

Национальный исследовательский университет ИТМО  
Факультет информационных технологий и программирования  
Прикладная математика и информатика

# Методы оптимизации

Отчет по лабораторной работе №3

⟨Собрано 6 июня 2023 г.⟩

**Работу выполнили:**

Бактурин Савелий Филиппович М32331

Вереня Андрей Тарасович М32331

Сотников Максим Владимирович М32331

**Преподаватель:**

Ким Станислав Евгеньевич

# Решение задачи нелинейной регрессии

Часто решая задачу создания регрессионной модели мы сталкиваемся с тем, что по жизни очень немногие рассматриваемые функции оказываются не представимы в виде обобщенной линейной зависимости или полиномиальной некоторой конечной степени  $k$ . Такая же ситуация часто случается и с некоторым набором данных, который нужно как-то обобщить. Именно в таких случаях к нам на помощь приходит более частный случай регрессионного анализа – *нелинейная регрессия*.

Идея построения нелинейной регрессии как и в случае с полиномиальной заключается в том, чтобы найти математическую функцию, которая максимально точно описывает зависимость между независимой переменной и зависимой от нее. Например, для построения нелинейной регрессии можно использовать функции типа полинома, логарифмической или экспоненциальной зависимости.

В целом весь процесс нахождения нелинейной регрессионной модели можно поделить на два этапа:

- ▷ Определить регрессионную модель  $f(w, x)$ , которая зависит от параметров  $w = (w_1, \dots, w_W)$  и свободной переменной  $x$ .
- ▷ Решить задачу по нахождению минимума суммы квадратов регрессионных остатков:

$$S = \sum_{i=1}^m r_i^2, \quad r_i = y_i - f(w, x_i)$$

Однако, решая в лоб такую задачу, мы сталкиваемся с оптимизационной задачей нахождения параметров нелинейной регрессионной модели. Тут к нам и приходят на помощь различные методы нахождения, в том числе и рассматриваемые ниже: *Gauss-Newton* и *Powell Dog Leg*.

## Gauss-Newton

Напомним, что мы решаем следующую задачу: дана нелинейная модель  $f(w, x)$ , где  $w \in \mathbb{R}^m$ , тогда сумма квадратов регрессионных остатков высчитывается как

$$S = \sum_{i=1}^{\text{sizeof } X} (f(w, x_i) - y_i)^2 \rightarrow \min$$

Итак, пусть  $n = \text{sizeof } X$  и введем некоторые новые объекты для решения задачи, пусть  $w^0 = (w_0^0, w_1^0, \dots, w_m^0)$  – начальное приближение, и

$$\mathbf{J} = \left( \frac{\partial f}{\partial w_j}(w^i, x_i) \right)_{n \times m} \quad - \text{Якобиан, или матрица первых производных}$$

$$\vec{f}_i = (f(w^i, x_i))_{n \times 1} \quad - \text{вектор значений функции } f$$

$$\delta_i = \text{const} \quad - \text{размера шага}$$

Тогда, формула  $i$ -й итерации рассматриваемого метода будет высчитываться как

$$w^{i+1} \leftarrow w^i - \delta_i \cdot \underbrace{(\mathbf{J}_i^T \mathbf{J}_i)^{-1}}_{\beta} \mathbf{J}_i^T (\vec{f}_i - y),$$

где  $\beta$  – это псевдообратная матрица к матрице  $\mathbf{J}_i$ , или решение некоторой задачи многомерной линейной регрессии, где мы ищем такой вектор  $\beta$ , что

$$\left\| \mathbf{J}_i \beta - (\vec{f}_i - y) \right\|^2 \rightarrow \min,$$

где  $y$  – вектор правильных/настоящих ответов нашей модели. Получается, для решения задачи, мы, так называемую, *невязку* пытаемся приблизить линейной комбинацией вектора из матрицы Якобиана так, что при следующем шаге итерации получить такой  $w^{i+1}$ , который бы сократил нам расстояние невязки. Причем, заметим, что на каждом шаге, задача будет новой, так как  $\mathbf{J}_i$  зависит от текущего приближения, чтобы решить задачу многомерной регрессии.

Заметим, что здесь, по алгоритму, мы видим достаточно очевидное ограничение:  $m \geq n$ , в ином случае для  $\mathbf{J}_i^T \mathbf{J}_i$  не будет существовать обратной матрицы и, в следствии, решения к уравнению.

## Исследования

### Powell Dog Leg

*Trust-region method* — это метод решения оптимизационных задач, который основывается на вычислении региона, в котором квадратичная модель аппроксимирует целевую функцию. Сам этот метод представляет из себя смесь сразу двух алгоритмов, решающих задачу:

- ▷ Линейный поиск используется для определения направления поиска и дальнейшего нахождения оптимального шага вдоль выбранного вектора пути.
- ▷ Сам по себе trust-region используется для определения области вокруг текущей итерации, в которой модель достаточно аппроксимирует целевую функцию. Причем, стоит заметить, что для поиска следующего радиуса рассматриваемого региона также будет использоваться линейный поиск.

В общем случае Trust-region на каждой итерации решает следующую квадратичную задачу:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p,$$

где  $f_k = f(x_k)$ ,  $g_k = \nabla f_k$ ,  $B_k = \nabla^2 f_k$  и  $\nabla_k > 0$  – изменяющийся радиус региона, причем всё это, при условии, что  $|p| \leq \nabla_k$ . Заметим, что в таком простейшем виде мы получаем безусловно почти бесполезный алгоритм: он чрезвычайно медленный из-за появления  $B_k$  – Гессиана функции. С другой стороны, если он положительно определен и  $|B_k^{-1} \nabla f_k| \leq \nabla_k$ , то решение легко определить:  $p_k^B = -B_k^{-1} \nabla_k$ . Но, опять же, высчитывать еще и обратную матрицу – дело долгое и медленное, поэтому, начиная отсюда и до конца все лабораторной работы, мы будем то и дело пытаться приближать наши значения к реальным/по настоящему посчитанным значениям Гессиан-функции.

Здесь мы рассмотрим один из методов оптимизации при аппроксимации квадратичной модели – *Powell Dog Leg*. Начнем, пожалуй, с определения радиуса рассматриваемого доверительного региона: в алгоритме dogleg обычно выбирают основываясь на сходстве функции  $m_k$  (та, что мы решаем изначально) и оригинальной

функции  $f$  на предыдущей итерации. Зададим  $\rho_k$  следующим образом:

$$\rho_k = \frac{f_k - f_k^*}{m_k(0) - m_k(p_k)},$$

где  $f_k^* = f(x_k + p_k)$ . А теперь посмотрим на то, как именно лучше поменять шаг: в том случае, если  $\rho_k$  меньше нуля, то это значит, что наша модель далека от функции и нужно обязательно уменьшить радиус; в том случае, изменение функции почти не изменилось и мы попали на границу региона, то есть смысл увеличить радиус; в ином другом случае – остается неизменным.

$$\Delta_{k+1} = \begin{cases} \frac{1}{4}\Delta_k, & \rho_k < \frac{1}{4} \\ \min(2\Delta_k, \Delta_{\max}), & \rho_k > \frac{3}{4} \wedge \|p_k\| = \Delta_k \\ \Delta_k, & \text{в ином другом случае} \end{cases}$$

Наконец, начинается самое интересное со стороны Powell Dog Leg. Итак, мы находимся на некоторой точки нашей модели, есть подсчитанный  $\Delta$ -радиуса доверительного региона, и посмотрим на полный шаг  $p^B = -B^{-1}g$ . Если  $p^B$  лежит в окружности региона, то мы можем его взять и более закончить алгоритм. В ином случае, рассмотрим анти-градиент  $-g$  и попробуем вдоль нее поискать минимум квадратичной модели, то есть решить

$$\min_{\|-\tau g\| \leq \Delta} m(-\tau g)$$

Для её решения мы можем взять некую новую точку без каких-либо ограничений в направлении анти-градиента и найти минимум модели

$$p^U = -\frac{g^T g}{g^T B g} g$$

Здесь снова две ситуации, где может находиться т.  $p^U$ :

- ▷ Если она находится вне рассматриваемой области, то мы можем взять точку на границе и шагнуть туда.
- ▷ Если же она находится в окружности, то построим отрезок  $p^U p^B$  и начнем искать минимум вдоль этих двух линий  $\left( \begin{array}{c} \text{текущая} \rightarrow p^U \text{ и } p^U \rightarrow p^B \end{array} \right)$ .

