Cs584 Assignment 1: report

**Camila García**

**Department of computing science**

**Illinois Institute of Technology**

**February 22th, 2016**

# 1 Problem Statement

In this homework it was needed to do several different kinds of regression on datasets with both single and multiple features. Getting to know each of the possible ways to regress a model and to get the chance to play around with the different parameters for each of them makes the theory much easier to understand. Moreover, I believe it also gives me the ability to judge each model and each technique properly, and gives me more experience to get better at deciding which one to use in future problems. This is of course important to get a better grip at the subjects treated in class, but is, in my opinion, especially important to start getting practical experience in this field.

# 2 Proposed solution

## Single feature

1. The loading of the data is done manually, reading through the lines containing the data (that is, not starting with a ‘#’) and loading the data into arrays that can be later turned into *numpy* arrays for a better handling of the data and returned. With this, the data can be plotted using the *matplotlib* library. The two methods required for this solution are **load\_data** and **plot**.
2. To fit the single feature data to a linear model, the linear algebra formula given in class was used:

Where b is the unknown theta vector and A is the following matrix:

Using the tools provided by python libraries, it is rather easy to compute this matrix and finally get the theta coefficients by solving the equation. Once this is done, the testing errors can be calculated by following the RSE formula after calculating , using the theta coefficients. This is done ten times, as specified by the ten-fold cross validation, calculating the testing and training error for each of the different partitions. The data can be plotted by applying the y-hat function to a wide range of x values and using a line to plot. The functions used for this solution are **cross\_validation**, **y\_hat**, **compute\_error** and **plot.**

1. The results are compared by using previously built in regression functions in python, and comparing the testing and training errors. The errors are used using cross validation too, to ensure they can be compared. The fucntions used for this solution are **py\_reg, py\_compute\_error** and **py\_cross\_validation.**
2. To test different polynomial methods, that x data vector is turned into the Z matrix with the necessary components elevated to the given degree, using python libraries. With this matrix it is once again possible to use the linear algebra function given in class to find the theta vector:

Once again, this allows us to calculate the training and testing errors using the ten-fold cross validation, both for the new algorithm and for python functions. The smallest sum of training and testing error is then found between all the analyzed degrees. This algorithm allows for only the data of the selected model to be plotted, or for all of them to be plotted by changing a parameter. The functions used for this solution are **cross\_validation, poly\_linreg, compare, plot, y\_hat, compute\_error, py\_reg, py\_compute\_error** and **py\_cross\_validation**.

1. To reduce the amount of training data, the procedure for the last point is followed, but instead of using 10-fold cross validation, 4-cross and 2-cross validation is used, reducing the percentage of training data from 90% to 75% and 50%. This change is made in the **run** function.

## Multiple feature

1. The loading of the data is done in pretty much the same way as it was done for the single feature case, with the exception that the feature data is now loaded into a *numpy* matrix instead of an array. Python *sckit* library is used to map the features into a higher dimension. The functions used for this solution are **load\_data** and **poly\_reg.**
2. When the data is mapped to a higher dimension, the Z matrix is created. That means that the previous method of regression used for the polynomial single feature regression can also be applied here, as well as the functions to calculate the error and the y\_hat. Finally, the model is chosen based on the smallest sum of errors. The functions used for this solution are  **cross\_validation\_multi, y\_hat, poly\_reg** and **compute\_error.**
3. For the iterative solution, the Newton method discussed in class was used. This method iterates over theta’s values until a point is reached where it is considered the iteration is making no more progress. That is interpreted as stopping when the difference of consecutive thetas is small enough that it won’t go lower. This threshold is hand chosen, and is currently 1e-15. To calculate each following theta the next function is used:

Finally, when the difference is small enough, we have the theta vector which we can use to calculate y\_hat and the different errors it produces for each degree. These can be now compared to the ones obtained with the previous algorithm. The functions used for this solution are **cross\_validation\_newton, y\_hat, compute\_error** and **calc\_theta**.

