Week 3 Assignment
Submitted by : Vaibhav Gusain

i) a) The figures below show the input data visualized by different angles.

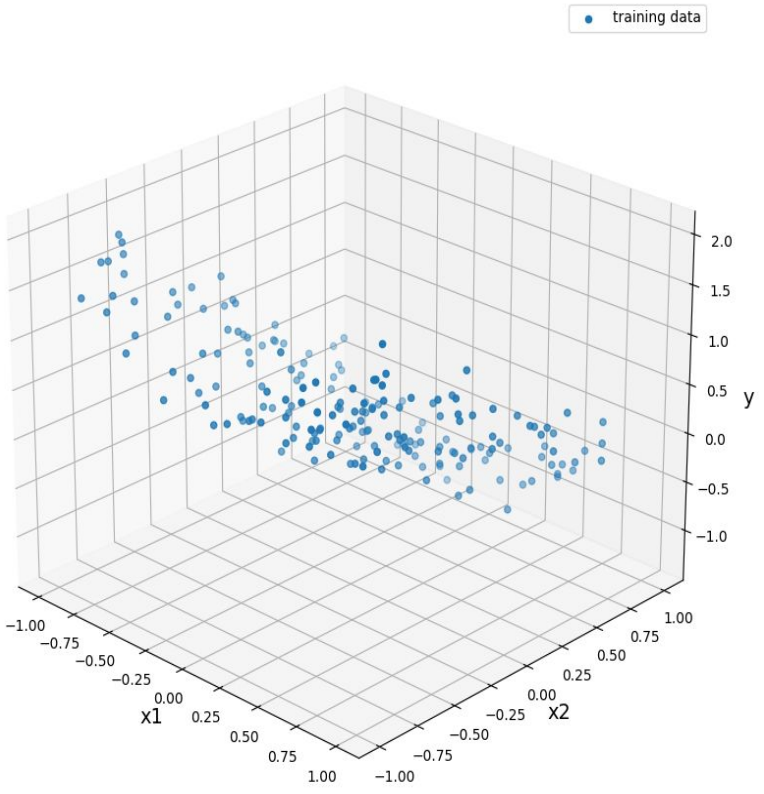

Fig1) a) This is a 3d plot the training data. Where the two axis x1 and x2 are the input features and the y axis is the output value.

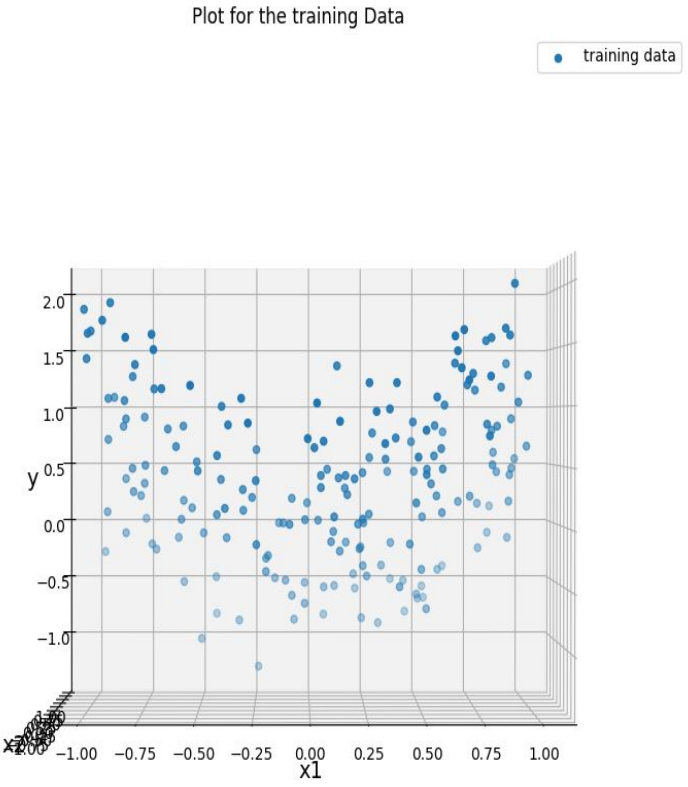

Fig 1) b) This is the plot of training data. Here the horizontal axis denotes the x1 feature and y axis denotes the output values.

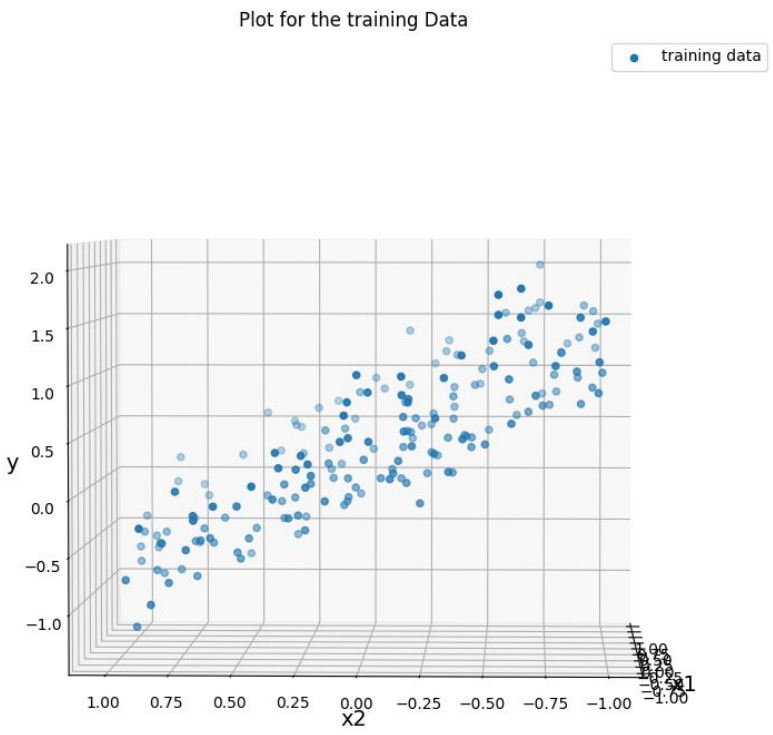

Fig 1) c) This is the plot of training data. Here the horizontal axis denotes the x2 feature and y axis denotes the output values.

If we look at all of the figures above then we can see that output y has a parabolic relationship with the feature x1 i.e value of y is the highest at the end of the points but is lower in the middle and has a linear relationship with the feature x2 i.e the value of y decreases if x2 is large and value of y increases if the value of x2 is small.

