Lab1 - Linear regression (Bike Rental dataset)

WEN Yue
CHEN MengYu

1. Before linear regression

(1) About this dataset

The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant.

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90

(2) Basic analyze

After using command ***data.info()***, we can see that there are no NULLs in this dataset, which means we do not need to clean this dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9568 entries, 0 to 9567
Data columns (total 5 columns):
AT    9568 non-null float64
V     9568 non-null float64
AP    9568 non-null float64
RH    9568 non-null float64
PE    9568 non-null float64
dtypes: float64(5)
memory usage: 373.8 KB
```

2. Scikit Learn

(1) We split the data set into two parts.

X = data[['AT', 'V', 'AP', 'RH']]
y = data[['PE']]

(2) We use ***train_test_split*** method to split our data set (X, y) into four parts: X_train, X_test, y_train, y_test.
(3) By using the linear regression model in sklearn, we can get the linear regression formula：

$$PE=460.05727267-1.96865472*AT-0.2392946*V+0.0568509*AP-0.15861467*RH$$

(4) Through the formula we can get the following values

```
MSE: 20.837191547220353
RMSE: 4.564777272465805
MAE: 3.676950252836232
R2: [0.92979357]
```

3. Custom method

(1) We split the data set into two parts.

train = data[0:7176]
test = data[7176:9568]

(2) We define y_test as 'PE'. We define y_test as 'PE', modify 'PE' in test to NaN

	AT	V	AP	RH	PE
7176	10.52	41.78	1013.54	71.52	NaN
7177	26.90	60.93	1007.10	67.32	NaN
7178	24.86	44.05	1005.69	66.65	NaN
7179	13.39	49.83	1007.14	90.88	NaN
7180	27.98	71.98	1005.58	81.00	NaN

(3) Because sometimes the independent variable and the dependent variable may be more than the relationship of y=ax+b, so we have added some new variables, hoping to get a better training model and get better results. These variables include AT*AT, V*V, AP*AP, RH*RH, AT*V, V*AP, AP*RH, RH*AT (after data standardization)

	AT	V	AP	RH	AT_AT	V_V	AP_AP	RH_RH	AT_V	V_AP	AP_RH	RH_AT
0	-0.629519	-0.987297	1.820488	-0.009519	0.396295	0.974755	3.314178	0.000091	0.621522	-1.797362	-0.017330	0.005993
1	0.741909	0.681045	1.141863	-0.974621	0.550429	0.463822	1.303851	0.949885	0.505274	0.777660	-1.112883	-0.723080
2	-1.951297	-1.173018	-0.185078	1.289840	3.807561	1.375970	0.034254	1.663686	2.288906	0.217099	-0.238720	-2.516861
3	0.162205	0.237203	-0.508393	0.228160	0.026311	0.056265	0.258463	0.052057	0.038476	-0.120592	-0.115995	0.037009
4	-1.185069	-1.322539	-0.678470	1.596699	1.404388	1.749109	0.460322	2.549449	1.567299	0.897303	-1.083313	-1.892198

(4) Define a random gradient drop function,

$$\theta_j := \theta_j - \alpha \frac{1}{m}(y^i - h_\theta(x^i))x_j^i$$

```python
def step_grad(points, learn_rate, M):
    N = points.shape[0]
    num_col = points.shape[1]

    new_M = np.zeros(num_col)

    for i in range(N):
        x = points[i, 0:num_col-1]
        x = np.append(x, 1)
        y = points[i, num_col-1]
        for j in range(num_col):
            new_M[j] += (-2/N) * (y - (M * x).sum()) * x[j]
        M = M - (learn_rate * new_M)

    return M
```

(5) Define grad_desc

```python
def grad_desc(points, learn_rate, num_iter):
    num_col = points.shape[1]
    M = np.zeros(num_col)

    for i in range(num_iter):
        M = step_grad(points, learn_rate, M)
        if i % 100== 0:
            print(i, "Cost= ", cost(points, M))
    print(i, "Cost= ", cost(points, M))
    return M
```

(6) Define cost function,

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

```python
def cost(points, M):
    total_cost = 0

    N = points.shape[0]
    num_col = points.shape[1]

    for i in range(N):
        x = points[i,0:num_col-1]
        x = np.append(x, 1)
        y = points[i,num_col-1]
        total_cost += (y - (M * x).sum()) ** 2

    total_cost = (1/N) * total_cost
    return total_cost
```

(7) Define run function.

```python
def run():
    data = data_train_1[:,0:data_train_1.shape[1]]
    # tweak learn_rate & num_iter to get better results; 0.001, 1000
    learn_rate = 0.0001
    num_iter = 1000

    m = grad_desc(data, learn_rate, num_iter)
    #print(m)
    return m[0:data_train_1.shape[1]-1], m[data_train_1.shape[1]-1]
```

(8) By calculating, we can get the correlation coefficient of the equation: m, c：

```
m
```

```
array([-13.61642886,  -3.6311436 ,   0.79233231,  -1.91396256,
         0.92490933,  -0.12851954,  -0.25278515,  -0.3947235 ,
         1.00681491,   0.30883716,  -0.40036628,  -0.6160933 ])
```

```
c
```

```
453.1648310371076
```

(9) Next, we can make predictions with trained models.

(10) Through the formula we can get the following values

```
MSE: 19.283391319352173
RMSE: 4.391285838948789
MAE: 3.397251451021495
R2: [0.9335685]
```

4. Conclusions

By comparing R2 of the two methods, we can see that the accuracy of custom functions is higher than that predicted by sklearn. This may be because we have further processed the dataset, adding new parameters (AT*AT, V*V, etc.). This operation brings the trained model equation closer to the optimal solution.