

SLR with future salary prediction

Code

```
1 # SLR with future salary prediction
2
3 # import packages
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import pandas as pd
7
8 # call dataset and split data into iv and dv
9 ds = pd.read_csv(r'D:\1. Professional\Data Science\08-11-2023\Salary_Data.csv')
10
11 X = ds.iloc[:, :-1].values
12 y = ds.iloc[:, 1].values
13
14 # Lets split the data into 80-20%
15
16 from sklearn.model_selection import train_test_split
17
18 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 0)
19
20 # as the dataset has 2 attributes we use SLR algorithm
21
22 from sklearn.linear_model import LinearRegression
23
24 regressor = LinearRegression()
25
26 # Built regression model
27 regressor.fit(X_train, y_train)
28
29 # Test the model and prepare future prediction table
30 y_pred = regressor.predict(X_test)
31
32 # visualize train data point ( 24 data)
33 plt.scatter(X_train, y_train, color = 'red')
34 plt.plot(X_train, regressor.predict(X_train), color = 'blue')
35 plt.title('Salary vs Experience (Training set)')
36 plt.xlabel('Years of Experience')
37 plt.ylabel('Salary')
38 plt.show()
39
```

```

40 # visualize test data point
41 plt.scatter(X_test, y_test, color = 'red')
42 plt.plot(X_train, regressor.predict(X_train), color = 'blue')
43 plt.title('Salary vs Experience (Testing set)')
44 plt.xlabel('Years of Experience')
45 plt.ylabel('Salary')
46 plt.show()
47
48 # To get value of slope
49 m = regressor.coef_
50 m
51
52 # To get intercept or constant
53 c = regressor.intercept_
54 c
55
56 # predict or forecast the future the data which we not trained before
57 y_12 = int(m) * 12 + c
58 y_12
59
60 y_20 = int(m) * 20 + c
61 y_20
62
63
64 # to check overfitting ( low bias high variance)
65 bias = regressor.score(X_train, y_train)
66 bias
67
68
69 # to check underfitting (high bias low variance)
70 variance = regressor.score(X_test,y_test)
71 variance
72
73
74

```

Dataset

Index	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2	43525
4	2.2	39891
5	2.9	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218

Index	YearsExperience	Salary
19	6	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

X & y data sets

The image shows two side-by-side NumPy array viewer windows. The left window is titled 'X - NumPy object array' and displays a 1D array of 11 elements: [1.1, 1.3, 1.5, 2, 2.2, 2.9, 3, 3.2, 3.2, 3.7, 3.9]. The right window is titled 'y - NumPy object array' and displays a 1D array of 11 elements: [39343, 46285, 37731, 43525, 39891, 56642, 60150, 54445, 64445, 57189, 63218]. Both windows have a 'Format' button, a 'Resize' button, a checked 'Background color' checkbox, a 'Save and Close' button, and a 'Close' button.

	0
0	1.1
1	1.3
2	1.5
3	2
4	2.2
5	2.9
6	3
7	3.2
8	3.2
9	3.7
10	3.9

	0
0	39343
1	46285
2	37731
3	43525
4	39891
5	56642
6	60150
7	54445
8	64445
9	57189
10	63218

Train test split data

The image shows five NumPy array viewer windows arranged in a grid. The top row contains three windows: 'X - NumPy object array' (same as before), 'X_train - NumPy object array' (first 10 elements: [9.6, 4, 5.3, 7.9, 2.9, 5.1, 3.2, 4.5, 8.2, 6.8]), and 'X_test - NumPy object array' (last element: [1.3]). The bottom row contains two windows: 'y_test - NumPy object array' (first 5 elements: [37731, 122391, 57081, 63218, 116969]) and 'y_train - NumPy object array' (last 6 elements: [56642, 60150, 54445, 64445, 57189, 63218]). All windows have the same controls as the previous ones.

	0
0	1.1
1	1.3
2	1.5
3	2
4	2.2
5	2.9
6	3
7	3.2
8	3.2
9	3.7
10	3.9

	0
0	9.6
1	4
2	5.3
3	7.9
4	2.9
5	5.1
6	3.2
7	4.5
8	8.2
9	6.8
10	1.3

	0
0	1.5
1	10.3
2	4.1
3	3.9
4	9.5
5	8.7

	0
0	37731
1	122391
2	57081
3	63218
4	116969
5	109431

	0
0	112635
1	55794
2	83088
3	101302
4	56642
5	66029
6	64445
7	61111
8	113812

80 – 20 % split results



```
In [19]: m = regressor.coef_  
...: m  
Out[19]: array([9312.57512673])  
  
In [20]: c = regressor.intercept_  
...: c  
Out[20]: 26780.099150628186  
  
In [21]: y_12 = int(m) * 12 + c  
...: y_12  
Out[21]: 138524.0991506282  
  
In [22]: y_20 = int(m) * 20 + c  
...: y_20  
Out[22]: 213020.0991506282  
  
In [23]:
```

```
In [23]: bias = regressor.score(X_train, y_train)  
...: bias  
Out[23]: 0.9411949620562126  
  
In [24]: variance = regressor.score(X_test, y_test)  
...: variance  
Out[24]: 0.988169515729126  
  
In [25]:
```

75 – 25 % split results



```
In [13]: m = regressor.coef_
...: m
Out[13]: array([9379.71049195])

In [14]: c = regressor.intercept_
...: c
Out[14]: 26986.691316737248

In [15]: y_12 = int(m) * 12 + c
...: y_12
Out[15]: 139534.69131673724

In [16]: y_20 = int(m) * 20 + c
...: y_20
Out[16]: 214566.69131673724

In [17]:
```

```
In [17]: bias = regressor.score(X_train, y_train)
...: bias
Out[17]: 0.9395413526983522

In [18]: variance = regressor.score(X_test, y_test)
...: variance
Out[18]: 0.9779208335417602

In [19]:
```

70 – 30 % split results



```
In [13]: m = regressor.coef_  
...: m  
Out[13]: array([9360.26128619])  
  
In [14]: c = regressor.intercept_  
...: c  
Out[14]: 26777.391341197632  
  
In [15]: y_12 = int(m) * 12 + c  
...: y_12  
Out[15]: 139097.39134119762  
  
In [16]: y_20 = int(m) * 20 + c  
...: y_20  
Out[16]: 213977.39134119762  
  
In [17]:
```

```
In [17]: bias = regressor.score(X_train, y_train)  
...: bias  
Out[17]: 0.9423777652193379  
  
In [18]: variance = regressor.score(X_test, y_test)  
...: variance  
Out[18]: 0.9740993407213511  
  
In [19]:
```