Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Лабораторная работа №1

«Линейная регрессия»

Выполнил: Чёрный Родион Павлович
магистрант кафедры информатики
группа №858642

Проверил: доцент кафедры информатики
Стержанов Максим Валерьевич

Минск 2019

Постановка задачи

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.
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.

Описание реализации

1. Код для загрузки файла представлен ниже. Данные хранятся в csv формате, поэтому для удобства воспользуемся методом read_csv из модуля pandas:

```
data = pd.read_csv(filename, header=None)
```

2. Зависимость прибыли ресторана от населения города, в котором он расположен, выглядит следующим образом:

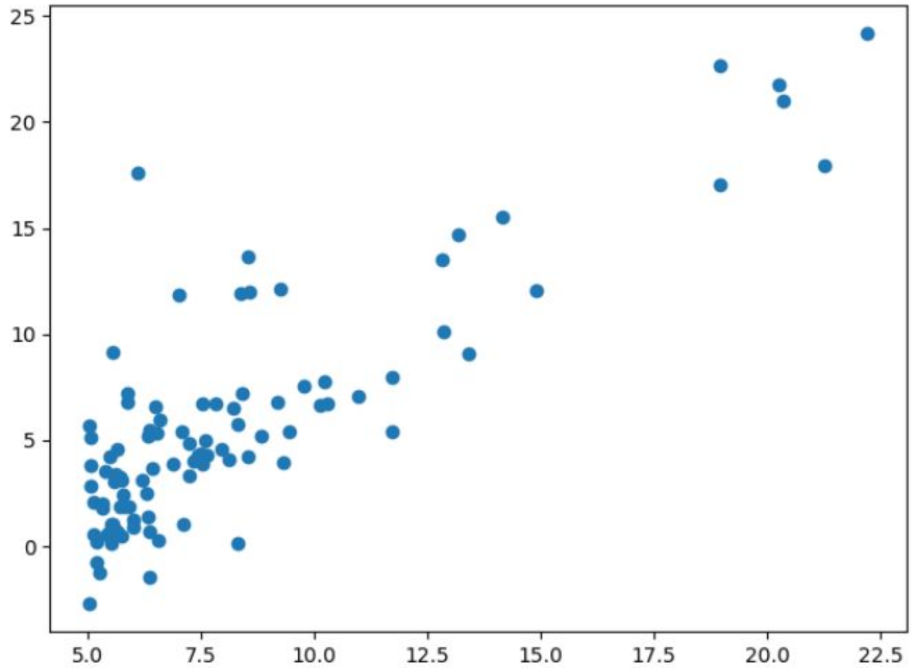

Рисунок 1 - Зависимость прибыли ресторана от населения города

3. Функция потерь представляет собой среднюю сумму квадратов ошибок (mean squared error), ее реализация:

```
def loss(self, w0, w1):
    error = self.Y - (w0 + w1 * self.X)
    return np.sum(np.dot(error, error.T)) / (2*self.n)
```

w0, w1 - параметры модели. X - исходный вектор значений населения города. Y - вектор ожидаемой прибыли, которую необходимо предсказать. n - длина вектора Y.

4. Реализация алгоритма градиентного спуска.

```
for iteration in range(number_of_iterations):
    w0_dir = (2/self.n) * np.sum(self.X * w1 + w0 - self.Y)
    w1_dir = (2/self.n) * np.sum((self.X * w1 + w0 - self.Y) * self.X)
    self.w0 = self.w0 - self.learning_rate * w0_dir
    self.w1 = self.w1 - self.learning_rate * w1_dir
```

Спустя 1000 итераций мы добились значения среднеквадратичной ошибки ~4.47. Полученной модели соответствует график:

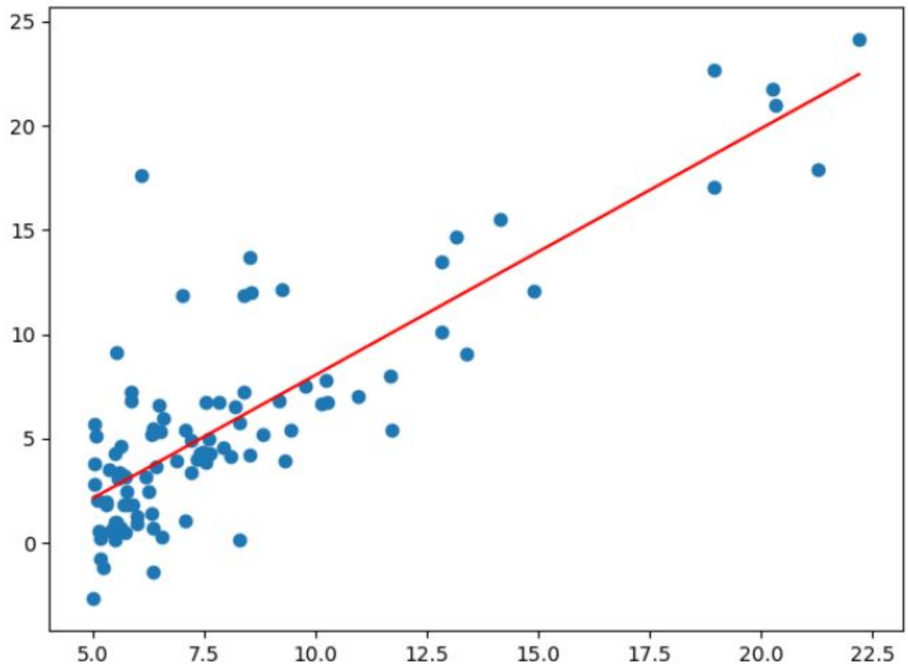

Рисунок 1 - Зависимость прибыли ресторана от населения города совмещенная с полученной функцей

5. По сути мы искали параметры модели, двигаясь в направлении противоположном градиенту в трехмерном пространстве. График зависимости функции потерь от параметров модели представлен ниже:

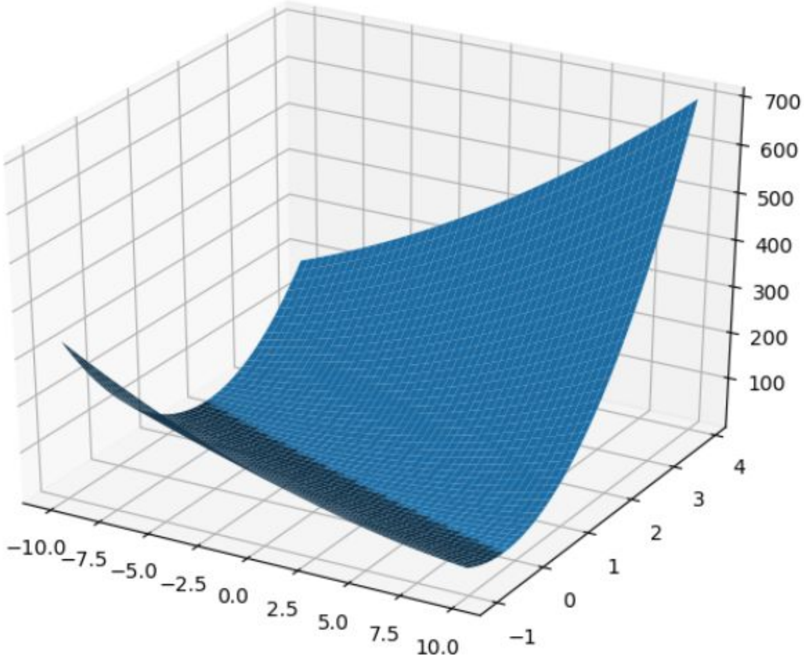

Рисунок 3 - График зависимости функции потерь от параметров модели в виде поверхности

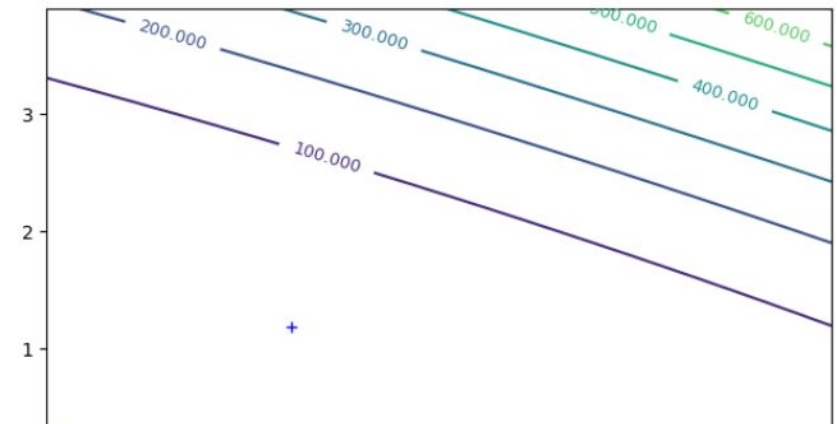

Рисунок 4 - График зависимости функции потерь от параметров модели в виде изолиний

6. Для загрузки данных из файла ex1data.txt воспользуемся уже реализованной ранее функцией.

```
data = load_data("ex1data2.txt")
```

Кривая обучения с новым набором данных принимает вид, указанный на рисунке 5. Можно увидеть, что функция достигает своего минимума где-то к 20-й итерации.

