Assignment 1 Report

Xiaopei Zhang   A20302816

I. Problem Statement

This assignment requires to implement various techniques in parametric regression from scratch, evaluate the performance of different algorithms, and figure out how to select the most suitable model for the given data sets. The first section of this assignment focuses on single variable regression. It includes both linear regression and polynomial regression. Mean squared error (MSE) is also implemented and used to select model for single variable data sets. The second section of this assignment is mainly around multivariate regression. The basic methods to map variables into higher dimension space is implemented. Gradient descent, as the iterative method, is also implemented and compared with the basic method. I also implemented Gaussian Kernel function to solve the dual linear regression problem and compare its MSE with that of the primal regression problem.

II. Proposed Solution

In the first section, I use the following steps to solve the linear and polynomial single variable regression problems.

1. Select model.

For linear regression:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

For polynomial regression:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$
$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$
$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

2. Define objective.

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2$$

$$\theta^* = \underset{\theta}{\arg\min}\ J(\theta)$$

3. Solve for parameters.

$$\nabla J(\theta)=0$$

$$\theta=(Z^TZ)^{-1}Z^Ty$$

4. Calculate the MSE. $MSE=\frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)}-y^{(i)})^2$

In the second section, I have the following different ways to solve the multivariate regression problems.

1. Higher dimensional model (one example for two variables).

$$\hat{y}=h_\theta(x)=\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1{}^2+\theta_4x_1x_2+\theta_5x_2{}^2$$

$$\hat{y}=h_\theta(x)=\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1{}^2+\theta_4x_1x_2+\theta_5x_2{}^2+\theta_6x_1{}^3+\theta_7x_1{}^2x_2+\theta_8x_1x_2{}^2+\theta_9x_2{}^3$$

2. Gradient descent. $\theta_1:=\theta_1-\alpha\frac{1}{\mathrm{m}}\sum_{\mathrm{i}=1}^{\mathrm{m}}((\mathrm{h}_\theta(\mathrm{x}^{(\mathrm{i})})-\mathrm{y}^{(\mathrm{i})})\cdot\mathrm{x}^{(\mathrm{i})})$

Feature scaling: $x'=\frac{x-\bar{x}}{\sigma}$

3. Gaussian kernel function. $f_1=\exp\left(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}\right)$

III. Implementation

1. Program Design

My implementations of all the functions are highly modular. According to the sequence of my implementation, I explained all the functions below. (Unused functions in my script are not listed below.)

load_data: read .dat file into a list of arrays, and return the list.

tenFold: shuffle a list of arrays, and yield a generator of training data set (90%) and testing data set (10%).

splitXy: split a list of arrays into two lists, of which one is all the features (X) and the other is the regressed variable (y).

getZLinear: add ones in front to transform X into Z for linear regression

regression: read Z and y, perform linear regression, and return theta

computeError: take theta, Z and y, and return MSE

plotLinearModel: plot linear regression line on testing data set

getZPolynomial: take a single variable data set X, and return its 2 to 5 degree polynomial data set

reduceTrainingData: take a dataset and return its first half

mapZ: map a multivariate data set X to its 2rd and 3rd higher dimensional spaces Z2 and Z3

gradientDescent: take initial theta, Z, and y, and return theta after convergence

gaussianKernel: take training data set and testing data set, and return the mapped K

dualLinearRegression: take X, y, and return alfa

In the main function, I first traverse the 4 single variable data sets and real data set 'iris.dat' and plot them. Then, under 10 fold cross validation I perform linear regression (plot line on testing data set) and 2 to 5 degree polynomial regression, and compare the results with those of Python regression. Also, I cut training data set into half, did linear and polynomial regression, and compare the results with my former results. After cross validation, I select the model which has the least averaged MSE. Second, I traverse the 4 multivariate data sets. Under 10 fold cross validation I perform linear regression after mapping X on higher dimensional spaces. Afterwards, I compare my explicit solution with gradient descent solution. I also performed dual linear regression with Gaussian kernel function, and compare its time performance and accuracy with my primal solution. Finally, I, again, selec the model which has the least averaged MSE.

