Assignment 1

Computational Intelligence, SS2018

Team Members		
STRUGER	Patrick	01530664
BÖCK	Manfred	01530598
HAUPT	Anna	01432018

1 Linear Regression

1.1 Derivation of Regularized Linear Regression

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

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

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

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

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

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

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 occur when the number of features is large.

1.1.1 Answer the following questions:

- Preliminary questions
  - **why is the design matrix X containing $n+1$ columns and not just n?**

    The first column of the design matrix consists of ones for *convenience* which allow estimation of y-intersept. The following columns contain the x-values associated with their related values of y.

  - **considering the function $J(\theta)$ as a function of all the variables $\theta_0, \theta_1 ...$, what is the definition of the gradient $\frac{\partial J(\theta)}{\partial \theta}$? This gradient is actually a vector, what is its dimension?**

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

Gradient definition

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{m} * \frac{\partial (X\theta - y)^T\ (X\theta - y)}{\partial \theta} + \frac{2\lambda}{m} * \frac{\partial (\theta)^T \theta}{\partial \theta} \tag{1}$$

$$\Rightarrow \frac{2}{m} * (X\theta - y)^T * \frac{\partial (X\theta - y)}{\partial \theta} + \frac{2\lambda}{m} * (\theta)^T * \frac{\partial (\theta)}{\partial \theta} \tag{2}$$

$$\Rightarrow \frac{2}{m} * (X\theta - y)^T * X + \frac{2\lambda}{m} * (\theta)^T \tag{3}$$

The gradient vector is formed with the partial derivatives. The dimension of the gradient is the amount of columns in the given matrix X (if $X = m * n$ then gradient of the form $1 * n$).

  - **What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix?**

    The Jacobian Matrix is a $m\ *\ n$ matrix of a differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ formed with the partial derivatives. The gradient matrix $1\ *\ n$ is used for functions of the form $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

- **the hypothesis $X\theta$ generates predictions of the output vector y, each coordinate of the vector corresponds to a different data point. What is the dimension of the vector $X\theta$? We also use the notation $\frac{\partial X(\theta)}{\partial \theta}$ to define the Jacobian matrix. What is the dimension of the Jacobian matrix $\frac{\partial X(\theta)}{\partial \theta}$? In this particular case, what is $\frac{\partial X(\theta)}{\partial \theta}$ equal to?**

  The vector $X\theta$ has the same dimension as the provided output vector $(m * 1)$ which refers to the number of training examples.
  The dimension of the Jacobian matrix $\frac{\partial X(\theta)}{\partial \theta}$ is m *(n+1) and this is equal to the design matrix X.

- Using the following hints, show that the optimal parameters which minimize the regularized linear regression cost (3) are given by, $\theta^* = (X^T X + \lambda I)^{-1} X^T y$, where I denotes the identity matrix of dimension $(n+1) \times (n+1)$.

$$\frac{\partial J(\theta)}{\partial \theta} = 0 \tag{4}$$
$$\Rightarrow \frac{2}{m} * (X\theta - y)^T * X * \frac{2\lambda}{m} * (\theta)^T = 0 \tag{5}$$
$$\Rightarrow \frac{2}{m}((X\theta - y)^T * X + \lambda * (\theta)^T)) = 0 \tag{6}$$
$$\Rightarrow ((X\theta - y)^T * X + \lambda * (\theta)^T)) = 0 \tag{7}$$
$$\Rightarrow (X\theta - y)^T * X = -\lambda(\theta)^T \tag{8}$$
$$\Rightarrow (X^T\theta^T - y^T) * X = -\lambda(\theta)^T \tag{9}$$
$$\Rightarrow (X^T\theta^T X - y^T X) = -\lambda(\theta)^T \tag{10}$$
$$\Rightarrow (X^T\theta^T X + \lambda(\theta)^T) = y^T X \tag{11}$$
$$\Rightarrow \theta^T(X^T X + \lambda) = y^T X \tag{12}$$
$$\Rightarrow \theta^T(X^T X + \lambda)(X^T X + \lambda)^{-1} = (X^T X + \lambda)^{-1} y^T X \tag{13}$$
$$\Rightarrow \theta^T = (X^T X + \lambda)^{-1} y^T X \tag{14}$$
$$\Rightarrow \theta^* = (X^T X + \lambda I)^{-1} * y X^T \tag{15}$$
$$\tag{16}$$

- (Optional) In practice the matrix $X^T X$ might not be invertible. Using the spectral theorem (eigenvalue decomposition of symmetric matrices), what can we say about the eigenvalues of $X^T X$? Explain why the matrix $X^T X + \lambda I$ is invertible for any small positive λ.

1.2 Linear Regression with polynomial features

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)

$$\phi_0 = 1, \phi_1 = x, \phi_2 = x^2, ..., \phi_n = x^n$$

- **Plot your results for each of the following polynomial degrees: 1, 2, 5, 20.**

