Eigenfaces of Faces and Music Classification

AMATH582: Homework 4
Maple Tan

March 8, 2019

1 Abstract

Applying the SVD to a data set of faces from Yale University allows us to extract the dominant modes in our set of images and also allows us to reconstruct a lower rank version of the images. We will also see how the reconstruction and dominant modes vary with uncropped and unaligned versions of our images. Then in the second part of this homework, we will attempt to classify music based on certain labels such as artist or genres.

2 Introduction and Overview

We begin with two folders containing two different types of black and white face pictures: one where the faces are all cropped and aligned and a set of black and white face pictures which are not aligned and cropped and also contains different expressions. Within the cropped faces folder, there will be a separate folder for each student with 64 pictures per a student, while all the pictures are in one folder for the uncropped case. We will reshape each images into a column vector and store the column vector into a big matrix for both picture types. We will also create a version where we apply a filter to the images in the frequency domain. Applying the SVD to our big matrix, we will find the dominant features and how many modes are necessary for good image reconstruction.

For classifying music, we have a series of five second samples from a variety of different artists and genres. We will look at three different tests for these samples:

- **Band Classification**: Consider three different artists and see if the algorithm can accurately assign samples to the correct artist.

- **The Case for Seattle**: Consider music by three artists in the same genre and see if the algorithm can accurately assign samples to the correct artist.
- **Genre Classification**: Group songs by genre and see if the algorithm can accurately assign samples to the correct genre.

3 Theoretical Background

In this specific example, the SVD can be used to generate a feature space for the images and will extract the dominant correlations among the images. Suppose we have our matrix of image data `X` with its SVD decomposition `X = usv'`. To extract the dominant correlations among the images, we will reshape and plot the first few columns of the `U` matrix. Since the columns of `U` represents features that are correlated or common throughout our set of images, it will give an idea of common physical traits that all the faces have. For `v`, this represents the importance of each feature to an individual image. We can use `v` this to determine if there are two different groups within our set of images, such as if we applied the SVD to a set of dog and cat images for example.

In terms of image reconstruction, we can use the singular values from our matrix `s` to determine the proper rank needed to construct a low-rank reconstruction of the original images. This is due to how the singular values of the SVD represent the amount of variance in each principal component. The larger the singular value, the more variance in that principal component. We will observe that a small portion of the principal components contains the majority of the variances. For reconstruction, we reduce the dimension of our `s` matrix so that it corresponds with our rank r and rebuild our original data matrix around that as below, where X_r represents the r-ranked reconstruction of X, and the numbers in the brackets below represent the dimensions.

$$\begin{aligned} X_r = &\quad U_r \qquad S_r \qquad V_r^* \\ [m \times n] = &[m \times r][r \times r][r \times n] \end{aligned}$$

We will also do this for images that have a filter applied to it in the frequency domain. For more details about the Fourier Transformation, check Homework 1. One thing to note for the Fourier Transformation in this case is that we will apply a filter that filters our the lower frequencies, which results in the outlines being more prominent in our image. This is because the higher frequencies of an image in the frequency domain correspond with the edges and details, which naturally are areas of high frequency.

When it comes to classification in machine learning, the goal ultimately is to take data and determine which group it belongs to. There are two different types: supervised and unsupervised. In our music classification example, we will employ supervised learning where we train the algorithm on data, known as features, with a specific label. The algorithm will then use the training data to predict where new data may be classified. One way to test how good our algorithm is at predicting thing is to feed in new data where we already know what label it belongs to. This is known as our testing data and we can compare where the algorithm classifies the data to where it actually belongs. It is important to note that in order to get the most accurate score, we must separate and keep separate our training and testing data. For this particular example, we will be utilizing three different machine learning algorithms: K Nearest Neighbors, Support Vector Machine, and Classification Tree Classifier.

- **K Nearest Neighbors**: The theory behind the this algorithm is that data belonging to the same label would theoretically be clustered close to one other. When training this algorithm in the supervised case, it essentially creates k different neighborhoods or groups of data points corresponding to each of the k labels. Then for a given new data point x, K Nearest Neighbors will find the neighborhood that is closest to that data point x and say that x belongs to the label corresponding with that neighborhood.
- **Support Vector Machine (SVM)**: SVM projects the training data into a higher dimension and splits the data with hyper planes. To find the best hyper plane, the algorithm will essentially find the plane that would cause the most separation between the different labeled clusters. Then for a new data point x, it then determines which side of the hyper plane it falls on to determines its label.
- **Decision Tree Classifier**: On a basic level, this algorithm takes a set of labels l and a matrix of features f and scans through each of the features to finds the feature f_j that is best associated with a label and then finds the best way to split the data on f_j, resulting in two branches. It then follows to repeat this for the resulting two branches. This algorithm then terminates once it reaches a unique cluster. For a new data point x, it simply follows the tree until it reaches the unique cluster.

