Introduction to Intelligent Systems - Lab 6

Tom Apol (s2701650)
Nielis Brouwer (s3850706)
Group: 23

October 19, 2018

Contents

1 Introduction

This report will describe the effect of the number of prototypes per class (K) on the number of misclassified data points with regard to Vector Quantisation of a 2D data set. In particular, we will be using the LVQ1 algorithm. This algorithm first initialises the prototypes on top of randomly selected data points of the same class, after which these prototypes are corrected by data points by 'pushing' and 'pulling' the closest prototype, depending on whether they belong to the same class.

2 Problem Description

The problem as mentioned in the assignment is as follows: Implement the LVQ1 algorithm using simple Euclidean distance as distance measure. Consider the following situations: LVQ1 with 1 - 4 prototypes per class.
Furthermore, we are advised to use the learning rate: $\eta = 0.002$ and a maximum amount of epochs: $100 \leq t\_max \leq 200$. The set used in our experiment is a 2D data set with 100x2 values belonging to a total of two classes, where the first 50 values belong to the first class, and the rest of the data belongs to the second class.

3 Results

For the direct results (plots of "Prototype positions over time" and "Misclassification rate over time"), see Appendix 6.1.
With regards to the code: the code is split up into several parts:

- Loading and initialising of variables and data structures: lines 1 - 36
- Computing the misclassification rate (E) and performing the algorithm: lines 38 - 114
- Plotting both the prototypes over time (and data points), as well as the misclassification rate over time: lines 117 - 159

Note that we did seed the random number generator with `'default'` to be able to reproduce the same results. Furthermore, we add a class label to all data points and prototypes, in order to make distinguishing the two classes easier.

From our plots (fig. 5 - 8) we can observe that all the misclassification rates stabilise after 200 epochs at around 20%. Note that there is not much change over the course of these 200 epochs, as all of our plots start at a point between 20% and 25%. Furthermore, to drive home the lack of change, for this particular data set, each percentage corresponds to one data point, hence the leaps between integer percentages.
Some other observations to note are:

- The stabilisation value of the misclassification rate tends to decrease when increasing K, as is to be expected, as with as many prototypes as data points, the misclassification rate will be 0. However, figure 7 for $K = 3$ does not seem to follow this trend.
- The overall stability (i.e. the lack of the misclassification rate going back up again) vaguely seems to increase when increasing K, though with results for only 4 values of K, together with the relative small amount of change overall and the inherent randomness of the entire procedure, it is very likely that this is heavily influenced by other factors than just K.

4 Discussion

Regarding the question "what is the influence of a higher value for K for the misclassification rate": we cannot conclude that there is a big influence, since the final values of the misclassification rate differ very little. The manner with which the misclassification rate reaches its final value differs, but not significantly. For value K=4 (fig. 8) there is a more stable decline, while with the other K values (figures 5 - 7) the misclassification rate tends to occasionally increase.

Furthermore, the fact that our plot for $K = 3$ (fig. 7) falls out line with regards to its stability and final 'stable' misclassification rate is most likely due to an unfortunate initialisation with our RNG.

Now on to the observations that the misclassification rate does not change much, stabilises around 20% and is not incredibly stable. This probably has to do with the relatively good initialisation of our prototypes (see figures 1 - 4) and, perhaps more importantly, because the data set is incredibly mixed: there are no two clear clusters with clear outliers, which results in the low change in misclassification rate (as stated before), as well as that the misclassification rate remains relatively high, as there will always be a fair number of 'outliers' that are misclassified. Furthermore, with so many 'outliers' mixed together, prototypes are prone to being pushed and pulled back and forth by the data points, as well as prone to be moved about when 'fighting' for data points with other prototypes, as our algorithm uses the 'winner takes all' heuristic. This in turn does not help a lot with stabilising the misclassification rate.

5 Conclusion

As mentioned in the discussion, the differences between the misclassification rate over time with regards to different values of K are rather small. This most likely has to do with the lack of clear clusters and outliers of the respective classes, where the data points of the two classes are mixed together a lot. Because of this, we cannot draw strong conclusions from results of just 4 K values. Therefore the conclusions we will draw are relatively weak and tightly bound to this particular data set.

- With a higher value for K, the misclassification rate over time seems to become more stable, where we define 'stability' as the lack of increasing the value over a longer period of time. However, since the differences over time are very small, all of our plots are quite stable already.
- With a higher value for K, the final 'stable' misclassification rates seem to decrease slightly. The reason why the misclassification rate is relatively high ($\approx 20\%$) is because the data points of the two classes are significantly mixed together, with a lack of clear clusters and outliers per class.

6 Appendix

6.1 Figures

6.1.1 Prototypes over time

Colours are used to separate our data, prototypes and their paths, namely black and red for class 1, and white and green for class 2 respectively. The progression of prototypes is signified by a change in colour from a relatively dark to relatively bright version of their respective colour.
Note that K here denotes the amount of prototypes per class.

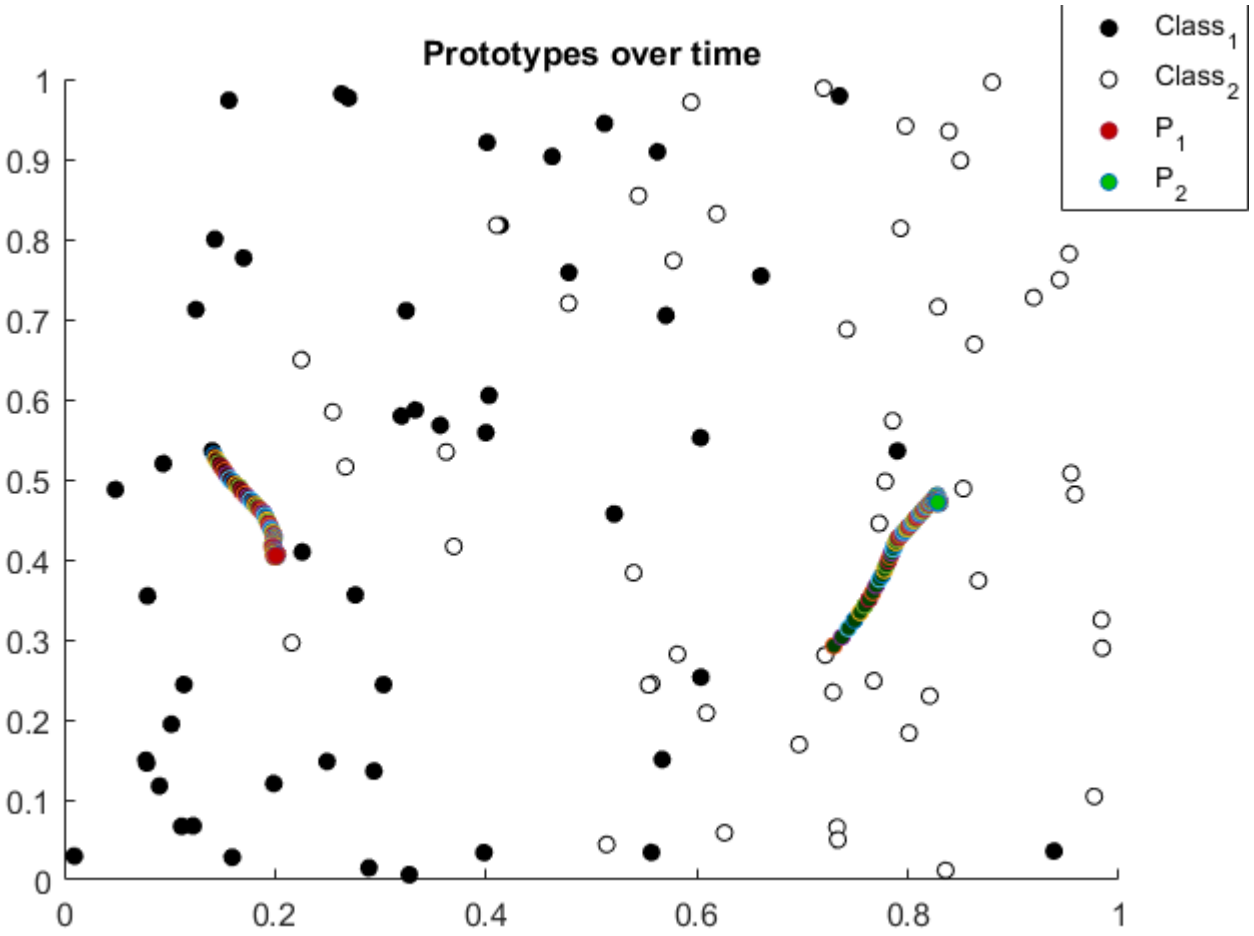

Figure 1: Prototypes over time with K=1

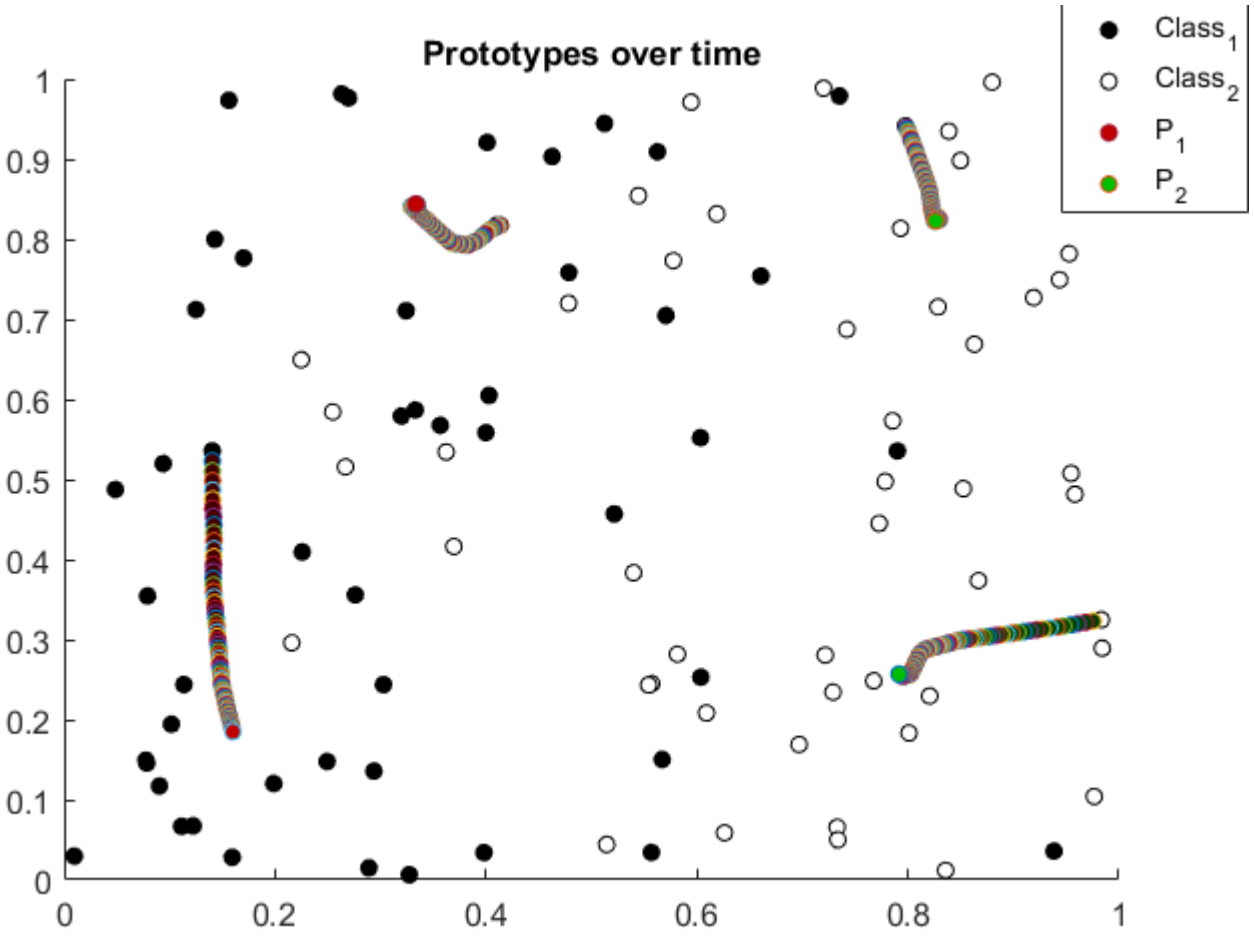

Figure 2: Prototypes over time with K=2

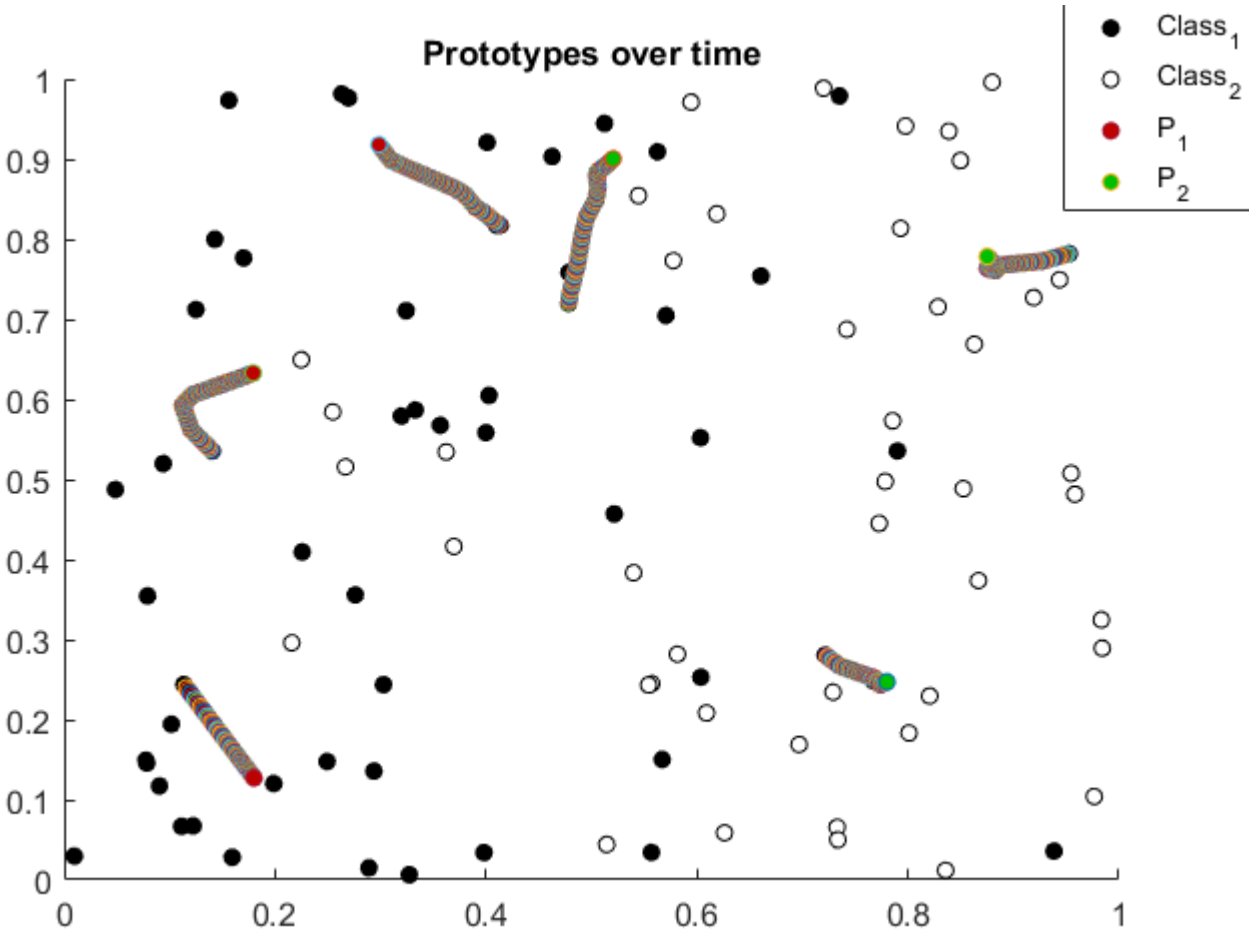

Figure 3: Prototypes over time with K=3

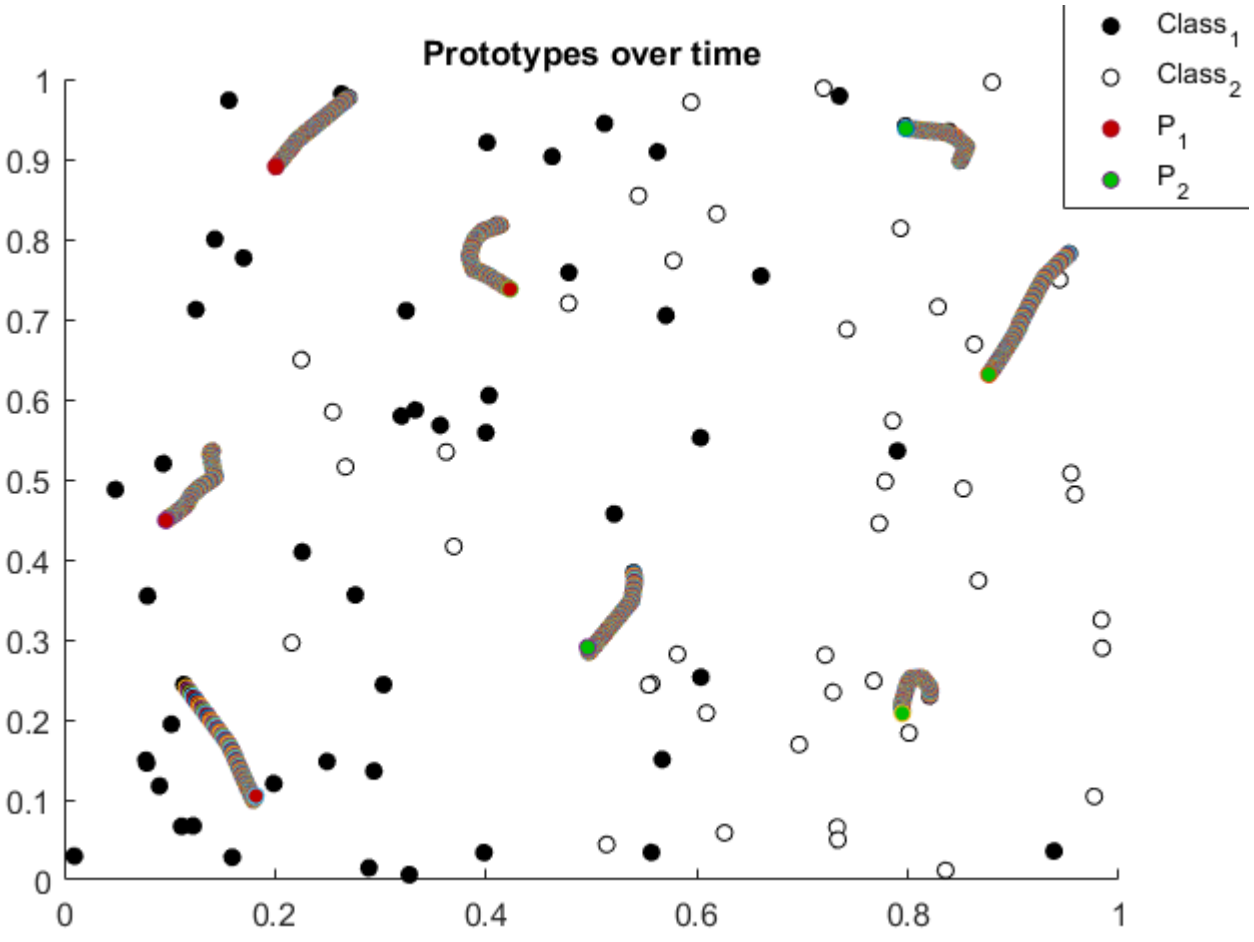

Figure 4: Prototypes over time with K=4

6.1.2 Misclassification rate over time

Note that K here denotes the amount of prototypes per class. Furthermore, the plots have been cropped in terms of the y-axis for readability's sake.

