Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

Лабораторная работа №1

“Линейная регрессия”

Выполнил: Реут Виктор Александрович

магистрант кафедры информатики

группа №858342

Минск 2019

Условие работы

Набор данных **ex1data1.txt** представляет собой текстовый файл, содержащий информацию о населении городов (первое число в строке) и прибыли ресторана, достигнутой в этом городе (второе число в строке). Отрицательное значение прибыли означает, что в данном городе ресторан терпит убытки.

Набор данных **ex1data2.txt** представляет собой текстовый файл, содержащий информацию о площади дома в квадратных футах (первое число в строке), количестве комнат в доме (второе число в строке) и стоимости дома (третье число).

Задание.

1. Загрузите набор данных **ex1data1.txt** из текстового файла.
2. Постройте график зависимости прибыли ресторана от населения города, в котором он расположен.
3. Реализуйте функцию потерь J(θ) для набора данных **ex1data1.txt**.
4. Реализуйте функцию градиентного спуска для выбора параметров модели. Постройте полученную модель (функцию) совместно с графиком из пункта 2.
5. Постройте трехмерный график зависимости функции потерь от параметров модели (θ_0 и θ_1) как в виде поверхности, так и в виде изолиний (contour plot).
6. Загрузите набор данных **ex1data2.txt** из текстового файла.
7. Произведите нормализацию признаков. Повлияло ли это на скорость сходимости градиентного спуска? Ответ дайте в виде графика.
8. Реализуйте функции потерь J(θ) и градиентного спуска для случая многомерной линейной регрессии с использованием векторизации.
9. Покажите, что векторизация дает прирост производительности.
10. Попробуйте изменить параметр α (коэффициент обучения). Как при этом изменяется график функции потерь в зависимости от числа итераций градиентного спуск? Результат изобразите в качестве графика.
11. Постройте модель, используя аналитическое решение, которое может быть получено методом наименьших квадратов. Сравните результаты данной модели с моделью, полученной с помощью градиентного спуска.

Ход выполнения работы

1.Загрузите набор данных **ex1data1.txt** из текстового файла.

```
In [2]: os.chdir("/path/to/data/folder/")
        data_row1 = np.genfromtxt('ex1data1.txt', delimiter=',')
        data1 = pd.DataFrame(data_row1, columns=list(['Population', 'Profit']))
```

2.Постройте график зависимости прибыли ресторана от населения города, в котором он расположен.

```
In [4]: plt.figure()
        data1.plot(x='Population', y='Profit',  style=['rx'])
        plt.show()

Out[4]: <Figure size 432x288 with 0 Axes>
```

3.Реализуйте функцию потерь J(θ) для набора данных **ex1data1.txt**.

```
In [5]: def computeCost(X, y, theta):
            h = [np.matmul(x, theta.T).sum() for x in X]
            return np.power(h - y, 2).sum() / (2 * m)
```

```
In [6]: m = data_row1.shape[0] # Size of training set
        n = data_row1.shape[1] # Size of feature vector
        X1 = data1[['Population']]
        X1.insert(0, 'theta_0', 1)
        y1 = data1['Profit']
        theta = np.zeros((1, n)) # theta coeficents for hypothesis func
        X1.head()
```

	theta_0	Population
0	1	6.1101
1	1	5.5277
2	1	8.5186
3	1	7.0032
4	1	5.8598

```
In [7]: cost = computeCost(X1.to_numpy(), y1.to_numpy(), theta)
        print('With theta = [0 ; 0]\nCost computed = %f' % (cost))

        With theta = [0 ; 0]
        Cost computed = 32.072734
```

4.Реализуйте функцию градиентного спуска для выбора параметров модели. Постройте полученную модель (функцию) совместно с графиком из пункта 2.

