hw4

Name: Ruffin White
ID: A53229660

March 9, 2018

Contents

HW4 : Math for Robotics

Author: Ruffin White
Course: CSE291
Date: Mar 9 2018

1.

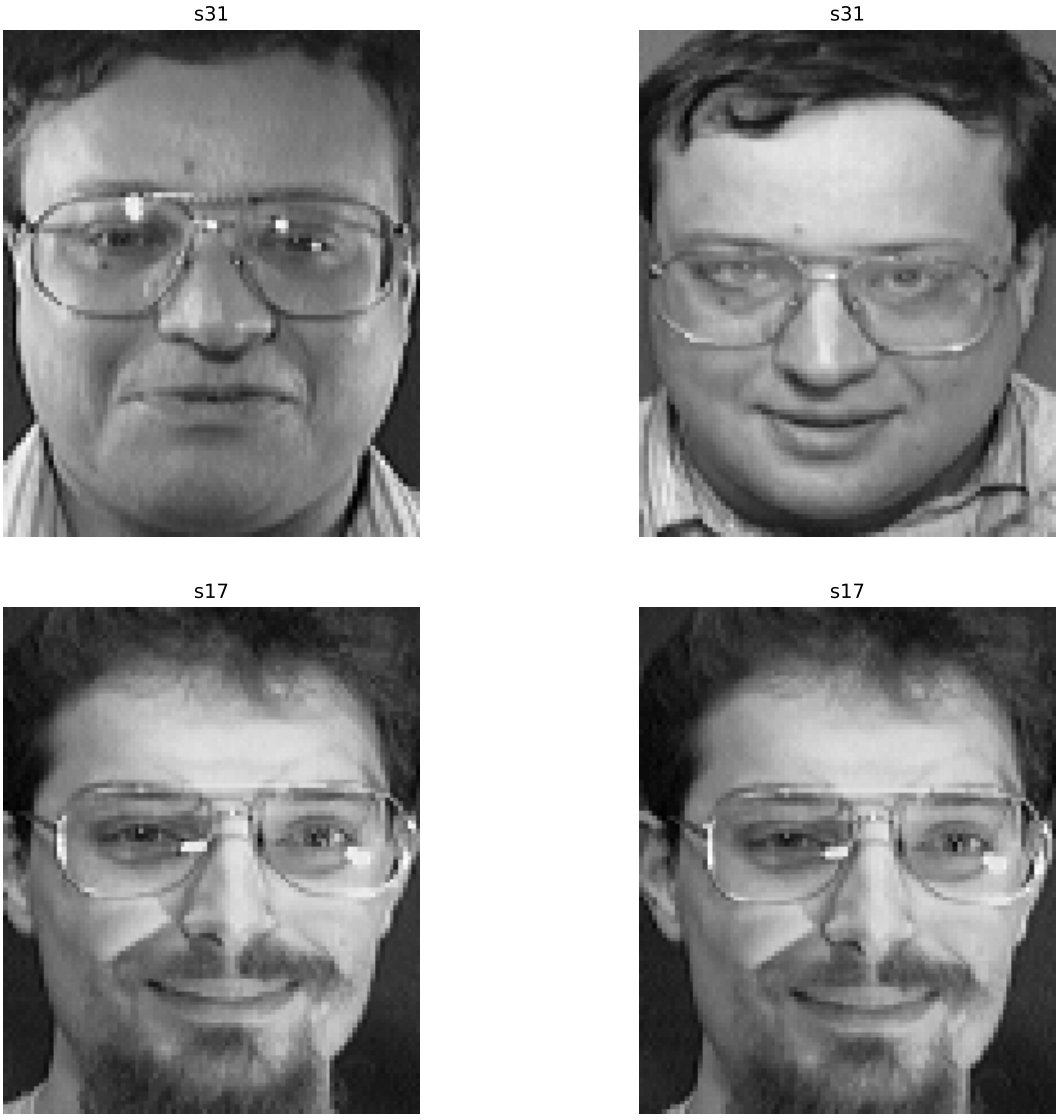
> Use the ATT dataset http://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att faces.zip. to compute subspaces for the PCA and LDA methods. Provide illustration of the respective 1st, 2nd and 3rd eigenvectors. Compute the recognition rates for the test-set. Report
>
> - Correct classification
> - Incorrect classification
>
> Provide at least one suggestion for how you might improve performance of the system

First we'll import some common numerical libraries, as well as some image, data structure libraries for manipulating the dataset.

Next we'll add a helper function to recursively search for the face images in the dataset, load them into flatten feature vectors, then interleave them into a dataframe with labels ascertained from the originating directory name.

We then read in the data and split the feature vectors and labels into separate yet commonly indexed dataframes.

