Методы Монте–Карло: сэмплинг

Сергей Николенко

Машинное обучение — ИТМО, осень 2006

Outline

Почему проблема

- Пусть у нас есть некоторое вероятностное распределение.
- Как с ним работать? Как, например, его симулировать?
- Мы не всегда можем приблизить (как по методу Лапласа) распределение каким-нибудь известным так, чтобы всё посчитать в явном виде.
- Например, в кластеризации: мультимодальное распределение с кучей параметров, что с ним делать?

Постановка задачи

- Пусть имеется некое распределение $p(x)$.
- Задача 1: научиться генерировать сэмплы $\{x^{(r)}\}_{r=1}^{R}$ по $p(x)$.
- Задача 2: научиться оценивать ожидания функций по распределению $p(x)$, т.е. научиться оценивать интегралы вида

$$E_p[\phi] = \int p(x)\phi(x)dx.$$

Постановка задачи

- Мы будем обычно предполагать, что x — это вектор из $\mathbb{R}^n$ с компонентами x_n, но иногда будем рассматривать дискретные множества значений.
- Функции ϕ — это, например, моменты случайных величин, зависящих от x.
- Например, если $t(x)$ — случайная величина, то её среднее — это $E_p[t(x)]$ ($\int p(x)t(x)dx$), а её вариация равна $E_p[t^2] - (E_p[t])^2$.
- И мы предполагаем, что явно вычислить не получается — слишком сложная функция p.

Ожидания и сэмплинг

- Мы будем заниматься только сэмплингом, потому что задача оценки ожиданий функций легко решится, если мы научимся делать сэмплинг.
- Как она решится?

Ожидания и сэмплинг

- Мы будем заниматься только сэмплингом, потому что задача оценки ожиданий функций легко решится, если мы научимся делать сэмплинг.
- Как она решится?
- Нужно взять сэмплы $\{x^{(r)}\}_{r=1}^{R}$ и подсчитать

$$\hat{\phi} = \frac{1}{R} \sum_{r} \phi(x^{(r)}).$$

- Ожидание $\hat{\phi}$ равно $E_p[\phi]$, а вариация убывает обратно пропорционально R.

Что же сложного в сэмплинге?

- Мы предполагаем, что дана функция $p^*(x)$, которая отличается от $p(x)$ только нормировочной константой $Z = \int p^*(x)dx$: $p(x) = p^*(x)/Z$.
- Почему трудно делать сэмплинг?
- Во-первых, мы обычно не знаем Z; но это не главное.
- Главное — обычно правильные сэмплы p^* часто попадают туда, где p^* велика. А как определить, где она велика, не вычисляя её *везде*?

Дискретизация пространства

- Простейшая идея: давайте дискретизуем пространство, вычислим p^* на каждом участке (пусть она гладкая), потом будем брать дискретные сэмплы, зная все вероятности (это нетрудно).
- Сколько же будет дискретных участков?
- Главная проблема — обычно велика размерность x. Например, если разделить каждую ось на 20 участков, то участков будет 20^n; а n в реальных задачах может достигать нескольких тысяч...
- Иными словами, такой подход никак не работает.

Равномерное сэмплирование

- Может быть, всё-таки получится решить хотя бы вторую задачу?
- Давайте брать сэмплы $\{x^{(r)}\}_{r=1}^{R}$ *равномерно* из всего пространства, затем вычислять там p^* и нормализовать посредством $Z_R = \sum_{r=1}^{R} p^*(x^{(r)})$.
- Тогда $\hat{\phi}$ можно будет оценить как

$$\hat{\phi} = \frac{1}{Z_R} \sum_{r=1}^{R} \phi(x^{(r)}) p^*(x^{(r)}).$$

- В чём проблема?

Равномерное сэмплирование

- Да в том же самом.
- Обычно значительная часть p^* сосредоточена в очень небольшой части пространства.
- Вероятность попасть в неё за R равномерно выбранных сэмплов тоже экспоненциально мала (например, если по каждой оси вероятность попасть $1/2$, и всё независимо, то получится вероятность 2^{-n}).
- Так что даже вторую задачу решить не получится.

Outline

Суть метода

- Выборка по значимости — importance sampling.
- Мы решаем только вторую задачу, а не первую.
- То есть нам нужно брать сэмплы, при этом желательно попадая в зоны, где функция p^* имеет большие значения.

