Assignment 2

Computational Intelligence, SS2020

Team Members		
Last name	First name	Matriculation Number
Blöcher	Christian	01573246
Bürgener	Max	01531577

1 Linear regression

1.1 Derivation of Regularized Linear Regression

- Why is the design matrix $\boldsymbol{X}$ containing $n+1$ and not just n?
  - The simplest form of linear regression is calculated by:

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{\omega}) = \underbrace{\boldsymbol{\omega_0}}_{bias} + \boldsymbol{X}\boldsymbol{\omega}$$

  - Modifying $\boldsymbol{X}$ with the additional column. The addition of the bias ω_0 is compensated and the calculation simplified.

$$\boldsymbol{y} = \begin{bmatrix} 1 & \dots & x_1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & x_N \end{bmatrix} \begin{bmatrix} \omega_0 \\ \vdots \\ \omega_N \end{bmatrix}$$

- What is the dimension of the gradient vector?
  - The gradient contains the derivation of all $J(\boldsymbol{\theta})$ for all variables. That means the gradient has the dimension $m \times 1$ and it contains a group of targets.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_m} \end{bmatrix}$$

- What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix?
  - The Jacobian matrix elements are calculated by the derivatives of all elements of a vector with respect to all variables.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

  - The gradient is computed from a scalar. It is a vector which contains the derivatives of a function for all variables.

- What is the dimension of the Jacobian matrix and what is it equal to?
  - The dimension of the Jacobian matrix not only equal the dimension of the Design matrix, but the result itself is the Design matrix. Because $\frac{\partial \boldsymbol{Ab}}{\partial \boldsymbol{b}} = \boldsymbol{A}$:

$$\boldsymbol{J} = \frac{\partial \boldsymbol{X}\boldsymbol{\theta}}{\partial \boldsymbol{\theta}} = \boldsymbol{X}$$

- Minimization of the regularized linear regression cost function

$$\begin{aligned} J(\boldsymbol{\theta}) &= \tfrac{1}{m}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \tfrac{\lambda}{m}\|\boldsymbol{\theta}^2\| \\ J(\boldsymbol{\theta}) &= \tfrac{1}{m}\left[(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^T \cdot (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})\right] + \tfrac{\lambda}{m}(\boldsymbol{\theta^T}\boldsymbol{\theta}) \quad | \ \tfrac{\partial}{\partial \boldsymbol{\theta}} \end{aligned}$$

Using Hint 2 from our exercise sheet the calculation is simplified to:

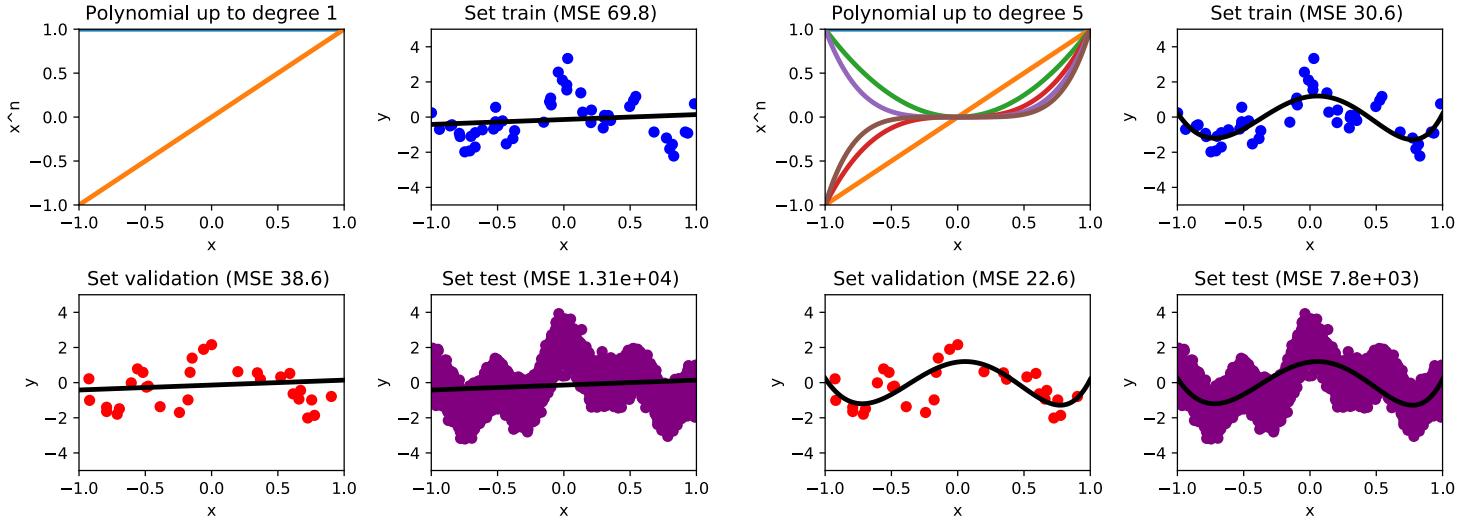
$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{2}{m}(\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot \boldsymbol{X} + \frac{2\lambda}{m}\,\boldsymbol{\theta}^T$$

We set the gradient to zero to calculate our solution θ. Due to the property of transpose: $(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$ we get:

