# 1. Одномерная линейная регрессия

В этой части упражнения используется линейная регрессия с одной переменной для предсказания прибыли, полученной в результате торговли в разных городах. Предположим, что нужно рассмотреть некоторые города для открытия новой торговой точки.

Имеются данные о прибыли, которую можно получить в этих городах, а также численность населения этих городов. Нужно использовать эти данные, чтобы выбрать, какой город использовать следующим.

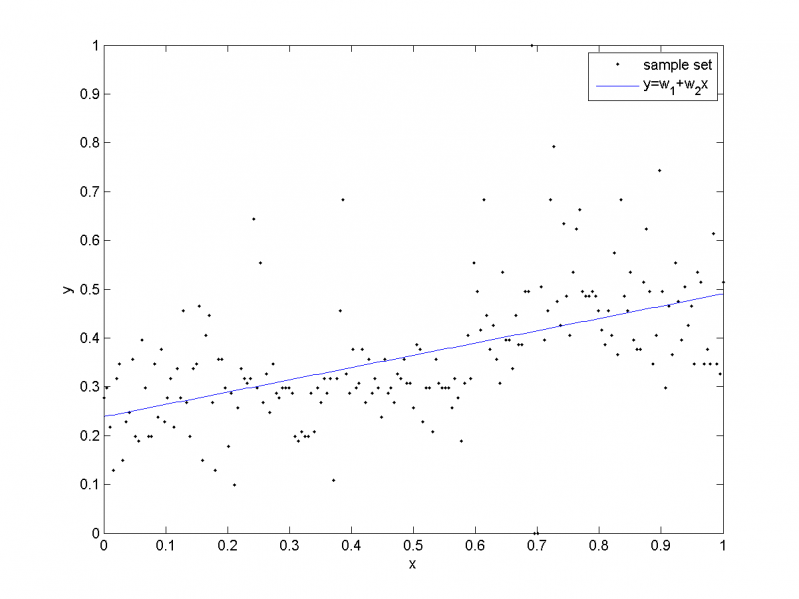
## Справка

**Регрессионный анализ** — статистический метод исследования влияния одной или нескольких независимых переменных на зависимую переменную.

**Линейная регрессия** — используемая в статистике регрессионная модель с линейной функцией зависимости.

В простейшем случае линейную модель можно представить так:

где — математическое ожидание зависимой переменной , когда переменная равна нулю; — ожидаемое изменение зависимой переменной при изменении на единицу. — случайная ошибка



Файл ex1data1.txt содержит набор данных для нашей задачи.

Первый столбец — численность населения города (в 10 000 чел.), второй столбец — прибыль, которую можно получить в этом городе (в 10 000-кратном размере). Отрицательное значение для прибыли свидетельствует об убыточности торговой точки.

**Задание1**. Напишите скрипт для загрузки этих данных в Python. Для этого рекомендуется создать текстовый файл с расширением .py, в котором вы будете записывать команды скрипта. Файлы скриптов Python должны располагаться в файлах именно с таким расширением.

При написании скрипта полезно бывает вставлять комментарии в код, поясняющие принцип работы скрипта. В Python для обозначения однострочного комментария используются символ #. Например

# пример создания матрицы NumPy в Python

a = np.array([[1, 2, 3], [4, 5, 6]])

Для загрузки данных из файла используется команда loadtxt из библиотеки numpy, в которую передаётся имя файла и параметр delimiter (разделитель значений в текстовом файле). В данной лабораторной работе в качестве файла с данными будет использоваться обычный текстовый файл с запятыми и переносами строк в качестве разделителей.

Если в качестве файла с данными используется обычный текстовый файл с разделителями, то возвращаемое функцией loadtxt значение необходимо присвоить некоторой переменной. Так как в файле хранится матрица значений, используем функцию matrix из numpy.

# в этом случае данные из файла будут сохранены в переменной data.

data = np.matrix(np.loadtxt('ex1data1.txt', delimiter=','))

Для того, чтобы загрузить данные из файла, может быть полезным перейти в директорию, содержащую данный файл. Для этого используется функция chdir модуля os. В качестве параметра передаётся абсолютный или относительный путь к нужной директории:

>>> import os

>>> os.chdir('D:')

Для проверки текущей рабочей директории используется команда getcwd модуля os:

>>> os.getcwd()

'D:\\'

**Задание2.** Нарисуйте данные, загруженные в задании 1 с помощью функции plot библиотеки matplotlib. Программно подпишите оси. Так как в нашей задаче всего 1 переменная, можно изобразить их в виде точечного графика на плоскости.

