Machine Learning

Lecture 05

Min-Kuan Chang

minkuanc@nchu.edu.tw

EE, College of EECS

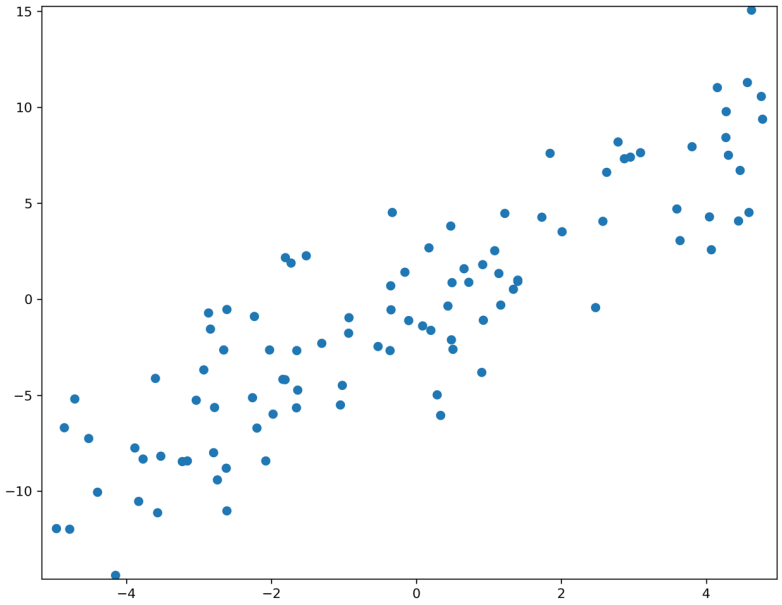
Implementation Using Scikit-Learn

Topic 01

Linear Regression

Step 01: Dataset Preparation

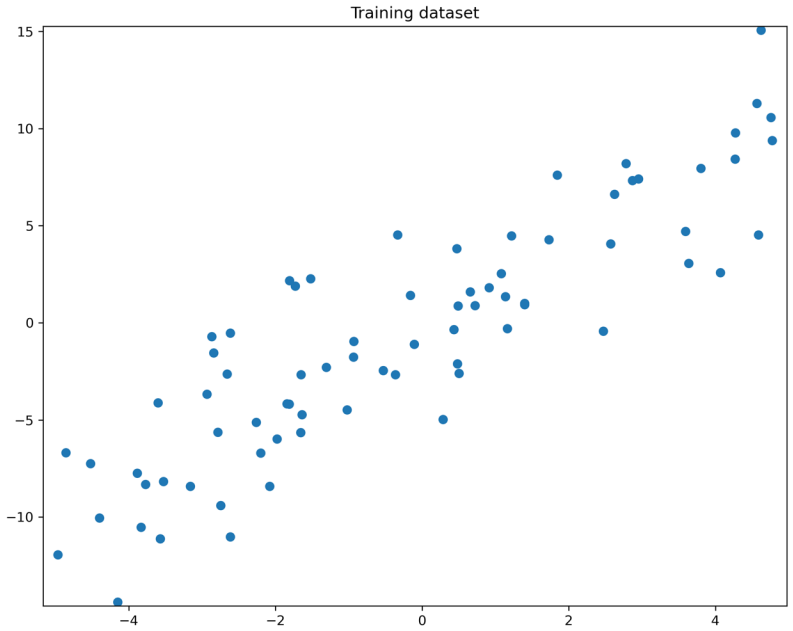
```
import numpy as np
X = np.random.uniform(-5,5,100)
y = 2*X + np.random.normal(0,3,100)
```

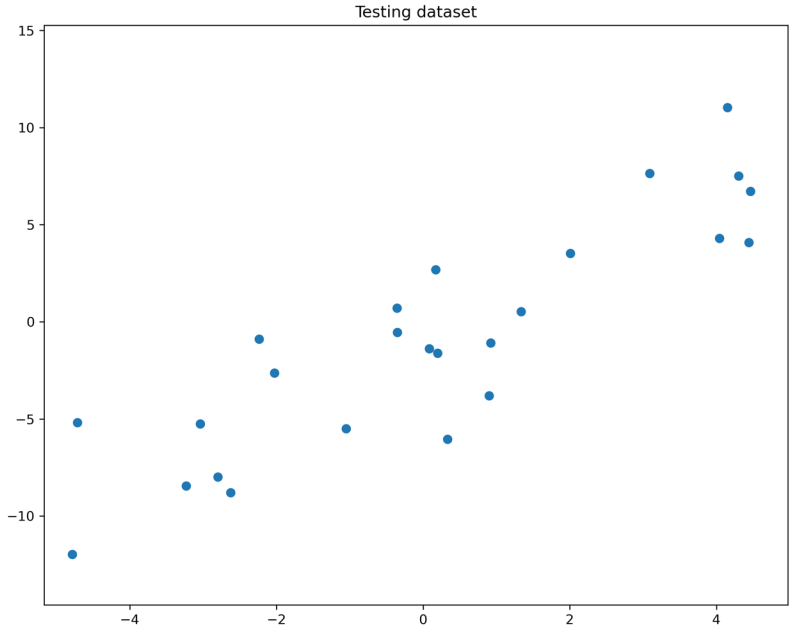


We are seeking a linear model
$\hat{y} = w_0 + w_1 x$ to fit the data

Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X=X.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)
```





Step 03: Build/Train ML Model

- 'LinearRegression' is responsible for building a linear regression model
- The object of class 'LinearRegression' is declared before training

```
from sklearn.linear_model import LinearRegression
LinearRegression_Model = LinearRegression()
```

  - no particular arguments are needed
- To train this model, we simply use the class method 'fit' to determine the bias and weights of the model

```
LinearRegression_Model.fit(X_train, y_train)
```

Step 04: Test the ML model

- We can use the method ‘predict’ to predict the output of a new data

```
LinearRegression_Model.predict([[3]])
```

  - the output is

```
array([5.77166998])
```

- We can use the method ‘score’ to understand the performance of the model

```
LinearRegression_Model.score(X_test, y_test)
```

  - the output is

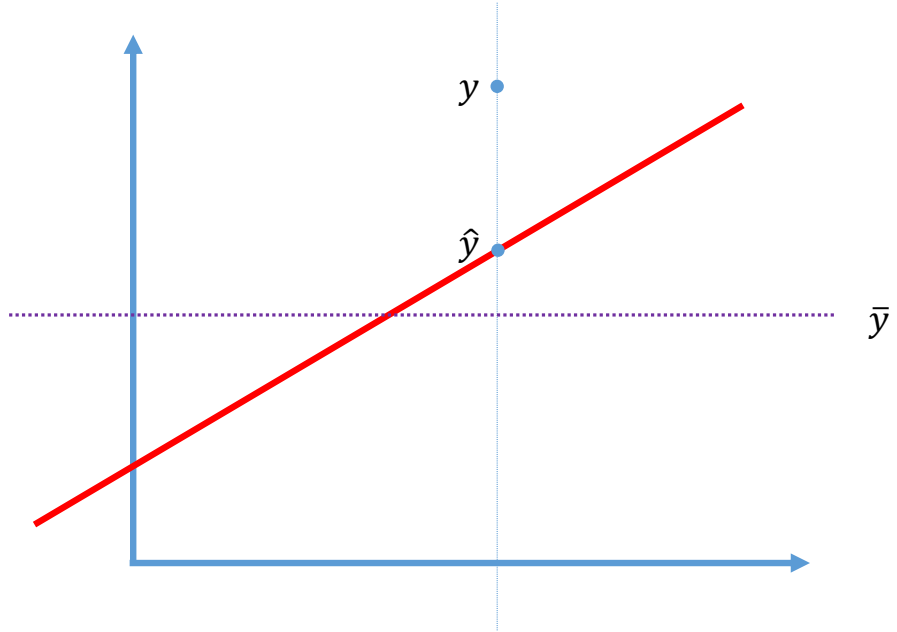
```
0.734295740699449
```

字幕預留位置

Step 04: Test the ML model

- The method 'score' returns the coefficient of determination or R^2 score

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y-\hat{y})^2}{\sum_{i=1}^{N}(y-\bar{y})^2}$$



Discussion

- LinearRegression_Model has many useful attributes
  - the information of the fitting function

```
LinearRegression_Model.intercept_
```

  - the bias term, w_0, of the linear model
  - the output is

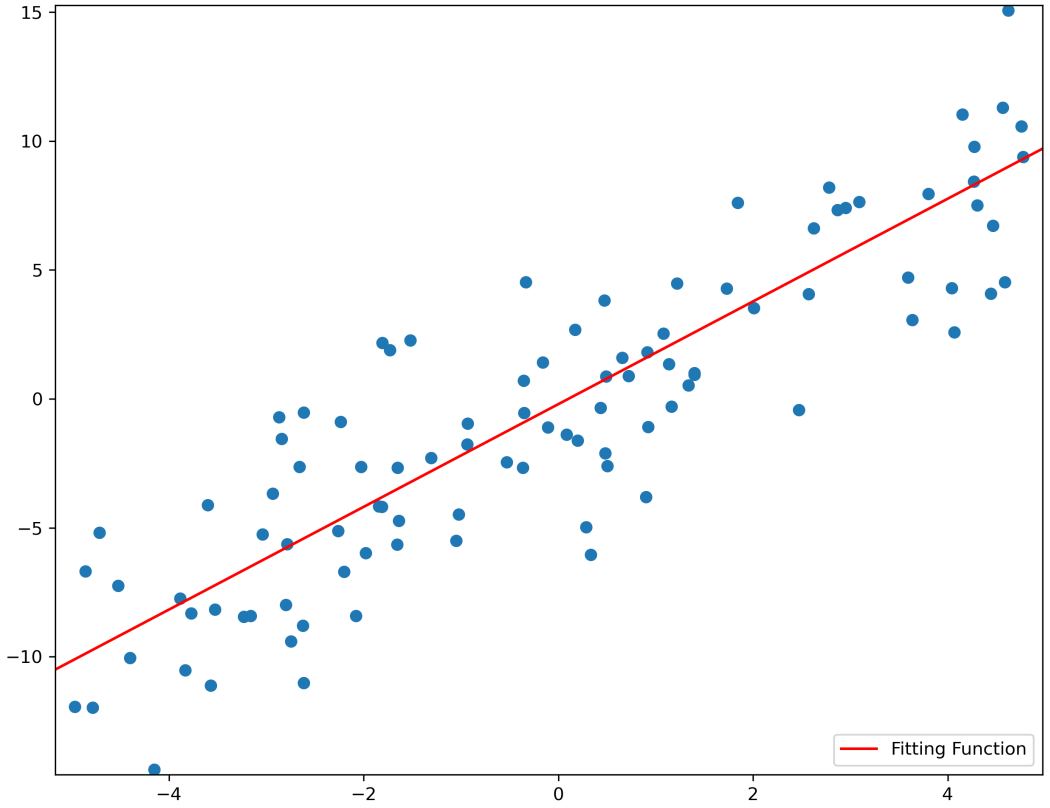
```
-0.2047650511796517
```

```
LinearRegression_Model.coef_
```

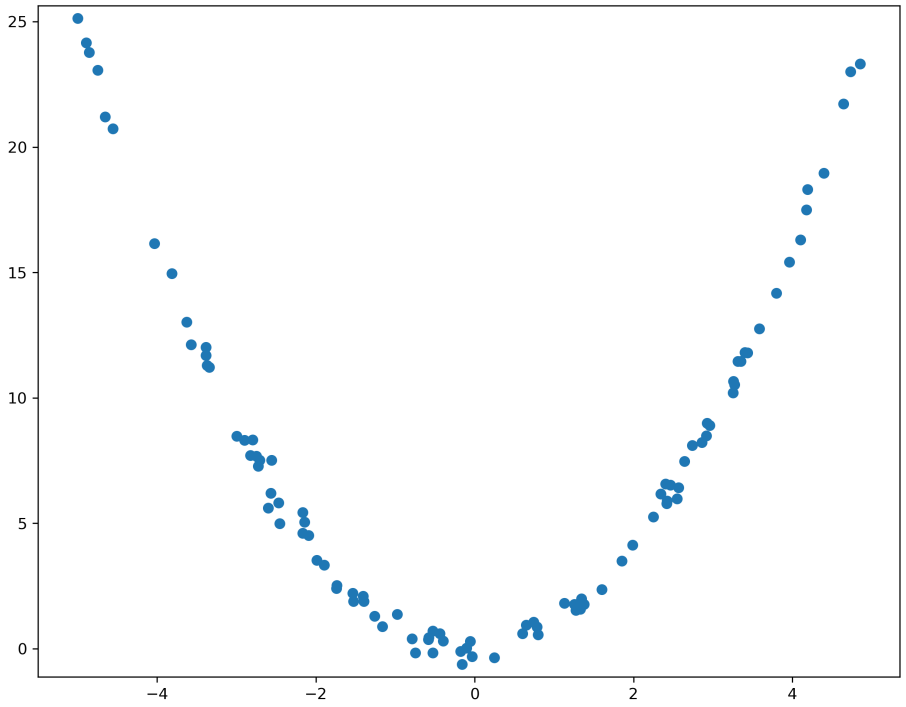
  - record the weights of the linear model
  - the output is

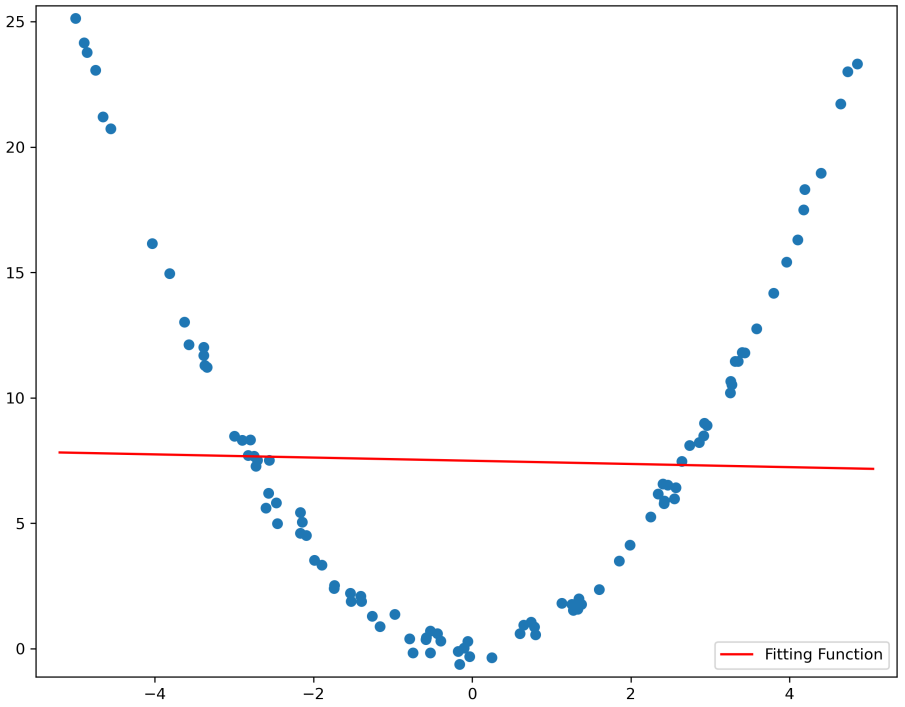
```
array([1.99214501])
```

Discussion

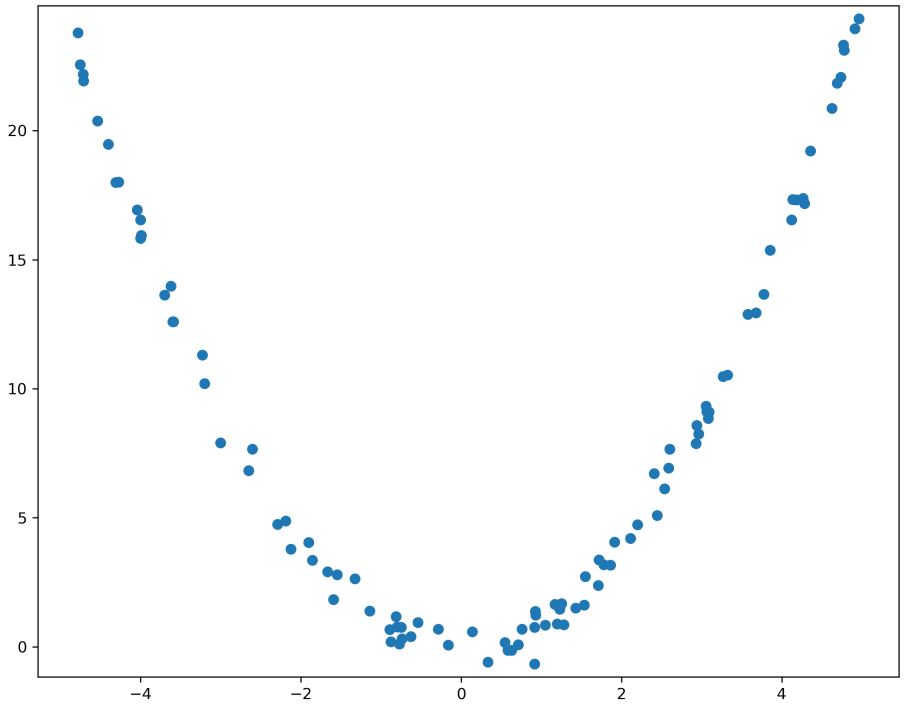


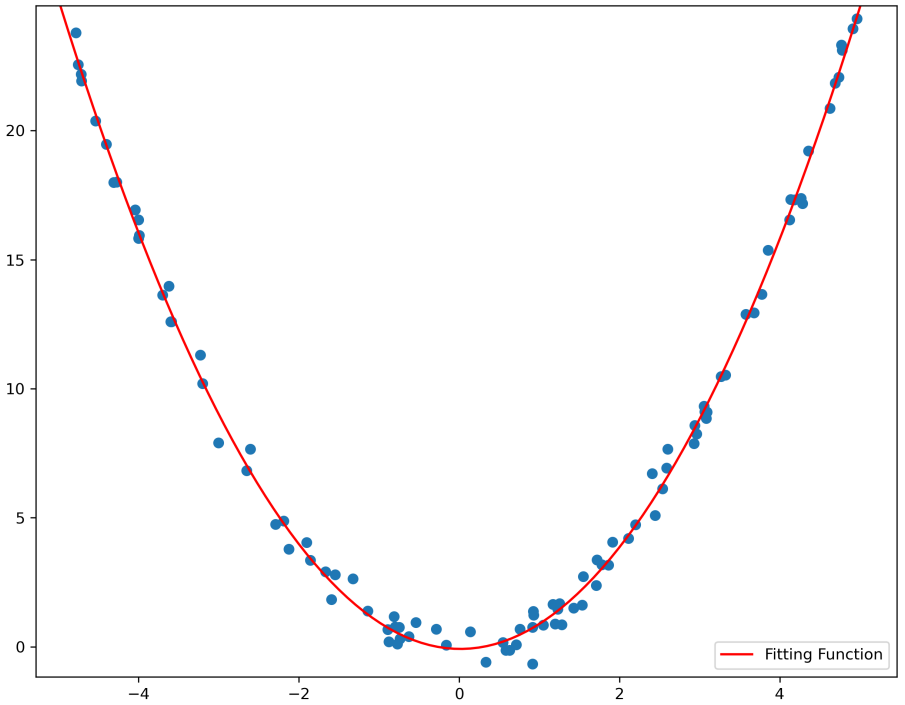
Discussion



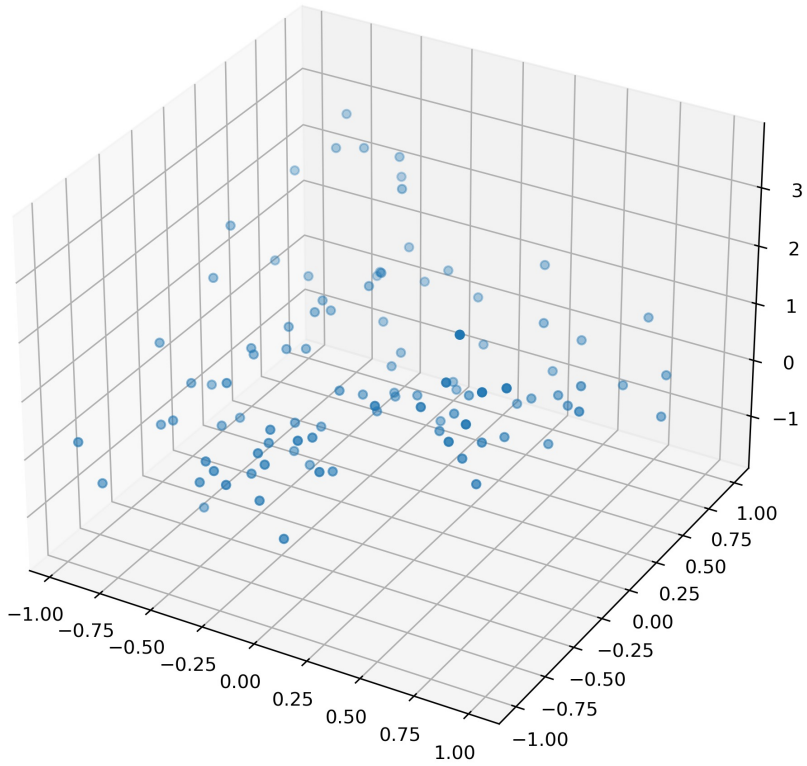


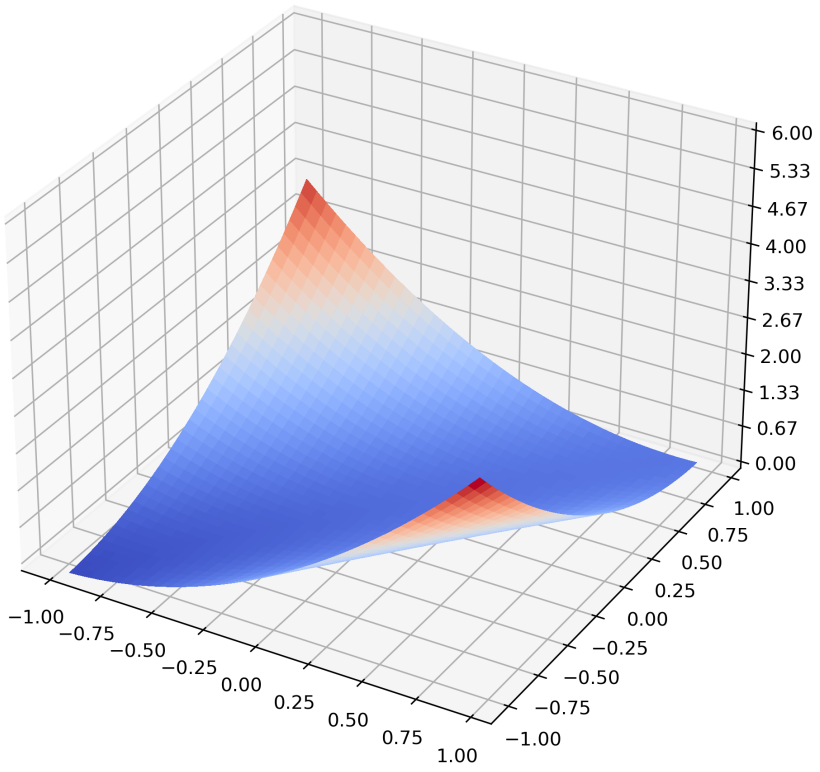
Discussion





Discussion


3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


6.00
5.33
4.67
4.00
3.33
2.67
2.00
1.33
0.67
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

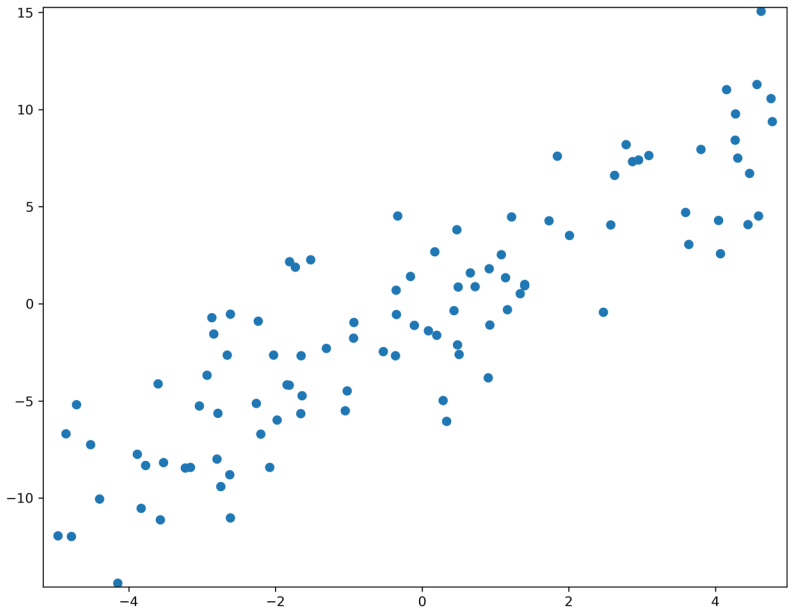
Implementation Using Scikit-Learn

Topic 02

Regularized Linear Regression - Ridge

Step 01: Dataset Preparation

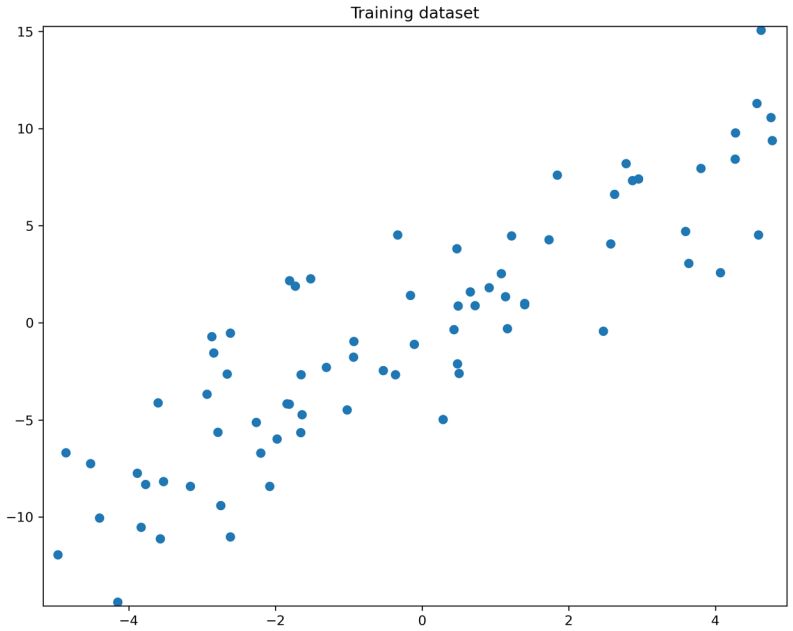
```
import numpy as np
X = np.random.uniform(-5,5,100)
y = 2*X + np.random.normal(0,3,100)
```

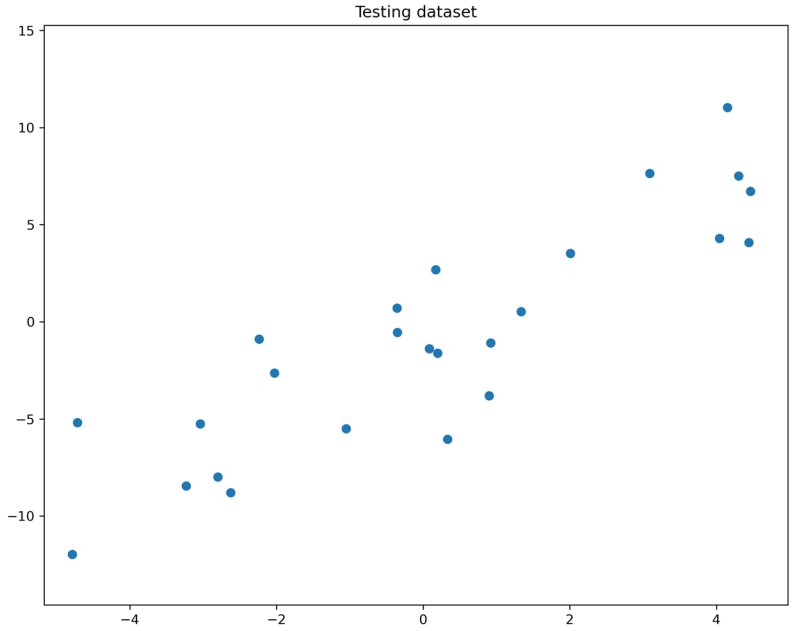


We are seeking for a Ridge model
$\hat{y} = w_0 + w_1 x$ to fit the data

Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X=X.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)
```





