CSE574 Introduction to Machine Learning Programming Assignment 1

Classification and Regression

Weibin Ma

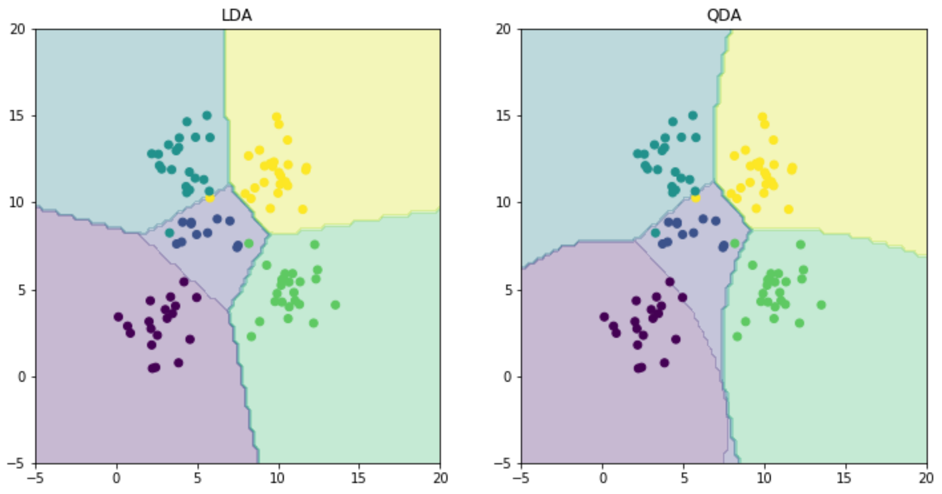
March 13th 2018

Problem 1: Experiment with Gaussian Discriminators

In this part, we aim to build the LDA and QDA model without any outside libraries and functions. Then to predict the test data based on the model learnt in training data and compute the accuracy of each classification model.

By coding the function to compute the accuracy of LDA and QDA, I got the accuracy of LDA is 0.97 while the accuracy of QDA is 0.96. This two classification method both have very high accuracy in classifying. Both LDA and QDA are derived from simple probabilistic models which model the class conditional distribution of the data P(X | y = k) for each class k. Predictions can be then obtained by Bayes' rule. As we learned in class, for LDA and QDA, P(X | y) can be modeled as a multivariate Gaussian distribution with density. So, we just need to estimate the class priors P(y = k), the class means μ (each class mean) and the covariance matrices.

For LDA, it is more ideal than QDA. Since each class shares the same covariance, so it is assumed to be a linear classifier without quadratic term. But for QDA, each class has different covariance matrix, so QDA is a quadratic classifier with quadratic term. Thus, they show different shape in the boundary. For LDA, the covariance is only linear, since the classes share the same covirance matrix, which means each class has the same shape for grouping, so we can consider the median line between the center of two classes as boundary which is linear. For QDA, since each class has different covariance matrix. So, after reflecting, their boundaries are quadratic so as to be more flexible, practical and experimental. See the plot below.



Problem 2: Experiment with Linear Regression

In this problem, we aim to implement ordinary least squares method to estimate regression parameters by minimizing the squared loss. The loss function can be written as:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

In order to find the minimized squared loss, we just need to set the function's first-order derivative equal to 0 and then compute the value of w. After derivation, w can be computed as:

w = (np.linalg.inv(X.T.dot(X))).dot(X.T).dot(y)

And then MSE can be computed by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

Now, we can compute the MSE without intercept and the MSE with intercept:

```
MSE without intercept : 106775.361533
MSE with intercept : 3707.8401812
```

From the final result, we can find that the MSE with intercept is smaller than the MSE without intercept. This is because the line without intercept learnt from training sample must pass through origin, which means is limited to rotate. On the other hand, when adding the intercept, the line is much more flexible and is much more practical and persuasive. And the accuracy is higher (about 29 times) than without adding intercept. Thus, adding the bias term is better.

