Задание № Б-1.1. Линейная регрессия

1. Введение

В этом упражнении вы реализуете линейную регрессию и увидите, как она работает с данными. Настоятельно рекомендуем вам использовать принципы векторизации везде, где это только возможно. В идеале у вас должен получиться код, состоящий исключительно из алгебраических матричных выражений – без циклов и даже без вызовов функции суммирования sum.

Описание файлов задания:

- ex1.m – главный скрипт для выполнения упражнения на однофакторную линейную регрессию,
- ex1_multi.m - главный скрипт для выполнения упражнения на многофакторную линейную регрессию,
- ex1data1.txt - датасет для линейной регрессии с одной переменной,
- ex1data2.txt - датасет для линейной регрессии с несколькими переменными,
- plotData.m - функция для визуализации датасета,
- computeCost.m - функция стоимости,
- gradientDescent.m - функция градиентного спуска,
- computeCostMulti.m - функция потерь для многофакторной линейной регрессии,
- gradientDescentMulti.m - градиентный спуск для многофакторной линейной регрессии,
- featureNormalize.m - функция для нормализации признаков,
- normalEqn.m - функция для решения нормального уравнения.

На протяжении всего упражнения вы будете запускать только скрипты ex1.m и ex1_multi.m. Они подготавливают датасеты для разных подзадач и выполняют вызовы функций, которые вы будете реализовывать. Файл ex1.m вам изменять не нужно. От вас требуется только реализовывать код функций в других файлах, следуя инструкциям в этом задании. Файл ex1_multi.m содержит фрагменты, которые вам нужно будет отредактировать. Что и где конкретно – будет сказано в задании.

2. Линейная регрессия с одной переменной

В этой части упражнения вы реализуете линейную регрессию с одной переменной (однофакторную линейную регрессию) для прогнозирования прибыли от потенциального расширения своего бизнеса. Предположим, вы являетесь генеральным директором ресторанной франшизы и анализируете, в каких еще городах можно открыть новые точки. У сети уже есть представительства в разных городах, и у вас есть данные о прибыльности и населении этих городов.

Вы хотели бы использовать эти данные, чтобы проанализировать, в каком городе открыть точку в следующий раз.

Файл ex1data1.txt содержит набор данных для нашей задачи линейной регрессии. Первая колонка - население города, вторая колонка - прибыльность в этом городе. Отрицательное значение указывает на убыточность бизнеса в этом городе.

Скрипт ex1.m загрузит эти данные для вас.

2.1 Визуализация данных

Прежде чем приступить к выполнению какой-либо задачи, часто бывает полезно понять данные, визуализировав их. Для датасета из данного упражнения вы можете использовать точечную диаграмму, поскольку у нас есть только два признака для отображения (прибыль и население).

Многие другие проблемы, с которыми вы столкнетесь в реальной жизни, являются многомерными и, очевидно, не могут быть визуализированы на двумерном и даже трехмерном графике.

В скрипте ex1.m обучающая выборка загружается из файла данных в переменные X и y:

```
data = load('ex1data1.txt'); % считывает данные, разделенные запятыми
X = data(:, 1); y = data(:, 2);
m = length(y); % размер обучающей выборки
```

Затем скрипт вызывает функцию plotData для отрисовки точечной диаграммы. Ваша задача – дописать соответствующий код в файле plotData.m. Для этого добавьте в него следующие строки:

```
plot(x, y, 'rx', 'MarkerSize', 10); % нарисовать график
ylabel('Profit in $10,000s'); % задать название оси Oy
xlabel('Population of City in 10,000s'); % задать название оси Ox
```

Теперь, когда вы будете запускать ex1.m, ваш итоговый результат должен выглядеть как на рисунке 1, с теми же красными маркерами x и метками осей.

Чтобы узнать больше о команде plot, вы можете ввести help plot в командной строке Octave/MATLAB или выполнить поиск соответствующей документации в Интернете. Чтобы изменить маркеры на красный X, мы использовали опцию 'rx' вместе с командой plot, т.е.

```
plot(..,[ваши параметры],.., 'rx');
```

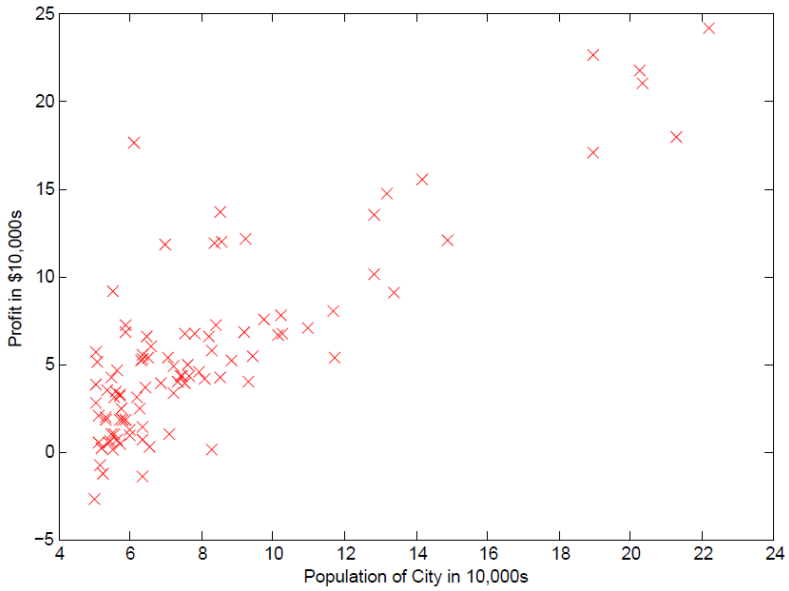


Рисунок 1: Точечный график обучающей выборки

2.2 Градиентный спуск

В этой части задания вы будете с помощью градиентного спуска подбирать параметры θ гипотезы $h_\theta(x)$, чтобы она наилучшим образом «объясняла» наш датасет, то есть иными словами – реализуете **машинное обучение** вашей модели.

2.2.1 Формулы итерационного алгоритма

Целью линейной регрессии является минимизация функции потерь (при таком выборе функции $J(\theta)$ этот метод еще называется методом наименьших квадратов – Least Squares):

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2,$$

где гипотеза $h_\theta(x)$ является линейной функцией:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1. \quad (1)$$

Напомним, что параметрами вашей модели являются значения θ_j. Именно их, **а не x_1**, вы будете подбирать, чтобы минимизировать значение функции потерь $J(\theta)$. Один из способов сделать это - использовать алгоритм пакетного градиентного спуска (batch gradient descent). При пакетном градиентном спуске на каждой итерации мы выполняем обновление параметров:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)},$$

при этом θ_j для всех j на каждом шаге обновляются синхронно, т.е. **одновременно**. Пакетным этот метод называется из-за того, что на каждом шаге мы пробегаемся по всему датасету от 1-го до m-го примера.

С каждым шагом градиентного спуска ваши параметры θ_j приближаются к оптимальным значениям, которые позволят достичь минимума функции $J(\theta)$.

Замечание. *Каждый обучающий пример $x^{(i)}$ мы храним в виде строки матрицы X. Для того чтобы можно было записать матричное уравнение (1) так, чтобы оно корректно работало со значением свободного параметра θ_0, добавим в матрицу X в самое начало столбец, состоящий из одних единиц. Это позволит нам рассматривать θ_0 просто как еще один признак.*

2.2.2 Реализация

В ex1.m уже содержится код для подготовки данных для линейной регрессии. Следующие строки добавляют еще одно измерение к нашему датасету, чтобы корректно работать с параметром θ_0. Мы также инициализируем θ_0 и θ_1 нулевыми значениями, а скорость обучения α = 0,01. Обратите внимание, что θ – это вектор-столбец!

```
X = [ones(m, 1), data(:,1)]; % добавляем столбец из единиц к матрице X
theta = zeros(2, 1); % инициализируем параметры модели
iterations = 1500;
alpha = 0.01;
```

2.2.3 Вычисление функции потерь $J(\theta)$

Когда вы выполняете градиентный спуск, минимизирующий функцию $J(\theta)$, полезно отслеживать ход работы, чтобы понимать, сходится ли алгоритм или нет. В этом разделе вы реализуете функцию для вычисления $J(\theta)$, чтобы ее можно было вызывать во время работы градиентного спуска и проверять тем самым, все ли в порядке.

Ваша следующая задача – написать в файле computeCost.m код, который вычисляет функцию $J(\theta)$. При этом помните, что переменные X и y являются не скалярными значениями, а матрицами размера $m \times n$ и $m \times 1$, соответственно, строки которых представляют примеры из датасета.

Как только вы завершите эту функцию, запустите снова ex1.m – скрипт вызовет computeCost один раз с вектором θ, инициализированным нулями, а затем со значениями [-1; 2], и выведет результаты на экран.

Корректные значения будут также выведены на экран с пометкой «Ожидаемая стоимость (примерно)». Сравните их с вашими.

2.2.4 Градиентный спуск

Далее вам необходимо написать код градиентного спуска в файле gradientDescent.m. Структура цикла в нем уже написана, и вам нужно только реализовать обновления θ на каждой итерации.

Убедитесь, что вы понимаете, что вы пытаетесь оптимизировать и что обновляете. Имейте в виду, что стоимость $J(\theta)$ параметризуется вектором θ, а не X и y. То есть мы минимизируем значение $J(\theta)$, **изменяя значения вектора $\boldsymbol{\theta}$**, а не X или y. Обратитесь к разделу 2.2.1, если не уверены в том, что нужно сделать.

Хороший способ убедиться, что градиентный спуск работает правильно – это посмотреть на значения $J(\theta)$ и проверить, уменьшаются ли они с каждым шагом цикла. Исходный код для gradientDescent.m вызывает computeCost на каждой итерации и выводит полученное значение. Если вы правильно все сделали, то получаемые вами значения $J(\theta)$ никогда не должны увеличиваться и должны сходиться к некоторому постоянному значению.

После того, как вы закончите, ex1.m использует ваши окончательные значения параметров θ для построения графика гипотезы. Результат должен выглядеть примерно так, как на рисунке 2.

Сравните полученные вами значения θ с корректными («Ожидаемое значение theta (примерно)»).

Ваши окончательные значения θ будут использоваться для прогнозирования прибыли в городах с населением 35 000 и 70 000 человек. У вас должно получиться примерно 4519.77 и 45342.45, соответственно. Обратите внимание на то, как в следующих строках из ex1.m для вычисления прогнозов используется умножение матриц, а не явное суммирование или циклы. Это пример векторизации кода в Octave/ MATLAB:

```
predict1 = [1, 3.5] * theta;
predict2 = [1, 7] * theta;
```

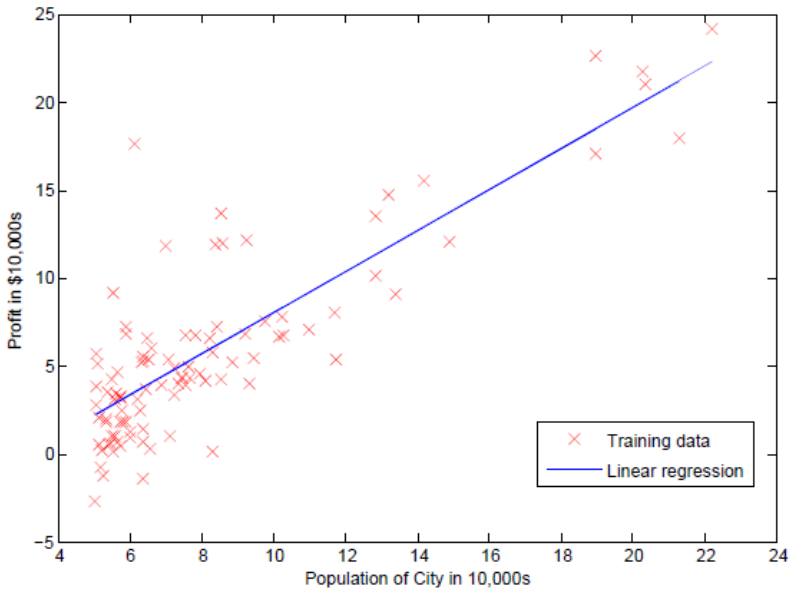


Рисунок 2: Датасет и график полученной в результате линейной регрессии гипотезы

2.3 Отладка

Вот некоторые вещи, которые следует иметь в виду при реализации градиентного спуска:

- Индексы массивов в Octave/MATLAB начинаются с единицы, а не с нуля. Если вы сохраняете θ_0 и θ_1 в векторе θ, то доступ к ним будет осуществляться как к $\theta(1)$ и $\theta(2)$.

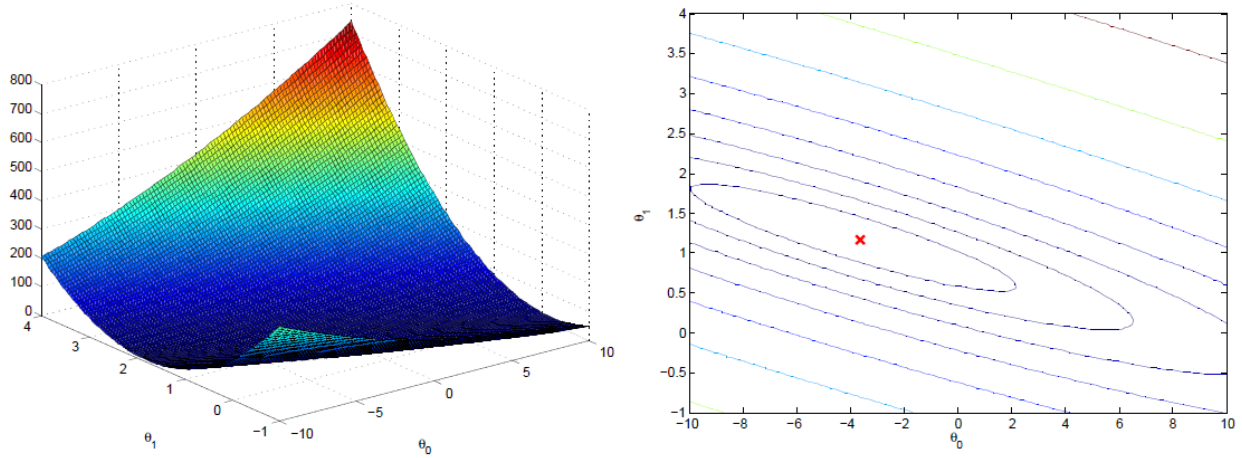
- Если вы видите много ошибок во время запуска, проверьте свои матричные операции, чтобы убедиться, что вы складываете и умножаете матрицы совместимых размеров. Печать размеров переменных с помощью команды size поможет вам в отладке.
- По умолчанию Octave/MATLAB интерпретирует математические операторы как матричные. Это распространенный источник ошибок несовместимости размеров. Если вы не хотите матричного умножения, вам нужно добавить обозначение "точка", чтобы явным образом указать это. Например, A * B выполняет матричное умножение, в то время как A .* B выполняет поэлементное умножение. В частности, обратите внимание, что A^2, это не поэлементное возведение элементов матрицы A в квадрат, а матричное умножение $A * A$. Если вам нужна поэлементная операция, то пишите A .^ 2.

2.4 Визуализация $J(\theta)$

Для лучшего понимания функции затрат $J(\theta)$ будет построен ее график по двумерной сетке значений θ_0 и θ_1. В этой части упражнения вам не нужно ничего кодировать, но вы должны понимать, как работает этот код.

```
% инициализация J_vals матрицей из нулей
J_vals = zeros(length(theta0_vals), length(theta1_vals));
% заполнение J_vals
for i = 1:length(theta0_vals)
      for j = 1:length(theta1_vals)
            t = [theta0_vals(i); theta1_vals(j)];
            J_vals(i,j) = computeCost(x, y, t);
      end
end
```

После выполнения этих строк у вас будет двумерный массив значений $J(\theta)$. Скрипт ex1.m затем использует эти значения для отрисовки графиков поверхности и контура $J(\theta)$ с помощью команд surf и contour, соответственно. Графики должны выглядеть примерно так, как на рисунке 3:



(a) Surface (b) Contour, showing minimum

Рисунок 3: Функция потерь $J(\theta)$

Их цель – показать вам, как $J(\theta)$ меняется в зависимости от θ_0 и θ_1. Функция потерь $J(\theta)$ имеет форму чаши и имеет, соответственно, глобальный минимум. (Это легче увидеть на контурном графике.) Этот минимум является оптимальным значением для θ_0 и θ_1, и каждый шаг градиентного спуска приближается к этой точке.

