Ridge and Lasso Regression (L1 and L2 regularization)

What is Regularization?

In a general manner, to make things regular or acceptable is what we mean by the term regularization. This is exactly why we use it for applied machine learning. In the domain of machine learning, regularization is the process which prevents overfitting by discouraging developers learning a more complex or flexible model, and finally, which regularizes or shrinks the coefficients towards zero. The basic idea is to penalize the complex models i.e. adding a complexity term in such a way that it tends to give a bigger loss for evaluating complex models.

In other words, a form of predictive modelling technique which investigates the relationship between a target variable to its predictor i.e., the independent variable is what we know as regression analysis. Mostly, this technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. A real-time example is the relationship between prediction of salary of new employees depending on years of work experience is best studied through regression.

Now, the next question arises is *why do we use Regression Analysis?*

Regression analysis provide us the easiest technique to compare the effects of variables measured on different range, such as the effect of salary changes and the number of upcoming, promotional activities. These benefits help market researchers', data analysts' and data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.

As already discussed above, regression analysis helps to estimate the relationship between dependent and independent variables. Let's understand this with an easy example:

Suppose we want to estimate the growth in sales of a company based on current economic conditions of our country. The recent company data available with us speaks that the growth in sales is around two and a half times the growth in the economy.

Using the regression insight, we can easily predict future sales of the company based on present & past information. There are multiple benefits of using regression analysis. They are as such as it provides a prediction by indicating the significant relationships between dependent variable and independent variable and depicting the strength of impact of multiple independent variables on a dependent variable.

Now, moving on with the next important part on what are the Regularization Techniques in Machine Learning.

Regularization Techniques

There are mainly two types of regularization techniques, namely Ridge Regression and Lasso Regression. The way they assign a penalty to β (coefficients) is what differentiates them from each other.

Ridge Regression (L2 Regularization)

This technique performs L2 regularization. The main algorithm behind this is to modify the RSS by adding the penalty which is equivalent to the square of the magnitude of coefficients. However, it is considered to be a technique used when the info suffers from multicollinearity (independent variables are highly correlated). In multicollinearity, albeit the smallest amount squares estimates (OLS) are unbiased, their variances are large which deviates the observed value faraway from truth value. By adding a degree of bias to the regression estimates, ridge regression reduces the quality errors. It tends to solve the multicollinearity problem through shrinkage parameter λ. Now, let us have a look at the equation below.

$$= \underset{\beta \in \mathrm{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}}$$

In this equation, we have two components. The foremost one denotes the least square term and later one is lambda of the summation of β2 (beta- square) where β is the coefficient. This is added to least square term so as to shrink the parameter to possess a really low variance.

Every technique has some pros and cons, so as Ridge regression. It decreases the complexity of a model but does not reduce the number of variables since it never leads to a coefficient tending to zero rather only minimizes it. Hence, this model is not a good fit for feature reduction.

Lasso Regression (L1 Regularization)

This regularization technique performs L1 regularization. Unlike Ridge Regression, it modifies the RSS by adding the penalty (shrinkage quantity) equivalent to the sum of the absolute value of coefficients.

Looking at the equation below, we can observe that similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition to this, it is quite capable of reducing the variability and improving the accuracy of linear regression models.

$$= \underset{\beta \in \mathrm{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_1}_{\text{Penalty}}$$

Limitation of Lasso Regression:

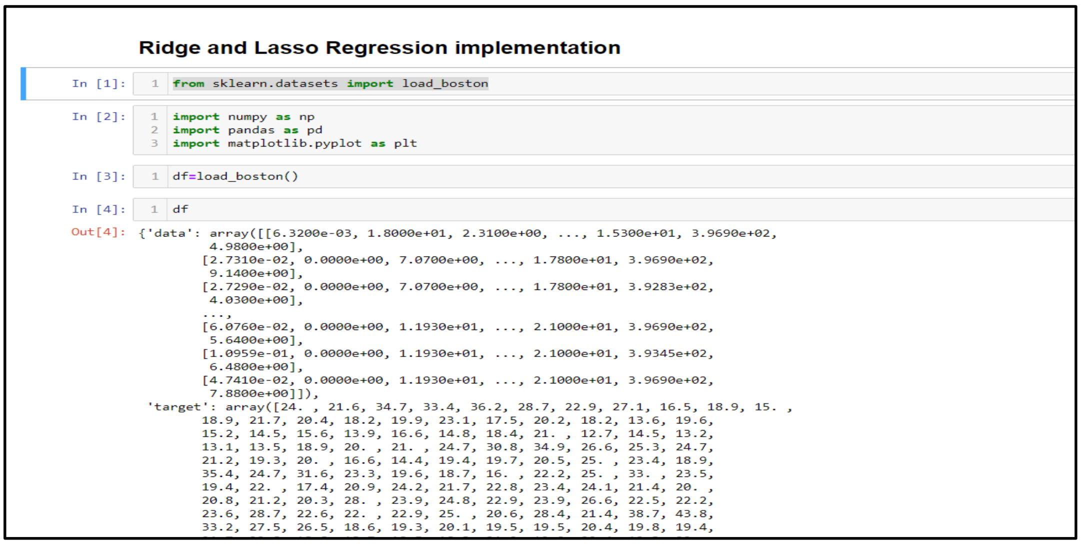
- If the number of predictors (p) is greater than the number of observations (n), Lasso will pick at most n predictors as non-zero, even if all predictors are relevant (or may be used in the test set). In such cases, Lasso sometimes really has to struggle with such types of data.
- If there are two or more highly collinear variables, then LASSO regression select one of them randomly which is not good for the interpretation of data.

Lasso regression differs from ridge regression in a way that it uses absolute values within the penalty function, rather than that of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. The more penalty is applied, the more the estimates get shrunk towards absolute zero. This helps to variable selection out of given range of n variables.

Practical Implementation using Python

Now, let's have a practical experience of ridge and lasso regression implementation in python programming language.

As like any other project, we import our usual libraries that will help us perform basic data manipulation and plotting. And then, we start off by importing our dataset and looking at its rows and columns.
The problem that we will try to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price.
At the very outmost, we will be importing the Boston dataset and display its data frame contents.



Ridge and Lasso Regression implementation

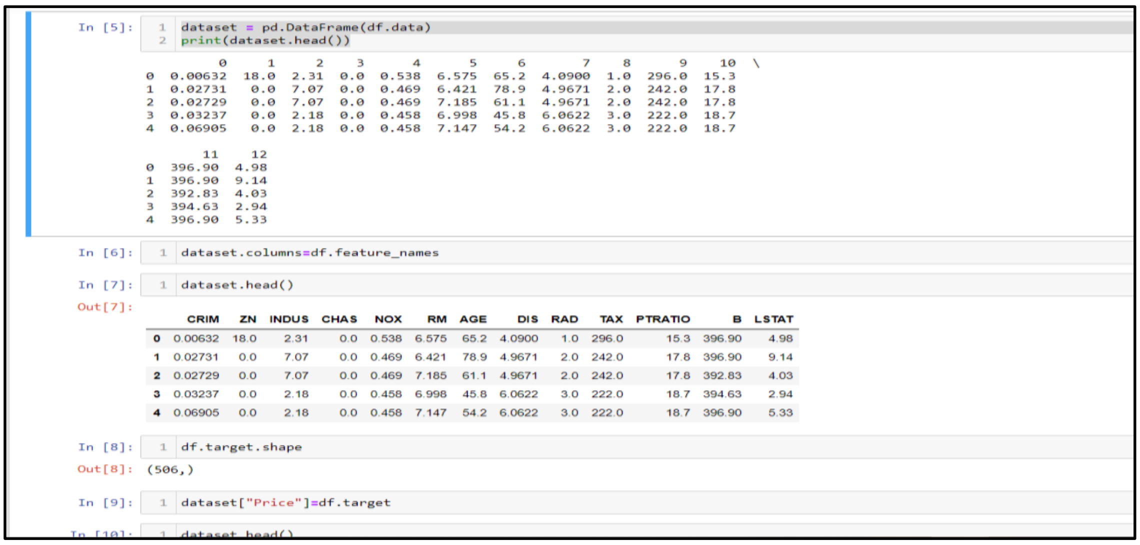
```
In [1]: from sklearn.datasets import load_boston
```

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [3]: df=load_boston()
```

```
In [4]: df
Out[4]: {'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
                 4.9800e+00],
                [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
                 9.1400e+00],
                [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
                 4.0300e+00],
                ...,
                [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
                 5.6400e+00],
                [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
                 6.4800e+00],
                [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
                 7.8800e+00]]),
         'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
                18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
                15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
                13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
                21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
                35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
                19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
                20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
                23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
                33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
```

The very next step is to visualize the data frame rows and columns which is known an Exploratory Data Analysis. The Boston data frame has 506 rows and 14 columns containing the columns: crim, zn, indus, chas, nox, rm, age, dis, rad, tax, ptratio, black, lstat and medv.



```
In [5]:  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 [6]:  dataset.columns=df.feature_names

In [7]:  dataset.head()
Out[7]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	4.98
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	9.14
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	4.03
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	2.94
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	5.33

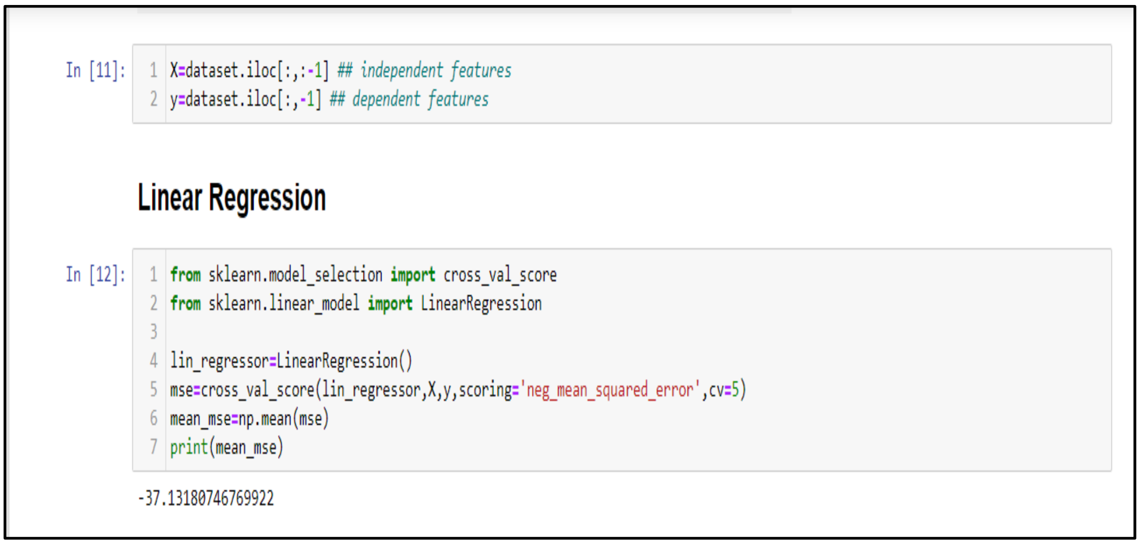
```
In [8]:  df.target.shape
Out[8]:  (506,)

In [9]:  dataset["Price"]=df.target

In [10]: dataset.head()
```

Now, we shall find the mean squared error using Linear Regression, Ridge Regression and Lasso Regression and try to find out which gives us the best result.

First, we import the Linear Regression and cross_val_score objects. The first one will allow us to fit a linear model, while the second object will perform k-fold cross-validation. Then, we define our features and target variable. The cross_val_score will return an array of MSE for each cross-validation steps. In our case, we have five of them. Therefore, we take the mean of MSE and print it. We are getting a negative MSE of -37.1318.



```
In [11]: X=dataset.iloc[:,:-1] ## independent features
         y=dataset.iloc[:,-1] ## dependent features
```

