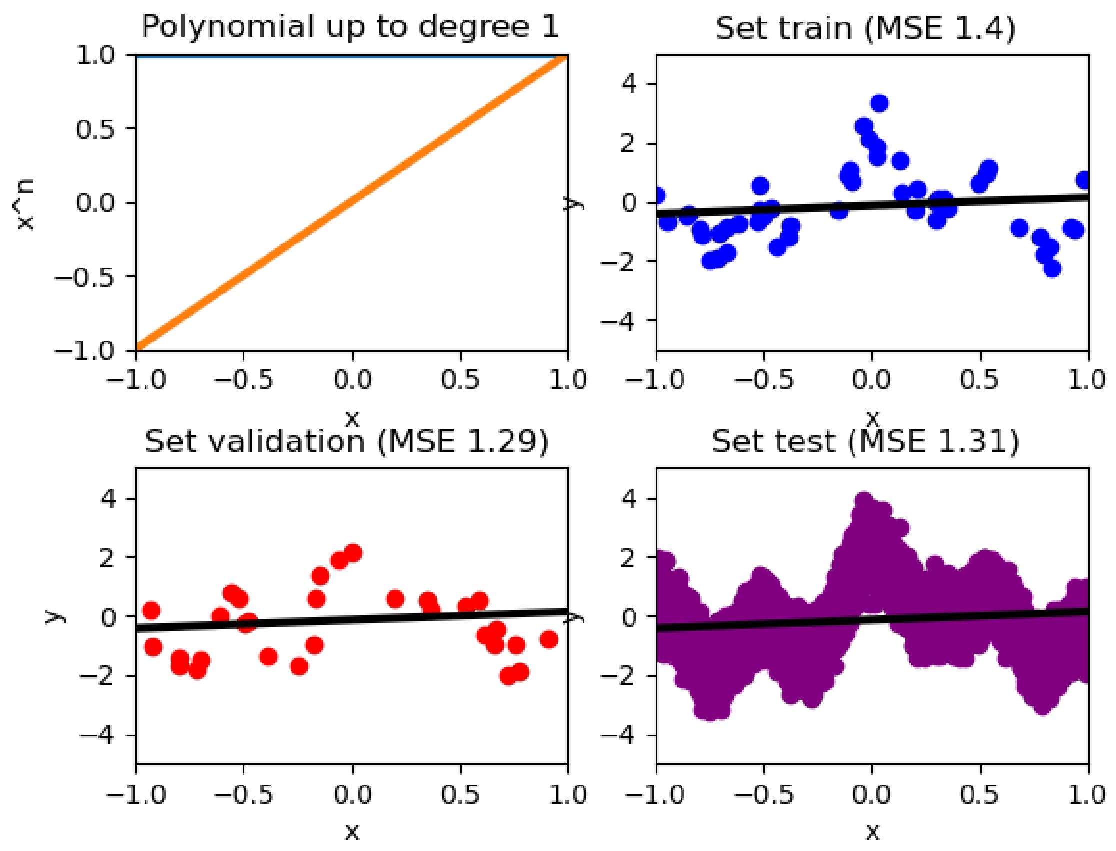
Assignment 2

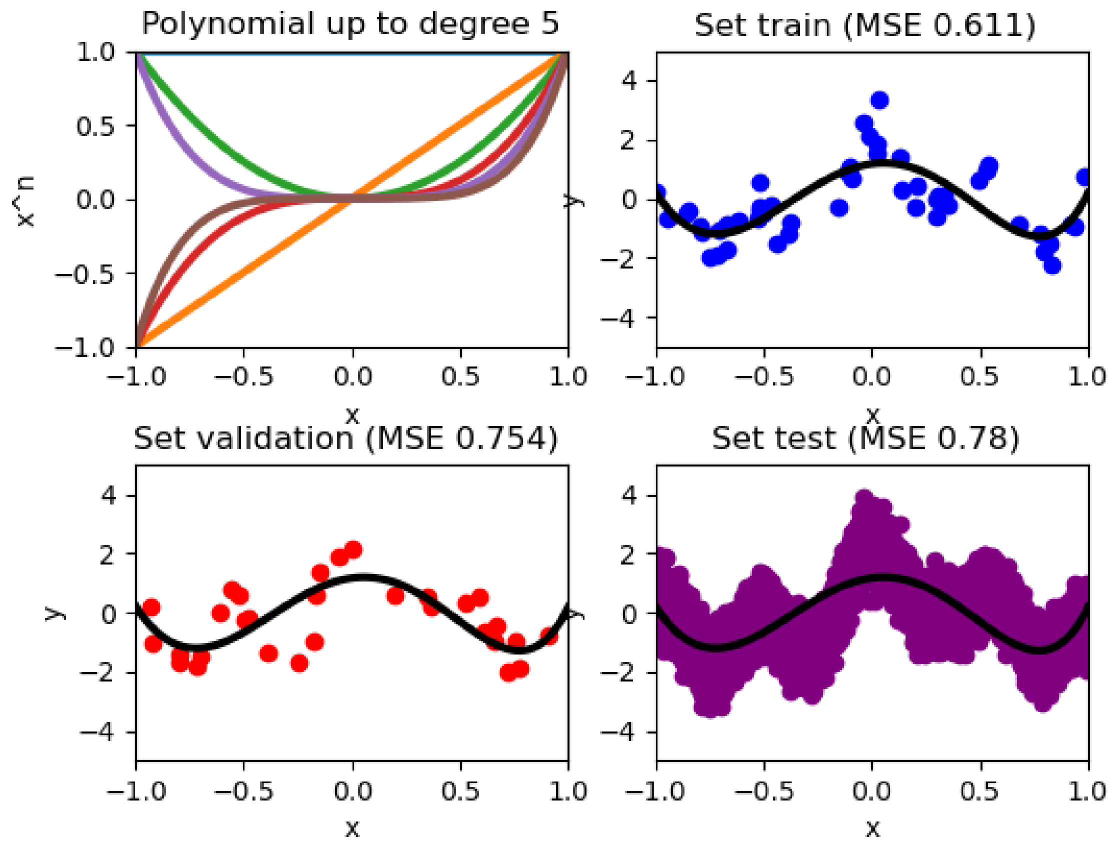
Computational Intelligence, SS2020

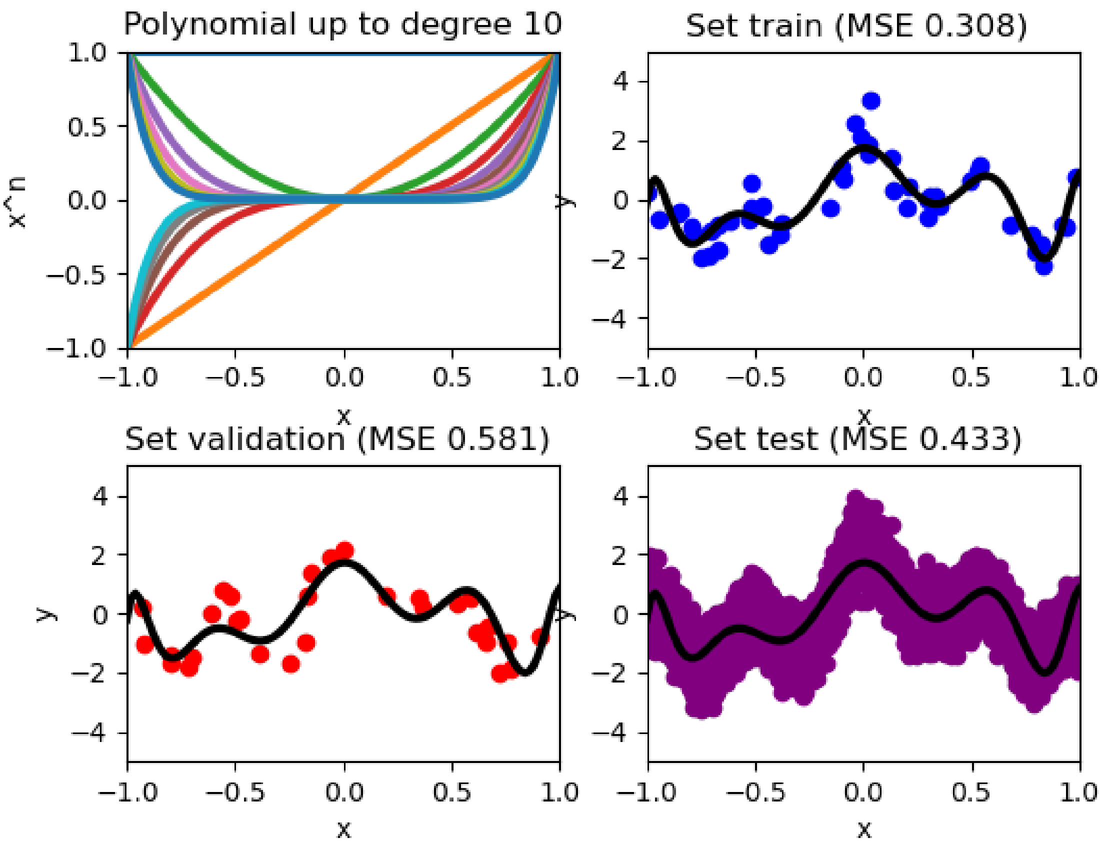
Team Members		
Last name	First name	Matriculation Number
Stiegler	Harald	9330054