Суть метода

- Предположим, что у нас есть какое-то другое распределение вероятностей q (точнее, q^*), попроще, и мы умеем брать его сэмплы.
- Тогда алгоритм такой: сначала взять сэмпл по q^*, а затем подправить его так, чтобы получился всё-таки сэмпл по p^*.

Формальный алгоритм

- Взять сэмплы $\{x^{(r)}\}_{r=1}^{R}$ по распределению q^*.
- Рассчитать веса
$$w_r = \frac{p^*(x^{(r)})}{q^*(x^{(r)})}.$$
- Оценить функцию по формуле
$$\hat{\phi} = \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r}.$$

Обсуждение

- Зачем нужно q? Чем это лучше равномерного распределения?

Обсуждение

- Зачем нужно q? Чем это лучше равномерного распределения?
- Проще говоря, распределение q должно помочь выбрать те участки, на которых имеет смысл сэмплить r.
- Если q хорошее, то может помочь, а если плохое, может только навредить.
- Но есть и более фундаментальные проблемы.

Проблемы

- Во-первых, сэмплер q не должен быть слишком узким.
- Например, если сэмплер гауссиановский с небольшой вариацией, то пики r далеко от центра q вообще никто не заметит.

Проблемы

- Во-вторых, может случиться, что все сэмплы будут напрочь убиты небольшим количеством сэмплов с огромными весами. Это плохо.
- Чтобы показать, как это бывает, давайте перейдём в многомерный случай.

Проблемы

- Пусть есть равномерное распределение r на единичном шаре и сэмплер q — произведение гауссианов с центром в нуле. Тогда большинство сэмплов q будут лежать в интервале
$$\frac{1}{(2\pi\sigma^2)^{n/2}} 2^{-\frac{N}{2}\pm\frac{\sqrt{2N}}{2}},$$
и ненулевые веса будут иметь значения порядка
$$(2\pi\sigma^2)^{n/2} 2^{\frac{N}{2}\pm\frac{\sqrt{2N}}{2}}.$$
- Это значит, что максимальный вес будет относиться к среднему примерно как $2^{\sqrt{2N}}$, а это очень много.

Заключение

- Если размерностей много, то у выборки по значимости есть две большие проблемы.
- Во-первых, чтобы получить разумные сэмплы, нужно уже заранее выбрать q так, чтобы оно хорошо аппроксимировало p.
- Во-вторых, даже если их получить, часто может так случиться, что веса у некоторых сэмплов будут слишком велики.
- В общем, для случая многих размерностей это не очень хороший метод.

Суть

- Выборка с отклонением — rejection sampling.
- Суть метода в том, что теперь у нас есть q^*, которое мы можем сэмплировать и про которое мы знаем константу c, такую, что
$$\forall x \quad cq^*(x) > p^*(x).$$
- Тогда мы сумеем сэмплировать p.

Алгоритм формально

- Взять сэмпл x по распределению $q^*(x)$.
- Выбрать случайное число u равномерно из интервала $[0, cq^*(x)]$.
- Вычислить $p^*(x)$. Если $u > p^*(x)$, x отклоняется (отсюда и название), иначе добавляется в сэмплы.

Обоснование

- Алгоритм работает, потому что выбирает точки $[x, u]$ равномерно из области под графиком $p^*(x)$, а это и значит, что получатся сэмплы p^*.

Упражнение. Реализовать алгоритмы выборки по значимости и выборки с отклонением в общем виде (чтобы можно было подставлять функции p^* и q^*, хотя бы в коде).

Проблемы

- Как и у предыдущего алгоритма, у выборки с отклонением начинаются проблемы в больших измерениях.
- Суть проблемы та же, что в предыдущем случае, а выражается она в том, что c будет очень большим (экспоненциальным от n), и почти все сэмплы будут отвергаться.

Outline

Общая идея

- Суть алгоритма похожа на выборку с отклонением, но есть важное отличие.
- Распределение q теперь будет меняться со временем, зависеть от текущего состояния алгоритма.
- Как и прежде, нужно распределение q, точнее, семейство $q(x'; x^{(t)})$, где $x^{(t)}$ — текущее состояние.
- Но теперь q не должно быть приближением p, а должно просто быть каким-нибудь сэмплируемым распределением (например, сферический гауссиан).
- Кандидат в новое состояние x' сэмплируется из $q(x'; x^{(t)})$.