Linear Regression

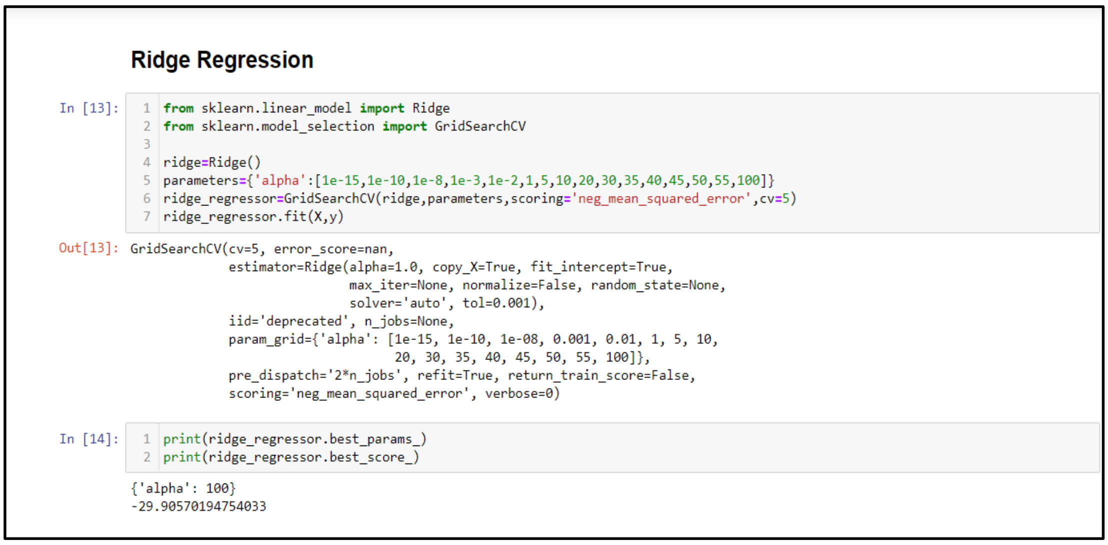
```
In [12]: from sklearn.model_selection import cross_val_score
         from sklearn.linear_model import LinearRegression

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

         -37.13180746769922
```

Now, let's see if ridge regression works better or lasso will be better. For ridge regression, we introduce GridSearchCV. This will allow us to automatically perform 5-fold cross-validation with a range of different regularization parameters in order to find the optimal value of alpha. You should see that the optimal value of alpha is 100, with a negative MSE of -29.90570. We can easily observe a slight improvement on comparing with the basic multiple linear regression.

The code looks like this:



Ridge Regression

```
In [13]:
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)
```

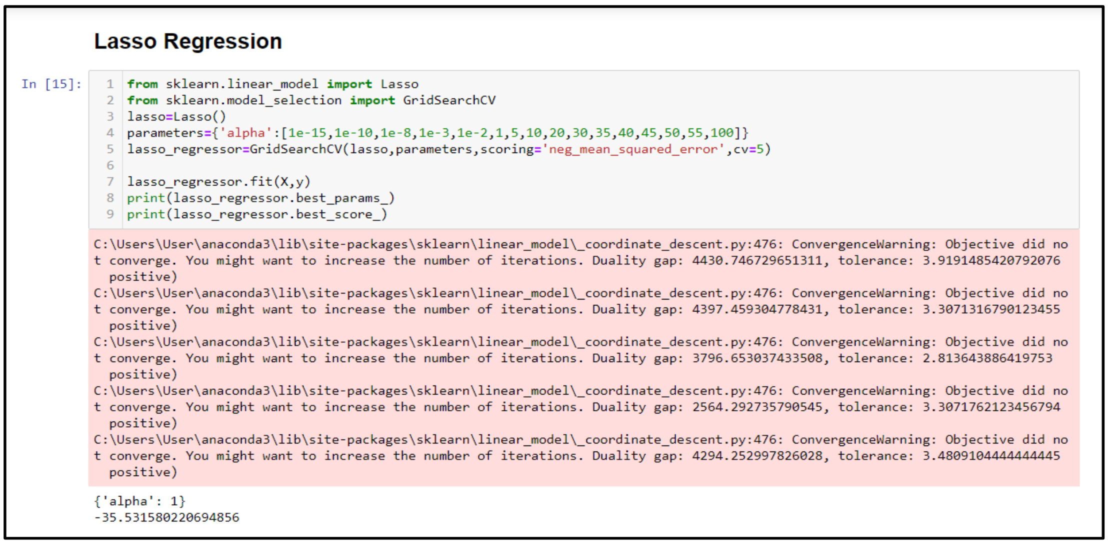
```
Out[13]: GridSearchCV(cv=5, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 10,
                                   20, 30, 35, 40, 45, 50, 55, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
In [14]:
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 100}
-29.90570194754033
```

