Ridge and LAsso Regression implementation

Regularization

When we use regression models to train some data, there is a good chance that the model will overfit the given training data set. Regularization helps sort this overfitting problem by restricting the degrees of freedom of a given equation i.e. simply reducing the number of degrees of a polynomial function by reducing their corresponding weights.
In a linear equation, we do not want huge weights/coefficients as a small change in weight can make a large difference for the dependent variable (Y). So, regularization constraints the weights of such features to avoid overfitting. Simple linear regression is given as:

$y = \beta_0 + \beta_1 x1 + \beta_2 x2 + \beta_3 x3 + \ldots + \beta_P xP$

Using the OLS method, we try to minimize the cost function given as:

To regularize the model, a Shrinkage penalty is added to the cost function. Let’s see different types of regularizations in regression:

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

LASSO(Least Absolute Shrinkage and Selection Operator) Regression (L1 Form)

LASSO regression penalizes the model based on the sum of magnitude of the coefficients. The regularization term is given by

regularization=$\lambda * \sum |\beta_j|$

Where, λ is the shrinkage factor.

and hence the formula for loss after regularization is:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = \mathrm{RSS} + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Ridge Regression (L2 Form)

Ridge regression penalizes the model based on the sum of squares of magnitude of the coefficients. The regularization term is given by

regularization=$\lambda * \sum |\beta_j^2|$

Where, λ is the shrinkage factor.

and hence the formula for loss after regularization is:

This value of lambda can be anything and should be calculated by cross validation as to what suits the model.

Let's consider β_1 and β_2 be coefficients of a linear regression and λ = 1:

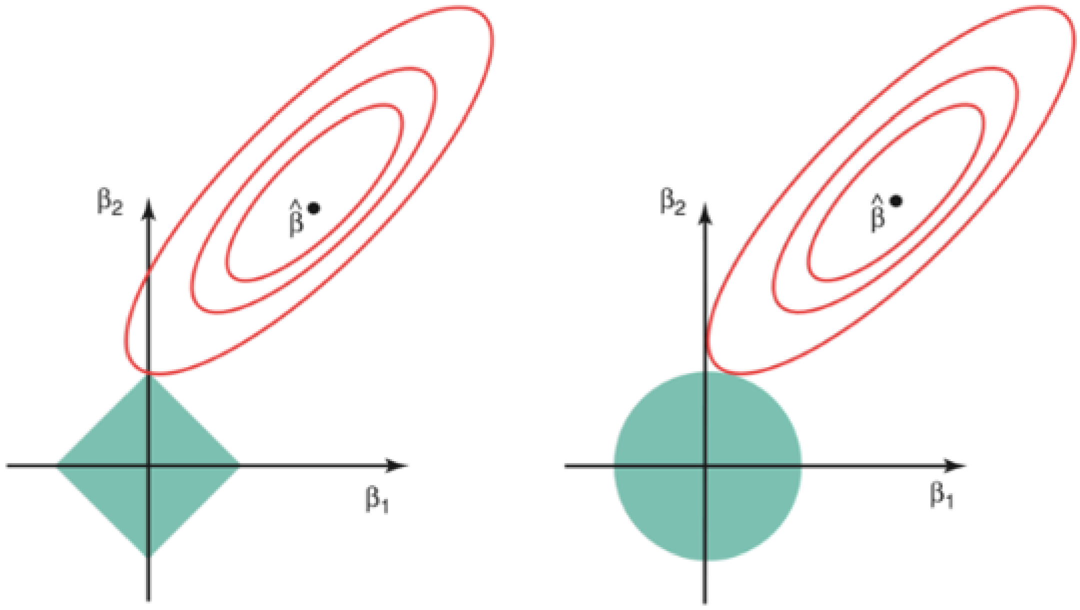
For Lasso, β_1 + β_2 <= s

For Ridge, β_1^2 + β_2^2 <= s

Where s is the maximum value the equations can achieve . If we plot both the above equations, we get the following graph:

The red ellipse represents the cost function of the model, whereas the square (left side) represents the Lasso regression and the circle (right side) represents the Ridge regression.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$



Difference between Ridge and Lasso

Ridge regression shrinks the coefficients for those predictors which contribute very less in the model but have huge weights, very close to zero. But it never makes them exactly zero. Thus, the final model will still contain all those predictors, though with less weights. This doesn't help in interpreting the model very well. This is where Lasso regression differs with Ridge regression. In Lasso, the L1 penalty does reduce some coefficients exactly to zero when we use a sufficiently large tuning parameter λ. So, in addition to regularizing, lasso also performs feature selection.

Why use Regularization?

Regularization helps to reduce the variance of the model, without a substantial increase in the bias. If there is variance in the model that means that the model won't fit well for dataset different that training data. The tuning parameter λ controls this bias and variance tradeoff. When the value of λ is increased up to a certain limit, it reduces the variance without losing any important properties in the data. But after a certain limit, the model will start losing some important properties which will increase the bias in the data. Thus, the selection of good value of λ is the key. The value of λ is selected using cross-validation methods. A set of λ is selected and cross-validation error is calculated for each value of λ and that value of λ is selected for which the cross-validation error is minimum.

In [1]:

```
from sklearn.datasets import load_boston
```

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
Bad key "text.kerning_factor" on line 4 in
C:\Users\91920\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_cla
ssic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
 (https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.templat
e)
or from the matplotlib source distribution
```

In [3]:

```
df=load_boston()
```

In [4]:

```
dataset = pd.DataFrame(df.data)
print(dataset.head())
```

```
         0     1     2    3      4      5     6       7    8      9    10  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7

       11    12
0  396.90  4.98
1  396.90  9.14
2  392.83  4.03
3  394.63  2.94
4  396.90  5.33
```

In [5]:

```
dataset.columns=df.feature_names
```

In [6]:

```
dataset.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

In [7]:

```
df.target.shape
```

Out[7]:

```
(506,)
```

In [8]:

```
dataset["Price"]=df.target
```

In [9]:

```
X=dataset.iloc[:,:-1] ## independent features
y=dataset.iloc[:,-1] ## dependent features
```

Linear Regression

In [10]:

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

lin_regressor=LinearRegression()
mse=cross_val_score(lin_regressor,X,y,scoring='neg_mean_squared_error',cv=5)
print(mse)
mean_mse=np.mean(mse)
print(mean_mse)
```

```
[-12.46030057 -26.04862111 -33.07413798 -80.76237112 -33.31360656]
-37.13180746769922
```

Ridge Regression

In [12]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge=Ridge()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=5)
ridge_regressor.fit(X,y)
```

Out[12]:

```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 1
0,
                                   20, 30, 35, 40, 45, 50, 55, 100]},
             scoring='neg_mean_squared_error')
```

In [13]:

```
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 100}
-29.905701947540372
```

Lasso Regression

In [14]:

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso=Lasso()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
lasso_regressor=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=5)

lasso_regressor.fit(X,y)
print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
```

```
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:532: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 4430.746729651311, tol
erance: 3.9191485420792076
  positive)
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:532: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 4397.459304778431, tol
erance: 3.3071316790123455
  positive)
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:532: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 3796.653037433508, tol
erance: 2.813643886419753
  positive)
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:532: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 2564.292735790545, tol
erance: 3.3071762123456794
  positive)
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:532: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 4294.252997826028, tol
erance: 3.4809104444444445
  positive)

{'alpha': 1}
-35.531580220694856
```

Elastic Net

In [19]:

```
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV, L
elasticCV = ElasticNetCV(alphas = None, cv = 5)
elasticCV.fit(X,y)
print(elasticCV.alpha_)
print(elasticCV.l1_ratio)
```

```
1.449640856754519
0.5
```

In [28]:

```
# calculate the prediction and mean square error
y_pred_elastic = elasticCV.predict(X)
mean_squared_error = np.mean((y_pred_elastic - y)**2)
print("Mean Squared Error on test set", mean_squared_error)
```

```
Mean Squared Error on test set 27.752516060229087
```

In [25]:

```
enet_cv_model.intercept_
```

Out[25]:

```
42.66300571110267
```

In [26]:

```
enet_cv_model.coef_
```

Out[26]:

```
array([-0.07026113,  0.05183146, -0.        ,  0.        , -0.        ,
        0.56092703,  0.02701927, -0.59693172,  0.2832434 , -0.01614854,
       -0.68609399,  0.0079763 , -0.78029054])
```

In [36]:

```
elasticCV.score(X, y)
```

Out[36]:

```
0.6712548925406525
```

Polynomial Regression

In [29]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
```

In [31]:

```
poly = PolynomialFeatures(degree=2, interaction_only=True) #change degree value to 3 and 4

poly_clf = linear_model.LinearRegression()
poly_clf.fit(X, y)
y_pred = poly_clf.predict(X)
```

In [32]:

```
mean_squared_error = np.mean((y_pred - y)**2)
print("Mean Squared Error on test set", mean_squared_error)
```

```
Mean Squared Error on test set 21.894831181729213
```

In [33]:

```
print(poly_clf.score(X, y))
```

0.7406426641094095

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: