

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

Методы оптимизации

Отчет по лабораторной работе №3

⟨Собрано 5 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 $w \in \mathbb{R}^n$, тогда сумма квадратов регрессионных остатков высчитывается как

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

Итак, теперь введем некоторые новые объекты для решения задачи, пусть $w^0 = (w_0^0, w_1^0, \dots, w_p^0)$ – начальное приближение, и

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

$$\vec{f}_t = (f(w^t, x_i))_{(\text{sizeof } X) \times 1} \quad - \text{вектор значений } f$$

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

Тогда, формула t -й итерации рассматриваемого метода будет высчитываться как

$$w^{t+1} \leftarrow w^t - \delta_t \cdot \underbrace{(\mathbf{J}_t^T \mathbf{J}_t)^{-1} \mathbf{J}_t^T}_{\Psi} (\vec{f}_t - y),$$

где Ψ – это псевдообратная матрица, или решение некоторой задачи многомерной линейной регрессии, где мы ищем такой вектор Ψ , что

$$\left\| \mathfrak{J}_t \Psi - (\vec{f}_t - y) \right\|^2 \rightarrow \min,$$

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^{t+1} , который бы сократил нам расстояние невязки. Причем, заметим, что на каждом шаге, задача будет новой, так как \mathfrak{J}_t зависит от текущего приближения, чтобы решить задачу многомерной регрессии.

Исследования

Powell Dog Leg

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). Тут следует сразу пояснить, что в квазиньютоновских методах для нахождения оптимальных параметров используется довольно медлительное определение *гесссиана* функции (или: матрица вторых производных). И вот тут данный алгоритм ускоряет работу на порядок: ибо он не явно каждый раз высчитывает матрицу, а лишь приближает к ней значения.

Рассмотрим идею этого алгоритма. Пусть дана нам некоторая функция $f(\vec{x})$ и, как обычно, решаем задачу оптимизации нахождения $\underset{\vec{x}}{\operatorname{argmin}} f(\vec{x})$. Пусть также $x_i = \{x_i^0, x_i^1, \dots, x_i^{n-1}\}$, где n – размерность рассматриваемого пространства; $x_0 \leftarrow \mathbf{INIT}$ – начальная точка; $H_0 = B_0^{-1}$ – начальное приближение, где B_0^{-1} – обратный гесссиан функции. Тогда:

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

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

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

$$x_{i+1} = x_i + \alpha \cdot p_i,$$

где α – коэффициент, удовлетворяющий условиям Вольфа, которые, напомним, выглядят вот так:

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

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

$$\begin{aligned} s_i &= x_{i+1} - x_i \\ y_i &= \nabla f_{i+1} - \nabla f_i \end{aligned}$$

- 4) Наконец, обновим гесссиан функции, зная, что \mathbf{I} – единичная матрица и $\lambda = \frac{1}{y_i^T s_i}$:

$$H_{i+1} = (\mathbf{I} - \lambda s_i y_i^T) H_i (\mathbf{I} - \lambda y_i s_i^T) + \lambda s_i s_i^T$$

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 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 i -ой итерации. Теперь, наконец, основная оптимизация: мы хотим оптимизировать основную рекуррентную формулу.

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

$$q_i = g_i$$

$$q_j = (\mathbf{I} - \rho_j y_j s_j^T) q_{j+1} \quad \forall j \setminus i$$

▷ Тогда рекурсивный алгоритм вычисления q_j от q_{j+1} состоит в том, чтобы определить $\alpha_j \leftarrow \rho_j s_j^T q_{j+1}$ и $q_j = q_{j+1} - \alpha_j y_j$.

▷ Определим также $\{z_{i-m}, z_{i-m+1}, \dots, z_i\}$, где $\forall j \quad z_j \leftarrow H_j q_j$.

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 $\beta_j \leftarrow \rho_j y_j^T z_j$ и $z_{j+1} = z_j + (\alpha_j - \beta_j) s_j$. Значение z_i тогда – наше направление восхождения.

Таким образом, мы можем вычислить направление спуска следующим образом:

```
1      q ← gi
2      for j ← i - 1, ..., i - m do
3          αj ← ρj sjT q
4          q ← q - αj yj
```

```

5       $\gamma_i \leftarrow \frac{s_{i-1}^T y_{i-1}}{y_{i-1}^T y_{i-1}}$ 
6       $H_i^0 \leftarrow \gamma_i \mathbf{I}$ 
7       $z \leftarrow H_i^0 q$ 
8      for  $j \leftarrow i - m, \dots, i - 1$  do
9           $\beta_j \leftarrow \rho_j y_j^T z$ 
10          $z \leftarrow z + s_j(\alpha_j - \beta_j)$ 
11      $z \leftarrow -z$ 
12

```

Исследования