Наконец, вдоль пути мы рассматриваем траекторию  $\hat{p}(\tau)$

$$\hat{p}(\tau) = \begin{cases} \tau p^U, & 0 \leq \tau \leq 1 \\ p^U + (\tau - 1)(p^B - p^U), & 1 \leq \tau \leq 2 \end{cases}$$

Подытожим. Мы получили, на самом деле, в чем-то схожий на метод Гаусса-Ньютона алгоритм нахождения схождения, в частности, кстати, точка  $p^B$  – это то, куда бы шагнул метод Гаусса-Ньютона, но при этом, если эта точка удовлетворяет нашим потребностям, то мы действуем как Гаусс-Ньютон, в ином случае – чуть по другому. Причем под «немного другим» способом предполагается, на самом деле, хитрая комбинация Гаусса-Ньютона и градиентного спуска (так как при маленьком доверительном регионе мы пойдем по направлению, близкому градиентному спуску).

## Исследования

# BFGS

*BFGS*, или Алгоритм Бройдена - Флетчера - Гольдфарба - Шанно – это тоже оптимизационный итерационный алгоритм для нахождения локального экстремума для не представимых данных или функций в линейном/полиномиальном виде.

Один из известных квазиньютоновских методов (то есть, тех, которые основаны на получении информации о кривизне функции). Как и в случае с Powell Dog Leg, данный метод, в отличии от многих квазиньютоновских, использует аналог довольно медлительного постоянного переопределения Гессiana функции. Но если предыдущий механизм никак не взаимодействовал с явным Гессианом, то BFGS, наоборот, ускоряет работу на порядок: ибо он не явно каждый раз высчитывает матрицу, а лишь приближает к ней значения, при этом посчитав по честному Гессиан лишь один раз.

Рассмотрим идею этого алгоритма. Обозначим за  $x_i = \{x_i^0, x_i^1, \dots, x_i^{n-1}\}$  – координата в пространстве, где  $n$  – размерность соответствующего пространства. Пусть дана нам некоторая функция  $f(x)$  и, как обычно, решаем задачу оптимизации нахождения  $\arg\min_x f(x)$ . Тогда, зададим некую начальную точку  $x_0$  и  $H_0 = B_0^{-1}$  – начальное приближение, где  $B_0^{-1}$  – обратный Гессиан функции, который или может быть посчитан в точке  $x_0$ , или выбран как  $\mathbf{I}$  – обратная матрица. Наконец, сам алгоритм:

- 0) Пусть  $k$  – текущий номер итерации алгоритма.
- 1) Находим точку, в направлении которой будем производить поиск, она определяется следующим образом

$$p_k = -H_k \times \nabla f_k,$$

где здесь и далее  $f_k = f(x_k)$ .

- 2) Вычисляем  $x_{k+1}$  через рекуррентное соотношение следующего вида:

$$x_{k+1} = x_k + \alpha \cdot p_k,$$

где  $\alpha$  – коэффициент, удовлетворяющий условиям Вольфа, которые, напомним, выглядят вот так

$$\begin{aligned} f(x_k + \alpha \cdot p_k) &\leq f(x_k) + c_1 \cdot \alpha \cdot \nabla f_k^T p_k \\ \nabla f(x_k + \alpha \cdot p_k)^T p_k &\geq c_2 \cdot \nabla f_k^T p_k \end{aligned}$$

- 3) Теперь определим размер шага алгоритма после данной итерации и изменение градиента следующими соответствующими образами

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f_{k+1} - \nabla f_k \end{aligned}$$

- 4) Наконец, обновим Гессиан функции, зная, что  $\lambda = \frac{1}{y_k^T s_k} \in \mathbb{R}$

$$H_{k+1} = (\mathbf{I} - \lambda s_k y_k^T) H_k (\mathbf{I} - \lambda y_k s_k^T) + \lambda s_k s_k^T$$

## Исследования

## L-BFGS

*L-BFGS*, или BFGS с ограниченной памятью – это оптимизационный алгоритм, который аппроксимирует оригинальный алгоритм BFGS с использованием заданного ограниченного объема памяти.

