Introduction to Intelligent Systems
Lab 6

Team 15:
Sara Bardají Serra and Albert Sallés Torruella

October 2019

Introduction

In this assignment we are asked to implement a Learning Vector Quantization algorithm (LVQ1). This is a type of prototype-based supervised classification algorithm.

Firstly, let's mention what supervised learning consists of. Supervised learning is the machine learning task of a function that maps an input to an output, therefore creating input-output pairs.

The basic idea of how it works is as follows:

- The first step is to initialize K prototype vectors
- Then take points in a random order and present them as examples
- The following step is to identify the closest prototype for each point to identify the winner
- Finally, if the example belongs to the same class as the winner, then bring the prototype closer, otherwise, move it further away.

The last three steps form an epoch of the algorithm, and must therefore be repeated as many times as epochs we want the algorithm to perform.

Methodology

The first thing to do is to implement the Learning Vector Quantization function following the steps from the assignment. Our function will have the following signature:

$$LVQ1(data, k, n, tmax)$$

Therefore, the function has four parameters, where the first one is the dataset of size N, k is the number of prototypes per class, n is the learning rate and t_{max} is the minimum amount of epochs in a row that must have the same training error in order to be considered there is convergence. The structure of the code and the steps it performs are as follows:

- First select k random points from each class from the dataset. These points will now be used as the prototypes.
- Then shuffle the points to randomly permutate the order in which the points will be presented as examples.
- Next, for every point calculate the distance to every prototype, and take the prototype to which it is nearest as the winner.
- Once the winner is found, move the winner closer to the point it is winner to if they both belong to the same class, otherwise, move the winner further away. The distance is moved closer or further away in n (learning rate) times the distance between the point and the winner.

When these steps are fulfilled for all points, an epoch has been completed. Therefore, we are performing t_{max} epochs of no improvement.

When an epoch has been completed, the misclassification error is computed in order to later plot the learning curve.

Once the implementation of the LVQ1 algorithm is completed, we run the function first with one prototype per class and then with two prototypes per class.

Results

In this assignment, we are given a dataset of 100 two-dimensional feature vectors. We assume that the first 50 points belong to class 1 and the second 50 belong to class 2.

Firstly, we have plotted the whole dataset in Fig. 1 to have an idea of how the points are distributed.

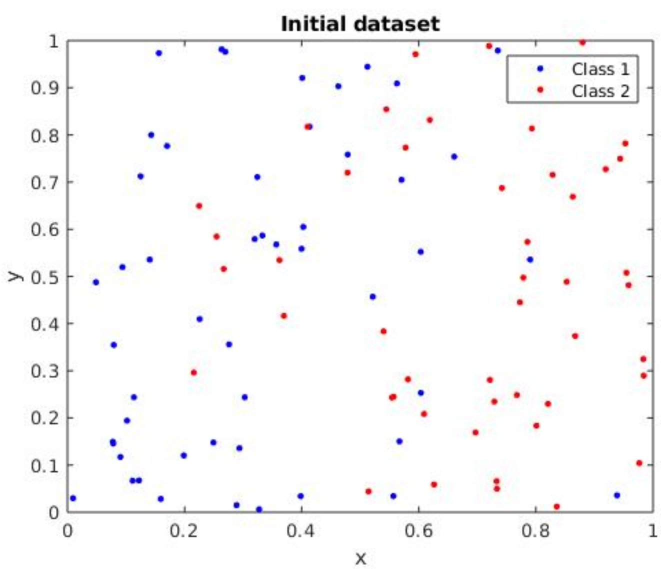

Figure 1: Given dataset distribution

As we can see in the plot, the points are very sparse, which means there are no clear clusters.

