Маргинализация на графах

Сергей Николенко

Машинное обучение — ИТМО, осень 2006

Outline

1. Суть и постановка задачи
   - Общий вид функции
2. Алгоритм sum–product
   - Структура графа
   - Сообщения
   - Работа алгоритма
   - Sum–product и разложение функции
3. Min-sum и деревья смежности
   - Min-sum алгоритм
   - Дерево сочленений
4. Метод Лапласа
   - Метод Лапласа в многомерном случае

Введение

- Мы уже много лекций подряд рассматривали задачи маргинализации, потому что они являются основными для байесовского вывода.
- Рассматривали много частных случаев.
- Сегодня мы наконец-то обобщим всё то, чем занимались, до практически самой общей из применимых на практике ситуаций.

Функция в общем виде

- Чтобы поставить задачу в общем виде, рассмотрим функцию

$$p^*(X) = \prod_{j=1}^{m} f_j(X_j),$$

где $X = \{x_i\}_{i=1}^{n}$, $X_j \subseteq X$.

- Т.е. мы рассматриваем функцию, которая раскладывается в произведение нескольких других функций.

Нормализованная функция в общем виде

- Мы написали p^*, потому что обычно, чтобы получилась вероятность, нужно ещё нормализовать:

$$p(X) = \frac{1}{Z} \prod_{j=1}^{m} f_j(X_j),$$

где $Z = \sum_X \prod_{j=1}^{m} f_j(X_j)$.

Пример

$$p^*(X) = f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1, x_2)f_5(x_2, x_3),$$

где

$$f_1(x_1) = \begin{cases} 0.05, & x_1 = 0 \\ 0.95, & x_1 = 1 \end{cases}$$

$$f_2(x_2) = \begin{cases} 0.05, & x_2 = 0 \\ 0.95, & x_2 = 1 \end{cases}$$

$$f_3(x_3) = \begin{cases} 0.95, & x_3 = 0 \\ 0.05, & x_3 = 1 \end{cases}$$

$$f_4(x_1x_2) = \begin{cases} 1, & x_1 = x_2 \\ 0, & x_1 \neq x_2 \end{cases}$$

$$f_5(x_2x_3) = \begin{cases} 1, & x_2 = x_3 \\ 0, & x_2 \neq x_3 \end{cases}$$

- Что это за функция? Вспомните прошлую лекцию.

Пример

- Это функция апостериорной вероятности повторяющего кода, если вероятность ошибки равна 0.05.
- f_4 и f_5 утверждают, что изначально все биты были одинаковые, а f_1, f_2 и f_3 указывают на вероятность ошибки.
- Задачи мы тоже формулировали: это были задачи маргинализации в общем и маргинализации побитовой.
- Давайте их обобщим.

Задачи

- Задача нормализации: найти $Z = \sum_X \prod_{j=1}^{m} f_j(X_j)$.
- Задача маргинализации: найти

$$p_i^*(x_i) = \sum_{k \neq i} p^*(X).$$

- Задача нормализованной маргинализации: найти

$$p(x_i) = \sum_{k \neq i} p(X).$$

- Также может понадобиться, например, $p_{i_1 i_2}$, но реже.

Задачи

- Все эти задачи NP-трудные.
- То есть, если мир не рухнет, сложность их решения в худшем случае возрастает экспоненциально.
- Но можно решить некоторые частные случаи.
- Этим мы и займёмся.

