Assignment 5: Image Classification with Feature Vectors

Michael Shepherd, 19059019

October 22, 2019

Question 1

1a

This first thing that we need to do is to find the average vector $\underline{a}$ of the training set. We do this using the following formula with $\underline{f}_i$ being the i^{th} training image flattened into a vector:

$$\underline{a} = \frac{1}{n}\sum_{i=1}^{n} \underline{f}_i$$

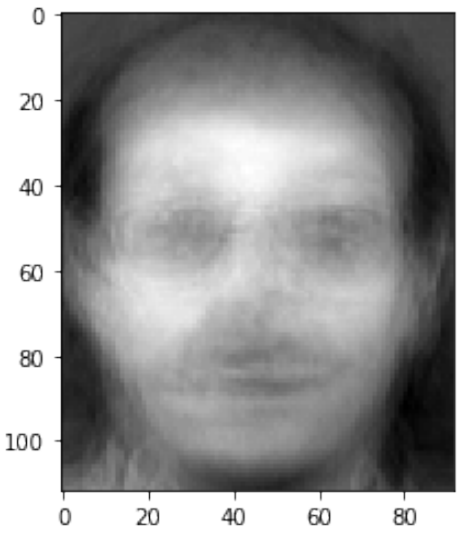


Figure 1: Average vector $\underline{a}$

We the use the average vector to find the $m \times n$ matrix X:

$$X = \frac{1}{\sqrt{n}}[\underline{x}_1 \cdots \underline{x}_n]$$

With:

$$\underline{x}_i = \underline{f}_i - \underline{a}$$

To decide what alpha value to use, we observer the singular values of X and attempt to observer a position where we will be able to keep as much useful information as possible, while still minimising the space that we use: We can

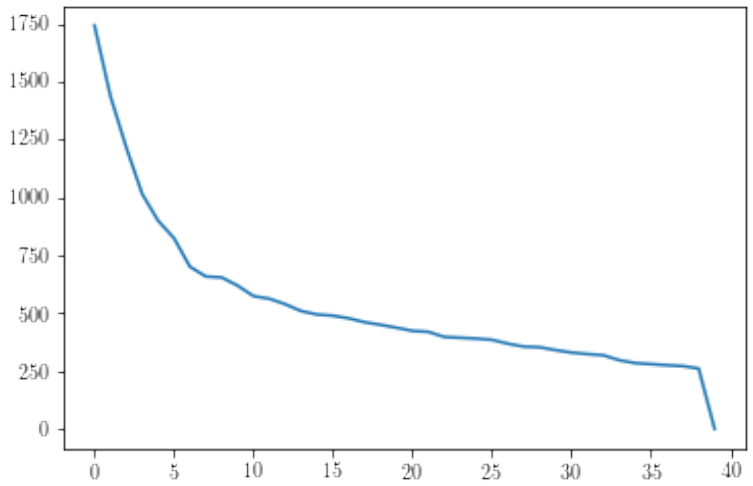
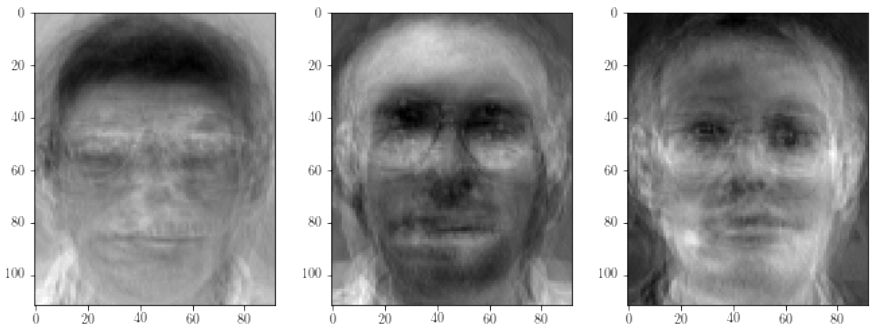


Figure 2: Singular values of X

observe that the graph of singular values starts to flatten at approximately the 15^{th} singular value, so we approximate α at 15 (note that we can find a more optimal α later when we observe the effects that it has on the accuracy of our simple facial recognition system). We can then find U_α by selecting the first α columns of the U in the SVD of X. Here we show the first 3 eigenfaces with $\alpha = 15$:



We displayed these images by re-scaling the eigenfaces from between the min and max values to between 0 and 255.

1b

To find the eigenface representations of an image $\underline{f}$, we solve for $\underline{y}$ in the over-determined system $U_\alpha \underline{y} = \underline{f} - \underline{a}$ in a least squares sense. Since the columns of U_α are orthogonal:

$$\underline{y} = U_\alpha^T(\underline{f} - \underline{a})$$

where The $\alpha \times 1$ vector $\underline{y}$ is a low dimensional representation of $\underline{f}$. We can then reconstruct $\underline{f}$ from its eigenface representation with the forlowing formula:

$$\hat{\underline{f}} = U_\alpha \underline{y} + \underline{a}$$

Using this formula to reconstruct the first three test images, we get:

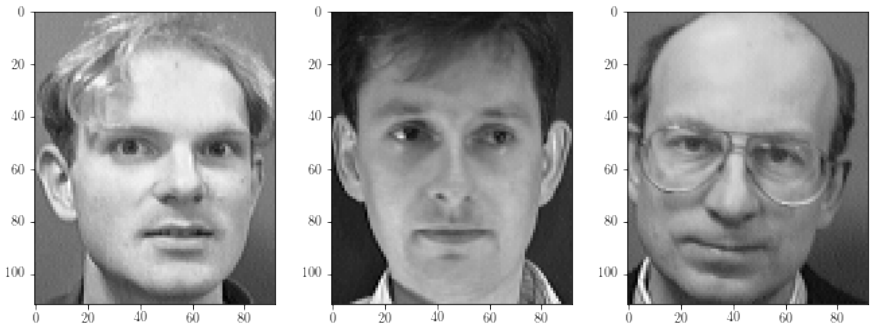


Figure 3: Test images

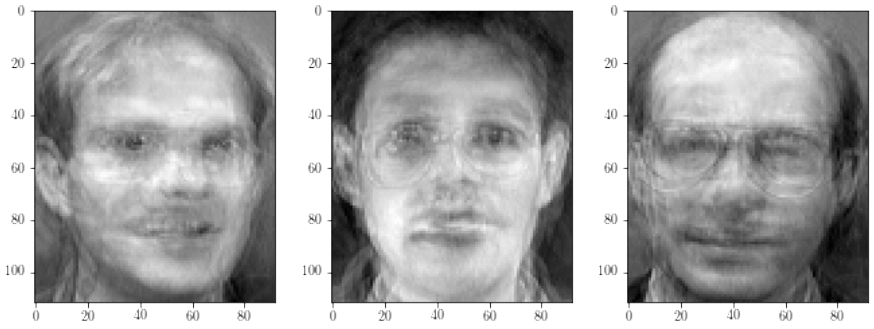


Figure 4: Reconstructions

As we can see, the re-constructions slightly resemble the original images, but they are clearly not exactly the same. This is expected as we have an extremely small training set, but it is effective enough to distinguish between the individuals.

1c

To find the nearest neighbour, we used the distance between each test image's eigenface representation and each training image's eigenface. At our original α value of 15, we found an accuracy of 77.5%. This did not feel like a very good accuracy, so to find a better accuracy, we experimented with different values of α and got the following information:

α	Accuracy %
5	66.25%
6	70%
7	72.5%
8	76.25%
9	75%
10	76.25%
11	75.25%
12	78.75%
13	77.5%
14	76.25%
15	77.5%
16	80%
17	77.5%
18	78.75%
19	78.75%
20	78.75%
21	78.75%

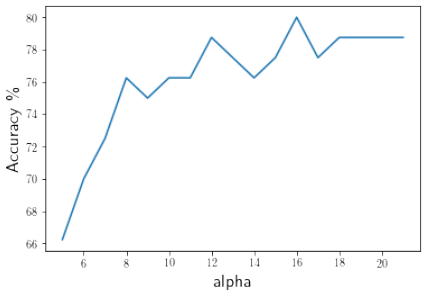


Figure 5: Accuracy and Alpha

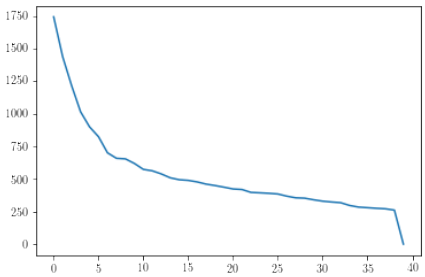


Figure 6: Singular values of X

As we can see, there is a relationship between the increasing of our α values and the accuracy of our system. This increase in accuracy also corresponds with the size of the singular values of X from earlier in the question. Using the accuracy graph, we can see that there is a point where there stops being a gain in accuracy from using a higher α value at $\alpha = 18$, but there is a more effective alpha value at $\alpha = 16$. We could have continued

increasing our alpha value, but after a certain point, we are not moving to a low enough dimensional subspace and might as well be comparing full images. Using our new optimised α value, we get matches such as these:

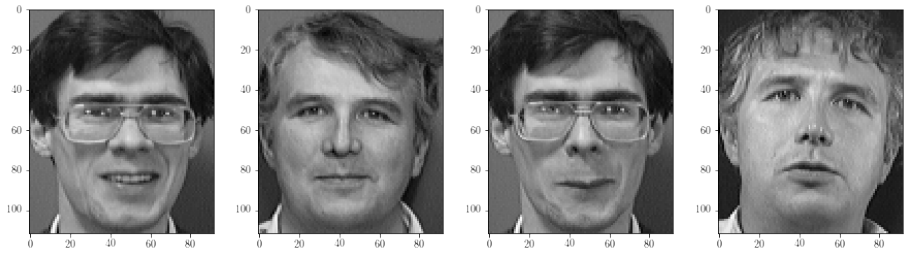


Figure 7: Test Images

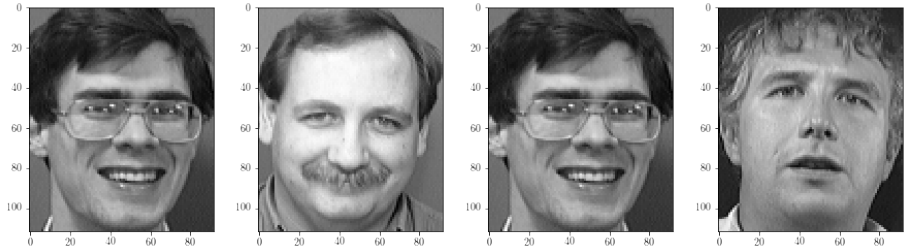


Figure 8: Training Image Matches

This shows us that even though the reconstructions may not look quite right to the human eye, the eigenface representations are sufficient enough to allow us to successfully match faces in the majority of cases. There are still some situations where faces looking in different directions (such as the incorrect match in the images above) or with completely different features will not be matched correctly, but for such a simple facial recognition system using α as only 16, 80% accuracy is very good. It is also noteworthy to say that at the higher α values, it is often the same test images that are being matched incorrectly, but are being matched to different incorrect training images at different α values.

Question 2

2a

To do this, I used the scikit-learn k-means clustering function on all of the provided SIFT feature descriptors. This allowed me to take the 295000 feature descriptors and map them each to one of the 50 cluster centres. The 50 128D cluster centres are our visual vocabulary.

We can now represent any image (in the training set or otherwise) as a nor-

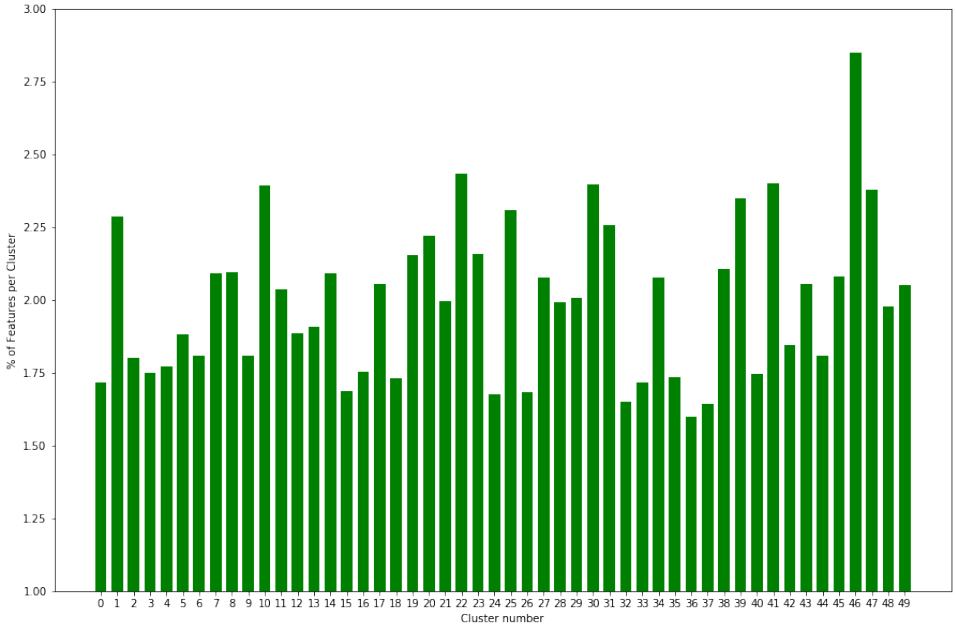
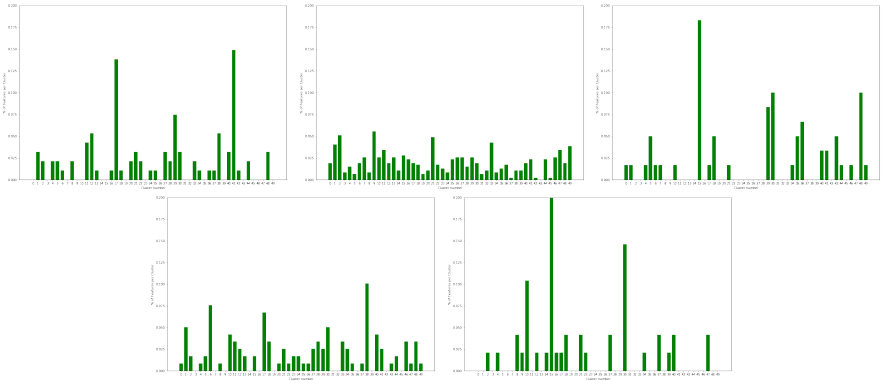


Figure 9: Cluster Centre Histogram

malized image histogram of these words. Below are the first five normalised histograms from the training set, as predicted by our k-means object.



2b

To answer this question, we needed to train a SVM for each pair of classes, which amounts to:

$$\frac{n\_class \times (n\_class - 1)}{2}$$

This means that we end up with 36 different SVMs so that we can classify with multiple classes. Each SVM votes on a class to assign to the test sample, and we pick the class with most votes for each image. Using the default LinearSVC function from Scikit-Learn's SVM library, I was able to get a 61% accuracy rate with class accuracies illustrated in the following normalised confusion matrix: This confusion matrix shows us that our classifications

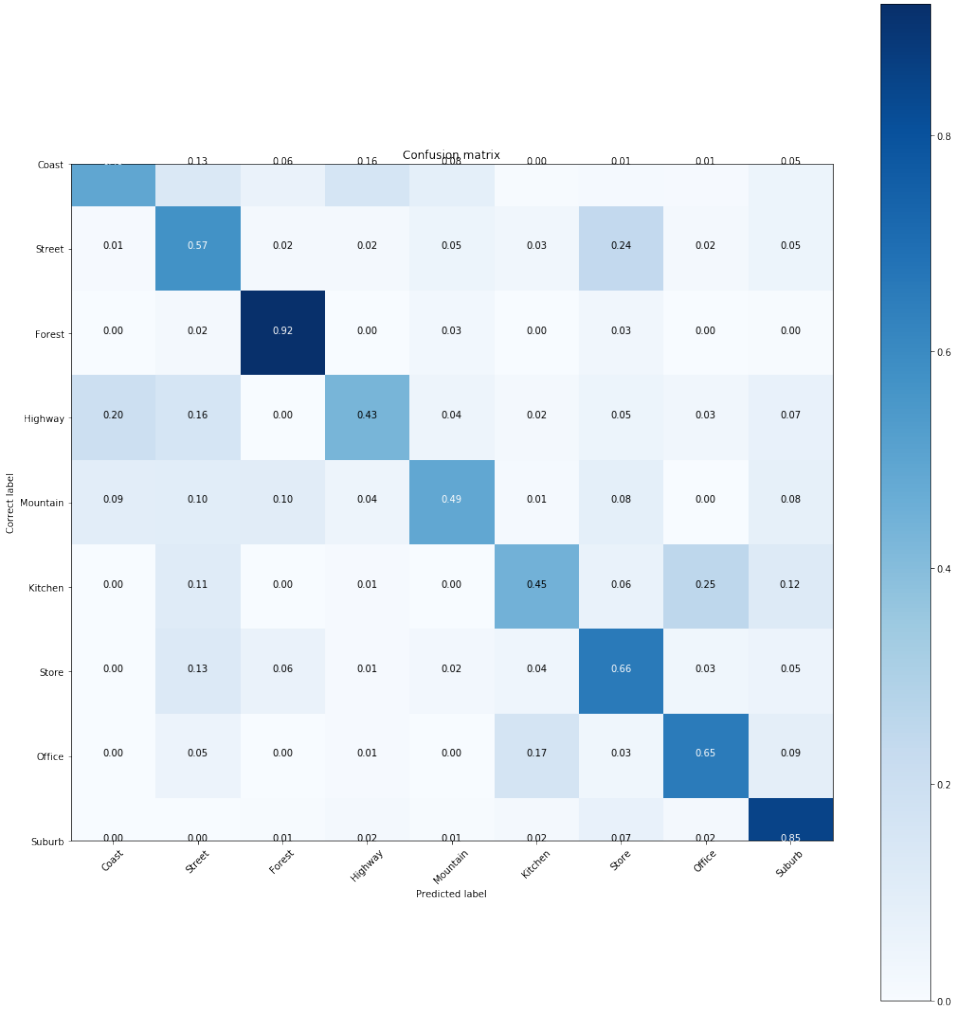


Figure 10: Confusion Matrix with Default Function

are mostly correct and that all of our classes are being predicted correctly

with above 40% accuracy. We do; however, have less than 50% accuracy for Coast, Highway, Kitchen and Mountain.

In a SVM we are searching for two things, a hyperplane with the largest minimum margin, and a hyperplane that correctly separates as many instances as possible. In an effort to try and better these scores, I increased the penalty parameter of the error term to 10. so as to seek out a hyperplane that seperates as many instances as possible. The logic behind this is well illustrated by this graphic from Kent Munthe Caspersen on Stackoverflow.

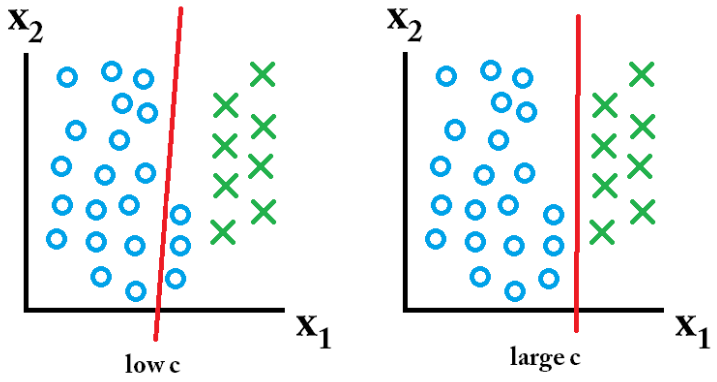


Figure 11: Illustrating the effect of large vs small penalty parameter of the error term

This Significantly increased the accuracy of the classifications for the classes of Highway, Kitchen and Mountain, but slightly decreased the accuracy of some other classes. This did bring our overall accuracy to 65% and increased all classes to more than 50% accuracy, which I believe makes this a fairly powerful classifier, given the small training sample size. Below, we can the the accuracy per class as well as the new and improved normalised confusion matrix.

Class	Accuracy %
Coast	50%
Street	58%
Forrest	92%
Highway	54%
Mountain	59%
Kitchen	51%
Store	68%
Office	75%
Suburb	84%
	65%

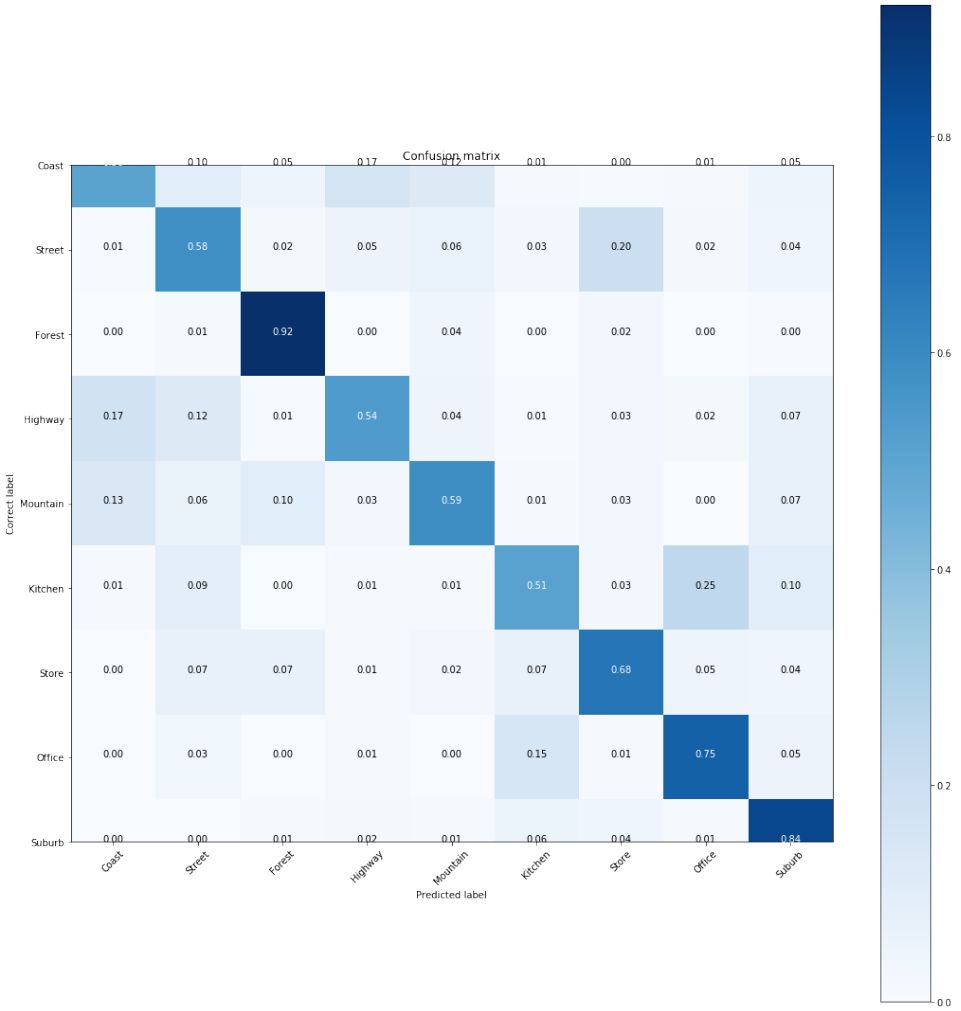


Figure 12: Confusion Matrix with Optimised Function

Below here, we can see some of the predictions that the classifier made. These classifications show that even though our classifier has a fairly high accuracy rate, It can make mistakes that are fairly obvious to the human eye. We could change the parameters of our classifier even further to try and iron out the errors that cause mis-classifications, such as the ones below, but since we do not know what kind of future images we can be expecting, it is advisable to keep the classifier as general as possible while reaching our target accuracy.

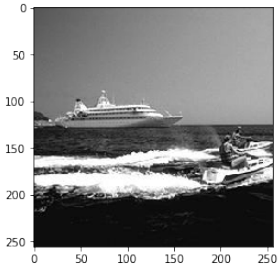


Figure 13: Correctly predicted as Coast

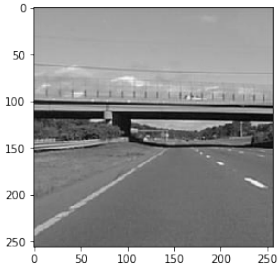


Figure 14: Incorrectly predicted as Coast

Figure 15: Correctly predicted as Suburb

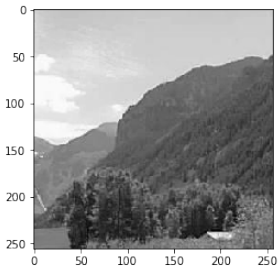


Figure 16: Incorrectly predicted as Suburb