Lab Course Machine Learning

Exercise Sheet 3

Krithika Murugesan(277537)

Dataset : Airfare data

This dataset has the data related to airlines fare, the independent variable is price and is a continuous variable, therefore linear regression can be used to predict the value.

Exercise 1 : Data Prepossessing

Reading the data from the HTML page and converting it to structured data

```
#Load the data using pandas
#Import all the required packages
import pandas as pd
import numpy as np
import io
import requests
import matplotlib.pyplot as plt

#Reading the content from the HTML page using requests package
url = "http://www.stat.ufl.edu/~winner/data/airq402.dat"
data = requests.get(url).content

#Splitting the data into rows and columns
cleanedData = data.decode('utf-8').split('\n')
rowsAndColumns = [each.split() for each in cleanedData]

#Adding a header to the data
header = ['city1','city2','avgFair','distance','avgPassengers','leadingAirline','marketShare','avgFare','lowPriceAi
rline','marketShare2','price']

#The final dataframe
df = pd.DataFrame(rowsAndColumns, columns = header)
df = df.loc[0:999,:]

#Converting numeric entries to float
numeric = ['avgFair','distance','avgPassengers','marketShare','avgFare','marketShare2','price']
for each in numeric:
    df[each] = df[each].astype('float')
print(df.describe())
```

1. To convert all the categorical variables to numerical values. One-hot encoding can be used, but the dataset has several different distinct classes and four categorical variables. This process will increase the matrix size largely and will increase the cost of computation largely. So integer encoding is used.
2.There are no NA values in the dataset
3.The data is split into test and train by using a split array assigned to true and false values randomly in the ratio given

Exercise 2 : Linear Regression with gradient descent

Least square loss and derivative are calculated for the function using the following functions

```
#3.Least square loss function
def leastSquareLoss(x,y,beta):
    yPredicted = x.dot(beta)
    leastSquareLoss = (np.square(y-yPredicted)).sum()
    return(leastSquareLoss)
```

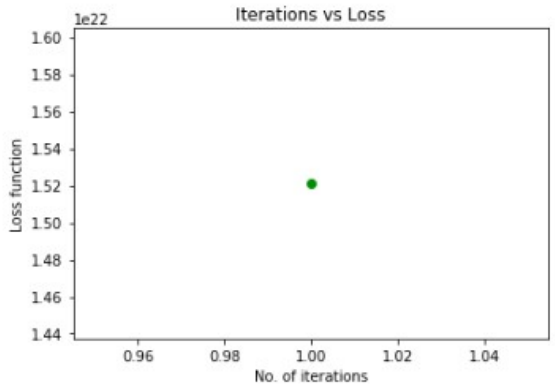
```
#Function for derivative
def derivative(x,y,beta):
    predicted = y - (x.dot(beta))
    return ((-2 * np.dot(x.T,predicted)))
```

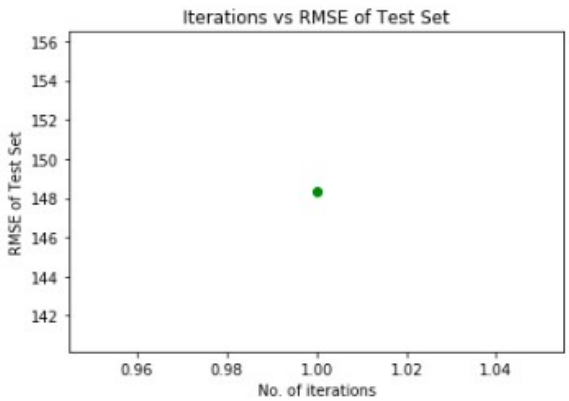
The **gradient descent** is implemented by the following function

```
# 4.learn-linregGD and minimize-GD algorithm
def gradientDescent(xTrain,yTrain,xTest,yTest,alpha):
    beta = np.zeros((xTrain.shape[1],1))
    print("Initial Beta : \n",beta)
    #Find Beta using minimize GD
    #Initializing list to plot the graphs
    loss = []
    ii = []
    rmse = []
    for i in range(1000):
        #Computing next beta
        betaNext = beta - (alpha*derivative(xTrain,yTrain,beta))
        betaLoss = leastSquareLoss(xTrain,yTrain,beta)
        betaNewLoss = leastSquareLoss(xTrain,yTrain,betaNext)
        lossValue = betaLoss - betaNewLoss
        #RMSE for test data
        eachRmse = leastSquareLoss(xTest,yTest,beta)
        #Appending the list with values for each loop
        loss.append(abs(lossValue))
        ii.append(i+1)
        rmse.append(np.sqrt(eachRmse/xTest.shape[0]))
        if float(lossValue) < 0.1:
            print("\nOptimal Beta found : \n",betaNext)
            print("No.of iterations: ",i+1)
            break
        if(i >= 1000):
            print("Gradient descent does not converge")
        beta = betaNext
    print("\nAlpha : ",alpha)
    plt.plot(ii,loss,'o',color = 'green')
    plt.xlabel("No. of iterations")
    plt.ylabel("Loss function")
    plt.title("Iterations vs Loss")
    plt.show()
    plt.plot(ii,rmse,'o',color = 'green')
    plt.xlabel("No. of iterations")
    plt.ylabel("RMSE of Test Set")
    plt.title("Iterations vs RMSE of Test Set")
    plt.show()
```

