Стохастический градиентный спуск. RMSProp. Бета-регрессия

Хромов Никита Андреевич

2024

1 Стохастический градиентный спуск

Ниже приведён исходный код реализации стандартного градиентного спуска для оптимизации произвольной функции $f \in C^1(\mathbb{R}^p)$.

```python
from numpy.linalg import norm
from numpy.typing import NDArray
from typing import Callable


def GradientDescent(
    start: NDArray,
    f_grad: Callable,
    f: Callable | None = None,
    learning_rate: float = 0.01,
    max_iter=1000,
    tol=1e-4,
    **kwargs
) -> dict:
    curr_point = start
    curr_value = None
    curr_grad = f_grad(curr_point, **kwargs)

    curr_iter = 0
    while curr_iter == 0 or (
        curr_iter < max_iter and learning_rate * norm(curr_grad) >= tol
    ):
        curr_point = curr_point - learning_rate * curr_grad
        curr_grad = f_grad(curr_point, **kwargs)
        curr_iter += 1

    if f is not None:
        curr_value = f(curr_point, **kwargs)

    return {
        "point": curr_point,
        "f_value": curr_value,
        "grad_value": curr_grad,
        "iterations": curr_iter,
    }
```

Листинг 1: Реализация градиентного спуска

Описание параметров функции:

- `start` — начальная точка x_0 для оптимизации функции.
- `f_grad` — градиент оптимизируемой функции.
- `f` — оптимизируемая функция (опционально, используется только чтобы вычислить значение в последней точке оптимизации).
- `learning_rate` — длина оптимизационного шага (на что умножается градиент при нахождении новой точки x_i).
- `max_iter` — максимальное число итераций алгоритма, при достижении этого значения алгоритм прекращает работу.
- `tol` — точность для критерия остановки: если $\|x_{i-1} - x_i\| < \texttt{tol}$, алгоритм прекращается.

- `**kwargs` — дополнительные параметры, передаваемые функциям `f` и `f_grad`.

Функция возвращает словарь, состоящий из последней точки оптимизационного процесса, значения функции `f` в этой точке (если она была передана), значения градиента `f_grad` в этой точке и количества итераций алгоритма.

Ниже приведён исходный код реализации стохастического градиентного спуска для функций вида $L(W, \mathbf{X}, Y)$, где $W \in \mathbb{R}^p$ — параметр, по которому происходит оптимизация, $\mathbf{X} \in \mathbb{R}^{n \times p}$ и $Y \in \mathbb{R}^n$. В частном случае: L — лосс-функция в некоторой регрессии, W — параметры регрессии, $\mathbf{X}$ — регрессоры, Y — отклики.

```python
import numpy as np
from numpy.linalg import norm
from numpy.random import choice
from numpy.typing import NDArray
from typing import Callable


def StochasticGradientDescent(
    start: NDArray,
    X: NDArray,
    y: NDArray,
    L_grad: Callable,
    batch_size: int,
    L: Callable | None = None,
    learning_rate: float = 0.01,
    max_iter=1000,
    tol=1e-4,
    **kwargs
) -> dict:
    curr_point = start
    W_error = None
    curr_value = None
    curr_iter = 0
    while W_error is None or (curr_iter < max_iter and W_error >= tol):
        idx = choice(X.shape[0], batch_size, replace=False)
        batch_X, batch_y = X[idx, :], np.array(y[idx]).reshape(idx.shape)

        curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)

        curr_point -= learning_rate * curr_grad
        W_error = norm(learning_rate * curr_grad)
        curr_iter += 1

    if L is not None:
        curr_value = L(curr_point, X, y, **kwargs)
    return {
        "point": curr_point,
        "L_value": curr_value,
        "grad_value": curr_grad,
        "iterations": curr_iter,
    }
```

Листинг 2: Реализация стохастического градиентного спуска

Описание параметров функции:

- `start` — начальная точка w_0 для оптимизации функции.
- `X`, `y` — матрица $\mathbf{X}$ и вектор Y, указанного выше вида.
- `L_grad` — градиент оптимизируемой функции.
- `batch_size` — размер подвыборки регрессоров и откликов, используемых для вычисления градиента на каждом шаге.
- `L` — оптимизируемая функция (опционально, используется только чтобы вычислить значение в последней точке оптимизации).
- `learning_rate` — длина оптимизационного шага.
- `max_iter` — максимальное число итераций алгоритма, при достижении этого значения алгоритм прекращает работу.

- `tol` — точность для критерия остановки: если $\|w_{i-1} - w_i\| < $ `tol`, алгоритм прекращается.
- `**kwargs` — дополнительные параметры, передаваемые функциям `L` и `L_grad`.

Функция возвращает словарь, состоящий из последней точки оптимизационного процесса, значения функции `L` в этой точке (если она была передана), значения градиента `L_grad` в этой точке и количества итераций алгоритма. Подвыборка регрессоров и откликов на каждом шагу получается выбором случайного набора индексов из множества $\overline{0:(n-1)}$ без повторений.

1.1 Проверка алгоритмов

Оба алгоритма применялись для нахождения минимума лосс-функции линейной регрессии:

$$L(W, \mathbf{X}, Y) = \frac{1}{n}\|\mathbf{X}W - Y\|^2, \tag{1}$$

$$\nabla L(W, \mathbf{X}, Y) = \frac{2}{n}\mathbf{X}^{\mathrm{T}}(\mathbf{X}W - Y). \tag{2}$$

В качестве $\mathbf{X}$ было взято 500 независимых реализаций четырёхмерной гауссовской величины $\mathrm{N}(\mathbf{0}, \mathbf{I}_4)$, а вектор Y был вычислен как $Y = \mathbf{X}^1 W + \varepsilon$, где $\mathbf{X}^1$ обозначает матрицу $\mathbf{X}$ с дописанным справа столбцом из единиц, $W = (2, -3, 1, 0.5, 4)^{\mathrm{T}}$, $\boldsymbol{\varepsilon}$ — выборка из 500 независимых реализаций распределения $\mathrm{N}(0, 1)$. Ниже приведена таблица с количеством итераций до сходимости каждого метода (при `max_iter` = 1000) и значением квадрата евклидова расстояния от полученного оптимизацией набора параметров до истинного значения (при `tol`=1e-4).

Метод	Итераций	$\|\widehat{W} - W\|^2$
Градиентный спуск	370	0.0114
Стохастический градиентный спуск	1000	0.0109

2 RMSProp

Ниже приведён исходный код реализации алгоритма SGD с эвристикой шага RMSProp.

```python
import numpy as np
from numpy.linalg import norm
from numpy.random import choice, permutation
from numpy.typing import NDArray
from typing import Callable


def SGD_RMSProp(
    start: NDArray,
    X: NDArray,
    y: NDArray,
    L_grad: Callable,
    batch_size: int | float,
    L: Callable | None = None,
    learning_rate: float = 0.01,
    decay_rate: float = 0.5,
    use_epoch: bool = True,
    max_iter=1000,
    tol=1e-4,
    n_iter_no_change: int = 5,
    **kwargs
) -> dict:
    curr_point = start
    min_error = None
    run_avg = np.zeros(np.size(start))
    curr_iter = 0
    curr_value = None
    n = X.shape[0]
    if L is not None:
        L_last = None
    if use_epoch:
        curr_epoch = 0

```

```python
    if type(batch_size) is float and batch_size < 1:
        batch_size = n * batch_size

    batch_size = int(batch_size)

    while min_error is None or (curr_iter < max_iter and min_error >= tol):
        if use_epoch:
            idx = permutation(n)
            X_perm, y_perm = X[idx], y[idx]

            if L is not None:
                if curr_epoch == 0:
                    L_start = L(curr_point, X, y, **kwargs)
                    L_last = L_start
                elif curr_epoch % n_iter_no_change == 0 and curr_epoch != 0:
                    if np.abs(L_last - L_start) < tol:
                        learning_rate /= 5
                    L_last = L_start

            for batch_start in range(0, n, batch_size):
                batch_end = batch_start + batch_size
                batch_X, batch_y = (
                    X_perm[batch_start:batch_end],
                    y_perm[batch_start:batch_end],
                )
                curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)

                run_avg = decay_rate * run_avg + (1 - decay_rate) * curr_grad**2

                curr_point -= learning_rate / np.sqrt(run_avg) * curr_grad
                curr_iter += 1

            W_error = norm(learning_rate * curr_grad)
            if L is not None:
                L_new = L(curr_point, X, y, **kwargs)
                L_error = np.abs(L_start - L_new)
                L_start = min(L_new, L_start)
                min_error = min(W_error, L_error)
            else:
                min_error = W_error

            curr_epoch += 1

        else:
            idx = choice(n, batch_size, replace=False)
            batch_X, batch_y = X[idx], y[idx]
            if L is not None:
                L_start = L(curr_point, X, y, **kwargs)

            curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)
            run_avg = decay_rate * run_avg + (1 - decay_rate) * curr_grad**2

            curr_point -= learning_rate / np.sqrt(run_avg) * curr_grad

            W_error = norm(learning_rate * curr_grad)
            if L is not None:
                L_error = np.abs(L_start - L(curr_point, X, y, **kwargs))
                min_error = min(W_error, L_error)
            else:
                min_error = W_error

            curr_iter += 1

    if L is not None:
        curr_value = L(curr_point, X, y, **kwargs)

    return {
        "point": curr_point,
        "L_value": curr_value,
        "grad_value": curr_grad,
        "iterations": curr_iter,
    }
```

Листинг 3: Реализация алгоритма SGD с эвристикой шага RMSProp

Описание параметров функции:

- `start`, `X`, `y`, `L_grad`, `learning_rate`, `max_iter`, `tol`, `**kwargs` — то же, что в алгоритме 1.
- `batch_size` — если число с плавающей точкой, то имеет смысл доли выборки, участвующей в оптимизации на каждом шагу, иначе, если целое число, то имеет тот же смысл, что и в алгоритме 1,
- `L` — оптимизируемая функция (опционально, используется чтобы критерий остановки мог учитывать изменение в значении оптимизируемой функции и чтобы вычислить значение в последней точке оптимизации). при достижении этого значения алгоритм прекращает работу.
- `decay_rate` — параметр скорости «забывания» старых градиентов.
- `use_epoch` — флаг того, применять ли в оптимизации эпохи (каждую эпоху выбирается перестановка строк `X` и `y` и выполняется последовательный проход по этой перестановке с шагом `batch_size`). Если `use_epoch == True`, то общее число итераций алгоритма может превысить значение `max_iter` на число не большее $n-1$.
- `n_iter_no_change` — после скольки эпох без сильного изменения целевой функции (разница между значениями `L` в текущей точке и в точке столько эпох назад меньше `tol`) уменьшить параметр `learning_rate` в 5 раз. Используется только если дана `L` и `use_epoch == True`.

Функция возвращает то же, что и функция стохастического градиентного спуска.

2.1 Проверка алгоритма

Рассматривалась та же задача, что и в разделе 1.1. Алгоритм был применён с параметрами `batch_size=100` и `decay_rate=0.9`, остальные параметры были взяты по умолчанию. Также алгоритму была передана функция потерь (1). Процесс оптимизации сошёлся за 555 итераций и квадрат евклидова расстояния от истинных параметров до полученных равен 0.0128.

3 Бета регрессия

Все утверждения и формулы взяты из статьи [1].

Пусть $\xi \sim \mathrm{B}(\alpha, \beta)$, тогда плотность ξ имеет вид

$$f_\xi(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}, \qquad x \in (0,1).$$

Для построения бета-регрессии удобнее работать в параметризации через среднее μ и "точность"φ:

$$\mu = \frac{\alpha}{\alpha+\beta}, \qquad \varphi = \alpha + \beta,$$
$$\mu \in (0,1), \qquad \varphi > 0.$$

Тогда старые параметры выражаются следующим образом:

$$\alpha = \mu\varphi, \qquad \beta = (1-\mu)\varphi.$$

Среднее и дисперсия хорошо выражаются через новые параметры:

$$\mathrm{E}(\xi) = \mu, \qquad \mathrm{D}(\xi) = \frac{\mu(1-\mu)}{1+\varphi}$$

В новой параметризации плотность ξ имеет вид

$$f_\xi(x) = \frac{\Gamma(\varphi)}{\Gamma(\mu\varphi)\Gamma((1-\mu)\varphi)} x^{\mu\varphi-1}(1-x)^{(1-\mu)\varphi-1}, \qquad x \in (0,1).$$

Пусть $\mathbf{X} \in \mathbb{R}^{n\times p}$ - выборка регрессоров, $Y \in \mathbb{R}^n$ - выборка откликов. Предполагается, что $y_i \sim \mathrm{B}(\mu_i, \varphi)$, где параметр φ неизвестен, а μ_i выражается через регрессоры:

$$g(\mu_i) = \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta}.$$

$g(t)$ - произвольная линк-функция, например логит:

$$g(\mu_i) = \log\left(\frac{\mu_i}{1-\mu_i}\right) = \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} \implies \mu_i = \frac{e^{\mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta}}}{1+e^{\mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta}}}.$$

Логарифм функции правдоподобия имеет вид

$$L(\mathbf{X}, \beta, \varphi; Y) = \sum_{i=1}^{n} l(\mu_i(\mathbf{x}_i, \beta), \varphi; y_i)$$

$$l(\mu_i(\mathbf{x}_i, \beta), \varphi; y_i) = \log\Gamma(\varphi) - \log\Gamma(\mu_i\varphi) - \log\Gamma((1-\mu_i)\varphi) + (\mu_i\varphi - 1)\log y_i + ((1-\mu_i)\varphi - 1)\log(1-y_i).$$

Пусть $\mathbf{X}$ и Y фиксированы, обозначим

$$y_i^* = \log(y_i/(1-y_i)), \qquad \mu_i^* = \psi(\mu_i\varphi) - \psi((1-\mu_i)\varphi), \qquad \mathbf{T} = \operatorname{diag}\left(1/g'(\mu_1), \ldots, 1/g'(\mu_n)\right),$$

$$Y^* = (y_1^*, \ldots, y_n^*)^{\mathrm{T}}, \qquad \boldsymbol{\mu}^* = (\mu_1^*, \ldots, \mu_n^*)^{\mathrm{T}},$$

где $\psi(z) = (\log\Gamma(z))'$ - дигамма-функция, тогда градиент логарифма функции правдоподобия равен $\nabla L(\beta, \varphi) = \left(L_\beta^{\mathrm{T}}(\beta, \varphi), L_\varphi(\beta, \varphi)\right)^{\mathrm{T}}$, где

$$L_\beta(\beta, \varphi) = \varphi \mathbf{X}^{\mathrm{T}}\mathbf{T}(Y^* - \boldsymbol{\mu}^*),$$

$$L_\varphi(\beta, \varphi) = \sum_{i=1}^{n} \big(\mu_i(y_i^* - \mu_i^*) + \log(1-y_i) - \psi((1-\mu_i)\varphi) + \psi(\varphi)\big).$$

