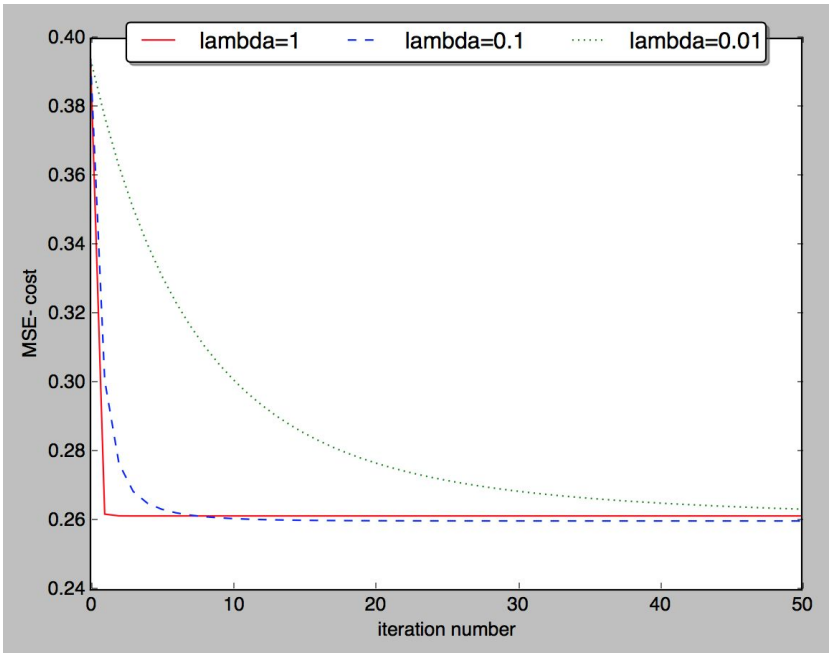
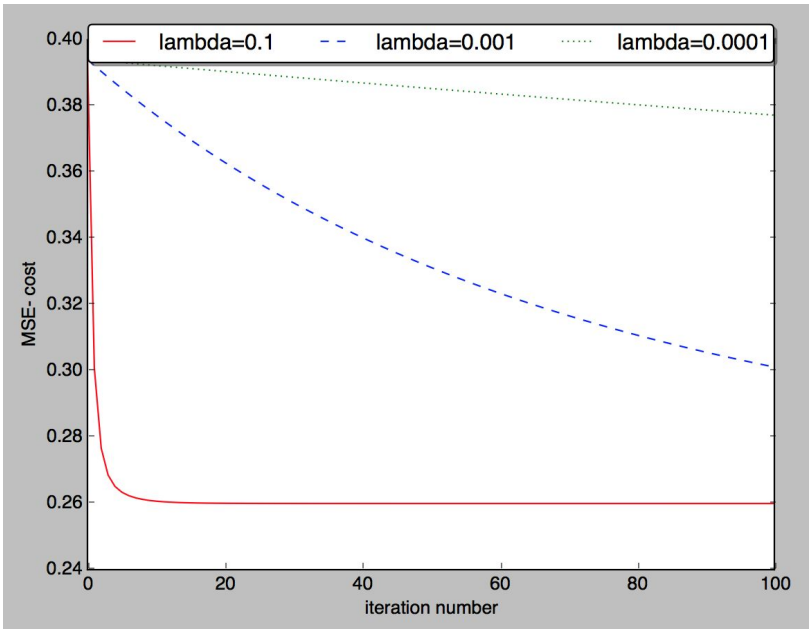
Web Economics Assignment B- Linear and Logistic Gradient Descent by Marcin Cuber

In this assignment I have imported the spambase.data file and then divided it into X_matrix and Y_vector. X_matrix is the entire data set apart from the last column which forms Y_vector. After splitting the data I preconditioned it using z-score (standard score). Z-score was only applied to X_matrix since we did not need to normalize Y_vector which has 0 or 1 entries. Once the data was preconditioned I formed 10 folds and using shuffle function (random library) I randomised data so that we don't have grouped spam emails or non-spam emails. In the original data set it is the case so this way we eliminated this disadvantage.
Randomisation of folds is particularly useful when dealing with stochastic gradient descent since we are taking a single example and we modife mean squared error (MSE). Such problem would not occur in batch gradient descent as we take entire data from a single fold and then modify MSE. Nevertheless, the above preconditions and randomisation have been applied to all gradient descent techniques used for this assignment.

Linear Stochastic Gradient Descent
In order to explore the effect of different fixed learning rates on gradient descent I used **convergence tolerance** = 0.0001.
Following graph presents results for learning rate (lambda) equal to 1, 0.1 and 0.01 over 50 iterations.



