Introduction to Machine Learning
Lab Session 3

Group 39
Lubor Budaj (s4167376) & Gasan Rzaev (s3553213)

October 2, 2022

Assignment 3

1. Introduction

The goal of this assignment is to implement and perform M-fold cross validation for LVQ1 training, using the data set provided in the first assignment of this course (**lvqdata.mat**). There are ($P = 100$) examples in the data and each example has x and y parameter. There are 2 classes of data. The first 50 examples belong to class 1, while the rest of the examples belong to class 2.

As it is suggested, firstly the data set is shuffled randomly (by rows, so that the data does not lose it's values and classes), and then split into M equally sized disjoint subsets, each containing $(P/M)\%$ of the data (since our implementation allows us to split the data set into any number of equally sized disjoint subsets, we stick to the variable M in this report). Since, the labeling process (addition of a new column for identifying the class of each data point) starts before the shuffling and the splitting of the data set, all the labels remain intact for each data point throughout the process.

During each training process, $M - 1$ of the M subsets are being used for the training, while the remaining one is used for the validation process. The training and the validation processes are done with a K amount of prototypes per class. In the report we mainly focused on the suggested range for $K \in \{1, 2, 3, 4, 5\}$, but we also experiment with higher values of K too upto $K = 15$.

Moving on, the prototypes are initialized randomly by choosing a data point from its class, as it was done in the first assignment. *Training error* and *validation error* are determined at the end of each training process (after 100 epochs). Furthermore, this process is repeated for M possible splits, so that we receive M amount of individual results.

Finally, for each K, the average and the standard deviation of training and validation errors are computed over the M individual results that were obtained and we plot them as a function of K (the number of prototype per class).

2. Implementation

Our implementation is more flexible than just testing the M-fold cross validation with just $M = 5$ (amount of split subsets of the given data set), as well as with just K (prototypes per class) being in range between 1 and 5. Moreover, this implementation can accept any amount of K and M, obviously lower than the amount of data points in the data set (P). Of course testing the algorithm

with $M = 100$ or $M = 50$ is not going to give valuable results. Additionally, using too high values for K may also result in insufficient results, due to the higher complexity of the model it would lead to the large variance and so called over-fitting.
It is important to mentioned that the learning rate is the same as suggested in the first assignment of this course ($\eta = 0.002$), and the figures were generated using this value.

