Ritika Sharma MSc Computer Science- Intelligent Systems
21334470 sharmari@tcd.ie

WEEK 2 ASSIGNMENT

Dataset Used - # id:17-17-17

i) a) The fig below shows a 3d scatter plot of the values from the dataset with id:17-17-17.

- X axis represents X1 feature (Column 1)
- Y-axis represents X2 feature (column 2)
- Z-axis represents y, target value (Column 3)
- It looks more like a curvy surface

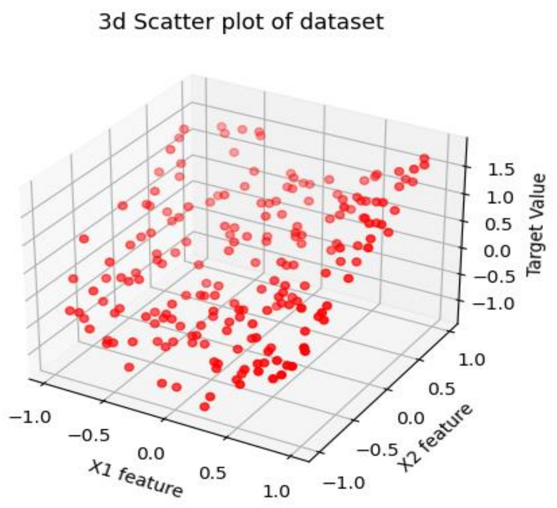


Fig 1. Scatter 3d plot of dataset

i) b) More features are added to the dataset by combining different powers of the existing features. For this purpose *PolynomialFeatures* class of sklearn.preprocessing is used. The degree set is 5, so we will have features till power 5.

- *Fit_transform* is used to transform the training data.

```
poly = PolynomialFeatures(degree=5)
X_poly = poly.fit_transform(X)
```

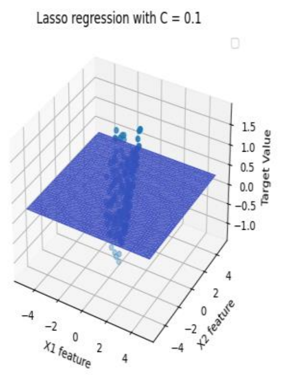
- To train the data using Lasso regression, the dataset has to be split into training and test data using *train_test_split* to avoid overfitting and underfitting of the data.
- It is also important to do hyperparameter tuning so that the optimal value is received for penalty parameters and the score achieved is the best attained.
- To check the optimum value of penalty parameter a wide range of values for C are used. Alpha = 1/(2*C)
- Lasso regression is used to train the dataset. Lasso regression introduces variance in the form of 1/ C $\sum_{j=1}^{n}$ |θj | to balance the bias, thus preventing over-fitting. Uses L1 penalty
- Allows to minimise the size of all coefficients thus removing unwanted features from model.

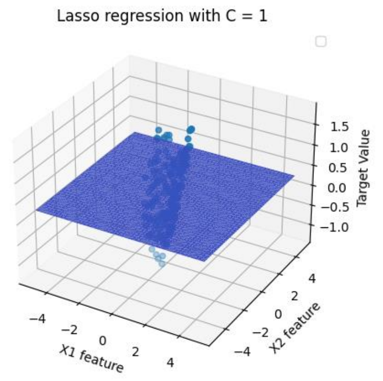
```
model = Lasso(alpha=1/(2*C))
model.fit(X_train, y_train)
```

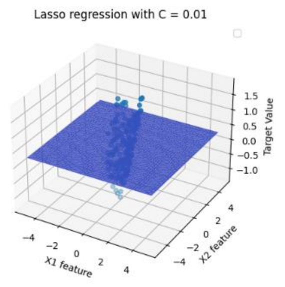
- The output of b) shows all the coefficients and intercepts for different values of C. From the output it can be observed that larger values of C provide better model. Thus, removing the effect of unwanted features. For values of C = [100, 1000, 5000] the coefficients hold significant value

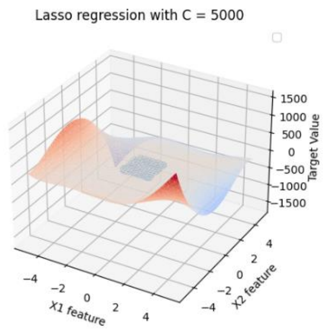
i) c) Here, the data prediction is done on a grid.

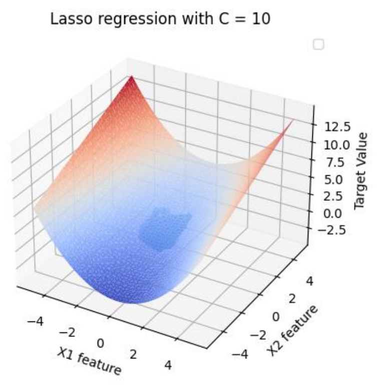
- The graph is plotted using *trisurf.* Below are the few graphs for different values of C.

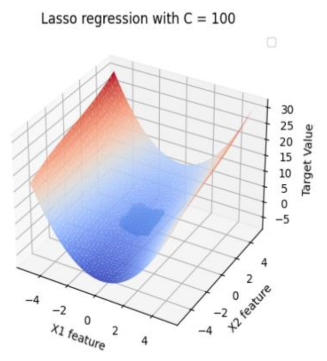












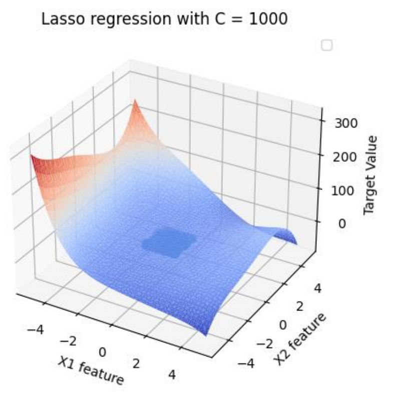


Fig 2. Different surface plots for Lasso Regression with diff values of C

C = 100 -> 'score_train': 0.9235489080196775, 'score_test': 0.9132090924850838
C = 1000 -> 'score_train': 0.9271217848585854, 'score_test': 0.91456749392692
C = 5000 -> 'score_train': 0.9294597725591695, 'score_test': 0.916778545462425

- Here the graph clearly shows for smaller values of C, the coefficients are zero and thus th slope is also 0.
- But for larger values of C like 100, 1000 a clear curved surface can be seen

i) d) Overfitting – Overfitting a model refers to when a model is trained too well on the training data set. This happens when a model learns the details and noises inside a training dataset very precisely thus impacting the performance of the model on the testing dataset. This negatively impacts the model tendency to generalise. Adding extra features generally add the risk of overfitiing.

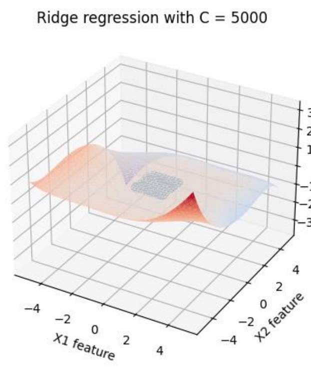
Underfitting – Underfitting a model refers to when it can neither model the training dataset nor generalise the features for new test dataset. This model is of no use. It can be deducted when a model has low accuracy scores for both training data and testing data.

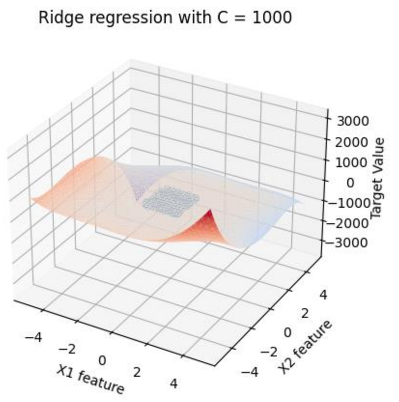
For a good fit, trade off between bias and variance should be maintained.

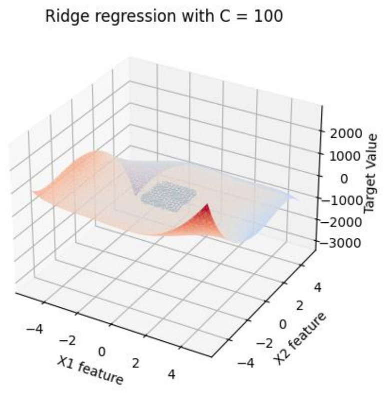
- For lower values of C, the coefficients are 0, thus providing 0 weight to the features and not introducing penalty to the model. Which again results in underfitting to the training dataset.
- For higher values of C, the coefficients have values and provide better accuracy scores for both training and testing dataset.
- For extremely high values of C, the alpha value decreases thus resulting in overfitting.

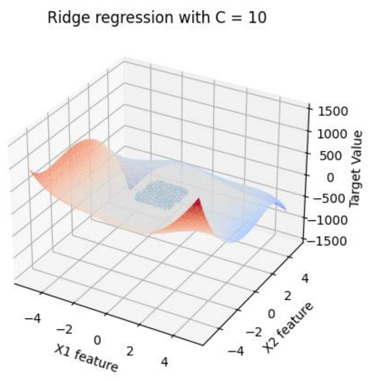
Ritika Sharma
21334470 MSc Computer Science- Intelligent Systems
sharmari@tcd.ie

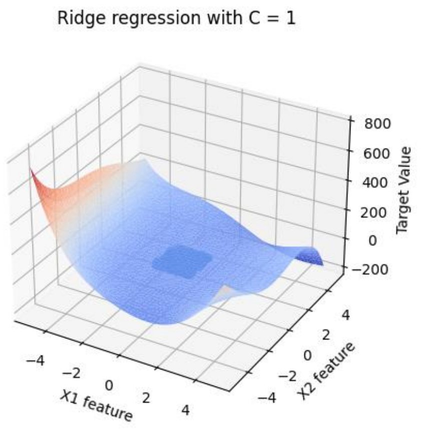
i) e) Ridge regression uses L2 penalty for cost function and has squared values of weight of the features. Below are the graphs plotted for different values of C.

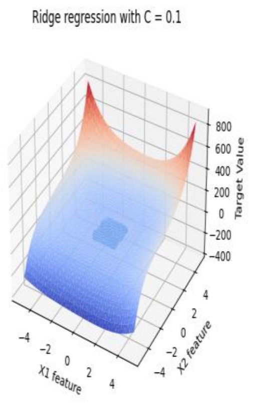












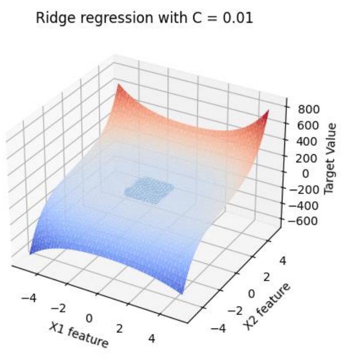


Fig 3. Surface graphs for different values of C for rige regression

```
model = Ridge(alpha=1/C)
model.fit(X_train, y_train)
y_pred = model.predict(X_poly_test)
```

- Here, for smaller values of C=1, better results are obtained and the accuracy score is 92% for training and testing data.
- For higher values of C, there is overfitting taking place. So better to take optimum value of C as 1.
- The difference between Lasso and Ridge is, ridge is used when almost all the features contribute towards the prediction whereas Lasso is used when some features do not contribute as the penalty parameter dissolves to 0. In this case, since there re a lot of features, Lasso regression is a better choice as it reduces the noise in the dataset.

ii) a) Here, the data is split using Cross-validation to 5 folds. This method is applied when the test data has too many noises. By changing the test data in every fold, it reduces the noise, smooths the dataset.

- The code for cross validation is done using *KFold* class of sklearn. The code is shared below. Here also different values of C are taken into consideration while building models in Lasso regression.

```
mean_error=[]
std_error=[]
for C in C_range:
  model = Ridge(alpha=1/C)
  temp=[]
  from sklearn.model_selection import KFold
  kf = KFold(n_splits=5)
  for train,test in kf.split(X_poly):
    model.fit(X_poly[train],y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test],ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())
```

- The KFold is done 5 times, thus providing different values of training and testing data in each iteration.

- The mean squared errors are stored in mean_error list and then plotted with C_range list to get the optimum value of C that provides the lease mean squared error for training and test dataset.
- The plot is drawn using errorbar fucntion of plot class

```
plt.errorbar(C_range,mean_error,yerr=std_error)
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.title('Zoomed Relationship between C and mean square error for Ridge Regre
ssion')
plt.xlim((0,200))
plt.show()
```

ii) b) With the help of graphs it can be observed that the value of C that provides least mean squared error is 1.

- After C>=1 and C<=50, the value of mean square gradually decrease and becomes constant after some time.

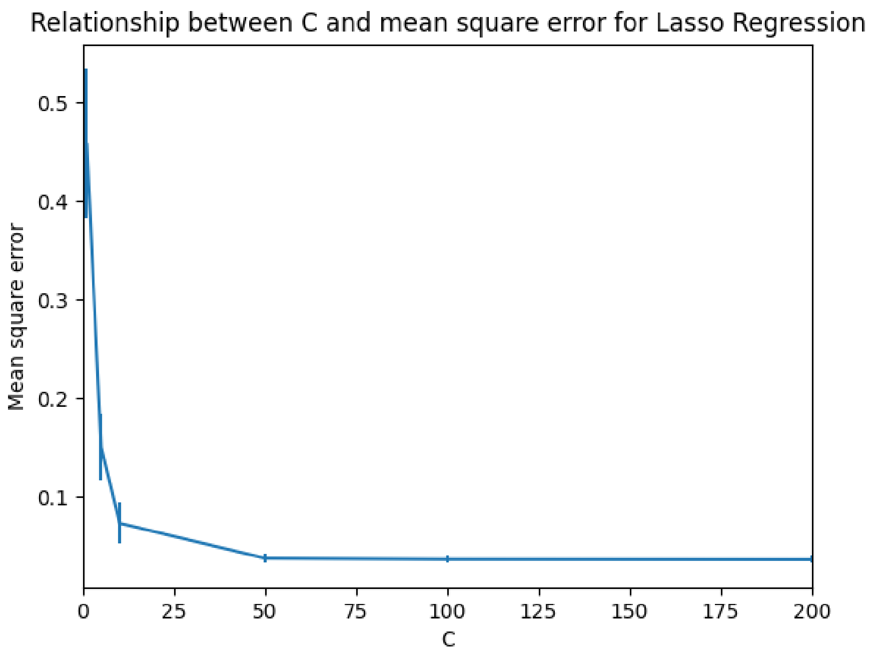


Fig 3. Relationship between values of C and mean square error Lasso Regression

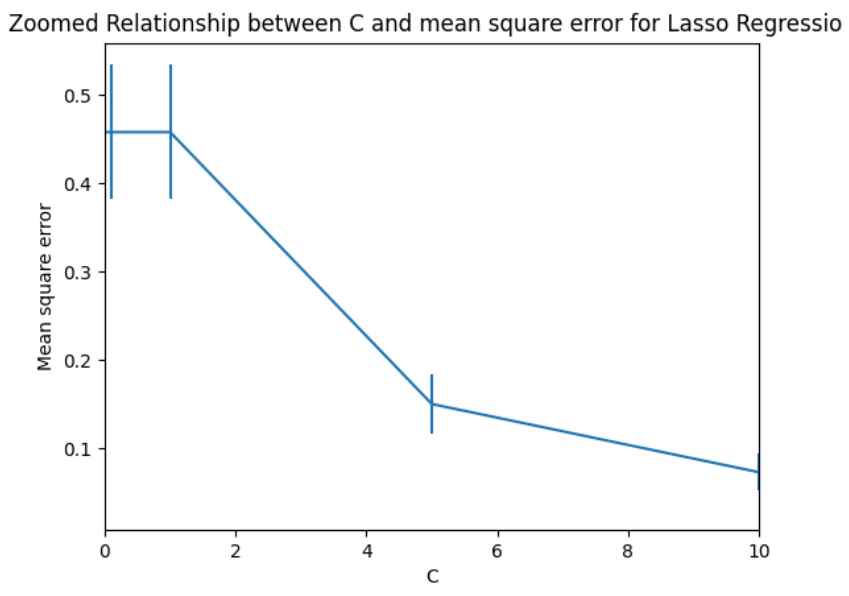


Fig 4. Zoomed image of Relationship between values of C and mean square error Lasso Regression

- In this zoomed image it can be seen for C<1, the values of mean square error is high, thus suggesting overfitting of the data.
- Thus C=1 would be a preferred choice.

ii) c) Plotting the same graph as above for Ridge regression. The logic and code remains almost equivalent. For different values of C, a Ridge model is trained and the mean square error for each corresponding C is stored in mean_error and std_error list.

```
model = Ridge(alpha=1/C)
  temp=[]
  from sklearn.model_selection import KFold
  kf = KFold(n_splits=5)
  for train,test in kf.split(X_poly):
    model.fit(X_poly[train],y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test],ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())
```

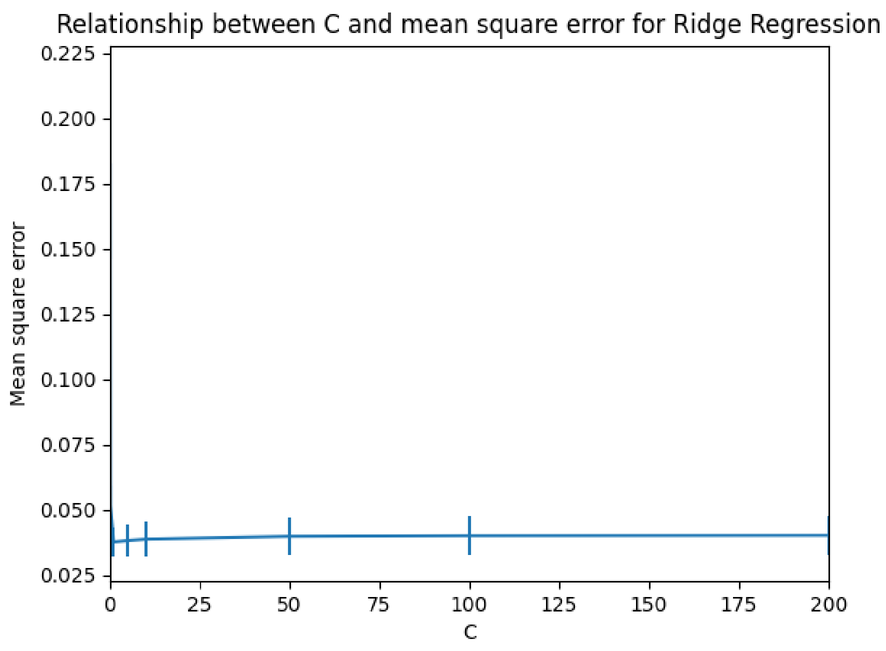


Fig 5. Relationship between values of C and mean square error Ridge regression

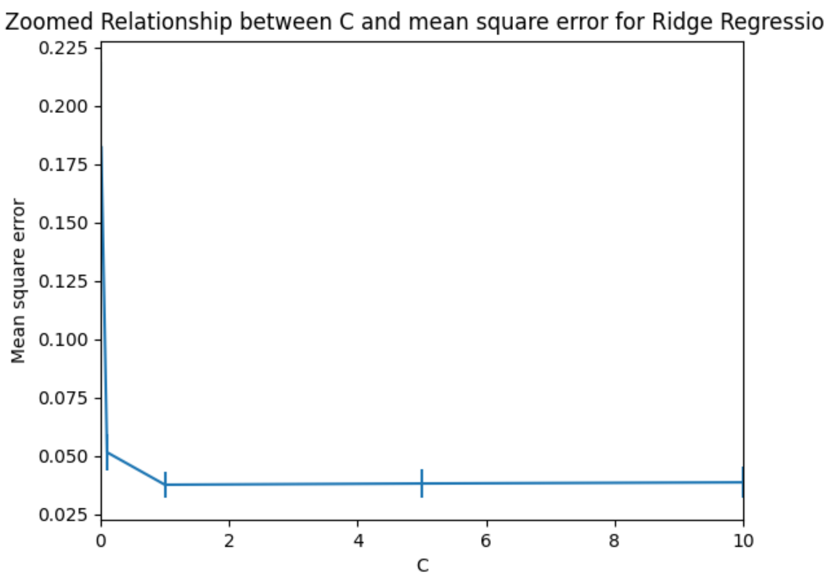


Fig 6. Zoomed image of Relationship between values of C and mean square error Ridge regression

- Here again, it can be clearly observed from the above graphs (normal and zoomed) that C=1 gives the best model and has least mean squared error.
- Higher values promote overfitting and lower values deflect too much from the data.
- Thus the optimum value of C =1.

Appendix

Part i) whole code –

```
# dataset id
# id:17-17-17

# import libraries
from operator import mod
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

# i) a)
# get data
df = pd.read_csv('week3.csv')
# print(df.head())
X1=df.iloc[ : , 0 ]
X2 = df.iloc[:, 1]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))
fig = plt.figure( )
ax = fig.add_subplot( 111 , projection ='3d' )
ax.scatter (X1, X2, y, c='r', marker='o' )

ax.set_xlabel('X1 feature')
ax.set_ylabel('X2 feature')
ax.set_zlabel('Target Value')
plt.title('3d Scatter plot of dataset')
plt.show()

# i) b) and c)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from matplotlib import cm

# create and fit transform
poly = PolynomialFeatures(degree=5)
X_poly = poly.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20)

Xtest =[]
grid = np.linspace(-5,5)
for i in grid :
    for j in grid :
        Xtest.append([i, j])
```

```python
Xtest = np.array( Xtest )
X_poly_test = poly.fit_transform(Xtest)

results=[]
C_range = [0.01,0.1,1,10,100,1000,5000]
for C in C_range:
    model = Lasso(alpha=1/(2*C))
    model.fit(X_train, y_train)
    y_pred = model.predict(X_poly_test)
    results.append( {
    'C' : C,
    'intercept' : model.intercept_,
    'coeff' : model.coef_,
    'score_train' : model.score(X_train,y_train),
    'score_test' : model.score(X_test,y_test),
    } )

    fig = plt.figure( )
    ax = fig.add_subplot( 111 , projection ='3d' )
    ax.scatter(X1,X2,y)
    surf = ax.plot_trisurf(Xtest[:,0], Xtest[:,1], y_pred, cmap=cm.coolwarm)
    ax.set_xlabel('X1 feature')
    ax.set_ylabel('X2 feature')
    ax.set_zlabel('Target Value')
    plt.title('Lasso regression with C = '+str(C))
    plt.legend()
    plt.show()

print(results)


# i) e)
from sklearn.linear_model import Ridge

for C in C_range:

    model = Ridge(alpha=1/C)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_poly_test)
    results.append( {
    'C' : C,
    'intercept' : model.intercept_,
    'coeff' : model.coef_,
    'score_train' : model.score(X_train,y_train),
    'score_test' : model.score(X_test,y_test),
    } )

    fig = plt.figure( )
    ax = fig.add_subplot( 111 , projection ='3d' )
    ax.scatter(X1,X2,y)
    surf = ax.plot_trisurf(Xtest[:,0], Xtest[:,1], y_pred, cmap=cm.coolwarm)
    ax.set_xlabel('X1 feature')
    ax.set_ylabel('X2 feature')
    ax.set_zlabel('Target Value')
    plt.title('Ridge regression with C = '+str(C))
    plt.show()

print(results)
```

Part ii) code –

```
# dataset id
# id:17-17-17

# import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# i) a)
# get data
df = pd.read_csv('week3.csv')
# print(df.head())
X1=df.iloc[ : , 0 ]
X2 = df.iloc[:, 1]
y=df.iloc[ : , 2]
X = np.column_stack((X1,X2))

poly = PolynomialFeatures(degree=5)
X_poly = poly.fit_transform(X)

C_range = [0.01,0.1,1,5,10,50,100,200, 500, 1000]

# ii) a)
from sklearn.linear_model import Lasso
mean_error=[]
std_error=[]
for C in C_range:
  model = Lasso(alpha=1/(2*C))
  temp=[]
  from sklearn.model_selection import KFold
  kf = KFold(n_splits=5)
  for train,test in kf.split(X_poly):
    model.fit(X_poly[train],y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test],ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())

plt.errorbar(C_range,mean_error,yerr=std_error)
plt.title('Relationship between C and mean square error for Lasso Regression')
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.xlim((0,200))
plt.show()

plt.errorbar(C_range,mean_error,yerr=std_error)
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.title('Zoomed Relationship between C and mean square error for Lasso Regre
ssion')
plt.xlim((0,10))
plt.show()

# ii) c)
from sklearn.linear_model import Ridge
mean_error=[]
std_error=[]
for C in C_range:
  model = Ridge(alpha=1/C)
  temp=[]
  from sklearn.model_selection import KFold
  kf = KFold(n_splits=5)
  for train,test in kf.split(X_poly):
    model.fit(X_poly[train],y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test],ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())

plt.errorbar(C_range,mean_error,yerr=std_error)
plt.title('Relationship between C and mean square error for Ridge Regression')
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.xlim((0,200))
plt.show()

plt.errorbar(C_range,mean_error,yerr=std_error)
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.title('Zoomed Relationship between C and mean square error for Ridge Regre
ssion')
plt.xlim((0,10))
plt.show()
```