Ниже приведён исходный код реализации функции правдоподобия, её логарифма и градиента её логарифма для бета регрессии.

```python
import numpy as np
from numpy.typing import NDArray
from scipy.special import gamma, digamma
from typing import Callable


def logit_inverse(x: NDArray, beta: NDArray) -> NDArray:
    t = np.exp(x.dot(beta))
    return t / (1 + t)


def logit_deriv(mu: NDArray) -> NDArray:
    return 1 / (mu - mu * mu)


def beta_log_likelihood(
    parameters: NDArray,
    X: NDArray,
    y: NDArray,
    link_inverse: Callable[[NDArray, NDArray], float],
    link_deriv: None = None,
) -> float:
    beta, phi = parameters[:-1], parameters[-1]
    mu = link_inverse(X, beta)

    alpha = mu * phi
    beta = (1 - mu) * phi
    return np.sum(
        (alpha - 1) * np.log(y)
        + (beta - 1) * np.log(1 - y)
        + np.log(gamma(phi) / (gamma(alpha) * gamma(beta)))
    )


def beta_inv_log_likelihood(
    parameters: NDArray,
    X: NDArray,
    y: NDArray,
    link_inverse: Callable[[NDArray, NDArray], float],
    link_deriv: None = None,
```

```
) -> float:
    return -beta_log_likelihood(parameters, X, y, link_inverse, link_deriv)


def beta_illh_grad(
    parameters: NDArray,
    X: NDArray,
    Y: NDArray,
    link_inverse: Callable[[NDArray, NDArray], NDArray],
    link_deriv: Callable[[NDArray], NDArray],
) -> NDArray:
    beta, phi = parameters[:-1], parameters[-1]
    mu_vec = link_inverse(X, beta)
    Y_star = np.log(Y / (1 - Y))
    mu_star = digamma(mu_vec * phi) - digamma((1 - mu_vec) * phi)
    T_mat = np.diag(1 / link_deriv(mu_vec))
    L_beta = phi * X.T.dot(T_mat).dot(Y_star - mu_star)
    L_phi = np.sum(
        mu_vec * (Y_star - mu_star)
        + np.log(1 - Y)
        - digamma((1 - mu_vec) * phi)
        + digamma(phi)
    )
    return -np.append(L_beta, L_phi)
```

Листинг 4: Реализация функции правдоподобия, её логарифма и градиента её логарифма

3.1 Проверка алгоритма RMSProp для нахождения параметров бета регрессии

Пусть матрица $\mathbf{X}$ равна матрице $\mathbf{X}^1$ из раздела 1.1, а $Y = (y_1, \ldots, y_n)$, где $y_i \sim \mathrm{B}(\mu_i, \varphi)$, $\mu_i = g^{-1}(\boldsymbol{x}_i \beta)$, $g(t) = \log(t/(1-t))$, $\beta = (0.1, 0.3, 0.01, 0.5, 0.4)^{\mathrm{T}}$, $\varphi = 3$, а $\boldsymbol{x}_i$ — i-я строка матрицы $\mathbf{X}$. Оптимизационный алгоритм RMSProp был запущен с параметрами `batch_size=50`, `learning_rate=0.01`, `decay_rate=0.1`, и остальными по умолчанию (также алгоритму был передан логарифм функции правдоподобия с отрицательным знаком в качестве оптимизируемой функции). Оптимизационный процесс завершился за 910 итераций и квадрат евклидова расстояния от истинных параметров до полученных равен 0.005.

4 Применение алгоритма RMSProp к данным в модели бета регрессии

Реализация SGD с RMSProp была применена к предварительно обработанным данным для оценки параметров в модели бета регрессии. Полученные параметры сравнивались с результатами функции `betareg` из одноимённого пакета [2] в языке R. Также полученная модель бета регрессии сравнивалась с моделью линейной регрессии, полученной аналогичным применением SGD с RMSProp, и с моделью линейной регрессии, полученной применением функции `SGDRegressor` из модуля `sklearn` в языке Python. Все модели были обучены на одной тренировочной подвыборке, содержащей 77% исходных наблюдений, предварительная обработка данных заключалась в приведении откликов к отрезку $(0, 1)$, и стандартизации регрессоров (со средними и стандартными отклонениями, посчитанными только по тренировочной выборке).

В таблице 1 приведены значения параметров бета регрессии, полученных разными функциями. Стоит заметить, что функция `betareg` также проверяет параметры на значимость, и в данном случае при уровне значимости $\alpha = 0.05$ значимыми оказались только β_3,, β_4 и φ. В таблице 2 приведены значения параметров линейной регрессии, полученных разными функциями. С уровнем значимости $\alpha = 0.05$ значимыми параметрами являются только β_3 и β_4.

	$\boldsymbol{\beta}$	φ
SGD + RMSProp	$(0.005, -0.001, -0.245, -1.696)$	3.22
betareg	$(0.004, -0.009, -0.244, -1.668)$	3.15

Таблица 1: Оценённые параметры бета регрессии

	$\boldsymbol{\beta}$
SGD + RMSProp	$(0.002, 0.0001, -0.064, 0.138)$
SGDRegressor	$(0.002, -0.0001, -0.064, 0.137)$

Таблица 2: Оценённые параметры линейной регрессии.

4.1 Проверка качества моделей

Для сравнения полученных моделей были использованы следующие метрики.

1. R^2:
$$1 - \frac{\sum_{i=1}^{n}(y_i^* - y_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2},$$
где n - число индивидов, y_i — значение признака, y_i^* — его предсказание по модели, $\overline{y}$ — выборочное среднее.

2. BIC:
$$\mathrm{BIC} = k\log(n) - 2\log(L_m).$$
где k - число оценённых параметров модели, L_m - функция правдоподобия модели.

3. WRMSE:
$$\sqrt{\frac{1}{n}\sum_{i=1}^{n} w_i(y_i - y_i^*)^2},$$
где $w_i = C/\widehat{\sigma}_i^2$, C — нормирующий веса множитель, а $\widehat{\sigma}_i^2$ — оценка дисперсии i-го остатка (WRMSE совпадает с RMSE в гомоскедастической модели).

Стоит заметить, что данные имеют сильно нелинейную структуру, так как регрессоры являются идентификаторами и скорее должны считаться категориальными переменными, а значения откликов заведомо не выходят за интервал $(0, 1)$. Поэтому показатель R^2 не является хорошим показателем качества модели в данном случае и приведён для справки.

В таблице 3 представлены значения метрик качества моделей, измеренных на тестовой выборке. Ожидаемо, R^2 выше у линейных моделей, но в целом его значение довольно мало. Значения BIC у

Модель регрессии	R^2	BIC	WRMSE
Линейная (SGD+RMSProp)	0.094	-744.6	0.202
Линейная (sklearn)	0.094	-744.2	0.202
Бета (SGD+RMSProp)	0.061	-4499.3	0.196
Бета (betareg)	0.059	-4500.5	0.196

Таблица 3: Показатели качества моделей на тестовой выборке.

моделей бета регрессии сильно меньше, чем у линейных, что говорит о том, что модель бета регрессии больше соответствует данным, чем модель линейной регрессии. Показатели WRMSE у моделей бета регрессии меньше, в том числе потому что эта модель учитывает гетероскедастичность откликов.

Таким образом, модель бета регрессии подходит данным больше, чем модель линейной регрессии. Но модели можно улучшить и далее, например заменив все регрессоры на их one-hot кодировки.

Список литературы

[1] Ferrari Silvia, Cribari-Neto Francisco. Beta Regression for Modelling Rates and Proportions // Journal of Applied Statistics. — 2004. — Vol. 31, no. 7. — P. 799–815.

[2] Zeileis Achim, Cribari-Neto Francisco, Grün Bettinas. — betareg: Beta Regression : 2004. — R package version 3.2-1. Access mode: `https://CRAN.R-project.org/package=betareg`.