

## Задание к лабораторной работе № 1

### Одномерная и множественная линейная регрессия

В этой работе Вы реализуете алгоритм линейной регрессии и примените его для решения практических задач. Начнем с реализации алгоритма одномерной линейной регрессии.

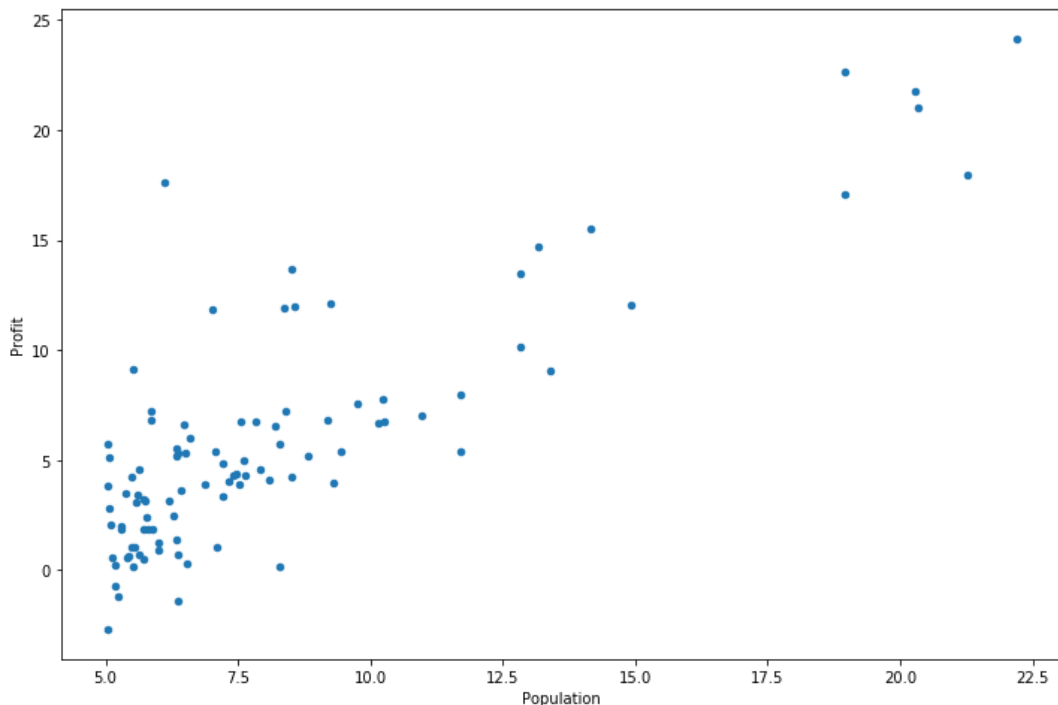
#### 1. Одномерная линейная регрессия

**Задача 1:** Вы являетесь владельцем сети ресторанов и рассматриваете варианты открытия дополнительных филиалов в других городах. Вы заметили закономерность между количеством жителей города и прибылью филиала Вашей сети ресторанов в этом городе.

Файл `ex1data1.txt` содержит данные о населении города (первый столбец) и прибыли ресторана в нем (второй столбец). Отрицательные значения во втором столбце говорят о том, что в этом городе Ваш бизнес является убыточным. Начнем с того, что посмотрим на эти данные.

**Загрузите таблицу с данными из файла `ex1data1.txt`, посмотрите на них. Для лучшей визуализации постройте точки с соответствующими двумя координатами (Population, Profit) на графике.**

У Вас должен получиться такой рисунок.



Очень полезно всегда анализировать имеющиеся данные. Мы видим, что у нас имеется некий кластер в районе значений низкой населенности городов и что то похожее на линейный тренд роста прибыли нашего бизнеса с ростом населенности города. Таким образом, метод линейной регрессии хорошо подходит для данной задачи.

Напомним, что мы собираемся минимизировать целевую функцию

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h(x_i) - y_i)^2, \quad (1)$$

где  $h(x) = \theta_0 + \theta_1 x$ ,  $m$  – число элементов в выборке. Неизвестными являются параметры  $\theta_0, \theta_1$ , которые мы и хотим подобрать так, чтобы значение  $J(\theta_0, \theta_1)$  было как можно меньше.

Мы будем это делать с помощью *алгоритма градиентного спуска*. Задав случайно начальные значения искомых параметров и шаг обучения  $\alpha$ , на каждой итерации мы обновляем параметры по формулам:

$$\theta_0^{new} = \theta_0^{old} - \alpha \cdot \frac{1}{m} \cdot \sum_{i=1}^m (h(x_i) - y_i), \quad (2)$$

$$\theta_1^{new} = \theta_1^{old} - \alpha \cdot \frac{1}{m} \cdot \sum_{i=1}^m (h(x_i) - y_i) \cdot x_i.$$

Следующее, что нам необходимо сделать, это написать функцию, которая будет вычислять значения целевой функции  $J(\theta_0, \theta_1)$  в зависимости от параметров  $\theta_0, \theta_1$ . Это нужно для того, чтобы мы могли убедиться, что это значение действительно уменьшается на каждом шаге итерационного алгоритма.

