

Большое практическое задание 2.

Классификация заемщиков линейными моделями

Курс «Машинное обучение 1»

Формулировка задания

Данное задание направлено на ознакомление с линейными моделями и градиентными методами обучения. В задании необходимо:

1. Написать на языке Python собственную реализацию линейного классификатора с произвольной функцией потерь и реализацию функции и градиента функции потерь для логистической регрессии. Реализации можно частично проверить через юнит-тесты (запускаются командой `pytest tests.py`).
2. Вывести все необходимые формулы, привести выкладки в отчёте.
3. Провести описанные ниже эксперименты с модельными данными и приложенным датасетом.
4. Написать отчёт о проделанной работе (в формате jupyter notebook).

Теоретическая часть (1 балл)

Выведите формулу градиента функции потерь для задачи бинарной логистической регрессии. Запишите вывод градиента в отчёт.

$$L(a(x), y) = \log(1 + \exp(-ya(x))), \quad y \in \{-1, 1\}, \quad a(x) \in (-\infty, \infty)$$

Реализация алгоритмов (9 баллов)

Прототипы функций должны строго соответствовать прототипам, описанным в спецификации и проходить все тесты. Задание, не проходящее все тесты, приравнивается к невыполненному. При написании необходимо пользоваться стандартными средствами языка Python, библиотеками `numpy`, `scipy` и `matplotlib`. Библиотекой `scikit-learn` для реализаций пользоваться запрещается, но разрешается использовать её в процессе экспериментов. Все подробности реализации алгоритмов подробно описаны в спецификации к заданию.

Список экспериментов (15 баллов)

В приложенном ноутбуке.

Бонусная часть (7 баллов)

В приложенном ноутбуке.

Линейная модель бинарной классификации

Рассмотрим задачу бинарной классификации. Пусть дана обучающая выборка $X = (x_i, y_i)_{i=1}^N$, где $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{Y} = \{1, -1\}$. Линейная модель классификации определяется следующим образом:

$$a(x) = \text{sign}(\langle w, x \rangle + w_0), \quad \text{где } w \in \mathbb{R}^d \text{ — вектор весов, } w_0 \text{ — сдвиг.}$$

$$\text{sign}(z) = \begin{cases} -1, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

Далее будем считать, что среди признаков есть константа. Тогда классификатор задаётся как:

$$a(x) = \text{sign}(\langle w, x \rangle)$$

Заметим, что на практике часто порог бинаризации α отдельно подбирается перебором по всем возможным значениям порога на отложенной выборке. Так как классы часто несбалансированы, выбор порога по отложенной выборке может приводить к лучшим результатам.

$$a(x) = \text{sign}(\langle w, x \rangle - \alpha)$$

Процесс обучения заключается в настройке вектора w . Величина $M_i(w) = y_i \langle w, x_i \rangle$ называется отступом (margin) объекта x_i относительно алгоритма $a(x)$. Если $M_i(w) < 0$, алгоритм допускает ошибку на объекте x_i . Чем больше отступ $M_i(w)$, тем более надёжно и правильно алгоритм классифицирует объект x_i . Пусть $\mathcal{L}(M(w))$ — монотонно невозрастающая функция, такая что $\mathbb{I}[M(w) < 0] \leq \mathcal{L}(M(w))$. Будем настраивать вектор весов w , оптимизируя функционал $Q(X, w)$:

$$Q(X, w) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(M_i(w)) \rightarrow \min_w$$

Метод обучения, использующий логистическую функцию потерь $\log(1 + \exp(-M))$, называется логистической регрессией. Одно из основных свойств логистической регрессии — возможность корректного оценивания вероятности принадлежности объекта к каждому из классов:

$$p(y = 1|x) = \frac{1}{1 + \exp(-\langle w, x \rangle)} = \sigma(\langle w, x \rangle)$$

Нетрудно заметить, что максимизация логарифма правдоподобия выборки при такой параметризации вероятностей эквивалентна минимизации суммы логистических функций потерь.

