Assignment 1

Computational Intelligence

Team Members		
Last name	First name	Matriculation Number
Merdes	Malte	01331649
Riedel	Stefan	01330219

1 Linear Regression

1.1 Derivation of Regularized Linear Regression

(a) Preliminary Questions:

n equals the number of features. However, to allow an offset parameter θ_0 as required in linear regression we need an additional column of ones ($x^0 = 1$) for X. Since we have $n+1$ parameters $\theta_0 ... \theta_n$ the design matrix X needs to be of dimension $m \times (n+1)$.

Gradient of dimension $(n+1) \times 1$:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = [\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, ..., \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n}]^T$$

Jacobi-Matrix:

$$\left(\frac{\partial f_i}{\partial \theta_j}(\boldsymbol{\theta})\right)_{i=1,...,m;\ j=1,...,n} = \begin{pmatrix} \frac{\partial f_1}{\partial \theta_1}(\boldsymbol{\theta}) & \frac{\partial f_1}{\partial \theta_2}(\boldsymbol{\theta}) & \dots & \frac{\partial f_1}{\partial \theta_n}(\boldsymbol{\theta}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \theta_1}(\boldsymbol{\theta}) & \frac{\partial f_m}{\partial \theta_2}(\boldsymbol{\theta}) & \dots & \frac{\partial f_m}{\partial \theta_n}(\boldsymbol{\theta}) \end{pmatrix}$$

The Jacobian matrix is needed when whe want to derive vector fields, other than scalar fields (gradient).

Dimension of $X\boldsymbol{\theta}$: $m \times 1$
Dimension of Jacobian matrix $\frac{\partial X\boldsymbol{\theta}}{\partial \boldsymbol{\theta}}$: $m \times (n+1)$

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

Assuming that θ is of dimension $(n+1) \times 1$, we calculate the solution of the regularized cost function:

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

The diagonal elements of $\boldsymbol{X^T X}$ are always greater or equal zero. Through addition of λ_i, we can assure non-zero eigenvalues, which corresponds to invertibility of $(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})$ in any case of $\lambda > 0$.

1.2 Linear Regression with polynomial features

As seen in figures 1 to 5 (polynomial degrees: 1, 2, 5, 20, 30), the higher the degree, the better we can fit our training data.

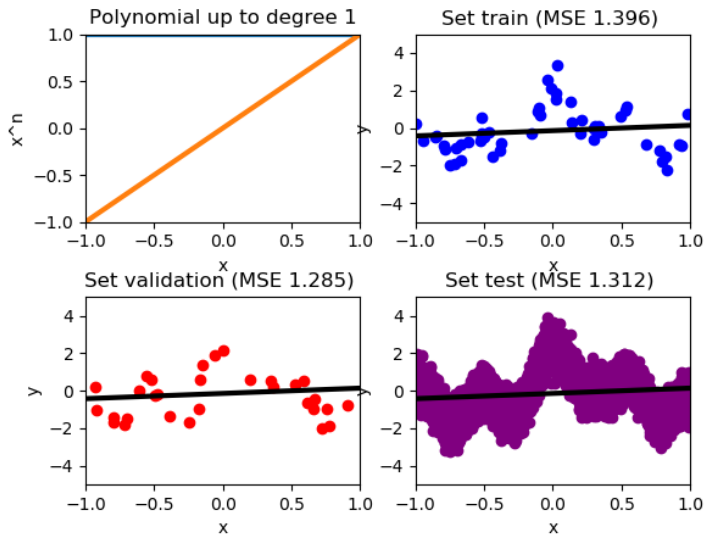

Figure 1: Linear Regression for polynomial degree 1

Logically, degree 30 gives the lowest training error as seen in figure 5.

However, the error on the validation set is lowest for the polynomial degree 13 (figure 6).

Training, validation and testing errors as a function of the polynomial degree can be seen in fig 7, which nicely summarizes the previous results.

Regarding figure 7 one can see over-fitting occurs roughly above degree 18. While the training error is further decreasing, validation and testing errors increase rapidly. The model selection yields the degree of 13, which also performs well on the unseen test set. Thus a validation set is important to avoid over-fitting.

