CSE 574 Assignment 1 Report

Team 66 members information:

Xian Zhou

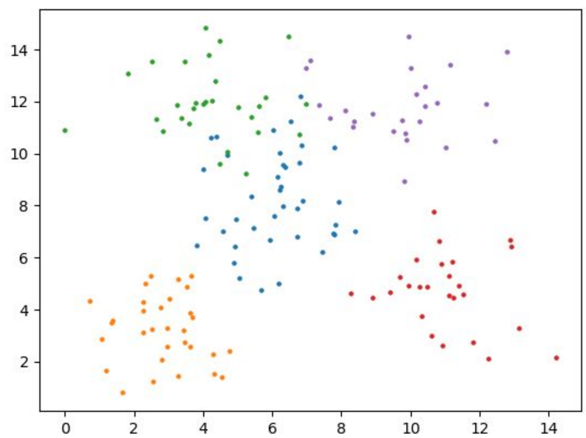
Yuxuan Zhang

Shuyan Chen

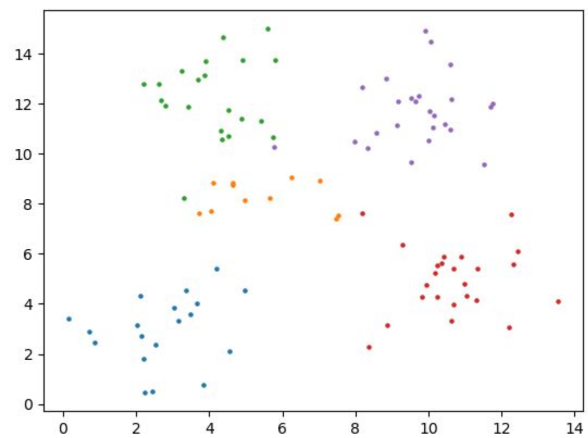
Prob 1: Experiment with Gaussian Discriminators

We implement the Linear Discriminant Analysis(LDA) and Quadratic Discriminant Analysis (QDA) for classification using functions ldaLearn, qdaLearn and ldaTest and qdaTest.

LDA and QDA assumes the data following Gaussian Distribution. Following are the images of distribution of training data (X) and test data (Xtest).



Training data



Test data

From the distribution we can see that it is meaningful to assume Gaussian distribution. We use ldaLearn and quaLearn to estimate means and covmats using training data. LDA assumes all the covmats the same, QDA assumes different covmats. For each class, QDA estimate MLE parameters for the multivariate normal distribution, LDA computes the MLE for covmat using all training data ignoring the class label.

Means and covmats are estimated by implementing the following formula:

$$\hat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_i} \triangleq \bar{\mathbf{x}},\ \hat{\Sigma}_{MLE} = \frac{1}{N}\left(\mathbf{x_i} - \bar{\mathbf{x}}\right)\left(\mathbf{x_i} - \bar{\mathbf{x}}\right)^{\mathbf{T}}$$

Aftering getting the parameters, we use the posterior for class classification.

$$p(y|x) \propto p(x|y) \cdot p(y) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} exp\left[-\frac{1}{2}\left(\mathbf{x}-\mu\right)^T \Sigma^{-1}\left(\mathbf{x}-\mu\right)\right] \cdot p(y)$$

Because there are only 150 training data, so we assume the same prior for every class, instead of using prior N_c / N. So the posterior:

$$p(y|x) \propto \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} exp\left[-\frac{1}{2}\left(\mathbf{x}-\mu\right)^T \Sigma^{-1}\left(\mathbf{x}-\mu\right)\right]$$

We choose the class label that has the biggest posterior.

