AIMasters

Спецификация к заданию «Классификация заемщиков линейными моделями»

курс «Машинное обучение 1», 2022

Везде выборкой объектов будем понимать `numpy.ndarray` размера $N \times D$, под ответами для объектов выборки будем понимать `numpy.ndarray` размера N, где N — количество объектов в выборке, D — размер признакового пространства.

Требования к реализации

Среди предоставленных файлов должны быть следующие модули и функции в них:

1. Модуль `losses.py` с реализацией функции потерь и ее градиента.

   Обратите внимание на то, что подсчёт всех функций может быть полностью векторизован (т.е. их можно реализовать без циклов). Функция потерь также должна поддерживать использование l2-регуляризации в виде слагаемого $\lambda \cdot \|weights\|_2^2$ в итоговой функции потерь. Обратите внимание, что признак для смещения (`bias`) не должен учитываться в регуляризаторе.

   Класс функции потерь наследуется от абстрактного класса `BaseLoss` и реализует два метода: `func` и `grad`.

   (a) `func(self, X, y, w)` — вычисление значения функции потерь на матрице признаков `X`, векторе ответов `y` с вектором весов `w`.

   (b) `grad(self, X, y, w)` — вычисление значения градиента функции потерь на матрице признаков `X`, векторе ответов `y` с вектором весов `w`.

   У обоих методов одинаковые аргументы:

   - `X` - выборка объектов
   - `y` - вектор ответов
   - `w` - вектор коэффициентов модели, одномерный `numpy.ndarray`.
     Вектор коэффициентов имеет вид: `w = [bias, weights]`, то есть нулевой элемент `w` - `bias`, остальное - веса, участвующие в скалярном произведении.

   В данном задании предлагается реализовать следующую функцию потерь:

   - `BinaryLogisticLoss` — функция потерь для бинарной логистической регрессии

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

2. Модуль `linear_model.py` с реализацией линейной модели, поддерживающей обучение через полный и стохастический градиентные спуски. Линейная модель должна задаваться в классе `LinearModel`. Параметр $\eta_k > 0$ — темп обучения (learning rate) для градиентного спуска, где k — номер эпохи, должен параметризовываться формулой:

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

   Обратите внимание, что пересчитывать темп обучения для k-й эпохи следует ДО обновления весов на этой же эпохе.

   Описание методов класса:

   (a) `__init__` — конструктор (инициализатор) класса с параметрами:
   - `loss_function` — функция потерь, заданная классом, наследованным от BaseLoss
   - `batch_size` — размер подвыборки, по которой считается градиент, если None, то необходимо использовать полный градиент
   - `step_alpha` — параметр выбора шага градиентного спуска
   - `step_beta` — параметр выбора шага градиентного спуска
   - `tolerance` — точность, по достижении которой, необходимо прекратить оптимизацию
   - `max_iter` — максимальное число итераций (в случае стохастического спуска - эпох)

(b) `fit(self, X, y, w_0=None, trace=False, X_val=None, y_val=None)` — обучение линейной модели

- `X` — выборка объектов
- `y` — вектор ответов
- `w_0` — начальное приближение вектора коэффициентов, если None, то необходимо инициализировать внутри метода. `w_0` имеет вид `[bias_0, weights_0]`.
- `trace` — индикатор, нужно ли возвращать информацию об обучении
- `X_val` — валидационная выборка
- `y_val` — вектор ответов для валидации

Метод поддерживает два варианта градиентного спуска - полный `(batch_size=None)` и стохастический. Стохастический градиентный спуск состоит из эпох (максимальное количество эпох - `max_iter`), а эпохи - из итераций. Темп обучения обновляется раз в эпоху. Веса обновляются каждую итерацию по градиенту, посчитанному на случайном батче размера `batch_size`, состоящем из объектов, которые еще не участвовали в подсчете градиента в этой эпохе. Когда все объекты по разу поучаствовали в подсчете градиента, эпоха заканчивается.
В случае полного градиентного спуска эпоха и итерация - одно и то же.

Если `trace=True`, то метод должен вернуть словарь `history`, содержащий информацию о поведении метода оптимизации во время обучении. Длина словаря `history` — количество эпох.

Элементы словаря в случае полного градиентного спуска:

- `history['time']` — содержит время потраченное на обучение каждой эпохи
- `history['func']` — содержит значения функционала на обучающей выборке на каждой эпохе
- `history['func_val']` — содержит значения функционала на валидационной выборке на каждой эпохе

Обратите внимание, что `trace=True` замедляет обучение методов, т.к. требует в конце эпохи подсчитывать значение функции на валидации. Не используйте его ни в каких экспериментах, кроме экспериментов, где необходимо исследовать поведение функции в зависимости от гиперпараметров.

Критерий останова метода — модуль разности значений функции потерь на соседних эпохах метода меньше `tolerance`.

(c) `predict(self, X, threshold=0)` — получение предсказаний модели

- `X` — выборка объектов
- `threshold` — порог бинаризации классов

Метод должен вернуть `numpy.ndarray` такого же размера, как и первая размерность матрицы `X`.

(d) `get_optimal_threshold(self, X, y)` — получение оптимального порога для бинаризации выходов модели

- `X` — выборка объектов
- `y` - вектор ответов

(e) `get_objective(self, X, y)` — вычисление значения функции потерь

- `X` - выборка объектов
- `y` - вектор ответов

Функция должна вернуть вещественное число.

(f) `get_weights(self)` — получить вектор линейных коэффициентов модели

(g) `get_bias(self)` — получить `bias` модели

3. Модуль `utils.py` с реализацией функции численного подсчёта градиента произвольного функционала.

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

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

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

В модуле должна быть реализована функция:

(a) `get_numeric_grad(f, x, eps)` — функция проверки градиента

- `f` — функция, возвращающая по вектору число
- `x` — вектор, подходящий для вычисления функции `f`, заданный в `numpy.ndarray`
- `eps` — число из формулы выше

Функция должна вернуть вектор численного градиента в точке x.

Замечание. Для всех функций можно задать аргументы по умолчанию, которые будут удобны вам в вашем эксперименте. Ко всем функция можно добавлять необязательные аргументы, а в словарь `history` разрешается сохранять необходимую в ваших экспериментах информацию.

Полезные советы по реализации

1. В промежуточных вычислениях стоит избегать вычисления $\exp(-b_i\langle x_i, w\rangle)$, иначе может произойти переполнение. Вместо этого следует напрямую вычислять необходимые величины с помощью специализированных для этого функций: `np.logaddexp`, `scipy.special.logsumexp` и `scipy.special.expit`. В ситуации, когда вычисления экспоненты обойти не удаётся, можно воспользоваться процедурой «клипинга» (функция `numpy.clip`).

2. Нет необходимости проводить честное семплирование для каждого батча в методе стохасического градиентного спуска. Вместо этого предлагается в начале одной эпохи сгенерировать случайную перестановку индексов объектов, а затем последовательно выбирать объекты для нового батча из элементов этой перестановки.

3. Функцию вычисления численного градиента можно использовать и для функций от двумерных входов. Достаточно написать обёртку, которая принимает на вход вектор, конструирует по нему матрицу и вычисляет значение функции.

4. Посчитав `grad_bias, grad_weights`, удобно соединить их в один `np.ndarray` для дальнейшего использования можно так: `np.r_[grad_bias, grad_weights]`