Step 03: Build/Train ML Model

- The class 'Ridge' is utilized to build a Ridge model
- We simply declare an object of 'Ridge' and initialize this model

```
from sklearn.linear_model import Ridge
Ridge_Model = Ridge(alpha = 1, max_iter = 1000, tol = 0.001)
```

  - 'alpha' controls the strength of the regularization
  - 'max_iter' determines the maximum number of iteration to find the weights of the models
  - 'tol' is the precision of the solution
- The training of the model is to call the 'fit' method

```
Ridge_Model.fit(X_train, y_train)
```

Step 04: Test the ML model

- When a new input comes, we can use the 'predict' method to predict its possible output

```
Ridge_Model.predict([[5]])
```

  - its output is

```
array([8.91188908])
```

- The 'score' method is to evaluate how fit the model is

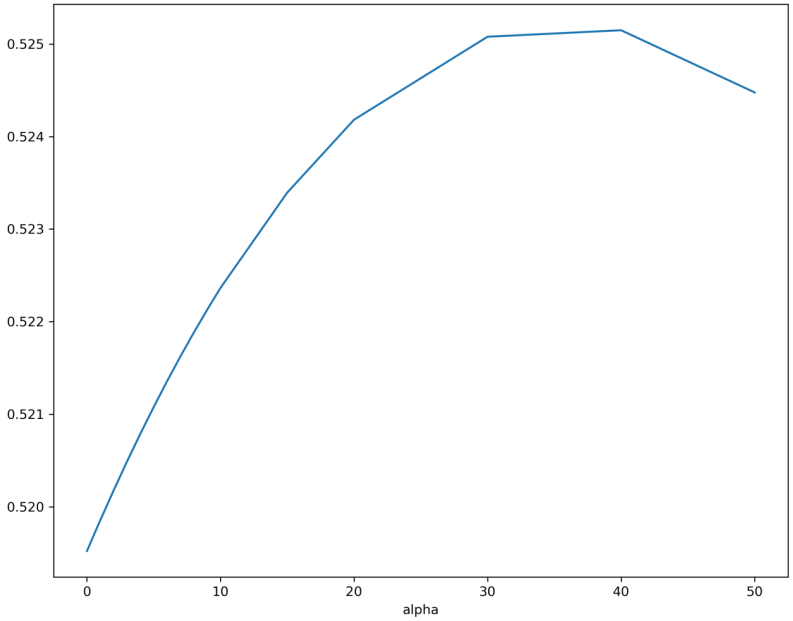
```
Ridge_Model.score(X_test, y_test)
```

  - its output is

```
0.5198518682703732
```

Discussion

- Since the regularization plays a part in Ridge, we need to access how the strength of regularization affects the R^2 score



Discussion

- The bias and the weight of the fitting function
  - bias: w_0

```
Ridge_Model.intercept_
```

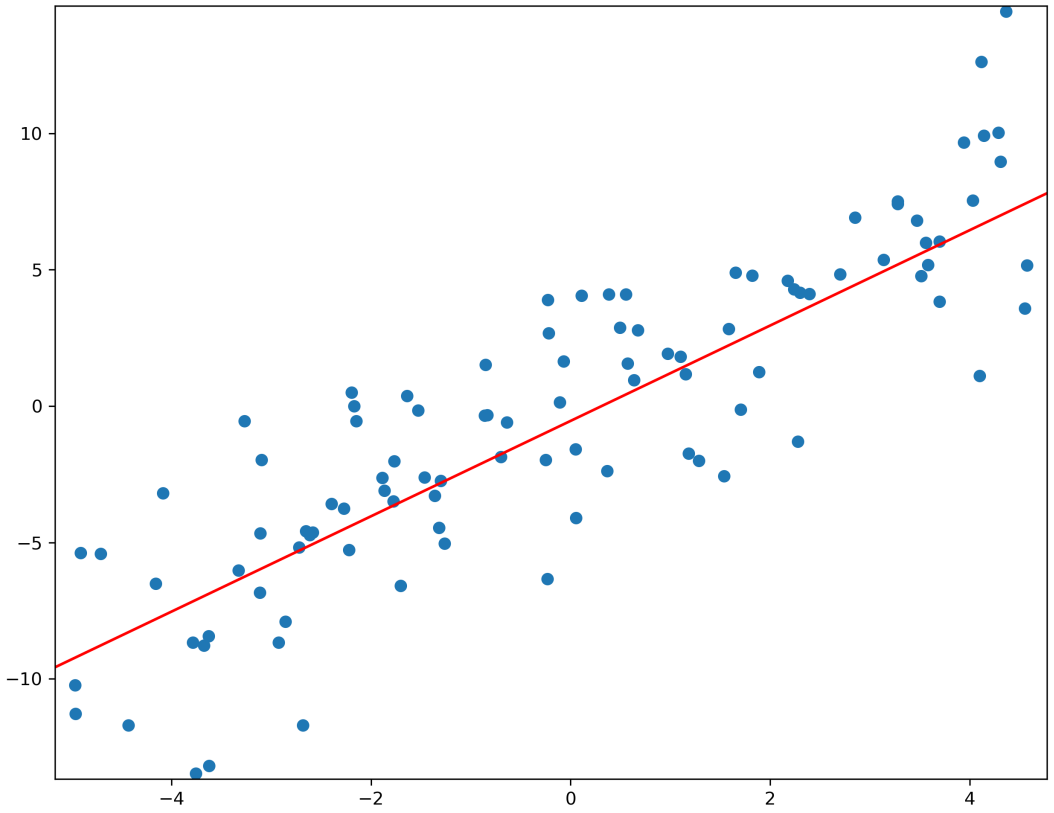
```
-0.5441973828355591
```

  - the weight: w_1

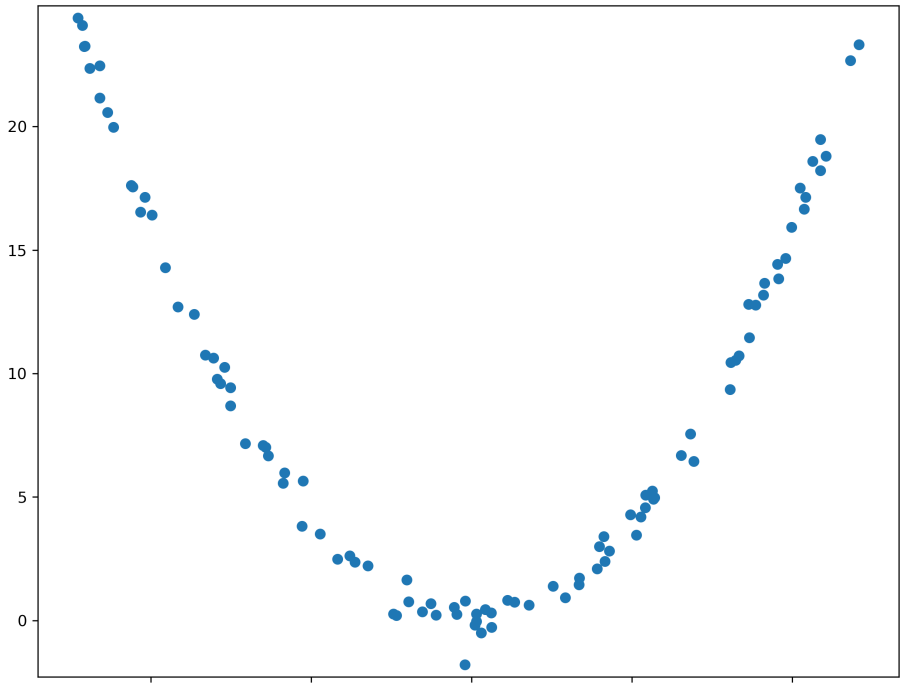
```
Ridge_Model.coef_
```

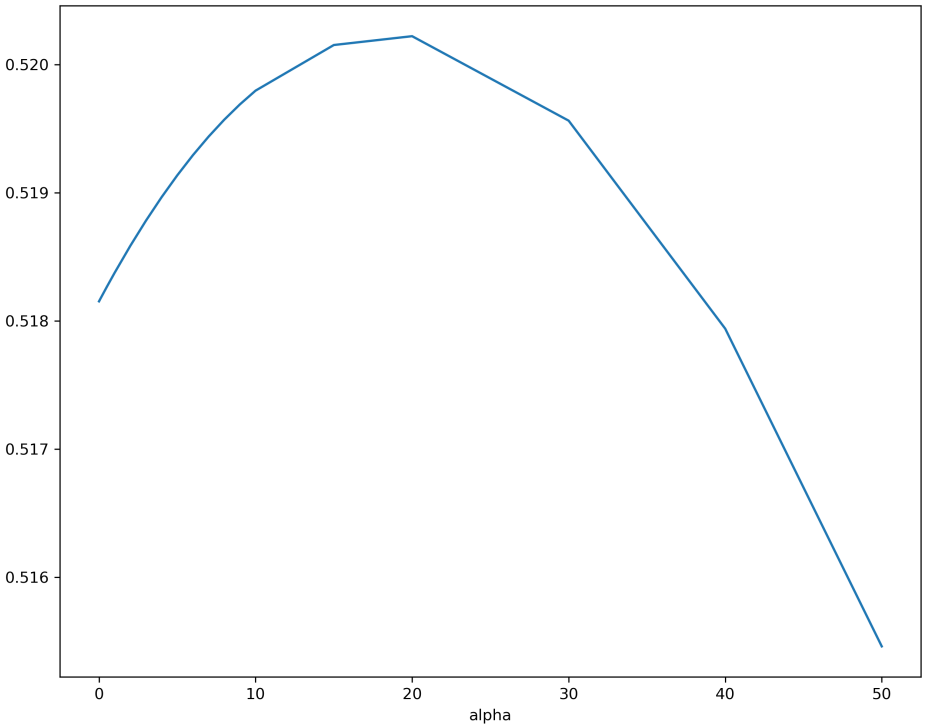
```
array([1.74817381])
```

Discussion

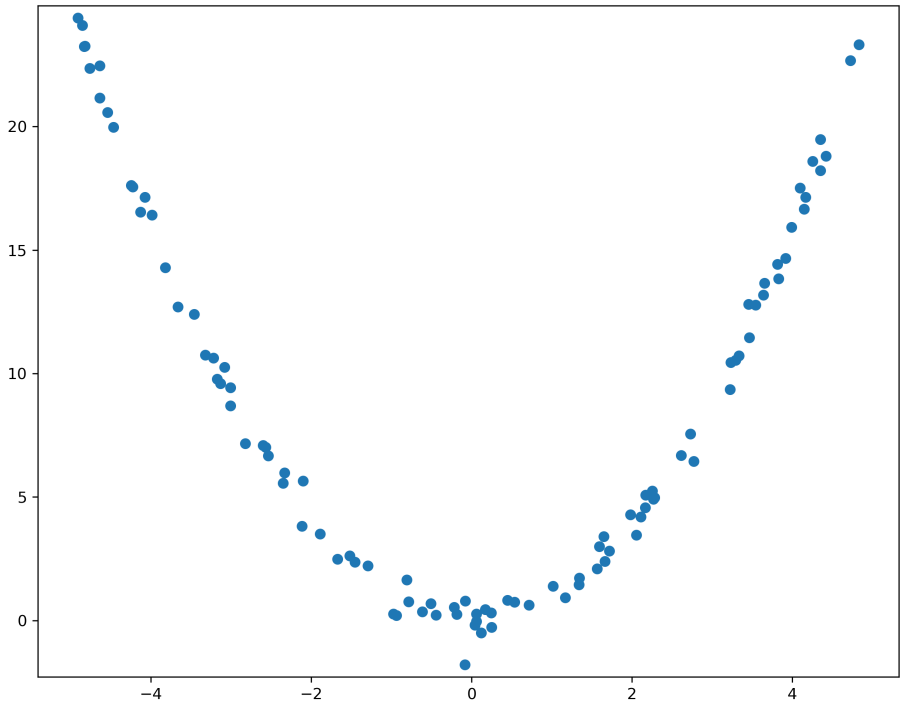


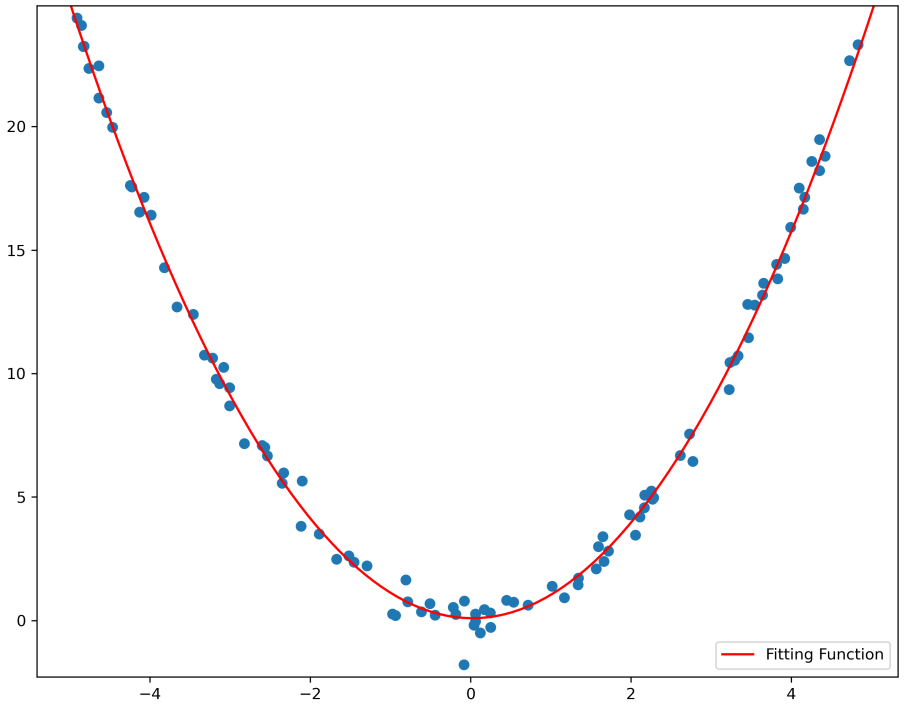
Discussion



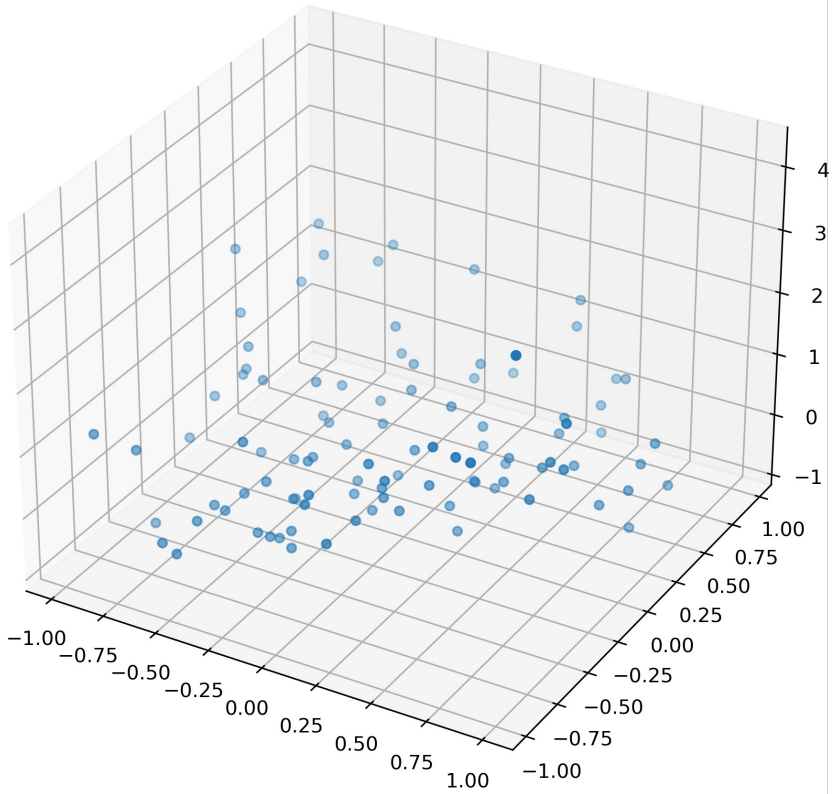


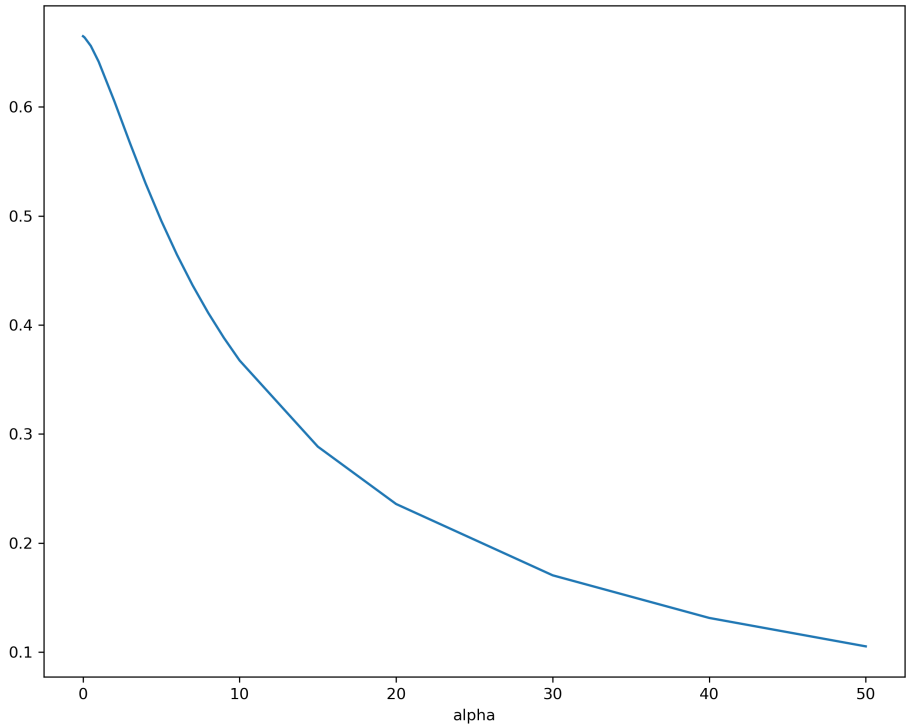
Discussion



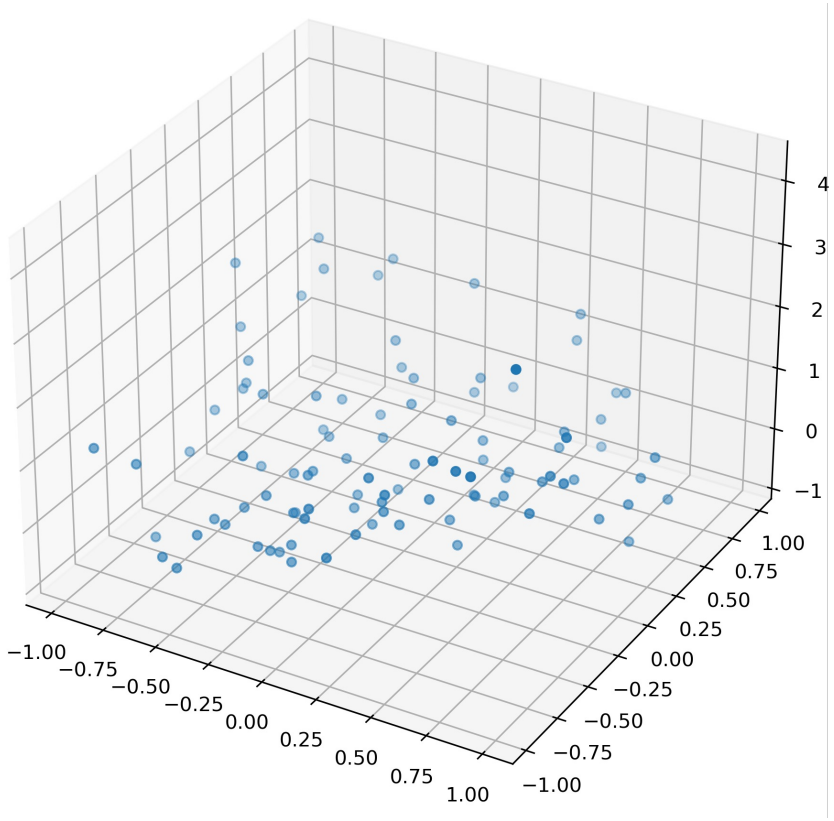


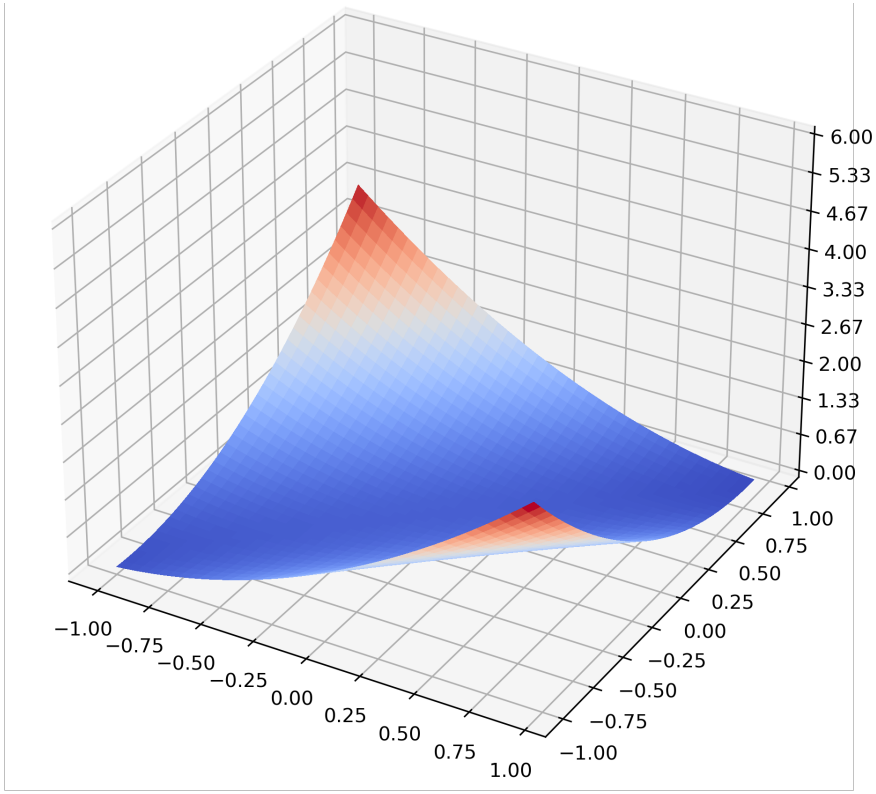
Discussion


4
3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
alpha

Discussion


4
3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


6.00
5.33
4.67
4.00
3.33
2.67
2.00
1.33
0.67
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

