Introduction to Data Science

COURSEWORK ASSESSMENT

Christopher White, 1908306

May 1, 2021

1 Question 1

Upon loading the `uspsdata.mat` dataset we can see the two matrices `uspstrain` and `uspstest`. Each matrix has 257 rows. The `uspstrain` matrix has 7291 observations and `uspstest` has 2007 observations. These observations are split into ten classes which are the digits from 0 to 9.

We can begin to split up the data into labels and features using the commands below.

```
training_labels = uspstrain(:,1);
training_features = uspstrain(:,2:end);

testing_labels = uspstest(:,1);
testing_features = uspstest(:,2:end);
```

The frequency of each class can be worked out by using the following commands.

```
tr_zeroes_ref = find(training_labels==0);
tr_zeroes_freq = length(tr_zeroes_ref);
```

Repeating the above for each class will give us a frequency value for each class which can then be collated into one vector.

```
training_freq = [tr_zeroes_freq, tr_ones_freq, tr_twos_freq, tr_threes_freq, tr_fours_freq...
    tr_fives_freq, tr_sixes_freq, tr_sevens_freq, tr_eights_freq, tr_nines_freq];
```

The above process can then be repeated for the testing set.

The frequencies of each class in both the training and test sets can be seen in figures 1 and 2. These figures are generated in `MATLAB` using the commands below.

```
figure();
pie(training_freq,[],string(training_freq));
legend('0','1','2','3','4','5','6','7','8','9');
title('Number Frequency in Training Set')
```

Figure 3 shows the visualisation of four random observations from the training set. This can be created in `MATLAB` using the following commands.

```
figure();
for i =1:4
    subplot(2,2,i);
    imagesc(reshape(training_features(randi(max(size(training_features))),:),16,16)');
end
```

A random image is selected using the `randi` function. This image is reshaped using the `reshape` function into a sixteen by sixteen matrix. In order to get the orientation of the image correct, the transpose of this matrix is taken. Finally, it is visualised using the `imagesc` command.

