机器学习练习 1 - 线性回归 Linear Regression

在本练习中，我们将使用梯度下降实现简单线性回归，并应用于一个示例问题。

单变量线性回归

在练习的第一部分中，我们的任务是使用一个变量实现线性回归，以预测食品卡车的利润。假设你是一家连锁餐厅的首席执行官，正在考虑在不同的城市开设一家新的分店。这家连锁店在各个城市都有卡车，你可以从这些城市获得人口数据及利润数据。

我们首先导入相关的工具库与数据

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
import os
path = 'ex1data1.txt'
data = pd.read_csv(path, header=None, names=['Population', 'Profit'])
data.head()
```

Out[2]:

	Population	Profit
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

In [3]:

```
data.describe()
```

Out[3]:

	Population	Profit
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

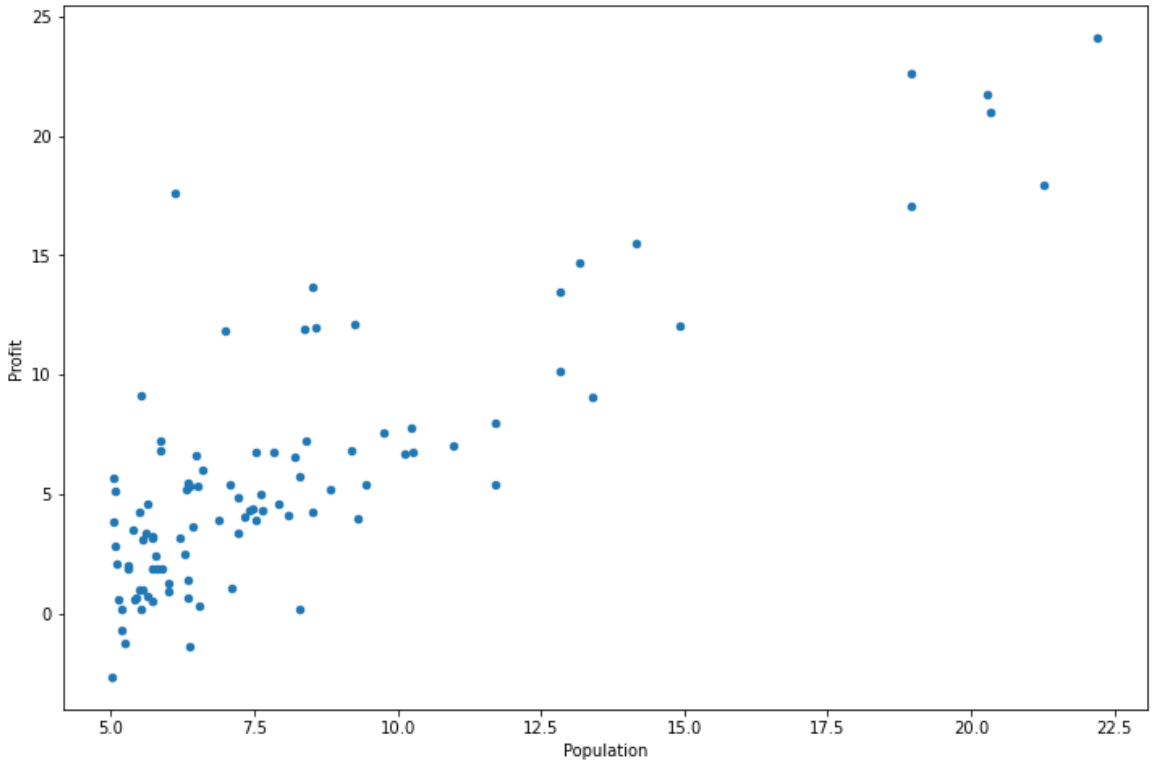
我们首先从直观的角度来观察一下数据

In [4]:

```
data.plot(kind='scatter', x='Population', y='Profit', figsize=(12,8))
```

Out[4]:

```
<AxesSubplot:xlabel='Population', ylabel='Profit'>
```



现在，我们使用梯度下降实现线性回归，以最小化代价函数。

我们首先定义一个函数实现代价函数的实现

In [5]:

```
def computeCost(X, y, theta):
    inner = np.power(((X * theta.T) - y), 2)
    return np.sum(inner) / (2 * len(X))
```

在数据中加入全1列，其目的是为了进行向量计算，提高计算效率

In [6]:

```
data.insert(0, 'Ones', 1)
```

现在我们初始化一些参数

In [7]:

```
# set X (training data) and y (target variable)
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]
```

现在我们来查看一下训练集中的X和y是否正确

In [8]:

```
X.head()
```

Out[8]:

	Ones	Population
0	1	6.1101
1	1	5.5277
2	1	8.5186
3	1	7.0032
4	1	5.8598

In [9]:

```
y.head()
```

Out[9]:

	Profit
0	17.5920
1	9.1302
2	13.6620
3	11.8540
4	6.8233

为了实验numpy矩阵计算，我们需要对X和y进行预处理，并对theta进行初始化

In [10]:

```
X = np.matrix(X.values)
y = np.matrix(y.values)
theta = np.matrix(np.array([0,0]))
```

theta 初始化结果显示

In [11]:

```
theta
```

Out[11]:

```
matrix([[0, 0]])
```

通过shape可以观察各个矩阵的结构

In [12]:

```
X.shape, theta.shape, y.shape
```

Out[12]:

```
((97, 2), (1, 2), (97, 1))
```

计算代价函数 注意，此时theta刚刚初始化为0

In [13]:

```
computeCost(X, y, theta)
```

Out[13]:

```
32.072733877455676
```

现在，我们来看看如何实现梯度下降，这里我们定义了一个函数

In [14]:

```
def gradientDescent(X, y, theta, alpha, iters):
    temp = np.matrix(np.zeros(theta.shape))
    parameters = int(theta.ravel().shape[1])
    cost = np.zeros(iters)

    for i in range(iters):
        error = (X * theta.T) - y

        for j in range(parameters):
            term = np.multiply(error, X[:,j])
            temp[0,j] = theta[0,j] - ((alpha / len(X)) * np.sum(term))

        theta = temp
        cost[i] = computeCost(X, y, theta)

    return theta, cost
```

初始化一些变量 -学习率 -循环次数

In [15]:

```
alpha = 0.01
iters = 1000
```

现在可以调用梯度下降函数，根据训练集，优化模型参数theta

In [16]:

```
g, cost = gradientDescent(X, y, theta, alpha, iters)
g
```

Out[16]:

```
matrix([[-3.24140214,  1.1272942 ]])
```

最后根据优化后的模型参数计算代价函数

In [17]:

```
computeCost(X, y, g)
```

Out[17]:

```
4.515955503078914
```

现在我们可以采用可视化方式直观地观察模型的拟合情况

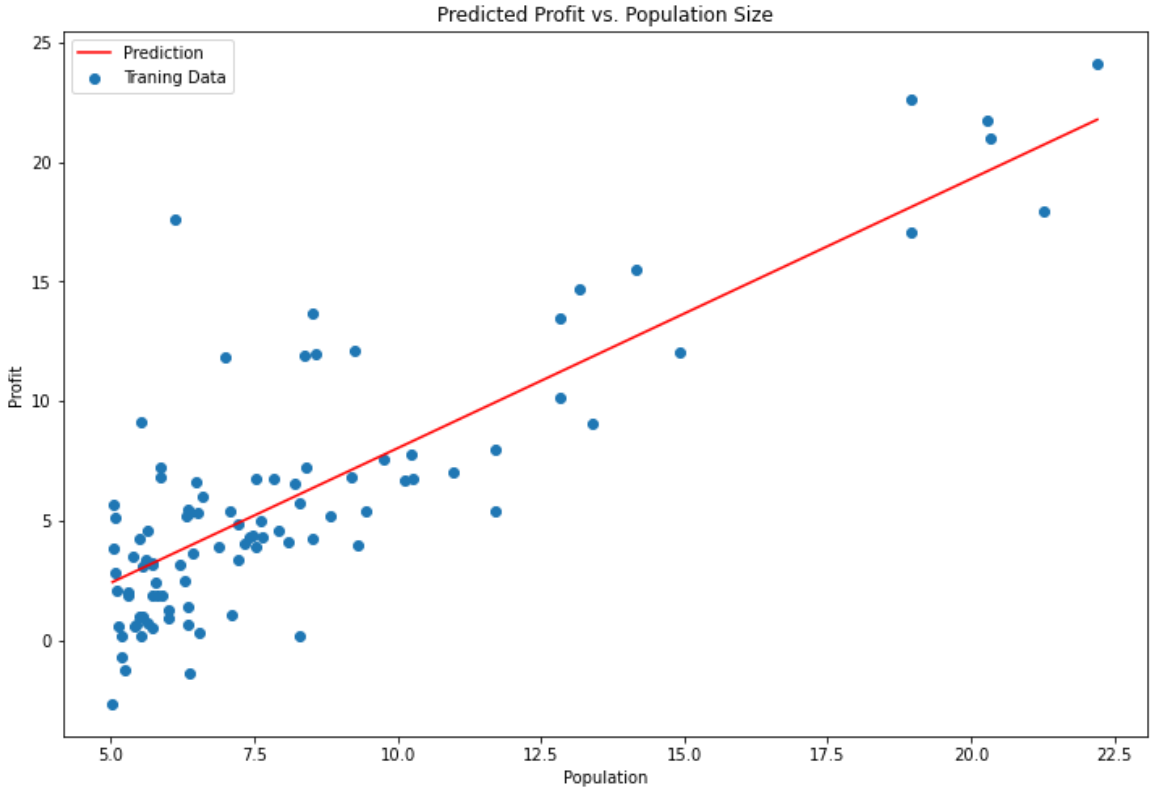
In [18]:

```
x = np.linspace(data.Population.min(), data.Population.max(), 100)
f = g[0, 0] + (g[0, 1] * x)

fig, ax = plt.subplots(figsize=(12,8))
ax.plot(x, f, 'r', label='Prediction')
ax.scatter(data.Population, data.Profit, label='Traning Data')
ax.legend(loc=2)
ax.set_xlabel('Population')
ax.set_ylabel('Profit')
ax.set_title('Predicted Profit vs. Population Size')
```

Out[18]:

```
Text(0.5, 1.0, 'Predicted Profit vs. Population Size')
```



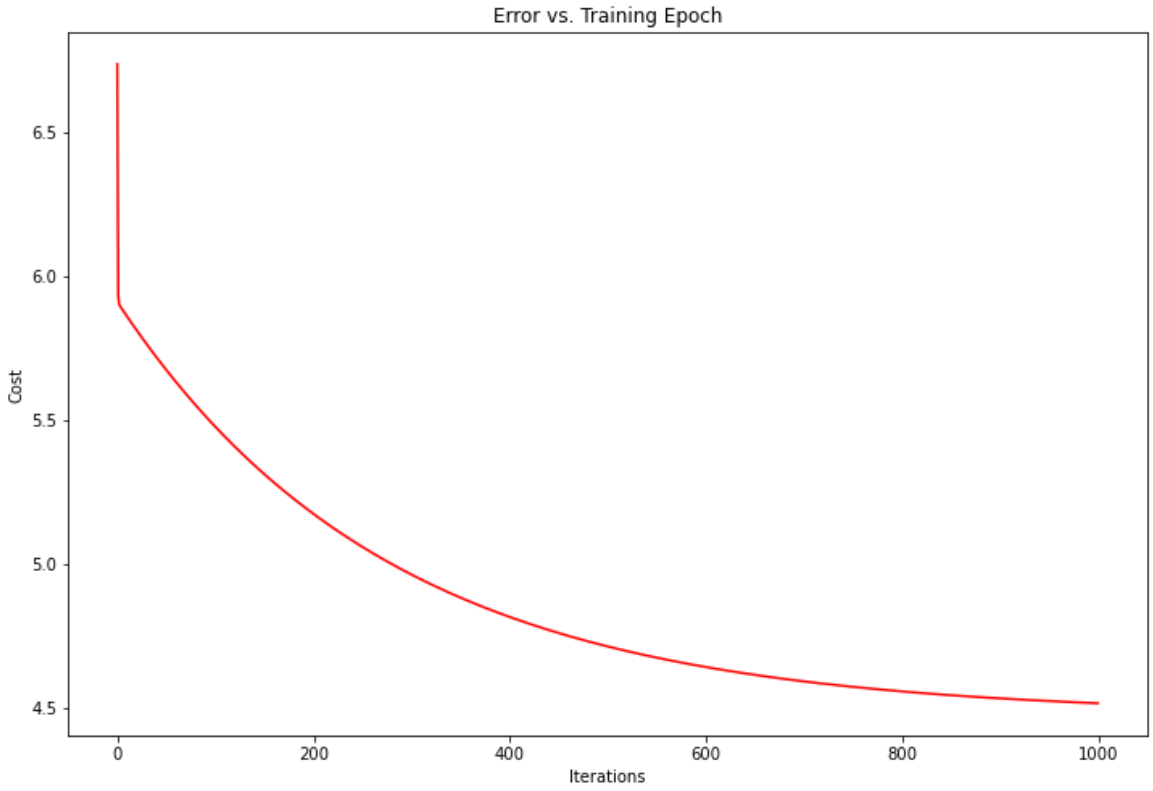
梯度下降函数每次调用时还会输出当前的代价（误差），我们可以将代价也画出来。 这里请注意，这个例子中代价函数是个凸函数，因此代价非常完美地始终呈现下降趋势，但是在很多问题中不是这样的。

In [19]:

```
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(np.arange(iters), cost, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Error vs. Training Epoch')
```

Out[19]:

```
Text(0.5, 1.0, 'Error vs. Training Epoch')
```



多变量线性回归

在第二个例子中，我们将尝试多变量线性回归。 这个例子是一个房产价格的例子。

In [27]:

```
path = 'ex1data2.txt'
data2 = pd.read_csv(path, header=None, names=['Size', 'Bedrooms', 'Price'])
data2.head()
```

Out[27]:

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

在这个实验中，我们增加了数据预处理的工作。 这里我们对数据进行了归一化。利用pandas可以非常方便的实现。

In [28]:

```
data2 = (data2 - data2.mean()) / data2.std()
data2.head()
```

Out[28]:

	Size	Bedrooms	Price
0	0.130010	-0.223675	0.475747
1	-0.504190	-0.223675	-0.084074
2	0.502476	-0.223675	0.228626
3	-0.735723	-1.537767	-0.867025
4	1.257476	1.090417	1.595389

现在我们针对第二个实验的数据，重复第一个实验并加入预处理部分。 注意，这次的数据X维度为3。

In [29]:

```
# add ones column
data2.insert(0, 'Ones', 1)

# set X (training data) and y (target variable)
cols = data2.shape[1]
X2 = data2.iloc[:,0:cols-1]
y2 = data2.iloc[:,cols-1:cols]

# convert to matrices and initialize theta
X2 = np.matrix(X2.values)
y2 = np.matrix(y2.values)
theta2 = np.matrix(np.array([0,0,0]))

# perform linear regression on the data set
g2, cost2 = gradientDescent(X2, y2, theta2, alpha, iters)

# get the cost (error) of the model
computeCost(X2, y2, g2)
```

```
(47, 4)
```

Out[29]:

```
0.13070336960771892
```

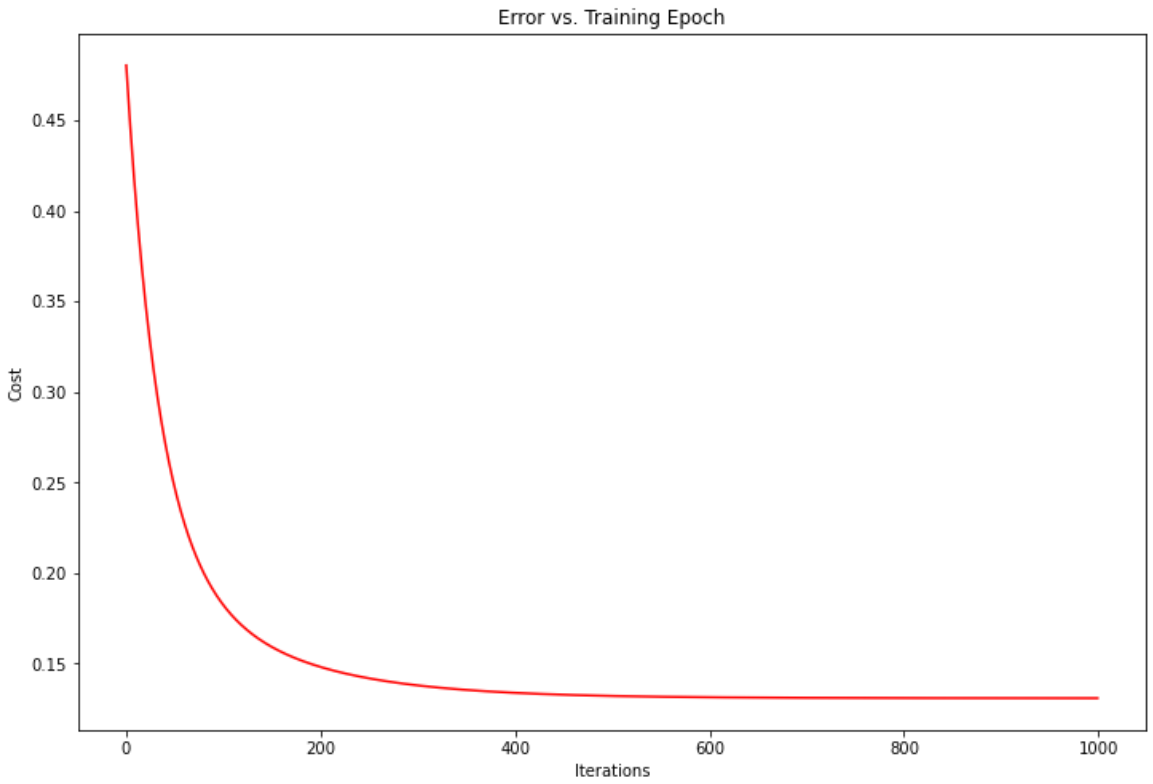
画图来直观看一下梯度下降的过程 上一步中cost2中存储了梯度下降算法中每一步的代价，因此我们可以绘制出来

In [30]:

```
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(np.arange(iters), cost2, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Error vs. Training Epoch')
```

Out[30]:

```
Text(0.5, 1.0, 'Error vs. Training Epoch')
```



到目前为止，我们通过手工码代码，实现了多变量线性回归。其实很多机器学习工具已经提供了相关的函数，可以直接调用。现在我们来看一下scikit-learn中线性回归函数的使用。由于多变量线性回归无法进行有效的可视化演示i，这里我们仅针对第一个单变量线性回归进行演示，多变量线性回归大家可以自己尝试，使用方法是完全一致的。

In [31]:

```
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(X, y)
```

Out[31]:

```
LinearRegression()
```

太简单了吧!

接下来我们可视化一下模型的训练结果

In [21]:

```
x = np.array(X[:, 1].A1)
f = model.predict(X).flatten()

fig, ax = plt.subplots(figsize=(12,8))
ax.plot(x, f, 'r', label='Prediction')
ax.scatter(data.Population, data.Profit, label='Traning Data')
ax.legend(loc=2)
ax.set_xlabel('Population')
ax.set_ylabel('Profit')
ax.set_title('Predicted Profit vs. Population Size')
```

```
---------------------------------------------------------------------
-------
IndexError                                Traceback (most recent cal
l last)
<ipython-input-21-6db56d5b5a9a> in <module>
      1 X.shape, y.shape
----> 2 x = np.array(X[:, 1].A1)
      3 f = model.predict(X).flatten()
      4
      5 fig, ax = plt.subplots(figsize=(12,8))

~\anaconda3\lib\site-packages\numpy\matrixlib\defmatrix.py in __geti
tem__(self, index)
    191
    192         try:
--> 193             out = N.ndarray.__getitem__(self, index)
    194         finally:
    195             self._getitem = False

IndexError: index 1 is out of bounds for axis 1 with size 1
```

线性回归的实验到此结束。关于线性回归，大家可以自行探索。

多项式回归

接下来，我们再探索一下多项式回归的问题

In [200]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 构造模拟数据，X特征(一维) , y真值
x = np.random.uniform(-3, 3, size=100)
X = x.reshape(-1, 1)
y = 0.5 * x**2 + x + 2 + np.random.normal(0, 1, 100)
```

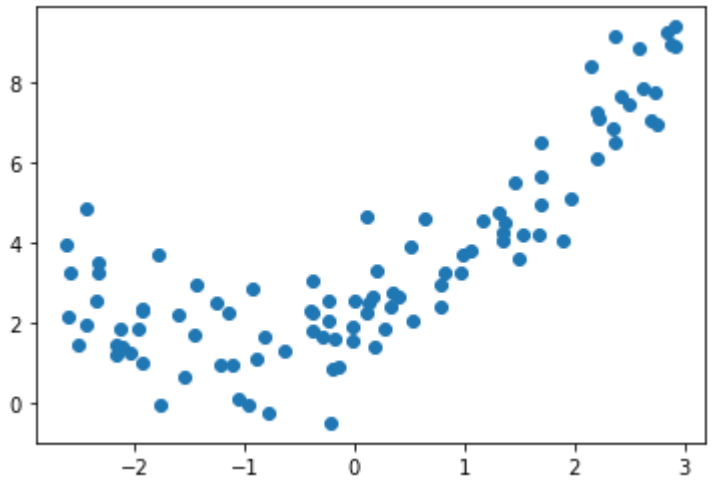
画图看一下数据

In [201]:

```
plt.scatter(x, y)
```

Out[201]:

```
<matplotlib.collections.PathCollection at 0x1fddd1c2580>
```



利用PolynomialFeatures构造特征

In [202]:

```
# 创建对象并填入将样本改成 最高degree次幂
poly = PolynomialFeatures(degree=2)
#fit原样本
poly.fit(X)
#修改原样本，构造新特征。
X2 = poly.transform(X)
X2[:5,:]
```

Out[202]:

```
array([[ 1.        ,  0.99215687,  0.98437525],
       [ 1.        ,  0.79021798,  0.62444446],
       [ 1.        ,  1.89539239,  3.59251231],
       [ 1.        , -2.59161161,  6.71645076],
       [ 1.        , -0.92766681,  0.8605657 ]])
```

第一例是x0特征，第二列原样本，第三列原样本2次幂样本'''

In [203]:

```
# 调用线性回归模型
lin_reg2 = LinearRegression()
lin_reg2.fit(X2, y)
y_predict2 = lin_reg2.predict(X2)
plt.scatter(x, y)
#np.sort排序，np.argsort排序索引
plt.plot(np.sort(x), y_predict2[np.argsort(x)], color='r')
```

Out[203]:

```
[<matplotlib.lines.Line2D at 0x1fddd480130>]
```