4 Algorithm Implementation and Development

In the first part of this assignment, we must begin by loading our data. Starting with importing `os`, `numpy as np`, and `matplotlib.pyplot as plt` in Python, we then start by using the `os.listdir(path)` method to obtain the name of all items in the listed in the folder directed by `path`. In particular, we will use this function to load and access all the images in each of the different folders. Now that we have access to all the images we need, we then initialize a matrix to store each image as a column, one version for unmodified images and the second for a FFT filter applied to the image. Next, we loop through each image, using the `open` method from the `Image` package of the `PIL` library to convert each image into a matrix of values on the gray scale. We can then use the `np.reshape` method from the `numpy` library to reshape our matrix into a column vector. Since `np.reshape` returns a (n,1) vector called `col` for example, in order to store this column into our matrix we have to take `col[:,0]`.

The data next must be mean centered, as necessary for the SVD. After mean centering the data, we then use `np.linalg.svd` apply the SVD to obtain u, s, and v. To obtain the dominant mode, we will use `np.reshape` again to reshape our u columns back into an image and then use `plt.imshow` to plot the dominant feature. Then for image reconstruction, we will begin by graphing the singular values squared over the sum of the singular values squared to obtain the amount of variance in each principal component. For each trial (the Fourier transformed and unmodified images), we then use a `while` loop to count how many principal components contain 90% of the variance and this will be our rank r. We then use this rank r to create the rank r reconstruction of the images. One important thing to note when it comes to using the SVD in the `numpy` library is that the returned `v` is already transposed. After reconstruction, we reshape the columns of our reconstructed matrix back into a matrix that we can use `plt.imshow` to plot the images.

For music classification, we will be using the SVD of the spectrogram of all our songs as our features. We will begin by creating a spectrogram of each five second sample. To do this, we need to load our music samples in a similar manner as the previous part, instead using the `wavfile` method from the `scipy.io` library to load our music. The song samples end up being different lengths so we have to loop through all the songs first to find the minimum length of each song. After finding the minimum length, we then need to resample the song to preserve memory. Due to memory issues, we will create a vector that samples every 8th point of our original signal up until our minimum length.

Now that we have resampled each song, we now create the spectrogram of each song. For more details on creating the spectrogram, reference Homework 2. After creating the spectrogram, we then reshape each spectrogram into a column vector and store it into a large matrix that will contain each of our spectrogram in a similar manner in the previous problem. For the sake of preserving memory, we will also save the large data matrix as a `.csv` file for each label so that we may use it later.

In a separate Python script, we will load each of the `.csv` files for each category and then compile them into one big matrix. Next we will mean center the matrix and then apply the SVD. We will then figure out how many principal components, r contain 90% of the variance and use the first r principal components in our classification algorithm as our features. We then use the `train_test_split` method from the `sklearn.model_selection` library to split our data into training and testing sets. In order to preserve memory, we use indexes instead of the actual data for the features.

Finally, we import inbuilt methods for each machine learning algorithm, all from the `sk.learn` library, and train our data in each. Then since there is a built in score function with each algorithm, we can use that to test and evaluate our algorithm against our testing data. We then plot the accuracy for each test. As a reference, we also attempt a trial where we use the raw spectrogram as our features instead.

5 Summary and Conclusion

Starting with the first part of our assignment, we will look at the amount of variation each

