Pattern Recognition Practical 4

Group 24: Maikel Withagen (s1867733) Steven Bosch (s1861948)

October 8, 2015

Assignment 1 Supervised learning: LVQ1

Using the code given in the appendix(1) we created the scatterplot in figure 1.

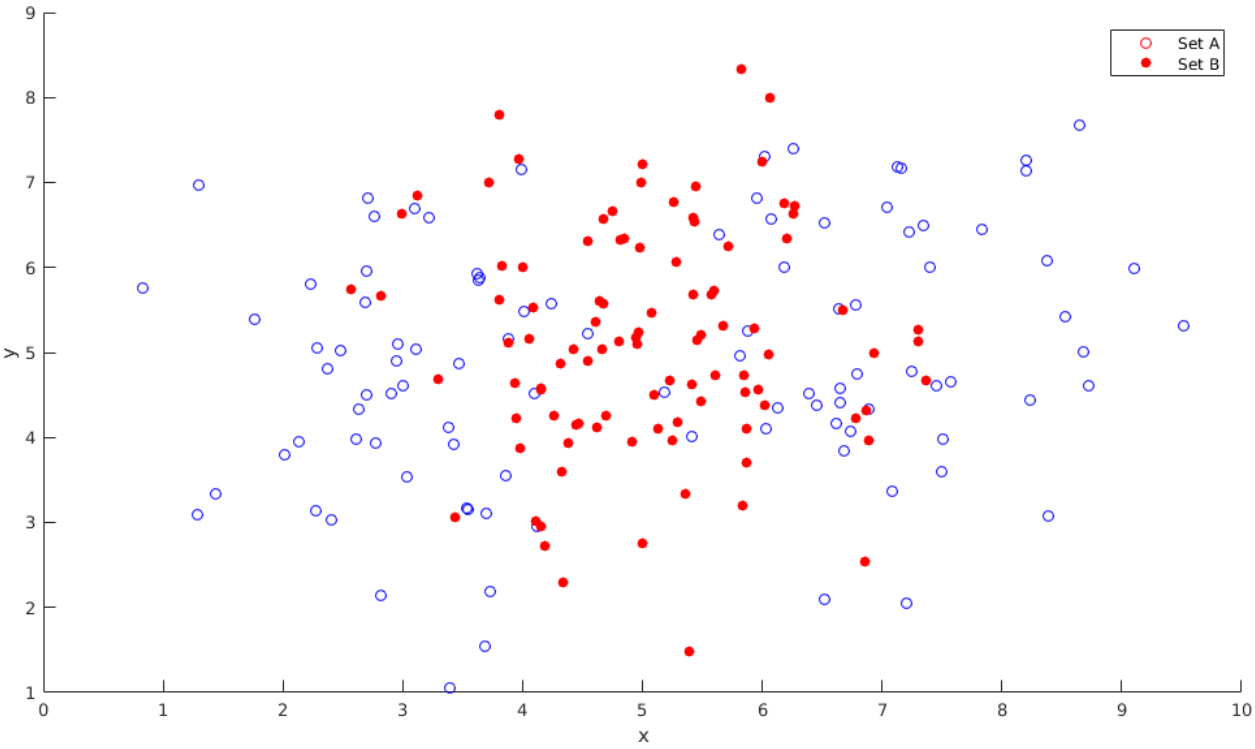

Figure 1: Scatterplot for the two classes.

The plot shows that there are at least three prototypes needed to approach a fairly well classification of these data. Two for set A, which should probably be located around $(3, 4.5)$ and $(7.5, 5.5)$, and one for set B somewhere around $(5, 5)$.

1

The code in the appendix(2) shows our implementation of the LVQ1 algorithm. We acquired the following results for the different settings.

a 1 Prototype for class A and 1 prototype for class B

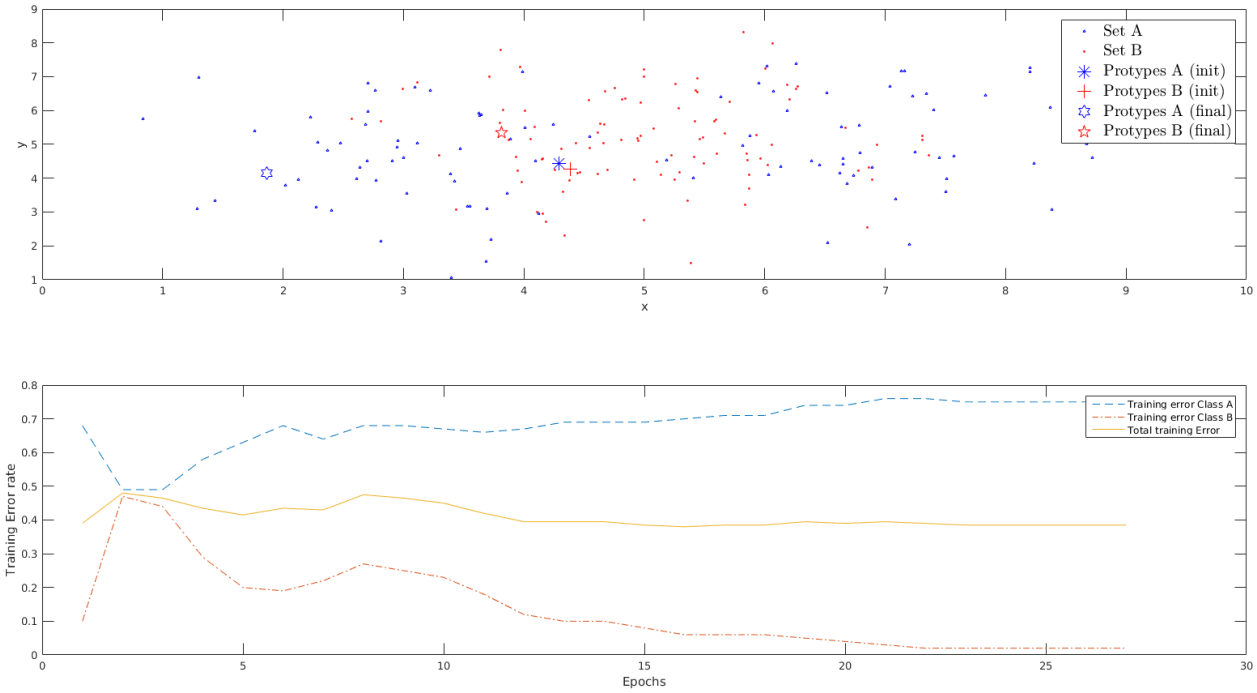

Figure 2: Classification for 1 prototype for class A and 1 prototype for class B. Plot two gives the corresponding error rates (Class 1 is A and class 2 is B).

As is expected with only one prototype per class, figure 2 shows that the prototype for class B is formed quite well, allowing it to correctly classify at least the data points that belong to class B (an error stabilising at around 0.03). However, since class A is distributed in two groups, with B in between them, the prototype for class A is formed at of one of the two outer clusters, which means it can only correctly classify a part of that cluster correctly. The other cluster will be incorrectly classified as belonging to class B, which follows from its error rate which stabilises at around 0.77, meaning that on average 77 out of 100 data points are incorrectly classified, which is quite high. Ultimately this results in a total error rate that stabilises at around 0.4.

b 1 Prototype for class A and 2 prototype for class B

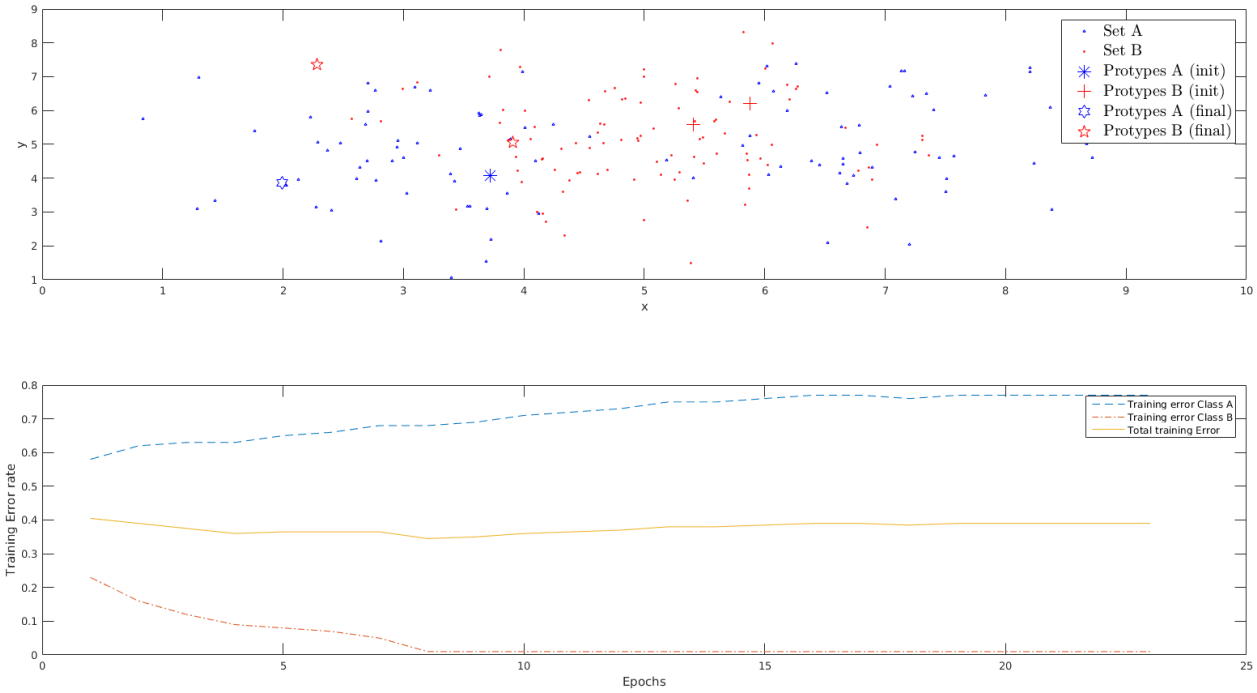

Figure 3: Classification for 1 prototype for class A and 2 prototypes for class B. Plot two gives the corresponding error rates (Class 1 is A and class 2 is B).

In this case figure 3 shows the same thing happening with the prototype for class A: it ends up in one of the two separated A-clusters. Its error is therefore also similar to that in the previous case. On the other hand we see that the two prototypes for class B, even though the data points belonging to class B are clustered in one big cluster, we see that a second prototype slightly improves the result, giving an error that stabilises near 0. This might be the case because it 'catches' some of the last data points that were mixed in with the left-side class A cluster, meaning that it also causes some of the class-B data points in that cluster to be misclassified. This explains the fact that the error of class A has become slightly higher, resulting in a total error that is approximately the same as in the previous case.

