Краткий конспект лекции «Линейные модели для классификации»

Попов Артём Сергеевич
курс «Практикум на ЭВМ» ММП ВМК МГУ

2 ноября 2018 г.

Линейная модель бинарной классификации

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

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

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

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

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 $\mathbb{I}[M(w) < 0] \leqslant \mathcal{L}(M(w))$. Будем настраивать вектор весов w, оптимизируя функционал $Q(X, w)$:

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

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:

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

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:

  $$X_j = (x_i, 2\mathbb{I}[y_i = j] - 1)_{i=1}^l$$

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

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

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}:

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

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

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

  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 $a_1(x), \dots, a_K(x)$, $a_j(x) = \text{sign}\left(\langle w_j, x \rangle\right)$. Каждая модель даёт оценку принадлежности объекта к определённому 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:

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

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 $Q(X, w)$:

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

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:

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

Остановка алгоритма может происходить при слишком малом изменении функционала или нормы вектора весов или после заданного числа итераций.

Функционал $Q(X, w)$ представляет собой сумму функций потерь на каждому 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:

$i \sim unif\{l\}$

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

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.

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:

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

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

Рекомендации

- Как сравнивать работу градиентного спуска и стохастического градиентного спуска?

  1 итерация GD == 1 эпоха SGD, поэтому сравнивать итерации некорректно! Сравнивать нужно либо по времени, либо по эпохам.

- Вычисление softmax

  Замечание. В промежуточных вычислениях стоит избегать вычисления значение $\exp(-b_i\langle x_i, w\rangle)$, 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`).

  Замечание. 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:

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