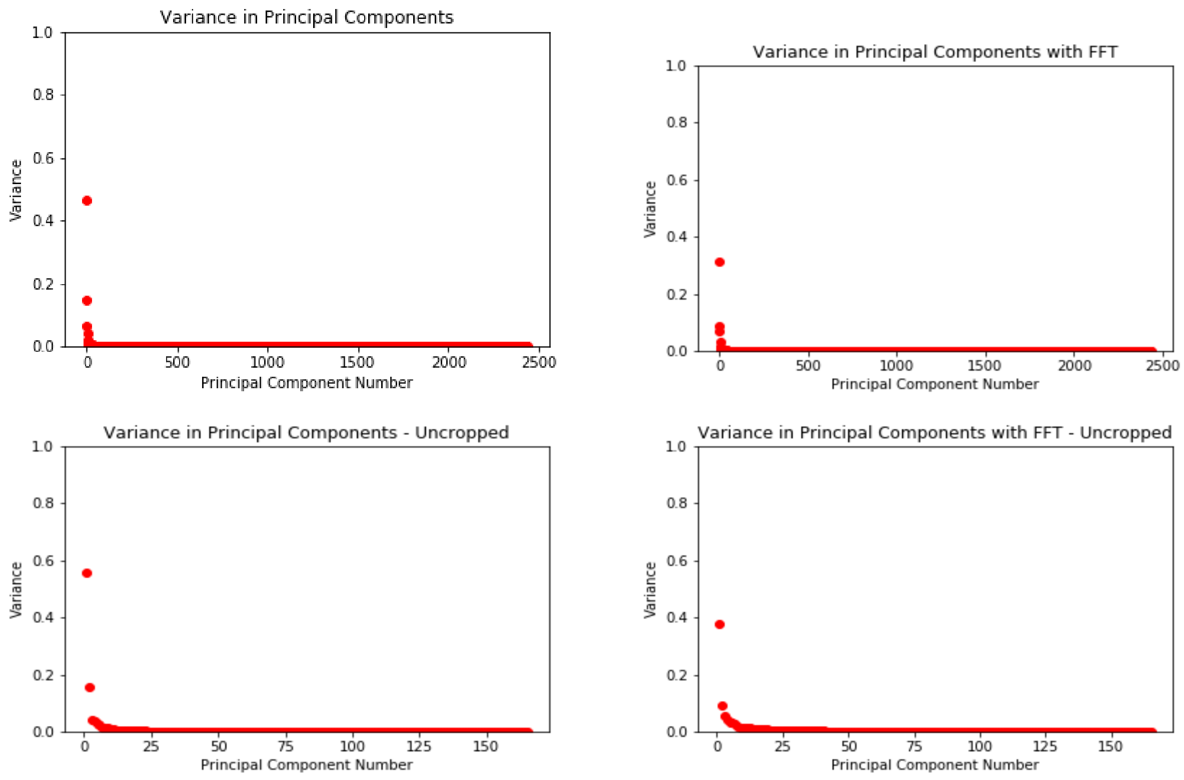


Figure 1: Here we see that the first few principal component contains the vast majority of the variance in all cases.

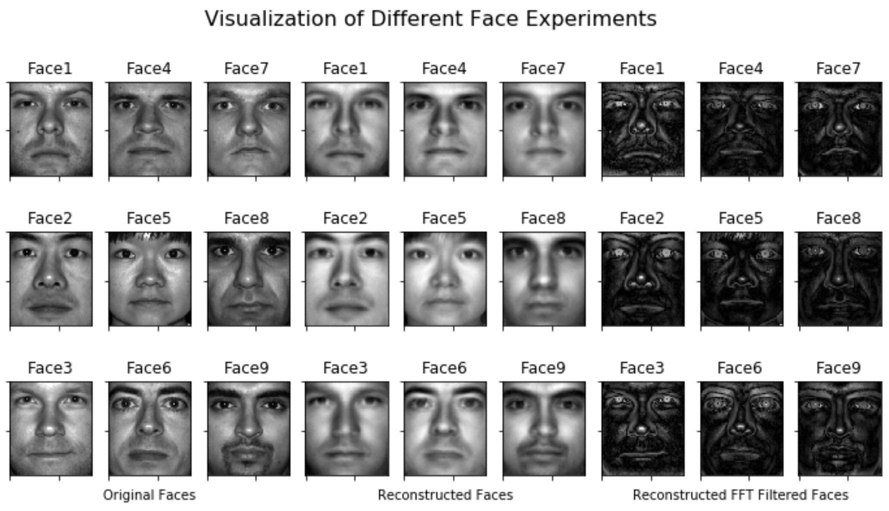


Figure 2: The reconstructed faces only required 36 modes to capture 90% of the variance opposed to the full 2500+ modes. As we can see, though the reconstruction is may not be as detailed as the original faces, the main details of the face are still present and very clear. Similarly, the FFT filtered images required 422 modes to capture 90% of the variance.

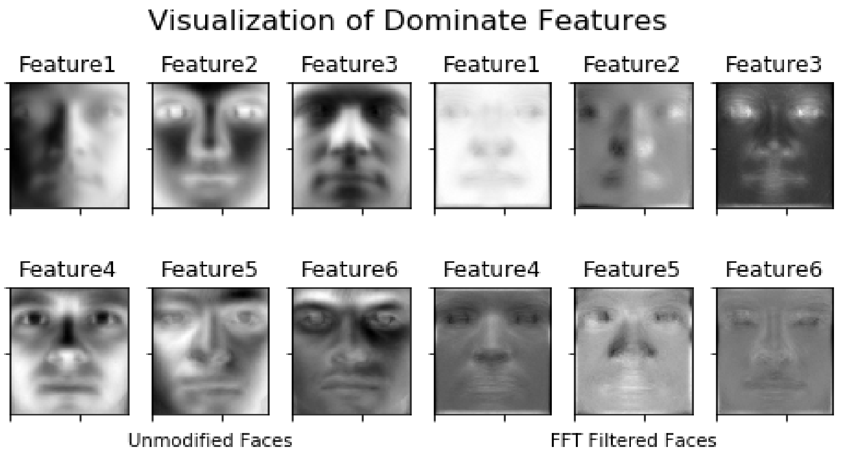


Figure 3: In both cases, certain facial features such as the eyes, nose and mouth are very prominent, suggesting that these are common features between out different images. One interesting thing is that after applying the FFT filtered images also show a feature that separates the lights nad shadows of a face from the outlines as seen in feature 2.

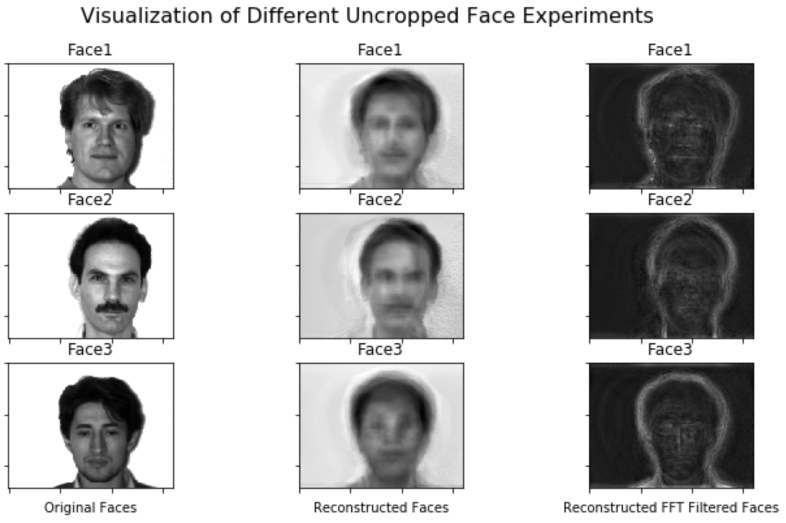

Visualization of Different Uncropped Face Experiments
Face1
Face2
Face3
Original Faces
Reconstructed Faces
Reconstructed FFT Filtered Faces

Figure 4: Compared to the cropped case, we see that a low amount of modes also contain 90% of the variance - 13 to be exact but we also see that the reconstruction is not nearly as good due to the aligned nature of the photos and different expressions. We see a lot of blurriness especially within the face, which implies a lack of correlation between the different images.

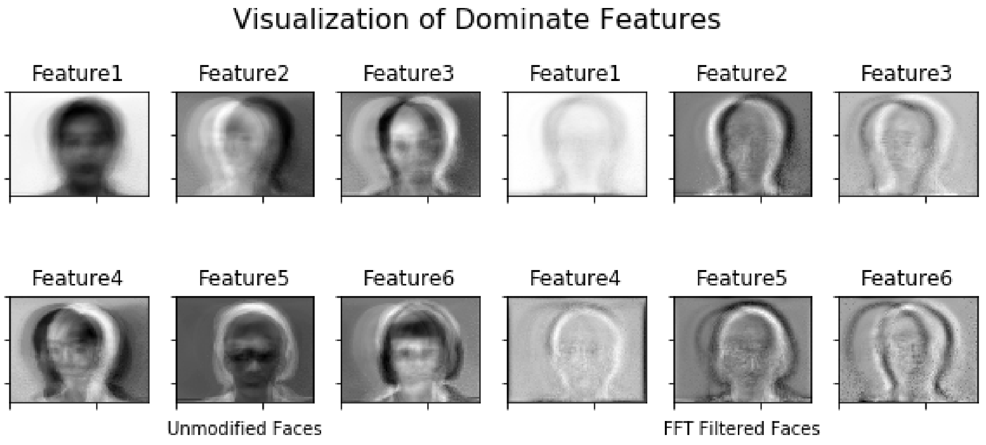


Figure 5: When comparing the dominate features of the unaligned and uncropped faces to the cropped faces, there is a definite lack of clarity in terms of head outlines and facial features. This makes sense because the lack of alignment throws off correlation between the different images.

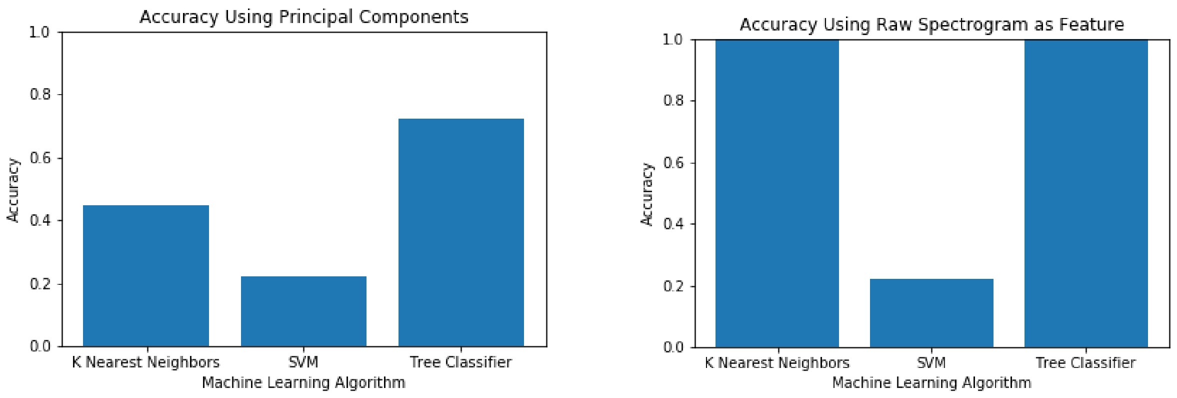


Figure 6: Accuracy using the different algorithms for Test 1. In this case, the Tree Classifier does the best job in terms of accuracy. We also see that using the raw spectrogram in this case results in major overfitting in the K Nearest Neighbors and Tree Classifier while SVM remains pretty similar in both cases.

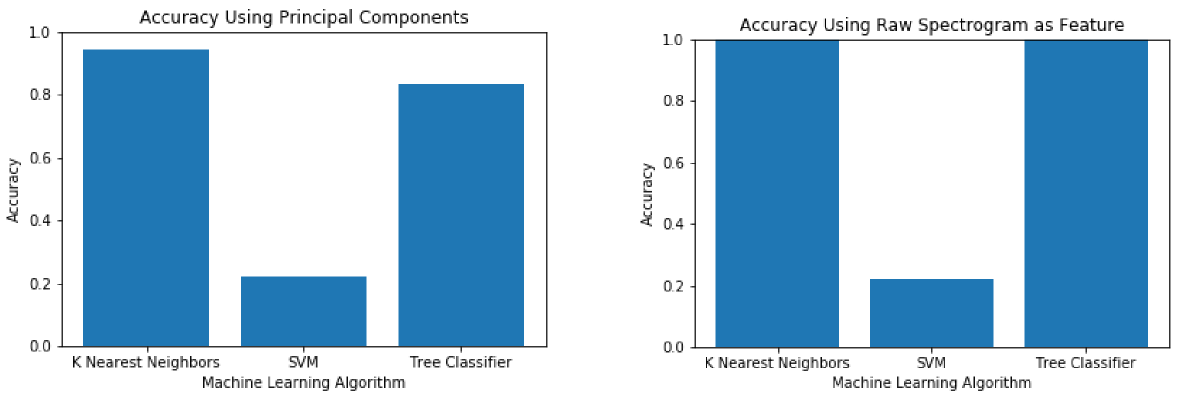
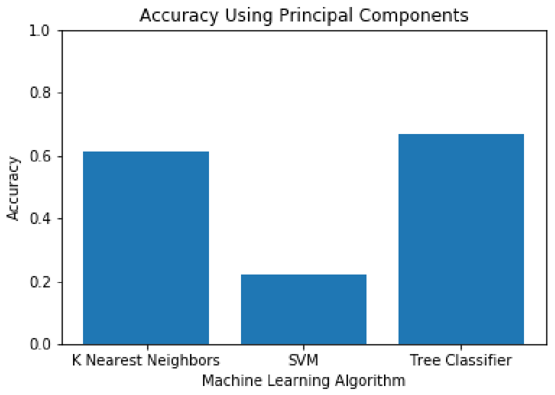


Figure 7: Accuracy using the different algorithms for Test 2. The algorithmns save for SVM do a better job compared to Test 1, with K Nearest Neighbors being the most accurate in this case. Again, there is issues with overfitting with K Nearest Neighbors and Tree Classifier



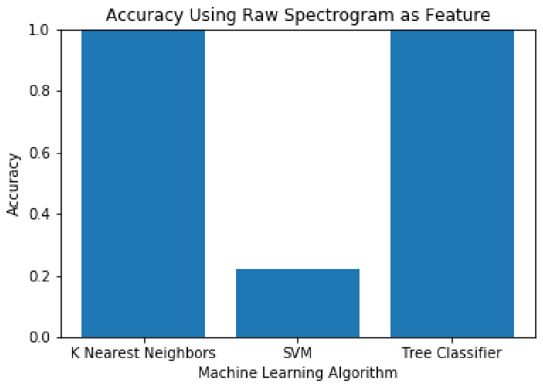


Figure 8: Accuracy for the differnet algorithms for Test 3. We see that the accuracy between Decision Tree and K Nearest Neighbors is rather similar, with Decision Tree being slightly better. Again overfitting is an issue when using the raw spectrogram.

In general, we see SVM not providing accurate results which may point to the data not being the best to use for the SVM algorithm. We see better results with Tree Classifier and K Nearest Neighbors. The principal components also provide a better feature space compared to the raw spectrogram.