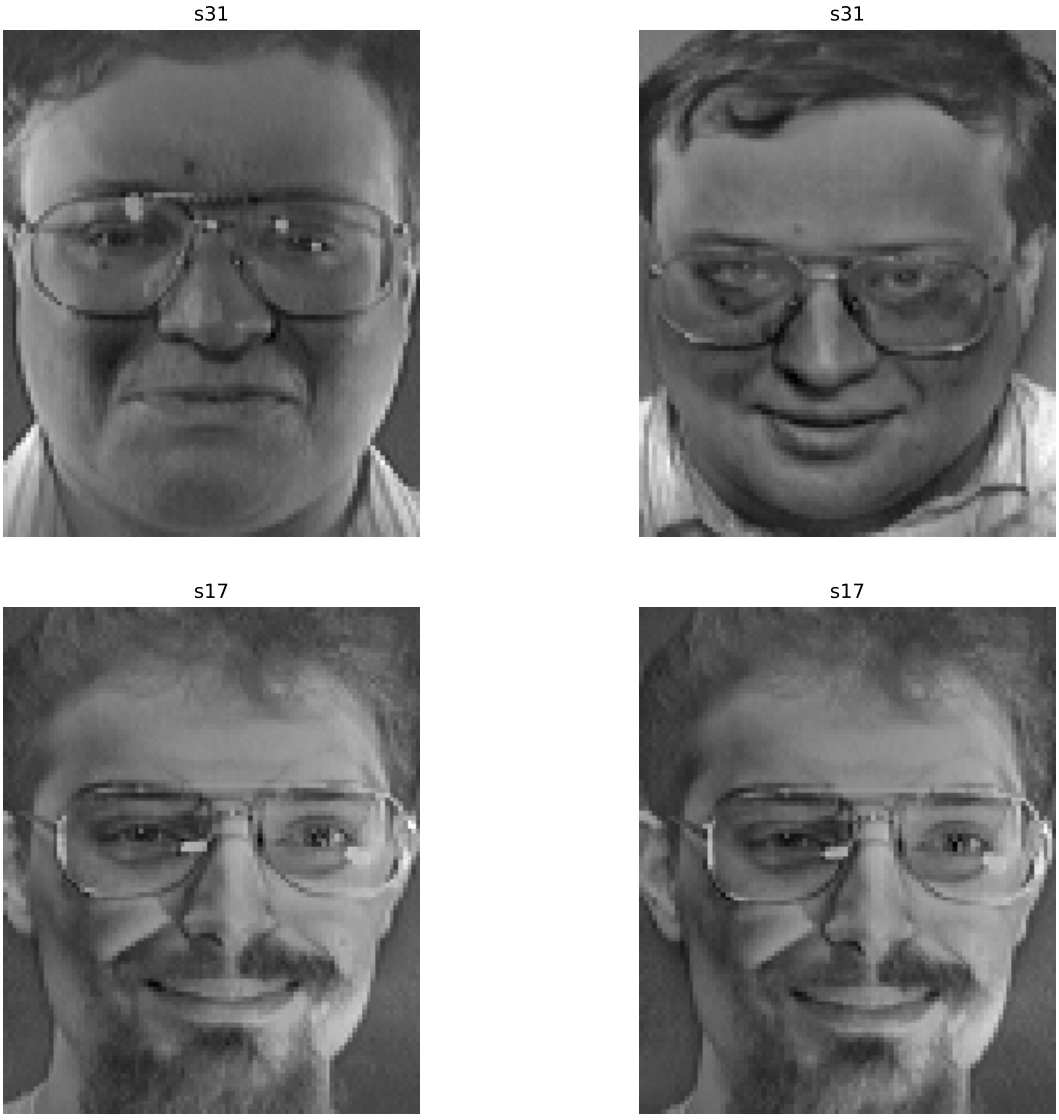
We can verify the integrity of the parsed data by recalling the original shape of the images to reshape the flattened features back into the pixelated images they encode, as shown below.



Additionally, we can also Standardize the feature vectors. Whether we should standardize the data prior to a PCA depends on the measurement scales of the original features. PCA yields a feature subspace that maximizes the variance along found axes, so it makes sense to standardize the data, especially, if it was measured on different scales (which isn't necessarily the case here given all pixel values fall within a common range).

This will transform the data onto unit scale (mean=0 and variance=1), which is a common requirement for the optimal performance of many machine learning algorithms. This should also heighten the sensitivity to pixels that change only subtly thought the hole dataset. We will see if this is to our classifiers benefit or detriment later on.

For comparision, the standerdized data is also visualed below:

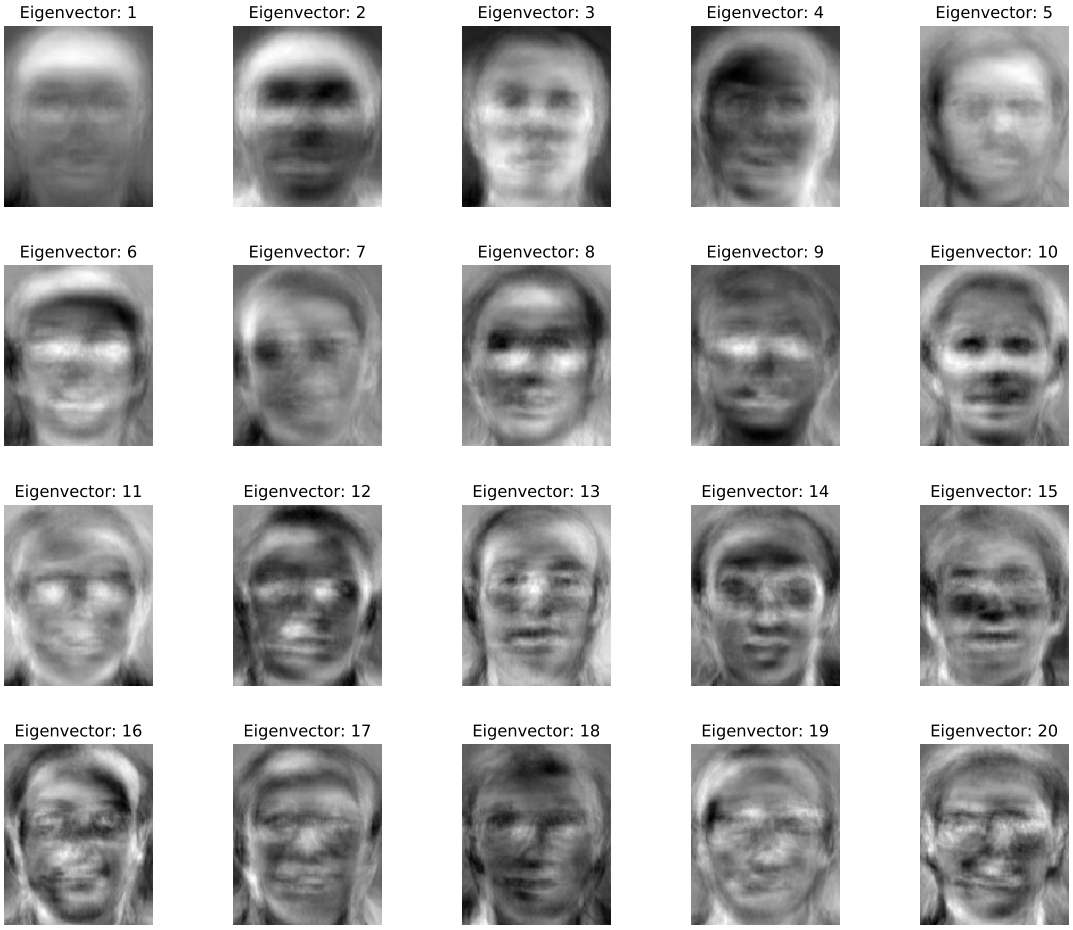


We now separate our data for train and testing in an 80:20 split, careful to make sure stratify over our labels to ensure all classes are evenaly represented in both training and testing.

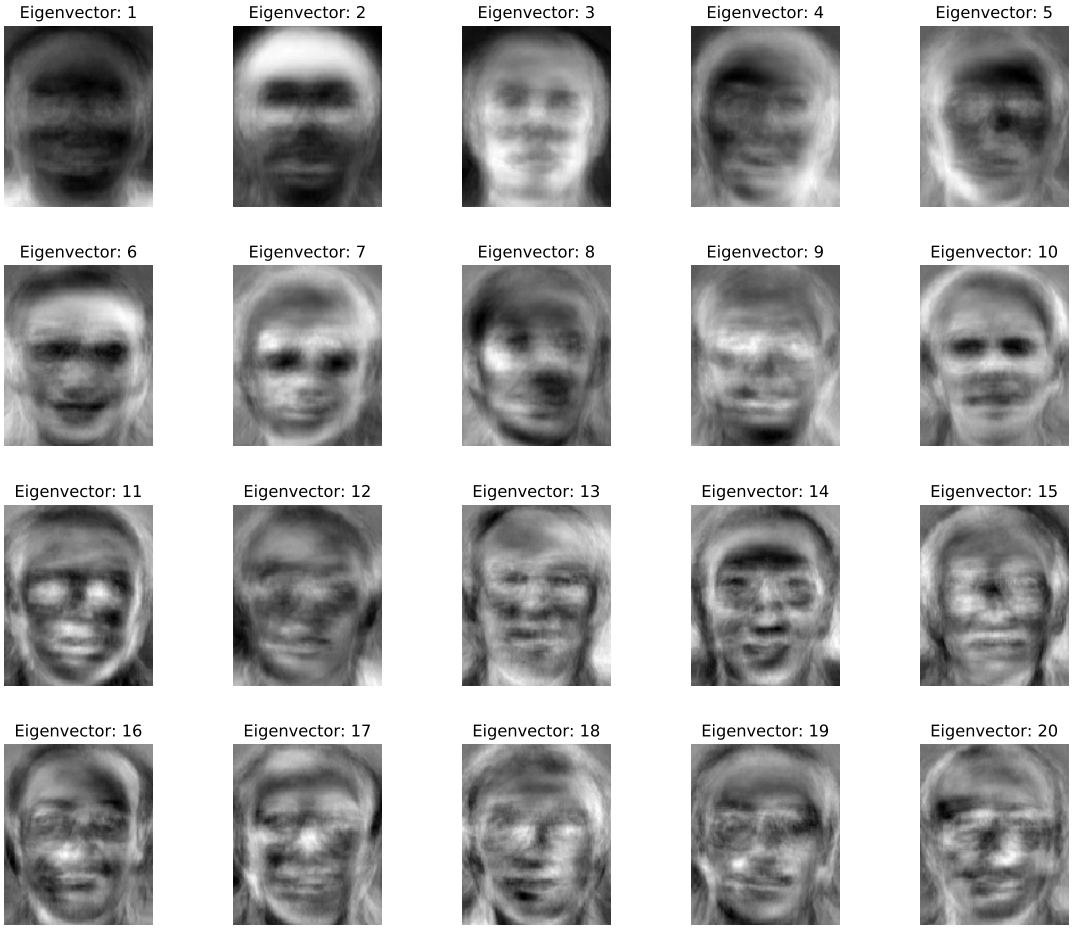
Next we fit a PCA model to both our original and standardized training data to determine the dominant eigenvectors for our PCA subspace.

```
X_std_pca.shape: (320, 320)
pca.components_: (320, 10304)
```

