Regularization and Bias and Variance Analysis

June 17, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        import pandas as pd
        import seaborn as sns

        %matplotlib inline

        from sklearn.datasets import load_boston
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error

        np.random.seed(9)
        boston_dataset = load_boston()

In [2]: print(boston_dataset.keys())
        print('****************************************************************************')
        print(boston_dataset.DESCR)
```

```
dict_keys(['data', 'feature_names', 'target', 'filename', 'DESCR'])
****************************************************************************
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usu

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
```

```
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Universit

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regressio
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the
```

In [3]:
```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

PTRATIO   B LSTAT

```
0      15.3  396.90   4.98
1      17.8  396.90   9.14
2      17.8  392.83   4.03
3      18.7  394.63   2.94
4      18.7  396.90   5.33
```

In [4]:
```python
boston['MEDV'] = boston_dataset.target
boston.head()
```

Out[4]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

In [5]:
```python
boston.isnull().sum()
```

Out[5]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

0.1 Let's split our dataset in training (60%), validation (20%) and test(20%)

In [6]:
```python
VARIABLE = 'LSTAT'
X = boston[VARIABLE].values.reshape((boston[VARIABLE].shape[0], 1))
y = boston['MEDV'].values.reshape((X.shape[0], 1))

df = pd.DataFrame(np.concatenate([X, y], axis=1))
#print(df.shape)
```

```
train, val, test = np.split(df.sample(frac=1), [int(.6*len(df)), int(.8*len(df))])
print(train.shape, val.shape, test.shape)

X_train, y_train = train[0], train[1]
X_val, y_val = val[0], val[1]
X_test, y_test = test[0], test[1]
print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)

X_train, y_train = np.reshape([X_train], (train.shape[0], 1)), np.reshape([y_train], (tr
X_val, y_val = np.reshape([X_val], (val.shape[0], 1)), np.reshape([y_val], (val.shape[0]
X_test, y_test = np.reshape([X_test], (test.shape[0], 1)), np.reshape([y_test], (test.sh

X_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_val = np.c_[np.ones((X_val.shape[0], 1)), X_val]
X_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(303, 2) (101, 2) (102, 2)
(303,) (101,) (102,)
(303,) (101,) (102,)
(303, 2) (101, 2) (102, 2)
```

1 Linear Regression with Regularization

1.1 2.1 Cost function (we do not regularize theta_0)

$$h_\theta(x) = \theta_0 + \theta_1 x = X\theta$$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y) + \frac{\lambda}{2m}(\theta_r^T\theta_r)$$

$$\theta_r = \begin{bmatrix} 0 \\ \theta_1 \\ \vdots \\ \theta_{n+1} \end{bmatrix}$$

1.2 2.2 Gradient

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

1.3 2.3 Gradient Vectorized

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} X^T (X\theta - y)$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} X^T (X\theta - y) + \frac{\lambda}{m}\theta_r$$

1.4 2.4 Update Rule

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

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

In [7]:
```
def regCost(theta, X, y, lambda_reg):
    m = y.shape[0]
    h = X.dot(theta)
    J = np.sum(np.square(h-y)) + lambda_reg*np.sum(np.square(theta[1:]))
    return (J/(2*m))

def regGradient(theta, X, y, lambda_reg):
    m = y.shape[0]
    theta_r = np.copy(theta)
    theta_r[0] = 0
    h = X.dot(theta)
    dJ = (1/m)*((X.T).dot(h-y)+ lambda_reg*theta_r)
    return (dJ.flatten())

def regGradientDescentVectorized( X, y, theta, alpha, n_iter, lambda_reg):
    m=y.shape[0]
    J = np.zeros(n_iter)
    for i in range(0,n_iter):
        h = X.dot(theta)
        gradient = regGradient(theta, X, y , lambda_reg)
        gradient = np.reshape([gradient], (gradient.shape[0],1))
        theta = theta - (alpha) * gradient
        J[i] = regCost(theta, X, y, lambda_reg)
    return (J,theta)
```

In [8]:
```
initial_theta = np.ones((X_train.shape[1], 1))
cost = regCost(initial_theta, X_train, y_train, 0)
gradient = regGradient(initial_theta, X_train, y_train, 0)
print(cost)
print(gradient)
```

```
160.7982498349835
[ -9.79808581 -25.97609439]
```

2 Bias and Variance Analysis

2.1 We introduce some tools to diagnose if the learning problem is suffering from bias or variance problem

2.2 If our problem suffers from high bias, it means that the model underfits. Then we can try 3 approaches:

2.2.1 1. To get additional features

2.2.2 2. To add polynomial features

2.2.3 3. To decrease the regularization term

2.3 If our problem suffers from high variance, it means that the model overfits. Then we can try 3 approaches:

2.3.1 1. To get more training examples

2.3.2 2. To reduce the set of features

2.3.3 3. To increase the regularization term

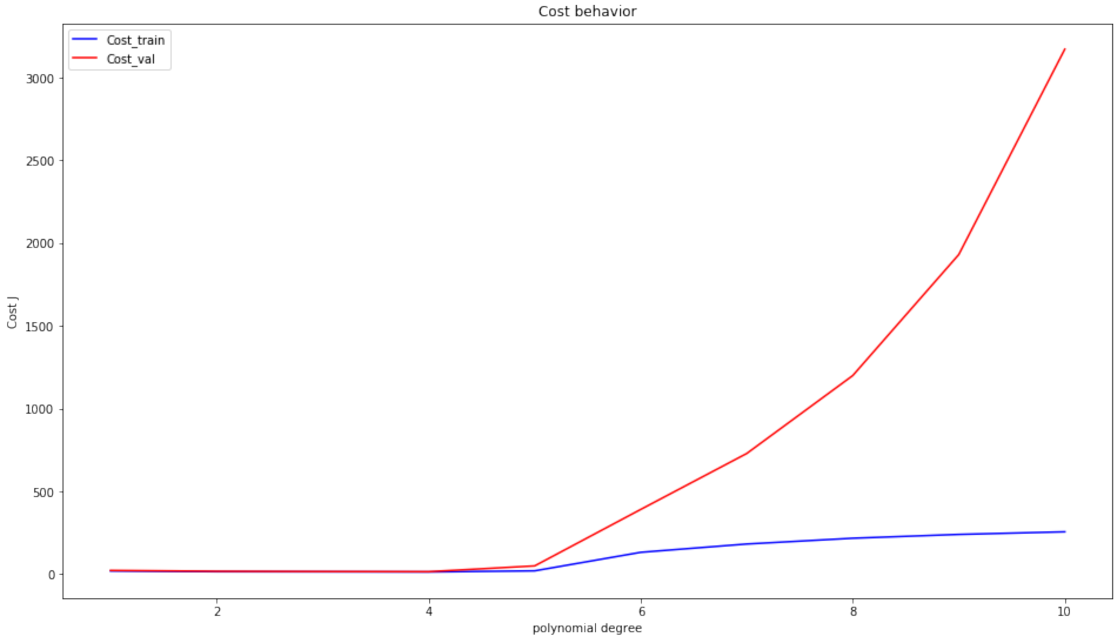
In [9]:
```
def polynomial_features(X, degree):
    for i in range(1, degree):
        colname = dataframe.columns[0]+'^'+str(i+1)
        X[colname] = np.power(X[VARIABLE], i+1)
    return X

def normalEquation(X, y):
    pseudo_inv = np.linalg.pinv(X.T.dot(X))
    product = pseudo_inv.dot(X.T)
    return product.dot(y)
```

2.4 Let's evaluate the cost function on the training set and the validation set to the increase of the polynomial degree and without regularization

In [10]:
```
dataframe = pd.DataFrame(X_train[:, 1], columns = [VARIABLE])
dataframev = pd.DataFrame(X_val[:, 1], columns = [VARIABLE])

costs_t = []
costs_v = []
for i in range(1, 11):
    X_t = np.concatenate([np.ones((X_train.shape[0],1)), polynomial_features(dataframe,
    theta_ne_t = normalEquation(X_t, y_train)
    cost_t = regCost(theta_ne_t, X_t, y_train, 0)
    costs_t.append(cost_t)
    X_v = np.concatenate([np.ones((X_val.shape[0],1)), polynomial_features(dataframev,
    cost_v = regCost(theta_ne_t, X_v, y_val, 0)
    costs_v.append(cost_v)
print(costs_t)
print(costs_v)
```

```
[18.533964398266033, 14.336088185911802, 13.595530380992917, 12.609790250734594, 19.281761790719
[21.907005175891513, 16.69923969942444, 15.203813493388687, 14.337475950012152, 49.2442483499470
```

```
In [11]: # Plot the convergence graph
         plt.figure(figsize=(16,9))
         plt.plot([i for i in range(1,11)], costs_t, '-b', label = 'Cost_train')
         plt.plot([i for i in range(1,11)], costs_v, '-r', label = 'Cost_val')
         plt.xlabel('polynomial degree') # Set the xaxis label
         plt.ylabel('Cost J') # Set the yaxis label
         plt.title('Cost behavior')
         plt.legend()
         plt.show()
```