If we look at the figures correctly then we can infer that the training data is on the curve. i.e y = f(x1,x2).
Where f is a function that gives output depending on the weighted sum of the square of feature X1 and feature X2.

The dummy regressor model which always predicts the mean of the data can be trained using the snippet below and gives a r2 score of 0.0 .

Here X is the input feature and y is the groundTruth of the Data.
getPolynomialFeature is just a function that returns the features with extra polynomial features equal to all combinations of powers of the two features up to power 5 in this case.

```
XPoly = getPolynomialFeature(X,5)
dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(XPoly,y)
score = dummy_regr.score(XPoly,y)
print('The score of dummy regressor is {}'.format(score))
```

i) b)

Parameter C	Intercept	Coefficient	Score	Figure number
1	0.39414069020558207	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0.0	Fig 2) a)
10	0.2079686508257265	0,0, -0.88624365, 0.43510669, 0, 0,0,0,0,0,0,0,0,0,0,0, 0,0,0, 0,0	0.8399812347749618	Fig 2) b)
100	0.03417740067757019	0,0, -0.99334966, 0.93017547, 0, 0,0, -0.07244852,0, 0,0,0,0,0,0,0, 0,0,0,0,0	0.9139897260287999	Fig 2) c)
1000	0.0300988387958403	0,-0.0126226, -0.99494037, 0.97212851, 0, -0.04999679, 0,-0.21357153, 0,0,0,0,0, 0.10806154, 0,0,0,0,0.04526124,0.14771844,	0.9187780289030455	Fig 2) d)

		0.03063837		
10000	0.025355474519896837	0,1.14450983e-01,-9.00312751e-01,9.93951537e-01,-9.10447969e-02,-6.01802219e-02,-6.81129215e-01,-5.35941366e-01,6.34685465e-04,-2.10104376e-01,-2.45972830e-02,9.45439152e-02,2.72681482e-02,1.60926265e-01,3.05783490e-03,6.50418284e-01,1.43997019e-02,-2.20818979e-02,5.66221467e-01,1.93674617e-01,1.04249766e-01	0.9210958048771817	Fig 2) e)
100000	0.026206401797652268||	0,0.13390214,0.8609131,1.00794272,-0.11563781,-0.0853756,-0.78705198,-0.66230404,0.01233442,-0.31685264,-0.04594015,0.12884158,0.04283768,0.17000866,0.02647131,0.75527751	0.921200369846944	Fig 2) f)

		,0.09170201 , -0.05371961 ,0.67619575 ,0.19812215 ,0.1754317		

Table 1) Table shows the parameters of our lasso regression model when we change our penalty parameter C. The column Parameter C denotes the value of the hyperparameter-used, intercept denotes the intercept value of the model, coefficient denotes the coefficient values of the model, score denotes the r2 score of the model on training data, i.e calculated using model.score function from scikit learn and figure column denotes the figure used to denote the test plot of the model.

While training the lasso regression model we add the penalty term (½*c)*θ (where C is our hyper parameter, θ are our model parameters) to the mean square error in the cost function. The penalty term is added to make sure that our model parameters do not become too large and are as close to zero as possible. However if we make our C too big lets say 1000 then the regularisation term θ/C will become insignificant and we might get model parameters that are really large. This is quite evident from the table 1 where as we increased C, our model parameters value increased along with C. We can see from the table that for lower values of C our model parameters were close to zero but as we increased the value of C our model parameters deviated,(increased) and were far from zero.
Also from the table it is quite evident that for a small value of C=1 all of the model parameters values are zero and we only have the model intercept.
All the regression models trained in this assignment are trained on the polynomial features which can be calculated using the function defined below: Here X are the original feature and polynomialValue is the maximum degree which we want our features to be.

```
def getPolynomialFeature(X,polynomialValue):
    '''
    return the polynomial feature for the feature X
    :param X: array of feature values
    :param polynomialValue: degree of polynomial
    :return: transformed feature
    '''
    return PolynomialFeatures(polynomialValue).fit_transform(X)
```

To get the polynomial features simply use:

```
XPoly = getPolynomialFeature(X,5)
```

Here X is the original 2d feature provided in the dataset.

Following function can be used to train and validate the lasso regression model on the test grid data using the polynomial features.

```
def trainLassoModel(features,outputValue,parameterC):
    """
    this function returns the Lassoregression model for the
hyperparameter C with features are the input data and
outputValue is the
    predicted value
    :param features: input features for the model
    :param outputValue: outputValue for the model
    :param parameterC: Value of the hyperParameter.
    :return: trained model
    """
    alpha = 1/(2*parameterC) # converting the penality
parameter for the Sklearn method
    model = linear_model.Lasso(alpha=alpha)
    model.fit(features,outputValue)
    return model
```

To train the lasso using above function simply use :

```
model = trainLassoModel(XPoly,y,hyperParameter)
```

Here Xpoly is the polynomial feature defined above, y are the corresponding output and the hyperparameter is the penalty term used to train the model.
To train the model on all of the penalty terms defined in the table the following snippet can be used.

```
valuesOfc = [1,10,100,1000,10000,100000,100000]
testX = getGridFeatures(-5, 5)
testXPoly = getPolynomialFeature(testX, polynomialValue=5)
XPoly = getPolynomialFeature(X,5)
for hyperParameter in valuesOfc:
    model = trainLassoModel(XPoly,y,hyperParameter)
    predictions = model.predict(testXPoly)
    print('Trained Lasso Regression model for
{}'.format(hyperParameter))
    print('model Parameters slope {} intercept {}'.format(
        model.coef_,model.intercept_
    ))

plot3dDataAndPrediction(X[:,0],X[:,1],y,testX[:,0],testX[:,1],
predictions)
    print('*********************************************')
```

i) c) Here are the figures that plot the training data as the scatter plot with blue dots and the test grid prediction as the yellow surface plot.

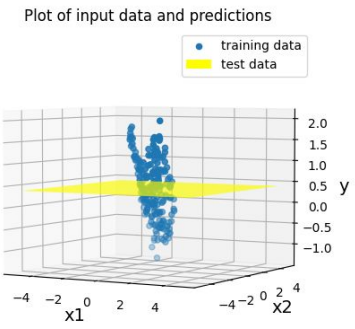

Fig 2) a) Plot for the trained Lasso Regression model with hyper parameter C=1

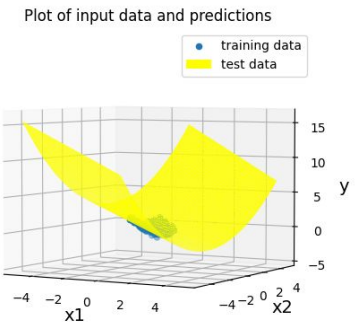

Fig 2) b) Plot for the trained Lasso Regression model with hyper parameter C=10

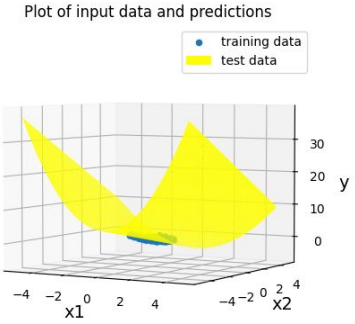

Fig 2) c) Plot for the trained Lasso Regression model with hyper parameter C=100

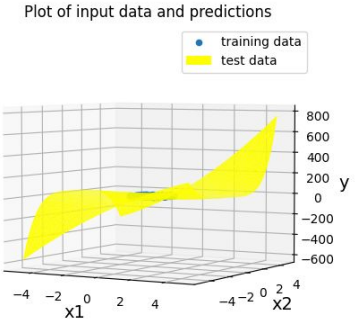

Fig 2) d) Plot for the trained Lasso Regression model with hyper parameter C=1000

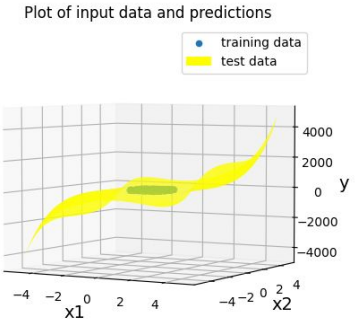

Fig 2) e) Plot for the trained Lasso Regression model with hyper parameter C=10000

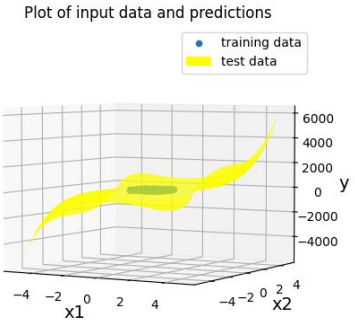

Fig 2) f) Plot for the trained Lasso Regression model with hyper parameter C=100000

If we take a close look at all the plots in the fig 2) then we can see that as we increase the value of C, the complexity of the model increases. i.e it is trying to fit itself correctly on the training data and also trying to fit the noise present in the data instead of just learning the general behaviour of the dataset.
We can see that in fig 2) b) 2 c) 2) d) the model is trying to learn the general behaviour of the training data but as we increased the value of C beyond that model is trying to fit all of the training data including the noise, and is getting more complex.

i) d) Overfitting is the scenario when our model becomes too complex and also tries to fit the noise in the training data. Such models might give good accuracy scores on the training data but might perform badly on unseen/new data. This is evident from the figure 2) f) and Table 1) As we have increased the value of our hyperparameter C, although the accuracy score of the model increases giving the maximum score at C=100000 but if we look at the surface plot of the predictions at C=100000 we can see that the model has become more complex and is also trying to fit the noise which is present in the training dataset. Such a model when given unseen data might perform badly on such a dataset.
Underfitting is the scenario when our model is too simple and is unable to capture the behavioural properties of the dataset. Since such a model is unable to capture the behavioural properties of the dataset, it also performs badly on the unseen/new dataset. This is evident from the fig 2) a) and table 1) when the value of our hyper parameter C=1, then all of the coefficients of our models are zero and we are left with a model that only has a single intercept. And we can see from fig 2) a) that such a model is too simple for this dataset and does not capture any behavioural properties of the dataset. Such a model can be termed as underfit.
Looking at fig 2) b) and fig 2) c) we can see that the model trained using the hyperparameters C=100 capture the behavioural properties of the dataset and such model are fairly simple, and doesn't try to fit the noise of the dataset and also gives a decent accuracy score on the dataset. Also looking at Table 1 we can also see that for C=100 that only three coefficients,which are non zero which is also a proof that our model is fairly simple and tries to learn a simple function to capture the behavioural properties of the dataset. Such

models should be used for testing as they capture the behavioral properties of the dataset ignoring the noise present in the training data. Also the r2 score of our models at C=100 is better than our Dummy regressor model, it also proves that our model has learnt something, not just noise and performs better than the dummy classifier.
Hence we should always chose the value of C which gives a good accuracy, with a very simple model.
i) e)

Model HyperParameter C	Intercept	Coefficient	Score	FigureNumber
0.00001	0.39	0.00000000e+00,-1.18953735e-06,-1.30481380e-03,3.82614931e-04,-9.88470341e-05-3.92911511e-06,-1.09073645e-05,-4.62452854e-04,-1.06730298e-05,-7.91481937e-04,3.17457845e-04,-7.00427380e-06,1.18035297e-04,-7.56000465e-05,-1.84986604e-05,-2.42326244e-05,-2.72125888e-04,-9.56716612e-06,-2.61877362e-04,-1.64231458e-05,-5.73316361e-04	0.0	Fig 3) a)
0.0001	0.39	0,0,-0.01,0,0,0,0,0,0,-0.01,0,0,0,0,0,0,0,0,0,0,0.01	0.03	Fig 3) b)

0.001	0.37	0,0,-0.11,0.04 -0.01,0,0,-0.04,0,0.06,0.03,0,0.01,-0.01,0,0,-0.02,0,-0.02,0,-0.05	0.26	Fig 3) c)
0.01	0.25	0,0.01,-0.43, 0.22,-0.01, -0.01,0,-0.15,0.01,-0.22, 0.18,0.01,0.06,0,-0.02,0, -0.09,0,-0.07,0.01,-0.14	0.76	Fig 3) d)
0.1	0.11	0,0,-0.76,0.51 ,0,-0.04,-0.01,-0.2,0.06, -0.2,0.35,0.02 ,0.11,0.06, -0.04,0,-0.12, 0.02,-0.02, 0.06 -0.03	0.91	Fig 3) e)
1.0	0.06	0,-0.01,-0.93 0.76,-0.06, -0.06,-0.12,-0.23,0.07,-0.14 0.21,0.03,0.06	0.92	Fig 3) f)

