Introduction to Intelligent Systems
Lab Session 5

Group 30
Sergio Calogero Catalano (s3540294) & Emmanouil Gionanidis (s3542068)

October 16, 2017

Introduction

Learning Vector Quantization is widely used in Supervised Learning due to its flexibility, simplicity and efficiency, as it is easy to implement and highly customizable (there are many possible variations to the algorithm, some of which have also been implemented for this assignment). The basic LVQ1 algorithm differs from the previous Unsupervised Learning VQ only in the update step, since, if the nearest prototype is not from the same class as the point we're considering, the prototype is distanced from that point.
We study a simple situation with only two classes, but we should keep in mind that this simple model is the basis of multi-class situations, as most of the algorithms usually reduce to an update of the two prototypes which currently define the decision boundary in each step. Therefore we can analyze this simple situation to gain insight into a broader class of problems.

Remarks

The criterion chosen to determine when the learning curve is "approximately constant" requires to compute the standard deviation of the last misclassification rates and to determine when this value is smaller than a limit, chosen differently for each plot (mostly based on the learning rate, because with a large learning rate we keep having a large variation in the training error, even after several epochs). In the following sections, we sometimes refer to the learning rate as η and to the maximum number of epochs as t_max, especially when referencing some matlab code. We also noticed that sometimes the learning curve becomes approximately constant, but after many epochs the misclassification rate changes significantly. Therefore we chose to determine a stopping point for the algorithm with the above criterion, but we also decided to show what would have happened if we hadn't stopped it: that is why in most of the plots there is a red vertical line, which shows when our stopping criterion would have normally stopped the process. In many executions, the outcome, as expected, has been that we could have just stopped where we decided, but in some cases the number of misclassifications reduced significantly after a period of constant behavior. Also note that every plot which displays the points and the prototypes, colored differently to highlight their classes and classification, contains a curve for each prototype, which represents its trajectory from its initialization point to its final position. The trajectory has been plotted using the code in **Listing 1** , which is execute after each epoch and draws the line between each old position of the prototypes and their new one, only if trajectory_on is set to 1.

Listing 1: *Code to draw the trajectory*

```
if trajectory_on == 1
    figure(fig(1)+plot_results);
```

```
        hold on;
        for i=1:K
            line([prototypes(i,1) old_prototypes(i,1)],[prototypes(i
    ,2) old_prototypes(i,2)]);
        end
        hold off;
    end
```

The complete matlab code is available in **Appendix A** .

Learning Vector Quantization

For this assignment we try different values for the number of prototypes, learning rate and number of epochs, to see how they influence the misclassification rate. But we should also keep in mind that the goal of a supervised learning vector quantization algorithm is to reduce the generalization error, and not the training error. Therefore we can accept results which don't represent a minimum value of the misclassification rate, to avoid overfitting and loss of generalization.
We executed lvq with a η of 0.002 and 2 prototypes, obtaining the prototypes shown in **Figure 1** . The misclassification rate reaches a small value very fast, but then it keeps changing while staying near that value, never really stabilizing, as shown in **Figure 2** . This happens because the order in which the example points are analyzed can differ a lot in each epoch and can move the prototypes away from the relative balance they have found.

