CS 584 Assignment 1 Report

Shreesh Kumara Bhat

February 22, 2016

Abstract

In this assignment, I implemented various algorithms for parametric regression. Used 10 fold cross validation method to estimate the general performance of the algorithms.

1 Problem Statement

Regression is a statistical method of analyzing relationships among attributes to predict the value of another attribute in a given data set. Regression is a problem of predicting a continuous variable from other attributes. Regression helps in understanding which attributes are important to estimate another attribute. If we look into the regression model's coefficients, we can understand this information.

2 Proposed solutions

2.1 Single Feature Linear Regression

Single Feature Linear Regression is applicable when there is only one attribute or feature in the data set and we would like to understand how it can be used to predict the value of another variable.

2.2 Single Feature Polynomial Regression

This algorithm works to the same data set as the previous one. The difference lies in the way it treats the data matrix. The feature of data set is mapped to higher dimensions using polynomial function $[1, X, X^2, X^3, \ldots X^n]$ See the code for Figure 2 in this section for an example.

2.3 Multiple Feature Regression

In Multiple Feature Regression, we map the features to higher dimensions by using polynomial function and combining features. For example, say the data set has 2 features : X_1, X_2 and degree of polynomial function is 2. Then, the generated higher dimension matrix will have features $[1, X_1, X_2, X_1^2, X_1 * X_2, X_2^2]$ See the code for Figure 2 in this section for an example.

2.4 Iterative solution using stochastic gradient descent

In this regression solution, the features are mapped to higher dimensions in the same way as multiple feature regression solution. However, parameters of the model are updated in each iteration over the modified data matrix to approach optimal solution using gradient descent until a specific error threshold is reached or max iteration value has been crossed.

2.5 Solving dual linear regression problem using Gaussian kernel function

In this regression problem, the features are higher in number compared to data samples available. Hence, to solve the problem efficiently we look into data samples instead of features to predict the output variable. We check each data sample row with given test data and do a similarity check. This method is called kernel function. In our case, we make use of Gaussian function to implement kernel similarity function to get a smooth transition of values.

3 Implementation details

3.1 Design issues and solutions

- The toughest design issue was **constructing Z matrix for second part of the problem. To construct Z matrix for general data set for any number of features and any degree was quite challenging. Took almost a day to implement it correctly.** Solved it by using itertools combinations with replacement and applying to each feature, degree combination.
- The second design issue was plotting single feature polynomial regression model. Tried various plotting methods to get a polynomial line in the plot. Settled for plotting predicted Y values instead.

3.2 Instructions to use IPython Notebook

- The assignment is divided across 2 notebooks. One for Single feature data set and another for Multi feature dataset.
- General way to execute cells are in top-down manner to ensure all necessary methods are defined. **Note :** All the data files should be present in a separate data folder where the ipython notebook will be present.
- Sample method calls are as follows:
- plot_data("svar-set1","Result1.pdf") # for plotting data and saving into pdf
- do_single_feature_linear_regression_experiment("svar-set1") # Loads data set without extension and does single feature linear regression. Prints the intercept and coefficients of scikit learn's model along with custom model for each fold. Then, prints average training error and testing error for custom model. Also, plots data points.

- do_single_feature_linear_regression_experiment("svar-set1",n_folds = 2,verbose = False) # Processes the same as the previous example, except doesn't print the coefficients of the models and plots the data. Instead, gives the flexibility of changing the fold number and prints the training error and testing error for each fold.

- do_single_feature_polynomial_linear_regression_experiment("svar-set2",n=2) # Does single feature polynomial regression with specified data & degree of polynomial = 2

- do_multi_feature_linear_regression_experiment("mvar-set1") # Does multi feature linear regression using polynomial degree = 2 and n_folds = 2 as default. Also, prints each fold's training and testing error as well as average across all folds.

- do_multi_feature_linear_regression_experiment("mvar-set2",degree=2, verbose = False) # Does same experiment as previously but allows flexibility of changing n_folds value, degree and suppressing each fold's error info.

- do_stochastic_gradient_descent_experiment("mvar-set1",max_iterations=10**8, error_threshold=0.01, learning_rate=1./(10**4)) # Does stochastic gradient descent regression and sets max_iterations, error threshold & learning rate for gradient descent algorithm.

- do_gaussian_kernel_regression_experiment("mvar-set2",sigma=1) # Does gaussian kernel regression to solve dual problem of regression with sigma = 1 and sets degree to 2 by default.