Outline

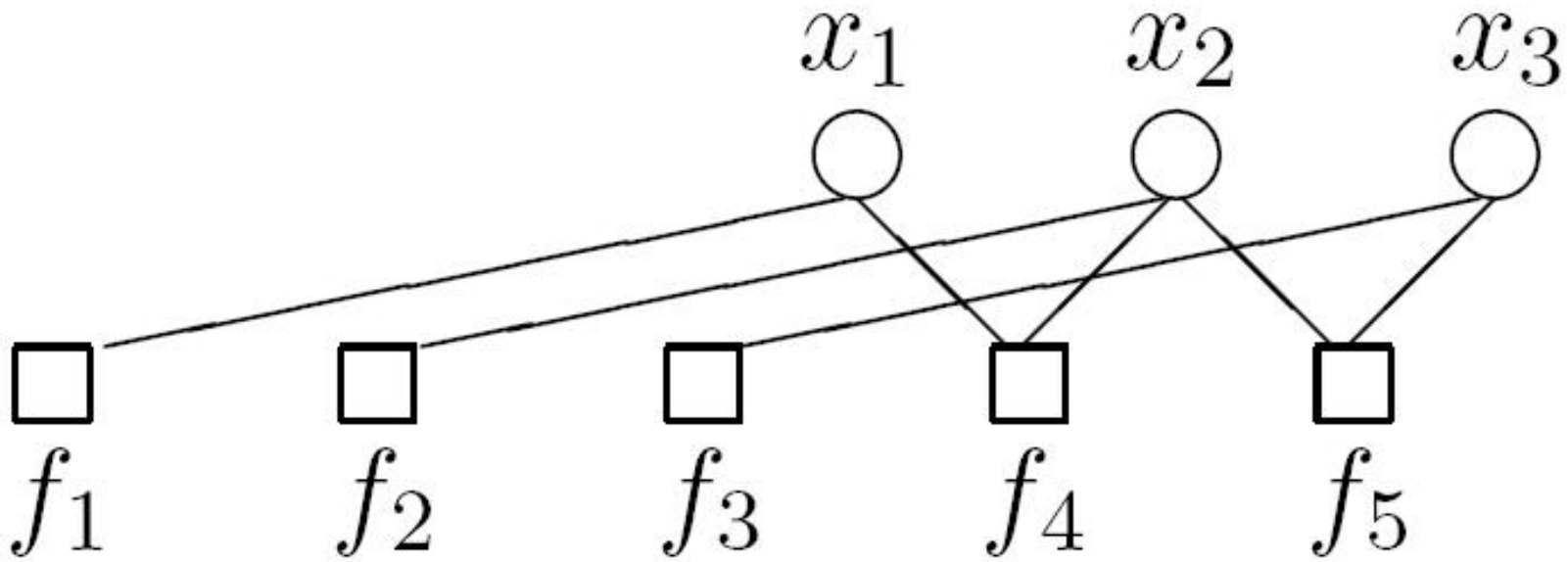
1. Суть и постановка задачи
   - Общий вид функции
2. Алгоритм sum–product
   - Структура графа
   - Сообщения
   - Работа алгоритма
   - Sum–product и разложение функции
3. Min-sum и деревья смежности
   - Min-sum алгоритм
   - Дерево сочленений
4. Метод Лапласа
   - Метод Лапласа в многомерном случае

Структура графа

- Граф состоит из вершин, соответствующих переменным, и вершин, соответствующих функциям.
- Функции соединены с переменными, которых они описывают.

Структура графа

$$p^*(X) = f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1, x_2)f_5(x_2, x_3).$$



Идея алгоритма

- Как и прежде, идея в том, чтобы пересылать сообщения.
- Сообщения двух видов: от функций к переменным и от переменных к функциям.

Сообщения

- От переменной x_i к функции f_j:

$$q_{i\to j}(x_i) = \prod_{k\in \mathrm{ne}(i)\setminus j} r_{k\to i}(x_i).$$

- От функции f_j к переменной x_i:

$$r_{j\to i}(x_i) = \sum_{X_j\setminus x_i} \left(f_j(X_j) \prod_{k\in \mathrm{ne}(j)\setminus i} q_{k\to j}(x_k) \right).$$

Сообщения в листьях

- Отметим, что если у вершины только один сосед, то её сообщение можно вычислить, не зная входящих сообщений.
- Пустое произведение равно единице.
- Поэтому функция f_j, зависящая только от одной переменной x_i, передаёт своей переменной $r_{j\to i} = f_j(x_i)$, а переменная x_i, участвующая только в одной функции, передаёт ей 1.

Работа алгоритма I

- Предположим, что граф — дерево.
- Тогда можно начать с листьев и, постепенно вычисляя сообщения, обойти все вершины.
- При этом применяется стандартное правило передачи сообщений: сообщение можно передавать, только если его можно полностью построить.
- За количество шагов, равное диаметру графа, работа алгоритма закончится.

