cs584 Assignment 1: Report

Arun Rajagopalan
Graduate student, A20360689
Department of Computer Science
Illinois Institute of Technology
February 22, 2016

Abstract:

The problem or the task is to implement techniques for parametric regression. Datasets used are the sample datasets provided in cs584 website, both synthetic and real. It is crucial that we calculate and analyze the performance of our algorithms and we should use 10-fold cross validation for that purpose. It is also to be noted that we cannot use the direct python functions for regressions, but have to implement them ourselves.

1 Problem statement:

To implement techniques for parametric regression. Two main types or models of regression are linear and polynomial and we have to implement techniques for both. Also, the experiments need to conducted with datasets that have single feature and multiple features as well. Based on the experiment outcomes, best model for the given datasets has to be decided.

2 Proposed solution:

The solution intends to perform regression using own functions and validate the results against in-built python functions. The major own functions implemented include that used for creating data(z) matrix, solving for theta values, calculating mean square errors, predicting outcomes, dividing data into folds and to perform cross validation. Also own functions were implemented for performing gradient descent, calculation of iterative/explicit solution, gaussian fitting and gaussian prediction. Plots were used for visualization wherever necessary and rest of the outputs are shown on the terminal screen.

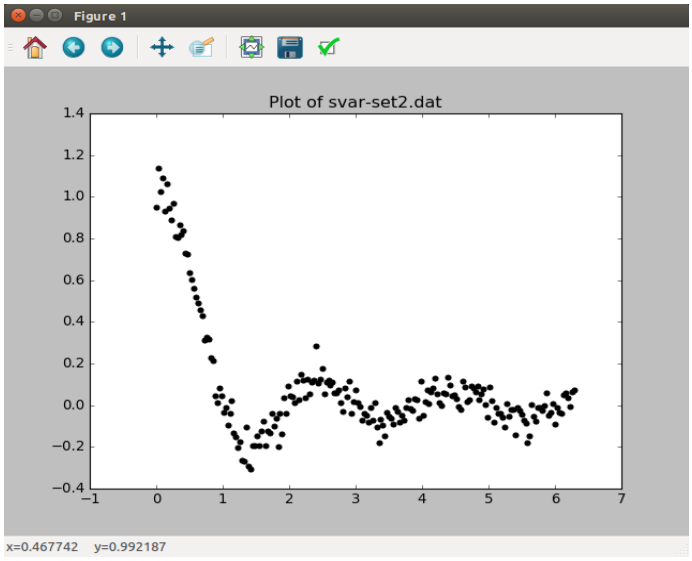
3 Implementation details:

To implement the solution, python 2.7.3 was used. Main packages used include 'numpy' (for numerical processing and sklearn/scipy packages for some regression related functions. Direct python functions that do regression are not used unless for validation.

Single variate:

First, we read the input files and get data (z) matrix and outcomes/labels matrix from them. The read data has two columns with the last column being our label/outcome column. The last column is saved as it is and the first column is saved with a column of ones inserted before it. These are our outcome and data matrices for calculating linear model. For polynomial model, feature vector is extended with powers of feature up until the specified degree. That is, for linear model involving single feature x, data matrix is of form [1, x] while for polynomial model of degree n, it is of the form [1, x, x^2, x^3, …. x^n].

Then we plot the datasets to get an idea of the problem. The picture shown below is the visualization of svar-set2.dat dataset.



Now, after getting data matrix and outcomes matrix, we divide them into predefined number of folds. For this series of experiments, the indicated value of 10 is used. This is for cross validation.

Cross validation is an attestation technique by which the given data is divided into some n folds and each fold is used exactly once for testing and n-1 times for training. Thus 10 iterations of subsequent training and testing can be done with varying subsets of given data and we can report the average of all accuracies as our final accuracy. But to get the final model, training is done one last time with all the data after 10 iterations and this model is taken as our final model.

