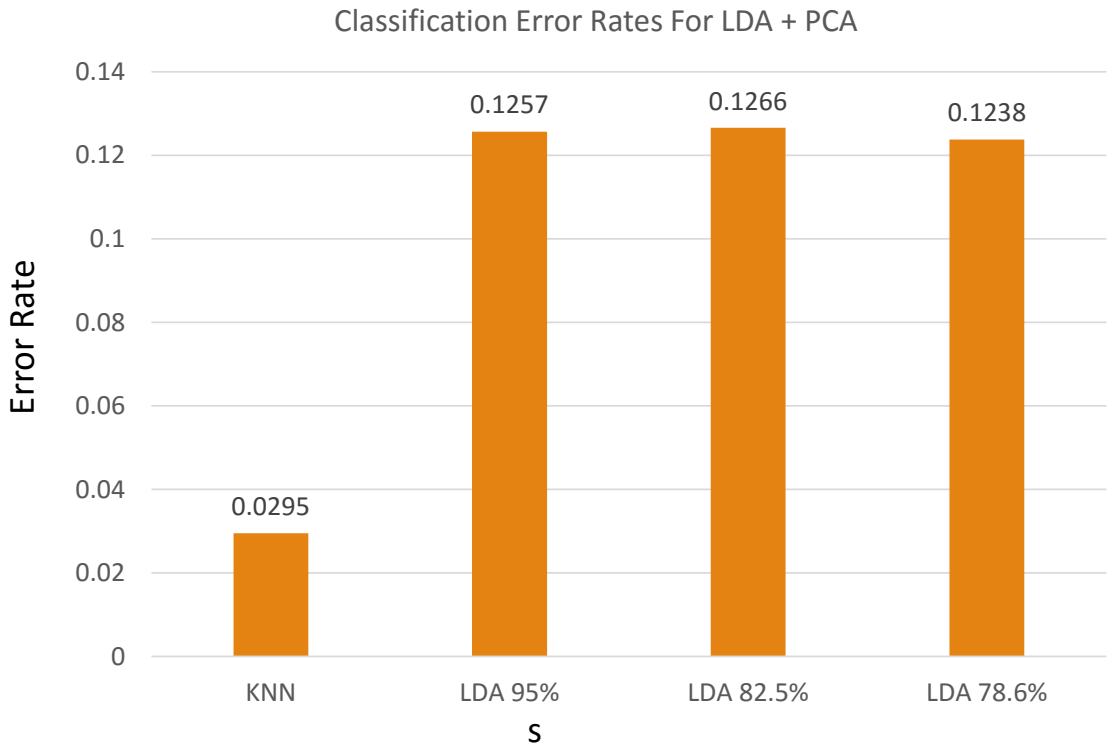
# Homework 3

ANDREW ZASTOVNIK

MATH 285

# Problem 1
## Linear Discriminate Analysis with different values of s

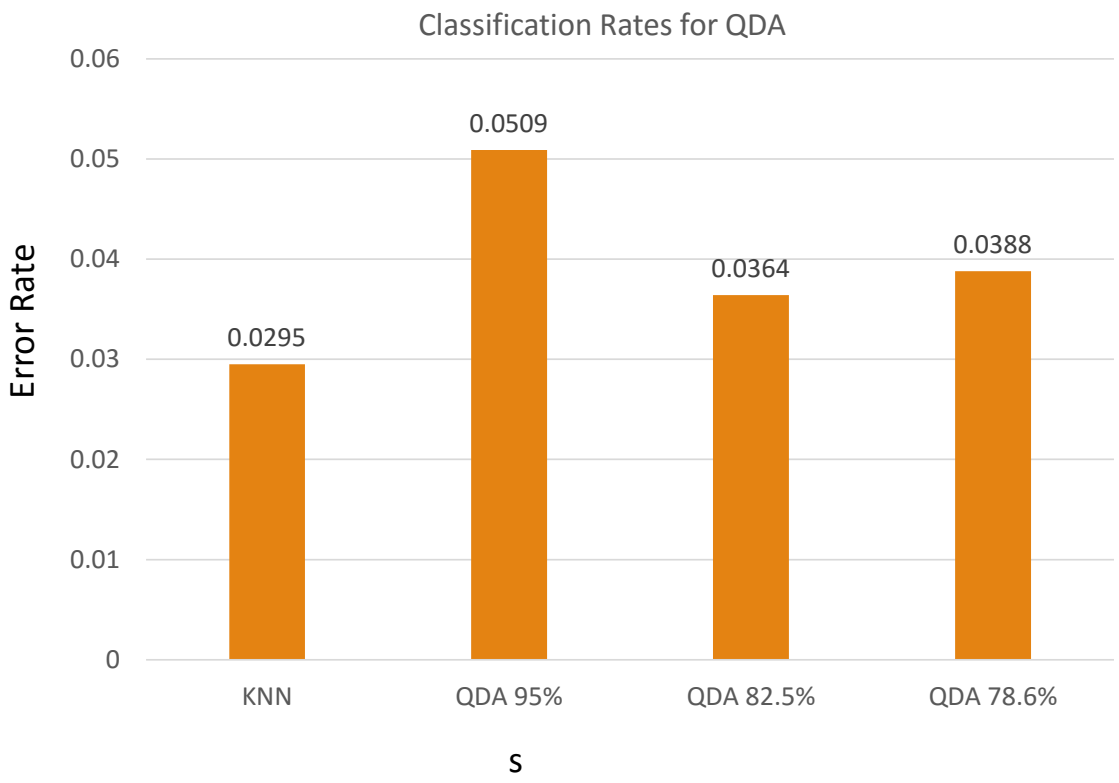



The classification error rate is much higher for all values of s when compared to K nearest neighbors