		0.17,-0.01, 0.13,-0.08, -0.02, 0.15,0.11,0 .09		
10	0.03	0,0.07,-0.9 ,0.96,0.11, -0.08,-0.5, -0.48, 0.06,-0.22, 0, 0.1,0.03,0. 18,0.02,0.5 ,0.01, -0.08,0.48, 0.17,0.13	0.92	Fig 3) g)
100	0.03	0,0.13,-0.8 6 1,-0.12,-0. 09,-0.75,-0 .65, 0.02,-0.31, -0.04,0.13, 0.04,0.17,0 .03 0.73,0.09, -0.06,0.66, 0.19 0.17	0.92	Fig 3) h)
1000	0.03	0,0.13,-0.8 6, 1.01,-0.12, -0.09,-0.79 ,-0.67,0.01 ,-0.33, -0.05,0.13, 0.04,0.17,0 .03 ,0.76,0.1, -0.06,0.69, 0.2,0.18	0.92	Fig 3) i)

Table 2) the above table shows the Ridge regression model parameter values (intercept, coefficient) when trained using different penalty terms (Hyper parameter C). The score represents how the model performs on the training data. The figure column tells where the plotted figure of the model is.

Note : I have chosen to round the values of model parameters upto two decimal places from row two onwards as if not then I was getting a very large floating point value. I have not rounded up the values of the first row just to demonstrate that all the model parameters never had the value zero in the case of the ridge regression model as opposed to the lasso regression model. This is due to the fact that in the lasso regression model we only penalise the weights but in the ridge regression model we penalize the square of those weights.

To train a ridge regression model following function can be used :

```
def trainRidgeModel(features,outputValue,parameterC):
    """
    this function returns the Ridgeregression model for the
hyperparameter C with features are the input data and
outputValue is the
    predicted value
    :param features: input features for the model
    :param outputValue: outputValue for the model
    :param parameterC: Value of the hyperParameter.
    :return: trained model
    """
    alpha = 1/(2*parameterC) # converting the penality
parameter for the Sklearn method
    model = linear_model.Ridge(alpha=alpha)
    model.fit(features,outputValue)
    return model
```

To train a ridge regression model simply use :

```
model = trainRidgeModel(XPoly,y,hyperParameter)
```

Where Xploy are the input features, y are the output features and hyperparameter is the value of the hyperparameter C.

To train the Ridge regression model on all the hyperparameter values a for loop can be used such as :

```
testX = getGridFeatures(-5, 5)
valuesOfc = [0.00001,0.0001, 0.001, 0.01,0.1,1,10, 100, 1000]

testXPoly = getPolynomialFeature(testX, polynomialValue=5)
XPoly = getPolynomialFeature(X,5)
for hyperParameter in valuesOfc:
    model = trainRidgeModel(XPoly,y,hyperParameter)
    predictions = model.predict(testXPoly)
    print('Trained Lasso Regression model for
{}'.format(hyperParameter))
    print('model Parameters slope {} intercept {}'.format(
        model.coef_,model.intercept_
```

```
    ))

plot3dDataAndPrediction(X[:,0],X[:,1],y,testX[:,0],testX[:,1],
predictions)
    print('**********************************************')
```

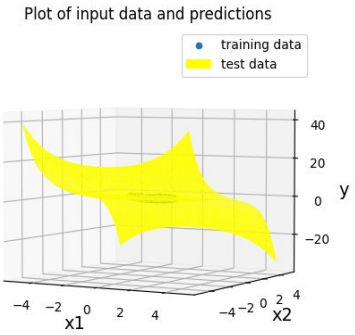

Fig 3) a) Plot for the trained Ridge Regression model with hyper parameter C=0.00001

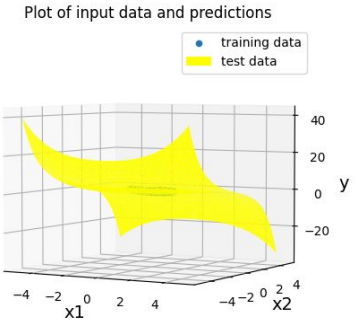

Fig 3) b) Plot for the trained Ridge Regression model with hyper parameter C=0.0001

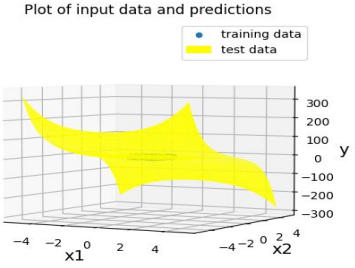

Fig 3) c) Plot for the trained Lasso Regression model with hyper parameter C=0.001