c 2 Prototype for class A and 1 prototype for class B

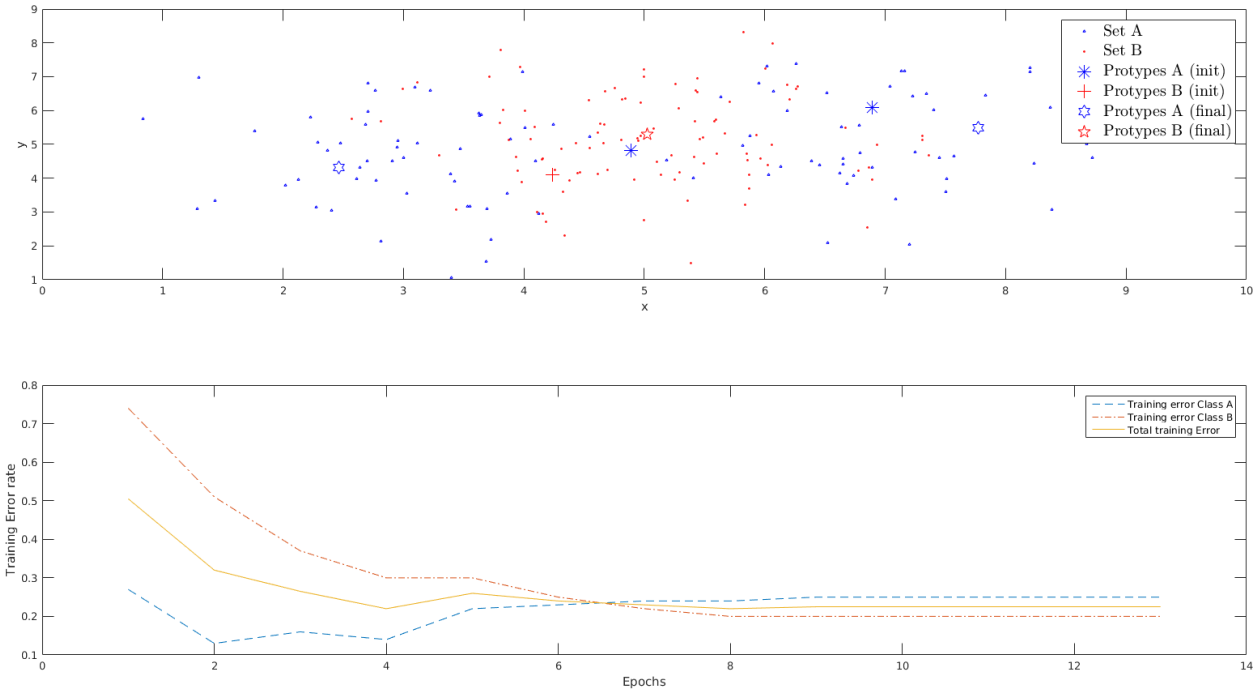

Figure 4: Classification for 2 prototypes for class A and 1 prototype for class B. Plot two gives the corresponding error rates (Class 1 is A and class 2 is B).

In this case figure 4 shows a much lower error rate than in the previous two cases. This is due to the fact that there are now two prototypes for class A, which can classify the seperated data points much better, since they are grouped into two clusters (one prototype for the left cluster, one for the right). Ultimately this results in a total error that stabilises around 0.22, which is significantly lower than in the previous two cases. However, there are also cases in which this amount of prototypes does not result in this error rate. This is the case when the initialization is such that both prototypes for class A begin somewhere near the same cluster of class A, meaning that they will never get to the other cluster, because they get 'pushed away' by the data points belonging to class B. In such a case the result yields about the same error rate as the case discussed in the previous paragraph.

d 2 Prototype for class A and 2 prototype for class B

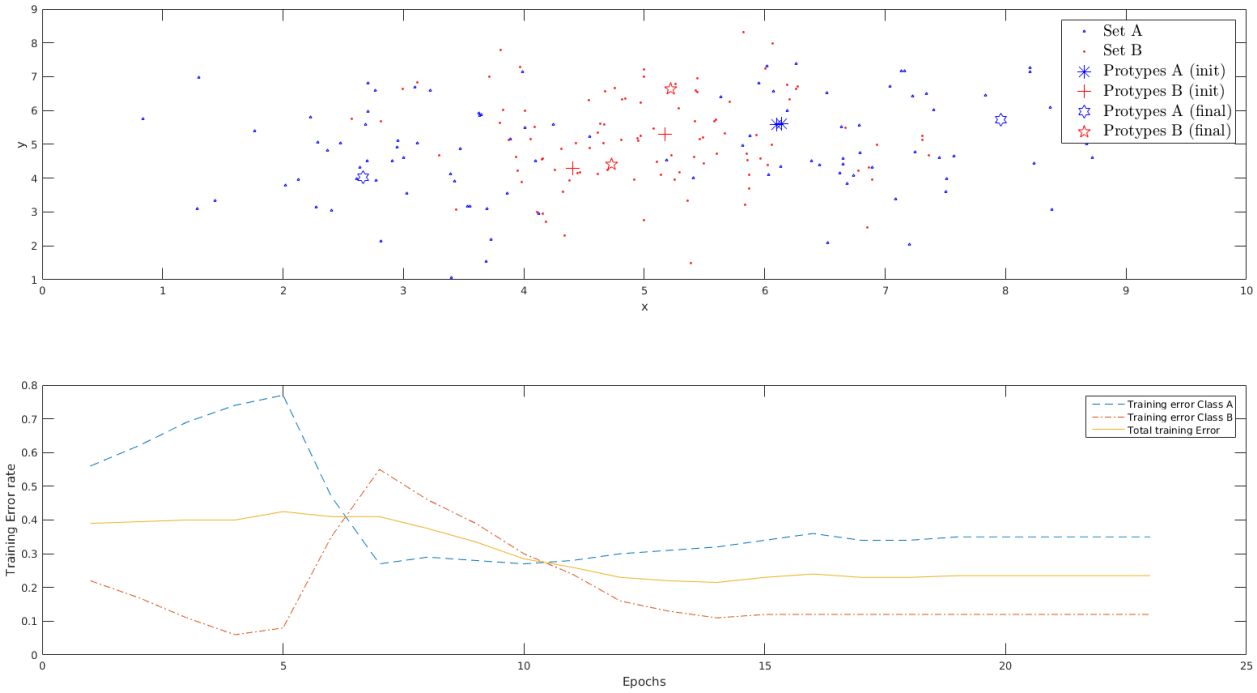

Figure 5: Classification for 2 prototypes for class A and 2 prototypes for class B. Plot two gives the corresponding error rates (Class 1 is A and class 2 is B).