Example points and prototypes after lvq execution

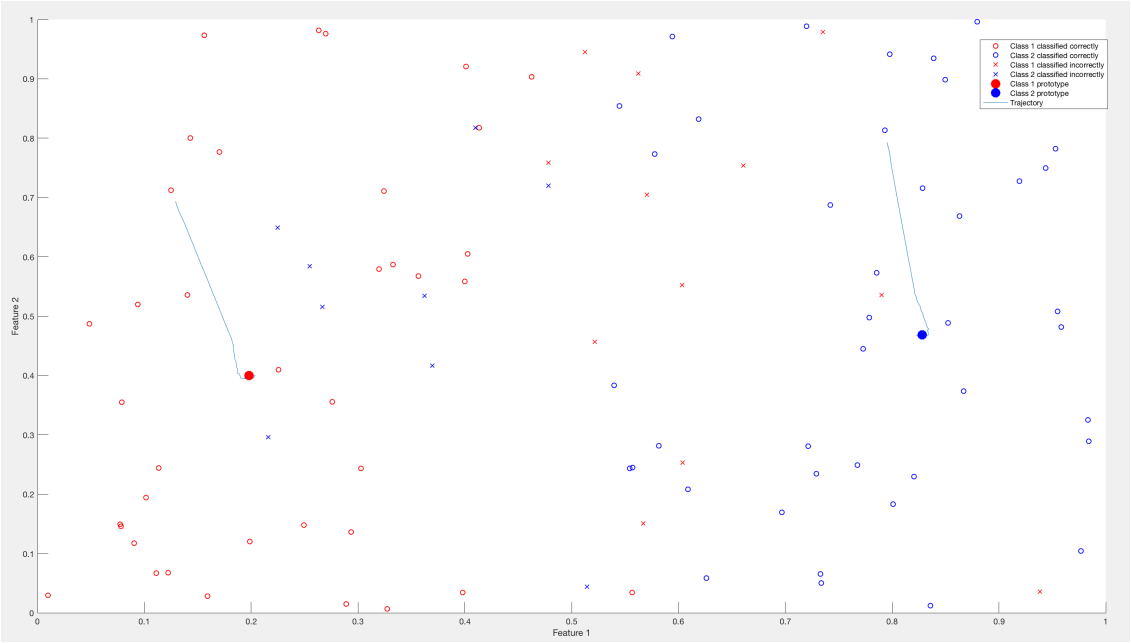

Figure 1: *shows feature 1 vs feature 2*

Learning curve with 2 prototypes and a 0.002 learning rate

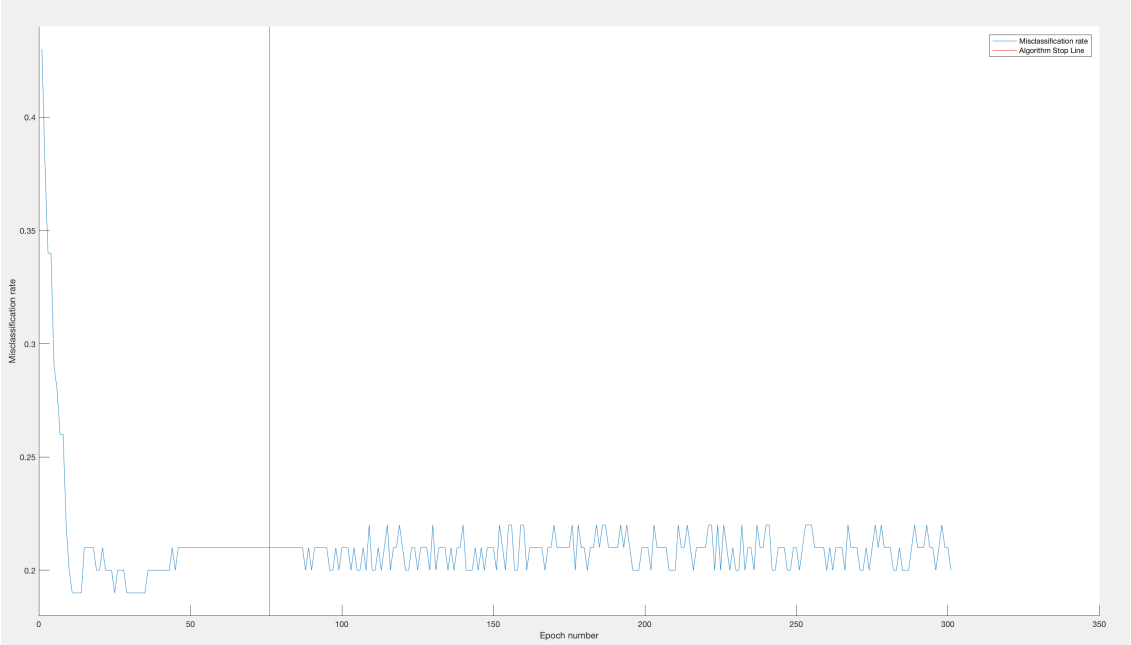

Figure 2: *shows the epoch number vs the misclassification rate*

We could avoid that behavior by decreasing the learning rate, but then the algorithm would require more epochs to reach good results, as each prototype would be updated less during each iteration. To show the influence of the learning rate on the misclassification rate, we created the plots in **Figure 3** . **Figure 3** shows the learning curve obtained after various executions of lvq(where h stands for learning rate and k for the number of prototype) with different values of η and 2 prototypes. The number of epochs used is 5000, because with very small learning rates it needs significantly more epochs to reach a good distribution of prototypes. What can be seen in **Figure 3** is that, decreasing the learning rate also leads to less wobbly learning curves, since it is harder to leave a local minimum with a small learning rate.