4 Results and discussion

- Plotted first data set in matplotlib. plot_data("svar-set1","Result1.pdf") See **Figure 1**

- Execute single feature linear regression on first data set and plot the model & print avg error. do_single_feature_linear_regression_experiment("svar-set1") . See **Figure 2**. **Analysis :** As we can see, training error is not the right estimate for how model will work for future test data. In this experiment, we find that test error is actually less compared to train error. Furthermore, I have compared the custom model with the scikit learn's model and find that **it is a perfect match**. (See **Figure 3**). Hence, have not compared their training and testing errors. I have **changed number of folds value** to reduce the amount of training data. For the first data set, training error increases as k fold value is increased. Logically, as we get more data, it should reduce training error. So, I m assuming it got different cross sections of data and becomes hard to predict with a linear model correctly. **For other data sets**, please refer ipython notebook used for homework. Fitting linear model for other data sets gives bad error because data is not linearly correlated.

- For second single feature data set, we will look into polynomial regression by calling : do_single_feature_polynomial_linear_regression_experiment("svar-set2",n=8). See **Figures 5,6,7** I choose degree 8 as model to be fit for

this data because values less than this gives underfit model and values more than this produces overfit model. This gives ideal value of testing error too, which gives a good estimate of how it will perform for future data. For dataset-3, degree will be 5 and for dataset-4, degree will be 8.

- For multivariate dataset 2, I perform linear regression in higher dimension and after varying the degree values, I observe that, **degree 3** gets highest accuracy while maintaining simple model as compared to degree 15 or so, which goes overfits the data. See **Figure 8**. Likewise, For dataset 1, degree will be 2. For dataset 3, degree will be 2.

- For iterative solution, using stochastic gradient descent, we ll compare the previous result with this model's output . The parameters which matters are max_iterations (which determine how many times theta will be at max, to avoid getting in an infinite loop), learning rate (if it is bigger than 0.001, it might skip over local minima and never get back. If it is too low, and max_iterations is not sufficiently big, it ll not reach local minima) and error_threshold (if it is around 0.001, it gives accurate result and breaks out of the loop in a proper manner). For this method, it might take more time to come to a convergence, but it doesn't face any problems of Non invertible matrix as the explicit solution. (see **Figure 9**)

- For dual regression solution, using gaussian kernel solution, we ll compare the data and model used in Figure 8. The main parameter which affects this model is sigma, which I have observed with different values that the value 1 gives good result. This method takes quite a long time to execute as compared to previous one because this method is meant to look into data instead of features & then apply kernel similarity function to it. See **Figure 10** for 10 fold cross validation testing and training error results and see **Figure 11** for time comparison between this model and the previous one.