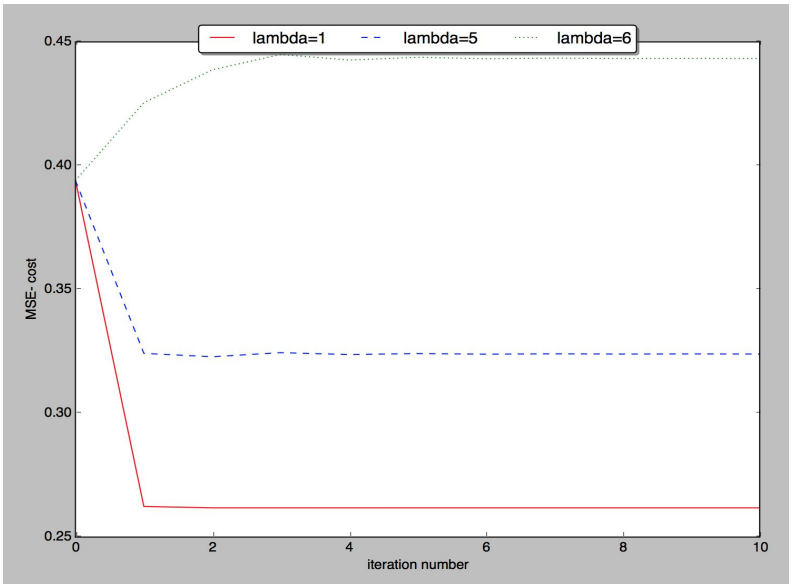
The results suggest that all three rates converge to MSE = ~0.26. However, we should point out that lambda = 1 has drop very quickly in first iteration therefore it is safer to use lambda =0.1. Although smaller lambda requires more iterations, we can see that after 10 iterations it reaches slightly smaller MSE than for lambda = 1.



Let's know consider another set of lambda values with more iterations.

Again, on the above plot we can see that lambda= 0.1 is nicely converging after around 10 iterations. However, for smaller lambda which are 0.001 and 0.0001 we can can easily see that the curves are converging a lot slower and 100 iterations is not even close to being enough for them to converge. Therefore we can conclude that rates between 1 and 0.1 should be good enough for our data.

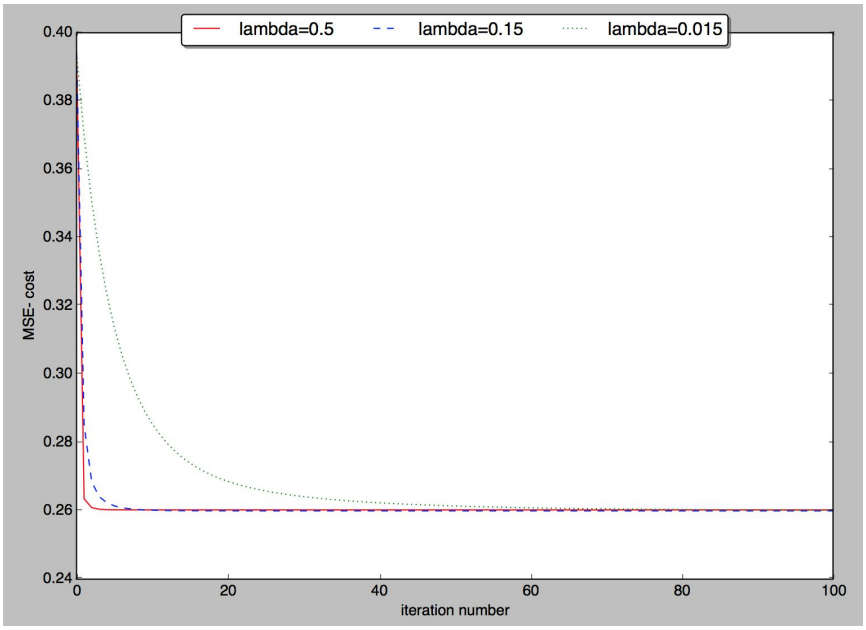
Now, lets see last example on which we will see that setting learning parameter to high leads to divergence and unpredicted results.

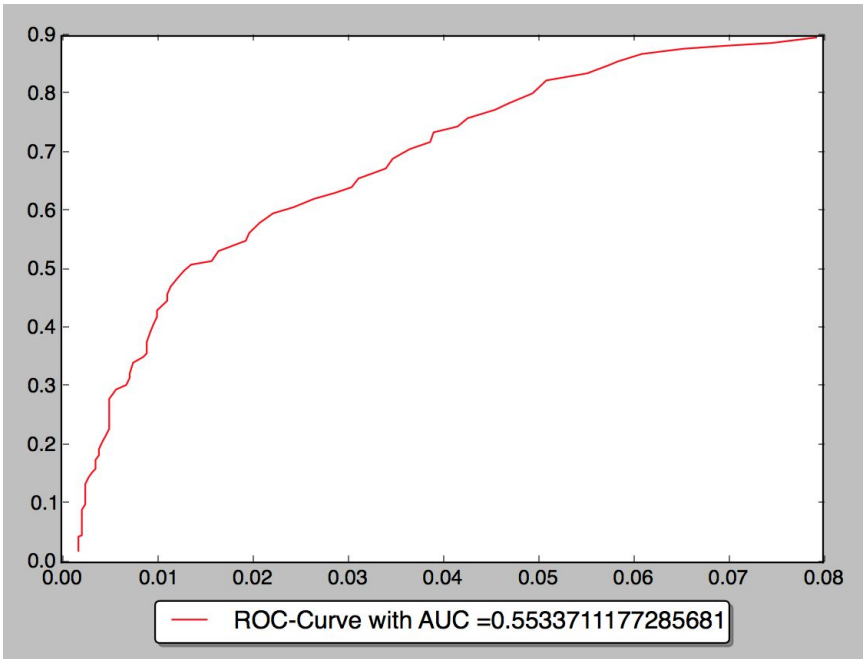


So, we can clearly see that lambda = 5 or 6 are just to high and do not converge at all and goes the other way.