Learning curves with 2 prototypes and various learning rates

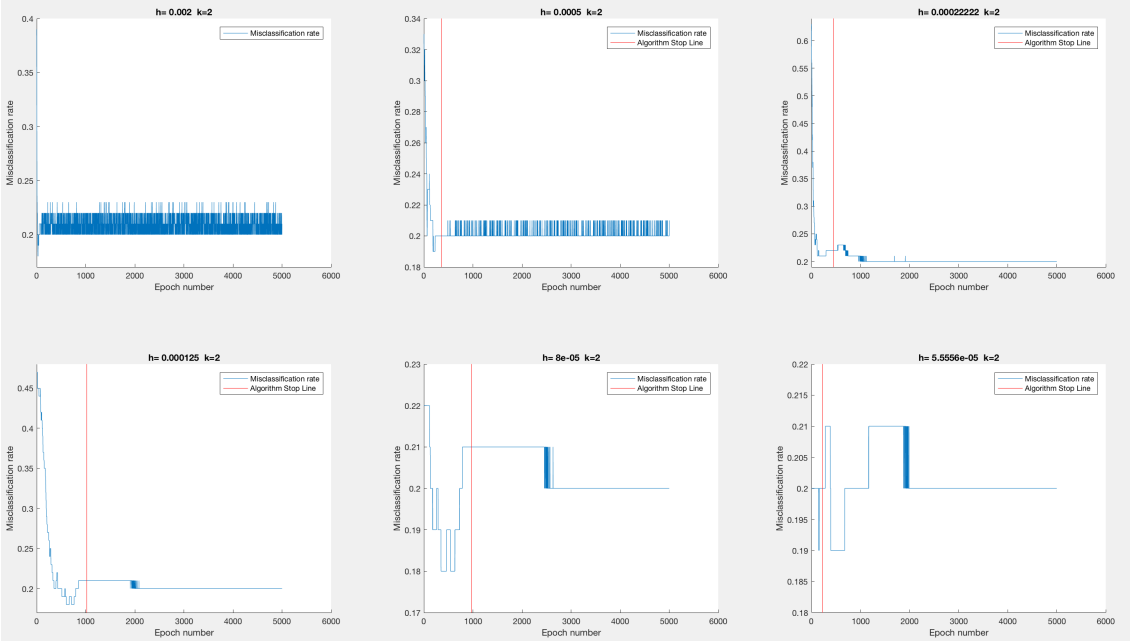

Figure 3: *shows the epoch number vs the misclassification rate for various learning rates*

To see how the number of prototypes changes the outcome of lvq, we tried to increase the number of prototypes used, expecting to obtain smaller misclassification rates: but that doesn't always happen, as the process particularly depends on the randomness of the updates order and on the initial position of the prototypes with respect to the data. We also expected to need more epochs of training, because we have more prototypes to position, and each prototype moves less (as the average distance of the points from the prototypes is smaller, also the average update is smaller). That is shown in **Figure 4** , where the number of prototypes chosen is 12, and the number of epochs necessary to reach a good point is much larger (at least 2000 epochs in this plot).

Learning curve with 12 prototypes and a learning rate of 0.0005

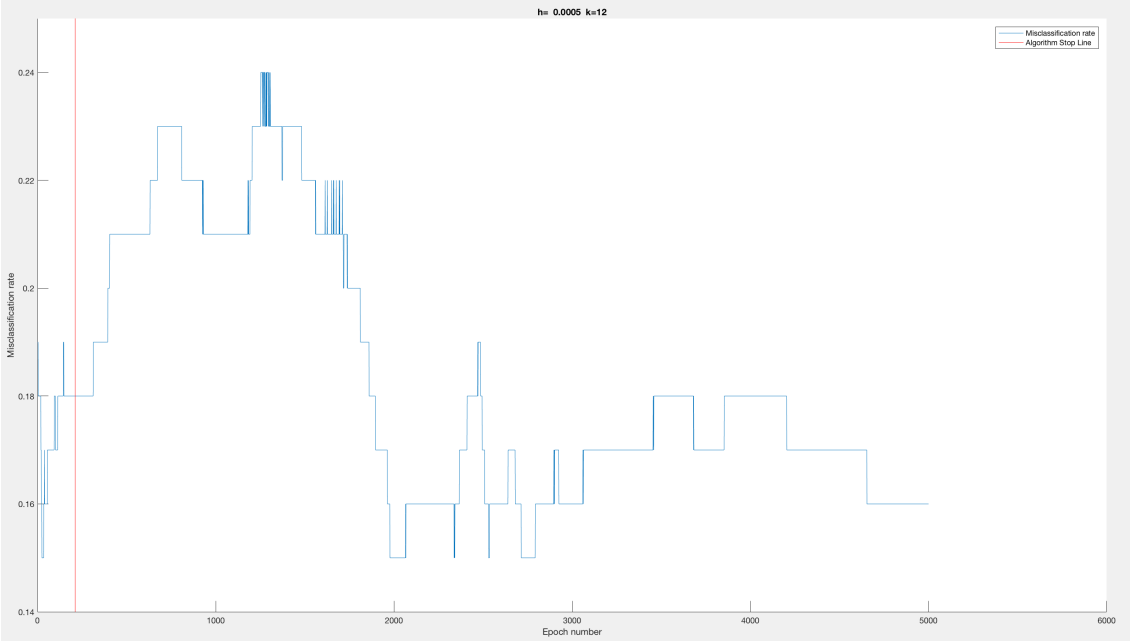

Figure 4: *shows the learning curve for a large number of prototypes*

Furthermore, the learning dynamics significantly depend on the precise initial positions of prototypes, since, if they're initialized near a local minimum, the algorithm is likely to end up in it (especially with a small learning rate). That's why we executed the algorithm twice, with the same parameters, changing only the initialization points of the prototypes, which were randomly chosen in the first plot and positioned near the class-conditional means in the second plot of **Figure 5** . **Figure 5** shows that the final value of the misclassification is almost the same in both plots, but when we initialize the prototypes near the class-conditional mean, we generally reach that result faster than with randomly initialized prototypes.