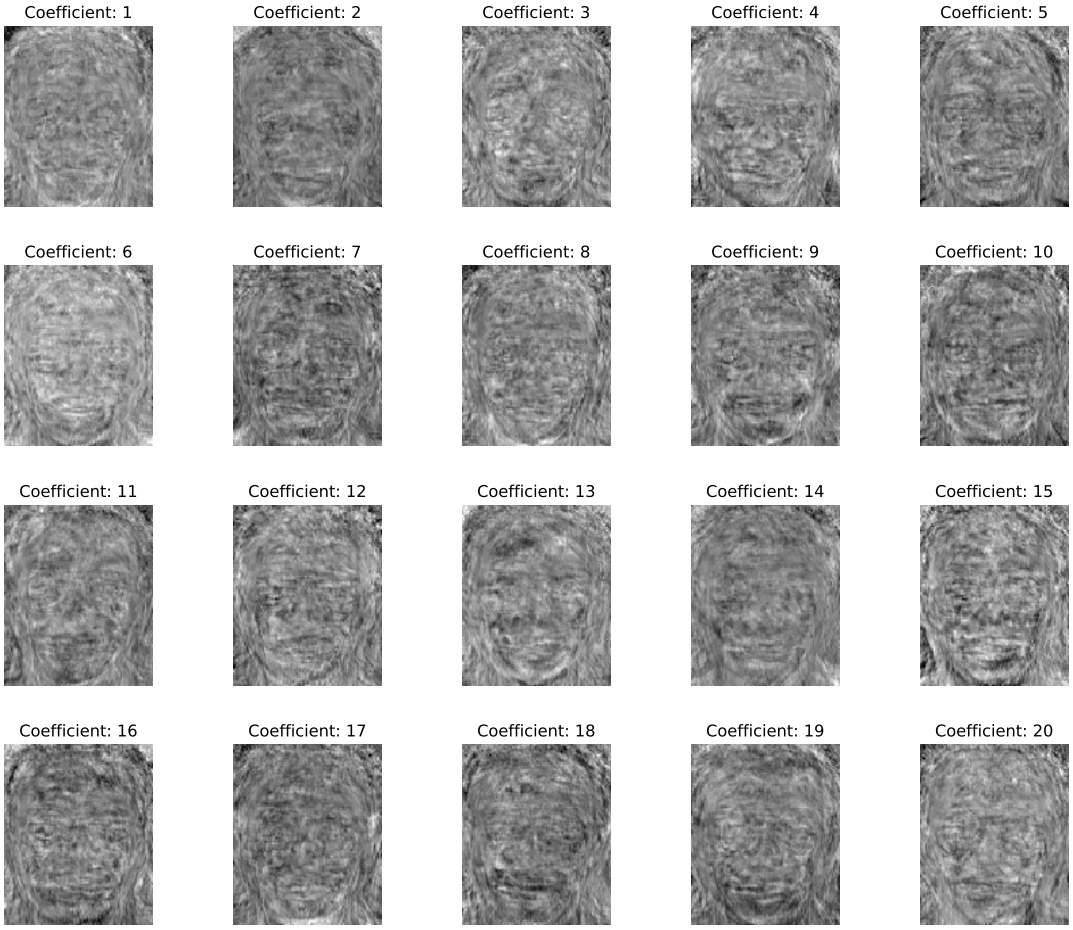
By extracting the eigenvectors from the fitted model, we can see what they look like in 2D by again reshaping the vecorts for pixel imagery. Below are the eigenvectors for the original data.



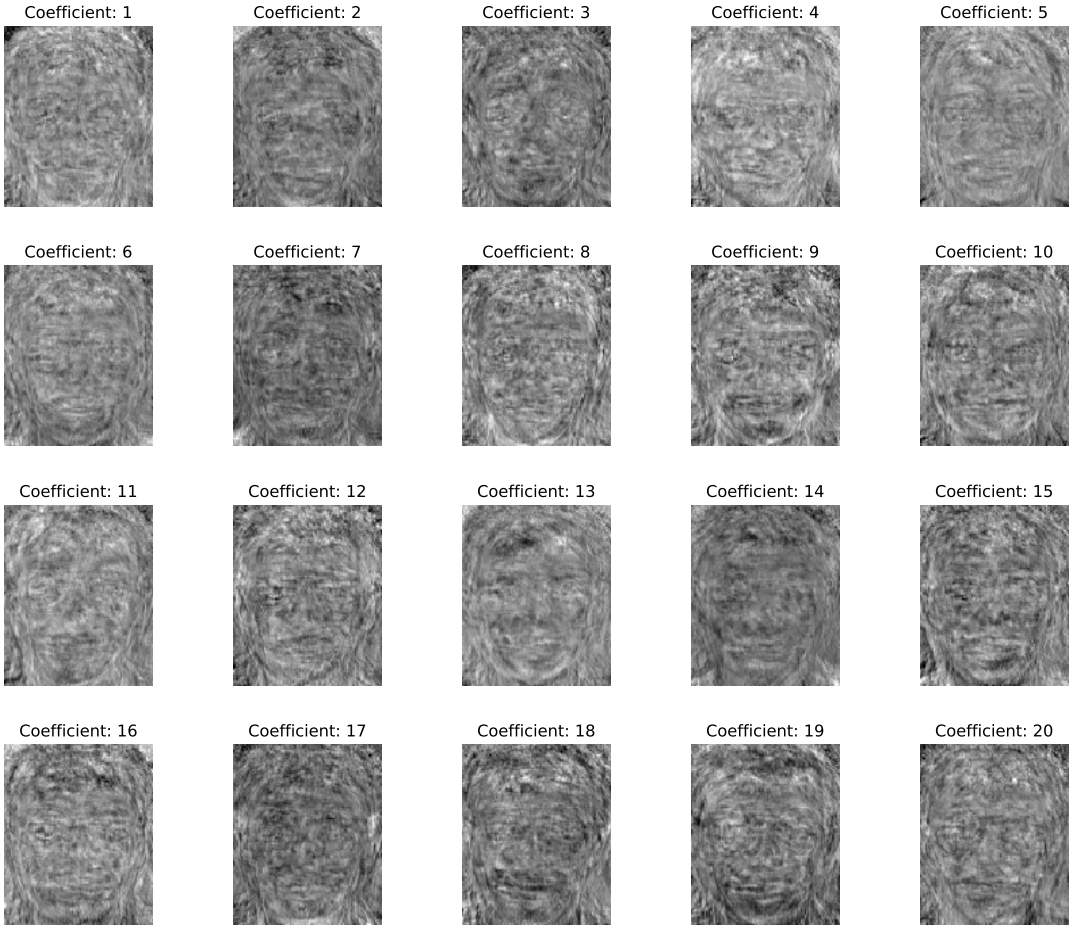
And below here we see the eigenvectors from the standardized data. Not much of difference for the first few dominate vectors other than shifted bias in grayscale, but smaller vectors seem swap around quite a bit.



