Assignment 2

Computational Intelligence, SS2020

Team Members		
Last name	First name	Matriculation Number
Reindl	Hannes	01532129
Samer	Philip	01634718
Rösel	David	01634719

Contents

Computational Intelligence SS20
Homework 2
Linear and Logistic Regression

Ceca Kraisnikovic
Horst Petschenig

Tutor:	Flora Feldner, flora.feldner@student.tugraz.at
Points to achieve:	20 pts
Bonus points:	2* pts
Info hour:	05.05.2020 Cisco WebEx Meeting, see TeachCenter
Deadline:	12.05.2020 23:55
Hand-in procedure:	Use the **cover sheet** to be found on the teach center. Submit your **python files and a colored report** on the teach center.
Course info:	TeachCenter

Contents

General remarks

Your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed)
- The quality of your plots (Is everything clearly visible in the print-out? Are axes labeled? . . .)
- Your submission should run with Python 3.5+

1 Linear Regression

1.1 Derivation of Regularized Linear Regression [5 points]

Given the design matrix $\boldsymbol{X}$ (m rows, $n+1$ columns), output vector $\boldsymbol{y}$ (m rows) and parameter vector $\boldsymbol{\theta}$ ($n+1$ rows), the mean squared error cost function for linear regression can be written compactly as,

$$J(\boldsymbol{\theta}) = \frac{1}{m}\|\boldsymbol{X\theta} - \boldsymbol{y}\|^2 \ . \tag{1}$$

The notation $\|\cdot\|$ refers to the Euclidean norm. The optimal parameters which minimize this cost function are given by,

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2}$$

Consider the following slightly extended, "regularized" cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\|\boldsymbol{X\theta} - \boldsymbol{y}\|^2 + \frac{\lambda}{m}\|\boldsymbol{\theta}\|^2 \tag{3}$$

The additional term penalizes parameters of large magnitudes. The constant $\lambda \geq 0$ controls the influence of this penalization (low $\lambda \rightarrow$ weak regularization, high $\lambda \rightarrow$ strong regularization). Such regularized cost functions are often used for linear regression (and other learning models) instead of (1) because regularization drastically reduces the effects of over-fitting that occurs when the number of features is large.
Answer the following questions:

1. Preliminary questions,
   (a) Why is the design matrix $\boldsymbol{X}$ containing $n+1$ columns and not just n?
   (b) Considering the function $J(\boldsymbol{\theta})$ as a function of all the variables $\theta_0, \theta_1 \ldots$, give one definition of the gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$? This gradient is actually a vector, what is its dimension?
   (c) What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix?
   (d) The hypothesis $\boldsymbol{X\theta}$ generates predictions of the output vector $\boldsymbol{y}$, each coordinate of the vector corresponds to a different data point. We also use the notation $\frac{\partial \boldsymbol{X\theta}}{\partial \boldsymbol{\theta}}$ to define the Jacobian matrix of the vector $\boldsymbol{X\theta}$. What is the dimension of the Jacobian matrix $\frac{\partial \boldsymbol{X\theta}}{\partial \boldsymbol{\theta}}$ and what is it equal to?
2. Using the hints below or as done in the practical 4, show that the optimal parameters which minimize the regularized linear regression cost (3) are given by,

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \ , \tag{4}$$

   where $\boldsymbol{I}$ denotes the identity matrix of dimension $(n+1)\times(n+1)$.

Some hints for question 1.1.2:

1. **Hint 1** There are three ways to compute the gradient of a function $f(\theta)$. (1) compute each component individually $\frac{\partial f}{\partial \theta_j}$, (2) use the matrix notation and definition of the gradient as the unique vector v that verifies $f(\theta + d\theta) - f(\theta) = v^T d\theta + o(d\theta)$ or (3) get used to manipulating the derivative of matrix operators directly with care; the wikipedia page on matrix calculus contains a useful summary of rules regarding matrix/vector derivatives (numerator layout).
2. **Hint 2** This is the derivation of the solution of the Linear Regression without regularization when using matrix operators:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{m}\frac{\partial(\boldsymbol{X\theta} - \boldsymbol{y})^T(\boldsymbol{X\theta} - \boldsymbol{y})}{\partial \boldsymbol{\theta}} \tag{5}$$

$$= \frac{2}{m}(\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot \frac{\partial(\boldsymbol{X\theta} - \boldsymbol{y})}{\partial \boldsymbol{\theta}} \tag{6}$$

$$= \frac{2}{m}(\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot \boldsymbol{X} \tag{7}$$

When the cost function is minimized the gradient of the cost function is zero:

$$\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}) = \mathbf{0} \tag{8}$$

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^T\boldsymbol{y} = \mathbf{0} \tag{9}$$

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta} = \boldsymbol{X}^T\boldsymbol{y} \tag{10}$$

$$\theta = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{11}$$

As long as $\boldsymbol{X}^T\boldsymbol{X}$ is invertible the solution θ exists and is unique.

1.2 Linear Regression with polynomial features [6 points]

For questions 1.2 and 1.3 download the associated zip file from the course website and unzip it. The file `data_linreg.json` contains training, validation and test sets of a regression problem with a single input dimension.

Your task is to fit polynomials of different degrees to the training data, and to identify the degree which gives the best predictions on the validation set (performing simple model selection). The features that should be used for linear regression are (x corresponds to the scalar input, and n is the polynomial degree)

$$\phi_0 = 1, \phi_1 = x, \phi_2 = x^2, \ldots, \phi_n = x^n, \tag{12}$$

Fill in the TODO boxes in the following places:

- The functions `design_matrix`, `train` and `compute_errors` in the file `poly.py`
- In `main_poly_model_selection.py`. Your code should find the degree of the polynomial that minimizes the cost function in either the training or validation set. You can re-use the code provided above.

The following should be included in your report:

1. Using `plot_poly` in `main_poly.py`, plot your results for each of the following degrees: 1, 5, 10, 22.
2. Report which degree $\in \{1, \ldots, 30\}$ gives the lowest cost function on the training set. Plot the results for that degree. Report the cost on the testing set.
3. Report which degree $\in \{1, \ldots, 30\}$ gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost on the testing set.
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`.
5. Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?

1.3 (Bonus) Linear Regression with radial basis functions [2* points]

Using the same data set, your task is to fit hypotheses with different numbers of radial basis functions to the training data, and to perform model selection. Let l be the number of RBFs. For a given l, the RBF centers should be chosen to uniformly span the whole input range $[-1, 1]$. The width of the basis functions should be set to $\sigma = 2/l$, i.e. with a higher l, the RBFs should be narrower (implementing a finer resolution). The features used for linear regression should be (x corresponds to the scalar input and c_j is the center of RBF j):

$$\phi_0 = 1,\ \phi_1 = e^{\frac{-(x-c_1)^2}{2\sigma^2}},\ \ldots\ ,\ \phi_l = e^{\frac{-(x-c_l)^2}{2\sigma^2}} \tag{13}$$

Fill in the TODO boxes in the following places::

- The functions `get_centers_and_sigma`, `design_matrix`, `train` and `compute_errors` in the file `rbf.py`.
- In `main_rbf_model_selection.py`, to get training/validation/test costs for each $l \in \{1, 2, \ldots, 39, 40\}$. You can re-use the code provided in `main_poly_model_selection.py`.
- In the function `plot_errors` in the file `plot_rbf.py`. This makes a plot with the number of RBFs l on the horizontal axis and the training/validation/test costs on the vertical axis. For each of the three cost values, plot a separate line in a different colour. You can base your implementation on the equivalent function in the file `plot_poly.py`.

The following should be included in your report:

1. Using `plot_rbf` in `main_rbf.py`, plot your results for each of the following degrees: 1, 5, 10, 22.
2. Report which number of RBFs $\in \{1, \ldots, 40\}$ gives the lowest cost function on the training set. Plot the results for that degree. Report the cost function on the testing set.
3. Report which number of RBFs $\in \{1, \ldots, 40\}$ gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost function on the test error.
4. Plot training, validation and testing costs as a function of the degree with your adaptation of `plot_errors`.
5. Briefly describe and discuss your findings in your own words. Is the polynomial or the RBF model better?

2 Logistic Regression

2.1 Derivation of Gradient [3 points]

The logistic regression hypothesis function is given by, $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{\theta}) = \sigma(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n)$ with the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$, parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^T$, and input vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)^T$. By convention the constant feature x_0 is fixed to $x_0 = 1$. With $\boldsymbol{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \ldots, x_n^{(i)})^T$, $x_0^{(i)} = 1$ and $\log(\cdot)$ referring to the natural logarithm, the logistic regression cost function can be written as,

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) + (1-y^{(i)})\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right) \tag{14}$$

Answer the following question:

1. Derive the gradient of the cost function, i.e. show that the partial derivative of the cost function with respect to θ_j equals

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)} \quad . \tag{15}$$

Hint Note that the derivative of the sigmoid function σ verifies $\frac{\partial\sigma(z)}{\partial z} = \sigma(z)\cdot(1-\sigma(z))$.

2.2 Logistic Regression training with gradient descent [6 points]

Download the associated zip file from the course website and unzip it. The file `data_logreg.json` contains training and test set of a binary classification problem with two input dimensions. Your task is to fit a logistic regression classifier with polynomial features of varying degrees to the data. The expansion of the input is already implemented. The features that should be used for logistic regression are all monomials

of the two inputs x_1 and x_2 up to some degree l (monomials are polynomials with only one term), i.e. all products $(x_1)^a(x_2)^b$ with $a, b \in \mathbb{Z}$ and $a + b \leq l$. For example, for degree $l = 3$ the features should be,

$$\phi_0 = 1, \tag{16}$$

$$\phi_1 = (x_1)^1,\ \phi_2 = (x_2)^1, \tag{17}$$

$$\phi_3 = (x_1)^2,\ \phi_4 = (x_1)(x_2),\ \phi_5 = (x_2)^2, \tag{18}$$

$$\phi_6 = (x_1)^3,\ \phi_7 = (x_1)^2(x_2)^1,\ \phi_8 = (x_1)^1(x_2)^2,\ \phi_9 = (x_2)^3 \tag{19}$$

Fill the following blocks in the code you are provided:

- In `logreg.py` implement the cost funtion of logistic regression as the python function `cost`
- In `logreg.py` implement the gradient of the cost with respect to theta as the python function `grad`
- In `gradient_descent.py` implement a generic gradient descent solver

Your report should include the following:

1. The function `check_gradient` in `toolbox.py` is here to test if your gradient is well computed. Explain what it is doing.
2. For degree $l = 1$ run GD for 20 and 2000 iterations (learning rate $\eta = 1$, 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.
3. For degree $l = 2$ run GD for 200 iterations and learning rates of $\eta = 0.1$, $\eta = 2$ and $\eta = 20$. 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.
4. Identify reasonably good pairs of values for the number of iterations and learning rate for each degree in $l \in \{1, 2, 7, 20\}$ 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.
5. Describe a possible stopping criterium using the gradient of the cost function.

