Санкт-Петербургский государственный университет
Прикладная математика и информатика

Отчет по учебной практике 2 (научно-исследовательской работе) (семестр 4)
Линейная регрессия для больших данных

Выполнил: Погребников Николай Вадимович, группа 21.Б05-мм

Научные руководители: к. ф.-м. наук, доцент Голяндина Н.Э.

Зотиков Д.Ю.

Санкт-Петербург
2023

1 Введение

В современном мире, где технологии проникают во все сферы жизни, машинное обучение становится неотъемлемой частью нашего повседневного опыта. Одним из методов машинного обучения является линейная регрессия, и именно задача её реализации была поставлена передо мной.

В ходе данной научно-исследовательской работы я разобрался в деталях данной модели, изучил работу градиентного спуска, написал код линейной регрессии на языке R, реализовав распараллеливание алгоритма и исследовал полученные результаты при разных параметрах модели.

2 Описание методов

2.1 Линейная регрессия

Линейная регрессия - это метод анализа данных, позволяющий устанавливать линейную зависимость между «признаками» («предикторами», «features») и «наблюдениями» («response», «outcome»). Пусть $\mathcal{D} = \{(\mathbf{x}^{(i)}, y_i)_{i=1}^n\}$ - множество данных, где $\mathbf{x}^{(i)} \in \mathrm{R}^d$ - признаки, а $y_i \in \mathrm{R}$ - наблюдения. Связующим звеном между $\mathbf{x}^{(i)}$ и y_i является модель линейной регрессии $y_i = f(\mathbf{x}^{(i)} \mid \boldsymbol{\beta}) + \epsilon_i$. Здесь $i \in 1:n$, $\epsilon_i \sim \mathrm{N}(0, \sigma^2)$, $f(\mathbf{x}^{(i)}|\boldsymbol{\beta}) = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \ldots + \beta_d x_d^{(i)}$, $\boldsymbol{\beta} \in \mathrm{R}^{d+1}$. Таким образом, наша модель является линейной комбинацией признаков, при этом шум имеет нормальное распределение.

Задача состоит в том, чтобы найти подходящий $\hat{\boldsymbol{\beta}}$ максимально близкий к $\boldsymbol{\beta}$. Более подробную информацию про линейную регрессию можно прочитать данной ссылке.

Существует несколько способов нахождения такого вектора-коэфицентов, однако в этой работе рассмотрены лишь два.

2.2 Метод наименьших квадратов

В курсе статистического обучения показывается, что нахождение оптимальной оценки $\hat{\boldsymbol{\beta}}$ эквивалентно нахождению такого $\hat{\boldsymbol{\beta}}$, что расхождения между $f\left(\mathbf{x}^{(i)} \mid \hat{\boldsymbol{\beta}}\right)$ и y_i были бы минимально возможные для всех $i \in 1:n$. В частности, величина

$$\epsilon_i = f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\beta}\right) - y_i, \quad i \in 1:n$$

называется остаток; сумма квадратов таких остатков по всем данным («Residual Sum of Squares») есть

$$\mathrm{RSS}(\boldsymbol{\beta} \mid \mathcal{D}) = \sum_{i=1}^{n} \left(f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\beta}\right) - y_i\right)^2$$

Однако удобнее минимизировать функцию, называющуюся Mean Squared Error, MSE или среднеквадратическим отклонением:

$$\mathrm{MSE}(\boldsymbol{\beta} \mid \mathcal{D}) = \frac{1}{n}\mathrm{RSS}(\boldsymbol{\beta} \mid \mathcal{D})$$

Существуют и другие функции потерь, но в данной работе будет рассмотрена именно эта.

