LVQ & LFM Implementation in MATLAB

Christos Panourgias

Contents

1 LVQ2

1.1 Introduction

Learning Vector Quantization 2 (LVQ2) is a learning algorithm used for data categorization. It consists of a collection of prototypes which are points/vectors in space. Each prototype has a class and is specifically defined to represent a class of data. LVQ2 then assigns the data to some of the prototypes, based on their category. The learning algorithm is repeated several times (epochs) and each time re-adjusts the prototypes to better represent the data. It achieves this by renewing the prototypes according to the following conditions:

1. The prototype, w_i, with the minimum distance from the training point, $x(t)$, must be in a different class than $x(t)$
2. The prototype, w_j, with the second smallest distance from the training point, $x(t)$, must be in the same class as $x(t)$
3. The training point, $x(t)$, must lie within a small symmetric region around the midpoint of the prototypes w_i and w_j

If the above conditions apply, then a refresh step is applied to the originals according to the following formulas :

$w_i(t+1) = w_i(t) - (t)(x(t) - w_i(t))$
$w_j(t+1) = w_j(t) + (t)(x(t) - w_j(t))$

Where $a(t)$ is the learning factor (learning rate) that LVQ2 uses to adjust the scale of change of prototypes in each epoch.

1.2 Implementation in MATLAB

1.2.1 Functions

- **data = extract_data(file, file_name, spare_lines)**
  This function accepts as parameters the file, the name of the file and the number of lines and returns the data of the file in an array.
  1. First the first lines that are not data are removed.
  2. Then using two nested loops for and regular expressions the data numbers are extracted from each line of the file.
  3. The line is added to the data array and then allowed that array.
- **[X_train, y_train, X_val, y_val, X_test, y_test] = data_split(X, y, division)**
  This function accepts as parameters the features (features), the categories (labels) and the division $\in \{1, 2, 3, ..., 10$ which will define how the data will be split into training set, validation set and test set. Returns the data split into training set, validation set and test set.

1. Initially, the size percentages that will correspond to each subset are determined.
2. Then using a switch expression, divide the data into training set, validation set, and test set according to the value entered in the variable division
3. The split data X_train, y_train, X_val, y_val, X_test, y_test is then returned.

- **dist = eucl(x, y)**
  This function takes as parameters two vectors and returns the Euclidean norm of their difference.

- **W = init_W(X, y, ppc)**
  This function accepts as parameters, the features (features), the labels of the categories (labels) and the number of neurons. Returns an array W whose columns are randomly selected vectors from the data.

  Note : the number of neurons will be needed to determine the dimensions of the W array.

  1. First the dimensions of the table W are determined ($nin \times nout \cdot ppc$)
  2. A for loop is created that iterates over the number of classes, in which lines belonging to the current class are extracted.
  3. A second for loop is created which runs through a number of neurons and is nested within the first loop. Inside this loop a pointer in the range $(1, length(r))$ is randomly selected. Using this pointer a row is randomly selected from the rows of the current class that were selected in the first loop. Then this row is added as a column to the W table, finally this row is removed from the rows of the current class so that there is no chance of it being selected a second time.
  4. After the two nested loops terminate, the W array is returned.

- **a_t = a(a0, Ka, t)**
  This function takes as parameters the initial learning rate, the variable K_a, and t which specifies the current time. Returns the learning rate at time t.

- **accuracy = accuracy(W, X, y, ppc)**
  This function accepts as parameters the array of trained prototypes W, the data on which the accuracy will be checked classification (X, y) and the number of neurons ppc. Returns the classification accuracy of the given set.

  1. The table y_pred is created which will contain the predictions of the trained model and the dimensions of the tables X and W are stored.

2. A loop is created that iterates over the number of rows of the array X, in which, using a second loop that iterates over the number of prototypes, the Euclidean distances between the current of the table X and the originals.
3. Then the prototype that has the smallest distance from the array element X, compared to the rest of the prototypes, is determined as the "winning" prototype.
4. After the original winner index is found then its class is calculated, and the array element X is categorized in the same class as the original winner.
5. Finally, the classes of the table of predictions, y_pred, are compared with the classes of the table of true classes, y, and the ratio of the correct predictions to the size of the samples is returned (that is, the accuracy of each total).

- **[W, test_accs, train_accs, val_accs] =**
  = LVQ_2(X, y, X_test, y_test, X_val, y_val, epochs, Ka, a0, s, ppc)
  This function accepts as parameters the training subset (X, y), the test subset (X_test , y_test), the subset of documentation (X_val, y_val) the number of times to read the training data (epochs), the learning rate parameters, K_a and a0, the window size s and the number of neurons ppc. Returns the trained matrix W and the accuracy of each subset (the accuracy metrics of the subsets are needed to generate the requested graphs).
  Note : This function implements the LVQ2 algorithm so that it returns the trained prototypes.

  1. Initialize array W using init_W()
  2. Initialize arrays to store the accuracy metrics so that the graphs are implemented.
  3. Two basic for loops are used in which the first iterates over the epochs and the second iterates over the number of rows of features in the training set.
  4. Within the above loops a third loop is created in which the Euclidean distances between the prototypes and the current training point are calculated. These distances are then stored in a table.
  5. Using the above array, store the minimum (which corresponds to the closest prototype to the training point) and its index, then remove it from the array and store the second minimum (which corresponds to the second closest prototype to the training point) and its index of.
  6. The pointers are used to find the class of the prototypes, and the y array is used to find the class of the current training point.
  7. It is then checked if the conditions of LVQ2 are valid and if they are, a refresh step is applied.
  8. Finally using the function accuracy() the accuracy measurements are stored in the lists for each epoch.

1.2.2 Basic Program

1. The file from which the data will be extracted is selected. The data is then extracted using the extract_data() function. After the data is stored, a permutation is applied to it to simulate a random selection of the data.