1.2.1. Using plot poly in main poly.py, plot your results for each of the following degrees: 1, 5, 10, 22.

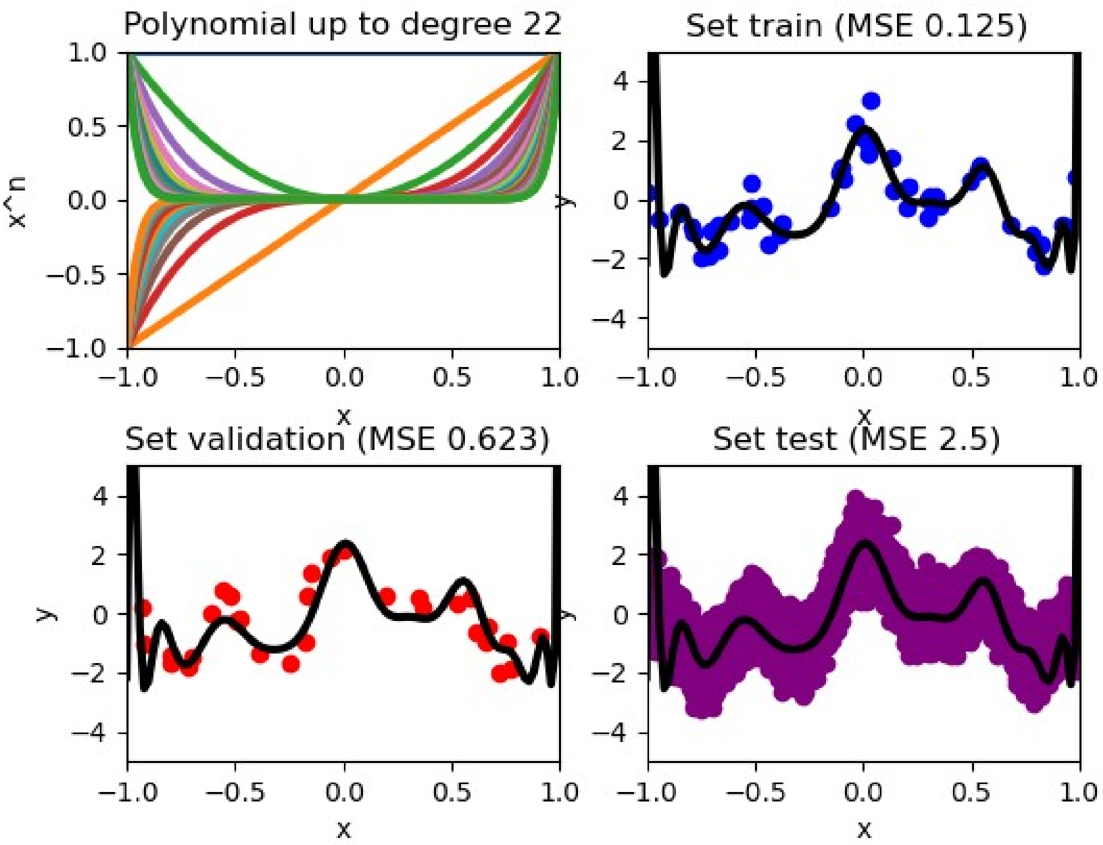
Python instruction: *Execute main_poly.py (Side effect: Figures below are stored on disk.)*





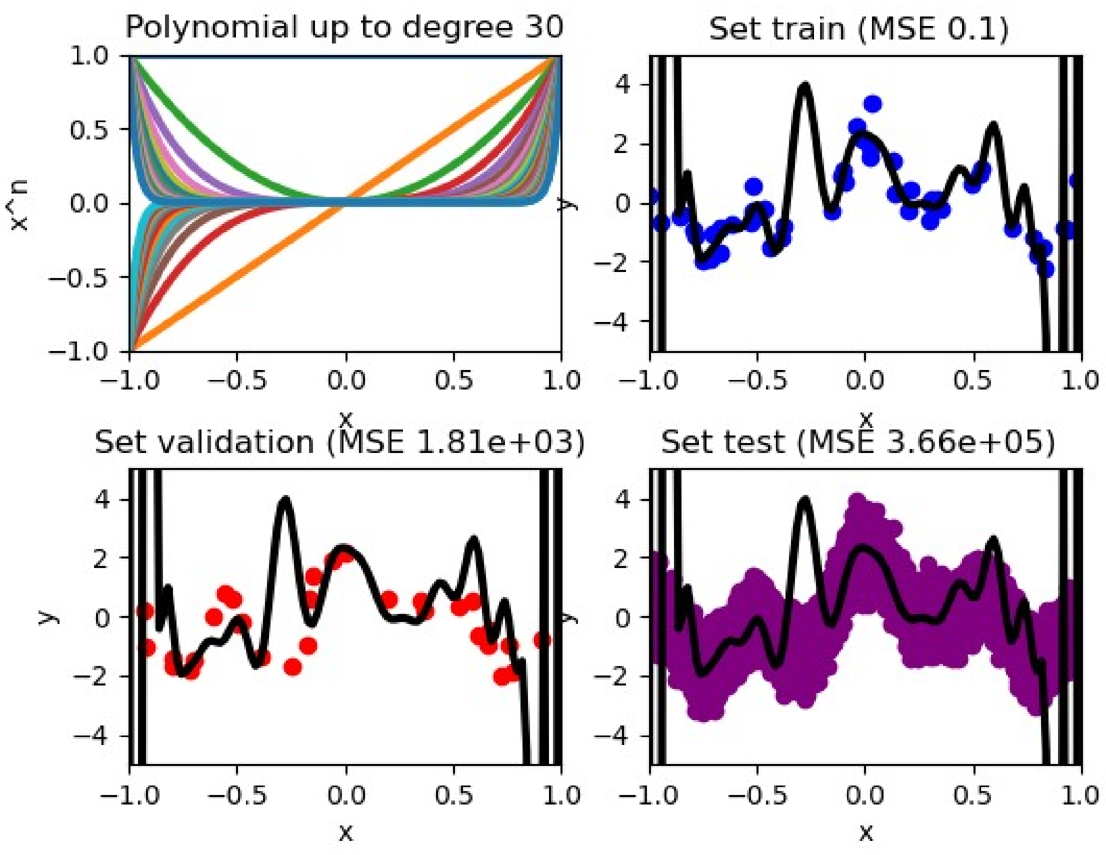


Polynomial up to degree 10
x^n
x
Set train (MSE 0.308)
y
x
Set validation (MSE 0.581)
y
x
Set test (MSE 0.433)
y
x



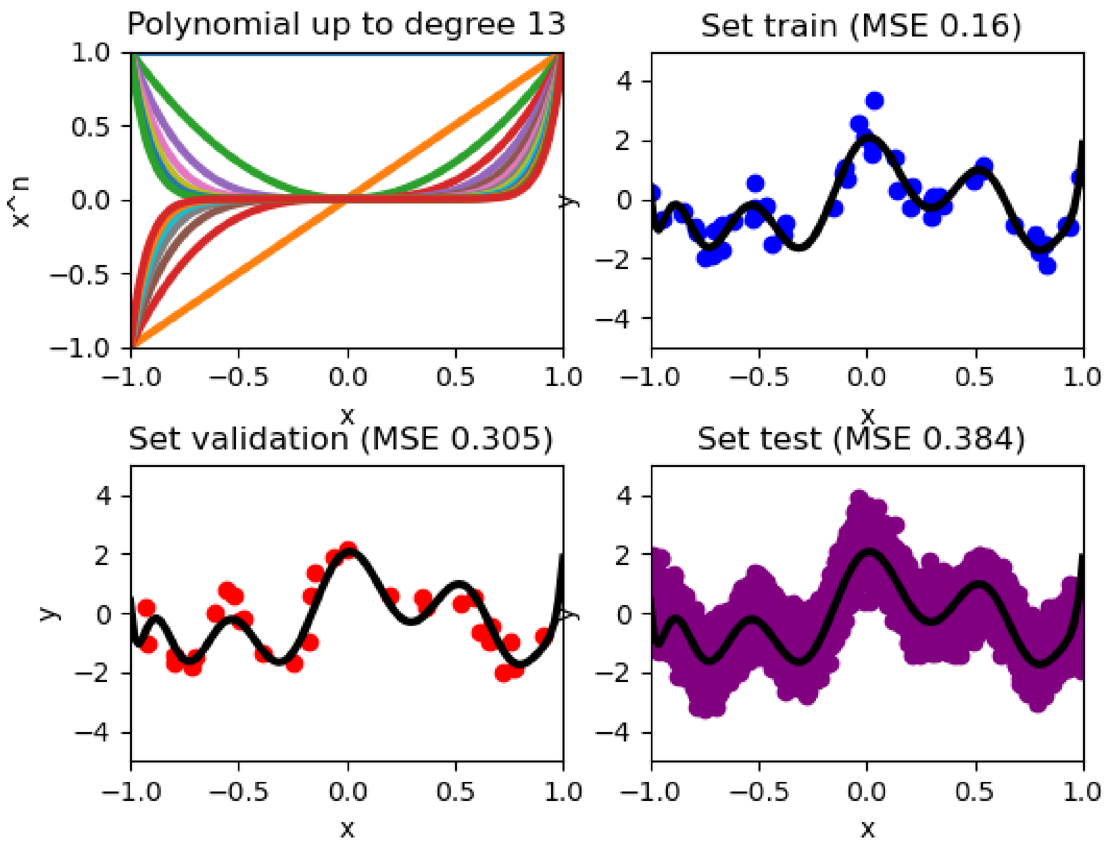
Polynomial up to degree 22
x^n
x
Set train (MSE 0.125)
y
x
Set validation (MSE 0.623)
y
x
Set test (MSE 2.5)
y
x

1.2.2. Report which degree ∈{1,...,30} gives the lowest cost function on the training set. Plot the results for that degree. Report the cost on the testing set.
Python instruction: *Execute main_poly_model_selection.py (No side effects.)*



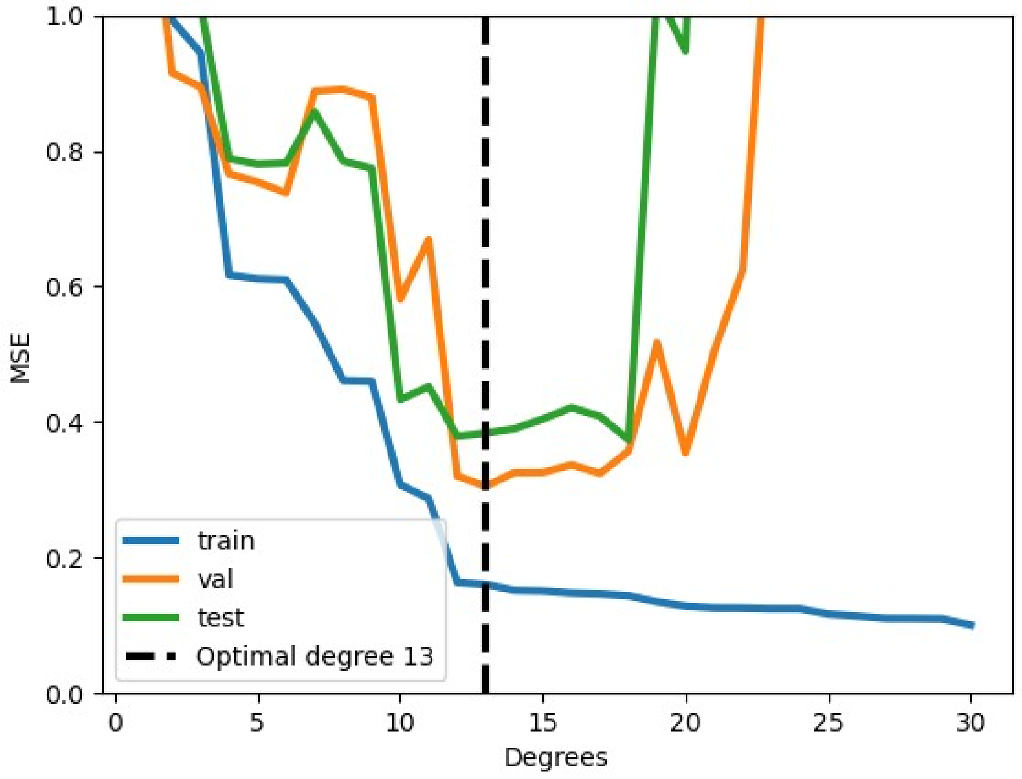
Best Degree on Training data set: 30
MSE training data set: 0.10026711102702786
MSE validation data set: 1806.112998730355
MSE test data set: 366166.96590486827