The best value of s that I tried was s = 45 which corresponds to 78.6% of variance

# Problem 2

## Quadratic Discriminate Analysis with different values of s




Quadratic Discriminate Analysis (QDA) does a much better job classifying the digits than linear discriminate analysis

The lowest error rate was 0.0364 at s = 50 which pretty decent

# Problem 2
## Confusion Matrices

Confusion matrix for KNN

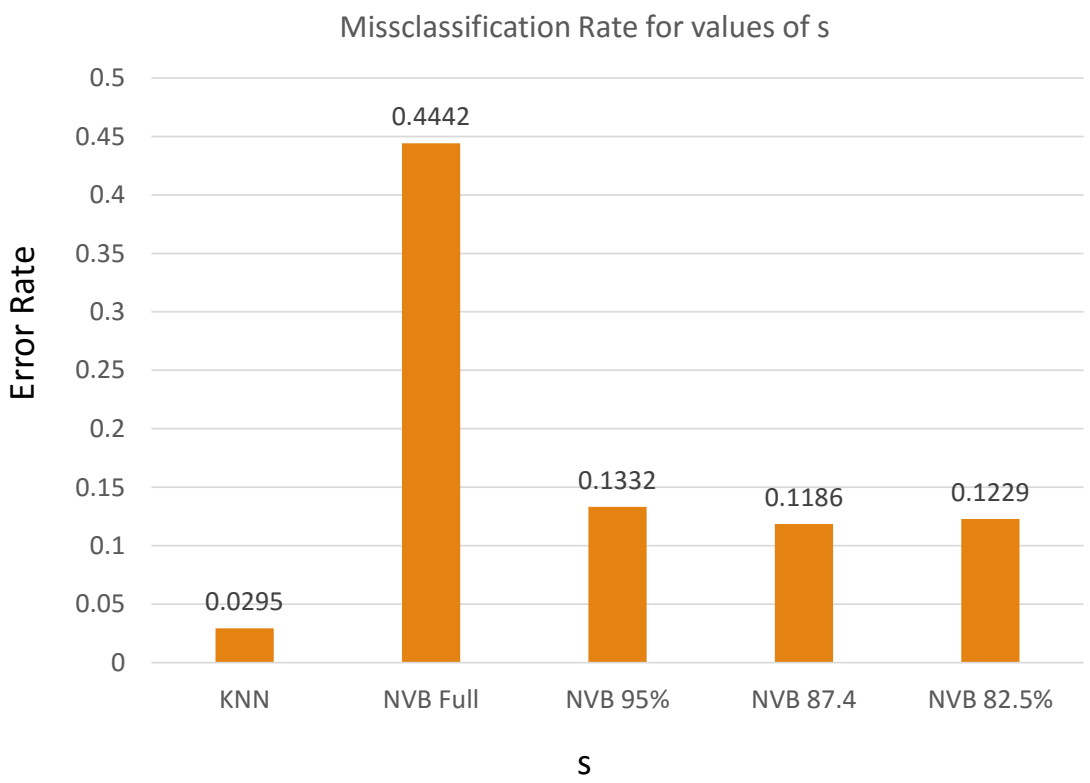
| True Value \ Predicted Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 972 | 1 | 1 | 0 | 0 | 1 | 4 | 1 | 0 | 0 |
| 1 | 0 | 1130 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 2 | 8 | 3 | 1004 | 0 | 1 | 0 | 1 | 12 | 3 | 0 |
| 3 | 0 | 0 | 2 | 977 | 1 | 11 | 0 | 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 QDA with s = 50

| True Value \ Predicted Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 970 | 0 | 1 | 0 | 0 | 2 | 1 | 1 | 5 | 0 |
| 1 | 0 | 1098 | 13 | 1 | 2 | 1 | 1 | 0 | 19 | 0 |
| 2 | 2 | 0 | 1000 | 3 | 3 | 0 | 2 | 2 | 20 | 0 |
| 3 | 2 | 0 | 9 | 970 | 0 | 5 | 0 | 2 | 18 | 4 |
| 4 | 1 | 0 | 4 | 0 | 964 | 0 | 3 | 2 | 2 | 6 |
| 5 | 2 | 0 | 1 | 18 | 0 | 860 | 2 | 0 | 9 | 0 |
| 6 | 8 | 1 | 2 | 0 | 4 | 12 | 924 | 0 | 7 | 0 |
| 7 | 1 | 2 | 28 | 1 | 3 | 2 | 0 | 959 | 14 | 18 |
| 8 | 4 | 0 | 8 | 10 | 1 | 5 | 1 | 2 | 938 | 5 |
| 9 | 5 | 0 | 10 | 6 | 11 | 2 | 0 | 6 | 16 | 953 |

QDA actually did a better job predicting many of the digits compared to knn
Its biggest weakness seems to be over predicting 8's