Обычно к оптимизируемому функционалу добавляют второе слагаемое — регуляризатор, не зависящий от данных. Второе слагаемое ограничивает вектор параметров модели тем самым уменьшая переобучение. Один из примеров регуляризации — L_2 регуляризатор:

$$Q(X, w) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(M_i(w)) + \frac{\lambda}{2} \|w\|_2^2 \rightarrow \min_w$$

Многоклассовая классификация

Пусть теперь множество $\mathbb{Y} = \{1, \dots, K\}$. Задачу многоклассовой классификации можно свести к набору бинарных задач.

- Один против всех (one-vs-all)

Обучается K классификаторов $a_1(x), \dots, a_K(x)$. Алгоритм $a_j(x)$ обучается по выборке X_j :

$$X_j = \{(x_i, 2\mathbb{I}[y_i = j] - 1) \mid (x_i, y_i) \in X\}$$

Таким образом, каждому классификатору $a_j(x)$ соответствует набор весов w_j :

$$a_j(x) = \text{sign}(\langle w_j, x \rangle)$$

Итоговый классификатор будет выдавать класс, соответствующий самому уверенному алгоритму:

$$a(x) = \arg \max_{j \in \{1, \dots, K\}} \langle w_j, x \rangle$$

- Каждый против каждого (all-vs-all)

Обучается C_k^2 классификаторов $a_{sj}(x)$, $s, j \in \{1, \dots, k\}$, $s < j$. Алгоритм $a_{sj}(x)$ обучается по выборке X_{sj} :

$$X_{sj} = \{(x_i, y_i) \mid y_i = s \text{ или } y_i = j, (x_i, y_i) \in X\}$$

Таким образом, каждому классификатору $a_{sj}(x)$ соответствует набор весов w_{sj} :

$$a_{sj}(x) = \text{sign}(\langle w_{sj}, x \rangle)$$

Итоговый классификатор будет выдавать класс, который наберёт больше всего голосов построенных алгоритмов:

$$a(x) = \arg \max_{k \in \{1, \dots, K\}} \sum_{s=1}^K \sum_{j \neq s} \mathbb{I}[a_{sj} = k]$$

Существует прямое обобщение логистической регрессии на случай многих классов — мультиномиальная регрессия. Пусть построено K линейных моделей $a_1(x), \dots, a_K(x)$, $a_j(x) = \text{sign}(\langle w_j, x \rangle)$. Каждая модель даёт оценку принадлежности объекта к определённом классу. Эти оценки можно перевести в вероятности с помощью функции $\text{softmax}(z_1, \dots, z_K)$, которая переводит произвольный вещественный вектор в дискретное вероятностное распределение:

$$\text{softmax}(z_1, \dots, z_K) = \left(\frac{\exp(z_1)}{\sum_{k=1}^K \exp(z_k)}, \dots, \frac{\exp(z_K)}{\sum_{k=1}^K \exp(z_k)} \right)$$

Вероятность k -ого класса можно выразить так:

$$P(y = j|x) = \frac{\exp(\langle w_j, x \rangle)}{\sum_{k=1}^K \exp(\langle w_k, x \rangle)}$$

Обучение производится с помощью метода максимального правдоподобия:

$$Q(X, w) = -\frac{1}{N} \sum_{i=1}^N \log P(y_i|x_i) + \frac{\lambda}{2} \sum_{k=1}^K \|w_k\|_2^2 \rightarrow \min_{w_1, \dots, w_K}$$

Стохастический градиентный спуск

Для минимизации функционала $Q(X, w)$ можно использовать метод градиентного спуска. В этом методе выбирается начальное приближение для вектора весов w , затем запускается итерационный процесс, на каждом шаге которого вектор w изменяется в направлении антиградиента функционала $Q(X, w)$:

$$w^{(k+1)} = w^{(k)} - \eta_k \nabla_w Q(X, w) = w^{(k)} - \frac{1}{N} \eta_k \sum_{i=1}^N \nabla_w \mathcal{L}(M_i(w))$$

Параметр $\eta_k > 0$ — темп обучения (learning rate), который может равняться константе, а может, например, монотонно уменьшаться с течением итераций. В задании вам предлагается использовать формулу:

$$\eta_k = \frac{\alpha}{k^\beta}, \quad \text{где } \alpha, \beta — \text{заданные константы}$$

Остановку алгоритма предлагается производить при малом изменении вектора весов или после заданного числа итераций.

Функционал $Q(X, w)$ представляет собой сумму функций потерь на каждом объекте. Вычислять градиент на каждой итерации при большом числе объектов может быть очень трудоёмко. Можно оценить градиент суммы градиентом одного слагаемого, на каждой итерации слагаемое выбирается случайно:

$$i \sim \text{unif}\{N\}$$

$$w^{(k+1)} = w^{(k)} - \eta_k \nabla_w \mathcal{L}(M_i(w))$$

Такой метод называется методом стохастического градиентного спуска. На каждой итерации можно выбирать не один объект, а небольшое подмножество (mini-batch), что ускорит сходимость алгоритма. Преимуществом алгоритма помимо маленькой вычислительной сложности является то, что на каждом шаге необходимо держать в память лишь один объект из обучающей выборки. Обратите внимание, что в случае стохастического градиентного спуска, темп обучения следует изменять только раз за эпоху (пробег метода оптимизации по всем объектам выборки).

Практический трюк. Для каждой эпохи необходимо сгенерировать случайную перестановку индексов всех объектов. Далее, на каждой итерации выбирать следующее подмножество индексов. Такой трюк ускоряет работу алгоритма.

Разностная проверка градиента

При написании собственной реализации линейной модели возникает необходимость проверить правильность её работы. Проверить правильность реализации подсчета градиента можно с помощью конечных разностей:

$$[\nabla f(w)]_i \approx \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon}$$

e_i — базисный вектор, $e_i = [0, 0, \dots, 0, 1, 0, \dots, 0]$, ε — небольшое положительное число.

Метрика качества

Данные, предложенные для анализа в задании, являются несбалансированными: объектов класса 1 сильно меньше по количеству чем объектов класса -1. В такой ситуации не совсем правильно использовать в качестве метрики качества стандартную точность (ассигасу), определяющуюся:

$$Q(y^{true}, y^{pred}) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[y_i^{pred} = y_i^{true}]$$

В выражении выше y_{true} — истинные метки, y_{pred} — предсказанные метки, N — количество объектов.

В качестве метрики качества в задании предлагается использовать сбалансированную точность (balanced assigasy). Сбалансированная точность для классификации с множеством классов Y определяется следующим образом:

$$Q_{balanced}(y^{true}, y^{pred}) = \frac{1}{|Y|} \sum_{y \in Y} \frac{\sum_{i=1}^N \mathbb{I}[y_i^{true} = y] \mathbb{I}[y_i^{pred} = y]}{\sum_{i=1}^N \mathbb{I}[y_i^{true} = y]}$$

Обратите внимание, что если мощности классов для всех $y \in Y$ равны, то выражение выше будет эквивалентно стандартной точности:

$$\begin{aligned} Q_{balanced}(y^{true}, y^{pred}) &= \frac{1}{|Y|} \sum_{y \in Y} \frac{\sum_{i=1}^N \mathbb{I}[y_i^{true} = y] \mathbb{I}[y_i^{pred} = y]}{N/|Y|} = \\ &= \frac{1}{N} \sum_{i=1}^N \sum_{y \in Y} \mathbb{I}[y_i^{true} = y] \mathbb{I}[y_i^{pred} = y] = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[y_i^{true} = y_i^{pred}] \end{aligned}$$