K-means Clustering & PCA

Andreas C. Kapourani

(Credit: Hiroshi Shimodaira)

08 February 2017

1 Introduction

In this lab session we will focus on *K-means clustering* and *Principal Component Analysis* (PCA). Clustering is a widely used exploratory tool, whose main task is to identify and group similar objects together. Clustering can be categorized as an *unsupervised* learning approach, since we try to interpret and discover hidden structure in unlabeled data. On the other hand, the problem of classification is to predict the correct label for some input data. The classifier is learned using a set of training data containing feature vectors and their labels. This is referred to as *supervised* learning, since the label for the training vector acts as supervision for the classifier during the learning phase. PCA is a technique that is widely used for applications such as dimensionality reduction, visualization and lossy data compression. In this lab we will focus on the visualization aspect.

2 Example Data

For this lab session we will use the `Old Faithful` data set (`http://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat`). The data set comprises 272 observations, each of which represents a single eruption and contains two variables corresponding to the duration in minutes of the eruption, and the time until the next eruption, also in minutes. You can download the Old faithful dataset from the course website:

`http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnLabSchedule.html`

You will find a file named `faithful.txt`, download it and save it in your current folder.

Let's read the file in a similar way we have done in the previous lab sessions.

```
% name of the file to be loaded
>> filename = 'faithful.txt';

% split data at specified delimiter
>> delimiterIn = '\t';

% read numeric data starting from line headerlinesIn+1
>> headerlinesIn = 1; % Note that the first line should be skipped

% Use importdata to load the data
file_data = importdata(filename, delimiterIn, headerlinesIn);
```

Since the Old Faithful dataset contains also a header line, MATLAB will store our data in a `structure array`. (**Exercise**: Check what a `struct` is in MATLAB.)

We can extract the information we want from the structure array by using the `.` (dot) operator and the name of the field we want to access.

```
% Check what the structure array contains
>> file_data
file_data =
            data: [272x2 double]
        textdata: {'eruptions'   'waiting'}
      colheaders: {'eruptions'   'waiting'}

% store the actual data in matrix A
>> A = file_data.data;
% store the column headers
>> col_headers = file_data.colheaders;
```

Figure 1 shows a plot of the time to the next eruption versus the duration of the eruptions. It can be seen that the time to the next eruption varies considerably, although knowledge of the duration of the current eruption allows it to be predicted more accurately. Also, it is clear that the observations are partitioned in two distinct groups (clusters).

