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 M32331
Вереня Андрей Тарасович M32331
Сотников Максим Владимирович M32331

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 $w = (w_1,\ \ldots,\ w_W)$ и свободной переменной x.
- ▷ Решить задачу по нахождению минимума сумма квадратов регрессионных остатков:

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

Однако, решая в лоб такую задачу, мы сталкиваемся с оптимизационной задачи 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

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

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

$$\begin{aligned}
&\mathbb{J} = \left(\frac{\partial f}{\partial w_j}(w^{\mathbf{i}}, x_i)\right)_{n\times m} \ -\ \text{Якобиан, или матрица первых производных}\\
&\vec{f_{\mathbf{i}}} = \left(f(w^{\mathbf{i}}, x_i)\right)_{n\times 1} \ -\ \text{вектор значений функции } f\\
&\eth_{\mathbf{i}} = \texttt{const} \ -\ \text{размера шага}
\end{aligned}$$

Тогда, формула $\mathbf{i}$-й итерации рассматриваемого метода будет высчитываться как

$$w^{\mathbf{i}+1} \leftarrow w^{\mathbf{i}} - \eth_{\mathbf{i}} \cdot \underbrace{\left(\mathbb{J}_{\mathbf{i}}^{\mathrm{T}}\mathbb{J}_{\mathbf{i}}\right)^{-1}\mathbb{J}_{\mathbf{i}}^{\mathrm{T}}}_{\beta}(\vec{f_{\mathbf{i}}} - y),$$

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

$$\left\| \mathbf{J_i}\beta - (\vec{f_\mathbf{i}} - y) \right\|^2 \to \min,$$

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 $w^{\mathbf{i}+1}$, 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: $m \geqslant n$, в ином случае для $\mathbf{J_i^T J_i}$ не будет существовать обратной матрицы и, в следствии, решения к уравнению.

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 *гессиана* 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 $x_i = \{x_i^0,\ x_i^1,\ \ldots,\ x_i^{n-1}\}$, где n – размерность рассматриваемого пространства; $x_0 \leftarrow \mathbf{INIT}$ — начальная точка; $H_0 = B_0^{-1}$ — начальное приближение, где B_0^{-1} – обратный гессиан 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, напомню, выглядят вот так:

$$f(x_i + \alpha \cdot p_i) \leqslant f(x_i) + c_1 \cdot \alpha \cdot \nabla f_k^T p_i$$
$$\nabla f(x_i + \alpha \cdot p_i)^T p_i \geqslant c_2 \cdot \nabla f_i^T p_i$$

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

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

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

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

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 $g_{\mathbf{i}} = \nabla f(x_{\mathbf{i}})$.

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

$$p_{\mathbf{i}} = -H_{\mathbf{i}} \times \nabla f_{\mathbf{i}}$$

2) Пусть мы сохранили m обновлений вида:

$$s_{\mathbf{i}} = x_{\mathbf{i}+1} - x_{\mathbf{i}}$$
$$y_{\mathbf{i}} = g_{\mathbf{i}+1} - g_{\mathbf{i}}$$

3) Определим $\rho_{\mathbf{i}} = \dfrac{1}{y_{\mathbf{i}}^{\mathrm{T}} s_{\mathbf{i}}}$ и $H_{\mathbf{i}}^0$ – «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 $\{q_{\mathbf{i}-m},\ q_{\mathbf{i}-m+1},\ \ldots,\ q_{\mathbf{i}}\}$, где

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

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

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

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

Таким образом, мы можем вычислить направление спуска следующим образом:

```
1        q ← ∇f_i;
2        for i ∈ [i − 1, i − 2, ..., i − m] do
3            α ← ρ_i s_i^T q;
4            q ← q − α_i y_i;
```

```
        end (for)
        r ← H_i^0 q;
        for i ∈ [k − m, k − m + 1, ..., k − 1] do
            β ← ρ_i y_i^T r;
            r ← r + s_i(α_i − β);
        end (for)
        return r ≡ H_i ∇f_i;

```

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

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

Исследования