2.3 Аналитическое решение

В курсе статистики доказывается, что точка минимума $\hat{\beta}$ может быть получена аналитически:

$$\hat{\beta} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y}$$

Здесь $\mathbf{X}$ - матрица признаков, а $\mathbf{y}$ - вектор-столбец наблюдений. На момент написания работы у меня не было курса статистики, поэтому было найдено объяснение данного метода в хэндбуке от Яндекса:

Пусть $\mathbf{x^{(1)}}, ..., \mathbf{x^{(d+1)}}$ - столбцы матрицы $\mathbf{X}$.

$$\mathbf{X}\hat{\boldsymbol{\beta}} = \hat{\beta}_1\mathbf{x^{(1)}} + ... + \hat{\beta}_d\mathbf{x^{(d+1)}}$$

Тогда и наша задача сводится к следующей: найти линейную комбинацию столбцов $\mathbf{x^{(1)}}, ..., \mathbf{x^{(d+1)}}$, которая наилучшим способом приближает столбец y по евклидовой норме - то есть найти проекцию вектора y на подпространство, образованное векторами $\mathbf{x^{(1)}}, ..., \mathbf{x^{(d+1)}}$. Отсюда уже можно вывести исходную формулу.

Посчитаем сложность данного алгоритма:

- Сложность транспонирования матрицы $\mathbf{X}$ равна $O(nd)$;
- Сложность умножения матриц(в нашем случае самым тяжёлым умножением будет $\mathbf{X}^\intercal\mathbf{X}$) равна $O(nd^2)$;
- Сложность нахождения обратной матрицы для $\mathbf{X}^\intercal\mathbf{X}$ равна $O(d^3)$.

Итоговая сложность равна $O(nd^2 + d^3)$.

2.4 Градиентный спуск

Аналитическое решение хорошо использовать на маленьких данных, однако для больших матриц вычислительно тяжело искать обратные матрицы. Поэтому можно использовать алгоритм градиентного спуска, являющийся итеративной процедурой, на каждом шаге которого происходит движение к минимуму объектной функции по её градиенту:

1. Пусть на шаге t есть текущее приближение $\hat{\boldsymbol{\beta}}^{(t)}$
2. Вычисляется градиент $\nabla R\left(\hat{\boldsymbol{\beta}}^{(t)} \mid \mathcal{D}\right)$
3. Следующее лучшее приближение есть $\hat{\boldsymbol{\beta}}^{(t+1)} = \hat{\boldsymbol{\beta}}^{(t)} - \alpha\nabla R\left(\hat{\boldsymbol{\beta}}^{(t)} \mid \mathcal{D}\right)$.α-предзаданный параметр, по смыслу задающий «скорость оптимизации» («learning rate»).
4. Алгоритм завершается, когда выполняется некоторый критерий останова.

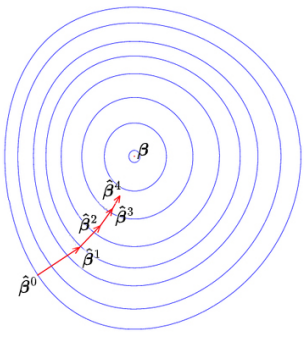

Рис. 1: Градиентный спуск

Поскольку MSE - выпуклая функция(доказательство данного факта), значит, минимум один, и при удачном выборе параметров мы всегда будем сходиться в нашем алгоритме.

2.4.1 Подсчёт вектора-градиента

Посчитаем элемент градиента функции:

$$R(\boldsymbol{\beta}|D) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x^{(i)}}|\boldsymbol{\beta}))^2 = \frac{1}{n}\text{RSS}(\boldsymbol{\beta}|D)$$

$$\frac{\partial R}{\partial \beta_j} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \ldots + \beta_d x_d^{(i)}) = \frac{-2}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x^{(i)}}|\boldsymbol{\beta}))\mathbf{x_j^{(i)}}$$

Рассмотрим данную формулу в векторном виде:
Пусть $\mathbf{X}$ - матрица признаков $N \times (d+1)$, $\boldsymbol{\beta}$ - вектор-столбец коэфицентов размерности $d+1$, $\boldsymbol{e} = y - f(\mathbf{x^{(i)}}|\boldsymbol{\beta})$ - вектор-столбец

$$\mathbf{X}^T = \begin{pmatrix} 1 & \ldots & 1 \\ x_{11} & \ldots & x_{1n} \\ \ldots & \ldots & \ldots \\ x_{d1} & \ldots & x_{dn} \end{pmatrix}, \epsilon = \begin{pmatrix} y_1 - f(\mathbf{x}^{(1)}|\boldsymbol{\beta}) \\ y_2 - f(\mathbf{x}^{(2)}|\boldsymbol{\beta}) \\ \ldots \\ y_n - f(\mathbf{x}^{(n)}|\boldsymbol{\beta}) \end{pmatrix}$$