A Python Code: Eigenfaces and Faces - Cropped

```
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image as im

# Set up Data

d = 'CroppedYale'   # Main folder Name
students = os.listdir(d)    # Lists directories in folder d

# Initialize matrix that will store all student images as columns
pic = im.open(os.listdir(d + '\\' + students[0])[0])
# Number of pictures total for all students
pnum = len(students)*len(os.listdir(d + '\\' + students[0]))
fw = pic.size[0]        # picture width
fh = pic.size[1]        # picture height
colength = fw*fh

studentall_fft = np.zeros([colength,pnum])      # FFT Filtered Images
studentall = np.zeros([colength,pnum])          # Unmodified Images

# Create filter for FFT Filter
x = np.arange(0, fw)
y = np.arange(0, fh)

X, Y = np.meshgrid(x,y)
fil = 1 - np.exp(-0.01*((X - fw / 2) ** 2 + (Y - fh / 2) ** 2))

k = 0;
for s in students:
    sf = d + '\\' + s      # Name of specific student folder
    files = os.listdir(sf)  # Obtains all files in student folder
    for f in files:
        # Reshapes image into column and add to matrix
        I = im.open(sf+'\\'+f)
        col = np.reshape(I, [colength,1])
        studentall[:, k] = col[:,0]
        # Applies FFT Filter onto Images and stores in matrix
        It = np.fft.fft2(I)
        Its = np.fft.fftshift(It)
        Itsf = Its*fil
        Itf = np.fft.ifftshift(Itsf)
        If = np.fft.ifft2(Itf)
        col = np.reshape(abs(If), [colength,1])
        studentall_fft[:, k] = col[:,0]

        k = k+1
# %% Preview Images as necessary
# plt.imshow(np.reshape(studentall[:,200], [fh, fw]), cmap = 'gray')
# plt.imshow(np.reshape(studentall_fft[:,200], [fh, fw]), cmap = 'gray')

#%% Receives a data set and returns the mean centered version of the data
studentall_ori = studentall
def center_mean(data):
    means = np.mean(data, axis = 0)
    data = data-means
    data = data*(1/(len(data)-1))
    return data

studentall = center_mean(studentall)
studentall_fft = center_mean(studentall_fft)

# %% Apply SVD - note that in numpy, v is returned transposed
u, s, v = np.linalg.svd(studentall, full_matrices=False, compute_uv=True)

# %% Apply SVD to FFT Filtered data
uf, sf, vf = np.linalg.svd(studentall_fft, full_matrices=False, compute_uv=True)

# %% Function receives a sigma vector of singular values and
#       plots the variances in each PC. Then it titles the graph
#       with the inputed title and saves it as the inputed file name
def plot_pcvars(sigma, file_name, title):
    plt.plot(np.arange(1,pnum+1), (sigma**2)/np.sum(sigma**2), 'ro')
    plt.title(title)
    plt.ylim([0,1])
    plt.xlabel('Principal Component Number')
    plt.ylabel('Variance')
    plt.savefig(file_name)

plot_pcvars(s, 'pc_var.png', 'Variance in Principal Components')
# %%
plot_pcvars(sf, 'pc_varz_fft.png', 'Variance in Principal Components with FFT')

# %% Receives the components of the SVD decomposition and returns
#       how many principal components contain 90% of the variance
def get_rank(u,s,v):
    var_tot = 0;
    rank = 0;
    while var_tot < 0.90:
        var_tot = var_tot + (s[rank]**2) / np.sum(s**2)
        rank = rank+1
    return rank

rank = get_rank(u,s,v)
rank_fft = get_rank(uf,sf,vf)

# %% Receives u,s,v, the components of the SVD decomposition and returns the
#        r-rank version of the original data
def recon_r_rank(rank,u,s,v):
    rank = rank
    u_rec = u[:, 0:rank]
    s_rec = np.diag(s)[0:rank, 0:rank]
    v_rec = v[0:rank,:]
    # Reconstruct data with r-rank versions of U,S,V
    x_rec = np.matmul(np.matmul(u_rec, s_rec), v_rec)
    return x_rec

x_rec = recon_r_rank(rank,u,s,v)
x_rec_fft = recon_r_rank(rank_fft,uf,sf,vf)

# %% Create all Visualizations
faces = np.arange(0,9)*64
pf = np.concatenate((studentall_ori[:,faces], ...
 x_rec[:,faces], x_rec_fft[:,faces]),axis=1)
fig,ax = plt.subplots(3,9, figsize=(12,6))
xl = np.shape(ax)[0]
yl = np.shape(ax)[1]
```

```
i = 0
j = 0
k = 0
ax[2][1].set_xlabel('Original Faces')
ax[2][4].set_xlabel('Reconstructed Faces')
ax[2][7].set_xlabel('Reconstructed FFT Filtered Faces')
plt.suptitle('Visualization of Different Face Experiments', ...
ha='center', fontsize=16)
for p in np.arange(0,xl*yl):
    pic = np.reshape(pf[:,p], [fh, fw])
    ax[i][j].imshow(pic, cmap='gray')
    ax[i][j].set_title('Face' + str(k+1))
    ax[i][j].set_xticklabels([])
    ax[i][j].set_yticklabels([])
    i = i+1
    k=k+1
    if i == xl:
        i = 0
        j = j+1
    if k == yl:
        k = 0
#plt.savefig('all_faces.png')
plt.show()
#%%
fig, ax = plt.subplots(2,6, figsize=(8,4))
i = 0
j = 0
pcs = u[:,0:6]
plt.suptitle('Visualization Dominate Features', ha='center', fontsize=16)
ax[1][1].set_xlabel('Unmodified Faces')
ax[1][4].set_xlabel('FFT Filtered Faces')
for p in np.arange(0,6):
    pcim = np.reshape(pcs[:,p], [fh, fw])
    ax[i][j].imshow(pcim, cmap='gray')
    ax[i][j].set_title('Feature' + str(p+1))
    ax[i][j].set_xticklabels([])
    ax[i][j].set_yticklabels([])
    j = j+1
    if j == 3:
        j = 0
        i = i+1

i = 0
j = 3
pcs = uf[:,0:6]
for p in np.arange(0,6):
    pcim = np.reshape(pcs[:,p], [fh, fw])
    ax[i][j].imshow(pcim, cmap='gray')
    ax[i][j].set_title('Feature' + str(p+1))
    ax[i][j].set_xticklabels([])
    ax[i][j].set_yticklabels([])
    j = j+1
    if j == 6:
        j = 3
        i = i+1
plt.savefig('dom_faces.png')
plt.show()
```