3. Линейная регрессия с несколькими переменными

В этой части упражнения вы реализуете линейную регрессию с несколькими переменными для прогнозирования цен на недвижимость. Предположим, вы продаете свой дом и хотите узнать оценку его рыночной стоимости. Один из способов сделать это – сначала собрать информацию о недавно проданных домах и затем составить модель цен на жилье.

Файл ex1data2.txt содержит обучающую выборку с ценами на жилье в некотором городе. В первом столбце указан размер дома (в квадратных метрах), во втором столбце указано количество спален, а в третьем столбце указана цена дома.

Скрипт ex1_multi.m поможет вам выполнить это упражнение.

3.1 Нормализация признаков

Скрипт ex1_multi.m начинается с загрузки и вывода некоторых значений из этого набора данных. Взглянув на значения, обратите внимание, что значения размеров домов примерно в 1000 раз превышают значения для количества спален. Когда признаки различаются на несколько порядков, предварительное их масштабирование может значительно ускорить сходимость градиентного спуска.

Ваша первая задача – завершить код в featureNormalize.m:

- вычтите из каждого признака его среднее значение,
- после этого масштабируйте (разделите) значения признаков на их соответствующие стандартные отклонения.

Стандартное отклонение – это способ измерения степени отклонения значений конкретного признака от своего среднего (большинство точек данных будут находиться в пределах ± 2 стандартных отклонения). Это альтернатива использованию диапазона значений (max-min). В Octave / MATLAB вы можете использовать функцию "std" для вычисления стандартного отклонения. Например, величина X(:,1) содержит все значения признака x_1 (размеры домов) в датасете, поэтому std(X(:,1)) вычисляет стандартное отклонение для размеров домов. В момент вызова featureNormalize.m дополнительный столбец 1, соответствующий $x_0 = 1$, еще не был добавлен в X.

Если вы сделаете это корректно, то ваш код будет работать с обучающими выборками любых размеров (любое количество признаков / примеров). Обратите внимание, что каждый столбец матрицы X соответствует одному признаку.

Замечание. *При нормализации признаков важно сохранить использованные значения средних и стандартных отклонений. После того как вы построите модель гипотезы и начнете использовать ее для произвольных входных значений, то предварительно вам нужно также будет нормализовать эти значения с помощью данных параметров, иначе результаты прогнозирования окажутся некорректными.*

3.2 Градиентный спуск

Ранее вы реализовали градиентный спуск для задачи одномерной регрессии. Единственная разница сейчас заключается в том, что в матрице X теперь есть еще один признак. Функция гипотезы и формула для градиентного спуска остаются неизменными.

Дополните код в computeCostMulti.m и gradientDescentMulti.m, чтобы он вычислял функции потерь и градиентного спуска для линейной регрессии с несколькими переменными.

Убедитесь, что ваш код поддерживает любое количество признаков и хорошо векторизован.

Подсказка. *В частности, если вы все правильно реализуете, то насколько сильно будет отличаться код для однофакторной регрессии от кода для многофакторной регрессии?*

3.2.1 Выбор скорости обучения

В этой части упражнения вы сможете опробовать различные скорости обучения и найти ту, при которой градиентный спуск сходится быстрее всего. Здесь вы должны внести соответствующие изменения в код ex1_multi.m.

Если вы выбрали скорость обучения в пределах хорошего диапазона, ваш график будет выглядеть примерно так же, как на рисунке 4. Если ваш график выглядит совсем по-другому, в частности, если значение $J(\theta)$ увеличиваются, то скорее всего скорость слишком большая – отрегулируйте ее и повторите попытку.