Для построения графиков в Python используется мощная библиотека matplotlib. Чтобы использовать её в своём скрипте, необходимо подключить её командой

import matplotlib.pyplot as plt

Для построения простых графиков проще всего использовать функцию plot. В качестве параметров ей передаётся либо 1 вектор (массив значений), либо 2, тогда первый будет содержать значения по оси абсцисс, а второй – значения по оси ординат. В качестве третьего параметра можно указать, как именно будет выглядеть график. Третий параметр представляет собой строку из 2 символов. Первый символ отвечает за цвет, например, r (red), g (green), b (blue), k (black). Второй символ отвечает за вид графика: - (непрерывная линия), o (кружочки), . (точки), \* (звёздочки).

‘ro’ – график в виде красных кружочков

‘b-‘ – график в виде непрерывной синей линии

‘b.’ – график в виде чёрных точек

Важно помнить, что если в функцию plot передаются данные по оси абсцисс и по оси ординат, то размер этих данных должен быть одинаковым, то есть, например, это должны быть вектора одинаковой длины.

Чтобы подписать оси и график, используются функции xlabel, ylabel и title соответственно. В качестве параметров все функции принимают строку. Для рисования сетки используется функция grid.

Пример построения точечного графика. Значения будут отмечены синими точками:

import matplotlib.pyplot as plt

from matplotlib import rc

font = {'family': 'Verdana', 'weight': 'normal'}

rc('font', \*\*font)

plt.plot(X, y, 'b.')

plt.title('Зависимость прибыльности от численности')

plt.xlabel('Численность')

plt.ylabel('Прибыльность')

plt.grid()

plt.show()

Обратите внимание на строки 2-4. Этот код необходим для того, чтобы на графике корректно отображались русские символы.

Вот как должен выглядеть итоговый график:

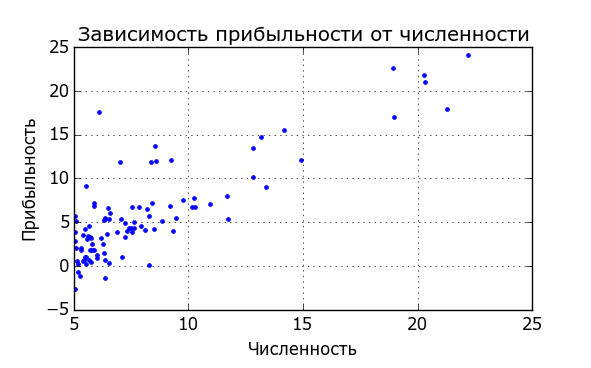


Рисунок 1 График зависимости прибыльности от численности города

Чтобы нарисовать такой график, вам потребуется выбрать значения из данных, которые вы загрузили из текстового файла в предыдущем задании. Первый столбец данных, отвечающий за численность города, необходимо сохранить в переменной X, второй столбец, отвечающий за прибыль, в переменной y.

В библиотеке NumPy присутствуют богатые возможности по выбору строк, столбцов и отдельных элементов из матриц:

>>> a = np.matrix([[1, 2], [3, 4], [5, 6]])

>>> a[0, 0] # выбрать первый элемент в матрице

1

Если в качестве одного из индексов указано двоеточие (:), значит подходят элементы с любым индексом. Так можно выбирать из матрицы целые строки или столбцы:

a[:, 1] # выбираем второй столбец (индекс строки – любой, индекс столбца – второй. Индексы начинаются с нуля)

a[2, :] # выбираем третью строку (индекс строки – третья, индекс столбца – любой)

Более того, в качестве индекса можно использовать массивы:

a[[0, 2], :] # берём из матрицы a первую и третью строки

Так же можно выбирать диапазоны значений:

a[0:2, 3:6] # из матрицы a будет выбрана подматрица с 1 по 3 строку и с 4 по 7 столбец

Используя полученные знания, выберите первый столбец из данных файла в переменную X, а второй столбец в переменную y.