2. The dataset, depending on the file selection, is divided into X features and y labels.

3. The parameters and the tables that will contain the results of the experiments are initialized.

4. Three for loops are created which run through the requested values of the hyperparameters. Within the loops, 10-fold cross-validation is applied using data_split() to split the data and LVQ_2 to train the data.

5. The accuracy metrics of each fold are calculated and stored.
   Also stores the accuracy count vectors for each epoch (the ones returned by the function LVQ_2())

6. The average of the accuracy measurements for each epoch is calculated separately so that a graph can then be created. The accuracy metrics at the end of the seasons are averaged (these metrics will be represented in the results table).

7. The results are saved and the execution time is recorded.

8. The index of the experiment that had the best
   performance in the documentation set is found and its results and hyperparameters are printed.

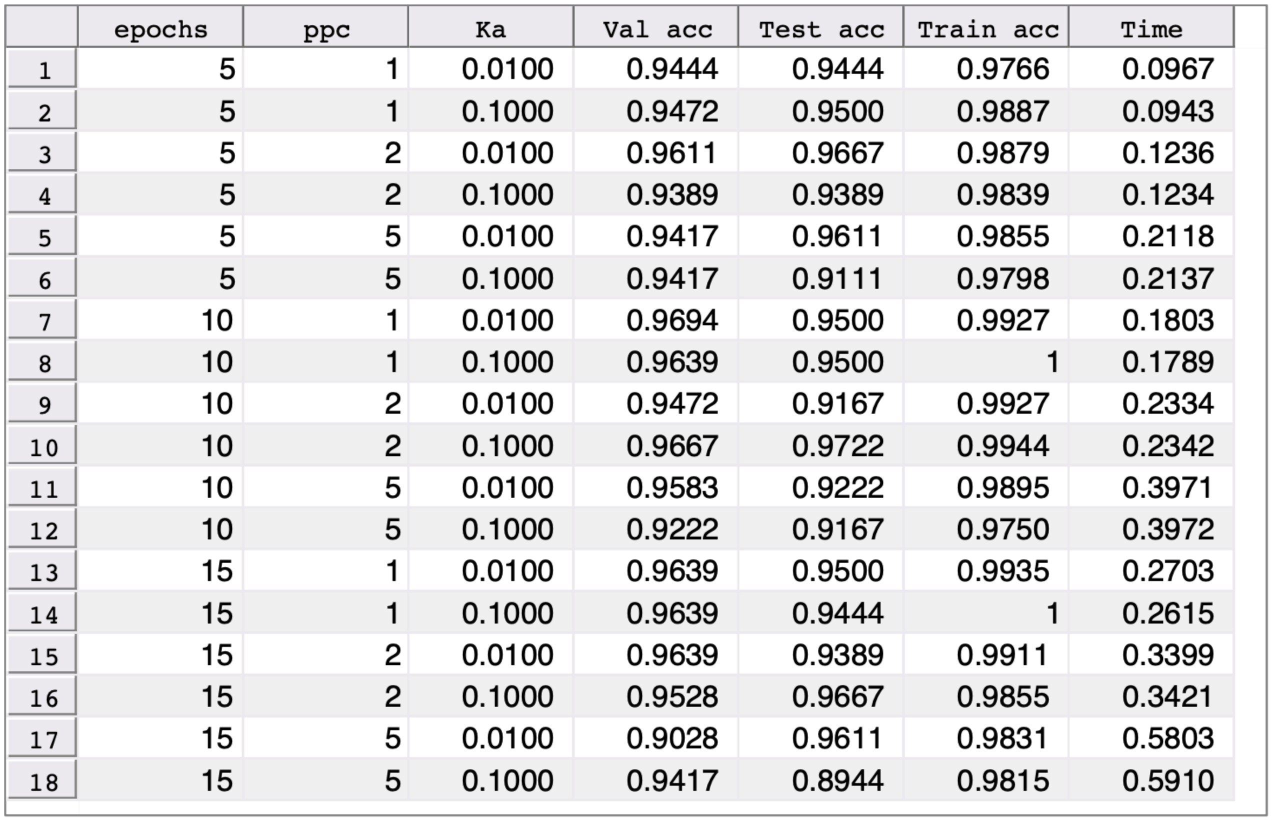
9. Using the precision metrics per epoch returned by the
   function LVQ_2(), graphs are created that represent the
   precision metrics of the training, documentation, and test sets
   with respect to epochs.

10. Finally a table is created using the function uitable() of MATLAB which contains the results and hyperparameters for each experiment.

