# Advanced Statistical Machine Learning
# Assignment #1

Robert Webb, (rjw14),
Imperial College London

February 23, 2015

## 1 Introduction

In this coursework, I implemented several projections: PCA, PCA-Whitening, LDA, LPP and ICA (Fast ICA). They were then used to pre-process data provided as input to facial recognition tasks using the PIE and YALEB corpora as training/testing data. To evaluate the transformations' performance I recorded the error rates (using the provided protocol.m file). The performance of the various algorithms at various levels of dimensionality reduction was also compared.

## 2 Methods

In this section, I explain how the various methods work, in my own words. They are pre-processing methods that return a linear transformation (a matrix), mapping the input data to a new vector space (latent space). These methods all allow for dimensionality reduction, which involves retaining the most 'important' dimensions in the new space and discarding the dimensions that do not add much information to the data. Lower-dimensional data is easier to store and process (since it is smaller) and can better model the 'intrinsic' dimension of the data (i.e. the dimensionality of the thing that is really being modelled) [Yu L].

### 2.1 PCA - Principal Component Analysis

Given some input data, this method returns a linear transformation (a matrix) that maximises the variance of the data (i.e. removes covariance). This method can be used to reduce the dimensionality of the data - the most 'important' dimensions (i.e. the ones with highest variance) can be preserved.

For example, if a data set consisting of three-dimensional samples is given, but dimensions one and two are strongly correlated then those two dimensions will have high covariance. Those two dimensions will be mapped to two new dimensions, one having high variance and one having low variance. The one with lower variance can be discarded, leaving behind two dimensions.

### 2.1.1 Algorithm

Given a matrix $X$, of dimensionality $F \times N$ ($F$ = number of dimensions, $N$ = number of samples):

1. De-mean the data. Subtract each data point from the average, so that the mean value for all dimensions is zero.
2. Perform eigenanalysis on $XX^T$ (dimensionality $F \times F$).
3. Return a matrix, consisting of the eigenvectors corresponding to the n largest eigenvalues.

## 2.2 PCA/w - Principal Component Analysis with Whitening

After performing PCA, the data can be whitened, which involves normalising the variances - mapping the data to a space where the variance of every dimension is 1. This is useful because often machine learning methods involve comparing distances between data points or determining norms, if one dimension has a much higher variance than another, it can bias the results.

### 2.2.1 Algorithm

1. perform PCA, keep the eigenvalues.
2. divide the data by the square root of the eigenvalues (i.e. the variances of the distribution in the projected vector space).

## 2.3 LDA - Linear Discriminant Analysis

This method operates on training data that has been split into classes. It aims to find a linear transformation that maximises the distance between classes (inter-class distance) and minimises the distance between data points in the same class (intra-class distance, i.e. the variance of the classes). This improves classification tasks as it makes it easier to distinguish between data points in different classes. This can be expressed in terms of an intra-class scatter matrix $S_w$ and an inter-class scatter matrix $S_b$.

$$\boldsymbol{S}_w = \sum_{j=1}^{C} \boldsymbol{S}_j = \sum_{j=1}^{C} \frac{1}{N_{c_j}} \sum_{\boldsymbol{x}_i \in c_j} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$$

$$\boldsymbol{S}_b = \sum_{j=1}^{C} (\boldsymbol{\mu}(c_j) - \boldsymbol{\mu})(\boldsymbol{\mu}(c_j) - \boldsymbol{\mu})^T$$

A set of weights $w$ must be found that maximises $w^T S_b w$, such that $w^T S_w w = 1$.

### 2.3.1 Algorithm

1. Perform eigenanalysis on $(I - M)X^T X(I - M) = V_w \Lambda_w V_w^T$
2. From the eigenvalues compute the transform to whiten $S_w$ (ie, to make $w^T S_w w = 1$).
3. Project to a new space $X_b = U^T XM$
4. Perform PCA on $X_b$ to find $Q$
5. Compute final transform $W = UQ$

## 2.4 LPP - Locality Preserving Projection

This methods aims to find a linear transformation that preserves local structure. This is defined as the neighbourhood structure, i.e. which data points are close to others - or the way in which the data is clustered. To perform LPP, a clustering method such as $k$ nearest neighbours is performed, then a linear transformation is found that maximises the variance of the data but keeps the data points in the same clusters. This is useful when the dimensionality of the data is very high (e.g. image recognition problems) because computing the eigenvalues for every dimension (as in PCA) will become computationally expensive. In this algorithm, a transformation is found that whitens $XDX^T$ using the eigenvectors of $(D - S)X^T X(D - S)$ (which has dimensionality $N \times N$ where $N$ is the number of training data samples) and the eigenvectors of $X_p(D - S)X_p^T$ (which also has dimensionality $N \times N$).

