# Introduction to Intelligent Systems - Lab 6

Diego Velasco Volkmann   Eduardo Faccin Vernier
S2851059   S3012875

October 31, 2015

## Exercise 1:

Unsupervised Learning - VQ
Our group chose to work on the w6_1x.mat dataset. After many runs of the algorithm, the final mean configurations would look like these with slight variations.

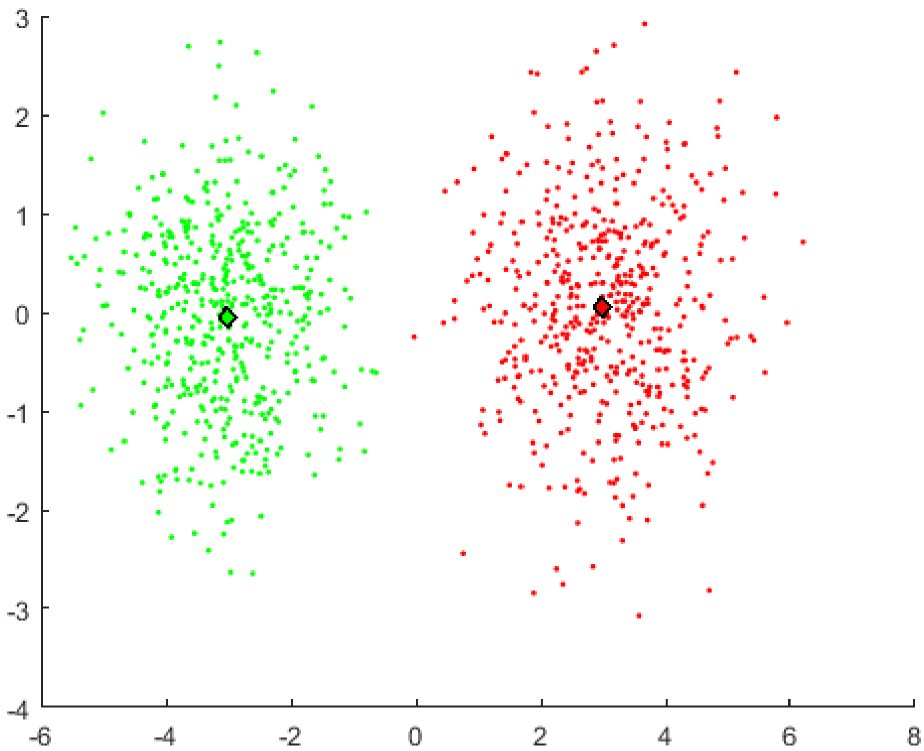


Figure 1: For k = 2