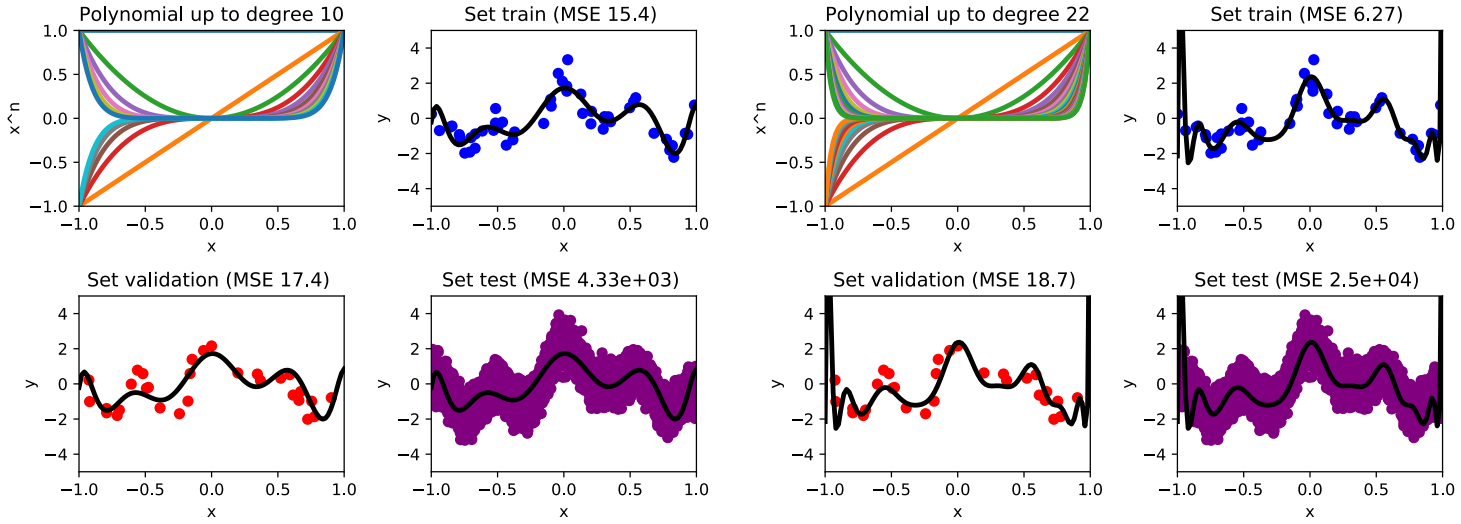
$$\begin{aligned}
\tfrac{2}{m}(\boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{y}^T\boldsymbol{X}) + \tfrac{2\lambda}{m}\boldsymbol{\theta}^T &= 0 \\
\boldsymbol{\theta}^T\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{\theta}^T &= \boldsymbol{y}^T\boldsymbol{X} \\
\boldsymbol{\theta}^T(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}) &= \boldsymbol{y}^T\boldsymbol{X} \\
\boldsymbol{\theta}^T &= (\boldsymbol{y}^T\boldsymbol{X})(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1} \quad |\,()^T
\end{aligned}$$

$$\begin{aligned}
\theta &= (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1^T}(\boldsymbol{y}^T\boldsymbol{X})^T \\
\theta &= (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}(\boldsymbol{X}^T\boldsymbol{y})
\end{aligned}$$

1.2 Linear Regression with polynomial features



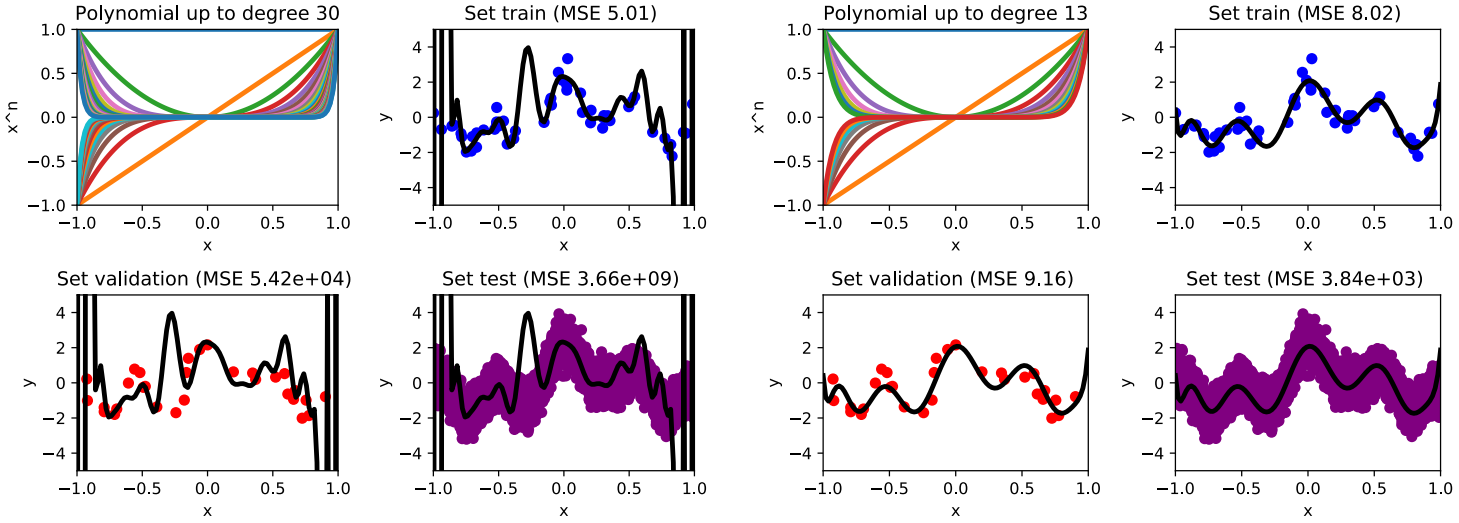
(a) $n = 1$ (b) $n = 5$



(c) $n = 10$ (d) $n = 22$

Figure 1: Results of Linear Regression for varying polynomial degree n.