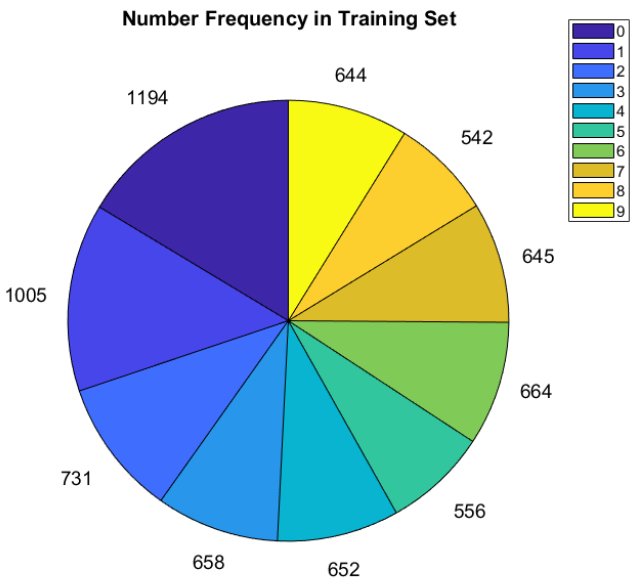

Figure 1: *Pie chart showing the frequency of each class in the training set.*

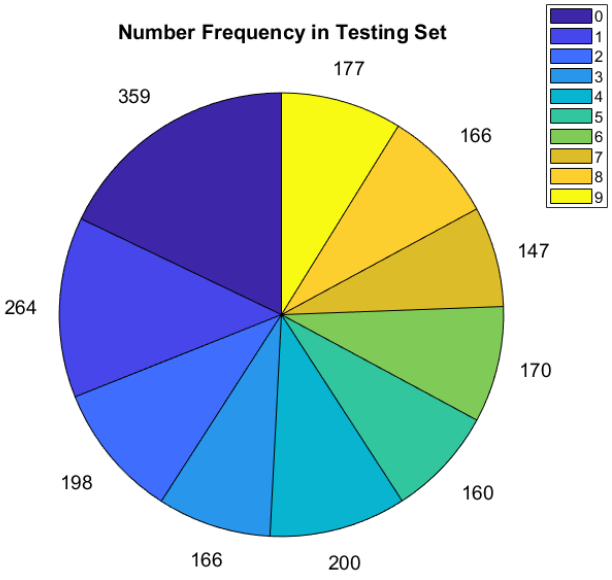

Figure 2: *Pie chart showing the frequency of each class in the testing set.*

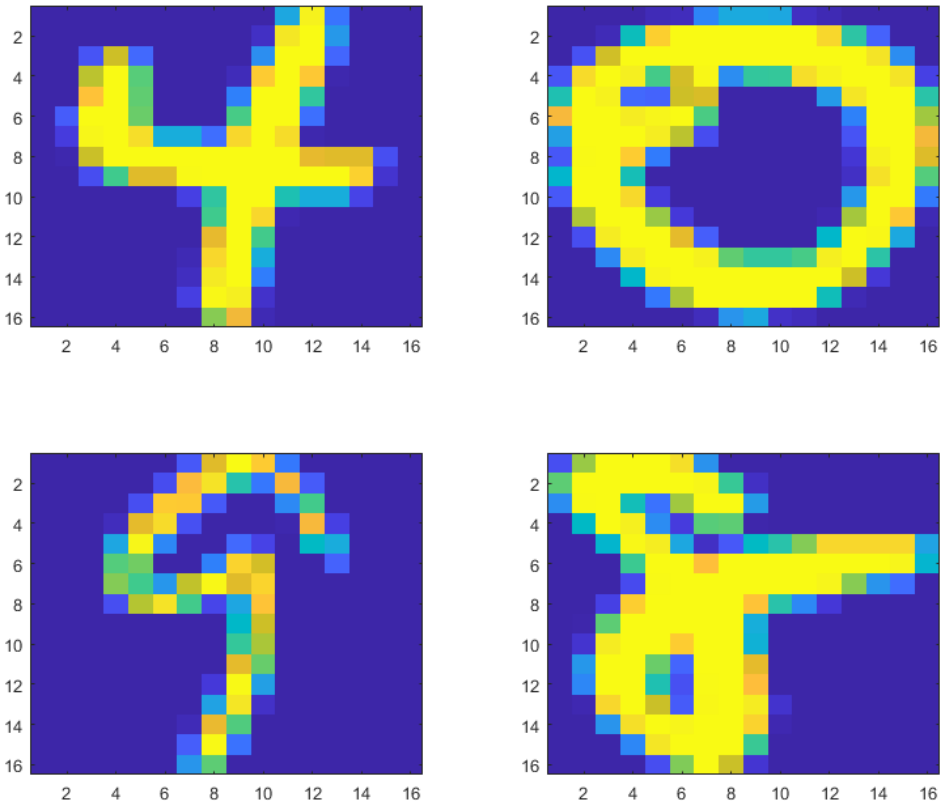

Figure 3: *A visualisation of four random numbers from the training set.*

2 Question 2

A principle components analysis (PCA) can be carried out on the data set. This is done by using the MATLAB commands below.

```
[coeff,score,latent] = pca(training_features);
```

The PCA scores output from this function can then be plotted into a two dimensional space.

```
figure();
plot(score(tr_zeroes_ref,1),score(tr_zeroes_ref,2),'*','MarkerSize',9);
hold on
plot(score(tr_ones_ref,1),score(tr_ones_ref,2),'*','MarkerSize',9);
plot(score(tr_twos_ref,1),score(tr_twos_ref,2),'*','MarkerSize',9);
plot(score(tr_threes_ref,1),score(tr_threes_ref,2),'*','MarkerSize',9);
plot(score(tr_fours_ref,1),score(tr_fours_ref,2),'*','MarkerSize',9);
plot(score(tr_fives_ref,1),score(tr_fives_ref,2),'*','MarkerSize',9);
plot(score(tr_sixes_ref,1),score(tr_sixes_ref,2),'*','MarkerSize',9);
plot(score(tr_sevens_ref,1),score(tr_sevens_ref,2),'k*','MarkerSize',9);
plot(score(tr_eights_ref,1),score(tr_eights_ref,2),'*','Color', pink,'MarkerSize',9);
plot(score(tr_nines_ref,1),score(tr_nines_ref,2),'*','Color',teal,'MarkerSize',9);
legend('0','1','2','3','4','5','6','7','8','9');
```

