Lab Course Machine Learning
Exercise Sheet 6

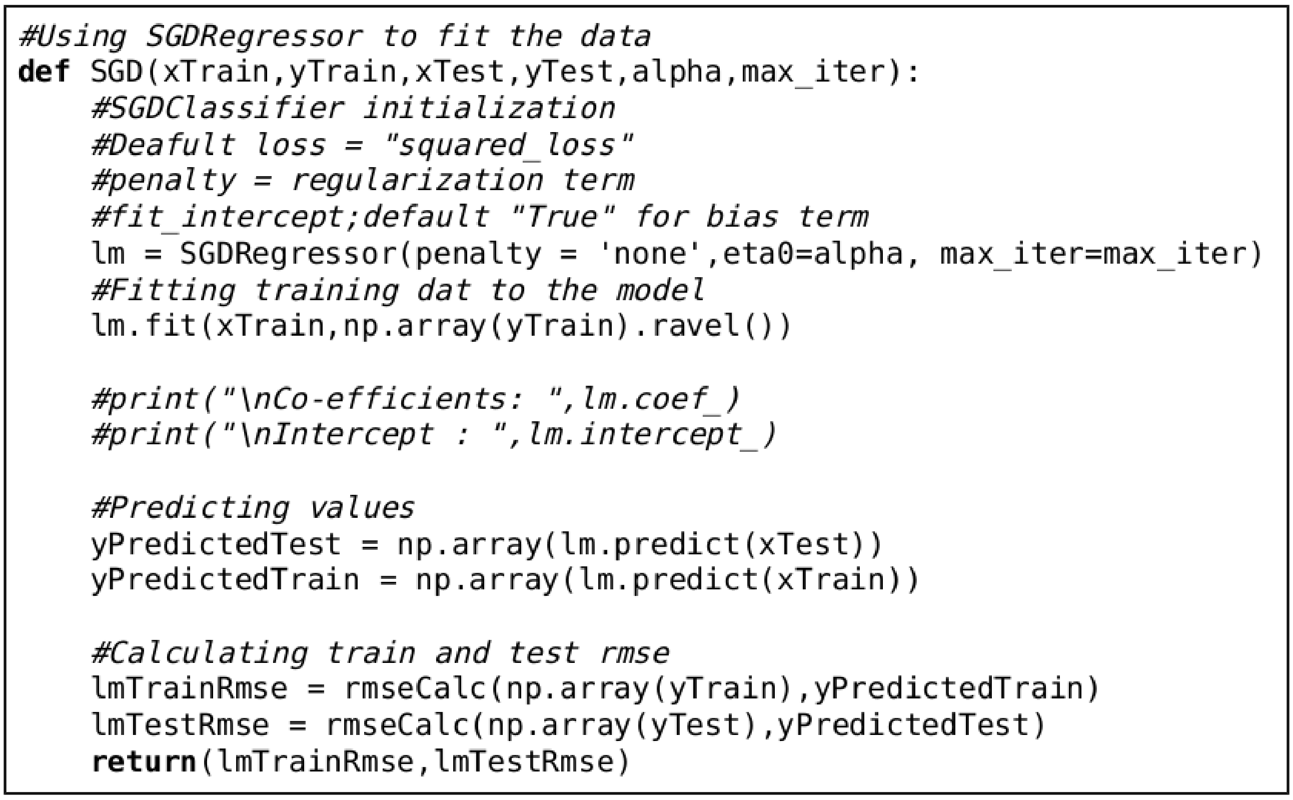
Krithika Murugesan(277537)

Exercise 1: Generalized Linear Models with Scikit Learn

Given is the wine dataset, the Quality is the dependent variable while the others are the dependent variables. The dataset has no null values. It is split into train and test data in the ratio 4:1 and used for the various learning algorithms.

The sklearn package function SGDRegressor() is used to compute Stochastic gradient descent for linear regression. The default loss function used is squared loss, the parameter penalty represents the regularization to be used and defaults to none. The bias term can be considered or not using the fit_intercept parameter of the function.

