CSL - 603
LAB - 2 REPORT
Linear and Logistic Regression

Eeshaan Sharma
2015CSB1011
Shivam Mittal
2015CSB1032

September 2017

1 Question 1

The goal of this exercise is to implement linear ridge regression and to predict the age of Abalone (is a type of snail). We perform various experiments and analyze the performance (by measuring the mean square loss). We will present all the experiments and the observations we got.

1.1 Experiment 1

The first column that is present in the dataset encodes whether the snail is male, female or an infant. Since this attribute has no order to itself. We represent this attribute as 3 attributes (x,y,z). We encode female as (1,0,0), infant as (0,1,0) and male as (0,0,1).
We also add a extra column in the attributes containing all 1, so that the linear fit may have the independence to choose an intercept.

1.2 Experiment 2

We standardize the all the independent variables except the first 4 columns. The first column is for the intercept term, and the other 3 columns encode the male, female and infant information.
We scale the features because it improves the convergence of gradient descent because all the features are on the same scale and we have uniform contour plots.

1.3 Experiment 3

Implemented the function mylinridgereg(X, Y, lambda). It calculates the linear least squares solution with lambda as the penalty parameter of ridge regression, and returns the weights which gives the best linear fit for the regression data given the ridge regression loss function. We learned the weights using the analytical solution which is

$$(X^T X + lambda * I)^{-1}(X^T Y)$$

But, if the matrix $(X^T X + lambda * I)$ is not invertible, we use gradient descent to learn the weights.
We used gradient descent using the loss function of ridge regression. We used the stopping criteria $error(prev) - error(curr) < 0.00001$ to stop.
The weight updates were as follows:

$$gradient = X^T(XW - Y)/N + lamda * W$$

$$W(new) = W(old) - (learning - rate * gradient)$$

We also implemented the function mylinridgeregeval(X, W) where the predicted target value was calculated as XW.

1.4 Experiment 4

Data was partitioned by randomly shuffling the array, and then choosing the first fraction of elements from the randomly shuffled array as training examples, and the remaining as test examples.
We partition the data into 0.8 training data and 0.2 test data, and learn the model for different values of lambda. In the upcoming section, we will plot graphs and share the observations we get for the different values of lambda.

1.5 Experiment 5

The function meansquarederr(T, Tdash) was implemented. For each data point, the square of the difference of the observed value and the predicted value was taken. The average of all these squared value was taken to get the mean square error.

1.6 Experiment 6

We observed the weights learned by the model, we take the absolute value of all the weights, and then remove the 2 weights which are the minimum (minimum absolute value). We also remove the attributes corresponding to those weights.
We observe that:
Before removing any attributes:
Training set mean square error = 4.96545740811
Test set mean square error = 4.88545308479

After removing 2 least significant attributes
Training set mean square error = 4.96542487813
Test set mean square error = 4.88257726356
No significant change in the squared error can be observed, both in the test set and the training set. This explains the fact that the attributes which we removed actually do not effect our target variable, even after removing those attributes, we get approximately the same result.

We see that the weight of the first 3 attributes, that is originally encoded from one atrribute i.e. (male, female, or infant) has the highest weight. Also, thes attribute number 6 (shucked weight) and 8 (shell weight) have the next most significant weights.
Hence, we can conclude that the sex, shucked weight and shell weight of an abalone are the most significant attributes in determining its age.

1.7 Experiment 7

We take different training set fractions and lambda values. For each fraction and lambda pair, we make 100 iterations and calculate the average error over the 100 iterations. We will make graphs over this data observed in experiment 8 and 9.

1.7.1 a. Does the effect of lambda on error change for different partitions of the data into training and testing sets?

Yes, the effect of lambda on error changes for different partitions of the data into training and test set. We will more clearly observe and try to generalize the trends observed during experiment 8 and experiment 9, that would more clearly answer this question.

1.7.2 b. How do we know if we have learned a good model?

There is only one way of knowing that we have learned a good model, and that is through checking the accuracy on a test dataset. We feed examples to the model which it has never seen till now. If the model is able to classify them well (i.e. the error is low), then we know that we have learned a good model. So, basically a low mean square error on test dataset is an indication of a good model.

1.8 Experiment 8

To see whether the training set fraction affects the effect of lambda on error, we plot graph of average mean square error vs lambda values, for different fractions of training set. In the above graphs, the red line shows the average MSE on the test set, and the blue line shows the average MSE on the training set.

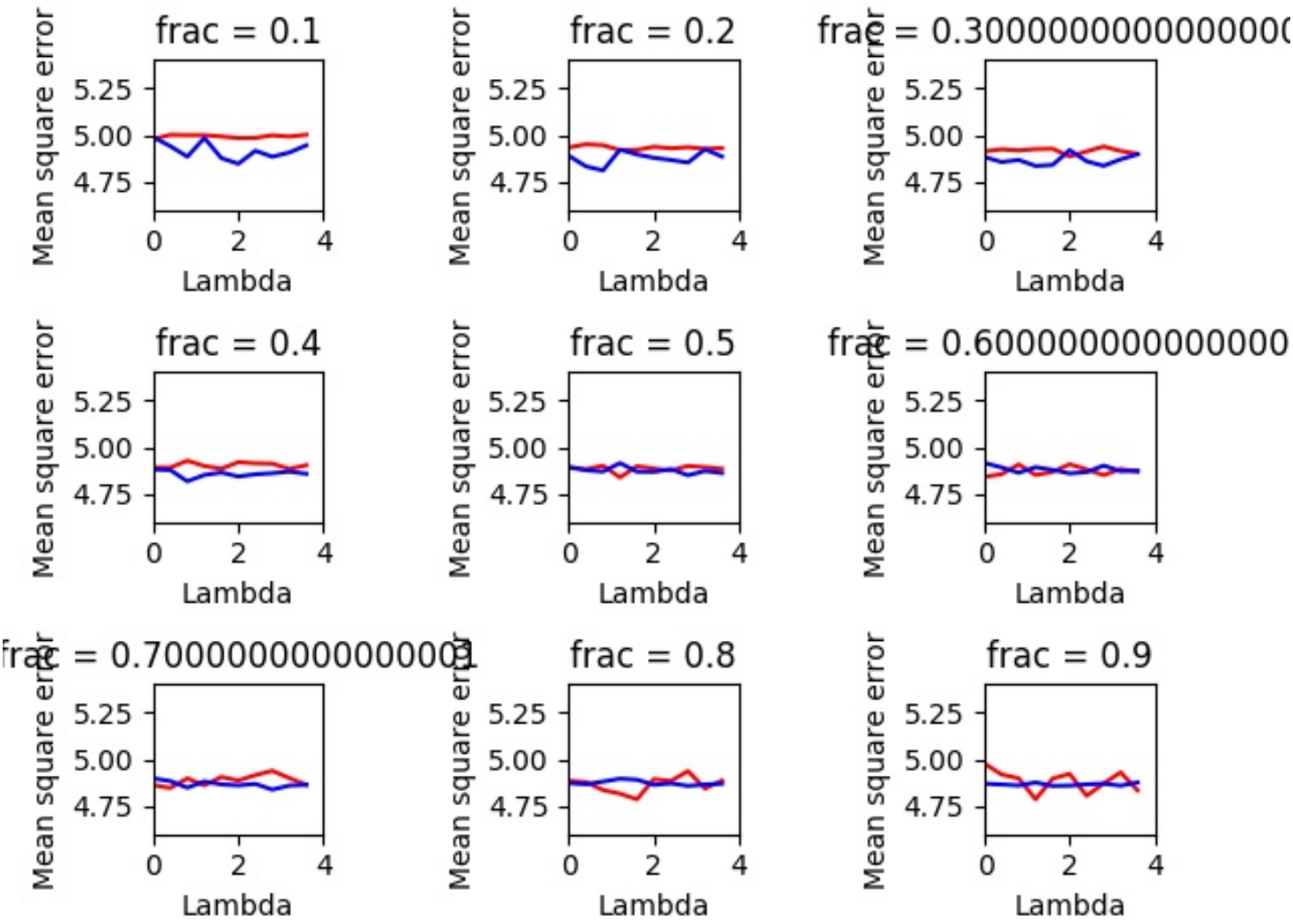


Figure 1: Experiment 8 : Average MSE vs lambda for different values of training set fraction

We can see that for lower values of training fraction, the blue plot (training error) is not constant and is oscillating, while the red plot (test error) is constant. While when, the fraction increases, then the blue plot (training error) becomes constant while the red error keeps oscillating.
When we have less training data, then the change in lambda changes the training error significantly. The change is significant because we have less training points, whereas when we have large number of training examples, the change in lambda can't influence the training error because the error is being averaged over large number of points, even if some points are influenced, when we average over a large number of examples, the result remains pretty much constant. The same explanation applies to the test error also.

1.9 Experiment 9

For more clear insight, we plot 2 more graphs.

1.9.1 Minimum average MSE vs training fraction

Figure 2: Experiment 9.1 : Minimum average MSE vs training fraction

We observe from the graph, that as the training fraction increases, the minimum mean square error decreases. This can be explained, as we increases the number of our training examples (as we are getting more and more data to

train), the effect of lambda on the error changes, the influence of lambda to fit a better model given the data increases as the data increases, and hence the error decreases.

1.9.2 Lambda that produced minimum average MSE vs training fraction

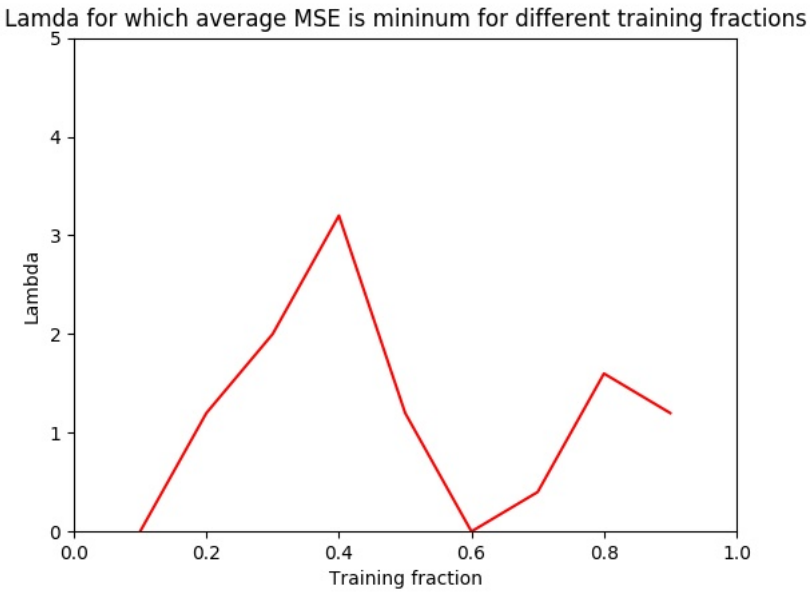


Figure 3: Experiment 9.2 : Lambda that produced minimum average MSE vs training fraction

We can see from the graph that there is no direct trend as to which lambda gives the best fit (minimum average mean square error), hence the determination of lambda is empirical, no general trend can be observed.

1.10 Experiment 10

To see the contribution of individual examples, we plot its predicted target value against the actual target value. We have made each of the point on the graph to have very low opacity, so the sparse regions will have low opacity, and the dense regions (containing more points) will have more opacity. We can observe that most of the points are close to the 45 degree line. The high density regions are close to this line, suggesting that the model learned is good.

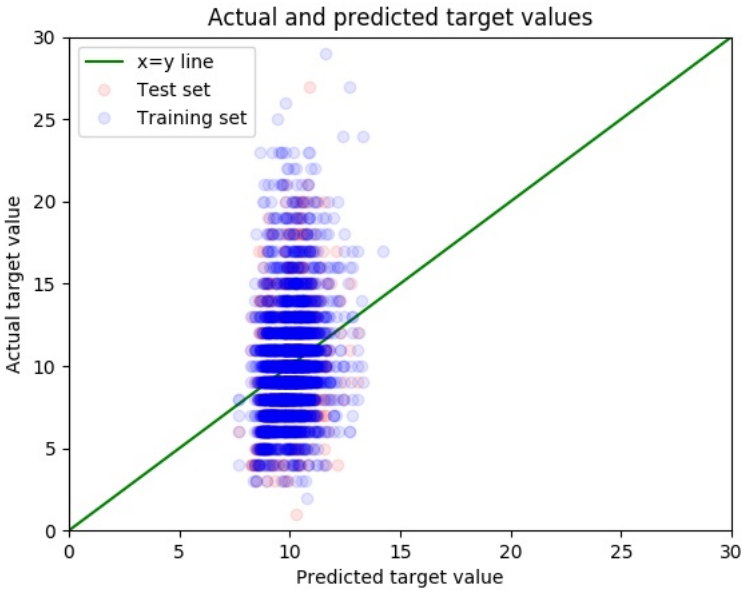


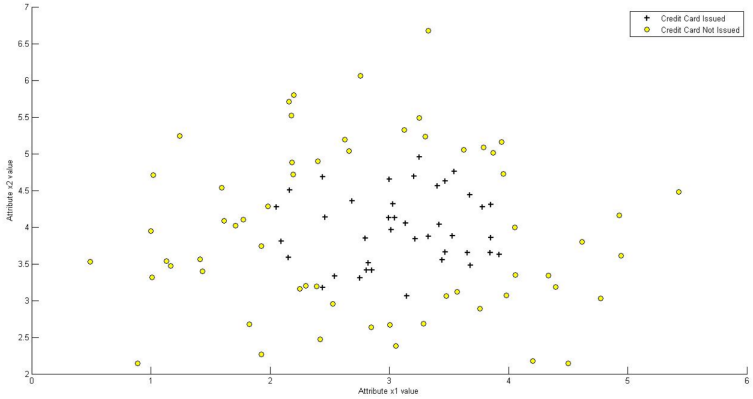
Figure 4: Experiment 10 : Actual value vs predicted target value

2 Question 2 - INTRODUCTION

The goal of this exercise is to implement regularized Logistic Regression using a number of techniques in order to predict whether a credit card can be issued to an individual or not on the basis of values of 2 attributes (x1 and x2). Regularized Logistic Regression is implemented using Gradient Descent and Newton Raphson as optimization techniques. To get a better model for understanding the data, Feature Transformation is implemented to incorporate higher order polynomials of the input attributes.

3 Plotting Data

The plot of training dataset with + indicating (y = 1) examples and o indicating (y = 0) examples is as follows -



4 Regularized Logistic Regression

In this part of the assignment, a logistic regression model is used to predict whether the credit card application of an individual is accepted or not. Individuals are characterized on the basis of values of 2 attributes (x1 and x2).

4.1 Using Gradient Descent

4.1.1 Implementation

The hypothesis for logistic regression is defined as -

$$h_W(x) = g(X * W) \tag{1}$$

where function g is the sigmoid function. The sigmoid function is defined as -

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

The cost function for regularized logistic regression consisting of 'm' training examples and 'n' features with regularization parameter lambda is defined as -

$$J(W) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_W(x^{(i)}) - (1-y^{(i)})log(1-h_W(x^{(i)})] + \frac{\lambda}{2m}\sum_{j=1}^{n} W_j^2 \quad (3)$$

Using the above cost function, the gradient obtained for each W is as follows -

$$\frac{dJ(W)}{dW} = (\frac{1}{m}\sum_{i=1}^{m}(h_W(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}W_j \quad (4)$$

Using the gradient obtained above the parameter update equation for regularized logistic regression using gradient descent optimization method with learning rate alpha is as follows -

$$W_j = W_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_W(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}W_j] \quad (5)$$

Using the parameter update equations learned above gradient descent is run for a fixed a number of iterations on an initial value of W, to obtain a classification model for the given dataset.

4.1.2 Results

For training purposes the model uses the training examples provided in the credit.txt file.
For test purposes 1000 new test examples are generated in the file credit2.txt using l2logreg.m script file provided via e-mail.

Gradient Descent is applied using the following parameters -

1. Learning Rate - $\alpha = 0.1$
2. Regularization Parameter - $\lambda = 10$
3. Number of Iterations = 5000

The following values for vector W are obtained -

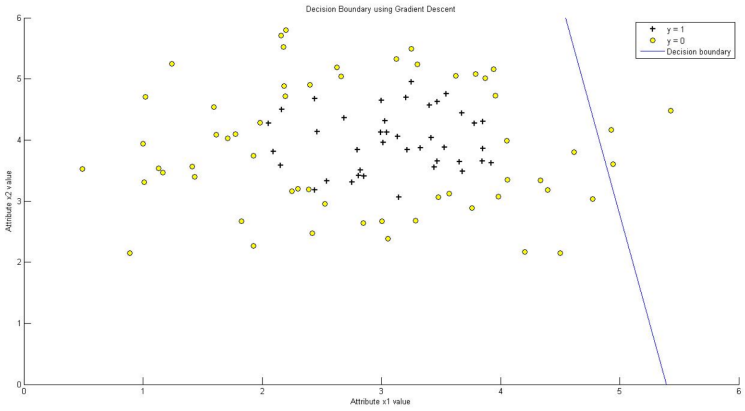
W_0 = -1.154362
W_1 = 0.214013
W_2 = 0.030233
This gives a decision boundary as follows -

4.2 Using Newton Raphson

4.2.1 Implementation

In Newton Raphson Optimization method we define a matrix H as follows -

$$H = X^T * R * X \quad (6)$$



where R is an NXN diagonal matrix with diagonal element - $(f(x_i)(1-f(x_i)))_{ii}$
The parameter update equation for Newton Raphson Optimization method is as follows -

$$W_j = W_j - H^{-1}\frac{dJ(W)}{dW} \tag{7}$$

where $\frac{dJ(W)}{dW}$ is the same as found in eqn(4) given under Gradient Descent. Using the parameter update equations learned above Newton Raphson Optimization is run for a fixed a number of iterations on an initial value of W, to obtain a classification model for the given dataset.

4.2.2 Results

For training and test purposes the same 2 files have been used as that of Gradient Descent.

Gradient Descent is applied using the following parameters -

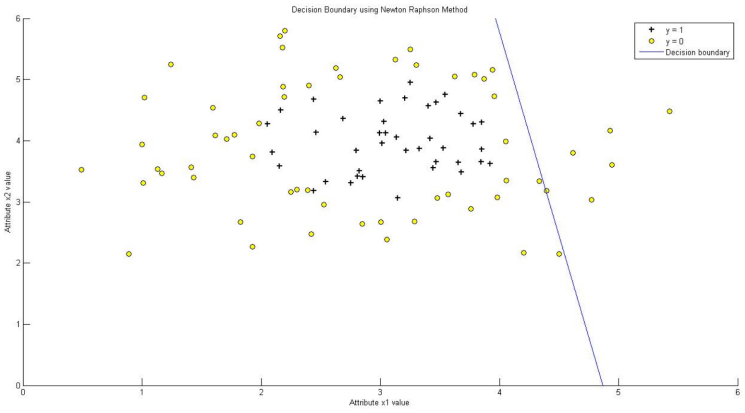
1. Regularization Parameter - $\lambda = 1$
3. Number of Iterations = 5000

The following values for vector W are obtained -

W_0 = -1.490216
W_1 = 0.306001
W_2 = 0.046066
This gives a decision boundary as follows -

4.3 Comparing Performance

To compare the performance of Gradient Descent and Newton Raphson optimization techniques for a fixed number of iterations, both of these techniques



are applied and their accuracy on the common Test Dataset is obtained. The result is as follows.

1. Accuracy of Gradient Descent on Training Data = 57%
2. Accuracy of Gradient Descent on Test Data = 58%

3. Accuracy of Newton Raphson on Training Data = 54%
4. Accuracy of Newton Raphson on Test Data = 51.4%

From the above results it can be seen that Gradient Descent performs slightly better than Newton Raphson Optimization Technique in learning a classification model for the given dataset.

5 Feature Transformation

From the above plots of data, it is evident that the dataset cannot be separated into positive and negative examples by a straight line through the plot. Therefore a straight forward application of logistic regression will not perform well on this dataset, since logistic regression will only be able to find a linear decision boundary.

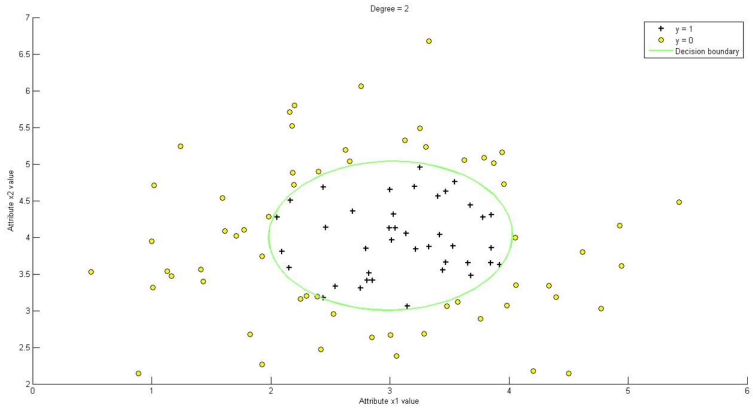
One method to overcome this difficulty is to create more features from each data point by mapping features to all polynomial terms of x1, x2 upto some degree.

As a result of this mapping, the original vector of 2 features is transformed into a higher dimensional vector. A logistic regression classifier trained on this higher dimensional feature vector will have a more complex decision boundary and will appear non-linear when drawn in a 2-D plot.

The features are transformed to create higher order polynomials of the input attributes and then the performance is measured by implementing Regularized Logistic Regression using Newton Raphson method on features transformed to different degrees.

5.1 Degree = 2

When the features are transformed to degree 2 the following figure shows plot of decision boundary.



Accuracy obtained on Training Data = 100%
Accuracy obtained on Test Data = 98.9%

5.2 Degree = 3

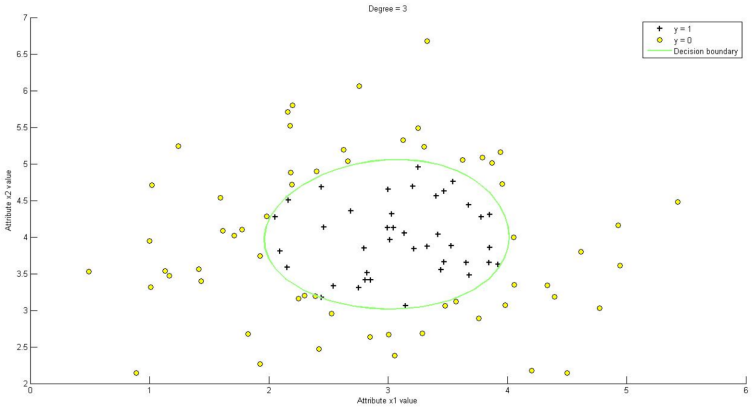
When the features are transformed to degree 3 the following figure shows plot of decision boundary.

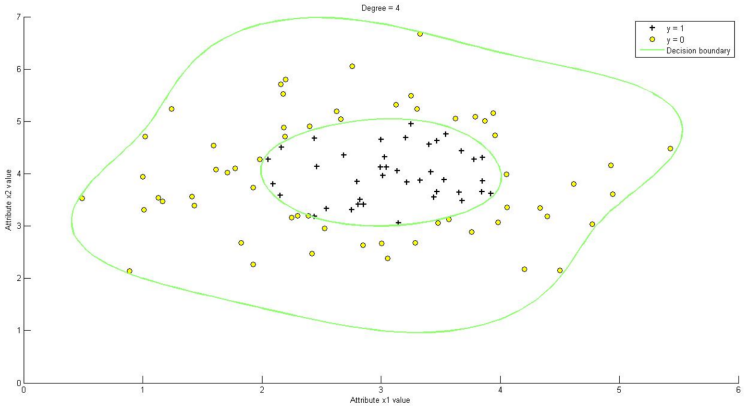
Accuracy obtained on Training Data = 100%
Accuracy obtained on Test Data = 98.5%

5.3 Degree = 4

When the features are transformed to degree 4 the following figure shows plot of decision boundary.

Accuracy obtained on Training Data = 100%
Accuracy obtained on Test Data = 95.9%


Degree = 3
y = 1
y = 0
Decision boundary
Attribute x2 value
Attribute x1 value


Degree = 4
y = 1
y = 0
Decision boundary
Attribute x2 value
Attribute x1 value

5.4 Choosing Optimum Degree

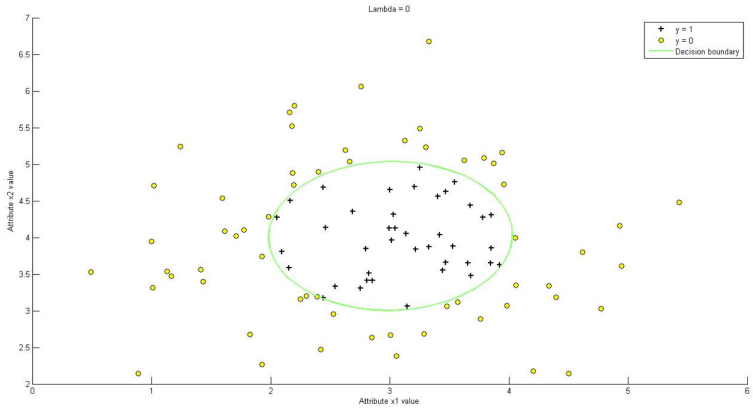
It is evident from the accuracy's obtained above that the feature transformation to a higher degree degrades the performance of Logistic Regression Classifier on the test dataset. The test dataset along with all other parameters is same for all the cases. This shows that although feature mapping allows us to build more expressive classifiers, but at the same time it is more susceptible to overfitting. Thus the appropriate degree of transformation obtained is 2.

6 Varying Regularization Parameter

As seen above, feature transformation with lower values of λ can result in the model to overfit the training data and thus perform poorly on previously unseen test data. To overcome this difficulty regularization is used to prevent overfitting.

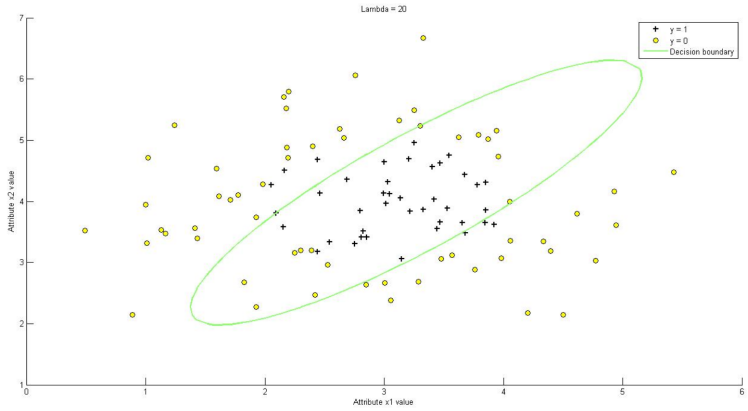
6.1 Overfitting Example

With a small value of λ it is observed that the classifier gets almost every training example correct, but produces a complicated boundary which is not a good decision boundary. This can be seen in the following example where $\lambda = 0$



6.2 Underfitting Example

With a larger value of λ a simpler decision boundary is obtained which still separates positive and negative examples fairly well. However, if λ is set too high, a good fit for the decision boundary is not obtained, which results in the classifier underfitting the data. This can be seen in the following example where $\lambda = 20$



References

[1] `http://archive.ics.uci.edu/ml/datasets/Abalone`

[2] Coursera course on Machine Learning by Stanford University, Week 3
`https://www.coursera.org/learn/machine-learning/home/welcome`