Для дальнейшего удобства переноса на случай многомерной линейной регрессии и с целью использования возможностей numpy в Python по матричным умножениям, введем обозначения:

$$X = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}.$$

Тогда (1) примет вид:

$$J(\theta) = \frac{(X\theta - Y)^T (X\theta - Y)}{2m}, \quad (3)$$

а формулы (2) для обновления  $\theta$  могут быть переписаны в следующем матричном виде:

$$\theta^{new} = \theta^{old} - \frac{\alpha}{m} \cdot X^T \cdot (X\theta - Y). \quad (4)$$

**Реализуйте функцию `computeCost(X, y, theta)`, вычисляющую значение целевой функции  $J(\theta)$  по формуле (3).**

**Найдите ее значение при  $\theta_0 = 0$ ,  $\theta_1 = 0$ . Проверьте, что оно равно 32,07.**

Далее нам нужно реализовать алгоритм градиентного спуска в задаче линейной регрессии. Наша функция будет возвращать не только новые значения  $\theta$ , полученные после *iters* итераций алгоритма с шагом обучения  $\alpha$ , но и хранить историю значений целевой

функции  $J(\theta)$  для всех промежуточных значений  $\theta$ . Это нужно нам для того, чтобы мы могли посмотреть, что происходило с целевой функцией во время обучения, правильно ли оно шло и шло ли вообще.

Кроме того, сейчас мы ведем обучение  $iters$  итераций и достаточно произвольно задаем это число. Другим возможным вариантом своевременной остановки обучения является достижение значения функции  $J(\theta)$  заданного порога. В случае необходимости мы легко сможем добавить в нашу функцию это дополнительное условие.

**Используя формулу (4) реализуйте функцию `gradientDescent(X, y, theta, alpha, iters)`, возвращающую новые значения параметров  $\theta_0, \theta_1$  и массив с  $iters$  значениями целевой функции на каждой итерации.**

**Для проверки ее работы убедитесь, что для  $\alpha = 0.01$  и  $iters = 1000$  и нулевых начальных значениях  $\theta_0, \theta_1$  у Вас получаются новые значения параметров  $\theta_0 = -3.24$ ,  $\theta_1 = 1.13$ .**

**Вычислите значение целевой функции для новых значений параметров. Стало ли оно меньше, чем 32.07?**

Давайте посмотрим, насколько хороши новые значения параметров  $\theta_0, \theta_1$ , которые мы получили после  $iters = 1000$  итераций алгоритма градиентного спуска. Как Вы помните, эти два параметра определяют положение прямой  $h(x) = \theta_0 + \theta_1 x$ . Поэтому далее мы построим полученную прямую и посмотрим на ее расположение относительно данных нам точек.

Кроме того, мы знаем, что поскольку алгоритм градиентного спуска минимизирует целевую функцию, то в процессе обучения ее значения должны уменьшаться. Проверим и это.

**Постройте график с точками (Population, Profit), как в начале этой работы и добавьте к нему прямую  $h(x) = \theta_0 + \theta_1 x$  с найденными значениями параметров  $\theta_0, \theta_1$ . Что Вы можете сказать о полученном рисунке?**

**Постройте график зависимости значений целевой функции от числа итераций и убедитесь, что Вы получили убывающую кривую.**

Мы решили задачу одномерной линейной регрессии.

Самостоятельно поэкспериментируйте с разными значениями шага обучения, числа итераций. Обратите внимание на то, как ведут себя значения целевой функции.

**Повторите решение задачи при  $\alpha = 0.05$ ,  $\alpha = 0.001$ . Объясните полученные результаты. Что изменилось на графике со значениями целевой функции и почему? Как быстро они уменьшаются?**

Теперь мы можем перейти к задаче множественной линейной регрессии.

## 2. Множественная линейная регрессия

Как правило, при прогнозировании значения некой величины нам нужно учитывать не один, а много факторов так или иначе влияющих на нее.

В этой части лабораторной работы Вы решите задачу предсказания цены на дом при известных значениях его площади (в фут<sup>2</sup>) и числе спален.

Исходные данные - в файле ex1data2.txt

Загрузите таблицу с данными, посмотрите на них. Ниже – часть таблицы.

Index	Size	Bedrooms	Price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900
5	1985	4	299900
6	1534	3	314900
7	1427	3	198999
8	1380	3	212000
9	1494	3	242500
10	1940	4	239999
11	2000	3	347000
12	1890	3	329999
13	4478	5	699900

Как видно, данные очень сильно отличаются по масштабу. Обычно у нас 2-5 спален, но площадь варьируется в пределах сотен, тысяч квадратных футов. Если "запустить" наш алгоритм на таких данных, вклад от переменной "size" превзойдет вклад от "bedrooms", которая просто "потеряется" на фоне "size". Чтобы избежать такой ситуации, нам нужно выровнять масштаб изменения наших переменных.

Нормируйте их. Для этого нужно вычесть из каждого значения признака среднее значения этого признака и разделить на его (признака) среднеквадратичное отклонение.

Помните про нормальную случайную величину с матожиданием 0 и ср.кв. отклонением 1?

Далее мы с радостью замечаем, что код программы, вычисляющий значение целевой функции и алгоритм градиентного спуска, которые мы не поленились и реализовали в прошлой задаче в матричном виде, не требует изменений в задаче множественной

линейной регрессии. Сейчас у нас два входных признака, но ясно, что все переносится без изменений и на любое другое их число.

**Найдите значение целевой функции при  $\theta_0 = \theta_1 = \theta_2 = 0$ . Проверьте, что оно равно 0,489.**

Теперь запустим алгоритм градиентного спуска для решения нашей задачи.

**Найдите значение параметров  $\theta_0, \theta_1, \theta_2$  после  $iters = 1000$  итераций алгоритма `gradientDescent(X, y, theta, alpha, iters)` с шагом  $\alpha=0.05$ .**

**Вычислите значение целевой функции для новых значений параметров. Убедитесь, что оно равно 0.13.**

Полезно также построить график поведения функции потерь в процессе обучения. Как он должен выглядеть?

**Постройте график изменения функции потерь в процессе обучения.**

**Можете ли Вы добиться меньшего, чем 0.13 значения этой функции? Можно ли получить 0?**

Убедитесь, что значения функции потерь уменьшаются с ростом числа эпох обучения и выходят на некоторое "плато" почти постоянного значения.

### 3. Возможности Python по решению задачи линейной регрессии

Мы большие молодцы, что реализовали алгоритм градиентного спуска для решения задачи линейной регрессии "с нуля". Ни в коем случае Вас не должно расстроить известие, что все это уже бесплатно реализовано в Python за счет его огромного сообщества разработчиков.

Мы, конечно, должны знать и встроенные возможности Python для машинного обучения. Однако для полного и настоящего понимания предмета такие решения "с нуля" безусловно необходимы. Теперь, когда Вы это умеете, можно раскрыть Вам секрет.

Давайте вернемся к первой задаче, вновь загрузим и приведем к надлежащему виду все наши данные.

Следующие строки кода выполняют всю работу метода градиентного спуска

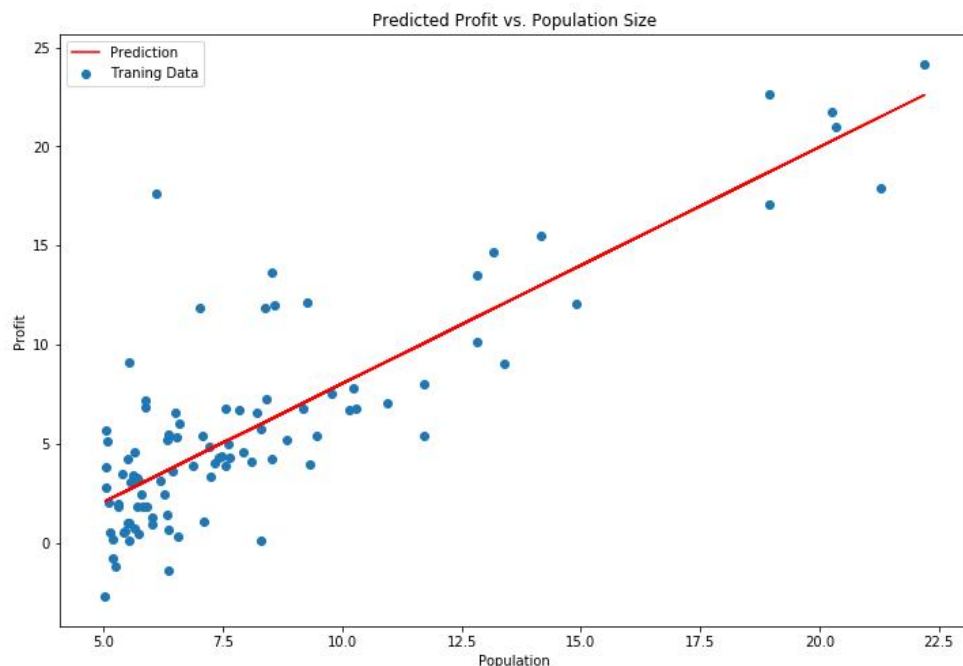
```
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(X, y)
```

У метода `fit` есть множество параметров, которые регулируют работу алгоритма [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LinearRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html), но значения по умолчанию довольно разумны и можно оставить `fit` в покое.

Теперь с помощью метода `predict` получим выходные значения обученной модели

```
f = model.predict(X)
```

и вновь построим точки и полученную прямую



**Продемонстрируйте свое умение решать обе рассмотренные задачи с помощью методов `fit` и `predict` из `sklearn`.**