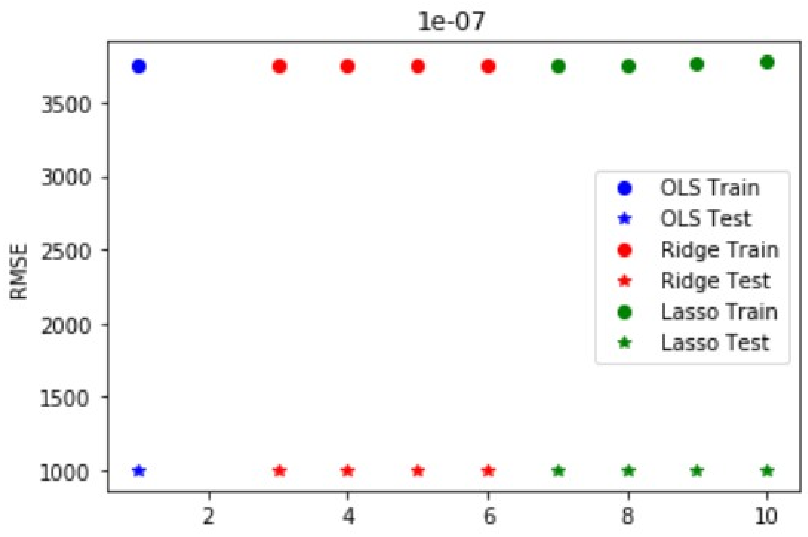
The function is initialized with the required parameters and later the model is fit to the training data. The predict function can be used to predict the results on test data and the RMSE is calculated.



```python
#Using SGDRegressor to fit the data
def SGD(xTrain,yTrain,xTest,yTest,alpha,max_iter):
    #SGDClassifier initialization
    #Deafult loss = "squared_loss"
    #penalty = regularization term
    #fit_intercept;default "True" for bias term
    lm = SGDRegressor(penalty = 'none',eta0=alpha, max_iter=max_iter)
    #Fitting training dat to the model
    lm.fit(xTrain,np.array(yTrain).ravel())

    #print("\nCo-efficients: ",lm.coef_)
    #print("\nIntercept : ",lm.intercept_)

    #Predicting values
    yPredictedTest = np.array(lm.predict(xTest))
    yPredictedTrain = np.array(lm.predict(xTrain))

    #Calculating train and test rmse
    lmTrainRmse = rmseCalc(np.array(yTrain),yPredictedTrain)
    lmTestRmse = rmseCalc(np.array(yTest),yPredictedTest)
    return(lmTrainRmse,lmTestRmse)
```

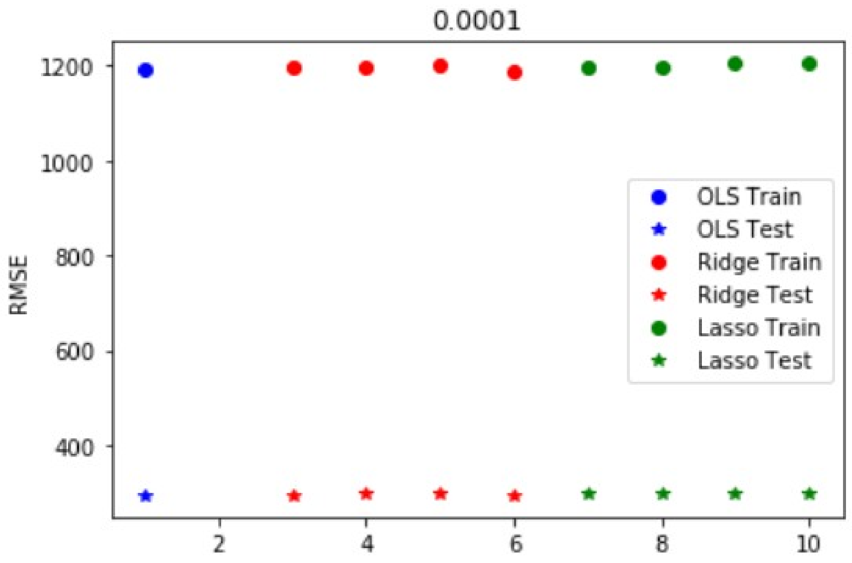
For Ridge and Lasso regression, the penalty term is added to the same function. Also, the regularization co-efficient can be passed to the function. The RMSE values obtained for the plot are as follows.

1. For learning rate = 1e-07



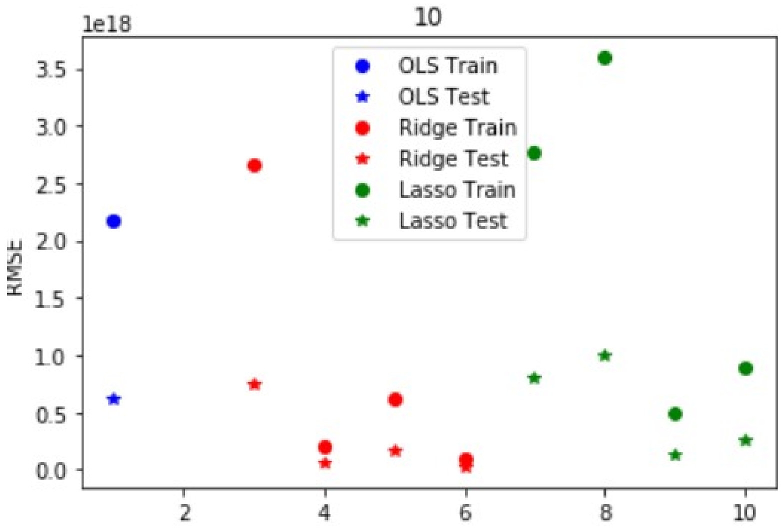
It can be seen that since the alpha is very small, the convergence of the gradient descent is very slow leading to higher values of RMSE. The descent has to be computed for several more iterations to reach lesser error also, this is observed across all models. Regularization is done to avoid over fitting of data where the model is just a replicate of the training data and is not generalized. The four points in the graph for Ridge and Lasso regression are for regularization co-efficients [0.00000001, 0.0001, 0.1, 1] respectively.

2. Alpha = 0.0001



The RMSE values are comparatively less for this alpha, the order of regularization co-efficients is the same as the previous example [0.00000001, 0.0001, 0.1, 1]. IT can be observed that the RMSE is still less for the test data, showing that the model is more generalized.

3.Alpha = 10



The learning rate is very high and considered only for understanding the models better. It can be seen that as the regularization co-efficient values increase the training error is very close to test error showing a mere under-fitting due to the high values. The gap between test and train RMSE is lesser for Ridge regression than Lasso regression.

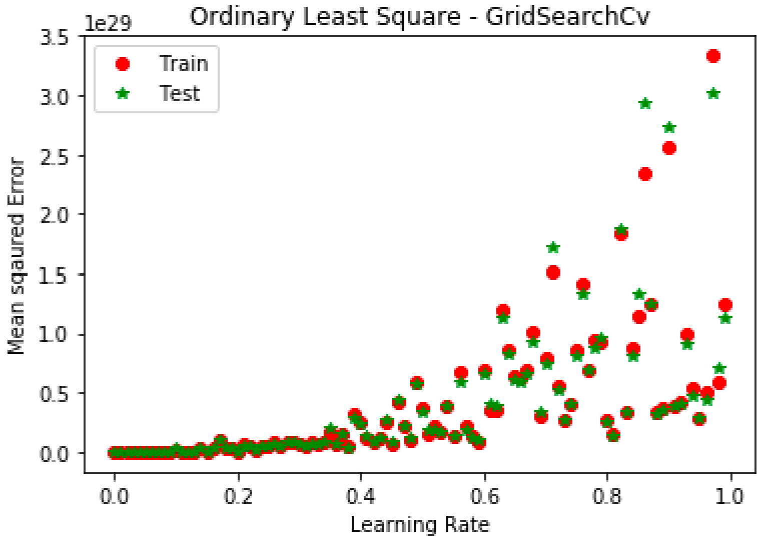
Ridge only shrinks the weights to the parameters, while Lasso zeros less significant features, hence doing both shrinkage and variable selection at the same time.