Работа алгоритма II

- Что делать, если граф — не дерево, и не ясно, с чего начинать?
- Вариант — начать с того, что все переменные передают сообщение 1, а потом уже его модифицируют, когда до них доходят сообщения от функций.
- Такой алгоритм не всегда работает правильно и делает много лишнего, но всё же полезен на практике.

Упражнение. Привести пример, когда алгоритм sum–product на графе с циклами работает некорректно.

Результат работы

- Когда рассылка сообщений закончится, можно будет вычислить маргиналы:

$$p_i^*(x_i) = \prod_{j \in \mathrm{ne}(i)} r_{j \to i}(x_i).$$

- Ясно, что $Z = \sum_i p_i^*(x_i)$, и $p(x_i) = \frac{1}{Z} p_i^*(x_i)$.

Упражнение. Доказать, что действительно получится $p_i^*(x_i)$.

Нормализация on-the-fly

- Возможно, мы интересуемся только нормализованными маргиналами (настоящими вероятностями).
- Тогда можно просто на каждом шаге нормализовать сообщения от переменных к функциям (от функций к переменным сообщения те же):

$$q_{i\to j}(x_i) = \alpha_{ij} \prod_{k\in \mathrm{ne}(i)\setminus j} r_{k\to i}(x_i),$$

где α_{ij} подобраны так, чтобы

$$\sum_i q_{i\to j}(x_i) = 1.$$

Упражнения

Упражнение. Пошагово применить вышеописанный алгоритм к функции из примера (функции правдоподобия повторяющего кода).

Упражнение. Реализовать алгоритм sum–product для маргинализации произведения функций (обе версии). На вход алгоритму подаётся набор функций (в любом формате), на выходе — маргиналы. Если на входе дерево, программа должна применять первую версию алгоритма, если не дерево — вторую.

Разложение функции p^*

- Что такое sum–product математически?
- Изначально было разложение

$$p^*(X) = \prod_{j=1}^{m} f_j(X_j).$$

- А мы перераскладываем функцию в произведение

$$p^*(X) = \prod_{j=1}^{m} \phi_j(X_j) \prod_{i=1}^{n} \psi_i(x_i),$$

где ϕ_j соответствуют узлам–функциям, а ψ_i — узлам–переменным.

- Изначально, до передачи сообщений, $\phi_j(X_j) = f_j(X_j)$ и $\psi_i(x_i) = 1$.

Разложение функции p^*

- Каждый раз, когда приходит сообщение $r_{j\to i}$ из функции в переменную, ϕ и ψ пересчитываются:

$$\psi_i(x_i) = \prod_{j\in \mathrm{ne}(i)} r_{j\to i}(x_i),$$

$$\phi_j(X_j) = \frac{f_j(X_j)}{\prod_{i\in \mathrm{ne}(j)} r_{j\to i}(x_i)}.$$

- Очевидно, общее произведение от этого не меняется.

Упражнение. Доказать, что в результате этого процесса ψ_i действительно станет маргиналом $p^*(x_i)$. Это докажет корректность алгоритма.

Outline

1. Суть и постановка задачи
   - Общий вид функции
2. Алгоритм sum–product
   - Структура графа
   - Сообщения
   - Работа алгоритма
   - Sum–product и разложение функции
3. Min-sum и деревья смежности
   - Min-sum алгоритм
   - Дерево сочленений
4. Метод Лапласа
   - Метод Лапласа в многомерном случае

Постановка задачи

- Маргинализация — не единственная возможная постановка задачи.
- Вот, например, задача максимизации: найти значения переменных из X, которые максимизируют функцию p^*.
- Мы уже решали её и видели, какие нужны для этого инструменты.

От sum–product к max–product

- Задача максимизации p^* решится, если в узлах операцию суммирования $\sum$ заменить на max.
- Тогда нормализационная константа $Z = \sum_X p^*(x)$ превратится в $\max_X p^*(x)$, а каждый маргинал $p^*(x_i)$ станет показывать максимальное значение p^*, которого можно достичь в зависимости от значения x_i; отсюда уже тривиально можно будет найти максимизирующий набор.

От max–product к min–sum

- Умножать дороже, чем складывать. Как бы нам начать складывать вместо того чтобы умножать?