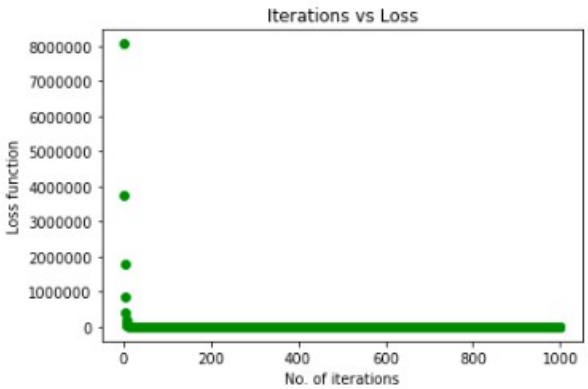
Plotting the iterations vs loss function value, for bigger alphas the loss difference is negative, so gradient descent does not function properly and comes out of the loop in one iteration and the function does not converge. For smaller alphas, the gradient descent approaches the minimum of the loss function, for very small alpha it takes long to converge. The following are the graphs for different alphas

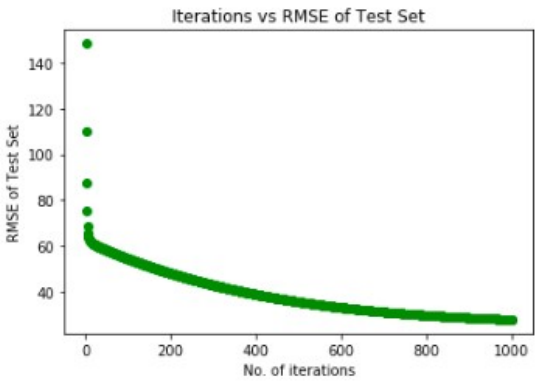
```
Alpha :  0.01
```



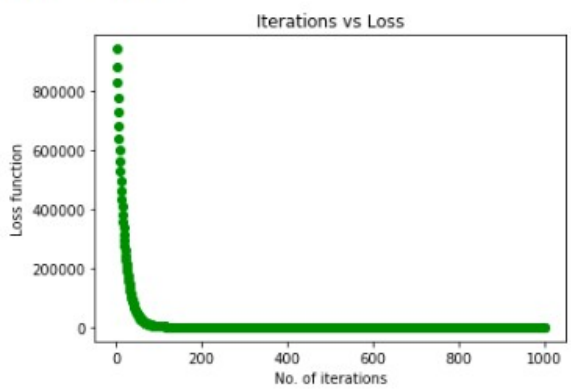



```
Alpha :  1e-10
```





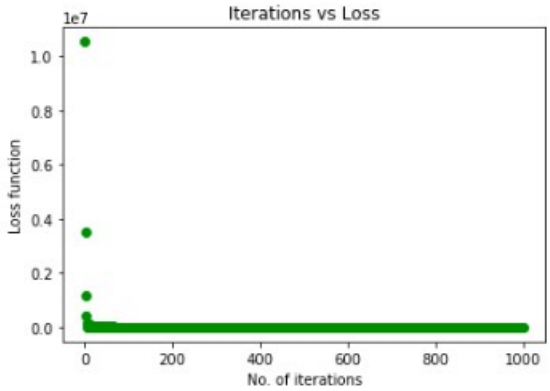
```
Alpha :  1e-11
```

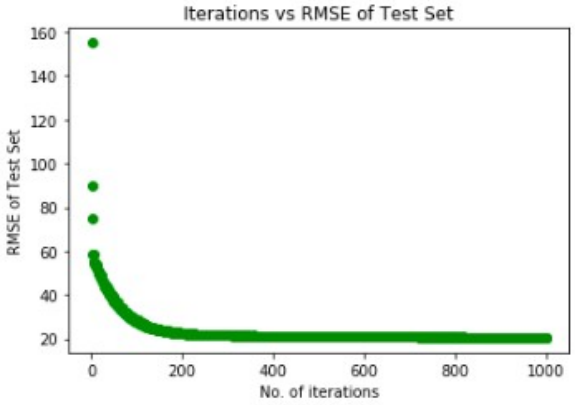


Using Armijo method to identify the learning rate, the results got are as follows,

```
while flag == False:
    #Computing condition
    condition = (alpha1*sigma)*((derivative(xTrain,yTrain,beta).T).dot(derivative(xTrain,yTrain,beta)))
    #New value to be passed to loss function
    betaNew = beta - (alpha1 * (derivative(xTrain,yTrain,beta)))