As can be seen in figure 3 the training cost decreases with increasing polynomial degree n. That means the lowest cost on the training set can be achieved by using polynomials up to the highest degree $n = 30$ (s. figure 2), but then the testing cost is maximised. This is due to overfitting: By finding parameters that suit the training data best, the solution becomes too specific for the validation and testing sets, leading to greater errors. Looking at figure 2 all training data points are in the close vicinity of the polynomial but the output function barely resembles the data. The best results can be obtained with degree $n = 13$, which minimises the validation cost (s. figure 3) and leads to a very low testing cost. Having a validation set in addition to the training set helps in finding the right degree for the linear regression process and greatly improves the quality of the solution.



(a) Training set, $n = 30$

(b) Validation set, $n = 13$

Figure 2: Results with lowest cost on training and validation set.

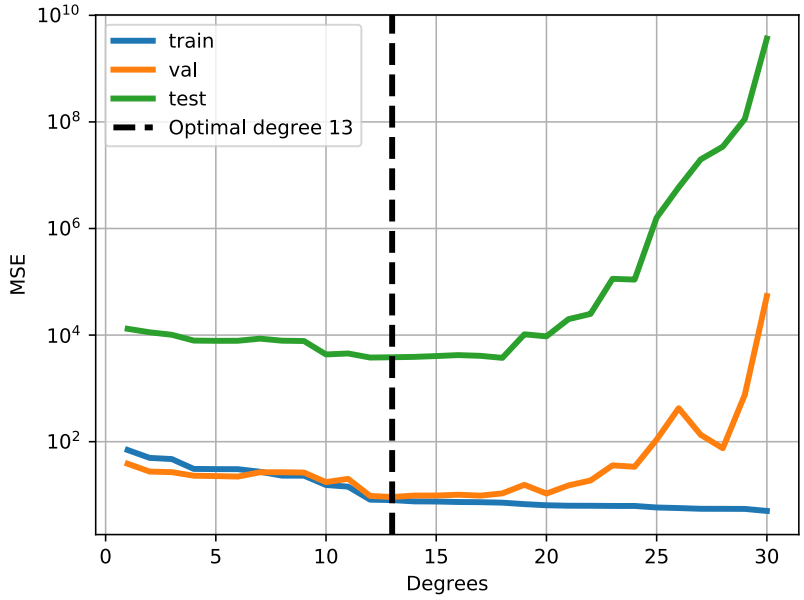
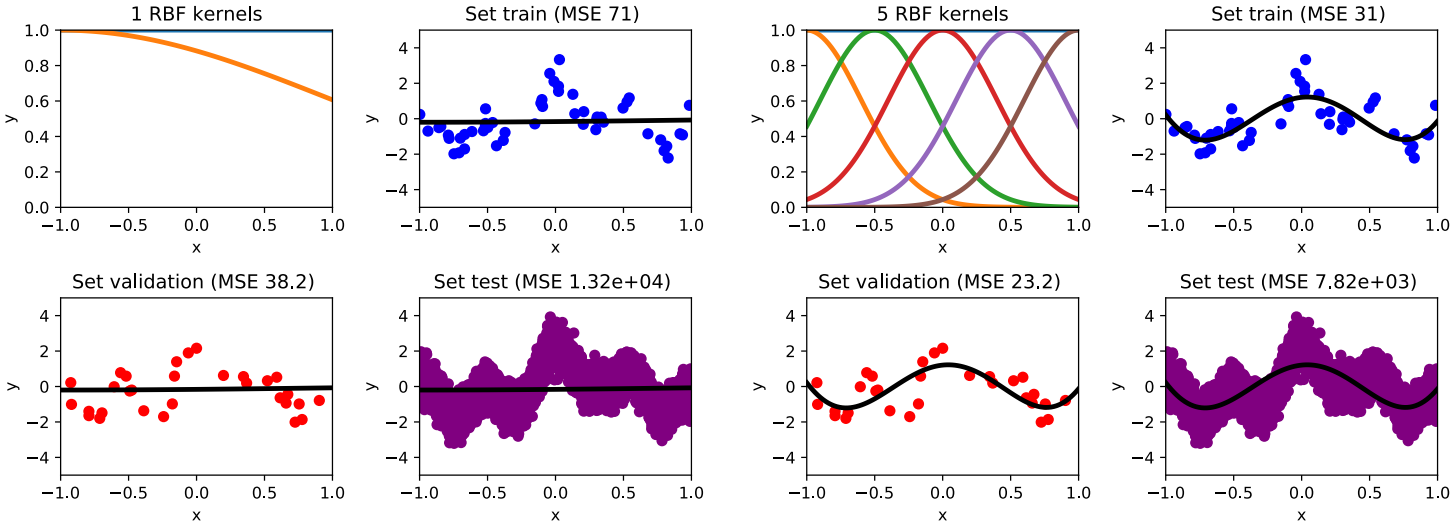


Figure 3: Training, validation and testing costs as a function of the polynomial degree n.

