Учреждение образования

«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе:

Лабораторная работа №1 “Линейная регрессия”

Выполнил: Карп Александр Игоревич
магистрант кафедры информатики
группа №858641

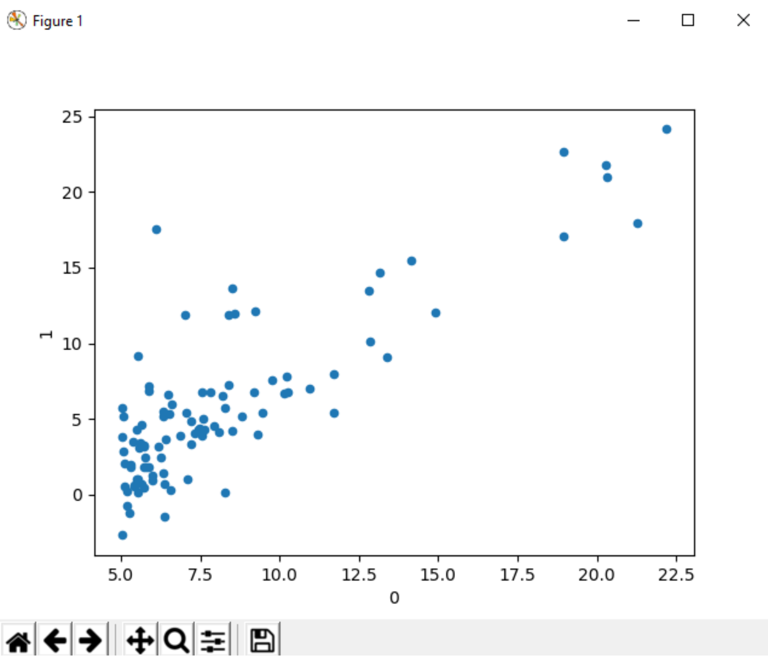
Минск 2019

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

```
data = pd.read_csv('ex1data1.txt', header=None)
num_columns = data.shape[1]

panda_X = data.iloc[:, 0:num_columns - 1]
panda_X.insert(0, 'Ones', 1)
X = panda_X.values
y = data[num_columns - 1].values
```

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

```
data.plot.scatter(x=0, y =1)
plt.show()
```



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

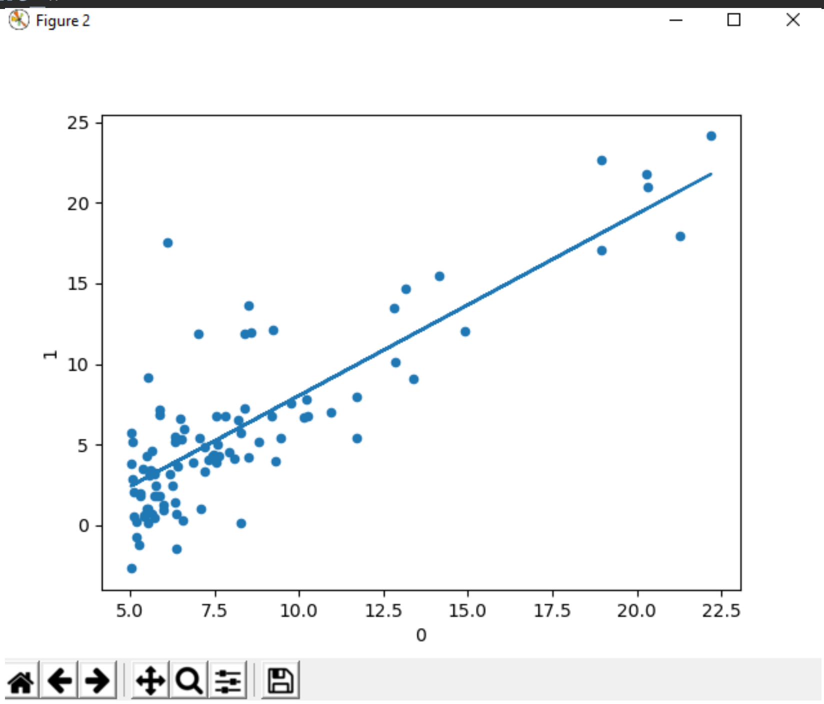
$$F_{loss} = 64.145, \theta = \bar{0}$$

```
def loss(X, y, w):
    units = np.full((len(X)), 1)  # единичный вектор
    return ((X @ w - y) ** 2 @ units) / len(X)
```