Now that we have divided our data into folds, we take 9 folds and combine them for training and use the remaining one for testing. We do this iteratively as mentioned above. Training and testing accuracies are calculated by solving for theta vector, using theta vector to predict the outcomes and checking whether the predicted outcome and the actual outcome are close or not. This closeness can be determined by using several methods like calculating mean square error or relative mean square error. For this series of experiments, the mean square error is used.

Solving linear model:
We form A and B matrices, with A having the structure [[Feature length, Feature sum],[Feature sum, Feature squared sum]] and B having the structure [[sum(outcomes)], [sum of product of features and outcomes)]]. We solve these two matrices with numpy.linalg.solve to get theta values.

Solving polynomial model:
Similar to the above, we use numpy.linalg.solve with the dot product of z and z^T and the dot product of z^T and y(outcomes) to get theta values.

Predicting and validating outcomes:
Once we get theta values, we predict the labels using training and testing datasets. Training estimate is the dot product of training data matrix and the theta vector and testing estimate is that of testing data matrix and theta vector. Then we calculate the mean square error between the predicted labels and the actual labels. Mean square error is nothing but the squared difference between predicted and the actual labels, divided by the number of examples/instances. Lastly we find the average training and testing errors for each model (linear and different degree polynomial models) on every dataset and choose that model as the best which has least errors and takes less time to compute.

Multi variate:

The single variate model is not an effective one. There is not much that we can predict by seeing a single feature. Usually for real world purposes, we use multi variate models only. Multi variate model is the one that derives its decisions based on the values of more than one feature vector. For this series of experiments, we perform Linear regression on multi feature datasets.

First we load the multiple datasets and map them to a higher dimensional space. Once we finalize upon the degree we need, we get a data matrix creator by using the PolynomialFeatures function available in the sklearn python stack. When we call this function with degree, it gives us a feature extending object which in turn has a function called fit_transform and this fit_transform function when called with the input data, extends the data. The outcome matrix is same as the previous case, the last column of the input file. By the end of this stage, we have data matrix (mapped to higher dimensional space) and the outcome matrix.

Then, again as the previous method, we split the data matrix and the outcome matrix into folds, solve for theta values, estimate the labels using theta values with the training and testing datasets and from the similarity between the training/testing estimates and the actual outcome matrix, we calculate the accuracies. Based on accuracies, we choose the model that will be the best match for our purpose.

Solving for theta:
Here to get the theta values, we apply pseudo-inverse operation on our input data and perform a dot product with this pseudo-inverse result and the outcomes matrix to get theta vector.

Predicting and validating outcomes:
The process of predicting and validating is same again, we do the dot product of training/testing data matrices with theta vectors to get training/testing estimates and we compare them with actual outcomes to scale the effectiveness of models. Here too, the mean square error is used as the measure of model effectiveness.

Explicit solution and iterative solutions:

The explicit solution is derived by calculating theta values, estimating outcomes and later comparing the results with actual outcomes. But iterative solutions use the concept of gradient descent to solve the regression problem. Gradient descent first guesses theta values and then iteratively updates its guess by subtracting the gradient of the objective multiplied by learning rate. We also specify a iteration count until which gradient descent updates the theta values.

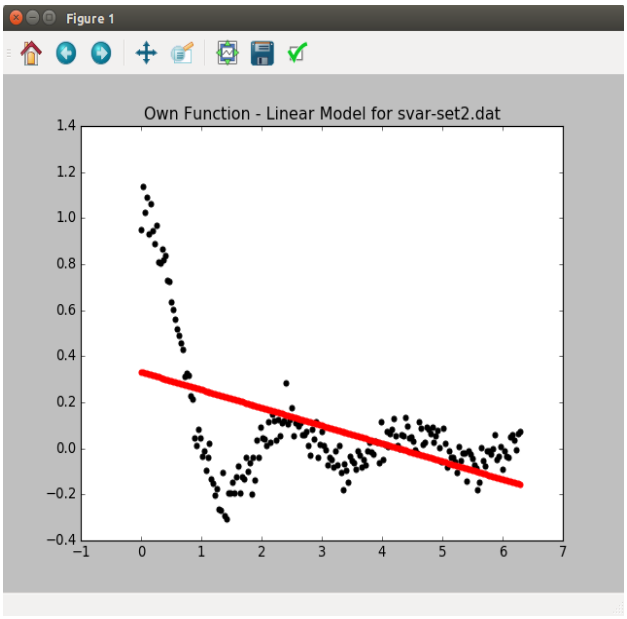
Dual regression problem:

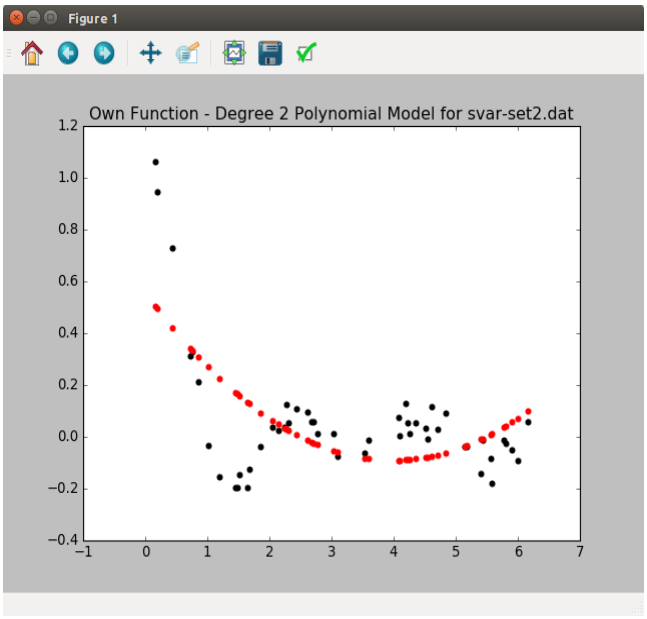
Dual regression problem is where we try to estimate labels from the data itself. Here theta is a weighted sum of input data and the weight we use is the alpha. We first calculate the gram matrix using cdist function available in scipy. Then we solve gram matrix and outcomes to get alpha with which predict outcomes by doing dot product with another gram matrix calculated from data and a data sample.

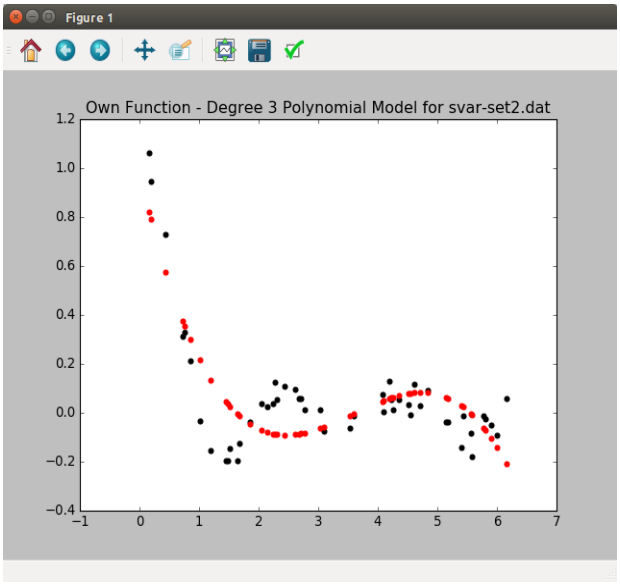
4 Results and discussion:

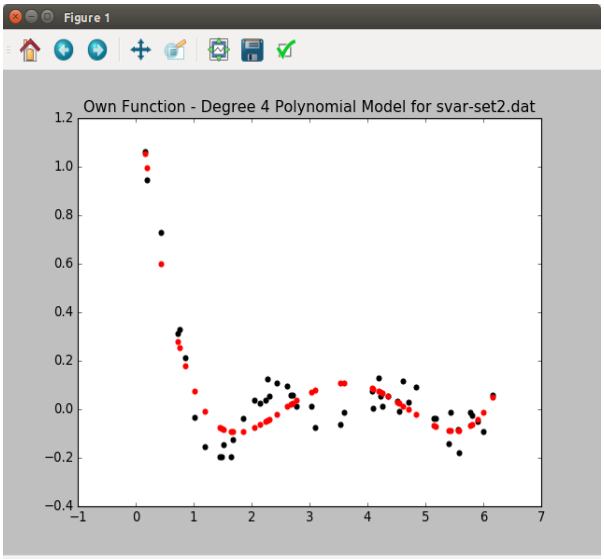
Single Variate:

The visualizations of linear and several variants of polynomial models for the single variate dataset svar-set2.dat are as follows.










```
rasuishere@ARajago6:~/Desktop/ml1/code$ python 1_single.py
Linear Model - Own Function - 10 fold cross validation
Theta Values
[[ 0.31737393 -0.07496823]
 [ 0.35026946 -0.08096405]
 [ 0.34529228 -0.08096825]
 [ 0.33611248 -0.07841667]
 [ 0.30030491 -0.07078898]
 [ 0.33514599 -0.07721364]
 [ 0.36012327 -0.08272288]
 [ 0.3363195  -0.07923753]
 [ 0.31099357 -0.07284993]
 [ 0.33283915 -0.07891878]]
Training Error          Testing Error
0.060502                0.051884
0.060114                0.055623
0.061359                0.043916
0.059983                0.055913
0.056477                0.090111
0.060471                0.051674
0.059262                0.064615
0.057541                0.077951
0.057203                0.082064
0.061960                0.038255
Mean Errors: Training: 0.059487 --- Testing: 0.061201
```

```
Linear Model - Pre-built Python Function - 10 fold cross validation
Theta Values
[[-0.         -0.07496823]
 [-0.         -0.08096405]
 [-0.         -0.08096825]
 [-0.         -0.07841667]
 [-0.         -0.07078898]
 [-0.         -0.07721364]
 [-0.         -0.08272288]
 [-0.         -0.07923753]
 [-0.         -0.07284993]
 [-0.         -0.07891878]]
Training Error          Testing Error
0.060502                0.051884
0.060114                0.055623
0.061359                0.043916
0.059983                0.055913
0.056477                0.090111
0.060471                0.051674
0.059262                0.064615
0.057541                0.077951
0.057203                0.082064
0.061960                0.038255
Mean Errors: Training: 0.059487 --- Testing: 0.061201
```

```
Degree 2 Polynomial Model - Own Function - 10 fold cross validation
Theta Values
[[ 0.58505844 -0.3533188   0.04453839]
 [ 0.58715788 -0.34400319  0.04309869]
 [ 0.41243463 -0.2365782   0.02912972]
 [ 0.50652233 -0.3002134   0.03756764]
 [ 0.5967165  -0.34455637  0.04269833]
 [ 0.46661482 -0.28434366  0.03706553]
 [ 0.6298939  -0.34603078  0.04241924]
 [ 0.56335675 -0.3274257   0.04138767]
 [ 0.60084328 -0.33543775  0.04083795]
 [ 0.56278959 -0.33619662  0.0430167 ]]
Training Error          Testing Error
0.037518                0.022612
0.036202                0.032217
0.030862                0.097342
0.036490                0.031812
0.036929                0.026644
0.032823                0.072357
0.030519                0.093255
0.038355                0.012536
0.033494                0.059825
0.039058                0.006438
Mean Errors: Training: 0.035225 --- Testing: 0.045504
```

```
Degree 3 Polynomial Model - Own Function - 10 fold cross validation
Theta Values
[[ 0.98203155 -1.05055595  0.32327626 -0.02989426]
 [ 1.01003635 -1.1060798   0.34978892 -0.03323228]
 [ 0.87761349 -0.93917219  0.29032815 -0.02694861]
 [ 0.945876   -1.03220108  0.32145815 -0.02993527]
 [ 1.0095215  -1.07633959  0.33106141 -0.03058182]
 [ 0.94224718 -1.03824057  0.32308961 -0.03008892]
 [ 0.99140496 -1.00123226  0.30038922 -0.02740032]
 [ 0.99269598 -1.07847249  0.33233573 -0.03059957]
 [ 0.98781866 -1.03497971  0.31422079 -0.02881227]
 [ 1.00099311 -1.09511894  0.33901609 -0.03133379]]
Training Error          Testing Error
0.019386                0.001756
0.016186                0.036503
0.017480                0.023743
0.018259                0.012928
0.017691                0.017709
0.017043                0.025521
0.014899                0.048429
0.018383                0.011944
0.016833                0.025636
0.017889                0.016552
Mean Errors: Training: 0.017405 --- Testing: 0.022072
```

```
Degree 4 Polynomial Model - Own Function - 10 fold cross validation
Theta Values
[[ 1.38548173 -2.28595681  1.19654849 -0.24392684  0.01692197]
 [ 1.37191686 -2.20486313  1.13564917 -0.22907851  0.01576762]
 [ 1.36733971 -2.24018912  1.16688722 -0.23754416  0.01648478]
 [ 1.356828   -2.25265407  1.18665539 -0.24339274  0.01697673]
 [ 1.38845962 -2.23545478  1.1516186  -0.23286546  0.01608587]
 [ 1.42572795 -2.30671323  1.18640312 -0.23965053  0.01657487]
 [ 1.34483429 -2.08553194  1.05643596 -0.21134055  0.01448126]
 [ 1.39448158 -2.28582363  1.19040209 -0.24198328  0.01676795]
 [ 1.36779452 -2.17906728  1.11570561 -0.22486377  0.015508  ]
 [ 1.39121322 -2.26576382  1.1717313  -0.23702445  0.01635473]]
Training Error          Testing Error
0.006852                0.006720
0.007094                0.004059
0.007424                0.001098
0.006029                0.014682
0.006783                0.006876
0.006644                0.009710
0.006352                0.013380
0.006506                0.009525
0.006466                0.010040
0.006685                0.007813
Mean Errors: Training: 0.006683 --- Testing: 0.008390
```