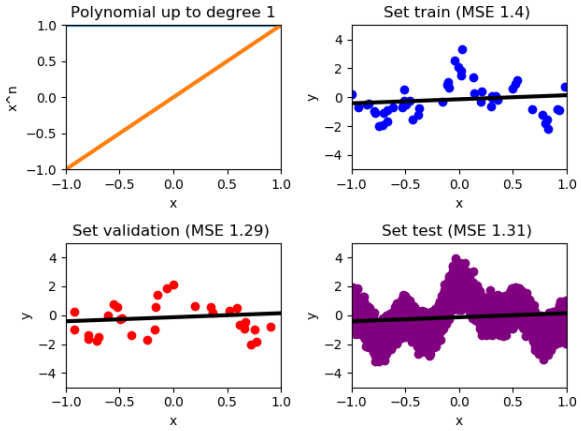

Figure 1: Plot with polynomial degree of 1

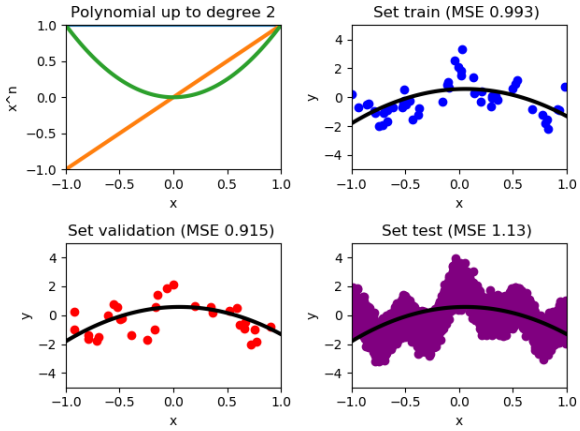

Figure 2: Plot with a polynomial degree of 2

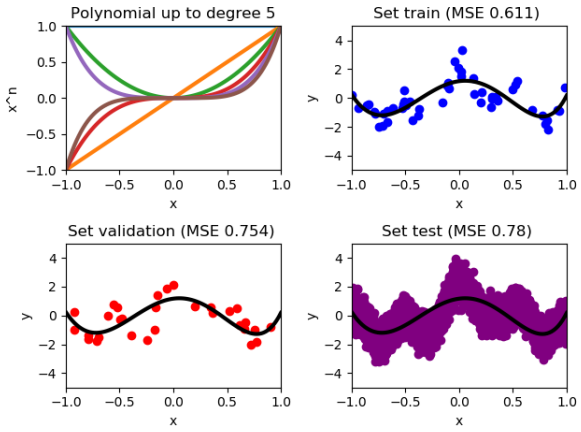

Figure 3: Plot with a polynomial degree of 5

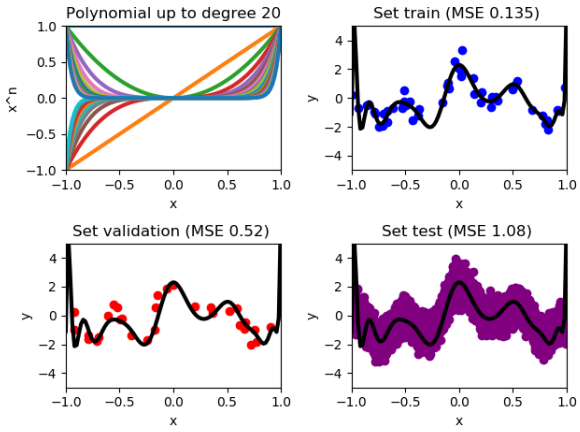

Figure 4: Plot with a polynomial degree of 20

- **Report which degree $\in \{1, ..., 30\}$ gives the lowest training error. Plot the results for that degree. Report the test error.**
  The lowest Trainings Error produced the plot with a degree of 30 it was only 0.1257193081096509. The related Testing Error was 14.888677131456754.

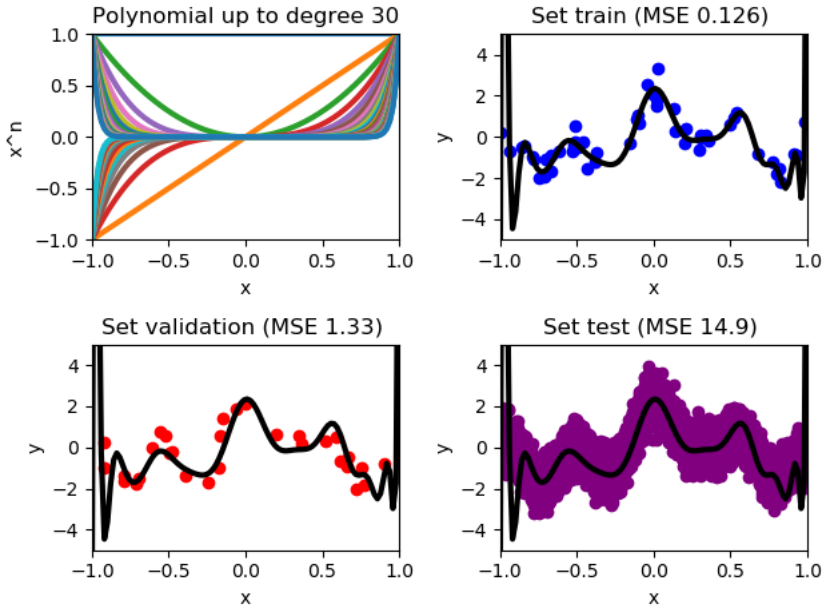

Figure 5: Lowest Trainings Error

- **Report which degree $\in \{1, ..., 30\}$ gives the lowest validation error. Plot the results for that degree. Report the test error.**
  The lowest Validation Error produced the plot with a degree of 13 it was only 0.30522863334865147. The related Testing Error was 0.38370160426301475.

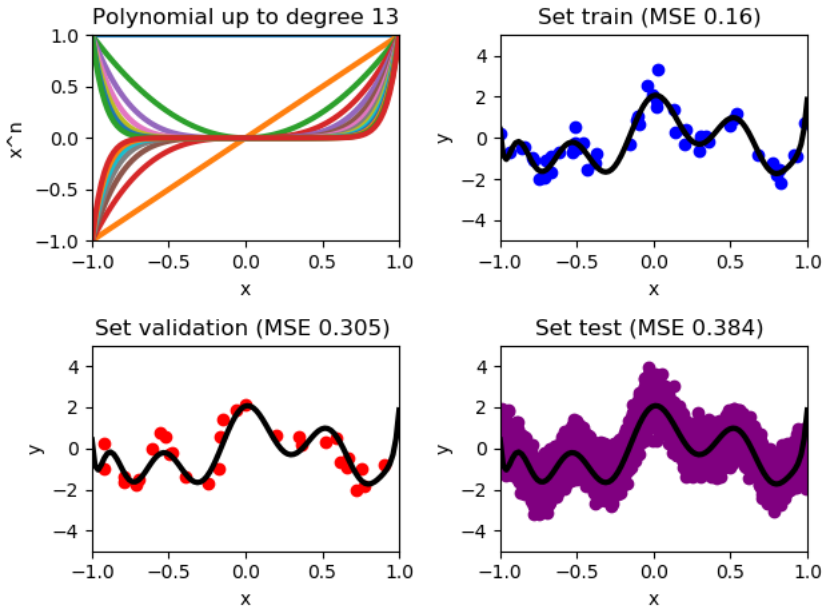

Figure 6: Lowest Validation Error

- **Plot training, validation and testing errors as a function of the polynomial degree.**

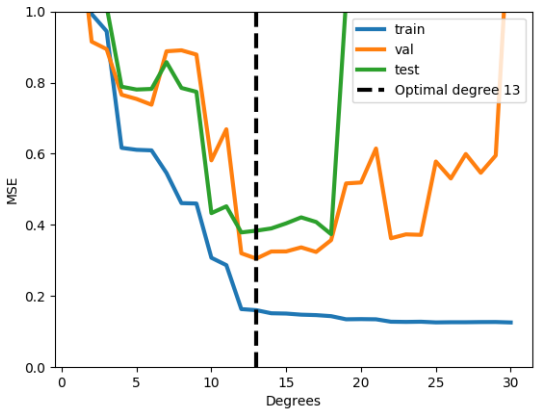

Figure 7: Training, Validation & Testing Errors as functions of the polynomial degree

- **Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?**
  We found out that at first the error for Trainings, Testing & Validation sets is reduced more or less equally but at a certain degree the errors in the training set still decreases but only minimally, whereas the errors in the Testing & Validation sets drastically deteriorated that phenomenon is called *over-fitting* It is important to use a validation set to get an ideal mean value for the used degree so that the error for all data is kept as small as possible to be precise to find the hypothesis which produces the best results .