# Problem 3
## Naïve Bayes with various values of s



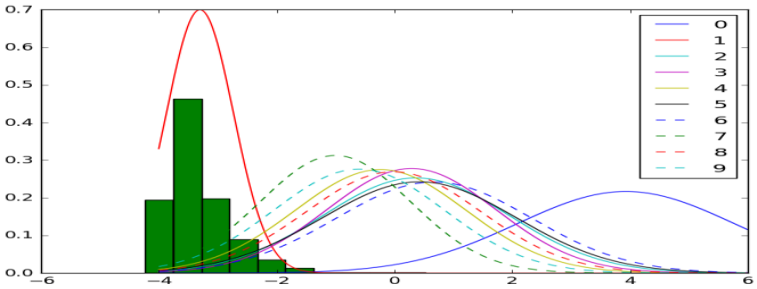

The Naïve Bayes classifier did a really bad job classifying digits in the unreduced dataset misclassifying almost half the digits

Reducing the dimensionality helped greatly but it still did poorly compared to knn

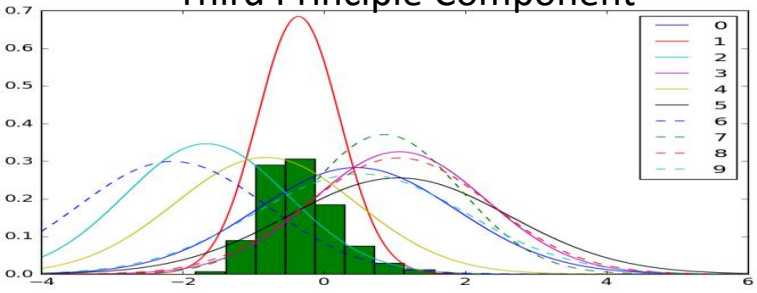
The best value of s I found was s = 70 which gave an error rate of 0.1186

