

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 $f \in C^1(\mathbb{R}^p)$.

```
1 from numpy.linalg import norm
2 from numpy.typing import NDArray
3 from typing import Callable
4
5
6 def GradientDescent(
7     start: NDArray,
8     f_grad: Callable,
9     f: Callable | None = None,
10    learning_rate: float = 0.01,
11    max_iter=1000,
12    tol=1e-4,
13    **kwargs
14 ) -> dict:
15     curr_point = start
16     curr_value = None
17     curr_grad = f_grad(curr_point, **kwargs)
18
19     curr_iter = 0
20     while curr_iter == 0 or (
21         curr_iter < max_iter and learning_rate * norm(curr_grad) >= tol
22     ):
23         curr_point = curr_point - learning_rate * curr_grad
24         curr_grad = f_grad(curr_point, **kwargs)
25         curr_iter += 1
26
27     if f is not None:
28         curr_value = f(curr_point, **kwargs)
29
30     return {
31         "point": curr_point,
32         "f_value": curr_value,
33         "grad_value": curr_grad,
34         "iterations": curr_iter,
35     }
```

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 $\|x_{i-1} - x_i\| < \text{tol}$, 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.

```

1 import numpy as np
2 from numpy.linalg import norm
3 from numpy.random import choice
4 from numpy.typing import NDArray
5 from typing import Callable
6
7
8 def StochasticGradientDescent(
9     start: NDArray,
10    X: NDArray,
11    y: NDArray,
12    L_grad: Callable,
13    batch_size: int,
14    L: Callable | None = None,
15    learning_rate: float = 0.01,
16    max_iter=1000,
17    tol=1e-4,
18    **kwargs
19 ) -> dict:
20     curr_point = start
21     W_error = None
22     curr_value = None
23     curr_iter = 0
24     while W_error is None or (curr_iter < max_iter and W_error >= tol):
25         idx = choice(X.shape[0], batch_size, replace=False)
26         batch_X, batch_y = X[idx, :], np.array(y[idx]).reshape(idx.shape)
27
28         curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)
29
30         curr_point -= learning_rate * curr_grad
31         W_error = norm(learning_rate * curr_grad)
32         curr_iter += 1
33
34     if L is not None:
35         curr_value = L(curr_point, X, y, **kwargs)
36     return {
37         "point": curr_point,
38         "L_value": curr_value,
39         "grad_value": curr_grad,
40         "iterations": curr_iter,
41     }

```

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 $\|w_{i-1} - w_i\| < \text{tol}$, 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 $0 : (n - 1)$ без повторений.

1.1 Проверка алгоритмов

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

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

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

В качестве \mathbf{X} было взято 500 независимых реализаций четырёхмерной гауссовской величины $N(\mathbf{0}, \mathbf{I}_4)$, а вектор Y был вычислен как $Y = \mathbf{X}^T W + \varepsilon$, где \mathbf{X}^T обозначает матрицу \mathbf{X} с дописанным справа столбцом из единиц, $W = (2, -3, 1, 0.5, 4)^T$, ε — выборка из 500 независимых реализаций распределения $N(0, 1)$. 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.

```

1 import numpy as np
2 from numpy.linalg import norm
3 from numpy.random import choice, permutation
4 from numpy.typing import NDArray
5 from typing import Callable
6
7
8 def SGD_RMSProp(
9     start: NDArray,
10    X: NDArray,
11    y: NDArray,
12    L_grad: Callable,
13    batch_size: int | float,
14    L: Callable | None = None,
15    learning_rate: float = 0.01,
16    decay_rate: float = 0.5,
17    use_epoch: bool = True,
18    max_iter=1000,
19    tol=1e-4,
20    n_iter_no_change: int = 5,
21    **kwargs
22 ) -> dict:
23     curr_point = start
24     min_error = None
25     run_avg = np.zeros(np.size(start))
26     curr_iter = 0
27     curr_value = None
28     n = X.shape[0]
29     if L is not None:
30         L_last = None
31     if use_epoch:
32         curr_epoch = 0
33 
```

```

34     if type(batch_size) is float and batch_size < 1:
35         batch_size = n * batch_size
36
37     batch_size = int(batch_size)
38
39     while min_error is None or (curr_iter < max_iter and min_error >= tol):
40         if use_epoch:
41             idx = permutation(n)
42             X_perm, y_perm = X[idx], y[idx]
43
44             if L is not None:
45                 if curr_epoch == 0:
46                     L_start = L(curr_point, X, y, **kwargs)
47                     L_last = L_start
48                 elif curr_epoch % n_iter_no_change == 0 and curr_epoch != 0:
49                     if np.abs(L_last - L_start) < tol:
50                         learning_rate /= 5
51                     L_last = L_start
52
53                 for batch_start in range(0, n, batch_size):
54                     batch_end = batch_start + batch_size
55                     batch_X, batch_y = (
56                         X_perm[batch_start:batch_end],
57                         y_perm[batch_start:batch_end],
58                     )
59                     curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)
60
61                     run_avg = decay_rate * run_avg + (1 - decay_rate) * curr_grad**2
62
63                     curr_point -= learning_rate / np.sqrt(run_avg) * curr_grad
64                     curr_iter += 1
65
66                 W_error = norm(learning_rate * curr_grad)
67                 if L is not None:
68                     L_new = L(curr_point, X, y, **kwargs)
69                     L_error = np.abs(L_start - L_new)
70                     L_start = min(L_new, L_start)
71                     min_error = min(W_error, L_error)
72                 else:
73                     min_error = W_error
74
75                 curr_epoch += 1
76
77             else:
78                 idx = choice(n, batch_size, replace=False)
79                 batch_X, batch_y = X[idx], y[idx]
80                 if L is not None:
81                     L_start = L(curr_point, X, y, **kwargs)
82
83                 curr_grad = L_grad(curr_point, batch_X, batch_y, **kwargs)
84                 run_avg = decay_rate * run_avg + (1 - decay_rate) * curr_grad**2
85
86                 curr_point -= learning_rate / np.sqrt(run_avg) * curr_grad
87
88                 W_error = norm(learning_rate * curr_grad)
89                 if L is not None:
90                     L_error = np.abs(L_start - L(curr_point, X, y, **kwargs))
91                     min_error = min(W_error, L_error)
92                 else:
93                     min_error = W_error
94
95                 curr_iter += 1
96
97             if L is not None:
98                 curr_value = L(curr_point, X, y, **kwargs)
99
100     return {
101         "point": curr_point,
102         "L_value": curr_value,
103         "grad_value": curr_grad,
104         "iterations": curr_iter,
105     }

```

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 сколько 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 $\xi \sim B(\alpha, \beta)$, тогда плотность ξ имеет вид

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

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

$$\begin{aligned} \mu &= \frac{\alpha}{\alpha + \beta}, & \varphi &= \alpha + \beta, \\ \mu &\in (0, 1), & \varphi &> 0. \end{aligned}$$

Тогда старые параметры выражаются следующим образом:

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

Среднее и дисперсия хорошо выражаются через новые параметры:

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

В новой параметризации плотность ξ имеет вид

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

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

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

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

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

Логарифм функции правдоподобия имеет вид

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

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

Пусть \mathbf{X} и Y фиксированы, обозначим

$$y_i^* = \log(y_i / (1 - y_i)), \quad \mu_i^* = \psi(\mu_i \varphi) - \psi((1 - \mu_i) \varphi), \quad \mathbf{T} = \text{diag}(1/g'(\mu_1), \dots, 1/g'(\mu_n)),$$

$$Y^* = (y_1^*, \dots, y_n^*)^T, \quad \boldsymbol{\mu}^* = (\mu_1^*, \dots, \mu_n^*)^T,$$

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

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

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

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

```

1 import numpy as np
2 from numpy.typing import NDArray
3 from scipy.special import gamma, digamma
4 from typing import Callable
5
6
7 def logit_inverse(x: NDArray, beta: NDArray) -> NDArray:
8     t = np.exp(x.dot(beta))
9     return t / (1 + t)
10
11
12 def logit_deriv(mu: NDArray) -> NDArray:
13     return 1 / (mu - mu * mu)
14
15
16 def beta_log_likelihood(
17     parameters: NDArray,
18     X: NDArray,
19     y: NDArray,
20     link_inverse: Callable[[NDArray, NDArray], float],
21     link_deriv: None = None,
22 ) -> float:
23     beta, phi = parameters[:-1], parameters[-1]
24     mu = link_inverse(X, beta)
25
26     alpha = mu * phi
27     beta = (1 - mu) * phi
28     return np.sum(
29         (alpha - 1) * np.log(y)
30         + (beta - 1) * np.log(1 - y)
31         + np.log(gamma(phi) / (gamma(alpha) * gamma(beta)))
32     )
33
34
35 def beta_inv_log_likelihood(
36     parameters: NDArray,
37     X: NDArray,
38     y: NDArray,
39     link_inverse: Callable[[NDArray, NDArray], float],
40     link_deriv: None = None,

```

```

41 ) -> float:
42     return -beta_log_likelihood(parameters, X, y, link_inverse, link_deriv)
43
44
45 def beta_illh_grad(
46     parameters: NDArray,
47     X: NDArray,
48     Y: NDArray,
49     link_inverse: Callable[[NDArray, NDArray], NDArray],
50     link_deriv: Callable[[NDArray], NDArray],
51 ) -> NDArray:
52     beta, phi = parameters[:-1], parameters[-1]
53     mu_vec = link_inverse(X, beta)
54     Y_star = np.log(Y / (1 - Y))
55     mu_star = digamma(mu_vec * phi) - digamma((1 - mu_vec) * phi)
56     T_mat = np.diag(1 / link_deriv(mu_vec))
57     L_beta = phi * X.T.dot(T_mat).dot(Y_star - mu_star)
58     L_phi = np.sum(
59         mu_vec * (Y_star - mu_star)
60         + np.log(1 - Y)
61         - digamma((1 - mu_vec) * phi)
62         + digamma(phi)
63     )
64     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 $Y = (y_1, \dots, y_n)$, где $y_i \sim B(\mu_i, \varphi)$, $\mu_i = g^{-1}(\mathbf{x}_i \beta)$, $g(t) = \log(t/(1-t))$, $\beta = (0.1, 0.3, 0.01, 0.5, 0.4)^T$, $\varphi = 3$, а \mathbf{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 .

	β	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

	β
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 :

$$1 - \frac{\sum_{i=1}^n (y_i^* - y_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2},$$

где n - число индивидов, y_i - значение признака, y_i^* - его предсказание по модели, \bar{y} - выборочное среднее.

2. BIC:

$$\text{BIC} = k \log(n) - 2 \log(L_m).$$

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 $w_i = C/\hat{\sigma}_i^2$, C - нормирующий веса множитель, а $\hat{\sigma}_i^2$ - 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. Кроме того, в силу

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] — betareg: Beta Regression : 2004. — R package version 3.2-1. Access mode: <https://CRAN.R-project.org/package=betareg>.