Problem 3: Experiment with Ridge Regression

In this part, we aim to Implement parameter estimation for ridge regression by minimizing the regularized squared loss. The regularized loss function can be found as follow:

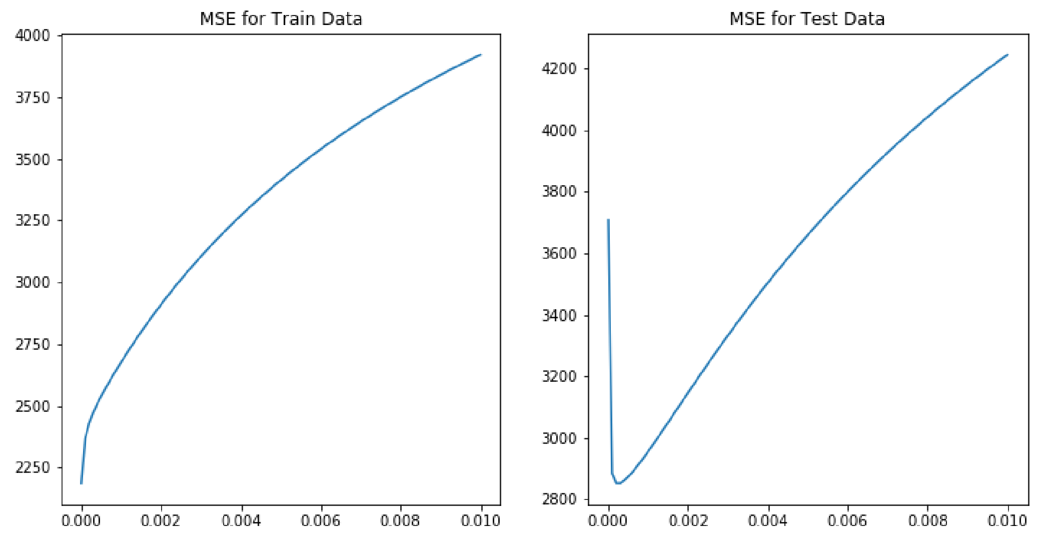
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \frac{1}{2}\lambda \mathbf{w}^\top \mathbf{w}$$

In order to find the minimized regularized squared loss, again, we just need to set the function's first-order derivative equal to 0 and then compute the value of w. After derivation, w can be computed as:

w = (np.linalg.inv(X.T.dot(X) + lambda_I)).dot(X.T).dot(y)

By running the code, I plot the MSE for train data and MSE for test data as below:

```
The optimal value for lambda is :  0.06 , where the MSE is :  2851.33021344
The total sum of weight learnt using OLE is :  57382.4955865
The total sum of weight learnt using Ridge Regression is :  1125.92527206
The variance of weight learnt using OLE is :  237806821.049
The variance of weight learnt using Ridge Regression is :  2546.2691635
Time Consuming : 0.04794120788574219
```



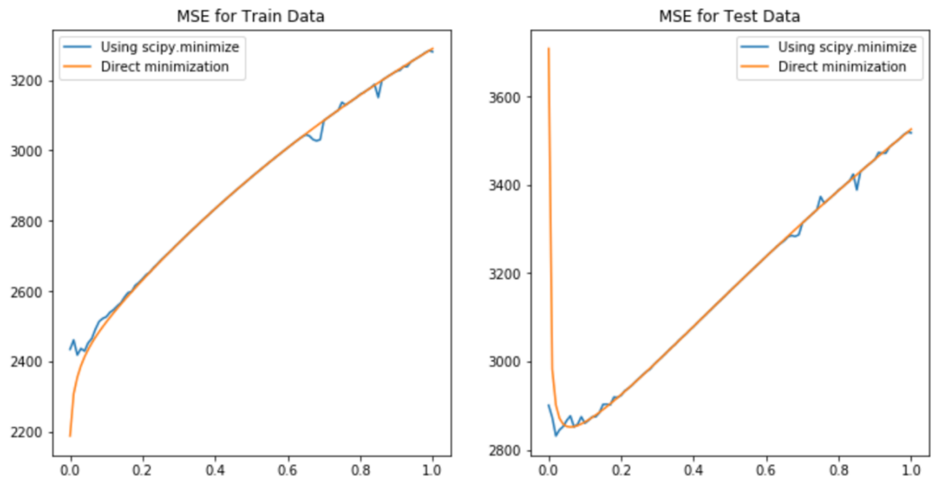
From the plot, the function 'learnRidgeRegression' was implemented by learning the train data, then the lambda was introduced into the function to learn controlling the weights vector so as to prevent from overfitting. As known, ridge regression is used for dealing with overfitting. From the plots, I find that the MSE for the Test Data shows a 'V' shape, since when labda = 0, it can't control the weights for overfitting, but as lambda increases, the lambda term in objective function become real values which will minimize the squared loss. So, as this point, the error in test data will begin to decrease. However, as lambda continues increasing, the test error raises again. This is because if the lambda continues increasing, it will restrict the growing of weight in learning process and then generates underfitting in turn in test data. So the optimal value for lambda is equal to the minimized value of lambda in plot. By computing, the optimal lambda is equal to 0.06 where the MSE value is equal to 2851.33021344.