Cross Validation

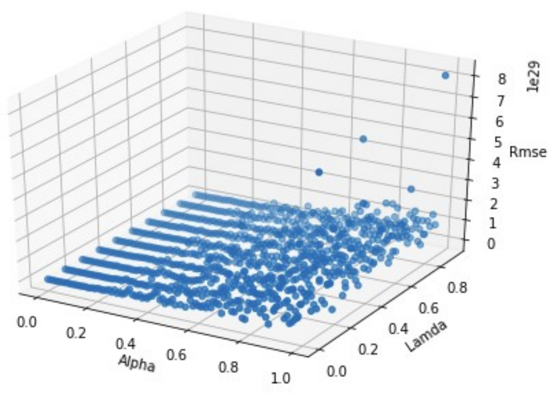
The hyper parameter such as initial alpha and regularization co-efficients cannot be computed using gradient descent, for which cross validation is used. The model is learnt with an array of hyper parameters. Then the values with the least error are selected, if they are validated on the test data the generalization will be lost. A separate part of the training data can be used as validation set to compute this, but it will take away entries from training set leading to under- fitting of the data. So cross-validation is used, the train data is split into k folds and for k iterations the kth split is used as test data and the parameters are validated.

This can be achieved easily in python using the GridSearchCv function, it takes in the model, param_grid(dictionary of the hyper parameter values to be evaluated), cv(number of folds) and the scoring method (‘mean squared’) are passed as parameters.

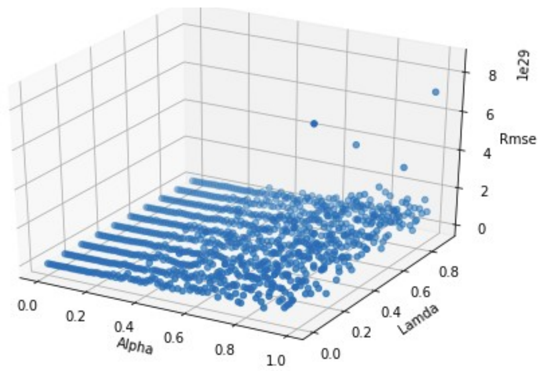
1. OLS Regression



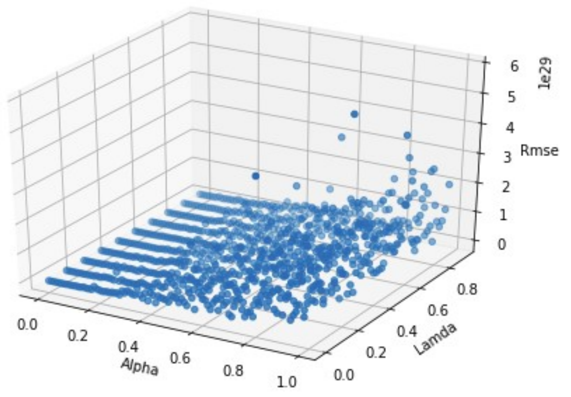
2. Ridge Regression(Train)



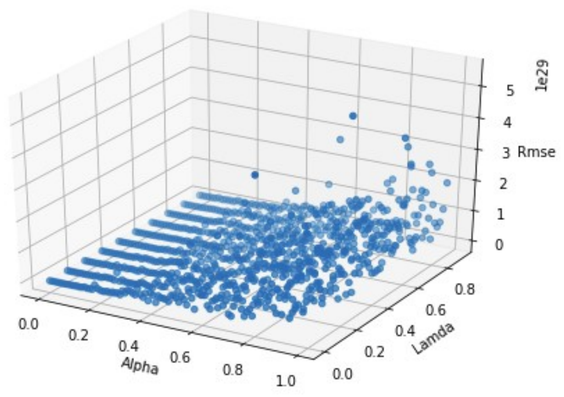
Ridge Regression(Test)