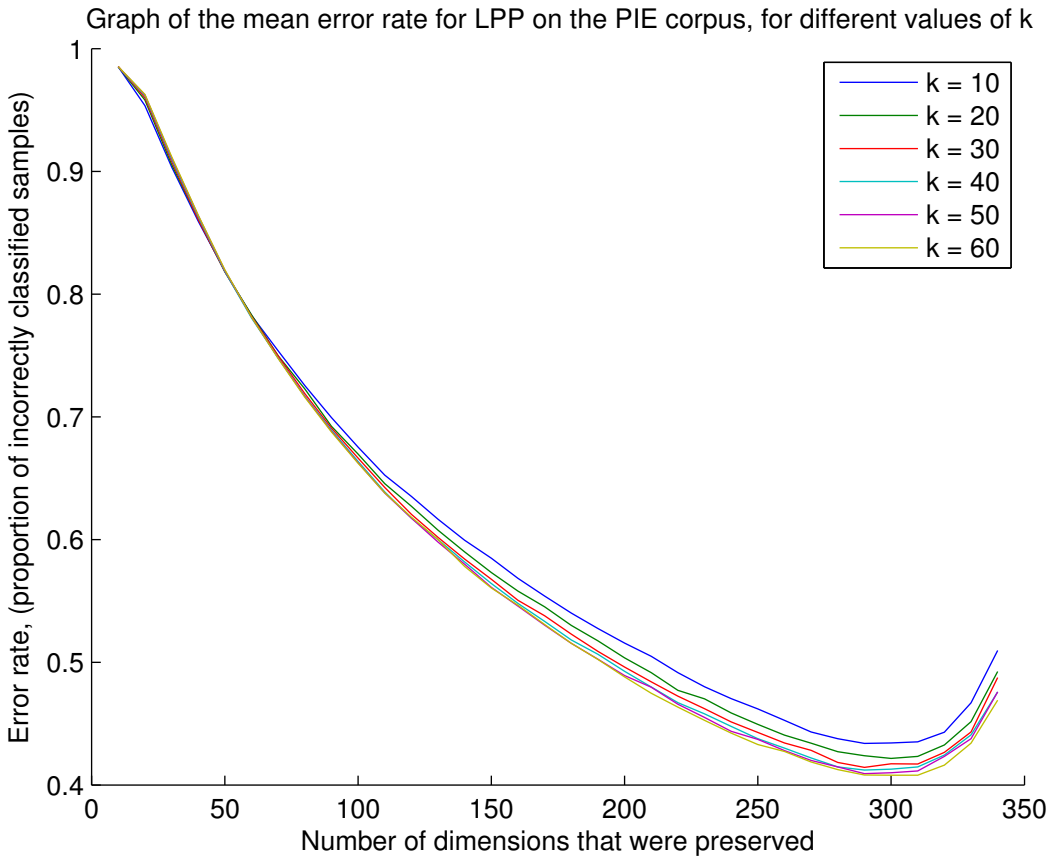
### 2.4.1 Algorithm

1. Use kNN to find a connectivity matrix, $S$
2. Find the matrix $U$, that whitens $XDX^T$
3. Transform $X$ with the matrix $U$
4. Find the eigenvectors of $X_p(D - S)X_p^T$
5. Choose the $n$ eigenvectors with the lowest eigenvalues