Finally in the case of two prototypes for each class, figure 5 shows slightly worse results than for the previous case, with a total error stabilising at approximately 0.23. This has probably to do with the two prototypes for class B now being divided over the one cluster of data points, which causes them to be slightly nearer to the data points that belong to class A. This causes some of those data points to be misclassified, resulting in a slightly higher error.

Overall we can state that the case for two prototypes for class A and one for class B yields the best results for this particular data set. This is of course also the intuitve explanation, since any human can immediately see that the clusters are distributed in such a way that this distribution of prototypes is needed.

e Learning rate

To investigate the influence of the learning rate, we've run a simulation with a higher, and a lower learning rate, both with the same parameters as Assignment 1c.

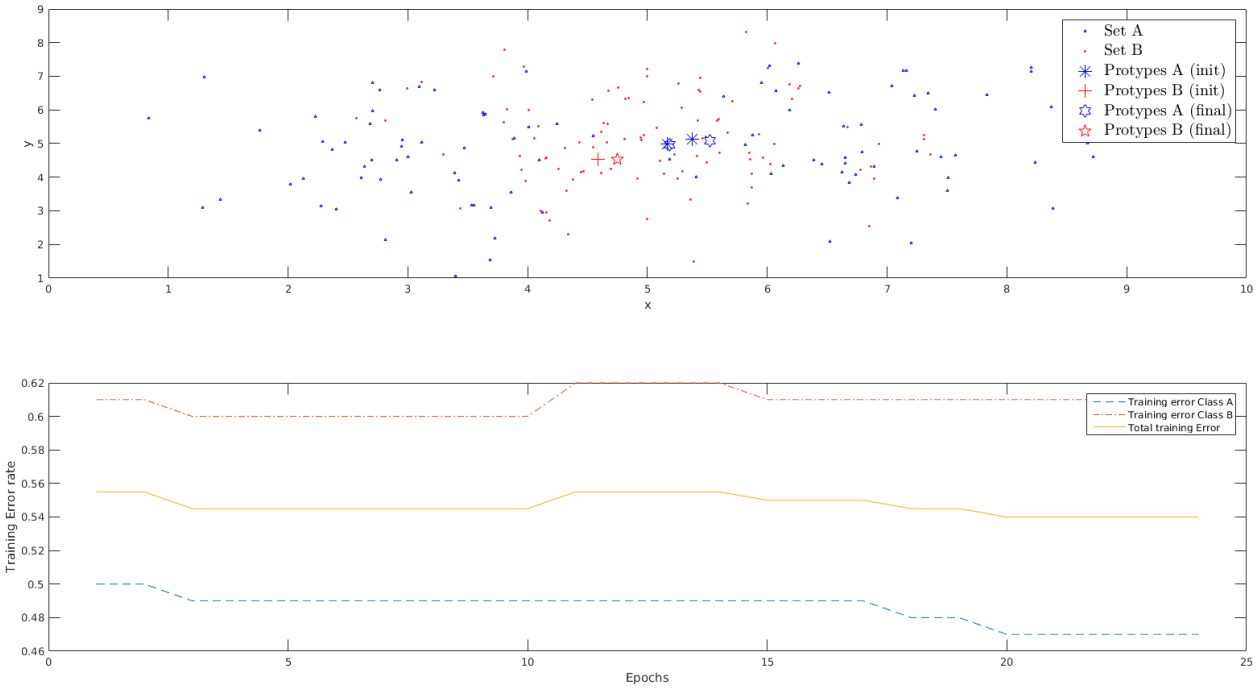

Figure 6: Classification as Assignment 1c, with $\eta = 0.0001$

Figure 6 shows a simulation with $\eta = 0.0001$. The prototype positions and the error rate both suggest that the algorithm was unable to classify with the same performance as simulations with a higher error rate. The error value do have stabilized, so it seems that lowering the learning rate "decreases" the pull that the data points have on the prototypes, resulting in non-optimal results with most likely local-optimal positions for the prototypes.

