Introduction to Machine Learning

Group 15
Lubor Budaj (s4167376) & Stefan Uta (s3933563)

October 2, 2022

Introduction

In machine learning, LVQ is a type of supervise learning classification. It is based on comparison of dissimilarity of example data with reference data (prototypes). Below, we firstly describe our approach using LVQ, afterwards we first present, and later interpret the results we obtained as learning curves and scatter plots.

Method

At first, we imported the data set and initialized the variables. Then we initialized each prototype with random example from its corresponding class. We added support for any number of prototypes (although the number shouldn't be higher than the number of examples per class). The number of prototypes can be changed by changing the value of K at line 7 in the code (Listing 1). In case of mean metric being used as starting point, we calculate mean for x and y features for each class and initialize the prototypes with it. This method is supported only for 1 prototype per class.

In the next step, for each example, we determined the winner (the closest prototype). To do this we used simple(squared) Euclidean distance. If the winner is of the same class as the example, we move the winner towards the example, otherwise we move it further from the example. We determine the new coordinates of the prototype by adding (or subtracting) the ratio of x and y distance multiplied by the learning rate to preserve the direction of the move.

After we determine the winner for each example and move it accordingly, we repeat the process until t_{max} (maximum epoch) is reached, or until the error rate becomes approximately constant. To determine this we tried different approaches. First we tried a simple way of stopping any time a worse result is encountered. Although this method sometimes produced better results than our current approach, often it produced very bad results, such as error rate of 40% for $K = 1$. This was caused by getting into a local maximum; there was no way for the program to get out of it. We solved this issue by comparing the running average of last 15 epochs. Anytime the running average worsens, we stop the run. For this reason the minimum number of epochs is 15 as well. Although in some cases this may cause worse results, in general the results are more stable.

Results

For each K (number of prototypes per class) we produced a learning curve and a scatter plot showing position of prototypes among the data.

In Figure 1 the learning curve for one prototype per class is shown. The x axis is the number of epochs and the y axis is the error rate. The learning rate for $K = 2$ can be seen at Figure 2, for $K = 3$ Figure 3 and for $K = 4$ Figure 4. The learning curve for prototypes initialized as the mean of each class can be seen at Figure 5.

At figures Figure 6 Figure 7 Figure 8 Figure 9 the scatter plots for each K can be seen. In these plots the x axis is feature x and y axis is feature y. Red and green colors distinguishes different classes. The smaller points example data. The bigger point are final position of the prototypes. The figure Figure 10 contains the scatter plot for prototypes initialized in mean positions.

Discussion

Based on the inspection of a Figure 6 for data with prototypes for $K = 1$, we can see that there are lot of outliers in the data. It can be seen by some green class points being encircled by red class points and vice versa. From this observation we deduct that the error rate shouldn't be too low for the implementation of LVQ. On the other hand, the data doesn't seem to be random either, so we expect better classification than a coin toss. In regards to error rate in general, our expectations are met with the results - for $K = 1$ the usual error rate is about 25%.

Our next expectation was that for higher number of prototypes, the error rate will become lower. For $K = 2$, the usual error rate is around 20% Figure 2. This meets our expectations. For $K = 3$ Figure 3 and $K = 4$ Figure 4 and increase in number of prototypes leads to further improvement, however, it is not as large as we anticipated. For this reason we tried running the program multiple times, but the gain in error rate was still marginal. We assume that this is caused by the nature of our method (simple LVQ) or relatively small data set. To verify the integrity of our code we tried running it with $K = 50$, where for each example there is an initialized prototype. As expected, we got the perfect match - 0% error rate.

When we initialized the prototypes with the mean of x feature and y feature for each class, we expected to get better results - lower error rate than when it is initialized with random data. However, this wasn't the case. In fact, we were surprised when we saw the learning curve (Figure 5) that is deteriorating. We verified our results by running the program multiple times. In each of our runs the result was similar and there was a clear upward trend for the error rate. Next, we compared the scatter plot of mean initialization Figure 10 method with random initialization scatter plot for $K = 1$ Figure 6. In these 2 plots we can see similar positions of the prototypes. Our theory is that there is a local maximum at mean for each class. LVQ steps out of this local maximum and eventually reaches a similar point for prototype as method with random initialization. This is caused by our method of not stopping the process too soon in case of being stuck in a local minimum, however, for this particular case it result in higher error rate than a standard simple stop-at-any-worse-epoch method.

When comparing the scatter plots for different number of prototypes per class, we expected the prototypes to be at different places. For $K = 1$ Figure 6 we expected the prototypes to be roughly at the middle of each group. This was indeed the case. For $K = 2$ Figure 7 we expected the prototypes to be distributed roughly equally for each group, which again was the case. For

$K = 3$ Figure 8we expected a prototype in the middle for few red data points stacked over there and again it met our expectations. We found the result for $K = 4$ Figure 9 interesting, we didn't expect the red prototype so far right in the 'green' territory.