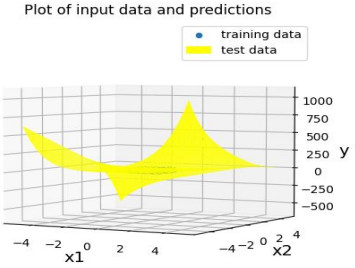

Fig 3) d) Plot for the trained Ridge Regression model with hyper parameter C=0.1

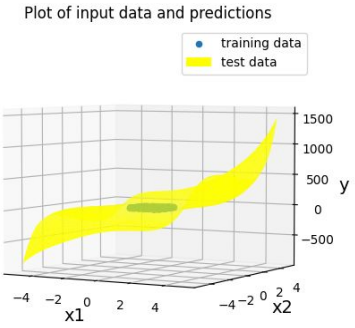

Fig 3) e) Plot for the trained Ridge Regression model with hyper parameter C=1

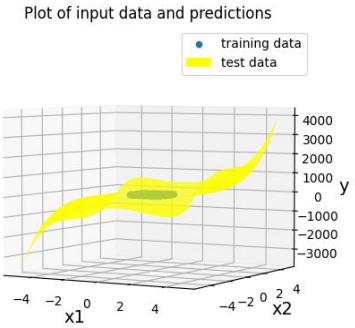

Fig 3) f) Plot for the trained Lasso Regression model with hyper parameter C=10

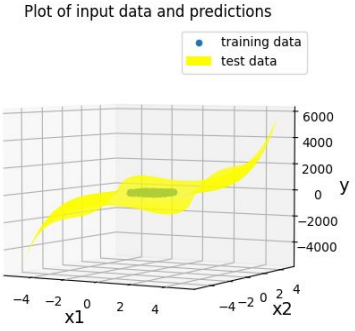

Fig 3) g) Plot for the trained Ridge Regression model with hyper parameter C=100

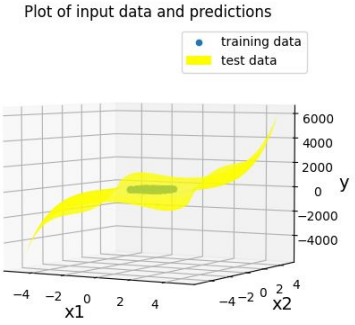

Fig 3) h) Plot for the trained Ridge Regression model with hyper parameter C=100

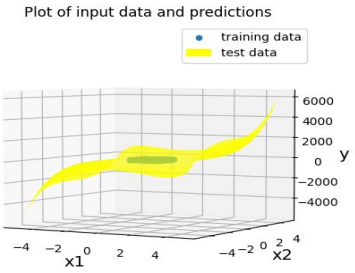

Fig 3) i) Plot for the trained Ridge Regression model with hyper parameter C=1000

From table 1 and table 2 it is evident that if we increase the value of C then the value of parameters for the lasso regression and ridge regression model increases. But the magnitude of increase is much more in the case of lasso regression than in the Ridge regression. I.e changing C from 1 to 10 we can see in the lasso regression model, that there are some parameters in the model who increased directly to 0.8 from 0. But if we see the change in ridge regression model (for C 1-10) then we can see that the model values increases but the magnitude of the increase value is very less as compared to that of the lasso model.
This behaviour happens because Ridge model is trained with a cost function that employs a squared penalty term i.e: (½*c)*θ*θT where as lasso model is trained with a cost function that employs a linear penalty ie: (½*c)*θ. Now from the penalty term it is evident that the model parameters will change quickly (if we change C) for the lasso model since we are only penalizing the weights but in the ridge regression model since we are penalizing the square of those weights the change in model parameters happens but its really slow.

Also in the lasso regression model when choosing a small values of C the model parameters were actually zero, however in the ridge regression model even though we have chosen a more smaller C=0.0001 the actual model parameters were never zero, there were always some model parameters that were non zero, they are zero, (from row two onwards) in the table because I rounded off the figures upto two decimal points just to make presentation better.

ii) a)

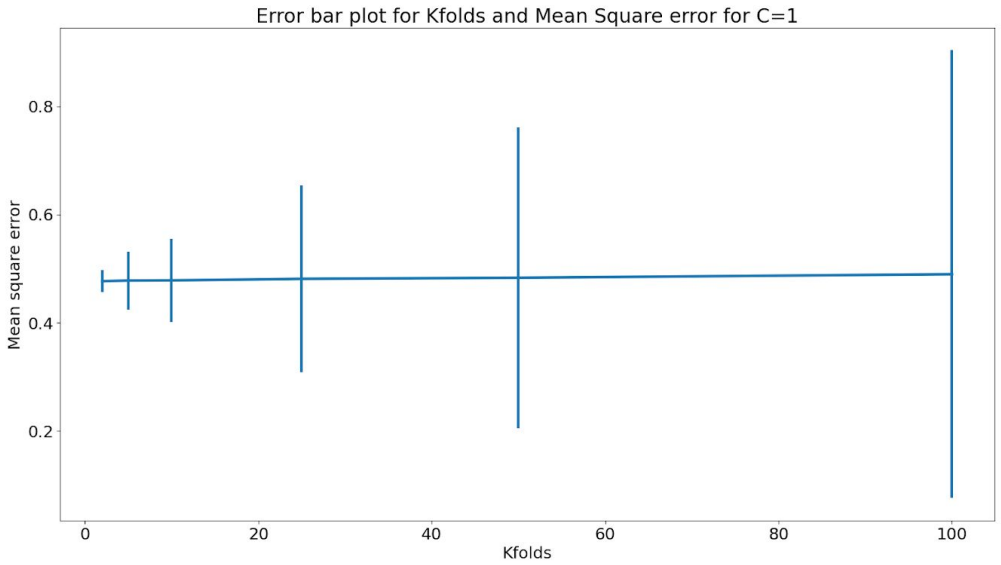

Fig 4) Error bar plot for kfolds and mean square error and square deviation. The horizontal axis denotes the number of folds used and the vertical axis denotes the mean square error and the bars over the line denotes the Standard deviation.

The table below shows the number of folds the meansq error for those folds and the std deviation for those folds

Kfolds	MeanSq error	Std Deviation
2	0.4768824042486902	0.02053094277450604
5	0.47808910611186467	0.05340646912238362
10	0.47840837339134124	0.07715253021437861
25	0.4813501604680042	0.1721974322698281
50	0.48315476756656817	0.277701827737670
100	0.4898330581795857	r0.4140071024453287

The plot is a straight line because at C=1 our model is a linear model as evident from the table 1 and fig 2)a) hence the mean square error remains the same even though we increase the folds. However, the variation of the predicted error increases as we increase the number of folds.
The number of folds denotes the parts in which we should divide our dataset into. We reserve 1 part of the dataset from the folds for validation and train on all the other parts. Hence if we increase the number of folds the number of testing data would decrease, i.e: if we chose 100 folds and if we have a dataset of 200 points we will be only left with 2 features (in one fold) each time to calculate the mean error and the variation error. Hence if we increase the value of folds by a very large amount we might end up with a validation dataset that is not representative of the real data. But if we chose a very small number of folds like 2 then our training data might decrease, (as it will divide the whole dataset into two sets having equal values and will use one set as testing and the other one as training) and since training data in such a case would be just half of the whole data, our model might not get trained properly.