## Функция стоимости (ошибка регрессии)

Для проведения градиентного спуска (алгоритма определения оптимальных коэффициентов, описывающих линию регрессии) нужно определить функцию стоимости (оценки), в качестве неё будем использовать квадратичную ошибку, формула 1:

(1)

где гипотеза получена из линейной модели, формула 2:

(2)

Формулу выше можно переписать следующим образом (формула 3):

(3)

Где вводят искусственно, и этот параметр всегда равен единице. Этот приём используют для того, чтобы можно было вычислить значение гипотезы в векторной форме (через перемножение векторов). Этот приём называется векторизацией вычислений и даёт помимо уменьшения кода по вычислению ещё и значительное увеличение быстродействия расчётов, так как во многих языках программирования (и в библиотеке NumPy в том числе) векторные и матричные операции хорошо оптимизированы и выполняются быстрее, чем написанные вручную циклы.

Для понимания процедуры векторизации рассмотрим пример. Допустим, требуется вычислить значение гипотезы для коэффициентов 1 и 2 и для значения равного 3. По формуле 2 вычислим значение :1+2\*3=7.

Теперь выполним то же самое, но с векторизацией вычислений. Опишем коэффициенты в виде вектора-столбца:

Значение можно тоже записать в виде вектора-столбца. Для того, чтобы можно было применять векторизацию вычислений, к добавим значение . Тогда вектор будет выглядеть так:

Тогда значение гипотезы может быть вычислено по формуле 2 в векторной форме:

Как видим, в векторной форме мы получили тот же результат, что и при поэлементном перемножении. Важно помнить, что для векторизации вычислений необходимо добавлять единичное значение .

В библиотеке NumPy для транспонирования матрицы или вектора используется метод transpose() (или краткий вариант записи функции: T):

>>> theta = np.matrix([1, 2])

>>> theta

matrix([[1, 2]])

>>> theta.**transpose()**

matrix([[1],

[2]])

>>> theta.**T**

matrix([[1],

[2]])

Следует добавить ещё одно важное замечание. Мы считали значение гипотезы для отдельного значения , которое представляло собой вектор-столбец. Но в нашей задаче существует целый набор значений . И к этому столбцу значений нужно присоединить в начале столбец единиц по размеру совпадающий с размером . Это можно сделать с помощью следующего кода Python:

m = X.shape[0] # количество элементов в X (количество городов)

X\_ones = np.c\_[np.ones((m, 1)), X] # добавляем единичный столбец к X

theta = np.matrix('[1; 2]') # коэффициенты theta представляют собой вектор-столбец из 2 элементов

# альтернативные способы создания вектора-столбца theta:

# theta = np.matrix([1, 2]).reshape(2, 1)

# theta = np.matrix([[1], [2]])

# theta = np.matrix([1, 2]).transpose()

h\_x = X\_ones \* theta # так можно вычислить значение гипотезы для всех городов сразу (подумайте почему и поэкспериментируйте с этим кодом в консоли Python).

Обратите внимание на 3 строку приведённого выше кода. Элементы вектора theta задаются в виде строки в одинарных кавычках. Такая запись позволяет создавать матрицы, используя более компактный синтаксис языка MatLab.

**Задание 3.** Реализуете функцию computeCost для вычисления для заданных коэффициентов и входных данных *X* и *y*.

Проверьте её работу, инициализировав нулевыми значениями.

Объявление функции Python начинается с ключевого слова def. Далее идёт имя функции и в скобках список её параметров. После закрывающей скобки ставится двоеточие.

Вот как должен выглядеть прототип создаваемой вами функции для вычисления ошибки:

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

# здесь будет размещаться ваш код вычисления ошибки cost

return cost

Функция принимает 3 параметра:

X – матрица, содержащая m строк и 2 столбца. m соответствует числу примеров (городов, для которых известна прибыль). Первый столбец в X состоит из единиц. Второй столбец соответствует численности каждого города. Единичный столбец в данном случае необходим для того, чтобы можно было использовать векторизацию вычислений.

y – прибыльность каждого города. y представляет собой вектор-столбец, состоящую из m элементов (по числу городов, для которых известна прибыль). y содержит значения прибыли в каждом городе.

theta – вектор-столбец коэффициентов . Всего в theta 2 элемента, которые полностью определяют форму регрессионной прямой. Первый отвечает за смещение прямой относительно начала координат, а второй за наклон прямой (по сути, второй коэффициент theta – это тангенс угла наклона прямой к оси абсцисс).

