OzonMasters

Спецификация к заданию «Применение линейных моделей для определения токсичности комментария»

курс «Машинное обучение 1», 2021

Везде выборкой объектов будем понимать numpy.ndarray размера $N \times D$ или разреженную матрицу scipy.sparse.csr_matrix того же размера, под ответами для объектов выборки будем понимать numpy.ndarray размера N, где N — количество объектов в выборке, D — размер признакового пространства. Подрузамевается, что первый столбец выборки объектов соответствует признаку для смещения и равен единице.

Требования к реализации

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

1. Модуль losses.py с реализациями функций потерь и их градиентов.

   Каждый класс в этом модуле задаёт конкретную функцию потерь, которую можно использовать для обучения линейной модели. Обратите внимание на то, что подсчёт всех функций может быть полностью векторизован (т.е. их можно реализовать без циклов). Все предложенные в задании функции потерь также должны поддерживать использование l2-регуляризации. Обратите внимание, что признак для смещения не должен учитываться в регуляризаторе.

   Классы должны поддерживать как плотные матрицы (numpy.ndarray), так и разреженные матрицы (scipy.sparse.csr_matrix). Каждый класс 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 аргменты:

   - X - выборка объектов
   - y - вектор ответов
   - w - вектор коэффициентов модели, одномерный numpy.ndarray для одноклассовой классификации, двумерный numpy.ndarray для многоклассовой классификации.

   В данном задании предлагается реализовать следующие функции потерь:

   - BinaryLogisticLoss — функция потерь для бинарной логистической регрессии

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

   - MultinomialLoss — функция потерь мультиномиальной регрессии (бонусная часть)

     $$p(y|x) = \mathrm{softmax}_{y \in Y}(a_y(x)) = \frac{\exp(a_y(x))}{\sum_{u \in Y} \exp(a_u(x))},\ Y = \{1, \ldots, K\},\ a_y(x) \in (-\infty, \infty)$$

     $$L(a(x), y) = -\log p(y|x),\ a(x) = \{a_y(x)\}_{y \in Y},\ y \in Y$$

2. Модуль linear_model.py с реализацией линейной модели, поддерживающей обучение через полный и

сто хастический 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 параметри зовываться формулой:

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

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

(b) fit(self, X, y, w_0=None, trace=False) — обучение линейной модели
- X — выборка объектов
- y — вектор ответов
- w_0 — начальное приближение вектора весов, если None, то необходимо инициализировать внутри метода
- trace — индикатор, нужно ли возвращать информацию об обучении

Если trace is 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 is True сильно замедляет обучение методов, т.к. требует в конце эпохи подсчитывать значение функции. Не используйте его ни в каких экспериментах, кроме экспериментов, где необходимо исследовать поведение функции в зависимости от гиперпараметров. Критерий останова метода — l2 норма разницы весов на соседних итерациях метода. (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

3. Модуль utils.py с реализацией функции численного подсчёта градиента произвольного функционала. При написании собственной реализации линейной модели возникает необходимость проверить правиль ность 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, $e_i = [0, 0, \ldots, 0, 1, 0, \ldots, 0]$, 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 специализиро ванных для этого функций: np.logaddexp, scipy.special.logsumexp и scipy.special.expit. В ситуации, когда вычисления экспоненты обойти не удаётся, можно воспользоваться процедурой «клипинга» (функ ция numpy.clip).

2. При вычислении нормировки $\frac{\exp(\alpha_i)}{\sum_k \exp(\alpha_k)}$ может произойти деление на очень маленькое число, близкое к нулю. Необходимо воспользоваться следующим трюком:

$$\frac{\exp(\alpha_i)}{\sum_k \exp(\alpha_k)} = \frac{\exp(\alpha_i - \max_j \alpha_j)}{\sum_k \exp(\alpha_k - \max_j \alpha_j)}$$

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

4. 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 вы числяет значение функции.