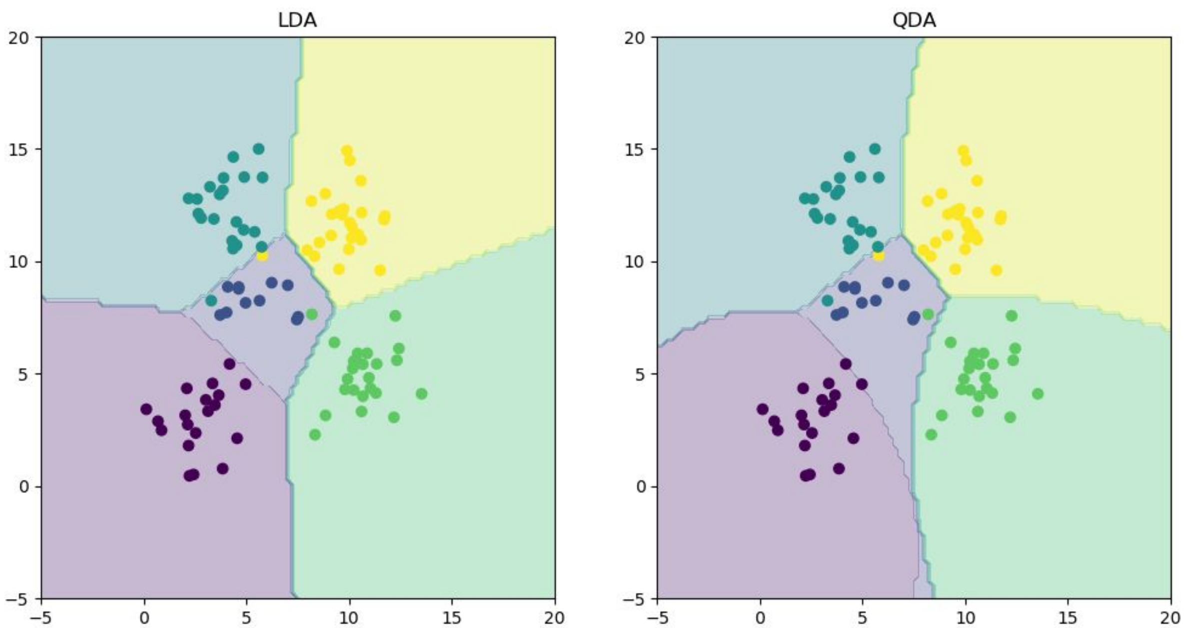
Results

We got the following accurate for LDA and QDA:

Accuracy for LDA: 97%

Accuracy for QDA: 96%

The images below are corresponding plots for the results of LDA and QDA:



Result analysis

As we can see from the above plots, LDA has linear surface. LDA assumes same covmat for all training data ignoring the class label. So LDA has no quadratic term: $\mathbf{x}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x}$. In contrast, QDA has non-linear boundaries. QDA computes the covariance differently for each class. QDA has the quadratic term, which will be different given different distribution.

There are total 100 test data, so we can see similar accuracy for LDA and QDA. But from the properties of LDA and QDA, given large and complex data set, LDA will be easy to calculate and obtained faster, but the it is a strong assumption that all the covmat be the same. QDA will gives better results.

Prob 2: Experiment with Linear Regression

We implement the linear regression using two main functions named learnOLERegression and testOLERegression.

Function learnOLERegression is used for estimating **w** using the training data. We use the ordinary least square method to estimate **w**.

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

By setting $\frac{\partial J(w)}{\partial w} = 0$, we can get

$$\hat{\mathbf{w}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

learnOLERegression is the implementation of this equation.

After we get , we can use the learnt weights for predication and evaluate the predication using MSE . Function testOLERegression is

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^{\mathrm{T}}x_i\right)^2 = \frac{1}{N}(\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Function testOLERegression is the implementation of the predication evaluation.

Results

1. Without using an intercept

That is $y = w^{T}x$, the line with **pass through the origin**.

In this case, **MSE for training data is 19099.446844570666**,

MSE for test data is 106775.36153159715.

2. With an intercept

That is $y = w^{T}x + w_0$, the line will **have an intercept $\mathbf{w_0}$** . We implement this by adding a column of 1s to X and Xtest.

In this case, **MSE for training data is 2187.160294930379,**

MSE for test data is 3707.8401813587807.

$w_0 \simeq 148.15487599654722$

Result Analysis:

From the results we can see that linear regression with intercept is better (i.e. lower MLE). It is obvious because not all the regression line pass through the origin.