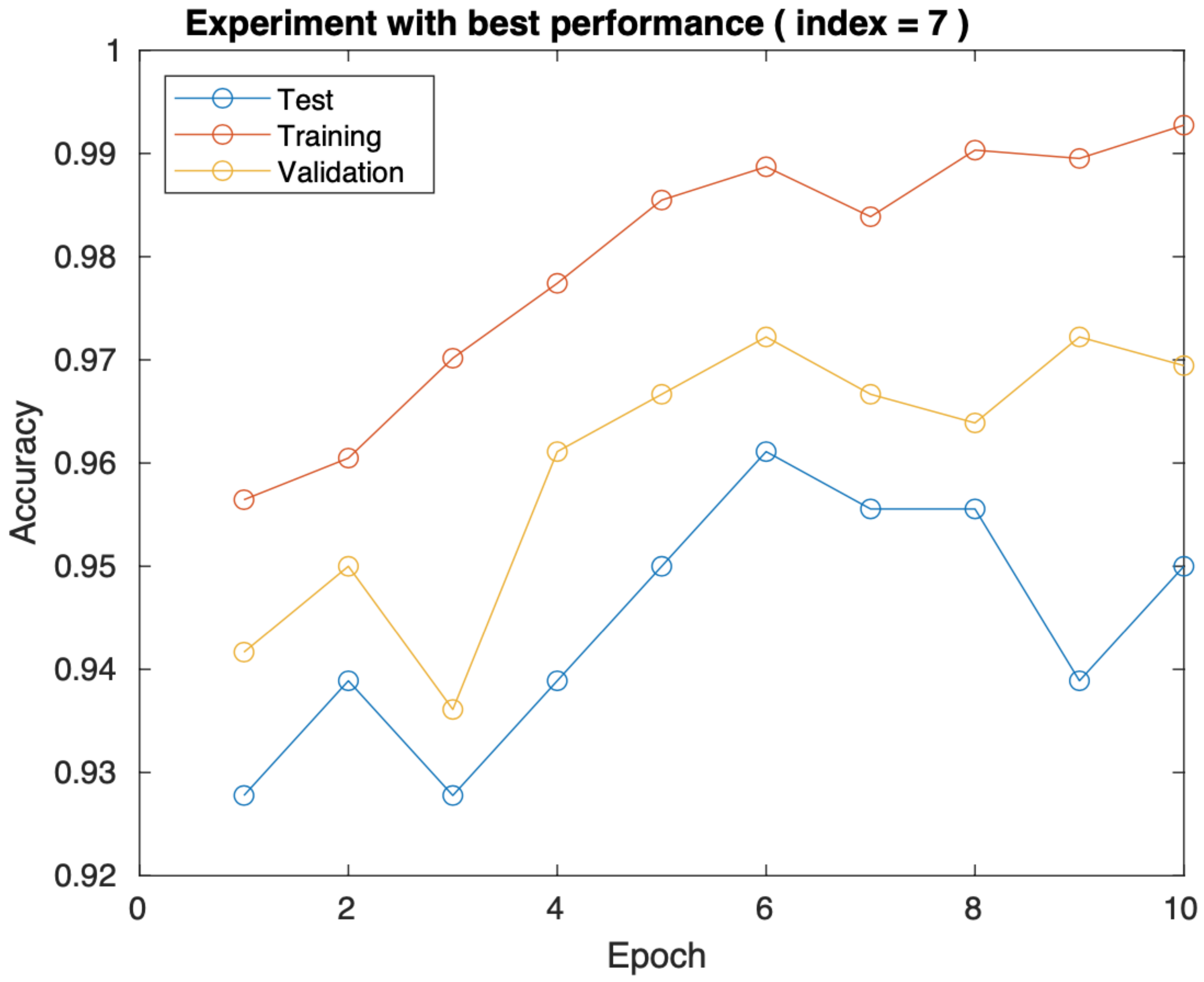
1.3 Results

1.3.1 WINE dataset

LVQ2-WINE.TXT



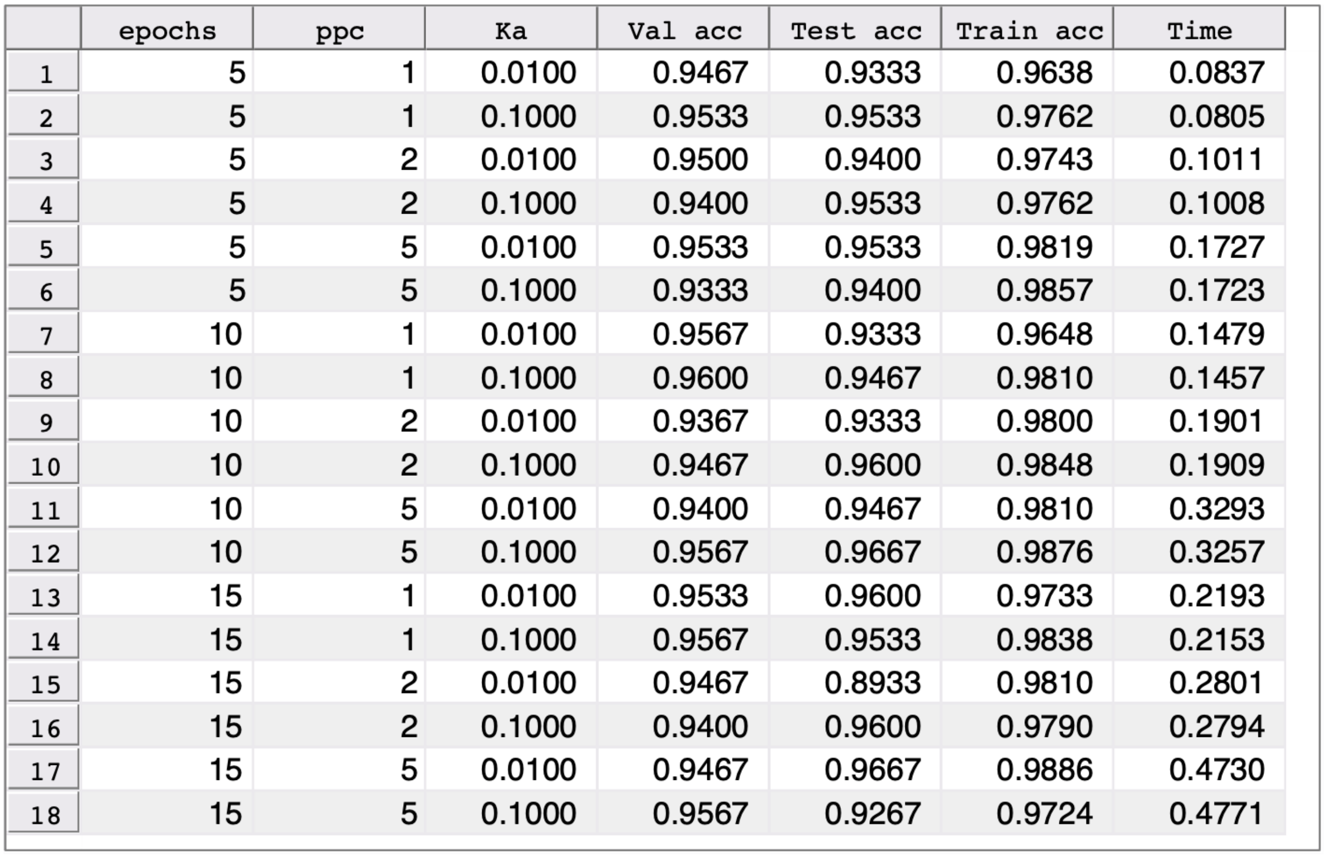
	epochs	ppc	Ka	Val acc	Test acc	Train acc	Time
1	5	1	0.0100	0.9444	0.9444	0.9766	0.0967
2	5	1	0.1000	0.9472	0.9500	0.9887	0.0943
3	5	2	0.0100	0.9611	0.9667	0.9879	0.1236
4	5	2	0.1000	0.9389	0.9389	0.9839	0.1234
5	5	5	0.0100	0.9417	0.9611	0.9855	0.2118
6	5	5	0.1000	0.9417	0.9111	0.9798	0.2137
7	10	1	0.0100	0.9694	0.9500	0.9927	0.1803
8	10	1	0.1000	0.9639	0.9500	1	0.1789
9	10	2	0.0100	0.9472	0.9167	0.9927	0.2334
10	10	2	0.1000	0.9667	0.9722	0.9944	0.2342
11	10	5	0.0100	0.9583	0.9222	0.9895	0.3971
12	10	5	0.1000	0.9222	0.9167	0.9750	0.3972
13	15	1	0.0100	0.9639	0.9500	0.9935	0.2703
14	15	1	0.1000	0.9639	0.9444	1	0.2615
15	15	2	0.0100	0.9639	0.9389	0.9911	0.3399
16	15	2	0.1000	0.9528	0.9667	0.9855	0.3421
17	15	5	0.0100	0.9028	0.9611	0.9831	0.5803
18	15	5	0.1000	0.9417	0.8944	0.9815	0.5910