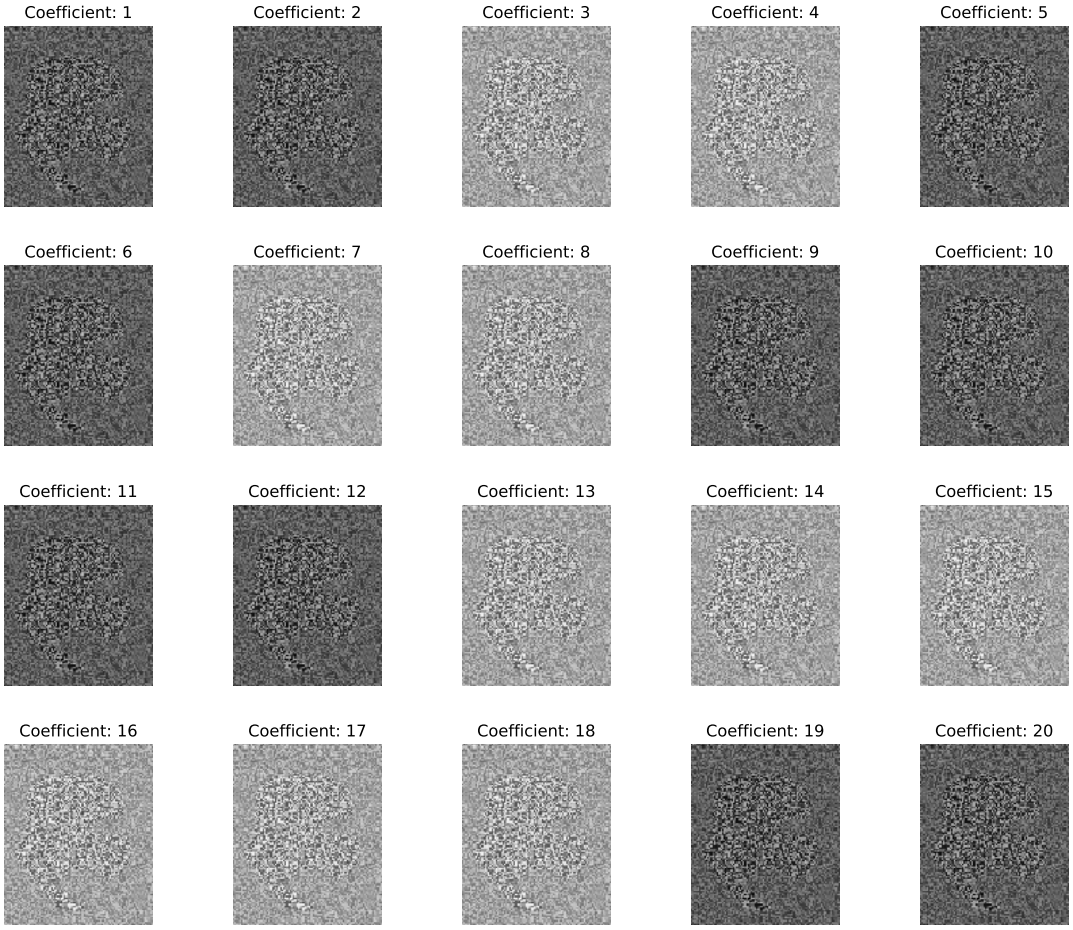
Similarly, we can fit an LDA model on to the original training data a view the coefficients learned.



Again, checking this for the standardized data renders just about the same results, where it appears that the LDA model is picking up on the frequency components of the images that separate the classes the best. Given the small sample sizes per class, I suspect that LDA may be overfitting here. Which leads us to another experimental avenue next.



Just for fun, let us also fit an LDA model on the transformed training data acquired from our first PCA model. We can see below that the LDA coefficients seem to be honeing onto the region of the image that most disittifies the class of the instance, that being the oblong area that contains the faces. It should be noted that proformaing the same method with the standardized data rendering nothing near as instreasing but instead for a few sparse dots, but is perhaps more or less functionally equivalent, but less pronounced.

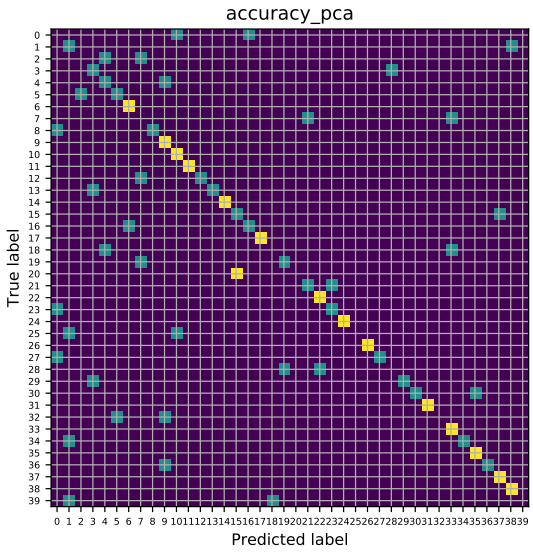


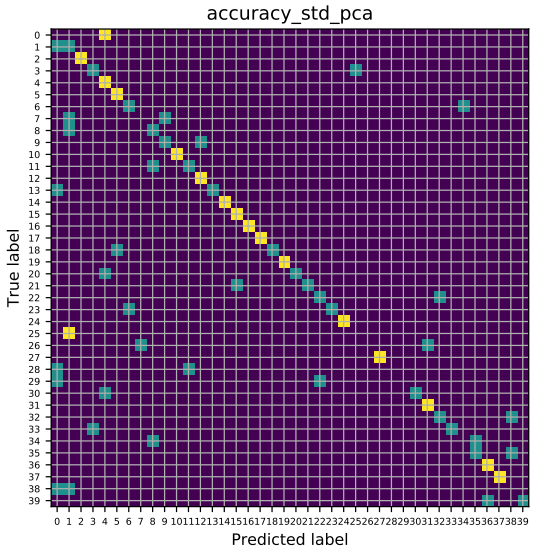
Now that we have fitted our subspace models, let us now attempt to train a classifier on the projected training data. This we'll use the simple random forest classifier setup for each subspace combination.

```
Out[21]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)
```

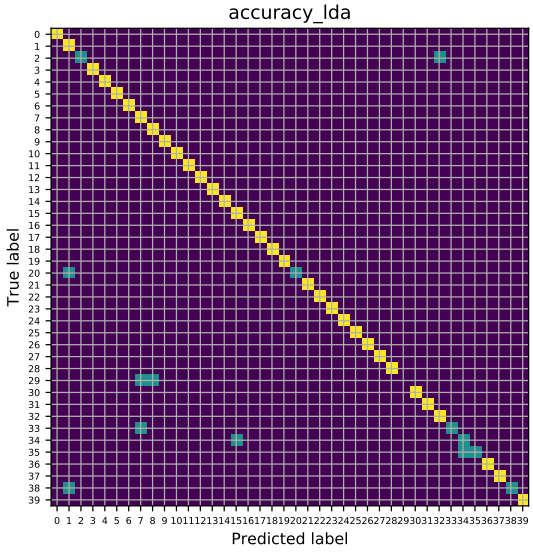
Next we'll transform the rest of our test data and predict using our list of classifiers.

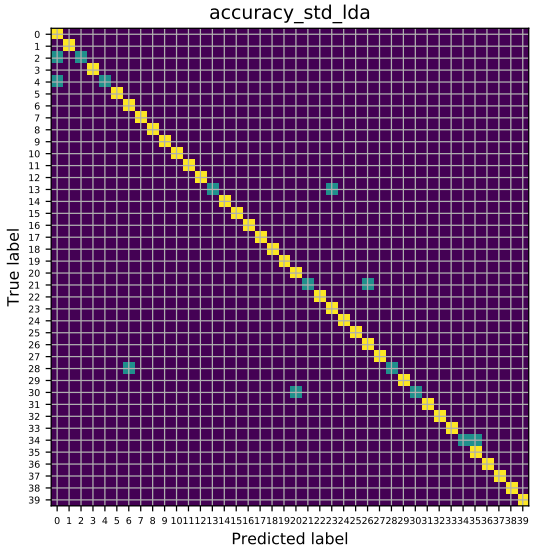
Finally, we can plot the confusion matrix over the number of classes to gain qualitative view the accuracy of the various approaches. First shown are the results for PCA with and without standardized data.



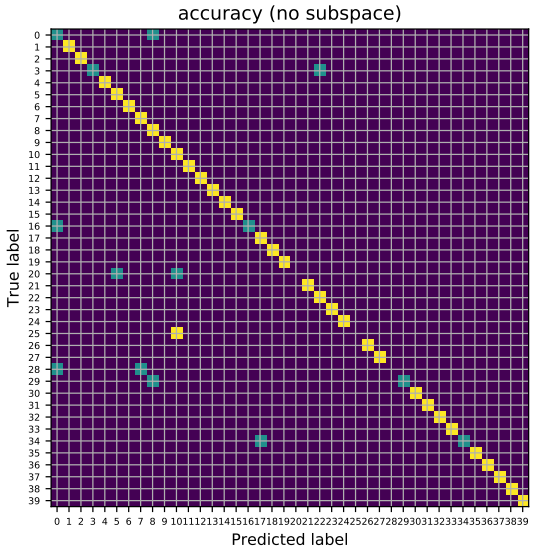


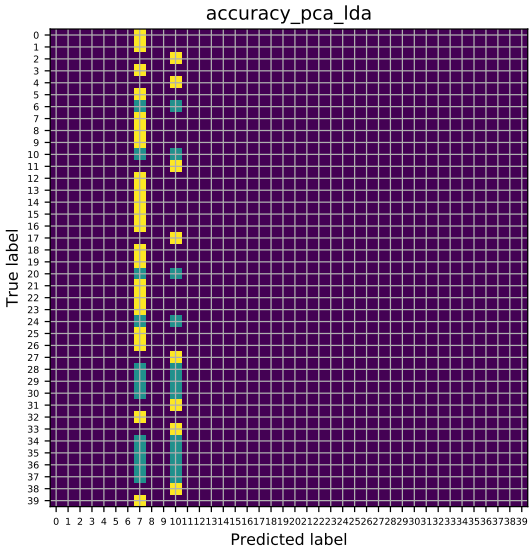
Next, show here is the same case but for LDA.





Lastly is the base case where the data is directly feed to the classifier v.s. our convoluted method of chaining LDA via a prior PCA transform with final random forest classifier fitting with the resulting cascaded output. As we can see this last experimental methods panned out be rather dismal. Additionally the stock methods without any subspace intervention only did slightly worse that the non standardized LDA approach previously.






```
<IPython.core.display.HTML object>
```

Approch	Accuracy
RFC PCA	0.5625
RFC PCA std	0.5875
RFC LDA	0.9
RFC LDA std	0.9125
RFC	0.8625
RFC std	0.85
RFC LDA PCA	0.0375

In summary, we can see that in most cases, standardizing the data did nothing to help the classification performance, as anticipated given the data was already of the same unit of grayscale intensity. Additionally, RFC LDA appear to be close to just feeding the data directly to the random forest classifier. This could say a few things about the scenario.

First, that data set is still quite small for modern PCs as opposed to systems back in 1994 when this att dataset was published, thus the bar or necessity for subspace methods has shifted dramatically further given that memory here is not much of an issue. Even with our LDA library using the pseudo inverse to handle the larger number of features vs instances, the majority of the descriptive power can still be ascertained the the RF classifier model. If the dimension of data was larger still, or the sparsity of samples in the feature space was worse, then this could warrant the use of feature space prepsoccessing.