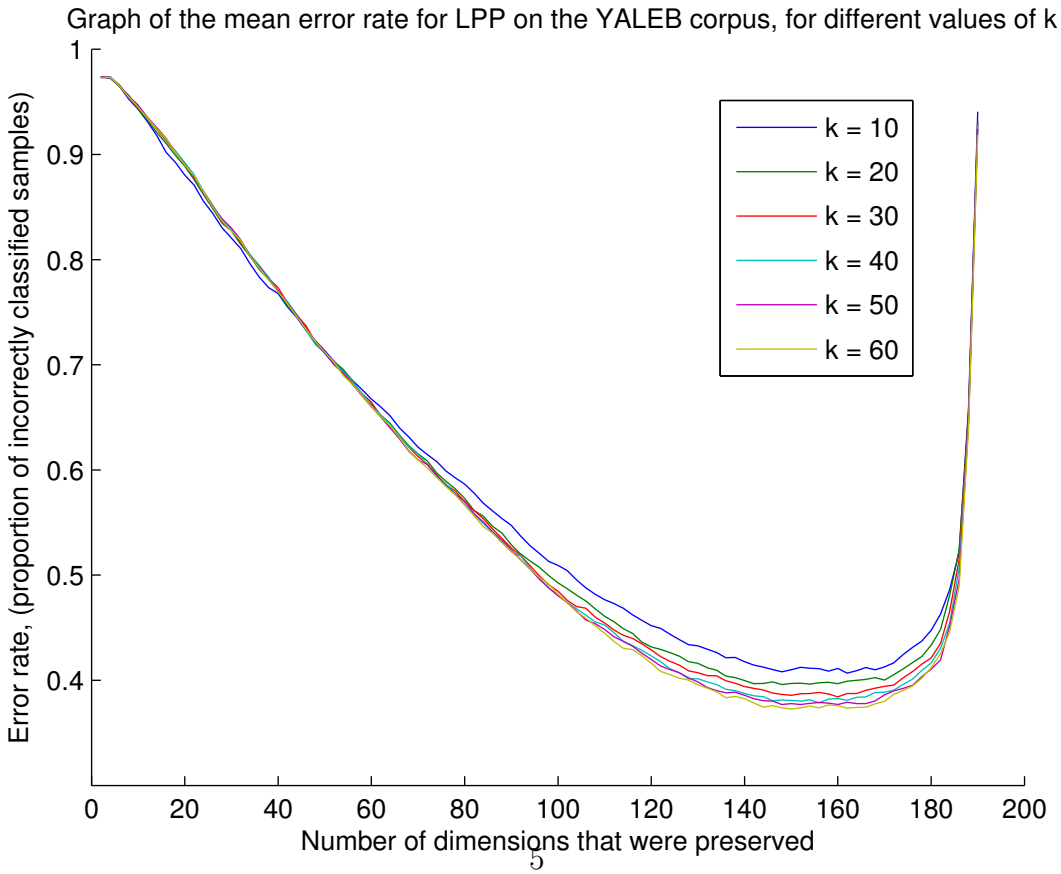
### 2.4.2 Optimisation

Unlike PCA, PCA/w and LDA, Locality Preserving Projections takes a parameter as input in addition to the training data. This parameter is k, which is the number of nearest neighbours to consider. Testing must be conducted, to find the ideal value of $k$. In the next two graphs, LPP has been performed for a range of $k$ values on the YALEB and PIE classification problems and the error has been plotted against the number of dimensions. The $k$ values considered were: $[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$.

It can be seen from the two graphs 1(a) and 1(b) that error rate improves as $k$ increases. This is because the distances to more neighbourhood data points is being considered. However, if more points are considered then the time taken to perform kNN clustering is longer, so there is a compromise to be made between efficiency and accuracy. The error rate does not increase as steeply after $k = 50$ in both the PIE and YALEB examples, so I decided to use that value for the final evaluation.


Graph of the mean error rate for LPP on the PIE corpus, for different values of k
Error rate, (proportion of incorrectly classified samples)
Number of dimensions that were preserved
k = 10
k = 20
k = 30
k = 40
k = 50
k = 60
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
300
350

(a)


Graph of the mean error rate for LPP on the YALEB corpus, for different values of k
Error rate, (proportion of incorrectly classified samples)
Number of dimensions that were preserved
k = 10
k = 20
k = 30
k = 40
k = 50
k = 60
1
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120
140
160
180
200



(b)

## 2.5 Fast ICA - Independent Component Analysis

Fast ICA [Hyvrinen, A] is an algorithm that solves the problem of independent component analysis. Given a set of $F$ components (e.g. signals, dimensions of training data) on $N$ training data samples, return a transformation matrix that de-mixes the input into a set of $F$ independent components. Dimensionality reduction is performed by only mapping the data to the most important components. The returned transformation matrix is of a slightly different form to the matrix returned by the other methods, instead of having dimensionality $F \times C$ (where $C$ is the number of reduced dimensions), it has dimensionality $C \times F$. So it must be transposed before being used on the input data.

