k] = error[k] + sum((label[k] != L[L_index[:,:,n]])[0])
    error = error / (len(L))
    return error

def local_kmeans_class(I, L, x, k):
    # A local kmeans function
    # takes training set I and training labels L
    # and uses them to classify x for 1:k nearest neighbors
    # Returns the predicted labels for x
    from scipy.spatial.distance import cdist
    sizex = len(np.atleast_2d(x)) # the number of obs in I
    columns = I.shape[1] # Number of factors in I
    label = np.zeros((sizex,k)) # place to put our labels
    for rowsx in range(0, sizex): # loop through every row of I
        dists = cdist(I, np.atleast_2d(x[rowsx]), metric='euclidean') # gets distances
        center = np.zeros((10,k,columns)) # place to put the centeres for each label
        label_order = np.unique(L)
        l=0 # this should be in the for loop instead of labs
        thing = np.zeros((k,columns)) # place to store the total sums
```

```
        for labs in np.unique(L): # luckly L are integers else we'd have a problem
            indices = L == labs # finds the indices in L that are labs
            k_smallest = np.argpartition(dists[indices],tuple(range(1,k)),axis=None)
            # sorts the thing
            for i in range(0,k):
                M = I[indices] #matrix with only labs probably wasting memory doing this
                #center[l,i,:] = np.average(M[k_smallest[:i+1]],axis = 0)
                if i == 0:
                    thing[i] = M[k_smallest[i+1]]
                else:
                    thing[i] = thing[i-1] + M[k_smallest[i+1]]
            center[l,:,:] = np.divide(thing,np.repeat(np.arange(1,11).reshape(10,1)
                ,columns ,axis=1))
            #that was suppose to be a faster way to compute the average but it isn't
            #but now we have the local averages for every lable and k
            l+=1 # Really shouldn't be here
        for i in range(k): # now we need to find the closed center basically knn again
            #print(cdist(center[:,i,:], np.atleast_2d(x[rowsx]), metric='euclidean'))
            dists2center = cdist(center[:,i,:], np.atleast_2d(x[rowsx]), metric='euclidean')
            k_smallest = np.argpartition(dists2center,tuple(range(1)),axis=None)
            label[rowsx,i] = label_order[k_smallest[0]]
        if rowsx % 1000 == 1: #keep track of where we are
            print(rowsx)
    return label
```

PCA

```
import numpy as np
import pickle
import pylab as plt
from NearestNeighbors import mfoldX,KNN,local_kmeans_class
from TicToc import tic,toc
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

class center_matrix_SVD:
    # A class to store our information about our centered matrix
    # center_matrix has 7 atributes
    # .size stores the shape of the original matrix
    # .centers stores the center of the dataset
    # a_centered is the centered original matrix
    # .U .s .V are the SVD decomposition of the centered matrix
    def __init__(self,a,dim=0):
        self.size = a.shape # Gets and stores the shape of a not sure it is really necessary
        self.centers = np.mean(a,axis=dim).reshape(1,self.size[1])
        # Reshaped as 1,n instead of ,n because that was causing problems
        self.a_centered = np.subtract(a,np.repeat(self.centers,self.size[dim],dim))
        #Creates an atribute a_centered to store the  centered a matrix
        self.U, self.s, self.V = np.linalg.svd(self.a_centered,full_matrices = False)
         # Runs SVD on our centered a matrix
        self.PCA = self.U@np.diagflat(self.s)
        # stores the U*S matrix in atribute PCA since s is diagnol we can just take rows
        # out of this instead of recalulating PCA for reduced dims

class mnist:
    # Creates a class that stores our mnist data. Hopefully it helps keep things more oragniz
    train_Images = pickle.load(open('mnistTrainI.p', 'rb'))
    test_Images = pickle.load(open('mnistTestI.p', 'rb'))
    train_Labels = pickle.load(open('mnistTrainL.p', 'rb'))
    test_labels = pickle.load(open('mnistTestL.p', 'rb'))

def class_error_rate(pred_labels,true_labels):
    # for calculating the error rate
    # Also returns a index vector with the position of incorrectly labeled images
```

```
    error = np.zeros(pred_labels.shape[0])
    error_index = np.zeros((pred_labels.shape[0],pred_labels.shape[1]))
    for i in range(pred_labels.shape[0]):
        error[i] = sum(pred_labels[i] != true_labels)/pred_labels.shape[1]
        # puts each
        error_index[i] = 1 - np.isclose(pred_labels[i],true_labels)
        #
    return error, error_index

def inboth_index(list1,list2):
    # returns a list of index's in list2 but not in list1
    index = np.zeros(list2.shape)
    for i in range(list2.shape[0]):
        if list2[i] not in list1:
            index[i] = 1
    index = np.nonzero(index.astype(int))[0]
    return index
```