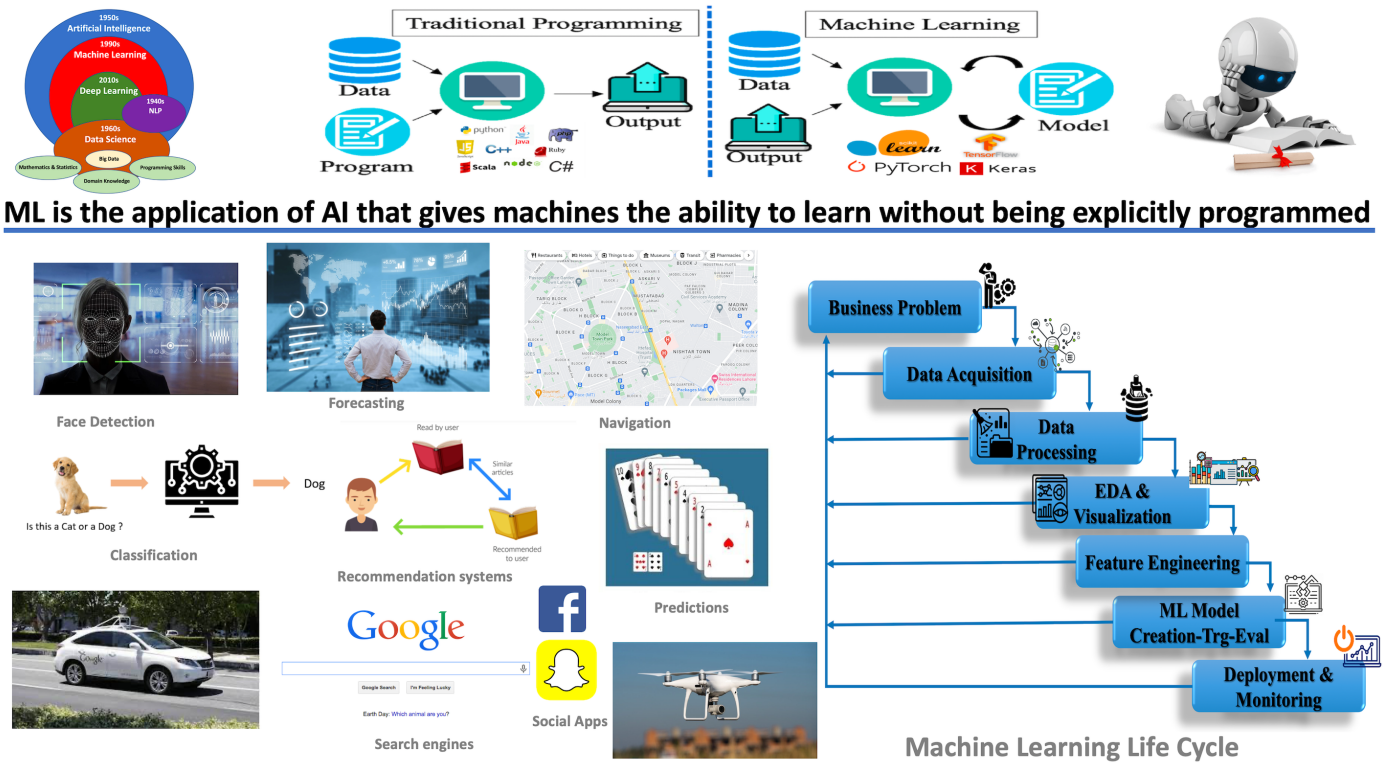
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.13 (Regularization: Ridge, Lasso and ElasticNet)

Open in Colab (https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)