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

$$\theta_{optimal} = [-3.24, 1.13]$$

```
def gradient(X, y, w):
    n = len(w)
    grad = [0.0] * n
    l = len(X)
    for k in range(0, n):
        for i in range(0, l):
            temp = 0
            for m in range(0, n):
                temp += w[m] * X[i][m]
            temp -= y[i]
            grad[k] += temp * X[i][k]
        grad[k] = (grad[k] * 2) / l

    return grad


def gradient_descent(X, y, w, learning_rate=0.0001, k=0.01, steps=10000):
    n = len(w)
    t = 1

    grad = gradient(X, y, w)
    next_w = [0.0] * n
    for i in range(0, n):
        next_w[i] = w[i] - k  * grad[i]

    while (t < steps):
        w = next_w
        grad = gradient(X, y, w)
        for i in range(0, n):
            next_w[i] = w[i] - k * grad[i]
        t += 1

    return next_w
```



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).

```
def loss_plot(X, y):
    fig = plt.figure()
    ax = fig.gca(projection='3d')

    u = np.arange(-5, 5, 0.1)
    v = np.arange(-5, 5, 0.1)
    z = np.zeros((len(u), len(v)))
    for i in range(0, len(u)):
        for j in range(0, len(v)):
            z[i][j] = loss(X, y, [u[i], v[j]])

    u, v = np.meshgrid(u, v)

    # Plot the surface.
    ax.plot_surface(u, v, z,
                           linewidth=0, antialiased=False)
    plt.show()

def loss_contour_plot(X, y):
    fig, ax = plt.subplots()

    u = np.arange(-20, 20, 0.2)
    v = np.arange(-10, 10, 0.1)
    z = np.zeros((len(u), len(v)))
    for i in range(0, len(u)):
        for j in range(0, len(v)):
            z[i][j] = loss(X, y, [u[i], v[j]])

    u, v = np.meshgrid(u, v)

    # Plot the contour plot.
    ax.contour(u, v, z, levels=[20, 50, 125, 300, 750, 1800, 4500, 11250,
28000])

    plt.show()
```