1 Linear Regression

1.1 Derivation of Regularized Linear Regression

1. Preliminary questions

   (a) Why is the design matrix $\mathbf{X}$ containing $n+1$ columns and not just n?
   n stands for the highest degree of the polynomial function. This results in n+1 features since there is one for every polynomial term starting at n=0. NOTE: m is the amount of datapoints.

   (b) Considering the function $J(\theta)$ as a function of all the variables θ_0, θ_1 ..., give one definition of the gradient $\frac{\partial J(\theta)}{\partial \theta}$? This gradient is actually a vector, what is its dimension?
   To get the dimension of $J(\theta)$ we can just look at what dimension θ has. The following equations show that:

$$
\begin{aligned}
dim(X) &= m \times n+1 \\
dim(y) &= m \times 1 \quad dim(\theta) = n+1 \times 1 \\
\frac{\partial J(\theta)}{\partial \theta} &= \frac{2}{m}(X\theta - y)^T \cdot \mathbf{X} + \frac{2\lambda}{m}\theta^T \cdot \mathbf{I} \\
dim(\frac{\partial J(\theta)}{\partial \theta}) &= dim(\frac{2}{m}(X\theta - y)^T \cdot \mathbf{X} + \frac{2\lambda}{m}\theta^T) \cdot \mathbf{I} \\
&= ((m \times n+1) \cdot (n+1 \times 1))^T \cdot (m \times n+1) + (1 \times n+1) \cdot (n+1 \times n+1) \\
&= (m \times 1)^T \cdot (m \times n+1) + (1 \times n+1) \\
&= (1 \times m) \cdot (m \times n+1) + (1 \times n+1) \\
&= (1 \times n+1)
\end{aligned}
$$

$$
\begin{aligned}
J(\theta) &= \frac{1}{m}\|X\theta - \mathbf{y}\|^2 + \frac{\lambda}{m}\|\theta\|^2 \\
\frac{\partial J(\theta)}{\partial \theta} &= \frac{2}{m}(X\theta - y)^T \cdot \frac{\partial (X\theta - y)}{\partial \theta} + \frac{2\lambda}{m}\theta^T \\
\frac{\partial J(\theta)}{\partial \theta} &= \frac{2}{m}(X\theta - y)^T \cdot \mathbf{X} + \frac{2\lambda}{m}\theta^T
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \|\theta\|^2}{\partial \theta} &= \frac{\partial (\theta^T \cdot \theta)}{\partial \theta} = \frac{\partial (\theta_0^2 + \cdots + \theta_{n+1}^2)}{\partial \theta} \\
&= \begin{pmatrix} 2\theta_0 & \cdots & 2\theta_{n+1} \end{pmatrix} \\
&= 2 \cdot \theta^T
\end{aligned}
$$

   (c) What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix?

The definition of the Jakobi Matrix is as follows. The function f is in this case a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.

$$Jf(x_1,...,x_n) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

The difference between the Jacobimatrix and the gradient is basicly very slim since one can argue that the gradient is basically the Jacobimatrix for m=1. So basicly the difference is that the Jacobimatrix is used if there are more than just a single function given given. In the case above representet by the m equations f1...fm.

(d) The hypothesis $\mathbf{X}\ \theta$ generates predictions of the output vector $\mathbf{y}$, each coordinate of the vector corresponds to a different data point. We also use the notation $\frac{\partial X\theta}{\partial \theta}$ to de
ne the Jacobian matrix of the vector $X\theta$. What is the dimension of the Jacobian matrix $\frac{\partial X\theta}{\partial \theta}$ and what is it equal to?

In the following equasion the $\mathbf{X}$ matrix can be taken as a multiplicative konstant, since it doesnt depend on theta. Therefor the only part of the equasion which is relevant is what results in $\frac{\partial \theta}{\partial \theta}$.

$$\theta = \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_{n+1} \end{pmatrix}$$

$$\frac{\partial \theta}{\partial \theta} = \begin{pmatrix} \frac{\partial \theta_0}{\partial \theta_0} & \frac{\partial \theta_0}{\partial \theta_1} & \cdots & \frac{\partial \theta_0}{\partial \theta_{n+1}} \\ \frac{\partial \theta_1}{\partial \theta_0} & \frac{\partial \theta_1}{\partial \theta_1} & \cdots & \frac{\partial \theta_1}{\partial \theta_{n+1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \theta_{n+1}}{\partial \theta_0} & \cdots & \cdots & \frac{\partial \theta_{n+1}}{\partial \theta_{n+1}} \end{pmatrix}$$

$$\frac{\partial \theta}{\partial \theta} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 1 \end{pmatrix} = \mathbf{I}$$

As one can see the result is unit matrix with dimensions of $n+1 \times n+1$. If we multiply it with the matrix $\mathbf{X}$ from before we just get the matrix $\mathbf{X}$ with the same dimensions as before $(m \times n+1)$.

2. Using the hints below or as done in the practical 4, show that the optimal parameters which minimize the regularized cost(3) are given by,

$$\theta^* = (X^T X + \lambda I)^{-1} X^T y, \tag{1}$$

where $\mathbf{I}$ denotes the identity matrix of dimension $(n+1)x(n+1)$.

$$
\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \frac{2}{m}(X\theta - y)^T \cdot \mathbf{X} + \lambda \theta^T \stackrel{!}{=} \mathbf{0}^T \\
((X\theta - y)^T \cdot \mathbf{X} + \frac{2\lambda}{m}\theta^T &= \mathbf{0} \\
\mathbf{X}^T(X\theta - y) + \lambda\theta &= \mathbf{0} \\
\mathbf{X}^T X\theta - \mathbf{X}^T\mathbf{y} + \lambda\theta &= \mathbf{0} \\
(\mathbf{X}^T \cdot \mathbf{X} + \lambda\mathbf{I})\theta &= \mathbf{X}^T\mathbf{y} \\
\theta &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1} \cdot \mathbf{X}^T\mathbf{y}
\end{aligned}
$$

1.2 Linear Regression with polynomial features

In this sub task a program should be written which fits a polynomial function onto some given data set. The degree of the polynomial which fits the data the best should be evaluated as well as the impact of over- and under-modelling.

1.2.1 Results for different given degrees

The following figures show the results of polynomials for degrees [1, 5, 10, 22]

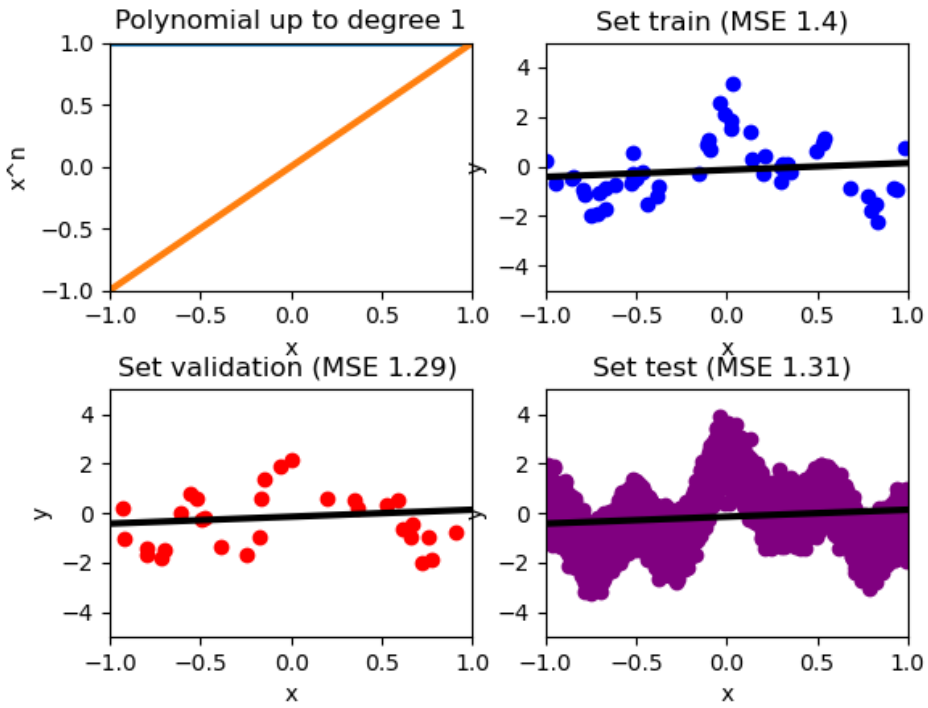

Figure 1: Fitted polynomial (*black*) of degree 1

The top left plot displays each polynomial term. The other plots displays the fitted black curve with different data sets.
These first set is called 'train' and is subset of the 'test'. This data set is used to in the Least-Squares-Algorithm to calculate the weights. The polynomial is the same for all three plots. The second set is called validation and its use is to validate the process of fitting a polynomial. Using again the training data set to validate for the best degree raises some issues, which will be discussed later. The last set contains all given samples and simply shows how to polynomial compares to the entire data.
Figure 1 shows the fitted curve for a polynomial with degree 1. A polynomial of degree 1 is just a linear function,

$$f(x) = w_0 + w_1 \cdot x = k \cdot x + d \tag{2}$$

which means there cant be any curvature. Hence it poorly resembles the true nature of the given data. The problem is under-modelled. The Mean-Squared-Error is also much bigger compared to the following figures.

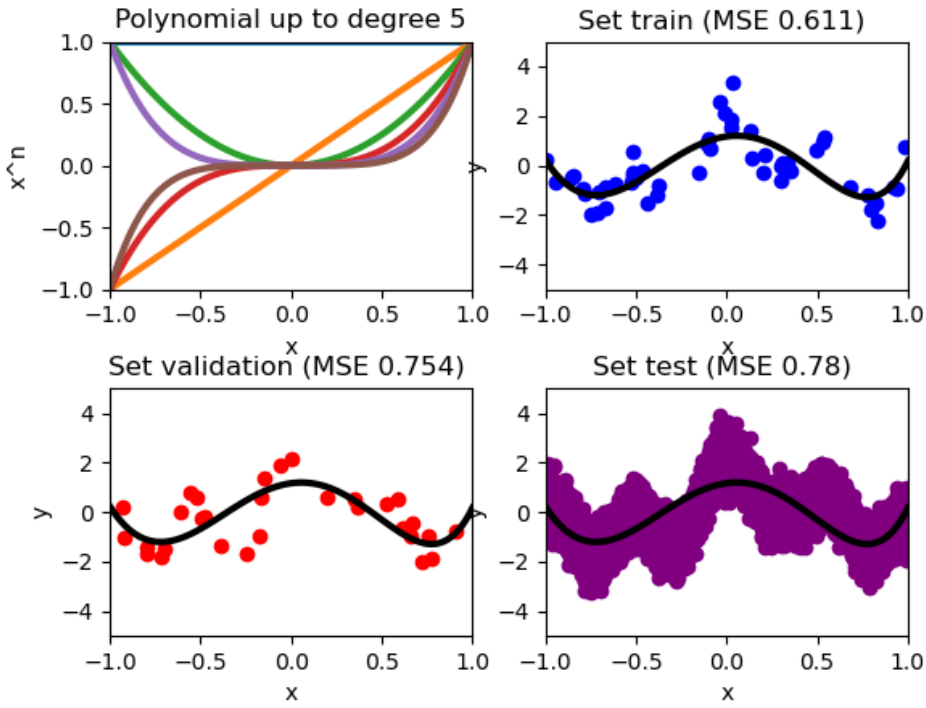

Figure 2: Fitted polynomial (*black*) of degree 5

By increasing the degree, we allow for a better modelling of the data set. The general form of a polynomial with degree 5 follows

$$f(x) = w_0 + w_1 \cdot x + w_2 \cdot x^2 + w_3 \cdot x^3 + w_4 \cdot x^4 + w_5 \cdot x^5 \tag{3}$$

with weight coefficients w_i. The higher order features allow for curvature and the fitted curve suits the data samples better now. But if we take a closer look on the test set plot (bottom right), it appears the true polynomial has more 'ups and downs' than our polynomial. It is still under-modelled and degree needs to be increased.

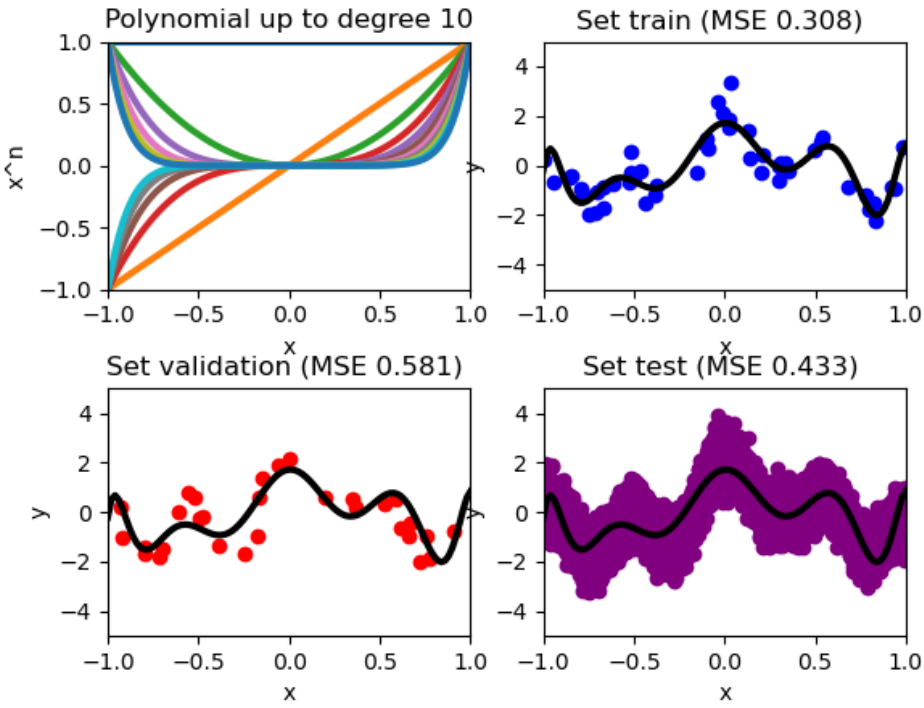

Figure 3: Fitted polynomial (*black*) of degree 10

Pumping up the degree even further shows, that our polynomial now can follow the frequent changes in the test set. But the peaks still appear to be a little bit lower than they should be, meaning the problem is still under-modelled. Evaluating the Mean-Squared-Error for the validation set, revealed that MSE reaches its minimum for a degree of 13, which means that the nature of data probably follows a polynomial with a degree of 13.

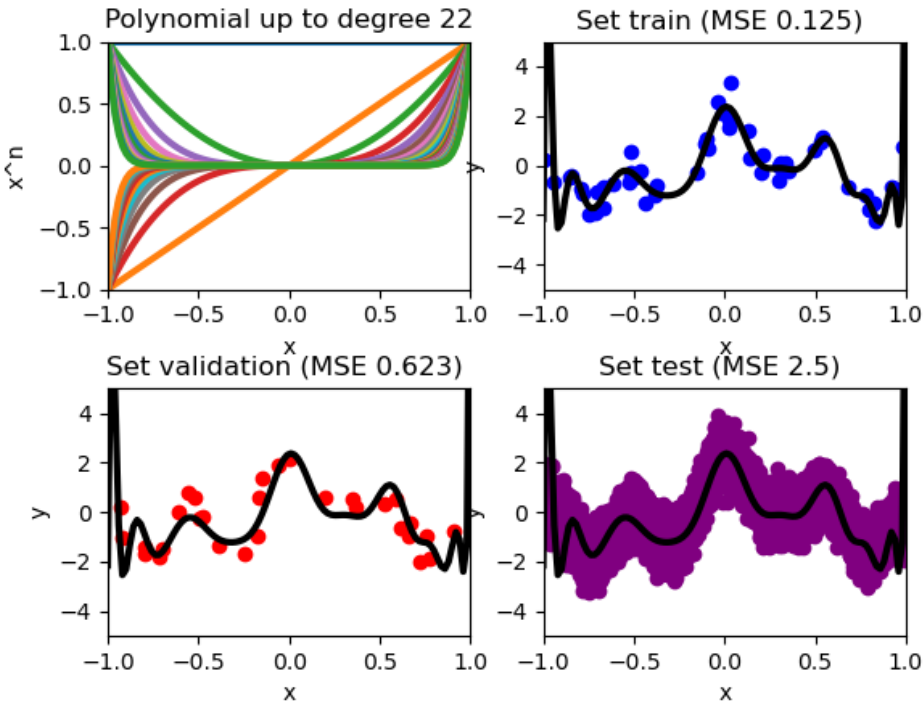

Figure 4: Fitted polynomial (*black*) of degree 22

Pumping up the degree too much causes oscillation near the edges and a worse MSE. Since the problem is over-modelled now, the Least-Squares-Algorithm does not 'look' on the overall trend anymore and tries to fit the curve to each individual point causing the fit to get worse.

1.2.2 Degree for lowest MSE on training set

In this subtask the degree which causes the minimum MSE using only the training set should be computed.

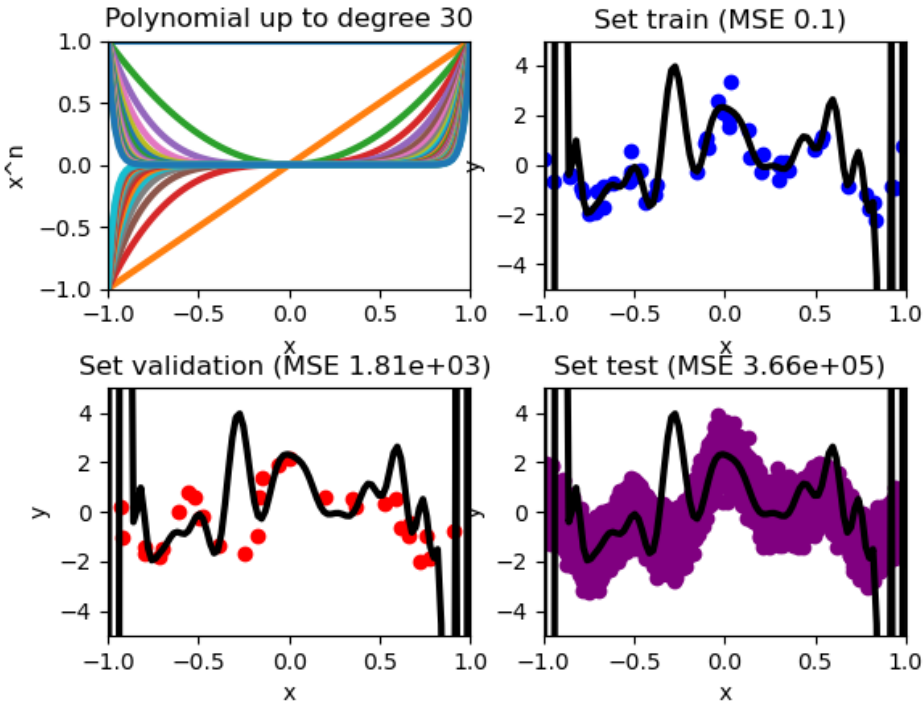

Figure 5: Fitted polynomial (*black*) of degree 13

The suggested degree i.e. the degree with the lowest MSE for the training set is 30. On top of each plot is the MSE for the set displayed. For the training set the Mean-Squared-Error with MSE = 0.1 is quite low. This is not the case if we take look at the validation or testing set.

Using only the training set, the suggested degree would increase until it could map every point of the training set onto the polynomial line, since this decreases the MSE. The only reason why the suggested degree is only 30, is because that's the highest degree which was allowed in the code. If another set was used instead to calculate the MSE (validation set) this problem would not occur.

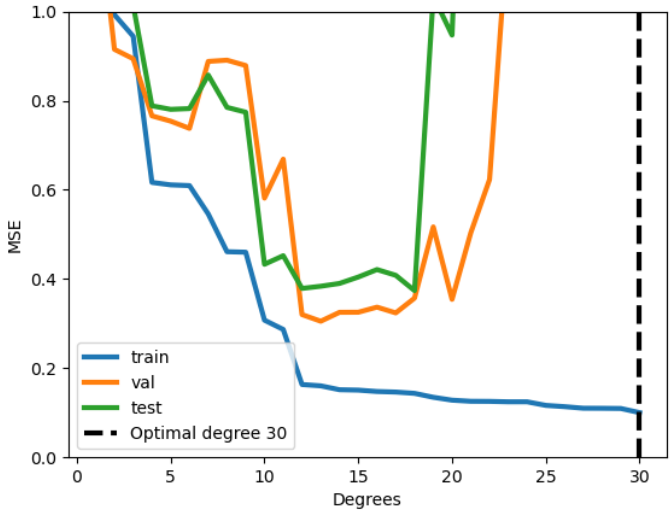

Figure 6: Mean Squared Error for different data sets

Figure 6 shows the change of the MSE depending on the polynomial degree for each data set. As one can see, the training set (*blue*) line keeps decreasing with increasing degrees, but the validation (*orange*) and testing set (*green*) had their minimum at lower degrees and kept increasing after that. Using the MSE of the training set is a bad metric to calculate the degree which fits the data the best.

1.2.3 Degree for lowest MSE on validation set

As stated before using the the same training data to evaluate the degree which fits the overall data the best causes some issues. This time the MSE to validate the result is another data set, but still drawn from the same overall data set. Written down mathematically, where χ denotes the overall data set.

$$\chi_{train} \subset \chi$$
$$\chi_{val} \subset \chi$$
$$\chi_{train} \cap \chi_{val} = \{\ \}$$

χ_{train} and χ_{val} are subsets of χ but they don't share any data samples. By doing this the calculated MSE wont decrease anymore, if the problem is already over-modelled.

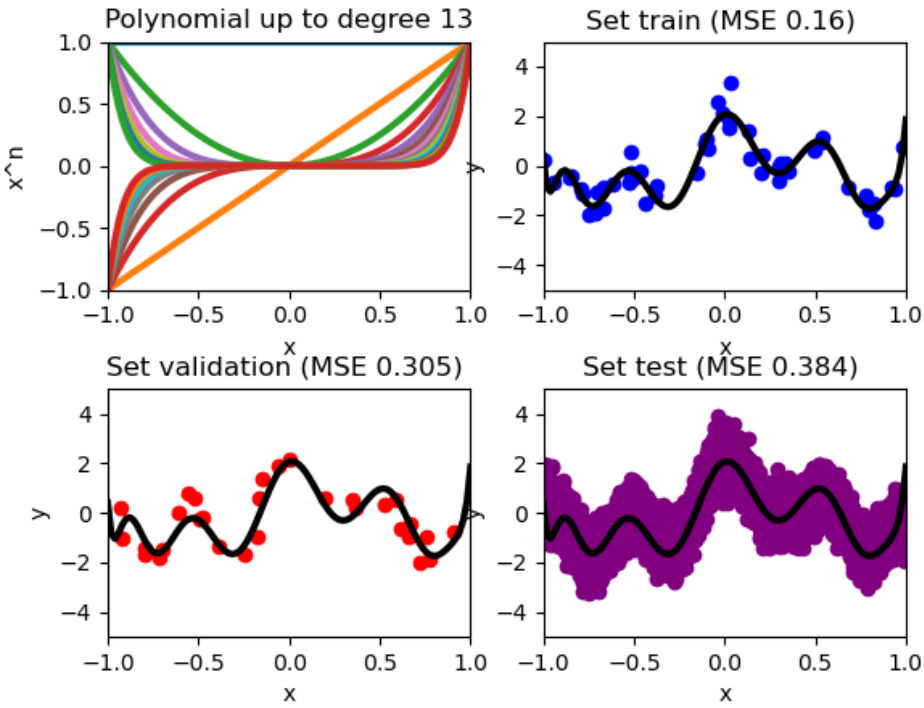

Figure 7: Fitted polynomial (*black*) of degree 13

Figure 7 shows the new fitted polynomial with MSE reaching its minimum at a degree of 13. The found polynomial follows the test set really well now.

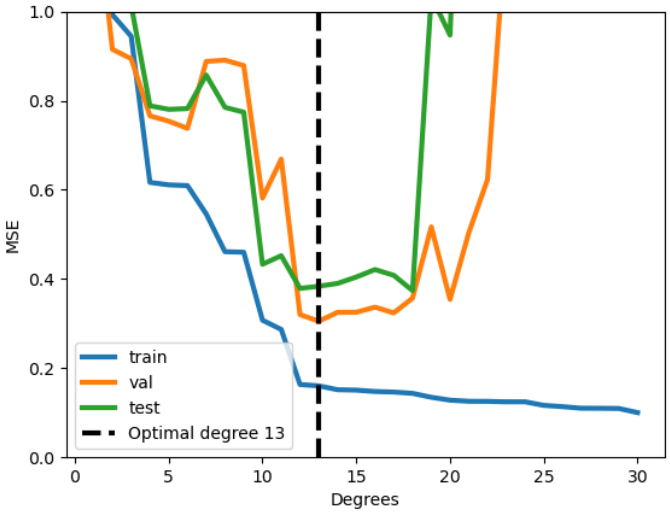

Figure 8: Mean Squared Error for different data sets

Figure 8 shows again the change of the MSE for the different sets. Since the MSE of the validation is used this time, the optimal degree settles to 13.

1.3 (Bonus) Linear Regression with radial basis functions

1.3.1 Results for different amount of Radial Basis Functions (RBF)

Since most of the plots were already described in section 1.2, I only discuss the changes. The biggest difference is the change of the basis function. Before a polynomial basis function with changing degree was used. This time a radial basis function in the form of

$$\Phi_i = e^{-\frac{(x-c_i)^2}{2\sigma^2}} \tag{4}$$

is used, where i denotes the control variable ('Laufvariable'), c_i is a shift constant and σ scales the width of the kernel and is calculated by

$$\sigma = \frac{2}{l} \tag{5}$$

where l denotes the amount of RBFs with $i \, \varepsilon \, [1, l]$. The zeroth element Φ_i is always 1.

$$\Phi_0 = 1 \tag{6}$$

The following figures shows the plots for the values $l = [1, 5, 10, 22]$

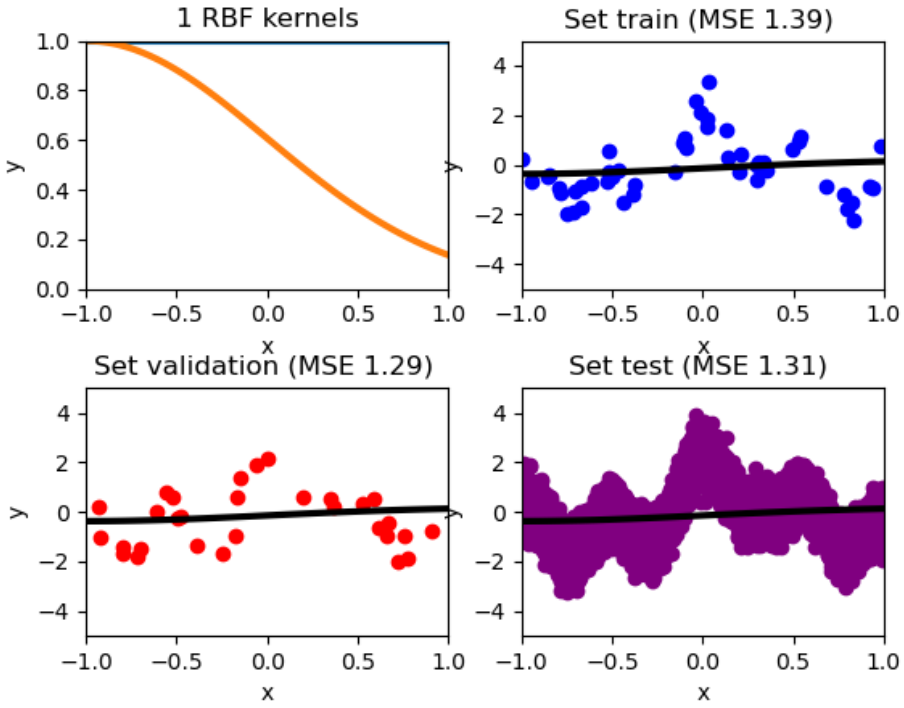

Figure 9: Fitted RBF (*black*) for l =1

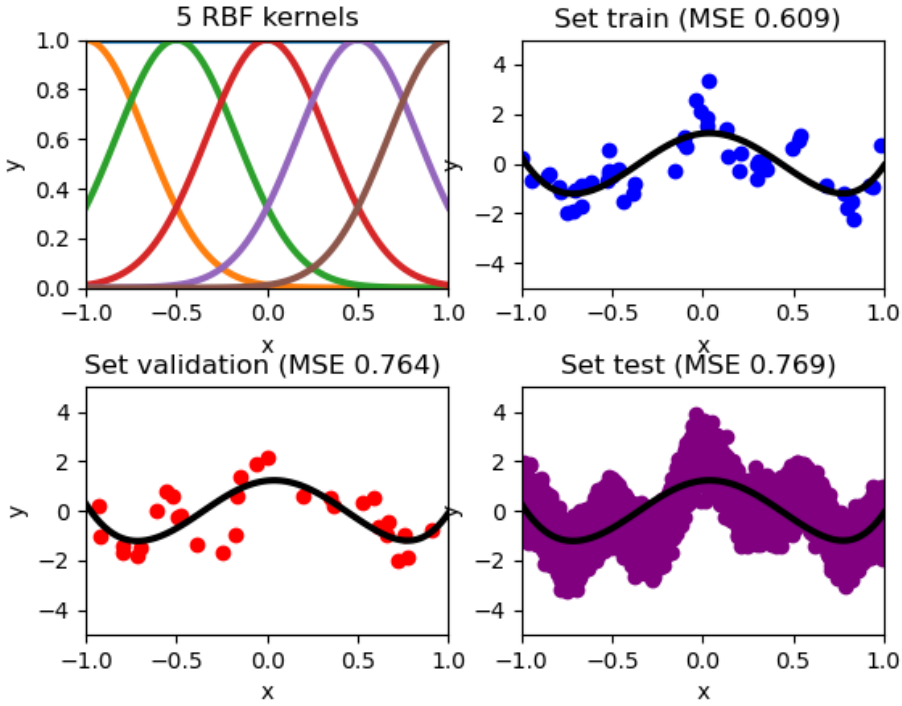

Figure 10: Fitted RBF (*black*) for l =5

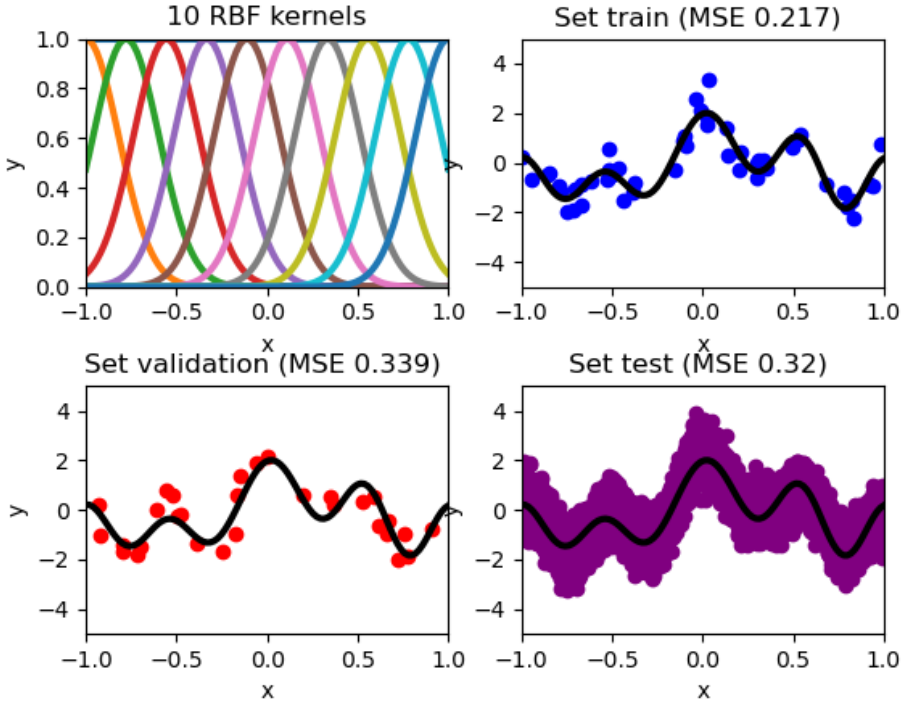

Figure 11: Fitted RBF (*black*) for l =10

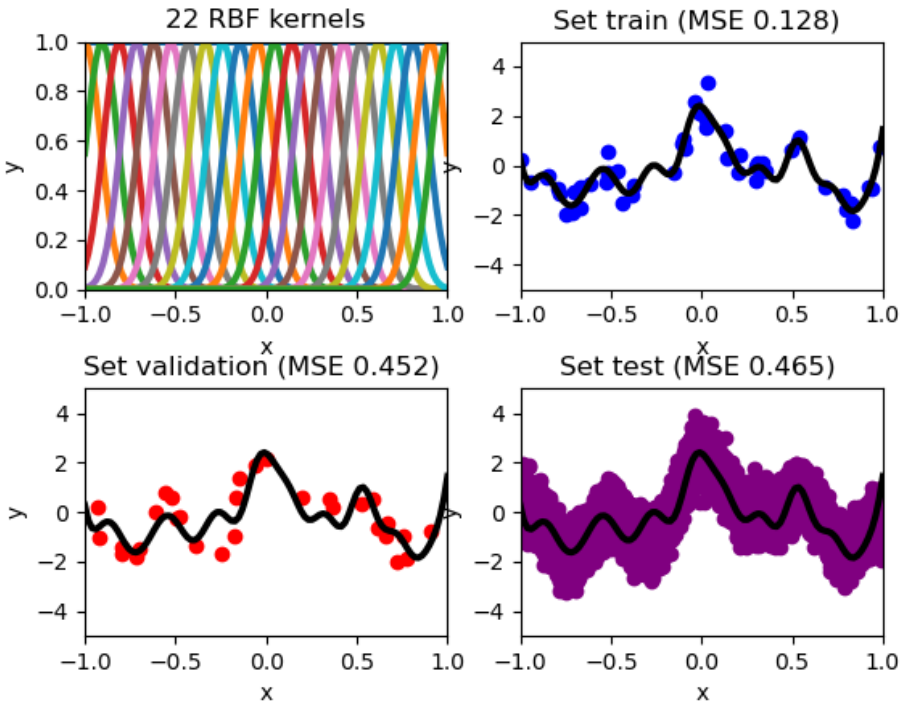

Figure 12: Fitted RBF (*black*) for l =22

1.3.2 Amount of RBFs for lowest MSE on training set

Using the MSE of the training set raises the same issues as for the polynomial basis functions. The 'ideal' RBF of figure 16 in the bottom left plot, oscillates a lot and does not fit the data well. The MSE is also quite big with a value of 213.

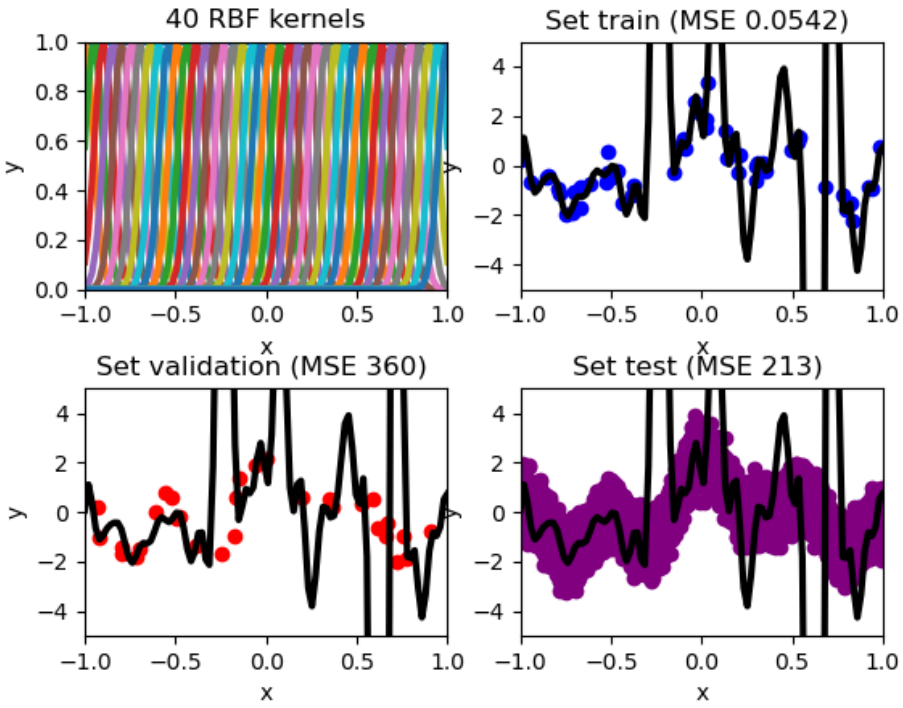

Figure 13: Fitted RBF (*black*) for $l = 40$

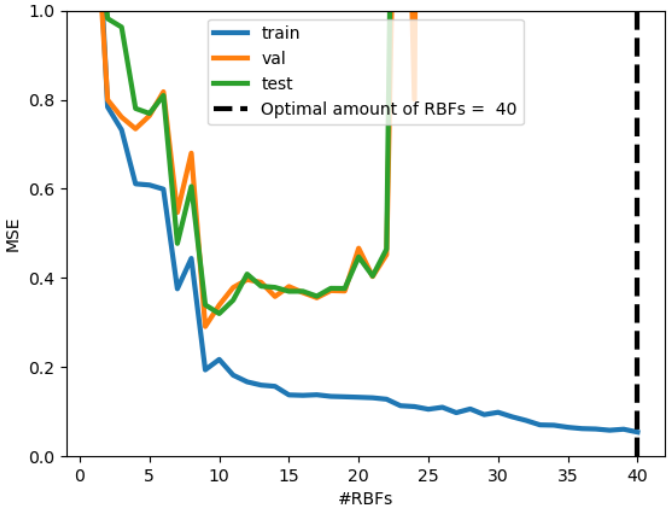

Figure 14: Mean Squared Error for different data sets

1.3.3 Amount of RBFs for lowest MSE on validation set

Here the MSE of the validation set is used to find the best fit.