Experiment with best performance (index = 7)
Test
Training
Validation
Accuracy
Epoch
1
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0
2
4
6
8
10

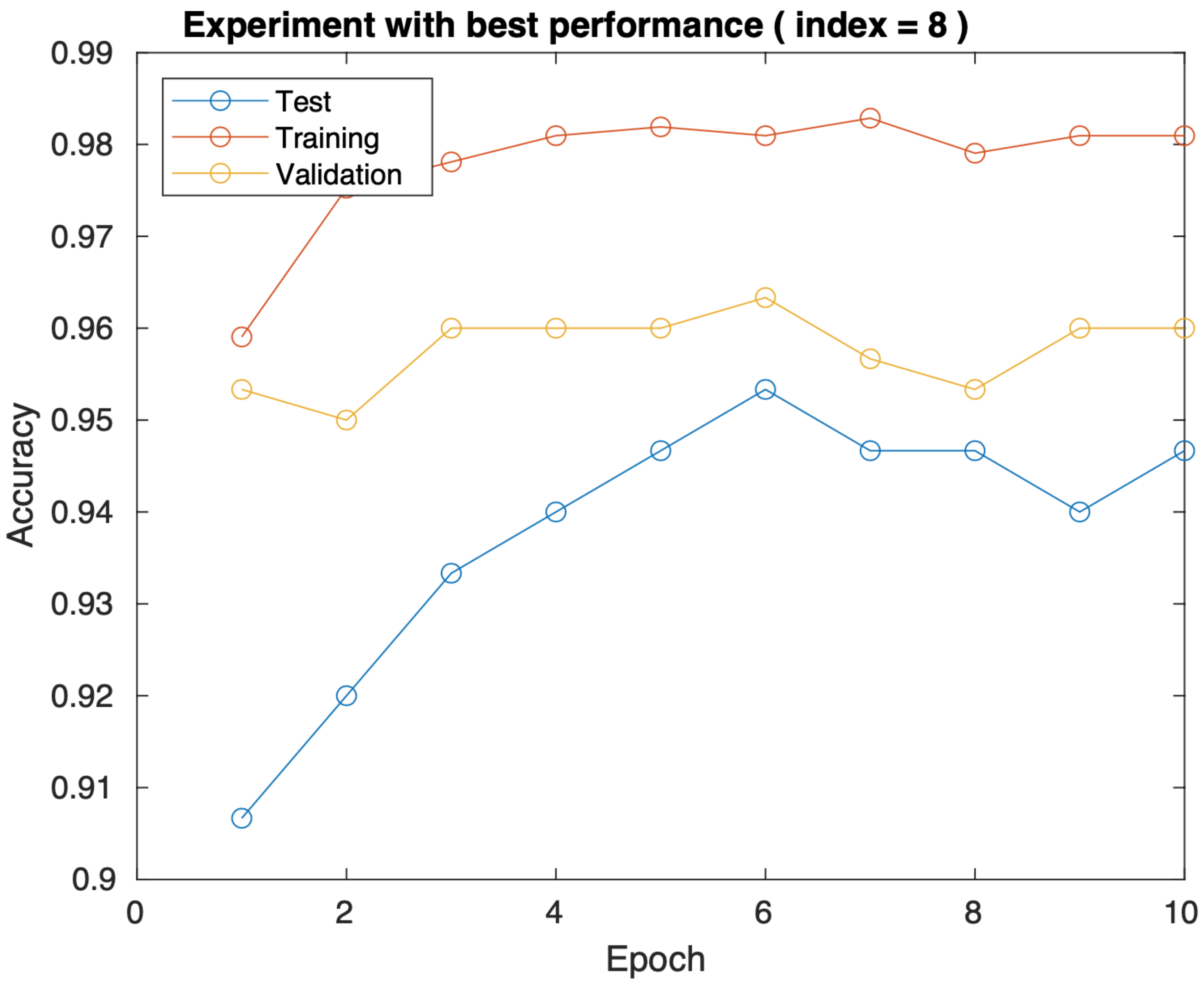
Note: The seventh experiment showed the best performance with respect to the documentation set. Overtraining is observed in the graphs after the sixth epoch because the performance on the training set increases while the performance on the test set decreases.