In [16]:
```
#function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
# GRADIENTDESCENT Performs gradient descent to learn theta
#   theta = GRADIENTDESCENT(X, y, theta, alpha, num_iters) updates theta by
#   taking num_iters gradient steps with learning rate alpha

def gradientDescent(X, y, theta, alpha, num_iters):
    m = y.shape[0]  # Size of training set
    n = X.shape[1]  # Size of feature vector
    j_history = []
    for i in range(0, num_iters):
        deltas = np.zeros(n)
        for j in range(0, n):
            xj = X[:, j]
            h = [np.matmul(x, theta.T)[0] for x in X]
            deltas[j] = ((h - y) * xj).sum() * alpha / m
        theta[0] -= deltas
        j_history.append(computeCost(X, y, theta))

    return theta, j_history
```

In [17]:
```
iterations = 1500
alpha = 0.01
(theta, j_history) = gradientDescent(X1.to_numpy(), y1.to_numpy(), theta, alpha, iterations)
```

In [18]:
```
print('Theta found by gradient descent: %s' % (theta))
```

```
Theta found by gradient descent: [[-3.89459687  1.1929147 ]]
```

In [19]:
```
print('For population = 35,000, we predict a profit of %f' %(np.matmul([1, 3.5], theta.T).sum() * 10000)) #predict1
```

```
For population = 35,000, we predict a profit of 2806.045736
```

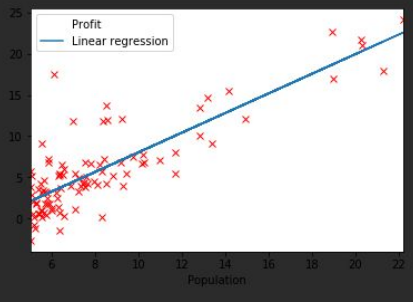
In [20]:
```
print('For population = 70,000, we predict a profit of %f' % (np.matmul([1, 7], theta.T).sum() * 10000)) #predict2
```

```
For population = 70,000, we predict a profit of 44558.060147
```

In [21]:
```
h = [np.matmul(x, theta.T).sum() for x in X1.to_numpy()]
data1_plot = data1.join(pd.DataFrame({'Linear regression': h}))
```

In [22]:
```
plt.figure()
ax = data1_plot.plot(x='Population', y='Profit',  style=['rx'])
data1_plot.plot(x='Population', y='Linear regression', ax=ax)
plt.show()
```

Out[22]: <Figure size 432x288 with 0 Axes>

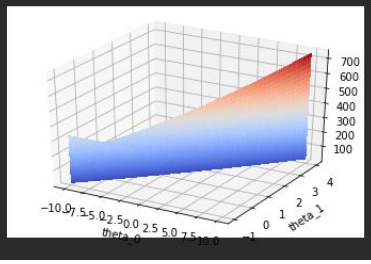


5.Постройте трехмерный график зависимости функции потерь от параметров модели (θ_0 и θ_1) как в виде поверхности, так и в виде изолиний (contour plot).

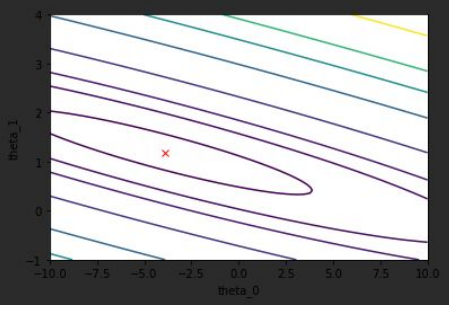
In [23]:
```
theta0_vals = np.linspace(-10, 10, num=100)
theta1_vals = np.linspace(-1, 4, num=100)
#theta0_vals, theta1_vals = np.meshgrid(theta0_vals, theta1_vals)
# initialize J_vals to a matrix of 0's
J_vals = np.zeros((theta0_vals.size, theta1_vals.size))
```

In [24]:
```
# Fill out J_vals
for i in range(0, theta0_vals.size):
    for j in range(0, theta1_vals.size):
        t = np.array([[theta0_vals[i], theta1_vals[j]]])
        J_vals[i, j] = computeCost(X1.to_numpy(), y1.to_numpy(), t)

J_vals = J_vals.T
```

In [25]:
```
# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(theta0_vals, theta1_vals, J_vals, cmap=cm.coolwarm, linewidth=0, antialiased=False)
plt.xlabel('theta_0')
plt.ylabel('theta_1');
```



In [26]:
```
fig, ax = plt.subplots()
ax.contour(theta0_vals, theta1_vals, J_vals, levels=[10, 30, 50, 100, 200, 300, 400, 600, 800])
ax.plot([theta[0, 0]], [theta[0, 1]], 'rx')
plt.xlabel('theta_0')
plt.ylabel('theta_1');
```



6.Загрузите набор данных **ex1data2.txt** из текстового файла.

In [27]:
```
data_row2 = np.genfromtxt('ex1data2.txt', delimiter=',')
data2 = pd.DataFrame(data_row2, columns=list(['size of the house', 'number of bedrooms', 'price']))
```

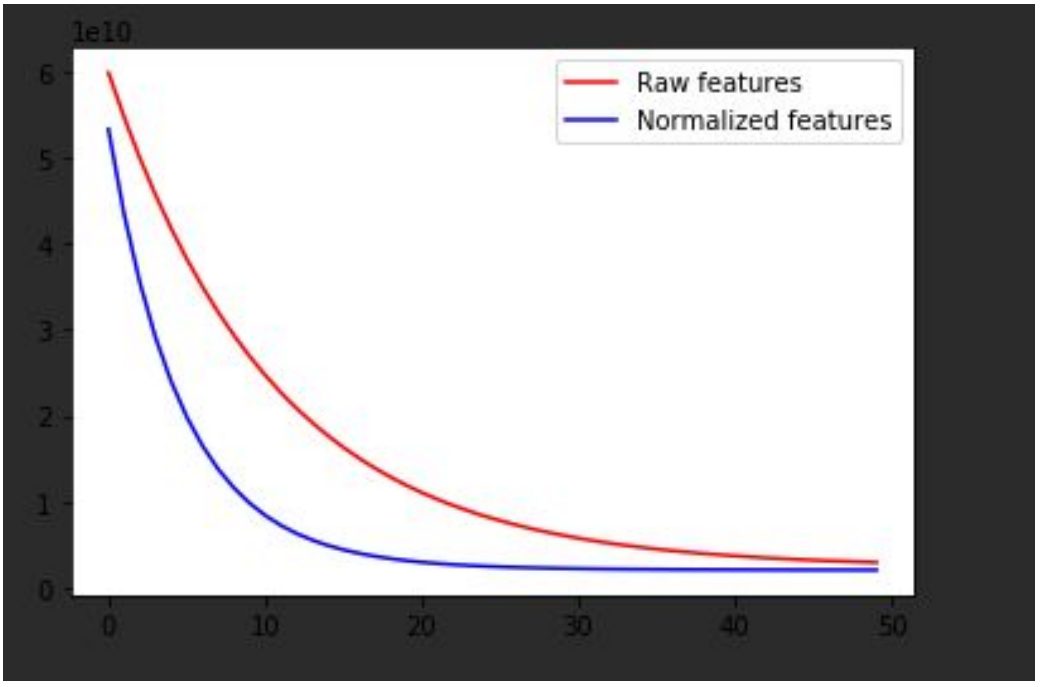
7.Произведите нормализацию признаков. Повлияло ли это на скорость сходимости градиентного спуска? Ответ дайте в виде графика.

In [28]:
```
#FEATURENORMALIZE Normalizes the features in X
#   FEATURENORMALIZE(X) returns a normalized version of X where
#   the mean value of each feature is 0 and the standard deviation
#   is 1. This is often a good preprocessing step to do when
#   working with learning algorithms.

def featureNormalization(X):
    norm = (X - X.mean(axis=0)) / X.std(axis=0)
    mu = X.mean(axis=0)
    sigma = X.std(axis=0)
    return norm, mu, sigma
```

In [30]:
```
y = data2['price']
m = y.size
n = data_row2.shape[1] # Size of feature vector
X.insert(0, 'theta_0', 1)
X_norm.insert(0, 'theta_0', 1)

theta1 = np.zeros((1, n)) # theta coeficents for hypothesis func
theta2 = np.zeros((1, n)) # theta coeficents for hypothesis func
```

In [31]:
```
(theta1, j_history) = gradientDescent(X.to_numpy(), y.to_numpy(), theta1, 0.00000001, 50)
(theta2, j_norm_history) = gradientDescent(X_norm.to_numpy(), y.to_numpy(), theta2, 0.1, 50)
```

In [32]:
```
p1 = plt.plot(range(0, len(j_history)), j_history, color='red')
plt.legend('Raw features')
p2 = plt.plot(range(0, len(j_norm_history)), j_norm_history, color='blue')
plt.legend((p1[0], p2[0]), ('Raw features', 'Normalized features'))
plt.show()
```



8.Реализуйте функции потерь J(θ) и градиентного спуска для случая многомерной линейной регрессии с использованием векторизации.

```
In [33]: def gradientDescentV(X, y, theta, alpha, num_iters):
             m = y.shape[0]  # Size of training set
             j_history = []
             XT = X.T
             for i in range(0, num_iters):
                 h = [np.matmul(x, theta.T)[0] for x in X]
                 loss = h - y
                 cost = np.sum(loss ** 2) / (2 * m)
                 gradient = np.matmul(XT, loss) / m
                 theta[0] -= alpha * gradient
                 j_history.append(cost)

             return theta, j_history
```

```
In [34]: iterations = 400
         alpha = 0.01
         theta_GD = np.zeros((1, n))  # theta coeficents for hypothesis func

         (theta_GD, j_history) = gradientDescentV(X_norm.to_numpy(), y.to_numpy(), theta_GD, alpha, iterations)
         print('Theta found by gradient descent: %s' % (theta_GD))

         Theta found by gradient descent: [[334302.06399328 100087.11600585  3673.54845093]]
```

```
In [35]: from __future__ import division
         price = np.array([1, (1650 - mu[0])/sigma[0], (3 - mu[1]) / sigma[1]]) @ theta_GD.T
         print('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): %f' % price)
```

Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): 289314.620338

9.Покажите, что векторизация дает прирост производительности.

In [36]:
```
from timeit import default_timer as timer

iterations = 1000
alpha = 0.02
theta = np.zeros((1, n))

start = timer()
(theta, j_history) = gradientDescent(X_norm.to_numpy(), y.to_numpy(), theta, alpha, iterations)
end = timer()
gd_exec_time = end-start
print("Theta %s | Execution time: %f" % (theta, gd_exec_time))
```

```
Theta [[340412.65900156 110620.78816241 -6639.21215439]] | Execution time: 0.279205
```

In [37]:
```
theta = np.zeros((1, n))

start = timer()
(theta, j_history) = gradientDescentV(X_norm.to_numpy(), y.to_numpy(), theta, alpha, iterations)
end = timer()
gdv_exec_time = end-start
print("Theta %s | Execution time: %f" % (theta, gdv_exec_time))
```

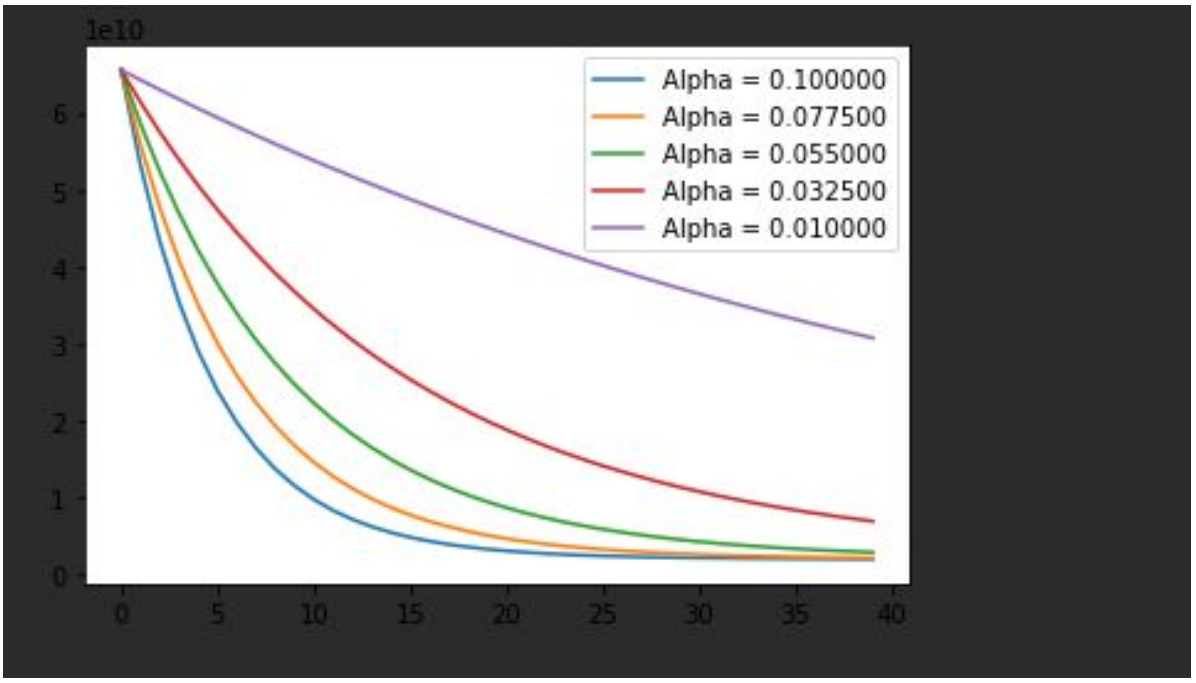
```
Theta [[340412.65900156 110620.78816241 -6639.21215439]] | Execution time: 0.074024
```