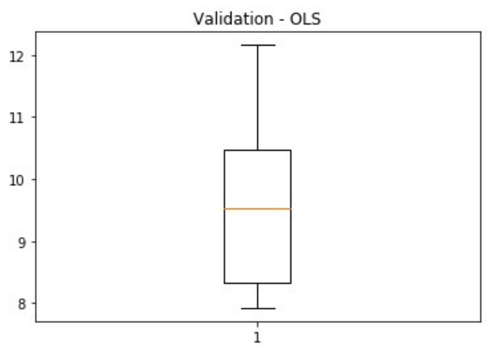
3. Lasso Regression(Train)

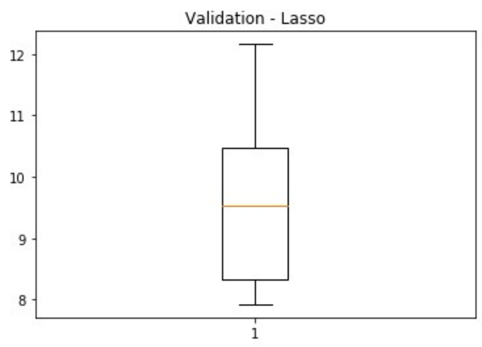


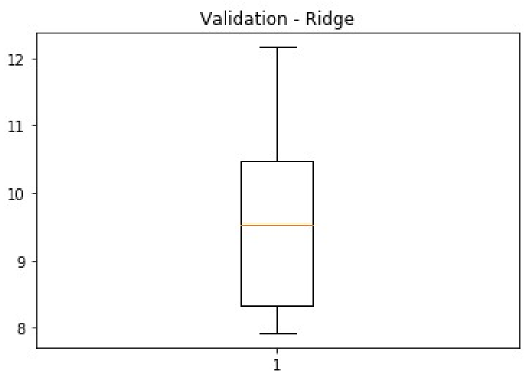
Lasso Regression(Test)



The cross validation scores are validated using cross_val_score function for the optimal values got from using the best_params_ function. And plotting the box plot for the same yields the following







The range for Lasso values are slighter on the larger side than the other two methods, while the mean is comparatively higher for Ridge regression. There are no outliers for any of the three models.

Exercise 2: Polynomial Regression

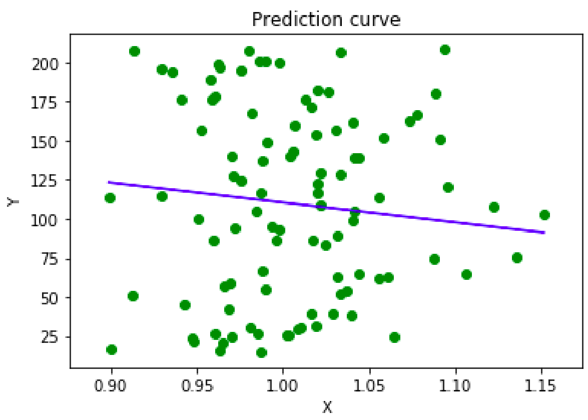
The dataset is generated using the following code snippet.

```
#TASK A
#Importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge

#Create x and y
x = np.random.normal(1,0.05,100)
x = x.reshape(-1,1)

eps = np.random.randint(200,size = 100)
eps = eps.reshape(-1,1)
y = ((1.3*x*x)+(4.8*x)+8)+eps
```

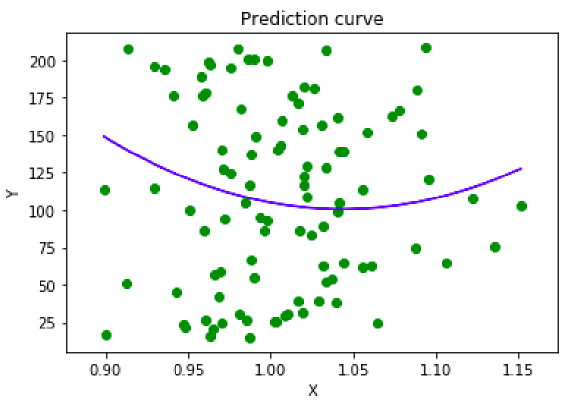
TASK A

The PolynomialFeatures function is used to generate features of higher degree, this is later used in the model to fit the training data. The same transformation is done on test data as well to predict the data. The following are the graphs for different degrees of polynomial.