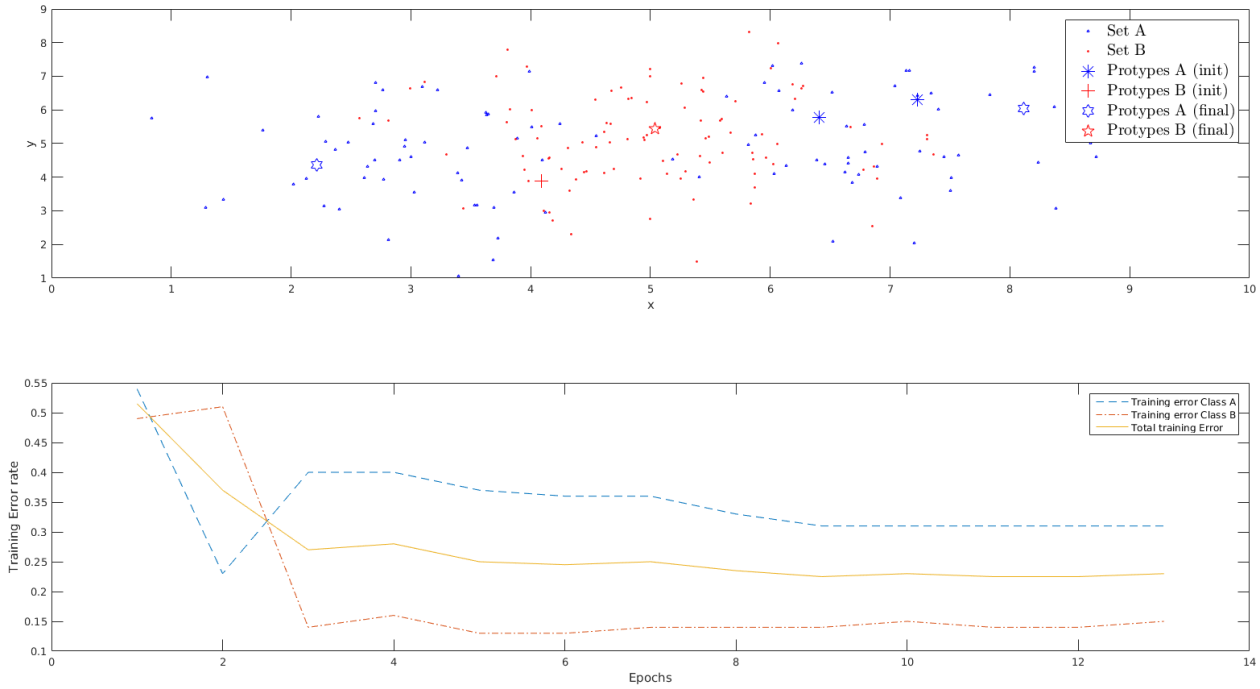

Figure 7: Classification as Assignment 1c, with $\eta = 0.03$

Figure 7 shows a simulation with $\eta = 0.03$, only slightly higher then the default value. This higher learning rate causing to points to "pull" harder on the prototypes. Even for this slight increase in learning rate, we had to decrease our stopping-threshold by a ten-fold, since the prototype positions did not stabilize.

In conclusion, we can say that the learning rate in LVQ1 corresponds to the strength of the "pull" that data points have on the prototype positions. If this pull is too strong, the prototype positions do not stabilize and will hover around their optimal positions. If this pull is too weak, the data points are unable to attract the prototypes to their optimal positions and the prototypes might end up in local, but not the optimal, optima positions.

Assignment 2 Cross validation

The code in the appendix(3) gives our implementation for this assignment. The computed mean of the 10 values of the classification error is 0.300. This is slightly higher than the error rates that were yielded in assignment 1.2(c), which ended up at a total error of around 0.22. It is worth mentioning that certain subsets yielded a high error rate in multiple experiments. This might suggest that the points in these subsets are important for correct training of the prototypes.

Assignment 3 Relevance LVQ

The code in the appendix(4) has produced the following image:

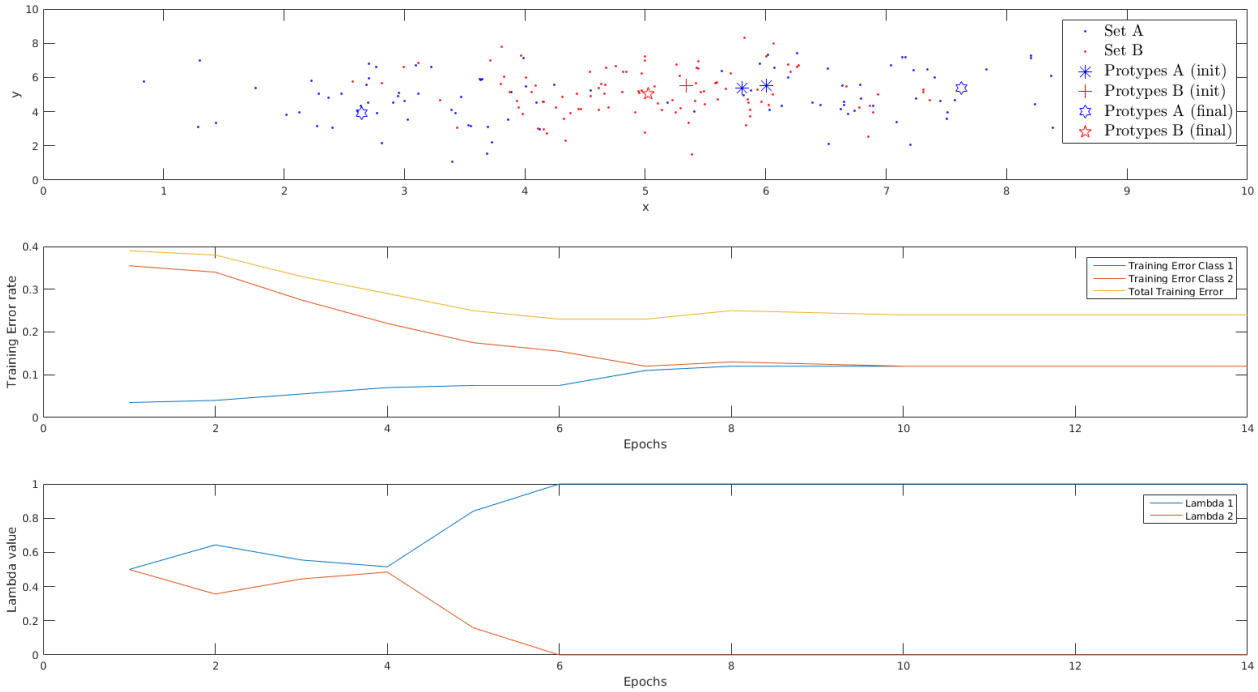

Figure 8

Looking at the final prototypes in Figure 8, we can see that their final positions are similar to ones we saw in earlier assignments. The final error rate also lies around the same value as earlier. This suggests that the prototypes are able to classify with the same performance as in earlier assignments. However, an interesting difference is the learning speed of the algorithm. By using a Relevance LVQ, the error rates have stabilized after only ≈ 15 epochs, which is less then earlier results. The learning curves for the relevances show a mirrorlike behavior due to the mandatory normalization step. Interesting to see is that after some initial movement, their values want to fan out (as was observed in experiments where the normalization step was skipped). This fanning out in combination with the normalization step results in two bottom/ceiling values for the learning values (i.e. 1.0 and 0.0), effectively further adjusting the prototypes only on a single dimension. This can be explained by the fact that the "separation" lines for the two data set are two vertical lines, so once the prototypes have acquired the correct vertical value (i.e. the y-value, learned with lambda 2), only corrections in the x-values (i.e. lambda 1) are neccesary.

