lab9

November 3, 2024

1 STOR 320: Introduction to Data Science

1.1 Lab 9

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
```

1. We use the following code to generate two columns of features and the target values. Based on the code below, what is the true linear model between `Target` and two features?

[ ]:
```
np.random.seed(0)

# Generate feature matrix X (100 samples, 2 features)
X = np.random.rand(100, 2)

# Define true coefficients for generating y
true_beta = np.array([5, 2, -3])  # Intercept, beta_1, beta_2

# Generate y with some added noise
y = true_beta[0] + X @ true_beta[1:] + np.random.normal(0, 0.5, X.shape[0])

# Combine X and y into a pandas DataFrame
data = pd.DataFrame(np.column_stack((X, y)), columns=['Feature_1', 'Feature_2',
 ↪'Target'])
data
```

[ ]:

	Feature_1	Feature_2	Target
0	0.548814	0.715189	4.515377
1	0.602763	0.544883	4.030911
2	0.423655	0.645894	3.335893
3	0.437587	0.891773	2.980945
4	0.963663	0.383442	5.527985
..	…	…	…
95	0.398221	0.209844	4.879017
96	0.186193	0.944372	2.610245

```
97    0.739551    0.490459   4.848061
98    0.227415    0.254356   5.037529
99    0.058029    0.434417   4.160183

[100 rows x 3 columns]
```

The true model is $y = 5 + 2 * F_1 - 3 * F_2$

2. Based on `data` table, separate X (features) and y (target) from the data. In other words, create a `100*2` numpy matrix for X and a numpy vector for y.

[ ]:
```python
x = data[["Feature_1", "Feature_2"]]
y = data["Target"]
```

[ ]:
```python
x
```

[ ]:
```
    Feature_1  Feature_2
0    0.548814   0.715189
1    0.602763   0.544883
2    0.423655   0.645894
3    0.437587   0.891773
4    0.963663   0.383442
..        …          …
95   0.398221   0.209844
96   0.186193   0.944372
97   0.739551   0.490459
98   0.227415   0.254356
99   0.058029   0.434417

[100 rows x 2 columns]
```

[ ]:
```python
y
```

[ ]:
```
0     4.515377
1     4.030911
2     3.335893
3     2.980945
4     5.527985
        …
95    4.879017
96    2.610245
97    4.848061
98    5.037529
99    4.160183
Name: Target, Length: 100, dtype: float64
```

3. Add a column of ones to X to account for the intercept term in the coefficient vector.

[ ]:
```
x_with_intercept = np.column_stack((np.ones(x.shape[0]), x))
x_with_intercept
```

[ ]:
```
array([[1.        , 0.5488135 , 0.71518937],
       [1.        , 0.60276338, 0.54488318],
       [1.        , 0.4236548 , 0.64589411],
       [1.        , 0.43758721, 0.891773  ],
       [1.        , 0.96366276, 0.38344152],
       [1.        , 0.79172504, 0.52889492],
       [1.        , 0.56804456, 0.92559664],
       [1.        , 0.07103606, 0.0871293 ],
       [1.        , 0.0202184 , 0.83261985],
       [1.        , 0.77815675, 0.87001215],
       [1.        , 0.97861834, 0.79915856],
       [1.        , 0.46147936, 0.78052918],
       [1.        , 0.11827443, 0.63992102],
       [1.        , 0.14335329, 0.94466892],
       [1.        , 0.52184832, 0.41466194],
       [1.        , 0.26455561, 0.77423369],
       [1.        , 0.45615033, 0.56843395],
       [1.        , 0.0187898 , 0.6176355 ],
       [1.        , 0.61209572, 0.616934  ],
       [1.        , 0.94374808, 0.6818203 ],
       [1.        , 0.3595079 , 0.43703195],
       [1.        , 0.6976312 , 0.06022547],
       [1.        , 0.66676672, 0.67063787],
       [1.        , 0.21038256, 0.1289263 ],
       [1.        , 0.31542835, 0.36371077],
       [1.        , 0.57019677, 0.43860151],
       [1.        , 0.98837384, 0.10204481],
       [1.        , 0.20887676, 0.16130952],
       [1.        , 0.65310833, 0.2532916 ],
       [1.        , 0.46631077, 0.24442559],
       [1.        , 0.15896958, 0.11037514],
       [1.        , 0.65632959, 0.13818295],
       [1.        , 0.19658236, 0.36872517],
       [1.        , 0.82099323, 0.09710128],
       [1.        , 0.83794491, 0.09609841],
       [1.        , 0.97645947, 0.4686512 ],
       [1.        , 0.97676109, 0.60484552],
       [1.        , 0.73926358, 0.03918779],
       [1.        , 0.28280696, 0.12019656],
       [1.        , 0.2961402 , 0.11872772],
       [1.        , 0.31798318, 0.41426299],
       [1.        , 0.0641475 , 0.69247212],
       [1.        , 0.56660145, 0.26538949],
       [1.        , 0.52324805, 0.09394051],
```