Learning curves with 4 prototypes and a learning rate of 0.001 with different initializations

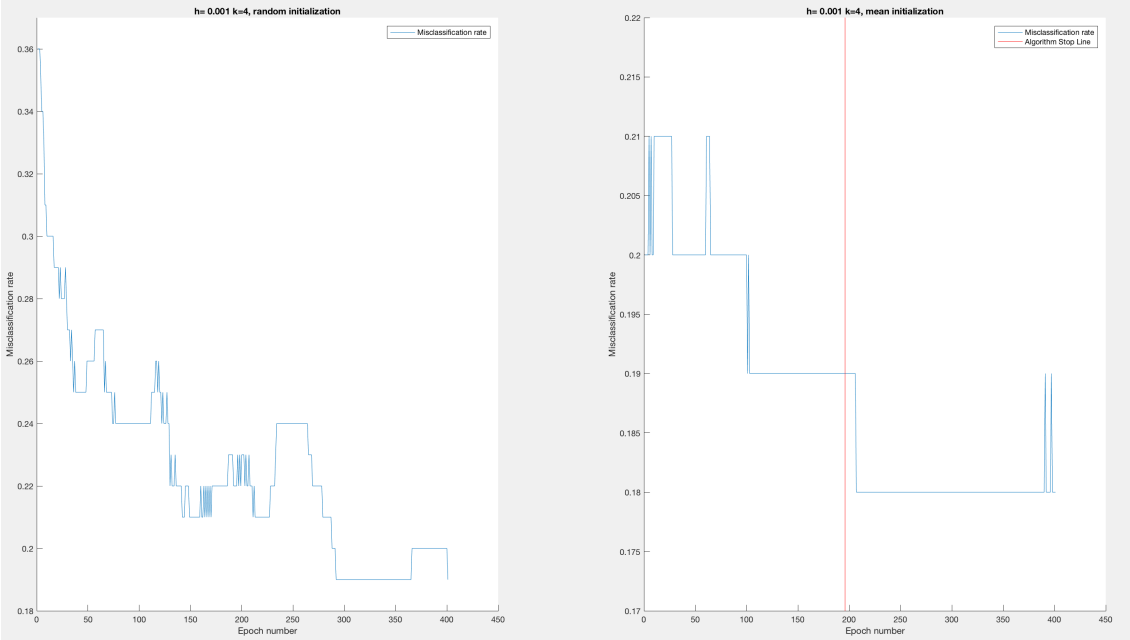

Figure 5: *shows the epoch number vs the misclassification rate for differently initialized executions*

A modification of the lvq algorithm is based on heuristics and aims at a more efficient separation of prototypes which represent different classes. Given a single example, we update two prototypes: the closest one which belongs to the same class as the data, and the closest one that represents a different class. The correct winner is moved towards the data whereas the wrong winner is moved farther away. This approach is called LVQ2. Using lvq2, the resulting learning curve has a larger standard deviation than lvq1 learning curves, since the prototypes move much more and are less likely to stabilize in a local minimum. This effect is increased when there is a clear distinction between the two classes, because the points of one class all push the prototypes of the other class in the same direction and, being the distance from a prototype of a different class often larger than the distance from a prototype of the same class, the pushing distance is more significant than the pulling distance. Thus using plain lvq2 leads to an unstable and divergent behavior, as the prototypes will tend to move away from the data (but the decision boundary will still be useful). There are different possible solutions to this problem: we could update the wrong winner based on its distance from the data, or stop updating it if the distance is larger than a predetermined value; we could define a window, close to the decision boundary, and accept updates only from points that fall into that window or we could just stop the algorithm when it start showing a divergent behavior. The solution we chose is the first one, but instead of reducing the update based on the distance, we reduce the update of a constant value, i.e. we use $\frac{\eta}{4}$ to update the wrong winner whereas we use η to update the correct winner(we call this "modified lvq2" in the plots). This way the learning curve results more stable than lvq1, as shown in **Figure 6** , which plots the learning curves obtained with same parameters, but different algorithms, by initializing the prototypes in the class-conditional means.

Learning curves with 4 prototypes and a learning rates of 0.001 with different algorithms