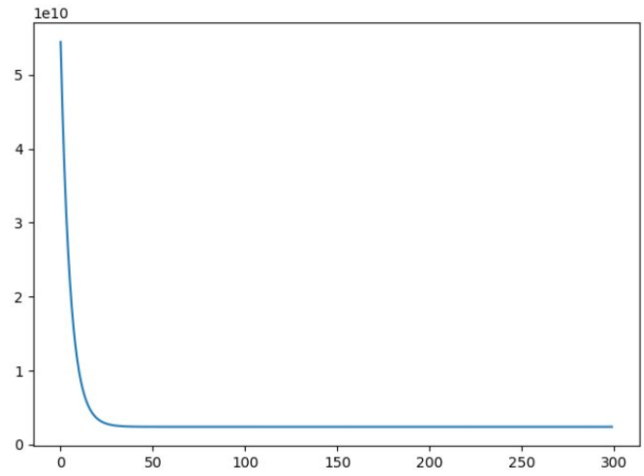

Рисунок 5 - кривая обучения со ненормализованными данными

7. Добавим к функции загрузки данных также нормализацию данных:

```
def load_data(filename, normalize=False):
   data = pd.read_csv(filename, header=None)
   if normalize:
       scaler = MinMaxScaler()
       scaled_values = scaler.fit_transform(data)
       data = pd.DataFrame(scaled_values)
   return data
```

После нормализации данных кривая обучения немного изменилась, схождение к минимуму стало медленнее.

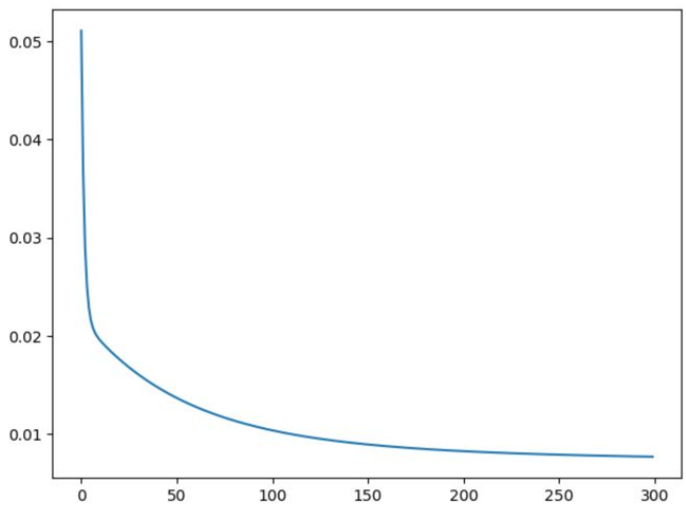

Рисунок 6 - кривая обучения с нормализованными данными

Пока что трудно тут сделать однозначным вывод, потому что для обоих случаев требовались разные шаги градиентного спуска: для ненормализованных данных - 1e-8, для нормализованных - 1e-1, потому что значение градиента в случае ненормализованных данных получается очень большим и, соответственно, большим становится шаг.

8. Векторная реализация функции потерь и алгоритма градиентного спуска:

```
def loss(self, model_parameters):
    error = self.Y - (np.dot(self.X, model_parameters))
    return np.sum(np.dot(error, error.T)) / (2*self.n)

def calculate_descent_direction(self):
    dw = []
    E = np.dot(self.X, self.model_parameters.T) - self.Y
    dw0 = np.dot(E, self.X.T[0]) / self.n
    dw.append(dw0)
    for i in range(1, len(self.model_parameters)):
        dwi = np.dot(E, self.X.T[i])
        dw.append(dwi / self.n)
    dw = np.array(dw)
    return dw

def train(self, iteration_count=1000):
    for it in range(iteration_count):
        dw = self.calculate_descent_direction()
        self.model_parameters = self.model_parameters - self.learning_rate * dw
        new_err = self.loss(self.model_parameters)
        self.learning_history.append(new_err)
```

9. Сравним скорости выполнения векторизированной и невекторизированной реализаций. Для большей точности, запустим тренировку каждой реализации по 25 раз и сравним время выполнения. График сравнения представлен на рисунке 7. Как видно из графика, разница существенная.