1. For this problem, it was necessary to build the gram matrix using the Gaussian kernel function:

Proven to be valid in class. This is done by iterating with two loops over each x feature data row and calculating the respective kernel function. Once this is done, alpha can be found like this:

and the theta vector can be found like this:

With the theta vector, it is only a matter of calculating y\_hat and the errors using 10-fold cross validaiton.

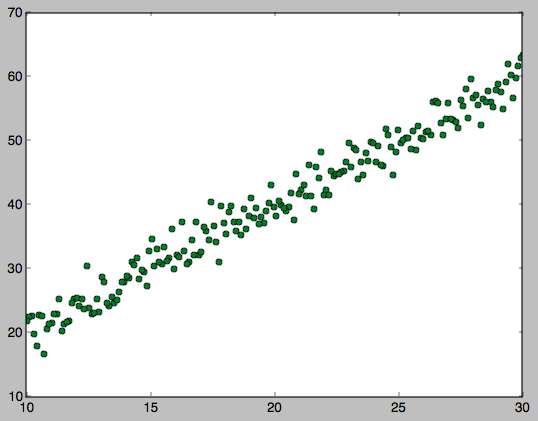
# 3 Implementation details

* Design issues are present in the program mostly as a learning experience. Some of the functions could have been implemented as a single function, but because the easiest problems were resolved first, it didn’t happen.
* Some of the biggest problems included getting familiar with the python language and the different libraries used for the program. Because they were mostly unknown, a long time was spent researching their correct use and debugging problems caused by a poor understanding of some of the functions used. I think it was also hard to put the theory into practice, as it might seem that the theory is understood completely but this understanding is challenged at the time of implementation. This, however, also helped me understand the theory much better.
* To run the program for each of the homework’s exercises it is enough to run the each file (homework\_p1.py for single feature and homework\_p2.py for multiple features) with **python 3**. It is necessary to inform that each time a plot is shown, it stops execution until it is manually closed. However, it is recommended to run the ipython notebook files as it was the intended way for the results to be shown. The maximum degree to compare can be changed by changing the **run** function’s parameters, if wished. Data files should be in the same folder.

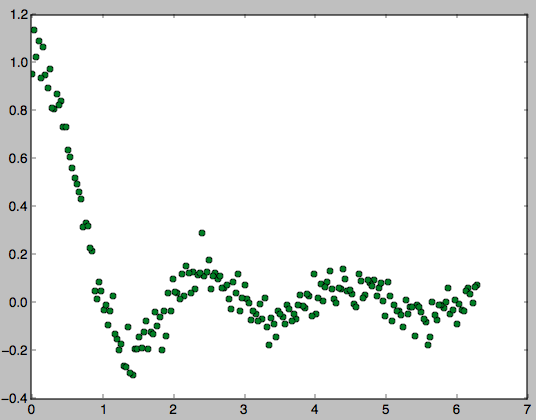
# 4 Results and Discussion

## Single feature

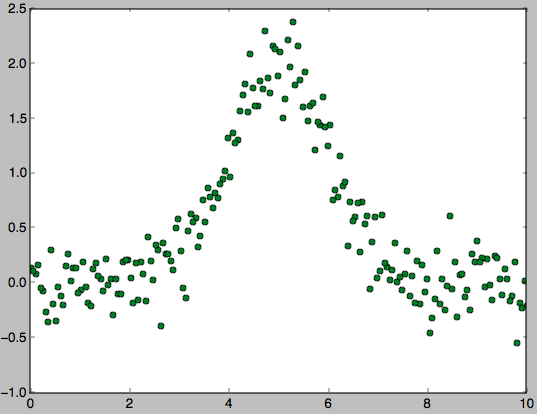
The output plot for each data given is:



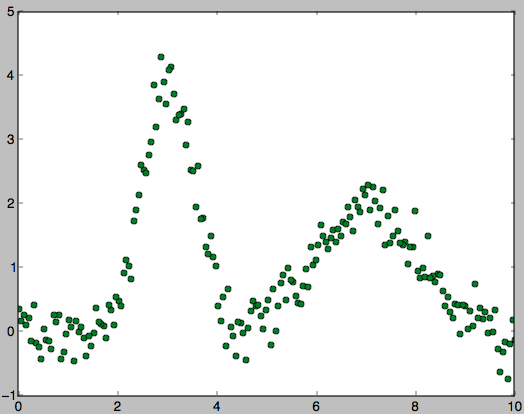
Data Set



Data Set



Data Set



Data Set

This inmmediately gives us a better understanding of the problem at hand. Even though it is clear that a simple regression won’t be good enough for data sets 2 through 4, the results are as follows:

### Data Set 1:

The RSE training error is: 0.003673

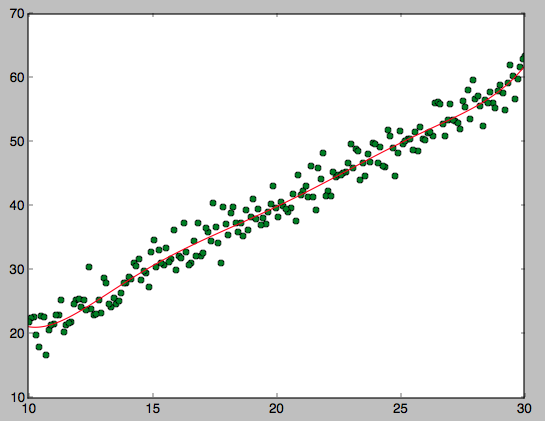
The RSE testing error is: 0.003845

The theta coefficients are:

theta\_0: 0.261203288658

theta\_1: 1.98610257105

Plot:



This fits very good, as expected.

### Data Set 2:

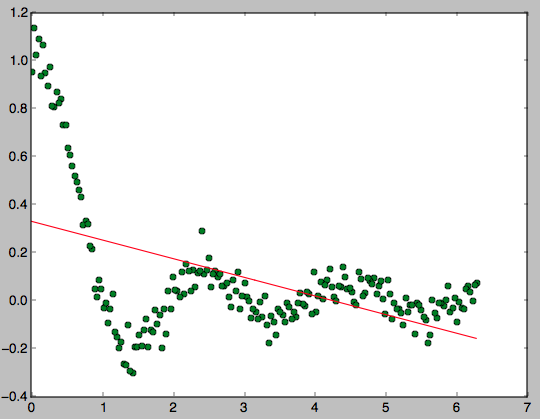
The RSE training error is: 100.343652

The RSE testing error is: 107.806416

The theta coefficients are:

theta\_0: 0.332447984563

theta\_1: -0.0776894595496



### Data Set 3:

The RSE training error is: 97.517621

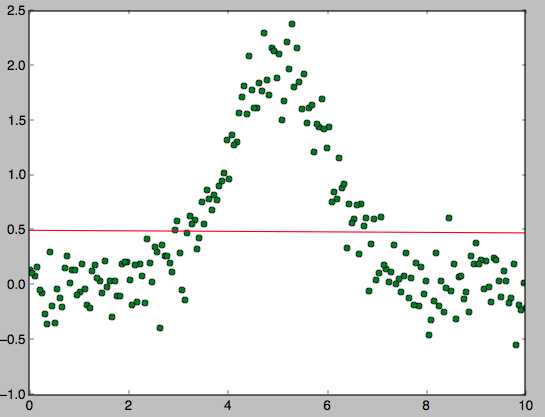
The RSE testing error is: 100.053993

The theta coefficients are:

theta\_0: 0.501018430927

theta\_1: -0.00234680272593

Plot:



### Data Set 4:

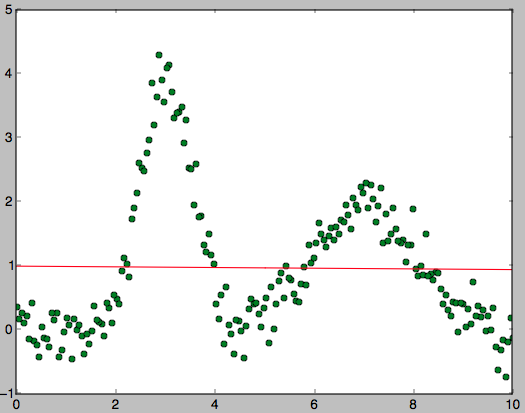
The RSE training error is: 1719.524581

The RSE testing error is: 1644.487543

The theta coefficients are:

theta\_0: 0.997775231806

theta\_1: -0.00537297700131



As expected, the fit is not at all good for these three data sets, as evidenced by their huge training and testing errors.

So, we now test different degrees up until degree 6 for each data set, and compare their errors and how they change with different percentages, ordered by smallest sum of errors for data set 1:

Using 90% for testing:

Using degree 6.

Training error: 0.003445

Python training error:0.003445

Testing error: 0.003974

Python testing error:0.003974

Using 75% for testing:

Using degree 6.

Training error: 0.003393

Python training error:0.003393

Testing error: 0.003949

Python testing error:0.003949

Using 50% for testing:

Using degree 6.

Training error: 0.003261

Python training error:0.003261

Testing error: 0.004045

Python testing error:0.004045

Using 90% for testing:

Using degree 1.

Training error: 0.003673

Python training error:0.003673

Testing error: 0.003845

Python testing error:0.003845

Using 75% for testing:

Using degree 5.

Training error: 0.003483

Python training error:0.003483

Testing error: 0.004041

Python testing error:0.004041

Using 50% for testing:

Using degree 5.

Training error: 0.003384

Python training error:0.003384

Testing error: 0.003952

Python testing error:0.003952

Using 90% for testing:

Using degree 2.

Training error: 0.003660

Python training error:0.003660

Testing error: 0.003911

Python testing error:0.003911

Using 75% for testing:

Using degree 4.

Training error: 0.003511

Python training error:0.003511

Testing error: 0.004050

Python testing error:0.004050

Using 50% for testing:

Using degree 4.

Training error: 0.003461

Python training error:0.003461

Testing error: 0.003903

Python testing error:0.003903

Using 90% for testing:

Using degree 4.

Training error: 0.003562

Python training error:0.003562

Testing error: 0.004023

Python testing error:0.004023

Using 75% for testing:

Using degree 1.

Training error: 0.003635

Python training error:0.003635

Testing error: 0.003961

Python testing error:0.003961

Using 50% for testing:

Using degree 3.

Training error: 0.003538

Python training error:0.003538

Testing error: 0.003928

Python testing error:0.003928

Using 90% for testing:

Using degree 5.

Training error: 0.003533

Python training error:0.003533

Testing error: 0.004082

Python testing error:0.004082

Using 75% for testing:

Using degree 2.

Training error: 0.003617

Python training error:0.003617

Testing error: 0.003999

Python testing error:0.003999

Using 50% for testing:

Using degree 2.

Training error: 0.003580

Python training error:0.003580

Testing error: 0.003903

Python testing error:0.003903

Using 90% for testing:

Using degree 3.

Training error: 0.003615

Python training error:0.003615

Testing error: 0.004002

Python testing error:0.004002

Using 75% for testing:

Using degree 3.

Training error: 0.003560

Python training error:0.003560

Testing error: 0.004135

Python testing error:0.004135

Using 50% for testing:

Using degree 1.

Training error: 0.003593

Python training error:0.003593

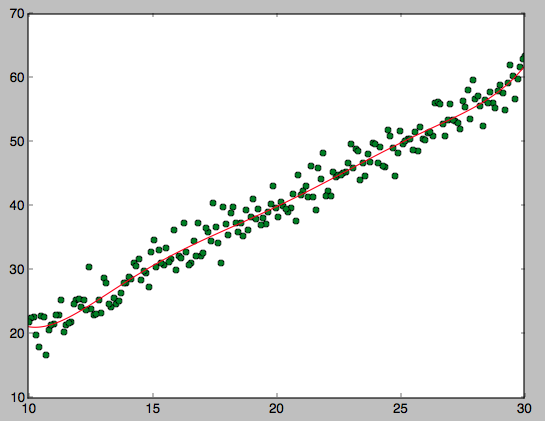
Testing error: 0.003930

Python testing error:0.003930

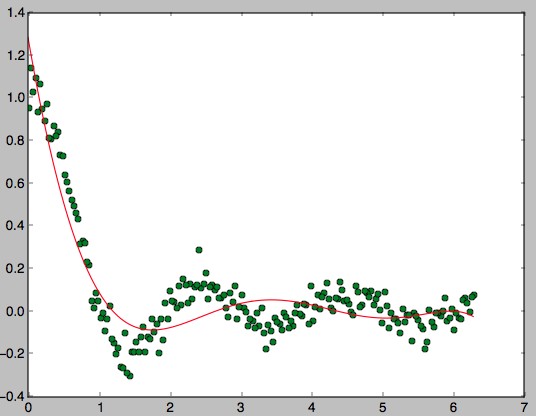
Here, we can observe multiple things. For one thing, it is possible to see that the testing and training errors are indeed the same ones as calculated by python functions, which is a sign of a well implemented algorithm. It is also possible to see that the testing error is generally bigger, as is to be expected. However, the percentages don’t have a very noticeable impact in the errors. This could be explained because less training data reduces variance but also increases bias, which both contribute towards the error.

The listing is not shown for the following data sets as they all seem to show the same characteristics. However, we will show the chosen model for each of the data set and its respective plot:

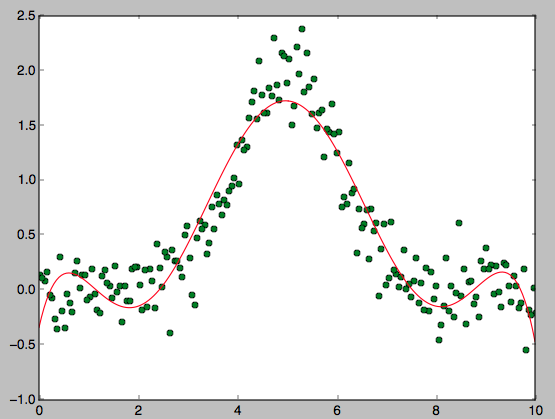
The regression with degree 6 has the smallest average of errors, and is therefore chosen (using 90 percent for testing). Plot:



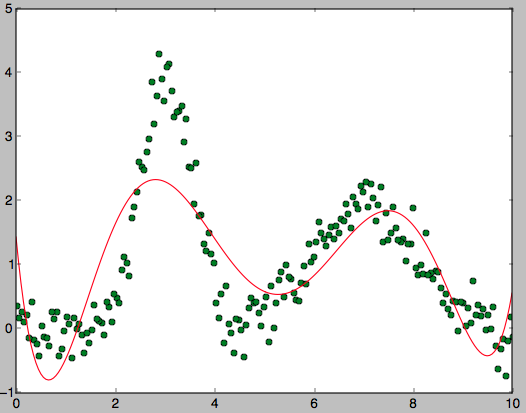
The regression with degree 6 has the smallest average of errors, and is therefore chosen (using 90 percent for testing). Plot:



The regression with degree 6 has the smallest average of errors, and is therefore chosen (using 90 percent for testing). Plot:



The regression with degree 6 has the smallest average of errors, and is therefore chosen (using 90 percent for testing). Plot:



It is important to note that this method of choosing might be causing an overfit of the data, as the highest degree seems to always be chosen. This could be bettered by a manual labor of looking at the graphs to decide. For example, we could decide that a better fit for data set 1 is actually the simple linear regression with degree 1.