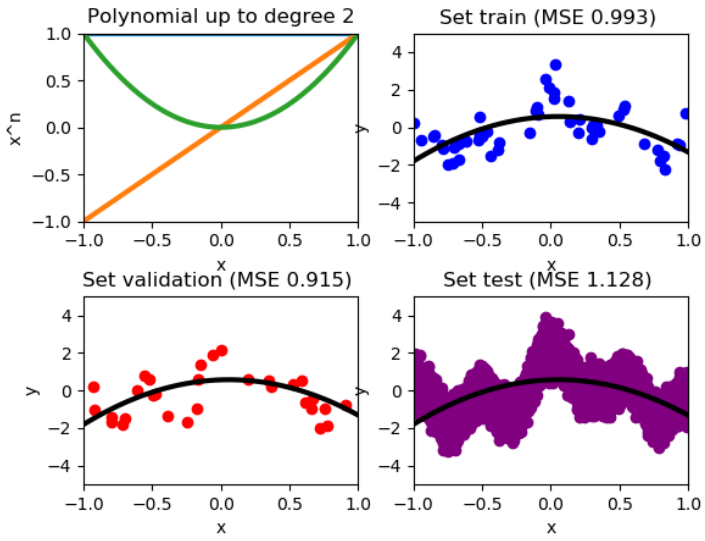

Figure 2: Linear Regression for polynomial degree 2

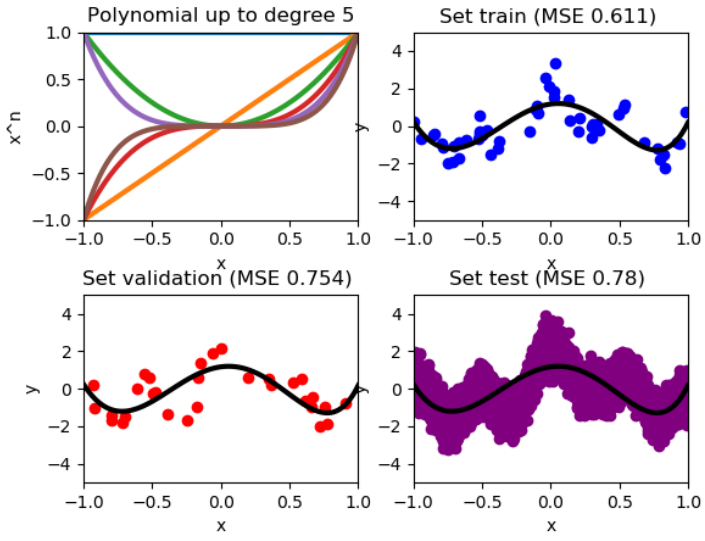

Figure 3: Linear Regression for polynomial degree 5

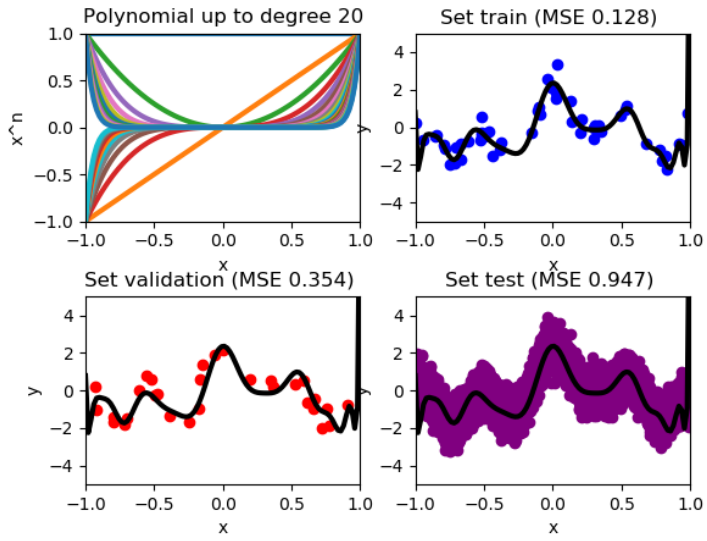

Figure 4: Linear Regression for polynomial degree 20

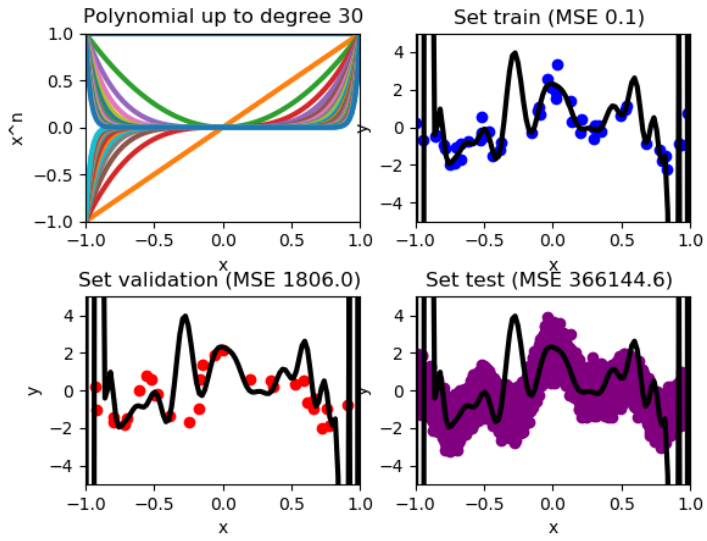

Figure 5: Linear Regression for polynomial degree 30, performs best in training set.

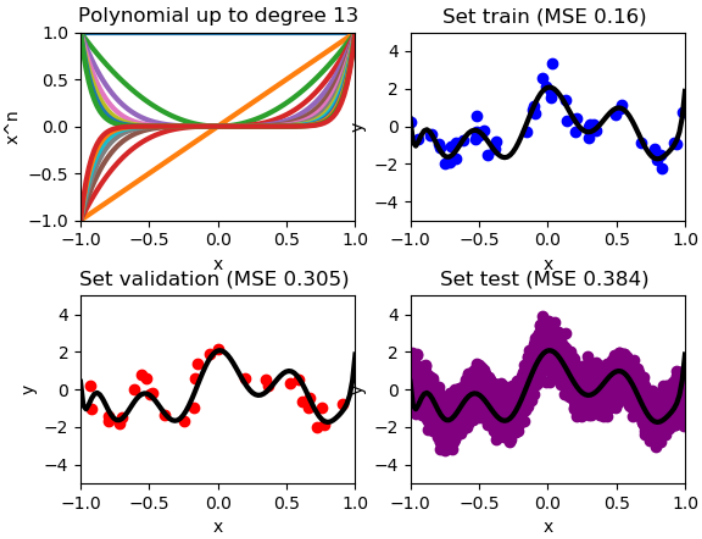

Figure 6: Linear Regression for polynomial degree 13, performs best in validation set.

