Lasso Regression

dhruv patel BDA 17162121014

Loading Libraries

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

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split,cross_val_score,GridSea

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso,Ridge

from sklearn.metrics import accuracy_score
```

In [3]:
```python
data = load_breast_cancer()  # Loading data
data
```

Out[3]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) d
ataset\n--------------------------------------------\n\n**Data Set Characterist
ics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 numeri
c, predictive attributes and the class\n\n    :Attribute Information:\n
- radius (mean of distances from center to points on the perimeter)\n        -
texture (standard deviation of gray-scale values)\n        - perimeter\n
- area\n        - smoothness (local variation in radius lengths)\n        - com
pactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave p
ortions of the contour)\n        - concave points (number of concave portions o
f the contour)\n        - symmetry\n        - fractal dimension ("coastline app
roximation" - 1)\n\n        The mean, standard error, and "worst" or largest (m
ean of the three\n        worst/largest values) of these features were computed
```

```
for each image,\n        resulting in 30 features.  For instance, field 0 is Me
an Radius, field\n        10 is Radius SE, field 20 is Worst Radius.\n\n
- class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n
:Summary Statistics:\n\n    ===================================== ====== ======
\n                                                  Min    Max\n    ==================
=================== ====== ======\n    radius (mean):                         6.
981  28.11\n    texture (mean):                        9.71   39.28\n    perimet
er (mean):                      43.79  188.5\n    area (mean):
143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n    comp
actness (mean):                   0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                0.0    0.201\n    symme
try (mean):                       0.106  0.304\n    fractal dimension (mean):
0.05   0.097\n    radius (standard error):              0.112  2.873\n    textu
re (standard error):              0.36   4.885\n    perimeter (standard error):
0.757  21.98\n    area (standard error):                6.802  542.2\n    smoot
hness (standard error):           0.002  0.031\n    compactness (standard erro
r):         0.002  0.135\n    concavity (standard error):           0.0    0.39
6\n    concave points (standard error):      0.0    0.053\n    symmetry (standa
rd error):           0.008  0.079\n    fractal dimension (standard error):
0.001  0.03\n    radius (worst):                        7.93   36.04\n    textur
e (worst):                      12.02  49.54\n    perimeter (worst):
50.41  251.2\n    area (worst):                         185.2  4254.0\n    smoo
thness (worst):                   0.071  0.223\n    compactness (worst):
0.027  1.058\n    concavity (worst):                    0.0    1.252\n    conca
ve points (worst):                0.0    0.291\n    symmetry (worst):
0.156  0.664\n    fractal dimension (worst):            0.055  0.208\n    =====
================================ ====== ======\n\n    :Missing Attribute Value
s: None\n\n    :Class Distribution: 212 - Malignant, 357 - Benign\n\n    :Creat
or:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor:
Nick Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Breast Can
cer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are com
puted from a digitized image of a fine needle\naspirate (FNA) of a breast mass.
They describe\ncharacteristics of the cell nuclei present in the image.\n\nSepa
rating plane described above was obtained using\nMultisurface Method-Tree (MSM-
T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Procee
dings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Societ
y,\npp. 97-101, 1992], a classification method which uses linear\nprogramming t
o construct a decision tree.  Relevant features\nwere selected using an exhaust
ive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actu
al linear program used to obtain the separating plane\nin the 3-dimensional spa
ce is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear
\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization M
ethods and Software 1, 1992, 23-34].\n\nThis database is also available through
the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-
learn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L.
Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS&
T/SPIE 1993 International Symposium on \n     Electronic Imaging: Science and T
echnology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Man
gasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n     prog
nosis via linear programming. Operations Research, 43(4), pages 570-577, \n
July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machin
e learning techniques\n     to diagnose breast cancer from fine-needle aspirate
s. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean
area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
```

```
           'radius error', 'texture error', 'perimeter error', 'area error',
           'smoothness error', 'compactness error', 'concavity error',
           'concave points error', 'symmetry error',
           'fractal dimension error', 'worst radius', 'worst texture',
           'worst perimeter', 'worst area', 'worst smoothness',
           'worst compactness', 'worst concavity', 'worst concave points',
           'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'C:\\Users\\dhruv\\anaconda3\\envs\\ml\\lib\\site-packages\\sklear
n\\datasets\\data\\breast_cancer.csv'}
```

In [4]:

```
data.target_names
```

Out[4]:

```
array(['malignant', 'benign'], dtype='<U9')
```

In [5]:

```
data.feature_names
```

Out[5]:

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [6]:

```
df = pd.DataFrame(data.data,columns=data.feature_names)
```

