Assignment 1

Computational Intelligence, SS2017

Team Members		
Last name	First name	Matriculation Number
Alié	Félix	1644819
Pesquita	Francisco	1645291
Machado	Rita	1645715

Computation Intelligence - 01

Linear and Logistic Regression

Félix Alié, Francisco Pesquita, Rita Machado

Monday 3rd of April

Contents

1 Introduction

In this report, we will explore the basics of machine learning by studying linear regression and logistic regression. The main goal is to try to describe clearly the mechanisms behind these two different methods, which are crucial for the proper understanding of regression and classification.

2 Linear Regression

2.1 Derivation of Regularized Linear Regression

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

The derivation of the solution of the regularized linear regression is analogous to the unregularized one, as follows. Given the regularized cost function $J(\boldsymbol{\theta})$ in expression 1, we can proceed to find the optimal parameters $\boldsymbol{\theta}$ by finding the gradient of $J(\boldsymbol{\theta})$: the optimal parameters are the ones that minimize the cost function, which takes place when the gradient of the cost function is zero.

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

$$= \frac{1}{m} \left(\frac{\partial}{\partial \boldsymbol{\theta}} \left((\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \right) + \lambda \frac{\partial}{\partial \boldsymbol{\theta}} \left(\boldsymbol{\theta}^T \boldsymbol{\theta} \right) \right) \tag{3}$$

$$= \frac{2}{m} \left((\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \frac{\partial}{\partial \boldsymbol{\theta}} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \lambda \boldsymbol{\theta}^T \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\theta}} \right) \tag{4}$$

$$= \frac{2}{m} \left((\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \mathbf{X} + \lambda \boldsymbol{\theta}^T \mathbf{I} \right) \tag{5}$$

Having found the expression of the gradient, we can now equate it to zero to find our solution.

$$\left((\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \mathbf{X} + \lambda \boldsymbol{\theta}^T \mathbf{I} \right)^T = \mathbf{0} \tag{6}$$

$$\left((\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T \mathbf{X} \right)^T + \left(\lambda \boldsymbol{\theta}^T \mathbf{I} \right)^T = \mathbf{0} \tag{7}$$

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

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

$$\left(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I} \right) \boldsymbol{\theta} = \mathbf{X}^T \mathbf{y} \tag{10}$$

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

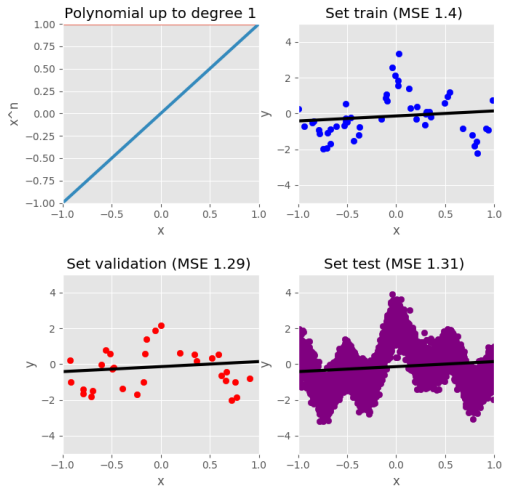
As can be seen, expression 11 is similar to the one presented in the assignment description in Hint 2, with the final result there having been the one here in equation :

$$\boldsymbol{\theta} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y} \tag{12}$$

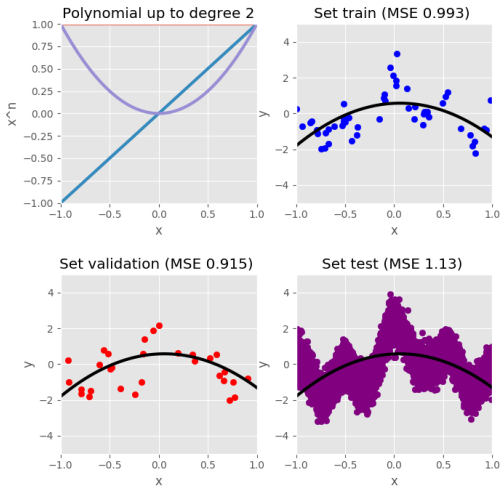
The only change caused by the "regularization" of the cost function is the addition of the penalization parameter λ to each element along the main diagonal of the factor $\mathbf{X}^T \mathbf{X}$.

2.2 Linear Regression with polynomial features

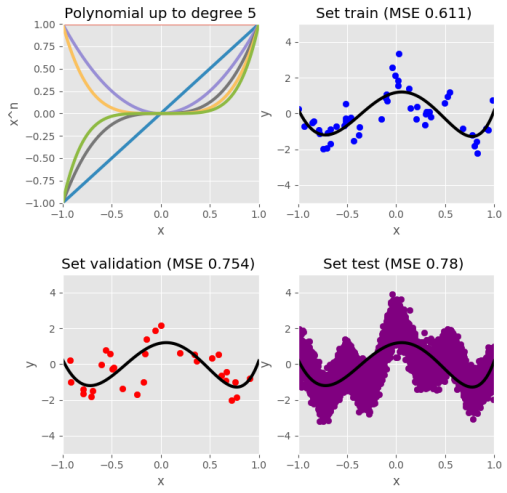
In this part, we will fit polynomials of various degrees to the provided data, and try to find which degree and corresponding polynomial best work as predictors. Firstly, let us plot each of the polynomials of degree n for $n \in \{1, 2, 5, 20\}$