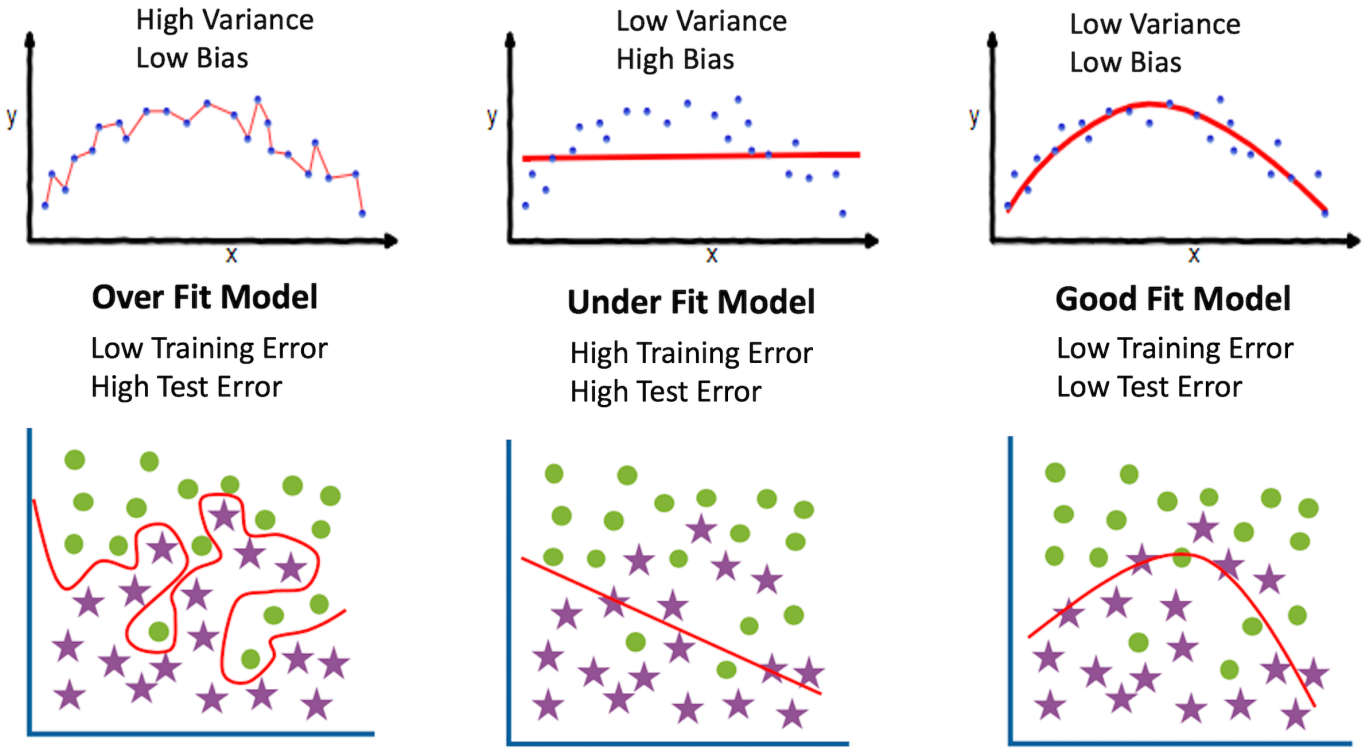


Learning agenda of this notebook

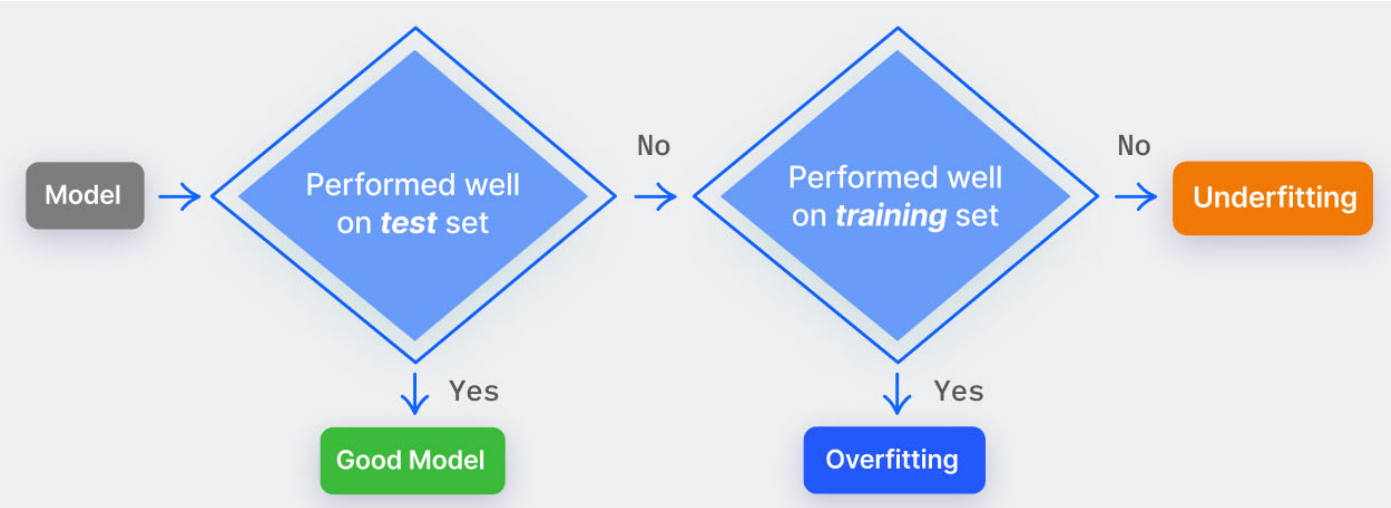
- Overfit vs Underfit vs Good Fit (A Recap)
  - What is Overfitting-Underfitting?
  - How to Detect Overfitting-Underfitting
  - Bias-Variance Tradeoff

1. Overfit vs Underfit vs Good Fit (A Recap)

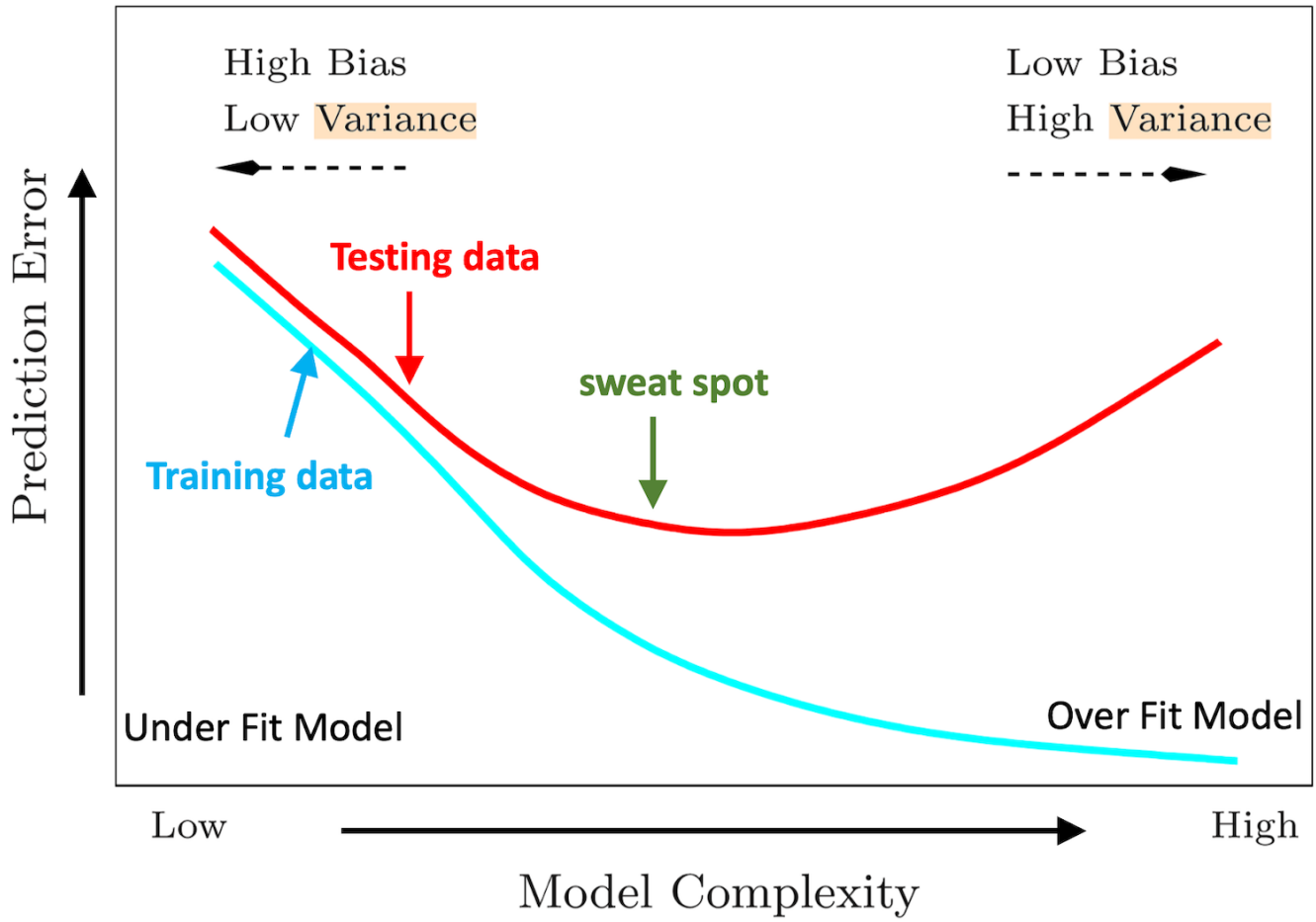
a. What is Overfitting-Underfitting?