# Problem 3
## A look at the Fitted Distributions

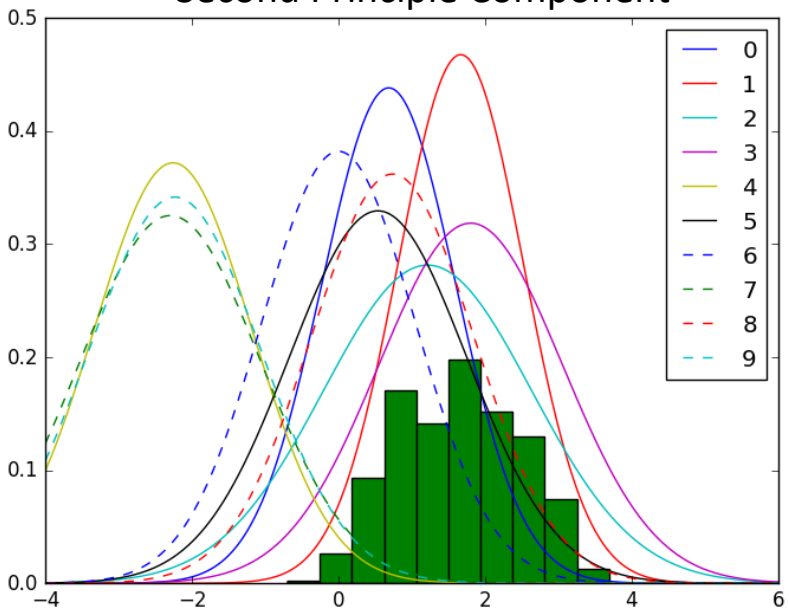
First Principle Component




Third Principle Component

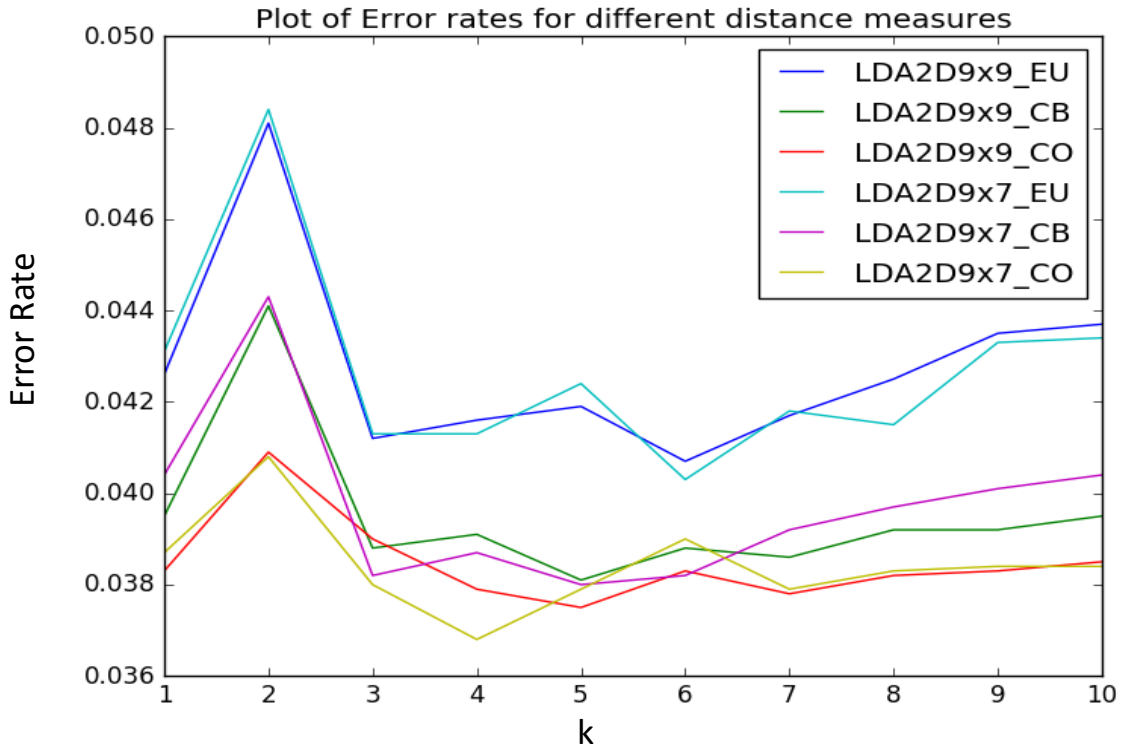



Second Principle Component




These plots show the normal distributions used for classification along with a histogram of the ones In the test data set to compare with.

# Problem 4
## 2DLDA + KNN with Different Distance measures

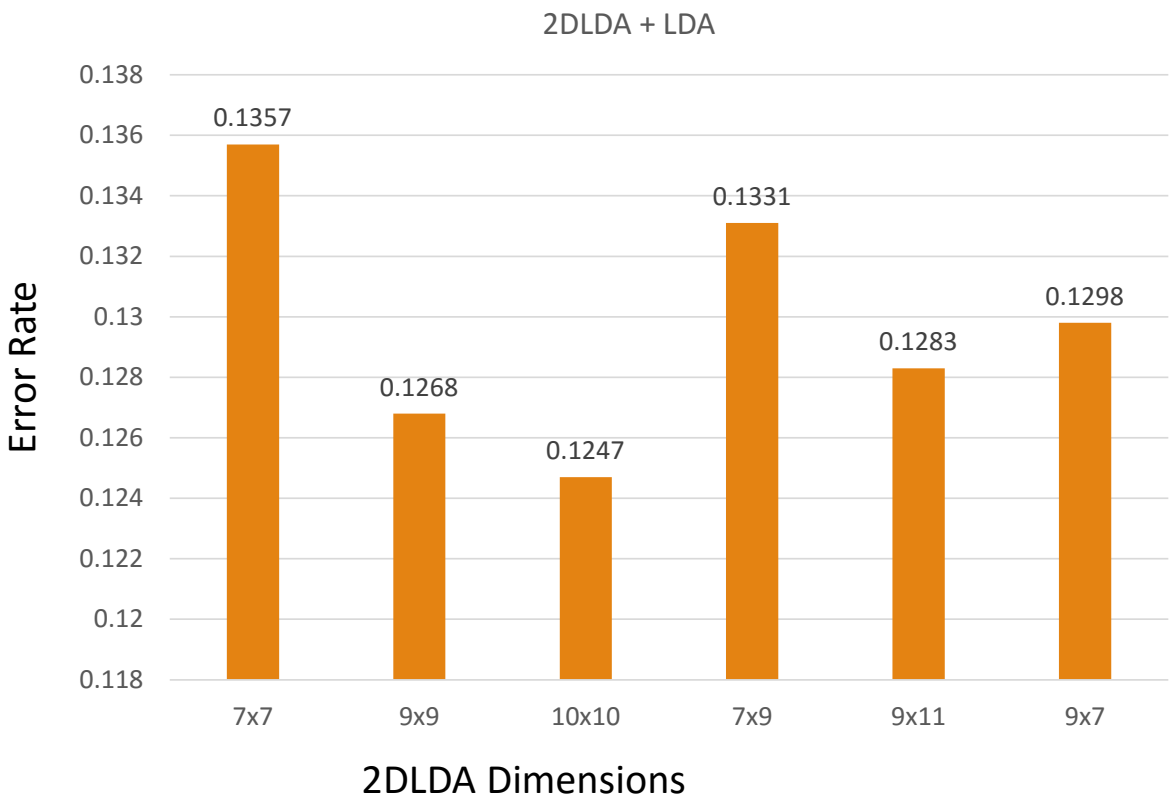



I tried many different dimension with k up to 10

The best that I found was a 9x7 2DLDA using the cosine distance measure at k = 4 with an error rate of .0368

The 9x9 2DLDA did the best with k = 5 with the cosine distances with an error rate of 0.0375

# Problem 5
## 2DLDA + LDA




The best predictions where the 10x10 2DLDA +LDA with an error rate of 0.1247

This is basically the same error rate as the best LDA +PCA which had an error rate of 0.1238