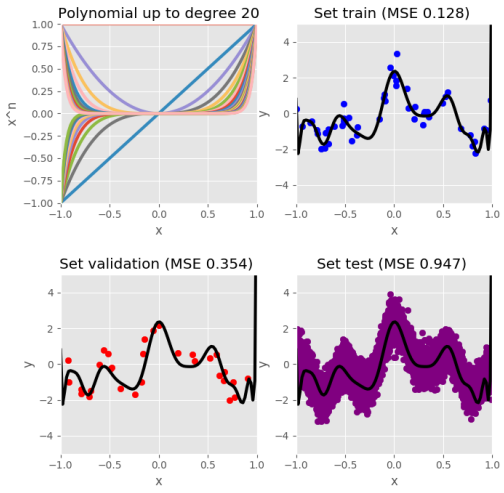
(a) Regression for $n = 1$



(b) Regression for $n = 2$



(c) Regression for $n = 5$

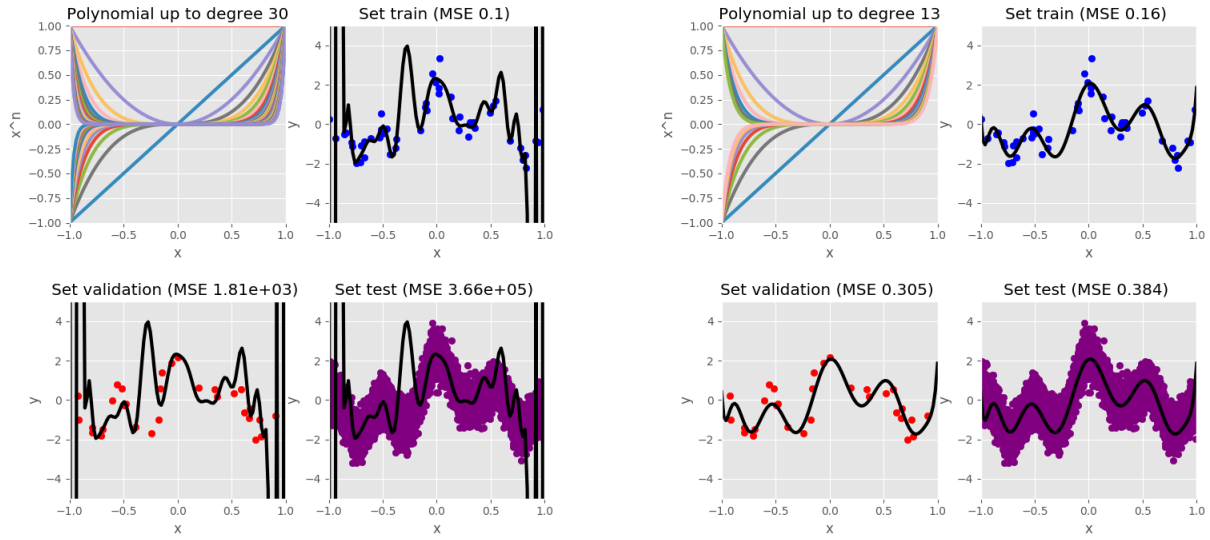


(d) Regression for $n = 20$

Figure 1: Regression for $n \in \{1, 2, 5, 20\}$

As can be seen, the higher n is, the better we can fit the training data. Running the associated script for $n = 1$ to 30, we find that the best fitting of the training data is indeed for $n = 30$

(Figure 2a), with a Mean Square Error of 0.1. The error for the test data, however, is then quite significant: $3.66 \cdot 10^5$. It is a clear case of overfitting.



(a) Regression for $n = 30$

(b) Regression for $n = 13$

Figure 2: Regression for $n = 30$ and $n = 13$

Looking instead for the value of n that minimizes this validation error, we find $n = 13$. This value gives a test error of only 0.384 (Figure 2b).

Let us now plot the training, validation and test errors as functions of n:

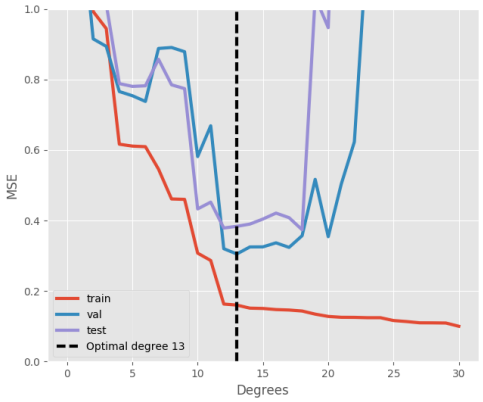


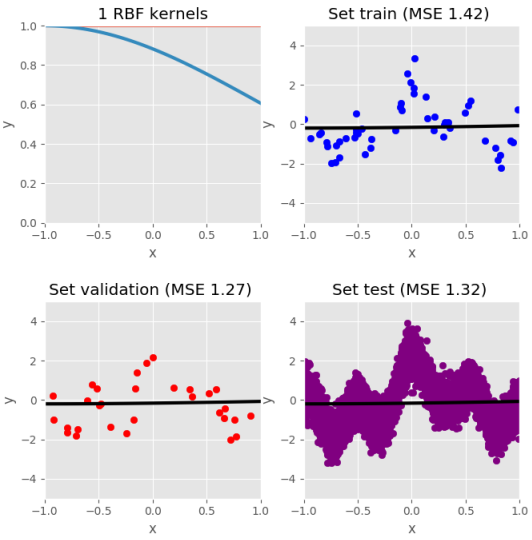
Figure 3: MSE for training, validation and test datasets depending on degree n

As we can see, while the training error is monotonic, decreasing as n increases, the validation error and test error are increase at a very high rate for $n > 13$. This is, once more, the phenomenon of "overfitting". As n increases beyond reason, the regression fits the training data *too much*, ignoring nuances and random elements in the data, which leads it to scores poorly on the other sets. This is why a validation set is useful, as it helps prevent overfitting. Therefore, the use of such a data set to validate the trained model and get a decent validation error should always prevail over the maximal minimisation of the training error.

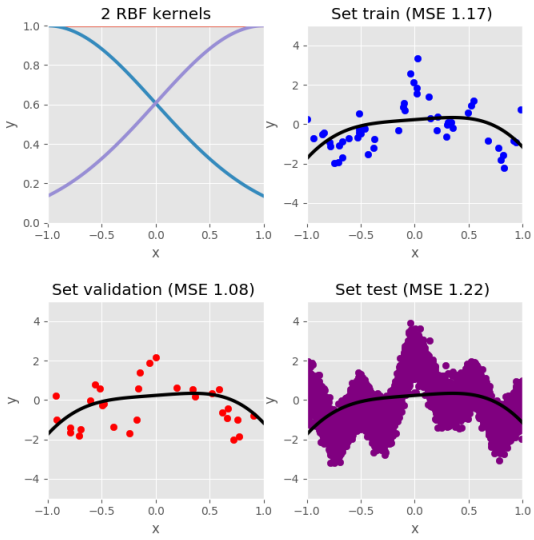
2.3 Linear Regression with radial basis functions

In this second part concerning linear regression, we will work on regression based on a linear combination of radial basis functions. The variable will be here l, the number of RBF.

As in the previous section, we plot the results for $l \in \{1, 2, 5, 20\}$:

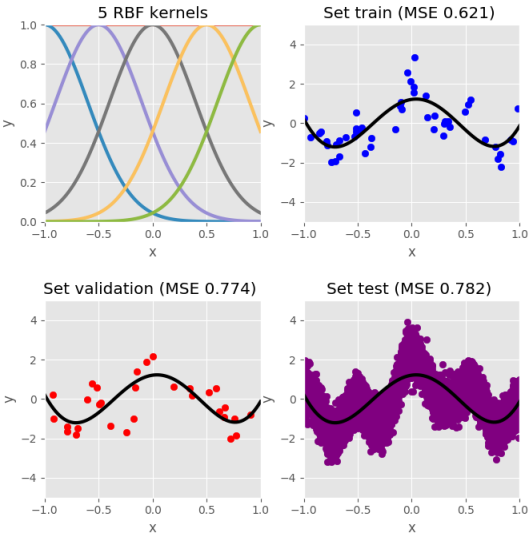


(a) Regression for $l = 1$

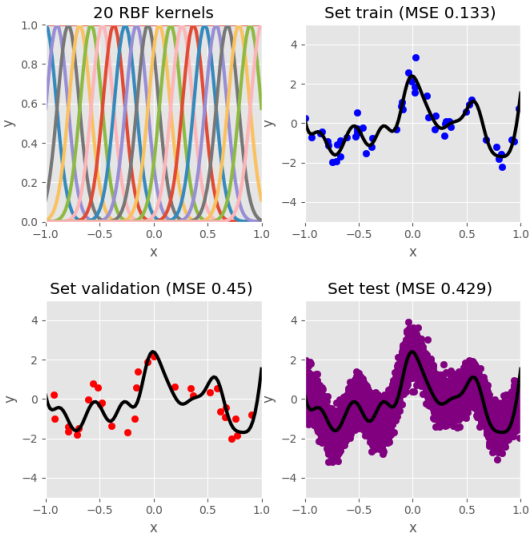


(b) Regression for $l = 2$

Figure 4: Regression for $l \in \{1, 2\}$



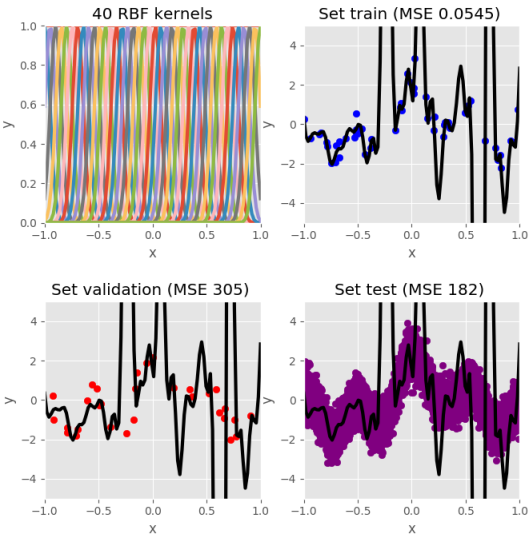
(a) Regression for $l = 5$



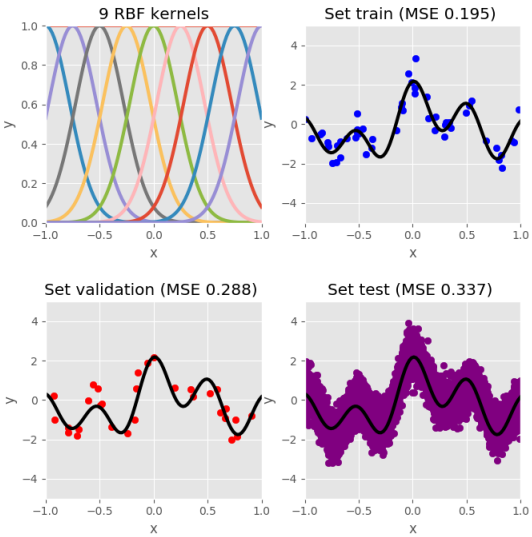
(b) Regression for $l = 20$

Figure 5: Regression for $l \in \{5, 20\}$

We then compute the values of l that minimise the training and validation errors:



(a) Regression for $l = 40$



(b) Regression for $l = 9$

Figure 6: Regression for $l = 40$ and $l = 9$

As before, the greater l is, the better our linear combination of RBF fits the training data. Testing for values from $l = 1$ to $l = 40$, 40 gives the best result with a training MSE of 0.0545 and a test MSE of 182. On the other hand, with 9 RBF - the value of l which minimises the validation error (0.288) - we obtain a slightly superior training error (0.195), but a significantly better test error: 0.337.
So once again, using a validation helps avoid overfitting. We can again visualise this by plotting the errors:

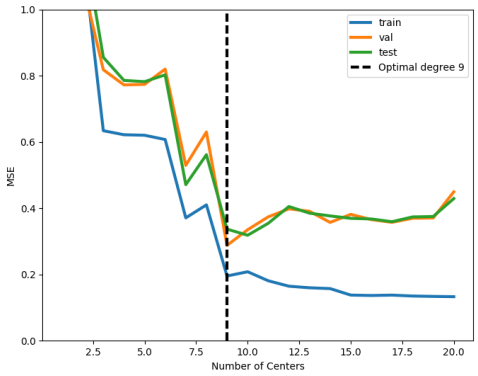


Figure 7: MSE for training, validation and test datasets depending on number of RBF l

2.3.1 Polynomial model vs RBF model

We can run a test speed using the time library, for the optimal values computed for each model:

Time and Errors of Regression models

	Time	Error
Polynomial	0.18s	0.384
RBF	0.006	0.337

As we can see, the polynomial model takes much longer than the RBF and also gives a higher error.

The difference in the error is a question of the data being more suited to one model than the other - and, most commonly, RBF does tend to be better suited for a majority of data sets. Yet RBF is not universally the better option. The polynomial model may be better for certain data sets, and it may be key to try different models and crossvalidate to find the best solution for each given problem. Previous knowledge and analysis of the data may allow us to determine which models might be best suited for it. An obvious example would be any data set made up of points which either approximately or exactly correspond to a polynomial expression being better fitted by the polynomial model. Additionally, of course, if we know the data to be linearly separable, it is better and faster to use a linear model instead, rather than either of the non-linear methods studied here.

Additionally, via a direct comparison of Figures 3 and 7, we can observe that overfitting is a much more significant concern with polynomial models: we believe this to be the case

because in order to improve fitting we must increase the degree of the polynomial, which may macroscopically lead to faster increases in rate of change across the domain of the data, and hence greater variation in between consecutive points of the training set as the degree increases, which causes a greater probability of error and greater errors when crossmatched with a test set. Meanwhile the RBF model relies only on having a greater number of RBF with centers spread out over the domain at regular intervals - the rate of change does not, then, vary wildly with a greater number of RBF as it does with increasing polynomial degrees, which leads to a smoother and more stable function throughout and a more stable error/resolution even at unnecessarily high degrees, which is one of the main factors that lead RBF to be the better general-case model.