2.5 We want to minimize both errors, then we choose the 4th degree as the best model.

2.6 Now let's evaluate the chosen model to the increase of the regularization term

2.7 For this purpose, let's introduce the regularized normal equation:

$$\theta = (X^T X + E)^{-1} X^T y$$

where

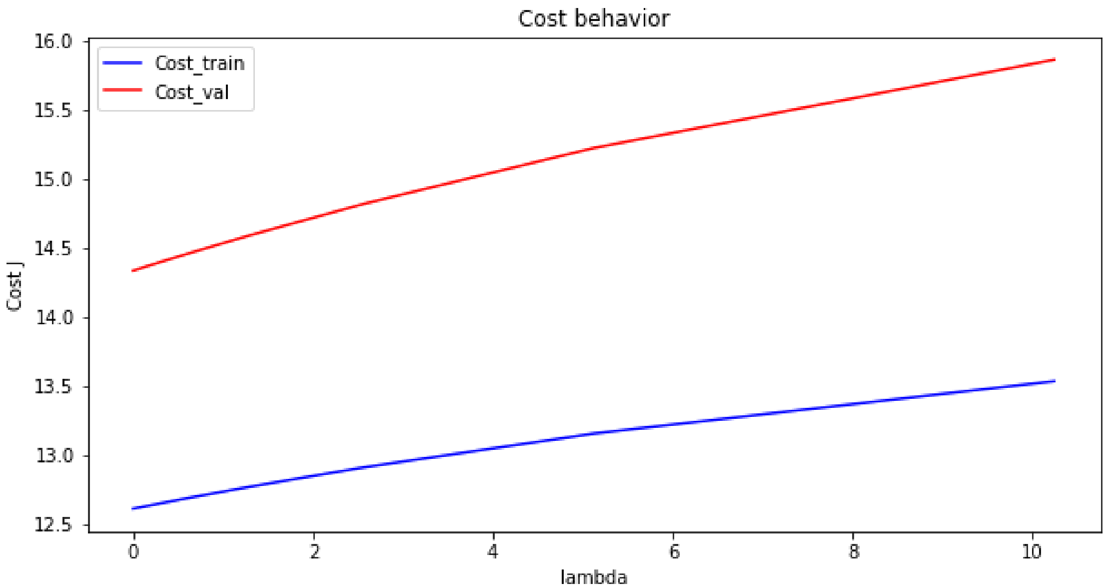
$$E = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & \lambda & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda \end{bmatrix}$$

```
In [12]: def regNormalEquation(X, y, lambda_reg):
             m=X.shape[1]
             E = np.identity(m)*lambda_reg
             E[0,0] = 0
             pseudo_inv = np.linalg.pinv(X.T.dot(X) + E)
             product = pseudo_inv.dot(X.T)
             return product.dot(y)
```

```
In [18]: dataframe2 = pd.DataFrame(X_train[:, 1], columns = [VARIABLE])
         dataframev2 = pd.DataFrame(X_val[:, 1], columns = [VARIABLE])

         X_t4 = np.concatenate([np.ones((X_train.shape[0],1)), polynomial_features(dataframe2, 4

         X_v4 = np.concatenate([np.ones((X_val.shape[0],1)), polynomial_features(dataframev2, 4)

         regcosts = []
         regcosts_v = []
         lamb = []
         theta_t4 = regNormalEquation(X_t4, y_train, 0)
         cost_reg = regCost(theta_t4, X_t4, y_train, 0)
         regcosts.append(cost_reg)
         cost_reg_v = regCost(theta_t4, X_v4, y_val, 0)
         regcosts_v.append(cost_reg_v)
         lamb.append(0)
         lamb.append(0.01)
         while lamb[len(lamb)-1]<11:
             theta_t4 = regNormalEquation(X_t4, y_train, lamb[len(lamb)-1])
             cost_reg = regCost(theta_t4, X_t4, y_train, lamb[len(lamb)-1])
             regcosts.append(cost_reg)
             cost_reg_v = regCost(theta_t4, X_v4, y_val, lamb[len(lamb)-1])
             regcosts_v.append(cost_reg_v)
             lamb.append(lamb[len(lamb)-1]*2)

         print(regcosts)
         print(regcosts_v)
```

```
[12.609790250734594, 12.61108657282577, 12.61238179027853, 12.614968916916139, 12.62012997651435
[14.337475950012152, 14.339545739980021, 14.341613664090154, 14.345744891884774, 14.353989502681
```

```
In [19]: plt.figure(figsize=(10,5))
         plt.plot(lamb[:len(lamb)-1], regcosts, '-b', label = 'Cost_train')
         plt.plot(lamb[:len(lamb)-1], regcosts_v, '-r', label = 'Cost_val')
         plt.xlabel('lambda') # Set the xaxis label
         plt.ylabel('Cost J') # Set the yaxis label
         #plt.ylim(13, 16)
         plt.title('Cost behavior')
         plt.legend()
         plt.show()
```



2.8 We want to minimize both errors, then we choose the regularization term equal to 0

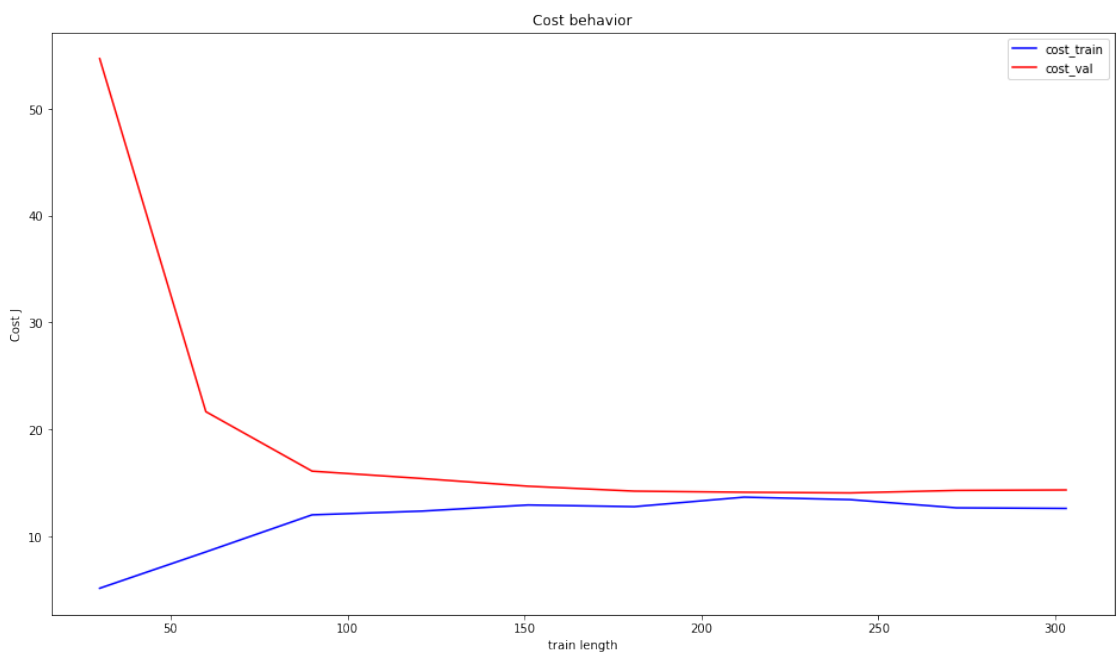
2.9 Now let's evaluate the chosen model and the chosen regularization term to the increase of the training set size

In [15]:
```
X_array = []
y_array = []
leng = []
used_lambda = 0

costs_t = []
costs_v = []
degree=4

for i in range(1,11):
    X_temp = X_train[:int(X_train.shape[0]*(i*0.1))]
    y_temp = y_train[:int(y_train.shape[0]*(i*0.1))]
    dataframe3 = pd.DataFrame(X_temp[:, 1], columns = [VARIABLE])
    dataframev3 = pd.DataFrame(X_val[:, 1], columns = [VARIABLE])
    X_i = np.concatenate([np.ones((X_temp.shape[0],1)), polynomial_features(dataframe3,
    theta_ne_t = regNormalEquation(X_i, y_temp, used_lambda)
    cost_t = regCost(theta_ne_t, X_i, y_temp, used_lambda)
    costs_t.append(cost_t)

    X_iv = np.concatenate([np.ones((X_val.shape[0],1)), polynomial_features(dataframev3
    cost_v = regCost(theta_ne_t, X_iv, y_val, used_lambda)
    costs_v.append(cost_v)
```

```
        leng.append(X_temp.shape[0])


    print(leng)
```

```
[30, 60, 90, 121, 151, 181, 212, 242, 272, 303]
```

```
In [17]: plt.figure(figsize=(16,9))
         plt.plot(leng, costs_t, '-b', label = 'cost_train')
         plt.plot(leng, costs_v, '-r', label = 'cost_val')


         plt.xlabel('train length') # Set the xaxis label
         plt.ylabel('Cost J') # Set the yaxis label
         plt.title('Cost behavior')
         plt.legend()
         plt.show()
```



2.10 This plot shows a high bias problem because there is an high error for both cost functions. Probably the problem is the way we represent the data.