In [7]:

```
df.head()
```

Out[7]:

mean :tness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wors smoothness
27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622
07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238
15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444
28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098
13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374

In [8]:
```
df.describe()
```

Out[8]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 30 columns

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [10]:
```
X = data.data
```

In [11]:
```
y = data.target
```

In [12]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_st
```

Building Models

In [13]:
```
lr = LinearRegression()
```

In [14]:
```
lr.fit(X_train,y_train)
```
Out[14]:
```
LinearRegression()
```

In [15]:
```
lr.score(X_test,y_test)  ## test score for linear regression
```
Out[15]:
```
0.739386998952047
```

In [16]:
```
lasso_1e1 = Lasso(alpha=0.001 , max_iter = 10e6)

```

In [17]:
```
lasso_1e1.fit(X_train,y_train)
```
Out[17]:
```
Lasso(alpha=0.001, max_iter=10000000.0)
```

In [19]:
```
lasso_1e1.score(X_test,y_test)  ## we can see that our score is quite low be
```
Out[19]:
```
0.7033245684677331
```

In [20]:
```
### trying to search best ALpha using GridSearchcv
```

In [21]:
```
lasso = Lasso()
param_vals = [1e-20,1e-10,1e-5,1e-1,1,10,100,20]
params = {'alpha':param_vals,"max_iter":[10e5]}

lassso_cv = GridSearchCV(lasso,params,scoring="neg_mean_squared_error",cv=5,

lassso_cv.fit(X_train,y_train)
```

```
C:\Users\dhruv\anaconda3\envs\ml\lib\site-packages\sklearn\linear_model\_coordi
nate_descent.py:529: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 7.5364956176210125, tol
erance: 0.007471698113207546
  model = cd_fast.enet_coordinate_descent(
C:\Users\dhruv\anaconda3\envs\ml\lib\site-packages\sklearn\linear_model\_coordi
nate_descent.py:529: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 8.030038076620464, tole
rance: 0.00742138364779874
  model = cd_fast.enet_coordinate_descent(
C:\Users\dhruv\anaconda3\envs\ml\lib\site-packages\sklearn\linear_model\_coordi
nate_descent.py:529: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 8.172295272619992, tole
rance: 0.007446855345911951
  model = cd_fast.enet_coordinate_descent(
C:\Users\dhruv\anaconda3\envs\ml\lib\site-packages\sklearn\linear_model\_coordi
nate_descent.py:529: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 8.426045348810645, tole
rance: 0.007485893416927903
  model = cd_fast.enet_coordinate_descent(
C:\Users\dhruv\anaconda3\envs\ml\lib\site-packages\sklearn\linear_model\_coordi
nate_descent.py:529: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 7.92288383145575, toler
ance: 0.007460815047021947
  model = cd_fast.enet_coordinate_descent(
```

Out[21]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [1e-20, 1e-10, 1e-05, 0.1, 1, 10, 100, 20],
                         'max_iter': [1000000.0]},
             scoring='neg_mean_squared_error')
```