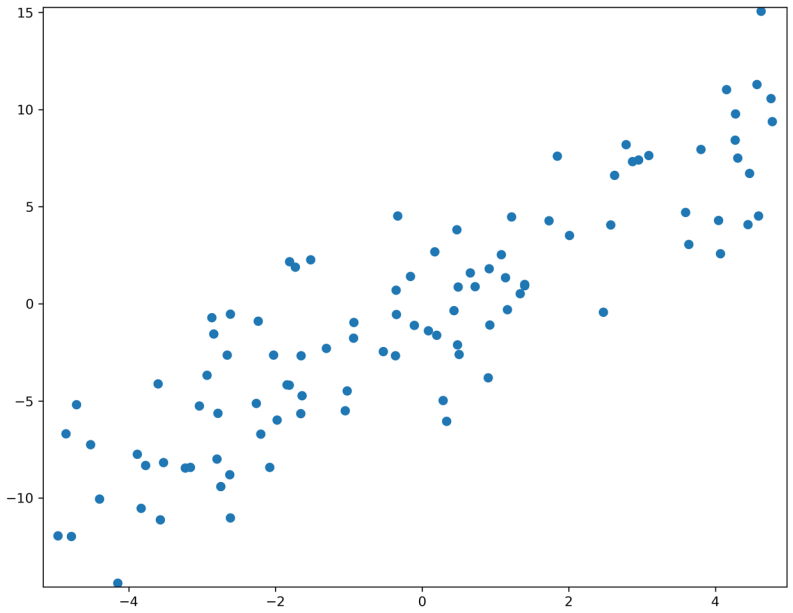
Implementation Using Scikit-Learn

Topic 03

Regularized Linear Regression - Lasso

Step 01: Dataset Preparation

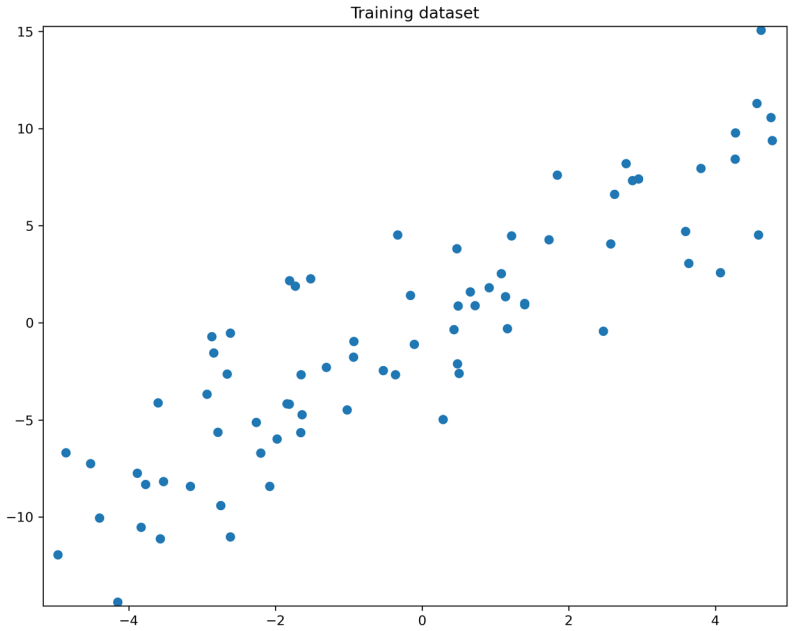
```
import numpy as np
X = np.random.uniform(-5,5,100)
y = 2*X + np.random.normal(0,3,100)
```

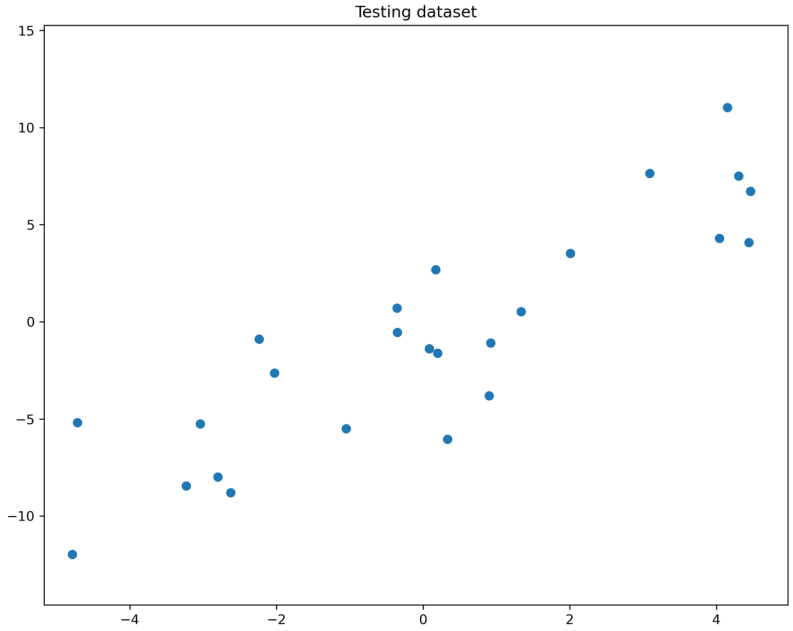


We are seeking for a Lasso model
$\hat{y} = w_0 + w_1 x$ to fit the data

Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X=X.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)
```





Step 03: Build/Train ML Model

- The class 'Lasso' is utilized to build a Lasso model
- An object of 'Lasso' is declared to initialize this model

```
from sklearn.linear_model import Lasso
Lasso_Model = Lasso(alpha = 1, max_iter = 1000, tol = 0.001)
```

  - 'alpha' controls the strength of the regularization
  - 'max_iter' determines the maximum number of iteration to find the weights of the models
  - 'tol' is the precision of the solution
- The training of the model is to call the 'fit' method

```
Lasso_Model.fit(X_train, y_train)
```

Step 04: Test the ML model

- When a new input comes, we can use the 'predict' method to predict its possible output

```
Lasso_Model.predict([[5]])
```

  - its output is

```
array([12.54081639])
```

- The 'score' method is to evaluate how fit the model is

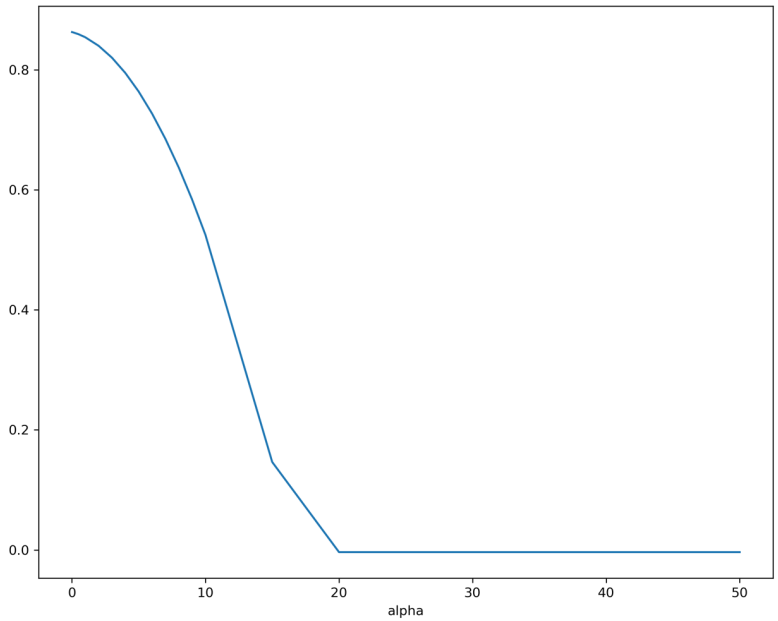
```
Lasso_Model.score(X_test, y_test)
```

  - its output is

```
0.8542612376965604
```

Discussion

- Since the regularization plays a part in Lasso, we need to access how the strength of regularization affects the R^2 score



Discussion

- The bias and the weight of the fitting function
  - bias: w_0

