Report
Martin Rosenfeld A20350531

Parametric Regression

1 Problem Statement

In this assignment we study *Parametric Regression*. In parametric regression we choose a function which describes the relationship between the response variable or label and the explanatory variables. In the training phase we find the parameters $\theta_1, \theta_2, ..., \theta_k$ of the model which best fit the data. With the result model we are able to classify and predict new examples.

There are diverse solutions for parametric regression. Each of them has advantages and disadvantages. Also, they have to be tunned with some parameters to make them work in a specific problem.

In this report I analyze three different solutions: *numerical* or *explicit* solution, the iterative algorithm *gradient descent*, and the dual problem using *gaussian kernel function*. For each of them I make an analysis of the algorithm and show the impact of its parameters. Later, I make a comparison of their performance.

2 Data Sets

To evaluate the algorithms I worked with both synthetic and real datasets. Synthetic datasets can be accessed in the CS 584 course web site[1] and include both univariate and multivariate samples. For this datasets, there is no background information of the meaning of data. However, they are useful to test the algorithms and evaluate their performance. Real datasets were downloaded from the UCI Machine Learning Repository[2]. I selected three of them to work with:

- Housing Data Set[3] concerns housing values in suburbs of Boston. It contains 506 examples, 13 predictor variables and a target attribute (median value of owner-occupied homes in $1000's).
- Auto MPG Data Set[4] concerns city-cycle fuel consumption in miles per gallon. It contains 398 instances, 8 predictor variables and a target attribute (miles per gallon).

[1] `http://www.cs.iit.edu/~agam/cs584/`
[2] `https://archive.ics.uci.edu/ml/datasets.html`
[3] `https://archive.ics.uci.edu/ml/datasets/Housing`
[4] `https://archive.ics.uci.edu/ml/datasets/Auto+MPG`

- Combined Cycle Power Plant Data Set[5] contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. It contains 4 predictor variables and a target attribute (net hourly electrical energy output (EP)).

To be able to do univariate analysis with real data sets, I chose one representative attribute from the predictor variables. To describe different cases, I chose different attributes in each case.

3 Implementation Details

I implemented all the algorithms from scratch. However, I used some existent libraries to facilitate the development of functions which are not part of the algorithms. The used libraries include *sklearn*, *scipy* and *numpy*. For example, to generate polynomial and interaction features I used *sklearn.preprocessing*, and to obtain the indices of folds to implement the K Fold cross validation algorithm I used *sklearn.cross_validation*.

I previously validated with Prof. Agam that the usage of this libraries was permitted.

3.1 Preprocessing

To work with the data sets in the multivariate case, I first normalized the data matrix on each feature. I normalized data with the function $x_j^{'(i)} = \frac{x_j^{(i)} - \bar{x_j}}{\sigma_j}$. In this way I avoid each of the features having different weights in the algorithms.

In the case of the real data sets, some of the values were missing. Instead of removing those examples, I decided to infer them from the data. Thus, I replaced the missing values with the *mean* of the corresponding feature. There are other methods to infer the values, e.g. by averaging in the example neighborhood, but they are out of the scope of this assignment.

3.2 Instructions

To make the code cleaner I created two python file libraries: *data_utils.py* contains functions to import data sets into numpy arrays; *regression.py* contains all the regression algorithm implementations, and reusable util methods, e.g. to evaluate performance.

The code used for the experiments contained in this report are organized in python (.py) scripts. The delivered code does not contain all the code necessary to generate all of the plots and graphs included in this report. They are a selection to show the implementation and evaluation of the algorithms. The selection includes the following files:

- *univariate.py*: For a univariate data set, fit the model and evaluate the performance using the three algorithms.

[5] `https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant`

- *multivariate.py*: For a univariate data set, fit the model and evaluate the performance using the three algorithms.
- *polynomial_degree.py*: Fit a univariate sample test set using three different polynomial degrees, and show the fit models in a plot.
- *parameter_analysis.py*: For the MPG real data set, evaluate the performance, using K fold cross validation, for different values of degree of polynomial and K.
- *housing_fitting.py*: For the Housing real data set, show the regression model on top of the test data.
- *convergence.py*: For the CCPP real data set, the convergence of the gradient descent algorithm for different learning weights.
- *gaussian_kernel.py*: For the Housing real data set, fit the model using the dual problem with gaussian kernel function.

To execute the python scripts, go to the directory where the code was downloaded and execute the code using *python* command. For example:

>>python univariate.py

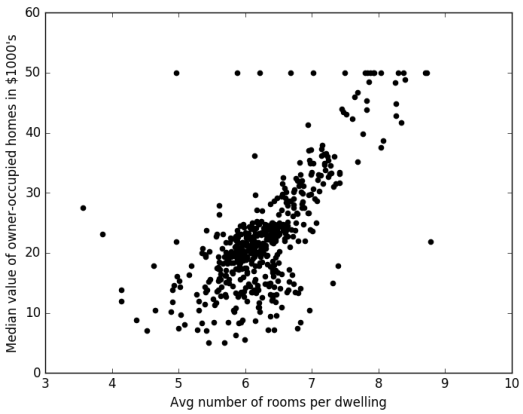
4 Explicit Solution

The explicit solution training phase consists in obtaining the coefficients θ which minimize the objective function Sum of Squared Errors $J(\theta) = (Z\theta - Y)^T(Z\theta - Y)$, where Z is the data matrix and Y is the target vector. The solution is $\theta = (Z^T Z)^{-1} Z^T Y$.

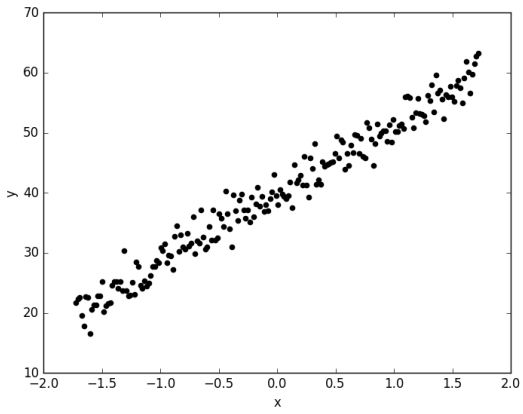
The simplest case is for the univariate case assuming a simple linear model. Figure 1 show plots of different data sets. In the first two, although some outliers, it is reasonable to think that a simple linear model could fit the data. For example, for the first plot of the Housing data set, there seems to be a straight linear relationship between the average number of rooms per dwelling (x) and the median value of owner-occupied homes (y). The latter increases in a straight linear fashion as the first one increases. However, it is not the case for the sample data in figure 1c, in which there is a clear curve relationship between x and y.

Clearly, fitting a linear model for the first two cases would be appropriate, but not for the third one. To see it graphically, I show in figure 2 the regression models obtained on top of the test data. The black points are testing examples, and the blue line is the fitted model. For these experiments, I split the data set leaving 90% for the training phase and 10% for the testing. It can be observed that the testing error (which graphically can be seen as the distance between the points and the line) is very high in the third case. In figure 2b, which is an idealized sample data set, the testing error is very small. In figure 2a, which is a real data set, the model fit the testing data with some reasonable testing error.

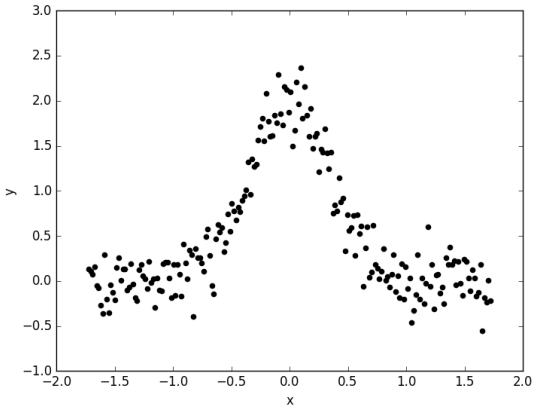
As a conclusion of the previous analysis, it is clear that the third case needs a more complex model. The simple linear model is too simple to capture the complexity of the data. This situation is called **underfitting**.



(a)

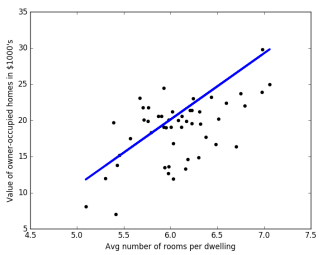


(b)

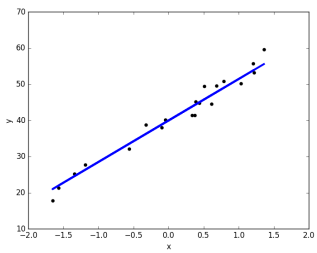


(c)

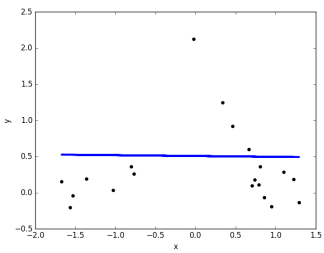
Figure 1: Data Set Plots



(a)

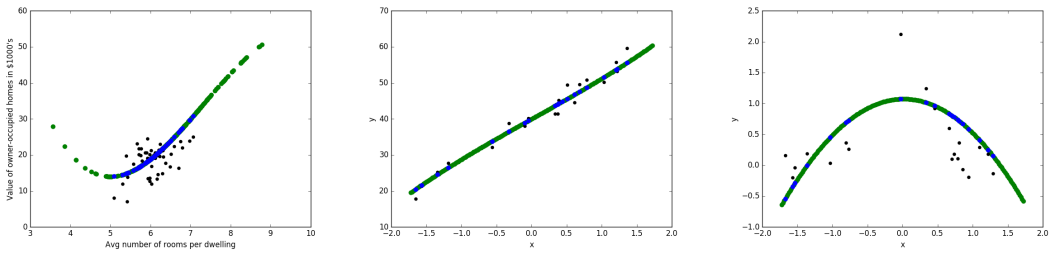


(b)



(c)

Figure 2: Regression linear model on top of the test data



(a) (b) (c)

Figure 3: Regression cubic model on top of the test data

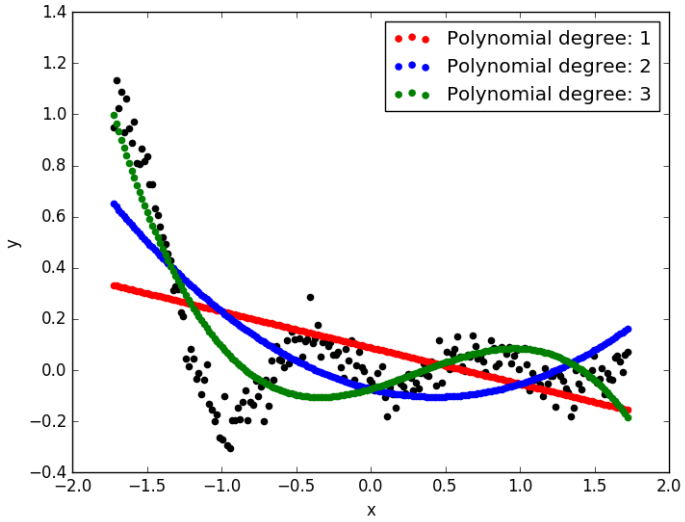


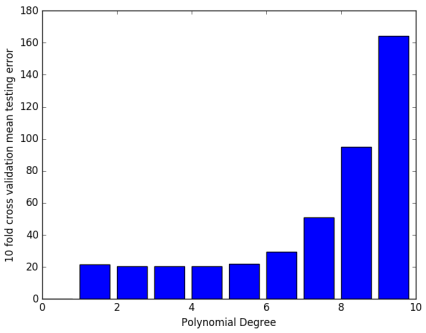
Figure 4: Different polynomial models

Now let's suppose that we decide to use a more complex cubic polynomial model instead of a simple linear model. We are assuming a model of the form $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ for the univariate case.

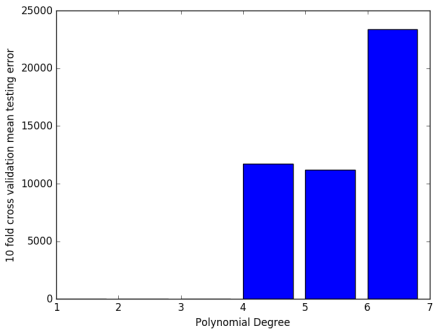
Figure 3 shows the new plots of the regression model over the testing data. Additionally, the green points show the fitted model over a larger range than the testing data. Now, the third plot shows an appropriate fit and a smaller testing error. However, for the first plot it happened what is called **overfitting**. The fitted model was so complex and adapted so well to the training data that new examples would be incorrectly predicted. For instance, the predicted value for a new example with avg rooms per dwelling of 4 will be much greater than it should be. From what the data shows, it would be unreasonable to predict a greater value for an avg number of rooms per dwelling of 4 than of 6.

To further see the impact of the degree of the polynomial, figure 4 shows the fit of three different degree of polynomial functions in a new sample test. Clearly the polynomial with degree 3 (color green) is the one that best fits this data set.

To formalize this discussion, I will analyze the performance of the regression model for different degree polynomial models. This analysis was made using the MPG data set, which relates different features of a car, e.g. the Horse Power, to MPG (Miles per Gallon). I used 10 fold cross validation to evaluate the performance of different polynomial degree



(a) Mean Error for different polynomial degrees (Univariate)



(b) Mean Error for different polynomial degrees (Multivariate)

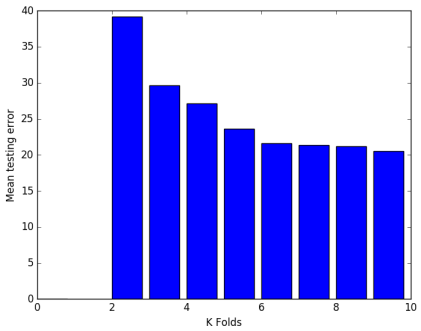
Figure 5

models. This algorithm splits the data in 10 folds and utilize each of them one time as testing data. I made the experiment for both the univariate case recently introduced and the multivariate case of the same data set, which contains 8 predictor variables for mpg. Figure 5 shows the result plots. The x-axis represents the degree of the polynomial, while the y-axis is the mean testing squared error of all the iterations of the 10 fold cross validation algorithm. In the univariate case, there is an improvement when using a quadratic (degree 2) polynomial compared to a linear model. However, as the degree of the polynomial increases, the performance is worse. This shows that using complex models can help better fit the training data but, because of the overfitting effect, the performance in the test set, which is the most important one, can be worse if the model is too complex. Also, it must be said that the higher the degree the more expensive in terms of time is the algorithm. This is because the dimensions of the matrices are much larger. For example, 8 predictor variables are mapped to 120 polynomial and interaction features if a degree 3 is used.

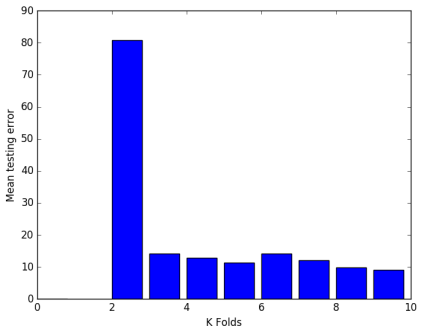
4.1 Training and Test Error

Another important thing to analyze is the effect of the size of the training set. Obviously, data is limited and we have to work with the available data. It is expected that the larger the training set size is, the best performance will be acquired. This is because the model will have more data to learn from.

To test this hypothesis I evaluate the performance for different values of K of K Fold Cross Validation for the same univariate and multivariate cases of the auto MPG data set. For the experiment I used the quadratic model. Figure 6 shows the results. Note that for smaller values of K, the training set size is smaller. For example, for K=2, only 50% of the data set is used for training in each iteration. As expected, the mean squared error obtained decreases as k, and thus the size of the training size increases.



(a) Effect of Training Size in Performance (Univariate)



(b) Effect of Training Size in Performance (Multivariate)

Figure 6

5 Iterative Solution

An iterative solution can be used when performing the explicit solution is not possible or is too expensive, e.g. when the data matrix is too big.

I implemented the iterative solution *Gradient Descent*. The gradient descent algorithm starts with random values for the parameters θ's and, in each iteration, it improves the solution by moving in the direction of the inverse of the gradient.

One fundamental parameter for this algorithm is the learning weight. It the learning weight is big, then the algorithm converges faster because each step in the direction of the inverse of the gradient is larger. However, if the learning weight is too big, the steps would be too large to get to an optimal solution and the algorithm would never converge.

On the other side, if the learning weight is small it will ensure that algorithm will go in the correct direction doing small steps. However, if the steps are too small, then it would take so long to get to the optimal solution than the algorithm would stop before that happens.

I tested the gradient descent algorithm using the Combined Cycle Power Plant (CCPP) Data Set. It must be remembered that I first normalized the values of the features. The stop conditions of the algorithm are reaching a maximum number of iterations or a minimum threshold of performance improvement is not achieved.

To see the impact of the learning weight, figure 7 shows the first 50 iterations of the gradient descent algorithm using three different learning weights. As the learning weight increases the algorithm converges more quickly. Note that the red line, which uses a learning weight of 0.000015 minimize the squared mean error in an early iteration.

Finally, figures 8a and 8b show graphically different runs of the algorithm for different learning weights. Clearly, as the learning weight increases, the number of iterations until it converges decreases. However, using a very big learning weight make the algorithm not find an optimal solution. Note that when using a learning weight of 0.001, the mean squared error obtained is high.

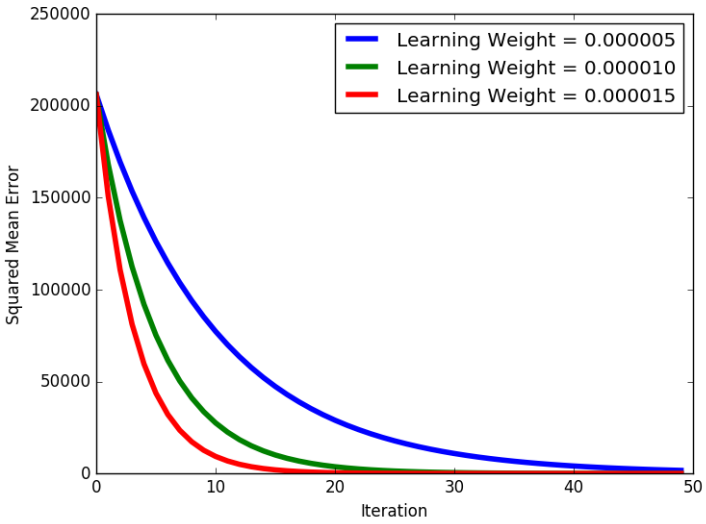
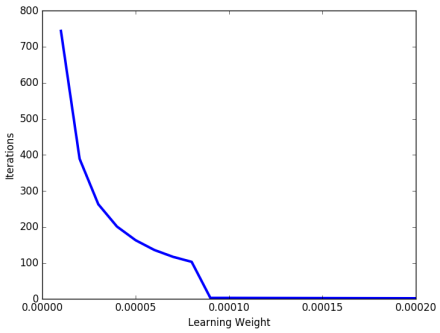
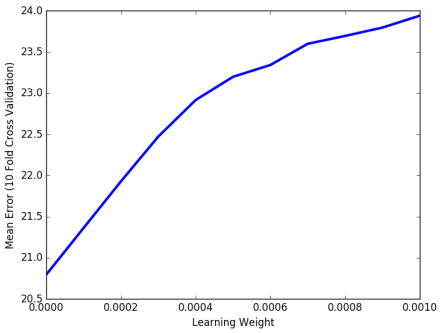


Figure 7: Convergence of gradient descent



(a) Learning Weight Vs Iterations



(b) Learning Weight Vs Mean Error

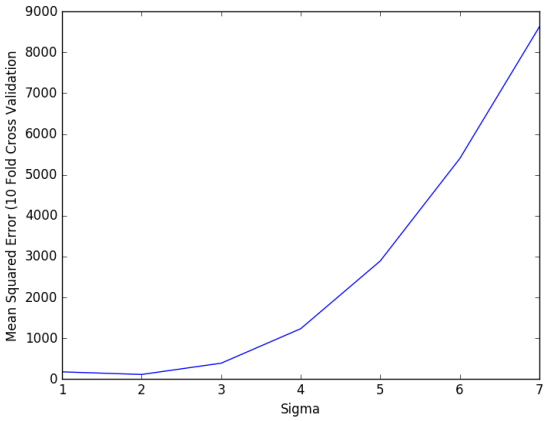


Figure 9: Dual Regression Performance for different values of σ

6 Dual Regression Problem

The last algorithm to solve the regression problem is the *dual regression problem*. Dual problem is frequently used when the number of features is bigger than the number of examples. This is because it reduces the number of unknown parameters by using one parameter per example. As the Kernel Function I used the *Gaussian Kernel Function*, which measures similarity with the function $exp(-(\frac{||x-y||^2}{2\sigma^2})$.

The performance obtained for the dual regression was similar to the other algorithms. In this case, an important parameter is the value of σ which is used in the gaussian kernel function. If the value of σ is too big, the kernel function will give low values, meaning that the vetcors are not similar. If the value is too small, the kernel function will give big values, meaning that the vetcors are similar.

Figure 9 shows different performance for different values of σ. Note that there is an improvement when switching from 1 to 2. However, the performance degrades when using larger values of σ.

6.1 Time Performance

The dual regression problem experiments showed that it is very expensive in terms of time. For example, for the CCPP data test which has 9568 examples and 4 features it takes around 50 seconds. In the contrary, both the explicit primal and the iterative solutions take less than a second.

As mentioned previously, th advantage of doing the dual regression is when the number of features is greater than the number of examples. In these cases, the tim performance of the dual solution would be better compared to the other algorithms.

7 Conclusions

During this assignment I could implement, test and evaluate the different algorithms to solve parametric regression. I showed that the algorithms could get to an optimal

solution, and they should be tunned with the right parameters.

I showed that the training set size has an important impact in the performance of the algorithm. Also, I went through different models and concluded that not all the models make a good fit to the data.

The three proposed solutions arrived to very similar results. In terms of time performance, the iterative solution is highly dependent on the learning weight. If the learning weight is small, then the iterative solution will have to do more iterations until converges, thus taking more time.