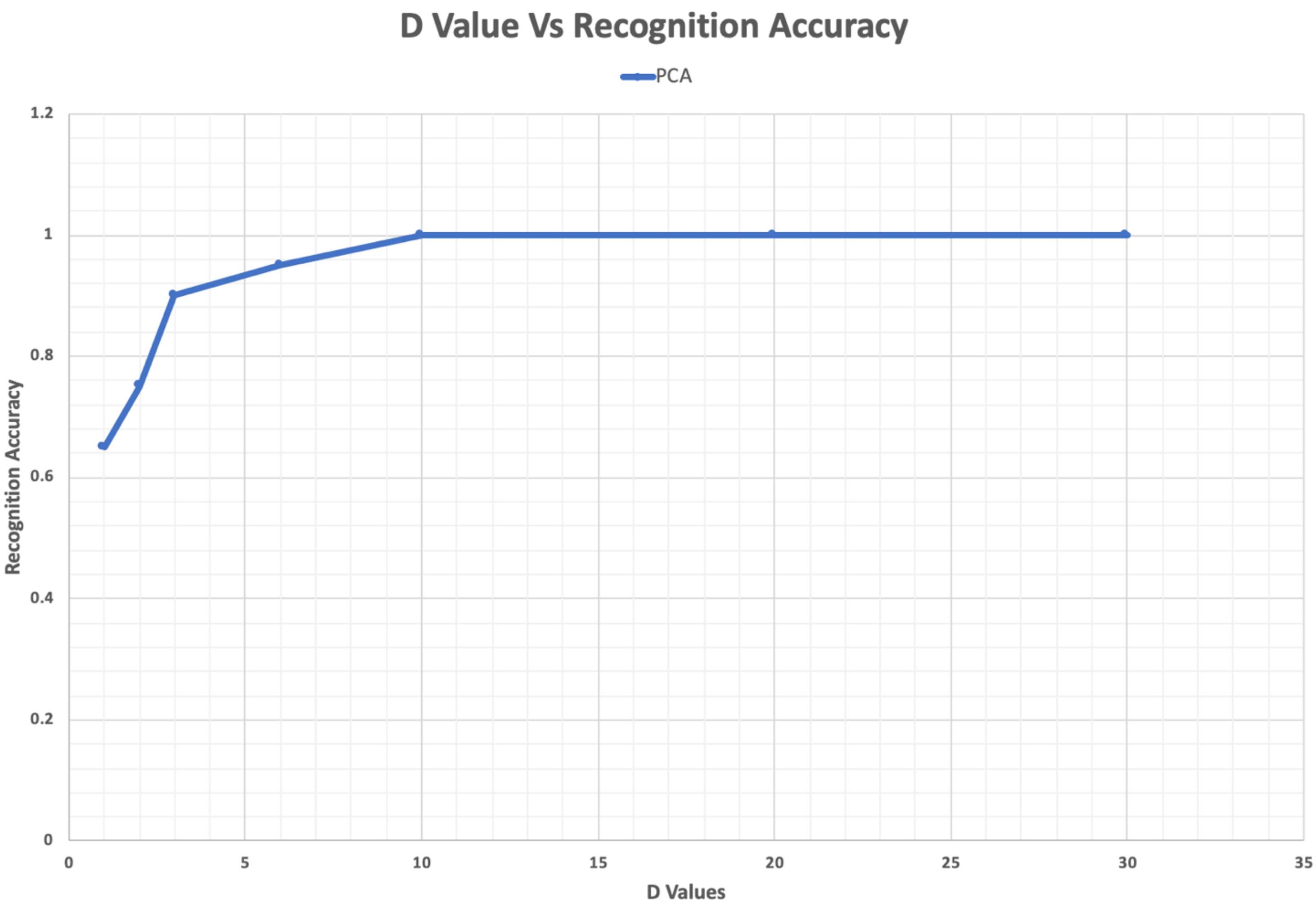
COEN 240 Machine Learning

Homework #6

Name: Alex Cherekdjian **Student ID:** 00001083236

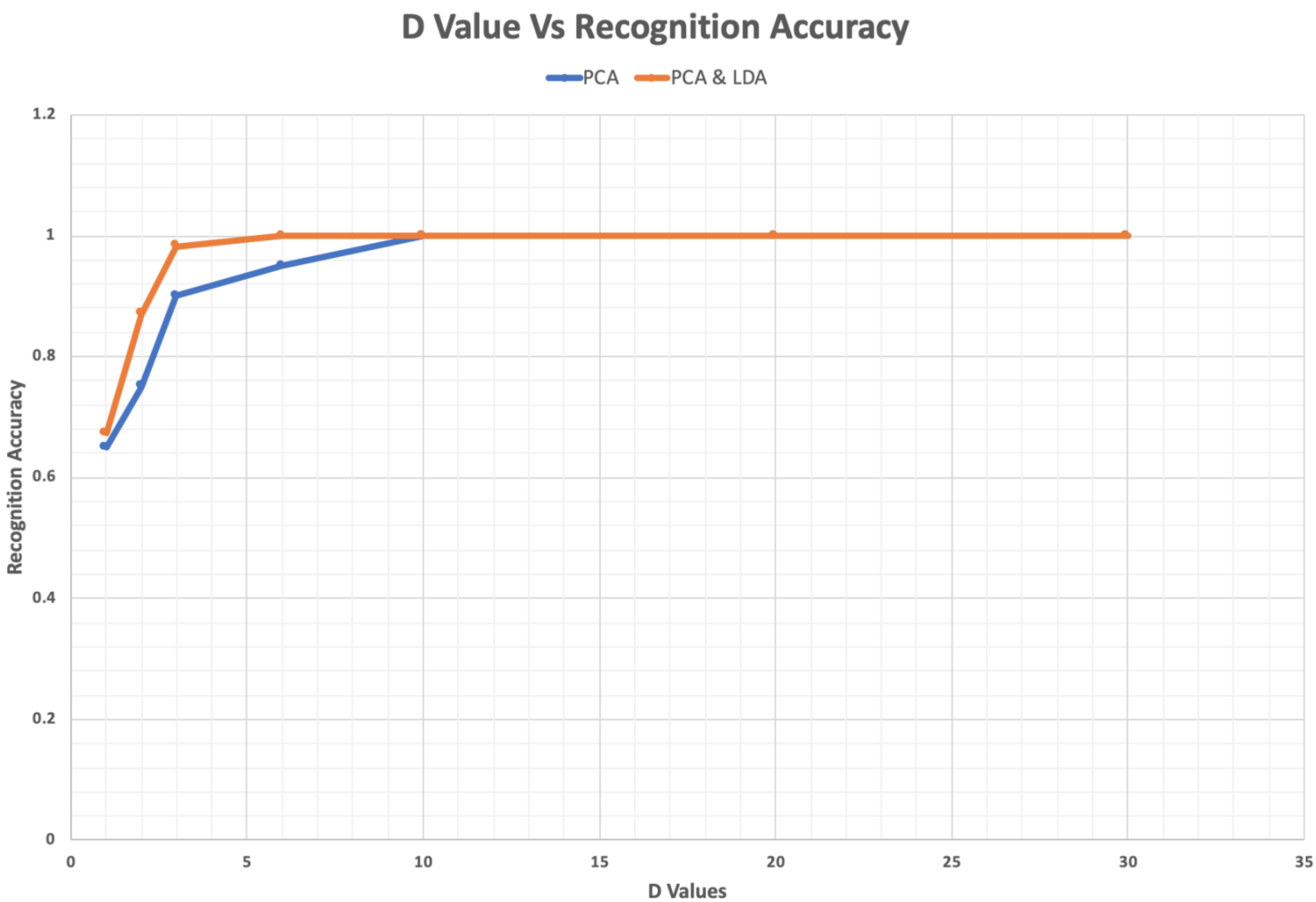
Problem 1.1



Comment: For PCA alone, trying different d values from 1, 2, 3, 6,10, 20, 30 proved to show an increase in recognition accuracy from a low of 65% to eventually getting 100% accuracy.

PCA Accuracy Results - 0.65, 0.75, 0.9, 0.95, 1.0, 1.0, 1.0

Problem 1.2



Comment: For PCA and LDA, trying different d values from 1, 2, 3, 6,10, 20, 30 proved to show an increase in recognition accuracy from a low of 67.25% to eventually getting 100% accuracy quicker than the PCA only model. Especially at lower dimensions, the accuracy is improved with the PCA and LDA model.

PCA Accuracy Results - 0.65, 0.75, 0.9, 0.95, 1.0, 1.0, 1.0
PCA & LDA Accuracy Results - 0.6725, 0.87, 0.9825, 1.0, 1.0, 1.0, 1.0

Attachment

Problem 1.1 and 1.2 Code (in zip file):

```
import os
import numpy as np
import pandas as pd
from skimage.io import imread
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# get working directory plus folder of pictures
current_dir = os.getcwd() + '/att_faces_10/'

# init samples and actual y values
samples = []
y = []

# read in folder by folder samples and y values
for i in range (1, 11):
    folder_path = current_dir +'s' + str(i)
    image_list = os.listdir(folder_path)

    for im in image_list:
        image = imread(folder_path + '/' + im)
        samples.append(image.flatten())
        y.append(i)

# init lists and variables for d and d0
d0 = 40
d_vals = [1,2,3,6,10,20,30]
recognition_accuracy_rates_pca = []
recognition_accuracy_rates_pca_lda = []

# init lda constant pca variable
pca0 = PCA(n_components=d0)

for d in d_vals:
    # init variables
    total_calculations = 0
    correct_classifications_pca = 0
    correct_classifications_pca_lda = 0

    # init lda models
```

```
    lda = LinearDiscriminantAnalysis(n_components=d) # FLD /LDA

    for i in range (0, 20):
        # initializing knn models
        knnModel_pca = KNeighborsClassifier(n_neighbors=1,
metric='euclidean')
        knnModel_pca_lda = KNeighborsClassifier(n_neighbors=1,
metric='euclidean')

        # splitting each class into train and test sets
        X_train1, X_test1, y_train1, y_test1 =
train_test_split(samples[0:10], y[0:10], test_size=0.2, random_state=25)
        X_train2, X_test2, y_train2, y_test2 =
train_test_split(samples[10:20], y[10:20], test_size=0.2, random_state=25)
        X_train3, X_test3, y_train3, y_test3 =
train_test_split(samples[20:30], y[20:30], test_size=0.2, random_state=25)
        X_train4, X_test4, y_train4, y_test4 =
train_test_split(samples[30:40], y[30:40], test_size=0.2, random_state=25)
        X_train5, X_test5, y_train5, y_test5 =
train_test_split(samples[40:50], y[40:50], test_size=0.2, random_state=25)
        X_train6, X_test6, y_train6, y_test6 =
train_test_split(samples[50:60], y[50:60], test_size=0.2, random_state=25)
        X_train7, X_test7, y_train7, y_test7 =
train_test_split(samples[60:70], y[60:70], test_size=0.2, random_state=25)
        X_train8, X_test8, y_train8, y_test8 =
train_test_split(samples[70:80], y[70:80], test_size=0.2, random_state=25)
        X_train9, X_test9, y_train9, y_test9 =
train_test_split(samples[80:90], y[80:90], test_size=0.2, random_state=25)
        X_train10, X_test10, y_train10, y_test10 =
train_test_split(samples[90:100], y[90:100], test_size=0.2, random_state=25)

        # combining all training and test sets
        x_train = X_train1 + X_train2 + X_train3 + X_train4 + X_train5 +
X_train6 + X_train7 + X_train8 + X_train9 + X_train10
        y_train = y_train1 + y_train2 + y_train3 + y_train4 + y_train5 +
y_train6 + y_train7 + y_train8 + y_train9 + y_train10
        x_test = X_test1 + X_test2 + X_test3 + X_test4 + X_test5 + X_test6 +
X_test7 + X_test8 + X_test9 + X_test10
        y_test = y_test1 + y_test2 + y_test3 + y_test4 + y_test5 + y_test6 +
y_test7 + y_test8 + y_test9 + y_test10

        # running pca analysis using specific d value
        pca = PCA(n_components=d)
        pca_operator = pca.fit(x_train)
        L0_pca = pca_operator.transform(x_train)
```

```
        # creating knn model and predicting values
        knnModel_pca.fit(L0_pca, y_train)
        y_pred_pca = knnModel_pca.predict(pca_operator.transform(x_test))

        # creating pca using d0=40 for input to lda
        pca0_operator = pca0.fit(x_train)
        L0 = pca0_operator.transform(x_train)

        # create lda operation from pca
        lda_operator = lda.fit(L0, y_train)
        train_proj_lda = lda_operator.transform(L0) # columns are examples

        # predict using knn
        knnModel_pca_lda.fit(train_proj_lda, y_train)
        y_pred_pca_lda =
knnModel_pca_lda.predict(lda_operator.transform(pca0_operator.transform(x_tes
t)))

        # counting predictions and total classifications
        for i in range(0, len(y_pred_pca_lda)):
            if y_pred_pca_lda[i] == y_test[i]:
                correct_classifications_pca_lda +=1
            if y_pred_pca[i] == y_test[i]:
                correct_classifications_pca +=1

            total_calculations +=1

    # append final results to list

recognition_accuracy_rates_pca.append(correct_classifications_pca/total_calcu
lations)

recognition_accuracy_rates_pca_lda.append(correct_classifications_pca_lda/tot
al_calculations)

# print out rates
print(recognition_accuracy_rates_pca)
print(recognition_accuracy_rates_pca_lda)
```