The improvement of this algorithm can also be observed in the K-fold estimation test error, which is 0.0250, as calculated with the code in the appendix(5. This value is lower than with the regular LVQ1 algorithm, suggesting that Relevance LVQ has learned to classify even better than LVQ1.

Appendix

1 Code 1.1

../Code/Ass1_1.m

```
hold off;
hold on;
scatter(matA(:,1),matA(:,2), 'blue');
scatter(matB(:,1),matB(:,2), 'red', 'filled');
xlabel('x'); ylabel('y'); legend('Set A', 'Set B');
```

2 Code 1.2

../Code/Ass1_2.m

```
load('data_lvq_A') % matA
load('data_lvq_B') % matB

close all
subplot(2,1,1)
plot(matA(:,1),matA(:,2), 'b^', 'markersize', 2);
hold on;
plot(matB(:,1),matB(:,2), 'rp', 'markersize', 2);
xlabel('x'); ylabel('y');

data = [matA ; matB];
data_labels = (floor( (0:length(data)-1) * 2 / length(data))).';
data = [data data_labels];

% The prototypes
w_A = 2;
w_B = 1;
w = zeros(w_A + w_B, ndims(data)+1);

eta = 0.03;
nrEpochs = 500000;

E_1 = zeros(1,nrEpochs);
E_2 = zeros(1,nrEpochs);

% Randomly initialize the prototypes between the minimum and maximum values
% last value being their class
for i = 1 : size(w,1)
    if i <= w_A
        w(i,:) = [mean(matA) + rand()*2*std(matA)-std(matA) 0];
    else
        w(i,:) = [mean(matB) + rand()*2*std(matB)-std(matB) 1];
    end
end

plot(w(1:w_A,1), w(1:w_A,2), 'b*', 'markersize', 12);
plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'r+', 'markersize', 12);

for epoch = 1:nrEpochs
    epoch
    % Training
    for point = 1 : size(data,1)
        % Find the row with the nearest prototype
        rowMin = find(pdist2(data(point,1:2), w(:,1:2)) == min(pdist2(data(point,1:2), w
            (:,1:2))),1);
        % If the classes of the data point and the nearest prototype are the same
```

```
46          if w(rowMin,end) == data(point, end)
47              % Move the row closer to the data point
48              w(rowMin,1:2) = w(rowMin,1:2) + eta * (data(point,1:2) - w(rowMin,1:2));
49          else
50              w(rowMin,1:2) = w(rowMin,1:2) - eta * (data(point,1:2) - w(rowMin,1:2));
51          end
52      end
53
54      % Testing
55      for point = 1 : size(data,1)
56          % Find the row with the nearest prototype
57          rowMin = find(pdist2(data(point,1:2), w(:,1:2)) == min(pdist2(data(point,1:2), w
                (:,1:2))),1);
58          if w(rowMin,end) ~= data(point, end)
59              if point <= size(matA,1)
60                  E_1(epoch) = E_1(epoch) + 1;
61              else
62                  E_2(epoch) = E_2(epoch) + 1;
63              end
64          end
65      end
66      E = E_1 + E_2;
67
68      if (epoch > 10 && var(E(:,epoch-4:epoch)) < 0.05)
69          E_1(:,epoch+1:end) = [];
70          E_2(:,epoch+1:end) = [];
71          E(:,epoch+1:end) = [];
72          break
73      end
74  end
75
76  plot(w(1:w_A,1), w(1:w_A,2), 'bh', 'markersize', 12);
77  plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'rP', 'markersize', 12);
78  lgnd = legend('Set A', 'Set B','Protypes A (init)', 'Protypes B (init)','Protypes A (final)'
        , 'Protypes B (final)');
79  set(lgnd, 'interpreter','latex', 'fontsize', 15);
80
81  subplot(2,1,2)
82  plot(E_1/100, '--')
83  hold on;
84  plot(E_2/100, '-.');
85  plot(E/200);
86  legend('Training error Class A', 'Training error Class B', 'Total training Error');
87  xlabel('Epochs')
88  ylabel('Training Error rate')
```