One thing not measured here was the training and evaluation time. Given that the subspace transformation used here does not have an equivalent number of eigenvectors as compared to the dimension of the original feature space, we know the transformation to be inherently lossy. This inevitably leads to reducing the reasoning power from the training data or inference during evaluation. however if the size of the data set is too large, then reducing the computational complexity by reducing the dimension of the data can be a suitable compromise.

If I wanted to improve upon this further, I think I look towards alternative Subspace methods, in particular embedding that have been learned using neural networks such as with open face, a framework that first detects the face using simple har detectors, then proceeds to generate a canonical Viewpoint of the face by undistorting the projection of the face with respect to the optical frame, then takes the 2D patch of the face and passed it through the Deep learned embedding, such that each individual has maximum separation between other individuals while at the same time having the maximum cohesion among samples

of the same individual. From there you could use any kind of clustering algorithm that works well with varying sized but isolated clusters.

2.

> In robotics sound source localization is a frequent challenge. In the file http://www.hichristensen.com/demo-delay.wav estimate the delay between the two sound channels using FFT. provide an explanation of how you computes the delay between the two channels.
>
> For both questions provide a description of the approach adopted, the associated code and a description of your results.

To determine the delay between two signals, we're essentially searching for the phase offset between them. This can be done by checking the cross correlation of one signal with the other. More specifically, this can be done using the FFT by reformulating the convolution operation in the time domain into a single multiplication in the fourier domain. I.e. using the convolution theorem:

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

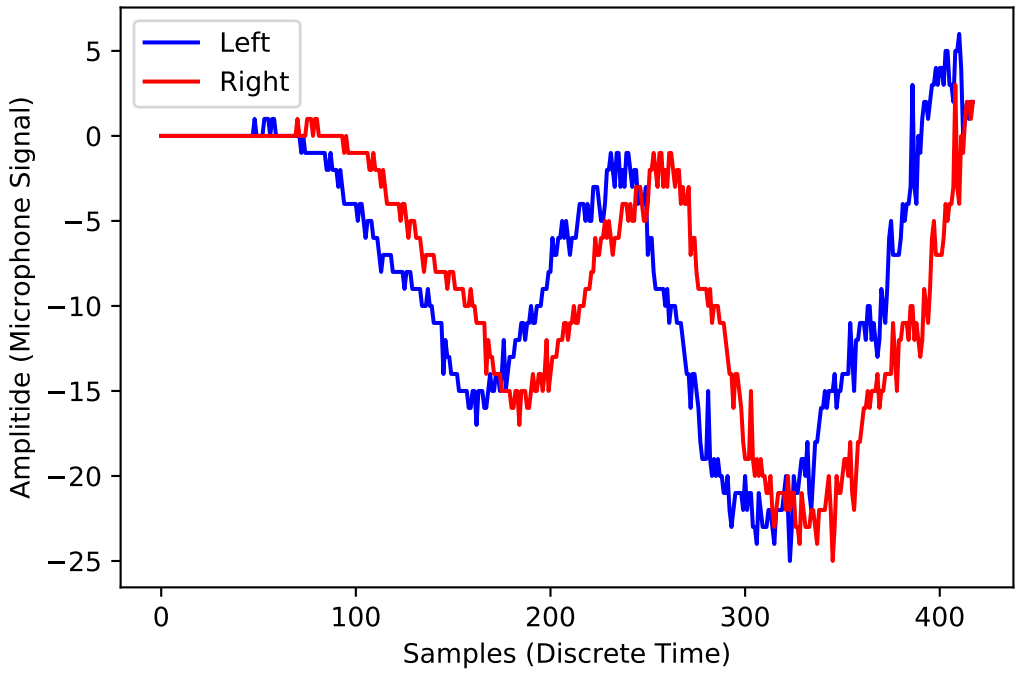
Thus, we can find the maximum response of the convolution between the two signals, i.e the counter shift rendering the signals in maximum alignment, by taking the argmax over the discrete inverse fourier transform of the product. From this we can determine k in $k \cdot dt$, where dt is the sample period for the discrete signal, k is the number of samples and thus $k \cdot dt$ is the total phase shift in time.

First we import some functions for reading in the WAV file, and calculating the forward and infevers FFT.

We then read in the file and split the two channels. From the stereo format of the file, we'll note the left channel to be the first, and the right channel to be the second, as is conventional in sound engineering and file formats.

From listening to the audio recording using a stereo headset, we can tell from the delay in arrival using our directional sense of hearing that the sound source is seems to originate from the left side from the perspective listener. If we take a look at the start of the signal closely, we can clearly see the subtle time shift with the right channel lagging behind the left.

This makes intuitive sense and matches with our sense of direction given that the time of arrival for a sound source on our left side would sound sooner in our left ear than our right. This ability for sub microsecond phase shift detection is what enables most binaural animals to localize sound sources, often coupled with servoing of the head to more accurately gauge bearing when sensing the change in phase shift over time and orientation.



Before processing and comparing the audio samples, it is generally a good idea to remove the DC component, as we don't yet exactly know how else the two signals may differ. Here we'll do so by removing the mean average respectively from both channels.

For microphone arrays separated by larger distances or some insulation that would otherwise result in greater signal attenuation in one recorded channel that the other, normalizing each channel by its standard deviation may also be a good idea. As it seems from the figure above that the two signals are of the exact same amplitude, the introduction of smaller floating point computation does not seem necessary in this case.