3. Results

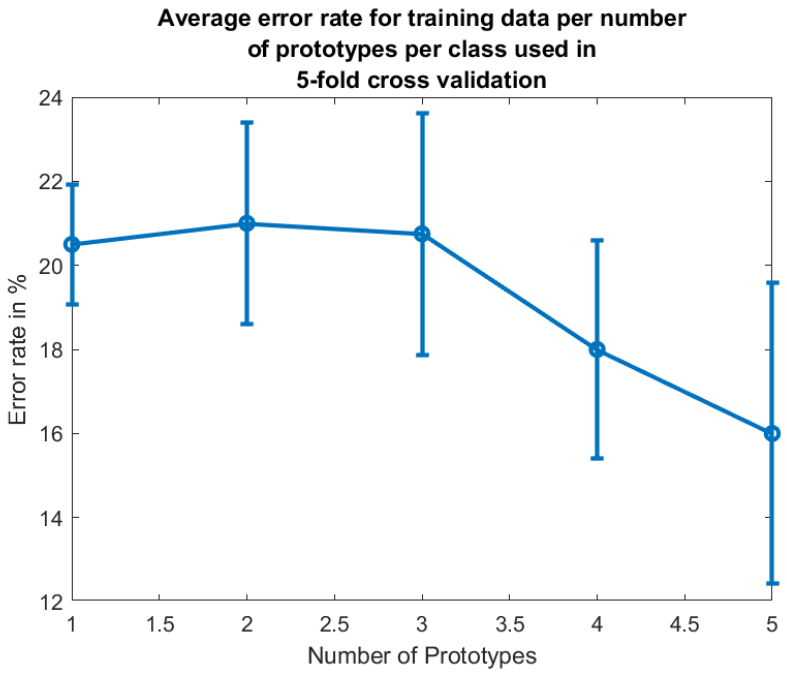

Figure 1

Figure 2

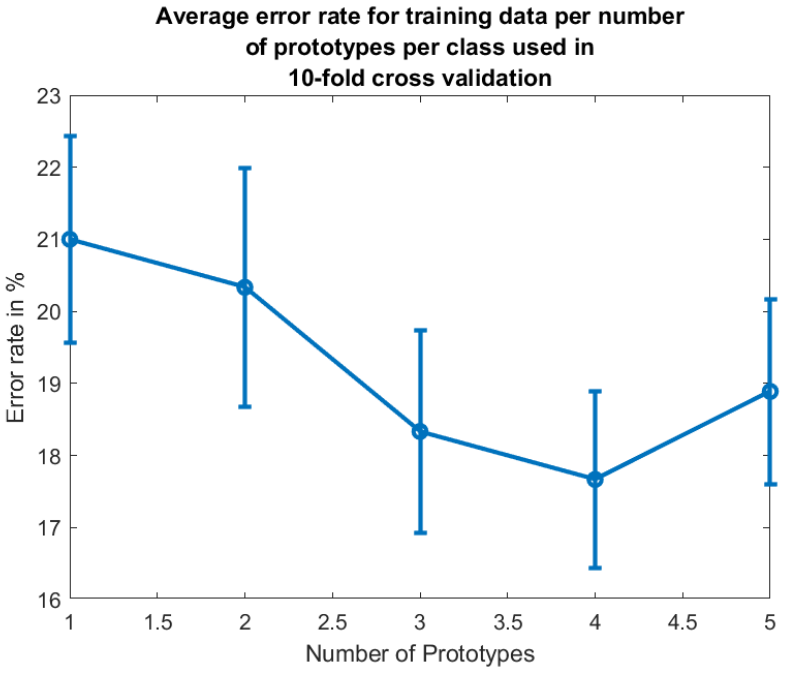

Figure 3

Figure 4

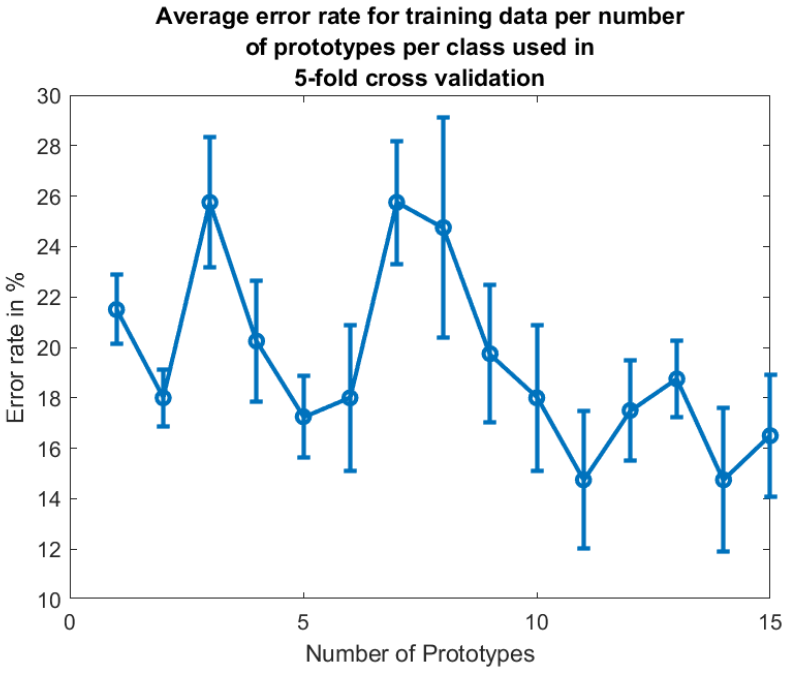

Figure 5

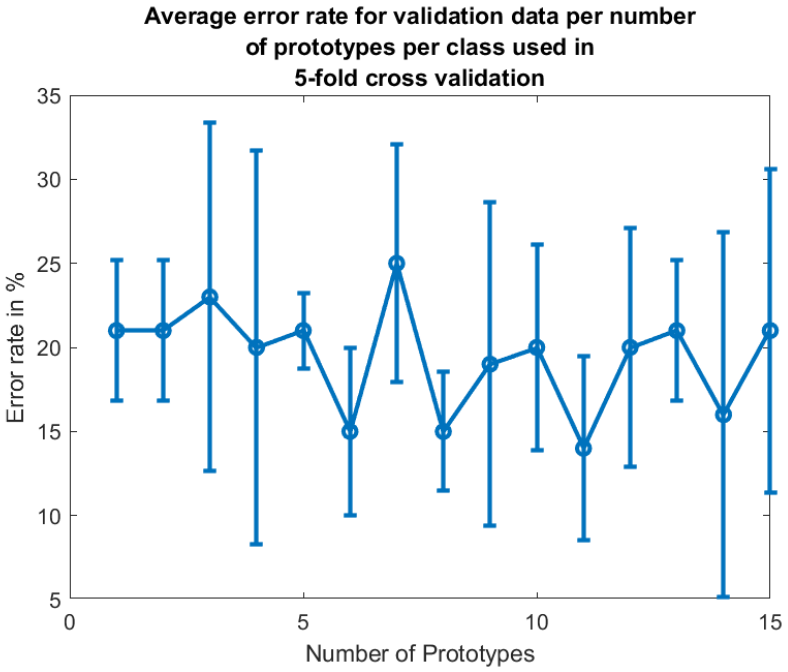

Figure 6

4. Discussion

First, let's compare the plots showing the error rate at 100^{th} epoch for different number of prototypes for training data (Figure 1) and validation data (Figure 2). In (Figure 1) we can see that the curve is down-ward slopping. On the other hand, (Figure 2) has 'V' shape. The difference between the curves for error rate in the training data and in the validation data can be explained by over-fitting in case of the training data. The prototypes positions are selected based on the training data, hence increasing the number of prototypes causes the error rate to decrease. On the other hand, concerning the validation data, we don't see this effect for increasing the number of prototypes per class. On the contrary, the error is actually increasing for higher values of K. Overall, the results match our expectations.

4.1. Finding the optimal number of prototypes per class

Secondly, let's discuss the resulting curve for 5-fold cross validation. In Figure 2 it can be seen that the lowest error rate was achieved when we used 2 or3 prototypes per class. Therefore we would choose one of these value as optimal value of prototypes for this application. To confirm or refute or choice, we tried running the program with number of prototypes per class up to 15. The resulting curves can be seen at Figure 5 and Figure 6. In the first figure showing the average training error we can see clear down-ward slopping trend, while the second figure showing the average error for validation data seems to be approximately linear. This confirms our earlier proposition of over-fitting at higher values for K. Although, there are some values of K, such as $K = 6$ or $K = 11$, when the average validation error rate is at it's lowest (15%), we believe this is caused purely by chance rather as there seem not to be any trend in the average error rate for different number of prototypes. We support our assumption by the fact that it is clearly noticeable that the standard deviation of the averages of errors in the validation data becomes significantly greater with increasing K than the standard deviation of averages of errors of the training data. It means that there is high difference between the different averages of validation error computed

during the cross validation, hence we don't think $K = 6$ or $K = 11$ are suitable choices for optimal K. It is computably more difficult to produce higher values of K, while at the same time there seem to be little benefit for doing so. Hence we choose $K = 2$ as our optimal value for number of prototypes per class.

4.2. Finding the optimal learning rate

We have experimented with different values of learning rate by running the program over and over again for different learning rates. Based on our observations, we concluded that the average error of both training and validation data drops down if we use more prototypes. However, it is important to say the results become even better if we used higher learning rate. We found a good value for the learning rate to be ($\eta = 0.01$). However, it is important to note that we found this learning rate to be optimal with fixed number of epochs ($t_{max} = 100$). If we increase the maximal number of epochs to higher number it is possible that other learning rates would be more optimal.

4.3. Trying increasing number of subsets M

By using higher order the cross validation, we expected the results to be more accurate. Comparing the figures obtained using 5-fold cross validation (Figure 1 and (Figure 2) and the ones using 10-fold cross validation (Figure 3 and (Figure 4), we cannot conclude our expectations. From the observation of the figures it seems that average error for particular number of prototypes increased for both training and validation data. Moreover, we can see is that the average standard deviation increased, showing that the results seem to be less accurate. This conclusion didn't meet our expectations We think this was caused by having too small validation subsets, which result in high bias. Hence, we believe doing 5-fold cross validation is sufficient for this particular dataset.

5. Work Distribution

The work among the group members was distributed in a following way:

- Code: 50% done by Gasan, 50% by Lubor
- Report: 50% done by Lubor, 50% by Gasan
- Graphs: 50% done by Lubor, 50% by Gasan

The code used to complete this assignment, that has not been included above, is presented in Listing 1 in Appendix A.

A. Assignment 3

Listing 1: *This is the file* `code.m`

```
%import
lvqdata = importdata("lvqdata.mat");
%initialization
dimensions = size(lvqdata);
N = dimensions(2) + 1;
P = dimensions(1);

%maximum amount of prototypes per class
K_max = 5;

%learning rate
n = 0.002;

%amount of subsets of the main dataset
m = 5;

%size of each subset
subsize = P / m;

%maximum amount of epochs
t_max = 100;

%initialization of prototypes
prototypes = zeros(K_max*2,N);

%initialization of error vectors
E_train = zeros(1,m);
E_validation = zeros(1, m);

%assigning class
for x = 1:P
    if x > P/2
        lvqdata(x,3) = 2;
    else
        lvqdata(x,3) = 1;
    end
end

%random permutation of indices
r = randperm(P);

%initialize the array of average errors
average_train = zeros(1, K_max);
average_validation = zeros(1, K_max);

%initialize the array of standard deviations of errors
std_train = zeros(1, K_max);
```

```
std_validation = zeros(1, K_max);

%loop for different amounts of prototypes per class
for K = 1:K_max

    %sorting the dataset by classes in order to initialize prototypes
    %correctly after each iteration for K
    lvqdata = sortrows(lvqdata, 3);

    %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

    %shuffle the data set to match other iterations
    lvqdata = lvqdata(r,:);

    %for each epoch
    for i = 1:m
        for t = 1:t_max

            %random permutation of examples
            p = randperm(P);

            %initialize error counters
            train_error = 0;
            validation_error = 0;

            %for each example
            for x = 1:P
                %checking if the datapoint is in the training subset
                if x > subsize*i || x <= subsize*(i-1)
                    %finding closest prototype
                    closest_prototype = 1;
                    minimal_distance = sqrt((lvqdata(p(x),1) -
   prototypes(1,1))^2 + (lvqdata(p(x),2) - prototypes(1,2))^2);
                    for k = 2:(2*K)
                        distance = sqrt((lvqdata(p(x),1) - prototypes(
   k,1))^2 + (lvqdata(p(x),2) - prototypes(k,2))^2);
                        if distance < minimal_distance
                            closest_prototype = k;
                            minimal_distance = distance;
                        end
                    end
                    %moving distance calculation
                    distX = n * (lvqdata(p(x),1) - prototypes(
   closest_prototype,1));
                    distY = n * (lvqdata(p(x),2) - prototypes(
```

```
closest_prototype,2));
                %direction
                if lvqdata(p(x),3) ~= prototypes(closest_prototype
,3)
                    distX = -distX;
                    distY = -distY;
                end
                %moving
                prototypes(closest_prototype, 1) = prototypes(
closest_prototype, 1) + distX;
                prototypes(closest_prototype, 2) = prototypes(
closest_prototype, 2) + distY;
            end
        end
        for x = 1:P
            %finding closest prototype
            closest_prototype = 1;
            minimal_distance = sqrt((lvqdata(p(x),1) - prototypes
(1,1))^2 + (lvqdata(p(x),2) - prototypes(1,2))^2);
            for k = 2:(2*K)
                distance = sqrt((lvqdata(p(x),1) - prototypes(k,1)
)^2 + (lvqdata(p(x),2) - prototypes(k,2))^2);
                if distance < minimal_distance
                    closest_prototype = k;
                    minimal_distance = distance;
                end
            end
            %error determination
            if lvqdata(p(x),3) ~= prototypes(closest_prototype,3)
                % if the example is in the validation or training
subset
                if x <= subsize*i && x > subsize*(i-1)
                    validation_error = validation_error + 1;
                else
                    train_error = train_error + 1;
                end
            end

        end

    end

    %appending training error percentage after the last epoch
    E_train(i) = (train_error / ((P/m)*(m-1)))*100;

    %appending validation error percentage after the last epoch
    E_validation(i) = (validation_error / (P/m))*100;

end

%calculating averaage errors and their standard deviation
```

```
    average_train(K) = mean(E_train);
    average_validation(K) = mean(E_validation);
    std_train(K) = std(E_train);
    std_validation(K) = std(E_validation);
end

%%% PLOTTING
errorbar([1:K_max], average_train, std_train, 'LineWidth', 2, 'Marker',
    "o");
xlabel('Number of Prototypes');
ylabel('Error rate in %');
name = strcat(int2str(m), "-fold cross validation");
title({"Average error rate for training data per number"; "of
   prototypes per class used in";name});

errorbar([1:K_max], average_validation, std_validation, 'LineWidth', 2,
    'Marker', "o");
xlabel('Number of Prototypes');
ylabel('Error rate in %');
title({"Average error rate for validation data per number"; "of
   prototypes per class used in";name});
```