Online Convex Optimization (OCO)

Михаил Лепехин и Роман Логинов, группа 694

24 декабря 2018 г.

Основные понятия и концепции OCO

В online выпуклой оптимизации, в отличие от обычной, изученной в курсе, нет фиксированной функции, которую надо минимизировать. Более того, процесс построения оптимального решения делится на несколько итераций. На каждой итерации известны те данные, которые были на предыдущих, а также требуемые для минимизации функции не предыдущих шагах.

Модель напоминает ту, что используется в теории игр. Итерационный процесс позволяет моделировать при помощи OCO процессы, происходящие в действительности.

По шагам это происходит так:

1. Игрок (оптимизатор) делает некоторый ход, выдавая вектор $x_t \in \mathcal{K}$
2. Система (реальный мир) выбирает функцию $f_t \in \mathcal{F}$
3. На этом шаге вычисляется функция потерь $f_t(x_t)$

Таким образом, на каждой итерации игрок терпит некоторые потери, однако узнаёт новую информацию о функциях. Пусть так будет происходить в течение T итераций.

Но в каком случае это вообще имеет смысл?

Во-первых, хочется, чтобы функции были не совсем произвольными, а **выпуклыми**, поскольку для их оптимизации существует множество известных алгоритмов.

Во-вторых, на каждом шаге сейчас функция выбирается произвольно и произвольного семейства. Поэтому если оставлять задачу в таком виде, то на каждом шаге система может выбирать неограниченную функцию! И тогда, если на первом шаге функция потерь получилась положительной, то, увеличивая и увеличивая её в дальнейшем, система может не позволить игроку восстановиться от потерь на первой итерации (когда

ничего не известно). Таким образом, значение функции потерь должно быть **ограниченным**.

Наконец, хочется, чтобы **ограниченным** было и множество $\mathcal{K}$. Конечно, оно может быть не обязательно конечным. Но в случае неограниченного пространства решений система может сопоставлять нулевую потерю для тех значений, которые игрок не выбирает, и огромные в противном случае. А вот, например, шар вполне подходит под такие ограничения (будет использован в дальнейшем)

Теперь формально опишем, чего мы хотим добиться. На каждом шаге мы меняем решение. Но приблизиться мы хотим к оптимальной функции потерь, которая была бы, если бы решение было фиксировано с самого начала. Поэтому для алгоритма $\mathcal{A}$ вводится величина:

$$regret_T(\mathcal{A}) = \sup_{\{f_1,\dots,f_T\}\subseteq\mathcal{F}} \left\{ \sum_{t=1}^{T} f_t(x_t) - \min_{x\in\mathcal{K}} \sum_{t=1}^{T} f_t(x) \right\}$$

Таким образом, отыскав оптимизирующие $regret$ алгоритмы, мы сможем решать задачи с обновляющейся функцией потерь. Среди примеров обучение на основании советов экспертов, задачи оптимального инвестирования на бирже (portfolio selection) и больше всего приближенная к IT задача фильтрации спама, на примере которой и будут рассмотрены алгоритмы OCO.

Применение Online Convex Optimisation к задаче фильтрации спама

Предположим, что признаки email-сообщений принадлежат множеству $\mathcal{X}$. В качестве признаков будем рассматривать частоты вхождений слов (или групп слов, чтобы размерность мн-ва признаков не получилась слишком большой) в сообщение.

На каждом шаге t функция $a_t : \mathcal{X} \to [0, 1]$, сопоставляет вектору $x \in \mathcal{X}$ значений признаков некоторое число из отрезка $[0, 1]$. По смыслу это значение является оценкой вероятности (уверенности) того, что сообщение с данными значениями признаков является спамом.

На каждом шаге t соперник выбирает вектор значений признаков x_t и индикатор y_t того, что данное сообщение является спамом.

Для оценки точности метода принятия решений a_t нужно взять некоторую функцию потерь f_t. Например, квадратичную функцию потерь:

$$f_t(a_t) := (y_t - a_t(x_t))^2.$$

На каждом шаге функция $a_t(x)$ выбирается из некоторого множества так, чтобы минимизировать $regret$:

$$\sum_{t=1}^{T} f_t(a_t) - \min_{a \in \mathcal{A}} \sum_{t=1}^{T} f_t(a) = \sum_{t=1}^{T} (y_t - a_t(x_t))^2 - \min_{a \in \mathcal{A}} \sum_{t=1}^{T} (y_t - a(x_t))^2$$

Выбор функции $a_t(x)$

В машинном обучении для решения задачи классификации спама часто делают следующее. При помощи некоторого алгоритма находят вектор фильтра a из шара $B_R(0)$ относительно некоторой нормы. А после - для определения, является ли сообщение с вектором значений признаков x спамом, рассматривают скалярное произведение $< a, x >$.

Если $< a, x > > 0$, то сообщение является спамом. Если же знак скалярного произведения отрицательный, то сообщение не является спамом. А если получилось так, что скалярное произведение равно 0, то считается, что тип сообщения не определён.

Будем строить функцию a_t из похожих соображений. Будем также подбирать вектор фильтра w_t из $W := B_R(0)$ и большим значениям скалярного произведения $< w_t, x >$ будет сопоставлять большую вероятность.

В качестве $\mathcal{X}$ возьмём множество векторов x из ${\mathbb{R}_+}^d$, что $\sum\limits_{i=1}^{n} x_i = 100$ (здесь каждой группе слов сопоставляется процент количества слов из этой группы по отношению ко всем словам в сообщении).

Покажем, что скалярного произведение $< w_t, x >$ ограничено. По неравенству Коши-Буняковского:

$$< w_t, x >^2 \leq ||w_t||_2^2 * ||x||_2^2 \leq R^2 * ||x||_2^2 \leq R^2 * 100^2.$$

Причём, равенство здесь достигается, если сразу выполняются 3 ограничения:

1) x коллинеарен w_t - получим равенство в нер-ве К-Б,

2) $w_t = R$ - получим 2 равенство,
3) $\exists i \in \{1, \dots, d\} : x_i = 100$.
Тогда определим $M := 100R$.
В качестве функции $a_t(x)$ возьмём

$$a_t(x) = \frac{< x, w_t > +M}{2M}.$$

Тогда функция f_t запишется следующим образом:

$$f_t(x) = \left(y_t - \frac{< x, w_t > +M}{2M}\right)^2$$

Свойства выбранной функции $a_t(x)$

Для нас очень важным свойством будет являться то, что выбранная функция $a_t(x)$ выпукла. Покажем это.

Вычислим её градиент.

$$\frac{\partial f_t}{\partial x}(x) = \frac{2}{2M}(< x, w_t > +M)\frac{w_t}{2M} = \frac{< x, w_t > +M}{4M^2}w_t$$

Продифференцируем градиент по x и получим гессиан.

$$\frac{\partial^2 f_t}{\partial x^2}(x) = \frac{1}{4M^2}w_t w_t^T \succeq 0$$

Положительная полуопределённость следует из того, что $\forall x \in \mathcal{X}$: $x^T w_t w_t^T x_t = (w_t^T x_t)^T w_t^T x_t =< w_t^T x_t, w_t^T x_t > \geq 0$ - по свойствам скалярного произведения.

По дифференциальному критерию выпуклости 2 порядка функция $f_t(x)$ выпукла.

Вычисление regret

Для того, чтобы проверять качество работы методов, очень полезно уметь получать значение *regret*.

С учётом выбора функции $a_t(x)$ *regret* можно записать следующим образом:

$$\sum_{t=1}^{T}(y_t - a_t(x_t))^2 - \min_{a\in\mathcal{A}}\sum_{t=1}^{T}(y_t - a(x_t))^2 =$$

$$= \sum_{t=1}^{T}\left(y_t - \frac{< x_t, w_t > +M}{2M}\right)^2 - \min_{w\in W}\sum_{t=1}^{T}\left(y_t - \frac{< x_t, w > +M}{2M}\right)^2$$

При этом заметим, что $\forall t = 1,\dots,T : g_t(w) = \left(y_t - \frac{<x_t,w>+M}{2M}\right)^2$ является выпуклой по w (доказательство аналогично выпуклости f_t по x). Значит, функция $g(w) = \sum\limits_{t=1}^{T}\left(y_t - \frac{<x_t,w>+M}{2M}\right)^2$ является выпуклой как сумма выпуклых функций.

Чтобы получить точное или приближённое значение *regret* нужно точно или приближённо решить следующую задачу оптимизации:

$$\min g(w)$$

$$s.t. w \in W$$

Эта задача является выпуклой, поскольку:

1) $g(w)$ выпукла в $\mathbb{R}^d$, как было показано выше;

2) множество $W = B_0(R)$ выпукло, поскольку любые 2 точки, лежащие в шаре можно соединить отрезком, каждая точка которого будет также принадлежать этому шару.

Поэтому для получения точного значения *regret* можно применить теорему Каруша-Куна-Таккера. В силу выпуклости задачи оптимизации стационарные точки лагранжиана будут точками минимума функции.

Но нам вполне хватит и приближённого означения *regret*, поэтому для решения вспомогательной задачи оптимизации воспользуемся пакетом *cvxpy*.

Методы первого порядка

В данном разделе мы рассмотрим базовые алгоритмы для Online Convex Optimization, которые достаточно неплохо применимы на практике.