1.3 Linear Regression with radial basis functions

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)

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

- **Plot your results for each of the following degrees: 1, 2, 5, 20.**

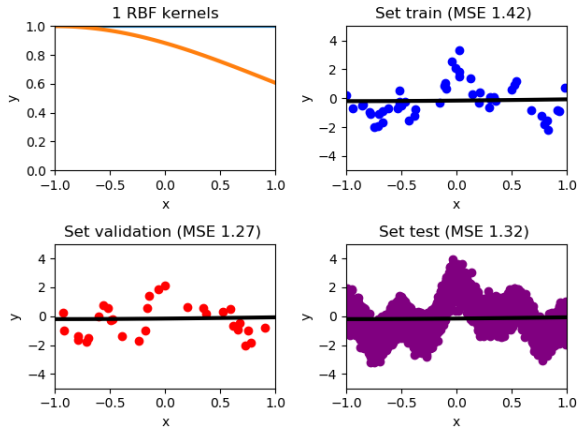

Figure 8: Radial Basis Function with a degree of 1

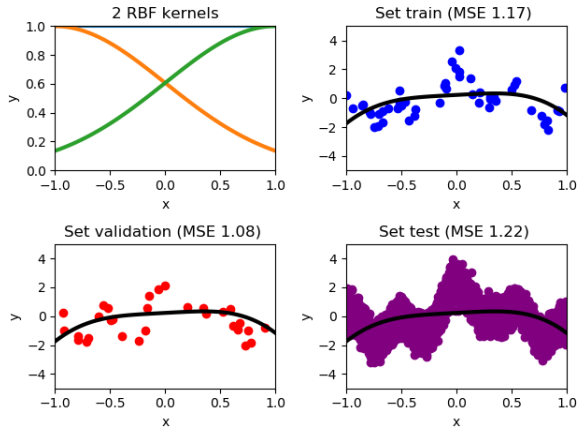

Figure 9: Radal Basis Function with a degree of 2

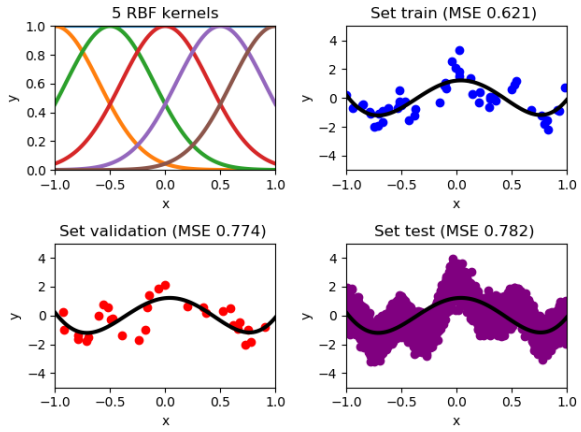

Figure 10: Radial Basis Function with a degree of 5

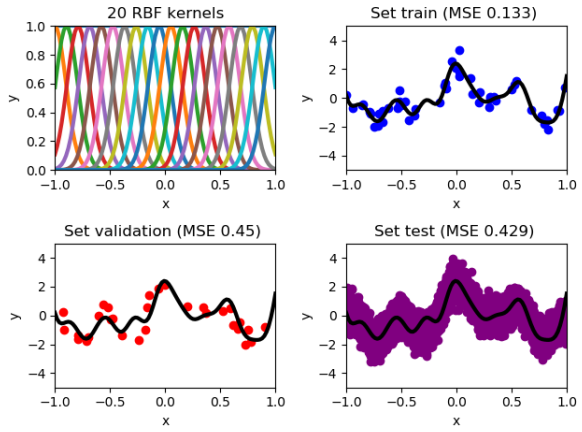

Figure 11: Radial Basis Function with a degree of 20

- **Report which number of RBFs $\in \{1, ..., 40\}$gives the lowest training error. Plot the results for that degree. Report the test error.**
  The lowest Trainings Error produced the plot with a degree of 40 it was only 0.054452916301053075. The related Testing Error was 181.6718279531847.

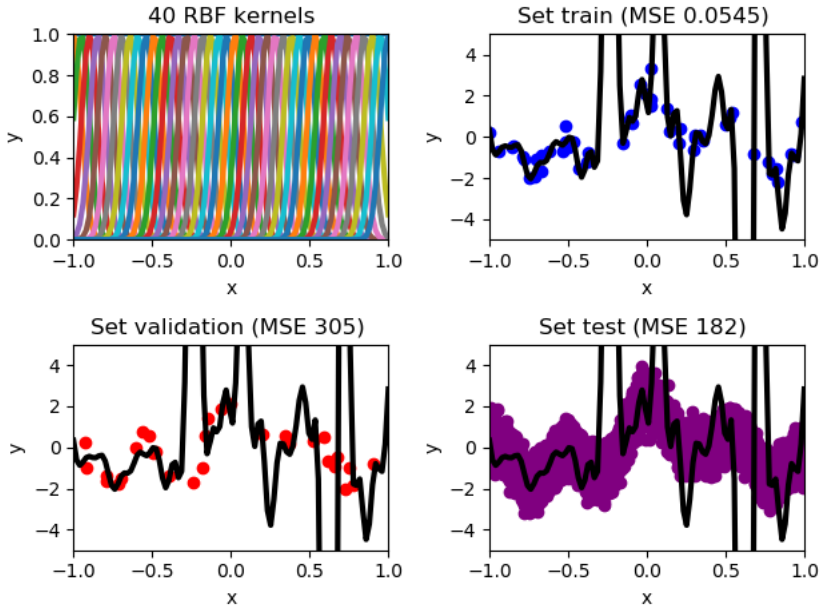

Figure 12: Lowest RBF Training Error

- **Report which number of RBFs $\in \{1, ..., 40\}$gives the lowest validation error. Plot the results for that degree. Report the test error.**
  The lowest Validation Error produced the plot with a degree of 9 it was only 0.2879598418720983. The related Testing Error was 0.33701300824089064.