2.4.2 Критерий останова

Процесс градиентного спуска можно продолжать сколь угодно долго, однако хочется выбрать условие, при котором стоит остановиться. Критерий останова определяет момент, при котором алгоритм градиентного спуска должен прекратить обновление параметров модели и завершиться. Правильный выбор критерия останова влияет на скорость сходимости алгоритма и точность полученных результатов.
В данной работе использовались два критерия останова:

1. Евклидова норма разности значений функции линейной регрессии оказалась меньше заранее заданного ϵ:

$$||f(\mathbf{x} \mid \hat{\boldsymbol{\beta}}^{(\boldsymbol{t+1})}) - f(\mathbf{x} \mid \hat{\boldsymbol{\beta}}^{(\boldsymbol{t})})||^2 < \epsilon$$

2. Модуль разности MSE оказался меньше заранее заданного ϵ:

$$|\text{MSE}(\hat{\boldsymbol{\beta}}^{(\boldsymbol{t+1})} \mid \mathcal{D}) - \text{MSE}(\hat{\boldsymbol{\beta}}^{(\boldsymbol{t})} \mid \mathcal{D})| < \epsilon$$

Поскольку с каждой итерацией алгоритм становится всё ближе и ближе к истинному значению $\boldsymbol{\beta}$, значения $||f(\mathbf{x} \mid \hat{\boldsymbol{\beta}}^{(\boldsymbol{t+1})}) - f(\mathbf{x} \mid \hat{\boldsymbol{\beta}}^{(\boldsymbol{t})})||^2$ и $|\text{MSE}(\hat{\boldsymbol{\beta}}^{(\boldsymbol{t+1})} \mid \mathcal{D}) - \text{MSE}(\hat{\boldsymbol{\beta}}^{(\boldsymbol{t})} \mid \mathcal{D})|$ становятся всё меньше и меньше. Помимо этого, выяснилось, что несколько проще распараллеливать алгоритм при выборе второго критерия, поэтому в конечном счёте было принято решение остановиться на нём.

Оба варианта работают в тандеме с дополнительным условием, ограничивающим количество итераций, чтобы, если уж и произошло зацикливание, то оно не было бы бесконечным. Кроме того, можно доказать, что потребуется $O(\frac{1}{\epsilon})$ итераций, чтобы достигнуть заранее заданной точности.

2.5 Распараллеливание алгоритма

Поскольку процедуры вычисления градиента и MSE вычислительно довольно сложны, хочется их распараллелить. Остаётся понять, каким образом?

MSE и градиент - это суммы. Разобьём данные суммы на m частей, в каждой k_i элементов, $i \in 1 : n$. Вычислим каждую из них на отдельных нодах, а затем сложим результаты. Рассмотрим на примере вычисления градиента:

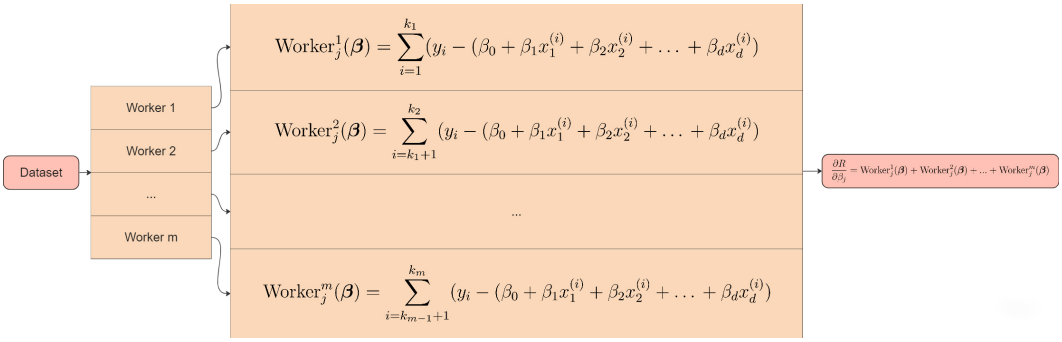

Рис. 2: Работа градиентного спуска

Таким же образом можно распараллелить вычисление функции MSE.

