Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей

Специальность “Обработка больших объемов информации”

Лабораторная работа №1

**Линейная регрессия**

Выполнил:

магистрант гр. 858641 Кальман В.А.

Проверил:

Стержанов М. В.

Минск 2019

**Данные.**

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).

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

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

%matplotlib inline

ex1data1 = np.loadtxt('Data/Lab 1/ex1data1.txt', delimiter=',')

X, y = np.expand\_dims(ex1data1[:, 0], axis=1), ex1data1[:, 1]

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

plt.figure(figsize=(16, 10))

plt.scatter(data[:, 0], data[:, 1], color='r')

plt.xlabel('Population', size=16)

plt.ylabel('Revenue', size=16)

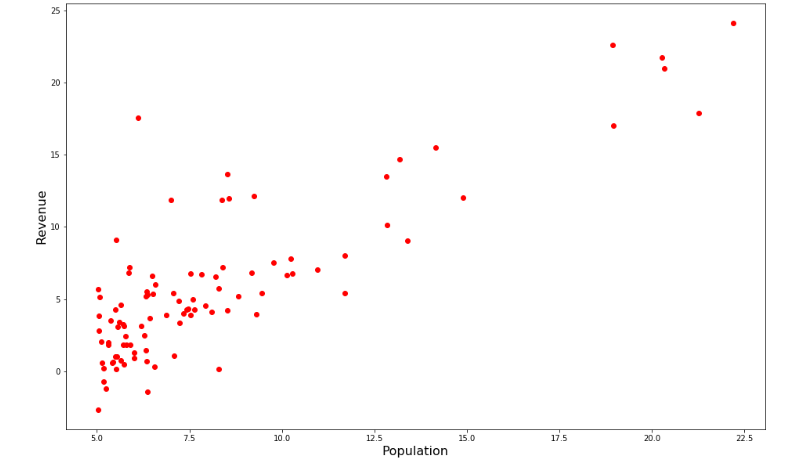
plt.show()

Рисунок 1 – график зависимости прибыли ресторана от населения (x – численность популяции, y – прибыль ресторана)

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

def compute\_cost(X, Y, theta):

m = len(X)

diff = []

for i in range(0, m):

val = pow(h0x(X[i], theta) - Y[i], 2)

diff.append(val)

cost = (1 / (2 \* m)) \* sum(diff)

return cost

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

def gradient\_descent(X, Y, theta, iterations, alpha):

"""

From Andrew Ng implementation: without ones vector in X

"""

m = len(X)

J = []

for i in range(iterations):

val = np.zeros(len(theta))

for j in range(0, m):

val[0] += h0x(X[j], theta) - Y[j]

for k in range(1, len(theta)):

val[k] += (h0x(X[j], theta) - Y[j]) \* X[j]

for z in range(0, len(theta)):

theta[z] = theta[z] - (alpha / m) \* val[z]

J.append(compute\_cost(X, Y, theta))

return [theta, J]

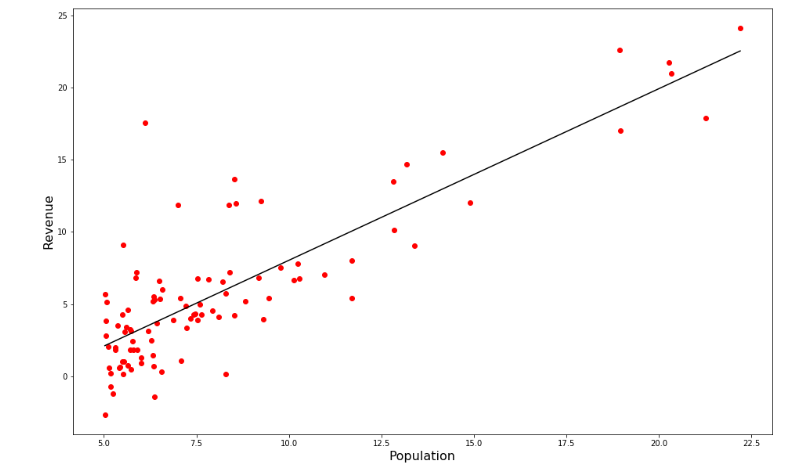


Рисунок 2 – график полученной модели

**5. Постройте трехмерный график зависимости функции потерь от параметров модели (θ0 и θ1) как в виде поверхности, так и в виде изолиний (contour plot).**

theta\_0\_range, theta\_1\_range = np.arange(-10, 10, 0.1), np.arange(-10, 10, 0.1)

theta\_0, theta\_1 = np.meshgrid(theta\_0\_range, theta\_1\_range)

values = []

**for** w1, w2 **in** zip(theta\_0.flatten(), theta\_1.flatten()):

model = Model(dim=1)

model.w = np.array([w1, w2])

loss = MSELoss(model)

loss\_value = loss(X, y)

values.append(loss\_value)

values = np.array(values)

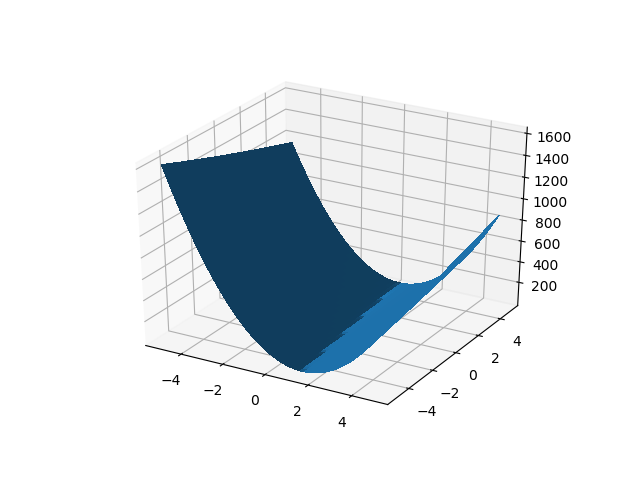
values = values.reshape((len(theta\_1\_range), len(theta\_0\_range)))

fig = plt.figure(figsize=(16,10))

plt.contour(theta\_0, theta\_1, values,

levels=[i\*10 **for** i **in** range(0,100)], cmap='viridis')

plt.scatter(\*trained\_model.w)

 Рисунок 3 – трёхмерный график зависимости потерь от параметров модели в виде поверхности

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

ex1data2 = np.loadtxt('Data/Lab 1/ex1data2.txt', delimiter=',')

X, y = ex1data2[:, :2], ex1data2[:, 2]

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

X\_norm = (X - X.mean(axis=0)) / X.std(axis=0)

results\_without\_norm = train\_model(X, y)

results\_with\_norm = train\_model(X\_norm, y)

print('number of iterations: **\n** without normalization: **{}**, with normalization: **{}**'.format(

results\_without\_norm['iter\_num'],

results\_with\_norm['iter\_num']))

plt.figure(figsize=(16,10)) plt.yscale('log') plt.plot(range(21), results\_without\_norm['loss\_steps']) plt.plot(range(21), results\_with\_norm['loss\_steps'])

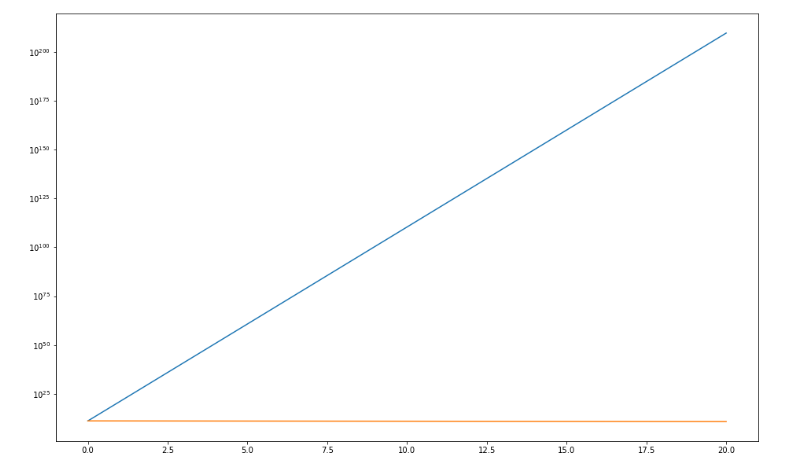


Рисунок 4 – график расхождения градиентного спуска без нормализации

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

def compute\_cost\_vectorized(X, Y, theta):

# J = (1 / (2 \* m)) \* (X \* theta - y)' \* (X \* theta - y); % equally (sum(power(X, 2)))

m = len(X)

temp = (h0x\_vectorized(X, theta) - Y)

return (1 / (2 \* m)) \* np.dot(temp.T, temp)[0][0]

def gradient\_descent\_vectorized(X, Y, theta, iterations, alpha):

m = len(Y)

J\_history = []

for i in range(iterations):

# theta = theta - alpha \* (1/m) \* (((X\*theta) - y)' \* X)'; % Vectorized

h0x = (h0x\_vectorized(X, theta) - Y).T

dt = np.dot(h0x, X).T

a = alpha \* (1 / m) \* dt

theta = theta - a

J\_history.append(compute\_cost\_vectorized(X, Y, theta))

return [theta, J\_history]

**9. Покажите, что векторизация дает прирост производительности.**

model = results\_with\_norm['model']

loss\_simple = MSELoss(model)

loss\_vectorized = MSELossVectorized(model)

print('Скорость вычисления лосса без векторизации:')

%timeit -n 100 loss\_simple(X\_norm, y)

Скорость вычисления лосса без векторизации

204 µs ± 91.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

print('Скорость вычисления лосса с векторизацией:')

%timeit -n 100 loss\_vectorized(X\_norm, y)

Скорость вычисления лосса с векторизацией:

180 µs ± 87 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

print('Скорость вычисления градиента без векторизации:')

%timeit -n 100 loss\_simple.grad(X\_norm, y)

Скорость вычисления градиента без векторизации:

432 µs ± 150 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

print('Скорость вычисления градиента с векторизацией:')

%timeit -n 100 loss\_vectorized.grad(X\_norm, y)

Скорость вычисления градиента с векторизацией:

278 µs ± 90.3 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

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

results\_01 = train\_model(X\_norm, y, alpha=0.01)

results\_001 = train\_model(X\_norm, y, alpha=0.001)

plt.figure(figsize=(16,10))

plt.yscale('log')

plt.plot(results\_01['loss\_steps'])

plt.plot(results\_001['loss\_steps'])

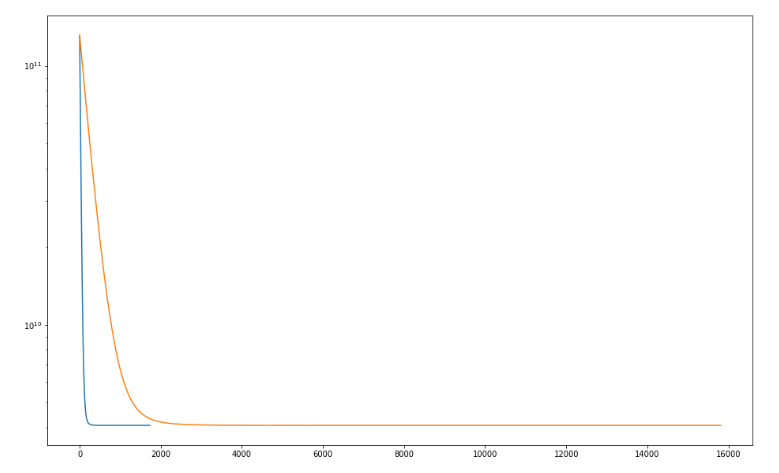


Рисунок 5 – график зависимости функции стоимости от количества итераций (alpha=0.01)

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

Solution:

[[340412.65957447]

[110631.05027885]

[ -6649.47427082]]

Normal equation:

[[340412.65957447]

[110631.05027885]

[ -6649.47427082]]

**Выводы**

Была изучена линейная регрессия с использованием метода градиентного спуска для минимизации функции потерь.