Стимулируемое обучение

Сергей Николенко

Машинное обучение — ИТМО, осень 2006

Outline

Постановка задачи

- До сих пор задача ставилась так: есть набор «правильных ответов», нужно его продолжить на всё пространство.
- Как работает обучение в реальной жизни? Мы далеко не всегда знаем набор правильных ответов, мы просто делаем то или иное действие и получаем результат.

Постановка задачи

- Отсюда и стимулируемое обучение (reinforcement learning).
- Агент взаимодействует с окружающей средой, предпринимая действия; окружающая среда его поощряет за эти действия, а агент продолжает их предпринимать.

Постановка задачи — формально

- На каждом шаге агент может находиться в состоянии $s \in S$.
- На каждом шаге агент выбирает из имеющегося набора действий некоторое действие $a \in A$.
- Окружающая среда сообщает агенту, какую награду r он за это получил и в каком состоянии s' после этого оказался.

Пример

- Диалог:

Среда: Агент, ты в состоянии 1; есть 5 возможных действий.
Агент: Делаю действие 2.
Среда: Даю тебе 2 единицы за это. Попал в состояние 5, есть 2 возможных действия.
Агент: Делаю действие 1.
Среда: Даю тебе за это −5 единиц. Попал в состояние 1, есть 5 возможных действий.
Агент: Делаю действие 4.
Среда: Даю тебе 14 единиц за это. Попал в состояние 3, есть 3 возможных действия...

- В этом примере агент успел вернуться в состояние 1 и исследовать ранее не пробовавшуюся опцию 4 (получив за это существенную награду).

И спросила кроха

- Мы хотим алгоритмы, которые «хорошо себя ведут».
- Что такое «хорошо»? Как оценивать поведение алгоритма в приведённом выше сеттинге?

Разные модели

- Ваши версии?

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?
- $$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

  где γ — некоторая константа (discount factor).

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?
- $$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$
  где γ — некоторая константа (discount factor).
- Модель *среднего вознаграждения* (average-reward model):
  $$\lim_{h\to\infty} E\left[\frac{1}{h}\sum_{t=0}^{h} r_t\right].$$

Что мы будем использовать

- Все модели разные, приводят к разным результатам. Получится ли у нас породить пример?
- Обычно используется модель бесконечного горизонта с некоторым фиксированным discount factor. Её и мы будем использовать, хотя для большинства алгоритмов у нас не получится ничего толкового доказать.

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
  - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
  - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
    - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
    - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?
- Минимизировать цену (regret), т.е. уменьшение общей суммы выигрыша по сравнению с оптимальной стратегией с самого начала. Это очень хорошая мера, но результаты о ней получить очень сложно.

Exploitation vs. exploration

- Каждый алгоритм должен и изучать окружающую среду, и пользоваться своими знаниями, чтобы максимизировать прибыль.
- Вопрос — как достичь оптимального соотношения? Та или иная стратегия может быть хороша, но вдруг она не оптимальная?
- Этот вопрос всегда присутствует в стимулируемом обучении.

Outline

Агенты с одним состоянием

- Формально всё то же самое, но $|S| = 1$, т.е. состояние агента не меняется. У него фиксированный набор действий A и возможность выбора из этого набора действий.
- Модель: агент в комнате с несколькими игровыми автоматами. У каждого автомата своё ожидание выигрыша. Нужно за ограниченное количество попыток выбрать лучший автомат.

Что мы будем рассматривать

- Подходы бывают доказуемо хорошие и ad hoc.
- Доказуемо хорошие:
  - Динамическое программирование
  - Индексы распределения (allocation indices)
  - Обучающиеся конечные автоматы
- Ad hoc:
  - Жадный алгоритм
  - Случайные стратегии (исследование по Больцману)
  - Интервальные оценки

Динамическое программирование

- Предположим, что агент действует на протяжении h шагов.
- Используем байесовский подход для определения оптимальной стратегии.
- Начинаем со случайных параметров $\{p_i\}$, например, равномерно распределённых, и вычисляем отображение из *belief states* (состояния после нескольких раундов обучения) в действия.
- Состояние выражается как $\mathcal{S} = \{n_1, w_1, \ldots, n_k, w_k\}$, где каждого бандита i запустили n_i раз и получили w_i единичек (считаем, что результат бинарный).