L-BFGS как BFGS использует приближенную оценку Гессиана, при этом в явном виде посчитав только один раз, а все остальные шаги лишь преобразовывая. Проблема: BFGS хранит всегда  $n \times n$  приближение к обратному Гессиану. Решение: хранить несколько векторов, которые неявно представляют приближение, представляющие из себя историю последних  $m$  обновлений положения  $\vec{x}$  и градиента  $\nabla f(\vec{x})$ . При этом,  $m$  обычно выбирается небольшим ( $m < 10$ ).

Рассмотрим идею этого алгоритма. Во многом она будет совпадать с предыдущим, поэтому пропустим обозначения и перейдем сразу алгоритму.

0) Пусть  $\mathbf{i}$  – текущий номер итерации алгоритма, возьмем  $g_i = \nabla f(x_i)$ .

1) Также находим точку, в направлении которой будем производить поиск:

$$p_i = -H_i \times \nabla f_i$$

2) Пусть мы сохранили  $m$  обновлений вида:

$$s_i = x_{i+1} - x_i$$

$$y_i = g_{i+1} - g_i$$

3) Определим  $\rho_i = \frac{1}{y_i^T s_i}$  и  $H_i^0$  – «начальная» аппроксимация обратного гессиана, с которого начинается наша оценка на  $\mathbf{i}$ -ой итерации. Теперь, наконец, основная оптимизация: мы хотим оптимизировать основную рекуррентную формулу.

▷ Для данного  $\mathbf{i}$  определим  $\{q_{i-m}, q_{i-m+1}, \dots, q_i\}$ , где

$$q_i = g_i$$

$$q_i = (\mathbf{I} - \rho_i y_i s_i^T) q_{i+1} \quad \forall i \setminus \mathbf{i}$$

▷ Тогда рекурсивный алгоритм вычисления  $q_i$  от  $q_{i+1}$  состоит в том, чтобы определить  $\alpha_i \leftarrow \rho_i s_i^T q_{i+1}$  и  $q_i = q_{i+1} - \alpha_i y_i$ .

▷ Определим также  $\{z_{i-m}, z_{i-m+1}, \dots, z_i\}$ , где  $\forall i \quad z_i \leftarrow H_i q_i$ .

Существует еще один рекурсивный алгоритм вычисления этих векторов, который заключается в определении  $z_{i-m} \leftarrow H_i^0 q_{i-m}$ , а затем рекурсивно определить  $\beta_i \leftarrow \rho_i y_i^T z_i$  и  $z_{i+1} = z_i + (\alpha_i - \beta_i) s_i$ . Значение  $z_i$  тогда – наше направление восхождения.

Таким образом, мы можем вычислить направление спуска следующим образом:

```

1      q ← ∇fi;
2      for i ∈ [i-1, i-2, ..., i-m] do
3          α ← ρi siT q;
4          q ← q - αi yi;
```

```

5         end (for)
6          $r \leftarrow H_{\mathbf{i}}^0 q$ ;
7         for  $i \in [k - m, k - m + 1, \dots, k - 1]$  do
8              $\beta \leftarrow \rho_i y_i^T r$ ;
9              $r \leftarrow r + s_i(\alpha_i - \beta)$ ;
10        end (for)
11        return  $r \equiv H_{\mathbf{i}} \nabla f_{\mathbf{i}}$ ;
12

```

4) Положим  $H_{\mathbf{i}}^0 = \gamma_{\mathbf{i}} \mathbf{I}$  следующим образом:

$$\gamma_{\mathbf{i}} = \frac{s_{\mathbf{i}-1}^T y_{\mathbf{i}-1}}{y_{\mathbf{i}-1}^T y_{\mathbf{i}-1}}$$

## Исследования