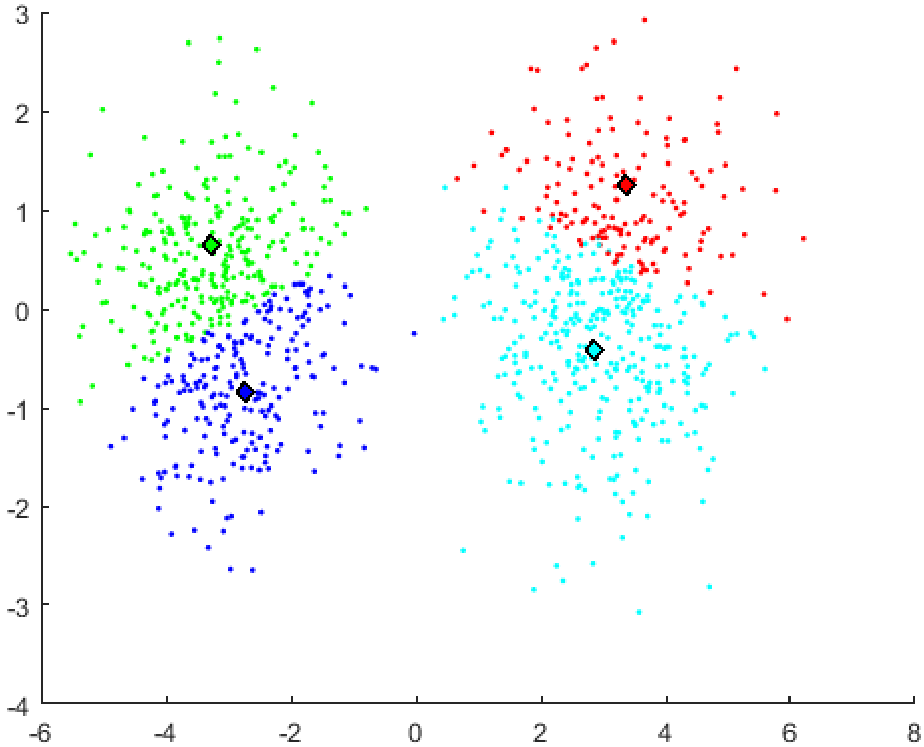


Figure 2: For k = 4

Because the algorithm has a random approach, the error quantization graphs for different runs might vary a lot. Below there are examples of the most common scenario, tested after many executions with same parameters.

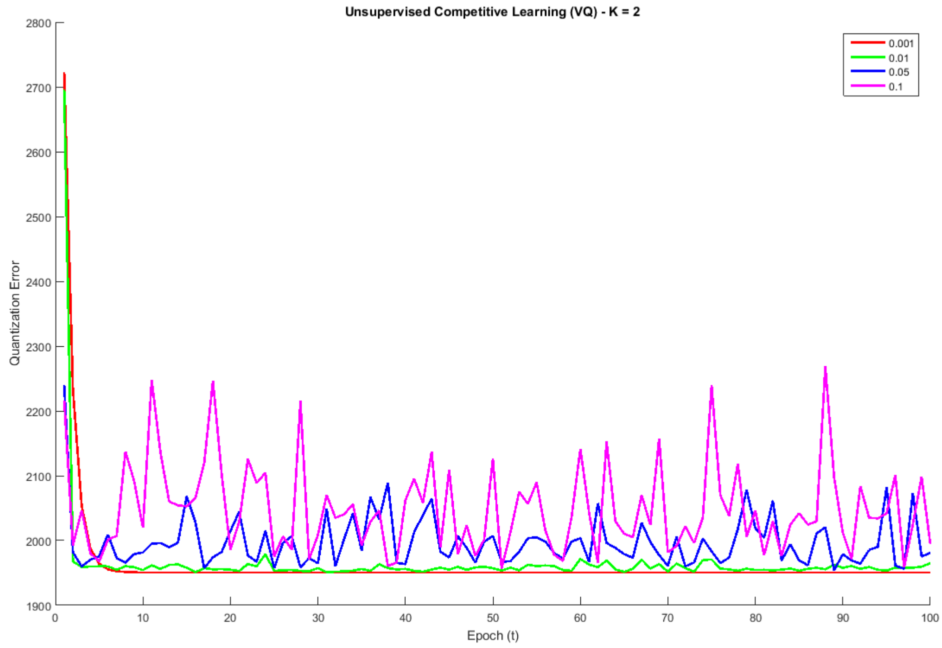


Figure 3: k = 2, learning rate values of 0.001, 0.01, 0.05, 0.1 and up to 100 epochs

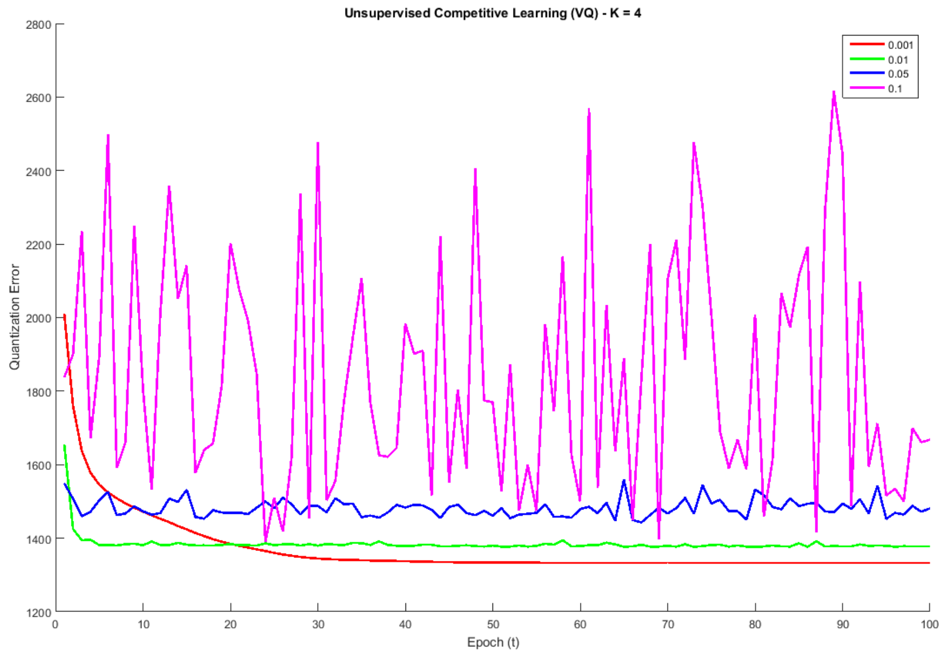


Figure 4: k = 4, learning rate values of 0.001, 0.01, 0.05, 0.1 and up to 100 epochs

During most runs for k = 2, it is possible to see how the learning rate directly affects the quantization error. For all executions, $\eta$ = 0.001 and $\eta$ = 0.01 had similar final results, even though the former would

take more epochs to stabilize (around 10 epochs versus 1). Also, for all execution with k = 2, $\eta$ = 0.1 was much more unstable than $\eta$ = 0.05. Therefore, the higher the $\eta$ value, the more unstable the curve becomes, as the prototypes get pulled a lot during the execution of the algorithm.
For k = 4 most results resemble Figure 4, following the same learning rate behaviour as k = 2 runs. For $\eta$ = 0.001, around 30 epochs were needed to reach a stable result. But given the random initialization aspects of the algorithm and ill fitting of the dataset for 4 clusters, situations where 3 prototypes landed on the same cluster generated interesting results, as seen below.

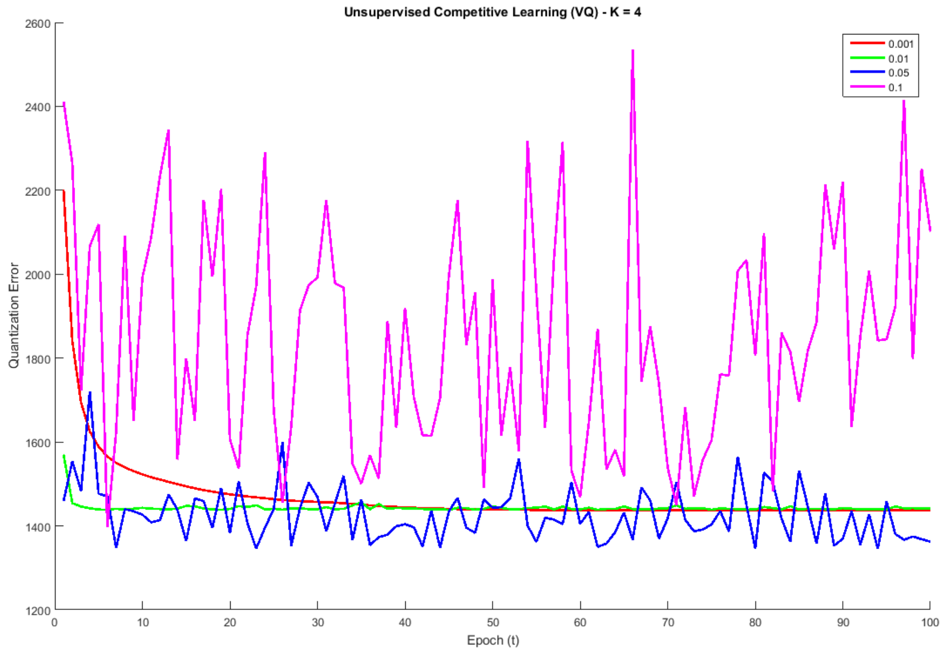


Figure 5: In this run, $\eta$ = 0.05 had a better overall performance than smaller values. It is also interesting to see that for $\eta$ = 0.001, around 40 epochs were needed to reach a stable configuration, meaning that the initialization state was very inadequate.

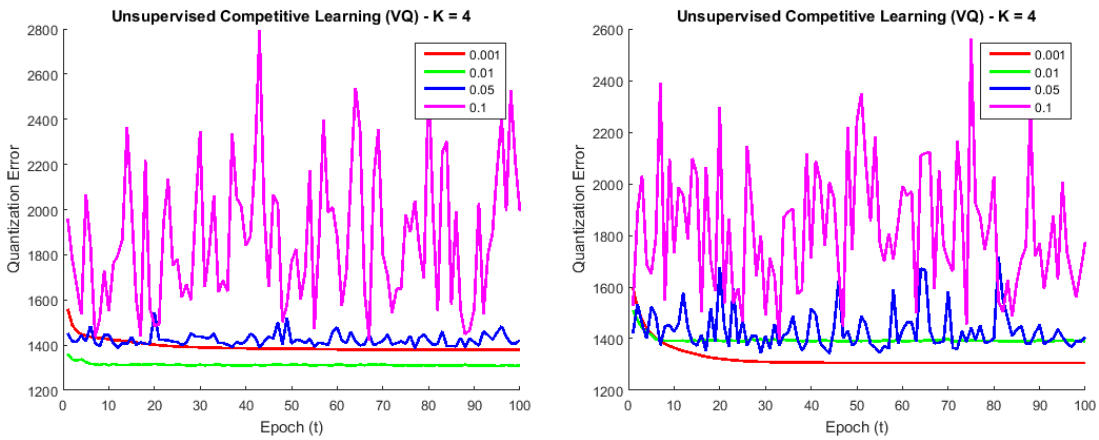


Figure 6: $\eta$ = 0.01 had a considerably lower quantization error on the plot on the left whilst $\eta$ = 0.001 had a better performance on the right plot

It is not possible to come to a conclusion on what is the best learning rate value, but it is reasonable to say that smaller values of $\eta$ (e.g. 0.001 and 0.01) have a more stable quantization error curve, as they don't get pulled around the dataset as much, and tend to reach a local minima. The trade off is that it takes more epochs, therefore more computation, to arrive at a good result.

# Division of work

Since there's only one exercise, the group worked together on the code.

# Appendix:

Listing 1: This is the ulvq.m file

```
function QEList = ulvq( data, K, lr, tMax )

% Seed based on clock time
rng('shuffle');

% Attach class identifier
data(:,3) = -1;

% Generate quantization error list
QEList = [];

% Generete prototype list of size K
protoList = [];
r = randperm(length(data));
for i = 1:K
    protoList = [protoList; [data(r(i),1) data(r(i),2)]];
end


for t = 1:tMax
    % Random permutation of examples
    permutedList = randperm(length(data));
    for i = permutedList
        % For each example, find nearest prototype
        nP = nearestPrototype(data(i, :), protoList);
        % Assign it to the prototype (for plotting and calculating
        % quatization error)
        data(i,3) = nP;
        % Move prototype next to example by a given learning rate
        protoList = moveP(data(i, :), protoList, nP, lr);
    end
    % Plots!
    % figure;
    % colorPoint(data);
    % plotPrototypes(protoList);
    QEList = [QEList  qError(protoList, data)];
end

% Plot of quantization error
% figure;
% plot(1:length(QEList), QEList);
end
```

Listing 2: This is the qError.m file

```
% Calculates quatization error
function qE = qError( protoList, data )
    qE = 0;
    for j = 1:length(protoList)
```

```
        p = protoList(j,:);
        for i = 1:length(data)
            if (j == data(i,3))
                qE = qE + ((p(1)-data(i,1))^2 + (p(2)-data(i,2))^2); % squared euclidian distance
            end
        end
    end
end
```