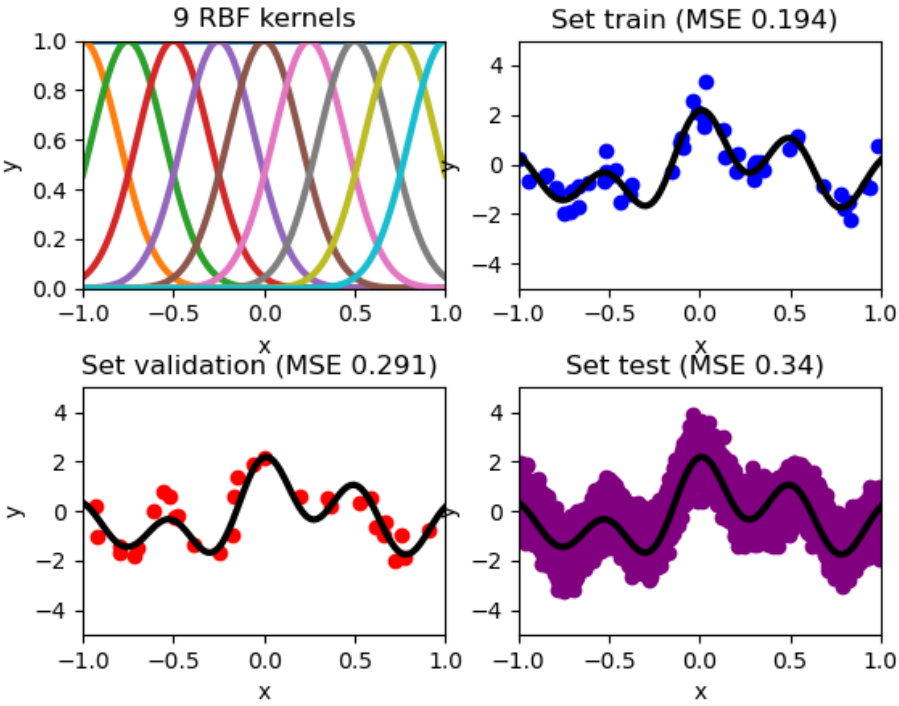

Figure 15: Fitted RBF (*black*) for $l = 9$

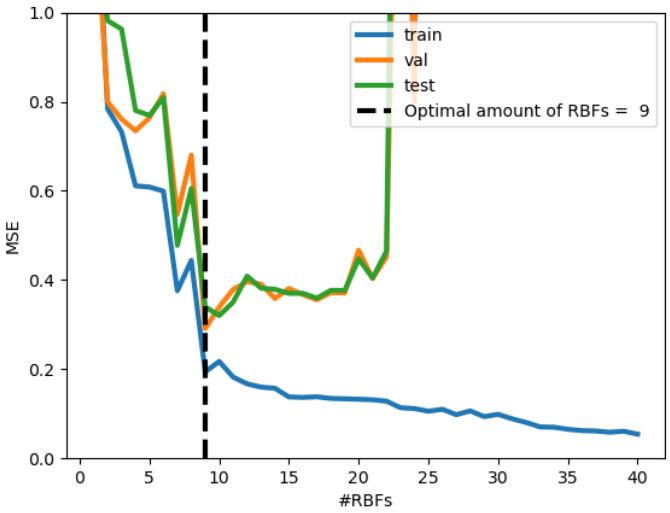

Figure 16: Mean Squared Error for different data sets

Comparing the MSE values of the RBF model to the MSE values of the polynomial, shows that the values of the RBF model are lower and therefore causes less error. The RBF model does a better job at fitting a curve.

2 Logistic Regression

2.1 Derivation of Gradient

- Derive the gradient of the cost function, i.e. show that the partial derivative of the cost function with respect to θ_j equals

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{n=1}^{m}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

$$h_\theta(\mathbf{x}) = \sigma(\mathbf{x}^T\theta) = \sigma(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n)$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} log(h_\theta(\mathbf{x}^{(i)})) + (1-y^{(i)}) log(1-h_\theta(\mathbf{x}^{(i)})))$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial(-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} log(h_\theta(\mathbf{x}^{(i)})) + (1-y^{(i)}) log(1-h_\theta(\mathbf{x}^{(i)}))))}{\partial \theta_j}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \frac{\partial log(h_\theta(\mathbf{x}^{(i)}))}{\partial \theta_j} + (1-y^{(i)}) \frac{\partial log(1-h_\theta(\mathbf{x}^{(i)})}{\partial \theta_j}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \frac{1}{h_\theta(\mathbf{x}^{(i)})} \cdot \frac{\partial h_\theta(\mathbf{x}^{(i)})}{\partial \theta_j} + (1-y^{(i)}) \frac{1}{1-h_\theta(\mathbf{x}^{(i)})} \cdot \frac{\partial 1 - h_\theta(\mathbf{x}^{(i)})}{\partial \theta_j}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \frac{1}{\sigma(\mathbf{x}^T\theta)} \cdot \frac{\partial \sigma(\mathbf{x}^T\theta)}{\partial \theta_j} + (1-y^{(i)}) \frac{1}{1-\sigma(\mathbf{x}^T\theta)} \cdot \frac{\partial 1 - \sigma(\mathbf{x}^T\theta)}{\partial \theta_j}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \frac{\sigma(\mathbf{x}^T\theta)\cdot(1-\sigma(\mathbf{x}^T\theta))}{\sigma(\mathbf{x}^T\theta)} \cdot \frac{\partial \mathbf{x}^T\theta}{\partial \theta_j} + (1-y^{(i)}) \frac{\sigma(\mathbf{x}^T\theta)\cdot(1-\sigma(\mathbf{x}^T\theta))}{1-\sigma(\mathbf{x}^T\theta)} \cdot \frac{\partial \mathbf{x}^T\theta}{\partial \theta_j}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}(1-\sigma(\mathbf{x}^T\theta)) \cdot x_j + (1-y^{(i)}) \cdot \sigma(\mathbf{x}^T\theta) \cdot x_j$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} (y^{(i)} - y^{(i)}\sigma(\mathbf{x}^T\theta) + \sigma(\mathbf{x}^T\theta) - y^{(i)}\sigma(\mathbf{x}^T\theta)) \cdot x_j$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m} (y^{(i)} - \sigma(\mathbf{x}^T\theta)) \cdot x_j$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \cdot x_j$$

$$\frac{\partial \mathbf{x}^T\theta}{\partial \theta_j} = x_j$$

2.2 Logistic Regression training with gradient descent

2.2.1

The *check_gradient* function uses the finite difference method to approximate the gradient. The approximation uses three different deltas (1e-2, 1e-4, 1e-6) and then calculates the difference between the implemented function df with the result of the approximation. If the difference is not too big the functions df and f are correct.

2.2.2

Iteration = 20 , η = 1, degree = 1, error = 0.48117520966815236 , Training - Data

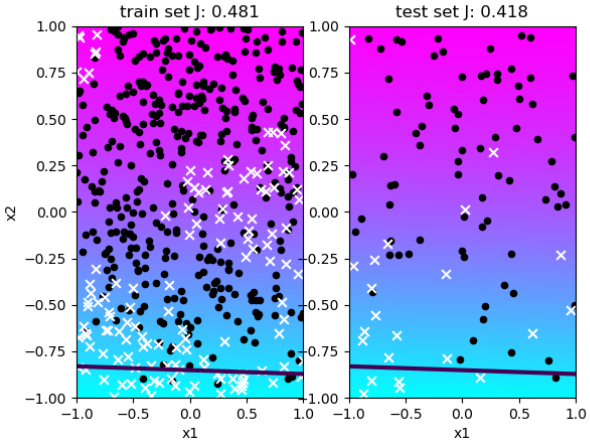

Figure 17: Decision Boundaries

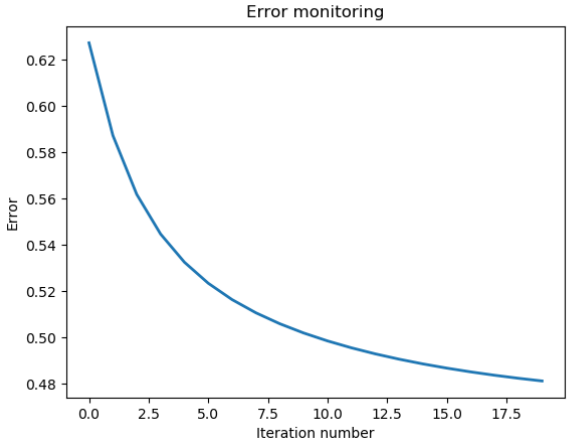

Figure 18: Error

Iteration = 20 , η = 1, degree = 1, error = 0.37168855271211126 , Testing - Data

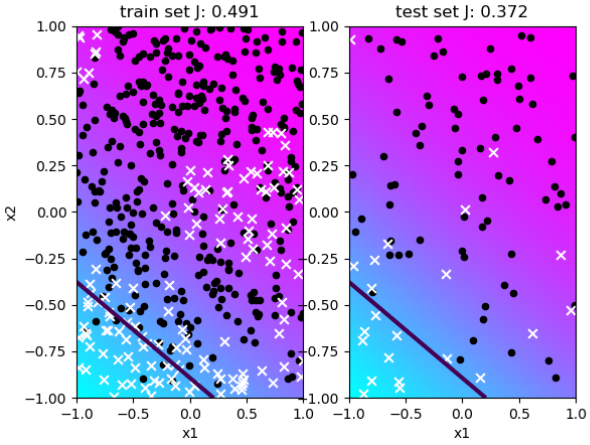

Figure 19: Decision Boundaries

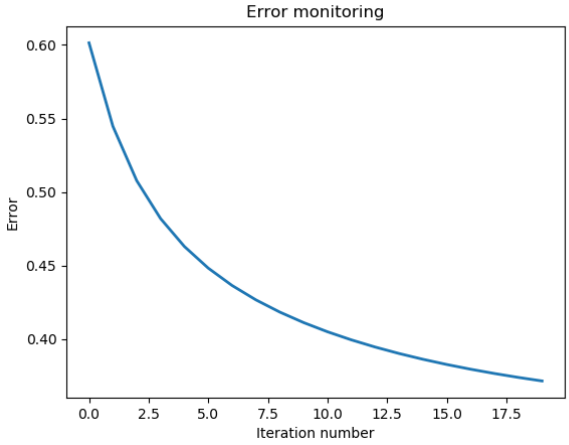

Figure 20: Error

Iteration = 2000 , η = 1, degree = 1, error = 0.46918315203779537 , Training - Data

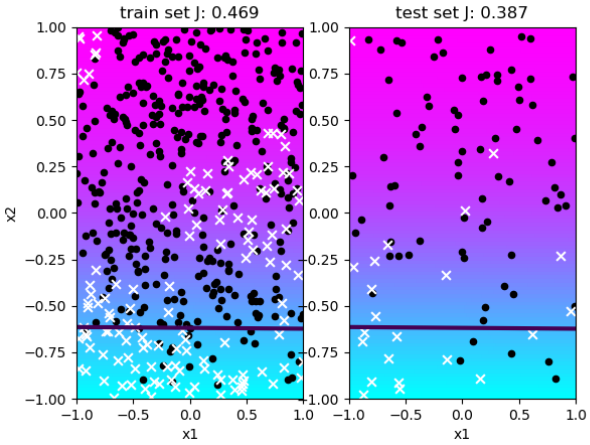

Figure 21: Decision Boundaries

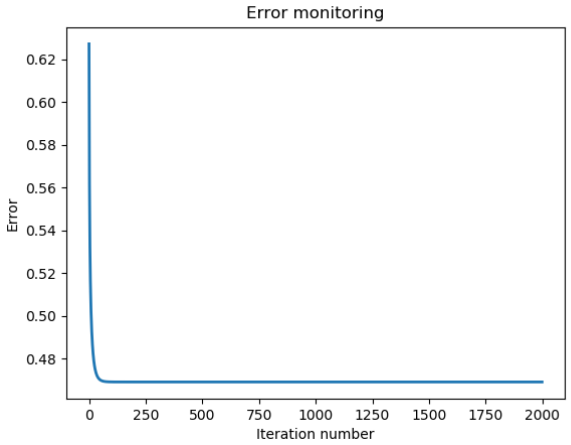

Figure 22: Error

Iteration = 2000 , η = 1, degree = 1, error = 0.33718964259351336 , Testing - Data

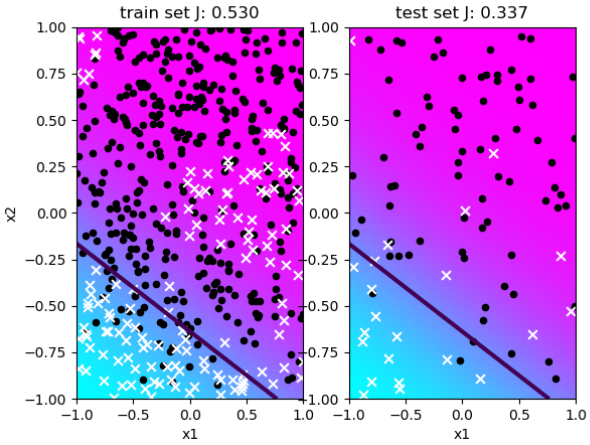

Figure 23: Decision Boundaries

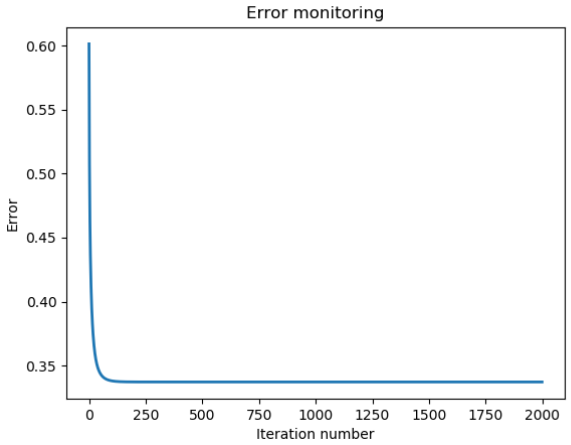

Figure 24: Error

If the iteration is to small we don't reach the error minimum. If the iteration is to high after reaching the minimum the computation only takes longer without improving the system.

2.2.3

iteration = 200 , η = 0.1, degree = 2, error = 0.48100547820950806 , Training - Data

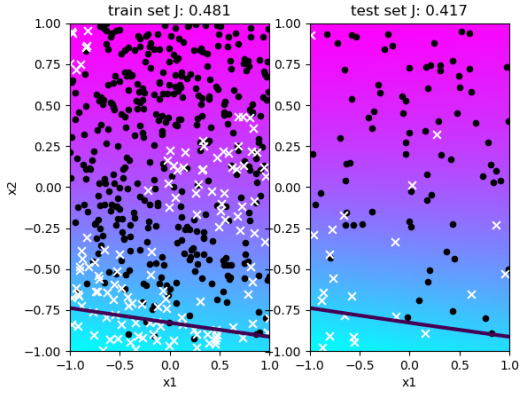

Figure 25: Decision Boundaries

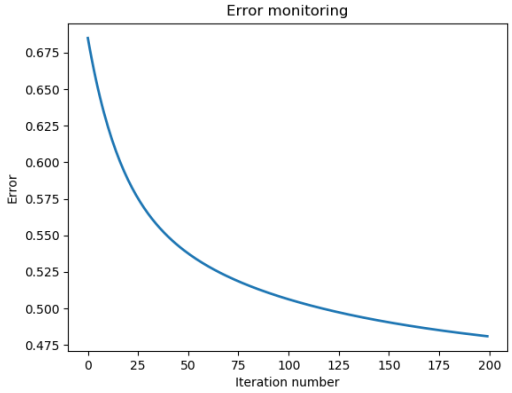

Figure 26: Error

iteration = 200 , η = 2, degree = 2, error = 0.44967555750709887 , Training - Data

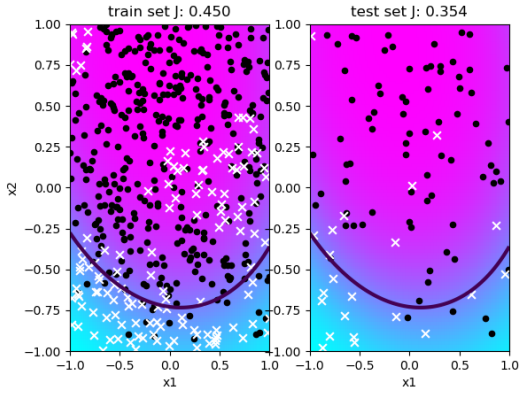

Figure 27: Decision Boundaries

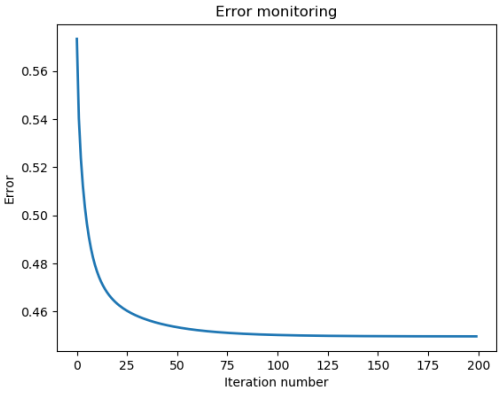

Figure 28: Error

iteration = 200 , η = 20, degree = 2, error = 0.7584680978078007 , Training - Data

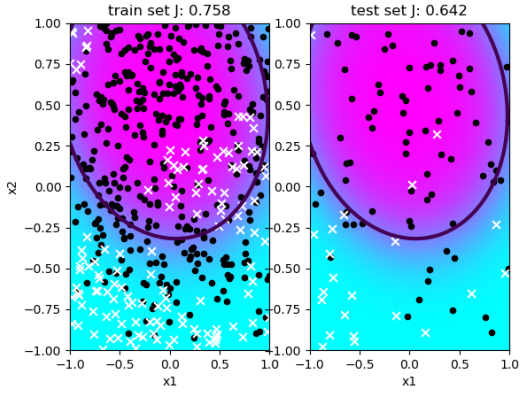

Figure 29: Decision Boundaries

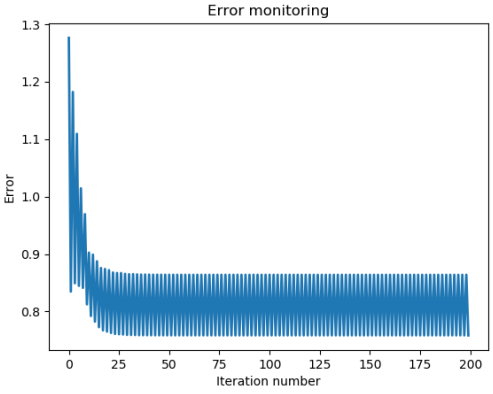

Figure 30: Error

iteration = 200 , η = 0.1, degree = 2, error = 0.36760050105672837 , Testing - Data

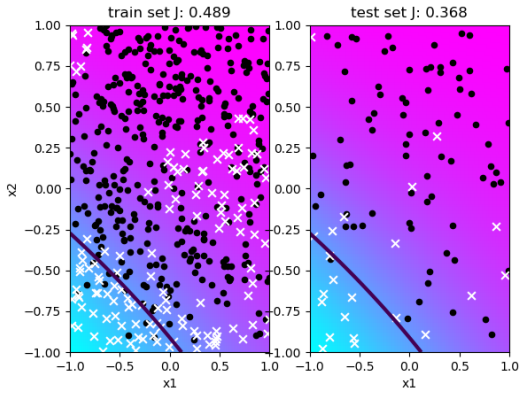

Figure 31: Decision Boundaries

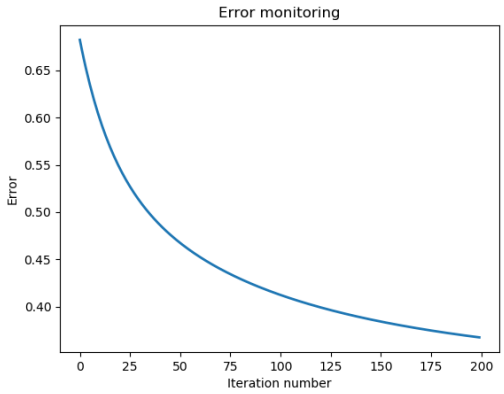

Figure 32: Error

iteration = 200 , η = 2, degree = 2, error = 0.3133315359994844 , Testing - Data

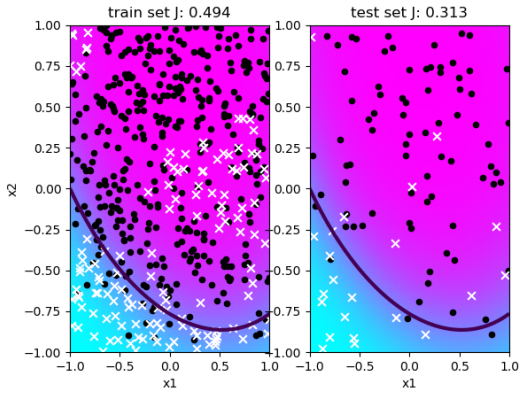

Figure 33: Decision Boundaries

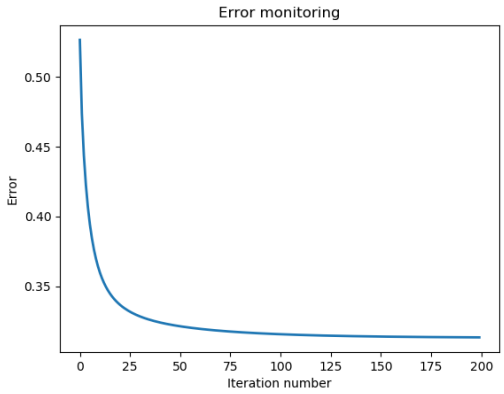

Figure 34: Error

iteration = 200 , η = 20, degree = 2, error = 0.3875239811075353 , Testing - Data

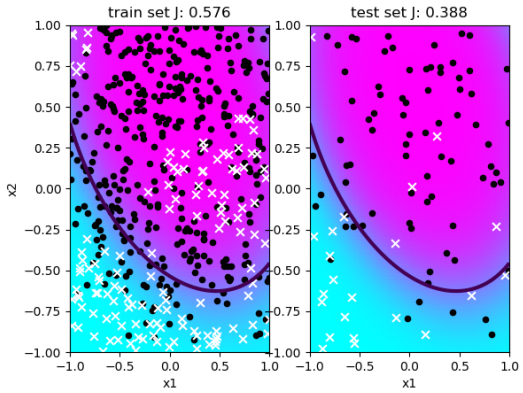

Figure 35: Decision Boundaries

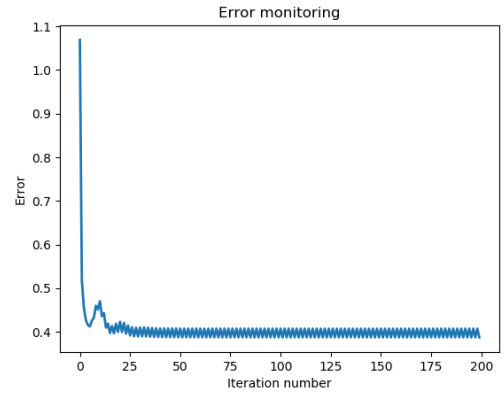

Figure 36: Error

After comparing the error of the different η we see, that a small η need many iteration to reach

the minimum. On the other hand a big η can lead to a bouncing between some values around the minimum, because the step-size is to big.

2.2.4

To find the good solutions we calculated some different pairs of values and then tried to use a loop to further improve the pairs. The first loop increases the η until it reaches the point where it has the lowest error. Then we increase the iteration until the error is at its *minimum*. The *minimum* is not the perfect solution but it increases the performance better than only trying some value pairs.

degree = 1, iteration = 65 , η = 12.6, error = 0.3371896383741966 , Testing - Data

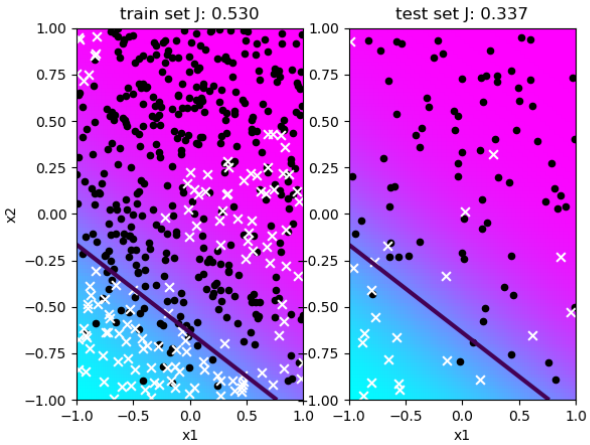

Figure 37: Decision Boundaries

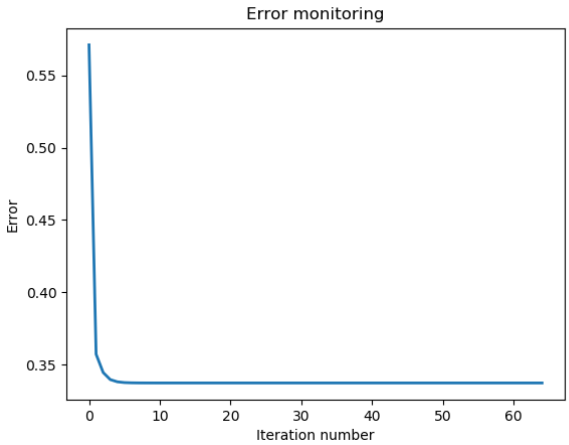

Figure 38: Error

degree = 2, iteration = 110 , η = 15, error = 0.31296818043517993 , Testing - Data

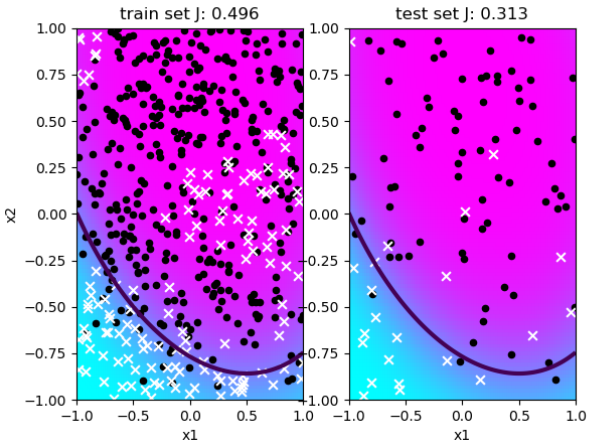

Figure 39: Decision Boundaries

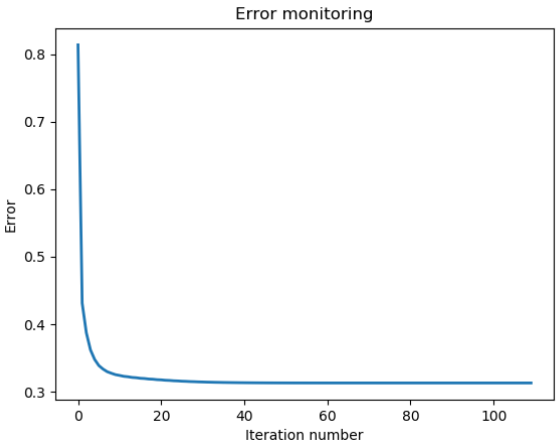

Figure 40: Error

degree = 7, iteration = 1820 , η = 24.1, error = 0.18541346106408038 , Testing - Data

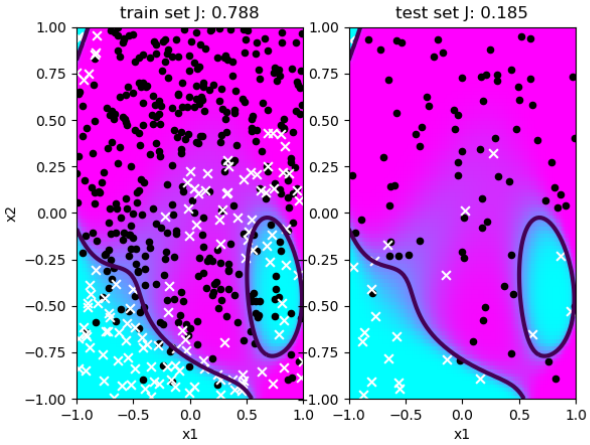

Figure 41: Decision Boundaries

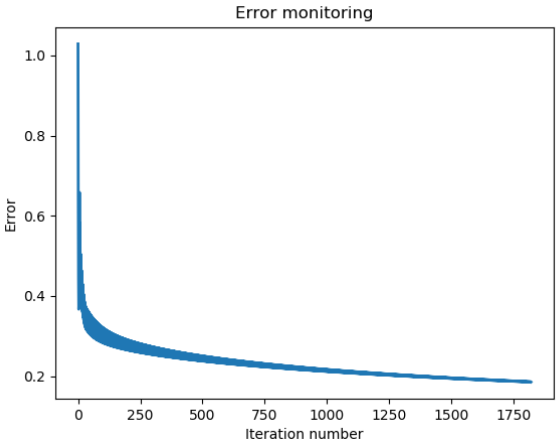

Figure 42: Error

degree = 20, iteration = 5420 , η = 33.1, error = 0.12605281939207935 , Testing - Data

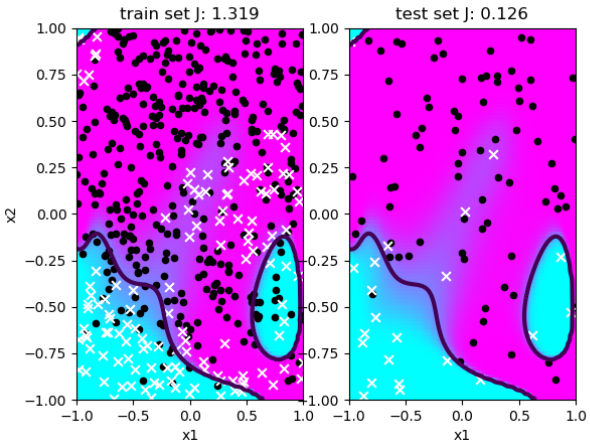

Figure 43: Decision Boundaries

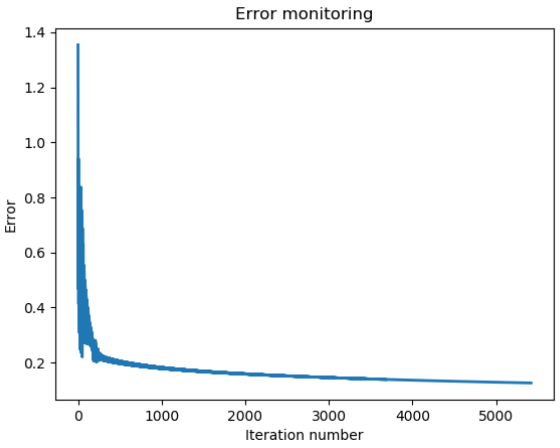

Figure 44: Error

2.2.5

To find a local minimum of a function we can use the gradient descent, we take steps proportional to the negative of the gradient of the function at the current point. If the gradient reaches 0 we arrive at the global minimum of the cost function. Due to numerical errors the gradient will never be exactly 0, hence we define a threshold and if the gradient is below that threshold we stop.