2. Problems Encountered

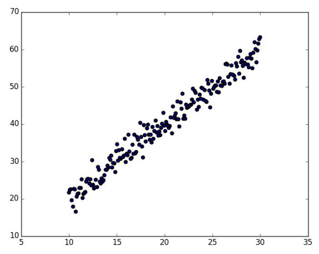
solve() does not always work when I use it to implement regression. Because many matrices are singular, I replaced solve() with dot(pinv()). I also find my implementations of dual linear regression and Gaussian kernel function increase MSE, and take much longer time to execute than my primal functions.

3. Instructions

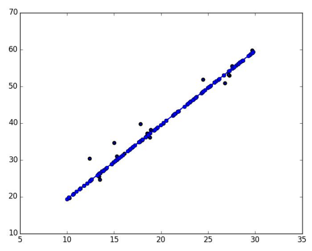
My script is written and tested under Python 2.7. I commented the lines (156-158, 169) that show the plotted picture of regression lines and dots to make the program continually executable. If preferred, uncommenting those lines would produce pictures, and closing picture windows would continue program execution.

IV. Results and Discussion

1. Results



svar_set1 all dots



svar_set1 testing dots & regression line

Sample training error and testing error and Python made errors.

```
My training MSE: 3.25690251549. Python training MSE: 3.25690251549.
My testing MSE: 2.99715775056. Python testing MSE: 2.99715775056.
My training MSE: 3.25690251549. Reduced training MSE: 3.02361074317.
My testing MSE: 2.99715775056. Reduced testing MSE: 3.28050532443.
```

Sample testing errors for all models and final selection (linear for svar_set1).

```
[3.3180524285818063, 3.3340047518122606, 3.3513892403424128, 3.3354453348679569,
3.3457790447833062]
The 1th degree is the minimum.
```

Sample testing errors for higher dimensional models and final selection (2 degree higher dimensional model).

```
2nd degree MSE: 0.267315187453. 3rd degree MSE: 0.268679636325.
[0.2639950423837531, 0.26349546019645953]
The 2th degree is the minimum.
The train MSE is 0.261505029375. The test MSE is 0.263495460196.
```

Sample testing errors between explicit solution and gradient descent.

```
Explicit MSE: 0.269886832129. Iterative MSE: 0.270515061032.
```

Sample testing errors and time performance between dual linear regression and primal solution.

```
Dual time: 0:03:07.784000. 2nd and 3rd degree time: 0:00:00.291000.
Dual MSE: 0.334722291489. 2nd degree MSE: 0.267315187453.
```

2. Demonstration of Correctness

My regression implementation is correct since my MSE is as the same as Python made MSE. My gradient descent, Gaussian kernel and dual linear regression implementations are correct given that they share similar MSE with that of my regression. My selection is based on the minimum MSE, and it can be observed in the plots that my selection of linear model is correct.

3. Performance Evaluation

(1) Strength. My regression implementation is totally based on the function of

regression. It is retrieved directly from data, and has high accuracy and good time performance.

(2) Weakness. My gradient descent, Gaussian kernel and dual linear regression implementation, though they produce similar results, takes minutes to execute on large scale of data sets.

4. Effect of Various Parameters on Results

(1) Reduced amount of training data generates lower training error but higher testing error. It is, in my opinion, because less training data exhibits less deviation from regression model, but less training data provides a less reliable regression model for testing data. (2) My regression produces a bit smaller testing error than that of gradient descent. With feature scaling, testing error of gradient descent improves. (3) My dual linear regression takes much longer time than my primal regression. My suspect is that dual linear regression does more inversing matrices operations.

References

[1]http://stats.stackexchange.com/questions/15798/how-to-calculate-a-gaussian-kernel-effectively-in-numpy

[2]https://en.wikipedia.org/wiki/Radial_basis_function_kernel

[3]https://en.wikipedia.org/wiki/Gradient_descent

[4]http://code.activestate.com/recipes/521906-k-fold-cross-validation-partition/

[5]https://www.cs.cmu.edu/~tom/10701_sp11/slides/Kernels_SVM_04_7_2011-ann.pdf

[6]http://javeeh.net/sasintro/intro151.html