3 Code 2

../Code/Ass2.m

```
1  load('data_lvq_A') % matA
2  load('data_lvq_B') % matB
3
4  K = 10;
5
6  %10 FOLD Cross validation
7  data = [matA ; matB];
8  data_labels = (floor( (0:length(data)-1) * 2 / length(data))).';
9  data = [data data_labels];
10 indices = 1:length(data)/K:length(data)+(length(data)/K);
11 E_K = zeros(1,K);
12
13 for fold = 1:K
```

```
14      train_data = data;
15      train_data(indices(fold):indices(fold+1)-1,:) = [];
16      test_data  = data(indices(fold):indices(fold+1)-1,:);
17
18      % The prototypes
19      w_A = 2;
20      w_B = 1;
21      w = zeros(w_A + w_B, ndims(data)+1);
22
23      eta = 0.01;
24      nrEpochs = 500;
25
26      E_1 = zeros(1,nrEpochs);
27      E_2 = zeros(1,nrEpochs);
28
29      % Randomly initialize the prototypes between the minimum and maximum values
30      % last value being their class
31      for i = 1 : size(w,1)
32          if i <= w_A
33              w(i,:) = [mean(matA) + rand()*2*std(matA)-std(matA)  0];
34          else
35              w(i,:) = [mean(matB) + rand()*2*std(matB)-std(matB)  1];
36          end
37      end
38
39      for epoch = 1:nrEpochs
40          % Training
41          for point = 1 : size(train_data,1)
42              % Find the row with the nearest prototype
43              rowMin = find(pdist2(train_data(point,1:2), w(:,1:2)) == min(pdist2(train_data(
                    point,1:2), w(:,1:2))),1);
44              % If the classes of the train_data point and the nearest prototype are the same
45              if w(rowMin,end) == train_data(point, end)
46                  % Move the row closer to the train_data point
47                  w(rowMin,1:2) = w(rowMin,1:2) + eta * (train_data(point,1:2) - w(rowMin,1:2)
                        );
48              else
49                  w(rowMin,1:2) = w(rowMin,1:2) - eta * (train_data(point,1:2) - w(rowMin,1:2)
                        );
50              end
51          end
52
53          % Testing
54          for point = 1 : size(train_data,1)
55              % Find the row with the nearest prototype
56              rowMin = find(pdist2(train_data(point,1:2), w(:,1:2)) == min(pdist2(train_data(
                    point,1:2), w(:,1:2))),1);
57              if w(rowMin,end) ~= train_data(point, end)
58                  if point <= size(matA,1)
59                      E_1(epoch) = E_1(epoch) + 1;
60                  else
61                      E_2(epoch) = E_2(epoch) + 1;
62                  end
63              end
64          end
65          E = E_1 + E_2;
66
67          if (epoch > 10 && var(E(:,epoch-4:epoch)) < 0.05)
68              fold
69              E_1(:,epoch+1:end) = [];
70              E_2(:,epoch+1:end) = [];
71              E(:,epoch+1:end) = [];
72              break
73          end
74      end
```

```
75
76      % Now test on the test set
77      % Testing
78      for point = 1 : size(test_data,1)
79          % Find the row with the nearest prototype
80          rowMin = find(pdist2(test_data(point,1:2), w(:,1:2)) == min(pdist2(test_data(point
                ,1:2), w(:,1:2))),1);
81          if w(rowMin,end) ~= test_data(point, end)
82              E_K(fold) = E_K(fold) + 1;
83          end
84      end
85  end
86
87  % The mean error rate over the 10 folds
88  mean(E_K/size(test_data,1))
```

4 Code 3a

../Code/Ass3.m

```
1   load('data_lvq_A') % matA
2   load('data_lvq_B') % matB
3
4   close all
5   subplot(3,1,1)
6   plot(matA(:,1),matA(:,2), 'bp', 'markersize', 2);
7   hold on;
8   plot(matB(:,1),matB(:,2), 'rp', 'markersize', 2);
9   xlabel('x'); ylabel('y');
10
11  data = [matA ; matB];
12  data_labels = (floor( (0:length(data)-1) * 2 / length(data))).';
13  data = [data data_labels];
14
15  % The prototypes
16  w_A = 2;
17  w_B = 1;
18  w = zeros(w_A + w_B, ndims(data)+1);
19  lambda = [0.5  0.5];
20
21  eta = 0.01;
22  etaL = 0.01;
23  nrEpochs = 200;
24
25  E_1 = zeros(1,nrEpochs);
26  E_2 = zeros(1,nrEpochs);
27  lambdaHist = zeros(2,nrEpochs);
28
29  % Randomly initialize the prototypes between the minimum and maximum values
30  % last value being their class
31  for i = 1 : size(w,1)
32      if i <= w_A
33          w(i,:) = [mean(matA) + rand()*2*std(matA)-std(matA)  0];
34      else
35          w(i,:) = [mean(matB) + rand()*2*std(matB)-std(matB)  1];
36      end
37  end
38
39  plot(w(1:w_A,1), w(1:w_A,2), 'b*', 'markersize', 12);
40  plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'r+', 'markersize', 12);
41
42  for epoch = 1:nrEpochs
43      % Save the old lambda values
```

```
44      lambdaHist(:,epoch) = lambda.';
45      % Training
46      for point = 1 : size(data,1)
47          % Calculate the distances for each prototype to the point
48          dist = zeros(1,size(w,1));
49          for prot = 1:size(w,1)
50              for dim = 1:size(matA,2)
51                  dist(prot) = dist(prot) + (lambda(dim) * (w(prot, dim) - data(point, dim))
                        ^2);
52              end
53          end
54
55          % Find the row with the nearest prototype
56          rowMin = find(dist == min(dist),1);
57          % If the classes of the data point and the nearest prototype are the same
58
59          if w(rowMin,end) == data(point, end)
60              % Move the row closer to the data point
61              w(rowMin,1:2) = w(rowMin,1:2) + eta * (data(point,1:2) - w(rowMin,1:2));
62              lambda = lambda - etaL * abs((data(point,1:2) - w(rowMin,1:2)));
63          else
64              w(rowMin,1:2) = w(rowMin,1:2) - eta * (data(point,1:2) - w(rowMin,1:2));
65              lambda = lambda + etaL * abs((data(point,1:2) - w(rowMin,1:2)));
66          end
67
68          if lambda(1) < 0
69              lambda(1) = 0;
70          end
71          if lambda(2) < 0
72              lambda(2) = 0;
73          end
74          lambda = lambda / sum(lambda);
75      end
76
77      % Testing
78      for point = 1 : size(data,1)
79          % Find the row with the nearest prototype
80          rowMin = find(pdist2(data(point,1:2), w(:,1:2)) == min(pdist2(data(point,1:2), w
                (:,1:2))),1);
81          if w(rowMin,end) ~= data(point, end)
82              if point <= size(matA,1)
83                  E_1(epoch) = E_1(epoch) + 1;
84              else
85                  E_2(epoch) = E_2(epoch) + 1;
86              end
87          end
88      end
89      E = E_1 + E_2;
90
91      if (epoch > 10 && var(E(:,epoch-4:epoch)) < 0.05)
92          E_1(:,epoch+1:end) = [];
93          E_2(:,epoch+1:end) = [];
94          E(:,epoch+1:end) = [];
95          lambdaHist(:,epoch+1:end) = [];
96          break
97      end
98  end
99
100 plot(w(1:w_A,1), w(1:w_A,2), 'bh', 'markersize', 12);
101 plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'rP', 'markersize', 12);
102 lgnd = legend('Set A', 'Set B','Protypes A (init)', 'Protypes B (init)','Protypes A (final)'
        , 'Protypes B (final)');
103 set(lgnd, 'interpreter','latex', 'fontsize', 15);
104
105 subplot(3,1,2)
```

```
plot(E_1/200)
hold on;
plot(E_2/200);
plot(E/200);
legend('Training Error Class 1', 'Training Error Class 2', 'Total Training Error');
xlabel('Epochs')
ylabel('Training Error rate')

subplot(3,1,3)
plot(lambdaHist(1,:));
hold on
plot(lambdaHist(2,:));
legend('Lambda 1', 'Lambda 2');
xlabel('Epochs')
ylabel('Lambda value')
```