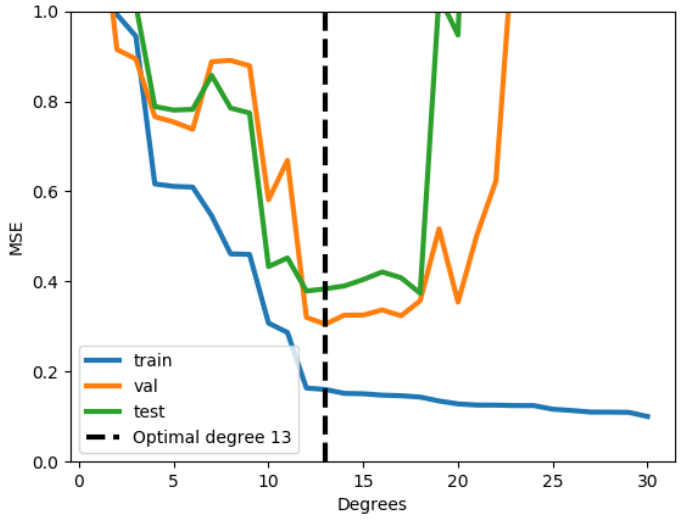

Figure 7: Training, validation and testing errors as a function of the polynomial degree.

1.3 Linear Regression with radial basis functions

As seen in figures 8 to 12 (number of radial basis functions: 1, 2, 5, 20, 40), the higher the number of radial basis functions, the better we can fit our training data.

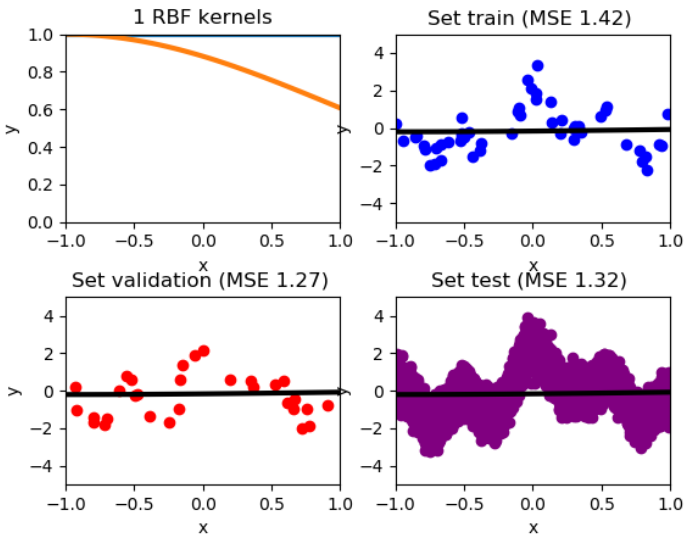

Figure 8: Linear Regression for 1 radial basis function

Logically, 40 RBF kernels result in the lowest training error as seen in figure 12.

However, the error on the validation set is lowest for 9 RBF kernels (figure 13).

Training, validation and testing errors as a function of the number of RBF kernels can be seen in fig 7, which nicely summarizes the previous results.

Regarding figure 14 one can see over-fitting occurs roughly above n = 9. While the training error is further decreasing, validation and testing errors increase rapidly. The model selection yields a number of 9 RBF kernels, which also performs well on the unseen test set. Comparing polynomial features with radial basis functions, we find a smaller test set error for RBF (mse = 0.34, n-centers = 9) than for polynomial regression (mse = 0.384, degree = 13).