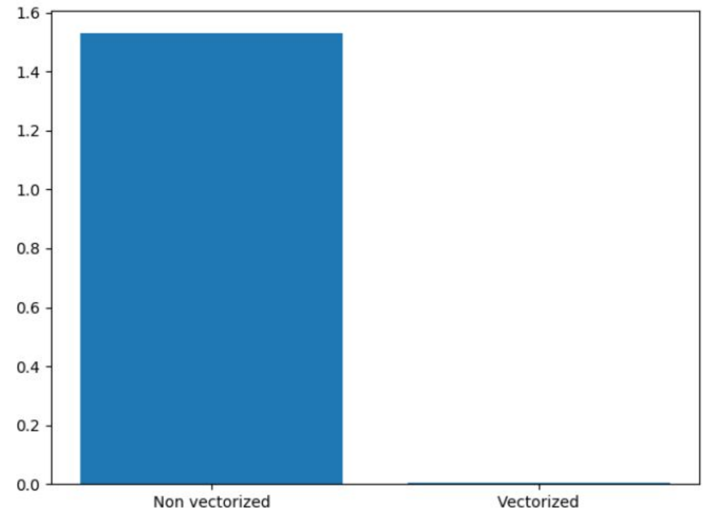

Рисунок 7 - сравнение производительности векторизированной реализации и невекторизированной

10. Меняя коэффициент обучения α увидим, что при его увеличении ускоряется сходимости к минимуму функции потерь. Сравнение кривых обучения изображено на рисунке 8.

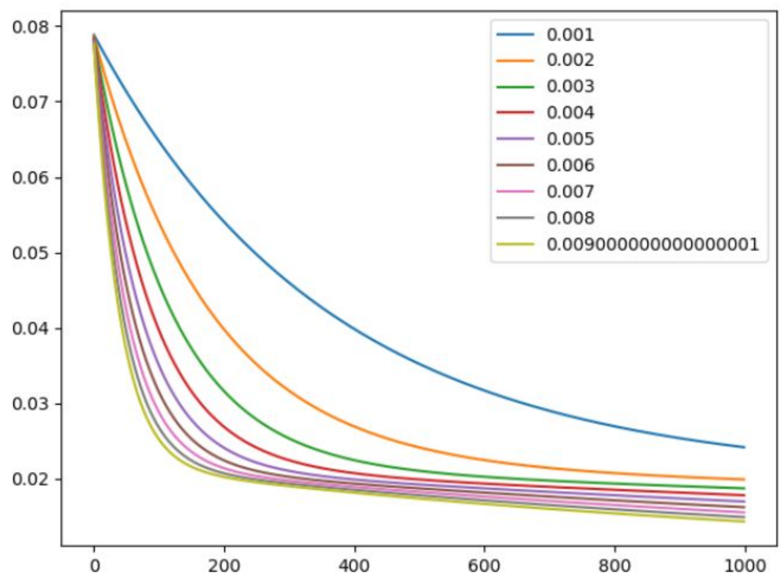

Рисунок 8 - кривые обучения при разных значениях α

11. При аналитическом способе нахождения параметры модели составили [0.0557875 0.952411 -0.0659473], соответствующее значение функции потерь: 0.00727.

После 1000 итераций градиентного спуска параметры модели составили [0.029095 0.886422 0.0211142] и значение функции потерь: 0.00738.

Выводы

Как можем увидеть, аналитический подход дал более точное решение и в нашем случае, при размере тренировочной выборки в 47 элементов этот подход гораздо точнее и быстрее. Но если бы элементов было в разы больше, то находить параметры модели аналитически стало бы очень трудоемко.

Также при реализации линейной регрессии все вычисления целесообразно выполнять в векторном виде, так как это дает значительный прирост производительности.

У различных признаков диапазон значений может сильно отличаться, например число комнат в доме (1 - 4) и площадь дома (1000 - 2500) и для того, чтобы все признаки оказывали одинаковое влияние на результат выходной функции, нам их необходимо нормализировать.

Среди плюсов использования линейной регрессии для предсказания значения функции можно выделить:

- хорошую изученность
- быстроту
- хорошо работают при большом количестве признаков