Next, we perform our first experiment with one prototype per class, learning rate of 0.002 as explained in the assignment and a maximum of 50 epochs for the convergence. The results are shown as follows:

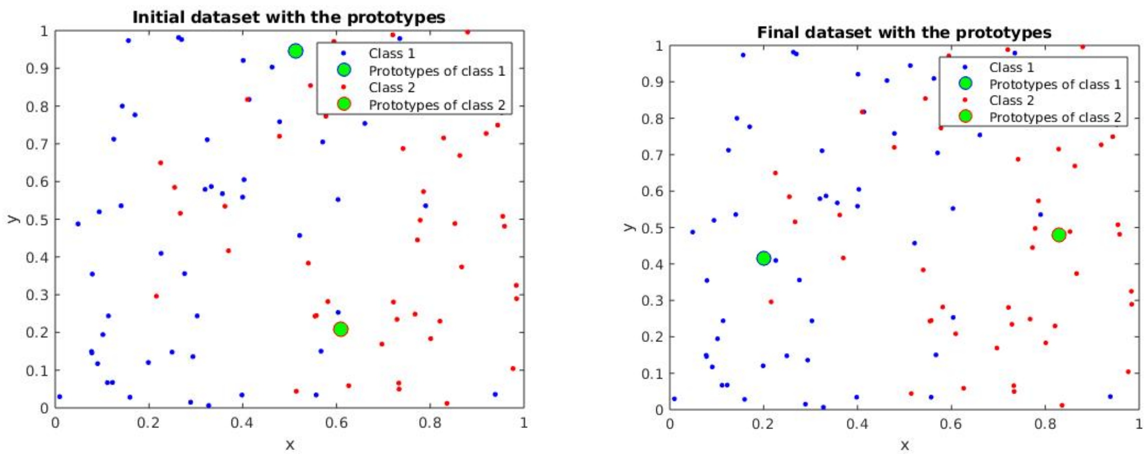

Figure 2: Initial (left) and final (right) distribution of the prototypes

So, in Fig. 2 we can see that the prototypes' final position divides the dataset in two halves. This happens every time we run the algorithm, therefore the two main clusters are in each side of the plot. Also, we can have a look at the learning curve in Fig. 3.

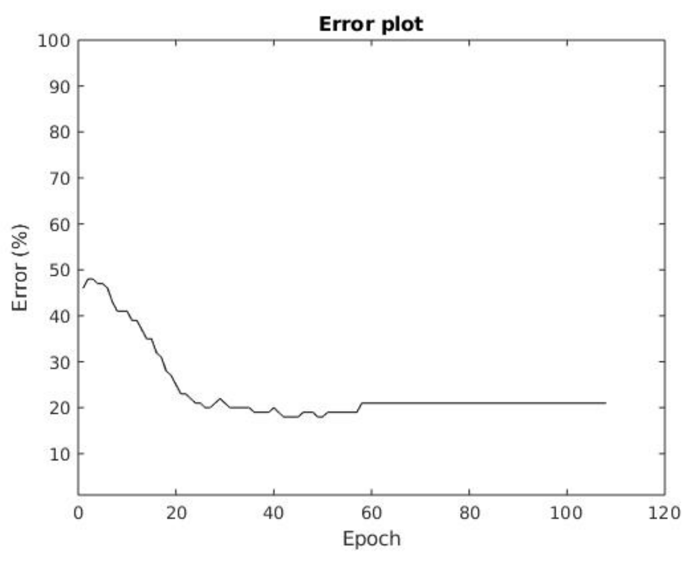

Figure 3: Learning curve of the first experiment

In the plot above, we can see that the training error converges in 20% of training error after 60 epochs.

Now, to compare results, we will perform a second experiment. For this experiment, we use the same configuration as in the previous experiment but with two prototypes per class instead. The results are shown as follows:

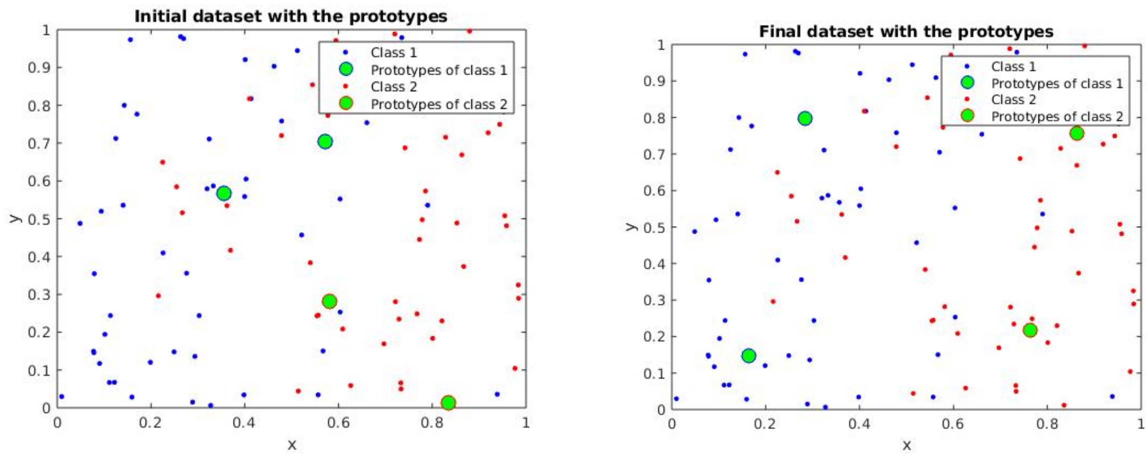

Figure 4: Initial (left) and final (right) distribution of the prototypes

So, in Fig. 4 we can see that the prototypes' final position divides the dataset in two halves again. Although this time the division is a little more complex.

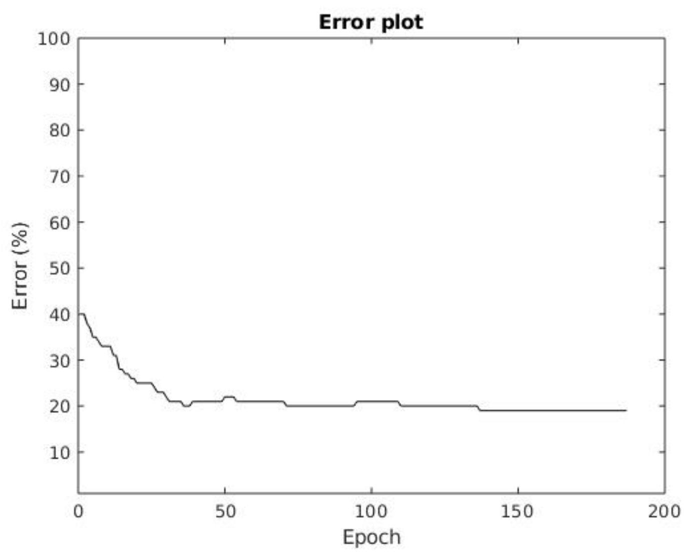

Figure 5: Learning curve of the second experiment

As shown in Fig. 5, the learning curve is now much flatter. However, the training error still converges in 20% of training error, this time at 150 epochs, even thought it reached that value at about 70 epochs. Moreover, in Fig. 6 and 7 we observe how the learning curves become less steep as we increase the number of prototypes per class. Furthermore, we can see that although the initial misclassification error decreases as the number of prototypes per class increases, the final training error does not reach a value below 20% except for when there are 8 and 16 prototypes per class, in those cases the training error is slightly lower than 20%. Then the accuracy of the clustering improves a little bit. In exchange, it takes more epochs for the training error to become stable.

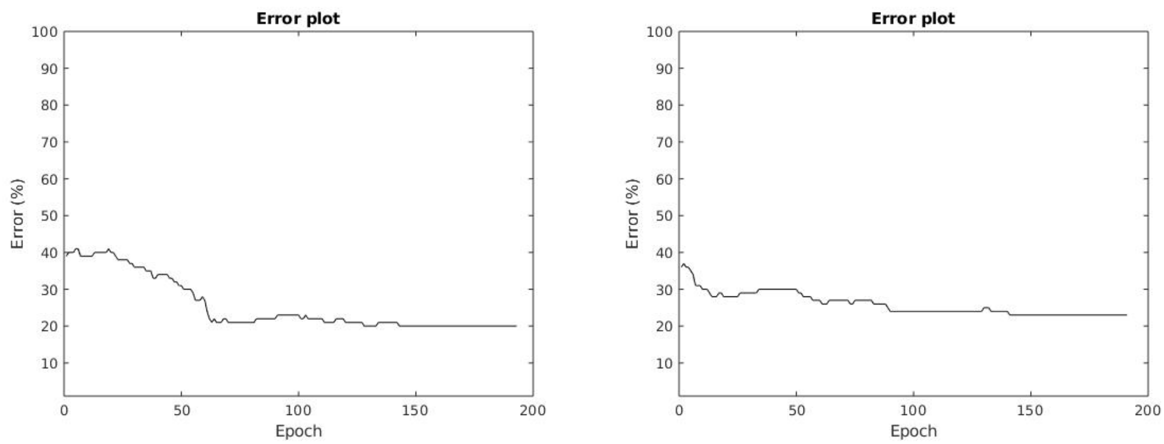

Figure 6: Learning curves for 3 prototypes per class (left) and 4 prototypes per class (right)

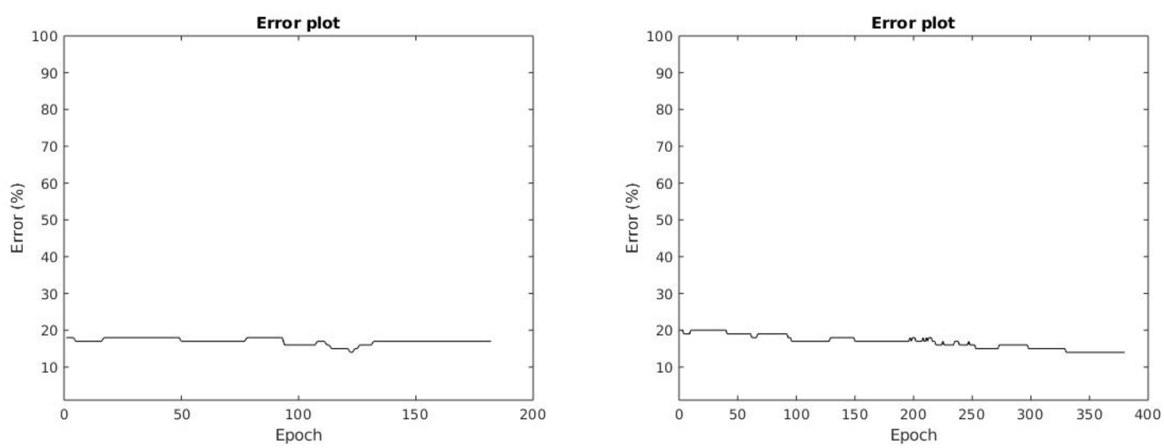

Figure 7: Learning curves for 8 prototypes per class (left) and 16 prototypes per class (right)

Coding details

For this assignment, we have coded the following function in Matlab to help us find the results of the experiments. The algorithm and main steps of the code below have already been explained before. However, there are two steps that need some explaining. First, in order to count how many times the error is stable, we created a counter to store the number of times that the error has not changed from one epoch to the next, but if the error does change, then this counter starts over from 0. Also, there is another tricky part, in which we need to know whether a data point and its winner belong to the same class. To do so, we use the indexes of the data point and the winner, as we know that the first half of the dataset and prototypes belong to class 1 whereas the other half belongs to class 2. [A]

Work division

During the realization of this assignment, we have been working and making decisions together, so we have divided the work equally. Also, the both of us have written and checked this report.