В целом данные методы похожи на соответствующие методы первого порядка для задач обычной выпуклой оптимизации. Но они принципиально отличаются целью применения. Ведь при помощи методов OCO мы стремимся минимизировать не ошибку оптимизации, а *regret*:

$$regret = \sum_{t=1}^{T} f_t(x_t) - \min_{x \in \mathcal{K}} \sum_{t=1}^{T} f_t(x)$$

Для сравнения $regret$ с ошибкой оптимизации полезно рассмотреть среднее значение $regret$, т.е. $\frac{regret}{T}$.

Введём обозначение:

$$\overline{x}_T := \frac{1}{T} \sum_{t=1}^{T} x_t$$

Пусть все функции f_t равны некоторой функции $f : \mathcal{K} \to \mathbb{R}$, то из неравенства Йенсена получим:

$$f(\overline{x}_T) - f(x^*) = f(\overline{x}_T) - \frac{1}{T} \sum_{t=1}^{T} f(x^*) \leq \frac{1}{T} \sum_{t=1}^{T} \big(f(x_t) - f(x^*)\big)$$

Таким образом мы показали следующий факт:

функция $f(x_T)$ сходится к $f(x*)$ не менее быстро, чем среднее значение $regret$.

Online gradient descent

Этот алгоритм, пожалуй, является одним из наиболее интуитивных и простых в Online Convex Optimization. Он базируется на известном нам методе градиентного спуска для offline выпуклой оптимизации.

На каждой итерации этот алгоритм делает шаг от предыдущей точки x_k в направлении градиента предыдущего веса. Но такой шаг может привести к выходу за границу допустимого выпуклого множества D. Для того, чтобы этого не произошло, алгоритм проецирует полученную точку обратно на множество D, находя ближайшую к ней в D.

Несмотря на то, что функция весов на следующем шаге может существенно отличаться от веса на предыдущем шаге, $regret$, получаемый алгоритмом все равно будет сублинейным.

Это следует из следующей теоремы.

Теорема. Online градиентный спуск с шагом, заданным по правилу $\alpha_t = \frac{D}{G\sqrt{t}}$, для любого $T \geq 1$ гарантирует:

$$regret = \sum_{t=1}^{T} f_t(x_t) - \min_{x \in \mathcal{K}} \sum_{t=1}^{T} f_t(x) \leq \frac{3}{2} GD\sqrt{T}$$

Кроме того, при выборе шага по данному правилу метод online градиентного спуска является асимптотически оптимальным по значению $regret$.

Теорема. Любой алгоритм для OCO в худшем случае выдаёт $regret = \Omega(DG\sqrt{T})$. Это утверждение верно даже при выборе функции веса из некоторого фиксированного распределения.

Stochastic gradient descent

Метод Ньютона для OCO

Экспоненциально вогнутые функции

Первоначально рассмотрим новый класс функций, не упоминавшийся в курсе - экспоненциально вогнутые (exp-concave) функции. Как мы покажем впоследствии, этот класс более широк, чем сильно выпуклые функции, а значит, когда мы предъявим алгоритм, работающий на этом классе функций, его возможности должны быть шире.

Например, теоремы сходимости для онлайн градиентного спуска показаны только для свойства сильной выпуклости. А в нашем случае такого свойства у функции нет. Также показывается, что аналогичная проблема появляется и при решении задачи protfolio selection.

Определение. Экспоненциально вогнутая с константой α функция - функция f, для которой $g = e^{-\alpha f(x)}$ - вогнутая функция.

Заметим, что в случае сильной выпуклости есть критерий второго порядка:

$$\nabla^2 f \succcurlyeq l\mathbf{I}$$

Здесь же суть заключается в том, что гессиан тоже велик, но только в направлении градиента. Покажем критерий второго порядка и в данном случае:

Лемма. *Дважды дифференцируемая функция f экспоненциально вогнута с константой $\alpha \iff \nabla^2 f(x) \succcurlyeq \alpha \nabla f(x) \nabla f(x)^\top$*

Доказательство. Покажем вогнутость через критерий второго порядка. Для этого вычислим гессиан и градиент.

$$\frac{\partial g}{\partial x_i} = -\frac{\partial f}{\partial x_i} \alpha e^{-\alpha f(x)}$$

$$\frac{\partial^2 g}{\partial x_i \partial x_j} = \frac{\partial \left(-\frac{\partial f}{\partial x_i} \alpha e^{-\alpha f(x)} \right)}{\partial x_j} = -\alpha \left[\frac{\partial^2 f}{\partial x_i \partial x_j} e^{-\alpha f(x)} - \alpha \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_i} e^{-\alpha f(x)} \right]$$