Division of Workload

In regards to code Lubor did approximately 80% and Stefan did 20%. Regarding the plots, Lubor did 60% and Stefan did 40%. Regarding the report, Lubor did 60% and Stefan did 40%. Although the contribution isn't equal, we are both satisfied with our own our partner's work on this assignment.

References

To make this report we used only the material presented in the lectures. The plots were generated in MATLAB.

Appendix

Listing 1: *This is the file LVQforKprototypes.m*

```
%import
lvqdata = importdata("lvqdata.mat");
%initialization
dimensions = size(lvqdata);
N = dimensions(2) + 1;
P = dimensions(1);
K = 1;
n = 0.002;
t_max = 300;
average_length = 15;
prototypes = zeros(K*2,N);
E = zeros(1,t_max);
%assigning class
for x = 1:P
    if x > P/2
        lvqdata(x,3) = 2;
    else
        lvqdata(x,3) = 1;
    end
end
%random prototype selection
r1 = randperm(50,K);
r2 = randperm(50,K)+50;
for k = 1:2:(2*K)
    prototypes(k,:) = lvqdata(r1((k+1)/2),:);
    prototypes(k+1,:) = lvqdata(r2((k+1)/2),:);
end
%for each epoch
for t = 1:t_max
    %random permutation of examples
```

```
    p = randperm(P);
    classified_correctly = 0;
    %for each example
    for x = 1:P
        %finding closest prototype
        closest_prototype = 1;
        minimal_distance = abs(lvqdata(p(x),1) - prototypes(1,1)) +
   abs(lvqdata(p(x),2) - prototypes(1,2));
        for k = 2:(2*K)
            distance = abs(lvqdata(p(x),1) - prototypes(k,1)) + abs(
   lvqdata(p(x),2) - prototypes(k,2));
            if distance < minimal_distance
                closest_prototype = k;
                minimal_distance = distance;
            end
        end
        if minimal_distance == 0 %no moving in case of a perfect match
            classified_correctly = classified_correctly + 1;
        else
            %distance ratio calculation
            distX = n / minimal_distance * (lvqdata(p(x),1) -
   prototypes(closest_prototype,1));
            distY = n / minimal_distance * (lvqdata(p(x),2) -
   prototypes(closest_prototype,2));
            %direction
            if lvqdata(p(x),3) ~= prototypes(closest_prototype,3)
                distX = -distX;
                distY = -distY;
            else
                classified_correctly = classified_correctly + 1;
            end
            %moving
            prototypes(closest_prototype, 1) = prototypes(
   closest_prototype, 1) + distX;
            prototypes(closest_prototype, 2) = prototypes(
   closest_prototype, 2) + distY;
        end

    end
    %ending condition
    E(t) = (P - classified_correctly) / P;
    if t > average_length && E(t-average_length) < E(t) %running
   average of last 5 inreases
        E = E(1,1:t);
        break;
    end
end
%data for plots
c = zeros(P+K*2,3);
sz = zeros(P+K*2,1);
plot_data = zeros(P+K*2,3);
```

```
for i = 1:P/2
    c(i,:) = [0.8,0.5,0.3];
    c(i+P/2,:) = [0.3,0.8,0.5];
end
for p = 1:P
    plot_data(p,:) = lvqdata(p,:);
    sz(p,1) = 30;
end
for k = 1:2:(2*K)
    plot_data(P+k,:) = prototypes(k,:);
    plot_data(P+k+1,:) = prototypes(k+1,:);
    c(P+k,:) = [0.4,0.3,0.1];
    c(P+k+1,:) = [0.1,0.4,0.3];
    sz(P+k,1) = 100;
    sz(P+k+1,1) = 100;
end
%plots
scatter(plot_data(:,1),plot_data(:,2),sz,c,"filled");
xlabel('x feature');
ylabel('y feature');
plot(E);
xlabel('Epoch');
ylabel('Error rate');
```

Listing 2: *This is the file LVQforMean.m*

```
%import
lvqdata = importdata("lvqdata.mat");
%initialization
dimensions = size(lvqdata);
N = dimensions(2) + 1;
P = dimensions(1);
K = 1;
n = 0.002;
t_max = 300;
average_length = 15;
prototypes = zeros(K*2,N);
E = zeros(1,t_max);
%assigning class
for x = 1:P
    if x > P/2
        lvqdata(x,3) = 2;
    else
        lvqdata(x,3) = 1;
    end
end
%mean calculation per class
x1 = 0;
y1 = 0;
x2 = 0;
y2 = 0;
for k = 1:(P/2)
```

```
    x1 = x1 + lvqdata(k,1);
    y1 = y1 + lvqdata(k,2);
    x2 = x2 + lvqdata(P/2+k,1);
    y2 = y2 + lvqdata(P/2+k,2);
end
prototypes = [x1*2/P, y1*2/P, 1; x2*2/P, y2*2/P, 2];
%for each epoch
for t = 1:t_max
    %random permutation of examples
    p = randperm(P);
    classified_correctly = 0;
    %for each example
    for x = 1:P
        %finding closest prototype
        closest_prototype = 1;
        minimal_distance = abs(lvqdata(p(x),1) - prototypes(1,1)) +
   abs(lvqdata(p(x),2) - prototypes(1,2));
        for k = 2:(2*K)
            distance = abs(lvqdata(p(x),1) - prototypes(k,1)) + abs(
   lvqdata(p(x),2) - prototypes(k,2));
            if distance < minimal_distance
                closest_prototype = k;
                minimal_distance = distance;
            end
        end
        if minimal_distance == 0 %no moving in case of a perfect match
            classified_correctly = classified_correctly + 1;
        else
            %distance ratio calculation
            distX = n / minimal_distance * (lvqdata(p(x),1) -
   prototypes(closest_prototype,1));
            distY = n / minimal_distance * (lvqdata(p(x),2) -
   prototypes(closest_prototype,2));
            %direction
            if lvqdata(p(x),3) ~= prototypes(closest_prototype,3)
                distX = -distX;
                distY = -distY;
            else
                classified_correctly = classified_correctly + 1;
            end
            %moving
            prototypes(closest_prototype, 1) = prototypes(
   closest_prototype, 1) + distX;
            prototypes(closest_prototype, 2) = prototypes(
   closest_prototype, 2) + distY;
        end

    end
    %ending condition
    E(t) = (P - classified_correctly) / P;
    if t > average_length && E(t-average_length) < E(t) %running
```

```
   average of last 5 inreases
        E = E(1,1:t);
        break;
    end
end
%data for plots
c = zeros(P+K*2,3);
sz = zeros(P+K*2,1);
plot_data = zeros(P+K*2,3);
for i = 1:P/2
    c(i,:) = [0.8,0.5,0.3];
    c(i+P/2,:) = [0.3,0.8,0.5];
end
for p = 1:P
    plot_data(p,:) = lvqdata(p,:);
    sz(p,1) = 30;
end
for k = 1:2:(2*K)
    plot_data(P+k,:) = prototypes(k,:);
    plot_data(P+k+1,:) = prototypes(k+1,:);
    c(P+k,:) = [0.4,0.3,0.1];
    c(P+k+1,:) = [0.1,0.4,0.3];
    sz(P+k,1) = 100;
    sz(P+k+1,1) = 100;
end
%plots
scatter(plot_data(:,1),plot_data(:,2),sz,c,"filled");
xlabel('x feature');
ylabel('y feature');
plot(E);
xlabel('Epoch');
ylabel('Error rate');
```