1.3.2 IRIS dataset

LVQ2-IRIS.TXT



	epochs	ppc	Ka	Val acc	Test acc	Train acc	Time
1	5	1	0.0100	0.9467	0.9333	0.9638	0.0837
2	5	1	0.1000	0.9533	0.9533	0.9762	0.0805
3	5	2	0.0100	0.9500	0.9400	0.9743	0.1011
4	5	2	0.1000	0.9400	0.9533	0.9762	0.1008
5	5	5	0.0100	0.9533	0.9533	0.9819	0.1727
6	5	5	0.1000	0.9333	0.9400	0.9857	0.1723
7	10	1	0.0100	0.9567	0.9333	0.9648	0.1479
8	10	1	0.1000	0.9600	0.9467	0.9810	0.1457
9	10	2	0.0100	0.9367	0.9333	0.9800	0.1901
10	10	2	0.1000	0.9467	0.9600	0.9848	0.1909
11	10	5	0.0100	0.9400	0.9467	0.9810	0.3293
12	10	5	0.1000	0.9567	0.9667	0.9876	0.3257
13	15	1	0.0100	0.9533	0.9600	0.9733	0.2193
14	15	1	0.1000	0.9567	0.9533	0.9838	0.2153
15	15	2	0.0100	0.9467	0.8933	0.9810	0.2801
16	15	2	0.1000	0.9400	0.9600	0.9790	0.2794
17	15	5	0.0100	0.9467	0.9667	0.9886	0.4730
18	15	5	0.1000	0.9567	0.9267	0.9724	0.4771



Note: The eighth experiment showed the best performance with respect to the documentation set. The graphs show overtraining after the sixth epoch for the same reasons as before.

2 LFM

2.1 Introduction

Learning From Mistakes (LFM) is an algorithm that involves
iteratively adjusting a model's parameters based on mistakes made during training. The goal of LFM is to improve model accuracy by minimizing the error between predicted and actual output. It achieves this with the following methodology:
For each training point there is its closest prototype with respect to the Euclidean norm.
If this prototype is in the same class as the training point, then no parameter adjustment is applied.
If the prototype is in a different class, then the prototype is moved away from the training point in proportion to their Euclidean distance
according to the following formula:

$w_i(t+1) = w_i(t) - (t)(x(t) - w_i(t))$

After this process, the closest prototype to the training point is found, which is in the same class as the training point. This prototype is moved closer to the training point in proportion to their Euclidean distance according to the following formula :

$w_j(t+1) = w_j(t) + (t)(x(t) - w_j(t))$

2.2 Implementation in MATLAB

Note: The LFM algorithm implementation in MATLAB is similar to the LVQ2 implementation. The difference lies in the function LVQ_2() which will be replaced by LeFM().

- **[W, test_accs, train_accs, val_accs] =**
  = LeFM(X, y, X_test, y_test, X_val, y_val, epochs , Ka, a0, ppc) This function accepts the same parameters as LVQ_2() and returns the same variables, with the difference between the two functions being in the training of the prototypes (i.e. the table W.

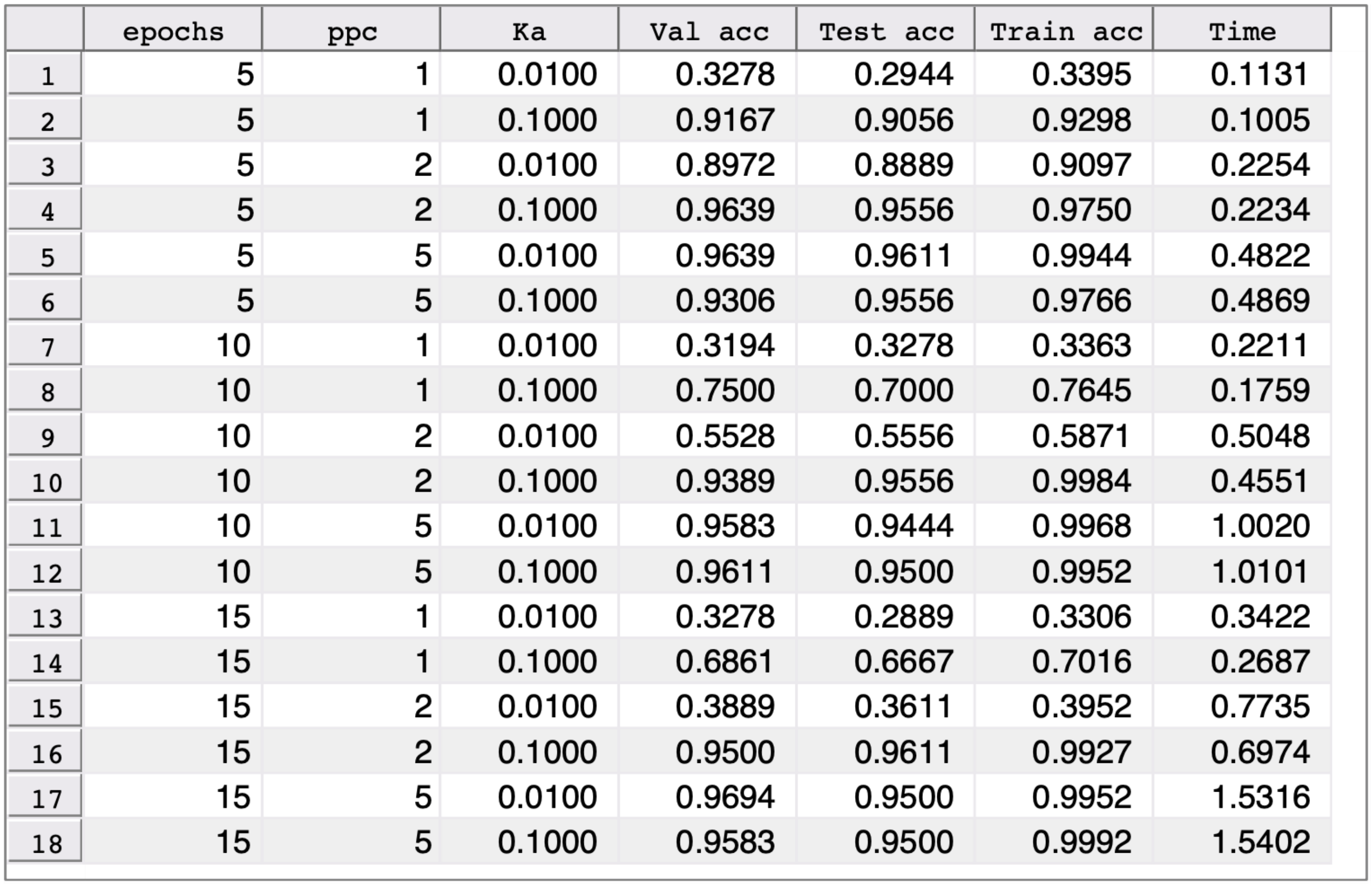
  1. In the two nested for loops the Euclidean distances between the prototypes and the current training point are calculated.
  2. Then the minimum of the distances from which
     the index is taken is calculated. Through the index the class of the prototype is calculated.