Lasso

As ridge regression, the same process is followed for lasso. In this case, the optimal value for alpha is 1, and the negative MSE is -35.5315, which is the best score of all three models!



Lasso Regression

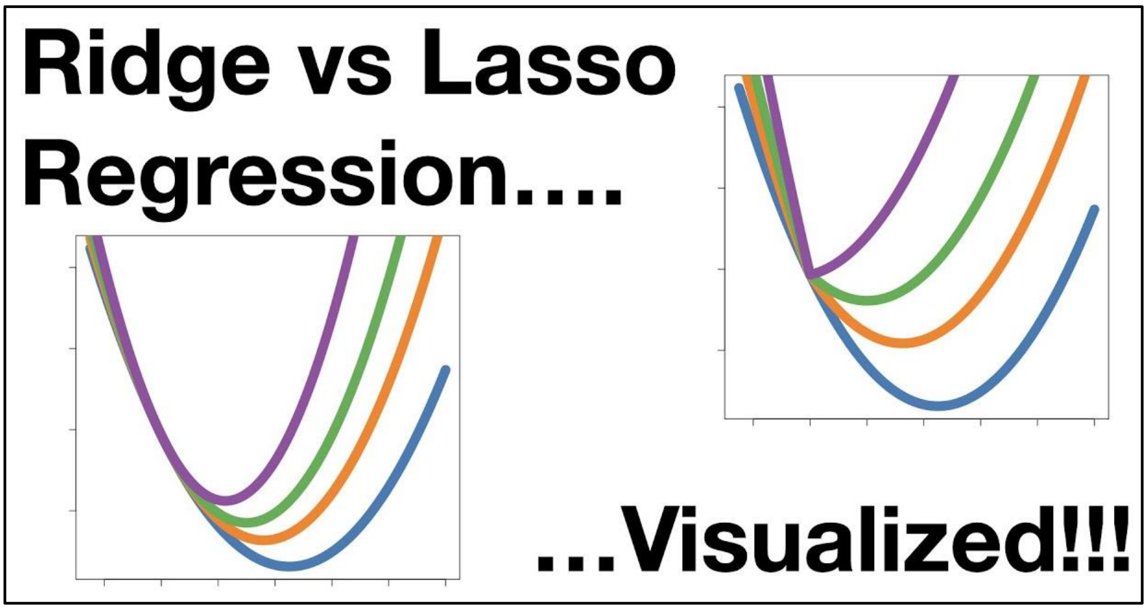
```
In [15]:
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso=Lasso()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
lasso_regressor=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=5)

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

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objective did no
t converge. You might want to increase the number of iterations. Duality gap: 4430.746729651311, tolerance: 3.919148542079207б
  positive)
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objective did no
t converge. You might want to increase the number of iterations. Duality gap: 4397.459304778431, tolerance: 3.3071316790123455
  positive)
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objective did no
t converge. You might want to increase the number of iterations. Duality gap: 3796.653037433508, tolerance: 2.813643886419753
  positive)
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objective did no
t converge. You might want to increase the number of iterations. Duality gap: 2564.292735790545, tolerance: 3.3071762123456794
  positive)
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: Objective did no
t converge. You might want to increase the number of iterations. Duality gap: 4294.252997826028, tolerance: 3.4809104444444445
  positive)
{'alpha': 1}
-35.531580220694856
```

The Conclusion…

Hope this articles has given you all a brief idea on Regularization, the types of techniques namely Ridge and Lasso Regression, their pros and cons and finally, implementation with the help of Python.



Featured Image