3 Logistic Regression

3.1 Derivation of Gradient

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\ln\left(h_\theta(x^{(i)})\right) + \left(1-y^{(i)}\right)\ln\left(1-h_\theta(x^{(i)})\right)\right) \tag{13}$$

Considering the logistic regression cost function as shown in (13), the logistic regression hypothesis and the sigmoid function it uses, we can start to simplify the cost expression before deriving it. Initially,

$$\ln\left(h_\theta\left(x^{(i)}\right)\right) = \ln\left(\frac{1}{1+e^{-x^T\theta}}\right) = \ln(1) - \ln\left(1+e^{-x^T\theta}\right) \tag{14}$$

$$= -\ln\left(1+e^{-x^T\theta}\right) \tag{15}$$

It is also possible to deduce,

$$\ln\left(1-h_\theta\left(x^{(i)}\right)\right) = \ln\left(1-\frac{1}{1+e^{-x^T\theta}}\right) = \ln\left(\frac{1+e^{-x^T\theta}}{1+e^{-x^T\theta}} - \frac{1}{1+e^{-x^T\theta}}\right) \tag{16}$$

$$= \ln\left(\frac{e^{-x^T\theta}}{1+e^{x^T\theta}}\right) = \ln\left(e^{-x^T\theta}\right) - \ln\left(1+e^{-x^T\theta}\right) \tag{17}$$

$$= -x^T\theta - \ln\left(1+e^{-x^T\theta}\right) \tag{18}$$

Replacing (15) and (18) in the inicial expression, (13), we obtain the following:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\left(-\ln\left(1+e^{-x^T\theta}\right)\right) + \left(1-y^{(i)}\right)\left(-x^T\theta - \ln\left(1+e^{-x^T\theta}\right)\right)\right) \tag{19}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x^T\theta - x^T\theta - \ln\left(1+e^{-x^T\theta}\right)\right) \tag{20}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x^T\theta - \ln\left(e^{x^T\theta}\right) - \ln\left(1+e^{-x^T\theta}\right)\right) \tag{21}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x^T\theta - \ln\left(e^{x^T\theta}\left(1+e^{-x^T\theta}\right)\right)\right) \tag{22}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x^T\theta - \ln\left(1+e^{x^T\theta}\right)\right) \tag{23}$$

Deriving now the gradient of the cost function as presented above in (23) we can see that the final result of the partial derivative of $J(\theta)$ with respect to θ_j is the expected one.

$$\frac{\partial J(\theta)}{\partial\theta_j} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x_j^{(i)} - \frac{\partial\ln(1+e^{x^T\theta})}{\partial\theta_j}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x_j^{(i)} - \frac{x^T}{e^{-x^T\theta}\left(1+e^{x^T\theta}\right)}\right) \tag{24}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}x_j^{(i)} - x^T\sigma\left(x^T\theta\right)\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)} \tag{25}$$

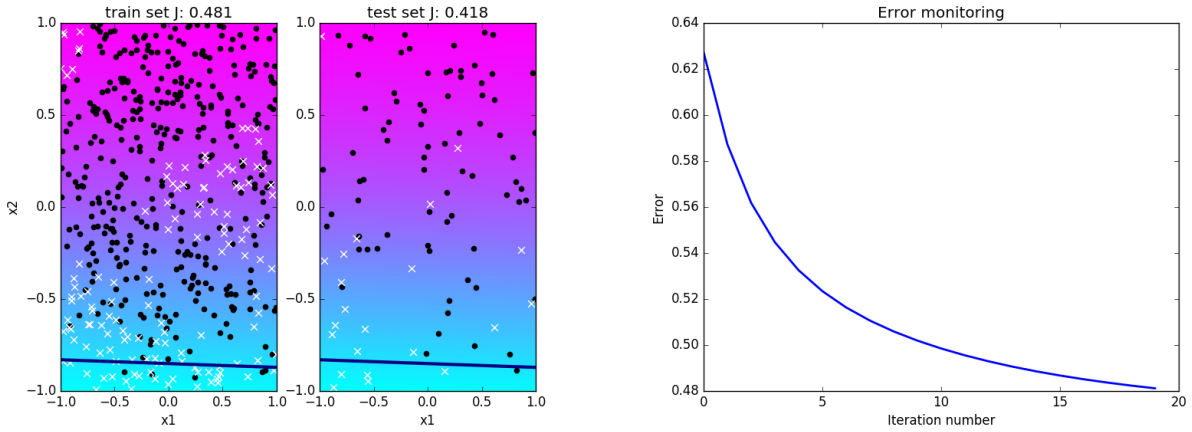
$$= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \tag{26}$$

3.2 Logistic regression of training with gradient descent and scipy.optimize

3.2.1 Gradient descent

1.

2. We are now only looking at degree 1, analyzing how this fits the data set provided when a learning rate of 1 is used and the number of iterations is either 20 or 2000.

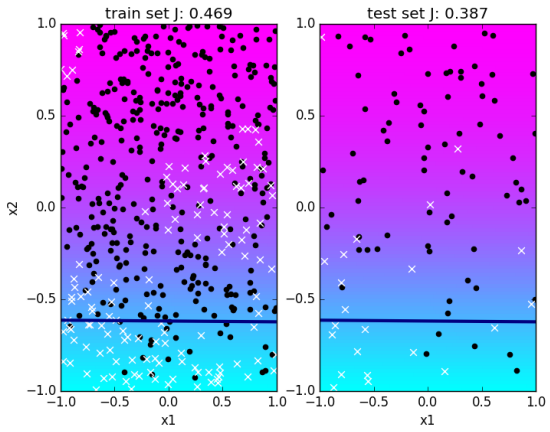


(a) Decision boundaries, training and test errors

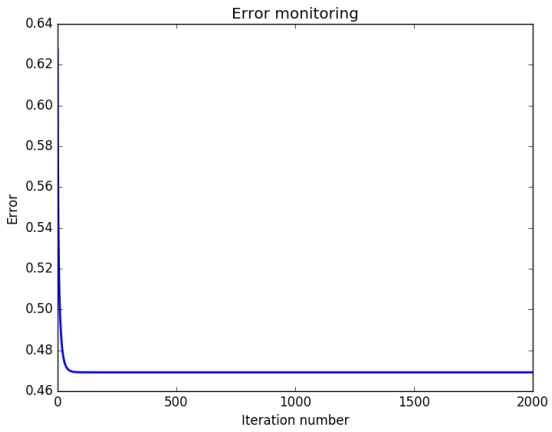
(b) Error monitoring

Figure 8: Logistic Regression training with gradient descent for $l = 1$ with 20 iterations

Analyzing the plot above and the correspondent error monitoring we can see that using a number as low as 20 iterations, although almost converging, causes the error of the train set, 0.481, and of the test set, 0.418, to be higher than they could by adjusting the parameters correctly. This is proved by the following plot where we can verify that the errors get lower (train set error is 0.469 and the test set error, 0.387) by increasing only the number of iterations.



(a) Decision boundaries, training and test errors

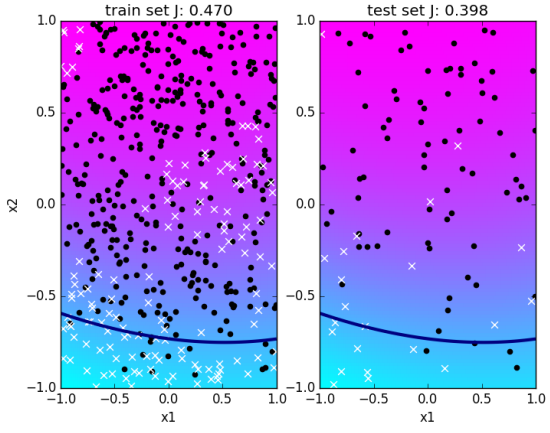


(b) Error monitoring

Figure 9: Logistic Regression training with gradient descent for $l = 1$ with 2000 iterations

However, it´s also clear in the error monitoring presented above that 2000 iterations is a higher number than necessary, such is visible in the fact that the error has already converged at about 100 iterations. This makes the time spent with all the remaining ones not required for a correct regression process.

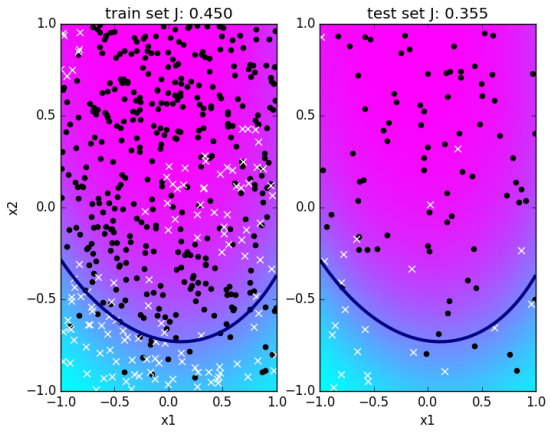
3. The objective is now to analyze the behavior of the regression and the influence the values of eta have in it, when using the second degree, 200 iterations and varying only the learning rate between 0.15;1.5;15.



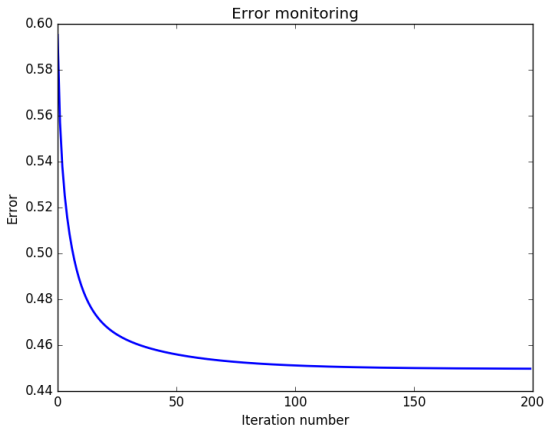
(a) Decision boundaries, training and test errors

(b) Error monitoring

Figure 10: Logistic Regression training with gradient descent when $\eta = 0,15$

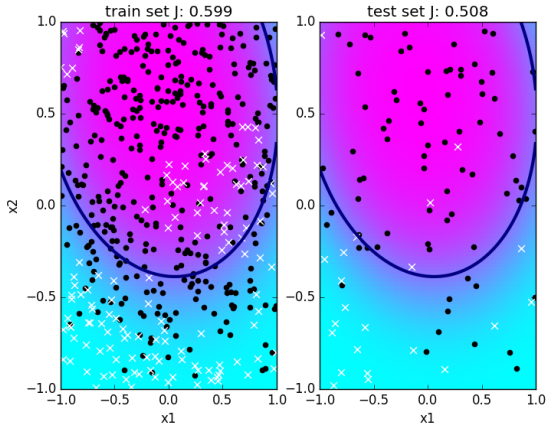


(a) Decision boundaries, training and test errors

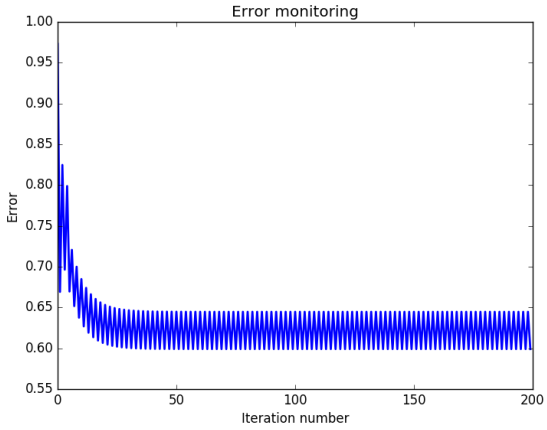


(b) Error monitoring

Figure 11: Logistic Regression training with gradient descent when $\eta = 1,5$



(a) Decision boundaries, training and test errors



(b) Error monitoring

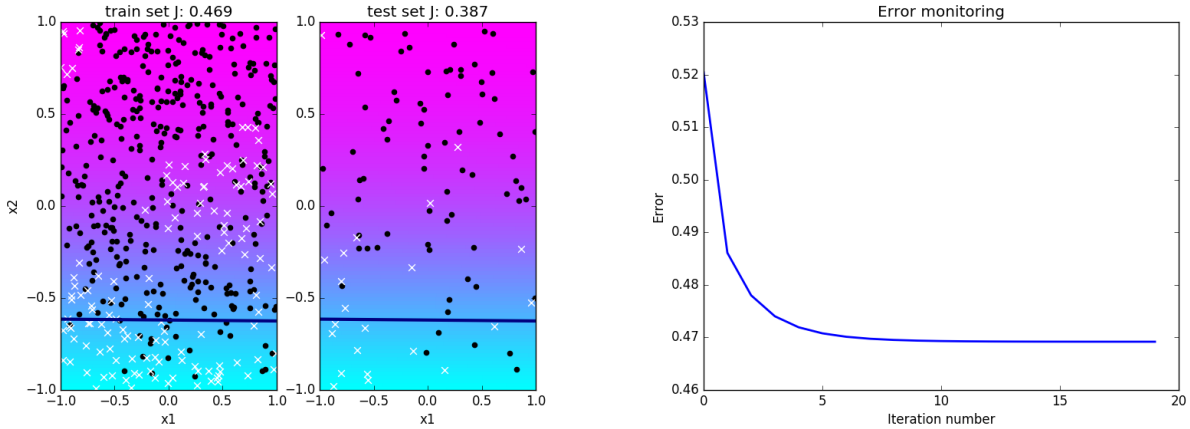
Figure 12: Logistic Regression training with gradient descent when $\eta = 15$

Observing the three sets of images above, it's possible to conclude that the most appropriate learning rate would be 1.5, also it´s visible in the error monitoring that, with this rate, 200 iterations are more than what would be necessary.
When η is too small, as seen in $\eta = 0,15$, the error of the train set (0,470) and the test set (0,398) are worse than expected since the number of iterations isn't altered when the learning rate is diminished. This is also supported by the fact that the error hasn´t converged by the end of the iterations.
In the last set, it´s visible that $\eta = 15$ is not appropriate, not only in the errors presented (train set error, 0.599; test set error, 0.508) which are higher than the ones obtained with any of the others learning rates. But also by observing the error monitoring where the visible oscillations

make it clear that the learning rate is too high.

4. The goal in this section is to find good pairs of values for iterations and learning rate, for each of the requested degrees. To find these pairs we considered the error values, the convergence of the error and the time of execution of each regression using the gradient descend.

The pairs selected were:

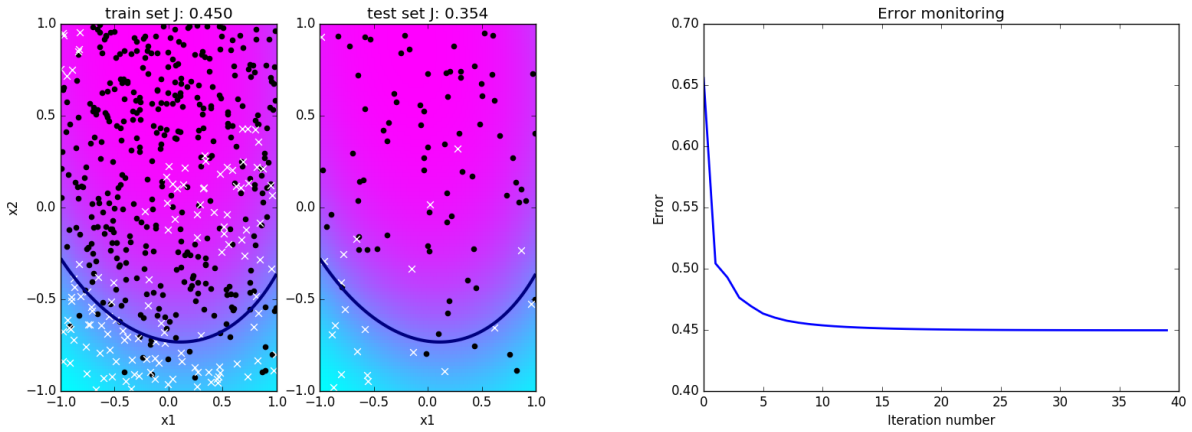


(a) Decision boundaries, training and test errors

(b) Error monitoring

Figure 13: Logistic Regression training with gradient descent when $l = 1$

Degree 1 -> learning rate of 6 and 20 iterations
Train set error->0.469
Testing set error->0.387
Time of execution-> 0,011 s

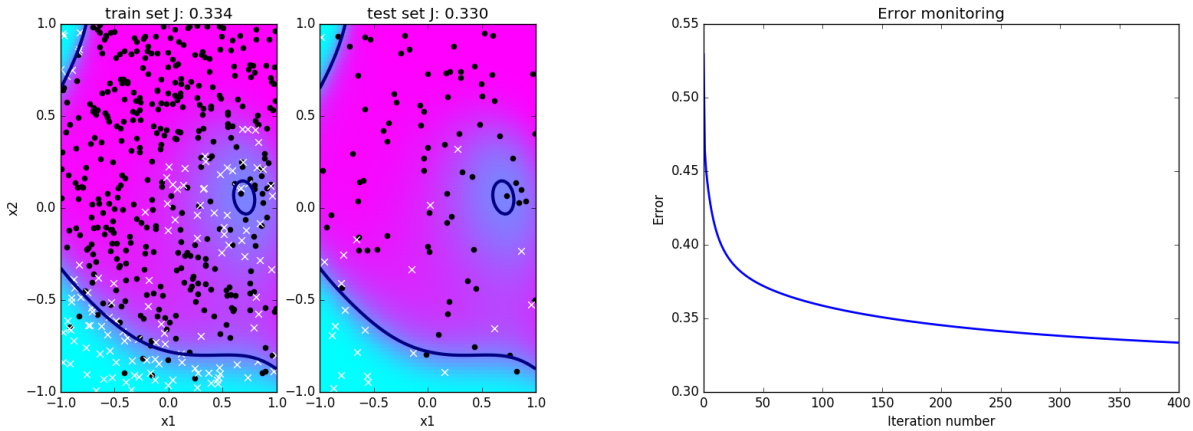


(a) Decision boundaries, training and test errors

(b) Error monitoring

Figure 14: Logistic Regression training with gradient descent when $l = 2$

Degree 2 -> learning rate of 9 and 40 iterations
Train set error-> 0.450
Testing set error-> 0.354
Time of execution-> 0,018 s

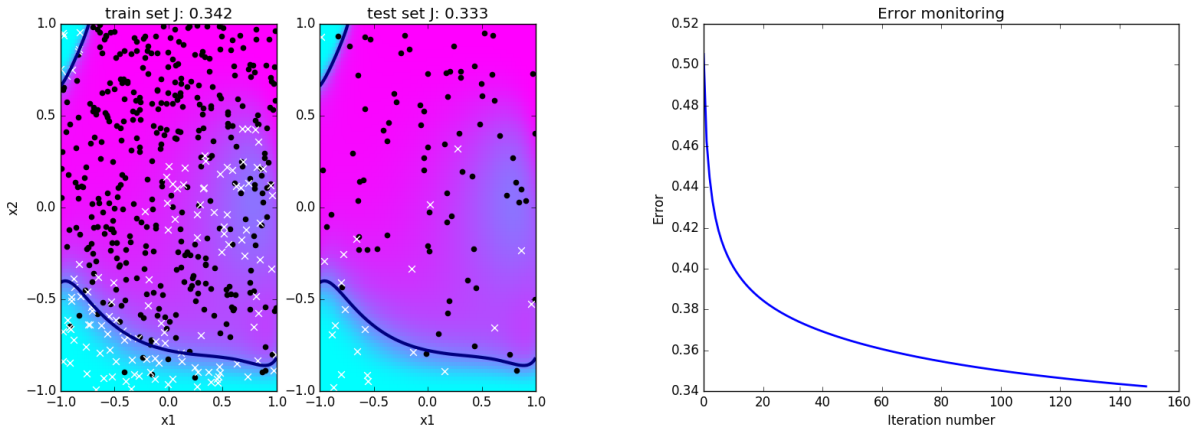


(a) Decision boundaries, training and test errors

(b) Error monitoring

Figure 15: Logistic Regression training with gradient descent when $l = 5$

Degree 5 -> learning rate of 6 and 400 iterations
Train set error-> 0.334
Testing set error-> 0.330
Time of execution-> 0,248 s



(a) Decision boundaries, training and test errors

(b) Error monitoring

Figure 16: Logistic Regression training with gradient descent when $l = 15$

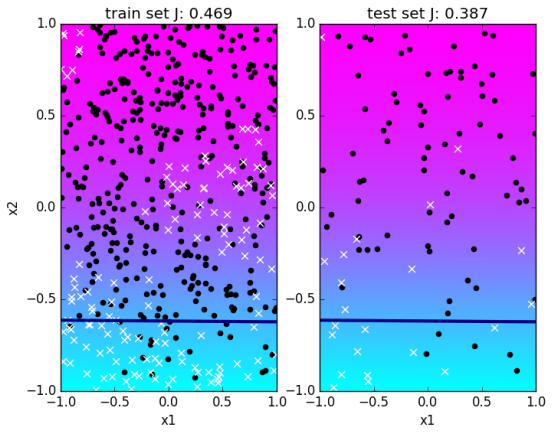
Degree 15 -> learning rate of 4 and 150 iterations
Train set error-> 0.342
Testing set error-> 0.333
Time of execution-> 0,41 s

Concluding, based on the errors and graphics provided above, the most appropriated degree appears to be l=5.

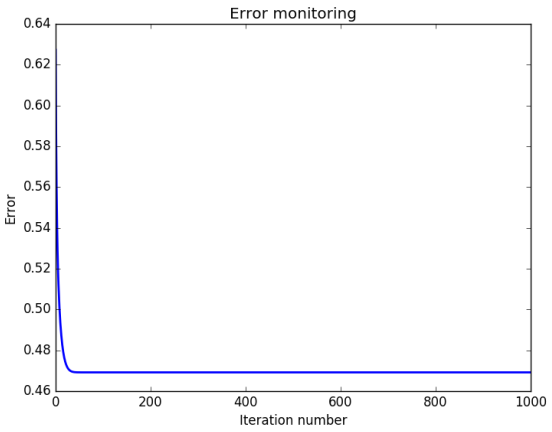
5.A possible stopping criterium to avoid doing too many iterations is the convergence of the error, that is, by analyzing the error monitoring graphics we can better approach the number of iterations necessary for the error to converge being that the required amount for a good regression, when combined with the right learning rate.

3.2.2 Adaptive gradient descent

In this section, we are using 1000 iterations, learning rate of 1 and varying the degrees between 1, 2, 5, 15. The goal is to analyze the impact of using the adaptive gradient descent on logistic regression training while comparing these results with the ones chosen in the prior question. We will now present the obtained hypothesis, including the final learning rates and corresponding errors.



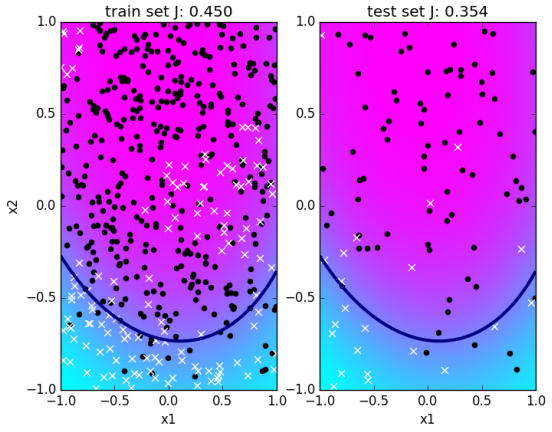
(a) Hypothesis obtained and respective training and test errors



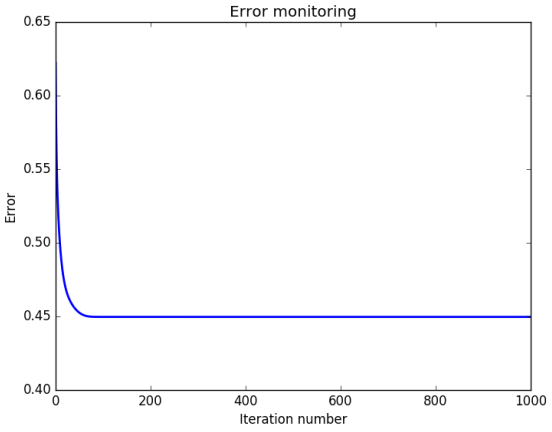
(b) Error monitoring

Figure 17: Logistic Regression training with an adaptive gradient descent when $l = 1$

Considering degree, l= 1, we saw a final learning rate of $5,34 * 10^{-5}$ and a time of execution of 0,231 seconds. With the non-adaptive gradient descent variant, the pair of values selected differ greatly from these but encounter the same final errors in a better time.



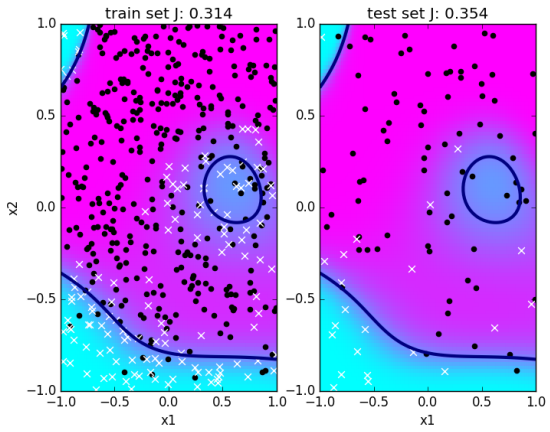
(a) Hypothesis obtained and respective training and test errors



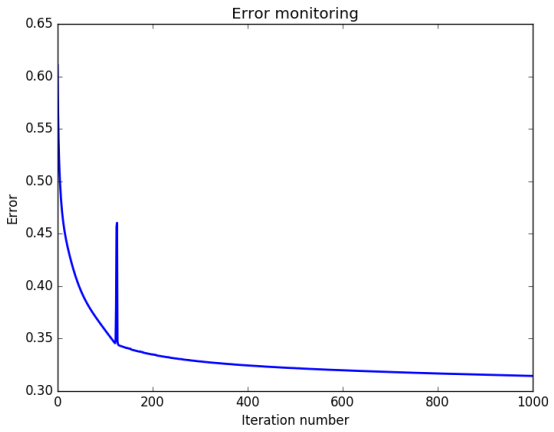
(b) Error monitoring

Figure 18: Logistic Regression training with an adaptive gradient descent when $l = 2$

With l=2, the final learning rate was $5,19 * 10^{-7}$ and the regression process took 0,293 s. The previously selected values are different since time of execution was valued so we opted for a lower number of iterations which needed to be accompanied by a higher learning rate.



