1. Learning Basics of Regression in Python

[1] Data Description summarization

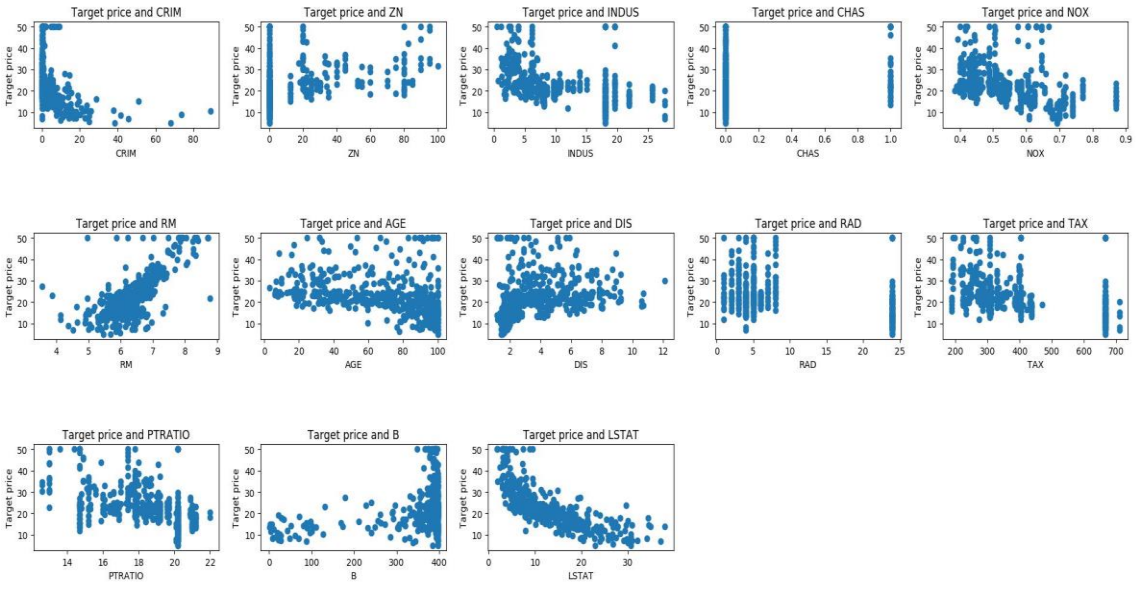
- Number of data points:
  It means the total number of training data and test data. In Q1.py, it is the numbers of column, namely np.shape(X)[0]. In this assignment the number of data points is 506.

- Features:
  It means the total dimension of data. In Q1.py, it is the numbers of row, namely np.shape(X)[1]. In this assignment the number of data points is 13, including ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO' 'B' 'LSTAT']

- Target:
  In Q1.py, it is the price of Boston houses, namely y In this assignment the number of data points is the median value of owner-occupied homes price in $1000's

[2] Visualization

In the plots, the dependent variable is the price of Boston houses, and the independent variables are features.



[3] Dividing data

In Q1.py, I use train_test_split to divide the data into 80% training data and 20% testing data.

```
my_list = [i for i in range(X.shape[0])]
random.shuffle(my_list)
test_num = int(X.shape[0] * 0.2)
train_index = my_list
test_index = []
for i in range(test_num):
    test_index.append(my_list[i])
    del train_index[i]
q1_x_train = X[train_index]
q1_y_train = y[train_index]
q1_x_test = X[test_index]
q1_y_test = y[test_index]
```

[4] Linear Regression

- Add bias:
  Add a column of 1 into data x

```
x = np.reshape(np.c_[np.ones(n_training_samples), X], [n_training_samples, n_dim + 1])
```

- X is the input data (with bias) and y is output price. $f_w(x)$ is prediction of y.
  $f_w(x) = w_0 + w_1x_1 + \cdots + w_dx_d = \mathbf{x}^T\mathbf{w}$ for w ∈ R^{d+1}

```
q1_y_predicts = np.dot(x_test, w)
```

- The optimal w* in w can be represented as follows:

$$w^* = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

- The optimal weight can be solved using np.linalg.solve function in python.

```
A = np.dot(x.T, x)
B = np.dot(x.T, y)
weights = np.linalg.solve(A, B)
```

[5] Table of feature and associated weight

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
-0.11	0.05	-0.01	2.69	-16	3.77	0.001	-1.6	0.32	-0.01	-0.92	0.01	-0.57

The third column (INDUS) means the proportion of non-retail business acres per town. The sign of INDUS is negative, which means that the more proportion INDUS is, the less the house price is. The sign does not match my

exception. Because, INDUS means less place to build houses that will cause house price raise. But the weight of INDUS is little. It makes less effect to house price.

[6] Error Matrix

1. **MSE:**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Y}_i - Y_i)^2$$

I also choose root mean square error and mean absolute error to evaluate the algorithm.

2. **RMSE:** $RMSE = \sqrt{MSE}$

3. **MAE:** $MAE = \frac{\sum_{i=1}^{n}|\widehat{Y}_i - Y_i|}{n}$

```
sum_mean = 0
abs_mean = 0
for i in range(len(q1_y_predicts)):
    sum_mean += (q1_y_predicts[i] - y_test[i]) ** 2
    abs_mean += abs(q1_y_predicts[i] - y_test[i])
sum_error_rmse = np.sqrt(sum_mean / len(q1_y_predicts))
sum_error_mse = sum_mean / len(q1_y_predicts)
sum_error_mae = abs_mean / len(q1_y_predicts)
```

4. **Result:**

```
MSE:   [ 16.42118452]
RMSE:   [ 4.05230607]
MAE:   [ 2.97246208]
```

[7] Feature Selection _

Among all the features, the most significant features are the features with large weights, namely, CHAS, NOX, RM, DIS. If the x changes, the y will change a lot.

2. Locally reweighted regression

[1] Weighted least square

The optimal **w**, $w^* = \arg min \frac{1}{2}\sum_{i=1}^{N} a^{(i)}(y^{(i)} - w^T x^{(i)})^2 + \frac{\lambda}{2}\|w\|^2$

We can get:

$$\mathrm{L(w)} = \frac{A}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|^2 = A(y - Xw)^T(\mathrm{y - Xw}) + \frac{\lambda}{2}w^T w$$

To get the optimal w, we get the gradient of L(w):

$\nabla \mathrm{L}(w^*) = (X^T AX + \lambda I)w^* - X^T Ay = 0$

Thus, we can get:

$w^* = (X^T AX + \lambda I)^{-1} X^T Ay$

[2] Locally reweighted least squares

To decrease the time complexity of the algorithm, I use matrix calculate instead of the loop. L2 is used to calculate the distance and dialogue matric to regulate A into dialogue matric.

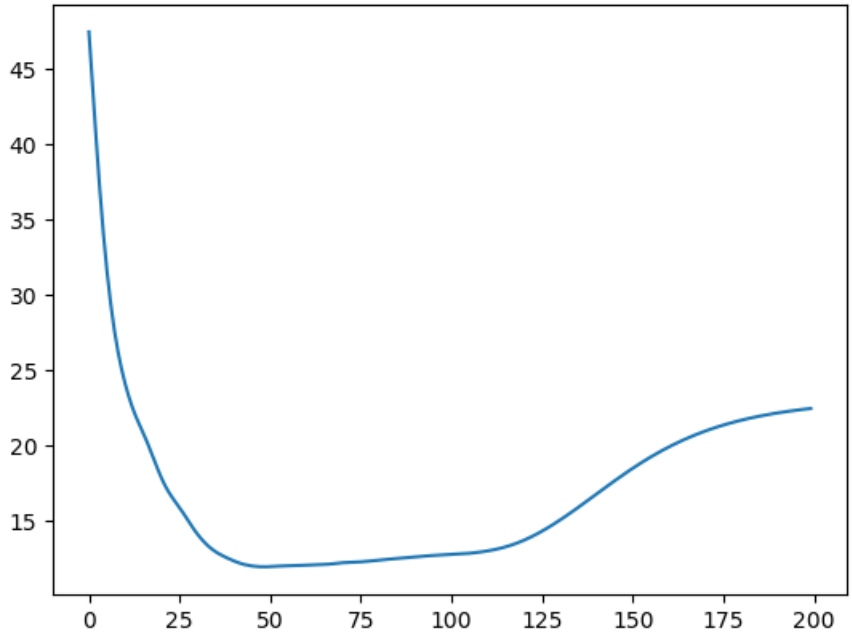
```
x_T = np.mat(x_train)
y_T = np.mat(y_train).T
m = np.shape(x_T)[0]
test = test_datum.T * np.ones([m, 1])
A = np.mat(l2(test, x_train)/(-2.0*tau**2))
A_max = A.max()
A = np.multiply(np.exp(A-A_max), np.mat(np.eye(m)))
A = A/np.sum(A)
xTAx = np.dot(np.dot(x_T.T, A), x_T) + lam*np.mat(np.eye(d))
xTAy = np.dot(np.dot(x_T.T, A), y_T)
weights = np.linalg.solve(xTAx, xTAy)
return np.dot(test_datum.T, weights)
```

[3] K-fold

In Q2, 5-fold cross validation is used to validation the algorithm. KFold() is used to split the data into 5 folds. In each turn, one fold is used to valid the algorithm and the others are used to train.

```
    my_list = [i for i in range(x.shape[0])]
    np.random.shuffle(my_list)
    for n in range(k):
        fold_test_index = []
        fold_train_index = []
        for j in range(x.shape[0]):
            if j % k == n:
                fold_test_index.append(my_list[j])
            else:
                fold_train_index.append(my_list[j])
        train_x, valid_x = x[fold_train_index], x[fold_test_index]
        train_y, valid_y = y[fold_train_index], y[fold_test_index]
        if n == 0:
            los = np.array(run_on_fold(valid_x, valid_y, train_x, train_y, taus))
        else:
            los += np.array(run_on_fold(valid_x, valid_y, train_x, train_y, taus))
    return los/k
```

Finally, we can get the graph from $\boldsymbol{\tau}$ [10,1000], we can get the result as fellow.



Min loss = 11.88667

[4] The algorithm behaves with regard to $\boldsymbol{\tau}$

According to:

$$a^i = \frac{\exp(-\|x - x^{(i)}\|^2/2\tau^2)}{\sum_j \exp(-\|x - x^{(j)}\|^2/2\tau^2)}$$

When $\boldsymbol{\tau} \to \infty$, $a^i \to 1$, the algorithm will approach to linear regression. It will cause under-fitting.

When $\boldsymbol{\tau} \to \mathbf{0}$, $a^i \to \infty$, the algorithm will approach to K-NN algorithm and become sensitive. It will cause over-fitting.

3. Learning Basics of Regression in Python

[1] We want to prove that $E_L\left[\frac{1}{m}\sum_{i\in L} a_i\right] = \frac{1}{n}\sum_{i=1}^{n} a_i$

Because the expectation is linear, we can get:

$$E[X+Y] = E[X] + E[Y]$$

In this question, by using the linearity of expectation:

$$E_L\left[\frac{1}{m}\sum_{i\in L} a_i\right] = \frac{1}{m}\sum_{i=1}^{m} E_L\,[a_i]$$

According to the definition of expectation:

$$E_L\,[a_i] = \sum_{j=1}^{n}\frac{1}{n}a_j$$

Equally, $E_L\left[\frac{1}{m}\sum_{i\in L} a_i\right] = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{n}a_j = \sum_{i=1}^{n}\frac{1}{n}a_i$

[2] We want to prove that $E_L[\nabla L_{\mathcal{L}}(x,y,\theta)] = \nabla L(x,y,\theta)$

According to loss function and linearity of differentiation

right hand side = $\nabla L(x,y,\theta) = \frac{1}{n}\sum_{i=1}^{n}\nabla\ell(x^{(i)},y^{(i)},\theta)$

left hand side = $E_L[\nabla L_{\mathcal{L}}(x,y,\theta)] = E_L\left[\frac{1}{m}\sum_{i=1}^{m}\nabla\ell(x^{(i)},y^{(i)},\theta)\right]$

$$E_L\left[\frac{1}{m}\sum_{i=1}^{m}\nabla\ell(x^{(i)},y^{(i)},\theta)\right] = \frac{1}{m}\sum_{i=1}^{m}E_L[\nabla\ell(x^{(i)},y^{(i)},\theta)]$$

From the definition of E[x]

$$E_L[\nabla\ell(x^{(i)},y^{(i)},\theta)] = \sum_{j=1}^{n}P(i=j)\cdot\nabla\ell(x^{(j)},y^{(j)},\theta)$$

Because the samples are chosen randomly, the probability is $1/n$, we can get that:

$$E_L[\nabla\ell(x^{(i)},y^{(i)},\theta)] = \frac{1}{n}\sum_{j=1}^{n}\nabla\ell(x^{(j)},y^{(j)},\theta) = \nabla L(x,y,\theta)$$

Which equal to the lift hand side $\frac{1}{m}\sum_{i=1}^{m}\nabla L(x,y,\theta) = \nabla L(x,y,\theta)$ = right side of equation

[3] We can get that the value of gradient of mini batch in SGD is the same as the value of true empirical gradient

[4] $\nabla L(x, y, \theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(x^{(i)}, y^{(i)}, \theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla (w^T x^{(i)} - y^{(i)})^2$

$= \frac{1}{n}\sum_{i=1}^{n} \nabla \left(w^2 x^{(i)^2} + y^{(i)^2} - 2 w x^{(i)} y^{(i)} \right)$

$= \frac{1}{n} \sum_{i=1}^{n} (2 x^{(i)} x^{(i)^T} w - 2 x^{(i)} y^{(i)})$

(b) **code to compute this gradient**

```
    Loss = np.zeros(X.shape)
    for i in range(len(X)):
        Loss[i] = 2*X[i].T*np.dot(X[i].T, w)-2*X[i].T*y[i]
    gradient = np.mean(Loss, axis=0)

    return gradient
```

[5] Cosine similarity & squared distance matric

Cosine similarity is a more meaningful measure. Because it limits the scale of the result.

```
# Question 5
batch_grad = []
for j in range(num_iter):
    X_b, y_b = batch_sampler.get_batch(m=50)
    batch_grad.append(lin_reg_gradient(X_b, y_b, w))
grad_batch = np.mean(batch_grad, axis=0)
grad_true = lin_reg_gradient(X, y, w)
cosine_s = cosine_similarity(grad_batch, grad_true)
print('cosine similarity is ' + str(cosine_s))
distance = ((grad_batch-grad_true)**2).mean()
distance = sqrt(distance)
print('squared distance metric is ' + str(distance))
```

```
Total data points: 506
Feature count: 13
Random parameters, w: [-0.35029347  0.1414773  -1.53463306  0.30050294 -2.48672583 -0.1110306
  0.00461172 -1.18791879  0.37891595 -0.47198755  3.16445653  0.27921423
 -0.0241888 ]
----------------------------------------


cosine similarity is 0.99999936161
squared distance metric is 121.232987167198
```

[6] Plot variance against m

We randomly choose a weight from the weight vector. we try the mini-batch size m in the range [0,400]. For each a randomly batch, we run the algorithm for 500 times and get the mean value of the variance. Finally, by using log m and log variance.