### 2.5.1 Algorithm

1. for $p$ in the range 1 to $C$ ($C$ is the number of desired components):
   (a) initialise $wp$, the vector that will express the pth component, to random numerical values. it has dimensionality $F \times 1$.
   (b) for i in the range 1 to num_iterations:
      i. iteratively update wp, with respect to the observed data and the previously considered wp vectors.
   (c) store wp as a row in the $C \times F$ matrix $X$
2. return the matrix $S$ (dimensionality $C \times F$), which consists of all of the past wp vectors (one for every p value considered), multiplied by the input data, $X$

As Fast ICA is an iterative algorithm that approximates an optimal solution, the more iterations that take place, the better the results will be but the longer the algorithm will take to run. A possible experiment would be to run Fast ICA with different number of iterations, but I did not have time to compute this.

# 3 Evaluation

To compare the performance of all of the different methods, I ran them all on the PIE and YALEB facial recognition problems. Fast ICA was run with the number of iterations set to 10 and LPP was run with a k value of 50. The PIE problem consisted of 340 training data samples and the YALEB problem consisted of 190.

Figure 1(c) and 1(d) show the results. The error rate decreases with every method as the number of dimensions is increased, but the error for PCA/whitening starts to increase at 250 on PIE and 150 on YALEB - this could be due to over-specialisation, that is, the

model has become unable to generalise to predict unseen data points. The error rate for LPP also curves upwards at 300 for PIE and 160 for YALEB, presumably for the same reason.

From the graphs it is possible to determine that PCA/whitening is the best method overall at any dimensionality, then either LDA or LPP, then PCA, then Fast ICA. I have some comments as to why this result may have occurred:

- Fast ICA achieved the worst error rate, because the distribution of the data is not appropriate for independent component analysis. ICA exploits the non-Gaussianity of the input data, so it may have performed badly because the data is normally or normally distributed. It also assumes that the original signals are independent. A way to test this hypothesis would be to calculate the non-Gaussianity of the data using a measure such as kurtosis or negentropy.

- PCA removes covariance, but it does not normalise the variances. This means that if one dimension has a high variance, then it could bias the decision made by the classifier.

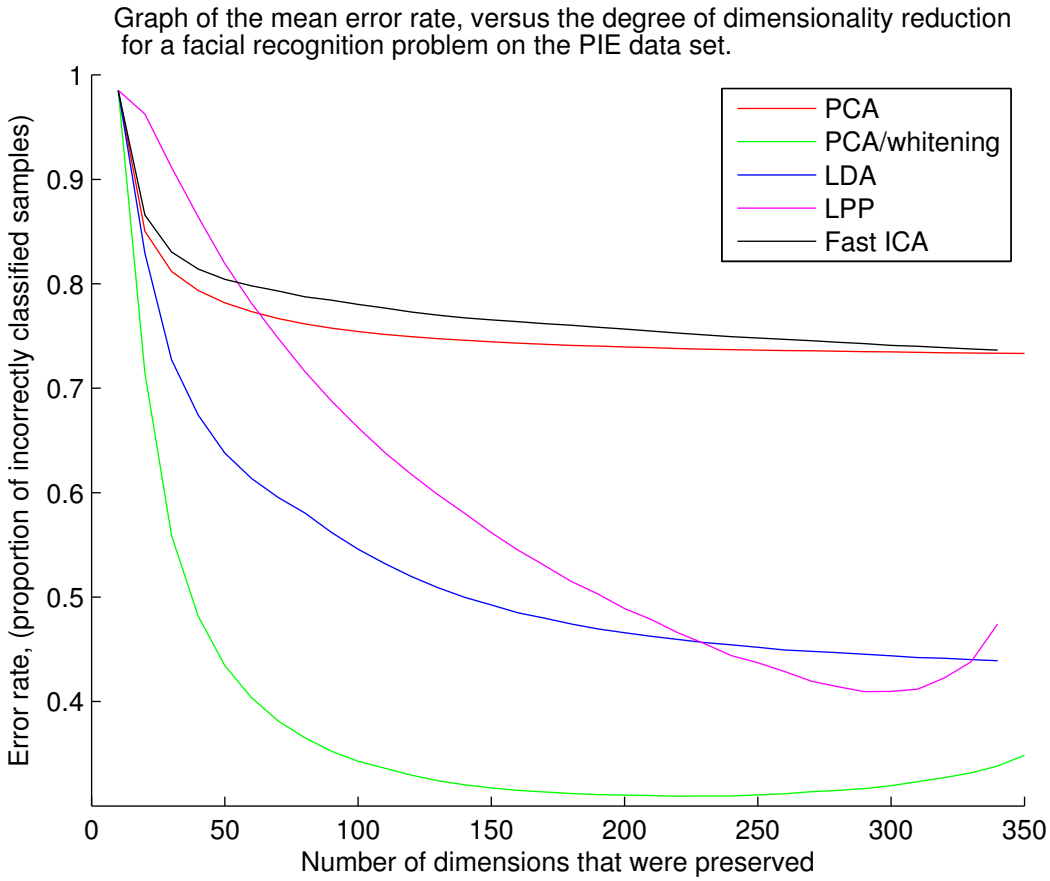
In the original research paper for LPP [He, Niyogi, 2004], it performs better than PCA and LDA on the YALEB facial recognition task, with an error rate of 16% (or 0.16) which is much lower than the lowest error rate in my experiment (about 0.4, or 40%). In this paper, PCA has an error rate of 25.3% and LDA has an error rate of 20%. Some of the parameters are different (e.g. the images are 32 by 32 pixel grayscale, compared with 64 by 64 in this experiment) but even so, this hints that further improvement may be possible.

# 4 Conclusion

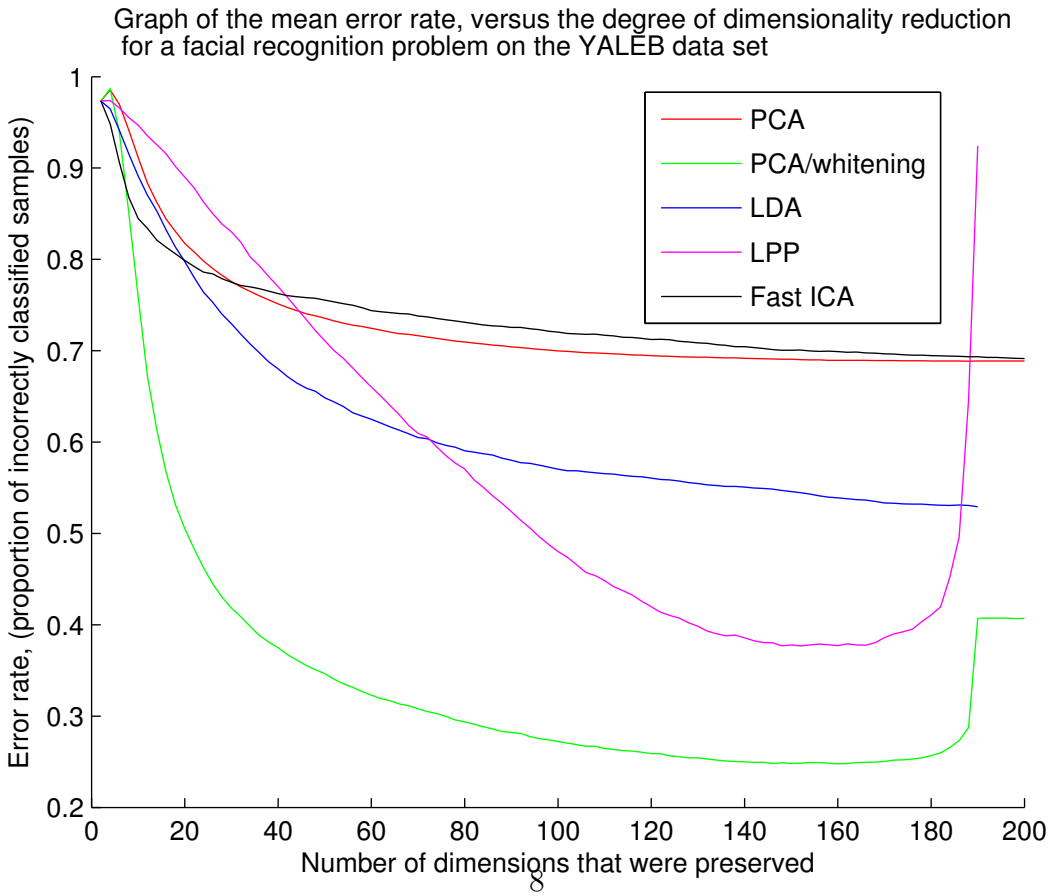
In this report, I described the PCA, PCA with whitening, LDA, LPP and Fast ICA algorithms as well as presented an evaluation of their performance on the PIE and YALEB facial classification tasks. In the appendix, the Matlab code used to implement them is enclosed, which contains comments about the algorithms. PCA with whitening achieved the lowest error rate, then LPP, LDA, PCA and Fast ICA. If I had more time, I would have investigated the effect of changing the number of iterations made by the Fast ICA algorithm and measured the running times of each algorithm.

# References

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.


Graph of the mean error rate, versus the degree of dimensionality reduction
for a facial recognition problem on the PIE data set.
PCA
PCA/whitening
LDA
LPP
Fast ICA
Error rate, (proportion of incorrectly classified samples)
Number of dimensions that were preserved
1
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
300
350

(c)


Graph of the mean error rate, versus the degree of dimensionality reduction
for a facial recognition problem on the YALEB data set
PCA
PCA/whitening
LDA
LPP
Fast ICA
Error rate, (proportion of incorrectly classified samples)
Number of dimensions that were preserved
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
120
140
160
180
200



(d)

[Yu L] Lei Yu, Jieping Ye and Huan Liu. Dimensionality Reduction for Data Mining - Techniques, Applications and Trends (Lecture notes) http://www.cs.binghamton.edu/ lyu/SDM07/DR-SDM07.pdf (accessed 22/02/15)

[He, Niyogi, 2004] He, X and Niyogi, P. Locality Preserving Projections http://papers.nips.cc/paper/2359-locality-preserving-projections.pdf

[Hyvrinen, A] Hyvrinen, A and Oja, E. Independent Component Analysis: Algorithms and Applications http://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf (accessed 22/02/15)

[Hyvrinen, A] Hyvrinen, A Fast and Robust Fixed-Point Algorithms for Independent Component Analysis http://www.cs.helsinki.fi/u/ahyvarin/papers/TNN99new.pdf (accessed 22/02/15)

[Hyvrinen, A] FastICA http://en.wikipedia.org/wiki/FastICA (accessed 22/02/15)