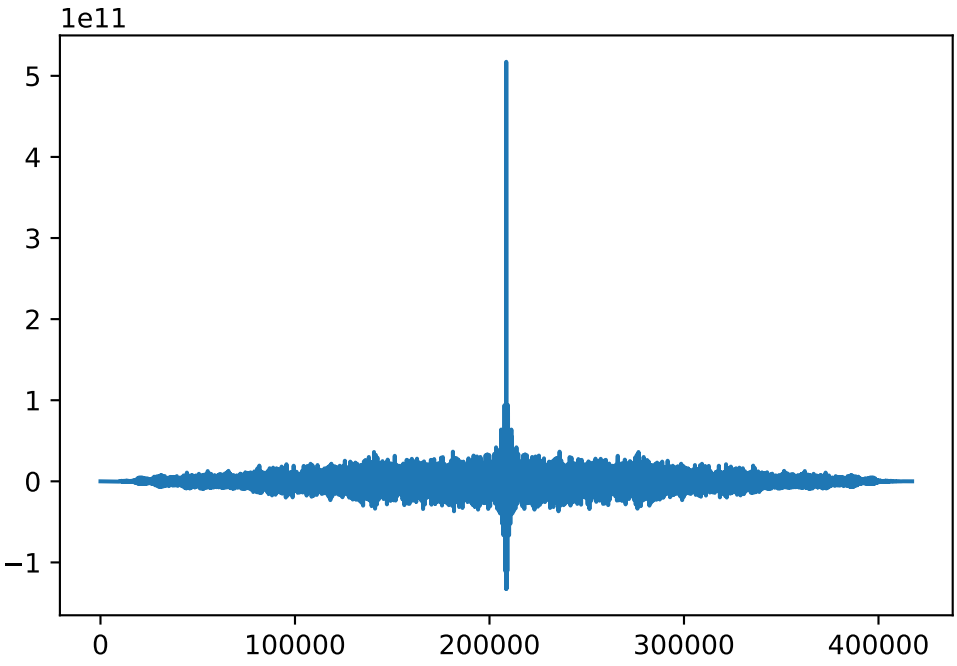
We'll use numpy to compute the N-dimensional discrete Fourier Transform for our real input signal. However, to ensure our fft computation is optimized for radix {2, 3, 4, 5}, we must find the next composite of the prime factors 2, 3, and 5 which are greater than or equal to target to pad with zeros, i.e. our maximum discrete sample size of our convolution operation, $|L| + |R| - 1$. This is also known as 5-smooth numbers, regular numbers, or Hamming numbers. We also compute the slice needed for later when shaving off the excess padding from our inverse operation.

Source: https://github.com/scipy/scipy/pull/3144

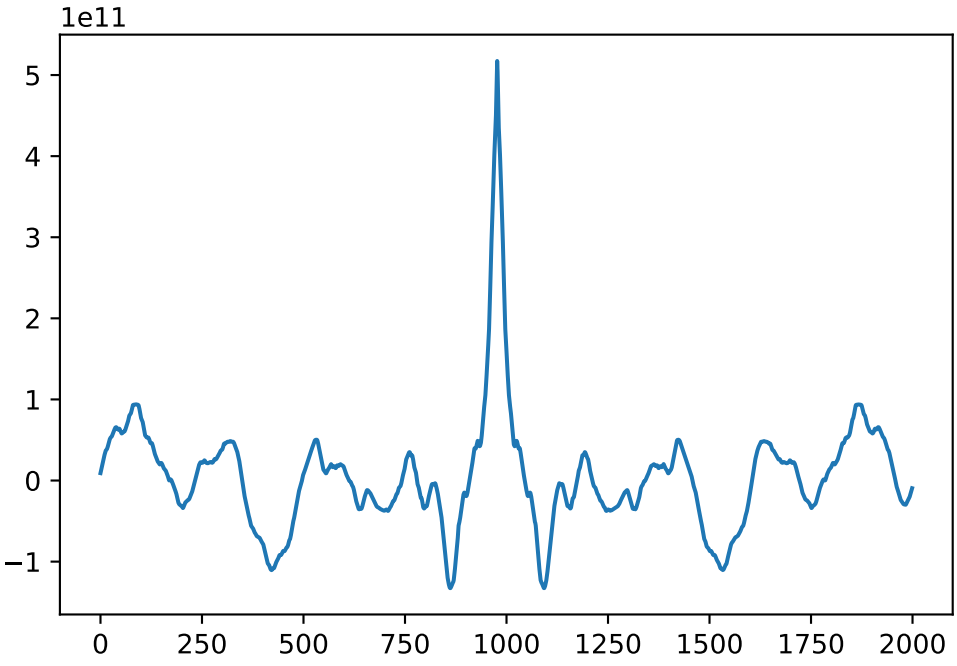
```
s1:  [208646]
s2:  [208646]
fshape:  [419904]
fslice:  (slice(0, 417291, None),)
```

Now we can finally compute the FFT for the left and right channels, we multiply them in the fourier domain, then recover the convolution by taking the reverse FFT of the product. Note that in order to compute the convolution correctly, we must first reverse the kernel, or the right channel in this case, before computing its FFT.

We can now plot the response. As shown below, we see the response grow from zero to some nominal noise floor as the kernel convolves over the left channel. From the sharp spike in response just past 200k on the x axis, we can tell that the signals are indeed quite similar, as they best aling at a narrow range in phase offset.



We can examine the response closer and see the residual harmonics in the sliding convictions, given the arbitrary waveform my happen to coincide to lesser extents at period lengths of other carrier frequencies of the audible voice sample.



Now, if we use the sample frequency as declared in the metadata of the file, $44.1kHz$ in this case, to derive the sample period dt, we can multiple this with k to determine the time shift. Note that we can't

really speak of the phase angle, as we are working with arbitrary waveforms that are not composed of a single frequency, thus a single angle would not make sense.

```
Recovered shift (samples):  -22
Recovered shift (seconds):  -0.000498866213152
```

We can do a simple sanity check by multiplying this time by the speed of sound, $343m/s$ in ambient air temperature to estimate a 'distance' between the two phased array microphones. We find this to be around $\sim 17cm$, which is an approximately on the scale for human ear separation, thus why the acoustic delay sound naturally from the left for us.

```
Array Displacement -0.171111111111
```

We can also check this by reversing the shift on the right channel as to realign with the left channel waveform. From the figure below, we see that $k = -22$ does indeed bring the two waveform back info perfect synchrony.