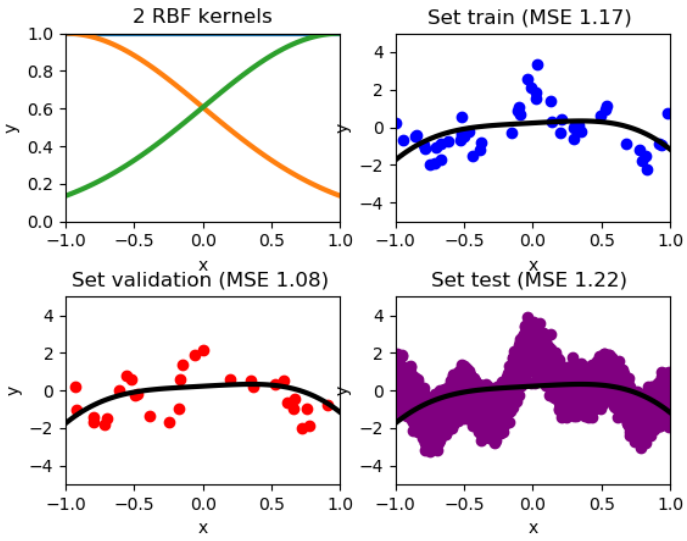

Figure 9: Linear Regression for 2 radial basis functions

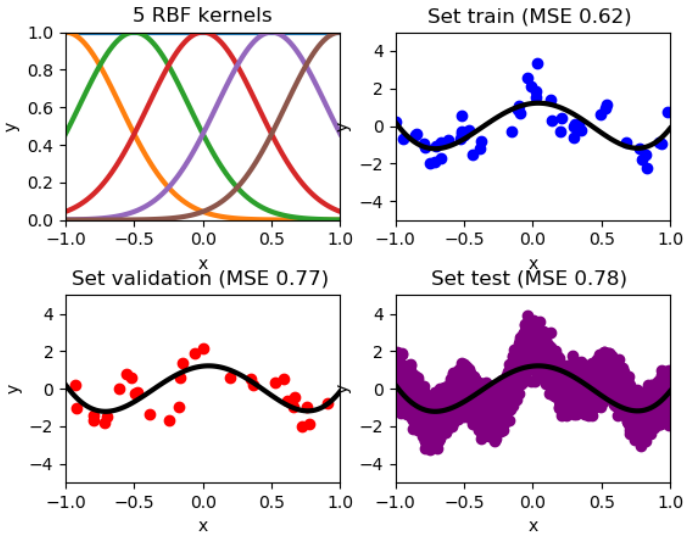

Figure 10: Linear Regression for 5 radial basis functions

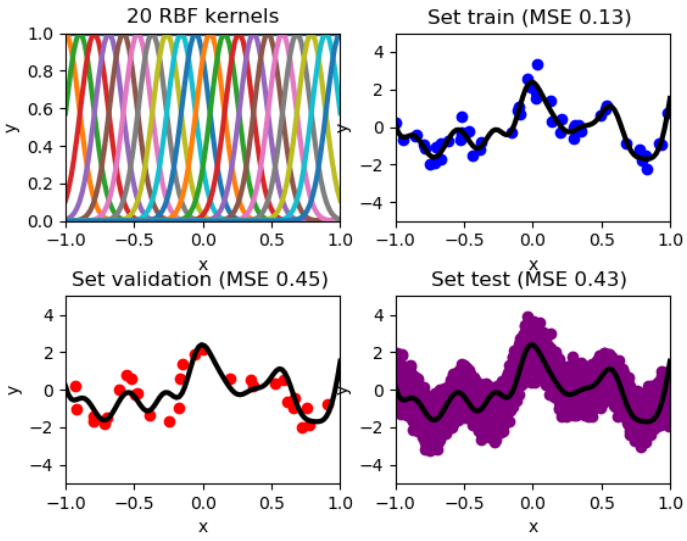

Figure 11: Linear Regression for 20 radial basis functions

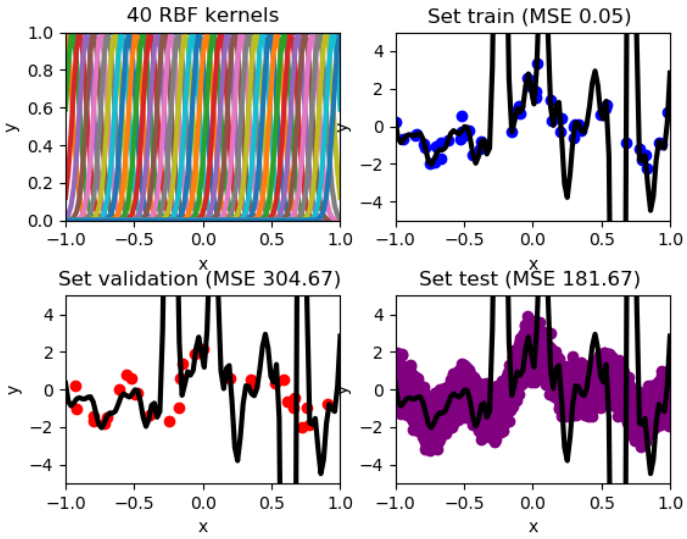

Figure 12: Linear Regression for 40 radial basis functions, performs best in training set.

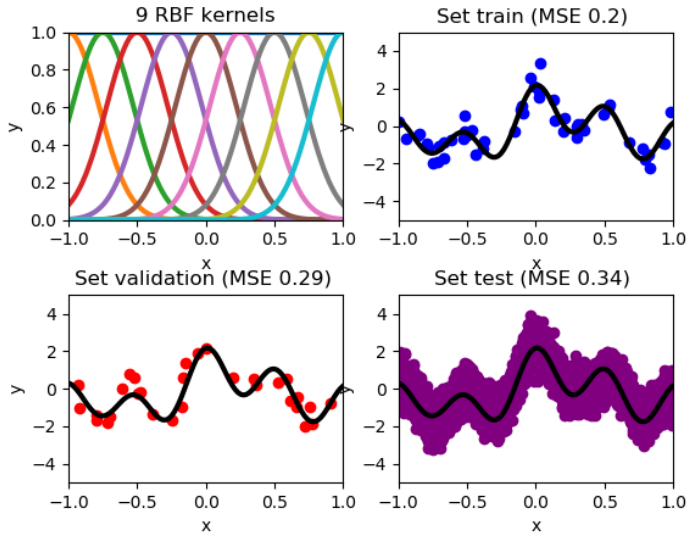

Figure 13: Linear Regression for 9 radial basis functions, performs best in validation set.