1.2.3. Report which degree ∈{1,...,30} gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost on the testing set.
Python instruction: *Execute main_poly_model_selection.py (No side effects.)*



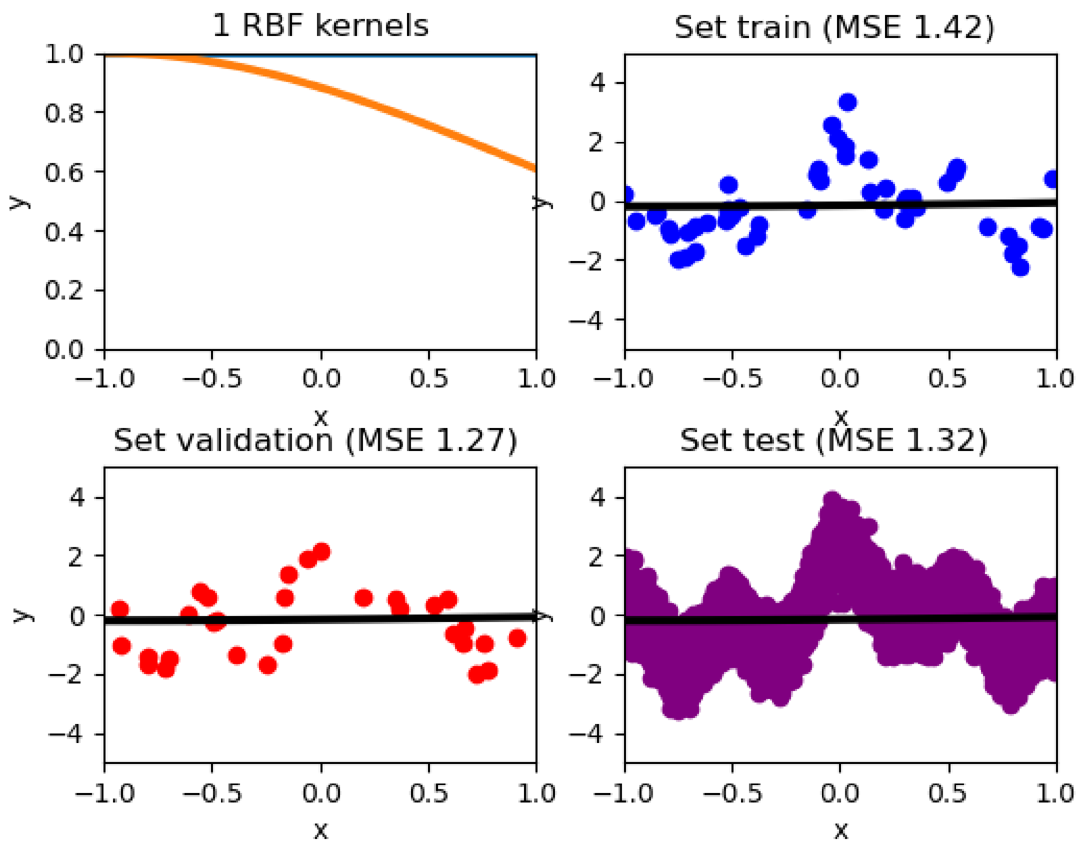
Best Degree on Validation data set: 13
MSE training data set: 0.16030297549817768
MSE validation data set: 0.3052286327052238
MSE test data set: 0.38370160465077324

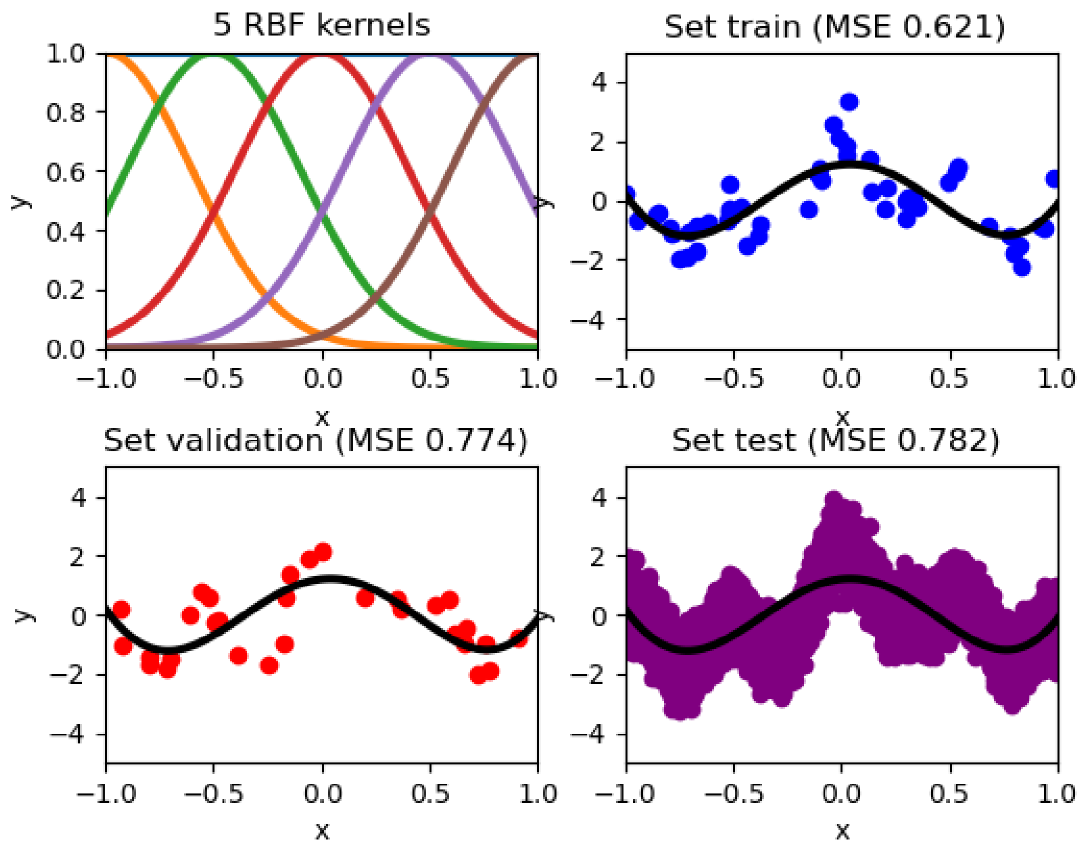
1.2.4. Plot training, validation and testing costs as a function of the polynomial degree using the provided function plot_errors in file plot_poly.py.
Python instruction: *Execute main_poly_model_selection.py (No side effects.)*