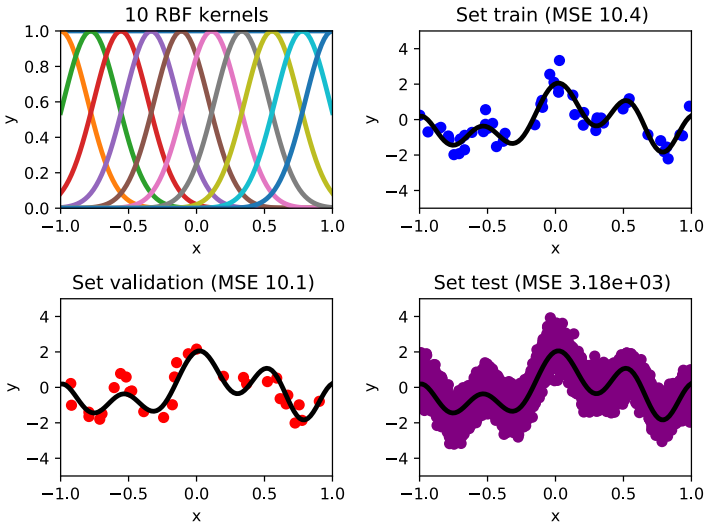
1.3 (Bonus) Linear Regression with radial basis functions

The results of the Linear Regression approach with radial basis functions are similar to the ones with polynomial functions: Increasing the degree/number of kernels l leads to lower training cost (s. figure 6) - minimised for $l = 40$ (s. figure 5) - but choosing l too high results in overfitting (s. figure 4). The cost function of the validation set is minimised for degree $l = 9$ (s. figure 5), leading to a low but not quite minimised testing cost. The actual testing cost minimum is found with degree $l = 10$. Improving on the results of Linear Regression with polynomial basis functions it is about 20% lower than the testing cost minimum with (higher) degree $n = 13$ (s. figures 4 and 2).

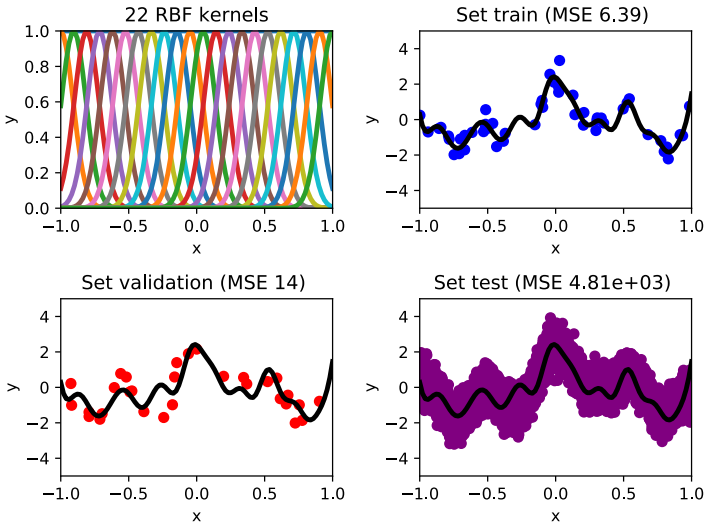


(a) $l = 1$

(b) $l = 5$

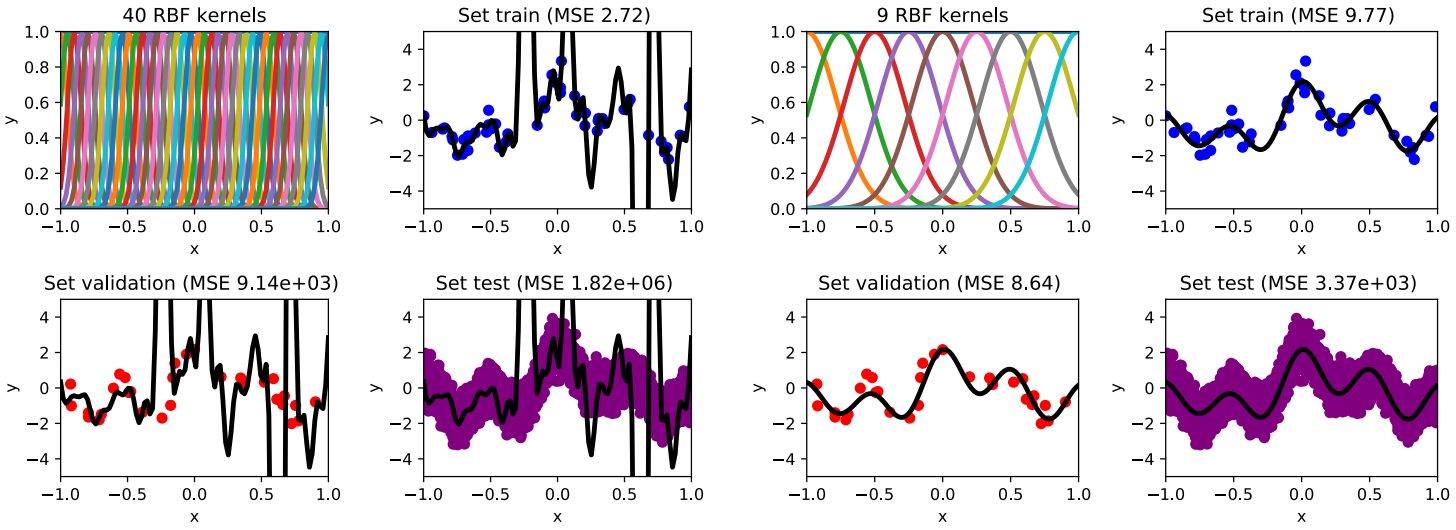


(c) $l = 10$



(d) $l = 22$

Figure 4: Results of Linear Regression for varying degree l.