Looking at the plot above, we can see that the variance of error is very less at fold = 5 and fold =10 and these folds also allow the model to have ample amount of training data to train on and also ample amount of test data to compute the accuracy of the model. Hence we should either chose fold=10 or fold=5

The function defined below trains a lasso regression model and validates it over different kfolds defined in the graph and also plots the value on the error bar graph.

```
def lassoWithKFold(kfolds = [2,5,10,25,50,100]):
    """
    this function trains and plots the mean and std error of
the dataset for different folds
    :param kfolds: list of folds to test
    :return:
    """
    mean_error = []
    std_error = []
    for kfold in kfolds:
```

```
        kfoldModel = KFold(n_splits=kfold)
        temp = []
        for train,test in kfoldModel.split(XPoly):
            model =
trainLassoModel(XPoly[train],y[train],parameterC=1)
            ypred = model.predict(XPoly[test])
            temp.append(mean_squared_error(y[test], ypred))
        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
        print('mean Sq Err {}, std Error{} , kfold
{}'.format(np.array(temp).mean(),np.array(temp).std(),kfold))

    barplot(kfolds,mean_error,std_error)
```

To use the function simply called :

```
kfoldsToUse = [2,5,10,25,50,100]
lassoWithKFold(kfoldsToUse)
```

b) Below is the plots of how the meansq error and the std deviation changes when we change the value of C for training the lasso regression model and validating it with kfold validation keep the folds = 10.

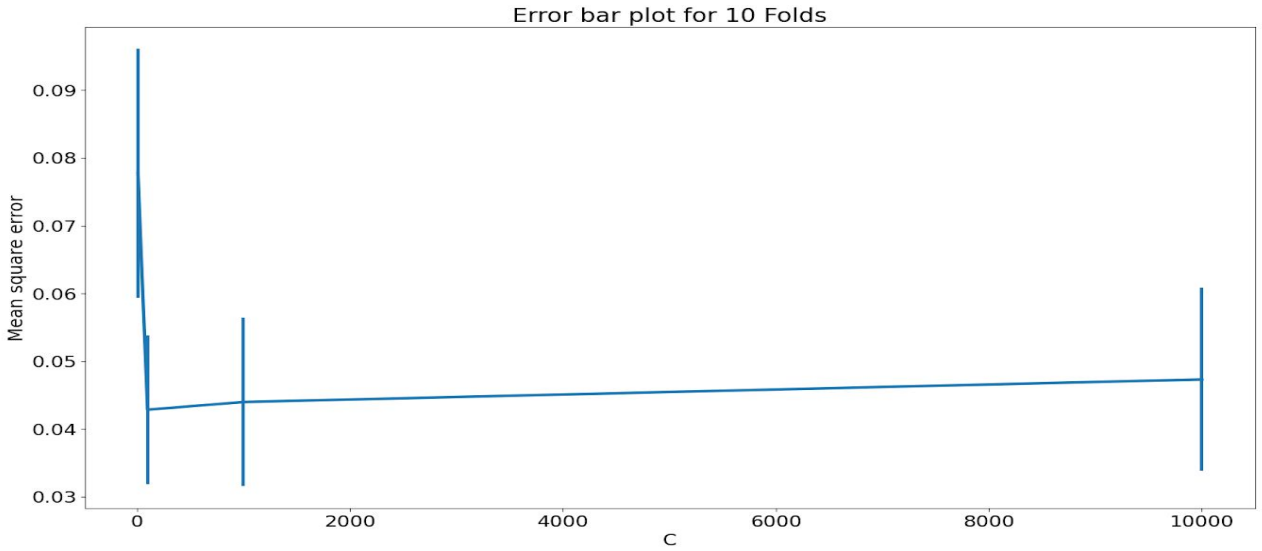

Fig 5) This is the plot of penalty parameter c vs the mean square error. Here the y axis denotes the mean square error the bar on the line shows the standard deviation error and x axis denotes the hyper parameter c.

Here on the plot hyperparameter value is denoted by the x axis, the mean square error value is denoted by the y axis and the std deviation error is plotted by the bar values on the line. Hyperparameter Values which were chosen to make the plot were 10,100,1000,10000, because from table 1 we can see that the model with C=1 was performing just as good as our baseline model and was giving the score of zero. However if we increased the hyperparameter values beyond 10000 there was an increase in score but it wasn't that much.
From the plot we can see that as we increase the penalty parameter there is a sharp dip in the mean square error and std deviation error from going to 10-100.
However, we can see that the mean error and std deviation error increases slightly if we go from 1000-10000 and then it remains constant for the rest of the values of C.

ii) c) Looking at the above plot I think we should go with the model at c=100 as that is the first model which has shown less standard deviations error, and the mean square error. Also both of the errors start to flatten out after that point, hence we can say that the model is not learning anything new about the data beyond those points and is just fitting the noise and is just getting more complex. This can also be backed by looking at table one where we can see that the models were fairly simple till c=100.

ii d) Below is the plot of how the meansq error and the std deviation changes when we change the value of C for training the Ridge regression model and validating it with kfold validation keeping the folds = 10.

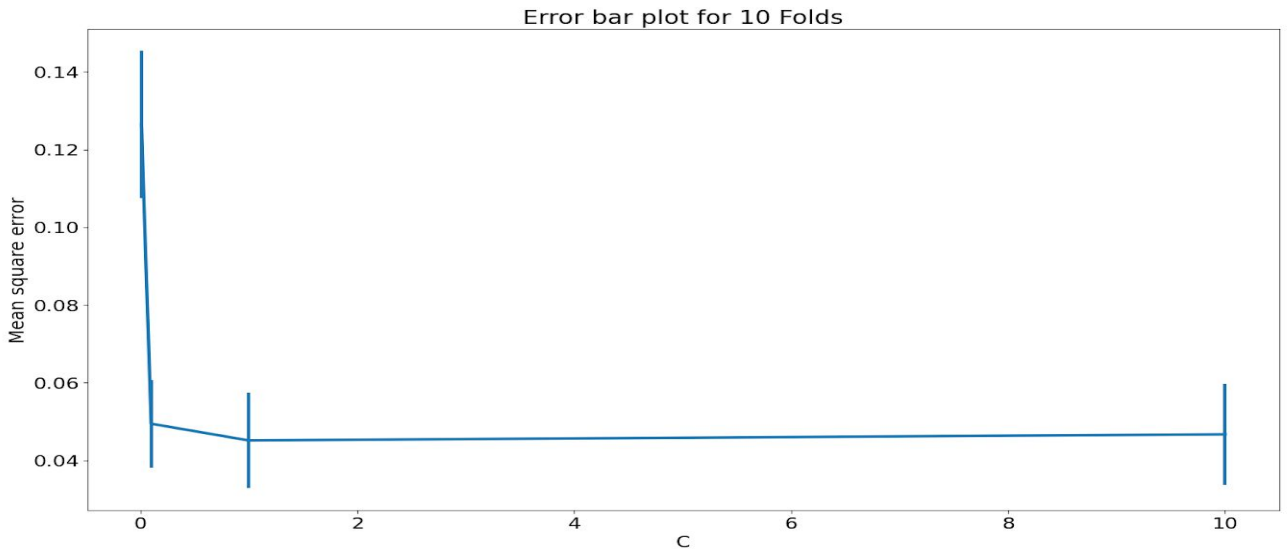

Fig 6) This is the plot of penalty parameter c vs the mean square error. Here the y axis denotes the mean square error the bar on the line shows the standard deviation error and x axis denotes the hyper parameter c.

Hyperparameter Values which were chosen to make the plot were 0.01,0.1,1,10, because from table 1 we can see that the model with C=0.001 was performing much better than our baseline model, but there was a significant accuracy increase while going from 0.001 to 0.01, hence 0.01 was chosed as a starting value. And if we increased the hyperparameter values beyond 10 there was an increase in score but it wasn't that much hence 10 was chosen as our final value.
From the plot we can see that as we increase the penalty parameter there is a sharp dip in the mean square error and std deviation error from going to 0.001 to 0.1. And then it steadily decreases till 1 and then flattens out.

Looking at the above plot I think we should go with the model at c=1 as it is the first model which has shown less standard deviations error, and the mean square error and also the errors starts to flatten out after that point.

Appendix

1) **Source code:**

```
# # id:22-22--22
import pandas as pd
import os
import numpy as np
import  matplotlib.pyplot  as  plt
from  mpl_toolkits.mplot3d  import Axes3D
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn import linear_model
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.dummy import DummyRegressor

def plot3dData(X,y,z):
    """
    this is a helper function to plot the graph on 3D Axis
    :param X: Data to plotted on Xaxis
    :param y: Data to be plotted on Yaxis
    :param z: Data to be plotted on Zaxis
    :return:
    """
    fig = plt.figure()
    ax = fig.add_subplot(111, projection ='3d' )
    ax.scatter(X, y, z,label='training data')
    ax.set_title('Plot for the training Data')
    ax.legend(loc='upper right')
    ax.set_xlabel('x1', size=14)
    ax.set_ylabel('x2', size=14)
    ax.set_zlabel('y', size=14)
    plt.show()
    return

def
plot3dDataAndPrediction(X,y,z,Xpred,Ypred,Zpred,fileSave=None)
:
    """
    this is a helper function to plot the graph on 3D Axis.
    We will plot predictions as a surface and training data as
scatterplot
    :param X: train Data to plotted on Xaxis
    :param y: train Data to be plotted on Yaxis
    :param z: train Data to be plotted on Zaxis
    :param Xpred: test Data to be plotted on Zaxis
    :param Ypred: test Data to be plotted on Zaxis
    :param zpredz: test Data to be plotted on Zaxis
    :param fileSave: if string then its the path of image to
save the plot
    :return:
    """
    fig = plt.figure()
    ax = fig.add_subplot(111, projection ='3d' )
    ax.scatter(X, y, z,label='training data')
```

```
    ax.set_title("Plot of input data and predictions")
    surf = ax.plot_trisurf(Xpred, Ypred,
Zpred,color='yellow',label='test data',shade=False)
    # patch to add colors to surface
    # taken from
https://stackoverflow.com/questions/54994600/pyplot-legend-pol
y3dcollection-object-has-no-attribute-edgecolors2d
    surf._facecolors2d = surf._facecolors3d
    surf._edgecolors2d = surf._edgecolors3d
    ax.legend(loc='upper right')
    ax.set_xlabel('x1', size=14)
    ax.set_ylabel('x2', size=14)
    ax.set_zlabel('y', size=14)
    if fileSave is None:
        plt.show()
    else:
        ax.view_init(4, -60) # angle calculated manually
        plt.savefig(fileSave)
    return

def getPolynomialFeature(X,polynomialValue):
    '''
    return the polynomial feature for the feature X
    :param X: array of feature values
    :param polynomialValue: degree of polynomial
    :return: transformed feature
    '''
    return PolynomialFeatures(polynomialValue).fit_transform(X)

def trainLassoModel(features,outputValue,parameterC):
    """
    this function returns the Lassoregression model for the
hyperparameter C with features are the input data and
outputValue is the
    predicted value
    :param features: input features for the model
    :param outputValue: outputValue for the model
    :param parameterC: Value of the hyperParameter.
    :return: trained model
    """
    alpha = 1/(2*parameterC) # converting the penality
parameter for the Sklearn method
    model = linear_model.Lasso(alpha=alpha)
    model.fit(features,outputValue)
```

```
    return model

def trainRidgeModel(features,outputValue,parameterC):
    """
    this function returns the Ridgeregression model for the
hyperparameter C with features are the input data and
outputValue is the
    predicted value
    :param features: input features for the model
    :param outputValue: outputValue for the model
    :param parameterC: Value of the hyperParameter.
    :return: trained model
    """
    alpha = 1/(2*parameterC) # converting the penality
parameter for the Sklearn method
    model = linear_model.Ridge(alpha=alpha)
    model.fit(features,outputValue)
    return model

def
getSmallestParameter(X,y,initialValue,divisionFactor,maxSteps)
:
    """
    this function is built to calculate the lowest value of C
for which lasso trained parameters are zero
    :param X: Input features
    :param y: output features
    :param initialValue: Intital value of the parameter C
    :param divisionFactor: We divide the value C by this after
each step and run again
    :param maxSteps: If model doesnot give zero parameters and
maxsteps has reached return the last value. this is a
fail-safe
    so that code doesnot go into forever loop.
    :return: value of hyperparameter for which we get model
parameters 0
    """
    runloop = True
    numIterations = 0
    valueOfC = initialValue
    while (runloop):
        numIterations +=1
        model = trainLassoModel(X,y,valueOfC)
        slope, intercept = model.coef_, model.intercept_
```

```
        zeroSlope = all(np.array(slope) == 0)
        interceptZero = True if model.intercept_ == 0 else
False
        if zeroSlope and interceptZero:
            runloop = False
        if numIterations > maxSteps:
            runloop = False
        valueOfC = valueOfC/float(divisionFactor)
    return valueOfC

def getGridFeatures(minValue,maxValue):
    """
    return the grid Test features between the min and max value
    :param minValue:
    :param maxValue:
    :return: TestSet
    """
    Xtest = []
    grid = np.linspace(minValue,maxValue)
    for i in grid:
        for j in grid:
            Xtest.append([i, j])