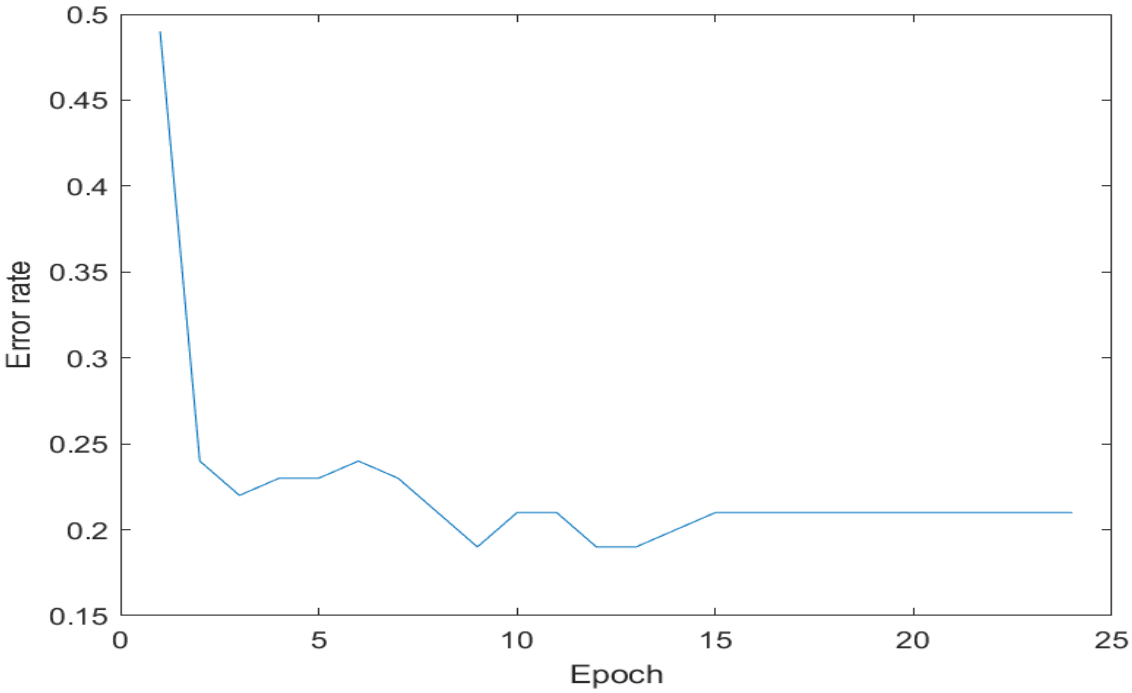

Figure 1: *Learning Curve for 1 prototype per class*

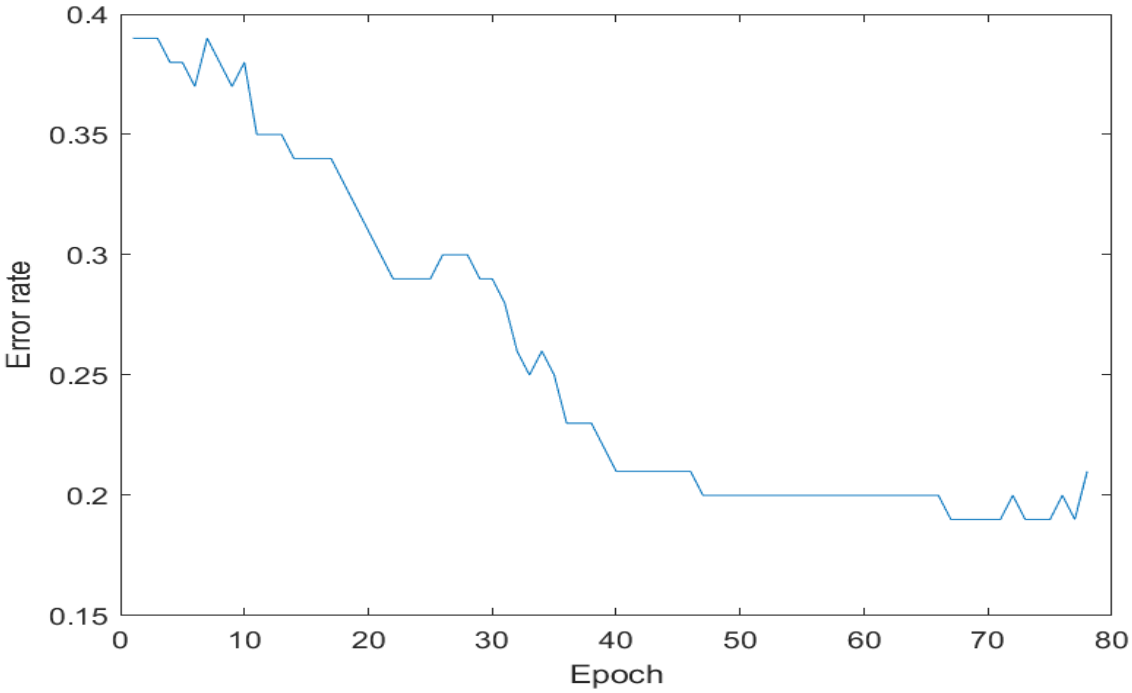

Figure 2: *Learning Curve for 2 prototypes per class*