Динамическое программирование

- $V^*(\mathcal{S})$ — ожидаемый оставшийся выигрыш.
- Рекурсивно: если $\sum_{i=1}^{k} n_i = h$, то больше нечего делать, и $V^*(\mathcal{S}) = 0$.
- Если знаем V^* для всех состояний, когда осталось t запусков, сможем пересчитать и для $t+1$:

$$V^*(n_1, w_1, \ldots, n_k, w_k) = \\ = \max_i \left(\rho_i (1 + V^*(\ldots, n_i + 1, w_i + 1, \ldots)) + \right. \\ \left. (1 - \rho_i) V^*(\ldots, n_i + 1, w_i, \ldots) \right),$$

где ρ_i — апостериорные вероятности того, что действие i оправдается (если изначально p_i равномерно распределены, то $\rho_i = \frac{w_i+1}{n_i+2}$).

Индексы распределения Гиттинса

- Пусть мы уже n раз сделали некое действие и получили w единичек. Существуют предвычисленные таблицы *индексов распределения Гиттинса* (Gittins allocation indices) $I(n, w)$, которые учитывают как ожидаемую прибыль, так и количество новой информации, которую мы получим, если предпримем это действие ещё раз. Можно их просто максимизировать.
- Работает отлично, но не существует аналогов для нескольких состояний.

Обучающиеся конечные автоматы

- Автомат Цетлина (Tsetlin) для решения задачи с двумя бандитами.
- В общем случае проще рассматривать агента не как алгоритм, а как набор вероятностей, с которыми предпринимаются действия.
- Эти вероятности можно изменять (обучать алгоритм, как нейронную сеть).

Обучающиеся конечные автоматы

- *Алгоритм линейного вознаграждения–бездействия* (linear reward-inaction) добавляет линейно к вероятности действия a_i, если оно успешно:

$$p_i := p_i + \alpha(1 - p_i),$$

$$p_j := p_j - \alpha p_j, \quad j \neq i,$$

а если оно безуспешно, то вероятности сохраняются.

Обучающиеся конечные автоматы

- Алгоритм сходится с вероятностью 1 к вектору из одной единички и остальных нулей.
- Не всегда сходится к оптимальной стратегии; но вероятность ошибиться можно сделать сколь угодно малой, уменьшая α.

Жадный алгоритм

- Всегда выбирать стратегию, максимизирующую прибыль.
- Оптимум легко проглядеть, если на начальной выборке не повезёт (что вполне возможно).
- Поэтому полезная эвристика — *оптимизм при неопределённости*. То есть выбирать жадно, но при этом прибыль ожидается весьма оптимистично, и нужны серьёзные отрицательные свидетельства, чтобы отклонить стратегию.
- Это не конкретный алгоритм, а общая идея, подробности позже.

Случайные стратегии

- Стратегия: выбрать действие с наилучшей ожидаемой прибылью с вероятностью p, а с вероятностью $1 - p$ выбрать случайное действие.
- Обычно начинают с маленьких p, затем увеличивают.
- Но алгоритм не различает хорошую альтернативу от бесполезной.
- Исследование по Больцману (Boltzmann exploration):

$$p(a) = \frac{e^{ER(a)/T}}{\sum_{a'} e^{ER(a')/T}},$$

  где ER — ожидаемая прибыль, T — *температура*.
- Тоже обычно температура понижается со временем.

Интервальные оценки

- Один из способов применить оптимистично–жадный метод.
- Для каждого действия мы храним статистику n и w, а потом вычисляем доверительный интервал для вероятности успеха (с границей $1 - \alpha$) и для выбора стратегии используем верхнюю границу этого интервала.
- Все ли помнят, как вычислять доверительные интервалы?

Интервальные оценки

- Пример: испытания Бернулли (монетка, например). Тогда с вероятностью .95 среднее лежит в интервале

$$\left(\bar{x} - 1.96\frac{s}{\sqrt{n}}, \bar{x} + 1.96\frac{s}{\sqrt{n}}\right),$$

где 1.96 берётся из таблицы (распределение Стьюдента), n — количество испытаний, $s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$.

Упражнения

Упражнение Реализовать алгоритм динамического программирования для решения задачи стимулируемого обучения.

Упражнение Реализовать алгоритм интервальных оценок для решения задачи стимулируемого обучения.