Prob 3: Experiment with Ridge Regression

We implement the parameter estimation for ridge regression by using the regularized squared loss function as the following(in matrix-vector notation):

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

To minimize loss function, we set $\frac{\partial J(w)}{\partial w} = 0$, and get the estimated **w,**

$\mathbf{w} = (X^T X + \lambda I)^{-1} X^T y$. (I is an identity matrix with the size of (d+1)*(d+1), we use data with interpret).

Then use estimated **w** in function testOLERegression implemented in Problem 2 to get the MSE for training and test data.

Results

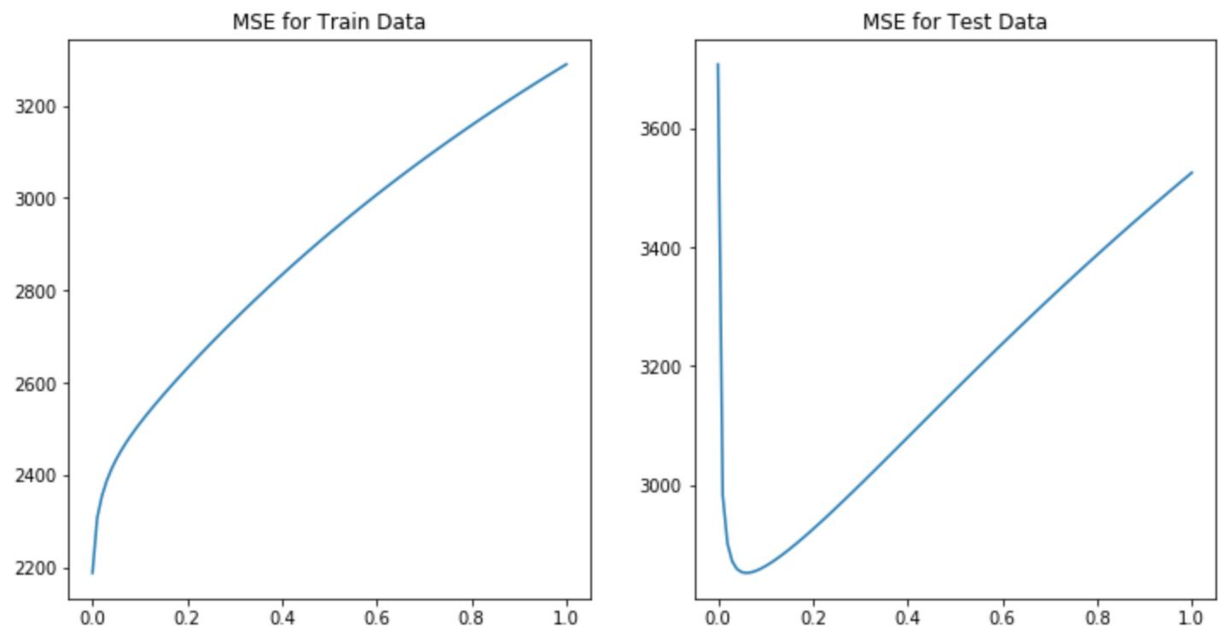
The chart of lambda and MLE for training and test data

Lambd	Train MSE	Test MSE		Lambd	Train MSE	Test MSE
0	**2187.16**	**3707.84**				
0.01	**2306.83**	**2982.45**		**0.51**	**2932.26**	**3166.92**
0.02	**2354.07**	**2900.97**		**0.52**	**2940.83**	**3174.81**
0.03	**2386.78**	**2870.94**		**0.53**	**2949.33**	**3182.69**
0.04	**2412.12**	**2858.00**		**0.54**	**2957.77**	**3190.55**
0.05	**2433.17**	**2852.67**		**0.55**	**2966.15**	**3198.39**
0.06	**2451.53**	**2851.33**		**0.56**	**2974.47**	**3206.21**
0.07	**2468.08**	**2852.35**		**0.57**	**2982.73**	**3214.01**
0.08	**2483.37**	**2854.88**		**0.58**	**2990.93**	**3221.79**
0.09	**2497.74**	**2858.44**		**0.59**	**2999.07**	**3229.55**
0.1	**2511.43**	**2862.76**		**0.6**	**3007.16**	**3237.29**
0.11	**2524.60**	**2867.64**		**0.61**	**3015.18**	**3245.00**
0.12	**2537.35**	**2872.96**		**0.62**	**3023.15**	**3252.70**
0.13	**2549.78**	**2878.65**		**0.63**	**3031.07**	**3260.36**
0.14	**2561.92**	**2884.63**		**0.64**	**3038.92**	**3268.01**
0.15	**2573.84**	**2890.86**		**0.65**	**3046.73**	**3275.63**
0.16	**2585.56**	**2897.31**		**0.66**	**3054.48**	**3283.23**

0.17	**2597.11**	**2903.94**		**0.67**	**3062.17**	**3290.80**
0.18	**2608.50**	**2910.74**		**0.68**	**3069.82**	**3298.34**
0.19	**2619.75**	**2917.68**		**0.69**	**3077.41**	**3305.86**
0.2	**2630.87**	**2924.75**		**0.7**	**3084.95**	**3313.35**
0.21	**2641.88**	**2931.94**		**0.71**	**3092.43**	**3320.82**
0.22	**2652.77**	**2939.23**		**0.72**	**3099.87**	**3328.26**
0.23	**2663.56**	**2946.60**		**0.73**	**3107.26**	**3335.68**
0.24	**2674.25**	**2954.07**		**0.74**	**3114.60**	**3343.06**
0.25	**2684.85**	**2961.60**		**0.75**	**3121.89**	**3350.42**
0.26	**2695.35**	**2969.20**		**0.76**	**3129.13**	**3357.76**
0.27	**2705.76**	**2976.86**		**0.77**	**3136.32**	**3365.06**
0.28	**2716.08**	**2984.56**		**0.78**	**3143.47**	**3372.34**
0.29	**2726.32**	**2992.32**		**0.79**	**3150.57**	**3379.59**
0.3	**2736.47**	**3000.12**		**0.8**	**3157.62**	**3386.81**
0.31	**2746.54**	**3007.95**		**0.81**	**3164.63**	**3394.01**
0.32	**2756.53**	**3015.81**		**0.82**	**3171.59**	**3401.17**
0.33	**2766.44**	**3023.70**		**0.83**	**3178.51**	**3408.31**
0.34	**2776.27**	**3031.61**		**0.84**	**3185.39**	**3415.42**
0.35	**2786.03**	**3039.55**		**0.85**	**3192.22**	**3422.51**
0.36	**2795.70**	**3047.49**		**0.86**	**3199.01**	**3429.56**
0.37	**2805.30**	**3055.45**		**0.87**	**3205.75**	**3436.59**
0.38	**2814.83**	**3063.42**		**0.88**	**3212.45**	**3443.59**
0.39	**2824.28**	**3071.40**		**0.89**	**3219.12**	**3450.56**
0.4	**2833.66**	**3079.39**		**0.9**	**3225.74**	**3457.50**
0.41	**2842.97**	**3087.37**		**0.91**	**3232.31**	**3464.42**
0.42	**2852.21**	**3095.35**		**0.92**	**3238.85**	**3471.30**
0.43	**2861.37**	**3103.34**		**0.93**	**3245.35**	**3478.16**
0.44	**2870.47**	**3111.32**		**0.94**	**3251.81**	**3484.99**
0.45	**2879.50**	**3119.29**		**0.95**	**3258.23**	**3491.80**

0.46	**2888.46**	**3127.25**		**0.96**	**3264.61**	**3498.57**
0.47	**2897.35**	**3135.21**		**0.97**	**3270.96**	**3505.32**
0.48	**2906.18**	**3143.16**		**0.98**	**3277.26**	**3512.04**
0.49	**2914.94**	**3151.09**		**0.99**	**3283.53**	**3518.73**
0.5	**2923.63**	**3159.01**		**1**	**3289.76**	**3525.39**

The image build in program of MSE for training and test data.



Result Analysis

We can conclude from the image above:
For training data, the MSE increases all the time. For test data, the MSE first decreases rapidly and then increases.
The optimal value for λ is **0.06**. Because we can observe from the chart of lambda and MSE for training and test data, **the MSE for test date is the smallest when λ=0.06**, which means when λ=0.06, test error is the smallest.
Ridge Regression use l_2 regularization. So we use l_2 norm to compare the magnitudes of weights learnt using OLE and ridge regression. For OLE, $||\mathbf{w}||_2^2$ **=15508101065.53**; for Ridge regression, $||\mathbf{w}||_2^2$ **= 920281.36** when λ=0.06. We can see that l_2 norm of OLE is much larger than that of Ridge regression. Complexity for OLE is larger.
For Ridge Regression: when λ=0.06, the training error is 2451.53, the testing error is 2851.33.
For Linear Regression (with bias): the training error is 2187.16, the testing error is 3707.84. We can see that **Ridge Regression has a larger training error and a smaller testing error in**

comparison with Linear Regression. This result is obvious because Ridge regression add an regularization term which is for complex penalty. As we all know, as the model complexity increase, the training error decrease and testing error decrease at first and then increase. **Ridge Regression solves the overfitting problem in Linear regression with the help of regularization term.**

Prob 4: Using Gradient Descent for Ridge Regression Learning

We implement the gradient descent using the function regressionObjVal.
Function regressionObjVal is used for estimating regularized squared error and its gradient by using the training data. By setting an initial **w,** we use the function minimizer to get optimal **w** with the minimum gradient error to create objective function.
The function below is used to get error.

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

By deriving J(w) with respect to w, we get the outcome below to get the gradient error.

$$\frac{\partial J(w)}{\partial w} = -X^T(\mathbf{y} - X\mathbf{w}) + \lambda\mathbf{w}$$

According the training rule for gradient descent $\mathbf{w} = \mathbf{w} - \eta\frac{\partial J(w)}{\partial w}$, we can finally get the optimal **w** when converge. Learning step η is the hyper-parameter, which can not be trained by training data. This area is implemented in the base code, through scipy.optimize.minimize in my opinion.

Results

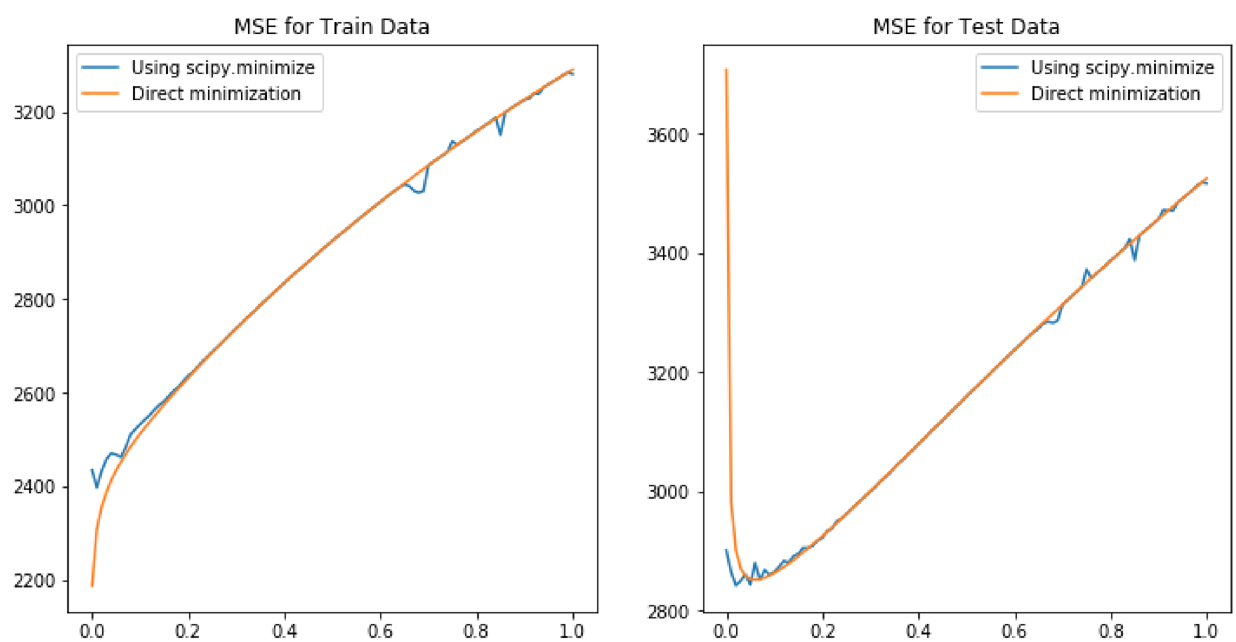
The following dataset represents the squared error for train and test data by using the gradient descent approach.

Lambda	MSE(train data)	MSE(test data)		Lambda	MSE(train data)	MSE(test data)
0	2433.665412	2900.545956		0.51	2932.263141	3166.923145
0.01	2396.708691	2864.511038		0.52	2940.881131	3174.797091
0.02	2431.419235	2841.57743		0.53	2949.356813	3182.679541
0.03	2457.27641	2849.18882		0.54	2957.775008	3190.549454
0.04	2470.338618	2860.49379		0.55	2966.169534	3198.367369
0.05	2467.309761	2843.083484		0.56	2974.475384	3206.204178
0.06	2462.046255	2879.738438		0.57	2982.745343	3214.029113
0.07	2482.84389	2851.198389		0.58	2990.930381	3221.784264
0.08	2509.379681	2868.064153		0.59	2999.130207	3229.54213

0.09	**2520.816255**	**2860.188624**		**0.6**	**3007.141354**	**3237.296728**
0.1	**2531.709701**	**2864.540583**		**0.61**	**3015.337706**	**3245.311011**
0.11	**2542.316314**	**2873.259762**		**0.62**	**3023.180117**	**3252.659352**
0.12	**2551.438345**	**2883.005794**		**0.63**	**3031.0425915]**	**3260.281568**
0.13	**2563.466391**	**2880.213124**		**0.64**	**3038.332616**	**3267.03008**
0.14	**2573.580881**	**2891.403223**		**0.65**	**3044.324949**	**3273.084071**
0.15	**2580.569941**	**2894.873276**		**0.66**	**3041.330581**	**3281.621963**
0.16	**2592.3115**	**2904.760641**		**0.67**	**3031.027044**	**3284.811341**
0.17	**2603.393**	**2904.415297**		**0.68**	**3027.107411**	**3282.717607**
0.18	**2612.18803**	**2908.247878**		**0.69**	**3030.658679**	**3286.560481**
0.19	**2624.518249**	**2916.969556**		**0.7**	**3084.889021**	**3313.288611**
0.2	**2635.888048**	**2921.799333**		**0.71**	**3092.43248**	**3320.816975**
0.21	**2643.5150796]**	**2933.760188**		**0.72**	**3099.882664**	**3328.275097**
0.22	**2654.889192**	**2937.092064**		**0.73**	**3107.28567**	**3335.672723**
0.23	**2666.417752**	**2949.498679**		**0.74**	**3114.6026473]**	**3343.068391**
0.24	**2675.891643**	**2954.033991**		**0.75**	**3136.868704**	**3372.830077**
0.25	**2686.313533**	**2961.541674**		**0.76**	**3129.054478**	**3357.776073**
0.26	**2696.408295**	**2969.638208**		**0.77**	**3136.302029**	**3365.089966**
0.27	**2706.395116**	**2977.262361**		**0.78**	**3143.449947**	**3372.290155**
0.28	**2716.198136**	**2984.878511**		**0.79**	**3150.541643**	**3379.556118**
0.29	**2726.839055**	**2992.368105**		**0.8**	**3159.42958**	**3387.873214**
0.3	**2736.958131**	**2999.617199**		**0.81**	**3164.590854**	**3393.946321**
0.31	**2746.194972**	**3006.444648**		**0.82**	**3171.993796**	**3401.689756**
0.32	**2757.202194**	**3015.91734**		**0.83**	**3178.698044**	**3408.455729**
0.33	**2766.110589**	**3022.989506**		**0.84**	**3188.13457**	**3423.456322**
0.34	**2775.782979**	**3030.099235**		**0.85**	**3150.503695**	**3387.956776**
0.35	**2786.527172**	**3039.493709**		**0.86**	**3199.049971**	**3429.560819**
0.36	**2795.921125**	**3047.566328**		**0.87**	**3205.97434**	**3436.64809**
0.37	**2805.009751**	**3054.380629**		**0.88**	**3212.933698**	**3443.918686**
0.38	**2815.109345**	**3063.521325**		**0.89**	**3218.877293**	**3450.056164**