# Problem 1

```
from PCA import mnist, center_matrix_SVD, class_error_rate
from Classifiers import nvb
import numpy as np
import pickle
import pylab as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

def main():
    digits = mnist()
    x = center_matrix_SVD(digits.train_Images)
    errors_154 = doLDA(x,digits,154)
    pickle.dump(errors_154,open('LDA_154.p','wb'))
    errors_50 = doLDA(x,digits,50)
    pickle.dump(errors_50,open('LDA_50.p','wb'))
    errors_10 = doLDA(x,digits,10)
    pickle.dump(errors_10,open('LDA_10.p','wb'))
    errors_60 = doLDA(x,digits,60)
    pickle.dump(errors_60,open('LDA_60.p','wb'))
    prob1_plots(digits)
    put_into_excel(digits)


def doLDA(x,digits,s):
    myLDA = LDA()
    myLDA.fit(x.PCA[:,:s],digits.train_Labels)
    newtest = digits.test_Images -x.centers
    newtest=newtest@np.transpose(x.V[:s,:])
    labels = myLDA.predict(newtest)
    errors = class_error_rate(labels.reshape(1,labels.shape[0]),digits.test_Labels)
    return errors

def prob1_plots(digits):
    labels_Full = pickle.load(open('KNN_Full','rb'))
    error_Full, error_Full_index = class_error_rate(labels_Full,digits.test_Labels)
    error_154,thing = pickle.load(open('LDA_154.p','rb'))
    error_50,thing = pickle.load(open('LDA_50.p','rb'))
    error_60,thing = pickle.load(open('LDA_60.p','rb'))
    plt.figure()
    plt.bar([0,1,2,3],[error_Full[2],error_154,error_50,error_60])
```

```
    plt.title('Bar Plot of Error Rates')
    plt.show()
    """
    errors_154= pickle.load(open('LDA_154.p','rb'))
    labels_Full = pickle.load(open('KNN_Full','rb'))
    df = pandas.DataFrame(errors_154)
    df.to_excel('Errors_154')
    """

def put_into_excel(digits):
    labels_Full = pickle.load(open('KNN_Full','rb'))
    error_Full, error_Full_index = class_error_rate(labels_Full,digits.test_Labels)
    error_154,thing = pickle.load(open('LDA_154.p','rb'))
    error_50,thing = pickle.load(open('LDA_50.p','rb'))
    error_60,thing = pickle.load(open('LDA_60.p','rb'))
    errors = np.hstack((error_Full,error_154,error_50,error_60))
    import pandas
    df = pandas.DataFrame(errors)
    df.to_excel('Errors.xls')
main()
```

## Problem 2

```
from PCA import mnist, center_matrix_SVD, class_error_rate
import numpy as np
import pickle
import pylab as plt
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as LDA

def main():
    digits = mnist()
    x = center_matrix_SVD(digits.train_Images)
    errors_154 = doQDA(x,digits,154)
    pickle.dump(errors_154,open('QDA_154.p','wb'))
    errors_50 = doQDA(x,digits,50)
    pickle.dump(errors_50,open('QDA_50.p','wb'))
    errors_10 = doQDA(x,digits,10)
    pickle.dump(errors_10,open('QDA_10.p','wb'))
    errors_60 = doQDA(x,digits,60)
    pickle.dump(errors_60,open('QDA_60.p','wb'))
    prob1_plots(digits)
    put_into_excel(digits)
    confusion(digits)

def confusion(digits):
    myLDA = LDA()
    x = center_matrix_SVD(digits.train_Images)
    myLDA.fit(x.PCA[:,:50],digits.train_Labels)
    newtest = digits.test_Images -x.centers
    newtest=newtest@np.transpose(x.V[:50,:])
    labels = myLDA.predict(newtest)
    import sklearn.metrics as f
    print(f.confusion_matrix(digits.test_Labels,labels))

def doQDA(x,digits,s):
    myLDA = LDA()
    myLDA.fit(x.PCA[:,:s],digits.train_Labels)
    newtest = digits.test_Images -x.centers
    newtest=newtest@np.transpose(x.V[:s,:])
    labels = myLDA.predict(newtest)
    errors = class_error_rate(labels.reshape(1,labels.shape[0]),digits.test_Labels)
    return errors
```

```
def prob1_plots(digits):
    labels_Full = pickle.load(open('KNN_Full','rb'))
    error_Full, error_Full_index = class_error_rate(labels_Full,digits.test_Labels)
    error_154,thing = pickle.load(open('QDA_154.p','rb'))
    error_50,thing = pickle.load(open('QDA_50.p','rb'))
    error_60,thing = pickle.load(open('QDA_60.p','rb'))
    plt.figure()
    plt.bar([0,1,2,3],[error_Full[2],error_154,error_50,error_60])
    plt.title('Bar Plot of Error Rates')
    plt.show()

def put_into_excel(digits):
    labels_Full = pickle.load(open('KNN_Full','rb'))
    error_Full, error_Full_index = class_error_rate(labels_Full,digits.test_Labels)
    error_154,thing = pickle.load(open('QDA_154.p','rb'))
    error_50,thing = pickle.load(open('QDA_50.p','rb'))
    error_60,thing = pickle.load(open('QDA_60.p','rb'))
    errors = np.hstack((error_Full[2],error_154,error_50,error_60))
    import pandas
    df = pandas.DataFrame(errors)
    df.to_excel('Errors_QDA.xls')
main()
```