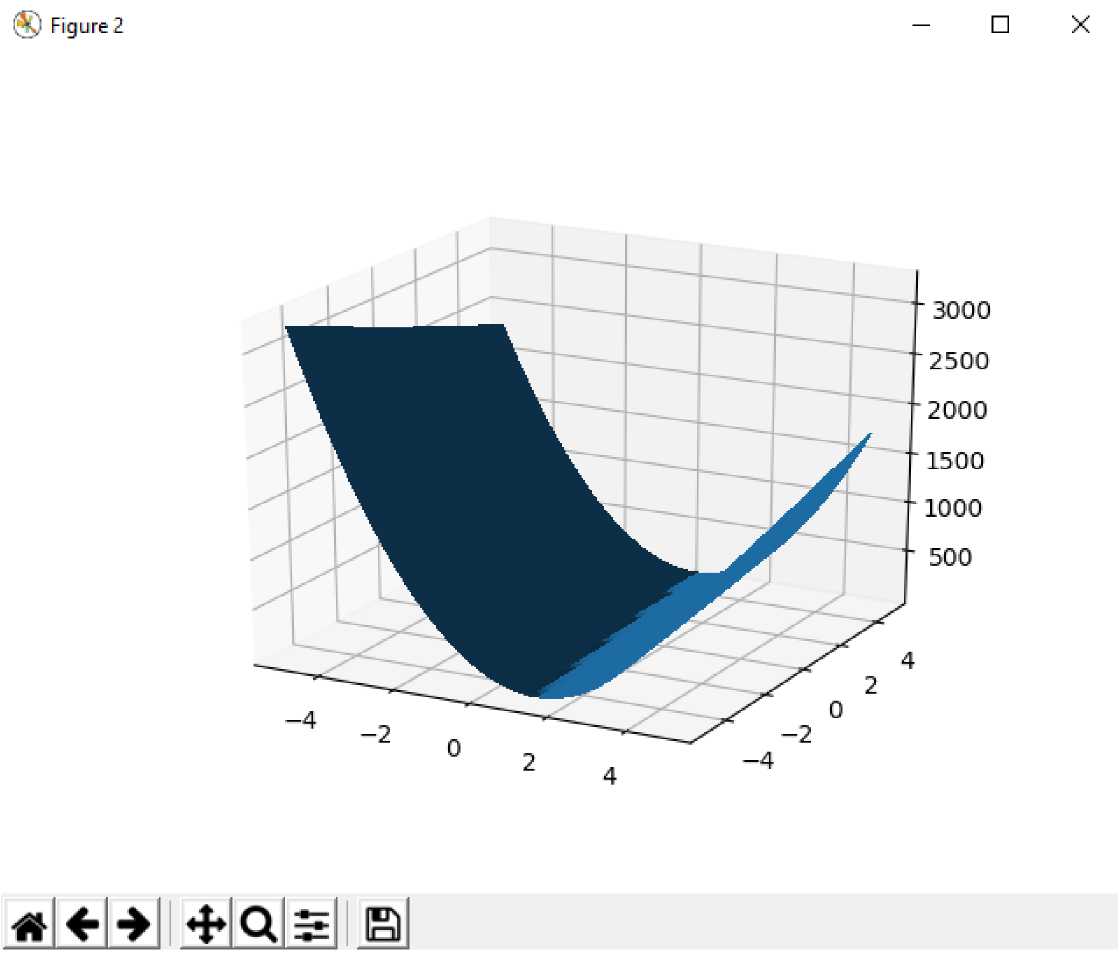


Figure 2
3000
2500
2000
1500
1000
500
−4
−2
0
2
4
4
2
0
−2
−4

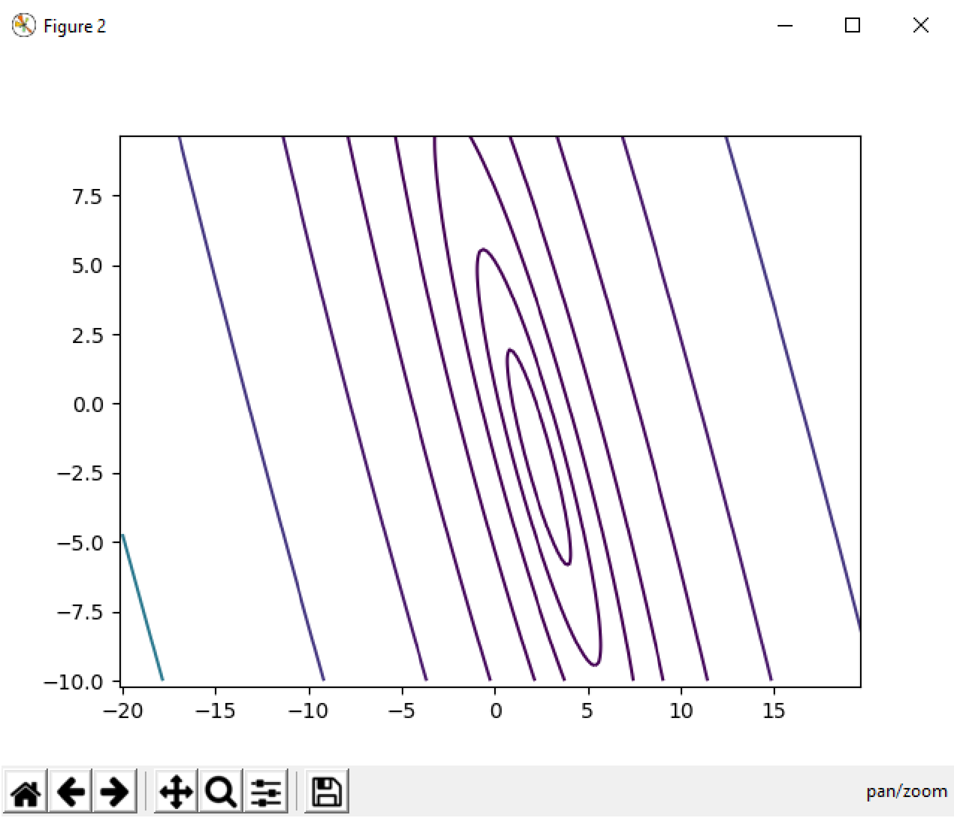


Figure 2
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−20
−15
−10
−5
0
5
10
15
pan/zoom

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

```
#6
data = pd.read_csv('ex1data2.txt', header=None)
num_columns = data.shape[1]
y = data[num_columns - 1].values
X = data.iloc[:, 0:num_columns - 1]
```

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

```
#7
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform()
df = pd.DataFrame(x_scaled)
df.insert(0, 'Ones', 1)
x_scaled = df.values
print(x_scaled)
```

Без нормализации алгоритм расходился, так что нормализация определенно повлияла на сходимость.

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

```
def loss(X, y, w):
    units = np.full((len(X)), 1)  # единичный вектор
    return ((X @ w - y) ** 2 @ units) / len(X)
def gradient_vectorized(X, y, w):
    return (2 / len(X)) * X.T @ (X @ w - y)


def gradient_descent_vectorized(X, y, w, learning_rate=0.0001, k=0.1,
steps=10000):
    t = 1
    next_w = w - k * gradient_vectorized(X, y, w)
    while np.linalg.norm(w - next_w) > learning_rate and t < steps:
        w = next_w
        next_w = w - k * gradient_vectorized(X, y, w)
        t += 1
    return next_w
```

$$\theta_{optimal} = [199467.37690402\ 504777.88748915\ -34952.05269712]$$