Сократив на $\alpha e^{-\alpha f(x)}$, получим, что эта матрица является неположительно определённой $\Leftrightarrow$

$$\nabla^2 f(x) \succcurlyeq \alpha \nabla f(x) \nabla f(x)^\top$$

□

Отсюда становится ясно, что сильно выпуклая функция будет и экспоненциально вогнутой. Во-первых, это ясно из интуиции определений. Экспоненциальная вогнутость означает сильную выпуклость лишь в направлении градиента, а значит этот класс функций получается шире. Также это видно и из формулы в лемме. Если $\nabla^2 f \succcurlyeq l\mathbf{I}$, то найдётся α, что $\alpha \nabla f \nabla f^\top \preccurlyeq l\mathbf{I}$

Также можно доказать более мощную лемму, которая используется для построения алгоритма:

Обозначим D - диаметр множества $\mathcal{K}$, а G - граница норм градиентов функции f

Лемма. *Пусть $f : \mathcal{K} \to \mathbb{R}$ - экспоненциально вогнутая функция с константой α. Тогда для $\gamma < \frac{1}{2} \min\{\frac{1}{4GD}, \alpha\}$ и для всех $x, y \in \mathcal{K}$*

$$f(x) \geqslant f(y) + \nabla f(y)^\top (x - y) + \frac{\gamma}{2} (x - y)^\top \nabla f(y) \nabla f(y)^\top (x - y)$$

Алгоритм

Сам алгоритм будет основан скорее не на методах второго порядка, а всё же на методах первого. Чаще всего в ОСО стандартный алгоритм Ньютона, учитывающий гессиан, не применятся, а используется идея квазиньютоновских методов. На каждой итерации будет использоваться шаг квазиньютоновского алгоритма с матрицей A_t, а градиент - это градиент предыдущей функции потерь (∇_t), вычисленный в сыгранной точке x_t.

Для обновления A_t используют следующее правило:

$$A_t = A_{t-1} + \nabla_t \nabla_t^\top$$

Тогда можно непосредственно проверить, что:

$$(A + xx^\top)^{-1} = A^{-1} - \frac{A^{-1}xx^\top A^{-1}}{1 + x^\top A^{-1}x}$$

По индукции ясно, что если взять на изначальном шаге матрицу $A_0 = \varepsilon \mathbf{I}$, то она будет симметричной и неотрицательно определенной на любом шаге как сумма симметричных неотрицательно определенных матриц. Таким образом, обратную матрицу можно отыскать на каждой итерации за квадратичную сложность.

Но шаг метода может вывести нас за пределы $\mathcal{K}$, поэтому применяется аналогичная методу проекций градиента идея. Берётся проекция на множество после каждого шага, но в данном алгоритме не по евклидовой норме, а по норме, порождённой матрицей A_t.

Таким образом, на каждом шаге решается следующая задача оптимизации, если после шага получилась точка y_t

$$\begin{aligned} &\min (x - y_t)^\top A_t (x - y_t) \\ s.t.\ &x \in \mathcal{K} \end{aligned}$$

В нашем случае требуется искать проекцию на шар с центром в нуле. В евклидовой норме это делать легко, а вот по норме матрицы - не так-то просто. Условия ККТ не дают аналитического решения, поэтому будем это делать приближёнными алгоритмами (тот же метод проекций градиента вполне подойдёт)

Теорема. *В обозначениях из леммы выше, описанный выше алгоритм, запущенный с параметрами* $\varepsilon = \frac{1}{\gamma^2 D^2}$ *и шагом* $\frac{1}{\gamma}$ *при более чем* $4x$ *итерациях гарантирует:*

$$regret_T \leqslant 5(\frac{1}{\alpha} + GD)n \log T$$

Доказательство. Доказательство достаточно громоздкое, его можно найти в указанных в плане источниках. □

Таким образом получили логарифмическую зависимость от числа итераций для экспоненциально-вогнутых функций, что должно быть лучше, чем при использовании онлайн градиентного спуска.

Покажем, какая константа для экспоненциальной вогнутости будет в нашем случае:

Вспомним, как в нашем случае считались гессиан и градиент $f_t(x)$:

$$\nabla^2 f(x) = \frac{1}{4M^2} w_t w_t^\top$$

$$\nabla f(x) = \frac{(x, w_t) + M}{4M} w_t$$

Теперь посмотрим на условие первой леммы этого раздела. Она говорит, что для подходящего α достаточно:

$$\alpha \left(\frac{(x, w_t) + M}{4M} \right)^2 \leqslant \frac{1}{4M^2}$$

$$\alpha((x, w_t) + M)^2 \leqslant 4$$

$$\alpha \leqslant \frac{4}{((x, w_t) + M)^2}$$

Используя рассуждения предыдущих разделов, по неравенству Коши-Буняковского имеем:

$$\alpha \leqslant \frac{4}{4M^2} = \frac{1}{M^2}$$

подходит. А это значит, что алгоритм можно писать, положив $\alpha = \frac{1}{M^2}$

Что показывает, что наша функция экспоненциально вогнута.