We explored effects of different learning rates so we can move onto producing more curves and additionally present ROC curve with area under the curve (AUC).

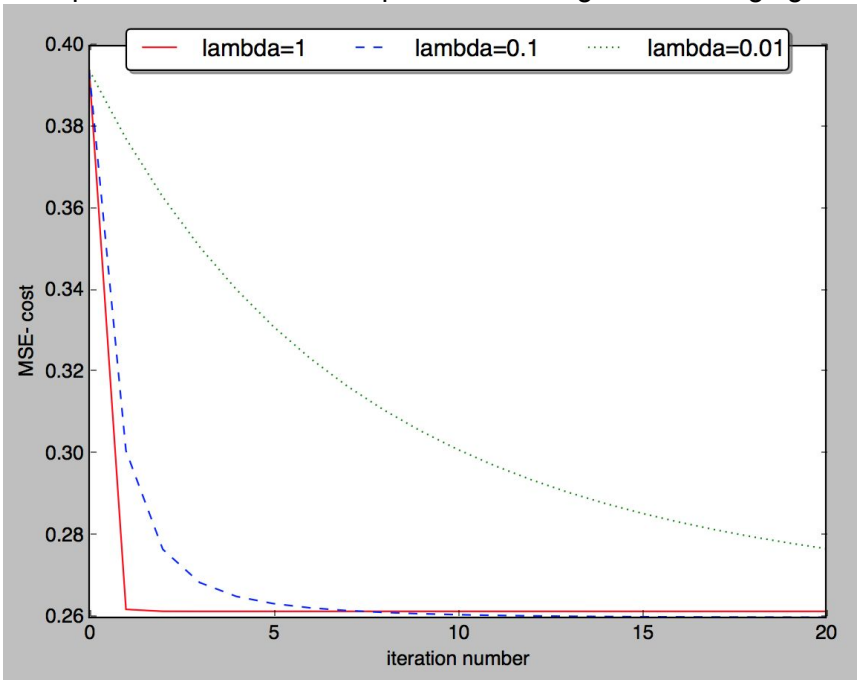
Lets use the following data where all lambdas converge in 100 iterations:

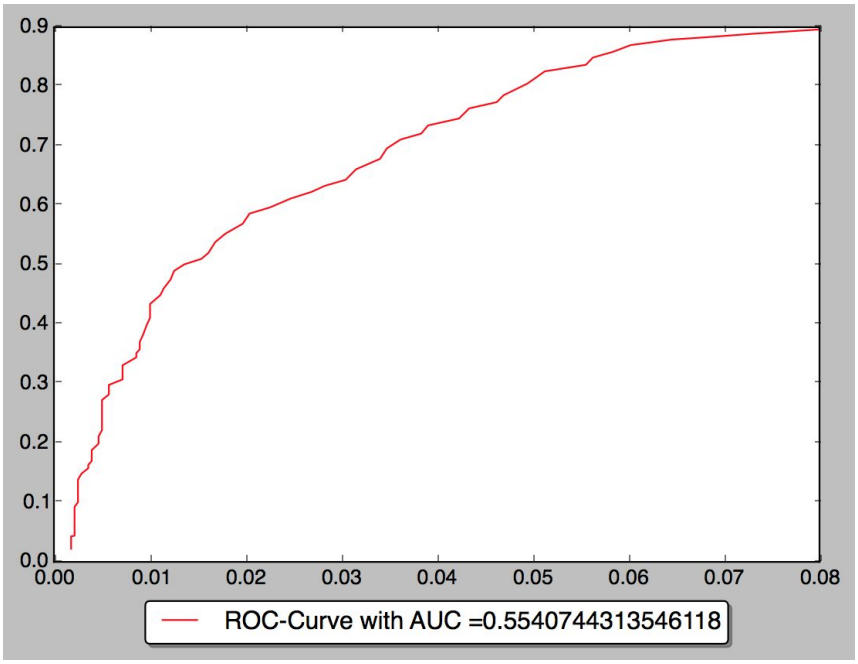




The values that we use to plot ROC curve are true positives along y axis and false positives along x axis. We get a very low score of 0.55337.

Let's present one more example with learning rate converging over 20 iterations.





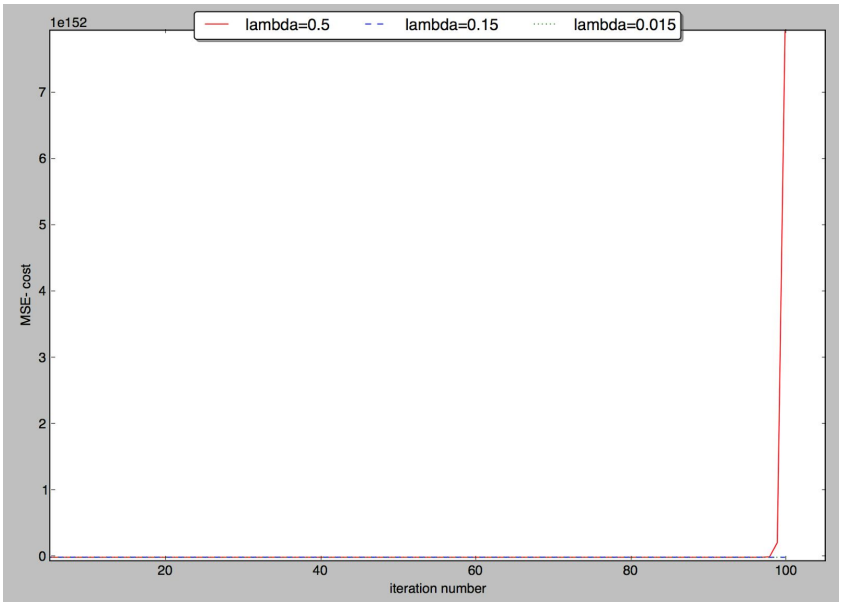
We can again see lambdas converging, it doesn't matter that lambda with value 0.01 does not yet converge since we use the lambda with minimum errors to create ROC curve. The AUC we get this time is again very low: 0.55407.

Note that convergence criteria that I have used for generation of the curves is number of iterations and convergence tolerance. At each iteration I am comparing errors to see whether they are getting smaller and if they smaller than convergence criteria we stop.

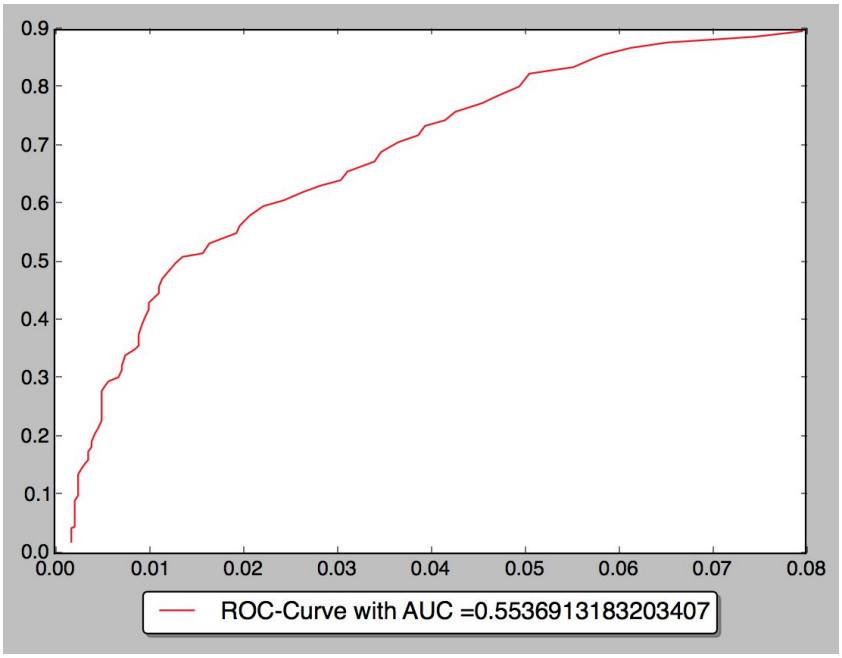
Linear Batch Gradient Descent

For the Batch gradient descent we use the same value of convergence tolerance and we will present examples which illustrates three learning rates converging over number of iterations. We will keep the lambdas the same so that we can compare them later.

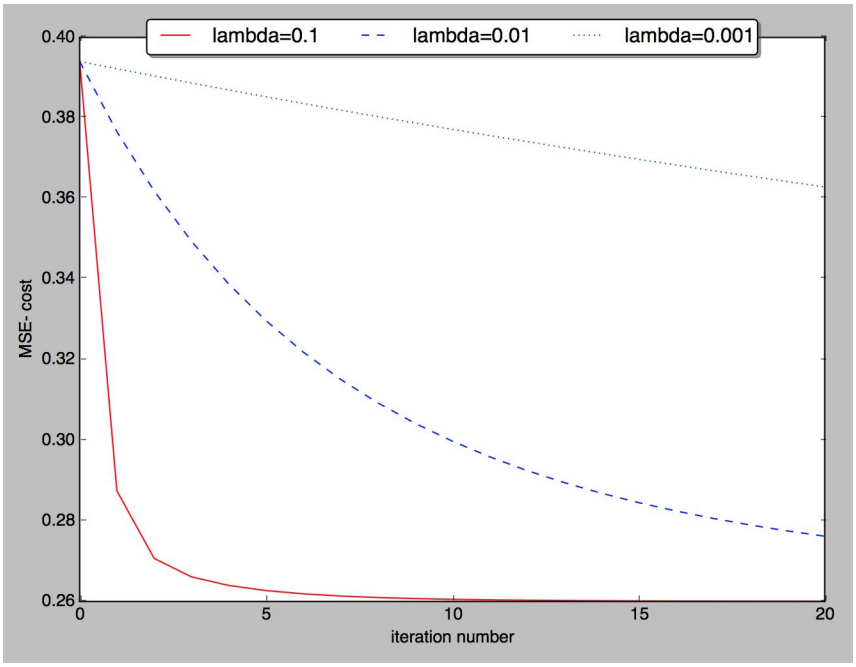
Let's consider lambdas equal to 0.5, 0.15 and 0.015 over 100 iterations.

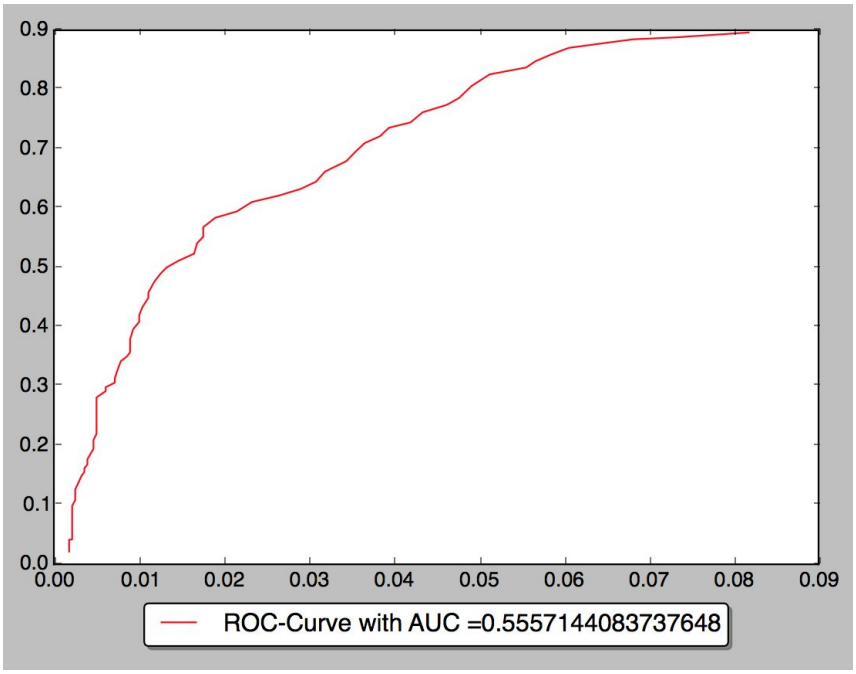


From the plot we can see that lambda with value 0.5 is too high to use as a learning rate and therefore we get a huge MSE. Nevertheless our ROC curve is fine since it takes the smallest errors and the curve looks as follows:



Similar to stochastic gradient descent ROC curve with the same values has similar AUC score. This is something that we expected since we take enough iterations for lambda to converge and therefore we have the same number of points to analyse.
For the second set of values we know that lambda = 0.5 is too high so we won't use lambda = 1 in the next example, instead we will use lambda = 0.001.





From the plots above we can see that lambda = 0.1 works well and converges after around 10 iterations whereas others converge slower. We only considered 20 iterations since it is enough

for us to reach the MSE that we were looking for. Consequently the ROC curve is using data from lambda = 0.1 as they have the smallest error values. These data points give us again very low AUC of 0.55571 which fairly close to what we got in stochastic gradient descent.

Linear Stochastic Gradient Descent vs Linear Batch Gradient Descent

In order to compare these linear methods I will use the above information and also provide an extra measure which is time taken to produce the results. For time measures I have used the same learning rates which are 0.1, 0.01 and 0.001. These lambdas were used because in both methods they converge and they are not too high.

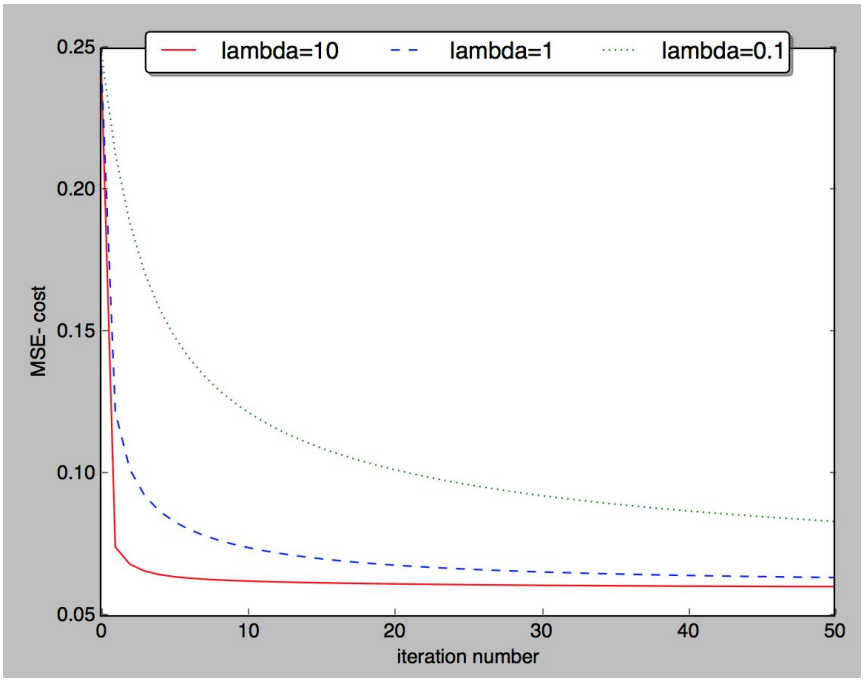
Number of iterations:	Linear Stochastic time(seconds):	Linear Batch time(seconds):
20	45.90	28.05
50	113.84	67.93
100	222.42	135.77
200	447.93	270.74

The obtained time results clearly show that Batch method in which we consider entire data set rather than a single feature perform a lot faster than Stochastic approach. In our case it is important to note that the data set is a matrix of size 4601 x 58. Considering data sets where there are millions of records, our set is very small and this could be the reason why Batch approach works more efficient. In much larger data sets it can turn out that Stochastic approach will converge at MSE a lot quicker. The reason for this is that Stochastic method takes a single data sample, calculates the MSE and updates it. In some cases it could be an advantage since it will reach convergence point quicker but it can also be a disadvantage which will result in not reaching the minimum error point.

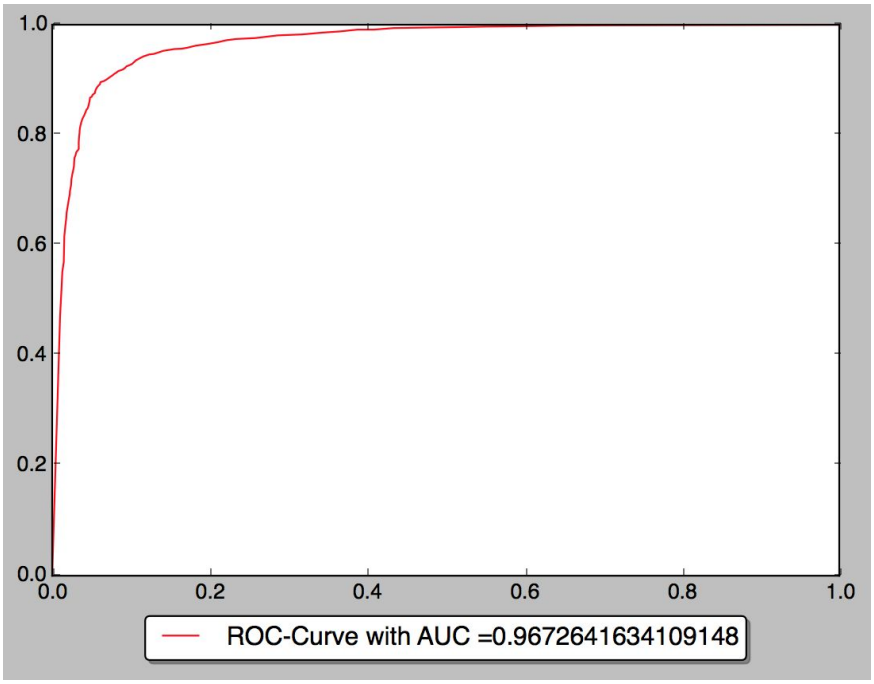
Comparing convergence rates, we could observe that lambda = 0.5 in the Stochastic approach required around 3 iterations to reach the convergence point. However, since we could not use the same value for Batch approach we cannot really compare which approach is better. However, for both methods we could observe that learning rate equal to 0.1 works well and on average 10 iterations were required to reach the MSE at which we converge. We can conclude that in terms computation time, linear Batch gradient descent performs a lot better. However, it could not be the case in much larger data sets. Also, we could observe that iterations required for certain lambdas are very close if not equal on average. Therefore, we should use Batch gradient descent method as it is more efficient and does not have a negative effect of missing the converging point with minimum MSE.

Logistic Stochastic Gradient Descent

Now we consider the logistic gradient descent for which we will use convergence tolerance again equal to 0.0001. We will first examine effect of different learning rates. So we start with the following lambda values: 10, 1 and 0.1, and we iterate 50 times.

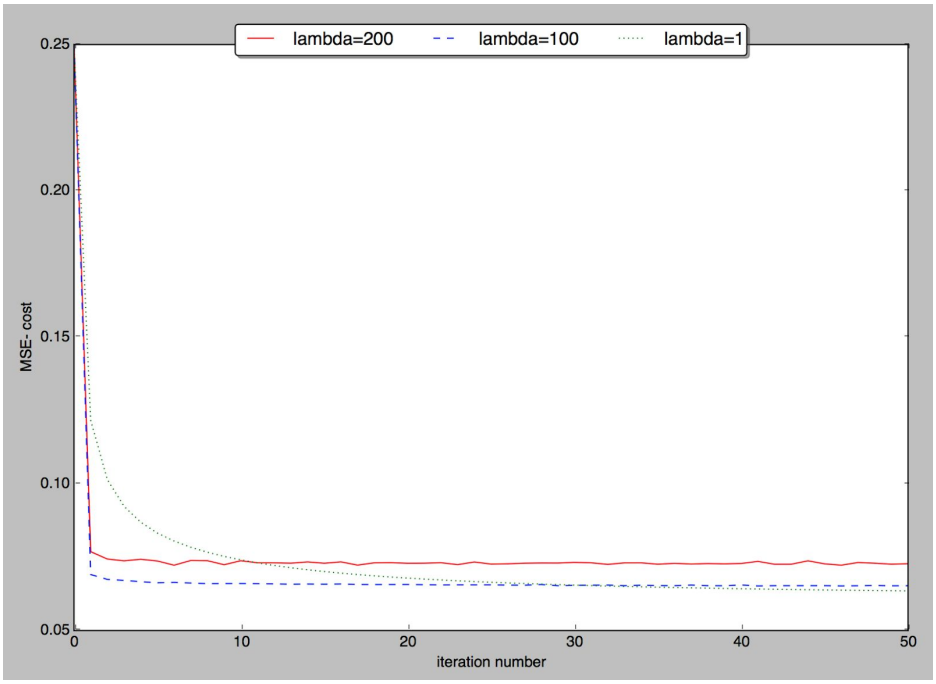


We can see that all values are converting and the fastest one is lambda=10. Since all the values seem to be converging we present AUC curve for minimum errors.



The result we got for this curve is impressing and AUC is 0.96726 which is a value very close to 1(perfect test). This means that our method has a excellent accuracy of the test so it very well separates the emails tested into those which are spam and not a spam.

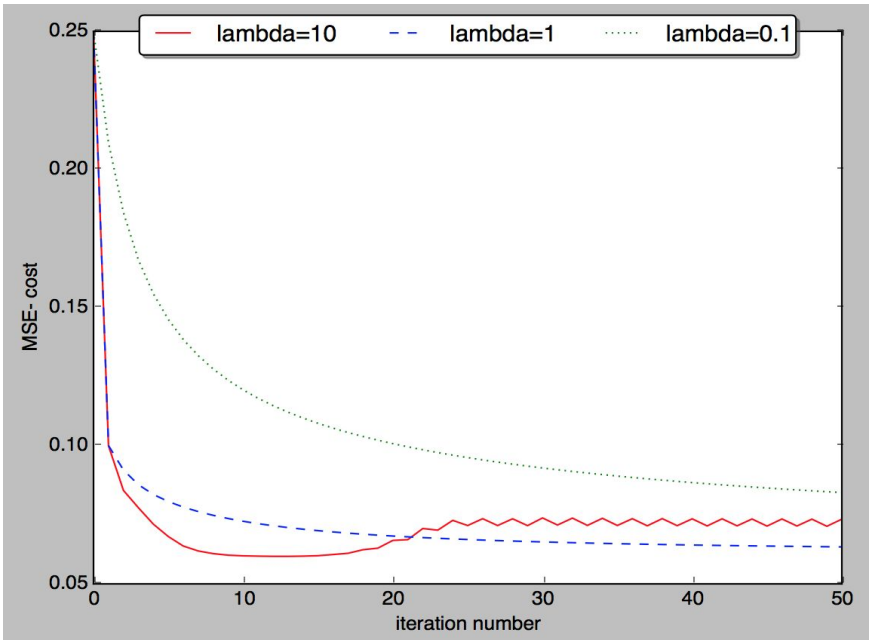
We will now test even higher lambda values 200, 100 and 1. Here is the plot we get:



It might not very clear but both lambdas equal to 100 and 200 are no converging so they are too high to consider. However, lambda=1 again seems to be fine and converges well after 40-50 iterations.

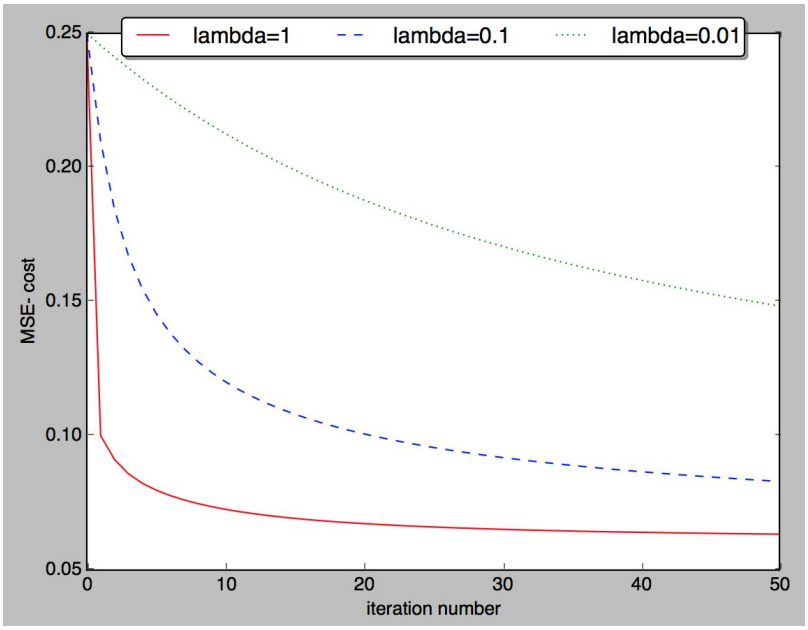
Logistic Batch Gradient Descent

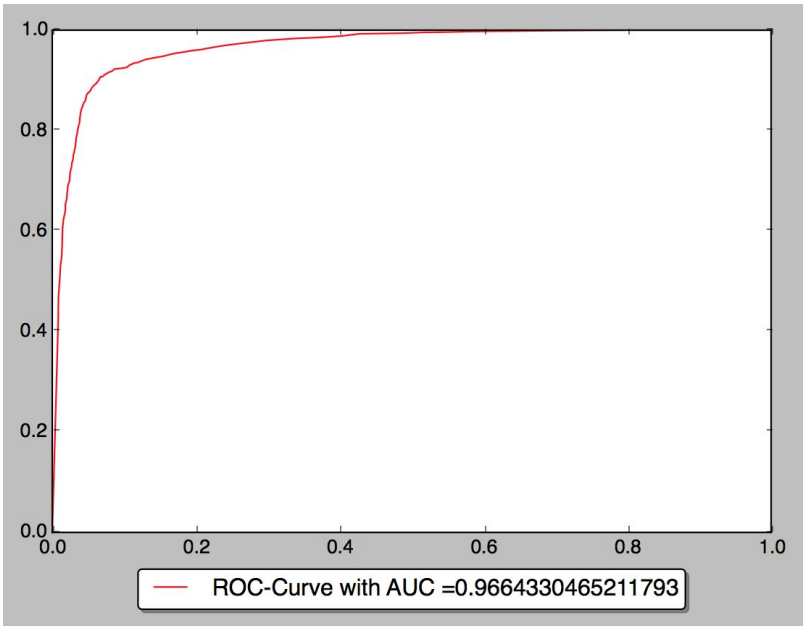
In order to compare Batch with Stochastic we need to check again whether the same lambda values are acceptable. So we use lambdas equal to 10, 1 and 0.1 over 50 iterations.



From the plot we can see that lambda =10 is again to high so to compare our methods I will use lambda=1, 0.1 and 0.01.

So let's see what we get for Batch approach using the above lambdas.



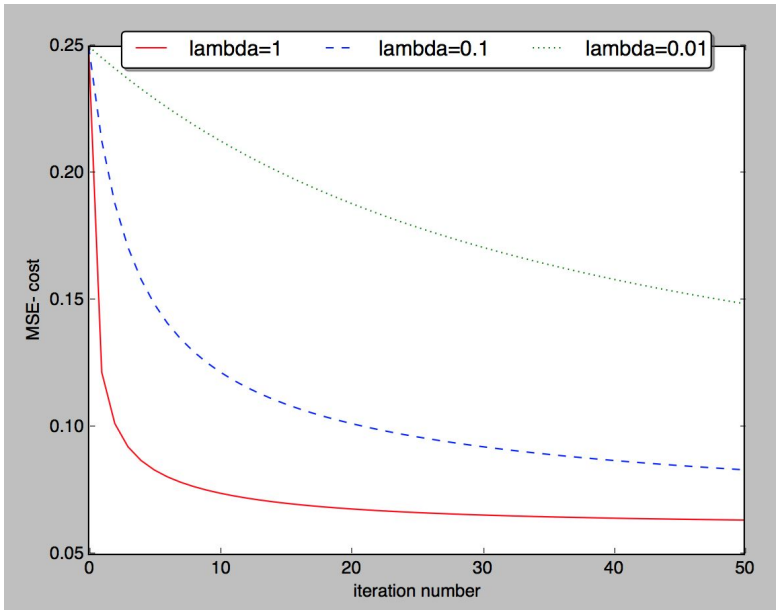


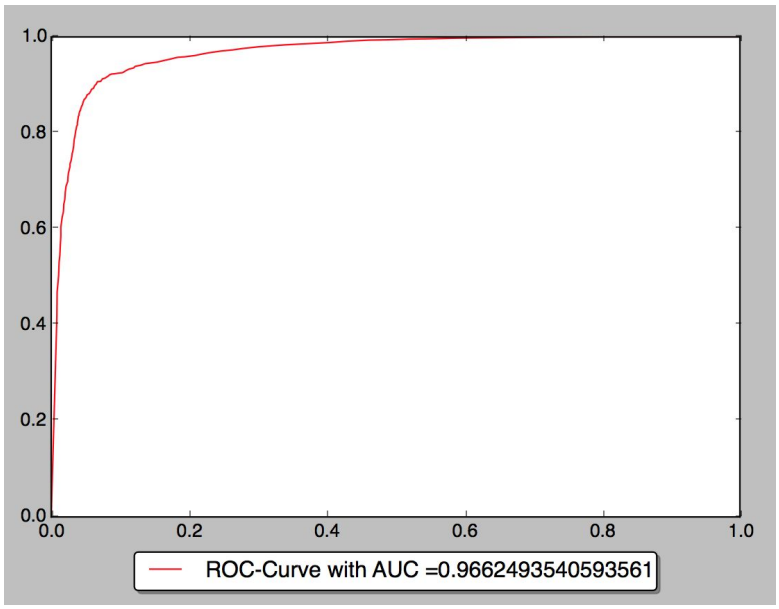
We have some good results are all values of lambda will eventually converge, but the one that seems to reach the minimum is lambda = 1 after 40-50 iterations. The AUC we getting for ROC-curve is also very good as it is close to 1(perfect test).

Logistic Stochastic Gradient Descent vs Logistic Batch Gradient Descent

In order to compare these linear methods I will use the above information and also provide an extra measure which is time taken to produce the results. For time measures I have used the same learning rates which are 1, 0.1 and 0.01. These lambdas were used because in both methods they converge and they are not too high.

Here is a plot for Stochastic approach over 50 iterations:





As we can see we do get converting rates, some after faster and some are slower but none of them are diverging.

Number of iterations:	Logistic Stochastic time(seconds):	Logistic Batch time(seconds):
50	166.97	117.47
100	320.19	239.33591
150	452.38	329.87
200	596.39	444.68

For the batch logistic method we can see that there is small difference in terms of convergence. We can see that stochastic gradient converges slightly quicker than batch method. Nevertheless, we reach the same mean squared error over the same number of iterations. Another difference is that stochastic method works well for higher learning rates, which is a problem for batch method. It is the case that we can spot in both logistic and linear gradient descent. Lastly, we can again see from the table above that stochastic approach takes a lot longer to converge therefore it is better to apply batch method.

Please note that for both linear and logistic stochastic gradient the graphs illustrate the number of iterations which is in fact the overall number of iterations. We also have that for each iteration there updates of theta when scanning through each feature in the data set. Therefore, internally in each iteration there are as many iterations as there are features in the data set. Such thing does not happen with batch methods therefore this explains the extra time taken to calculate mean squared errors.