(a) Training set, $l = 40$

(b) Validation set, $l = 9$

Figure 5: Results with lowest cost for training and validation set.

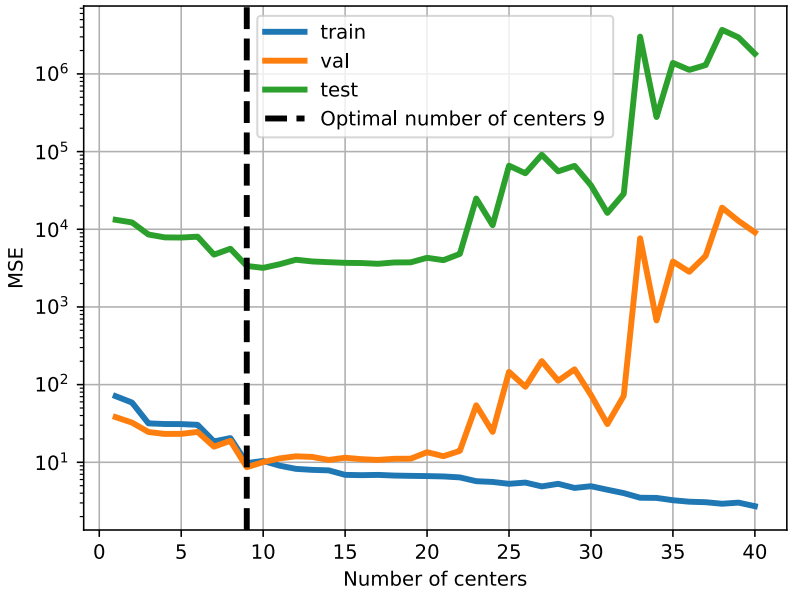


Figure 6: Training, validation and testing costs as a function of the degree l.

2 Logistic Regression

2.1 Derivation of Gradient

$$J(\boldsymbol{\theta}) \quad = \quad -\tfrac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\,\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) + (1-y^{(i)})\,\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\right) \quad \mid \tfrac{\partial}{\partial \boldsymbol{\theta}_j}$$

Derivation of the hypothesis function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$:

$$\tfrac{\partial h_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j} = \tfrac{\partial}{\partial \theta_j}\sigma(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n) = \sigma(\boldsymbol{x}^T\boldsymbol{\theta})\left(1-\sigma(\boldsymbol{x}^T\boldsymbol{\theta})\right)x_j = h_{\boldsymbol{\theta}}(\boldsymbol{x})\,(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))\,x_j$$

Using the chain rule and the given hint for the derivation of the sigmoid function σ, we get:

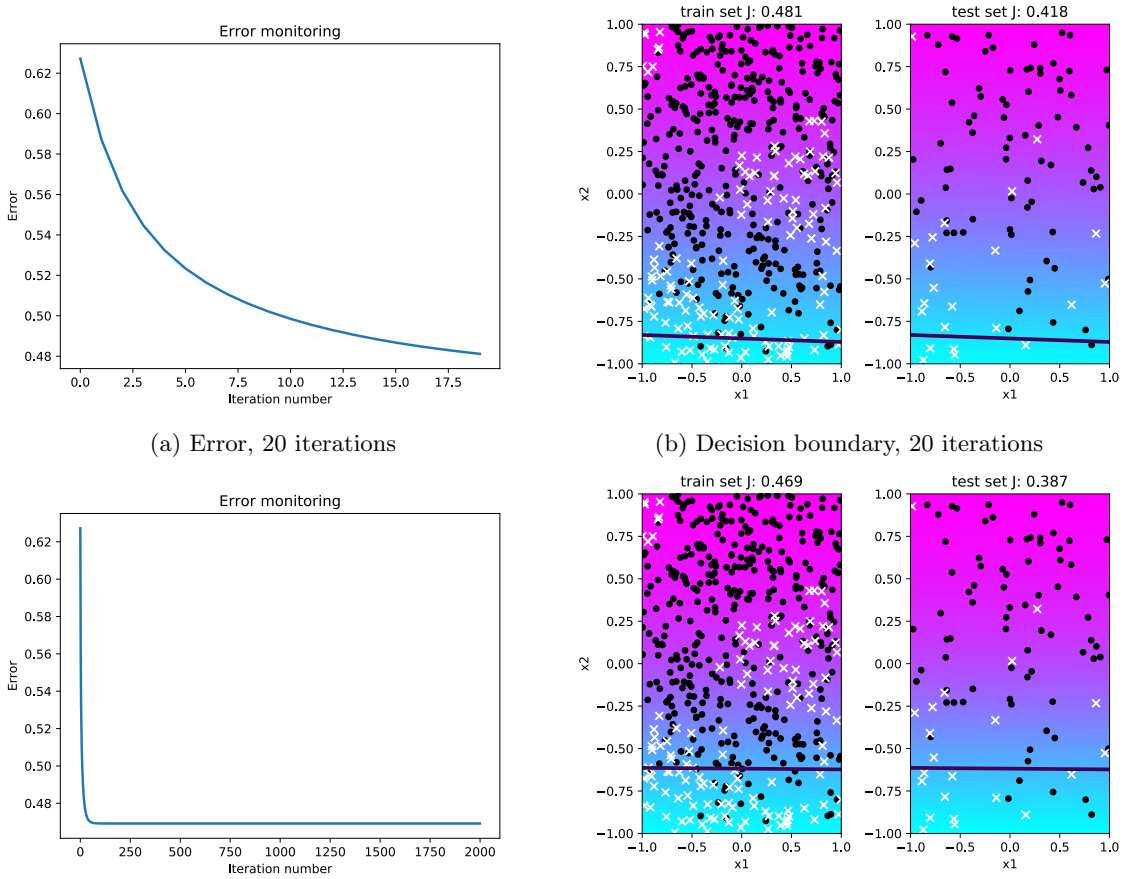
$$\begin{aligned}
\tfrac{J(\boldsymbol{\theta})}{\partial \theta_j} \quad &= \quad -\tfrac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\,x_j^{(i)} + \ldots\right. \\
&\qquad \left.\ldots(1-y^{(i)})\tfrac{1}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}\left(-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\,(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\,x_j^{(i)}\right)\right) \\
\tfrac{J(\boldsymbol{\theta})}{\partial \theta_j} \quad &= \quad -\tfrac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)x_j^{(i)} \\
\tfrac{J(\boldsymbol{\theta})}{\partial \theta_j} \quad &= \quad \tfrac{1}{m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\,x_j^{(i)}
\end{aligned}$$