By comparing the relative magnitudes of weights learnt using OLE (Problem 2) and weights learnt using ridge regression. I see that weights learnt using ridge regression is much more stable while the weights learnt using OLE is much more fluctuant and higher. Moreover, the variance for OLE is much higher than the variance for ridge regression, the variance of weight learnt using OLE is :237806821.049 and The variance of weight learnt using Ridge Regression is only :2546.2691635. Since the ridge regression uses regularization and the lambda part in ridge regression can help control the value of weights. Thus, the value of weights obtained is in a much lower magnitude and more stable after ridge regression.

Problem 4: Using Gradient Descent for Ridge Regression Learning

In this segment, we aim to use another efficient method to optimize the ridge regression: Gradient Descent for Ridge Regression. As discussed in class, regression parameters can be calculated directly using analytical expressions (as in Problem 2 and 3). To avoid computation of $(X^\top X)$-1, another option is to use gradient descent to minimize the loss function (or to maximize the log-likelihood) function. By implementing the code, I compare the gradient descent method and regular ridge regression method in terms of MSE and plot them as below:

```
The optimal value for lambda is :  0.02 , where the MSE is :  2831.18722376
Time Consuming : 0.4116849899291992
```



As given information, gradient descent is used to minimize the loss function. From the plots above, I find that the plot for minimizing the lose function with gradient descent is similar with the plot for minimizing the regularized squared loss with ridge regression. They both have similar weight values and variance. Both of the MSE show a 'V' shape as lambda increases as well.

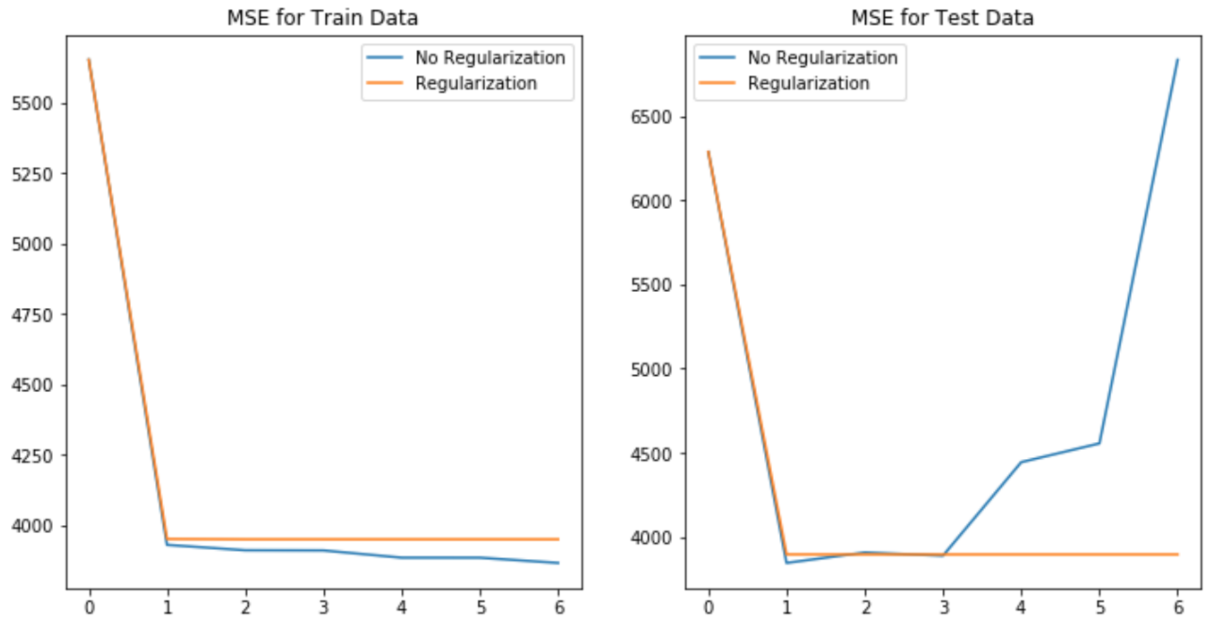
But, there is an obvious difference between two methods in mathematics. The regularized loss function contains an inverse term. So, when the matrix is very large or is a singular matrix (irreversible), the covariance matrix in the regular loss function will become unstable due to the inverse term in the formula. So, at this point, we can use gradient descent method which has similar effect for minimizing loss function to replace regular method.

Problem 5: Non-linear Regression

In this part, we aim to investigate the impact of using higher order polynomials for the input features. So, we need to implement the function ‘mapNonLinear’ which converts a single attribute x into a vector of p attributes, 1, x, x2, ..., xp.

By implementing the code, I got two graphs as below:

```
The optimal value for p in training data is :  6 , where the MSE is :  3866.88344945
The optimal value for p in test data is :  1 , where the MSE is :  3845.03473017
Time Consuming : 0.03581690788269043
```



By observing this two plots, I find that when I set lamda = 0 on training data, as p value increases, both of the MSE of ridge regression and gradient descent methods will decrease.

In the test data, we can find the error will decrease sharply when p value is in range (0,1). This is because when p = 0, it means using a horizontal line as the regression line, and then when p > 0, it is the same as the linear ridge regression. So, during this range in (0,1), the error will decrease quickly due to the function of linear ridge regression. Another observation in test data, when I set lambda = optimal lambda (0.06), the smallest of error happens in p = 1 for gradient descent method and then increases sharply after p = 1. This is because when p value is more than 1, the attribute x will be converted into higher polynomial terms which will increase the complexity of weight function so it's easier to fit the original data more, it then generates overfitting and the error raises quickly.

Problem 6: Interpreting Results

In the final part, we aim to make final recommendations for anyone using regression for predicting diabetes level using the input features based on the previous 4 methods. I compared them in two sides: MSE and Taking time.

```
MSE with intercept in Linear Regression :  3707.8401812
MSE in Ridge Regression :  2851.33021344
MSE in Gradient Descent for Ridge Regression :  2831.18722376
MSE in Non-Linear Regression :  3845.03473017

Time Consuming in Linear Regression : 0.0048558712005615234
Time Consuming in Ridge Regression : 0.04794120788574219
Time Consuming in Gradient Descent for Ridge Regression : 0.4116849899291992
Time Consuming in Non-Linear Regression : 0.03581690788269043

                             MSE  Taking_Time
LinearRegression     3707.840181     0.004856
RidgeRegression      2851.330213     0.047941
Gradient Descent     2831.187224     0.411685
NonLinearRegression  3845.034730     0.035817
```

First side is in MSE, from the data frame I plot above, we can easily find that the MSE value in Ridge Regression and Gradient Descent for Ridge Regression is much lower than it in Linear Regression and Non Linear Regression. This is because we put lambda into formula so as to control the weights. But, remembered, the ridge regression loss function contains an inverse term, when the data become large, this inverse term will become unstable due to the unstable covariance matrix. And as we learned, there might be a singular matrix (irreversible) term in the formula. In this situation, the ridge regression will become poor feasible, in turn the gradient descent method will become best choice, since it doesn't contain any inverse term in the formula.

Second side is in time consuming, I find the Linear Regression takes lowest time consuming while the gradient descent method takes highest. So, if we only want to consider time saving side, the Linear Regression will be the best choice.