-

5 Figures

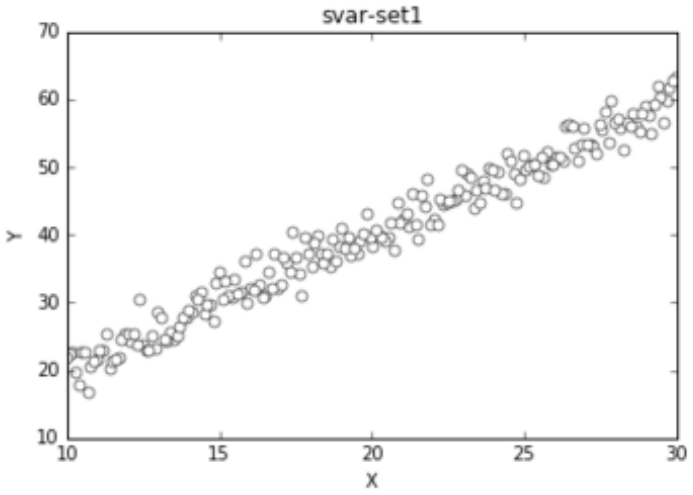

Figure 1: Load and plot data set 1

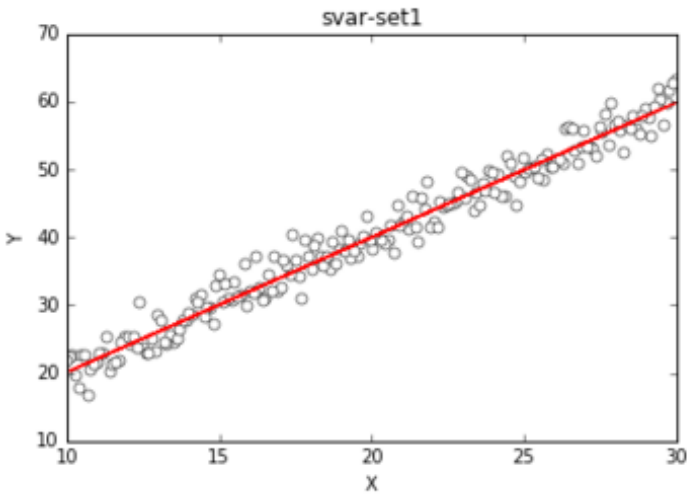

Figure 2: Avg Training error: 386.2078, Avg Testing error: 19.0557 for 10 folds

```
Differences in theta values of custom and scikitlearn's model
 Intercept of scikit's model : 0.327556
 Intercept of custom's model : 0.327556
 Intercept of scikit's model : 1.987673
 Intercept of custom's model : 1.987673
```

Figure 3: Comparison of custom model and sklearn's model for linear regression svar-set1

```
Custom model's Avg Training error : 16.3983 & Avg Testing error : 16.4769 for 2 folds
Custom model's Avg Training error : 36.3521 & Avg Testing error : 4.5930 for 5 folds
Custom model's Avg Training error : 162.4824 & Avg Testing error : 1.3289 for 8 folds
Custom model's Avg Training error : 386.2078 & Avg Testing error : 19.0557 for 10 folds
```

Figure 4: Changing k fold value for linear regression svar-set1

```
do_single_feature_polynomial_linear_regression_experiment("svar-set2",n=2)
```

```
Custom model's Avg Training error : 1030.4597 & Avg Testing error : 34.3051 for 10 folds
```

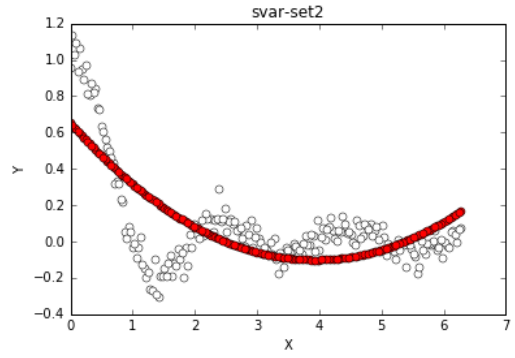

Figure 5: Under fit polynomial regression degree 2 for svar-set2

```
do_single_feature_polynomial_linear_regression_experiment("svar-set2",n=8)
```

```
Custom model's Avg Training error : 116.6110 & Avg Testing error : 5.4436 for 10 folds
```

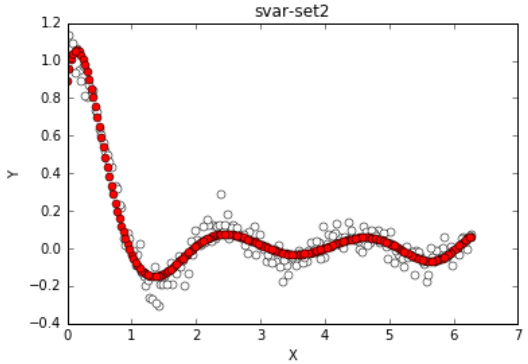

Figure 6: Correct fit polynomial regression degree 8 for svar-set2

```
do_single_feature_polynomial_linear_regression_experiment("svar-set2",n=10)
```

```
Custom model's Avg Training error : 116.8402 & Avg Testing error : 8.9512 for 10 folds
```