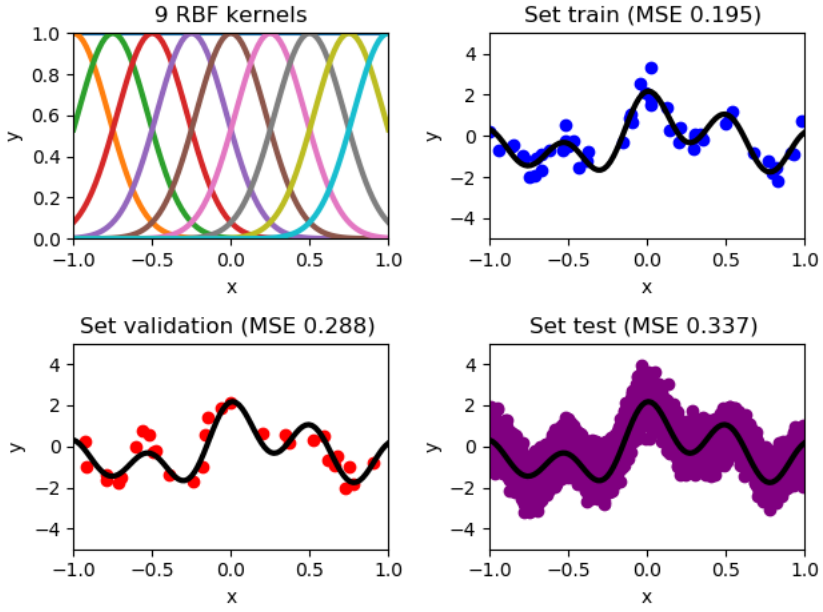

Figure 13: Lowest RBF Validation Error

- **Plot training, validation and testing errors as a function of the degree.**

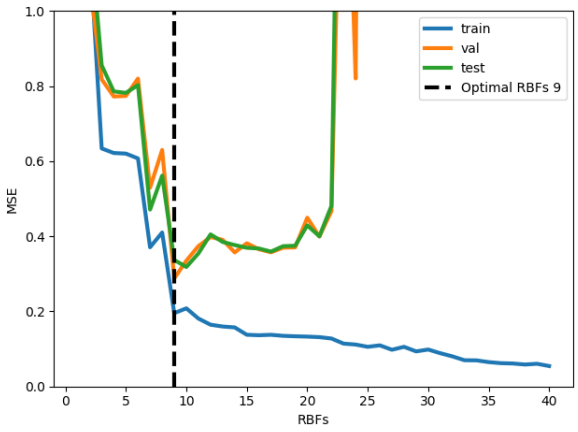

Figure 14: Training, Validation & Testing errors as a function of the degree

- **Briefly describe and discuss your findings in your own words. Is the polynomial or the RBF model better?**
  We found out that at first the error for Trainings, Testing & Validation sets is reduced more or less equally until 9 RBFs. From this moment the errors in the training set still decreases, whereas the errors in the Testing & Validation sets begin to rise again. From RBF 22 the errors for the Testing & Validation sets begin to increase immensely fast.
  We got to the conclusion that the Radial Basis Function model superior to the Polynomial Function model because we got in almost all cases lower testing errors using the Radial Basis Function model. For example the lowest validation error had a testing error of 0,337 using the RBF model and a testing error of 0,384 using the Polynomial Function model.

2 Logistic Regression

2.1 Derivation of Gradient

The logistic regression hypothesis function is given by,

$$h_\theta(x) = \sigma(x^T\theta) = \sigma(\theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n)$$

with the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$, parameters $\sigma = (\sigma_0, \sigma_1, ..., \sigma_n)^T$, and input vector $x = (x_0, x_1, ..., x_n)^T$. By convention the constant feature x_0 is fixed to $x_0 = 1$. With $x^{(i)} = (x_0^{(i)}, x_1^{(i)}, ..., x_n^{(i)})^T, x_0^{(i)} = 1$ and $log(\cdot)$ referring to the natural logarithm, the logistic regression cost function can be written as,

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

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

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

$Hint: \frac{\partial\sigma(z)}{\partial\sigma} = \sigma(z)\cdot\left(1-\sigma(z)\right)$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\left(-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1+y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)\right)\right) \tag{17}$$

Since only the log functions contain θ, we just have to do the derivation of them as follows:

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{\partial}{\partial\theta_j}\log\left(h_\theta(x^{(i)})\right) + (1+y^{(i)})\frac{\partial}{\partial\theta_j}\log\left(1-h_\theta(x^{(i)})\right)\right) \tag{18}$$

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{h\theta(x^{(i)})}\frac{\partial h\theta(x^{(i)})}{\partial\theta_j} + (1+y^{(i)})\frac{1}{1-h\theta(x^{(i)})}\frac{\partial(1-h\theta(x^{(i)}))}{\partial\theta_j}\right) \tag{19}$$

In the next step we focuse on the partial derivative $\frac{\partial(1-h\theta(x^{(i)}))}{\partial\theta_j}$

$$\frac{\partial(1-h\theta(x^{(i)}))}{\partial\theta_j} = \frac{\partial}{\theta_j}\left(1-\frac{1}{1+e^{-x^{(i}\theta^{(i)}}}\right)*(x^{(i)}) = \frac{e^{-x^{(i)}\theta^{(i)}}*(x^{(i)})}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)^2}*(x^{(i)}) = \tag{20}$$

$$\frac{\left(-1+1+e^{-x^{(i)}\theta^{(i)}}\right)}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)^2}*(x^{(i)}) = \left(\frac{1}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)} - \frac{1}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)^2}\right)*(x^{(i)}) \tag{21}$$

$$= \frac{1}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)}*\left(1-\frac{1}{\left(1+e^{-x^{(i}\theta^{(i)}}\right)}\right)*(x^{(i)}) = \left(h\theta(x^{(i)})*(1-h\theta(x^{(i)}))\right)*(x^{(i)}) \tag{22}$$

In the next step we compute the partial derivative $\frac{\partial h\theta(x^{(i)})}{\partial \theta_j}$ with the same scheme:

$$\frac{\partial(h\theta(x^{(i)}))}{\partial \theta_j} = \frac{\partial}{\theta_j}\Big(\frac{1}{1+e^{-x^{(i}\theta^{(i)}}}\Big) * (x^{(i)}) \tag{23}$$

$$= -\frac{e^{-x^{(i)}\theta^{(i)}} * (x^{(i)})}{\Big(1+e^{-x^{(i}\theta^{(i)}}\Big)^2} * (x^{(i)}) = ... = \Big(h\theta(x^{(i)}) * (h\theta(x^{(i)}) - 1)\Big) * (x^{(i)}) \tag{24}$$

Finally the partial derivatives are inserted in the original function

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\left(\frac{y^{(i)}}{h\theta(x^{(i)})}\Big(h\theta(x^{(i)}) * (h\theta(x^{(i)}) - 1)\Big) * (x^{(i)}) + \frac{(1+y^{(i)})}{1-h\theta(x^{(i)})}\Big(h\theta(x^{(i)}) * (1-h\theta(x^{(i)}))\Big) * (x^{(i)})\right) \tag{25}$$