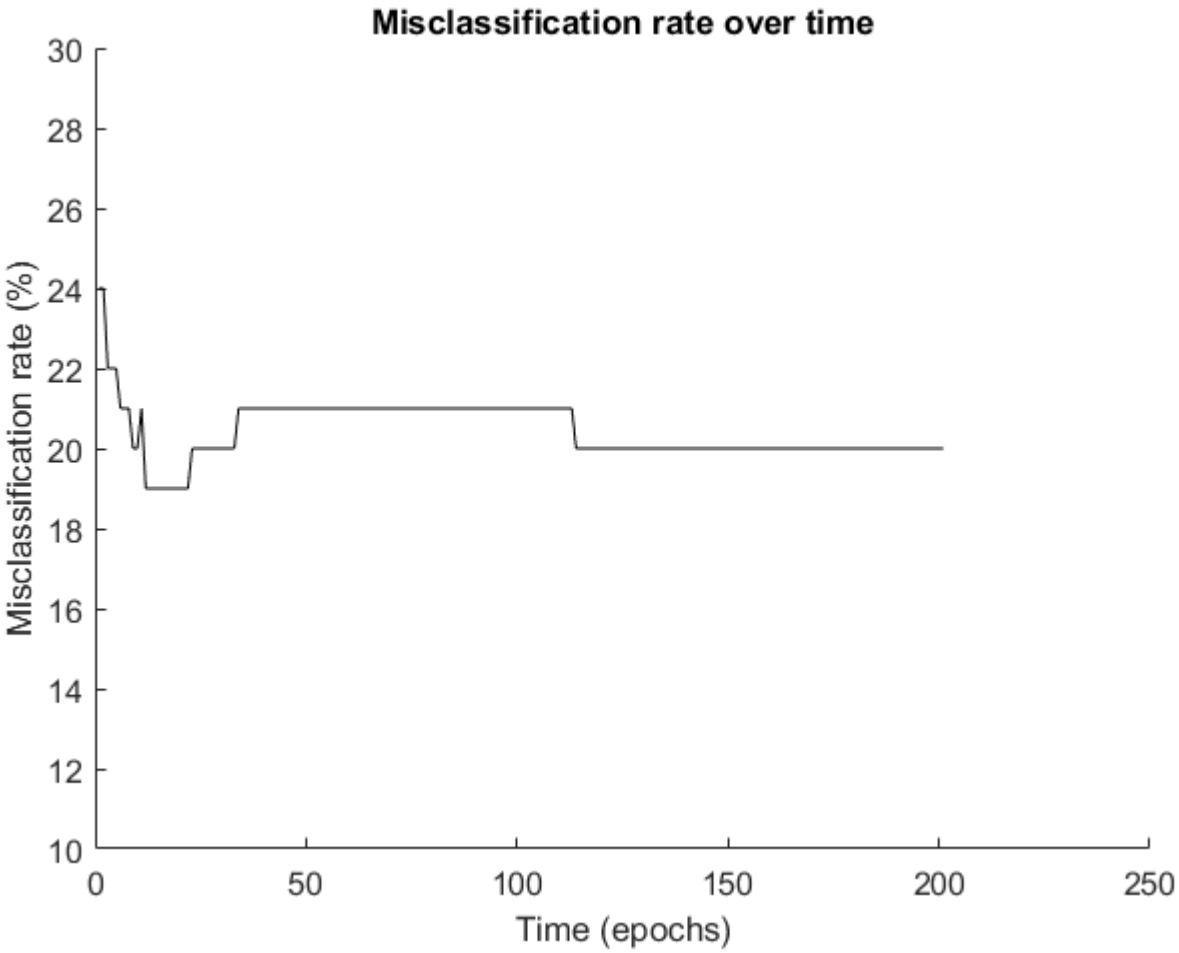

Figure 5: Misclassification rate over time with K=1

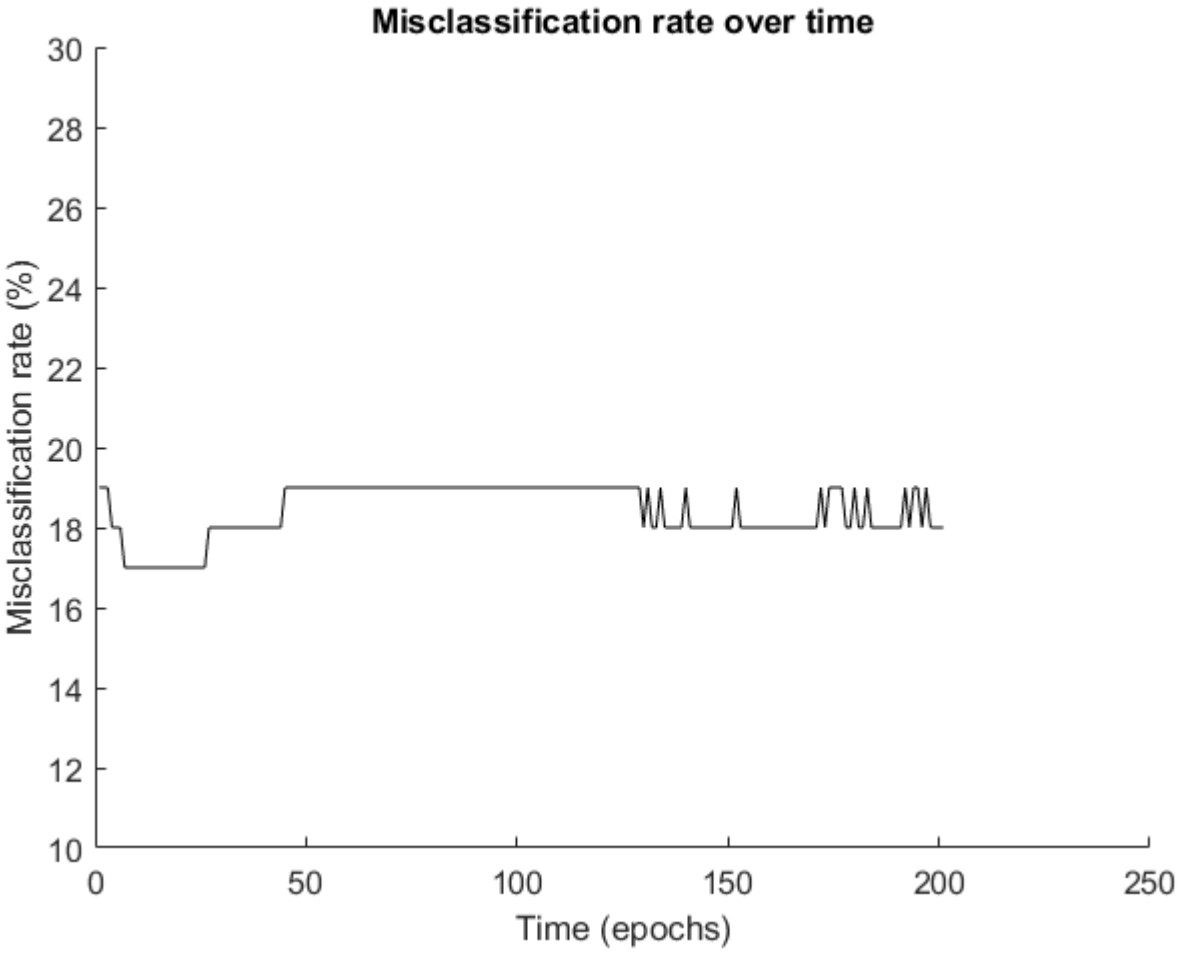

Figure 6: Misclassification rate over time with K=2

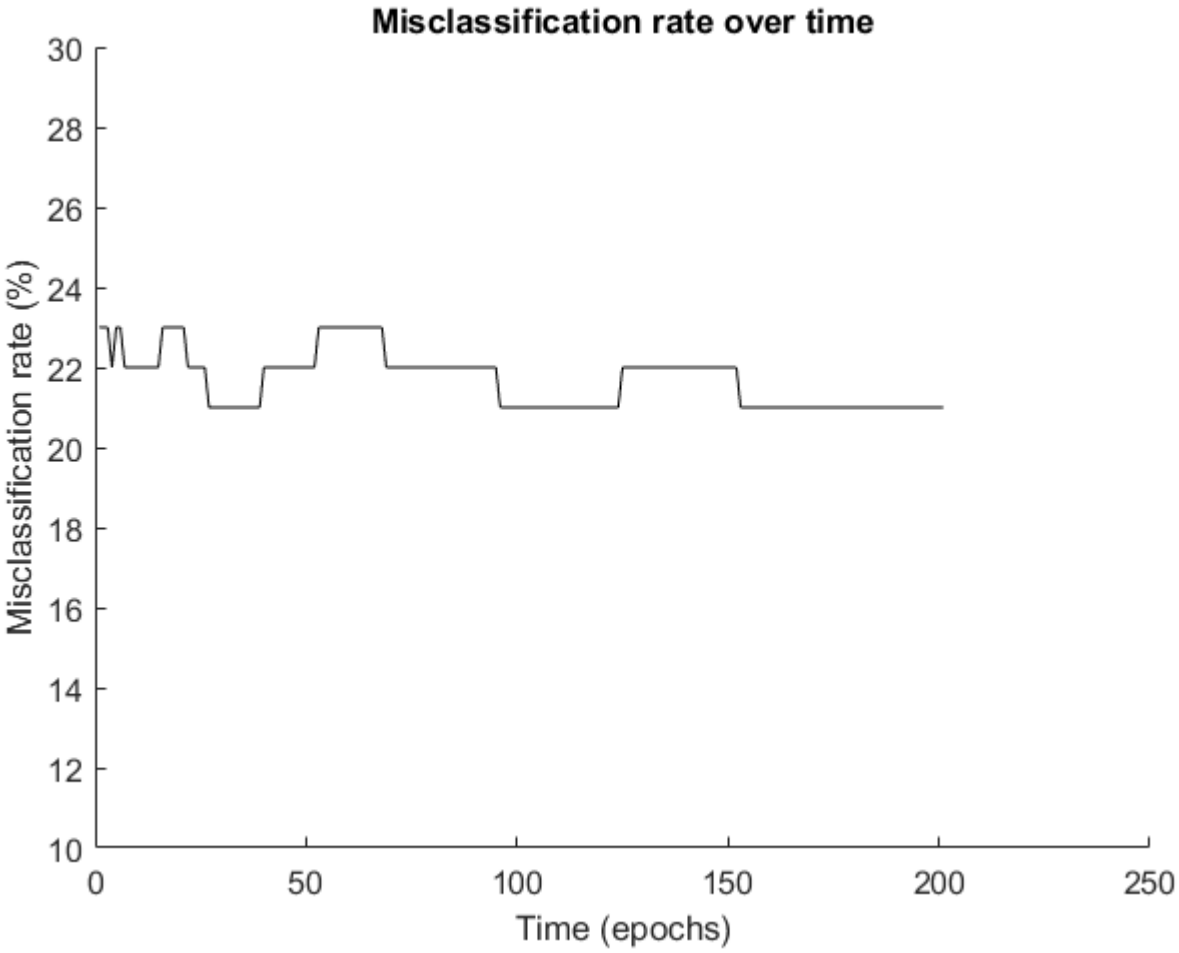

Figure 7: Misclassification rate over time with K=3

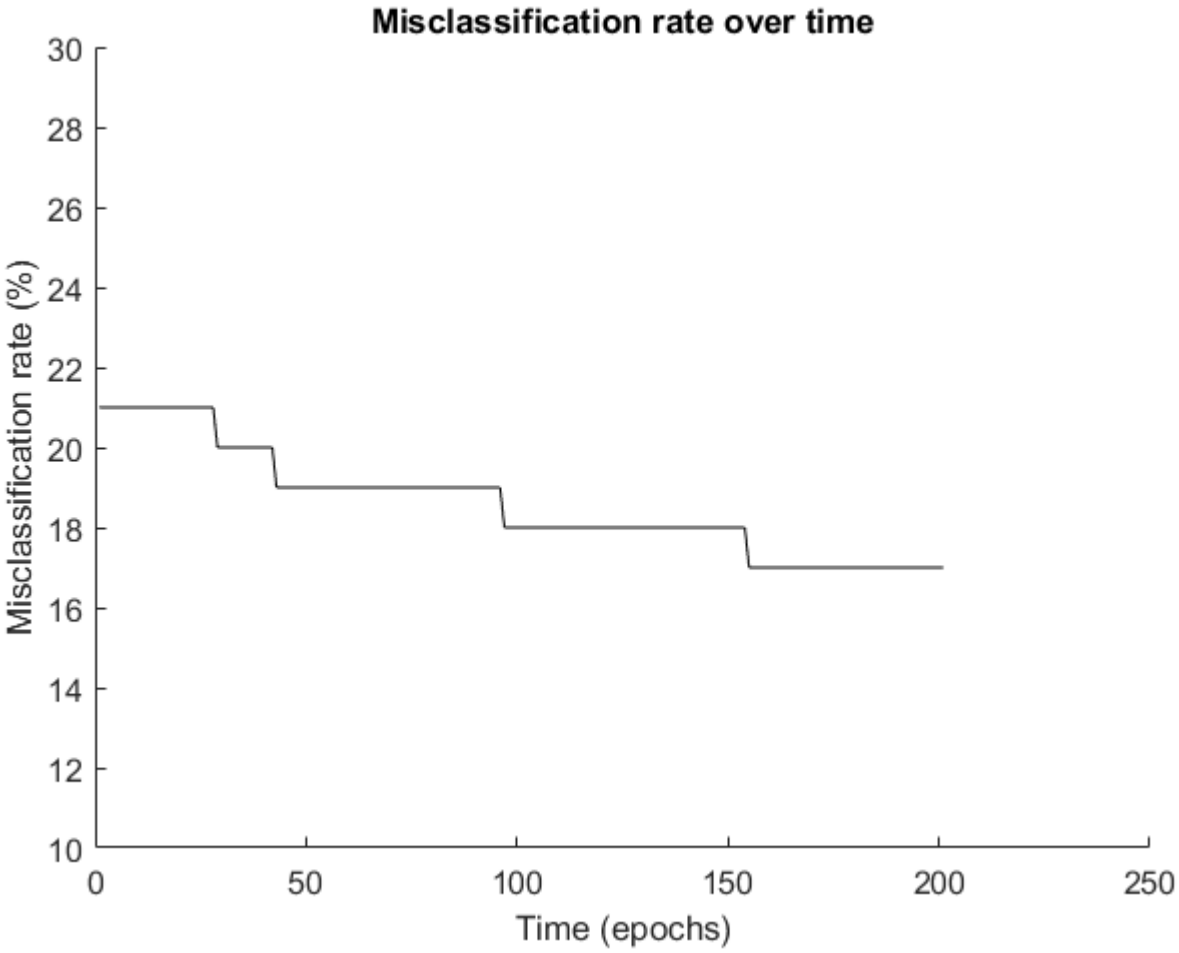

Figure 8: Misclassification rate over time with K=4

6.2 Code

```
% clear previous set data and close all figures
clear all;
close all;

%============================ Initialisation ============================%
% load data
d = load('data_lvq.mat');
d = d.w5_1;
[rows, cols] = size(d);
data = ones(rows, cols+1);
data(:, 1:cols) = d;
for i = 1:50  % add label to data
  data(i, cols+1) = 0;
end

t_max = 200;
learning_rate = 0.002;

% Initialise RNG
rng('default');

% initialise prototypes
NR_OF_CLASSES = 2;
K = 2; % prototypes per class

prototypes = zeros(NR_OF_CLASSES * K, cols+1);

temp_order_class_1 = randperm(50, K);
temp_order_class_2 = randperm(rows-50, K)+50;
for i = 1:K  % get K random data points per class to assign to prototypes
  prototypes(i, :) = data(temp_order_class_1(i), :);
  prototypes(i+K, :) = data(temp_order_class_2(i), :);
end

prototypes_over_time = zeros(t_max, NR_OF_CLASSES * K, cols+1);
E_over_time = zeros(t_max+1,1);

%========================= Actual computation =========================%

% Compute initial E
E_over_time(1) = 0;
for i = 1:rows
  winner = 1;
  winner_distance = intmax;
```

```
45
46    for j = 1:NR_OF_CLASSES*K
47      distance = pdist([prototypes(j, 1:2); data(i, 1:2)]);
48      if distance < winner_distance
49        winner_distance = distance;
50        winner = j;
51      end
52    end
53
54    E_over_time(1) = E_over_time(1) +  ...
55        (prototypes(winner, cols+1) ~= data(i, cols+1));
56  end
57
58
59  % Permutations
60  for t = 1:t_max
61
62    %  Shuffle data set and perform permutations
63    for j = randperm(rows)
64      example = data(j, :);
65
66      % initialise winner
67      winner = 1;
68      winner_distance = intmax; % represents infinity
69
70      % get closest prototype
71      for k = 1:NR_OF_CLASSES * K % take rows
72        prototype = prototypes(k, :);
73        distance = pdist([example(1:cols); prototype(1:cols)]);
74        if distance < winner_distance
75          winner_distance = distance;
76          winner = k;
77        end
78      end
79
80      % update winner
81      sign = 1;
82      if prototypes(winner, cols+1) ~= example(cols+1)
83        sign = -1;
84      end
85
86      prototypes(winner, 1:cols) = prototypes(winner, 1:cols) + ...
87        sign * learning_rate * ...
88        (example(1:cols) - prototypes(winner, 1:cols));
89    end
90
```

```
91    %Compute E
92    E = 0;
93    for i = 1:rows
94      winner = 1;
95      winner_distance = intmax;
96
97      for j = 1:NR_OF_CLASSES*K
98        distance = pdist([prototypes(j, 1:2); data(i, 1:2)]);
99        if distance < winner_distance
100         winner_distance = distance;
101         winner = j;
102       end
103     end
104
105     E = E + (prototypes(winner, cols+1) ~= data(i, cols+1));
106   end
107
108   % store prototypes and misclassification error for this iteration
109   prototypes_over_time(t, :, :) = prototypes;
110   E_over_time(t+1) = E;
111 end
112
113 % normalise and convert E_over_time to percentage
114 E_over_time = E_over_time./rows.*100;
115
116
117 %============================== Plots ================================%
118 figure; hold on;
119
120 plot(data(1:50, 1), data(1:50, 2), 'ko', 'MarkerFaceColor', [0 0 0], ...
121   'DisplayName', 'Class_1');
122 plot(data(51:rows, 1), data(51:rows, 2), 'ko', ...
123   'MarkerFaceColor', [1 1 1], 'DisplayName', 'Class_2');
124 i = 0;
125 for t = 1:t_max
126   for c = 0:NR_OF_CLASSES-1
127     for k = 0:K-1
128       % Make sure only the final prototypes are used in the legend
129       if t == t_max && k == K-1
130         handle_visibility = 'on';
131       else
132         handle_visibility = 'off';
133       end
134
135       row_nr = c*K + k + 1;
136       plot(prototypes_over_time(t, row_nr, 1), ...
```

```
137           prototypes_over_time(t, row_nr, 2), 'o', 'MarkerFaceColor', ...
138           %{
139           Dynamically set gradient colour, depending on t and prototype class
140           %}
141           [(prototypes_over_time(t, row_nr,3) == 0) * (t/t_max/2 + .25), ...
142            (prototypes_over_time(t, row_nr,3) ~= 0) * (t/t_max/2 + .25), ...
143           0], 'HandleVisibility', handle_visibility, ...
144           'DisplayName', strcat('P_', num2str(c+1))');
145       end
146     end
147   end
148   title('Prototypes over time');
149   lgd = legend('show');
150   hold off;
151
152   figure; hold on;
153   plot(1:t_max+1, E_over_time, 'k-');
154   plot(1, 10, 'w');  % Scaling for readibility purposes.
155   plot(1, 30, 'w');
156   xlabel('Time (epochs)');
157   ylabel('Misclassification rate (%)');
158   title('Misclassification rate over time');
159   hold off;
```