## Multiple features

The output of all data sets is presented and discussed, with the exception of the gaussian kernel function, which seems to have a very hard time running the data sets with a higher number of examples, and is only run for the first data set.. This is due to the fact that the gram matrix is of size mxm, and as m increases, the number of iterations the gaussian function has to perform increases dramatically.

**Data set 1**

For degree 2:

Time for explicit method was: 0.028508

Time for iterative method was: 0.098884

Explicit Iterative

Training error 0.258184561507020850.2596166333121631

Testing error 0.258184561507020960.2596166333121632

For degree 3:

Time for explicit method was: 0.034438

Time for iterative method was: 0.102735

Explicit Iterative

Training error 0.25749801005237450.26066030119483397

Testing error 0.257498010052374440.26066030119483397

For degree 4:

Time for explicit method was: 0.041602

Time for iterative method was: 0.116993

Explicit Iterative

Training error 0.25629458290637520.2603054771042634

Testing error 0.256294582906375360.26030547710426316

Gaussian kernel:

Time elapsed was: 632.014813

Training error: 1995288355792247324672.000000

Testing error: 2015602069853649764352.000000

**Data set 2**

For degree 2:

Time for explicit method was: 0.027425

Time for iterative method was: 0.097202

Explicit Iterative

Training error 0.0199027676400618770.020037595733541413

Testing error 0.0199027676400618770.02003759573354142

For degree 3:

Time for explicit method was: 0.032389

Time for iterative method was: 0.101427

Explicit Iterative

Training error 0.0102984689620366170.010402279276244758

Testing error 0.0102984689620366190.010402279276244762

For degree 4:

Time for explicit method was: 0.038582

Time for iterative method was: 0.108203

Explicit Iterative

Training error 0.01028715971831670.010439267920393043

Testing error 0.0102871597183167020.010439267920393042

**Data set 3**

For degree 2:

Time for explicit method was: 1.683877

Time for iterative method was: 4.466816

Explicit Iterative

Training error 0.250704832520689370.2508363347199878

Testing error 0.25070483252068940.2508363347199879

For degree 3:

Time for explicit method was: 3.622288

Time for iterative method was: 6.665207

Explicit Iterative

Training error 0.250623709285978160.2509436613825019

Testing error 0.25062370928597830.2509436613825019

For degree 4:

Time for explicit method was: 8.903549

Time for iterative method was: 12.069670

Explicit Iterative

Training error 0.250426408069515660.25111545892110676

Testing error 0.250426408069515660.2511154589211065

**Data set 4**

For degree 2:

Time for explicit method was: 1.670986

Time for iterative method was: 4.481821

Explicit Iterative

Training error 0.00388690298932496460.003888310776487546

Testing error 0.0038869029893249640.003888310776487547

For degree 3:

Time for explicit method was: 3.526022

Time for iterative method was: 6.509479

Explicit Iterative

Training error 0.00388588956147888160.0038894914547551616

Testing error 0.003885889561478880.0038894914547551616

For degree 4:

Time for explicit method was: 8.808991

Time for iterative method was: 11.655184

Explicit Iterative

Training error 0.0034595423945154330.0034682541701159253

Testing error 0.00345954239451543370.0034682541701159266

It is possible to observe from this results that for one thing, the error for the explicit and iterative solutions does not differ by a noticeable amount, making them both just as accurate according to this standart.. We can also see that the amount of time differs only slightly, and that the time for the iterative solution grows slightly as both degree and number of examples increase.

It is also very important to note that the time it takes the gaussian to complete is extremely high, and the errors it produces even more so. This can be attributed to the fact that in applying the gaussian kernel function, we are increasing the number of features and ideally we would always like to have more examples than features, hence this is not a good example to apply gaussian.