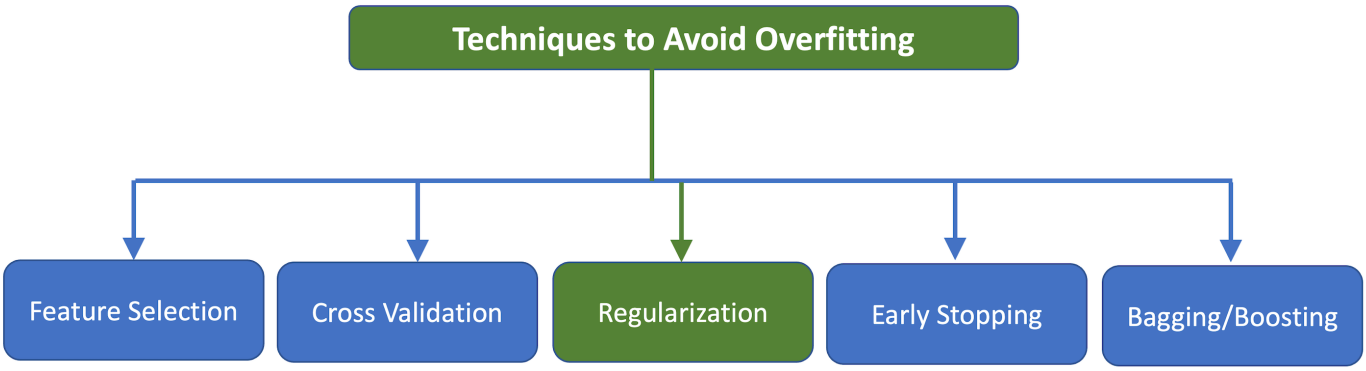
b. How to Detect Overfitting-Underfitting



c. Bias-Variance Tradeoff



d. How to Prevent Overfitting?

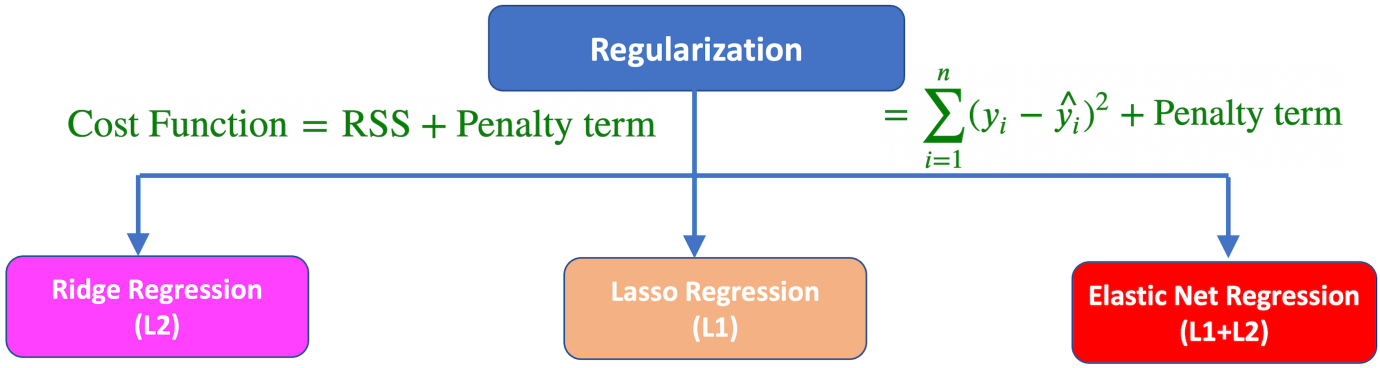


- **Feature Selection:** Keep only those features in your dataset which are correlated with output column and not with each other. Two ways to proceed are Forward selection and Backward selection.
- **Early Stopping:** Used when you are training a learning algorithm iteratively and is most commonly used in Deep Learning. However, if you stop the training/learning process too early may lead to underfitting as well.
- **Cross Validation:** Generate multiple train-test splits from your training data and tune hyperparameters of your model using techniques like GridSearch and RandomizedSearch CV.
- **Bagging/Boosting:** Bagging and Boosting are ensemble techniques that combine prediction using multiple estimators. Bagging is an acronym for Boot Strap Aggregation, which attempts to reduce the chance of overfitting of complex models. Example is Random Forrest Classifier. Boosting attempts to train large number of weak learners in sequence. Examples are AdaBoost and XGBoost.
- **Regularization:**

2. Regularization

Regularization techniques are used to calibrate the linear regression models in order to minimize the adjusted loss function and prevent overfitting and underfitting

a. Regularization Techniques



$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \times \sum_{i=1}^{m} \beta_i{}^2$$