3. The class of the prototype is compared to the class of the current training point and in case they are different,
   the corresponding refresh step of the prototype is applied.
4. A list (same_class) is initialized which will contain all prototypes of the same class as the class of the current
   training point. And a list that will contain the pointers to the array W
5. A loop is created which stores the prototypes that have the same class as the current training point and their
   pointers in the lists above. The distances of these prototypes with the current training point are then calculated and stored in a list.
6. From these distances, the minimum distance is selected and the index of the prototype corresponding to the matrix W is extracted. Finally, According to this index, the renewal of the
   prototype is applied.

2.3 Results

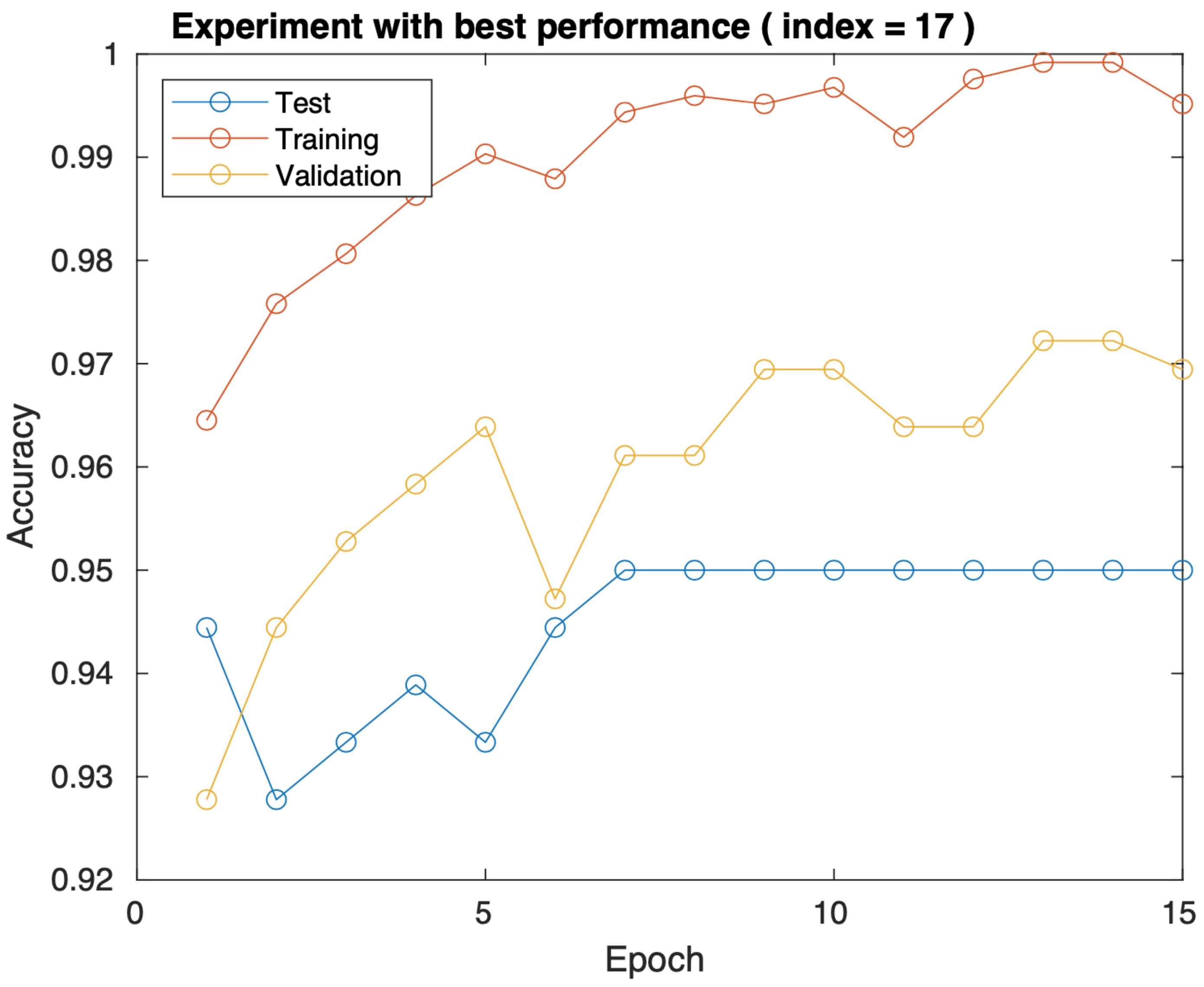
2.3.1 WINE dataset

LFM-WINE.TXT



	epochs	ppc	Ka	Val acc	Test acc	Train acc	Time
1	5	1	0.0100	0.3278	0.2944	0.3395	0.1131
2	5	1	0.1000	0.9167	0.9056	0.9298	0.1005
3	5	2	0.0100	0.8972	0.8889	0.9097	0.2254
4	5	2	0.1000	0.9639	0.9556	0.9750	0.2234
5	5	5	0.0100	0.9639	0.9611	0.9944	0.4822
6	5	5	0.1000	0.9306	0.9556	0.9766	0.4869
7	10	1	0.0100	0.3194	0.3278	0.3363	0.2211
8	10	1	0.1000	0.7500	0.7000	0.7645	0.1759
9	10	2	0.0100	0.5528	0.5556	0.5871	0.5048
10	10	2	0.1000	0.9389	0.9556	0.9984	0.4551
11	10	5	0.0100	0.9583	0.9444	0.9968	1.0020
12	10	5	0.1000	0.9611	0.9500	0.9952	1.0101
13	15	1	0.0100	0.3278	0.2889	0.3306	0.3422
14	15	1	0.1000	0.6861	0.6667	0.7016	0.2687
15	15	2	0.0100	0.3889	0.3611	0.3952	0.7735
16	15	2	0.1000	0.9500	0.9611	0.9927	0.6974
17	15	5	0.0100	0.9694	0.9500	0.9952	1.5316
18	15	5	0.1000	0.9583	0.9500	0.9992	1.5402

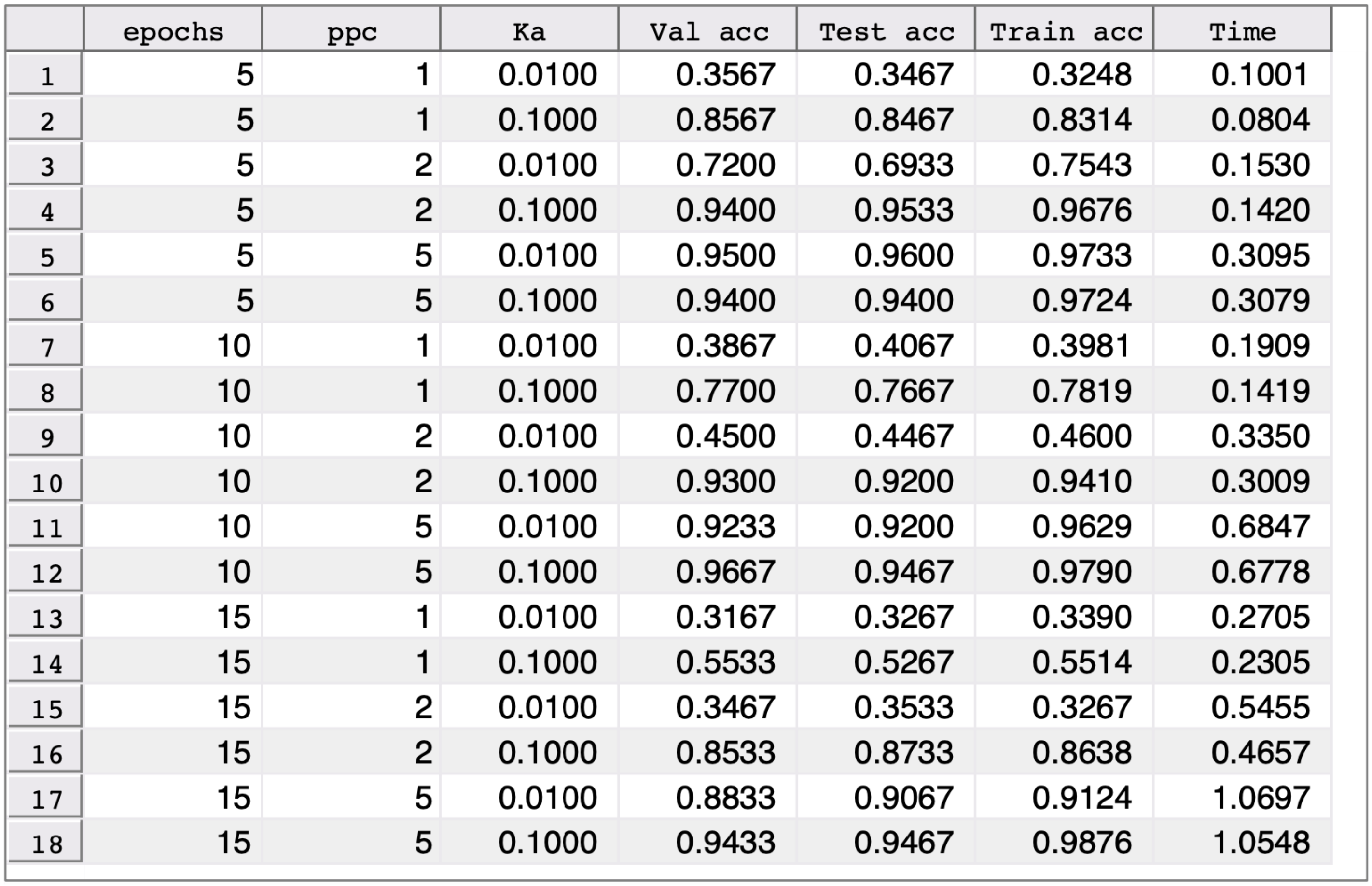
Remark: Compared to the LVQ2 algorithm the overall accuracy measures have a more moderate performance.



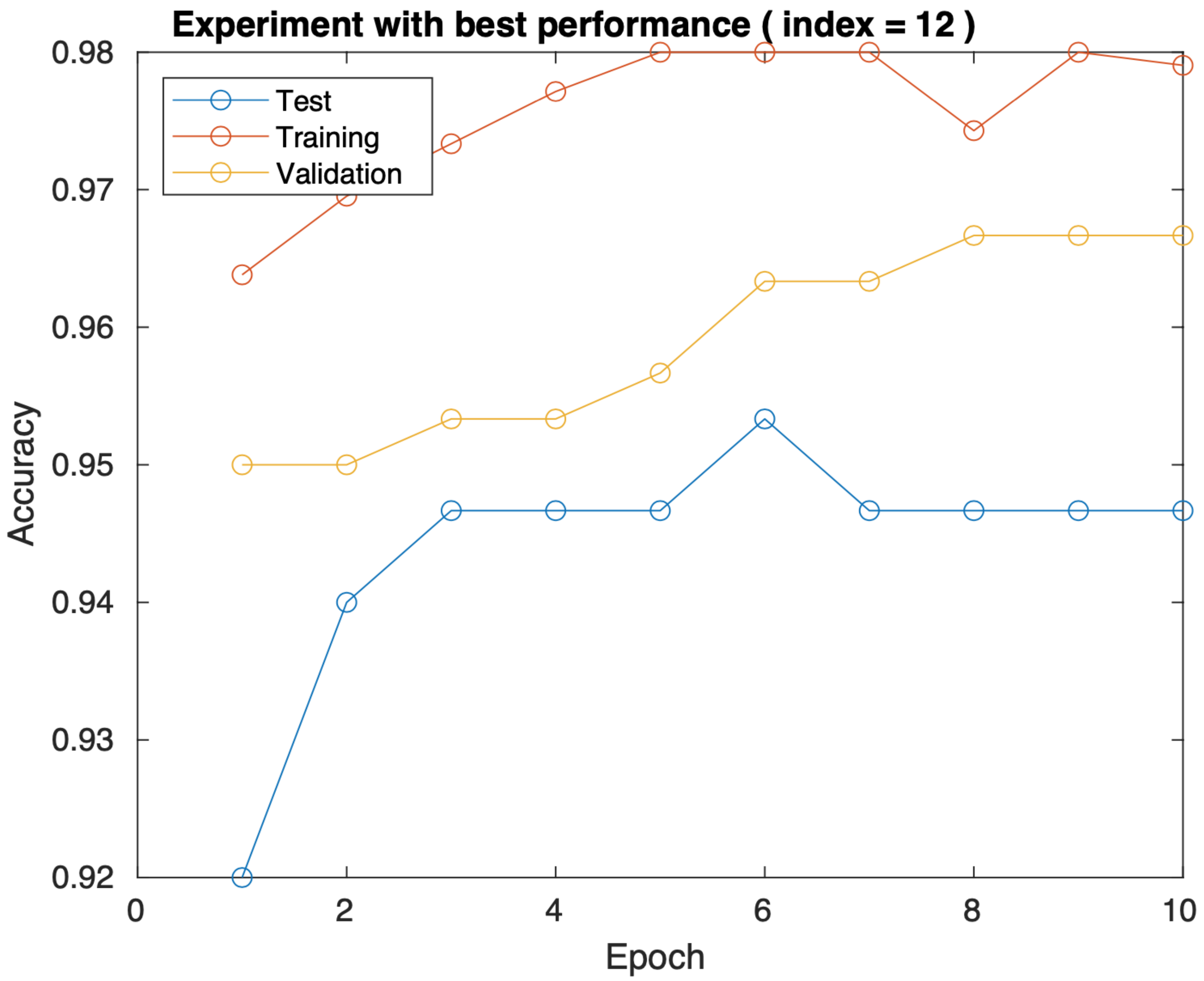
Note: A stabilization of the accuracy measurement of the test set is observed after the seventh epoch, while the accuracy of the other two sets has an increasing trend.

2.3.2 IRIS dataset

LFM-IRIS.TXT



	epochs	ppc	Ka	Val acc	Test acc	Train acc	Time
1	5	1	0.0100	0.3567	0.3467	0.3248	0.1001
2	5	1	0.1000	0.8567	0.8467	0.8314	0.0804
3	5	2	0.0100	0.7200	0.6933	0.7543	0.1530
4	5	2	0.1000	0.9400	0.9533	0.9676	0.1420
5	5	5	0.0100	0.9500	0.9600	0.9733	0.3095
6	5	5	0.1000	0.9400	0.9400	0.9724	0.3079
7	10	1	0.0100	0.3867	0.4067	0.3981	0.1909
8	10	1	0.1000	0.7700	0.7667	0.7819	0.1419
9	10	2	0.0100	0.4500	0.4467	0.4600	0.3350
10	10	2	0.1000	0.9300	0.9200	0.9410	0.3009
11	10	5	0.0100	0.9233	0.9200	0.9629	0.6847
12	10	5	0.1000	0.9667	0.9467	0.9790	0.6778
13	15	1	0.0100	0.3167	0.3267	0.3390	0.2705
14	15	1	0.1000	0.5533	0.5267	0.5514	0.2305
15	15	2	0.0100	0.3467	0.3533	0.3267	0.5455
16	15	2	0.1000	0.8533	0.8733	0.8638	0.4657
17	15	5	0.0100	0.8833	0.9067	0.9124	1.0697
18	15	5	0.1000	0.9433	0.9467	0.9876	1.0548



Note: While a maximum value is noted in the sixth epoch, from the third epoch onwards there is a tendency to stabilize the accuracy of the test set. However, from the third epoch onwards, the accuracy metrics of the other two ensembles show an upward trend.

3 Conclusions

In this particular training set, the two algorithms perform similarly on both datasets. It is observed that for several combinations of hyperparameters, in contrast to the LVQ2 algorithm, the LFM algorithm has moderate performances. Moreover, in this training set, the accuracy of the test set for the LVQ2 algorithm shows steeper declines when the training set is overtrained.