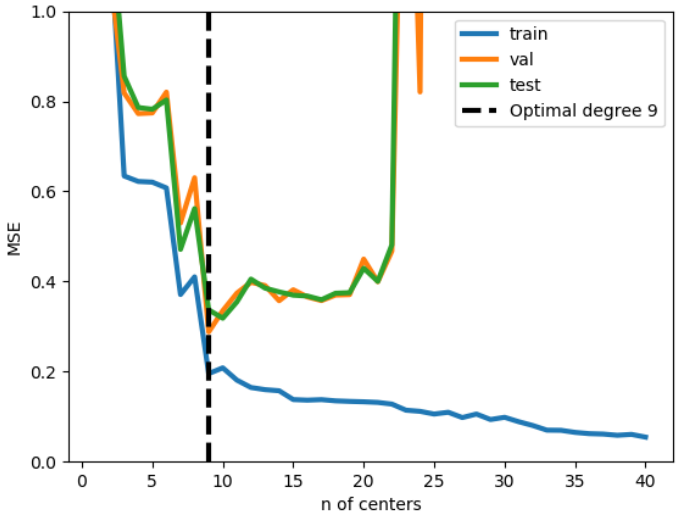

Figure 14: Training, validation and testing errors as a function of the number of radial basis functions.

2 Logistic Regression

2.1 Derivation of Gradient

$$\begin{aligned}\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} &= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}\cdot h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\cdot(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\cdot x_j^{(i)}\right.\\ &\left.+(1-y^{(i)})\frac{1}{1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})}\cdot(-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\cdot(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\cdot x_j^{(i)}\right)\\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}-y^{(i)}h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})+y^{(i)}h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right]\cdot x_j^{(i)}\\ &= \frac{1}{m}\sum_{i=1}^{m}\left[h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})-y^{(i)}\right]\cdot x_j^{(i)}\end{aligned}$$

2.2 Logistic Regression training with gradient descent

2.2.1 Gradient descent

1. The function "check-gradient" in toolbox.py starts with computing a random point x0 and evaluating the cost function and our gradient at this point.
Then it generates a random vector dx and computes the inner product of this random vector and the analytical gradient function evaluated at the point x0 (df-g = np.inner(dx, g0)).
Our gradient calculation was correct, if the finite difference approximation (df = (f(x0 + d * dx) - f0) / d) converges against df-g.
2. Considering figures 15 and 16 we find that after 2000 iterations the algorithm finds better weights $\boldsymbol{\theta}$ for a linear degree $l = 1$ (reduced test set and training error). However, the larger the iteration number the longer it takes the algorithm to converge. Contrary, if the iteration number is too low the algorithm will finish quickly, but produce large test and training errors.

3. If the learning is too low ($\eta = .15$, fig. 18) the gradient descent doesn't converge towards the minimum with the given 200 iterations. For a learning rate of $\eta = 1.5$ seen in fig. 20 we converge towards a lower test error after the 200 iterations. Lastly, with $\eta = 15$ the error decreases quickly but then oscillates around a higher value (fig. 22).

4. Comparing degrees $l \in 1, 2, 5, 15$ we find the lowest test error (0,333) for degree $l = 5$ (fig. 23). For the degrees 5 and 15 we had to increase both learning rate and iteration number to achieve convergence in a reasonable time (figures 24 and 26).

5. A stopping criterium can be formulated as $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) < \delta$ (in python: df(theta) < delta) .

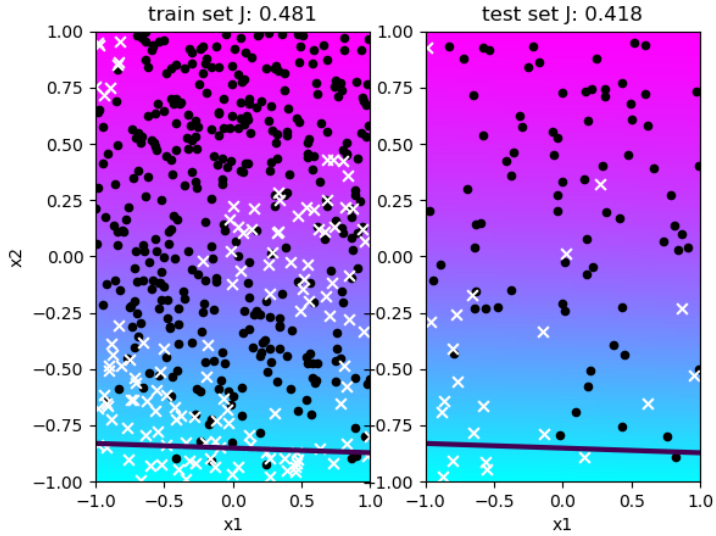

Figure 15: Training and test errors after gradient descent of 20 iterations and degree $l = 1$, $\eta = 1$.

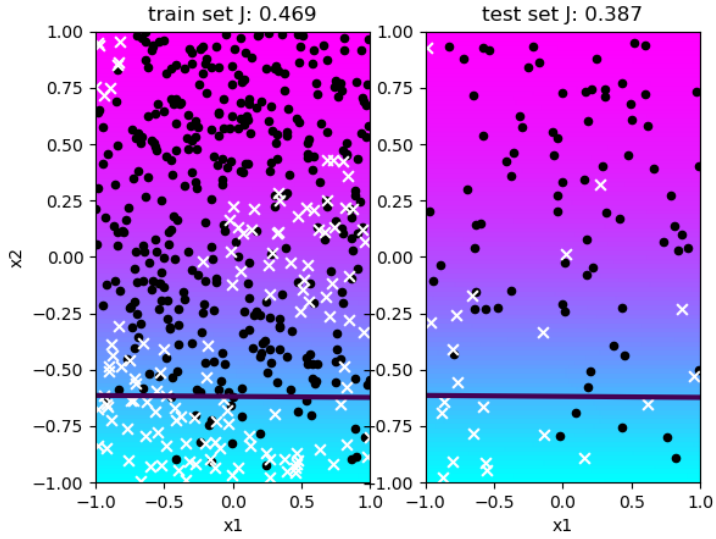

Figure 16: Training and test errors after gradient descent of 2000 iterations and degree $l = 1$, $\eta = 1$.

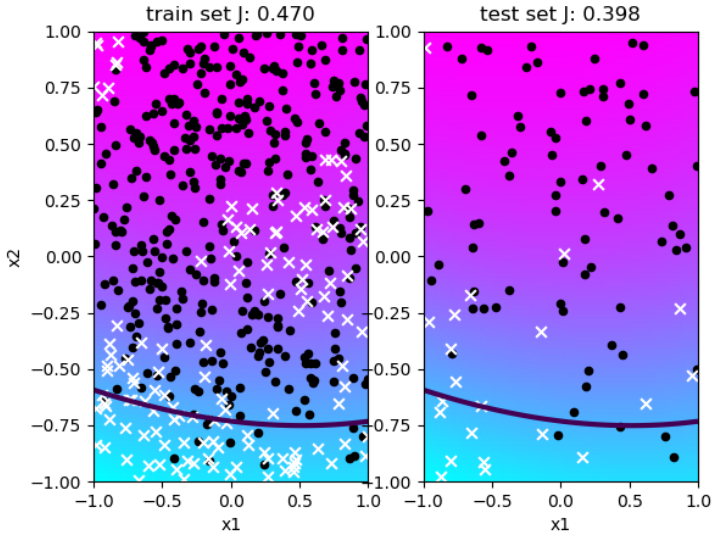

Figure 17: Training and test errors after gradient descent of 200 iterations and degree $l = 2$, $\eta = .15$.

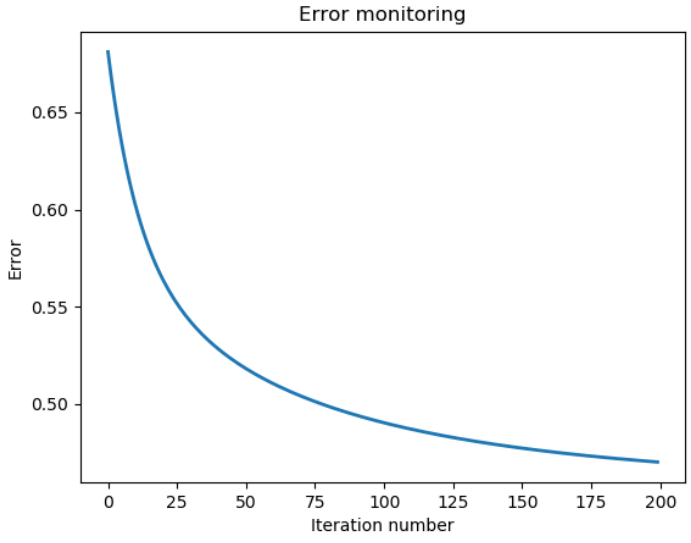

Figure 18: Training error plotted over the 200 iterations for degree $l = 2$, $\eta = .15$.

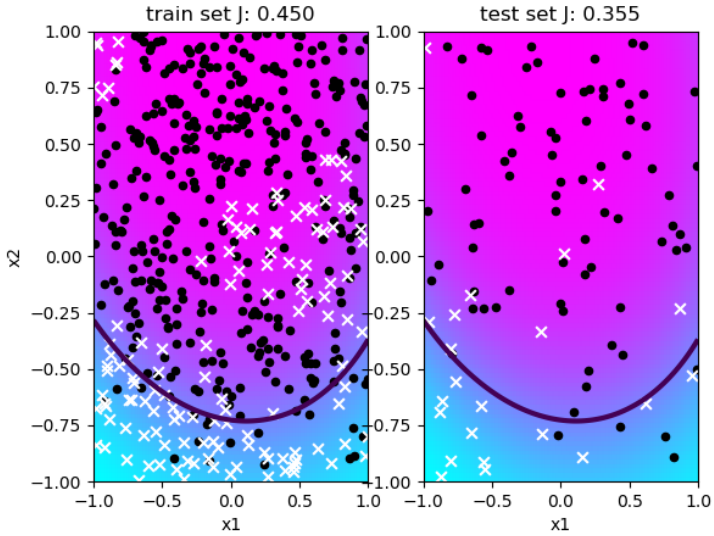

Figure 19: Training and test errors after gradient descent of 200 iterations and degree $l = 2$, $\eta = 1.5$.