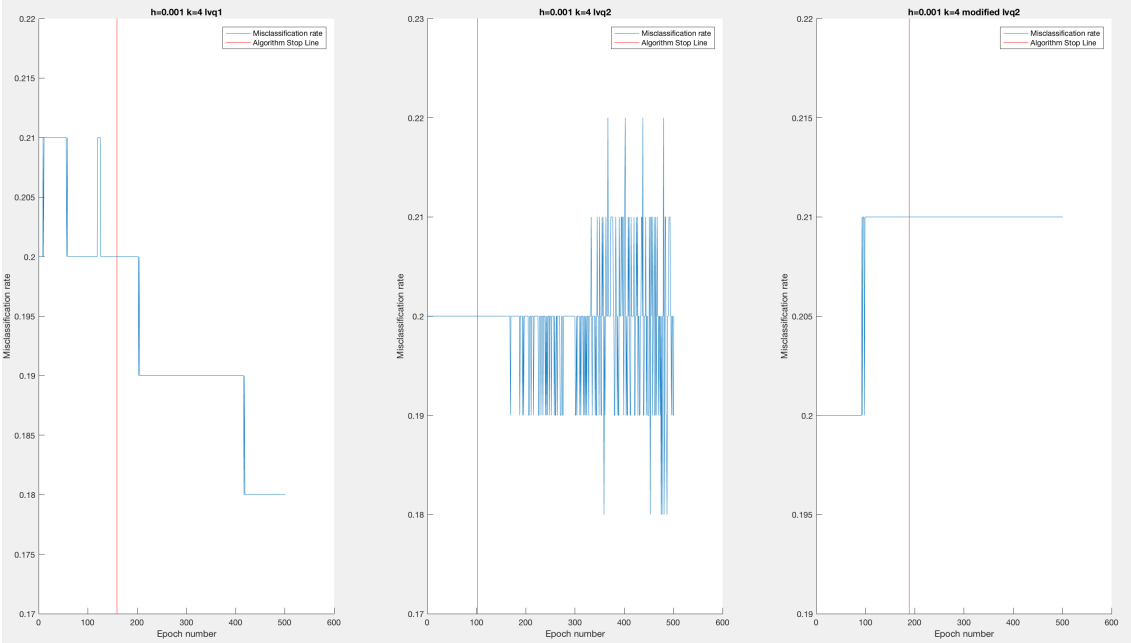

Figure 6: *shows the epoch number vs the misclassification rate for lvq1, lvq2 and modified lvq2, respectively*

The resulting prototype distributions are shown in **Figure 7** for the lvq1 algorithm, in **Figure 8** for the lvq2 algorithm and in **Figure 9** for the modified lvq2: the prototypes of standard lvq2 moved away from the data and are outside the shown space, but they still define a decision boundary which allows an acceptable classification; the prototypes of the other plots show a difference in the prototypes trajectory, as they move more directly to their destination (the trajectory looks like a line, and not a curve) with modified lvq2, while lvq1 seems more unstable from that point of view.

Example points and prototypes after lvq1 execution with 4 prototypes and a learning rates of 0.001

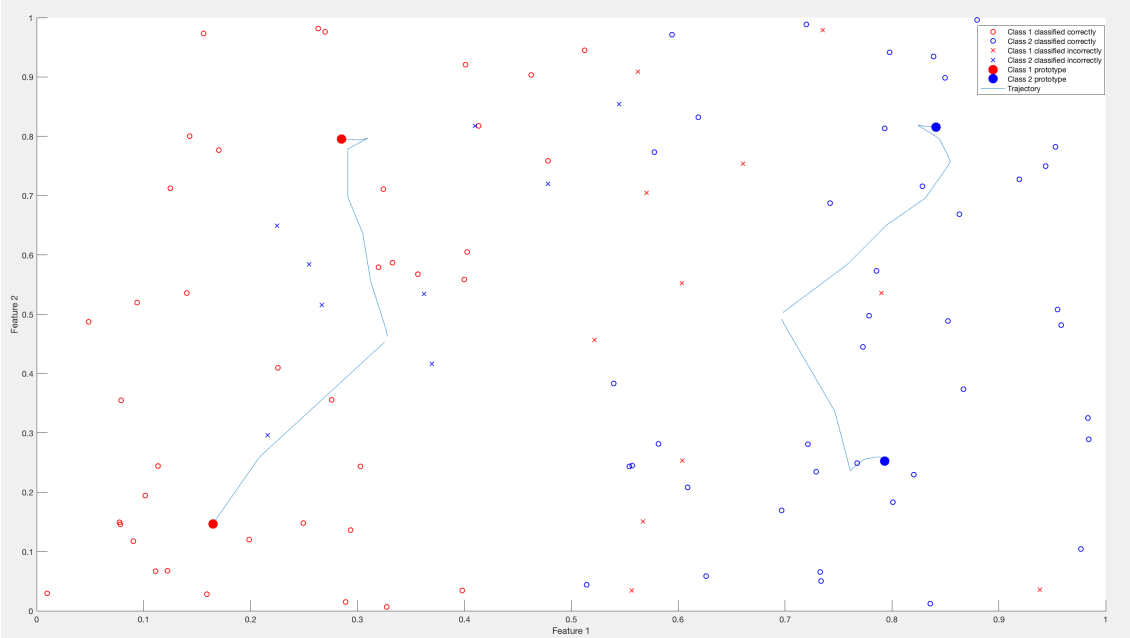

Figure 7: *shows feature 1 vs feature 2 for lvq1, with highlighted prototypes and trajectories*

Example points and prototypes after lvq2 execution with 4 prototypes and a learning rates of 0.001