In [22]:
```
lassso_cv.best_params_
```

Out[22]:
```
{'alpha': 1e-05, 'max_iter': 1000000.0}
```

In [23]:
```
lassso_cv.best_score_
```

Out[23]:
```
-0.062488634237653815
```

In [24]:
```
lassso_cv.best_estimator_
```

Out[24]:
```
Lasso(alpha=1e-05, max_iter=1000000.0)
```

In [25]:
```
lassso_cv.cv_results_
```

Out[25]:
```
{'mean_fit_time': array([3.58654497e+01, 3.28113751e+00, 1.23047647e+00, 4.6011
9247e-03,
        1.00030899e-03, 8.00228119e-04, 4.00066376e-04, 8.00132751e-04]),
 'std_fit_time': array([5.39236675e-01, 4.65809269e-01, 2.15030476e-01, 3.32364
010e-03,
        3.87384339e-07, 4.00114329e-04, 4.89979335e-04, 4.00066575e-04]),
 'mean_score_time': array([0.00020013, 0.00020003, 0.00060024, 0.00040002, 0.00
019999,
        0.00020013, 0.        , 0.        ]),
 'std_score_time': array([0.00040026, 0.00040007, 0.0004901 , 0.00048992, 0.000
39997,
        0.00040026, 0.        , 0.        ]),
 'param_alpha': masked_array(data=[1e-20, 1e-10, 1e-05, 0.1, 1, 10, 100, 20],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_iter': masked_array(data=[1000000.0, 1000000.0, 1000000.0, 1000000.
0, 1000000.0,
                    1000000.0, 1000000.0, 1000000.0],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'alpha': 1e-20, 'max_iter': 1000000.0},
  {'alpha': 1e-10, 'max_iter': 1000000.0},
  {'alpha': 1e-05, 'max_iter': 1000000.0},
  {'alpha': 0.1, 'max_iter': 1000000.0},
  {'alpha': 1, 'max_iter': 1000000.0},
  {'alpha': 10, 'max_iter': 1000000.0},
  {'alpha': 100, 'max_iter': 1000000.0},
  {'alpha': 20, 'max_iter': 1000000.0}],
 'split0_test_score': array([-0.07428915, -0.07428914, -0.07464274, -0.0856067
4, -0.10070146,
        -0.10542019, -0.12704672, -0.10528853]),
 'split1_test_score': array([-0.06303485, -0.06303483, -0.06197237, -0.0839421
7, -0.10919212,
        -0.10909901, -0.13063961, -0.10878617]),
 'split2_test_score': array([-0.06124805, -0.06124805, -0.0609032 , -0.0706723
6, -0.07527377,
        -0.08466295, -0.12563457, -0.08735557]),
 'split3_test_score': array([-0.05335686, -0.05335682, -0.05185672, -0.0667246
, -0.12056159,
        -0.11816192, -0.13079112, -0.11674674]),
 'split4_test_score': array([-0.06737639, -0.06737634, -0.06306814, -0.0741482
7, -0.10757998,
        -0.10998106, -0.14368361, -0.1124399 ]),
 'mean_test_score': array([-0.06386106, -0.06386104, -0.06248863, -0.07621883,
-0.10266178,
        -0.10546502, -0.13155913, -0.10612338]),
 'std_test_score': array([0.00691305, 0.00691305, 0.00726465, 0.00738882, 0.015
1075 ,
        0.01120303, 0.00638593, 0.01012692]),
 'rank_test_score': array([3, 2, 1, 4, 5, 6, 8, 7])}
```

In [35]:
```
lasso_best_estimator = Lasso(alpha=1e-05,max_iter=1000000.0)

```

In [37]:
```
lasso_best_estimator.fit(X_train,y_train)
```

Out[37]:
```
Lasso(alpha=1e-05, max_iter=1000000.0)
```

In [38]:
```
lasso_best_estimator.score(X_test,y_test)
```

Out[38]:
```
0.7438235254978831
```

In [29]:
```
lasso_1e1.score(X_test,y_test)
```

Out[29]:
```
0.7033245684677331
```

thus after using gridsearchcv we found the best hyperparameters for our Lasso Reg and got a score more than LinearRegression

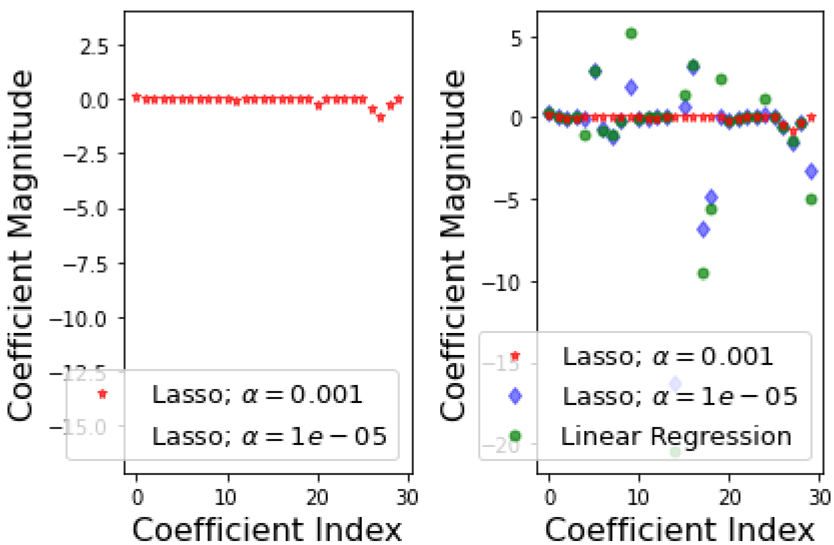
In [45]:

```
plt.subplot(1,2,1)
plt.plot(lasso_1e1.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,
plt.plot(lasso_best_estimator.coef_,alpha=1e-05,linestyle='none',marker='d',

plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)

plt.subplot(1,2,2)
plt.plot(lasso_1e1.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,
plt.plot(lasso_best_estimator.coef_,alpha=0.5,linestyle='none',marker='d',ma

plt.plot(lr.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color='
plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.tight_layout()
plt.show()
```



thus we successfully demonstrated how we can improve the accuracy of our model using Normalization techniques like Lasso Regression