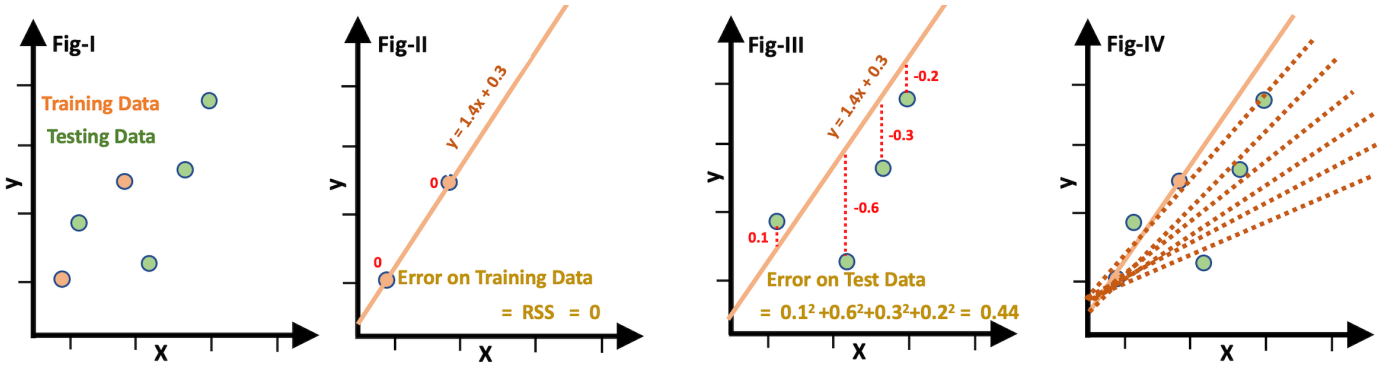
$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \times \sum_{i=1}^{m} |\beta_i|$$

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \times \sum_{i=1}^{m} \beta_i{}^2 + \lambda_2 \times \sum_{i=1}^{m} |\beta_i|$$

- **How to find the coefficients for the minimum value of RSS error function?**

  - Solve the model parameters analytically and arrive at a closed form solution by setting the gradient of loss function equal to zero and solving the resultant equation for the coefficient vector β.
  - Use Gradient Descent, which is an iterative optimization algorithm used to find the local minima of a differentiable function. The main idea is to take repeated steps in the opposite direction of the gradient of the function at the current point.

b. Regularization for 2-D Data



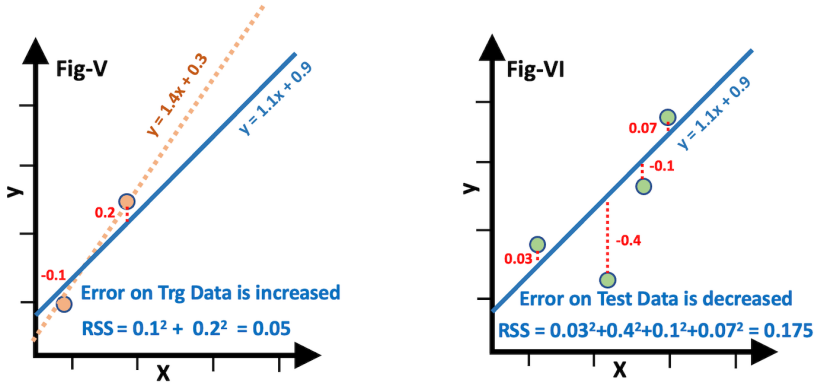
$$E(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \text{Penalty Term}$$

Ridge Penalty Term = $\lambda \beta_1^2$

Lasso Penalty Term = $\lambda|\beta_1|$

EN Penalty Term = $\lambda_1|\beta_1| + \lambda_2 \beta_1^2$

The value of λ can range from zero to infinity



3. Cost Functions for Ridge, Lasso, and ElasticNet Regression

a. Cost Function for Standard Linear Regression

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2$$

- From the lecture of Linear Regression:

$$\begin{aligned} y &= Xb \\ X^T y &= X^T Xb \\ (X^T X)^{-1} X^T y &= (X^T X)^{-1}(X^T X)b \\ (X^T X)^{-1} X^T y &= Ib \\ b &= (X^T X)^{-1} X^T Y \end{aligned}$$

$$\beta = (X^T X)^{-1} X^T Y$$

b. Cost Function for Ridge Regression (L2)

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \text{Penalty Term}$$

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \times \sum_{i=1}^{m} {\beta_i}^2$$

- The penalty term $\lambda \times \sum_{i=1}^{m} {\beta_i}^2$ maintains a trade-off between `fit of the model to the data` and `square of norm of the coefficients`.
  - If model is fitted poorly, the first term is large.
  - If coefficients have high values, the second term (penalty term) is large.
- Large λ penalizes coefficient values more.
- The cost function of Ridge Regression is still quadratic, and we can find closed form solution by taking gradient with respect to β, after gradient we have a solution of the ridge regression.

$$\beta = (X^T X + \lambda I)^{-1} X^T Y$$

- $\lambda = 0$, we have non-regularized solution.
- $\lambda = \infty$, the regression coefficients approaches to zero value but not exactly zero.
- `L2` regularization forces the weights to be small but does not make them zero and does non-sparse solution.
- `L2` is not robust to outliers as square terms blows up the error differences of the outliers and the regularization term tries to fix it by penalizing the weights
- Ridge regression performs better when all the input features influence the output and all with weights are of roughly equal size.

c. Cost Function for Lasso Regression (L1)