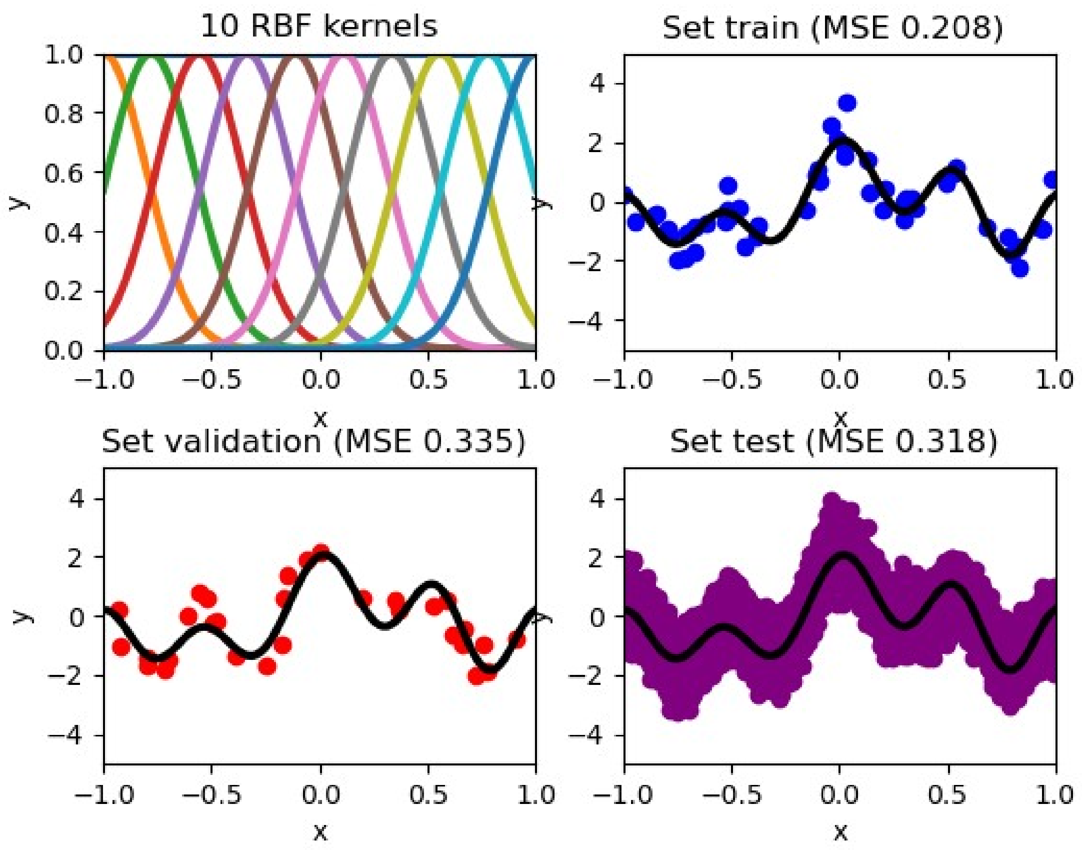


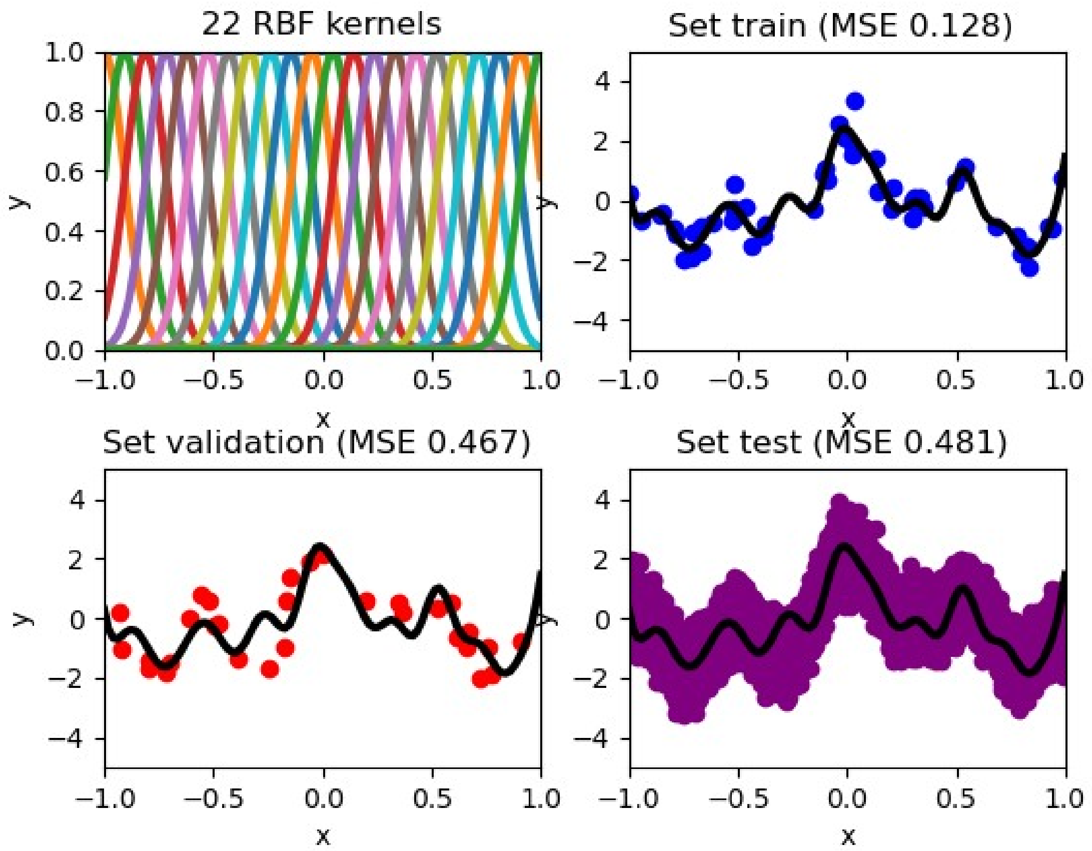
1.2.5. Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?
Weights of the linear regression model are adapted based on the training set. The higher the degree, the more data points of the training set will be hit exactly by the polynomial. But the training error is irrelevant. The model‘s quality is decided based on the error by the validation data set. The minimum error at a validation set determines the best model‘s parameters. The model‘s performance indicators are not derived from the validation data set but from the test data set! Error on the test data set is therefore not relevant too.

1.3.1. Using plot_rbf in main_rbf.py, plot your results for each of the following degrees: 1, 5, 10, 22
Python instruction: *Execute main_rbf.py (Side effects: Figures below are stored on disk.)*






10 RBF kernels
Set train (MSE 0.208)
Set validation (MSE 0.335)
Set test (MSE 0.318)
x
y

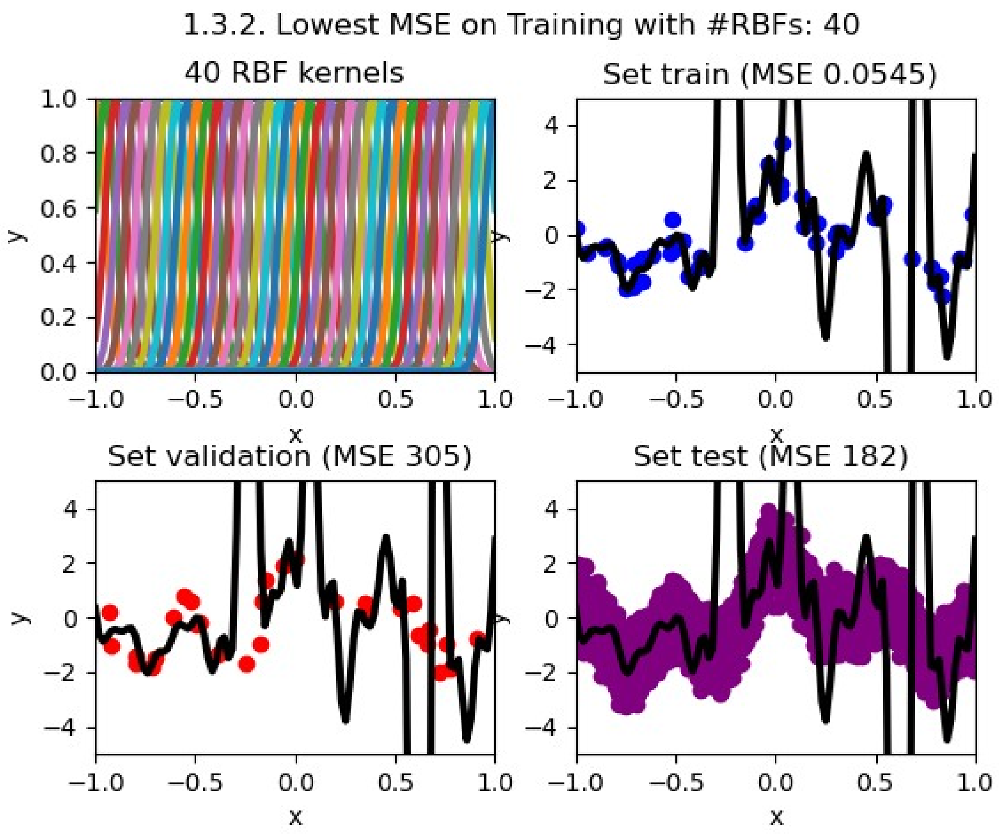

22 RBF kernels
Set train (MSE 0.128)
Set validation (MSE 0.467)
Set test (MSE 0.481)
x
y

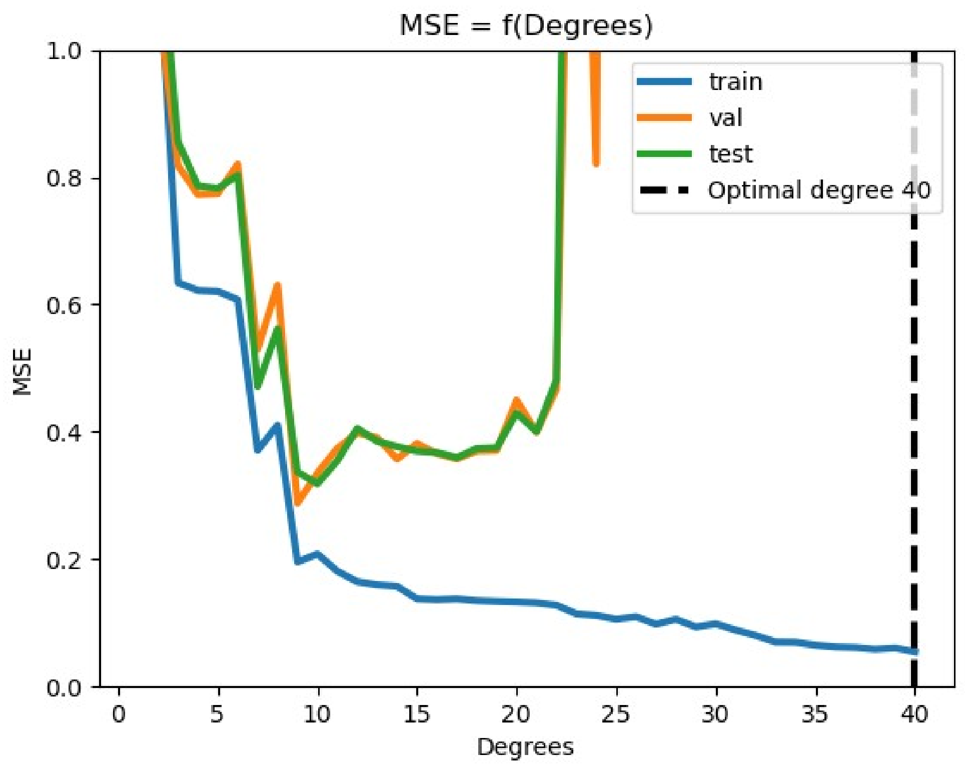
1.3.2 Report which number of RBFs ∈ {1,...,40} gives the lowest cost function on the training set. Plot the results for that degree. Report the cost function on the testing set.

Python instruction: *Execute main_rbf_model_selection.py (Side effects: Figures below are stored on disk. On the GUI only one figure is shown at a time deviating from the standard behaviour of showing two figures at the same time.)*

Best Number of RBFs on Training data set: 40 RBF kernels

MSE training data set:	0.05445291630105152
MSE validation set:	304.6665688638052
MSE test data set:	**181.67182614229617**