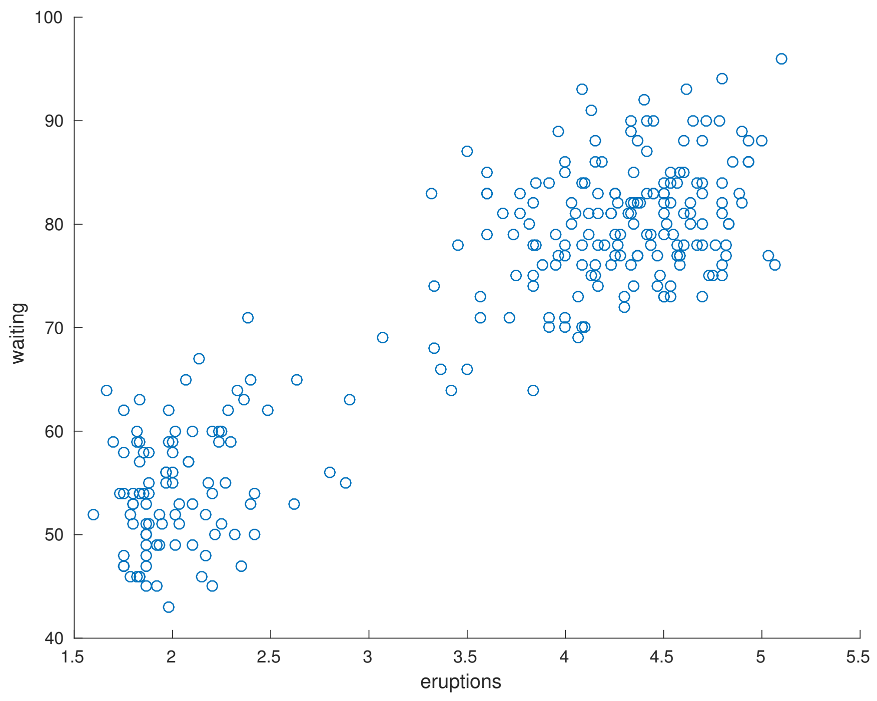


Figure 1: *Plot of the time to the next eruption in minutes (y-axis) versus the duration of the eruption in minutes (x-axis) for the Old Faithful dataset. It is clear that the observations are clustered in two distinct groups.*

The code for creating the above plot is the following:

```
% Plot of the Old Faithful data
scatter(A(:,1), A(:,2))
% Extract label information from the file headers
xlabel(col_headers{1}); ylabel(col_headers{2})
```

3 K-means clustering

K-means algorithm is one of the most employed clustering algorithms. K-means aims to partition N observations into K clusters in which each observation belongs to the cluster with the nearest mean. The algorithm proceeds as follows:

1. Pick K random points as cluster centre positions.
2. Assign each point to the nearest centre.
3. Recompute each cluster mean as the mean of the vectors assigned to that cluster.
4. If centres moved, goto 2.

The algorithm requires a distance measure to be defined in the data space, and the Euclidean distance is often used. Taking the recommender system as an example, we could use the Euclidean distance between the critics in order to find K similar groups of critics, e.g. the ones that rate higher drama movies and lower thriller movies, etc.

3.1 Euclidean distance (again...)

Let's try and implement the K-means algorithm using the steps shown in the previous section. First of all, since we need to compute distances between observations and cluster centres, we need again to use the `square_dist` function from the previous lab session. If you do not have this file, create a new file, name it `square_dist.m` and copy & paste the following MATLAB code:

```
function sq_dist = square_dist(U, v)
% Compute 1 x M row vector of square distances for M x N and 1 x N data U
    and v, respectively.
  sq_dist = sum(bsxfun(@minus, U, v).^2, 2)';
end
```

Note that the `square_dist` function computes the distance between the row vector `v` and each row of matrix `U`. Thus, the number of columns of matrix `U` and the vector length should be the same.

Reminder: The Euclidean distance $r_2(\mathbf{u}, \mathbf{v})$ between two 2-dimensional vectors $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2)$ is given by the following expression:

$$r_2(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2} = \sqrt{\sum_{i=1}^{2}(u_i - v_i)^2} \qquad (1)$$

3.2 Pick K random points as cluster centre positions

From the plot in Figure 1, we observe that a sensible choice for the total number of clusters is K = 2. We will pick randomly two points in this 2-dimensional space:

```
% Total number of clusters K
>> K = 2;
% pick random points
>> Cc0 = [2 90; 5 50];

% Cell array of different MATLAB color points
>> colors = {'r.', 'g.', 'b.', 'k.'};
>> figure
>> scatter(A(:,1), A(:,2), '+');
% show each cluster centre (cluster1 = red, cluster2 = green)
>> for c = 1:K
      hold on;
      scatter(Cc0(c,1), Cc0(c,2), 300, colors{c});
  end
```