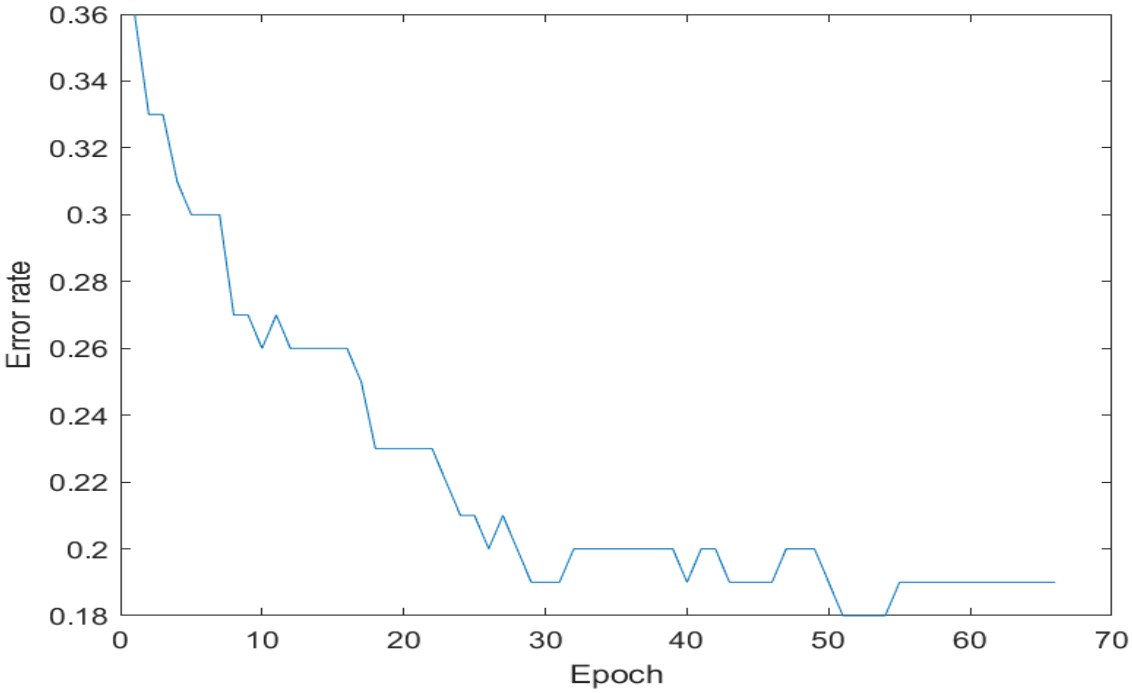

Figure 3: *Learning Curve for 3 prototypes per class*

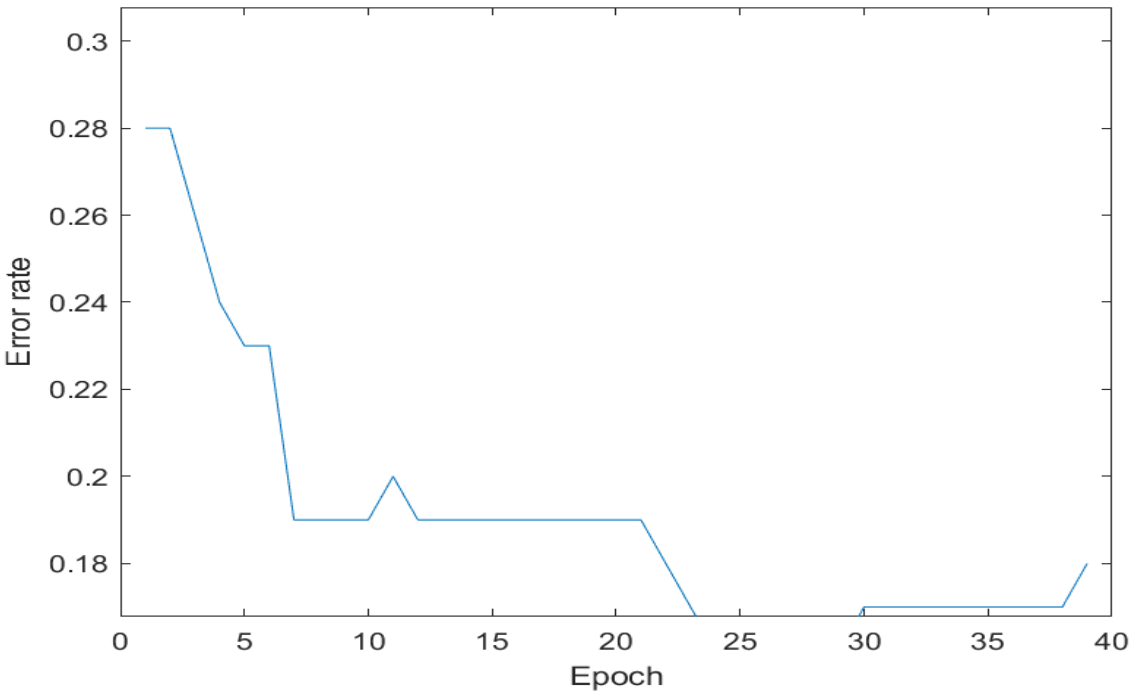

Figure 4: *Learning Curve for 4 prototypes per class*