    if float(leastSquareLoss(xTrain,yTrain,beta)-leastSquareLoss(xTrain,yTrain,betaNew)) < condition:
        #Modifying alpha
        alpha1 = alpha1/2
    else:
        #Setting the flag to True to come out of the loop
        flag = True
        alpha = alpha1
```

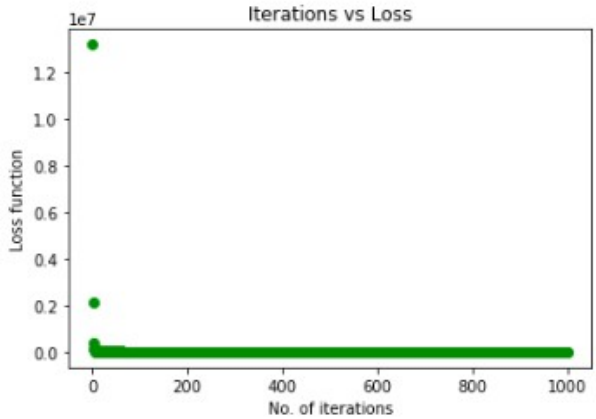
```
Alpha :  9.313225746154785e-10
```

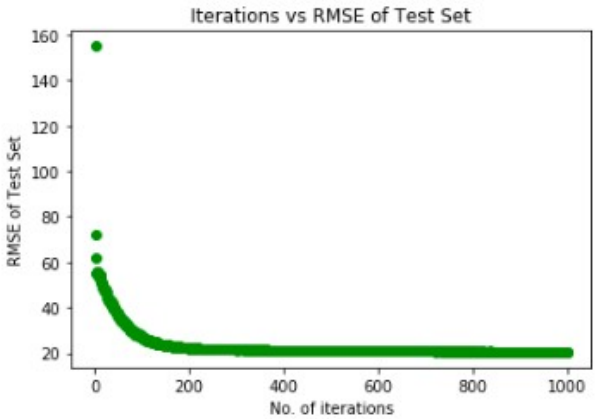




Using bold driver for the same data, the results are

```
Alpha :  8.195638656616212e-10
```





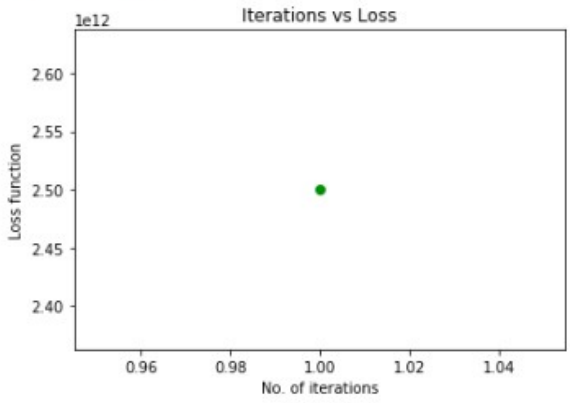
The alpha learnt by armijo and bold driver are almost similar, both the loss function and the rmse keep decreasing as the number of iterations increase, which implies the function is converging. But the number of iterations for the function is converge is large this can be attributed to the huge difference in the scale. By normalizing the data using min-max normalization or z normalization the beta convergence will increase as there is no change iarmbalanced change in weights of dependent variables as in the normal gradient descent.