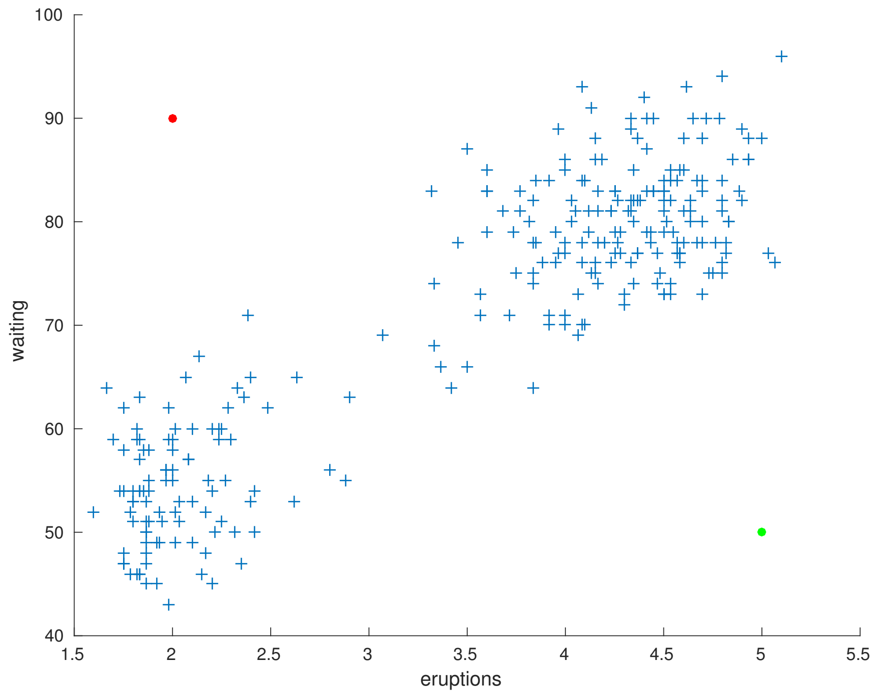


Figure 2: *Plot of the Old Faithful dataset together with initial cluster centres. Red colour point is Cluster 1 and green colour point is Cluster 2.*

3.3 Implement K-means algorithm

The core idea of K-means algorithm is to implement steps 2 and 3 shown in the previous section. This is quite straightforward, since we have already implemented the square distance function (i.e. step 2). The following MATLAB code implements the K-means algorithm (write the following code in a `script` file instead of running each line in the `Command Window`):

```
% Very simple version of K-means algorithm
centres = [2 90; 5 50]; % initial cluster centres

K = size(centres, 1);    % extract number of clusters

[N dim] = size(A);       % dataset dimensions

maxiter = 100;           % Maximum number of iterations

D = zeros(K, N);         % KxN matrix for storing distances between
                         % cluster centres and observations
```

```
fprintf('[0] Iteration: ')
centres                 % show cluster centres at iteration 0

% Iterate 'maxiter' times
for i = 1:maxiter
    % Compute Squared Euclidean distance (i.e. the squared distance)
    % between each cluster centre and each observation
    for c = 1:K
        D(c,:) = square_dist(A, centres(c,:));
    end

    % Assign data to clusters
    % Ds are the actual distances and idx are the cluster assignments
    [Ds, idx] = min(D); % find min dist. for each observation

    % Update cluster centres
    for c = 1:K
        %check the number of samples assigned to this cluster
        if( sum(idx==c) == 0 )
            warn('k-means: cluster %d is empty', c);
        else
            centres(c, :) = mean( A(idx==c,:) );
        end
    end

    fprintf('[%d] Iteration: ', i)
    centres      % show cluster centres at iteration i
end
```

When running the above code, the final cluster centres should be the following:

```
% Final cluster centre locations
>> centres
centres =
    4.2979   80.2849
    2.0943   54.7500
```

Check Figure 2, and see if these points are actually the 2 cluster centres in the Old Faithful data (**Note**: Each centre point is given by the rows of matrix `centres`).

Exercise

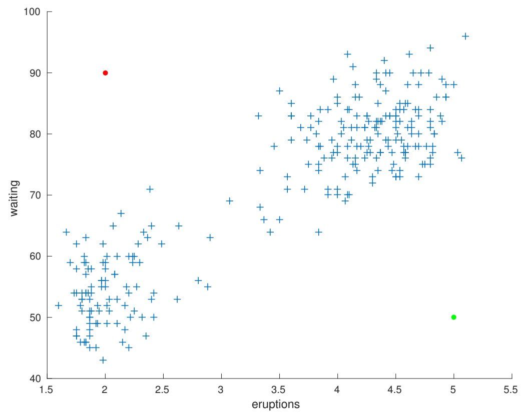
The given K-means MATLAB code prints the cluster centres on each iteration. Scrolling up in the first iterations of the algorithm, you will realize that the actual cluster centres were learned from the 3rd iteration! Modify the given algorithm so it terminates when the actual cluster assignments (or equivalently the cluster centres) have not moved.