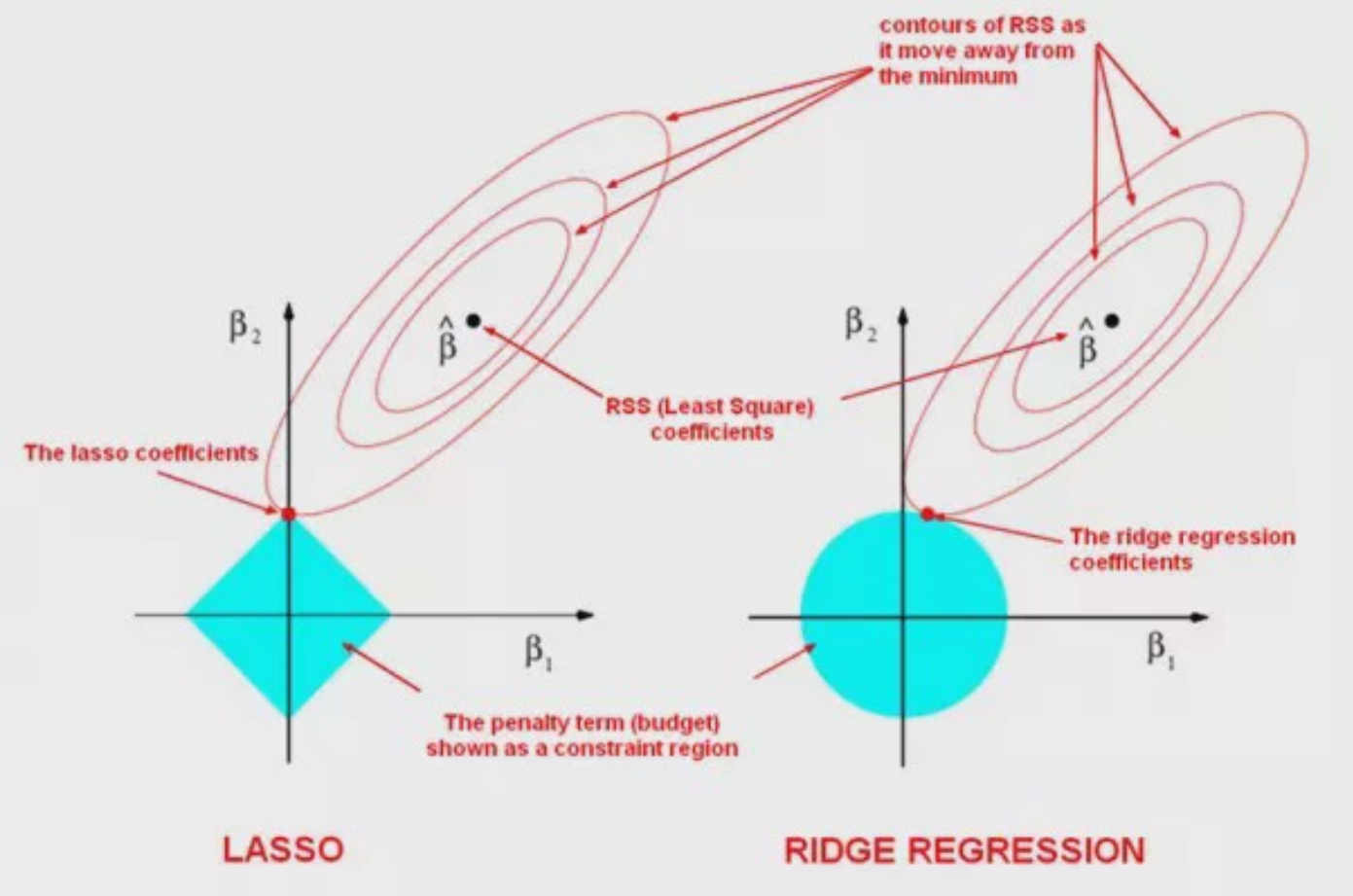
$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \text{Penalty Term}$$

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{m} x_{ij}\beta_j)^2 + \lambda \times \sum_{i=1}^{m} |\beta_i|$$

- Lasso regularization is referred to as Least Absolute Shrinkage or Selection Operator.
- L1 norm shrinks the parameters to zero.
  - When input features have weights closer to zero that leads to sparse L1 norm. In Sparse solution majority of the input features have zero weights and very few features have non-zero weights.
- Not all input features have the same influence on the prediction. L1 norm will assign a zero weight to features with less predictive power.
- Lasso regularization does feature selection. It does this by assigning insignificant input features with zero weight and useful features with a non-zero weight.
- Since Lasso use L1 norm, so it is not effected much by outliers as compared to Ridge.
- Lasso regression gives us sparse solution.

- **Why Lasso have NO closed form solution?** Due to L1 penalty the cost function is no longer continuously differentiable, therefore, no closed form solution exist for LASSO Regression. (On the contrary we do have a closed form solution for Ridge Regression). One can use discrete optimization techniques like Pathwise Coordinate Descent, Least Angle Regression, Forward Stepwise Regression to minimize the cost function for Lasso Regression.

d. Graphical Visualization



- The two contuour plots are of the least square function (RSS) and the black dot at the center represents the minimum of the function.
- The penalty term for Ridge regression is represented by a circle, while the penalty term for Lasso regression is represented by a diamond, with minimum at the origin.
- For Ridge as well as for Lasso regression, we want to minimize both the RSS and the penalty term function.
- In case of Lasso regression, there is a high probability that the contuour plot will meet with the diamond at the point
  - either along the β_2 line in which case β_1 will become zero or
  - along the β_1 line in which case β_2 will become zero.
- Therefore, the Lasso regression gives us a sparse solution as it makes some of the parameters exactly zero
- In case of Ridge regression, irrelevant features get a very small coefficient and in Lasso regression irrelevant features get exactly zero coefficient and are not considered in the model. That is the reason we say that Lasso do Feature Selection, and it does this by assigning insignificant input features with zero weight and useful features with a non-zero weight.

- Moreover, Lasso use L1 norm, so it is not effected much by outliers as compared to Ridge.
- It seems as Lasso is better than Ridge, but Lasso is computationally more expensive than Ridge, because it has a non-differentiable cost function and do not have a closed form solution like Ridge.

d. Cost Function for ElasticNet Regression (L1+L2)

- Ridge and Lasso are special cases of elastic net regression.
- ElasticNet Regression combines the strength of both, but require tuning of two hyper-parameters λ_1 and λ_2.
- Ridge and Lasso are special cases of ElasticNet regression.