    Xtest = np.array(Xtest)
    return Xtest


def barplot(x,y,z):
    """
    plot the error between the X vs Y and Z
    :param x: horizontal axis data
    :param y: y axis data usually mean error here
    :param z: z axis data usually Std error here
    :return:
    """
    plt.rc("font", size = 18)
    plt.title('Error bar plot for 10 Folds')
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.errorbar(x, y, yerr=z, linewidth=3)
    # plt.xticks([0.001,1,10])
    plt.xlabel("C")
    plt.ylabel("Mean square error")
    plt.show()

def trainLassoRegressionModel(valuesOfc,X,y,testX):
```

```
    '''
    Helper function to train the lasso regression model and
show the plots of test.
    :param valuesOfc: different values of hyperparameter to
train lasso regression model on
    :param X: Input training data for the regressor
    :param y: output label for the training data
    :param testX: testing data for the model
    :return:
    '''
    testXPoly = getPolynomialFeature(testX, polynomialValue=5)
    XPoly = getPolynomialFeature(X,5)
    for hyperParameter in valuesOfc:
        model = trainLassoModel(XPoly,y,hyperParameter)
        predictions = model.predict(testXPoly)
        print('Trained Lasso Regression model for
{}'.format(hyperParameter))
        print('model Parameters slope {} intercept {}'.format(
            model.coef_,model.intercept_
        ))

plot3dDataAndPrediction(X[:,0],X[:,1],y,testX[:,0],testX[:,1],
predictions)

print('************************************************')

def trainRidgeRegressionModel(valuesOfc,X,y,testX):
    '''
    Helper function to train the Ridge regression model and
show the plots of test.
    :param valuesOfc: different values of hyperparameter to
train lasso regression model on
    :param X: Input training data for the regressor
    :param y: output label for the training data
    :param testX: testing data for the model
    :return:
    '''
    testXPoly = getPolynomialFeature(testX, polynomialValue=5)
    XPoly = getPolynomialFeature(X,5)
    for hyperParameter in valuesOfc:
        model = trainRidgeModel(XPoly,y,hyperParameter)
        predictions = model.predict(testXPoly)
        print('Trained Lasso Regression model for
{}'.format(hyperParameter))
```

```
        print('model Parameters slope {} intercept {}'.format(
            model.coef_,model.intercept_
        ))

plot3dDataAndPrediction(X[:,0],X[:,1],y,testX[:,0],testX[:,1],
predictions)

print('************************************************')

def lassoWithKFold(kfolds = [2,5,10,25,50,100]):
    """
    this function trains and plots the mean and std error of
the dataset for different folds
    :param kfolds: list of folds to test
    :return:
    """
    mean_error = []
    std_error = []
    for kfold in kfolds:
        kfoldModel = KFold(n_splits=kfold)
        temp = []
        for train,test in kfoldModel.split(XPoly):
            model =
trainLassoModel(XPoly[train],y[train],parameterC=1)
            ypred = model.predict(XPoly[test])
            temp.append(mean_squared_error(y[test], ypred))
        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
        print('mean Sq Err {}, std Error{} , kfold
{}'.format(np.array(temp).mean(),np.array(temp).std(),kfold))
    barplot(kfolds,mean_error,std_error)

if __name__ == '__main__':
    df = pd.read_csv("dataset.csv", comment='#', header=None)
    # print(df.head())
    X1 = df.iloc[:, 0]
    X2 = df.iloc[:, 1]

    y = np.asarray(df.iloc[:, 2])
    plot3dData(X1,X2,y)
    X = np.column_stack((X1, X2))
    #getting the polynomial features
    XPoly = getPolynomialFeature(X,5)
    # making the test set
```

```
    testX = getGridFeatures(-5, 5)
    # Training the dummy regressor
    dummy_regr = DummyRegressor(strategy="mean")
    dummy_regr.fit(XPoly,y)
    score = dummy_regr.score(XPoly,y)
    print('The score of dummy regressor is {}'.format(score))
    # initialising the different Penalty values for Lasso And
Ridge
    valuesOfcLasso = [1,10,100,1000,10000,100000,100000] #
check why this value is always there 0.39414069020558207
    valuesOfcRidge = [0.00001,0.0001, 0.001, 0.01,0.1,1,10,
100, 1000]
    # training the lasso regression model and plotting its
prediction on the test data
    trainLassoRegressionModel(valuesOfcLasso,X,y,testX)
    # training the ridge regression model and plotting its
prediction on the test data
    trainRidgeRegressionModel(valuesOfcRidge,X,y,testX)
    # plotting the barlot for kfolds vs mean error and std
deviation for C= 1
    kfoldsToUse = [2,5,10,25,50,100]
    lassoWithKFold(kfoldsToUse)
    # plotting the bar plot for hyperparameter C vs the mean
error and std deviation for Lasso model using folds = 10.
    mean_error = []
    std_error = []
    kfoldModel = KFold(n_splits=10)
    valuesOfcLasso = [10,100,1000,10000]
    for valueOfc in valuesOfcLasso:
        temp = []
        for train,test in kfoldModel.split(XPoly):
            model =
trainLassoModel(XPoly[train],y[train],parameterC=valueOfc)
            ypred = model.predict(XPoly[test])
            temp.append(mean_squared_error(y[test], ypred))
        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
        print('meanError {} stdError {} Hyperparam {}'.format(
            np.array(temp).mean(),np.array(temp).std(),valueOfc
        ))
        print('**************************************')
    barplot(valuesOfcLasso,mean_error,std_error)
    # plotting the bar plot for hyperparameter C vs the mean
error and std deviation for Ridge model using folds = 10.
```

```
    valuesOfcRidge = [0.01,0.1,1,10]
    for valueOfc in valuesOfcRidge:
        temp = []
        for train,test in kfoldModel.split(XPoly):
            model =
trainRidgeModel(XPoly[train],y[train],parameterC=valueOfc)
            ypred = model.predict(XPoly[test])
            temp.append(mean_squared_error(y[test], ypred))
        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
        print('meanError {} stdError {} Hyperparam {}'.format(
            np.array(temp).mean(),np.array(temp).std(),valueOfc
        ))
        print('****************************************')
    barplot(valuesOfcRidge,mean_error,std_error)

    pass
```