Hint 1: Create a vector `idx_prev` for storing the previous cluster assignments so you can compare if cluster assignments have changed. Remember to update the previous vector of cluster assignments on each `for` loop iteration.

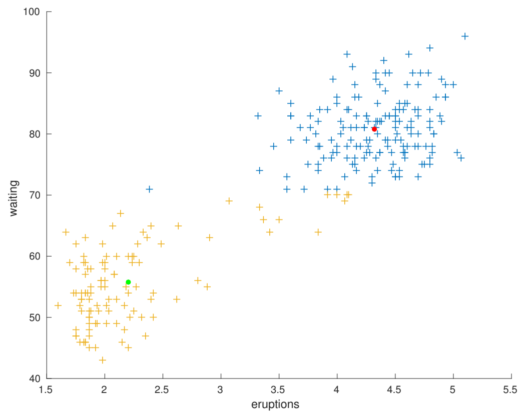
Hint 2: To terminate execution of a `for` loop you can use the `break` command.

3.4 Cluster assignment plots on each iteration

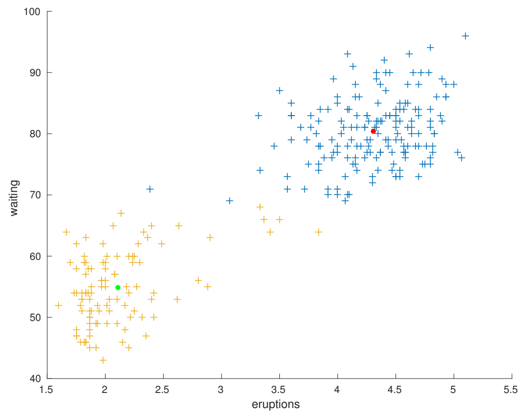
We can also plot the cluster centres and cluster assignments on each iteration of the K-means algorithm, as shown in Figure 3. It is clear that after two iterations the algorithm had found the actual cluster centres and has assigned each observation to its actual group.



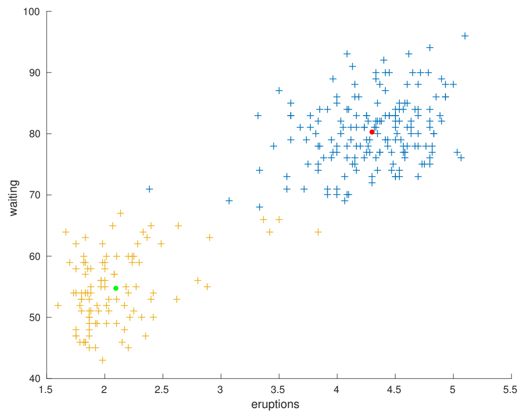
(a) Iteration 0



(b) Iteration 1



(c) Iteration 2



(d) Iteration 3

Figure 3: *K-means cluster assignments on each iteration of the algorithm.*

Exercises

1. Create a function `myKmeans` which will take a dataset `A` and random initial cluster centres, and will apply the K-means algorithm as described above. You could also create a plot function which can be called on each iteration to plot the cluster assignments as shown in Figure 3.
2. MATLAB has a built-in function `kmeans`, which performs K-means clustering on a given dataset of observations. Check what parameters this function takes as input and apply it for partitioning the Old Faithful data into K = 2 clusters. Do the cluster centres agree with your implementation?
3. Run your K-means implementation using a higher number of clusters, e.g. K = 4, are now the results similar to MATLAB's implementation?
4. One way to measure the quality of the K-means clustering solution is to compute the sum-squared error:

$$E = \frac{1}{N} \sum_{k=1}^{K} \sum_{n=1}^{N} z_{kn} \|\mathbf{x}_n - \mathbf{m}_k\|^2 \tag{2}$$

where, $\mathbf{m}_k$ is the centre of cluster k, and $z_{kn} = 1$ when point $\mathbf{x}_n$ belongs to cluster k and $z_{kn} = 0$ otherwise.

