Linear regression, Lasso, Ridge, and ElasticNet Regression

@Yuanyuan Tang

Contents

References

https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

https://towardsdatascience.com/whats-the-difference-between-linear-regression-lasso-ridge-and-elasticnet-8f997c60cf29

https://www.datacamp.com/tutorial/tutorial-lasso-ridge-regression

https://www.geeksforgeeks.org/python-seaborn-pairplot-method/

1. Linear regression

Suppose you are given a set of data $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, where x_i is a m-dimensional vector and y_i is a scalar. Our goal is to estimate a parameter w with

$$y = x^T w + b$$

Such that we can minimize

$$L(w,D) = \sum_{i=1}^{n} (y_i - x_i^T w - b)^2 = \sum_{i=1}^{n} (y_i - \tilde{x}_i^T \tilde{w})$$

$$\hat{x}_i^T = [x_i^T, 1], \quad \hat{w} = \begin{bmatrix} w \\ b \end{bmatrix}$$

$$L(w,D) = \min_{w} \| y - \tilde{X}^T \tilde{w} \|_2^2$$

$$\frac{\partial L(w,D)}{\partial w} = 2\tilde{X}(y - \tilde{X}^T \tilde{w}) = 0$$

$$\Rightarrow \tilde{w} = (\tilde{X}\tilde{X}^T)^{-1} \tilde{X} y$$

Where he cost function is the mean square error defined as

$$\sum_{i=1}^{n} (y_i - h(x_i))^2$$

Solution: Based on the derivation, it requires that X should be nonsingular. If X is not singular or we want to reduce the computational complexity of inverse, we may use gradient methods, where

$$W^{t+1} = W^{t} - \alpha \frac{\partial L(W^{\phi}, D)}{\partial W}\Big|_{W=t}$$

Possible issues: When it comes to training models, there are two major problems one can encounter: overfitting and underfitting.

- **Overfitting**: happens when the model performs well on the training set but not so well on unseen (test) data.
- **Underfitting**: happens when it neither performs well on the train set nor on the test set.

In particular, during the training stage, if the model feels like one particular feature is particularly important, the model may place a large weight to the feature. This sometimes leads to overfitting in small datasets.

Regularization is a technique used in feature selection to avoid overfitting of the data, especially when there is a large variance between train and test set performances. With regularization, the number of features used in training is kept constant, yet the magnitude of the coefficients (w) is reduced.

2. Lasso regression: L1 Regularization

The cost function of Lasso regression is:

$$\sum_{i=1}^{n} (y_i - w_i \tilde{x}_i)^2 + \lambda \sum_{j=1}^{m} |w_i|$$

λ is the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation. With λ set to zero, you will find that this is the equivalent of the linear regression model. A larger value λ penalizes the optimization function. Therefore, lasso regression shrinks the coefficients and helps to reduce the model complexity and multi-collinearity.

Solution: The parameters will be obtained by gradient methods, where the gradient will be replaced by sub-gradient.

$$W^{t+1} = W^{t} - \alpha \underbrace{\frac{\partial L(\overset{\phi}{W}, D)}{\partial W}}_{\to \text{Subgradient}} \Bigg|_{W=t}$$

Feature selection: Due to the fact that coefficients will be shrunk towards a mean of zero, less important features in a dataset are eliminated when penalized. The shrinkage of these coefficients based on the alpha value provided leads to some form of automatic feature selection, as input variables are removed in an effective approach.

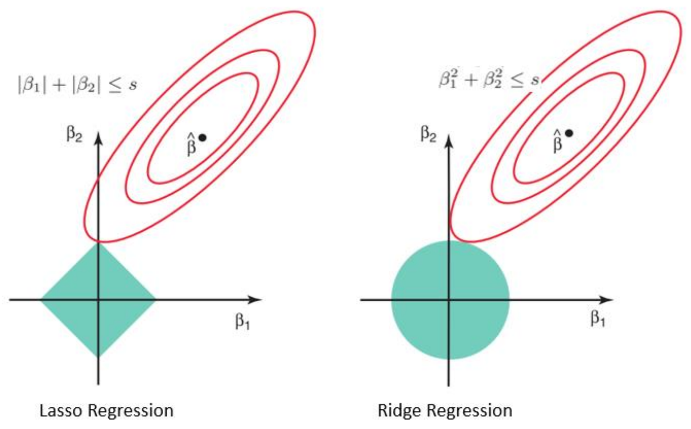
3. Ridge regression: L2 Regularization

The cost function of Ridge regression is shown below, where the constraint is over $l2$-norm of coefficients w.

$$\sum_{i=1}^{n} (y_i - w_i \tilde{x}_i)^2 + \lambda \sum_{j=1}^{m} W_i^2$$

Property: the weights not only tend to have smaller absolute values, but also really tend to penalize the extremes of the weights, resulting in a group of weights that are more evenly distributed.

4. Question: why cannot Ridge regression selection features?



Considering the geometry of both the lasso (left) and ridge (right) models, the elliptical contours (red circles) are the cost functions for each. Relaxing the constraints introduced by the penalty factor leads to an increase in the constrained region (diamond, circle). Doing this continually, we will hit the center of the ellipse, where the results of both lasso and ridge models are similar to a linear regression model.

However, both methods determine coefficients by finding the first point where the elliptical contours hit the region of constraints. Since lasso regression takes a diamond shape in the plot for the constrained region, each time the elliptical regions intersect with these corners, at least one of the coefficients becomes zero. This is impossible in the ridge regression model as it forms a circular shape and therefore values can be shrunk close to zero, but never equal to zero.

5. ElasticNet regression-L1+L2-regularization

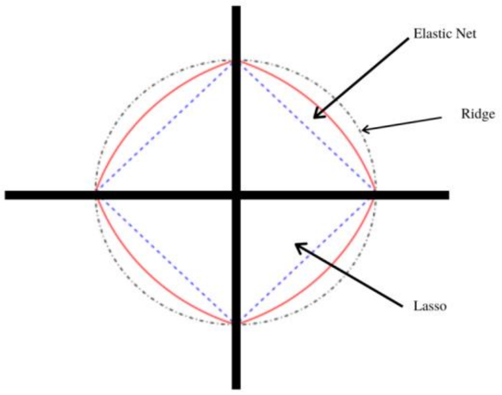
ElasticNet is a hybrid of Lasso and Ridge, where both the absolute value penalization and squared penalization are included, being regulated by:

$$L(w,D) = \sum_{i=1}^{n} (y_i - x_i w)^2 + \lambda_1 \sum_{j=1}^{m} |w_j| + \lambda_2 \sum_{j=1}^{m} w_j^2$$

The function in skit_learner uses the following function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

Where the geometrical representation of the constraint is shown in the following figure.



6. Scale data

Since different features of datasets may have different range. Therefore, it is better to scale or normalize data before applying them.

7. Application scenarios

(1) sklearn's algorithm cheat sheet suggests you to try Lasso, ElasticNet, or Ridge when you data-set is smaller than 100k rows. Otherwise, try SGDRegressor.

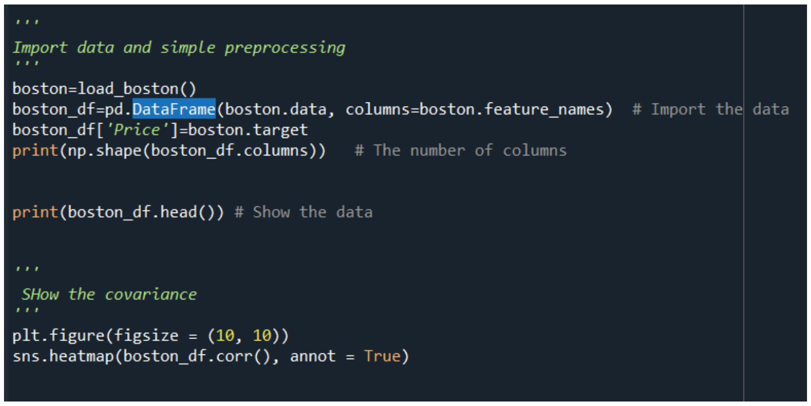
(2) Lasso and ElasticNet tend to give sparse weights (most zeros), because the l1 regularization cares equally about driving down big weights to small weights, or driving small weights to zeros. If you have a lot of predictors (features), and you suspect that not all of them are that important, Lasso and ElasticNet may be really good idea to start with.

(3) Ridge tends to give small but well distributed weights, because the l2 regularization cares more about driving big weight to small weights, instead of driving small weights to zeros. If you only have a few predictors, and you are confident that all of them should be really relevant for predictions, try Ridge as a good regularized linear regression method.