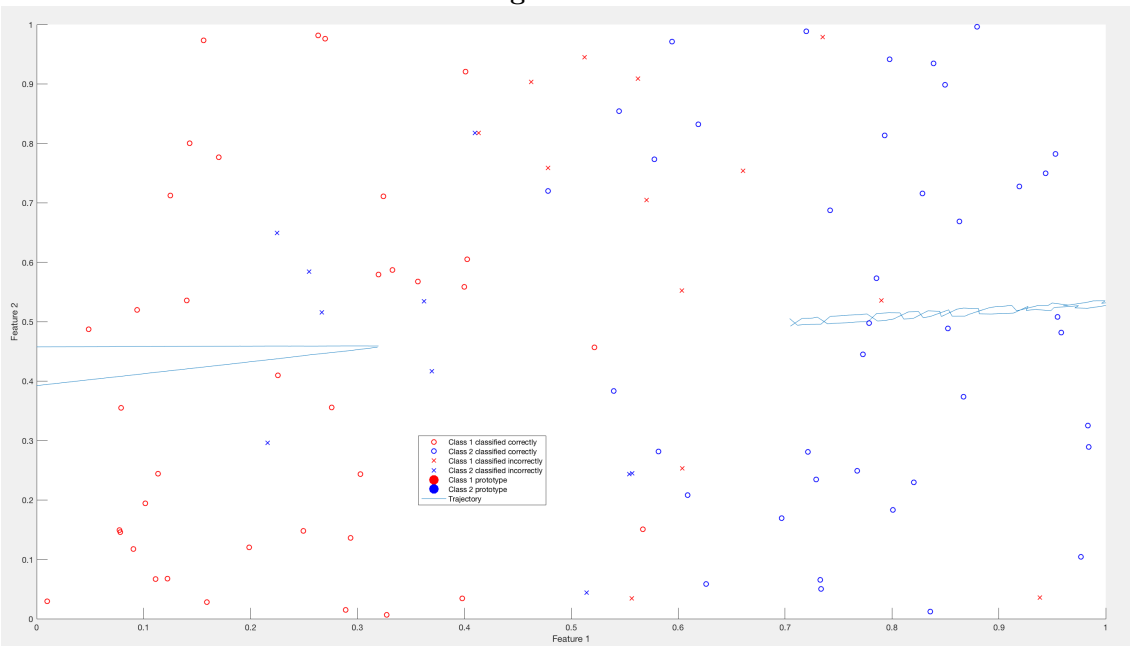

Figure 8: *shows feature 1 vs feature 2 for lvq2, with highlighted prototypes and trajectoriess*

Example points and prototypes after modified lvq2 execution with 4 prototypes and a learning rates of 0.001

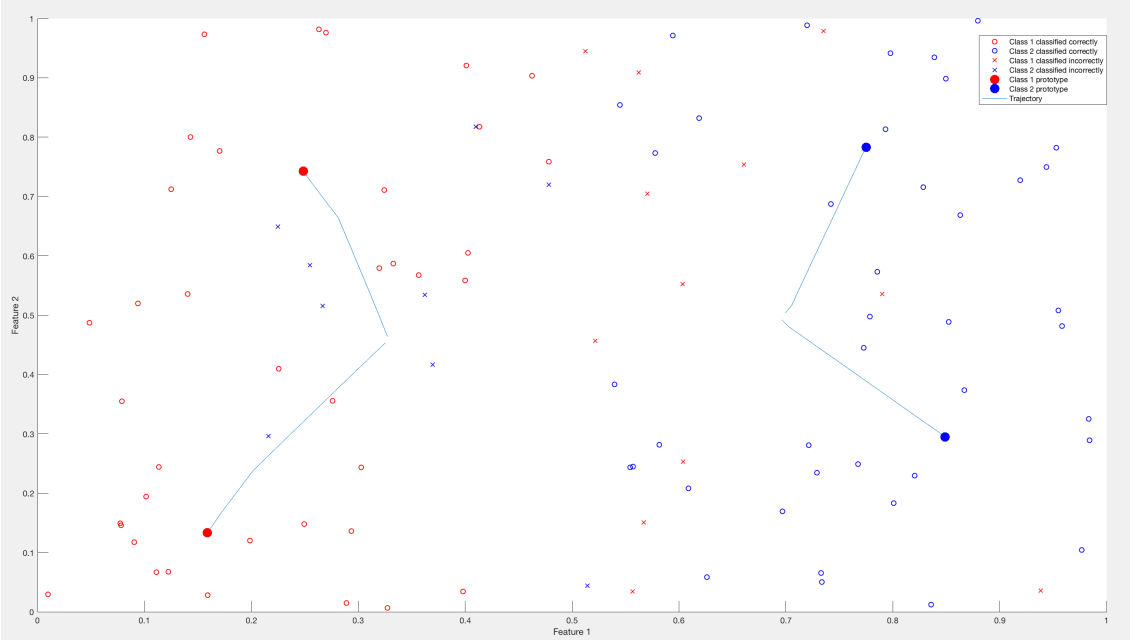

Figure 9: *shows feature 1 vs feature 2 for modified lvq2, with highlighted prototypes and trajectories*

During the analisys of the plots, we also discussed the stability (preciselly steady the misclassification rate is), but we should always keep in mind that we use a lvq algorithm mainly to obtain generalization, which means that it should be able to classify well new points. Thus we should not focus too much on minimizing or on stabilizing the misclassification rate, because, usually, tuning the prototypes to fit too well the training data leads to more classification errors with new general data.

A. Assignment

Listing 2: *This is the file* `script1.m`

```
%Read input data and delete not needed struct
S = load('data_lvq.mat');
vectors = S.w5_1;
clear S;

%Provide first plot: learning curve with 2 prototypes
completeLearningProcess(vectors, 0.002, 2, [1 1 1 1], 100, 0.0045, 300,
    0, 1, 0, 1);

%Provide plots with different learning rates and 2 prototypes
for i = 1:6
    h = 0.002/(i*i);
    completeLearningProcess(vectors, h, 2, [3 2 3 i], 100+i*20, 0.0025,
    5000, 0, 0, 0, 0);
    title(['h= ' num2str(h) '  k=' num2str(k)]);
end

%Provide plots with same learning rate and different numbers of
   prototypes
completeLearningProcess(vectors, 0.0005, 12, [4 1  1 1], 100, 0.002,
   5000, 0, 0, 0, 0);
title('h=0.0005  k=12');

%Provide plots with same parameters but different initialization
   points for the prototypes
completeLearningProcess(vectors, 0.001, 4, [5 1 2 1], 100, 0.0025, 400,
    0, 0, 0, 0);
title(['h= 0.001 k=4, random initialization']);
completeLearningProcess(vectors, 0.001, 4, [5 1 2 2], 100, 0.0025, 400,
    1, 0, 0, 0);
title(['h= 0.001 k=4, mean initialization']);

%Provides plots with same parameters, but the first runs lvq1 and the
   second runs lvq2 and the third one runs modified lvq2
completeLearningProcess(vectors, 0.001, 4, [6 1 3 1], 100, 0.0025, 500,
    1, 1, 0, 1);
title('h=0.001 k=4 lvq1');
completeLearningProcess(vectors, 0.001, 4, [6 1 3 2], 100, 0.0025, 500,
    1, 1, 1, 2);
title('h=0.001 k=4 lvq2');
completeLearningProcess(vectors, 0.001, 4, [6 1 3 3], 100, 0.0025, 500,
    1, 1, 4, 3);
title('h=0.001 k=4 modified lvq2');

%
   ------------------------------------------------------------------------
```

```
% Executes the algorithm with the given parameters.
% M is the data set, for which the first half is class1 and the second
    half is class 2
% h is the learning rate
% K is the number of prototypes (in total, not for each class)
% fig is a vector containing: the figure in which we plot the data(1),
    the row subplots(2), the columns subplots(3) and the current
   subplot number(4)
% analisys_length determines how many misclassification values are
   considered to compute the std deviation
% limit determines the limit value of the standard deviation under
   which we can stop the algorithm. If set to 0 the algorithm stops at
    t_max epochs
% t_max determines the maximum number of epochs. If set to 0, the
   algorithm will stop when the limit requirement is satisfied, or
   when it reaches 1000 epochs (to prevent infinite cycles)
% prototypes_in_mean is 1 if the prototypes are to be initialized in
   the class conditional mean of their class
% trajectory_on is 1 if we want to plot the trajectory of the
   prototypes
% lvq2 is 1 if we want to use the lvq2 update algorithm instead of a
   normal lvq
% plot_results is n if we want it to plot the points with the
   prototypes on figure fig(1)+ n, otherwise it is 0
function completeLearningProcess(M, h, K, fig, analisys_length, limit,
    t_max, prototypes_in_mean, trajectory_on, lvq2, plot_results)
    %Preprocessing
    if t_max == 0
        t_max = 1000;
        stop = 1;
    end

    %Saves in P  the number of data points. In the next computations,
   we assume that M is divided in 2 classes which have the same
   dimension
    P = size(M, 1);

    %Creates a prototypes matrix which has in the first half of the
   rows prototypes for class1 and in the second half prototypes of
   class2
    prototypes = [initializePrototypes(M(1:P/2, :), K/2,
   prototypes_in_mean); initializePrototypes(M(P/2 + 1 : P, :), K/2,
   prototypes_in_mean)];

    %Initializes variables
    misclassifications = zeros(1,t_max);
    t = 1;
    [prototypes, E] = epoch(M, P, K, prototypes, h, 0, lvq2);
    misclassifications(1) = E/100;
    stop = t_max;
```

```
%Executes all epochs
for t = 2:t_max
    old_prototypes = prototypes;
    [prototypes, E] = epoch(M, P, K, prototypes, h, 0, lvq2);
    misclassifications(t) = E/100;

    %Draws trajectory lines if enabled
    if trajectory_on == 1
        figure(fig(1)+plot_results);
        hold on;
        for i=1:K
            line([prototypes(i,1) old_prototypes(i,1)],[prototypes
(i,2) old_prototypes(i,2)]);
        end
        hold off;
    end

    %Computes standard deviation and the stopping point, if enough
 epochs have passed
    if t>analisys_length+1
        val = std(misclassifications(t-analisys_length:t));
        if val < limit
            if stop == 1
                stop = t;
                break;
            else