Modify your k-means implementation so as to compute the sum-squared error on each iteration. Does this error decrease after each iteration? Increase the number of clusters K, and check the values of error function. Now you will be able to re-create the figure shown in Lecture 3, Slide 23. **Note**: The error function probably will not decrease in the same way, since we are using a different dataset from the example shown in the lecture slides.

4 Principal Component Analysis

A way for obtaining a better understanding of the data is to visualise it; however, it is not straightforward to visualise high-dimensional data. This was a problem, for example, in the critics-movie ratings dataset for recommender systems, where we could only choose pairs of movies (or critics) and plot them.

Principal Component Analysis (PCA) is a technique that is widely used for applications such as dimensionality reduction, visualization and lossy data compression. In this lab we will focus on the visualization aspect. PCA can be defined as the orthogonal linear transformation of the data to a lower dimensional linear space, known as the *principal subspace*, such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Intuitively, PCA finds a *meaningful* coordinate basis to express our data (see Lecture 3 for algebraic details).

Let's create some synthetic data and show how we can use PCA to transform them to a new basis, and finally project them to a lower dimension.

```
% Synethic data in 2 dimensions (x,y)
X = [2.5 2;
    0.5 0.7;
    2.2 2.4;
    1.9 2.5;
    3.1 3.9;
    2.3 2.7;
    2 1.8;
    1.5 1.6;
    1 1.3];
% Number of observations
N = length(X);
```

You can plot the data using the `scatter` function.

The first step in PCA is to subtract off the mean for each dimension (in our case just two dimensions), thus, each dimension will have zero mean.

```
% Calculate the mean for each column
x_mean = mean(X, 1);

% Mean shift the original matrix
X = bsxfun(@minus, X, x_mean);
```

Next we need to compute the covariance matrix between the two dimensions.

```
% Compute covariance matrix of X
>> covar_m = 1/(N-1) * (X' * X)
covar_m =
    0.6236    0.6600
    0.6600    0.8500
```

Now we compute the eigenvectors and eigenvalues of the covariance matrix using the `eig` MATLAB function. These will be our new basis vectors.

```
% find the eigenvectors and eigenvalues
% PC are the principal components, i.e. eigenvectors
% and V are the corresponding eigenvalues
>> [PC, V] = eig(covar_m)
PC =
   -0.7645    0.6446
    0.6446    0.7645
V =
    0.0672         0
         0    1.4064
```

Our new basis are orthogonal, since their dot product is zero:

```
% Show that the new basis vectors are orthogonal to each other
>> PC(:,1)' * PC(:,2)
ans =
     0
```

Since the first component should *explain* most of the variance, and the second most of the remaining variance, and so on, we should sort the variances in decreasing order, this can be done by ordering the eigenvalues in the following way:

```
% extract diagonal of matrix as vector
V = diag(V);

% sort the variances in decreasing order
[tmp, ridx] = sort(V, 1, 'descend');

V = V(ridx);

PC = PC(:,ridx);
```

Now we can plot the principal component basis together with the original data:

```

% Show the data in original basis
scatter(X(:,1), X(:,2));
xlabel('x'); ylabel('y');
axis([-2 2 -2 2])
hold on;

% Plot the first new basis
hh = refline(PC(2,1)/PC(1,1), 0);
hh.Color = 'r';
hold on;

% Plot the second new basis
hh = refline(PC(1,2)/PC(2,2), 0);
hh.Color = [0.7 0.7 0.1];
axis([-2 2 -2 2])
box on;
```

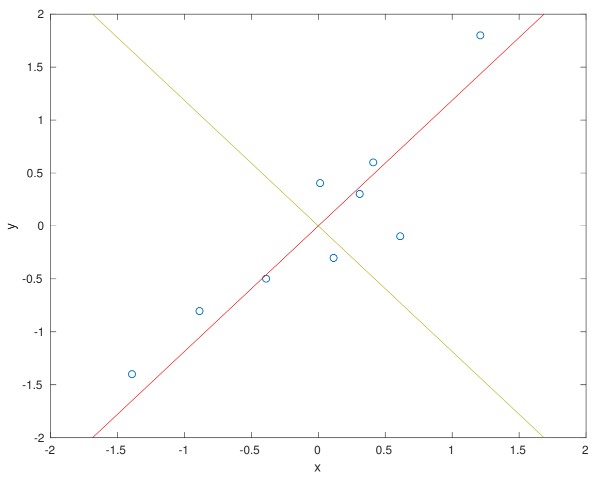


Figure 4: *Original data, together with the first principal component (red line) and second principal component (yellow line).*

Now we can transform our original dataset X and project it in the new principal subspace.

```
% Transform the original data X to the principal subspace
PC_X = X * PC;
% Plot the data in the new basis
plot(PC_X(:,1), PC_X(:,2), 'o')
xlabel('1st Principal Component');
ylabel('2nd Principal Component');
box on
xlim([-2.3 2.3]); ylim([-2.3 2.3])
```

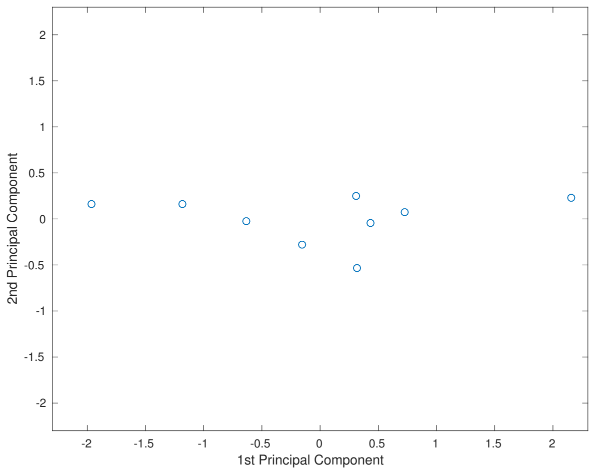


Figure 5: *Plot of the data in the 2-dimensional space obtained by the first two principal components.*

We can check that the data after the transformation are uncorrelated, by computing the covariance matrix for each dimension and see if the non-diagonal elements are zero:

```
% Compute covariance matrix using Matlab's function cov
>> cov(PC_X)
ans =
    1.4064   -0.0000
   -0.0000    0.0672
```

In the introduction of the PCA, we mentioned that we mainly use it to project data onto a lower dimensional space so as to be able to visualize them; however, until now we just transformed the data from one basis (our x-y Cartesian coordinate system) to another basis (the principal subspace).

However, the dimensionality reduction part is straightforward, we just keep the first *P* principal components and discard the less important components (**Note**: We will lose information but not that much if the eigenvalues are really small). In our case we had just two components, to keep only the first principal component we take the first column of the `PC` matrix and transform the `X` dataset using only that component:

```
% We took the transpose only to show the results as a row vector
>> (X * PC(:,1))'
ans =
    0.3175   -1.9656    0.4299    0.3130    2.1568    0.7237   -0.1577
   -0.6329   -1.1846
```

Exercises

- Create a function `myPCA` which given a matrix `A`, will perform Principal Component Analysis on it. The function should return: the sorted eigenvectors `PC`, the corresponding eigenvalues `V` and the transformed data `PC_X`.
- Apply PCA to the `Old Faithful` data set and then apply K-means algorithm on the 1st principal component. Do the cluster assignments agree when using K-means on the original dataset? What happens when you use the 2nd principal component for performing clustering?
- Use the critics-movie ratings dataset from Lab 3. This is a 6×6 matrix, hence we cannot plot the critics preferences in a 2-dimensional plot. Apply PCA on this matrix and plot the critics in the 2-dimensional space obtained by the first two principal components. The result should be similar to *Figure 6*, where each number corresponds to the specific critic, i.e. number 2 corresponds to the second row in the original dataset, and so on.
- Plot the movies in the 2-dimensional space obtained by the first two principal components. **Hint**: Just transpose the matrix so rows correspond to movies and columns to critics.

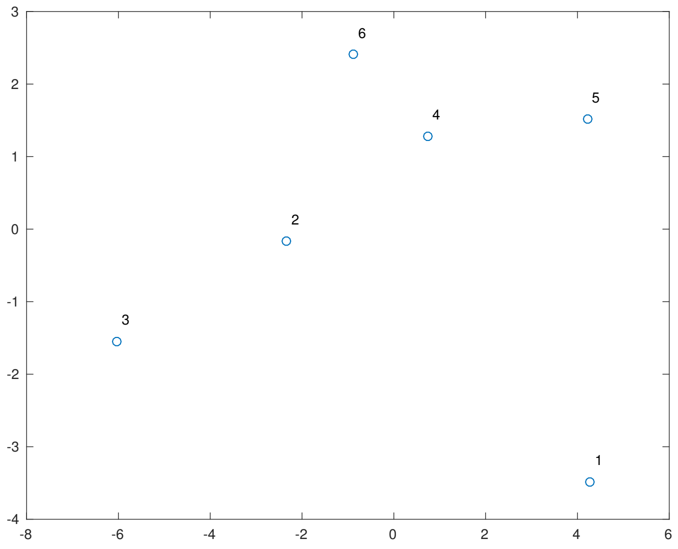


Figure 6: *Plot of the* ***critics*** *in the 2-dimensional space obtained by the first two principal components.*

5 Appendix A: Simple K-means function

```
%
%  A very simple Kmeans clustering.
%
%  This is a slightly modified version from the original code written
%  by Hiroshi Shimodaira <h.shimodaira@ed.ac.uk>  February 2015
%
%  Input:  A(N,dim)        : N samples (row vector of dim-dimension)
%          centres(K,dim)  : Initial K cluster centers (row vector)
%          maxiter         : Integer, maximum number of iteration
%          verbose         : Logical, print results during each iteration
%  Output: idx:            : Cluster index table of samples (row vector)
%          centres(K,dim)  : Final cluster centres
%

function [idx, centres] = simpleKmeans(A, centres, maxiter, verbose)
    % If 'verbose' argument is not given, we set by default to false
    if nargin < 4
        verbose = false;
    end
    % If 'maxiter' argument is not given, we set by default to 100
    if nargin < 3
        maxiter = 100;
    end

    K = size(centres, 1);    % extract number of clusters
    [N dim] = size(A);       % dataset dimensions
    D = zeros(K, N);         % KxN matrix for storing distances between
                             % cluster centres and observations

    idx_prev = zeros(1, N); % 1xN vector storing cluster assignments

    if verbose
        fprintf('[0] Iteration: ')
        centres              % show cluster centres at iteration 0
    end

    for i = 1:maxiter
        % Compute Squared Euclidean distance (i.e. the squared distance)
        % between each cluster centre and each observation
        for c = 1:K
            D(c,:) = square_dist(A, centres(c,:));
        end

        % Assign data to clusters
        % Ds are the actual distances and idx are the cluster assignments
        [Ds, idx] = min(D); % find min dist. for each column

        % Update cluster centres
        for c = 1 : K
            %check the number of samples assigned to this cluster
            if( sum(idx==c) == 0 )
                warn('k-means: cluster %d is empty', c);
            else
                centres(c, :) = mean( A(idx==c,:) );
            end
        end

        % Check for convergence
        if( sum( abs(idx - idx_prev) ) == 0 )
          break;
        end
        % update previous assignment vector with current
        idx_prev = idx;

        if verbose
            fprintf('[%d] Iteration: ', i)
            centres     % show cluster centres at iteration i
        end
    end
end
```