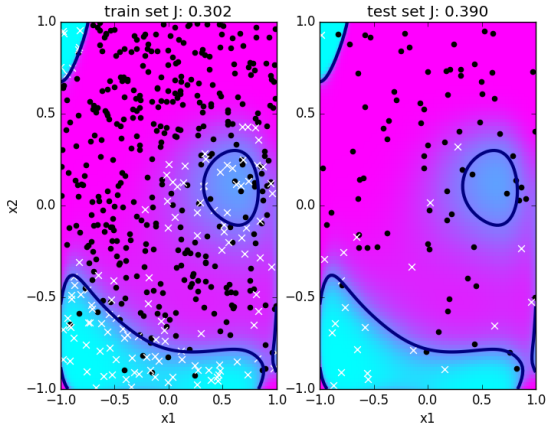
(a) Hypothesis obtained and respective training and test errors



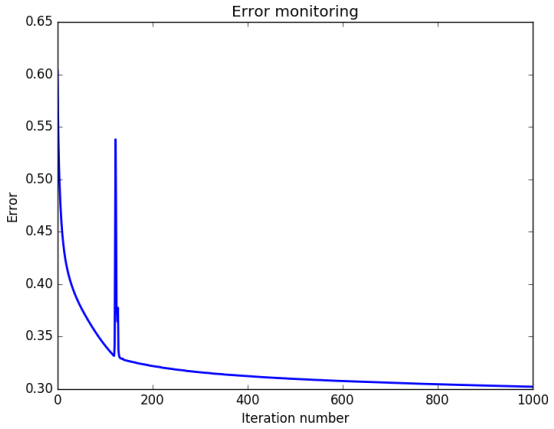
(b) Error monitoring

Figure 19: Logistic Regression training with an adaptive gradient descent when $l = 5$

When executed with l=5, the final eta was 12,5 for 0,65 s of execution time. In this case, we can observe that there is some oscillation showed in the error monitoring graphic which suggests that the learning rate was eventually too high before being decreased again. When compared to the non-adaptive gradient descent variant, it´s clear that there is still a big gap between the values selected and the ones now obtained. This is explained by the fact that previously, values which provided a lower test set error were preferred over the ones showed above. A slight overfitting is visible in the results presented by the adaptive gradient.



(a) Hypothesis obtained and respective training and test errors



(b) Error monitoring

Figure 20: Logistic Regression training with an adaptive gradient descent when $l = 15$

Finally, when the chosen degree was 15, the final eta was 12,5 and it took 2,83 seconds for the execution to be completed. Here, the conclusion is similar to the one presented with the previous degree, being now very clear the overfitting in this result. With the results presented

for this degree on the prior questions this did not happen.
We can conclude that these numbers and results were not coherent with our previous guesses since we choose results were timing and lower test set errors, or more balanced results in the error values, were given a higher importance than by the non-adaptive gradient.
GDad, the adaptive gradient descent, can be useful because it adapts the learning rate automatically depending on the value of the cost function at each iteration. In this way, it's possible to make the regression more correct, specially when the right degree and amount of iterations are provided.

3.2.3 Scipy optimizer

In this section we shall look at how using the scipy optimizer, instead of the adaptive or non-adaptive gradient descent, affect the logistic regression training. For that intent we will now analyze the hypothesis obtained for $l \in \{1, 2, 5, 15\}$.

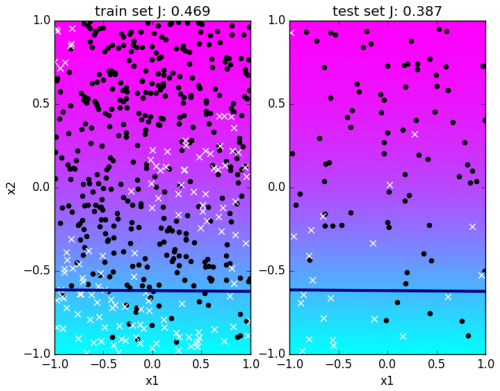


Figure 21: Logistic regression training using the scipy optimizer with degree, l=1

The values here presented are consistent with the ones obtained with GD and GDad the difference being in the number of iterations used.
Based on the error monitoring of the plot obtained in the GD experience we could conclude that 20 iterations were enough for the error to converge, this is here confirmed by the simple 17 iterations needed for the scipy optimizer.

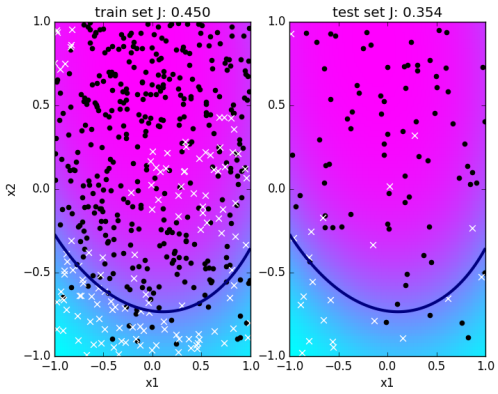


Figure 22: Logistic regression training using the scipy optimizer with degree, l=2

For the case above, the error results are again equal to the ones obtained in the sections presented before. Also, the 38 iterations used here verify the comments in section 4 of the gradient descent filter analysis, referring to degree 2, where by 40 iterations the error had converged, as proved in the error monitoring there presented.

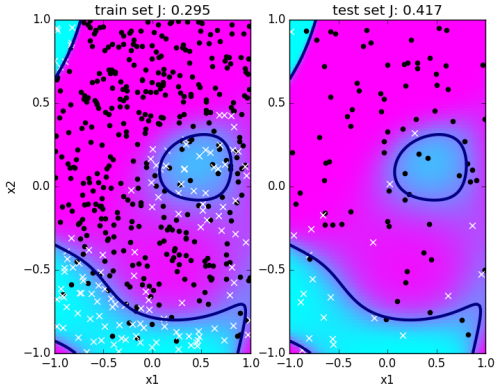


Figure 23: Logistic regression training using the scipy optimizer with degree, l=5

When l=5, the errors presented above are not as good or balanced as the ones obtained with the use of the adaptive or non-adaptive gradient. Here a lower number of iterations, 241, than before is used and the overfitting to the train set is clear, creating a higher test set error than the values obtained in the prior sections.

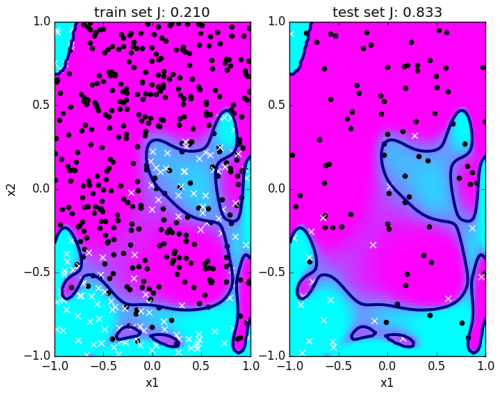


Figure 24: Logistic regression training using the scipy optimizer with degree, l=15

Finally, when l=15 the results are impacted by the same factors as in degree 5 although here the test set was different enough for the it's error to highly increase. Also the number of iterations used is 628, higher than already proven necessary, which also leads to the overfitting issues.

We can now observe that although the adaptive gradient descent and the scipy optimizer are more automated and simple and fast to use, these do not consider every criterium responsible for a good regression training. This way, if the gradient descent is used and adapted depending on the plots obtained, after several attempts a better or equal result can be obtained than with the other two options. For instance, for degree 1 and 2 we can in this section confirm the pairs of values chosen initially.
However, if the process must be automated the adaptive gradient descent is also not the better option since the number of iterations is still decided by the user. Only the scipy optimizer can be totally automatic, not depending on any parameter but the degree in question. Nevertheless, this method seems to present overfitting issues when higher degrees are studied.