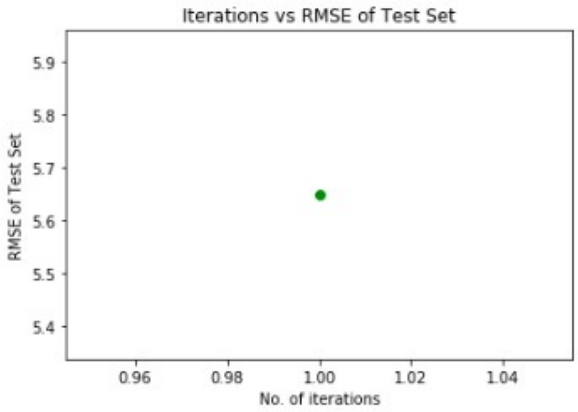
Dataset - Wine Quality

This data set has two different sets of data, one for white wine and one for red wine. In both the cases the independent variable is a categorical variable. Using the given loss function only linear regression can be done. The predicted values are real numbers unlike the independent variable which is an indicator of nominal variable. This may lead to an increase in the losses.

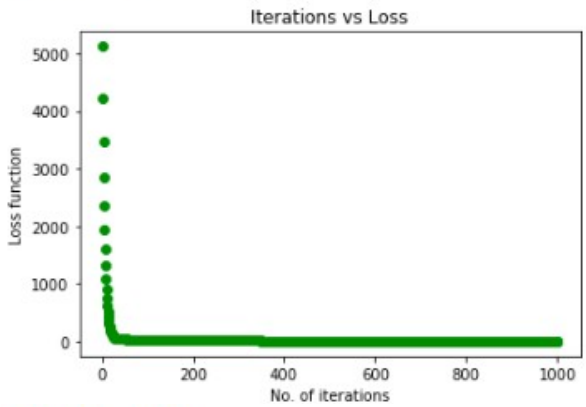
Performing the same pre-processing as Airfare data and using integer encoding to avoid huge computation cost with hot-one encoding involved. There are no NA values in both the datasets. Following are the iteration vs loss graphs for red wine dataset,

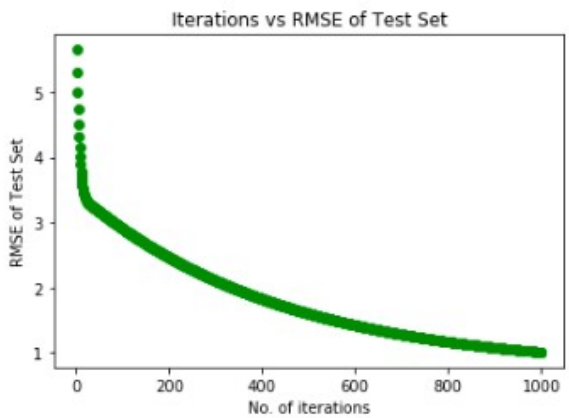
```
Alpha :  0.001
```



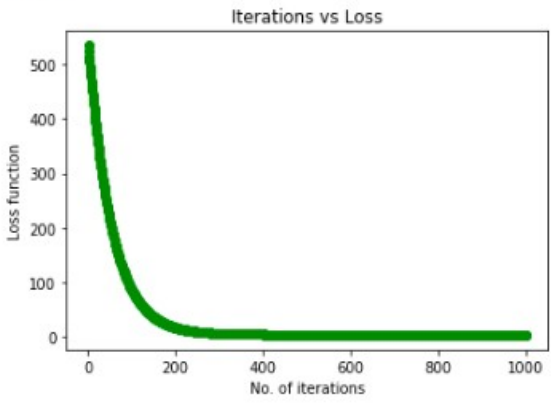


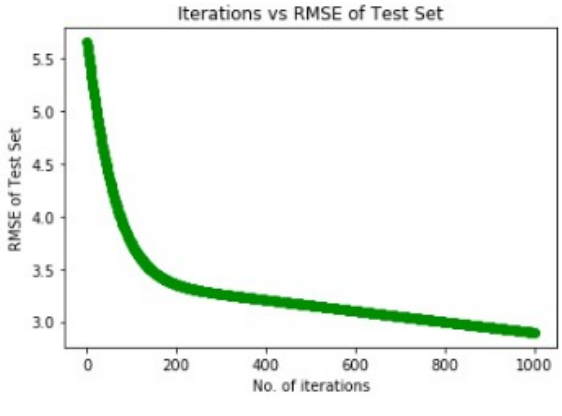

```
Alpha :  1e-08
```





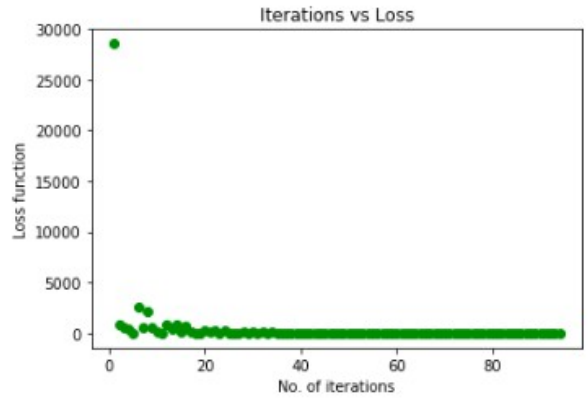

```
Alpha :  1e-09
```

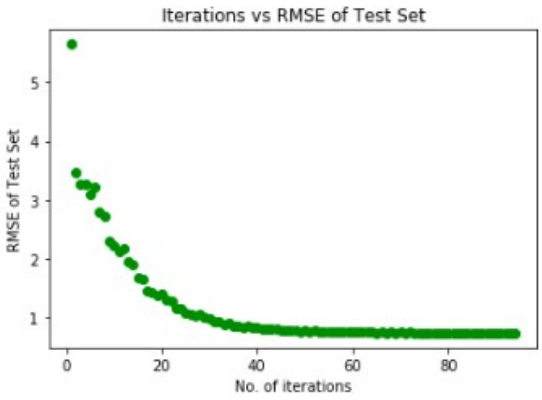




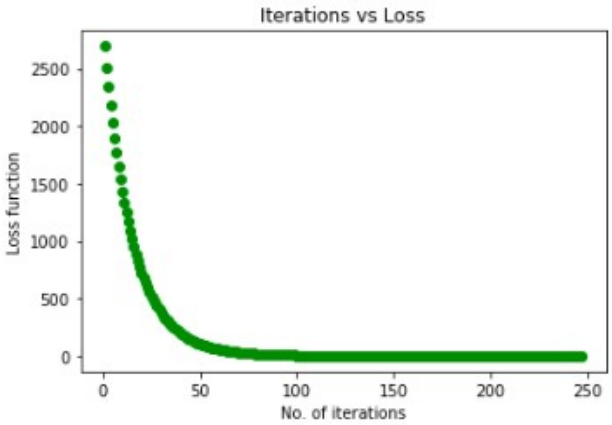
Armijo step length - Gradient Descent

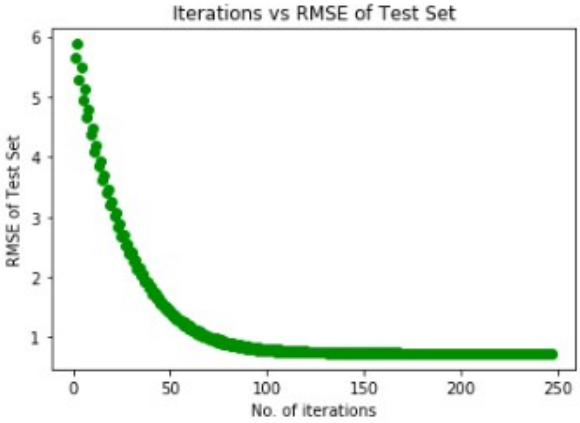
```
Alpha :  2.384185791015625e-07
```





Bold driver - Gradient descent

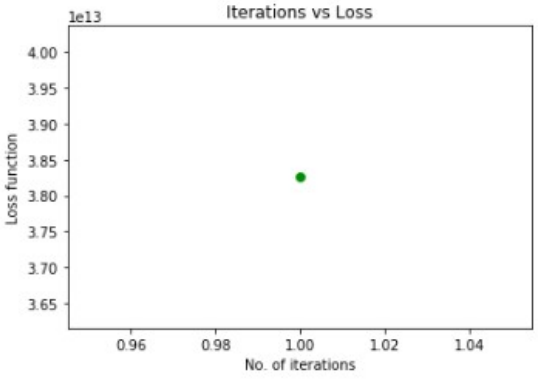


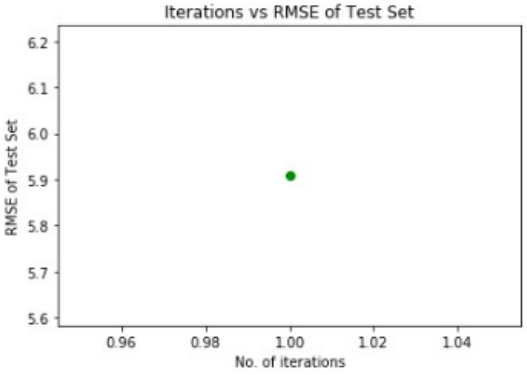


White Wine dataset

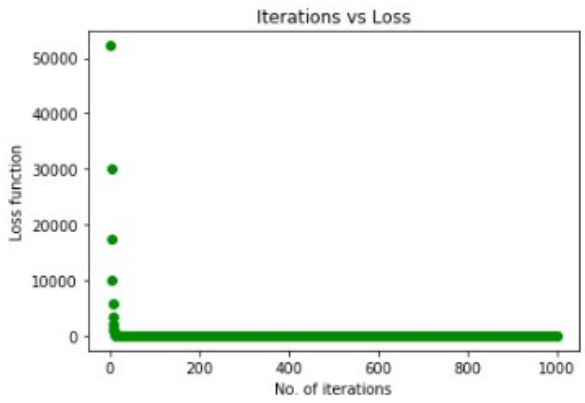
Gradient descent for three alpha values

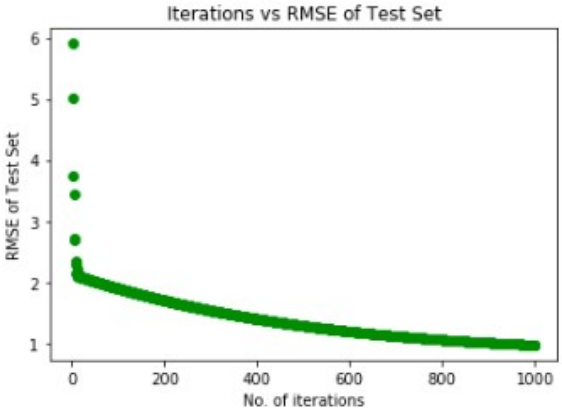
```
Alpha :  0.0001
```



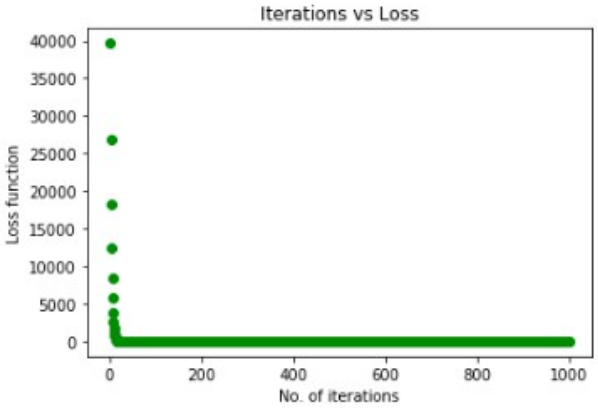


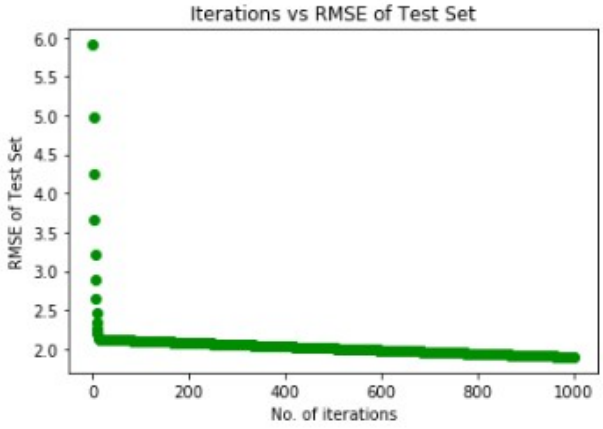
Alpha : 1e-08



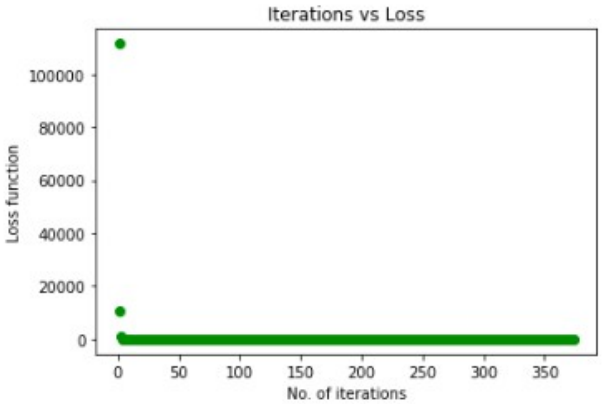


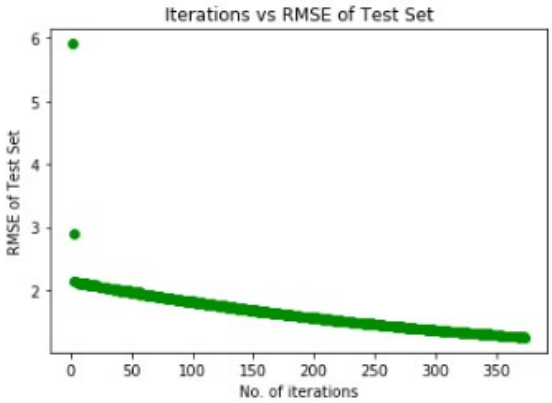
Alpha : 1e-09





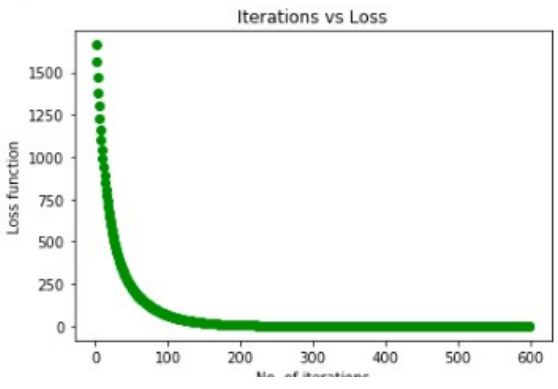
Armijo - Gradient Descent

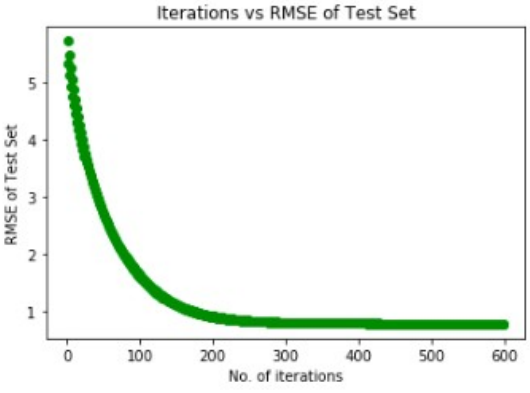




Bold driver - Gradient Descent

Alpha : 4.1961669921875006e-07





As seen in the airfare data, the loss function does not converge for huge alphas and for very small alpha values it takes long to converge. The RMSE value is also similar to loss function for all the methods of descent.