5 Code 3b

../Code/Ass3.m

```
load('data_lvq_A') % matA
load('data_lvq_B') % matB

close all
subplot(3,1,1)
plot(matA(:,1),matA(:,2), 'bp', 'markersize', 2);
hold on;
plot(matB(:,1),matB(:,2), 'rp', 'markersize', 2);
xlabel('x'); ylabel('y');

data = [matA ; matB];
data_labels = (floor( (0:length(data)-1) * 2 / length(data)))';
data = [data data_labels];

% The prototypes
w_A = 2;
w_B = 1;
w = zeros(w_A + w_B, ndims(data)+1);
lambda = [0.5 0.5];

eta = 0.01;
etaL = 0.01;
nrEpochs = 200;

E_1 = zeros(1,nrEpochs);
E_2 = zeros(1,nrEpochs);
lambdaHist = zeros(2,nrEpochs);

% Randomly initialize the prototypes between the minimum and maximum values
% last value being their class
for i = 1 : size(w,1)
    if i <= w_A
        w(i,:) = [mean(matA) + rand()*2*std(matA)-std(matA) 0];
    else
        w(i,:) = [mean(matB) + rand()*2*std(matB)-std(matB) 1];
    end
end

plot(w(1:w_A,1), w(1:w_A,2), 'b*', 'markersize', 12);
plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'r+', 'markersize', 12);

for epoch = 1:nrEpochs
    % Save the old lambda values
```

```
44      lambdaHist(:,epoch) = lambda.';
45      % Training
46      for point = 1 : size(data,1)
47          % Calculate the distances for each prototype to the point
48          dist = zeros(1,size(w,1));
49          for prot = 1:size(w,1)
50              for dim = 1:size(matA,2)
51                  dist(prot) = dist(prot) + (lambda(dim) * (w(prot, dim) - data(point, dim))
                        ^2);
52              end
53          end
54
55          % Find the row with the nearest prototype
56          rowMin = find(dist == min(dist),1);
57          % If the classes of the data point and the nearest prototype are the same
58
59          if w(rowMin,end) == data(point, end)
60              % Move the row closer to the data point
61              w(rowMin,1:2) = w(rowMin,1:2) + eta * (data(point,1:2) - w(rowMin,1:2));
62              lambda = lambda - etaL * abs((data(point,1:2) - w(rowMin,1:2)));
63          else
64              w(rowMin,1:2) = w(rowMin,1:2) - eta * (data(point,1:2) - w(rowMin,1:2));
65              lambda = lambda + etaL * abs((data(point,1:2) - w(rowMin,1:2)));
66          end
67
68          if lambda(1) < 0
69              lambda(1) = 0;
70          end
71          if lambda(2) < 0
72              lambda(2) = 0;
73          end
74          lambda = lambda / sum(lambda);
75      end
76
77      % Testing
78      for point = 1 : size(data,1)
79          % Find the row with the nearest prototype
80          rowMin = find(pdist2(data(point,1:2), w(:,1:2)) == min(pdist2(data(point,1:2), w
                (:,1:2))),1);
81          if w(rowMin,end) ~= data(point, end)
82              if point <= size(matA,1)
83                  E_1(epoch) = E_1(epoch) + 1;
84              else
85                  E_2(epoch) = E_2(epoch) + 1;
86              end
87          end
88      end
89      E = E_1 + E_2;
90
91      if (epoch > 10 && var(E(:,epoch-4:epoch)) < 0.05)
92          E_1(:,epoch+1:end) = [];
93          E_2(:,epoch+1:end) = [];
94          E(:,epoch+1:end) = [];
95          lambdaHist(:,epoch+1:end) = [];
96          break
97      end
98  end
99
100 plot(w(1:w_A,1), w(1:w_A,2), 'bh', 'markersize', 12);
101 plot(w(w_A+1:size(w,1),1), w(w_A+1:size(w,1),2), 'rP', 'markersize', 12);
102 lgnd = legend('Set A', 'Set B','Protypes A (init)', 'Protypes B (init)','Protypes A (final)'
        , 'Protypes B (final)');
103 set(lgnd, 'interpreter','latex', 'fontsize', 15);
104
105 subplot(3,1,2)
```

```
106 plot(E_1/200)
107 hold on;
108 plot(E_2/200);
109 plot(E/200);
110 legend('Training Error Class 1', 'Training Error Class 2', 'Total Training Error');
111 xlabel('Epochs')
112 ylabel('Training Error rate')
113
114 subplot(3,1,3)
115 plot(lambdaHist(1,:));
116 hold on
117 plot(lambdaHist(2,:));
118 legend('Lambda 1', 'Lambda 2');
119 xlabel('Epochs')
120 ylabel('Lambda value')
```