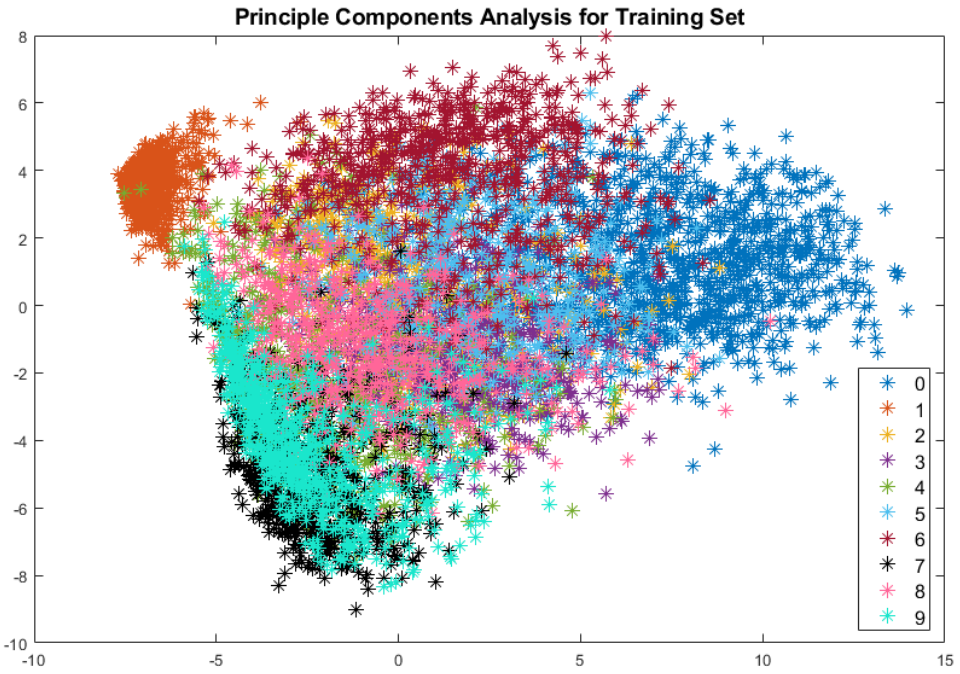

Figure 4: *Visualisation of the datapoints from the training set using Principle Components Analysis. The points from each class are coloured differently, shown in the legend.*

The two dimensional visualisation from the principle components analysis of the training set can be seen in figure 4.

3 Question 3

3.1 3A

The training data can be clustered using the `kmeans` function and the result can be cross-tabulated with the training labels using the commands below. A number of clusters of ten was chosen since there are ten classes in the data set.

```
[cluster_id, centroids] = kmeans(training_features,10);
crosstab(cluster_id, training_labels);
```

```
ans =

   0    0   15   10   90    8    0  596    9  492
 509    0   10    1    2    1   21    0    5    0
   0 1001    5    2   43    0    9    7   14   16
   4    0  539   16    7   10   26    2    6    0
  13    0   65    5  496   35   18   33   15  126
   9    0   47  549    0   13    0    0   34    2
  21    1    4   28    0  395   72    1    7    0
 515    0    3    2    0    5   26    0    2    0
   7    3   39   43    7   33    2    6  448    8
 116    0    4    2    7   56  490    0    2    0
```

The code below can be used to calculate the accuracy of the categorisation using the matrix

above.

```
>> sum(diag(ans))/sum(sum(ans))
```

This run of the kmeans algorithm had a final accuracy of 0.1440. The `kmeans` function in `MATLAB` has different distance measures with which the kmeans algorithm can be carried out. The code for doing so with four different distance measures can be seen below, along with the cross-tabulation of the kmeans results with the training labels.

```
[cluster_id, centroids] = kmeans(training_features,10, 'Distance', 'sqeuclidean');
sqeuclidean_crosstab = crosstab(cluster_id, training_labels);

[cluster_id, centroids] = kmeans(training_features,10, 'Distance', 'cityblock');
cityblock_crosstab = crosstab(cluster_id, training_labels);

[cluster_id, centroids] = kmeans(training_features,10, 'Distance', 'cosine');
cosine_crosstab = crosstab(cluster_id, training_labels);

[cluster_id, centroids] = kmeans(training_features,10, 'Distance', 'correlation');
correlation_crosstab = crosstab(cluster_id, training_labels);
```

```
sqeuclidean_crosstab =

 13     0    49     8   490    43    15    35    14   136
  0     0    14    10    92     9     0   596    11   483
 16     0    36   544     0   300     1     0    35     2
 11     0     2     0     3    16   322     0     0     0
  0  1001     5     1    45     0     5     7    14    16
519     0     3     2     0    11    11     0     2     0
  4     3    46    68     6    64     1     4   445     7
507     0    10     1     0     5     3     0     4     0
  4     0   561    12     9    10    20     2     8     0
120     1     5    12     7    98   286     1     9     0
```

Squared Euclidean Accuracy = 0.0156.

```
cityblock_crosstab =

 18     1   274    17   275    33    46    94    42   188
504     0    36     5     1    16    41     0    11     1
142     0    20     1     8   135   508     0     7     1
  2  1002    67     5    72     9    22     5    26    14
500     0     7     2     1     7    42     0     2     0
 11     0    55   397     0    86     1     0    43     2
  3     0   150     7    20    12     0   520    11   312
  2     0     4     1   275    13     0    22    10   116
  4     2   117    17     0    19     1     4   378    10
  8     0     1   206     0   226     3     0    12     0
```

Cityblock Accuracy = 0.0727.

```
cosine_crosstab =

 10     0    27   539     0   265     0     0    26     1
506     0     3     2     1     9    43     0     2     0
  7     1    39    73     3    69     1     2   441     5
  0  1003    12     3    39     1    13     4    16    19
  0     0    19     9    82    11     0   574    11   453
  0     0   436     6     0     1     3     2     2     0
  0     0     5     2   285    14     0    37     5   142
126     0     8     1     6   123   517     0     3     0
 25     1   171    21   235    45    51    26    29    24
520     0    11     2     1    18    36     0     7     0
```

Cosine Accuracy = 0.0225.

```
correlation_crosstab =
```

```
519    0   13   13    2   27   65    0    2    1
  0 1003    5    1   53    1    9    3   12    6
  0    0  442    8    2    4    1    4   15    1
  0    0    3    4  363   25    0   23   14  151
  0    0   18    5   16    7    0  523    6   88
544    0   19    3    3   16   79    2    7    1
 16    0  169    0   62   16   12    0    0    0
  2    2   16   38  139   21    2   90  406  394
  4    0   37  586    0  313    0    0   69    2
109    0    9    0   12  126  496    0   11    0
```

Correlation Accuracy = 0.2978.

It can be seen that using correlation distance, when carrying out the kmeans algorithm, results in the greatest accuracy, significantly higher than the other distance measures.

The centroids used in the kmeans algorithm can be visualised using the code below. The figure that is output from this can be seen in figure 5.

```
figure();
[cluster_id, centroids] = kmeans(training_features,10);
for i=1:10
   subplot(2,5,i);
   imagesc(reshape(centroids(i,:),16,16)');
end
```

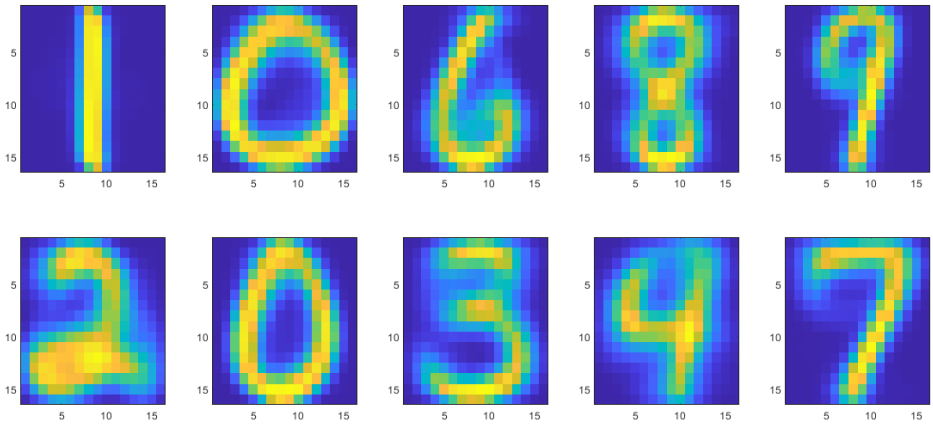

Figure 5: *Visualisation of the centroid for each cluster.*

3.2 3B

The kmeans algorithm is heuristic and works by finding local minima to the optimisation problem that results from trying to minimise the residual sum of squares. Since the algorithm does not find the global minimum, on different runs it may find different local minima resulting in different results from identical initial conditions.

```
[cluster_id_1, centroids] = kmeans(training_features,10);
[cluster_id_2, centroids] = kmeans(training_features,10);
crosstab(cluster_id_1, cluster_id_2);
```

```
ans =
    0    0    1    1    0    4    3    0  224  587
    0    1  639    1    0    5   10    0  324    0
    0    0    0   28  552    1    0  541    0    0
    0    1   14    0    2    1  905    2    7    4
    0  465    0    0    3   25    0    4    2    1
  687    0    0    0    5    1    0    0    0    0
  351    1   10    0   21    3    0    0    0    0
    1   40   37   12  411    9  122    3    3    1
    6    2   67   12   82  510    6    6    0    0
    0    0    0  465    8   13    0   38    0    0
```

By using the code above we can quantify the similarity of two consecutive runs of the algorithm. We can see that when cross tabulating two successive runs they only share 0.11% of the labels assigned by the algorithm. This shows that two successive runs can give very different outcomes.

4 Question 4

Firstly, a random subset of the training set can be taken using the code below. It is chosen to be the same size as the ones class.

```
random_sample_refs = randperm(max(size(training_features)),max(size(tr_ones)));
random_sample = training_features(random_sample_refs,:);
```

The pair-wise distance between every pair of points in both subsets can be taken using the code below.

```
pdist_ones = pdist(tr_ones);
pdist_rand_sample = pdist(random_sample);
```

This pair wise distance can be plotted as a histogram using the code below. The output can be seen in figure 6.

```
figure();
histogram(pdist_ones,'FaceColor','r');
hold on
histogram(pdist_rand_sample,'FaceColor','b');
legend('pdist for Ones','pdist for Rand Sample');
xlabel('Pair-Wise Distance')
ylabel('Frequency')
title('Comparison Between Pair-Wise Distances for Random Sample and Ones Class')
```

It can be seen from the figure that the ones class has much lower distances between its points than the points selected to be in the random subset.