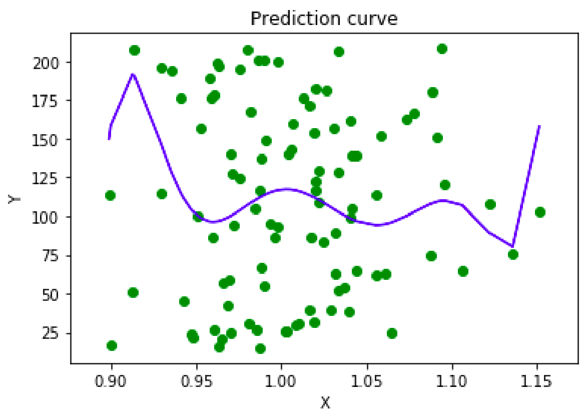
```
Polynomial degree :  1
```



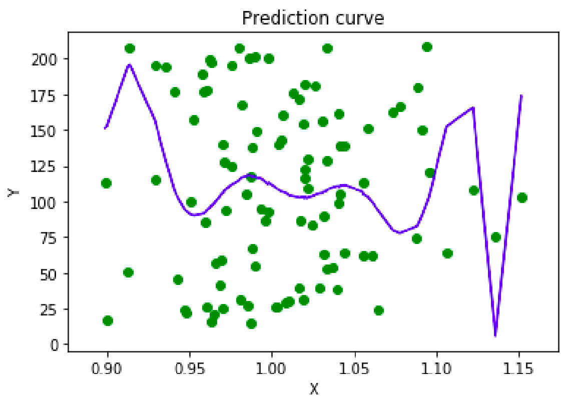
```
Polynomial degree :  2
```



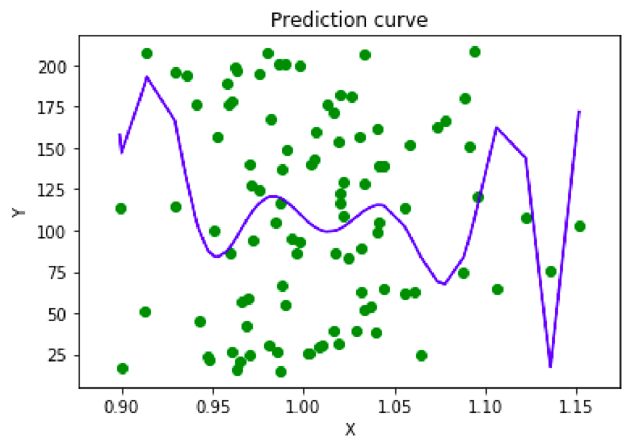
```
Polynomial degree :  7
```



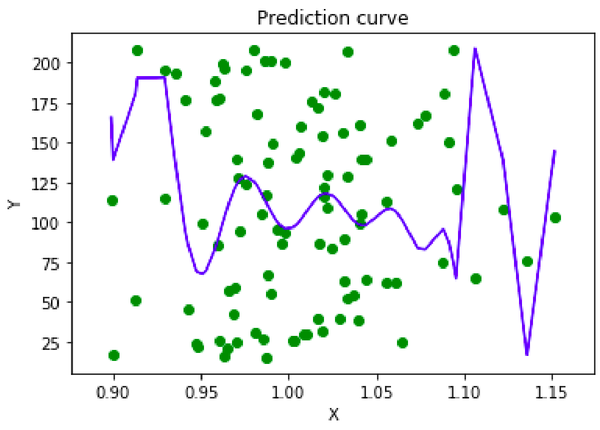
```
Polynomial degree :  10
```



```
Polynomial degree :  16
```



```
Polynomial degree :  100
```

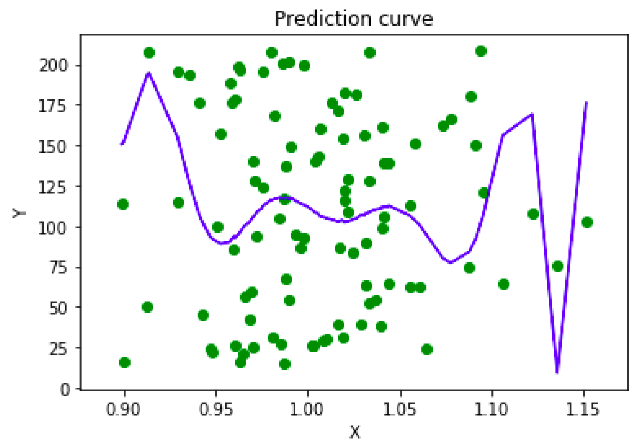


It can be seen that as the degree of the polynomial increases, the model fits the data better i.e., the prediction curve passes through more number of points as the degree increases. This may lead to overfitting with higher polynomial degrees as the model will fail to generalize.

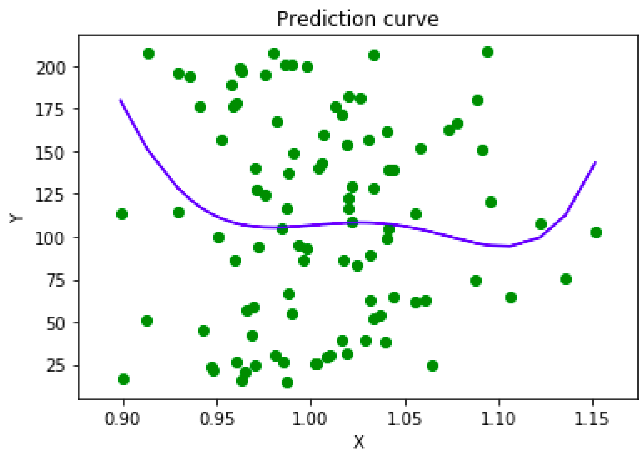
TASK B

When Ridge regression is done on the polynomial of degree 10 for different Alpha's, following are the observations

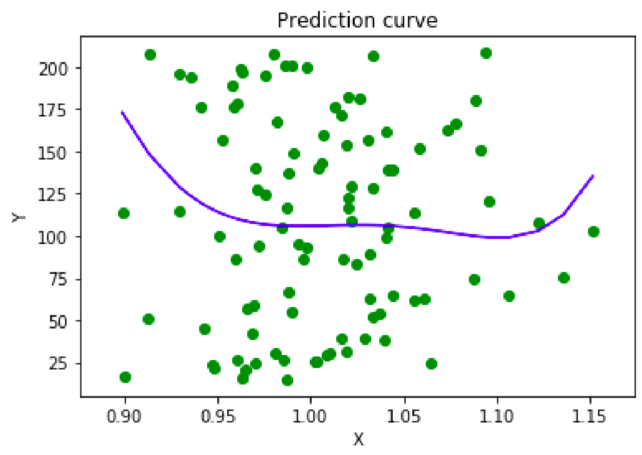
```
Alpha :  0
```



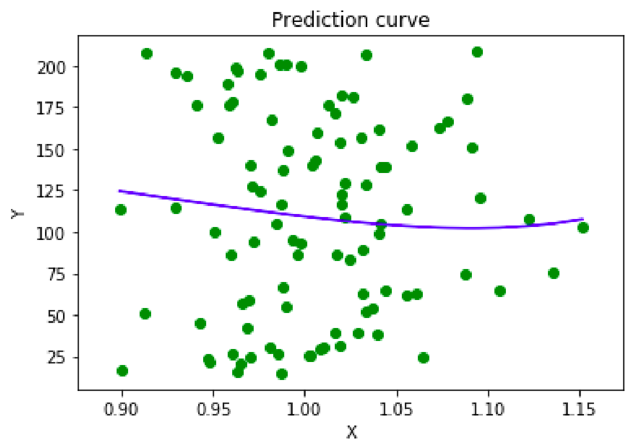
```
Alpha :  1e-11
```



```
Alpha :  1e-06
```



```
Alpha :  1
```