После того, как в функции compute\_cost вы реализуете вычисление гипотезы h\_x для всех городов X в векторной форме, необходимо будет вычислить значение стоимости по формуле 1. Для этого из каждого значения h\_x нужно будет вычесть соответствующее значение y, разности возвести в квадрат, просуммировать и поделить на удвоенное число примеров (городов). Например, возведение в квадрат разностей двух векторов может выглядеть так:

np.power(h\_x - y, 2)

Для суммирования элементов в векторе используется функция sum из библиотеки NumPy. Для определения количества примеров в матрице X используйте свойство shape.

Используя данную справочную информацию, реализуйте функцию compute\_cost, которая будет вычислять ошибку между реальными данными y и предсказанными с помощью гипотезы h\_x значениями по формуле 1.

Для проверки правильности реализованной вами функции вызовите её с параметрами X\_ones (это столбец X, содержащий значения численности городов, к которому добавлен единичный столбец), y (прибыльность каждого города) и коэффициентами theta = [1, 2]

Если функция реализована правильно, вы должны получить значение 75.203.

## Градиентный спуск

Необходимо подобрать такой набор коэффициентов , чтобы значение было минимальным. Один из способов сделать это - использование алгоритма градиентного спуска. Каждая итерация алгоритма обновляет параметры модели, формула 4:

(4)

где — скорость обучения (0 < < 1).

С каждым шагом градиентного спуска коэффициенты приближаются к оптимальным значениям, которые позволят обеспечить наименьшую стоимость .

При выполнении градиентного спуска нужно контролировать сходимость путем вычисления общей стоимости.

Далее необходимо реализовать функцию градиентного спуска gradient\_descent.

Помните, что стоимость параметризуется вектором , а не *X* и *у*. То есть, мы минимизируем значение , изменяя значения вектора , не изменяя *X* или *y*. Хороший способ проверить, что градиентный спуск работает правильно, это посмотреть на значение и убедиться, что оно уменьшается с каждым шагом, Значение никогда не должны расти, и должны сходиться к постоянному значению к концу алгоритма.

В нормальном случае значения сходятся как показано на рисунке:

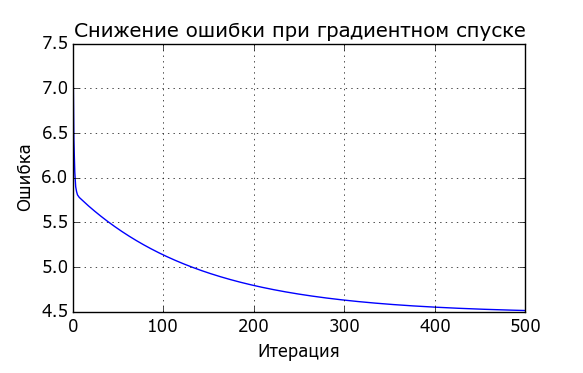


Рисунок 2 Снижение уровня ошибки с каждой итерацией

В общем виде алгоритм градиентного спуска выглядит так:

1. инициализировать = (0 , 1 , … 1)
2. вычислить значение для всех входов
3. вычислить значение функции стоимости
4. если значение функции стоимости изменилось меньше нужного порога, то градиентный спуск завершен, перейти на п. 7, иначе перейти на следующий пункт
5. изменить значения по формуле 3.
6. перейти на п. 2
7. конец работы

Для того чтобы избежать зацикливания, нужно сделать ограничение на максимальное количество итераций.

Если растёт, то нужно подобрать другую скорость обучения и попробовать другие начальные . Чем больше , тем быстрее происходит сходимость, но можно пропустить оптимальную точку.

**Задание 4.** Реализуйте описанную выше функцию градиентного спуска gradient\_descent. Входные параметры — *X*, *Y*, скорость обучения (например, = 0.01), количество итераций. Не забудьте добавить к X единичный столбец ( — свободный коэффициент).

Функция gradient\_descent должна иметь следующий вид:

def gradient\_descent(X, y, alpha, iterations):

# определение m – количество городов в матрице X (матрица X должна содержать единичный столбец). Используйте свойство X.shape для определения m и n.

# определение n – количество столбцов в матрице X (в нашей задаче n = 2)

# создать вектор-столбец theta и инициализировать его первый элемент нулем, а остальные элементы единицами. Количество элементов в theta равно n.