In [38]:
```
print('Vectorized gradient descent is %0.1fX faster' % (gd_exec_time / gdv_exec_time))
```

```
Vectorized gradient descent is 3.8X faster
```

10.Попробуйте изменить параметр α (коэффициент обучения). Как при этом изменяется график функции потерь в зависимости от числа итераций градиентного спуск? Результат изобразите в качестве графика.

In [39]:
```
alphas = np.linspace(0.1, 0.01, num=5)
plots = []
for alpha in alphas:
    theta = np.zeros((1, n))
    (theta, j_history) = gradientDescentV(X_norm.to_numpy(), y.to_numpy(), theta, alpha, 40)
    p = plt.plot(range(0, len(j_history)), j_history)
    plots.append(p[0])

plt.legend(plots, ["Alpha = %f" % (x) for x in alphas])
plt.show()
```



11.Постройте модель, используя аналитическое решение, которое может быть получено методом наименьших квадратов. Сравните результаты данной модели с моделью, полученной с помощью градиентного спуска.

```
In [42]: # computes the closed-form solution to linear regression using the normal equations
         def normalEqn(X, y):
             XX = np.asmatrix(X)
             XT = XX.T
             return ((XT @ XX).I @ XT) @ y
```

```
In [43]: theta_A = normalEqn(X.to_numpy(), y.to_numpy())
         print('Theta computed from the normal equations: %s' % (theta_A))
```

Theta computed from the normal equations: [[89597.9095428 1067402 -8738.01911233]]

```
In [52]: print('Theta computed from the normal normalized gradient descent: %s' % (theta_GD))
```

Theta computed from the normal normalized gradient descent: [[334302.06399328 100087.11600585 3673.54845093]]

```
In [53]: price = np.array([1, 1650, 3]) @ theta_A.T
         print('Predicted price of a 1650 sq-ft, 3 br house (using normal equations): %f' % price)
```

Predicted price of a 1650 sq-ft, 3 br house (using normal equations): 293081.464335

```
In [54]: price = np.array([1, (1650 - mu[0]) / sigma[0], (3 - mu[1]) / sigma[1]]) @ theta_GD.T
         print('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): %f' % price)
```

Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): 289314.620338

Выводы

В данной работе было показано как работает модель Линейной регрессии, для нахождения решения задачи линейной регрессии были использованы вычислительный метод градиентного спуска и аналитический метод наименьших квадратов.

Для градиентного спуска с помощью графиков показана зависимость скорости сходимости в зависимости от параметров и количества итераций

Также в пункте 7 было исследовано влияние нормализации на скорость сходимости градиентного спуска и показано, что нормализация увеличивает скорость сходимости для градиентного спуска.

В пункте 10 была показана зависимость сходимости от параметра Alpha. Если скорость обучения alpha слишком мала, у нас будет медленная сходимость, если alpha слишком велика, функция потерь может не уменьшаться на каждой итерации и может не сходиться.