Listing 3: This is the plotPrototypes.m file

```
function [ output_args ] = plotPrototypes( protoList )

l = length(protoList);
hold on;

% Not best way to do it, but we couldn't get other methods to work
scatter(protoList(1,1),protoList(1,2), 'd','MarkerEdgeColor','k',...
'MarkerFaceColor','r', 'LineWidth',1.5)
if (l == 1) return; end
scatter(protoList(2,1),protoList(2,2), 'd','MarkerEdgeColor','k',...
'MarkerFaceColor','g', 'LineWidth',1.5)
if (l == 2) return; end
scatter(protoList(3,1),protoList(3,2), 'd','MarkerEdgeColor','k',...
'MarkerFaceColor','b', 'LineWidth',1.5)
if (l == 3) return; end
scatter(protoList(4,1),protoList(4,2), 'd','MarkerEdgeColor','k',...
'MarkerFaceColor','c', 'LineWidth',1.5)


end
```

Listing 4: This is the nearestPrototype.m file

```
function nearestIndex = nearestPrototype( e, pList )
%NEARESTPROTOTYPE Finds nearest prototype using squared euclidian distance
    nearestDist = intmax;
    nearestIndex = -1;
    for i = 1:length(pList)
        if ((e(1)-pList(i,1))^2 + (e(2)-pList(i,2))^2 < nearestDist)
            nearestDist = (e(1)-pList(i,1))^2 + (e(2)-pList(i,2))^2;
            nearestIndex = i;
        end
    end
end
```

Listing 5: This is the moveP.m file

```
function pList = moveP( e, pList, pIndex, lr )
%MOVEP moves prototype towards example
    p = pList(pIndex,:);
    if (p(1) < e(1))
        pList(pIndex, 1) = p(1) + (e(1) - p(1))*lr;
    else
        pList(pIndex, 1) = p(1) - (p(1) - e(1))*lr;
    end

    if (p(2) < e(2))
        pList(pIndex, 2) = p(2) + (e(2) - p(2))*lr;
    else
        pList(pIndex, 2) = p(2) - (p(2) - e(2))*lr;
    end


end
```

Listing 6: This is the l6.m

```
close all;

data = load('w6_1x.mat');
data = data.w6_1x;

% k = 2
QE0 = ulvq(data, 2, 0.001, 100);
QE1 = ulvq(data, 2, 0.01, 100);
QE2 = ulvq(data, 2, 0.05, 100);
QE3 = ulvq(data, 2, 0.1, 100);

figure;
hold on;
plot (1:100, QE0, 'r', 'LineWidth', 2);
plot (1:100, QE1, 'g', 'LineWidth', 2);
plot (1:100, QE2, 'b', 'LineWidth', 2);
plot (1:100, QE3, 'm', 'LineWidth', 2);
xlabel('Epoch (t)');
ylabel('Quantization Error');
legend('0.001', '0.01', '0.05', '0.1');
title('Unsupervised Competitive Learning (VQ) - K = 2');


% k = 4
QE0 = ulvq(data, 4, 0.001, 100);
QE1 = ulvq(data, 4, 0.01, 100);
QE2 = ulvq(data, 4, 0.05, 100);
QE3 = ulvq(data, 4, 0.1, 100);

figure;
hold on;
plot (1:100, QE0, 'r', 'LineWidth', 2);
plot (1:100, QE1, 'g', 'LineWidth', 2);
plot (1:100, QE2, 'b', 'LineWidth', 2);
plot (1:100, QE3, 'm', 'LineWidth', 2);
xlabel('Epoch (t)');
ylabel('Quantization Error');
legend('0.001', '0.01', '0.05', '0.1');
title('Unsupervised Competitive Learning (VQ) - K = 4');
```

Listing 7: This is the colorPoint.m file

```
function [ output_args ] = colorPoint( data )
%COLORPOINT Summary of this function goes here
%   Detailed explanation goes here
for i = 1:length(data)
    hold on;
    if (data(i,3) == 1)
        scatter(data(i,1),data(i,2), '.r');
    elseif (data(i,3)== 2)
        scatter(data(i,1),data(i,2), '.g');
    elseif (data(i,3) == 3)
        scatter(data(i,1),data(i,2), '.b');
    elseif (data(i,3) == 4)
        scatter(data(i,1),data(i,2), '.c');
    end
end
end
```