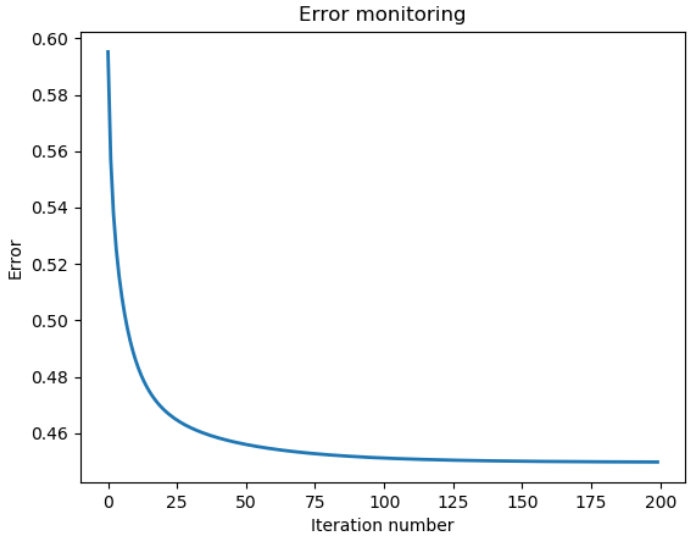

Figure 20: Training error plotted over the 200 iterations for degree $l = 2$, $\eta = 1.5$.

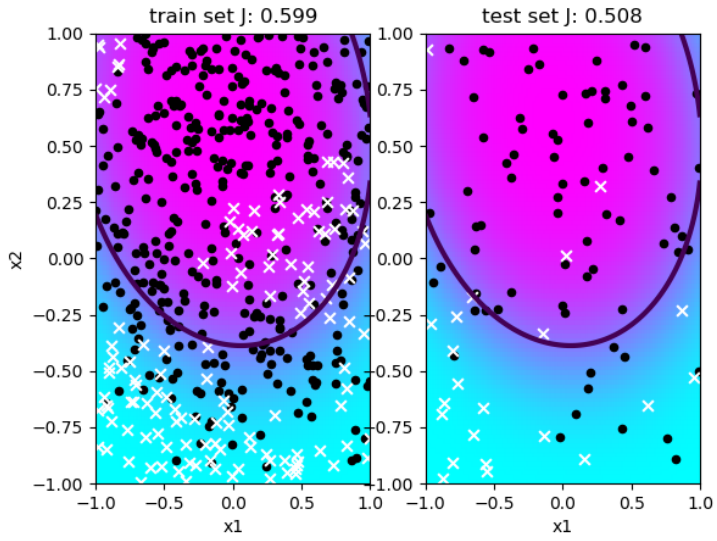

Figure 21: Training and test errors after gradient descent of 200 iterations and degree $l = 2$, $\eta = 15$.

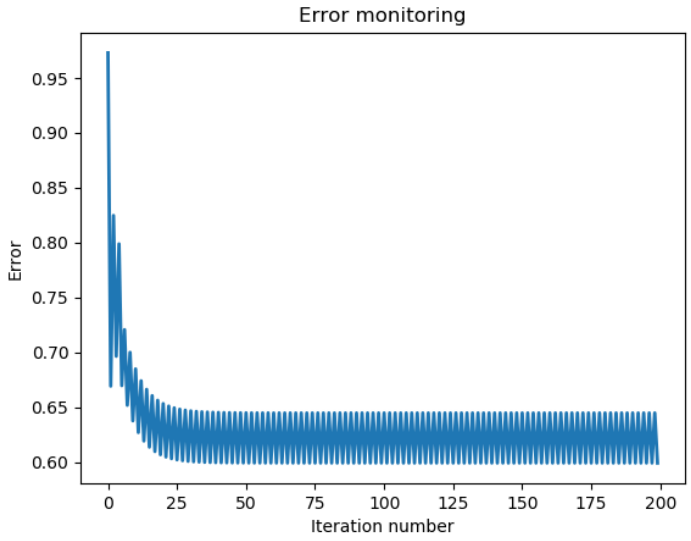

Figure 22: Training error plotted over the 200 iterations for degree $l = 2$, $\eta = 15$.