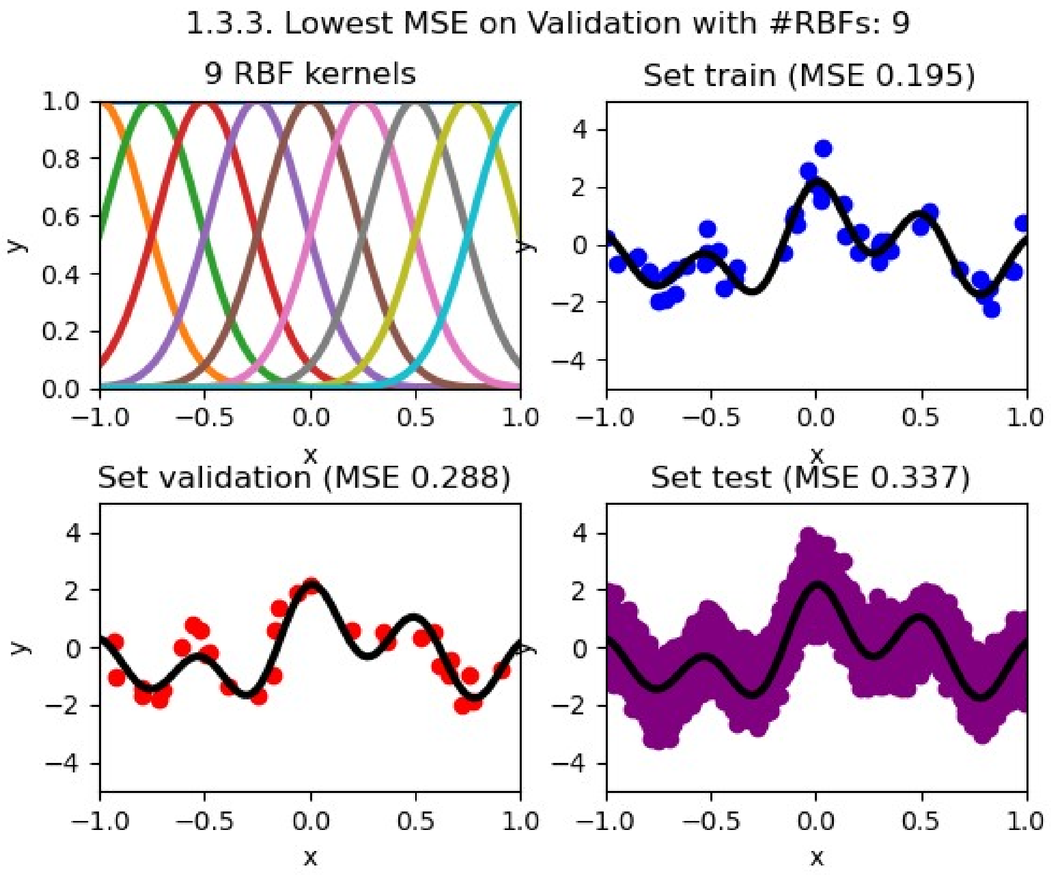


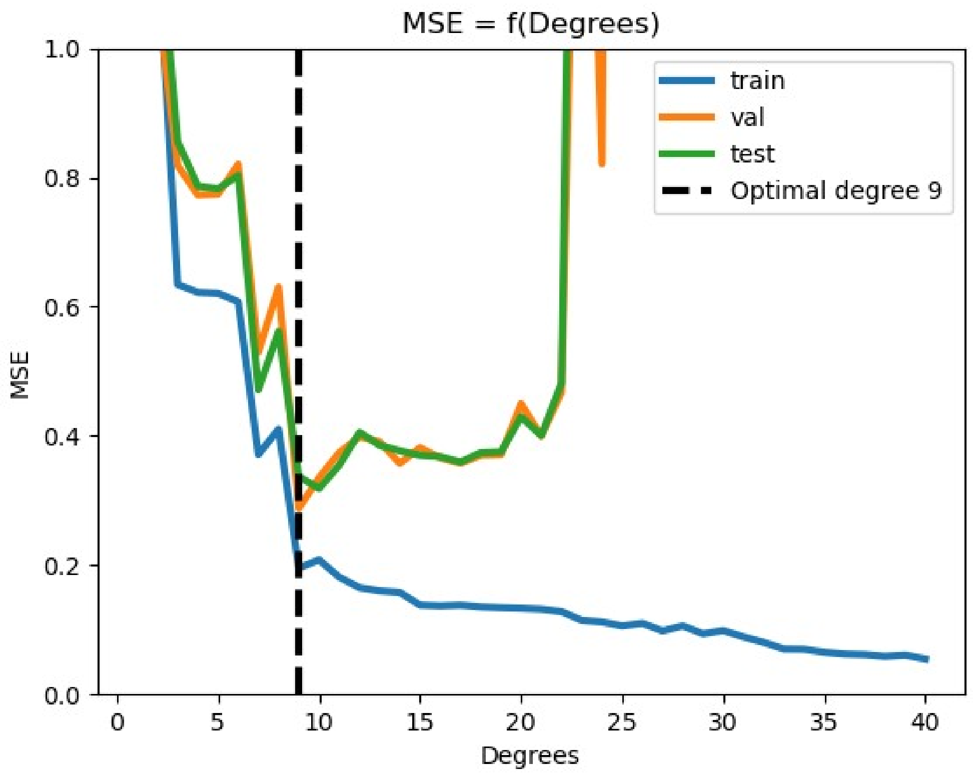


1.3.3 Report which number of RBFs ∈{1,...,40} gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost function on the test error.
Python instruction: *Execute main_rbf_model_selection.py (Side effects: Figures below are stored on disk. On the GUI only one figure is shown at a time deviating from the standard behaviour of showing two figures at the same time.)*

Best number of RBFs on Validation data set: 9 RBF kernels

```
MSE training data set:    0.19547366580986747
MSE validation data set:  0.28795984187213014
```
MSE test data set: 0.33701300824089886





1.3.4 Plot training, validation and testing costs as a function of the degree with your adaptation of plot_errors
Please take a look at last two error plots with title „MSE = f(Degrees)“

1.3.5 Briefly describe and discuss your findings in your own words. Is the polynomial or the RBF model better?

Polynomial: Minimum MSE on validation data set:	0.30522863270522 38
RBF: Minimum MSE on validation data set:	0.287959841872130 14

RBF outperforms Polynomial model in terms of minimum MSE. It depends on computational cost of 9 RBF kernels in comparison to polynomial model of degree 13. If both polynomial model and RBF model cost same in terms of required computational power, RBF should be used preferably.
I guess that some computation time can be saved at RBF by using less kernels. By this approach here more and more RBF kernels are centered at equidistant steps and are weighted differently. Maybe it is sufficient to use only those kernels with the highest weights, as these should be the most important ones. Additionally they could be centered exactly at the maxima on the x-axis. In order to determine those maxima, the polynomial model could be used and its maxima extracted. At these maxima the RBF kernels can be placed exactly and trained furthermore.

2.2.1. The function check_gradient in toolbox.py is here to test if your gradient is well computed. Explain what it is doing.

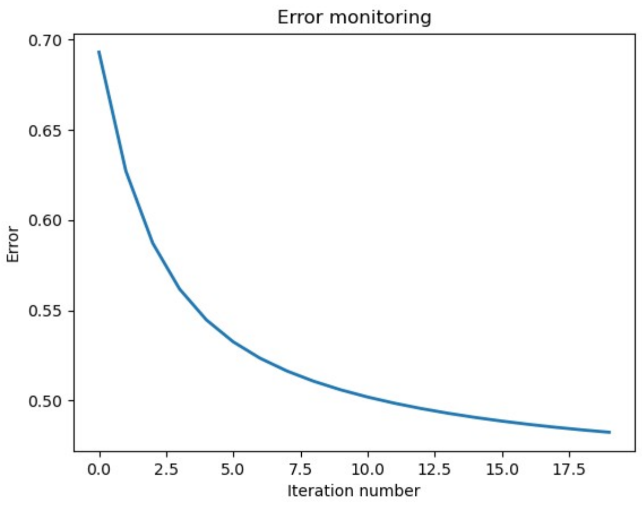
'check_gradient' compares "untrusted", externally computed gradient with "trusted" internally computed gradient. At first, a random point x0 (=theta here) and a random direction dx are chosen. A reference "trusted" gradient is computed by evaluation of the discrete derivation formula of (f(x0+dx)-f(x0))/dx.
The external "untrusted" gradient computation function is called for x0. This gradient vector is projected into direction of dx (by inner product) in order to make this value comparable to the "trusted" gradient value.
If the difference between "trusted" and "untrusted" gradient value deviates less than 1e-20 from the external gradient, everything is fine, otherwise an error message is output.
A bit more documentation has been added to the python code.

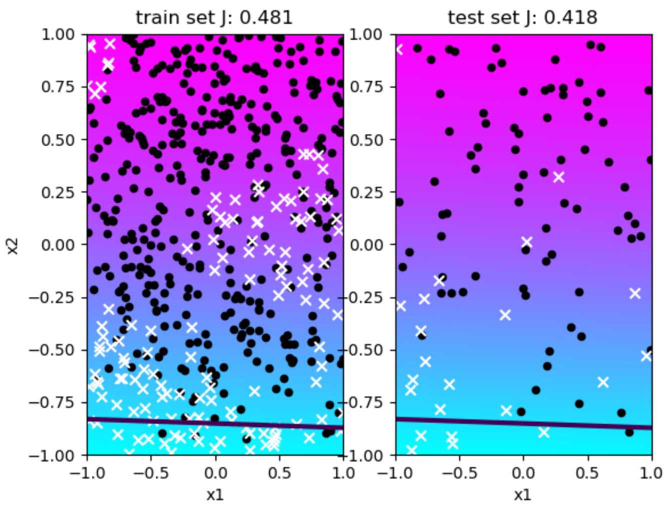
2.2.2 For degree l = 1 run GD for 20 and 2000 iterations (learning rate η = 1, all three parameters initialized at zero). Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain why the number of iterations should be neither too low nor too high

Python instruction: *Execute main_logreg.py (Side effects: Figures below are stored on disk. On the GUI no figure is shown! Function main() performs tasks 2.2.2 – 2.2.4 depending on configuration string „active_task“. Please set this string accordingly.)*

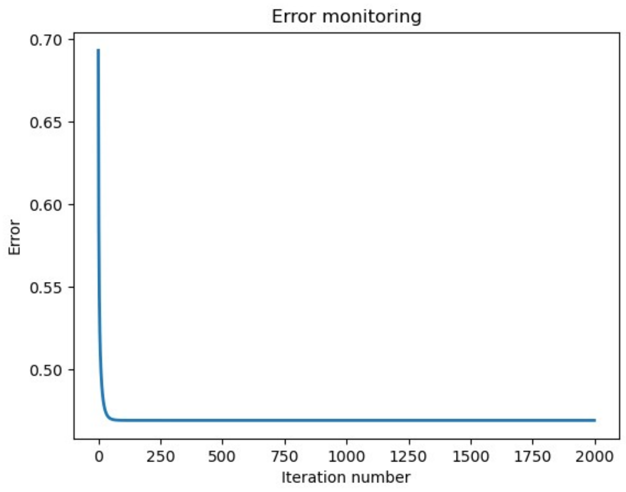
Training data set

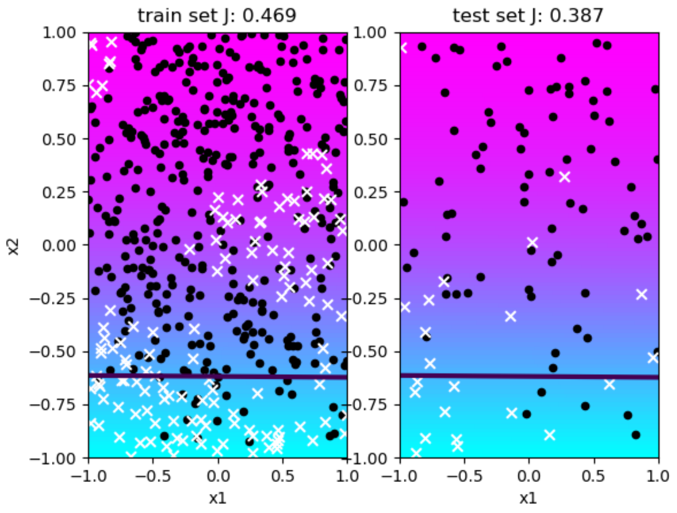
l=1, iter=20, error=0.4823504808597247





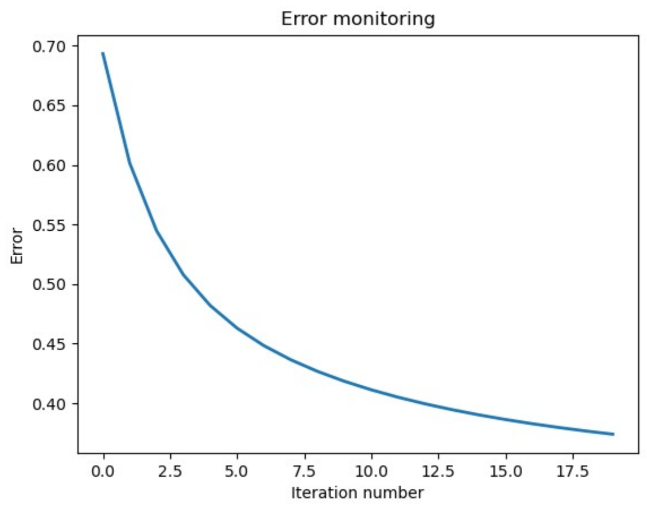
l=1, iter=2000, error=0.4691831520377954