                stop = t;
                limit =-1;
            end
        end
    end
end

%Execute last epoch
t = t + 1;
if plot_results == 0
    plot_results = -fig(1);
end
[~, E] = epoch(M, P, K, prototypes, h, fig(1) + plot_results, lvq2
);
misclassifications(t) = E/100;

%Plot the learning curve in the appropriate figure/subfigure
figure(fig(1));
subplot(fig(2),fig(3),fig(4));
hold on;
misclassifications_plot = plot(1:t, misclassifications(1:t));
if stop ~= t_max
    stop_line = line([stop stop],[0 1],'Color','red');
    legend([misclassifications_plot stop_line], 'Misclassification
 rate', 'Algorithm Stop Line');
```

```
    else
        legend(misclassifications_plot, 'Misclassification rate');
    end
    xlabel('Epoch number');
    ylabel('Misclassification rate');
    ylim([min(misclassifications(1:t))-.01 max(misclassifications(1:t)
    )+.01])
    hold off;

end

% Function which performs an epoch
% the parameter lvq2 determines which algorithm will be used
function [prototypes, E] = epoch(M, P, K, prototypes, h, fig, lvq2)
    %Initializations and permutation of example data
    groups = [ones(1, 50) 2*ones(1, 50)];
    r1 = randperm(50);
    r2 = randperm(50) + 50;
    M = M([r1 r2],:);
    E = 0;

    %Iterations on the example data
    for i = 1:P
        if lvq2 > 0
            %Find winner prototype class1
            [winner1, distance1] = findWinnerPrototype(M(i,:),
   prototypes(1:K/2, :), K/2);
            %Find winner prototype class2
            [winner2, distance2] = findWinnerPrototype(M(i,:),
   prototypes(K/2+1:K, :), K/2);
            winner2 = winner2 + K/2;

            %Update prototypes according to lvq2 algorithm
            if i <= P/2
                prototypes(winner1,:) = prototypes(winner1,:) - h*(
   prototypes(winner1,:) - M(i,:));
                prototypes(winner2,:) = prototypes(winner2,:) + h/lvq2
   *(prototypes(winner2,:) - M(i,:));
                if distance2 < distance1
                    E = E + 1;
                    groups(i) = groups(i) + 2;
                end
            else
                prototypes(winner1,:) = prototypes(winner1,:) + h/lvq2
   *(prototypes(winner1,:) - M(i,:));
                prototypes(winner2,:) = prototypes(winner2,:) - h*(
   prototypes(winner2,:) - M(i,:));
                if distance1 < distance2
                    E = E + 1;
                    groups(i) = groups(i) + 2;
                end
```

```
            end
        else
            %Find winner prototype
            [winner, ~] = findWinnerPrototype(M(i,:), prototypes, K);

            %Update winner prototype
            if ((winner <= K/2) && (i <= P/2)) || ((winner > K/2) && (
    i> P/2))
                prototypes(winner,:) = prototypes(winner,:) - h*(
    prototypes(winner,:) - M(i,:));
            else
                E = E + 1;
                prototypes(winner,:) = prototypes(winner,:) + h*(
    prototypes(winner,:) - M(i,:));
                groups(i) = groups(i)+2;
            end
        end
    end

    %Draw the points and the prototypes in the feature space
    if fig > 0
        figure(fig);
        hold on;
        g1 = gscatter(M(:,1), M(:,2), groups, 'rbrb', 'ooxx');
        g2 = gscatter(prototypes(:,1), prototypes(:,2), [ones(1,K/2)
    ones(1,K/2)*2],[1 0 0; 0 0 1], '.', 40);
        l = line([0.5 0.5],[0.5 0.500001]);
        ylim([0 1]);
        xlim([0 1]);
        xlabel('Feature 1');
        ylabel('Feature 2');
        legend([g1(1) g1(2) g1(3) g1(4) g2(1) g2(2) l],'Class 1
    classified correctly','Class 2 classified correctly','Class 1
    classified incorrectly','Class 2 classified incorrectly','Class 1
    prototype', 'Class 2 prototype', 'Trajectory');
        hold off;
    end


end

%Function which finds the winner prototype and its distance from the
    data
%example v
function [min_index, min_distance] = findWinnerPrototype(v, prototypes,
     K)
    min_distance = norm(prototypes(1,:) - v);
    min_index = 1;
    for k = 2:K
        distance = norm(prototypes(k,:) - v);
        if distance < min_distance
```

```
            min_distance = distance;
            min_index = k;
        end
    end
end

%Function which initializes prototypes depending on mode
function prototypes = initializePrototypes(M, K, mode)
    if mode == 0 %RANDOM MODE
        p = ceil(rand(1, K) * size(M,1));
        prototypes = M(p, :);
    else %MEAN
        prototypes = zeros(K,2);
        cc_mean = [mean(M(:,1)) mean(M(:,2))];
        for k=1:K
            prototypes(k,:) = cc_mean;
        end
    end
end
```