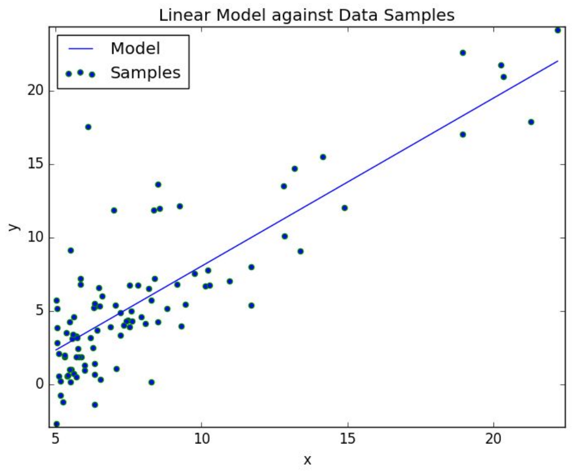
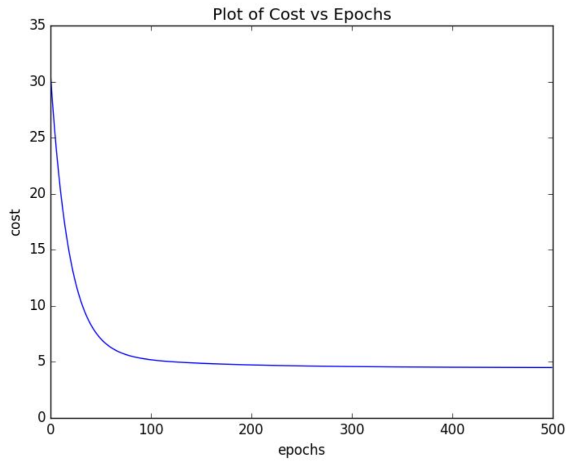
IST597: Foundations of Deep Learning - Assignment 1

Submitted by : Anand Gopalakrishnan (aug440@psu.edu)

Problem 1 - Univariate Linear Regression

Q1.1) Fit your linear regression model on the food truck data and save a plot showing your linear model against the data samples. Furthermore, write code to produce a plot that shows the cost you are minimizing as a function of epoch (or full pass through the dataset).





(a) (b)

Figure 1. (a) shows Linear Model against Test Data Sample and (b) shows cost as function of epochs

Q1.2) You should record the settings you tried and what you found that worked also in the answer sheet.

Values of hyper-parameters used to generate results shown in Fig.1 are:
Learning rate = 0.024
N_epochs = 500

To set the hyper-parameters I started with a learning rate = 0.001 and ran a fixed number of training epochs (say 500) which is clearly more than sufficient for the cost to converge. Then I incremented the learning rate by a factor of 10 and repeated the above process until the point where the learning rate becomes too high causing the cost function to diverge during gradient descent. Looking at the shape of the cost vs epochs curve of these various learning rate settings gives me a good idea about how optimal that learning rate is. (optimal learning rate shows a nice decay shape vs epochs)

Q1.4) How many epochs did you need to converge to a reasonable solution (for any given step size?

Number of epochs needed to converge to a reasonable solution (for any step size) is observed to be around 45000.

Learning has converged when (cost(t) - cost(t-1) < eps) eps=0.00001

Table shown below shows the step-sizes, corresponding number of epochs needed for convergence and final loss values.

Step Size (alpha)	Number of epochs	Final Loss value
0.0001	45662	4.75434
0.001	10952	4.50471
0.01	1733	4.47973
0.024	822	4.47812
0.03	Does not converge	Nan
0.05	Does not converge	Nan

Figure 2. Shows the different step size values tested

Q1.5) Things to discuss: What happens when you change the step-size α ?

When we increase step-size alpha the number of epochs needed for convergence decreases. This is because we are now taking larger steps in the direction of the gradient towards the local minima. However, beyond a certain value for step size the learning is unstable and the cost diverges with each epoch.

Problem 2 - Polynomial Regression & Regularization

Q2.1) With respect to the feature mapping, what is a potential problem that the scheme we have designed above might create?

The feature mapping we've employed to convert the 1-D variable 'x' into a N-D vector **x** is a polynomial feature mapping scheme where **x =** {x, x^2, x^3, x^4, …, x^N}. The problems with this type of feature mapping is as follows:

1. Polynomial functions have poor interpolatory properties. High degree polynomials show large oscillations between exact-fit values.
2. Polynomial models have poor extrapolatory properties. Polynomials may provide good fits within the range of data, but they deteriorate rapidly outside the range of the data.
3. Polynomial models have poor asymptotic properties. Polynomials have a finite response for finite x values and have an infinite response if and only if the x value is infinite. Thus polynomials may not model asympototic phenomena very well.
4. Polynomial models have a tradeoff between complexity/degree tradeoff. In order to model data with a complicated structure, the degree of the polynomial must be high and this can result in highly unstable models.

More generally, a when we use such a "fixed" feature mapping, we are constraining ourselves to a particular class of hypothesis functions (like in this example polynomial functions). Designing such a feature mapping strategy would probably need the human to first understand the properties of the data (i.e. how many modes it has, how is the variance around these modes etc.) and evaluate the fitness of a class of hypothesis functions before using one such "fixed" feature mapping strategy. More importantly in many cases, it might not be feasible for a human to find such an optimal hypothesis class of functions in real-life datasets.

Q2.2) What is one solution to fixing any potential problems created by using this scheme (and what other problems might that solution induce)?

A solution to instability of the polynomial feature mapping is to normalize the input data to have zero mean and unit variance and hence keep the range of the data in a small interval which will prevent the higher order polynomials from blowing up.
Another possible solution for preventing the coefficients from becoming extremely large is to add some regularization term (such as l2 penalty) to the cost function which severely penalizes large values of coefficients. The only drawback is that this adds another hyperparameter to the learning process which needs to be tuned.

An alternative solution would be learn the features themselves. In this technique, we could learn to generate a hierarchical set of features and train the entire system in an end-to-end manner using say a regular feedforward neural network. The advantage of this technique is that these networks can approximate any class of continuous functions and therefore we are now NOT restricting ourselves to any particular hypothesis class of functions but exploring all possible classes of functions to find/learn the best possible function class to use as the feature mapping.

Q2.3) Fit a 1st order (i.e., linear), 3rd order, 7th order, 11th order, and 15th order polynomial regressor to the data in prob2_data.txt.

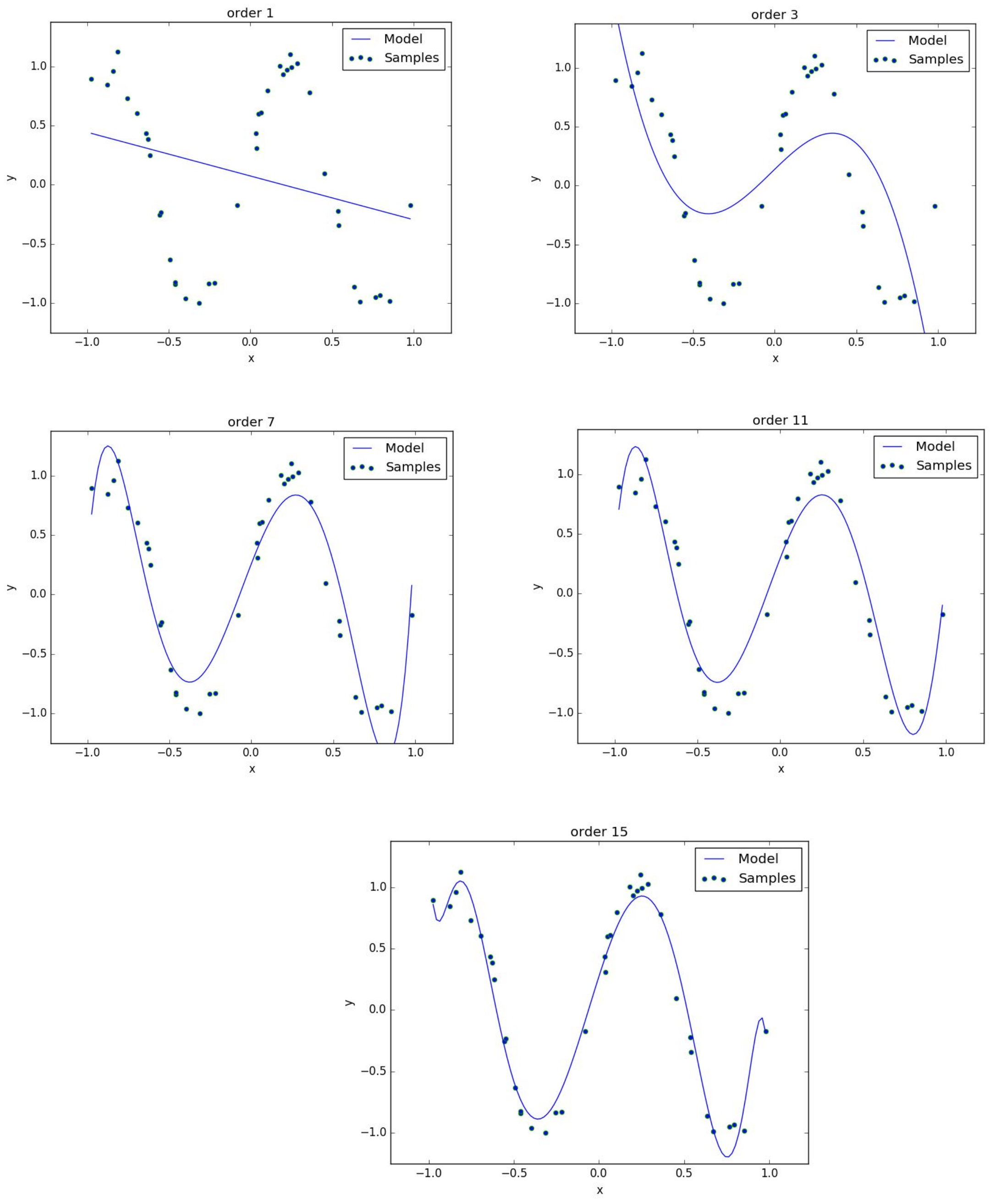


Figure 3. Shows the different order of polynomial regression models fit to data

Q2.4) What do you observe as the capacity of the model is increased? Why does this happen?

As the model capacity increases it's able to fit the data much better. This is because as the model capacity increases, we're essentially increasing the number of free parameters of the model that can be learnt during training. Now the model with greater number of parameters can better capture the complexity of the data distribution. However, as we can see in order=15 case, it starts to overfit to the noise in data sampling process and will not generalise well if appropriate regularization is not applied.

Q2.5) Refit your 15th order polynomial regressor to the same data but this time vary the β meta-parameter using the specific values {0.001, 0.01, 0.1, 1.0} and produce a corresponding plot for each trial run of your model.

Following plots show the polynomial regressor for the β values {0.001, 0.01, 0.1, 1.0}

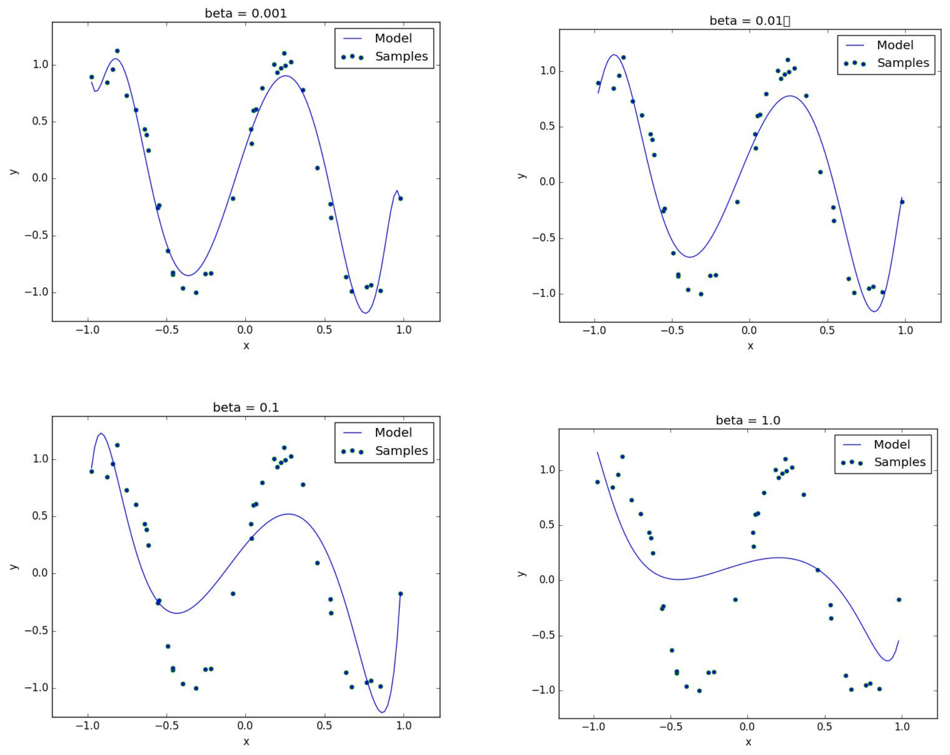


Figure 4. Shows the polynomial regression (order =15) fit to data at different l2-regularization strengths with other hyperparameters fixed at alpha = 1.55, n_epochs=5000 and eps=0.0

Q2.6) What do you observe as you increase the value of β ? How does this interact with the general model fitting process (such as the step size α and number of epochs needed to reach convergence)?

As you increase the value of β , the number of epochs needed to reach convergence decreases. This is because as you increase the value of β , you are in effect reducing the model capacity. In other words, since the l2-regularization strongly penalizes larger values of weights it effectively reduces the effective search space (since now larger values are in effect not "good" fits) for the learning process in the weight space and hence the learning converges faster.

In simpler terms, since the increase in value of β is equivalent to lowering the effective model capacity, now the model has lesser free parameters to optimize and hence converges faster.

In relation to step size, we reach convergence faster with a greater value of step size (as long it's within stable learning regime where cost decays with epochs) as we're taking greater strides in the direction of the gradient and hence reach the local minima faster. Also step size and number of epochs are inversely related, i.e. lower the step size greater the number of epochs needed for convergence and vice versa.

Step-Size (alpha)	Number of Epochs	Final Loss value
0.001	48455	0.159437
0.01	23259	0.094417
0.1	5768	0.082648
1.0	1281	0.008013

Figure 5. Shows relationship between β and step size α and number of epochs needed to reach convergence. For results shown in table order =15, β = 0.1 and convergence parameter (eps) = 0.000001 are kept fixed throughout.

Q2.7) Comment as to how many steps it then took with this early halting scheme to reach convergence.

Beta	Number of epochs	Final Loss value
0.001	5611	0.00931
0.01	4320	0.02760
0.1	904	0.07999
1.0	151	0.17366

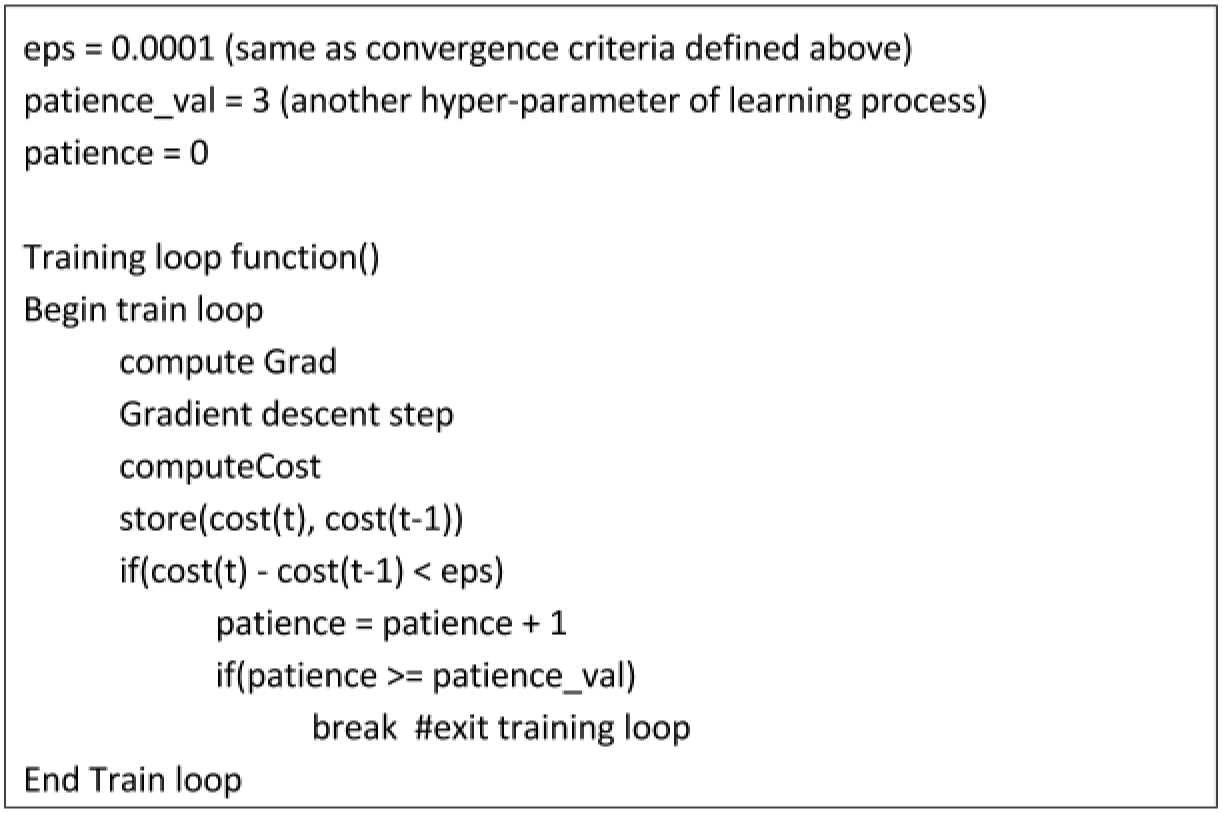
Figure 6. Order = 15, eps = 0.000001, alpha = 1.6 kept fixed throughout

Q2.8) What might be a problem with a convergence check that compares the current cost with the previous cost (i.e., looks at the deltas between costs at time t and t − 1), especially for a more complicated model? How can we fix this?

For more complicated models (highly non-linear models) and loss functions, the error surface will not be convex and in such cases, the error surface will display complicated topologies where we may have to traverse over flat/nearly flat stretches before we begin to descend to lower local minima. The complicated error surface will have many local minima some better than the others.

In such cases, a better solution would be to show greater patience when traversing these flat surfaces. We could use an additional patience criteria defined as follows:

Convergence check function:



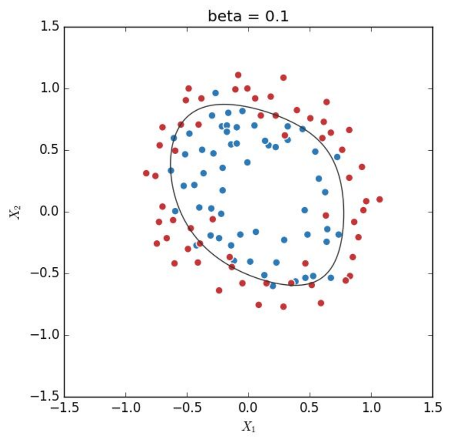
```
eps = 0.0001 (same as convergence criteria defined above)
patience_val = 3 (another hyper-parameter of learning process)
patience = 0

Training loop function()
Begin train loop
    compute Grad
    Gradient descent step
    computeCost
    store(cost(t), cost(t-1))
    if(cost(t) - cost(t-1) < eps)
        patience = patience + 1
        if(patience >= patience_val)
            break  #exit training loop
End Train loop
```

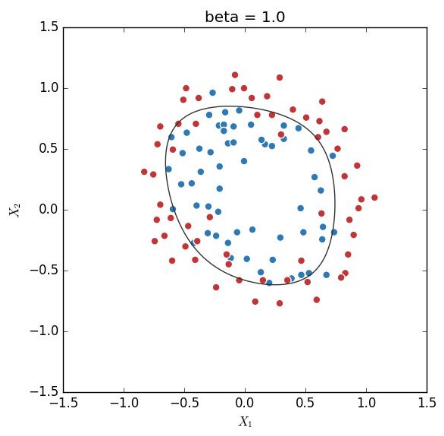
This function when substituted with the earlier convergence check function will wait for “patience_val” number of epochs in which the cost did not sufficiently decrease before it terminates the training. If we set patience_val high enough this can help us traverse the nearly flat surfaces and proceed towards deeper minima.

Problem 3 - Multivariate Regression & Decision Boundaries

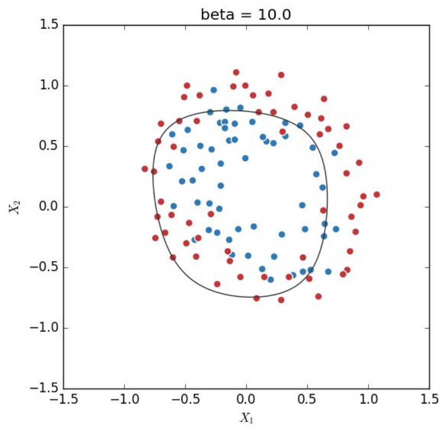
Q3.1) For the last part of this assignment, re-fit your logistic regression but this time with β = {0.1, 1, 10, 100}, again, tuning the number of epochs and the learning rate. Copy each of the resultant contour plots to your answer sheet and report your classification error for each scenario.



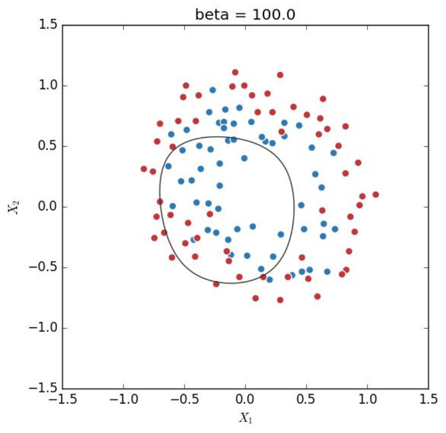
(a) error % = 16.1016



(b) error % = 16.9491



(c) error % = 25.4237



(d) error % = 33.898

Figure 6. Shows the decision boundary of a p=6 polynomial regression model and classification error rates at different β values

Q3.2) Comment how the regularization changed your model's accuracy as well as the learned decision boundary. Why might regularizing our model be a good idea if it changes what appears to be such a well-fit decision boundary?

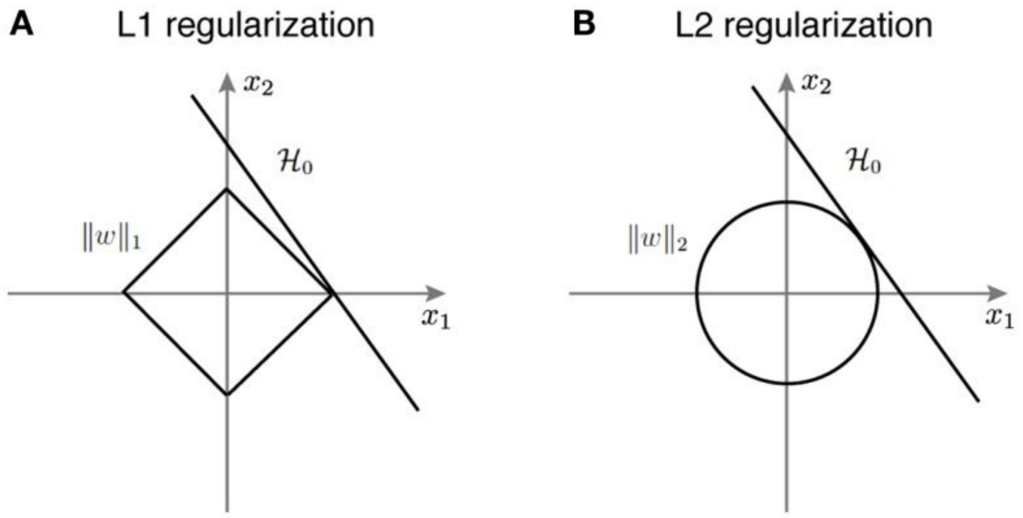


Figure 7. A schematic illustration of how different regularization terms lead to sparse and non-sparse solutions in the linear classifier.

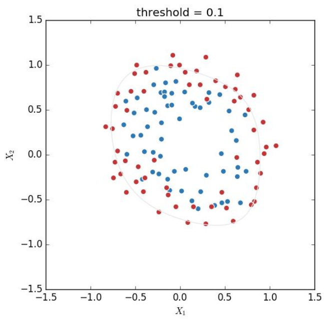
Regularization helps in preventing the model from overfitting itself to the training data. It attempts to simplify the model and thereby reduces variance but it increases bias. If our model capacity is too large for the training set, it essentially uses its free parameters to memorize the training data points and thereby will not generalize well to new data. In this case, adding regularization helps to reduce effective model capacity and helps the model generalize better to unseen testing data.

The L2 regularization we've used in this case, creates a spherical ball of tension, wherein as the parameters go farther away from the origin the tension increases in proportion to the squared radius and creates a pulling force that keeps the parameters from growing too large. From Figure 6. we can see that as the strength of regularization term is increased the decision boundary starts to get more strongly attracted towards a circle centered at 0 and having a radius of 0.5 (as observed most clearly in beta=100.0 case)

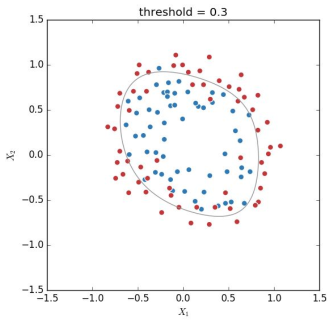
However, we must set the regularization strength optimally. As seen in beta=100 case, if the regularization strength is too large it oversimplifies the model thereby making it unable to capture the complex distribution of data and results in a poor fit.

Experimental observations with different threshold settings for logistic function

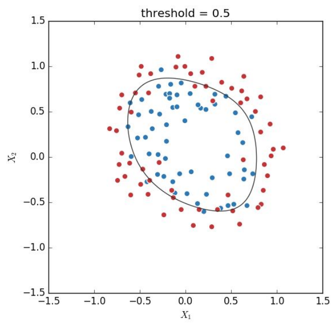
Hyper-parameter settings kept fixed throughout these experiments are:
step size = 4.5 ; eps = 0.0 ; beta = 0.1 ; n_epoch = 500



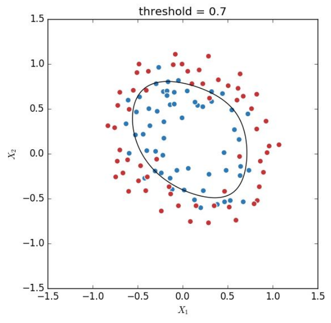
(a) th = 0.1 error% = 32.203



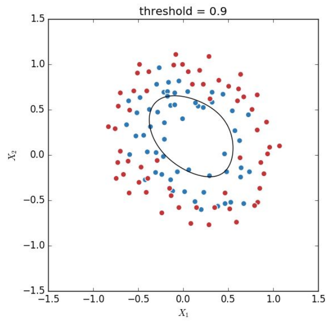
(b) th = 0.3 error% = 16.101



(c) th = 0.5 error% = 16.949



(d) th = 0.7 error% = 18.644



(e) th = 0.9 error% = 38.135

Figure 7. Shows the decision boundary drawn for different values of threshold with all other hyper-parameters kept fixed

In some specific machine learning problems we're dealing with has different costs for false positive and true negatives then it makes sense to appropriately set threshold to different value other than 0.5.
As we can observe from the decision boundaries above, for the extreme cases of threshold=0.1, the model is hardly misclassifying blue dots as red dots but it's classifying a lot of red dots as blue dots. And in other extreme case of threshold = 0.9, it's does NOT misclassify any red dots as blue dots but it's misclassifying a lot of blue dots as red dots.