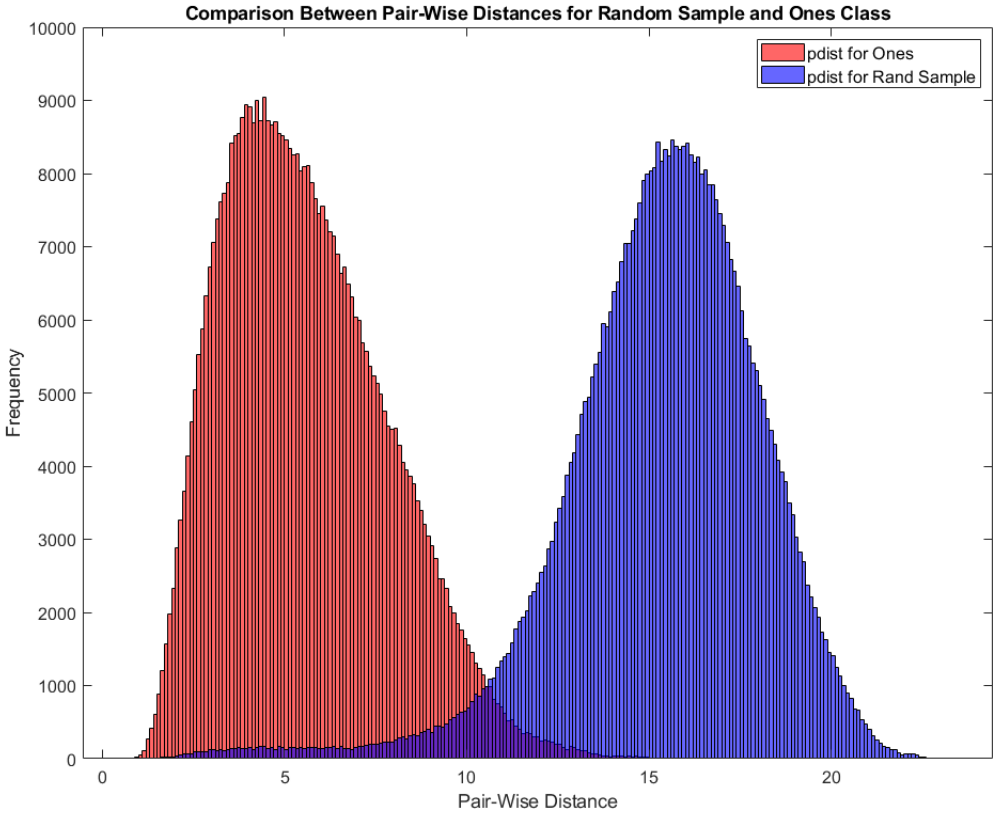

Figure 6: *Histogram showing the pair-wise distances for the ones class and a random subset of the same size from the training set.*

The mean squared distance (MSD) can be calculated by taking the mean of the pair-wise distances calculated previously.

```
MSD_ones = mean(pdist_ones);
MSD_rand_sample = mean(pdist_rand_sample);
```

The mean squared distance for the ones class is 5.5878 and for the random subset it is 15.3636.

The code below is used to generate a series of random subsets and plot their MSDs as a histogram. The output of this can be seen in figure 7. From this figure it can be seen that a random subset of the data is likely to have a MSD of around 15. This is significantly larger than the MSD for the ones class. Therefore, we can conclude that the observations in the ones class are much closer together (and much more similar) than the average spread of points throughout the data set.

```
for i = 1:50
    random_sample_refs = randperm(max(size(training_features)),max(size(tr_ones)));
    random_sample = training_features(random_sample_refs,:);
    pdist_rand_sample = pdist(random_sample);
    MSD_rand_samples(i) = mean(pdist_rand_sample);
end

histogram(MSD_rand_samples,'FaceColor','b');
```