```
[1.        , 0.5759465 , 0.9292962 ],
[1.        , 0.31856895, 0.66741038],
[1.        , 0.13179786, 0.7163272 ],
[1.        , 0.28940609, 0.18319136],
[1.        , 0.58651293, 0.02010755],
[1.        , 0.82894003, 0.00469548],
[1.        , 0.67781654, 0.27000797],
[1.        , 0.73519402, 0.96218855],
[1.        , 0.24875314, 0.57615733],
[1.        , 0.59204193, 0.57225191],
[1.        , 0.22308163, 0.95274901],
[1.        , 0.44712538, 0.84640867],
[1.        , 0.69947928, 0.29743695],
[1.        , 0.81379782, 0.39650574],
[1.        , 0.8811032 , 0.58127287],
[1.        , 0.88173536, 0.69253159],
[1.        , 0.72525428, 0.50132438],
[1.        , 0.95608363, 0.6439902 ],
[1.        , 0.42385505, 0.60639321],
[1.        , 0.0191932 , 0.30157482],
[1.        , 0.66017354, 0.29007761],
[1.        , 0.61801543, 0.4287687 ],
[1.        , 0.13547406, 0.29828233],
[1.        , 0.56996491, 0.59087276],
[1.        , 0.57432525, 0.65320082],
[1.        , 0.65210327, 0.43141844],
[1.        , 0.8965466 , 0.36756187],
[1.        , 0.43586493, 0.89192336],
[1.        , 0.80619399, 0.70388858],
[1.        , 0.10022689, 0.91948261],
[1.        , 0.7142413 , 0.99884701],
[1.        , 0.1494483 , 0.86812606],
[1.        , 0.16249293, 0.61555956],
[1.        , 0.12381998, 0.84800823],
[1.        , 0.80731896, 0.56910074],
[1.        , 0.4071833 , 0.069167  ],
[1.        , 0.69742877, 0.45354268],
[1.        , 0.7220556 , 0.86638233],
[1.        , 0.97552151, 0.85580334],
[1.        , 0.01171408, 0.35997806],
[1.        , 0.72999056, 0.17162968],
[1.        , 0.52103661, 0.05433799],
[1.        , 0.19999652, 0.01852179],
[1.        , 0.7936977 , 0.22392469],
[1.        , 0.34535168, 0.92808129],
[1.        , 0.7044144 , 0.03183893],
[1.        , 0.16469416, 0.6214784 ],
```

```
       [1.        , 0.57722859, 0.23789282],
       [1.        , 0.934214  , 0.61396596],
       [1.        , 0.5356328 , 0.58990998],
       [1.        , 0.73012203, 0.311945  ],
       [1.        , 0.39822106, 0.20984375],
       [1.        , 0.18619301, 0.94437239],
       [1.        , 0.7395508 , 0.49045881],
       [1.        , 0.22741463, 0.25435648],
       [1.        , 0.05802916, 0.43441663]])
```

4. Calculate the estimation of parameter β manually using NumPy's matrix operations.

- Hint: You can refer to the lecture notes to find the closed-form solution for the regression coefficients
- Hint: You can use `np.linalg.inv` to calculate the inverse of a matrix.

[ ]:
```
XTX_inv = np.linalg.inv(x_with_intercept.T @ x_with_intercept)
beta_hat = XTX_inv @ x_with_intercept.T @ y
print(f"Manually calculated beta: {beta_hat}")
```

```
Manually calculated beta: [ 5.05725163  1.78685022 -2.98521247]
```

5. Compute R squared manually. You can refer to the lecture notes to see the definitino of R-squared.

[ ]:
```
y_pred = x_with_intercept @ beta_hat
SS_res = np.sum((y - y_pred) ** 2)
SS_total = np.sum((y - np.mean(y)) ** 2)
R_squared = 1 - (SS_res/SS_total)
R_squared
```

[ ]:
```
0.8308337212672314
```

6. **Comparison with statsmodels**

We fit a linear regression model using `statsmodels`, then print and compare both the manually calculated coefficients and R-squared values with those from statsmodels. Are they the same?

[ ]:
```
x_sm = sm.add_constant(x)
mod = sm.OLS(y, x_sm)
results = mod.fit()
results.summary()
```

[ ]:

Dep. Variable:	Target	**R-squared:**	0.831
Model:	OLS	**Adj. R-squared:**	0.827
Method:	Least Squares	**F-statistic:**	238.2
Date:	Sun, 03 Nov 2024	**Prob (F-statistic):**	3.74e-38
Time:	11:37:57	**Log-Likelihood:**	-64.212
No. Observations:	100	**AIC:**	134.4
Df Residuals:	97	**BIC:**	142.2
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.0573	0.130	39.044	0.000	4.800	5.314
Feature_1	1.7869	0.166	10.750	0.000	1.457	2.117
Feature_2	-2.9852	0.164	-18.217	0.000	-3.310	-2.660

Omnibus:	0.410	**Durbin-Watson:**	2.045
Prob(Omnibus):	0.815	**Jarque-Bera (JB):**	0.502
Skew:	0.144	**Prob(JB):**	0.778
Kurtosis:	2.807	**Cond. No.**	5.58

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[ ]:
```

```

[ ]:
```

```