От max–product к min–sum

- Умножать дороже, чем складывать. Как бы нам начать складывать вместо того чтобы умножать?
- Естественно, надо логарифмировать (а потом добавить знак «минус»); тогда из max–product получится min–sum. Мы это уже много раз делали.
- Такой алгоритм называется алгоритмом Витерби (Viterbi).

Упражнение. Реализовать алгоритм Витерби для поиска максимального значения функции. Вход — тот же, выход — максимизирующий набор.

Суть

- Sum–product работает корректно, только если граф — дерево (ну, разве что скрестить пальцы и помолиться...).
- Что делать, когда граф содержит циклы?
- В прошлом году мы уже отвечали на этот вопрос — нужно использовать деревья сочленений.

Деревья сочленений — неформально

- Если цикл не сдаётся, его уничтожают, то есть заменяют весь цикл на одну вершину.
- Получается дерево, в котором уже можно работать обычным sum–product'ом; но при этом, конечно, замена нескольких вершин одной приводит к её экспоненциальному раздуванию (множество значений соответствующей переменной должно содержать все комбинации значений исходных переменных).
- О том, как это сделать формально и правильно, см. лекцию 9 предыдущего курса (о байесовских сетях).

Другие методы

- Есть приближённые методы, ими мы будем заниматься на семинарах и следующих лекциях.
- Но, как мы уже видели, есть очень хороший метод работать тогда, когда нельзя применять sum–product, а именно применять sum–product. :)
- Он работает довольно часто даже тогда, когда в принципе работать не обязан (когда есть циклы).

Outline

1. Суть и постановка задачи
   - Общий вид функции
2. Алгоритм sum–product
   - Структура графа
   - Сообщения
   - Работа алгоритма
   - Sum–product и разложение функции
3. Min-sum и деревья смежности
   - Min-sum алгоритм
   - Дерево сочленений
4. Метод Лапласа
   - Метод Лапласа в многомерном случае

Аппроксимационные методы

- Кроме точной маргинализации, можно рассматривать и приближённую. Особенно это важно, когда распределения непрерывные — тогда точно может просто вообще не получиться.
- Сейчас мы рассмотрим простейший метод Лапласа, а более хитрые приближённые методы маргинализации будут на семинарах.
- Этот метод прямого отношения к маргинализации не имеет — мы просто заменим сложное распределение на простое.

Суть метода Лапласа

- Рассмотрим ненормализованную плотность распределения p^* с нормализационной константой $Z = \int p^*(x)dx$.
- Предположим, что у p^* имеется максимум в точке x_0; разложим в ряд Тейлора логарифм $p^*(x)$:

$$\ln p^*(x) = \ln p^*(x_0) - \frac{c}{2}(x - x_0)^2 + \ldots,$$

где

$$c = -\left.\frac{\partial^2}{\partial x^2} \ln p^*(x)\right|_{x=x_0}.$$

Суть метода Лапласа

- А теперь приблизим p^* ненормированным гауссианом:

$$q^*(x) = p^*(x_0)e^{-\frac{c}{2}(x-x_0)^2},$$

а нормализационную константу приблизим константой гауссиана:

$$Z_q = p^*(x_0)\sqrt{\frac{2\pi}{c}}.$$

- Получится гауссиан, логарифм которого совпадает с первыми двумя членами ряда Тейлора p^*.

Обобщение метода Лапласа

- Пусть теперь $p^*(x_1, \ldots, x_k)$ работает в многомерном пространстве.
- Тогда появляется целая матрица вторых производных A:

$$A_{ij} = \frac{\partial^2}{\partial x_i \partial x_j} \ln p^*(x) \Big|_{x=x_0}.$$

- А ряд Тейлора выглядит как

$$\ln p^*(x) = \ln p^*(x_0) - \frac{1}{2}(x - x_0)^t A(x - x_0) + \ldots,$$

где x и x_0 — k–мерные векторы.

Обобщение метода Лапласа

- Наконец, нормализационная константа гауссиана будет равна

$$Z_q = p^*(x_0)\sqrt{\frac{(2\pi)^k}{\det A}}.$$

Упражнение. Докажите это.

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`