$$F_{loss\,\theta_{optimal}} = 4086560101.205672$$

9) Покажите, что векторизация дает прирост производительности.

```
#8
start1 = time.time()
theta = gradient_descent_vectorized(x_scaled,y,np.array([0,0,0]))
time1 = time.time() - start1
print(theta)
print(loss(x_scaled, y, theta))


#9
start2 = time.time()
theta = gradient_descent(x_scaled,y, np.array([0,0,0]))
time2 = time.time() - start2
print(time1, time2)
```

Время сходимости алгоритма без использования векторизации: $t_1 = 4.25$

С использованием: $t_2 = 0.05688$

Скорость сходимости с использованием векторизации увеличилась в 74 раза

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

```
def lsm(X, y):
    return ((np.linalg.inv((X.T.dot(X)))).dot(X.T)) @ y
```

$$\theta_{optimal}(least\ square\ method) = [199467.38469349\ 504777.90398791\ -34952.07644931]$$

$$F_{loss\,\theta_{optimal}}(least\ square\ method) = 4086560101.2056565$$

Результаты метода наименьших квадратов и метода градиентного спуска практически идентичны.

Выводы

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

К плюсам линейных моделей можно отнести:

- Хорошо изучены
- Очень быстрые, могут работать на очень больших выборках
- Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные
- Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

К минусам:

- Плохо работают в задачах, в которых зависимость ответов от признаков сложная, нелинейная