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \text{Penalty Term}$$

$$\text{Cost Function} = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \lambda_1 \times \sum_{i=1}^{m}{\beta_i}^2 + \lambda_2 \times \sum_{i=1}^{m}|\beta_i|$$

- When $\lambda_1 = 0$ and $\lambda_2 = 0$, we get non-regularized solution.
- When $\lambda_1 > 0$ and $\lambda_2 = 0$, we get Ridge regression.
- When $\lambda_1 = 0$ and $\lambda_2 > 0$, we get Lasso regression.
- When $\lambda_1 > 0$ and $\lambda_2 > 0$, we get ElasticNet regression.

f. Scikit-Learn Estimators for Regularization

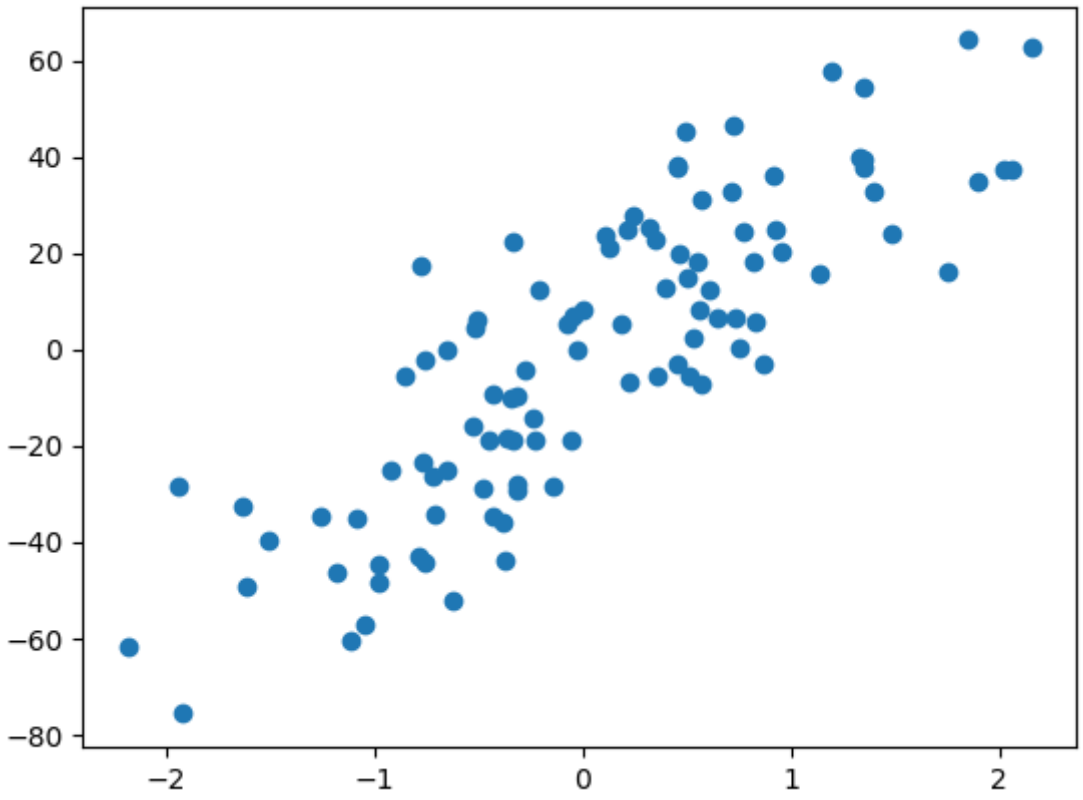
In sklearn, an estimator is a Python object that implements the fit(X, y) and predict(X) methods

```
from sklearn.model_family import Estimator

from sklearn.linear_model import Ridge, Lasso, ElasticNet

from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
```

Example 1: Ridge Regression (Intutition)

In [1]:

```
from sklearn import datasets
from matplotlib import pyplot as plt
X, y = datasets.make_regression(n_samples=100,
                        n_features=1,
                        noise=20,
                        n_targets=1,
                        random_state=13)
plt.scatter(X,y);
```



Use Standard Linear Regression to calculate Coefficients:

In [2]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X,y)
print("β1: ", lr.coef_)
print("β0: ",lr.intercept_)
```

```
β1:  [27.82809103]
β0:  -2.2947445586769795
```

Use Ridge Regression to calculate Coefficients (with `alpha=0`):

In [3]:

```
from sklearn.linear_model import Ridge

rr1 = Ridge(alpha=0)
rr1.fit(X,y)
print("β1: ", rr1.coef_)
print("β0: ",rr1.intercept_)
```

```
β1:  [27.82809103]
β0:  -2.29474455867698
```

Use Ridge Regression to calculate Coefficients (with `alpha=10`):

In [4]:

```
rr2 = Ridge(alpha=10)
rr2.fit(X,y)
print("β1: ", rr2.coef_)
print("β0: ",rr2.intercept_)
```

```
β1:  [24.9546267]
β0:  -2.1269130035235726
```

Use Ridge Regression to calculate Coefficients (with `alpha=100`):

In [5]:

```
rr3 = Ridge(alpha=100)
rr3.fit(X,y)
print("β1: ", rr3.coef_)
print("β0: ",rr3.intercept_)
```