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\Big((h\theta(x^{(i)}) - 1)\Big) * (x^{(i)}) + (1+y^{(i)})\Big(h\theta(x^{(i)})\Big) * (x^{(i)})\right) \tag{26}$$

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x^{(i)} - y^{(i)}x^{(i)}h\theta(x^{(i)}) - x^{(i)}h\theta(x^{(i)}) + y^{(i)}x^{(i)}h\theta(x^{(i)})\right) \tag{27}$$

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)}x^{(i)} - x^{(i)}h\theta(x^{(i)})\Big) \tag{28}$$

$$\Rightarrow -\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)} - h\theta(x^{(i)})\Big) * x^{(i)} \tag{29}$$

$$\Rightarrow \frac{1}{m}\sum_{i=1}^{m}\Big(h\theta(x^{(i)}) - y^{(i)}\Big) * x^{(i)} \tag{30}$$

2.2 Logistic Regression training with gradient descent and *scipy.optimize*

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, and to compare different optimization techniques for this task. The optimization techniques that should be compared are gradient descent with a constant learning rate, gradient descent with an adaptive learning rate (optional), and the advanced optimization method implemented by the Scipy function minimize. 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 $a, b \in Z$ and $a + b \leq l$. For example, for degree $l = 3$ the features should be,

$$\phi_0 = 1,\ \phi_1 = (x_1)^1, \phi_2 = (x_2)^1,\ \phi_3 = (x_1)^2, \phi_4 = (x_1)(x_2), \phi_5 = (x_2)^2,$$
$$\phi_6 = (x_1)^3, \phi_7 = (x_1)^2(x_2)^1, \phi_8 = (x_1)^1(x_2)^2, \phi_9 = (x_2)^3$$

2.3 Gradient descent (GD)

1. The function check_gradient in toolbox.py is here to test if your gradient is well computed. Explain what it is doing.
   Function check_gradient in toolbox.py checks if the gradient & cost functions have been computed correctly. It tries for different variations if the gradient is with finite difference well approximated. If the gradient is well approximated a message about the success of the check is printed. On the other hand if something is not right with the cost function or the gradient an exception is raised.

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.

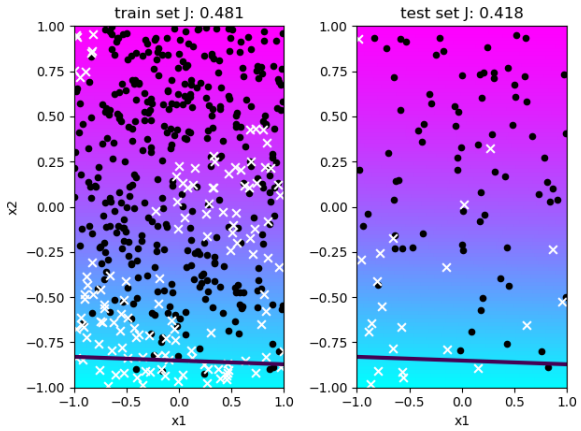

Figure 15: GD for 20 Iterations & Learning Rate of 1

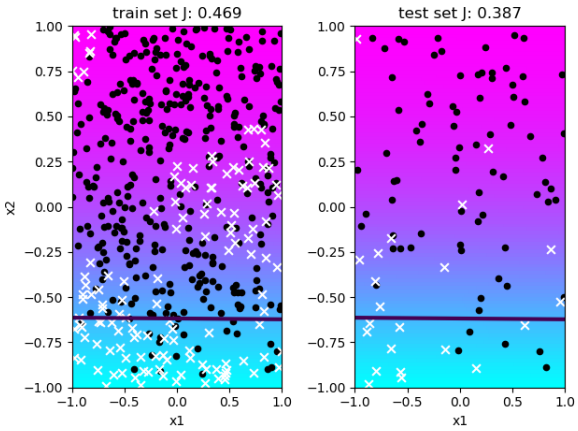

Figure 16: GD for 2000 Iterations & Learning Rate of 1

For 20 iterations the training error was 0.481 and the testing error was 0.418 in contrast the training & testing error for 2000 iteration was 0.469 & 0.387 respectively
With a smaller iteration we are getting a higher error rate while more iterations get a lower error rate, but with minimal difference. The conclusion is that increasing the iteration, the error rate is optimized just by a minimum but more time consuming. The iterations should be neither too low nor too high because if the iteration number is too high then although the result might be correct it just takes too long to compute. On the other side they are too low the learning rate has not much time/chances to influence on the decision boundaries

3. For degree $l = 2$ run GD for 200 iterations and learning rates of $\eta = .15$, $\eta = 1.5$ and $\eta = 15$.. 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.

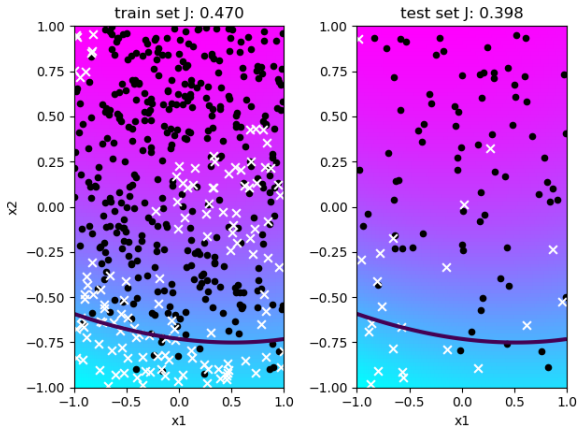

Figure 17: GD for 200 Iterations & Learning Rate of 0.15

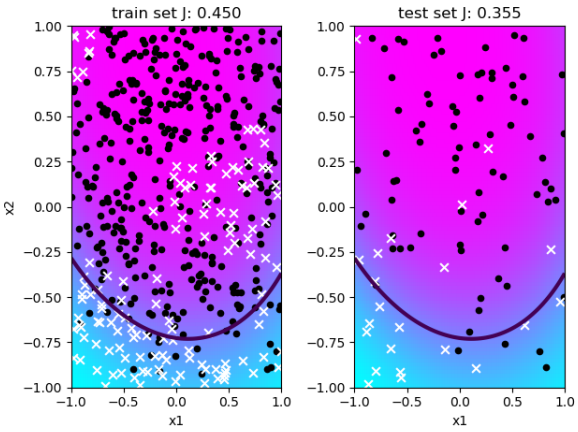

Figure 18: GD for 200 Iterations & Learning Rate of 1.5

For 200 iterations and a learning rate of 0.15 the training error was 0.470 and the testing error was 0.398. With a learning rate of 1.5 the training error was 0.450 and the testing error was 0.355. And with a learning rate of 15 the training error was 0.599 and the testing error was 0.508.
If the learning rate is too large the algorithm may overshoot the optimal point, so you end up at a non-optimal point with a higher error, on the other hand if it's to small than we have