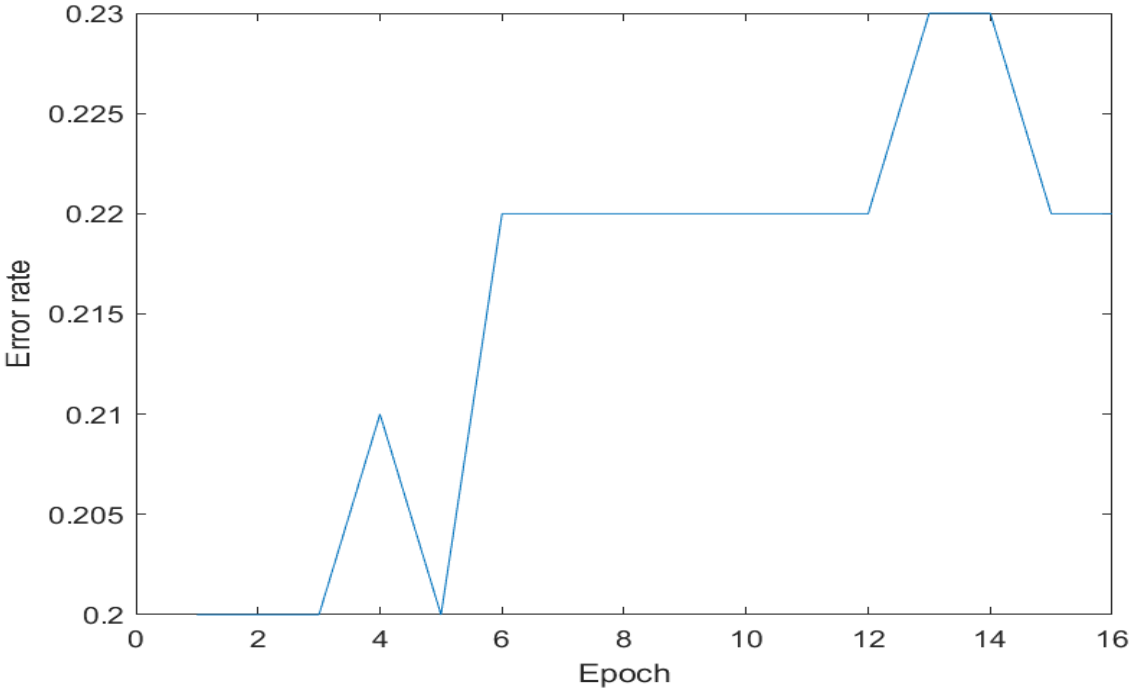

Figure 5: *Learning Curve for the mean prototypes*

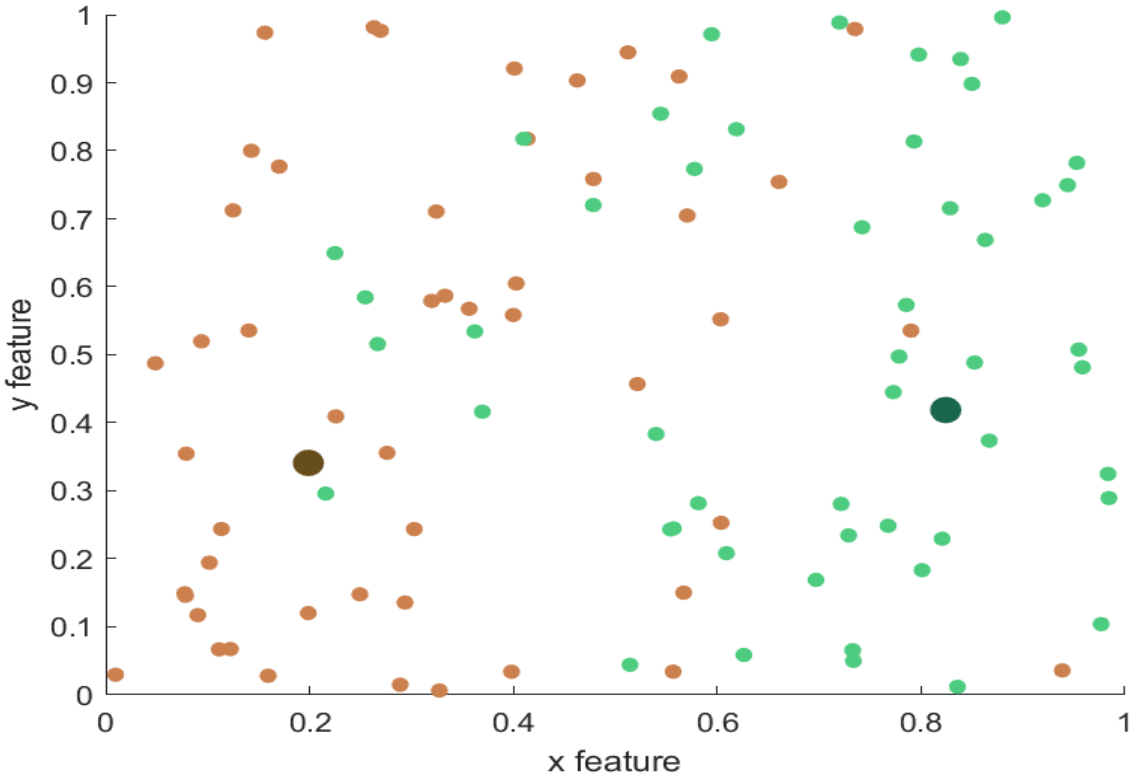

Figure 6: *Scatter plot for 1 prototype per class*

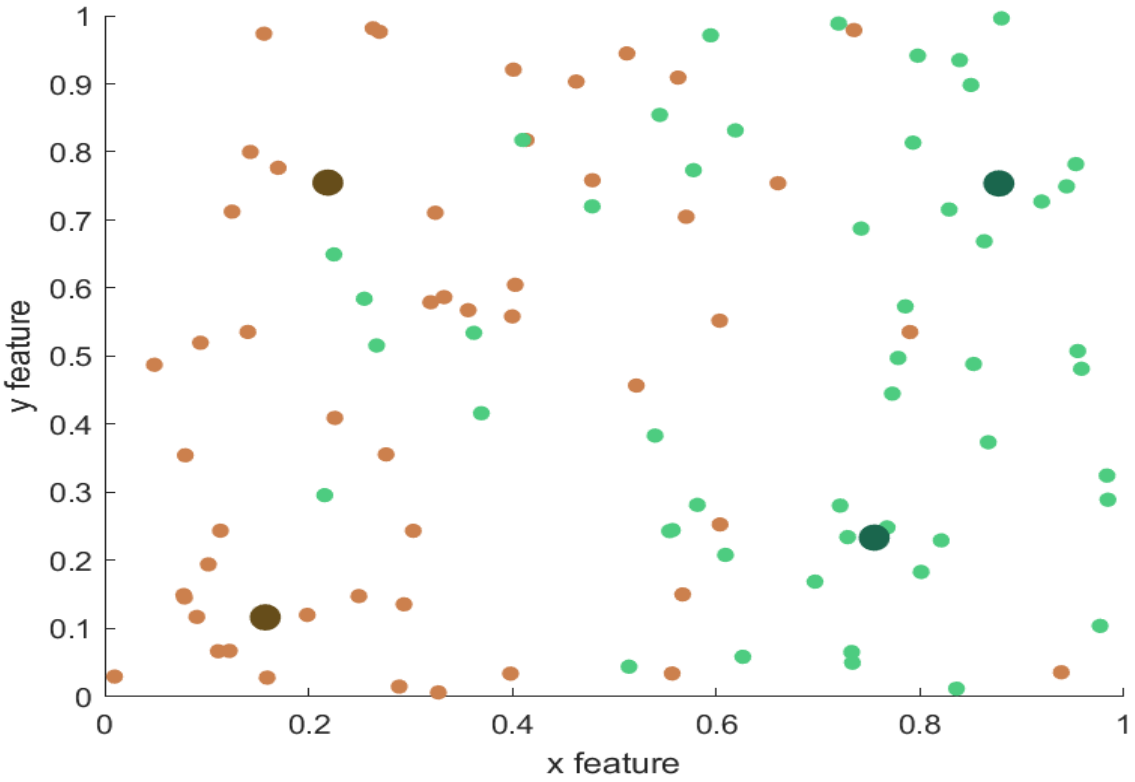

Figure 7: *Scatter plot for 2 prototypes per class*

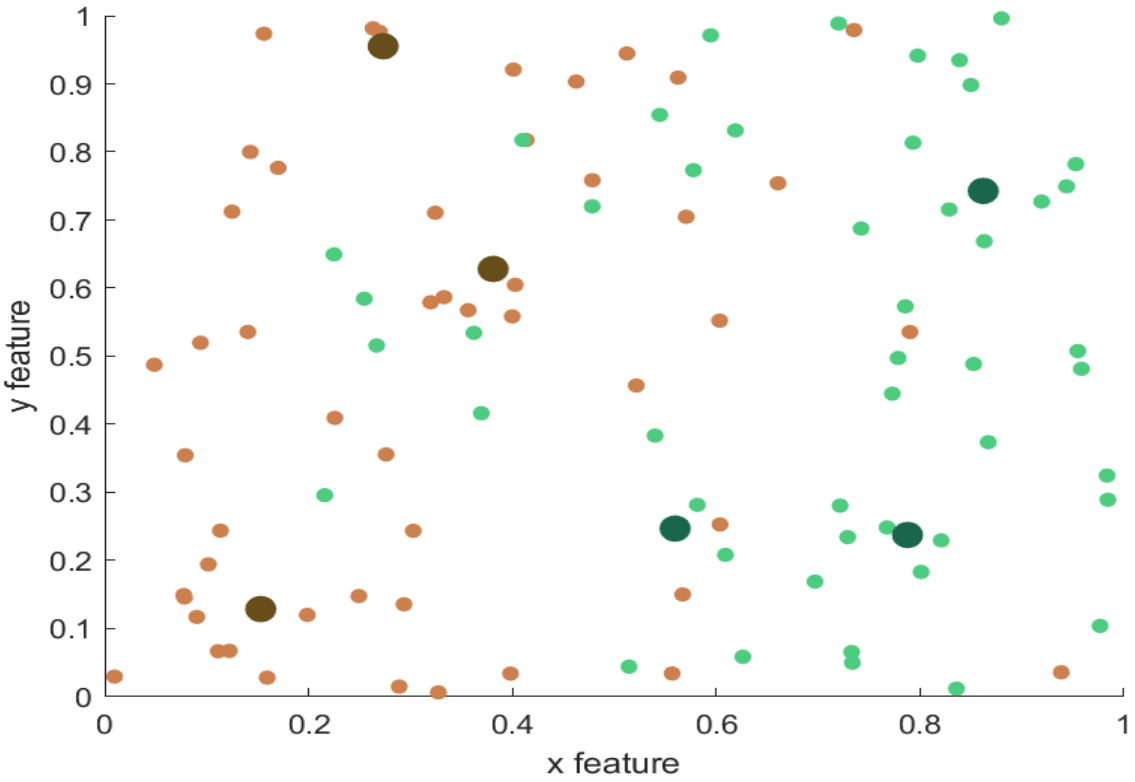

Figure 8: *Scatter plot for 3 prototypes per class*

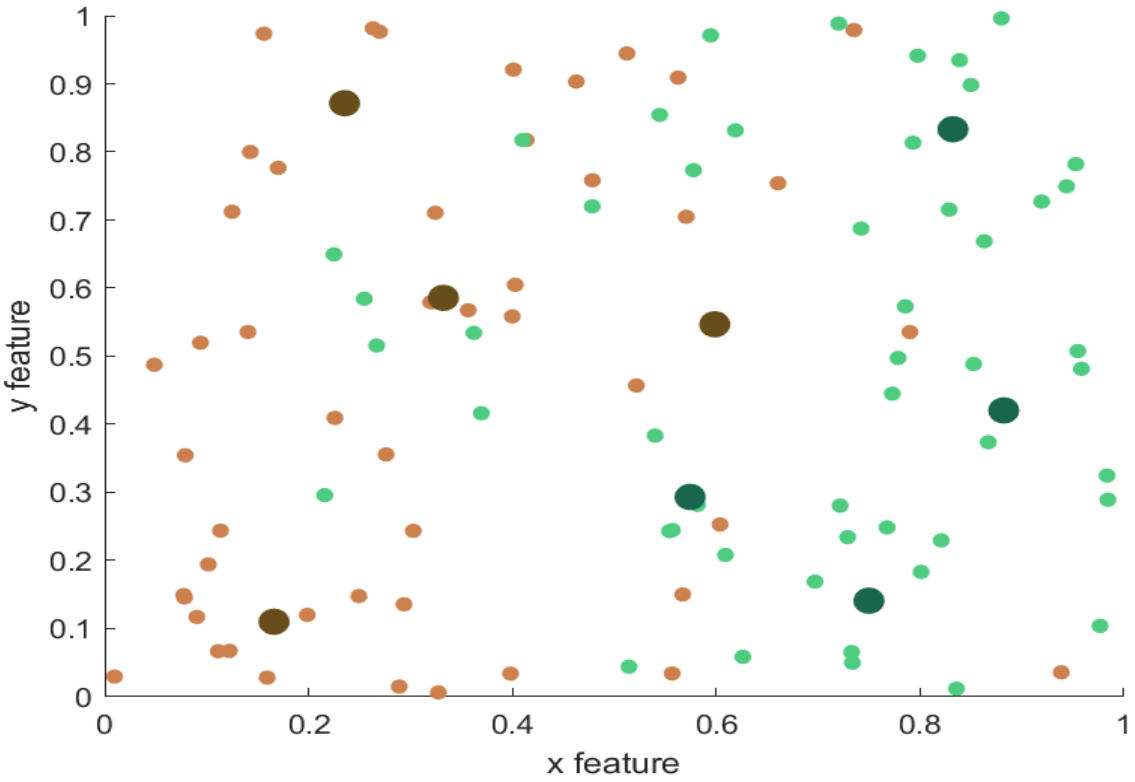

Figure 9: *Scatter plot for 4 prototypes per class*

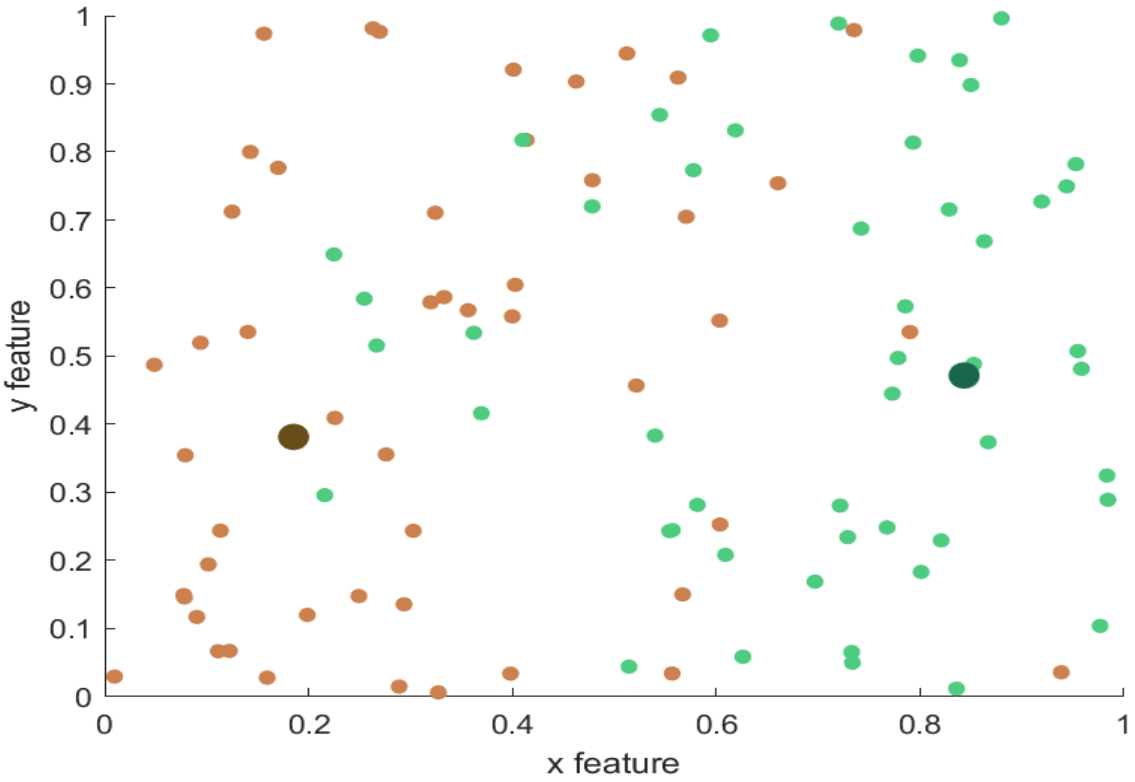

Figure 10: *Scatter plot for the mean prototypes*