3 Описание вычислительного эксперимента и результаты

В ходе данной научно-исследовательской работы теоретическая часть была преобразована в практические функции с использованием языка R. Передо мной были поставлены 3 основных вычислительных экспериментов, выполнение каждого из которых приведено ниже.

3.1 Генерация данных и написание основных функций

Исходные данные $\mathcal{D}$ были преобразованы в вектора и матрицы, написаны вспомогательные функции:

1. Линейная регрессия f(x, b), где x - матрица признаков, b - текущий вектор $\boldsymbol{\beta}$.
2. Генератор матрицы X make_X(n, d, x_min, x_max), где n и d задают множество данных, x_min и x_max - минимальное и максимальное значения в матрице X.
3. MSE(x, b, y), берущая среднее значения квадрата разности значений f(x, b) и y, y - вектор наблюдений.

Алгоритм градиентного спуска был описан в теоретической части данной работы и реализован двумя способами: с запоминанием данных MSE для каждой итерации(gd) и без(gd_without_plot_data). Это обусловлено тем, что иногда хочется посмотреть на сходимость алгоритма и увидеть график своими глазами, однако эти данные также забивают память и замедляют работу. Эти функции принимают на вход следующие

параметры: b_init - начальное значение $\hat{\boldsymbol{\beta}}^{(1)}$, X - матрица признаков, y - вектор наблюдений, a - скорость обучения, epsilon - точность, n_iter_max - количество итераций после которых стоит остановиться(чтобы не было бесконечных циклов ни в каком из случаев).

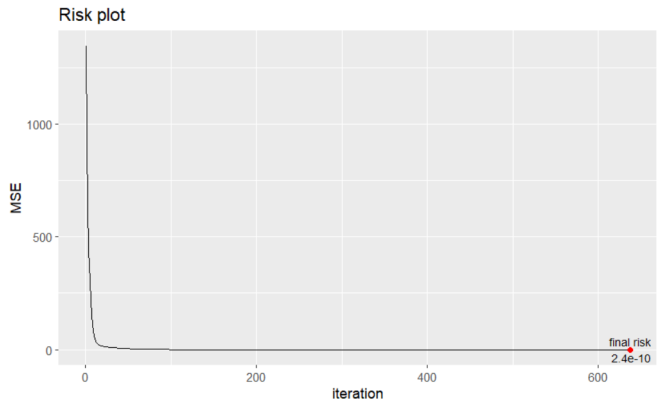

Рис. 3: Пример графика MSE в зависимости от итерации алгоритма
Начальные данные: b_init = 1:6, n = 1000, d = 5, x_min = -5, x_max = 5, $\epsilon_i \sim \mathrm{N}(0, 0)$, a = 1e-2, epsilon = 1e-10, n_iter_max = 1e6.

При этом, если данные зашумлены, то MSE не сможет опуститься ниже σ:

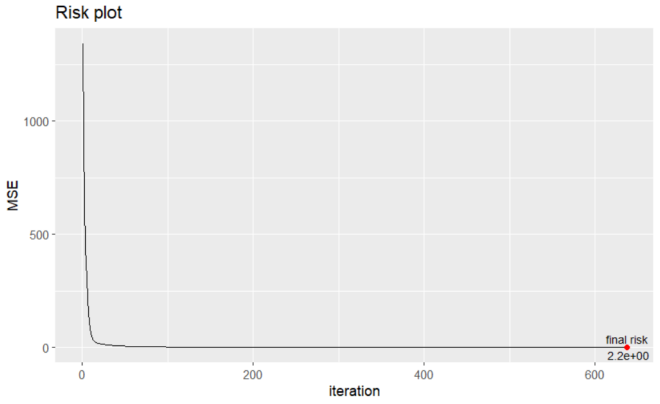

Рис. 4: Пример графика MSE в зависимости от итерации алгоритма
$\epsilon_i \sim \mathrm{N}(0, 2^2)$

Кроме того, были рассмотрены различные α для понимания, когда алгоритм сходится, а когда расходится для тех же данных, что использовались при исследовании графика MSE на рисунке 3. Изначально α(alpha) равнялся 1e-3 и постепенно увеличивался на 1e-3. В последствии эксперимента выяснилось, что при $\alpha >= 0.108$ сходимости уже нет.