```
β1:  [12.93442104]
β0:  -1.4248441496033308
```

In [6]:

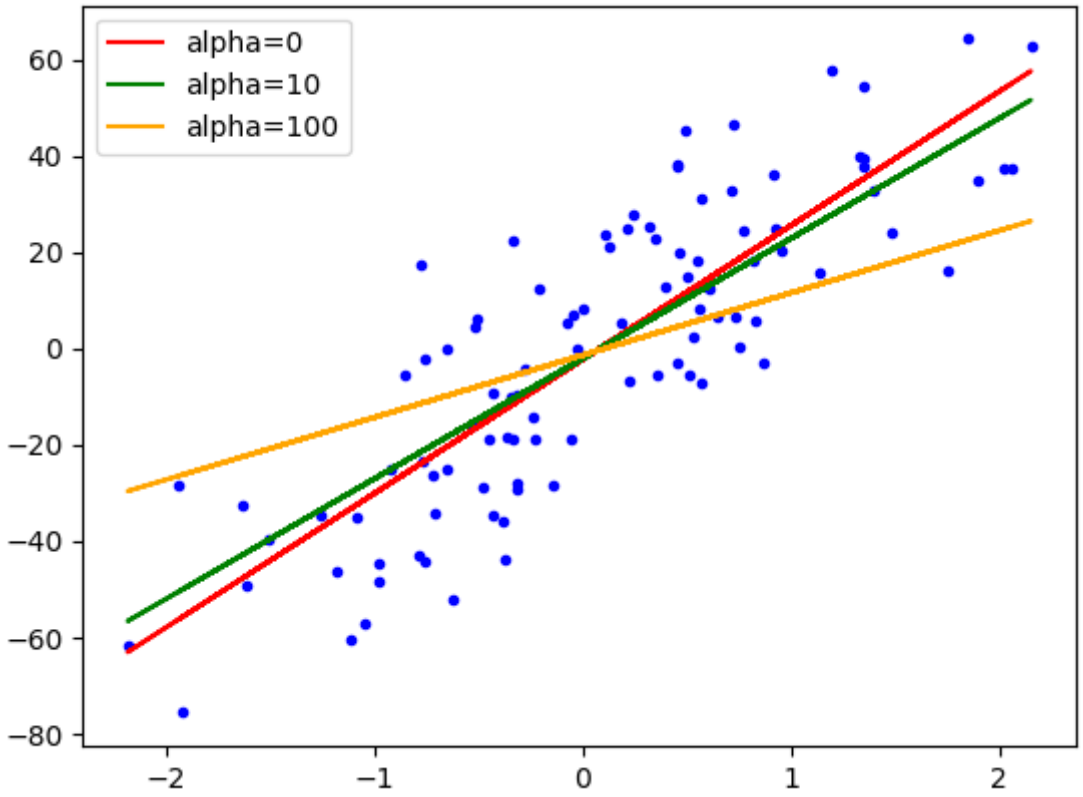
```
# For Lasso with large alpha value the regression coefficient becomes zero
from sklearn.linear_model import Lasso
a = Lasso(alpha=100)
a.fit(X,y)
print("β1: ", a.coef_)
print("β0: ",a.intercept_)
```

```
β1:  [0.]
β0:  -0.669378360869679
```

Visualize the Regression Lines with different alpha values:

In [7]:

```
plt.plot(X, y, 'b.')
plt.plot(X, rr1.predict(X), color='red', label='alpha=0')
plt.plot(X, rr2.predict(X), color='green', label='alpha=10')
plt.plot(X, rr3.predict(X), color='orange', label='alpha=100')
plt.legend();
```



Example 2: Ridge Regression (Prevent Overfitting)

In [8]:

```
import pandas as pd
df = pd.read_csv("datasets/Melbourne_housing.csv")
df
```

Out[8]:

	Suburb	Rooms	Type	Method	SellerG	Distance	Regionname	Propertycount	C
0	Abbotsford	2	h	S	Biggin	2.5	Northern Metropolitan	4019.0	
1	Abbotsford	2	h	S	Biggin	2.5	Northern Metropolitan	4019.0	
2	Abbotsford	3	h	SP	Biggin	2.5	Northern Metropolitan	4019.0	
3	Abbotsford	3	h	PI	Biggin	2.5	Northern Metropolitan	4019.0	
4	Abbotsford	4	h	VB	Nelson	2.5	Northern Metropolitan	4019.0	
...	...	...	...	...	...	...	...	...	
27239	Yarraville	4	h	PI	Jas	6.3	Western Metropolitan	6543.0	M
27240	Yarraville	2	h	SP	Sweeney	6.3	Western Metropolitan	6543.0	M
27241	Yarraville	2	t	S	Jas	6.3	Western Metropolitan	6543.0	M
27242	Yarraville	3	h	SP	hockingstuart	6.3	Western Metropolitan	6543.0	M
27243	Yarraville	2	h	PI	RW	6.3	Western Metropolitan	6543.0	M

27244 rows × 15 columns

In [9]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27244 entries, 0 to 27243
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Suburb          27244 non-null  object
 1   Rooms           27244 non-null  int64
 2   Type            27244 non-null  object
 3   Method          27244 non-null  object
 4   SellerG         27244 non-null  object
 5   Distance        27244 non-null  float64
 6   Regionname      27244 non-null  object
 7   Propertycount   27244 non-null  float64
 8   CouncilArea     27244 non-null  object
 9   Bedroom2        27244 non-null  float64
 10  Bathroom        27244 non-null  float64
 11  Car             27244 non-null  float64
 12  Landsize        27244 non-null  float64
 13  BuildingArea    27244 non-null  float64
 14  Price           27244 non-null  float64
dtypes: float64(8), int64(1), object(6)
memory usage: 3.1+ MB
```

Handle categorical variables:

In [10]:

```
1 df = pd.get_dummies(df, drop_first=True)
2 df.head()
```

Out[10]:

	Rooms	Distance	Propertycount	Bedroom2	Bathroom	Car	Landsize	BuildingArea	Pric
0	2	2.5	4019.0	2.0	1.0	1.0	202.0	160.2564	1480000
1	2	2.5	4019.0	2.0	1.0	0.0	156.0	79.0000	1035000
2	3	2.5	4019.0	3.0	2.0	0.0	134.0	150.0000	1465000
3	3	2.5	4019.0	3.0	2.0	1.0	94.0	160.2564	850000
4	4	2.5	4019.0	3.0	1.0	2.0	120.0	142.0000	1600000

5 rows × 745 columns

Train | Test Split:

In [11]:

```
X = df.drop('Price', axis=1)
y = df['Price']
X.shape, y.shape
```

Out[11]:

```
((27244, 744), (27244,))
```

In [12]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

Standard Linear Regression

In [13]:

```
lr = LinearRegression()
lr.fit(X_train,y_train)
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
from sklearn.metrics import r2_score
print("R2 score on Training set: ", lr.score(X_train, y_train))
print("R2 score on Training set: ", r2_score(y_train, lr.predict(X_train)))
```

```
R2 score on Training set:  0.6827792395792723
R2 score on Training set:  0.6827792395792723
```

In [15]:

```
print("R2 score on Test set: ", lr.score(X_test, y_test))
print("R2 score on Test set: ", r2_score(y_test, lr.predict(X_test)))
```

```
R2 score on Test set:  0.1385368316175627
R2 score on Test set:  0.1385368316175627
```

> Above model is Overfitting, because it is giving reasonably good results on training data but very bad results on test data :(

Ridge Linear Regression

In [16]:

```
from sklearn.linear_model import Ridge
rr = Ridge(alpha=1)
rr.fit(X_train,y_train)
```

Out[16]:

```
Ridge(alpha=1)
```

In [17]:

```
print("R2 score on Training set: ", rr.score(X_train, y_train))
print("R2 score on Training set: ", r2_score(y_train, rr.predict(X_train)))
```

```
R2 score on Training set:  0.6796668251040214
R2 score on Training set:  0.6796668251040214
```

In [18]:

```
print("R2 score on Test set: ", rr.score(X_test, y_test))
print("R2 score on Test set: ", r2_score(y_test, rr.predict(X_test)))
```

```
R2 score on Test set:  0.6701765758295282
R2 score on Test set:  0.6701765758295282
```

> Above model is a good fit, because it is giving almost same results on both training and testing data :)

> Try increasing the alpha value to 10000 and see how the model underfits.