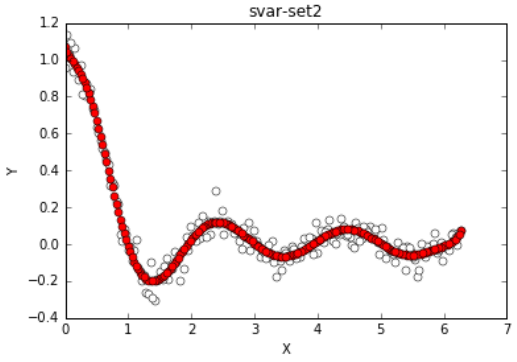

Figure 7: Over fit polynomial regression degree 10 for svar-set2

```
do_multi_feature_linear_regression_experiment("mvar-set2",degree=2, verbose = False)
```

```
Avg Training error : 0.0199 & Avg Testing error : 0.0200 for 10 folds
```

```
do_multi_feature_linear_regression_experiment("mvar-set2",degree=3, verbose = False)
```

```
Avg Training error : 0.0103 & Avg Testing error : 0.0104 for 10 folds
```

```
do_multi_feature_linear_regression_experiment("mvar-set2",degree=4, verbose = False)
```

```
Avg Training error : 0.0103 & Avg Testing error : 0.0104 for 10 folds
```

```
do_multi_feature_linear_regression_experiment("mvar-set2",degree=5, verbose = False)
```

```
Avg Training error : 0.0047 & Avg Testing error : 0.0048 for 10 folds
```

```
do_multi_feature_linear_regression_experiment("mvar-set2",degree=15, verbose = False)
```

```
Avg Training error : 0.0023 & Avg Testing error : 0.0026 for 10 folds
```

Figure 8: For mvar-set2, multi feature linear regression in higher dimension

```
do_stochastic_gradient_descent_experiment("mvar-set2",max_iterations=10**8, error_threshold=0.2, learning_rate=1/(10**4))
```

```
For Fold : 1
Training error : 0.0259 & Testing error : 0.0243
For Fold : 2
Training error : 0.0258 & Testing error : 0.0248
For Fold : 3
Training error : 0.0256 & Testing error : 0.0269
For Fold : 4
Training error : 0.0253 & Testing error : 0.0292
For Fold : 5
Training error : 0.0256 & Testing error : 0.0267
For Fold : 6
Training error : 0.0258 & Testing error : 0.0251
For Fold : 7
Training error : 0.0258 & Testing error : 0.0250
For Fold : 8
Training error : 0.0261 & Testing error : 0.0224
For Fold : 9
Training error : 0.0257 & Testing error : 0.0260
For Fold : 10
Training error : 0.0256 & Testing error : 0.0267

Avg Training error : 0.0257 & Avg Testing error : 0.0257 for 10 folds
```

Figure 9: For mvar-set2, stochastic gradient descent

```
do_gaussian_kernel_regression_experiment("mvar-set2",sigma=1)
```

```
For Fold : 1
Training error : 0.0000 & Testing error : 0.0029
For Fold : 2
Training error : 0.0000 & Testing error : 0.0032
For Fold : 3
Training error : 0.0000 & Testing error : 0.0030
For Fold : 4
Training error : 0.0000 & Testing error : 0.0035
For Fold : 5
Training error : 0.0000 & Testing error : 0.0033
For Fold : 6
Training error : 0.0000 & Testing error : 0.0029
For Fold : 7
Training error : 0.0000 & Testing error : 0.0030
For Fold : 8
Training error : 0.0000 & Testing error : 0.0028
For Fold : 9
Training error : 0.0000 & Testing error : 0.0032
For Fold : 10
Training error : 0.0000 & Testing error : 0.0033

Avg Training error : 0.0000 & Avg Testing error : 0.0031 for 10 folds
```

Figure 10: For mvar-set2, gaussian kernel dual regression

```
start_time = time.time()
do_gaussian_kernel_regression_experiment("mvar-set2")
print("Time taken :  %s seconds" % (time.time() - start_time))
```

```
For Fold : 1
Training error : 0.0000 & Testing error : 0.0029
For Fold : 2
Training error : 0.0000 & Testing error : 0.0032
For Fold : 3
Training error : 0.0000 & Testing error : 0.0030
For Fold : 4
Training error : 0.0000 & Testing error : 0.0035
For Fold : 5
Training error : 0.0000 & Testing error : 0.0033
For Fold : 6
Training error : 0.0000 & Testing error : 0.0029
For Fold : 7
Training error : 0.0000 & Testing error : 0.0030
For Fold : 8
Training error : 0.0000 & Testing error : 0.0028
For Fold : 9
Training error : 0.0000 & Testing error : 0.0032
For Fold : 10
Training error : 0.0000 & Testing error : 0.0033

Avg Training error : 0.0000 & Avg Testing error : 0.0031 for 10 folds
Time taken :  608.017000198 seconds
```

```
start_time = time.time()
do_multi_feature_linear_regression_experiment("mvar-set2",degree=2, verbose = False)
print("Time taken :  %s seconds" % (time.time() - start_time))
```

```
Avg Training error : 0.0199 & Avg Testing error : 0.0200 for 10 folds
Time taken :  0.309000015259 seconds
```

Figure 11: For mvar-set2, time comparison