Lab Course Machine Learning
Exercise 4
Krithika Murugesan (277537)

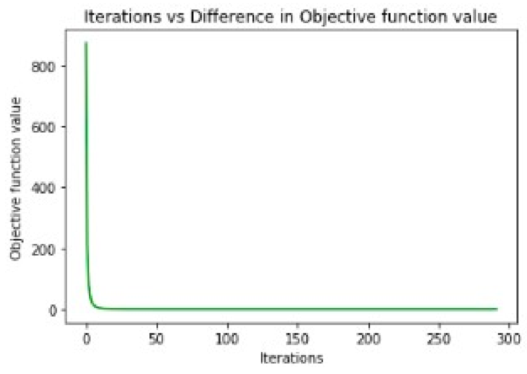
Classification Dataset : Bank Marketing

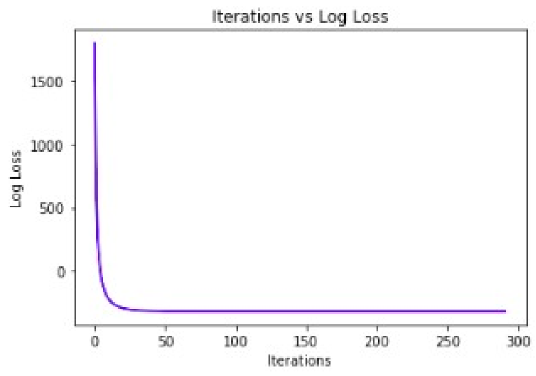
Considering the bank-additonal.csv, the output variable is a binary variable indicating if the client has subscribed a term deposit. There are twenty independent variables of which ten are categorical, they are converted into numerical values using hot-one encoding. Hot-one encoding is used as it does not add any additional bias to the variables and is sparse matrix thereby not contributing much to the computation.

There are no missing values in the dataset. The data is also normalized for quicker convergence. Then the data is split into test and training data. 'y_yes' is taken as the output variable.

Exercise 1 : Linear Classification with Stochastic Gradient Descend/Ascend

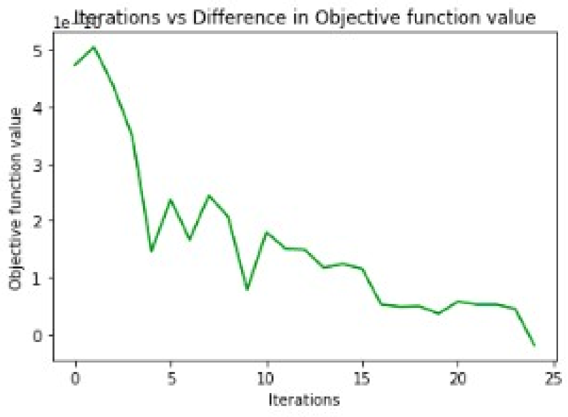
Starting with the gradient ascend, the function takes a considerable amount of time to converge. Following are the log likelihood and log loss plots against iterations for the same

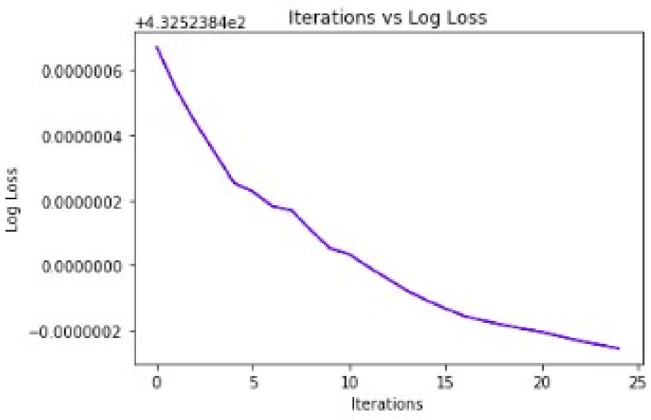




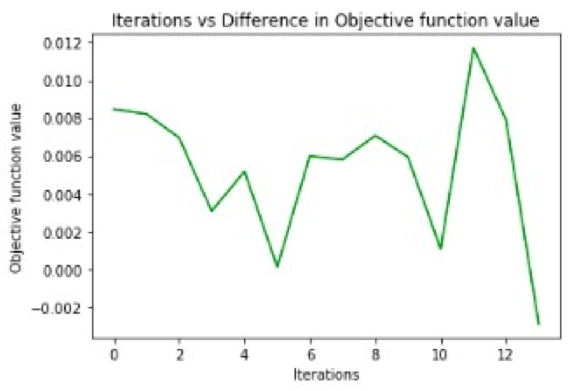
Implementing Stochastic Gradient Ascend for the bank, by considering the gradient at random points in the independent variable and updating the beta values the log likelihood converges at a faster rate. The step length is computed using two different methods **Bold driver** and **AdaGrad** respectively. Find below the graphs for the same where iterations vs Objective function difference is plotted for Training data and iterations vs negative log loss is plotted for Test data.

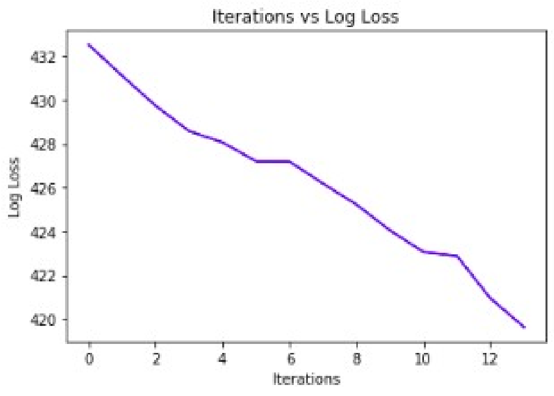
SGA : Bold driver : The log likelihood and log loss decreases after every step as the log likelihood is being maximized in ascend





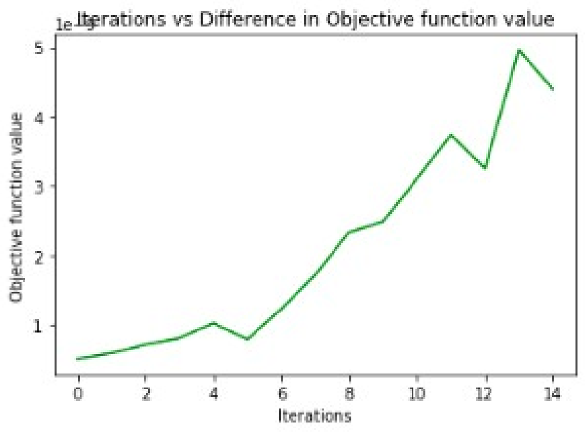
SGA - AdaGrad - It is similar to previous result but converges considerably faster and has many fluctuations as the gradient is taken for a random point.

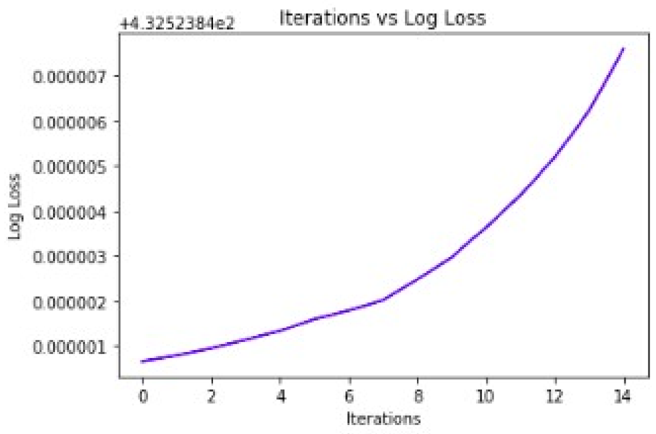




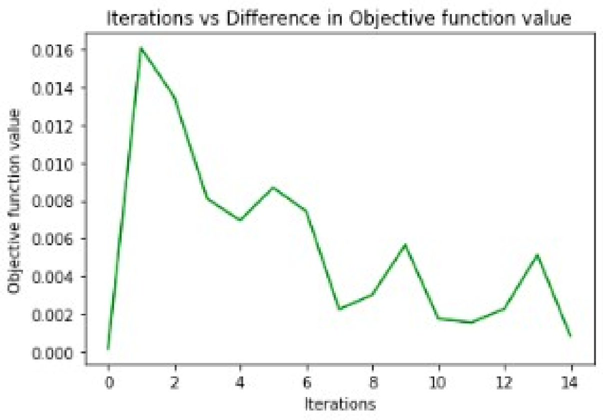
Applying Stochastic gradient descent for the same data by taking negative of gradient at random points in the dataset, the following are the results.

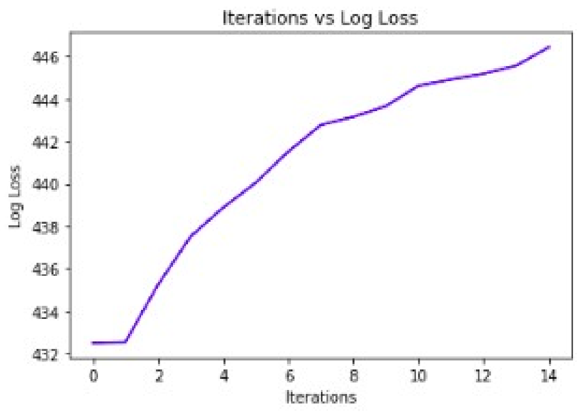
SGD - Bold driver - The log likelihood and log loss increases after every step as the log likelihood is being minimized in descend, as we are taking the negative slope





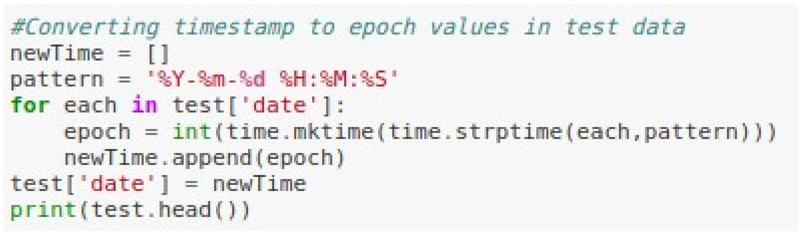
SGD - AdaGrad - It is similar to previous result but has many fluctuations as the gradient is taken for a random point. The loss increases in every iteration





Dataset 2 : Occupancy detection dataset

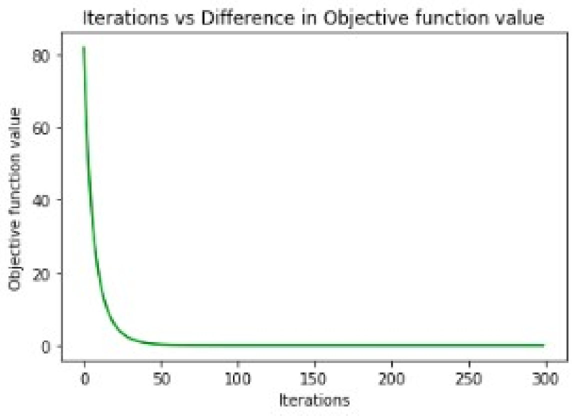
The occupancy has to be predicted using the parameters. Time stamp is a part of the dataset, it can be converted to epoch time which can be later normalized so the influence of time stamp is not removed from the regression function.

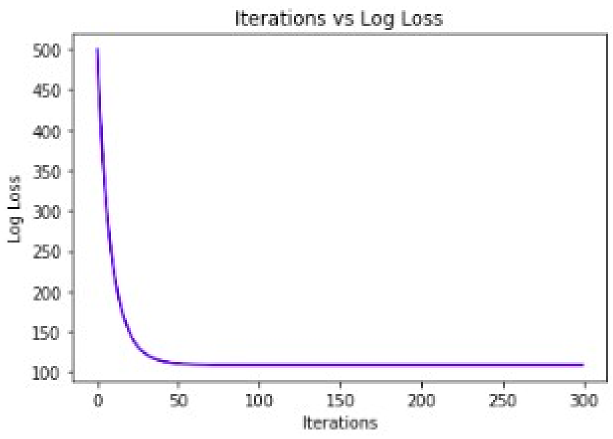


```
#Converting timestamp to epoch values in test data
newTime = []
pattern = '%Y-%m-%d %H:%M:%S'
for each in test['date']:
    epoch = int(time.mktime(time.strptime(each,pattern)))
    newTime.append(epoch)
test['date'] = newTime
print(test.head())
```

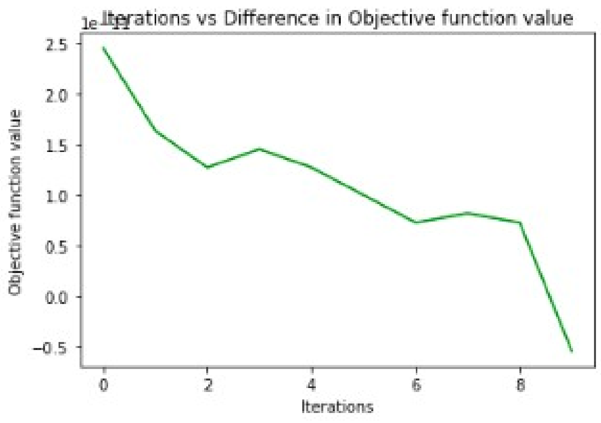
There are no categorical variables in the data, so encoding is not needed. Following are the graphs for the different step length learning algorithms in SGD and SGA

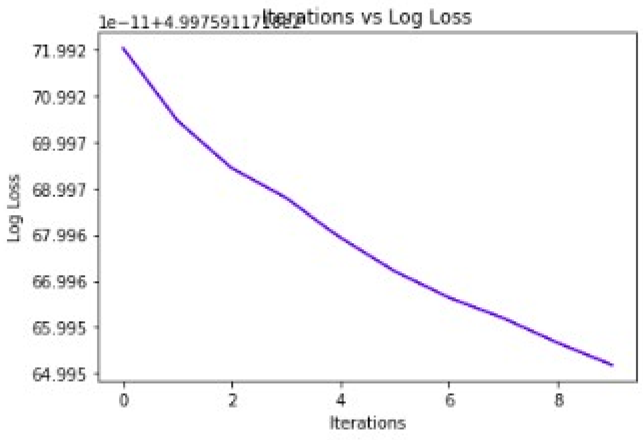
Gradient Ascent



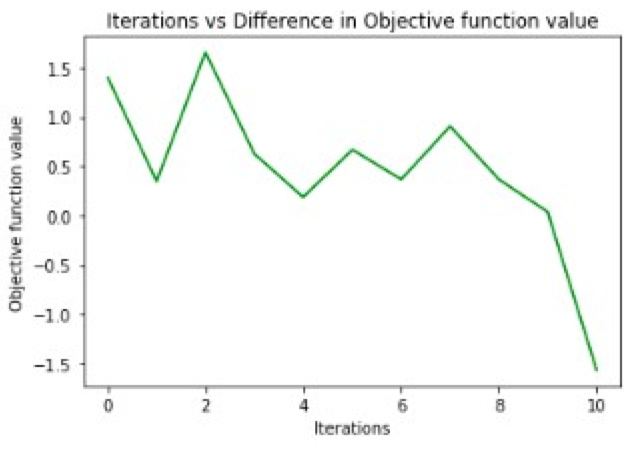


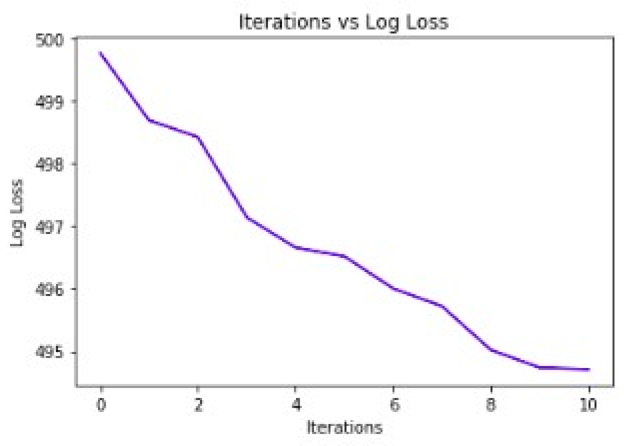
SGA - Bold driver



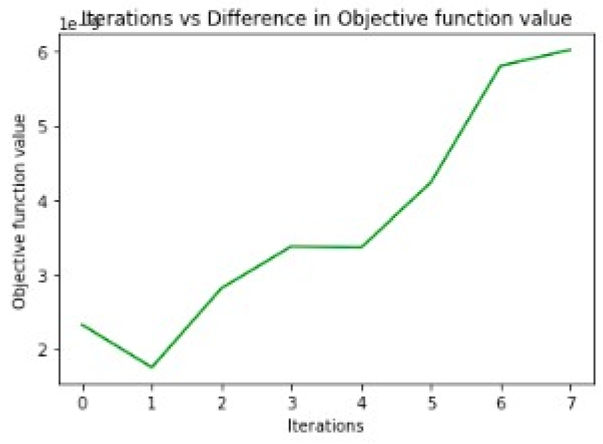


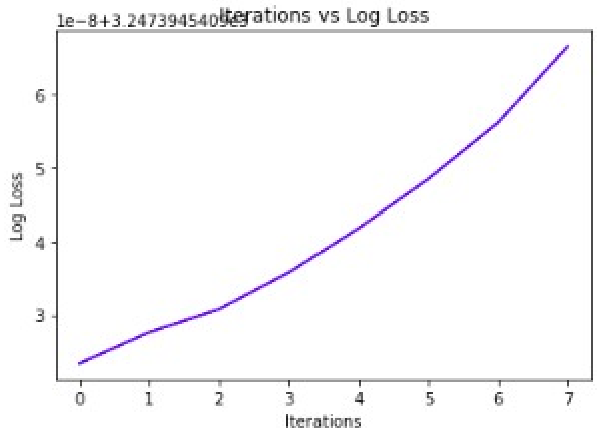
SGA - AdaGrad



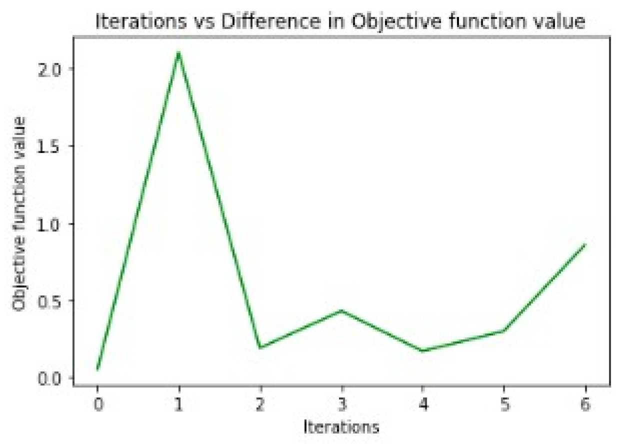


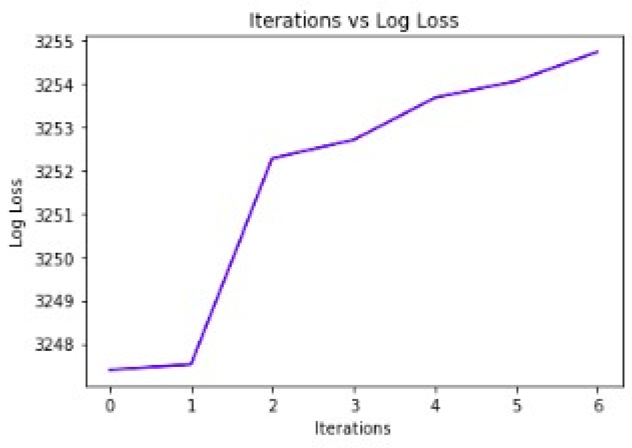
SGD - Bold driver





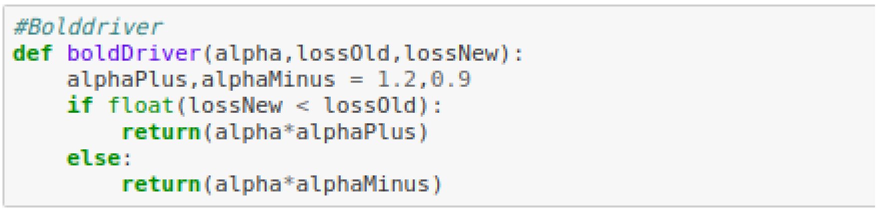
SGD - Adagrad



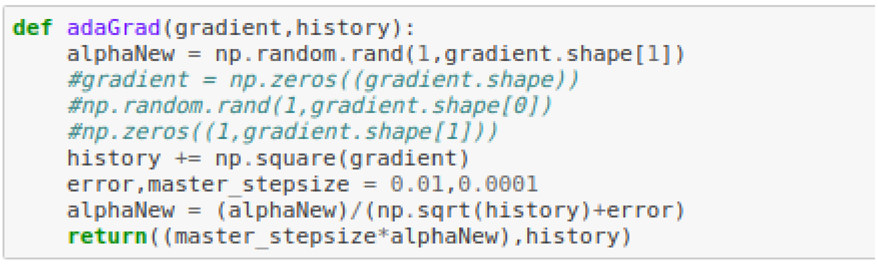


All the graphs are similar to that of the Bank dataset.

The code snippets used for bold driver and adagrad are



```
#Bolddriver
def boldDriver(alpha,lossOld,lossNew):
    alphaPlus,alphaMinus = 1.2,0.9
    if float(lossNew < lossOld):
        return(alpha*alphaPlus)
    else:
        return(alpha*alphaMinus)
```



```
def adaGrad(gradient,history):
    alphaNew = np.random.rand(1,gradient.shape[1])
    #gradient = np.zeros((gradient.shape))
    #np.random.rand(1,gradient.shape[0])
    #np.zeros((1,gradient.shape[1]))
    history += np.square(gradient)
    error,master_stepsize = 0.01,0.0001
    alphaNew = (alphaNew)/(np.sqrt(history)+error)
    return((master_stepsize*alphaNew),history)
```

Adagrad or adaptive gradient allows the learning rate to adapt based on the parameters, while bold driver learns a single beta for all the parameters. It performs larger updates for infrequent parameters and smaller updates for frequent ones. As it considers the gradient history there are many jitters in the graph for Adagrad.