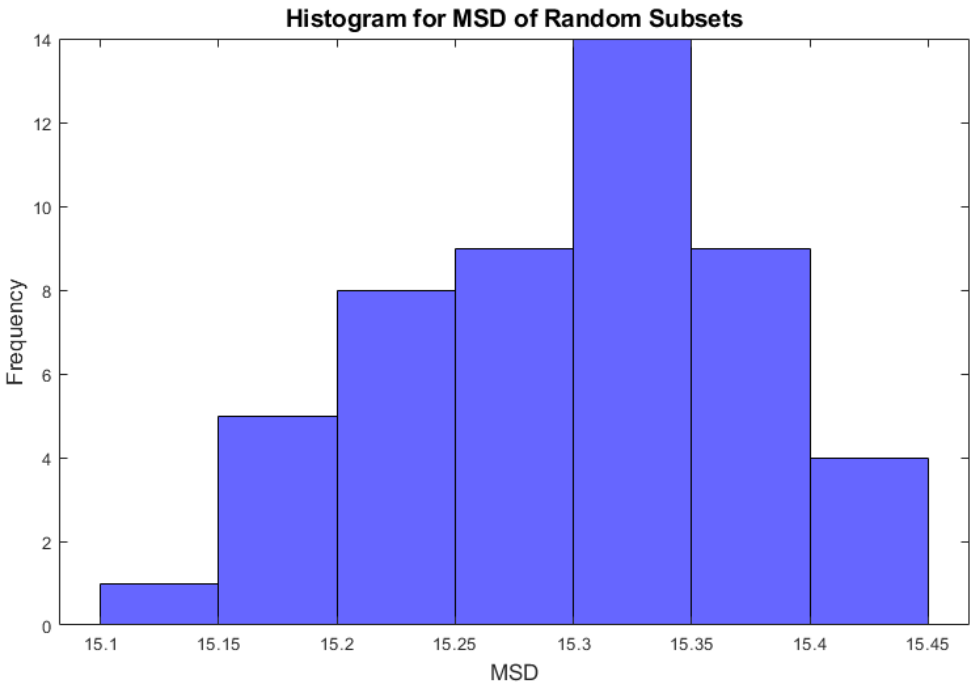

Figure 7: *Histogram showing the MSD of 50 random subsets of the training set.*

5 Question 5

In order to separate the fives class from the other observations, the labels for the other classes first need to be changed. This is done using the code below.

```
two_class_training_labels = training_labels;
two_class_training_labels(find(two_class_training_labels ~= 5)) = 1;

two_class_testing_labels = testing_labels;
two_class_testing_labels(find(two_class_testing_labels ~= 5)) = 1;
```

Then a decision tree can be trained using the training set and predictions can be made on the testing set. This is done using the code below.

```
tree_two_class = fitctree(training_features, two_class_training_labels);
ts_Treepredictions_two_class = predict(tree_two_class, testing_features);
```

The effectiveness of this categorisation technique can be assessed by creating a confusion matrix using the command below.

```
two_class_ts_Treeconmat = confusionmat(ts_Treepredictions_two_class,two_class_testing_labels)
two_class_ts_Treeconmat =

       1801         50
         46        110
```

This confusion matrix suggests that the fives class has been separated well, with an accuracy of 95%.

The above process can be recreated using a different type of categorisation. A source vector

machine (SVM) can be used by using the follow code.

```
SVMhyperplane = fitclinear(training_features,two_class_training_labels,'Learner','svm');
ts_SVMpredictions = predict(SVMhyperplane,testing_features);
```

Again, a confusion matrix can be computed to assess the effectiveness of this technique.

```
two_class_ts_SVMconmat = confusionmat(ts_SVMpredictions,two_class_testing_labels)
two_class_ts_SVMconmat =

       1819         23
         28        137
```

The SVM has been found to have an accuracy of 98%. This result is greater than the accuracy obtained by using a decision tree. This can also be seen from the confusion matrices, where the SVM confusion matrix shows fewer wrongly classified observations (the elements not in the leading diagonal).

6 Question 6

The process outlined in question 5 can be repeated using a different type of classifier, the K Nearest Neighbour algorithm. The classifier can be trained on the training set and tested on the test set using the code below.

```
KNN = fitcknn(training_features,two_class_training_labels);
ts_KNNpredictions = predict(KNN,testing_features);
```

To assess the performance of this method, a confusion matrix can be calculated as below.

```
ts_KNNconmat = confusionmat(ts_KNNpredictions,two_class_testing_labels)
ts_KNNconmat =

       1837         15
         10        145
```

This classification method is found to have an accuracy of 99% when used on this data set. This makes this the best method out of the three trialled.