Example 3: Impact of Alpha Values of Ridge Regression

Load Dataset:

In [19]:

```
from sklearn.datasets import load_diabetes
diabetes=load_diabetes(as_frame=True)
```

In [20]:

```
print(diabetes.get('DESCR'))
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n
=
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive value
s

  :Target: Column 11 is a quantitative measure of disease progression
one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and s
caled by the standard deviation times `n_samples` (i.e. the sum of squ
ares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html (https://www
4.stat.ncsu.edu/~boos/var.select/diabetes.html)

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (20
04) "Least Angle Regression," Annals of Statistics (with discussion),
407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [21]:

```
X = diabetes.get('data')    #diabetes.data      #diabetes['data']
y = diabetes.get('target')  #diabetes.target    #diabetes['target']
```

Train | Test Split:

In [22]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

Standard Linear Regression

In [23]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)              # Fit a ridge regression on the training da
lr_pred = lr.predict(X_test)            # Use this model to predict the test data
print(pd.Series(lr.coef_, index = X.columns)) # Print coefficients
```

```
age    -31.805370
sex   -214.671676
bmi    445.137506
bp     376.354944
s1    -802.451729
s2     525.951855
s3     106.399671
s4     172.311225
s5     799.479042
s6      57.167687
dtype: float64
```

Ridge Regression with `alpha=0`

In [24]:

```
from sklearn.linear_model import Ridge
rr1=Ridge(alpha=0)
rr1.fit(X_train, y_train)              # Fit a ridge regression on the training d
rr1.predict(X_test)           # Use this model to predict the test data
print(pd.Series(rr1.coef_, index=X.columns)) # Print coefficients
```

```
age    -31.805370
sex   -214.671676
bmi    445.137506
bp     376.354944
s1    -802.451729
s2     525.951855
s3     106.399671
s4     172.311225
s5     799.479042
s6      57.167687
dtype: float64
```

Ridge Regression with `alpha=1`

In [25]:

```
from sklearn.linear_model import Ridge
rr1=Ridge(alpha=1)
rr1.fit(X_train, y_train)              # Fit a ridge regression on the training d
rr1.predict(X_test)           # Use this model to predict the test data
print(pd.Series(rr1.coef_, index=X.columns)) # Print coefficients
```

```
age     38.755675
sex    -52.971195
bmi    234.615921
bp     193.163458
s1      27.821247
s2      -2.904829
s3    -139.601939
s4     116.142663
s5     242.458241
s6     101.463602
dtype: float64
```

Ridge Regression with `alpha=10000`

In [26]:

```
from sklearn.linear_model import Ridge
rr1=Ridge(alpha=10000)
rr1.fit(X_train, y_train)
rr1.predict(X_test)
print(pd.Series(rr1.coef_, index=X.columns)) # Print coefficients
```

```
age     0.025811
sex     0.004663
bmi     0.062004
bp      0.052696
s1      0.026870
s2      0.021778
s3     -0.044786
s4      0.049758
s5      0.067498
s6      0.042867
dtype: float64
```

Lasso Regression with `alpha=10000`

In [27]:

```
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=10000)
lasso_reg.fit(X_train, y_train)
lasso_reg.predict(X_test)
print(pd.Series(lasso_reg.coef_, index=X.columns))
```

```
age     0.0
sex     0.0
bmi     0.0
bp      0.0
s1      0.0
s2      0.0
s3     -0.0
s4      0.0
s5      0.0
s6      0.0
dtype: float64
```

A Quick Comparison

Ridge/L2 Regularization

- L2 regularization penalizes sum of square weights.
- L2 has a non sparse solution
- L2 has one solution
- L2 has no feature selection
- L2 is not robust to outliers
- L2 gives better prediction when output variable is a function of all input features
- L2 regularization is able to learn complex data patterns

Lasso/L1 Regularization

- L1 penalizes sum of absolute value of weights.
- L1 has a sparse solution
- L1 has multiple solutions
- L1 has built in feature selection
- L1 is robust to outliers
- L1 generates model that are simple and interpretable but cannot learn complex patterns

How to choose the best alpha value for Ridge, Lasso and ElasticNet Regression?

In [ ]:

```
1
```

In [ ]:

```
1
```

Task To Do

In [28]:

```
1 import pandas as pd
2 teams = pd.read_csv("datasets/teams.csv")
3 teams
```

Out[28]:

	team	year	athletes	events	age	height	weight	prev_medals	medals
0	AFG	1964	8	8	22.0	161.0	64.2	0.0	0
1	AFG	1968	5	5	23.2	170.2	70.0	0.0	0
2	AFG	1972	8	8	29.0	168.3	63.8	0.0	0
3	AFG	1980	11	11	23.6	168.4	63.2	0.0	0
4	AFG	2004	5	5	18.6	170.8	64.8	0.0	0
...	...	...	...	...	...	...	...	...	...
2009	ZIM	2000	26	19	25.0	179.0	71.1	0.0	0
2010	ZIM	2004	14	11	25.1	177.8	70.5	0.0	3
2011	ZIM	2008	16	15	26.1	171.9	63.7	3.0	4
2012	ZIM	2012	9	8	27.3	174.4	65.2	4.0	0
2013	ZIM	2016	31	13	27.5	167.8	62.2	0.0	0

2014 rows × 9 columns