3.2 Сравнение аналитического решения и решения с применением градиентного спуска

Мною была написана функция analitical_solution(x, y), вычисляющая вектор $\hat{\boldsymbol{\beta}}$ исключительно с помощью матричных операций. К сожалению, не удалось вывести ситуацию, при которой градиентный спуск оказался бы быстрее, в моей реализации, поскольку матричные операции очень хорошо оптимизированы на языке R, а итерации в R довольно медленные. Однако удалось выяснить, что на "небольших"данных выгоднее использовать именно аналитический метод.

При параметрах $n = 1e5, d = 100$ аналитическое решение оказалось быстрее в 4.1 раза, чем градиентный спуск. Если увеличивать данные(особенно параметр d) метод градиентного спуска окажется выгоднее.

3.3 Распараллеливание градиентного спуска

Теоретическая часть распараллеливания была описана ранее, остаётся понять, как всё это оформить на компьютерном языке. Для начала стоит определиться с количеством рабочих нод. В теории, наилучшим решением является сделать на одну меньше рабочую ноду, чем количество ядер в компьютере, оставив одно ядро процессора на работу системы.

Разобьём вычисления градиента следующим образом:

1. Для каждой ноды будет своя часть матрицы признаков $\mathbf{X}$ и своя часть наблюдений y. Данное разбиение примерно равномерное и конкретно на языке R это можно сделать через parallel::splitIndices.
2. Каждая нода на выходе даст свою часть вектора градиента, после чего всё это суммируется и получается обычный градиент.

Кроме того, чтобы не считать значение функции линейной регрегрессии на главной ноде, было принято решение заменить критерий остановки на абсолютную фактическую погрешность между значением MSE на предыдущей итерации и текущей. Таким образом, можно распараллелить и рассчёт MSE по тому же принципу.

При таком распараллеливании алгоритма приходится передавать в кластер каждую итерации только один единственный параметр - текущее значение коэфицентов линейной комбинации $\hat{\boldsymbol{\beta}}$. То есть $d + 1$ значение.

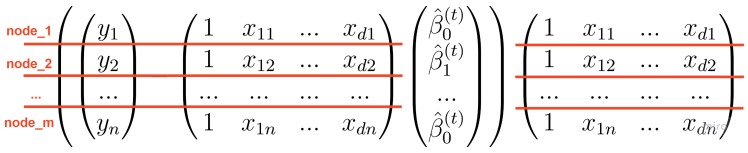

Рис. 5: Разбиение векторов и матриц

На рисунке 5 изображено разбиение вычисления градиента, после чего следует суммирование результатов и домножение их на $\frac{-2}{n}$. Таким образом получается вектор-градиент.

Мой компьютер имеет 8 ядер и при сравнении выполнения работы на одних и тех же довольно больших данных($n = 1e7, d = 20$), распараллеленный градиентный спуск отработал в 3.166 раз быстрее обычного. На небольших же данных ситуация оказывается обратной. Это происходит из-за того, что тратится время на копирование данных для каждой ноды.

Кроме того, был запущен алгоритм с вычислением оптимального количества нод(m). В результате вывелась таблица, с информацией о количестве нод и временем исполнения алгоритма. Для моего компьютера наилучшее значение $m = 10$.

4 Заключение

В ходе данной научно-исследовательской работы я разобрался с теоретической частью одного из методов машинного обучения, линейной регрессии, реализовал её с помощью языка R, при этом распределив вычисления на ядра процессора, рассмотрел зависимость конечных результатов от начальных данных.

5 Приложения

1. GitHub со всем кодом на языке R: https://github.com/statmod-courseworks/lr-gd-r-npogrebnikov/
2. Учебник по машинному обучению от Яндекса: https://academy.yandex.ru/handbook/ml/
3. Доказательство выпуклости функции MSE: https://math.stackexchange.com/questions/3669797/how-to-prove-a-mse-loss-function-is-convex-for-linear-regression/36698123669812
4. Доказательство оценки зависимости итераций от заранее заданной точности https://www.cs.ubc.ca/ schmi W18/L4.pdf