2.2 Logistic Regression training with gradient descent

- What is the function `check_gradient` doing?
  - The cost f_0 and gradient $\boldsymbol{g}_0$ for a randomly chosen $\boldsymbol{x}_0$ are determined via the functions implemented in `logreg.py`. Then a random direction $\boldsymbol{dx}$ in which to check the gradient is generated and a scalar df_g containing the sum of the gradient elements weighted by the elements of $\boldsymbol{dx}$ is computed via the dot product. For three predetermined deltas $d = 10^{-2}, 10^{-4}, 10^{-6}$ the cost f_1 at $\boldsymbol{x}_0 + d \cdot \boldsymbol{dx}$ is computed (again via the implemented function) and the gradient df is determined as:

$$df = \frac{\Delta f}{\Delta x} = \frac{f_1 - f_0}{d}$$

    Then the logarithm of the error $|df_g - df|$ is computed for each d (the added constant 10^{-20} prevents the logarithm from becoming zero). If for decreasing d the log-error always decreases by more than 1 or if all log-errors are smaller than -20 the gradient is well approximated and a confirmation message is printed. Otherwise an error is raised.



(a) Error, 20 iterations

(b) Decision boundary, 20 iterations

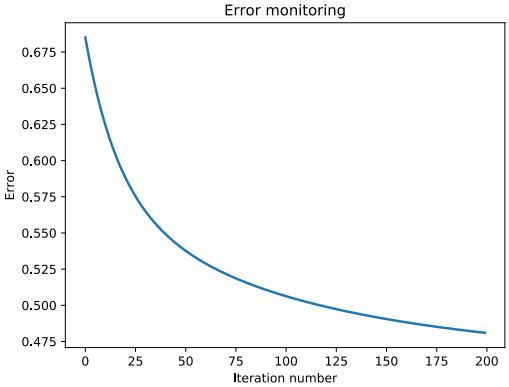
(c) Error, 2000 iterations

(d) Decision boundary, 2000 iterations

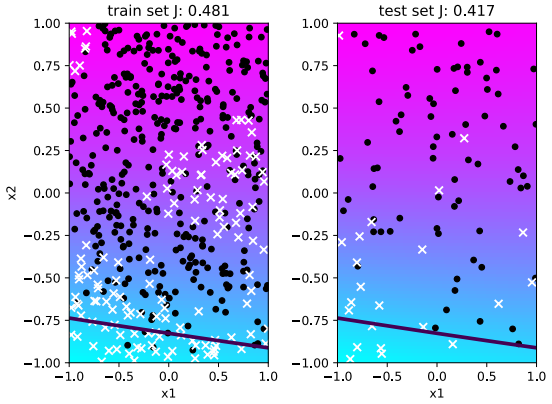
Figure 7: GD errors and decision boundaries for varying number of iterations, degree $l = 1$ and learning rate $\eta = 1$.

- For degree $l = 1$ and learning rate $\eta = 1$ the GD was run for 20 and 2000 iterations (s. figure 7). With only 20 iterations the GD algorithm stops before convergence leading to an inaccurate decision boundary and higher training and testing costs. Performing 2000 iterations one gets a more accurate decision boundary but the algorithm already converges

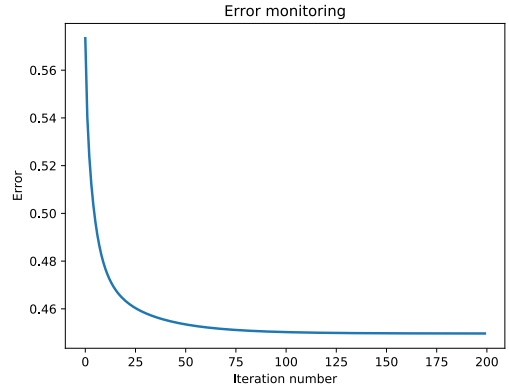
after a fraction of the iterations, essentially wasting the majority of the computation time. A possible solution to prevent this would be the stopping criterion proposed at the end of this section.