(4) You will need to scale your data before using these regularized linear regression methods. Use StandardScaler first, or set 'normalize' in these estimators to 'True'.

8. Examples

- Import data into Pandas

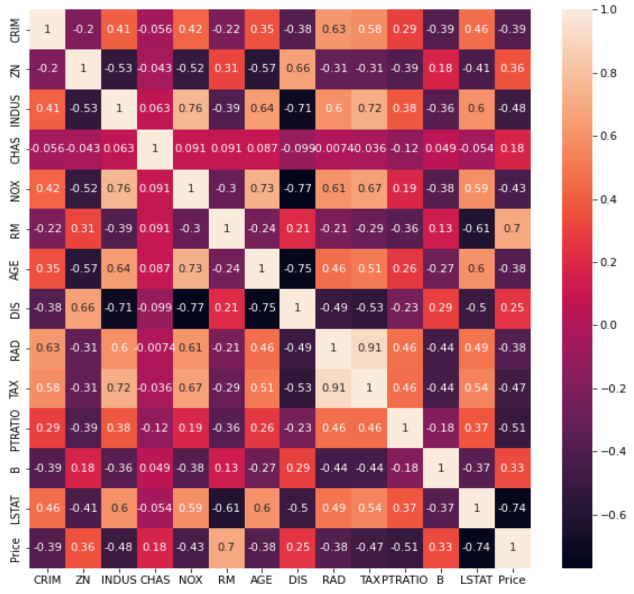


```
'''
Import data and simple preprocessing
'''
boston=load_boston()
boston_df=pd.DataFrame(boston.data, columns=boston.feature_names)  # Import the data
boston_df['Price']=boston.target
print(np.shape(boston_df.columns))   # The number of columns


print(boston_df.head()) # Show the data


'''
 SHow the covariance
'''
plt.figure(figsize = (10, 10))
sns.heatmap(boston_df.corr(), annot = True)
```

1) **Analyzing relationship among features: correlation and co-distribution**

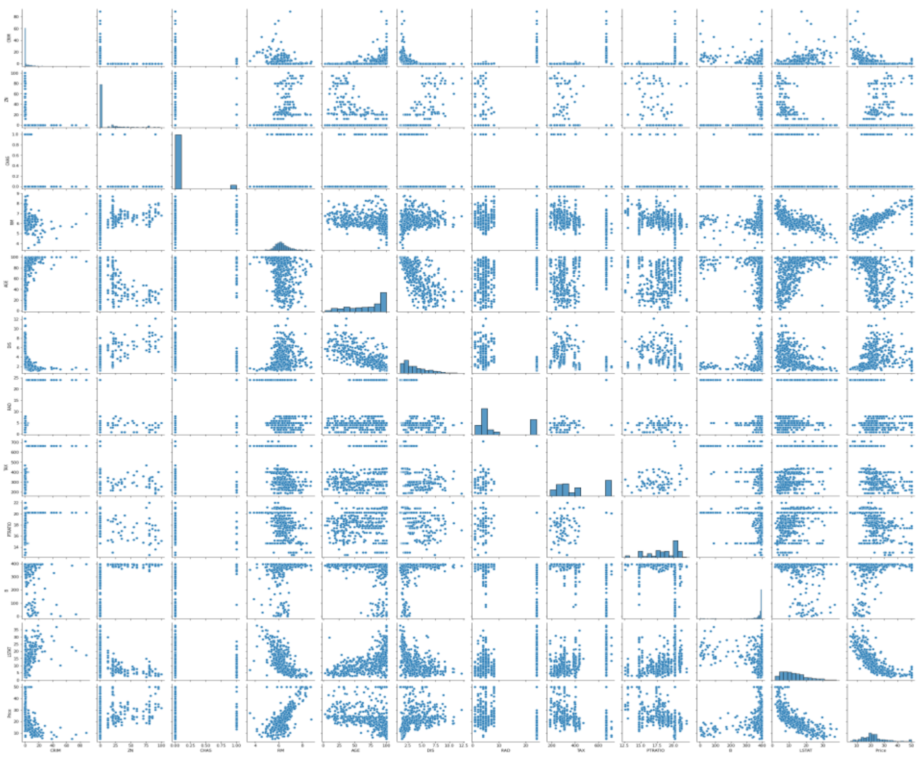
- df.corr(): Analyzing the correlation among different features, which can be used to select suitable features

$$r = \frac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{Var}(x) \cdot \mathrm{Var}(y)}}$$



- **seaborn.pairplot():** To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function. By this method, in particular, you can analyze the relationship between the features and the labels.
  This shows the relationship for (n, 2) combination of variables in a DataFrame as a matrix of plots and the diagonal plots are the univariate plots.
  seaborn.pairplot(data, **kwargs)
  Seaborn.pairplot uses many arguments as input, main of which are described below in form of table:
  https://www.geeksforgeeks.org/python-seaborn-pairplot-method/

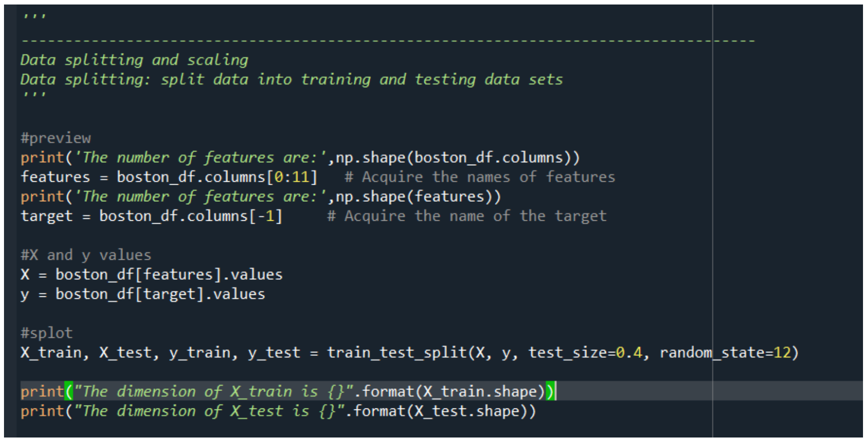
data	Tidy (long-form) dataframe where each column is a variable and each row is an observation.	DataFrame
hue	Variable in "data" to map plot aspects to different colors.	string (variable name), optional
palette	Set of colors for mapping the "hue" variable. If a dict, keys should be values in the "hue" variable. vars : list of variable names, optional	dict or seaborn color palette
{x, y}_vars	Variables within "data" to use separately for the rows and columns of the figure; i.e. to make a non-square plot.	lists of variable names, optional
dropna	Drop missing values from the data before plotting.	boolean, optional



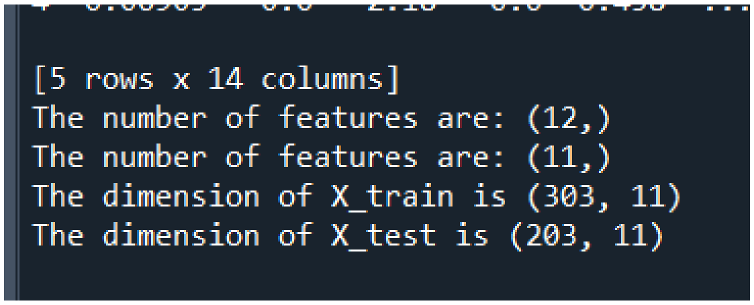
9. Splitting datasets: training and test

- **sklearn.model_selection.train_test_split() function**: The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X_test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model.
  Parameters of the function:

- ***arrays: sequence of indexables.** Lists, numpy arrays, scipy-sparse matrices, and pandas dataframes are all valid inputs.*
- ***test_size: int or float, by default None.** If float, it should be between 0.0 and 1.0 and represent the percentage of the dataset to test split. If int is used, it refers to the total number of test samples. If the value is None, the complement of the train size is used. It will be set to 0.25 if train size is also None.*
- ***train_size: int or float, by default None.***
- ***random_state : int,by default None.** Controls how the data is shuffled before the split is implemented. For repeatable output across several function calls, pass an int.*
- ***shuffle: boolean object , by default True.** Whether or not the data should be shuffled before splitting. Stratify must be None if shuffle=False.*
- ***stratify: array-like object , by default it is None**. If None is selected, the data is stratified using these as class labels.*



```python
'''
--------------------------------------------------------------------------------------
Data splitting and scaling
Data splitting: split data into training and testing data sets
'''

#preview
print('The number of features are:',np.shape(boston_df.columns))
features = boston_df.columns[0:11]   # Acquire the names of features
print('The number of features are:',np.shape(features))
target = boston_df.columns[-1]     # Acquire the name of the target

#X and y values
X = boston_df[features].values
y = boston_df[target].values

#splot
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=12)

print("The dimension of X_train is {}".format(X_train.shape))
print("The dimension of X_test is {}".format(X_test.shape))
```



```
[5 rows x 14 columns]
The number of features are: (12,)
The number of features are: (11,)
The dimension of X_train is (303, 11)
The dimension of X_test is (203, 11)
```

10. Scaling data: standard distribution & Maxmin

- **StandardScaler():** scale features based on a standard distribution. The main parameters are shown below.

- The **fit(data)** method is used to compute the mean and std dev for a given feature so that it can be used further for scaling.
- The **transform(data)** method is used to perform scaling using mean and std dev calculated using the .fit() method.
- The **fit_transform()** method does both fit and transform.

Syntax: *class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)*

Parameters:

- ***copy:*** *If False, inplace scaling is done. If True , copy is created instead of inplace scaling.*
- ***with_mean:*** *If True, data is centered before scaling.*
- ***with_std:*** *If True, data is scaled to unit variance.*

Standardization:

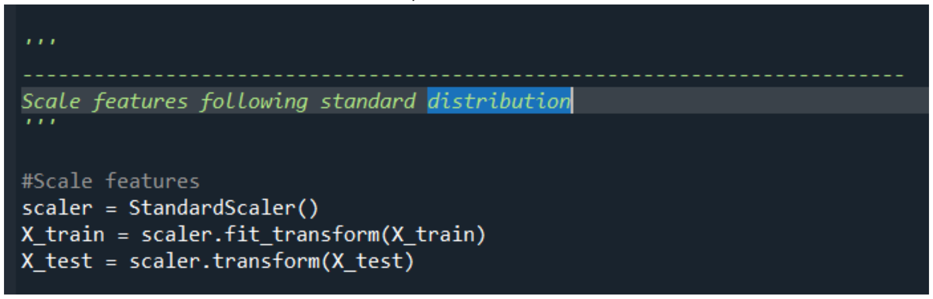
$$z = \frac{x-\mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$



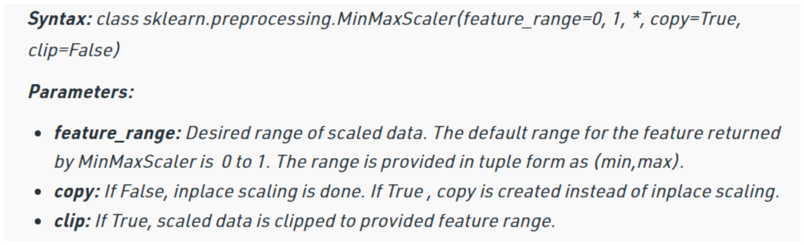
```
'''
---------------------------------------------------------------------------
Scale features following standard distribution
'''

#Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

MinMax Scaler(): shrinks the data within the given range, usually of 0 to 1. The formal definition is shown below.

x_std = (x – x.min(axis=0)) / (x.max(axis=0) – x.min(axis=0))

*x_scaled = x_std * (max – min) + min*

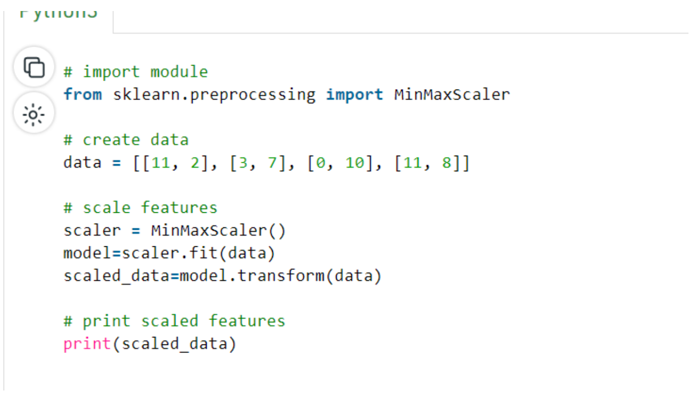
With parameters:



Syntax: *class sklearn.preprocessing.MinMaxScaler(feature_range=0, 1, *, copy=True, clip=False)*

Parameters:

- ***feature_range:*** *Desired range of scaled data. The default range for the feature returned by MinMaxScaler is 0 to 1. The range is provided in tuple form as (min,max).*
- ***copy:*** *If False, inplace scaling is done. If True , copy is created instead of inplace scaling.*
- ***clip:*** *If True, scaled data is clipped to provided feature range.*



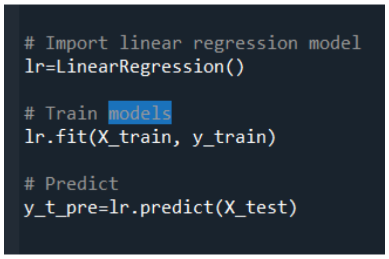
```
# import module
from sklearn.preprocessing import MinMaxScaler

# create data
data = [[11, 2], [3, 7], [0, 10], [11, 8]]

# scale features
scaler = MinMaxScaler()
model=scaler.fit(data)
scaled_data=model.transform(data)

# print scaled features
print(scaled_data)
```

Many machine learning algorithms like Gradient descent methods, KNN algorithm, linear and logistic regression, etc. require data scaling to produce good results.
https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/

11. Model training and evaluation

- **Training model**: contains three steps

a) Import model
b) training model:
c) predict values



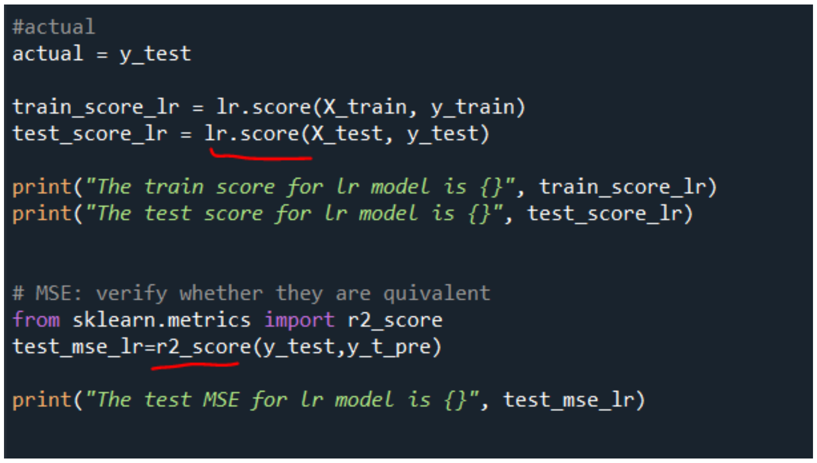
```
# Import linear regression model
lr=LinearRegression()

# Train models
lr.fit(X_train, y_train)

# Predict
y_t_pre=lr.predict(X_test)
```

- **Evaluating models**: https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

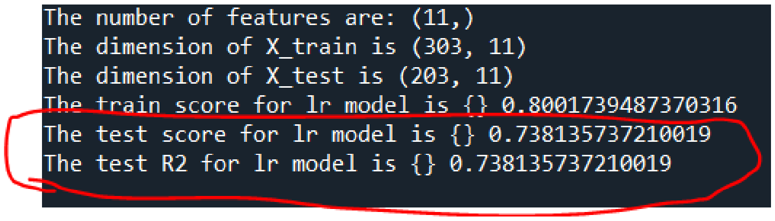
R-Square: It determines how much of the total variation in Y (dependent variable) is explained by the variation in X (independent variable). Mathematically, it can be written as:

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}, \\ = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}.$$

Where $\bar{y} = \frac{1}{n}\sum_i y_i$ is the mean of true label values, not predict values.



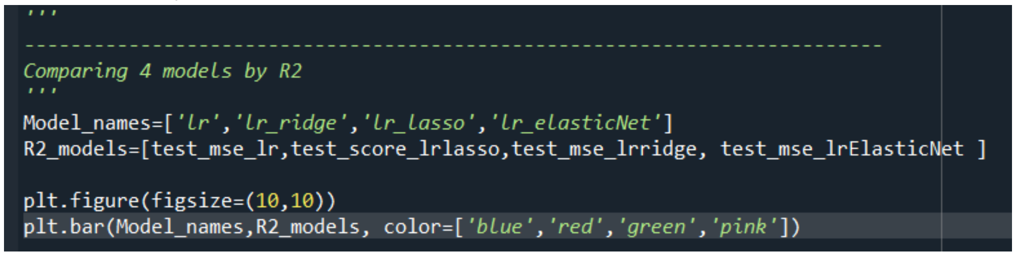
```
#actual
actual = y_test

train_score_lr = lr.score(X_train, y_train)
test_score_lr = lr.score(X_test, y_test)

print("The train score for lr model is {}", train_score_lr)
print("The test score for lr model is {}", test_score_lr)


# MSE: verify whether they are quivalent
from sklearn.metrics import r2_score
test_mse_lr=r2_score(y_test,y_t_pre)

print("The test MSE for lr model is {}", test_mse_lr)
```



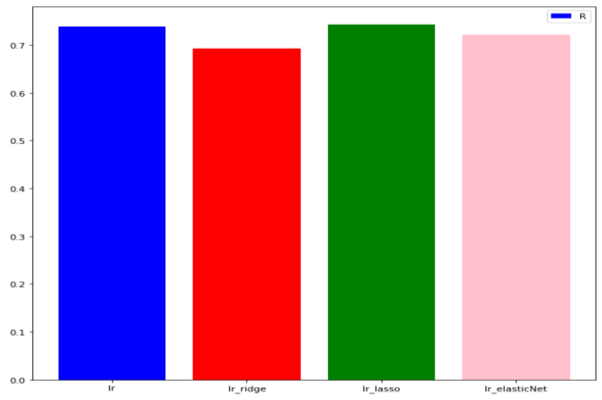
```
The number of features are: (11,)
The dimension of X_train is (303, 11)
The dimension of X_test is (203, 11)
The train score for lr model is {} 0.8001739487370316
The test score for lr model is {} 0.738135737210019
The test R2 for lr model is {} 0.738135737210019
```

12. Comparison of different models

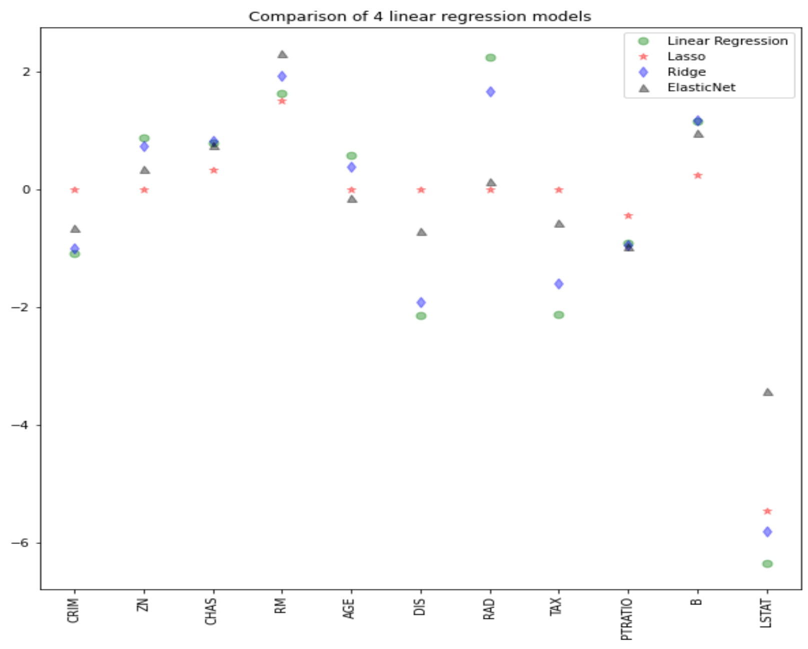
- Comparison of R-sequre



```
'''
--------------------------------------------------------------------------
Comparing 4 models by R2
'''
Model_names=['lr','lr_ridge','lr_lasso','lr_elasticNet']
R2_models=[test_mse_lr,test_score_lrlasso,test_mse_lrridge, test_mse_lrElasticNet ]

plt.figure(figsize=(10,10))
plt.bar(Model_names,R2_models, color=['blue','red','green','pink'])
```



- Comparison of coefficients



2) Comparison of different models

Search suitable parameters in a range: by using lassoCV, RidgeCV, ElasticCV functions.

13. Conclusion:

- Get familiar with multiple functions including **correlation**, **dual-distribution**, **rescaling**, **dataset splitting**, **training models**, and **R-square evaluation**.
- Data for linear regression should be rescaled before being applied training
- Linear regression is simple, but may suffer overfitting
- Using regularizations can restrict the weight to specific features, including l1-norm penalty, l2-norm penalty, and hybrid of both l1 and l2 penalties
- Lasso can achieve feature selections but has a smaller R-square value.
- A larger regularization coefficient leads to higher penalty on weights.