Алгоритм

- Очередная итерация начинается с состояния $x^{(i)}$.
- Выбрать x' по распределению $q(x'; x^{(i)})$.
- Вычислить
$$a = \frac{p^*(x')}{p^*(x^{(i)})} \frac{q(x^{(i)}; x')}{q(x'; x^{(i)})}.$$
- С вероятностью a (1, если $a \geq 1$) $x^{(i+1)} := x'$, иначе $x^{(i+1)} := x^{(i)}$.

Обсуждение

- Суть в том, что мы переходим в новый центр распределения, если примем очередной шаг.
- Получается этакий random walk, зависящий от распределения p^*.
- $\frac{q(x^{(i)};x')}{q(x';x^{(i)})}$ для симметричных распределений (гауссиана) равно 1, это просто поправка на асимметрию.
- Отличие от rejection sampling: если не примем, то не просто отбрасываем шаг, а записываем $x^{(i)}$ ещё раз.

Обсуждение

- Очевидно, что $x^{(i)}$ — отнюдь не независимы.
- Независимые сэмплы получаются только с большими интервалами.
- Поскольку это random walk, то если большая часть q сосредоточена в радиусе ϵ, а общий радиус p^* равен D, то для получения независимого сэмпла нужно будет минимум $\left(\frac{D}{\epsilon}\right)^2$ шагов (и это оценка снизу).
- Хорошие новости: это верно для любой размерности. То есть времени надо много, но нет катастрофы при переходе к размерности 1000.

Когда размерность велика

- Когда размерность большая, можно не сразу все переменные изменять по $q(x'; x)$, а выбрать несколько распределений q_j, каждое из которых касается части переменных, и принимать или отвергать изменения по очереди.
- Тогда процесс пойдёт быстрее, чаще принимать изменения будем.

Упражнение. Реализовать алгоритм Метрополиса–Гастингса в общем виде (чтобы можно было подставлять функции p^* и q, хотя бы в коде) и посмотреть на несколько примеров.

Идея

- Пусть размерность большая. Что делать?
- Давайте попробуем выбирать сэмпл не весь сразу, а покомпонентно.
- Тогда наверняка эти одномерные распределения окажутся проще, и сэмпл мы выберем.

На двух переменных

- Пусть есть две координаты: x и y. Начинаем с (x^0, y^0).
- Выбираем x^1 по распределению $p(x|y = y^0)$.
- Выбираем y^1 по распределению $p(y|x = x^1)$.
- Повторяем.

Общая схема

- В общем виде всё то же самое: x_i^{t+1} выбираем по распределению
$$p(x_i|x_1^{t+1},\ldots,x_{i-1}^{t+1},x_{i+1}^{t},\ldots,x_n^{t})$$
и повторяем.
- Это частный случай алгоритма Метрополиса (какие тут распределения q?).
- Поэтому сэмплирование по Гиббсу сходится, и, так как это тот же random walk по сути, верна та же квадратичная оценка.

Обсуждение

- Нужно знать $p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Это, например, особенно легко знать в байесовских сетях.
- Как будет работать сэмплирование по Гиббсу в байесовской сети?
- Для сэмплирования по Гиббсу не нужно никаких особенных предположений или знаний. Можно быстро сделать работающую модель, поэтому это очень популярный алгоритм.
- В больших размерностях может оказаться эффективнее сэмплить по несколько переменных сразу, а не по одной.

BUGS

- Гиббс в байесовских сетях — the BUGS Project.
- http://www.mrc-bsu.cam.ac.uk/bugs/

Упражнение. Реализовать сэмплирование по Гиббсу для декодирования линейного кода (например, (7,4)–кода).

Марковские цепи

- Марковская цепь задаётся начальным распределением вероятностей $p^0(x)$ и вероятностями перехода $T(x';x)$.
- $T(x';x)$ — это распределение следующего элемента цепи в зависимости от следующего; распределение на $(t+1)$–м шаге равно

$$p^{t+1}(x') = \int T(x';x)p^t(x)dx.$$

- В дискретном случае $T(x';x)$ — это матрица вероятностей $p(x'=i|x=j)$.

Свойства марковских цепей: инвариантное распределение

- Не всякая марковская цепь нам подойдёт.
- Во-первых, цепь должна сходиться к распределению, которое нас интересует.
- Это называется *инвариантным распределением*; инвариантное распределение π удовлетворяет

$$\pi(x') = \int T(x'; x)\pi(x)dx.$$

- Нам нужно, чтобы инвариантным распределением нашей цепи было $p(x)$, которое мы хотим сэмплировать.

Свойства марковских цепей: эргодичность

- Ну, и нужно, чтобы собственно сходилось:

$$\forall p^0(x) \quad p^t(x) \longrightarrow \pi(x) \text{ при } t \to \infty.$$

- Какие могут быть примеры неэргодичных цепей?

Из чего делают марковские цепи

- Есть несколько удобных конструкций, с помощью которых можно построить достаточно сложную функцию T, сохраняя её свойства.
- Давайте их рассмотрим.

Из чего делают марковские цепи: конкатенация

- Можно конкатенировать распределения, запуская их друг за другом:

$$T(x', x) = \int T_2(x', x'') T_1(x'', x) dx''.$$

- При этом сохраняется инвариантное распределение (докажите).

Из чего делают марковские цепи: смесь

- Можно смешивать распределения. Если были функции $T_i(x', x)$, то можно ввести новую

$$T(x', x) = \sum_i p_i T_i(x', x), \text{ где } \sum_i p_i = 1.$$

Условие баланса

- Ещё одно полезное свойство:

$$\forall x, y \quad T(x, y)p(y) = T(y, x)p(x).$$

- Т.е. вероятность того, что мы выберем x и дойдём до y равна вероятности выбрать y и дойти до x.
- Такие цепи называются *обратимыми* (reversible).
- Если выполняется условие баланса, то $p(x)$ — инвариантное распределение. Это свойство может пригодиться.

Суть

- Slice sampling — ещё один алгоритм, похожий на алгоритм Метрополиса.
- Применяется в тех же ситуациях, но в нём больше настраиваемых параметров и вообще гибкости.

Алгоритм в одномерном случае

- Мы хотим сделать random walk из одной точки под графиком p^* в другую точку под графиком p^*, да так, чтобы в пределе получилось равномерное распределение.
- Вот как будем делать переход $(x, u) \to (x', u')$:
    - Вычислим $p^*(x)$ и выберем u' равномерно из $[0, p^*(x)]$.
    - Сделаем горизонтальный интервал (x_l, x_r) вокруг x.
    - Затем будем выбирать x' равномерно из (x_l, x_r), пока не попадём под график.
    - Если не попадаем, модифицируем (x_l, x_r).
- Осталось понять, как сделать (x_l, x_r) и как его потом модифицировать.

Дополнения к алгоритму

- Исходный выбор (x_l, x_r):
  - Выбрать r равномерно из $[0, \epsilon]$.
  - $x_l := x - r$, $x_r := x + (\epsilon - r)$.
  - Раздвигать границы на ϵ, пока $p^*(x_l) > u'$ и $p^*(x_r) > u'$.
- Модификация (x_l, x_r): Если x' лежит выше p^*, сокращаем интервал до x'.

Свойства

- В алгоритме Метрополиса нужно было выбирать размер шага. И от него всё зависело квадратично.
- А тут размер шага подправляется сам собой, и эта поправка происходит за линейное время (а то и логарифм).
- В задачах с большой размерностью нужно сначала выбрать (случайно или совпадающими с осями) направление изменения y, а потом проводить алгоритм относительно параметра α в распределении $p^*(x + \alpha y)$.

Домашнее задание

Упражнение. Реализовать алгоритм slice sampling в общем виде (чтобы можно было подставить функцию p^*, хотя бы в коде).

Спасибо за внимание!

- Lecture notes, слайды и коды программ появятся на моей homepage:
  http://logic.pdmi.ras.ru/~sergey/index.php?page=teaching
- Присылайте любые замечания, коды программ на других языках, решения упражнений, новые численные примеры и прочее по адресам:
  sergey@logic.pdmi.ras.ru, smartnik@inbox.ru