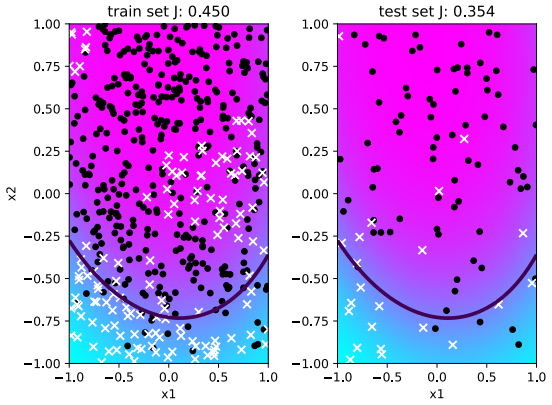
(a) Error, $\eta = 0.1$



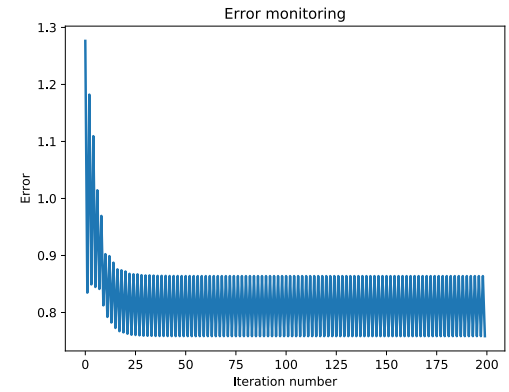
(b) Decision boundary, $\eta = 0.1$



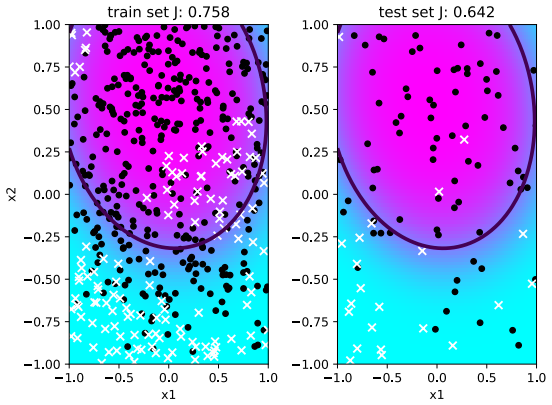
(c) Error, $\eta = 2$



(d) Decision boundary, $\eta = 2$



(e) Error, $\eta = 20$



(f) Decision boundary, $\eta = 20$

Figure 8: GD errors and decision boundaries for varying learning rate η, degree $l = 2$ and 200 iterations.

- For degree $l = 2$ the GD algorithm was run for 200 iterations with learning rates $\eta = 0.1, 2, 20$ (s. figure 8). The learning rate $\eta = 0.1$ is too small, preventing the algorithm from converging in time. The result is an inaccurate decision boundary and higher training and testing costs. $\eta = 2$ works well with the other set parameters. The GD algorithm converges, the decision

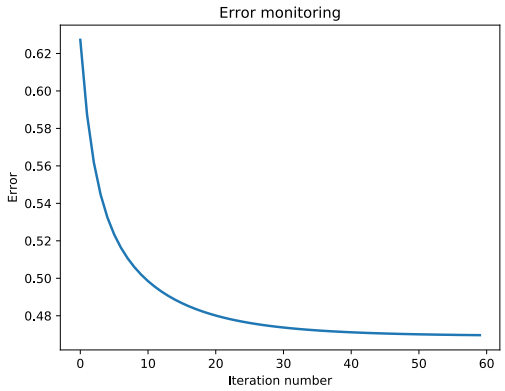
boundary works is quite accurate considering the low degree used and the training and testing costs are the lowest of the three results. The last learning rate $\eta = 20$ on the other hand is much too large. Rather than reaching the location of the minimum the algorithm overshoots repeatedly, preventing convergence and causing the algorithm to oscillate. Of course the resulting decision boundary is basically useless and the training and testing costs are very high.

- For degrees $l = 1, 2, 7, 20$ reasonably good pairs of values for the number of iterations and learning rate have been identified (s. figure 9 and table 1). Especially with higher degrees it has turned out to be quite difficult to find the right balance between keeping the computational time at acceptable levels and finding good learning rates, because even after the initial steep descent the error continues to slowly decrease. Of all tested degrees $l = 7$ seems to fit the data best although the testing cost is slightly lower with $l = 2$. The lowest training cost is obtained by using $l = 20$ but this results in overfitting.

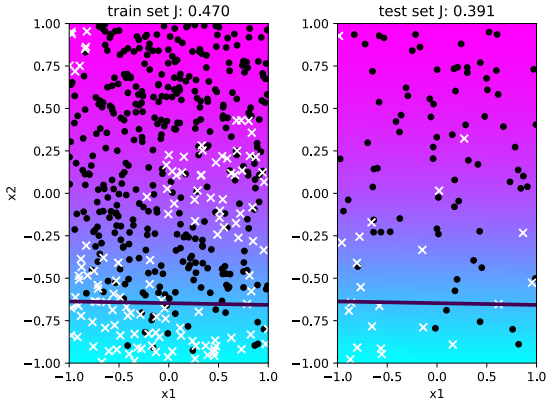
degree l	learning rate η	iterations	training cost	testing cost
1	1	60	0.470	0.391
2	5	75	0.450	0.354
7	10	1000	0.310	0.361
20	8	1500	0.297	0.398

Table 1: Chosen values for learning rate η and number of iterations and obtained training and testing costs for degrees $l = 1, 2, 7, 20$.