```
Lasso_Model.intercept_
```

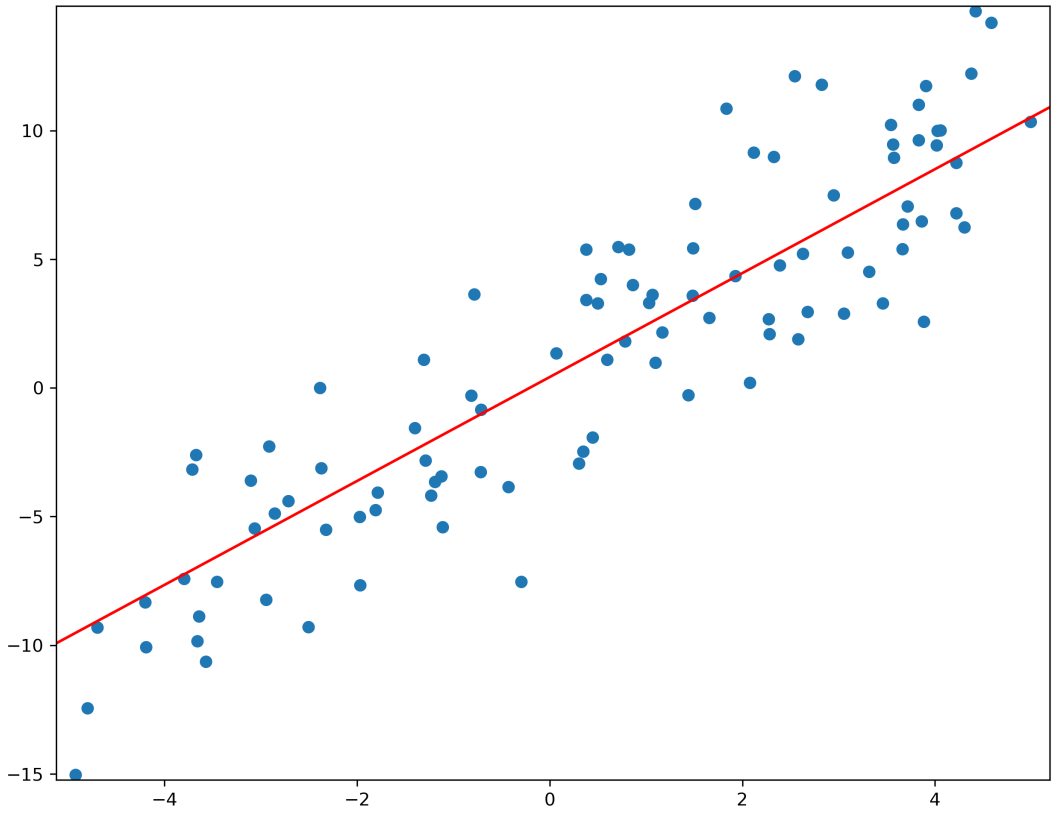
```
0.4235520049058268
```

  - the weight: w_1

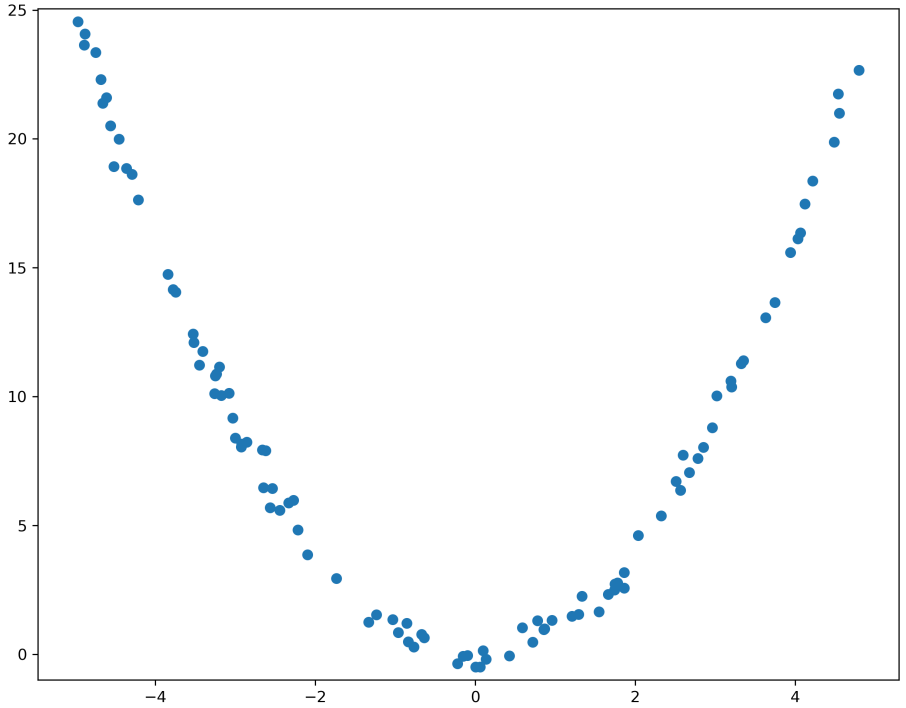
```
Lasso_Model.coef_
```

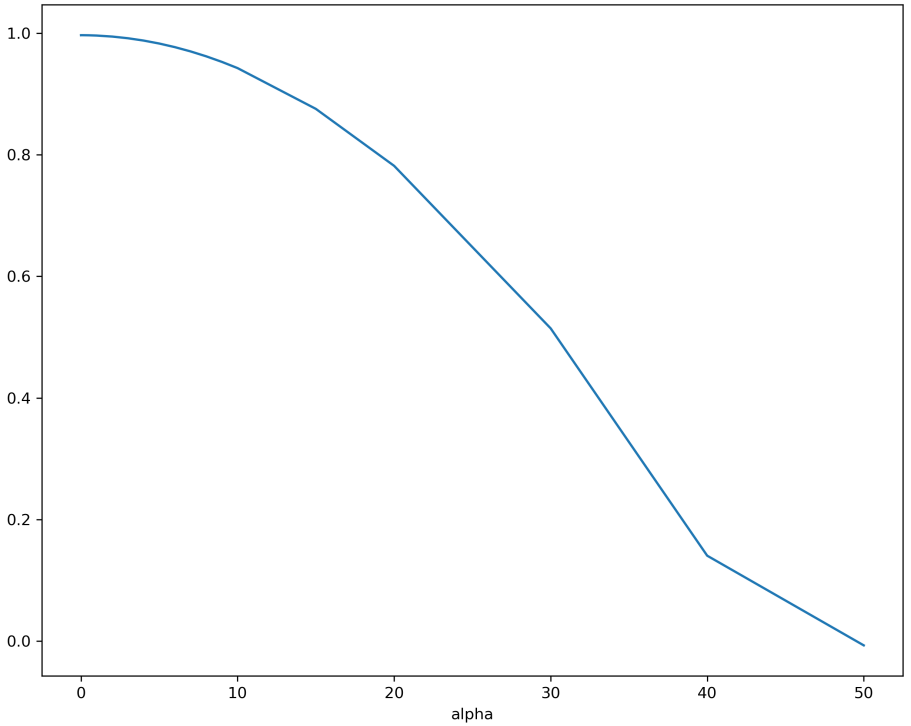
```
array([2.01954406])
```

Discussion

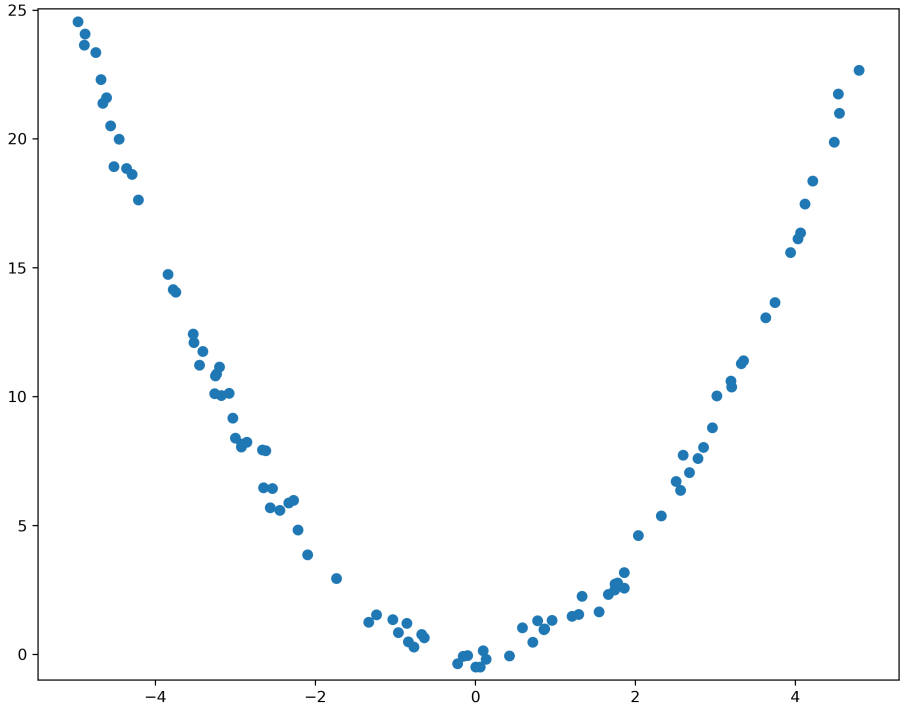


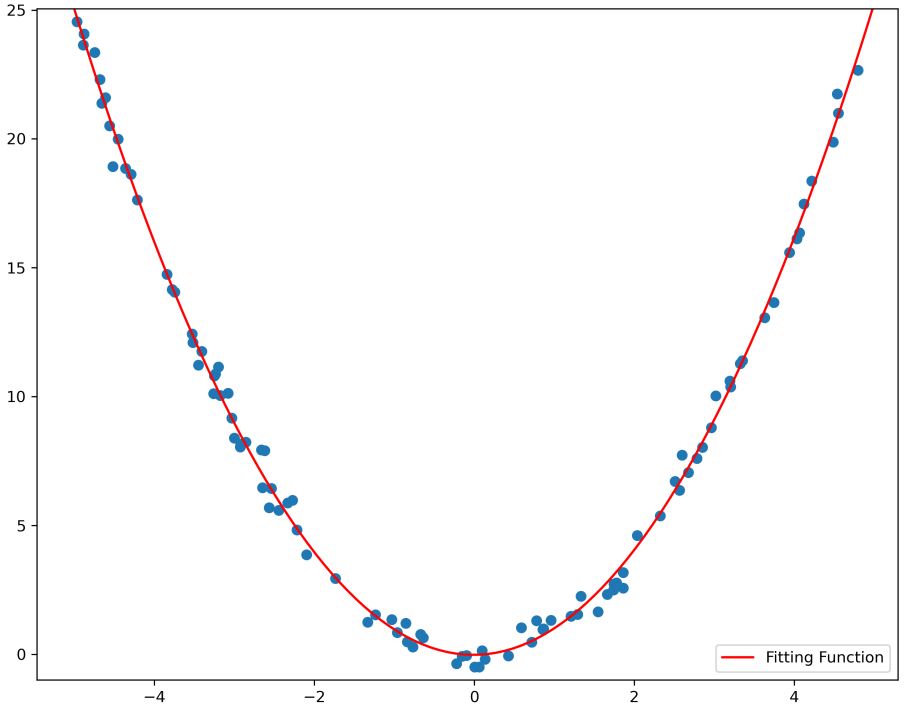
Discussion



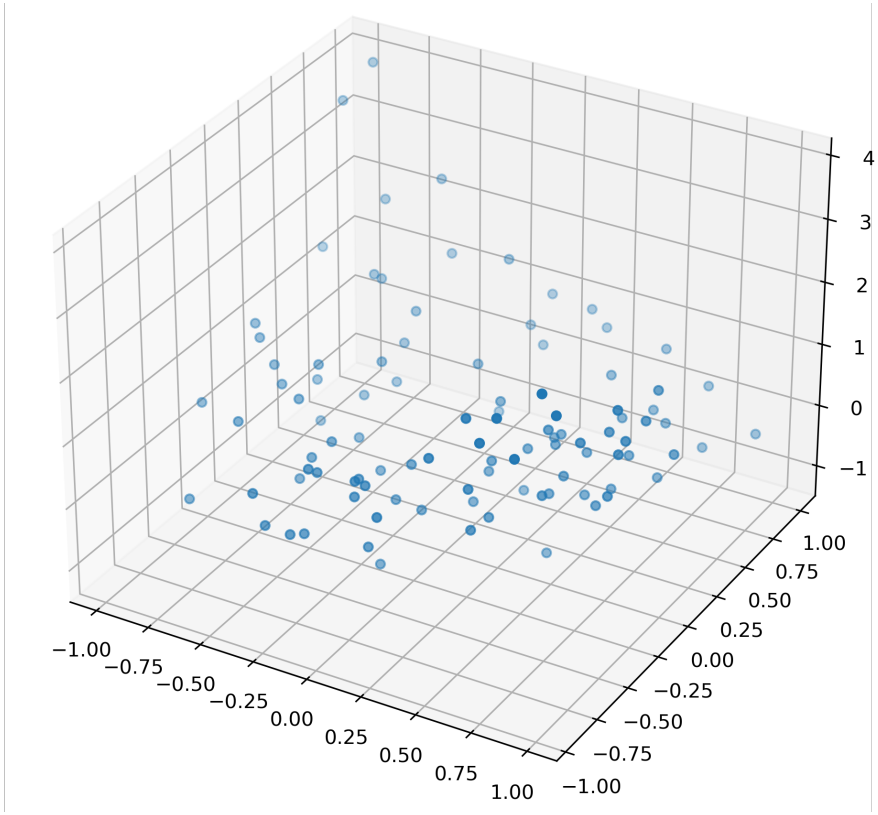


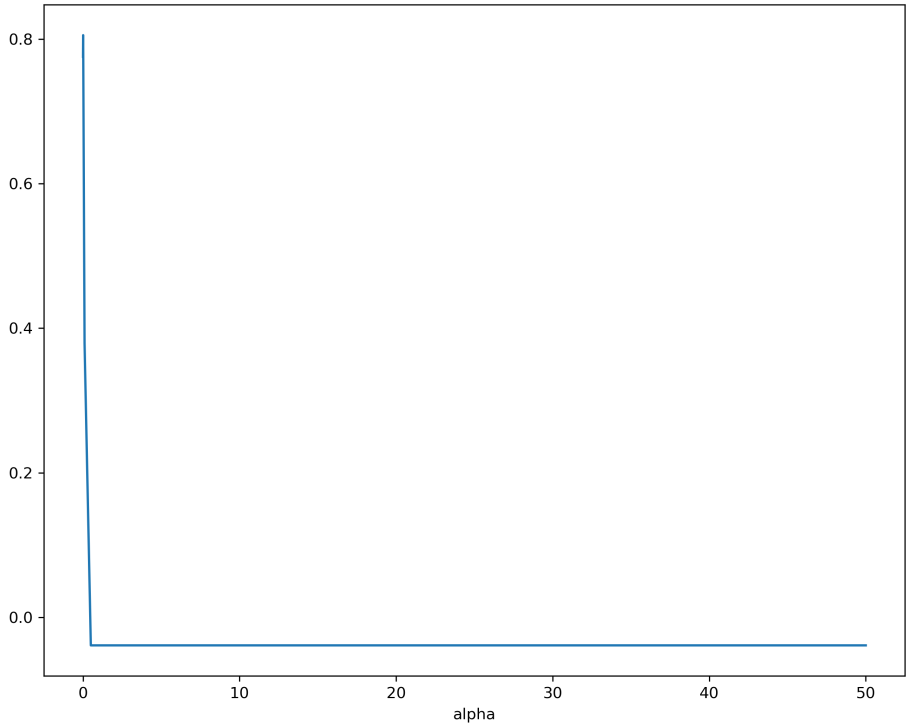
Discussion



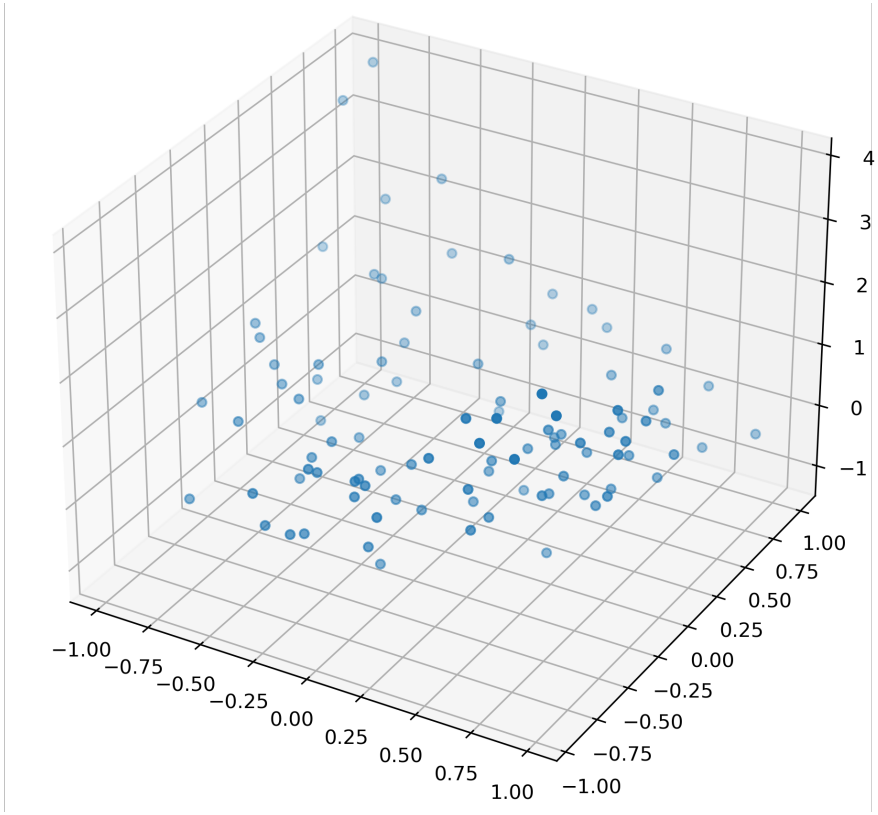


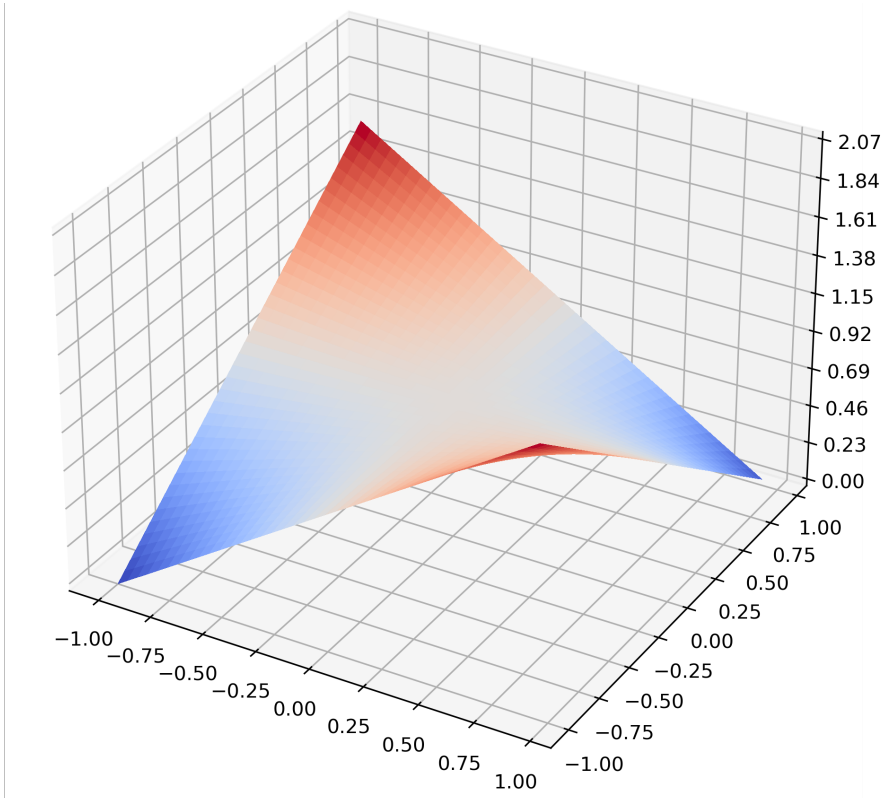
Discussion


4
3
2
1
0
−1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
alpha

Discussion


4
3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


2.07
1.84
1.61
1.38
1.15
0.92
0.69
0.46
0.23
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

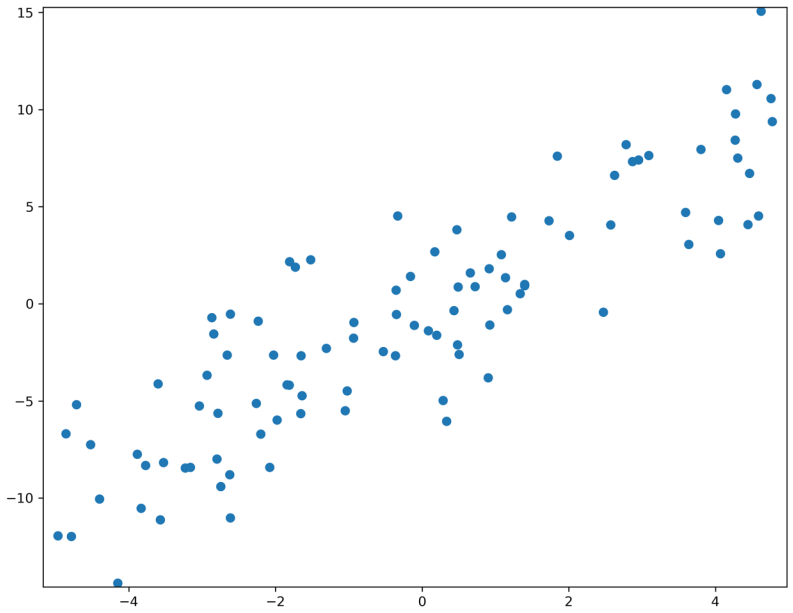
Implementation Using Scikit-Learn

Topic 04

Bayesian Regression

Step 01: Dataset Preparation

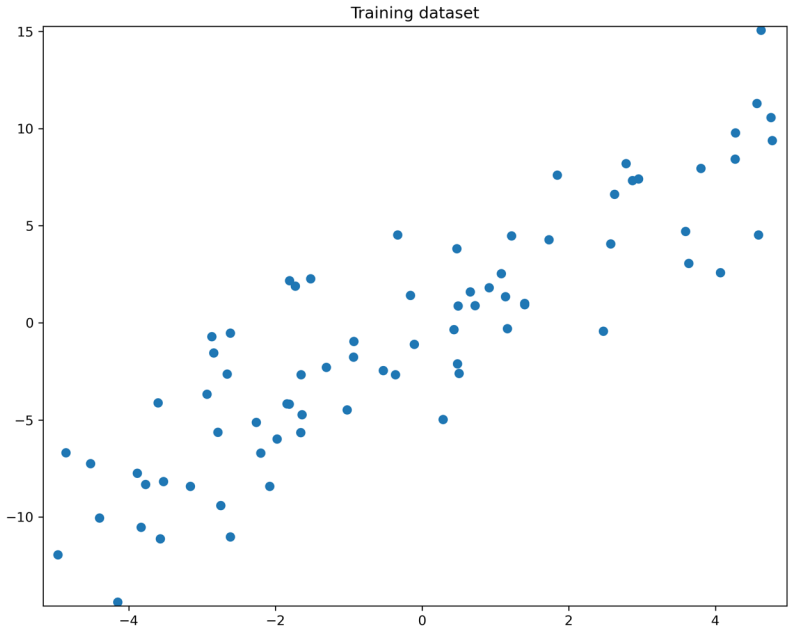
```
import numpy as np
X = np.random.uniform(-5,5,100)
y = 2*X + np.random.normal(0,3,100)
```

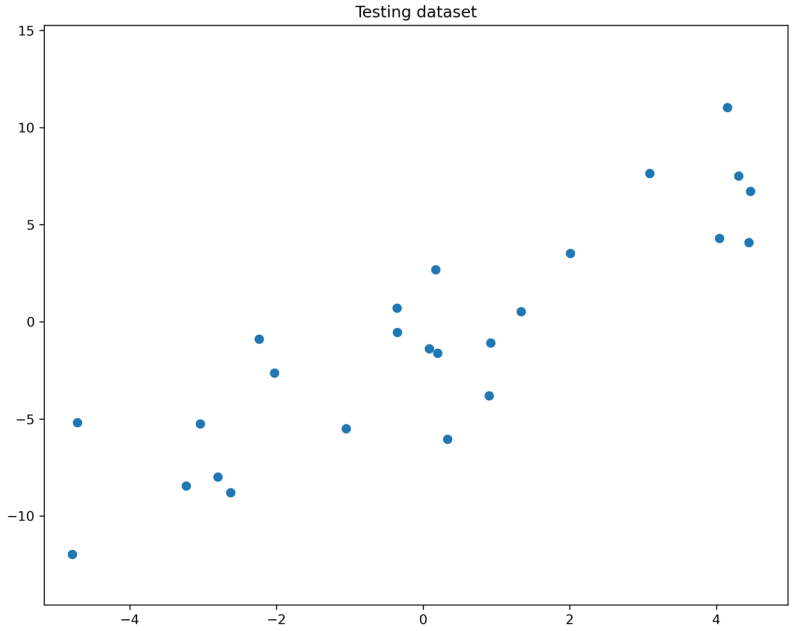


We are seeking for a Bayesian regression model $\hat{y} = w_0 + w_1 x$ to fit the data

Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X=X.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)
```





Step 03: Build/Train ML Model

- 'BayesianRidge' helps build the Bayesian regression in Scikit-learn
- Two prior distributions are assumed in 'BayesianRidge'
  - the prior distribution of the variance of observation follows the gamma distribution with parameters 'alpha_1' and 'alpha_2'
  - the prior distribution of the variance of weights follows the gamma distribution with parameters 'lambda_1' and 'lambda_2'

Step 03: Build/Train ML Model

```
from sklearn.linear_model import BayesianRidge
BayesianRidge_Model = BayesianRidge(
   n_iter=300, tol=0.001,
   alpha_1=1e-06, alpha_2=1e-06,
   lambda_1=1e-06, lambda_2=1e-06
)
```

- 'n_iter' determines the maximum number of iteration to find the weights of the models
- 'tol' is the precision of the solution

- The training of the model is to call the 'fit' method

```
BayesianRidge_Model.fit(X_train, y_train)
```

Step 04: Test the ML model

- When a new input comes, we can use the 'predict' method to predict its possible output

```
BayesianRidge_Model.predict([[1.6]])
```

  - its output is

```
array([3.27289468])
```

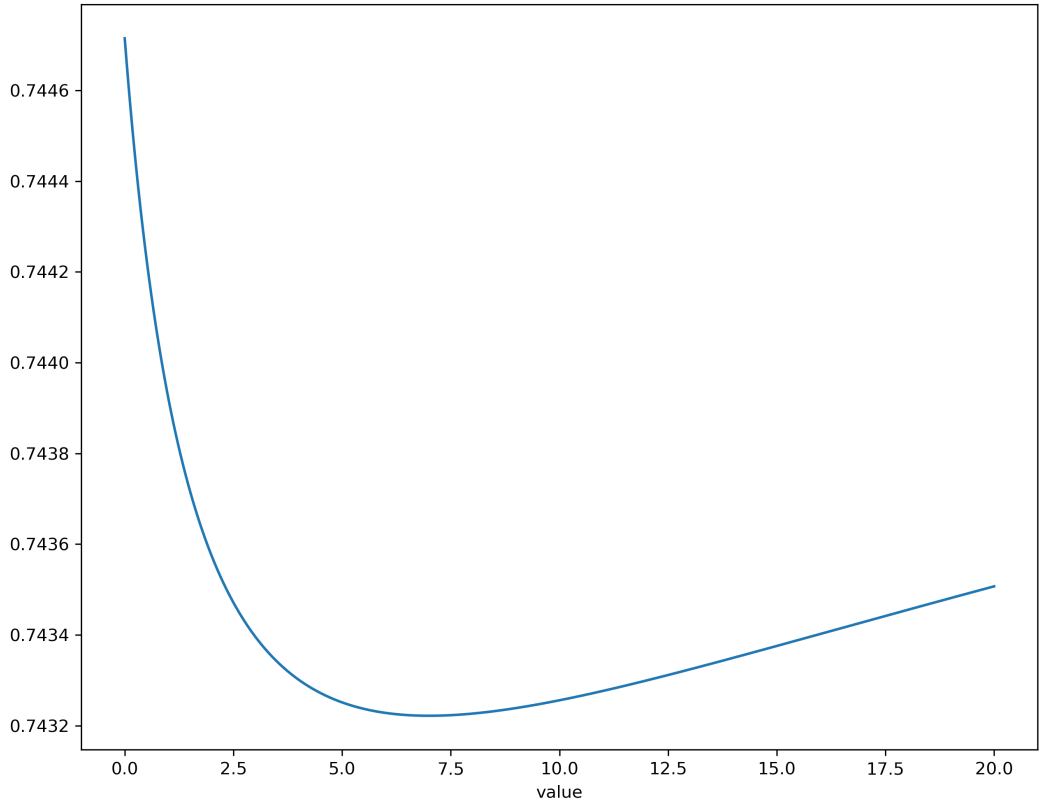
- The 'score' method is to evaluate how fit the model is

```
BayesianRidge_Model.score(X_test, y_test)
```

  - its output is

```
0.7440097954763434
```

Discussion


0.7446
0.7444
0.7442
0.7440
0.7438
0.7436
0.7434
0.7432
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
value

Discussion

- The bias and the weight of the fitting function
  - bias: w_0

```
BayesianRidge_Model.intercept_
```

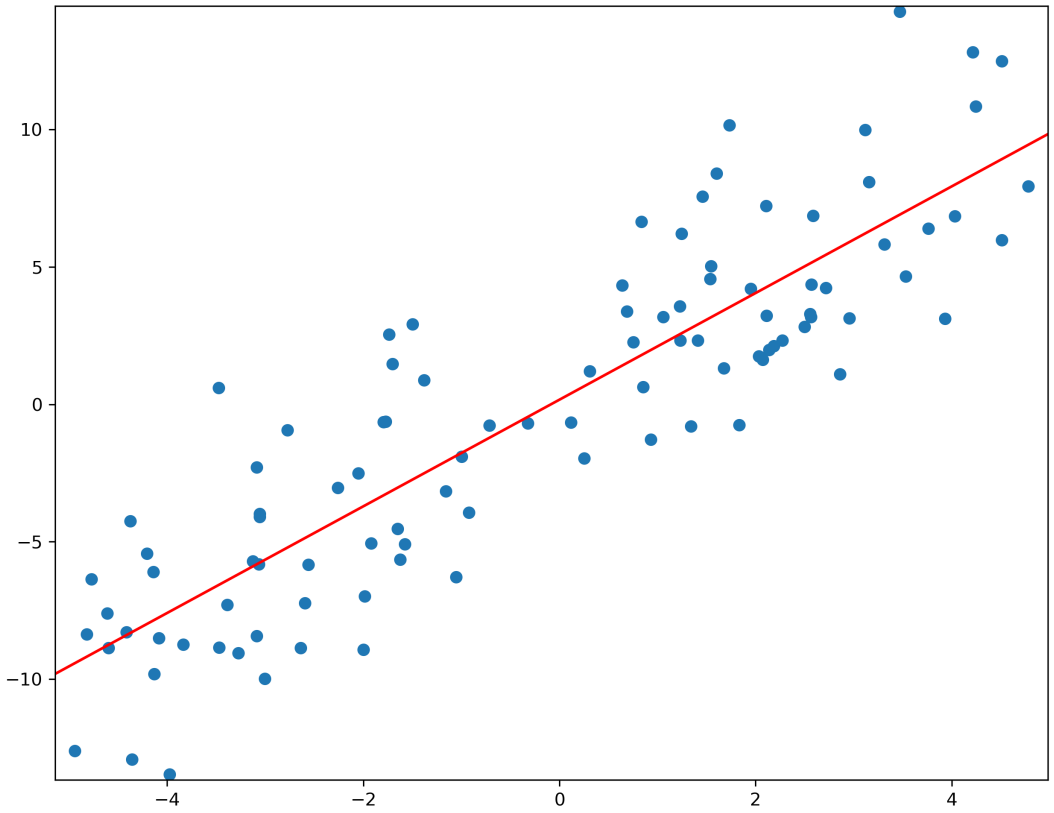
```
0.1670676923915525
```

  - the weight: w_1

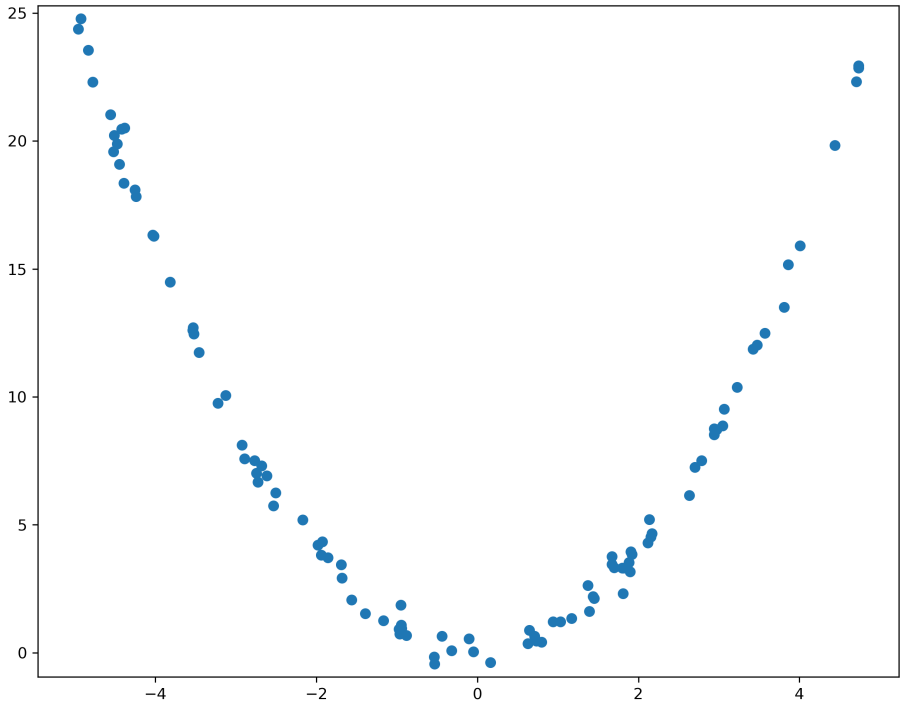
```
BayesianRidge_Model.coef_
```

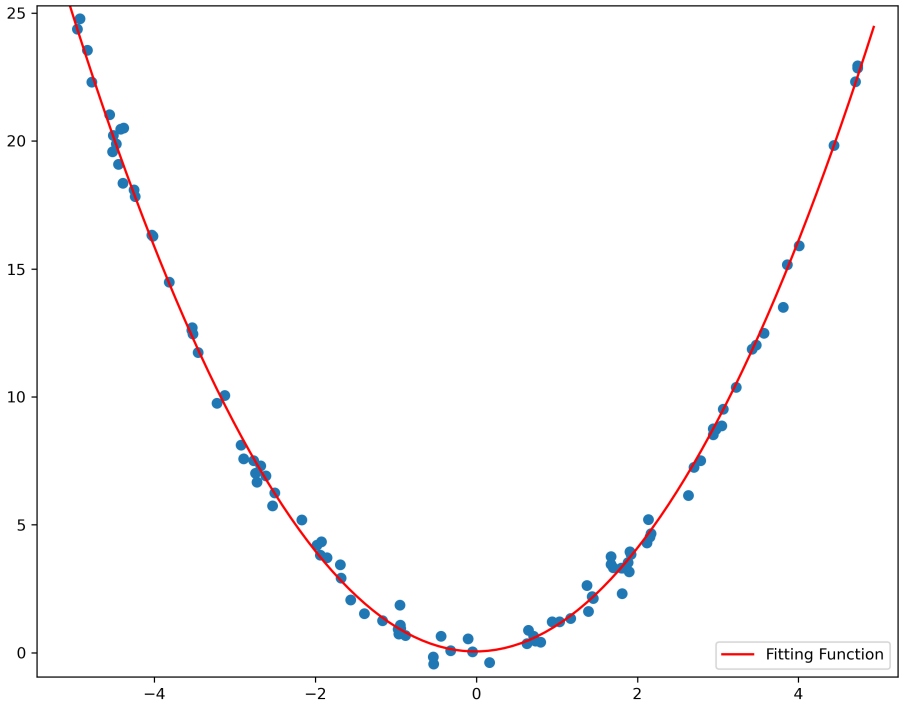
```
array([1.94114187])
```

Discussion

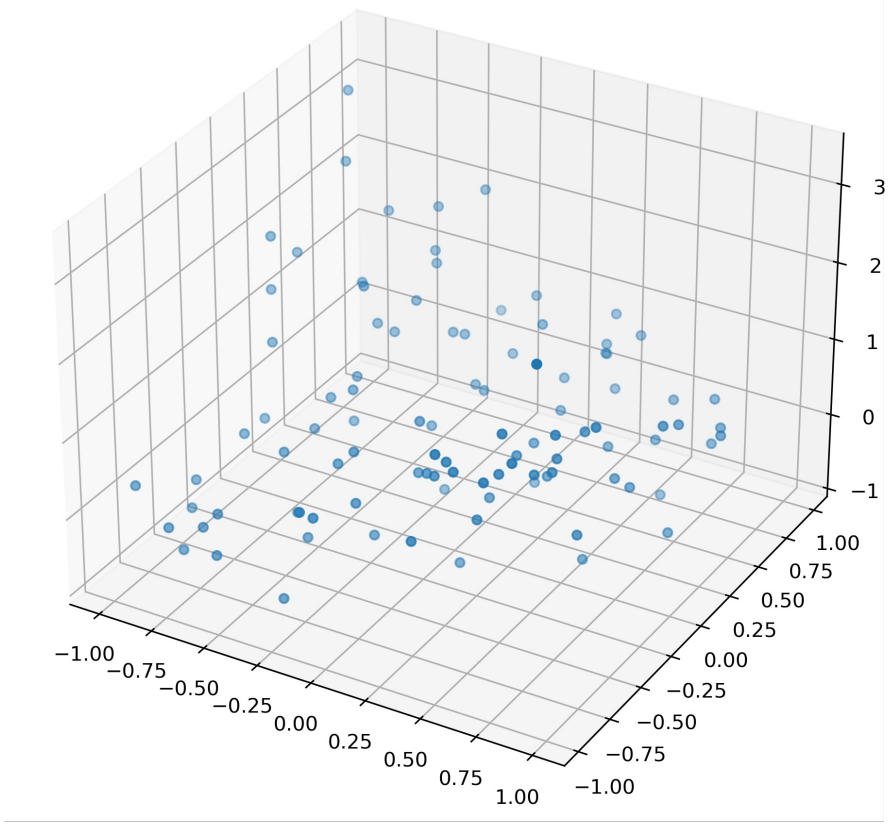


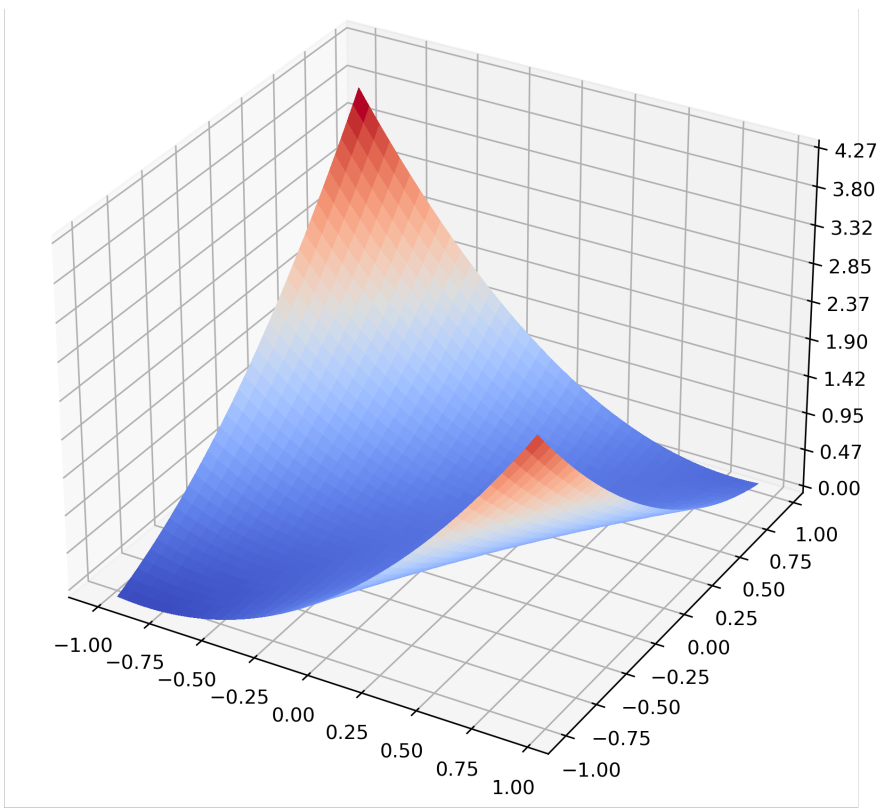
Discussion





Discussion


3
2
1
0
−1
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


4.27
3.80
3.32
2.85
2.37
1.90
1.42
0.95
0.47
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Q&A