## Problem 3

```
from PCA import mnist, center_matrix_SVD,class_error_rate
from Classifiers import nvb
import numpy as np

import pylab as plt

def main():
    digits = mnist()
    mynvb = nvb()
    x = center_matrix_SVD(digits.train_Images)
    mynvb.fit(digits.train_Images,digits.train_Labels)
    labels = mynvb.predict(digits.test_Images)
    errors_Full, error_Full_index = class_error_rate(labels,digits.test_Labels)
    mynvb.fit(x.PCA[:,:154],digits.train_Labels)
    newtest = (digits.test_Images -x.centers)@np.transpose(x.V[:154,:])
    labels = mynvb.predict(newtest)
    errors_154, error_Full_index = class_error_rate(labels,digits.test_Labels)
    mynvb.fit(digits.train_Images,digits.train_Labels)
    mynvb.fit(x.PCA[:,:50],digits.train_Labels)
    newtest = (digits.test_Images -x.centers)@np.transpose(x.V[:50,:])
    labels = mynvb.predict(newtest)
    errors_50, error_Full_index = class_error_rate(labels,digits.test_Labels)
    mynvb.fit(digits.train_Images,digits.train_Labels)
    mynvb.fit(x.PCA[:,:70],digits.train_Labels)
    newtest = (digits.test_Images -x.centers)@np.transpose(x.V[:70,:])
    labels = mynvb.predict(newtest)
    errors_70, error_Full_index = class_error_rate(labels,digits.test_Labels)
    print(errors_Full)
    print(errors_154)
    print(errors_50)
    print(errors_70)
    prob3_plots(mynvb,digits,newtest,pc=0)
    prob3_plots(mynvb,digits,newtest,pc=1)
    prob3_plots(mynvb,digits,newtest,pc=2)
    prob3_plots(mynvb,digits,newtest,pc=3)


def prob3_plots(mynvb,digits,newtest,pc):
    z= (np.arange(1000)/1000)*10 - 4 #
```

```
    y0 = mynvb.likelihood[str(pc)+'l'+str(0)](z)
    y1 = mynvb.likelihood[str(pc)+'l'+str(1)](z)
    y2= mynvb.likelihood[str(pc)+'l'+str(2)](z)
    y3 = mynvb.likelihood[str(pc)+'l'+str(3)](z)
    y4 = mynvb.likelihood[str(pc)+'l'+str(4)](z)
    y5 = mynvb.likelihood[str(pc)+'l'+str(5)](z)
    y6 = mynvb.likelihood[str(pc)+'l'+str(6)](z)
    y7 = mynvb.likelihood[str(pc)+'l'+str(7)](z)
    y8 = mynvb.likelihood[str(pc)+'l'+str(8)](z)
    y9 = mynvb.likelihood[str(pc)+'l'+str(9)](z)
    plt.plot(z,y0, label='0')
    indices = np.nonzero(np.array(digits.test_Labels == 1))
    #plt.plot(newtest[indices,0],np.zeros(len(indices)),marker='o')
    weights = np.ones_like(np.transpose(newtest[indices,pc]))/len(np.transpose(newtest[indices,pc]))
    plt.hist(np.transpose(newtest[indices,pc]),weights=weights)
    plt.plot(z,y1, label='1')
    plt.plot(z,y2, label='2')
    plt.plot(z,y3, label='3')
    plt.plot(z,y4, label='4')
    plt.plot(z,y5, label='5')
    plt.plot(z,y6, label='6',ls='--')
    plt.plot(z,y7, label='7',ls='--')
    plt.plot(z,y8, label='8',ls='--')
    plt.plot(z,y9, label='9',ls='--')
    plt.legend(loc='upper right')
    plt.show()

main()
```

## Problem 4

```
from PCA import mnist, center_matrix_SVD,class_error_rate
import numpy as np
from Classifiers import KNN
import LDA2D
import pickle
import pylab as plt

def main():

    digits = mnist()
    do_LDA2D_KNN(digits,7,7)
    do_LDA2D_KNN(digits,9,9)
    do_LDA2D_KNN(digits,10,10)
    do_LDA2D_KNN(digits,7,9)
    do_LDA2D_KNN(digits,9,11)
    do_LDA2D_KNN(digits,9,7)
    to_plt(digits,7,7)
    plt.legend(loc='upper right')
    plt.title('Plot of Error rates for different distance measures')
    plt.show()
    to_plt(digits,9,9)
    plt.legend(loc='upper right')
    plt.title('Plot of Error rates for different distance measures')
    plt.show()
    to_plt(digits,10,10)
    plt.legend(loc='upper right')
    plt.title('Plot of Error rates for different distance measures')
    plt.show()
    to_plt(digits,7,9)
    plt.legend(loc='upper right')
    plt.title('Plot of Error rates for different distance measures')
    plt.show()
    to_plt(digits,9,11)
    plt.legend(loc='upper right')
    plt.title('Plot of Error rates for different distance measures')
    plt.show()
    to_plt(digits,9,9)
    to_plt(digits,9,7)
    plt.legend(loc='upper right')
```

```
    plt.title('Plot of Error rates for different distance measures')
    plt.show()

def do_LDA2D_KNN(digits,p,q):
    l,r = LDA2D.iterative2DLDA(digits.train_Images, digits.train_Labels, p, q, 28, 28)

    new_train = np.zeros((digits.train_Images.shape[0],p*q))
    for i in range(digits.train_Images.shape[0]):
        new_train[i] = (np.transpose(l)@digits.train_Images[i].reshape(28,28)@r).reshape(p*q)
    new_test = np.zeros((digits.test_Images.shape[0],p*q))
    for i in range(digits.test_Images.shape[0]):
        new_test[i] = (np.transpose(l)@digits.test_Images[i].reshape(28,28)@r).reshape(p*q)

    labels, nearest = KNN(new_train,digits.train_Labels,new_test,10,'euclidean')
    pickle.dump(labels, open('LDA2D'+ str(p) + 'x' + str(q) + '_EU.p','wb'))
    #pickle.dump(nearest, open('NLDA2D'+ str(p) + 'x' + str(q) + '_EU.p','wb'))
    labels, nearest = KNN(new_train,digits.train_Labels,new_test,10,'cityblock')
    pickle.dump(labels, open('LDA2D'+ str(p) + 'x' + str(q) + '_CB.p','wb'))
    #pickle.dump(nearest, open('NLDA2D'+ str(p) + 'x' + str(q) + '_CB.p','wb'))
    labels, nearest = KNN(new_train,digits.train_Labels,new_test,10,'cosine')
    pickle.dump(labels, open('LDA2D'+ str(p) + 'x' + str(q) + '_CO.p','wb'))
    #pickle.dump(nearest, open('NLDA2D'+ str(p) + 'x' + str(q) + '_CO.p','wb'))

def to_plt(digits,p,q):
    thing = pickle.load(open('LDA2D'+ str(p) + 'x' + str(q) + '_EU.p','rb'))
    error, idw = class_error_rate(thing,digits.test_Labels)
    print(error)
    plt.plot(np.arange(10)+1,error,label='LDA2D'+ str(p) + 'x' + str(q) + '_EU')
    thing = pickle.load(open('LDA2D'+ str(p) + 'x' + str(q) + '_CB.p','rb'))
    error, idw = class_error_rate(thing,digits.test_Labels)
    print(error)
    plt.plot(np.arange(10)+1,error,label='LDA2D'+ str(p) + 'x' + str(q) + '_CB')
    thing = pickle.load(open('LDA2D'+ str(p) + 'x' + str(q) + '_Co.p','rb'))
    error, idw = class_error_rate(thing,digits.test_Labels)
    print(error)
    plt.plot(np.arange(10)+1,error,label='LDA2D'+ str(p) + 'x' + str(q) + '_CO')

main()
```

## Problem 5

```
from PCA import mnist, center_matrix_SVD,class_error_rate
import numpy as np
import LDA2D
import pickle
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

def main():
    digits = mnist()
    do_LDA2D_LDA(digits,7,7)
    do_LDA2D_LDA(digits,9,9)
    do_LDA2D_LDA(digits,10,10)
    do_LDA2D_LDA(digits,7,9)
    do_LDA2D_LDA(digits,9,11)
    do_LDA2D_LDA(digits,9,7)
    to_plt(digits,7,7)
    to_plt(digits,9,9)
    to_plt(digits,10,10)
    to_plt(digits,7,9)
    to_plt(digits,9,11)
    to_plt(digits,9,7)
def do_LDA2D_LDA(digits,p,q):
    l,r = LDA2D.iterative2DLDA(digits.train_Images, digits.train_Labels, p, q, 28, 28)

    new_train = np.zeros((digits.train_Images.shape[0],p*q))
    for i in range(digits.train_Images.shape[0]):
        new_train[i] = (np.transpose(l)@digits.train_Images[i].reshape(28,28)@r).reshape(p*q)
    new_test = np.zeros((digits.test_Images.shape[0],p*q))
    for i in range(digits.test_Images.shape[0]):
        new_test[i] = (np.transpose(l)@digits.test_Images[i].reshape(28,28)@r).reshape(p*q)
    myLDA = LDA()
    x = center_matrix_SVD(new_train)
    myLDA.fit(new_train-x.centers,digits.train_Labels)
    labels = myLDA.predict(new_test-x.centers)
    pickle.dump(labels, open('LDA2DLDA'+ str(p) + 'x' + str(q) + '.p','wb'))


def to_plt(digits,p,q):
    thing = pickle.load(open('LDA2DLDA'+ str(p) + 'x' + str(q) + '.p','rb'))
    error, idw = class_error_rate(thing,digits.test_Labels)
```

```
        print(error)

main()
```

## Classifiers

```
import numpy as np
import math

def KNN(I, L, x, k,metric='euclidean',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=metric)
        # 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)]
        if rowsx % 1000 == 1:
```

```
            print(rowsx)
    return label, nearest

class nvb:
    # A Naive Bayes Classifier
    # Has atributes fit distribution and predict
    # fit creates a dictionary of likelihood functions
    def fit(self,train_data,train_labs):
        # Creates the distributions associated with each factor and class
        self.train_labs = train_labs
        self.likelihood = {} #  a dictionary to store our dist functions
        lab = np.unique(train_labs)
        for l in range(lab.shape[0]):
            for factors in range(train_data.shape[1]):
                # Get the indices for a specific class
                indices = np.nonzero(np.array(train_labs == lab[l]))
                # Create likelihood functions
                self.likelihood[str(factors)+'l'+str(lab[l])] = self.distribution(train_data[indices,factors])

    def distribution(self,factor):
        # Creates a distribution give a factor
        mu = np.mean(factor) #  Finds the mean
        s = np.var(factor)  #  Findes the variance
        def normal(x):
            if s == 0: # what do we do if variance is 0?
                # Create a very small box with length 2e-10 around the mean
                if x <=mu +1e-10 and x >= mu - 1e-10:
                    return (0.5e10) # The height of our box +1
                else:
                    return 1e-100  # if it is outside our range likelihood is 0 + 1
                # ok can't add 1 to the likelihood functions
            else:
                # Now our normal distributions
                l = (1/(2*3.14*s)**.5)*2.71828**(-((x-mu)**2)/(2*s))
                if l ==0:
                    # to handel underflow
                    l=1e-100
                return l
        return normal

    def predict(self,test_data):
```

```
        # The thing the gives us predictions give test data
        labels = np.zeros(test_data.shape[0])
        for i in range(test_data.shape[0]):
            pofl = np.zeros(np.unique(self.train_labs).shape[0])
            for l in range(np.unique(self.train_labs).shape[0]):
                for factors in range(test_data.shape[1]):
                    pofl[l] += math.log(self.likelihood[str(factors)+'l'+str(l)](test_data[i,factors]))
            labels[i] = np.unique(self.train_labs)[np.argmax(pofl)]
            print(i)
        return labels

    def count_unique(self):
        # Maybe I should use this to get the prior probabilities but I'm not
        labs = np.unique(self.train_labs)
        labs_count = np.zeros(labs.shape) #this probabily will need to be checked
        for l in self.train_labs.shape[0]:
            labs_count[l] = sum(self.train_labs == labs[l])
        return labs, labs_count

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(np.min(d),np.max(d)))
    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 also in list1
    index = np.zeros(list2.shape)
    for i in range(list2.shape[0]):
        if list2[i] in list1:
            index[i] = 1
    return index
```

# PCA

```
import numpy as np
import pickle

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):
        size = a.shape # Gets and stores the shape of a
        self.centers = np.mean(a,axis=dim).reshape(1,size[1])
        # Reshaped as 1,n instead of ,n because that was causing problems
        a_centered = np.subtract(a,np.repeat(self.centers,size[dim],dim))
        #Creates a_centered to store the  centered a matrix
        self.U, self.s, self.V = np.linalg.svd(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 oragnized
    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 dump_the_svd(digits):
    # Gets SVD for the training dataset
    train = center_matrix_SVD(digits.train_Images)
    pickle.dump(train,open('Training SVD Data','wb')) # dump the result to a file
    # You won't be able to load this unless you have the center_matrix_SVD class avalible
    # Not sure it really saves much time to put this into a file since svd is pretty fast

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
```

```
    if len(pred_labels.shape)> 1:
        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)
            #
    else:
        error = sum(pred_labels != true_labels)/pred_labels.shape[0]
        error_index = 1 - np.isclose(pred_labels,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
```

# LDA2D

```
import numpy as np

def iterative2DLDA(Trainset, LabelTrain, p, q,r, c):
    # This was basically just copied from the function in matlab
    Trainset = np.transpose(Trainset) #  Make it so images are columns
    m = Trainset.shape
    classnumber = max(LabelTrain)
    aa = {}
    for i in range(classnumber):
        temp= np.arange(LabelTrain.size).reshape(LabelTrain.shape)[np.where(LabelTrain==i)]
        temp1=temp
        m1 = temp1.shape
        Trainset1 = Trainset[:,temp1]
        aa[i] = np.mean(Trainset1,axis=1)
    bb = np.mean(Trainset,axis=1)
    bb1 = np.transpose(bb)
    R = np.vstack((np.eye(q,q),np.zeros((c-q,q))))
    for j in range(10):
        sb1=np.zeros((r,r))
        sw1=np.zeros((r,r))
        for i in range(classnumber):
            temp= np.arange(LabelTrain.size).reshape(LabelTrain.shape)[np.where(LabelTrain==i)]
            temp1=temp
            m1 = temp1.shape
            Trainset1=Trainset[:,temp1]
            m2 = Trainset1.shape
            for s in range(m2[1]):
                sw1=sw1+(Trainset1[:,s].reshape(r,c) - aa[i].reshape(r,c))\
                        @R@np.transpose(R)@\
                        np.transpose((Trainset1[:,s].reshape(r,c) - aa[i].reshape(r,c)))
            sb1=sb1+m1[0]*(aa[i].reshape(r,c) - bb1.reshape(r,c))\
                    @R@np.transpose(R)@np.transpose((aa[i].reshape(r,c)-bb1.reshape(r,c)))
        s,u = np.linalg.eig(np.linalg.pinv(sw1)@sb1)
        tt = s
        ind = tt.argsort()[::-1]
        u11 = u[:,ind]
        L = u11[:,:p]
        #Seriously?
        sb2 = np.zeros((c,c))
```

```
        sw2 = np.zeros((c,c))
        for i in range(classnumber):
            temp= np.arange(LabelTrain.size).reshape(LabelTrain.shape)[np.where(LabelTrain==i)]
            temp1=temp
            m1 = temp1.shape
            Trainset1 = Trainset[:,temp1]
            m2 = Trainset1.shape
            for s in range(m2[1]):
                sw2=sw2+np.transpose((Trainset1[:,s].reshape(r,c) - aa[i].reshape(r,c)))\
                        @L@np.transpose(L)@\
                        (Trainset1[:,s].reshape(r,c) - aa[i].reshape(r,c))
            sb2=sb2+m1[0]*np.transpose((aa[i].reshape(r,c) - bb1.reshape(r,c)))\
                                                @L@np.transpose(L)@(aa[i].reshape(r,c)-bb1.reshape(r,c))
        s1,u1 = np.linalg.eig(np.linalg.pinv(sw2)@sb2)
        tt1 = s1
        ind1 = tt1.argsort()[::-1]
        u12 = u1[:,ind1]
        R = u12[:,:q]
        print(j)
    return L,R
```