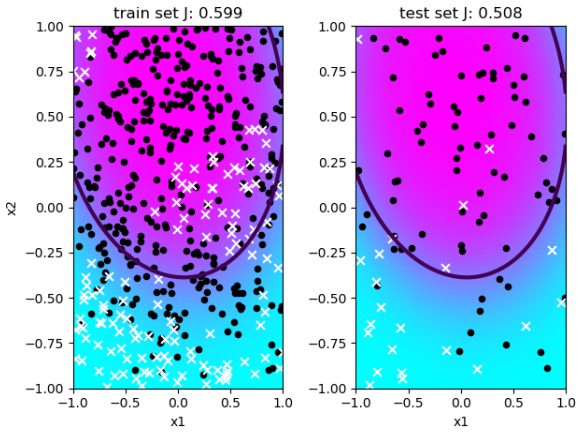

Figure 19: GD for 200 Iterations & Learning Rate of 15

much more iterations to find the optimal point. Finding the optimal learning rate impacts directly the efficiency and reliability of the algorithm.

4. Identify reasonably good pairs of values for the number of iterations and learning rate for each degree in $l \in 1, 2, 5, 15$ 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 appropriate to fit the given data.

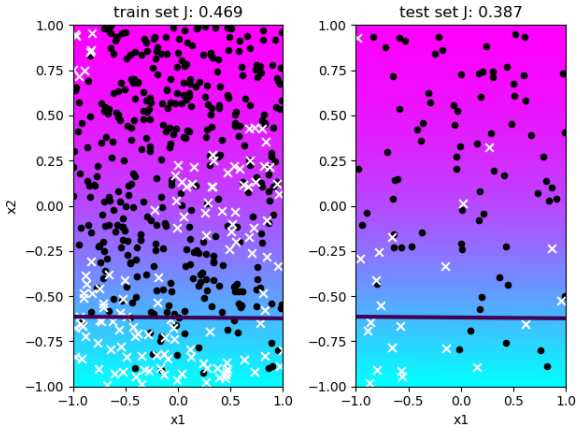

Figure 20: Gradient Descend for degree 1, 250 iterations & and a learning rate of 2

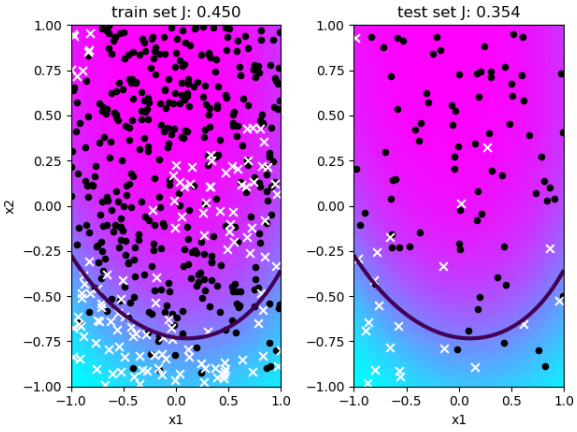

Figure 21: Gradient Descend for degree 2, 250 iterations & and a learning rate of 2

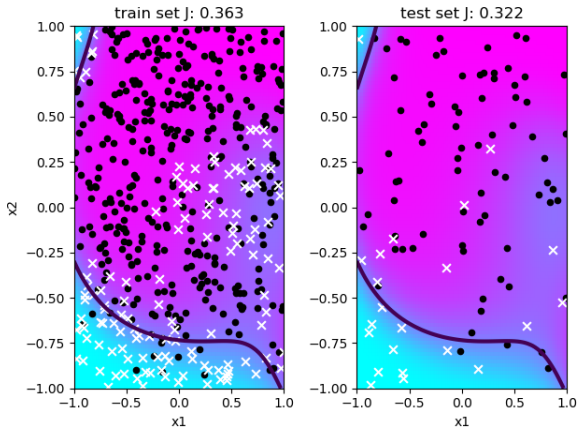

Figure 22: Gradient Descend for degree 5, 250 iterations & and a learning rate of 2

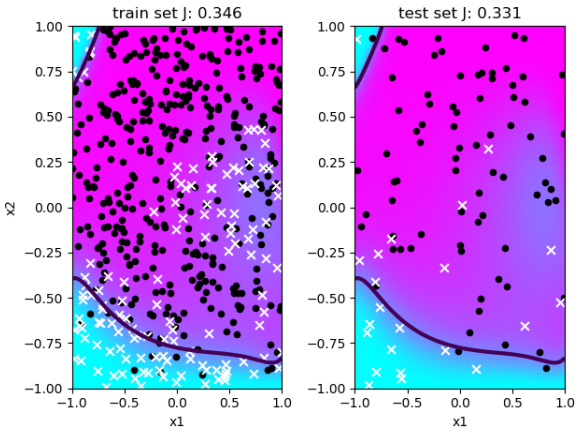

Figure 23: Gradient Descend for degree 15, 250 iterations & and a learning rate of 2

By experimenting with different parameters we found out that a reasonably good pair of values for the number of iterations and learning rate for each degree in $l \in 1, 2, 5, 15$ are 250 iterations & a learning rate of 2. Using a fixed number of iteration and a fixed learning rate we got for degree 1 0.469 Trainings & 0.387 Testing errors. For degree 2 we got 0.450 Trainings & 0.354 Testing errors. For degree 5 we got 0.363 Trainings & 0.322 Testing errors. And for degree 15 we got 0.346 Trainings & 0.331 Testing errors. Due to a relatively low iteration number the computing was quite fast.
With 250 iterations and a learning rate of 2, the degree 5 seems to be the most appropriate to fit the given data as we achieved with this degree the lowest testing error: 0.322.

5. Describe a possible stopping criterium using the gradient of the cost function ?
   A possible stopping criterium could be to stop as soon as the function gets close to zero or is equal zero.

2.3.1 Adaptative gradient descent (GDad)

Fill the following blocks in the code you are provided:

- In gradient_descent.py implement the adaptive learning rate as follows: After every update check whether the cost increases or decreases.
  - If the cost increased, reject the update (go back to the previous parameter setting) and multiply the learning rate by 0.7.
  - If the cost decreased, accept the update and multiply the learning rate by 1.03.

  The iteration count should be increased after every iteration even if the update was rejected

1. Run GDad for varying degrees $l \in 1, 2, 5, 15$ (with zero initialization of parameters, 1000 iterations, and initial learning rate $\eta = 1$). Report training and test errors, final learning rates and plot the hypothesis obtained in each case with function plot_logreg in file logreg_toolbox.py.

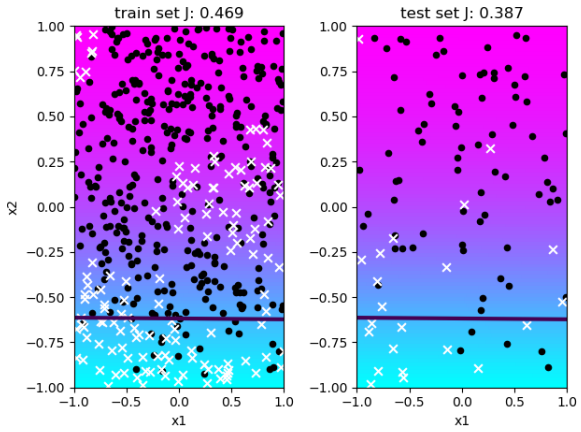

Figure 24: Hypothesis for GDad for 1000 Iterations, a Learning Rate of 1 & a Degree of 1

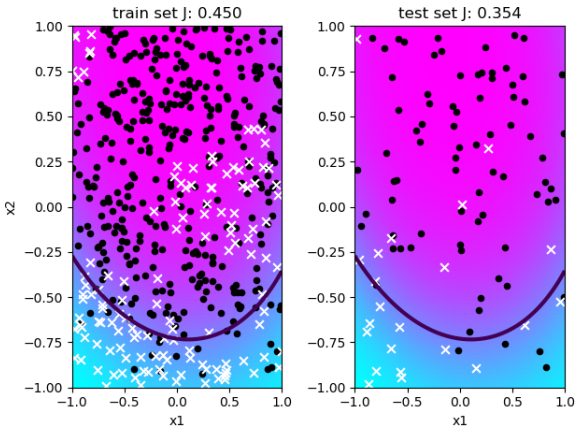

Figure 25: Hypothesis for GDad for 1000 Iterations, a Learning Rate of 1 & a Degree of 2

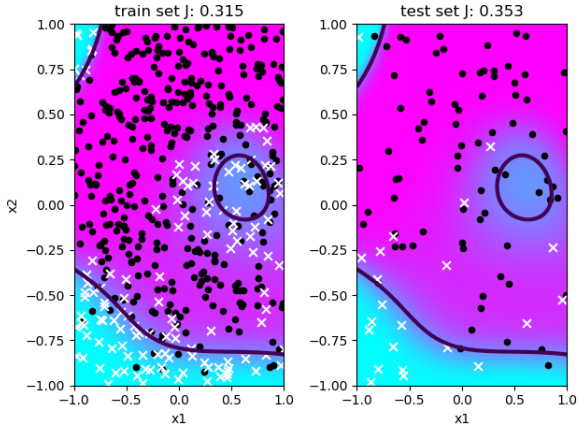

Figure 26: Hypothesis for GDad for 1000 Iterations, a Learning Rate of 1 & a Degree of 5

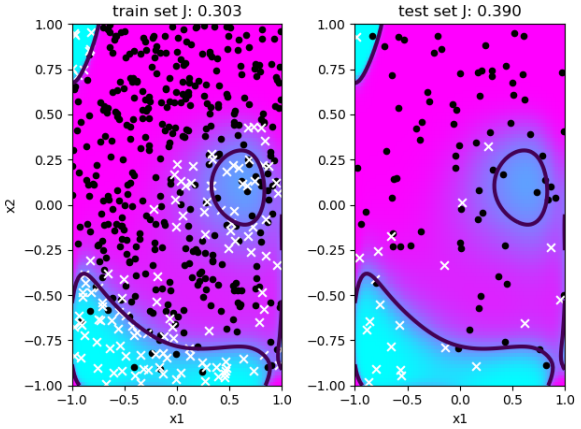

Figure 27: Hypothesis for GDad for 1000 Iterations, a Learning Rate of 1 & a Degree of 15

With different degrees but constant iteration of 1000 and learning rate of 1 different testings & trainings were computed as shown in the table below.

Degree	Training Error	Testing Error	Final Learning Rate
1	0.469	0.387	3.42e-09
2	0.450	0.354	1.6e-08
5	0.315	0.353	12.5
15	0.303	0.390	12.5

2. For each degree compare with the non-adaptive gradient descent variant. In particular, is the evolution of the learning rate numerically coherent with your previous guess of the optimal learning rate ? Discuss why GDad can be useful.

If we compare the results for each degree of the adaptive gradient descent to the results of

Degree	Training Error	Testing Error
1	0.469	0.387
2	0.450	0.354
5	0.349	0.323
15	0.333	0.342

the non-adaptive gradient descent it clearly shows that for low degrees for example a degree

of 1 or 2 trainings & testing errors are similar if not even the same. In Contrast for higher degrees like a degree of 5 or 15 the adaptive gradient descent performs better in terms of lower testing errors but it performs worse in terms of higher trainings errors. The evolution of the learning rate is not numerically coherent with our previous guess.
GDad can be useful since it can often eliminates slow convergence and divergence issues. To do that the learning rate is decreased if the cost increased and increased if the cost decreased.

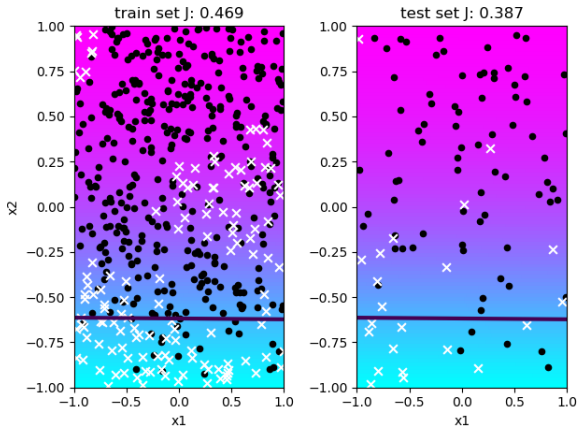

Figure 28: Hypothesis for GD for 1000 Iterations, a Learning Rate of 1 & a Degree of 1

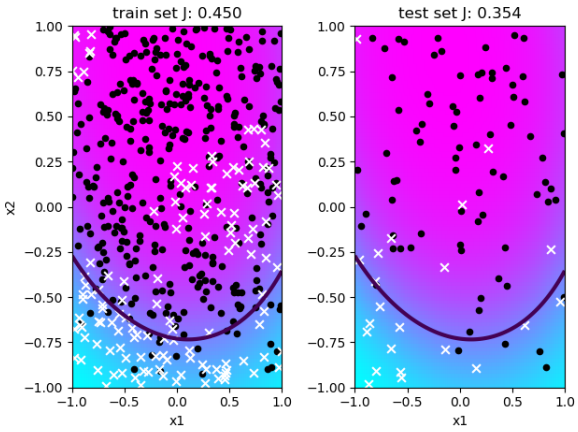

Figure 29: Hypothesis for GD for 1000 Iterations, a Learning Rate of 1 & a Degree of 2

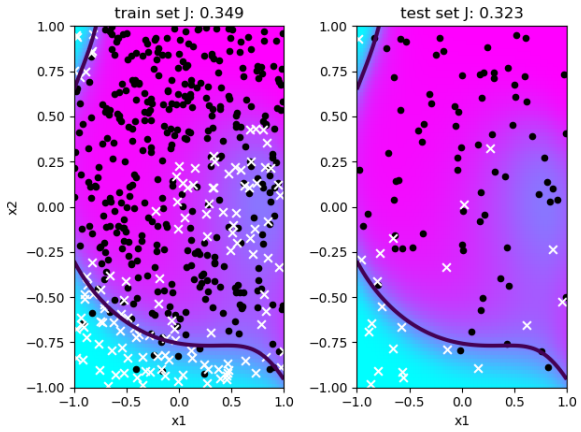

Figure 30: Hypothesis for GD for 1000 Iterations, a Learning Rate of 1 & a Degree of 5

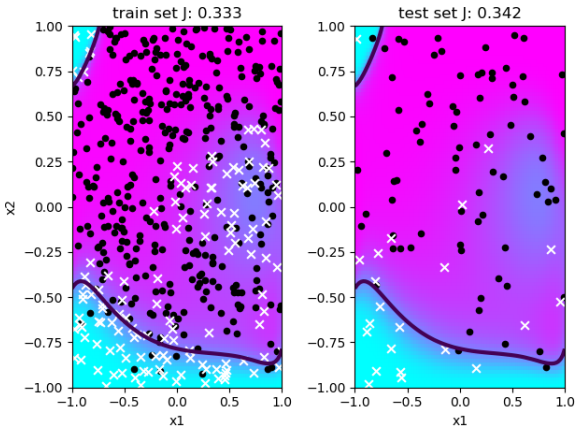

Figure 31: Hypothesis for GD for 1000 Iterations, a Learning Rate of 1 & a Degree of 15

2.3.2 Scipy optimizer

In main_logreg.py the VARIANT 3 is commented out. Un-comment the code and compare the results to the previous implementations.

1. Report training and test errors, and plot the hypothesis obtained for the four degrees $l \in 1, 2, 5, 15$ with function plot_logreg.

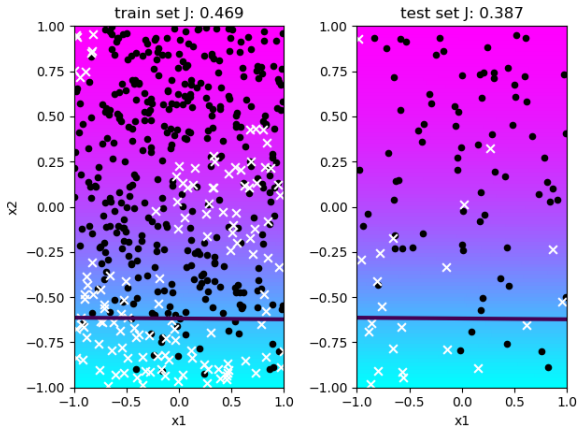

Figure 32: Scipy Optimizer Plot for a Degree of 1

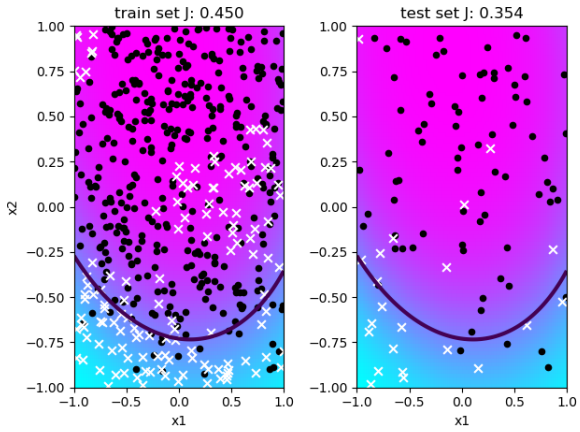

Figure 33: Scipy Optimizer Plot for a Degree of 2

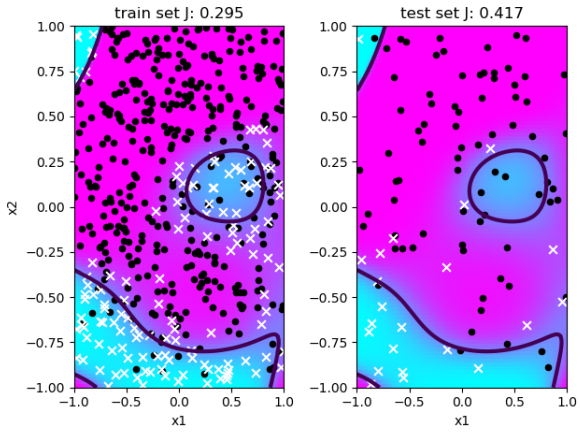

Figure 34: Scipy Optimizer Plot for a Degree of 5

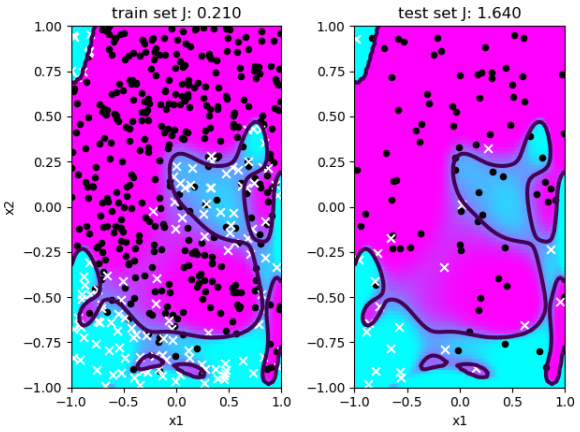

Figure 35: Scipy Optimizer Plot for a Degree of 15

Table 1: scipy

Degree	**Trainings Error**	**Testing Error**
1	0.469	0.387
2	0.450	0.354
5	0.295	0.417
15	0.210	1.640

2. Compare the results with those above (GD/GDad) and briefly discuss and interpret your observations. In particular does it change your opinion on which degree is best to fit the data?
   The results for GD, GDad and Scipy optimizer are for low degrees for example a degree of 1 or 2 for trainings & testing errors similar if not even the same. In Contrast for higher degrees like a degree of 5 or 15 the scipy optimizer performs better in terms of lower trainings errors but it performs worse in terms of higher testing errors compared to GD&GDad.
   It doesn't change our opinion on which degree is best to fit the data.