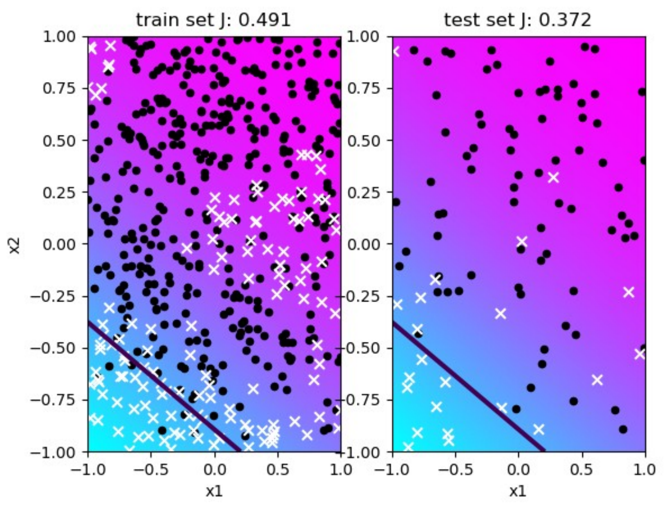




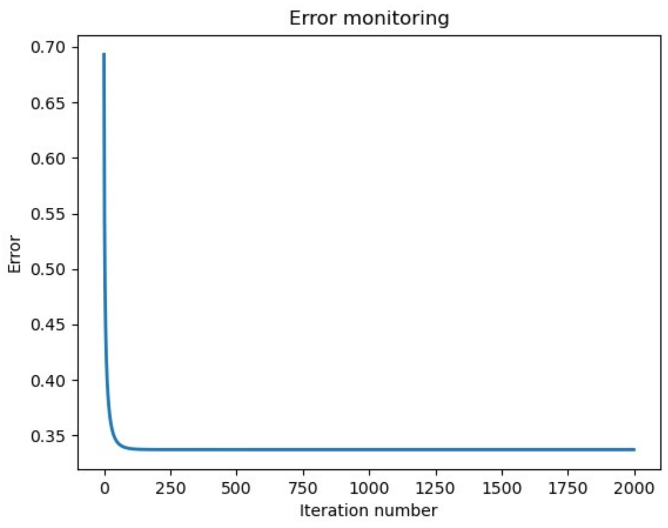
Test data set

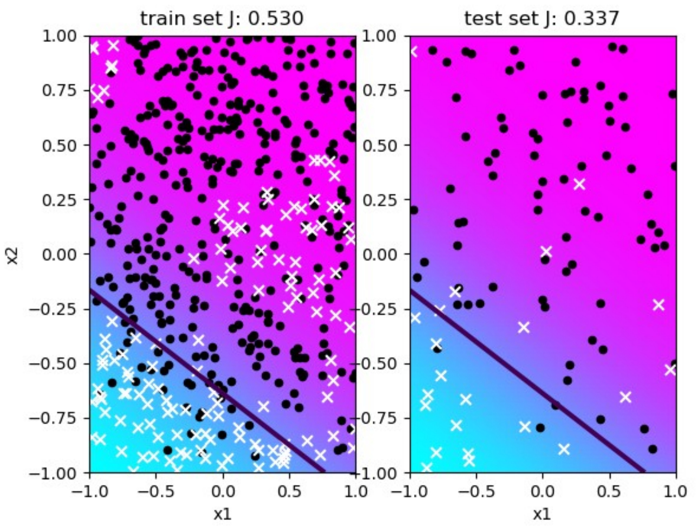
l=1, iter=20, error=0.37411397816550596





l=1, iter=2000, error=0.3371896425935134





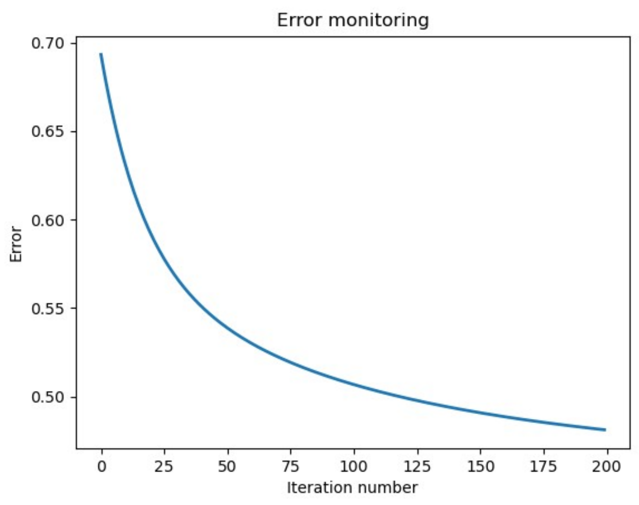
Number of iterations: If too low, classes will not be separated correctly (underfitting). If number of iterations is too high, error will not decrease any more and computation power is wasted.

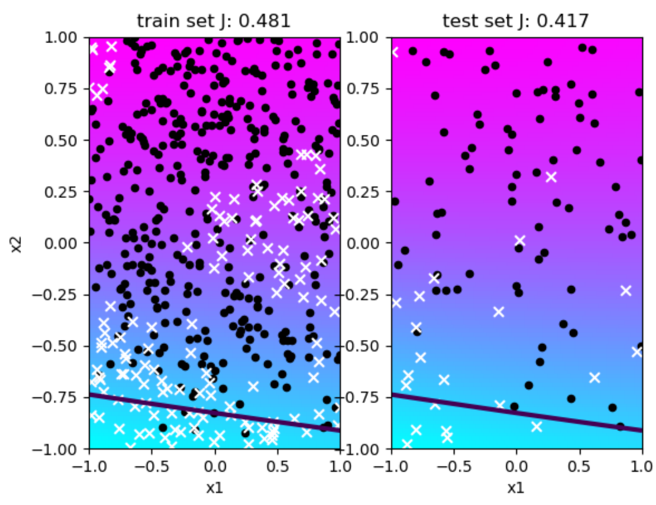
2.2.3 For degree l = 2 run GD for 200 iterations and learning rates of η = 0.1, η = 2 and η = 20. Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain what is happening when η is too large or too small.
Python instruction: *Execute main_logreg.py (Side effects: Figures below are stored on disk. On the GUI no figure is shown! Function main() performs tasks 2.2.2 – 2.2.4 depending on configuration string „active_task“. Please set this string accordingly.)*

In order to provide a compact overview on a single page, content is shifted to next page.

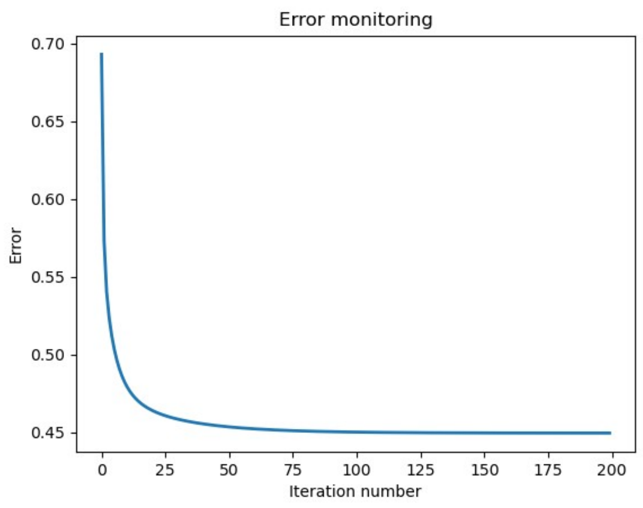
Training data set

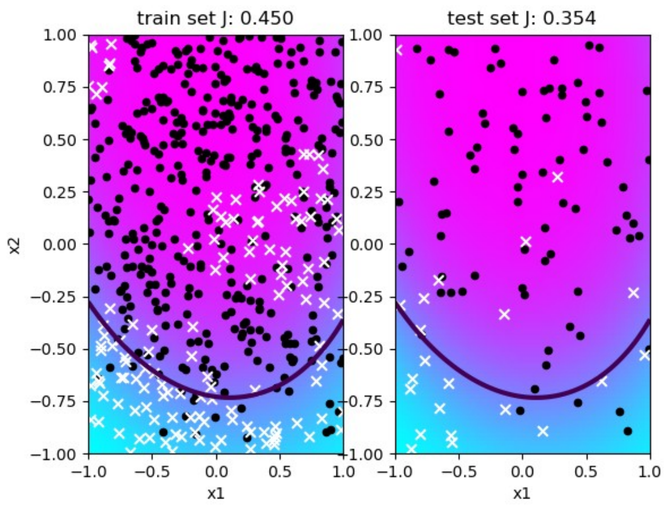
η=0.1, degree=2, iter=200, error=0.4811614505203645