B Python Code: Eigenfaces and Faces - Uncropped

Note that only the Data initialization and visualization are significantly different in this case, so only those sections of the code have been included.

```
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image as im

# Set up Data

d = 'yalefaces'   # Main folder Name
students = os.listdir(d)    # Lists directories in folder d

# Initialize matrix that will store all student images as columns
pic = im.open(d + '\\' + students[0])
pnum = len(students)    # Number of pictures total for all students
fw = pic.size[0]        # picture width
fh = pic.size[1]        # picture height
colength = fw*fh
#%%
```

```
studentall_fft = np.zeros([colength,pnum])
studentall = np.zeros([colength,pnum])

# Create filter
x = np.arange(0, fw)
y = np.arange(0, fh)

X, Y = np.meshgrid(x,y)
fil = 1 - np.exp(-0.01*((X - fw / 2) ** 2 + (Y - fh / 2) ** 2))

k = 0;
for s in students:
    I = im.open(d+'\\'+s)
    col = np.reshape(I, [colength,1])
    studentall[:, k] = col[:,0]

    # Applies FFT Filter onto Images
    It = np.fft.fft2(I)
    Its = np.fft.fftshift(It)
    Itsf = Its*fil
    Itf = np.fft.ifftshift(Itsf)
    If = np.fft.ifft2(Itf)
    col = np.reshape(abs(If), [colength,1])
    studentall_fft[:, k] = col[:,0]

    k = k+1

# %% Create all Visualizations
num = 2
faces = np.arange(0,3)*11
pf = np.concatenate((studentall_ori[:,faces], ...
 x_rec[:,faces], x_rec_fft[:,faces]),axis=1)
fig,ax = plt.subplots(3,3, figsize=(12,6))
xl = np.shape(ax)[0]
yl = np.shape(ax)[1]
i = 0
j = 0
k = 0
ax[2][0].set_xlabel('Original Faces')
ax[2][1].set_xlabel('Reconstructed Faces')
```

```
ax[2][2].set_xlabel('Reconstructed FFT Filtered Faces')
plt.suptitle('Visualization of Different Uncropped Face Experiments',...
 ha='center', fontsize=16)
for p in np.arange(0,xl*yl):
    pic = np.reshape(pf[:,p], [fh, fw])
    ax[i][j].imshow(pic, cmap='gray')
    ax[i][j].set_title('Face' + str(k+1))
    ax[i][j].set_xticklabels([])
    ax[i][j].set_yticklabels([])
    i = i+1
    k=k+1
    if i == xl:
        i = 0
        j = j+1
    if k == yl:
        k = 0
plt.subplots_adjust(wspace=0.0001)
plt.savefig('all_faces_uncropped.png')
plt.show()
```