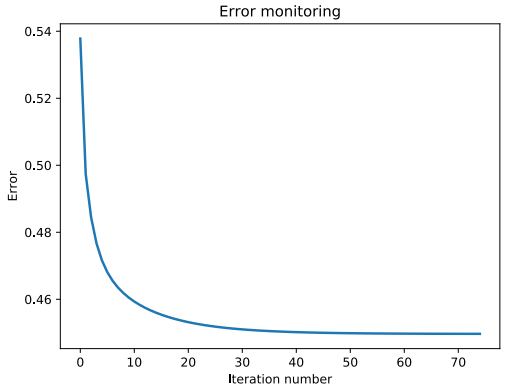
- What could a possible stopping criterion look like?
  - In the GD update equation $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ the elements of the gradient vector contain the updates for each dimension of $\boldsymbol{\theta}$. If an element of the gradient is very small the solution will only change very little in that dimension. So on each iteration of the GD before updating $\boldsymbol{\theta}$ one could compare the elements of the gradient to a threshold. If all elements are smaller than the threshold the location of the minimum of the cost function is reasonably well approximated and the update loop can be aborted.



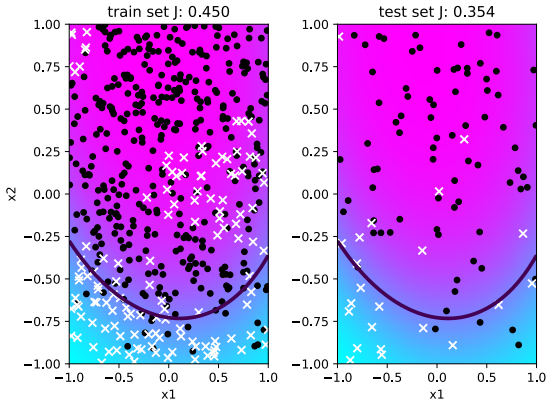
(a) Error, $l = 1$, $\eta = 1$, 60 iterations



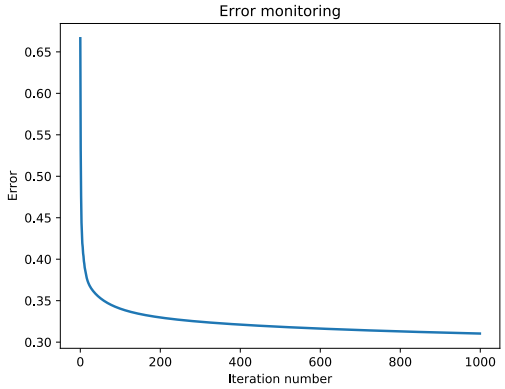
(b) Decision boundary, $l = 1$, $\eta = 1$, 60 iterations



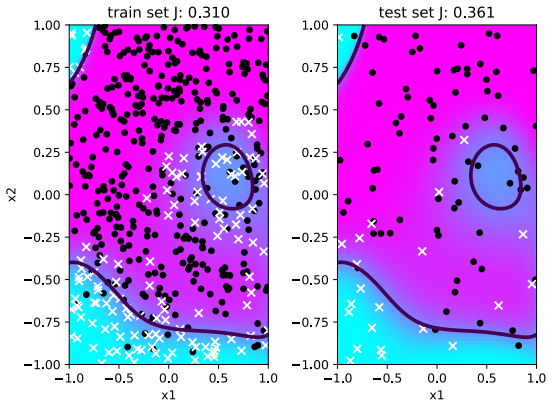
(c) Error, $l = 2$, $\eta = 5$, 75 iterations



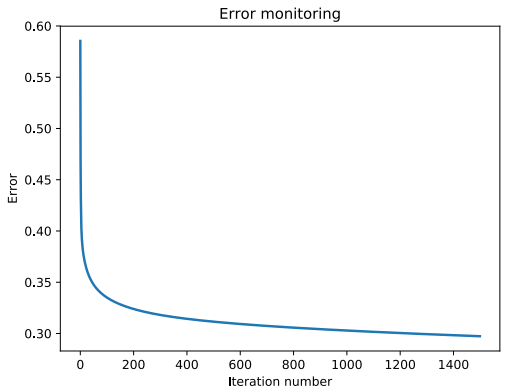
(d) Decision boundary, $l = 2$, $\eta = 5$, 75 iterations



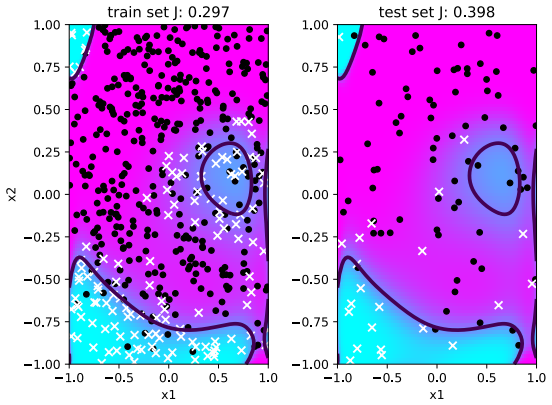
(e) Error, $l = 7$, $\eta = 10$, 1000 iterations



(f) Decision boundary, $l = 7$, $\eta = 10$, 1000 iterations



(g) Error, $l = 20$, $\eta = 8$, 1500 iterations



(h) Decision boundary, $l = 20$, $\eta = 8$, 1500 iterations

Figure 9: GD errors and decision boundaries for degrees $l = 1, 2, 7, 20$ and chosen parameter values.