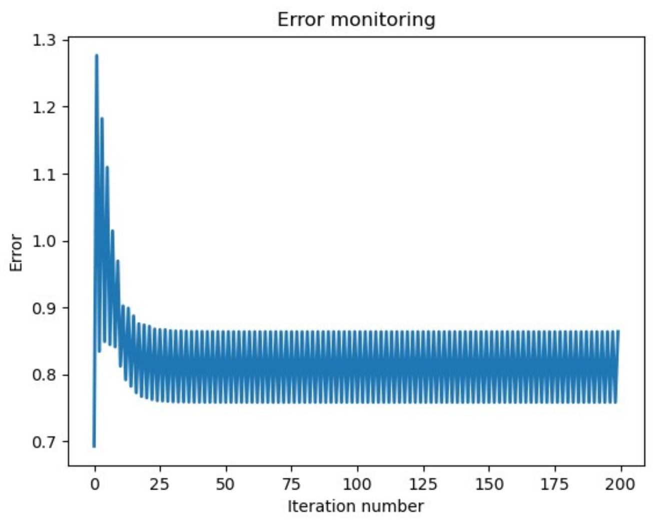


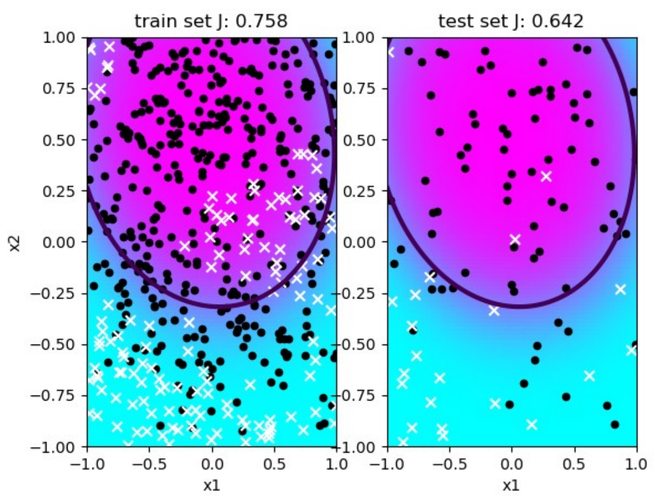
η=2, degree=2, iter=200, error=0.449676188182135 45





η=20, degree=2, iter=200, error=0.6931471805599454

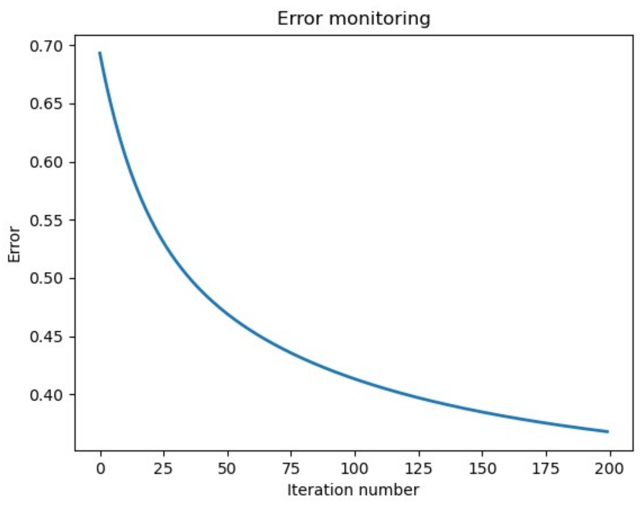


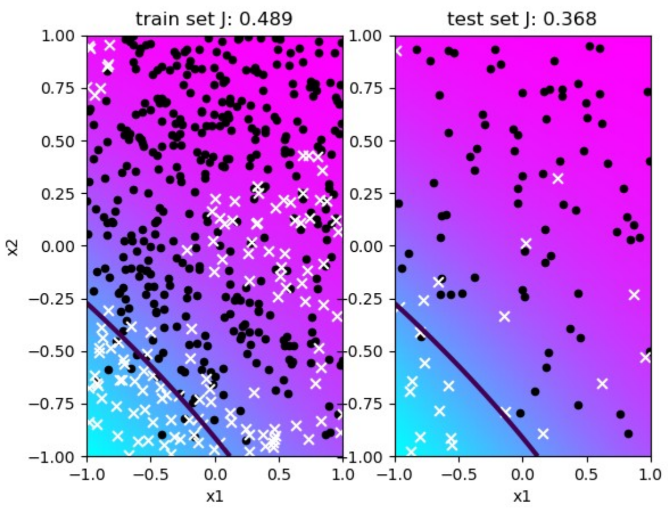


Note: Training set: For values above (η=20, degree=2, iter=200) the error=0.6931471805599454 differs from the error in title „train set J: 0.758“. After some debugging it turned out that cost computation in `plot_logreg` is based on a linspace meshgrid, whereas cost computation in the used framework is based on train/test data values.

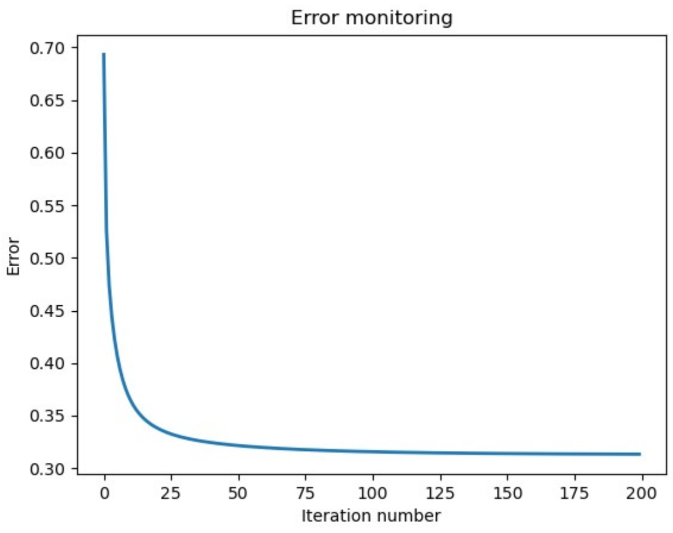
Test data set

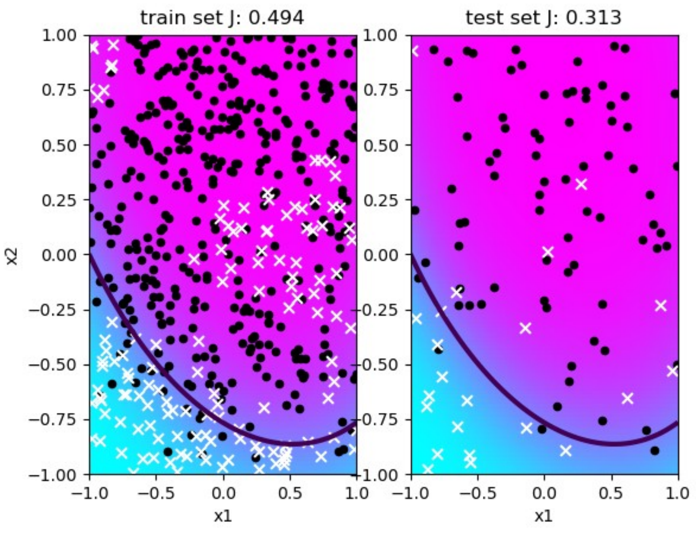
η=0.1, degree=2, iter=200, error=0.36787244582232326



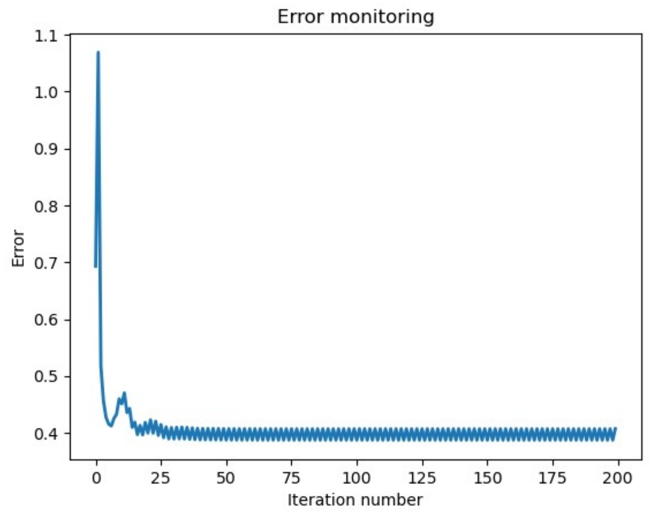


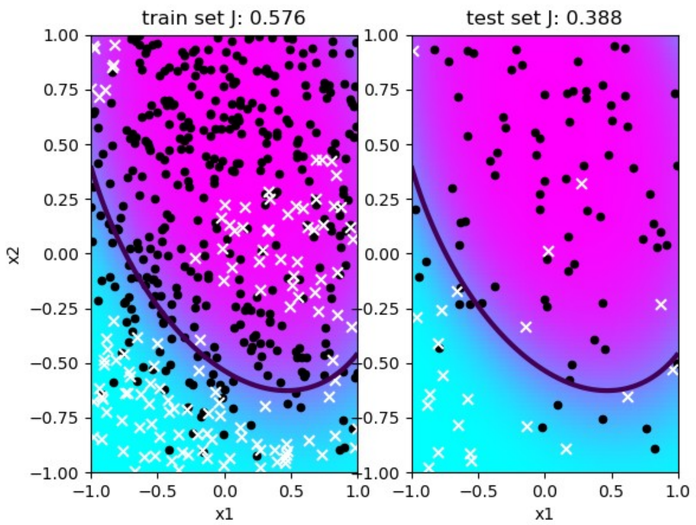
η=2,degree=2, iter=200, error=0.31333856269534466





η=20, degree=2, iter=200, error=0.3875239811079388





If η is too small, it will take a long time until a reasonable small error is achieved. Maybe training is cancelled before a good solution is found. But if classification training comes close to an optimal solution, it will be able to approach very close to it. If η is too big, classification training will go very fast to a solution after beginning, but classification error will finally oscillate around a good solution. Due to the large step size, a good solution can be „never“ achieved.

2.2.4 Identify reasonably good pairs of values for the number of iterations and learning rate for each degree in l ∈{1,2,7,20} that provides a good solution in reasonable time. Report these pairs, the performance obtained on training and testing set and plot each solution. Conclude by describing which degree seems the most approriate to fit the given data.

Python instruction: *Execute main_logreg.py (Side effects: Figures below are stored on disk. On the GUI no figure is shown! Function main() performs tasks 2.2.2 – 2.2.4 depending on configuration string „active_task“. Please set this string accordingly.)*

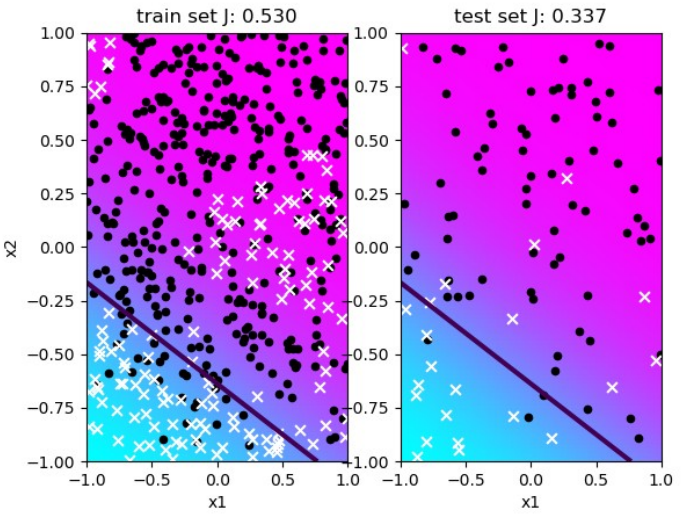
Description: I did a brute force search over a grid of parameters:

```
degree_list=[1,2,7,20]
max_iter_list=[50, 100, 150, 200, 250, 300, 350, 400, 600, 800, 900, 1000, 1100, 1200, 1300,
1400, 1500, 1600, 1700]
eta_list=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
```

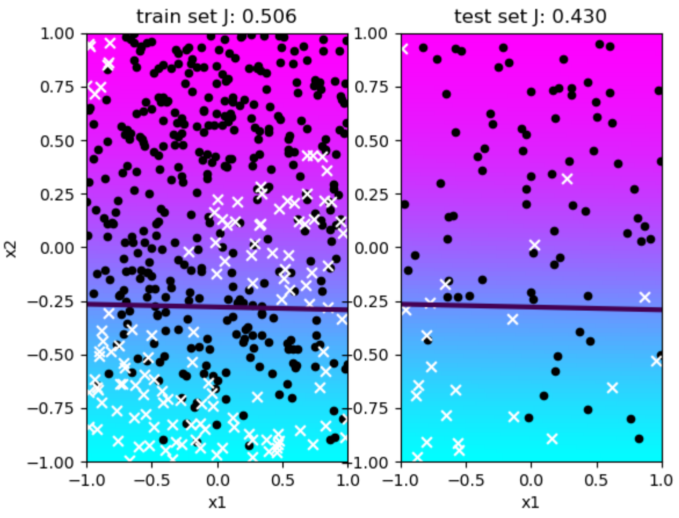
Each value of *degree_list* was combined with each element of *max_iter_list* with each element of *eta_list*. The „optimal“ pair was considered of being the one with the least error on the test set (error on training set was ignored) and with least computational requirements (least number of iterations, eta somewhere between 1 and 18):

Degree	**Iterations**	**Eta**	**Error Test data set**	**Error Training data set**
1	50	13	0.33718964259351564	0.5055197479823257

Test data set

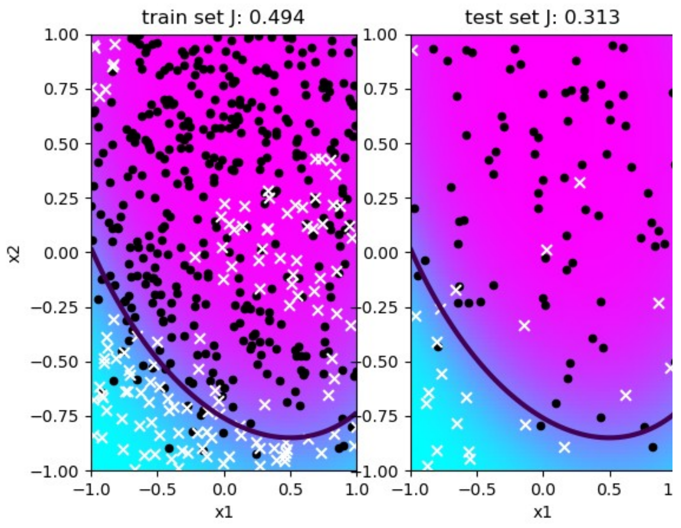


Training data set

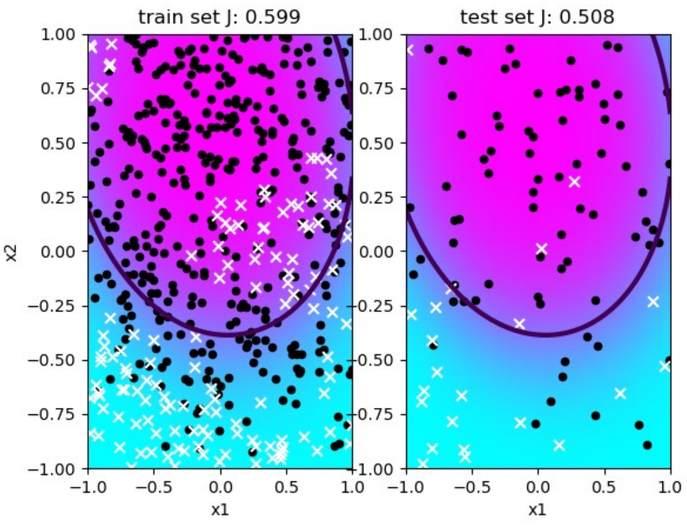


Degree	Iterations	Eta	Error Test data set	Error Training data set
2	50	14	0.31298915602071087	0.5657007971339023

Test data set

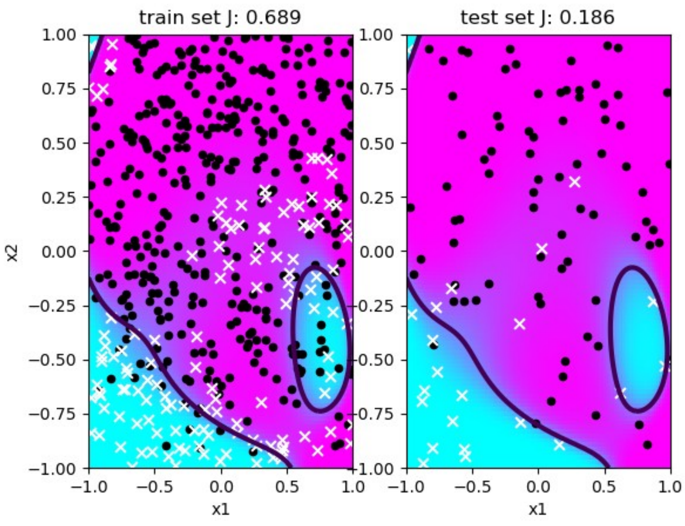


Training data set

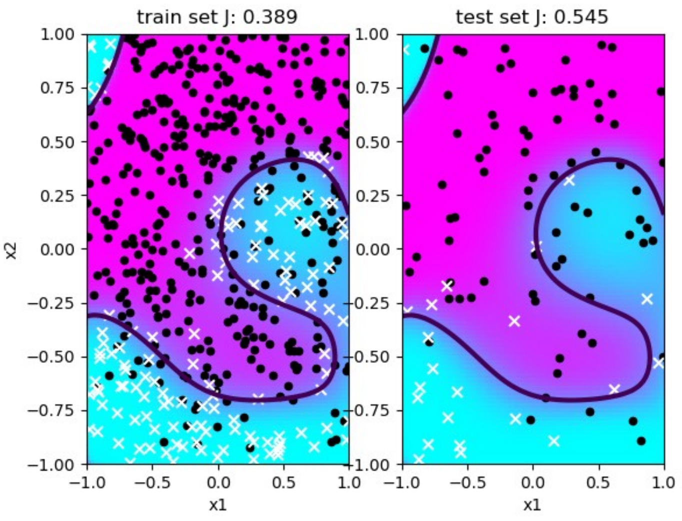


Degree	Iterations	Eta	Error Test data set	Error Training data set
7	1700	18	0.18600019323569	0.3885326491684223

Test data set

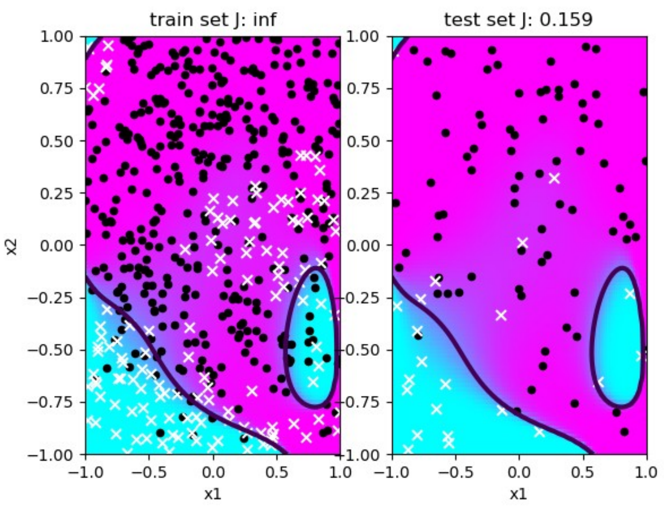


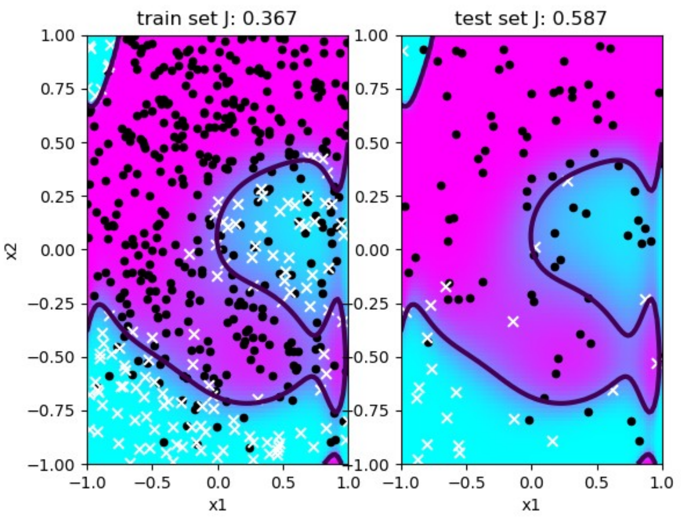
Training data set



Degree	Iterations	Eta	Error Test data set	Error Training data set
20	1700	18	0.1593203510272915	0.36732703005878237

Test data set **Training data set**





2.2.5. Describe a possible stopping criterium using the gradient of the cost function.
As a single criterion will never result in good solutions for all tasks, a set of different termination criterions is required:

- Maximum number of iterations
- Error value below an absolute threshold
- Relative change of error value from one step to the next below a threshold (smooth descends assumed): close to zero if error does not change much any more
- Mean of relative changes of error value below a threshold (a bit more robust against zig-zag)
- If error value starts to grow again: Perform N steps, if error does not become smaller, then algorithm is stuck in local minima: terminate
- If gradient‘s vector length becomes smaller than a certain threshold, no distinct direction exists any more. Maybe stuck in a local minima.
- Gradient‘s vector length evaluation over time: If it becomes shorter and shorter, terminate, if change in length becomes smaller than a threshold.
- Gradient‘s vector orientation: If gradient vector turns fully around within N iterations (and error does not change much), probably it is stuck in a local minima.