# создать вектор-столбец J\_theta, размер которого равен количеству итераций, заполненный нулями

# запустить цикл по итерациям: for i in range(iterations):

# вычислить значение ошибки J\_theta[i] для текущих значений theta

# модифицировать коэффициенты theta по формуле 4 (не забудьте про использование временной переменной temp\_theta)

**Важное замечание по модификации коэффициентов theta**. В формуле 4 коэффициенты меняются, но в то же самое время они и используются при вычислении гипотезы . Чтобы избежать этой ошибочной ситуации, исходные коэффициенты theta копируют во временную переменную temp\_theta, которая будет меняться на каждой итерации цикла по формуле 4. Исходные коэффициенты theta при этом не изменяются. После того, как все коэффициенты temp\_theta модифицированы, их значения присваивают исходным коэффициентам theta.

Запустите написанную функцию gradient\_descent, передав в неё параметры X\_ones, y, alpha=0.02 и количество итераций 500 и проверьте, какие значения theta возвращает функция. Если всё реализовано верно, должны получиться значения около -3.277, 1.131.

С помощью функции plot нарисуйте график уменьшения ошибки J\_theta с увеличением итераций. Программно подпишите график и оси. График должен выглядеть примерно как на рисунке выше.

**Задание 5.** Сделайте предсказание для двух неизвестных ранее городов, используя полученные выше коэффициенты .

**Задание 6.** Добавьте к изображению из задания 2 полученную линию регрессии (для рисования на существующем графике дважды вызывайте функцию plot). Для рисования линии регрессии используйте следующий код:

x = np.arange(min(X), max(X))

plt.plot(x, theta[1]\*x + theta[0], 'g--')

Не забудьте программно подписать график и оси и отобразить сетку.

Вот как должен выглядеть итоговый график:

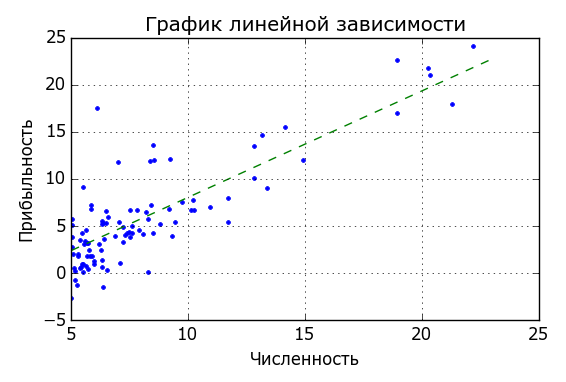


Рисунок 3 Итоговый график зависимости прибыли от числа жителей

# 2. Многомерная линейная регрессия

В этой части, вы реализуете линейную регрессию с несколькими переменными для прогнозирования цены дома. Предположим, вы продаете свой дом, и вы хотите знать, какая у него рыночная цена. Один из способов сделать это – сначала собрать информацию о последних проданных домах и построить модель цен.

Файл ex1data2.txt содержит обучающий набор цен на жилье. Первый столбец — площадь в футах, второй столбец — количество комнат, и третий столбец цена дома.

Обратите внимание, что размеры дома в абсолютных значениях на несколько порядков больше, чем количество комнат. Нормализация может ускорить градиентный спуск.

**Задание 7.** Загрузите входные данные. Для каждого входного столбца проведите нормализацию: вычислите среднее значение и вычтите его из набора данных. Посчитайте стандартное отклонение и разделите на него каждый элемент.

**Задание 8.** Постройте модель многомерной линейной регрессии. Сделайте прогноз для двух неизвестных ранее квартир. Можно использовать ваши предыдущие реализации функций compute\_cost и gradient\_descent, если они работают с произвольной размерностью. Иначе сделайте их универсальными.

# 3. Метод наименьших квадратов

Согласно методу наименьших квадратов искомые коэффициенты могут быть найдены из формулы 4:

Используемая формула не требует каких-либо нормализаций данных и многократных итераций. Однако в некоторых случаях решение не может быть получено, так как *XTX* может оказаться необратимой (вырожденной) матрицей (это может получиться если входные столбцы линейно зависимы).

**Задание 9.** Постройте модель многомерной линейной регрессии с использованием МНК. Сделайте прогноз для двух неизвестных ранее квартир.

При реализации не забудьте добавить к X первый единичный столбец. Для транспонирования матрицы используйте её метод transpose() (или краткий вариант: T). Для вычисления обратной матрицы используйте функцию np.linalg.pinv библиотеки NumPy.

**Задание 10.** Сравните модели, полученные методом МНК и методом градиентного спуска.