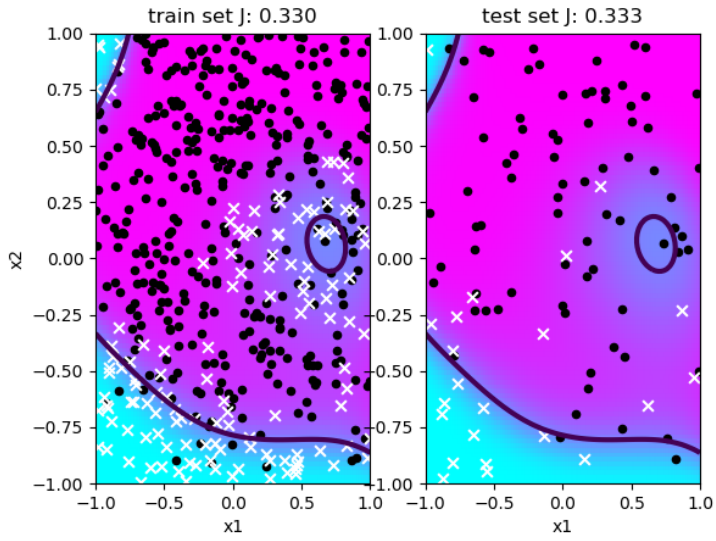

Figure 23: Training and test errors after gradient descent of 300 iterations and degree $l = 5$, $\eta = 10$.

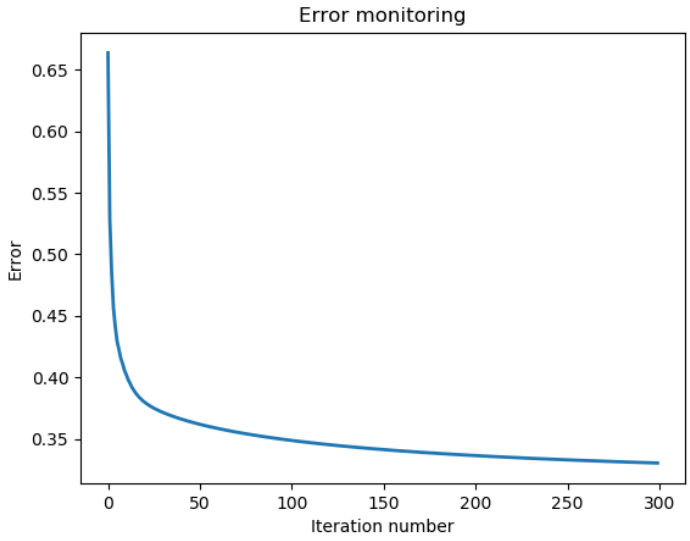

Figure 24: Training error plotted over 300 iterations and degree $l = 5$, $\eta = 10$.

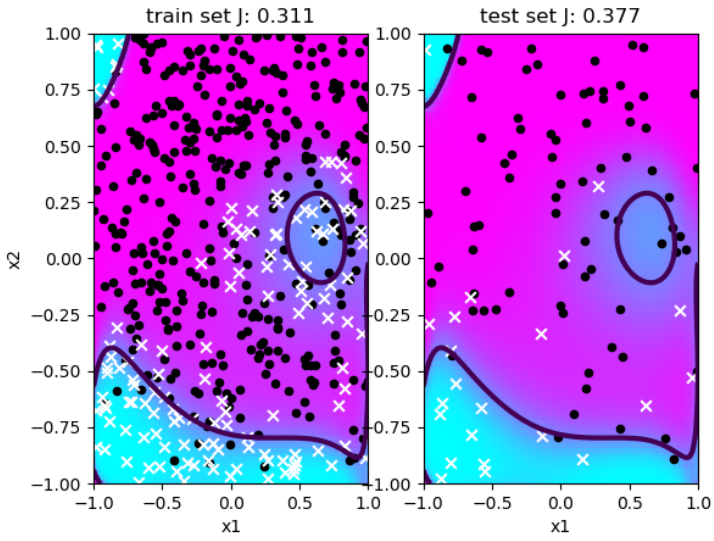

Figure 25: Training and test errors after gradient descent of 500 iterations and degree $l = 15$, $\eta = 10$.

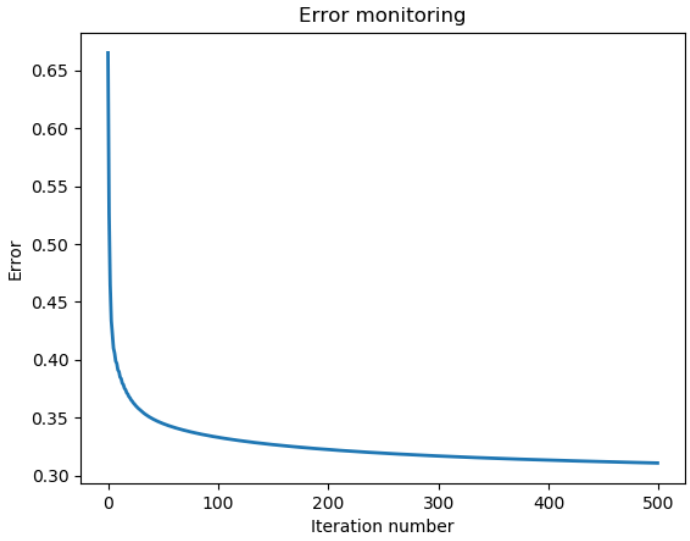

Figure 26: Training error plotted over 500 iterations and degree $l = 15$, $\eta = 10$.