# 5 Appendix - Code

```
function [ W, WD ] = PCA( Data, n )
%PCA Returns the principal component analysis transformation
   matrix
    % de-mean X
    Data_d = bsxfun(@minus, Data, mean(Data,1));
    X = Data_d';

    % eigenanalysis
    [W,WD] = eigs(X * X', n);
end
```

```
function [ W ] = PCAWhitening( Data, n )
%PCAWHITENING Returns a matrix that applies PCA and whitening
    to the data
    % de-mean X
    Data_d = bsxfun(@minus, Data, mean(Data,1));
    X = Data_d';

    % eigenanalysis
    [V,L] = eigs(X * X', n);

    % Apply whitening
    eV = 1 ./ sqrt(abs(L));
    eV(isinf(eV)) = 0;
    W = V * eV;
```

```
end
```

```
function [ W ] = LDA( Data, Gnd, N )
%LDA Perform linear discriminant analysis on a dataset 'Data'
    (samples X features), given a list of class numbers 'Gnd'
    and number of dimensions to preserve N
    num_clusters = size(unique(Gnd),1);
    cluster_freqs = hist(Gnd, num_clusters);

    E_array = cell(num_clusters);
    for i = 1:num_clusters
        E_array{i} = ones(cluster_freqs(i)) * cluster_freqs(i
           );
    end
    M = blkdiag(E_array{:});

    X = Data';

    I = eye(size(M,1));

    % 1. Perform eigenanalysis of Xw
    Xw = X * (I - M);
    Ue = Xw' * Xw;
    [Vw, Lw] = eig(Ue);

    % Remove zero eigenvalues and corresponding vectors
    a = diag(Lw);
    inds = find(a);
    Vw_clean = Vw(inds,:);
    Lw_clean = Lw(inds,:);

    % 2. From the eigenvalues compute the transform to whiten
        Sw (ie, to make w' Sw w = 1).
    U = X * (I - M) * Vw_clean / Lw_clean;

    % 3. Project to a new space Xb = U' X M
    Xb = U' * X * M;

    % 4. Perform PCA on Wb to find Q
    Q = PCA(Xb, N);

    % 5. Compute final transform
    W = U * Q;
end
```

```
function [ W ] = LPP( Data, n, k)
%LPP Find the locality preserving projection of the data 'Data', with k nearest neighbours and n dimensions.
    % In the eigenanalysis part, each column is a data entry
    % but the KNearestNeighbours command takes rows as data entries
    X = Data';

    % 1. Calculate S, the connectivity matrix
    S = KNearestNeighbours(Data, k);

    % 2. Find the matrix U, that whitens XDX'
    D = diag(sum(S,1));
    Ds = sqrt(D);

    L = eig(Ds * (X' * X) * Ds);
    Lp = diag(1 ./ sqrt(L));

    U = Lp * Ds * X';

    % 3. Project X with U
    Xp = U * X;

    % 4. Eigenanalysis
    [Vq, Vl] = eig(Xp * (D - S) * Xp');

    % 5. Find best eigenvectors, perform dimensionality reduction if desired
    [~, eig_indices] = sort(diag(Vl), 'ascend');
    top_indices = eig_indices(1:min(size(eig_indices,1), n));

    Q = Vq(:,top_indices);
    W = U' * Q;
end
```

```
function [ W, S ] = FastICA( Data, C )
%FASTICA Performs independent component analysis using the FastICA algorithm.
    max_iter = 10;

    X_nowhiten = Data';

    Whit = PCAWhitening(X_nowhiten, size(Data,1));

    % X must be NxM
```

```
10      X = X_nowhiten * Whit;
11
12      % M = number of samples
13      % C = number of desired components
14      % N = number of dimensions
15
16      % ICA
17      N = size(X,1);
18      M = size(X,2);
19      W = zeros(C, N);
20
21      for p = 1:C
22          % initialise wp, the vector that will express the pth
                component
23          w = rand(N,1);
24
25          for i = 1:max_iter
26              %iteratively update wp
27              p1 = X * G(w' * X)';
28              p2 = Gp(w' * X) * ones(M,1) * w;
29              w = (p1 - p2) / M;
30
31              err = 0;
32              for j = 1:(p-1)
33                  wj = W(j,:)';
34                  d = w' * wj * wj;
35                  err = err + d;
36              end
37              w = w - err;
38
39              w = w / norm(w);
40          end
41
42          W(p,:) = w;
43      end
44
45      % return the matrix S
46      S = W * X;
47  end
48
49  % definitions of the first and second derivative of a
       nonquadratic nonlinearity function f
50  function [X] = G(w)
51      X = tanh(w);
52  end
53
```

```
function [X] = Gp(w)
    X = 1 - pow2(tanh(w));
end
```

```
function [ S ] = KNearestNeighbours( X, k )
%KNEARESTNEIGHBOURS An implementation of the k nearest
   neighbours clustering algorithm
    N = size(X, 1);
    S = zeros(N, N);

    % for every data point
    for i = 1:N
        % calculate all of the distances
        distances = zeros(N);
        for j = 1:N
            distances(j) = L2Norm(X(i,:), X(j,:));
        end

        % find the nearest neighbours
        [~, Nearest] = sort(distances);
        kNearestNeighbours = Nearest(2:k+1);

        % generate connectivity matrix
        for l = 1:k
            nbr = kNearestNeighbours(l);
            S(i,nbr) = 1;
            S(nbr,i) = 1;
        end
    end
end
```