- As we see from all the figures above, we get best fitting model when we increase the degree, but care has to be taken not to increase the degree so much that we overfit the data.

- The best model for this svar-set2.dat and the other datasets is the degree 4 polynomial model as it fits the datasets better than linear and other polynomial models. Also as we could see from the error details, the training and the testing errors are least for the degree 4 polynomial model with the training error mean standing at 0.006683 and the testing error standing at 0.008390. This is very less when compared to the performance of the linear model which has a training error of 0.059487 and testing error of 0.061201.

- Also we get the same error values when we use the inbuilt python functions (LinearRegression, fit and predict functions.)

- One more thing that we could observe when we ran the experiments on different subsets of data (polynomial models were run on different subsets of data) is that linear model has high bias (error/distance between models and data points is high) and the polynomial model, especially degree 4 has high variance (difference between other subset models is high).

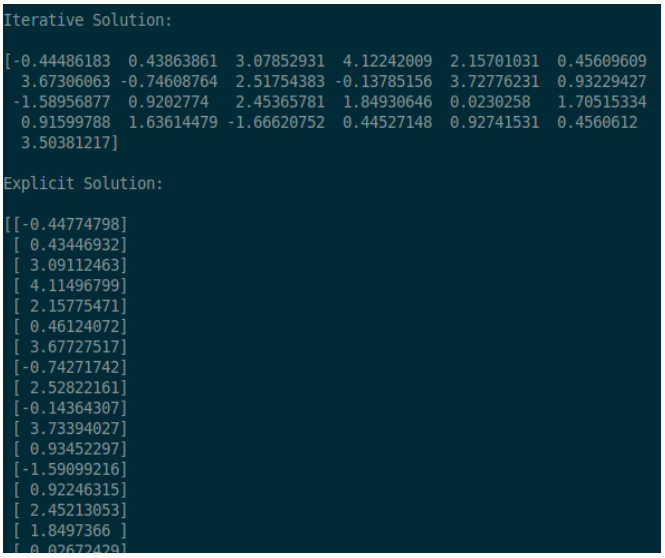
Multi Variate:

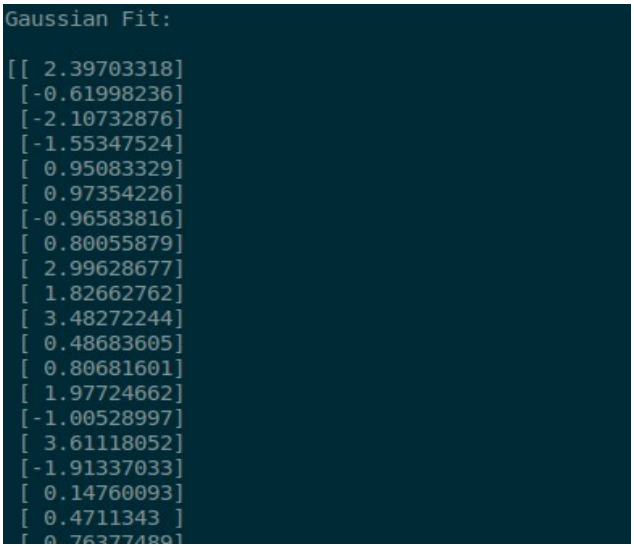
The terminal outputs for the multivariate program with mvar-set2.dat as input are as follows.

```
Mapping data to higher dimemsions and analyzing it - Degree: 6

Mapped data:
[[  1.00000000e+00   1.67346939e+00   4.48979592e-01 ...,   1.13799918e-01
    3.05316852e-02   8.19142774e-03]
 [  1.00000000e+00  -4.08163265e-02   5.30612245e-01 ...,   1.32061706e-04
   -1.71680218e-03   2.23184283e-02]
 [  1.00000000e+00  -7.75510204e-01  -1.59183673e+00 ...,   3.86161634e+00
    7.92647565e+00   1.62701342e+01]
 ...,
 [  1.00000000e+00  -2.00000000e+00  -6.93877551e-01 ...,   9.27238252e-01
    3.21694904e-01   1.11608436e-01]
 [  1.00000000e+00  -1.83673469e+00   1.02040816e+00 ...,   3.65753555e+00
   -2.03196419e+00   1.12886900e+00]
 [  1.00000000e+00  -2.85714286e-01   1.42857143e+00 ...,   3.39994390e-01
   -1.69997195e+00   8.49985975e+00]]
```

```
Predicted & actual outcomes - Output limited to initial 10 values
Predicted
[[ 3.0459166 ]
 [ 1.52501194]
 [-1.45728112]
 [-2.43673314]
 [ 1.41787641]
 [ 1.15435073]
 [-1.46485232]
 [ 1.20237545]
 [ 2.61374915]
 [ 2.52928222]]
Actual
[[2.861980967575097], [0.5195252261416367], [-2.245760962347731],
085317769937612], [0.8038785273144268], [3.18204127016383], [2.351

Mean Square Error for degree : 6 is 0.255133
```

```
Training Error          Testing Error
0.257600                0.264785
0.258196                0.259166
0.260973                0.235385
0.259365                0.248477
0.256997                0.269972
0.255306                0.286811
0.257189                0.268174
0.262928                0.216553
0.255843                0.280255
0.257447                0.266588

Gradient Descent:

[ 1.02445099  1.038856    0.91416978 -0.02779879 -0.00802481
 -0.00673911  0.00793707 -0.01318302  0.01883043]
```





- It is observable from the outputs that successful mapping of data to higher dimensional space was done and theta values were calculated, from which predictions were made and the accuracies were found out.

- Similar to the previous case, we get good accuracies in higher degree models for the given mvar-set2.dat dataset. Also we had calculated theta values and predictions using explicit and iterative solutions (gradient descent) and found that both are nearly same.

- Then we were also able to solve the dual regression problem (calculating theta as a weighted sum of input data) using a gaussian kernel. Though the accuracies were in the acceptable order, it was also noted that the dual problem solution takes more time to calculate.

Thus the given specifications were successfully met and the experiments were concluded.

5 References:

http://stats.stackexchange.com/
http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning