COL774 – Machine Learning

Assignment 1

Nilaksh Agarwal
2015PH10813

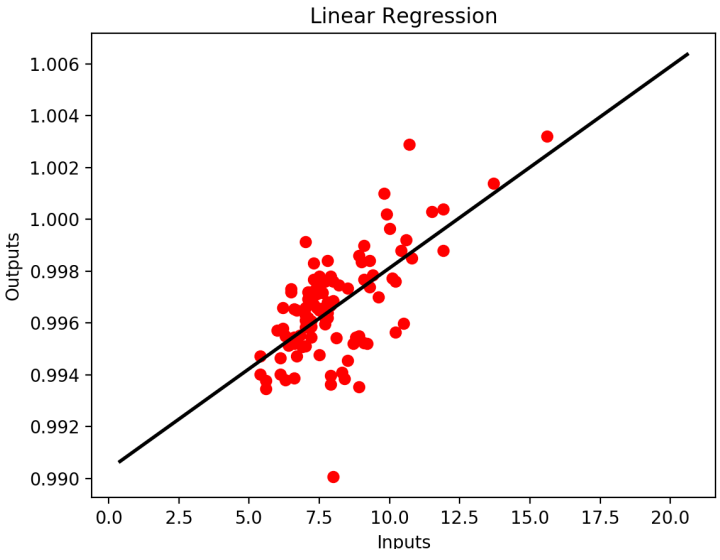
Part 1:

In this part, we implement linear regression in 1 dimension. So, our curve fitted turns out to be a line. Since a line can be perfectly represented by two parameters (slope, intercept) we minimize our cost function to find values of this.
To speed up this process, I use numpy's matrix multiplication to directly get our predicted values and gradients for all parameters (instead of summing over all examples individually)

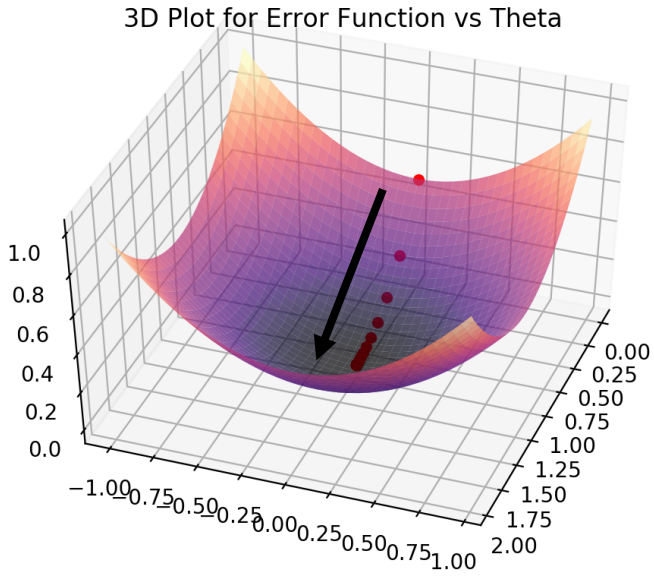
The stopping criteria is two-fold. First we have a max_iter which provides a hard limit on the number of iterations we update our theta parameters over. Secondly we have a condition over the change in our cost function (J[i+1] – J[i]) which has to be less than 10^{-10} times our learning rate. This is done so that even a very small learning rate will still iterate and not instantly converge just due to minimal increments in J.

a) Learning Rate: 0.3
Stopping Criteria: $(J[i+1] - J[i]) / LR < 10\text{-}10$
Final Parameters (θ): [0.99034, 0.00078]

b)

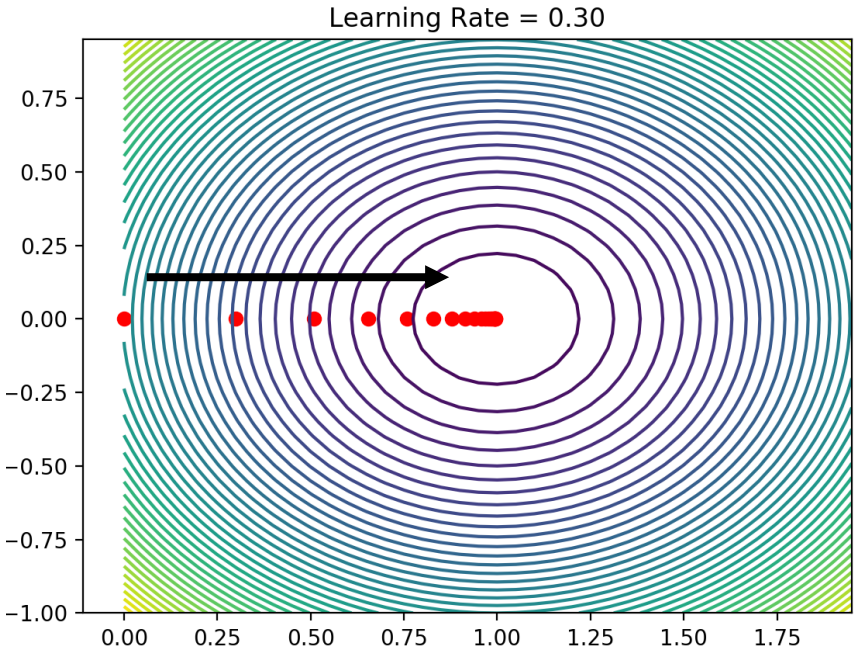


c)



Arrow Shows direction of moving cost function

d)



Arrow Shows direction of moving cost function

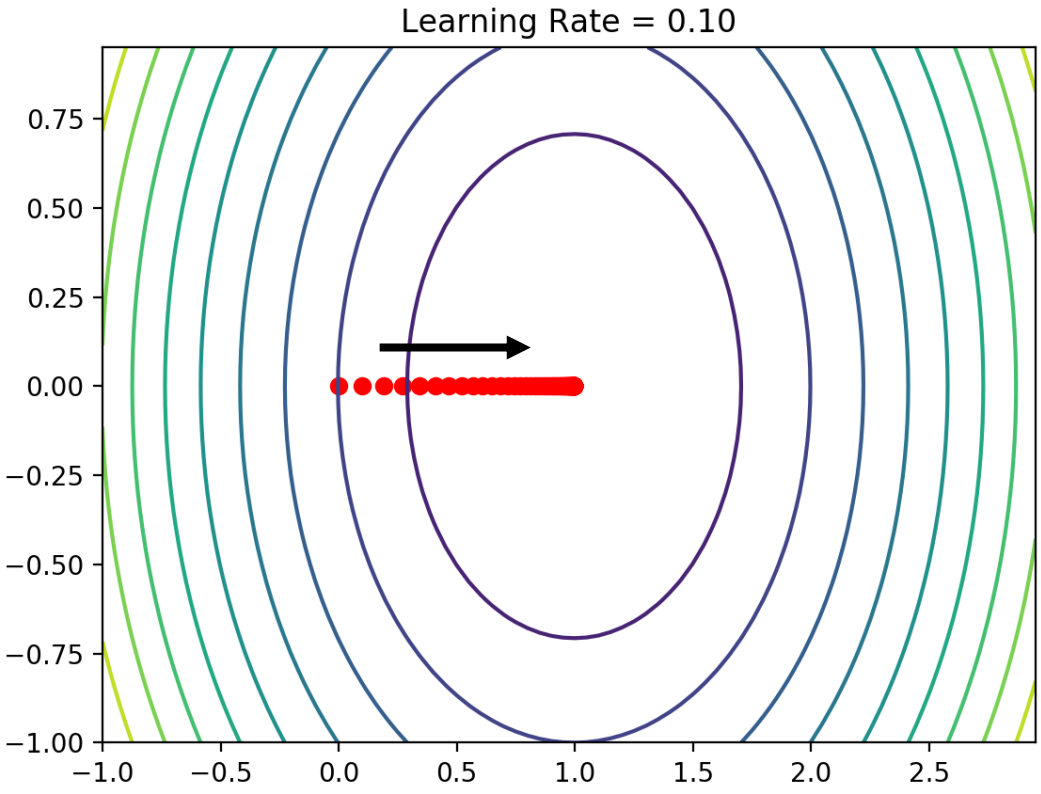
e) On increasing the learning rate from 0.1 to 2.5, we see that initially, for [0.1, 0.5, 0.9] the cost function moves faster (in fewer iterations) towards the minima.

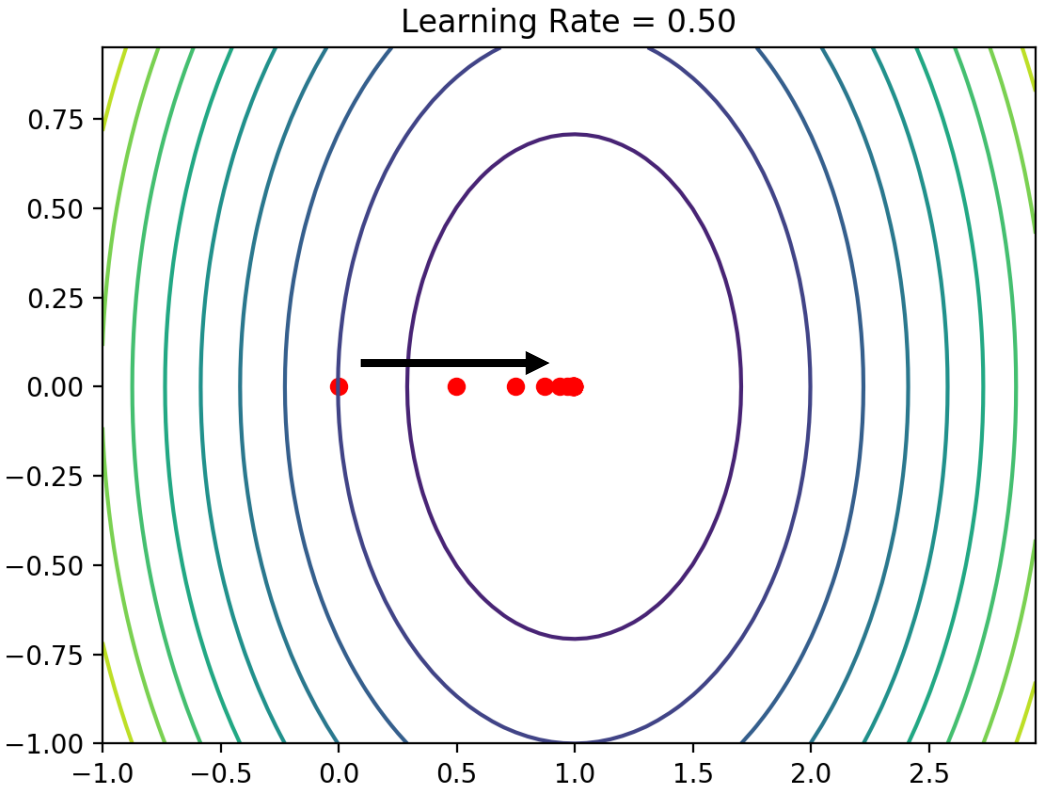
After that for [1.3,1.7], the cost function overshoots the minima, then comes back but overshoots it again, and keeps spiraling towards the minima.

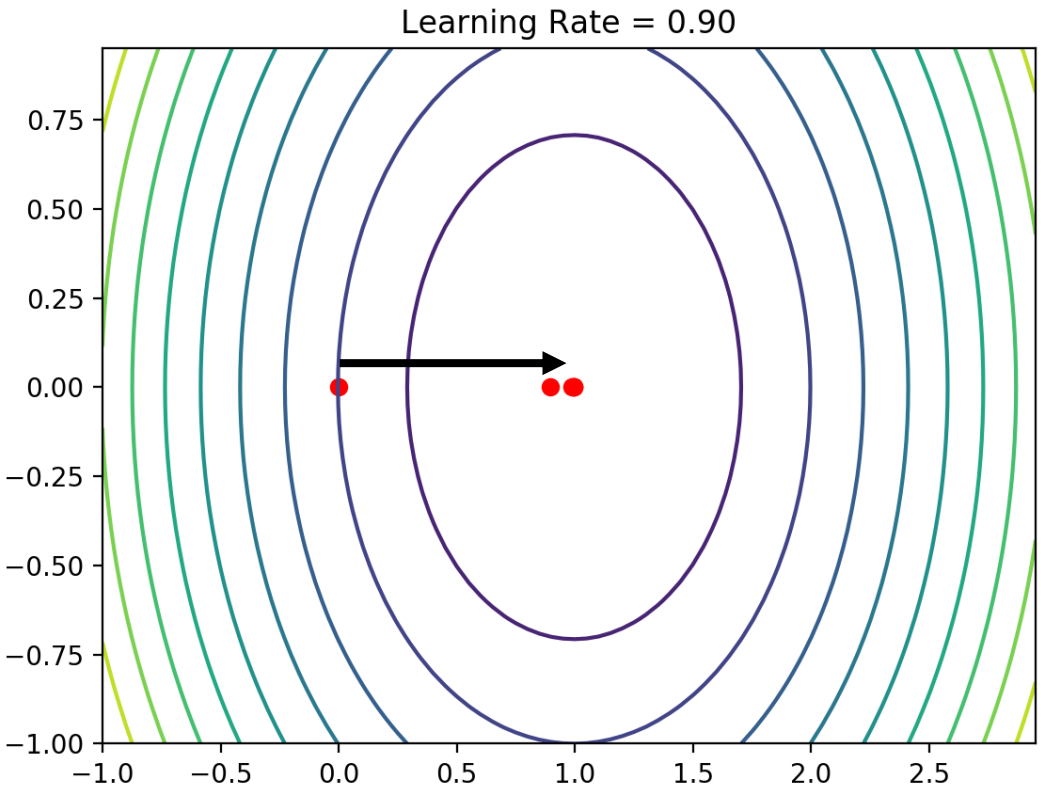
For [2.1,2.5], the cost function overshoots the minima and keeps increasing the overshoot, progressively, the cost function keeps increasing at each iteration. In this way it never reaches the minima.

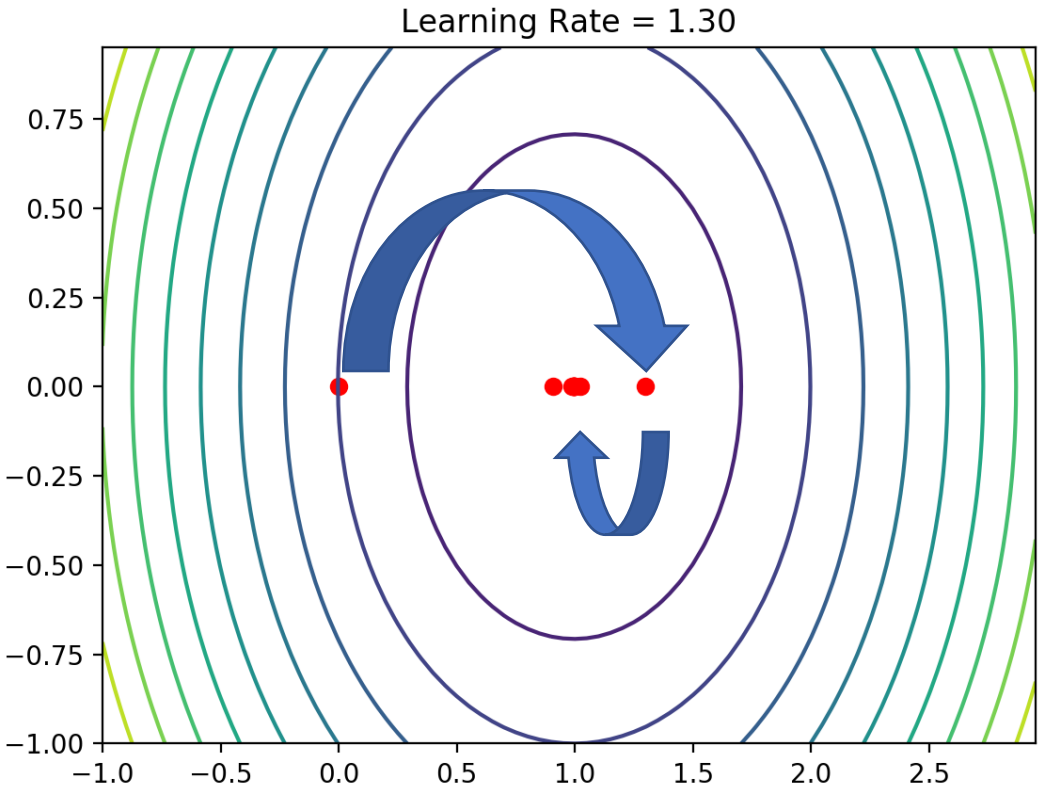
Plot always starts from [0,0]

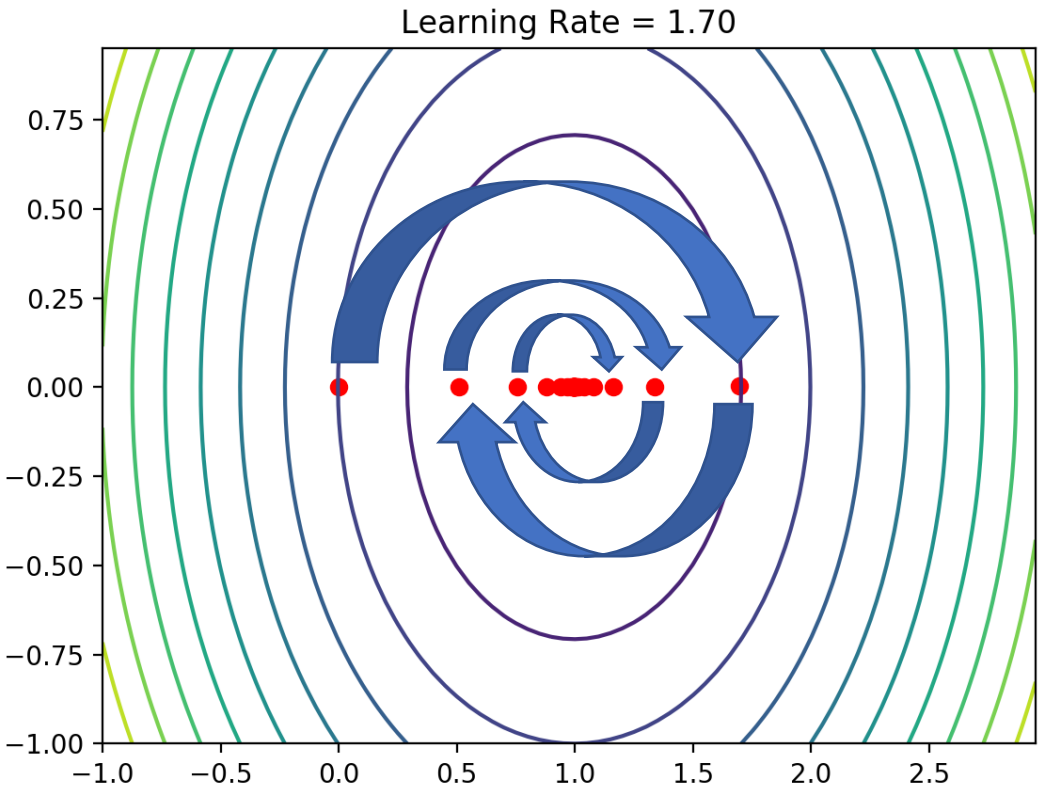
[0.1, 0.5, 0.9] – linear reducing cost function
[1.3, 1.7] – reducing spiral cost function
[2.1,2.5] – increasing spiral cost function

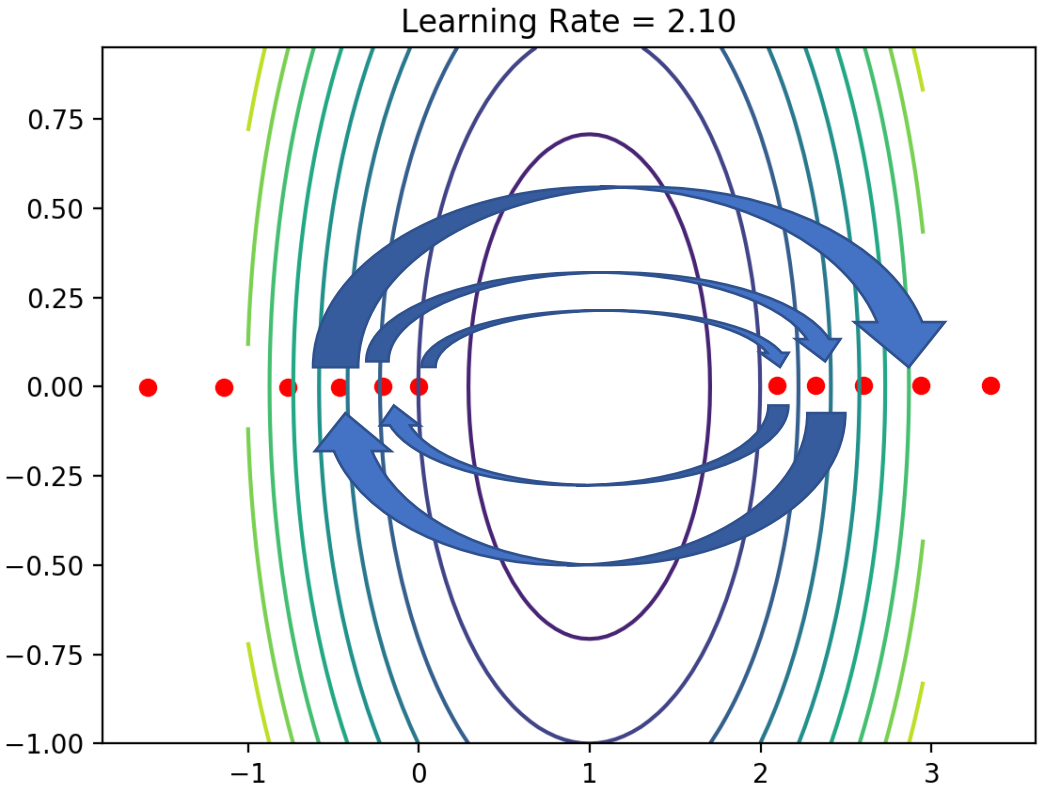


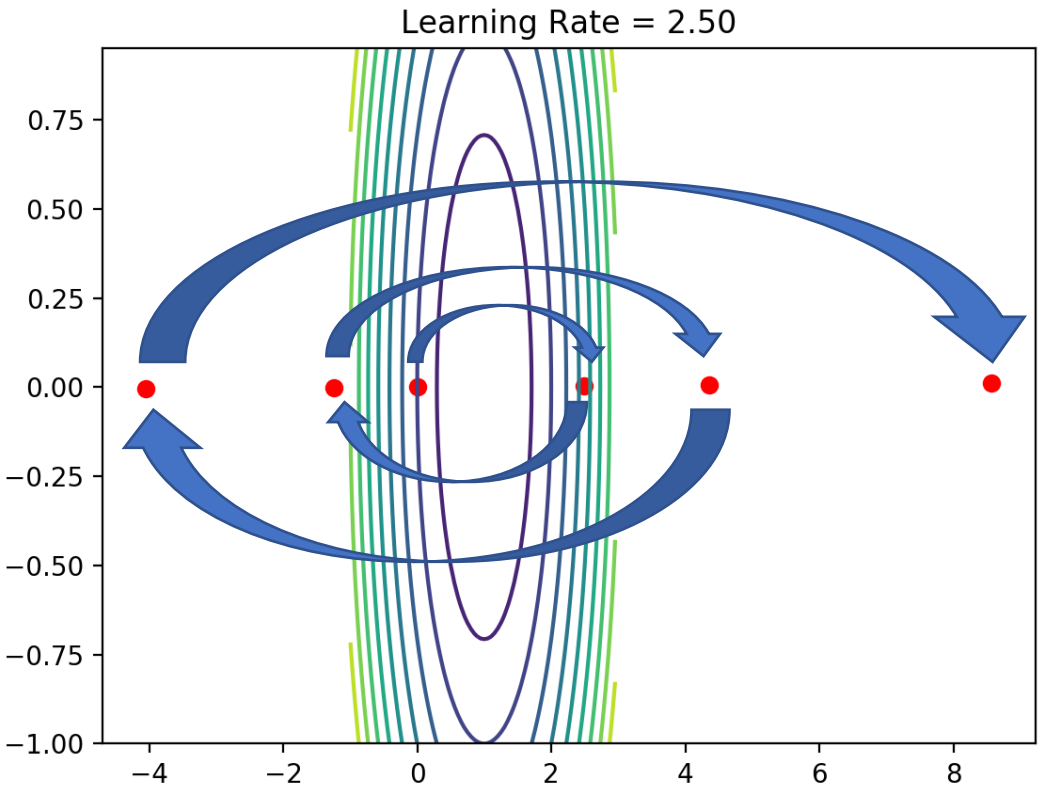

Learning Rate = 0.50
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Learning Rate = 0.90
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Learning Rate = 1.30
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Learning Rate = 1.70
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Learning Rate = 2.10
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1
0
1
2
3

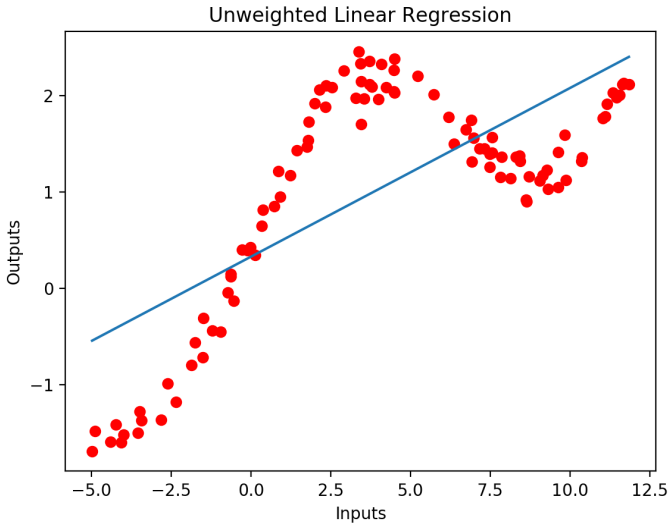

Learning Rate = 2.50
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−2
0
2
4
6
8

Part 2:

In this part, we implement weighted linear regression. We calculate the weights matrix for each individual X[i] point with respect to all m points in the space. Using this calculated W matrix, we put it into the derived normal equation formula to get the value of the parameters for each point individually. So, we have multiple sets of θ, one for each point in our X-space.

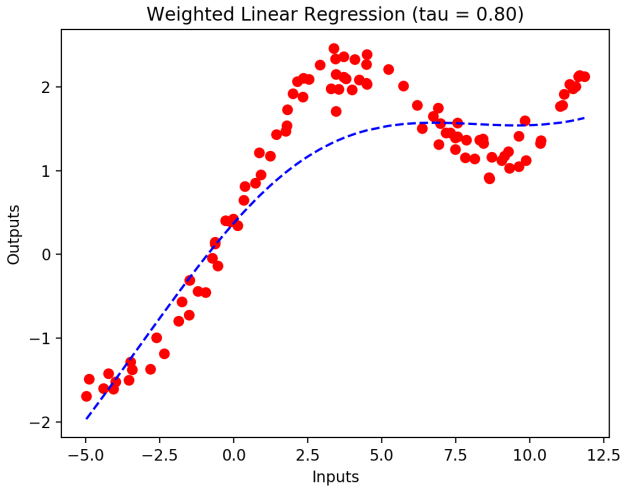
a) $J(\theta) = \frac{1}{2m} (X\theta - Y)^T (X\theta - Y)$

$\theta = (X^TX)^{-1} X^TY$

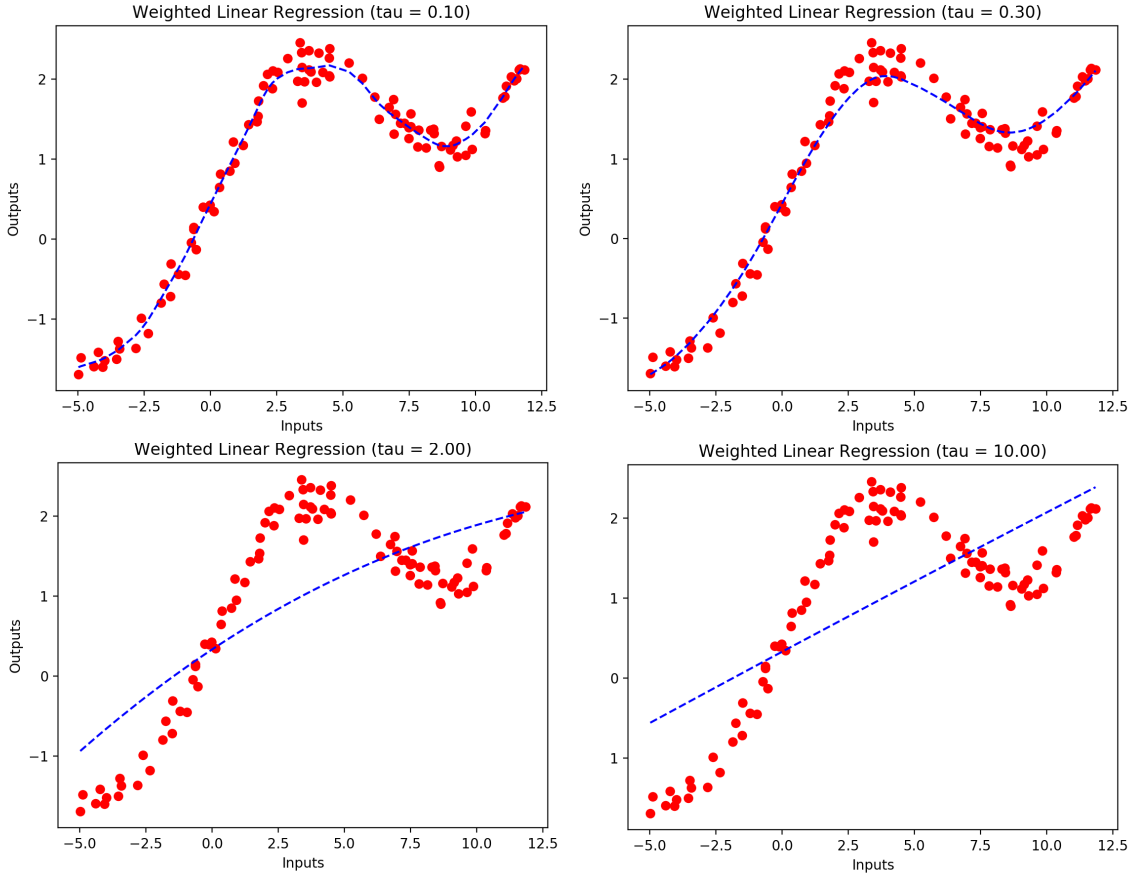


b) $J(\theta) = \frac{1}{2m} (X\theta - Y)^T W (X\theta - Y)$

$\theta = (X^T(W+W^T)X)^{-1} X^T(W+W^T)Y$



c) When τ is too small, we overfit to the data, i.e., the curve follows the data too exactly, and it might not be able to generalize to new points.
When τ is too large, we underfit to the data, i.e., the curve is not a good general representation, and the curve is unable to capture some features of the data.



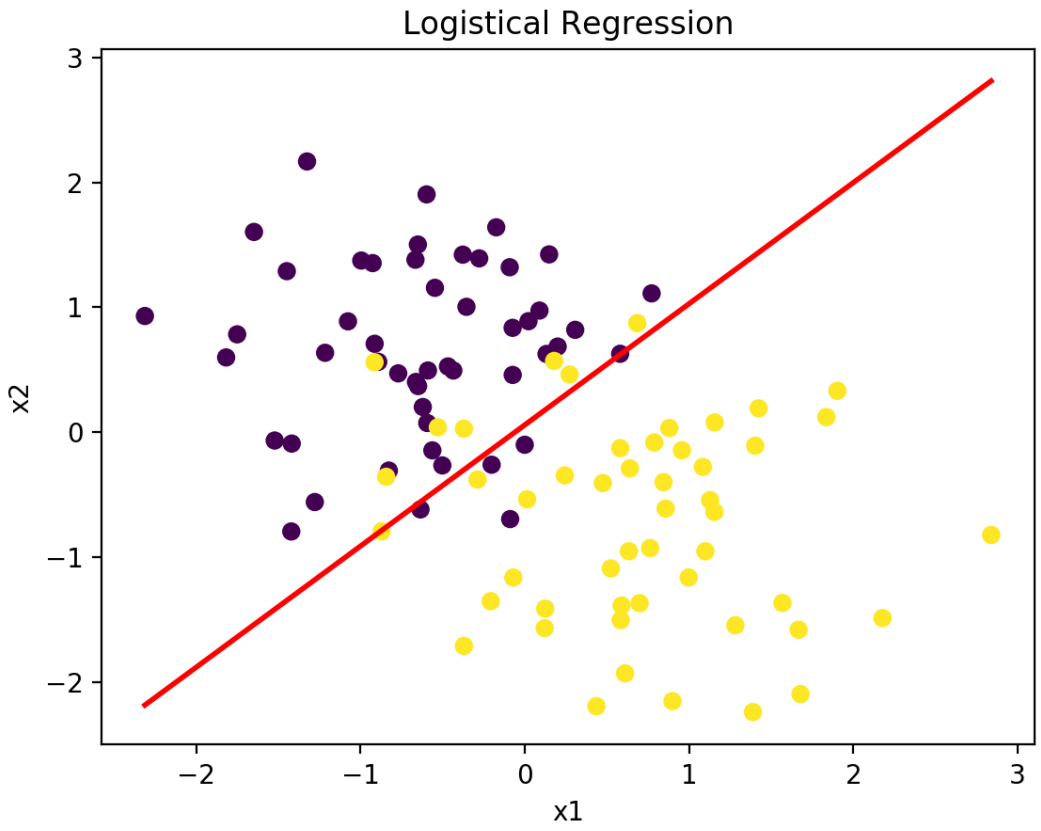
Part 3:

Here we implement the newtons method to calculate the values of the parameters. We define two functions to calculate the gradient and hessian of J w.r.t. the parameters. For both gradient and hessian, we use the right side derivative since for our chosen initial θ = 0, the functions on 0+ε and 0-ε were equal and cancelled out. Similarly for the hessian. This method is much faster (no. of iterations wise) than regular gradient descent but can be costly if there are a large number of parameters.

a) $\theta := \theta - H^{-1}\nabla_\theta \ell(\theta)$.

Final Parameters (θ): [0.15547, 2.59393]

b) Data plot with decision boundary



Part 4:

Here we implement Gaussian Discriminant Analysis using both a quadratic and linear separator. First we calculate the values of means and co-variance(s) from the data. Now, using these values we obtain the parameters of the line(linear) and parabola(quadratic) that becomes the decision boundary for the same.

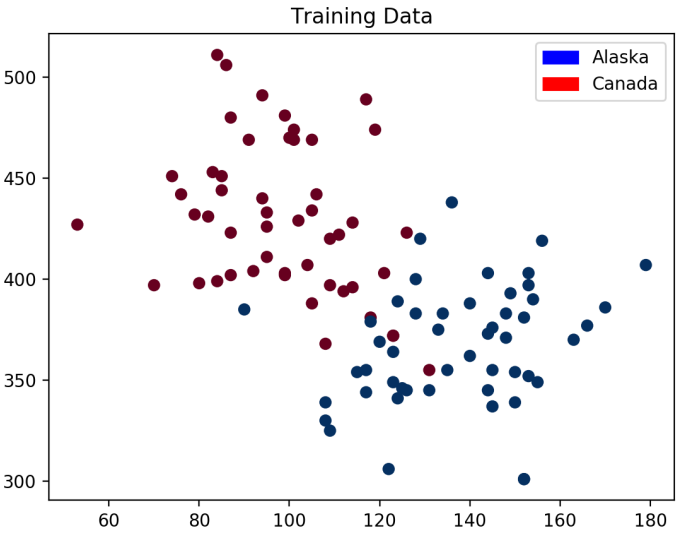
a) Parameter Values:

μ_0 = [98.38, 429.66]

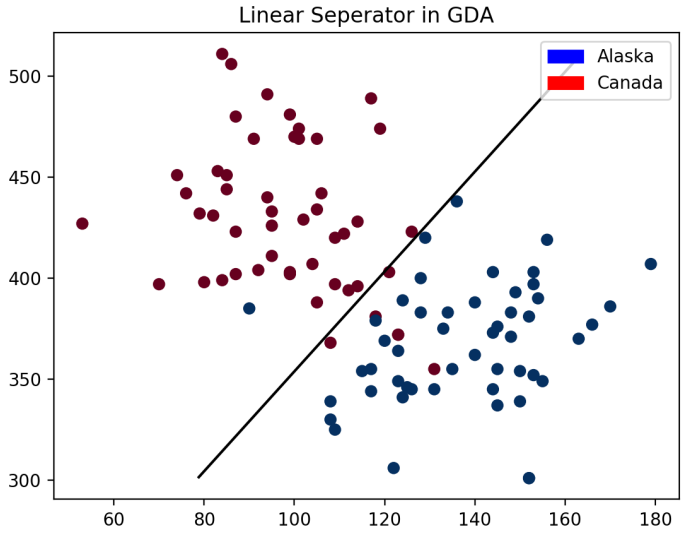
μ_1 = [137.46, 366.62]

Σ = [287.482, -26.748
-26.748, 1123.25]

b)



c)



d) Parameter Values:

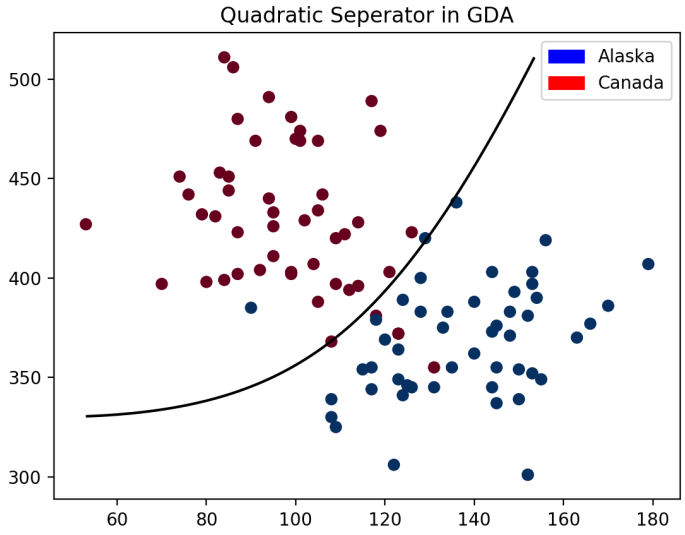
μ_0 = [98.38, 429.66]

μ_1 = [137.46, 366.62]

Σ_0 = [255.3956, -184.3308
-184.3308, 1371.1044]

Σ_1 = [319.5684, 130.8348
130.8348, 875.3956]

e)



f) On analysing the two separators, we see that the Quadratic Separator is able to give a slightly better distinction between the two classes.

This is because the linear separator assumes that the observations vary consistently across classes. This might not be the case, and a more general separator (quadratic) which models the covariance of each class independently is able to do a slightly better job.

So we see, that for a generalisation perspective, the quadratic separator is able to make better predictions about the underlying distribution of the individual classes.

Files included with this:

1_part.py
2_part.py
3_part.py
4_part.py
run.sh