0.39	2824.415889	3070.608054		0.9	3225.378033	3457.175784
0.4	2833.881617	3079.431202		0.91	3228.207441	3472.573028
0.41	2843.16505	3086.719599		0.92	3238.89473	3471.3411
0.42	2852.341146	3095.337904		0.93	3238.348832	3471.208421
0.43	2861.374579	3103.115639		0.94	3251.905382	3485.119242
0.44	2870.020145	3110.834533		0.95	3258.267736	3491.856724
0.45	2878.973673	3118.506869		0.96	3264.615109	3498.088107
0.46	2888.527637	3127.312246		0.97	3270.957197	3505.308367
0.47	2897.325607	3135.227377		0.98	3278.399721	3513.51091
0.48	2906.29128	3143.239217		0.99	3283.699879	3518.783514
0.49	2914.468016	3150.513886		1	3280.783437	3517.191683
0.5	2923.685175	3159.007267				

From the table above, **we found that when lambda is 0.02 the MSE for testing data is minimum**, so we use this for our function.



Result Analysis

The MSE tendency for training data and test data for different λ is similar between direct minimization and using scipy.minimize (gradient descent). As the λ increase from 0 to 1, for training data, the MSE increases all the time. For test data, the MSE first decreases rapidly and then increases.

In the above figure we found that the MSE for test data is greater than train data. Test data is more error prone.

The best λ we get is 0.02, with train MSE 2431.42 and test MSE 2841.58. The result we get from Pro3 is: λ=0.06, the training error is 2451.53, the testing error is 2851.33. The two result are similar.

Prob 5: Non-linear Regression

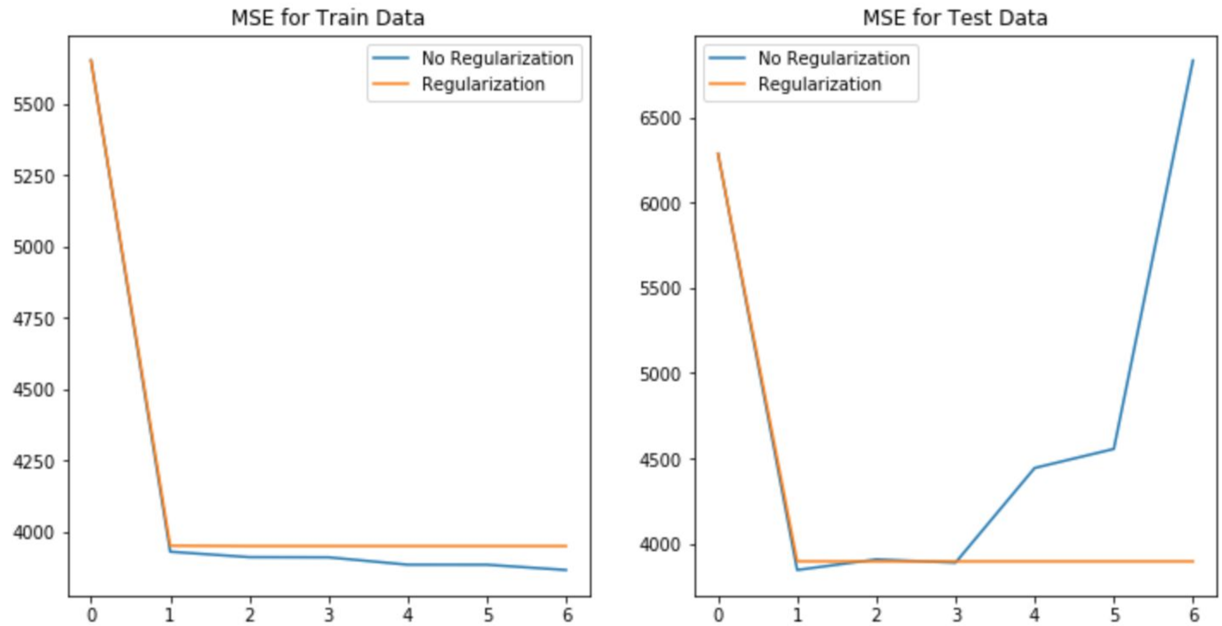
We converts a single attribute x into a vector of p attributes 1, x, x^2, ..., x^p to investigate the impact of using higher order polynomials for the input features. Using the λ = 0 and the optimal value of λ found in Problem 3, vary p from 0 to 6 to train ridge regression weights using the non-linear mapping of the data. And compare their errors.

Results

The result of MSE for training and test date with different λ and p

p	MSE for training data(λ=0)	MSE for training data(λ=0.06)		MSE for test data(λ=0)	MSE for test data(λ=0.06)
0	5650.710539	5650.711907		6286.40	6286.881967
1	3930.915407	3951.839124		3845.03	3895.856464
2	3911.839671	3950.687312		3907.13	3895.584056
3	3911.188665	3950.682532		3887.98	3895.582716
4	3885.473068	3950.682337		4443.33	3895.582668
5	3885.407157	3950.682335		4554.83	3895.582669
6	3866.883449	3950.682335		6833.46	3895.582669

The image build in program of MSE for training and test data with different λ.



Result Analysis

We can conclude from the image above:

For training data, the errors of both $\lambda=0$(no regularization) and $\lambda=0.06$(regularization) first are very high but decrease rapidly, then decrease very less amount.

For test data, the error of $\lambda=0$(no regularization) first are very high, then decrease rapidly, but after a few test, it increase to a high number and goes on. The error of $\lambda=0.06$(regularization) first are very high but decrease rapidly, then decrease very less amount.

The optimal value of p in terms of test error in $\lambda=0$ is 1.

The optimal value of p in terms of test error in $\lambda=0.06$ is 4.

Prob 6: Interpreting Results

Results and result analysis

Comparison of MSE for training data and testing data for Prob2-5:

Approaches	Train MSE	Test MSE
OLE without intercept	19099.44	106775.36
OLE with intercept	2187.16	3707.84
Ridge Regression	2451.53	2851.33
Ridge using Gradient descent	2431.42	2841.58
Non-linear Regression without regularization	3930.92	3845.03
Non-linear Regression with regularization	3950.68	3895.58

For non-linear regression, we choose the pair with minimum testing MSE.

From above table, we can see that **Ridge regression approaches (both direct minimization and using gradient descent) have the best performance on prediction (with the lowest test MSE).** We will analyze the result afterwards.

For Linear Regression with and without bias, the OLE with bias has better performance(i.e. lower MLE). It is obvious because not all the regression line pass through the origin.

Linear Regression vs Ridge Regression:

(We use the result from Prob3 for analysis here, gradient descent is only another approaches for implement the ridge regression. The results for Prob3 and Prob4 are similar.)
For Ridge Regression: the training error is 2451.53, the testing error is 2851.33. For Linear Regression (with bias): the training error is 2187.16, the testing error is 3707.84. We can see that **Ridge Regression has a larger training error and a smaller testing error in comparison with Linear Regression**. This result is obvious because Ridge regression add an regularization term which is for complex penalty. As we all know, as the model complexity increase, the training error decrease and testing error decrease at first and then increase. **Ridge Regression solves the overfitting problem in Linear regression with the help of regularization term.** Ridge Regression use l_2 regularization. So we use l_2 norm to compare the magnitudes of weights learnt using OLE and ridge regression. For OLE, $\|\mathbf{w}\|_2^2$ **=15508101065.53**; for Ridge regression, $\|\mathbf{w}\|_2^2$ **= 920281.36** when λ=0.06. We can see that l_2 norm of OLE is much larger than that of Ridge regression. Complexity for OLE is larger.

Two approaches of Ridge Regression:
We can see that the performance of the two approaches of ridge regression are similar. But I suggest to use gradient descent because it is computationally expensive and unstable to calculate a matrix inversion. For gradient descent, it is more widely used, but it is highly depend on the step size, a hyper-parameter. But this problem can be solved by using Hessian matrix (second derivation).

Non-linear regression can solve the problem of underfitting for linear regression, but we need to compare model with different p and find the best one.

In summary, Ridge Regression performs best. Gradient Descent for Ridge regression is more suggested because it doesn't need to calculate the matrix inversion.