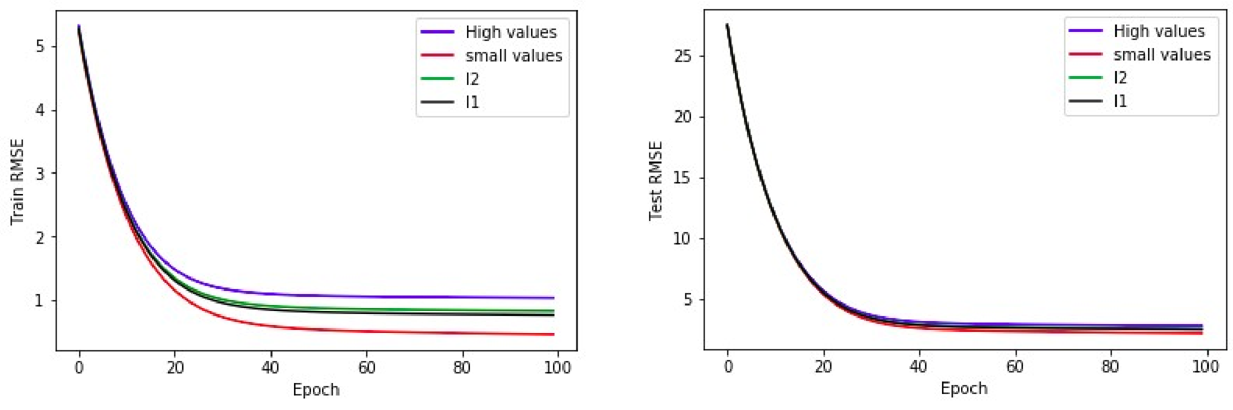
It can be seen that for alpha equal to zero, the model is a replica of the normal Linear Regression model. As alpha increases the fit reduces, giving the maximum value 1 for alpha under-fitting can be observed as the curve does not include many points, this is due to the high penalty imposed on the loss function due to the high value of alpha used. The other two alphas are almost similar in their behavior.

Bonus: Implement Elastic Net using Stochastic Gradient Descent

Elastic net is a linear combination of both l1 and l2 regularization. It combines the pro's of the two models and overcomes the cons. The loss function includes both l1 and l2 normalized terms.

```
#Loss function
def loss(x,y,beta,r1,r2):
    yPredicted = np.array(x.dot(beta))
    l2 = 2*r1*((beta.T.dot(beta)).sum())
    l1 = r2*(np.abs(beta).sum())
    leastSquareLoss = (np.square(y-yPredicted)).sum()
    return(leastSquareLoss+l1+l2)
```

On performing the elastic net search using stochastic gradient, the following is the result.

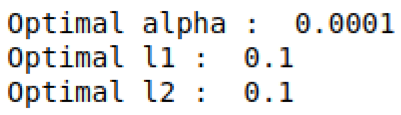


High values of l1 and l2 take longer to converge while small values converge sooner. The Lasso normalization is comparatively faster and more efficient than Ridge in this example as the dataset is small.

Defining a grid for alpha,l1 and l2 and performing cross validation on the data gives the optimal value of hyper-parameters.

```
#CrossValidation
def cv(trainCvalid,alphaGrid,l1,l2,totalFolds,batchsize,epoch):
    alpha,l1List,l2List,rmseCv = [],[],[],[]
    for eachAlpha in alphaGrid:
        for eachL1 in l1:
            for eachL2 in l2:
                alpha.append(eachAlpha)
                l1List.append(eachL1)
                l2List.append(eachL2)
                eachFoldRmse = []
                for each in range(1,totalFolds+1):
                    trainC = trainCvalid.copy()
                    trainCv,testCv = cvTestTrain(trainC,each,totalFolds)
                    xTest = testRed.loc[:,testCv.columns != 'quality']
                    yTest = testRed.loc[:,testCv.columns == 'quality']
                    ii,rmse,rmseTest = mini_BGD(trainCv,beta,xTest,yTest,eachAlpha,eachL2,eachL1,batchsize,epoch)
                    eachFoldRmse.append(np.mean(rmseTest))
                temp = (np.mean(eachFoldRmse))
                rmseCv.append(float(temp))
    return(alpha,l1List,l2List,rmseCv)
```

The combination of values for hyper parameters which give the least value are



```
Optimal alpha :  0.0001
Optimal l1 :  0.1
Optimal l2 :  0.1
```