Tutorial on Supervised Learning

Part 1 : Linear Regression (implemented in Python from scratch)

Quick Recap

- Machine Learning systems are usually classified according to the amount and type of supervision they get during training.
- There are four major categories: Supervised Learning, Unsupervised Learning, Semi-supervised Learning, and Reinforcement Learning. Let us focus on **Supervised Learning** for now.

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called *labels*. Generally there are two kinds of tasks:

- Regression : Given a set of features (predictors), predict a target numeric value.
- Classification (in the next part)

Let's try **Linear Regression** first !

Univariate Linear Regression (ULR)

There can be several features. For simplicity, let's use only one feature for regression.

The dataset

We will use a sample dataset called *Portland Housing Prices*, wherein we are given some features of a house (i.e. area, no. of rooms, etc) and predict the target price. For ULR, assume the predictor is the **area** of a house.

Problem statement

- The data file (ex1data2.txt) contains a training set of housing prices in Portland,Oregon.
- Data format: <size of the house (in square feet), number of bedrooms, price of the house>
- Need to train on this data, and predict market price of new houses.

Implementation

In [1]:
```
#importing dependencies

import numpy as np  #python library for scientific computing
import pandas as pd #python library for data analysis and dataframes
```

In [2]:

```
# load data

data = pd.read_csv('./ex1data2.txt', header=None)
data.columns =(['Size','Bedrooms','Price'])
data.head()
```

Out[2]:

	Size	Bedrooms	Price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

In [3]:

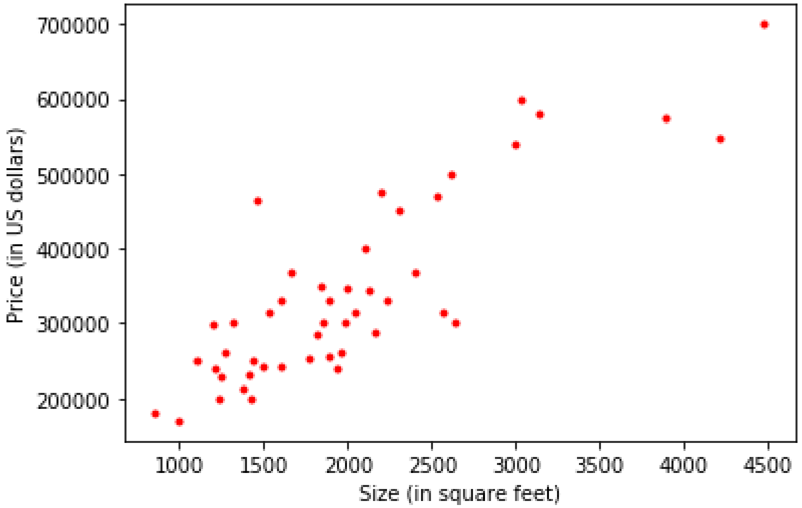
```
# Since we assume predictor variable is only area (size), remove the other featu
re
data.drop('Bedrooms', axis=1, inplace=True)
data.head()
```

Out[3]:

	Size	Price
0	2104	399900
1	1600	329900
2	2400	369000
3	1416	232000
4	3000	539900

In [4]:

```
# necessary dependencies for plotting
import matplotlib.pyplot as plt #python library for plot and graphs
%matplotlib inline

plt.plot(data.Size, data.Price, 'r.')
plt.xlabel('Size (in square feet)')
plt.ylabel('Price (in US dollars)')
plt.savefig('data_scatter.png')
plt.show()
```



Observation : High correlation between Housing Area and Housing Price. Intuitively, we could use a line (linear model) to fit this data!

In [5]:
```
data.corr()
```

Out[5]:

	Size	Price
Size	1.000000	0.854988
Price	0.854988	1.000000

The idea in Linear Regression

$$y = k + \beta_1 x_1 + \beta_2 x_2 + \ldots. + \beta_n x_n$$

Dependent Variable

Intercept

Coefficient

Predictors

In [6]:

```
X = np.array(data.drop('Price',axis=1))
y = np.array(data.Price)
m = len(data)

print(X.shape)
print(y.shape)
```

```
(47, 1)
(47,)
```

In [7]:

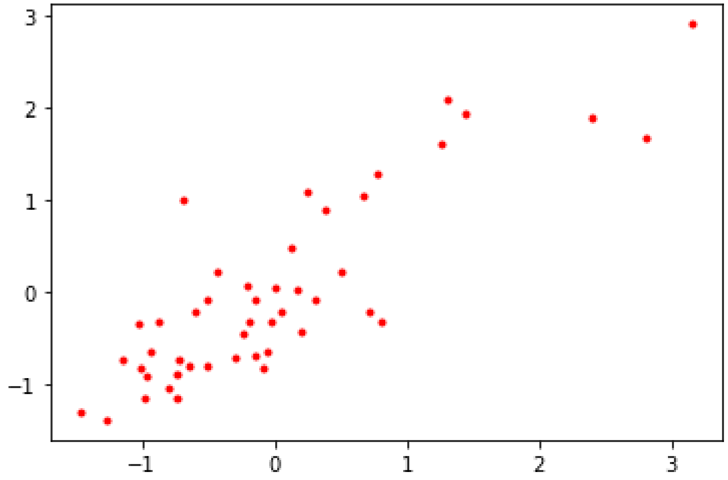
```
y = y.reshape((m,1))   # reshaping into a matrix
print(y.shape)
```

```
(47, 1)
```

In [8]:

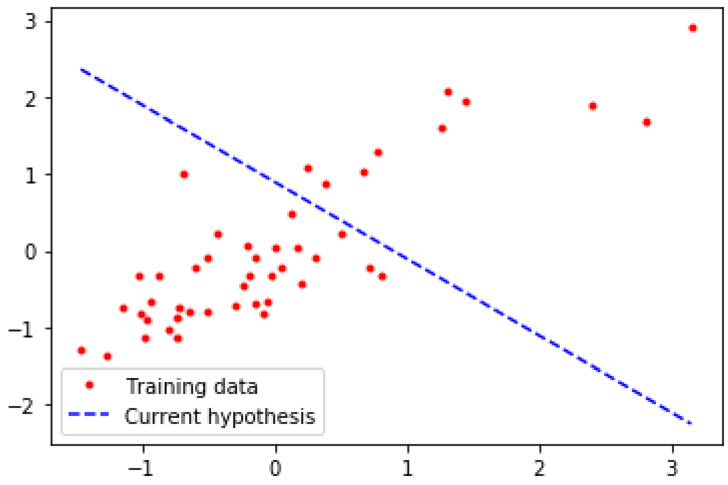
```
# feature scaling and normalization

def normscaler(Z, normal=False, scale='max'):
    Zn = np.zeros(Z.shape)
    for col in range(Zn.shape[1]):
        std = Z[:,col].std()
        clm = Z[:,col]
        mn = Z[:,col].mean()
        mx = Z[:,col].max()
        nrm = 0
        sclr = 1
        if normal:
            nrm = mn

        if scale =='max':
            sclr = mx
        elif scale == 'std':
            sclr = std
        Zn[:,col] = (clm-nrm)/sclr

    return Zn
```

In [9]:
```
Xn = normscaler(X, normal=True, scale='std')
yn = normscaler(y, normal=True, scale='std')

plt.plot(Xn, yn, 'r.')
plt.show()
```



In [11]:

```
# random parameter initialization
theta = np.array([0.9,-1])

lineX = np.linspace(Xn.min(), Xn.max(), 100)
liney = [theta[0] + theta[1]*xx for xx in lineX]

plt.plot(Xn,yn,'r.', label='Training data')
plt.plot(lineX,liney,'b--', label='Current hypothesis')
plt.legend()
plt.show()
```



In [12]:
```
def cost_function(X, y, theta, deriv=False):
    z = np.ones((len(X),1))           # column of all 1's (x_0 column of matrix
 X)
    X = np.append(z, X, axis=1)

    if deriv:
        loss     = X.dot(theta)-y
        gradient = X.T.dot(loss)/len(X)
        return gradient, loss

    else:
        h = X.dot(theta)
        j = (h-y.flatten())
        J = j.dot(j)/2/(len(X))
        return J                      # returns cost (in this case, MSE cost)

cost_function(Xn, yn, theta)
```

Out[12]:
```
2.259987592878125
```

In [13]:
```
def GradDescent(features, target, param, learnRate=0.01, multiple=1, batch=len(X
), log=False):

    iterations = batch*len(features)
    epochs     = iterations*multiple
    y          = target.flatten()
    t          = param
    b          = batch
    a          = learnRate

    theta_history  = np.zeros((param.shape[0],epochs)).T
    cost_history   = [0]*epochs

    for ix in range(epochs):

        i    = epochs%len(X)
        cost = cost_function(features[i:i+b], y[i:i+b], t)

        cost_history[ix]   = cost
        theta_history[ix]  = t

        g, l = cost_function(features[i:i+b], y[i:i+b], t, deriv=True)
        t    = t-a*g

        if log:
            if ix%250==0:
                print("iteration :", ix+1)
                #print("\tloss     = ", l)
                print("\tgradient = ", g)
                print("\trate     = ", a*g)
                print("\ttheta    = ", t)
                print("\tcost     = ", cost)

    return cost_history, theta_history

alpha = 0.01
```

```
mul = 10
bat = 8
ch, th = GradDescent(Xn,yn,theta,alpha,mul,bat,log=True)
```

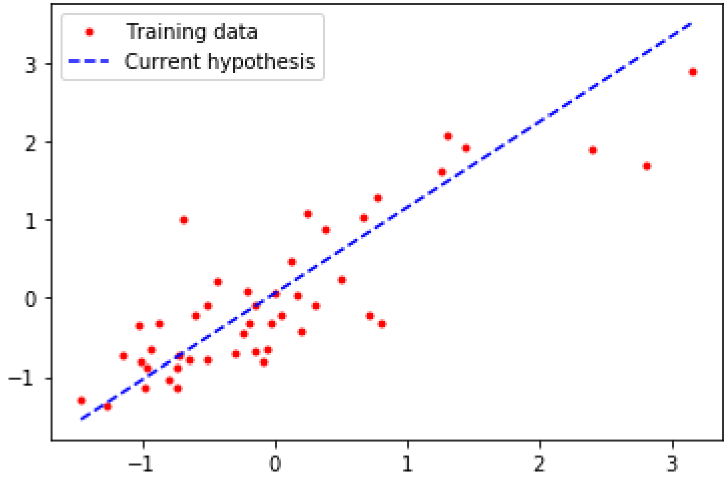
```
iteration : 1
        gradient =  [ 1.02497703 -1.01186205]
        rate     =  [ 0.01024977 -0.01011862]
        theta    =  [ 0.88975023 -0.98988138]
        cost     =  1.5476729221946035
iteration : 251
        gradient =  [ 0.04767076 -0.29470635]
        rate     =  [ 0.00047671 -0.00294706]
        theta    =  [0.04532149 0.43827016]
        cost     =  0.15187018177727896
iteration : 501
        gradient =  [-0.00724839 -0.09542187]
        rate     =  [-7.24839206e-05 -9.54218677e-04]
        theta    =  [0.02877192 0.87724003]
        cost     =  0.06550859731882473
iteration : 751
        gradient =  [-0.00425784 -0.0317633 ]
        rate     =  [-4.25783718e-05 -3.17633010e-04]
        theta    =  [0.04415607 1.0214539 ]
        cost     =  0.05620850392203404
iteration : 1001
        gradient =  [-0.00157275 -0.01064365]
        rate     =  [-1.5727469e-05 -1.0643646e-04]
        theta    =  [0.05096421 1.06962801]
        cost     =  0.05516264135740254
iteration : 1251
        gradient =  [-0.00053931 -0.00357219]
        rate     =  [-5.39313224e-06 -3.57218620e-05]
        theta    =  [0.05337906 1.08578419]
        cost     =  0.05504474088262382
iteration : 1501
        gradient =  [-0.00018197 -0.00119932]
        rate     =  [-1.81969074e-06 -1.19932415e-05]
        theta    =  [0.05420001 1.09120753]
```

```
        cost     =  0.05503144815633974
iteration : 1751
        gradient =  [-6.11700564e-05 -4.02694948e-04]
        rate     =  [-6.11700564e-07 -4.02694948e-06]
        theta    =  [0.05447646 1.09302844]
        cost     =  0.05502949452873095
iteration : 2001
        gradient =  [-2.0544911e-05 -1.3521487e-04]
        rate     =  [-2.0544911e-07 -1.3521487e-06]
        theta    =  [0.05456935 1.09363985]
        cost     =  0.05502978047978815 4
iteration : 2251
        gradient =  [-6.89893279e-06 -4.54019757e-05]
        rate     =  [-6.89893279e-08 -4.54019757e-07]
        theta    =  [0.05460054 1.09384515]
        cost     =  0.0550297614287185 6
iteration : 2501
        gradient =  [-2.31653611e-06 -1.52449331e-05]
        rate     =  [-2.31653611e-08 -1.52449331e-07]
        theta    =  [0.05461102 1.09391408]
        cost     =  0.05502975928078325 5
iteration : 2751
        gradient =  [-7.77842106e-07 -5.11889717e-06]
        rate     =  [-7.77842106e-09 -5.11889717e-08]
        theta    =  [0.05461453 1.09393723]
        cost     =  0.05502975903861176 4
iteration : 3001
        gradient =  [-2.61181676e-07 -1.71880779e-06]
        rate     =  [-2.61181676e-09 -1.71880779e-08]
        theta    =  [0.05461571 1.093945  ]
        cost     =  0.0550297590113078 5
iteration : 3251
        gradient =  [-8.76988099e-08 -5.77136082e-07]
        rate     =  [-8.76988099e-10 -5.77136082e-09]
        theta    =  [0.05461611 1.09394761]
        cost     =  0.05502975900822943 6
iteration : 3501
        gradient =  [-2.94472426e-08 -1.93789009e-07]
```

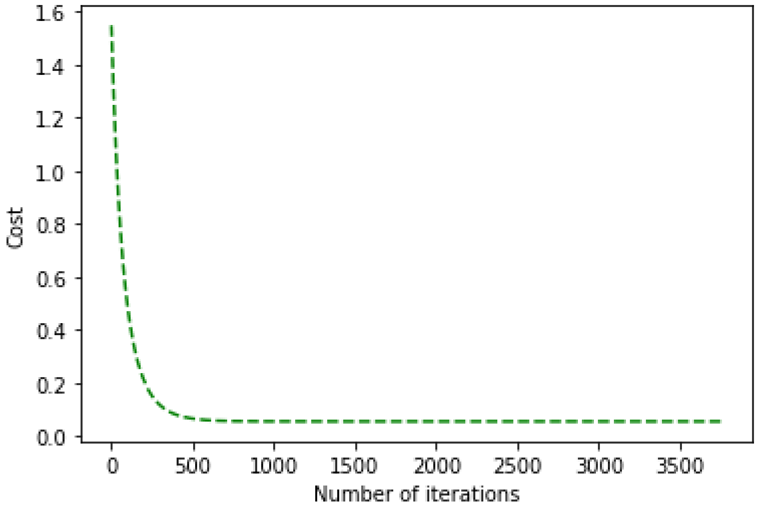
```
        rate     =  [-2.94472426e-10 -1.93789009e-09]
        theta    =  [0.05461624 1.09394849]
        cost     =  0.05502975900788236
iteration : 3751
        gradient =  [-9.88770626e-09 -6.50698878e-08]
        rate     =  [-9.88770626e-11 -6.50698878e-10]
        theta    =  [0.05461629 1.09394878]
        cost     =  0.055029759007843224
```

In [14]:

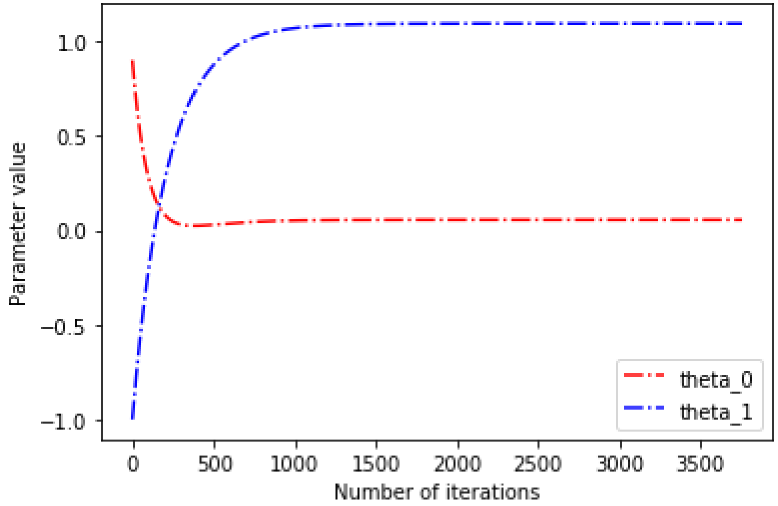
```
# training resuls

lineX = np.linspace(Xn.min(), Xn.max(), 100)

# the final values of theta are used for the fit
liney = [th[-1,0] + th[-1,1]*xx for xx in lineX]

plt.plot(Xn,yn,'r.', label='Training data')
plt.plot(lineX,liney,'b--', label='Current hypothesis')
plt.legend()
plt.show()
```



In [19]:
```
# Loss plot

plt.plot(ch,'g--')
plt.ylabel('Cost')
plt.xlabel('Number of iterations')
plt.show()
```



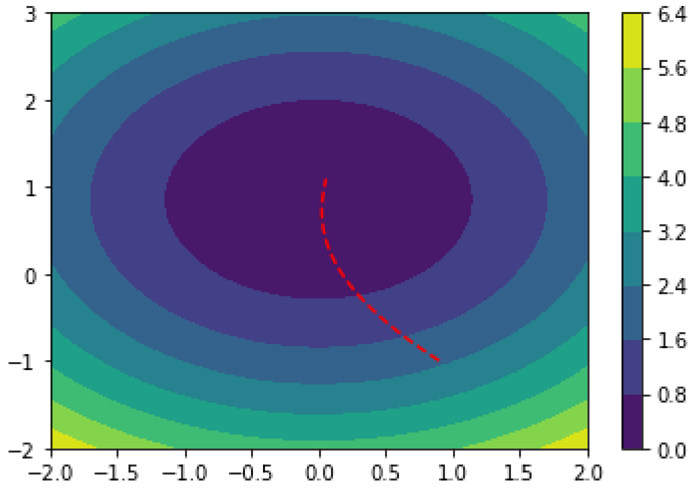
In [20]:
```
# How parameters are changing

plt.plot(th[:,0],'r-.', label = 'theta_0')
plt.plot(th[:,1],'b-.', label = 'theta_1')
plt.ylabel('Parameter value')
plt.xlabel('Number of iterations')
plt.legend()
plt.show()
```



In [17]:

```python
#Grid over which we will calculate J
theta0_vals = np.linspace(-2, 2, 100)
theta1_vals = np.linspace(-2, 3, 100)

#initialize J_vals to a matrix of 0's
J_vals = np.zeros((theta0_vals.size, theta1_vals.size))

#Fill out J_vals
for t1, element in enumerate(theta0_vals):
    for t2, element2 in enumerate(theta1_vals):
        thetaT = np.zeros(shape=(2, 1))
        thetaT[0][0] = element
        thetaT[1][0] = element2
        J_vals[t1, t2] = cost_function(Xn, yn, thetaT.flatten())

#Contour plot
J_vals = J_vals.T
```

In [18]:
```
A, B = np.meshgrid(theta0_vals, theta1_vals)
C = J_vals

cp = plt.contourf(A, B, C)
plt.colorbar(cp)
plt.plot(th.T[0],th.T[1],'r--')
plt.show()
```



End of Part 1