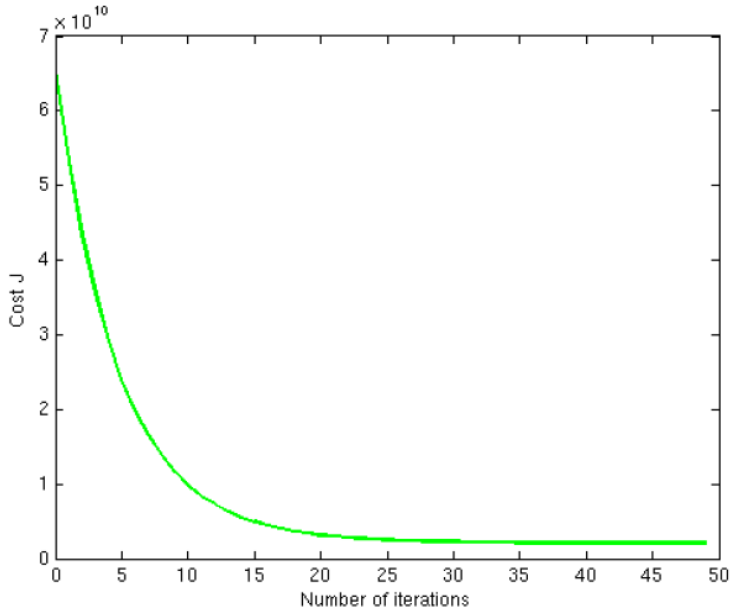


Рисунок 4: Сходимость градиентного спуска с адекватной скоростью обучения

Замечание. *Если ваша скорость обучения слишком велика,* $J(\theta)$ *может расходиться и даже "взрываться", что приводит к значениям, которые слишком велики для компьютерных вычислений. В этих ситуациях Octave / MATLAB будет возвращать NaN. NaN означает «не число» и часто является результатом неопределенных операций, использующих* $-\infty$ *и* ∞.

Подсказка. *Чтобы сравнить, как разные скорости обучения влияют на сходимость, полезно построить график* $J(\theta)$ *для нескольких скоростей обучения на одном и том же рисунке. В Octave / MATLAB это можно сделать, выполнив градиентный спуск несколько раз и с помощью команды hold – «удерживать» отрисованные графики на одном холсте. Если более конкретно, то если вы перебрали три разных значения* α *и сохранили значения функции потерь в J1, J2 и J3, вы можете использовать следующие команды для их визуализации на одном и том же рисунке:*

```
plot(1:50, J1(1:50), 'b');
hold on;
plot(1:50, J2(1:50), 'r');
plot(1:50, J3(1:50), 'k');
```

Аргументы `'b'`, `'r'` *и* `'k'` *определяют разные цвета для графиков.*

Изучите отличия в кривых сходимости в зависимости от скорости обучения. При небольшой скорости вы должны обнаружить, что для достижения оптимального значения градиентного спуска

требуется очень много времени. И наоборот, при большой скорости обучения градиентный спуск может не сходиться или даже расходиться!

Используя наилучшую найденную вами скорость обучения, запустите скрипт ex1_multi.m, чтобы найти конечные значения θ. Затем используйте это значение θ, чтобы спрогнозировать цену дома площадью 1650 квадратных метров с 3 спальнями. Это значение вы будете использовать позже, чтобы проверить свою реализацию нормальных уравнений. Не забудьте нормализовать значения признаков, когда будете делать этот прогноз!

У вас должно получиться значение $\theta = (340412.66,\ 110631.05,\ -6649.47)$ и прогноз $293 081.46.

3.3 Нормальное уравнение

Из теории вы знаете, что решение задачи линейной регрессии в аналитической форме есть

$$\theta = (X^T X)^{-1} X^T \bar{y}.$$

Использование этой формулы не требует нормализации признаков, и вы получите точное решение всего за один шаг: нет необходимости запуска циклического алгоритма и ждать, пока он сойдется, как при градиентном спуске.

Допишите код в normalEqn.m, который, используя вышеуказанную формулу, рассчитает θ. Помните, что, хотя вам и не нужно масштабировать свои признаки, но вам все равно нужно добавить столбец из единиц в матрицу X.

Вычислив с помощью метода нормального уравнения значение θ для вашего датасета, используйте его теперь для прогнозирования цены на дом площадью 1650 квадратных метров с 3 спальнями. У вас должен получиться тот же самый результат, что и в случае градиентного спуска из раздела 3.2.1. При этом обратите внимание, что вектор $\theta = (89597.91,\ 139.21,\ -8738.02)$ у вас будет другой – как вы можете это объяснить?