A Code

```
% Function to apply LVQ1 algorithm to a dataset
%  data is the entire dataset in a matrix
%  k is the number of prototypes per class
%  n is the learning rate
%  tmax is the minimum number of epochs to consider the error stable

function LVQ1(data, k, n, tmax)

    % We have 2 classes so we have 2*k prototypes
    k = k*2;

    % Size of the whole dataset
    N = size(data, 1);

    points_class1 = data(1:N/2,:);
    points_class2 = data((N/2 + 1):end,:);

    prototypes = [datasample(points_class1, k/2) ; datasample(points_class2, k/2)];

    % Plot initial state of the data
    figure
    plot(points_class1(:, 1), points_class1(:, 2), 'b.', 'MarkerSize', 10)
    hold on
    plot(prototypes(1:k/2, 1), prototypes(1:k/2, 2), 'bo', 'MarkerFaceColor','g', 'MarkerSize', 10)
    plot(points_class2(:, 1), points_class2(:, 2), 'r.', 'MarkerSize', 10)
    plot(prototypes(k/2+1:end, 1), prototypes(k/2+1:end, 2), 'ro', 'MarkerFaceColor','g', 'MarkerSize', 10)
    title('Initial dataset with prototypes')
    xlabel('x')
    ylabel('y')
    legend('Class 1', 'Prototypes of class 1', 'Class 2', 'Prototypes of class 2')
    hold off

    % List of the error in each epoch
    error = [];
    t = 1;
    equal_epochs = 0;
    % Loop over the epochs
    while equal_epochs ~= tmax
        % Add one element to the error list
        error = [error 0];

        % We shuffle the indices
        shuffled_indexes = randperm(N);

        for idx = shuffled_indexes
            distances = pdist2(data(idx,:),prototypes);
            [~, p] = min(distances);
```

```
            % If the winner belongs to class 1
            if p <= k/2
                % If the data point belongs to class 1
                if idx <= N/2
                    % Move it closer
                    prototypes(p,:) = prototypes(p,:) - (n*(prototypes(p,:) - data(idx,:)));
                % If the data point belongs to class 2
                else
                    % Move it further
                    prototypes(p,:) = prototypes(p,:) + (n*(prototypes(p,:) - data(idx,:)));
                end
            % If the winner belongs to class 2
            else
                % If the data point belongs to class 1
                if idx <= N/2
                    % Move it further
                    prototypes(p,:) = prototypes(p,:) + (n*(prototypes(p,:) - data(idx,:)));
                % If the data point belongs to class 2
                else
                    % Move it closer
                    prototypes(p,:) = prototypes(p,:) - (n*(prototypes(p,:) - data(idx,:)));
                end
            end
        end

        for i = 1:N
            distances = pdist2(data(i,:),prototypes);
            [~, p] = min(distances);

            if i <= N/2
                if p > k/2
                    error(t) = error(t) + 1;
                end
            else
                if p <= k/2
                    error(t) = error(t) + 1;
                end
            end
        end

        error(t) = (error(t)*100)/N;
        if t > 1 && error(t-1) == error(t)
            equal_epochs = equal_epochs + 1;
        else
            equal_epochs = 0;
        end
        t = t + 1;
    end
```

```

    % Plot final state of the data
    figure
    plot(points_class1(:, 1), points_class1(:, 2), 'b.', 'MarkerSize', 10)
    hold on
    plot(prototypes(1:k/2, 1), prototypes(1:k/2, 2), 'bo', '
       MarkerFaceColor','g', 'MarkerSize', 10)
    plot(points_class2(:, 1), points_class2(:, 2), 'r.', 'MarkerSize', 10)
    plot(prototypes(k/2+1:end, 1), prototypes(k/2+1:end, 2), 'ro', '
       MarkerFaceColor','g', 'MarkerSize', 10)
    title('Final dataset with the prototypes')
    xlabel('x')
    ylabel('y')
    legend('Class 1', 'Prototypes of class 1', 'Class 2', 'Prototypes of
       class 2')
    hold off

    % Plot quantization error or all epochs
    figure
    plot(1:numel(error), error, 'k')
    xlabel('Epoch')
    ylabel('Error (%)')
    title('Error plot')
    ylim([1,100])
    hold off

end
```

Code