C Python Code: Music Classification - Part 1

```
import numpy as np
import os
from scipy.io import wavfile

F = 'Music'    # Main Folder
tests = ['test1', 'test2', 'test3']

artist_path = F + '\\' + tests[2]
artists = os.listdir(artist_path)
the_a = artists[2]
song_path = artist_path + '\\' + the_a
songs = os.listdir(song_path)


# %% First Find Minimum Song Length
fs, dat = wavfile.read(song_path + '\\' + songs[0])
s_min = len(dat)
```

```
for a in artists:
    song_path2 = artist_path + '\\' + a
    songs2 = os.listdir(song_path2)
    for s in songs2:
        fs, dat = wavfile.read(song_path2 + '\\' + s)
        if len(dat) < s_min:
            s_min = len(dat)


#%% Create Spectrogram
rs = 8
sl = int(s_min/rs)
spc_song = np.zeros([sl*51,len(songs)])
nn = 0
for stu in songs:
    fs, dat = wavfile.read(song_path + '\\' + stu)
    sn = s_min
    # Resample Data - take every 8th point
    s1d = (dat[np.arange(0, sn, rs),0] + dat[np.arange(0, sn, rs),1])/2
    pn = len(s1d)

    step = 0.1
    timeslide = np.arange(0, 5+step, step)
    time = np.linspace(0, 5, pn)
    pk = (2*np.pi)*np.concatenate((np.arange(0,pn/2), np.arange(-pn/2,0)))
    tlen = len(timeslide)

    width = 1000
    spc = np.zeros([tlen, pn])
    k = 0
    width = 100
    for t in timeslide:
        g = np.exp(-width*(time-t)**10)
        s1df = s1d*g
        s1dft = np.fft.fft(s1df)
        spc[k, :] = abs(np.fft.fftshift(s1dft))
        k=k+1

    spc_song[:, nn] = np.reshape(spc, [tlen*pn,1])[:,0]
    nn=nn+1
```

```
# %% Save the matrix
np.savetxt(the_a + '.csv', spc_song, delimiter=',')
```

D Python Code: Music Classification - Part 2

```
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

test = 'test2'
artists = os.listdir('music\\' + test)

# Create Labels
samples = 30
labels = []
k = 0
for s in np.arange(0,samples*3):
    labels.append(artists[k])
    if (s+1) % 30 == 0:
        k = k+1
# %%
X1 = np.genfromtxt(artists[0] + '.csv', delimiter=',')
#%%
X2 = np.genfromtxt(artists[1] + '.csv', delimiter=',')
#%%
X3 = np.genfromtxt(artists[2] + '.csv', delimiter=',')

#%%
X = np.concatenate((X1, X2, X3),axis=1)

#%% Mean Center the data
means = np.mean(X, axis = 1)
means = np.reshape(90, 1)
X = X-means
X = X*(1/((np.shape(X)[1]-1)))

#%% Apply SVD to Data Matrix
```

```
u,s,v = np.linalg.svd(X, full_matrices = False)

#%% Get necessary number of modes
var_tot = 0;
rank = 0;
while var_tot < 0.90:
    var_tot = var_tot + (s[rank]**2) / np.sum(s**2)
    rank = rank+1
#%% Obtain the first r = rank columns of V for our features
feats = v[:, 0:rank];


#%% Split in to Training and Testing Data
X_ind = np.arange(0, 90)
X_train, X_test, y_train, y_test = train_test_split(X_ind, labels, ...
test_size=0.2, random_state=42)

# %% KNearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
n = 3
clf = KNeighborsClassifier(n)
clf.fit(feats[X_train,:], y_train)
score_kmean = clf.score(feats[X_test,:], y_test)

# %%  SVM
from sklearn.svm import SVC
clf = SVC(gamma='auto')
clf.fit(feats[X_train,:], y_train)
score_SVM = clf.score(feats[X_test,:], y_test)

# %% TreeClassifier
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
clf.fit(feats[X_train,:], y_train)
score_CTree = clf.score(feats[X_test,:], y_test)

# %% Plot the accuracy

plt.title('Accuracy Using Principal Components')
plt.bar(['K Nearest Neighbors', 'SVM', 'Tree Classifier'],...
 [score_kmean, score_SVM, score_CTree])
plt.ylim([0, 1])
plt.xlabel('Machine Learning Algorithm')
plt.ylabel('Accuracy')
plt.savefig(test+'_acc_v.png')

# %% KNearest Neighbors with Raw Data
n = 3
clf = KNeighborsClassifier(n)
clf.fit(X[X_train,:], y_train)
score_kmeanX = clf.score(X[X_test,:], y_test)

# %%  SVM with Raw Data
clf = SVC(gamma='auto')
clf.fit(X[X_train,:], y_train)
score_SVMX = clf.score(X[X_test,:], y_test)

# %% TreeClassifier with Raw Data
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X[X_train,:], y_train)
score_CTreeX = clf.score(X[X_test,:], y_test)

# %% Plot the accuracy for Raw Data
plt.title('Accuracy Using Raw Spectrogram as Feature')
plt.bar(['K Nearest Neighbors', 'SVM', 'Tree Classifier'],...
 [score_kmeanX, score_SVMX, score_CTreeX])
plt.ylim([0, 1])
plt.xlabel('Machine Learning Algorithm')
plt.ylabel('Accuracy')
plt.savefig(test+ '_acc.png')
```