Outline

Марковские процессы

- *Марковский процесс принятия решений* (Markov decision process) состоит из:
  - множества состояний S;
  - множества действий A;
  - функции поощрения $R : S \times A \to \mathbb{R}$;
  - функции перехода между состояниями $T : S \times A \to \Pi(S)$, где $\Pi(S)$ — множество распределений вероятностей над S. Вероятность попасть из s в s' после a — $T(s, a, s')$.
- Модель *марковская*, если переходы не зависят от истории предыдущих переходов.

Две основные задачи

- Предположим, что мы уже точно знаем нашу модель. Задача — найти оптимальную стратегию поведения для агента в этой модели.
- Реальная ситуация: модель не знаем, надо и модель узнать, и оптимально себя повести.
- Сначала решим первую задачу, без неё все равно не будет второй.

Оптимальные значения состояний

- *Оптимальное значение состояния* — ожидаемая суммарная прибыль, которую получит агент, если начнёт с этого состояния и будет следовать оптимальной стратегии:
$$V^*(s) = \max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$
- Эту функцию можно определить как решение уравнений
$$V^*(s) = \max_{a\in A}\left(R(s,a) + \gamma \sum_{s'\in S} T(s,a,s') V^*(s')\right),$$
а затем выбрать оптимальную стратегию
$$\pi^*(s) = \mathrm{argmax}_a \left(R(s,a) + \gamma \sum_{s'\in S} T(s,a,s') V^*(s')\right).$$
- Как решать уравнения?

Итеративное решение (по значениям)

- Ищем оптимальные значения состояний простым итеративным алгоритмом.
- `ValueIteration`:
  - Инициализировать $V(s)$.
  - Пока стратегия недостаточно хороша:
    - Для всех $s \in S$ и $a \in A$
      $$Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s').$$
    - $V(s) := \max_a Q(s,a)$.

Другой вариант

- Пересчёт в предыдущем алгоритме использует информацию от всех состояний–предшественников. Можно сделать другой вариант:

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)).$$

- Он работает, если каждая пара (s,a) встречается бесконечное число раз, s' выбирают из распределения $T(s,a,s')$, а r сэмплируют со средним $R(s,a)$ и ограниченной вариацией.

Итеративное решение (по стратегиям)

- Ищем оптимальную стратегию простым итеративным алгоритмом.
- `PolicyIteration`:
  - Инициализировать π.
  - Повторять:
    - Вычислить значения состояний для стратегии π, решив систему линейных уравнений

$$V_\pi(s) = R(s,\pi(s)) + \gamma \sum_{s'\in S} T(s,\pi(s),s')V_\pi(s').$$

    - Улучшить стратегию на каждом состоянии:

$$\pi'(s) := \mathrm{argmax}_a \left(R(s,a) + \gamma \sum_{s'\in S} T(s,a,s')V_\pi(s') \right).$$

  - пока $\pi \neq \pi'$.

Итеративное решение (по стратегиям)

- Сходится, т.к. на каждом шаге строго улучшаем целевую функцию, а всего существует конечное число ($|A|^{|S|}$) стратегий.

Адаптивный эвристический критик

- Основная идея — пусть один элемент алгоритма ищет стратегию, а другой — функцию значений, и пусть они друг с другом соперничают.
- Адаптивный эвристический критик (adaptive heuristic critic) состоит из *критика* AHC и *компонента стимулируемого обучения* RL.
- На месте RL может быть любой алгоритм стимулируемого обучения для решения задачи о k бандитах.

Адаптивный эвристический критик

- RL максимизирует не прибыль, а значение эвристики v, вычисляемое критиком.
- AHC в это время использует полученное от RL значение для пересчёта ожидаемых значений прибыли от состояний.
- Как будто бы по очереди: фиксируем стратегию π и подсчитываем функцию значений V_π. Затем фиксируем V_π и учим новую стратегию π', которая мексимизирует V_π. И так далее.

Адаптивный эвристический критик

- Как АНС выучит V_π по π?
- $\langle s, a, r, s' \rangle$ — *experience tuple* (s — состояние перед переходом, a — действие, r — поощрение, s' — следующее состояние.
- Значение можно выучить по правилу $TD(0)$:

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s)).$$

Адаптивный эвристический критик

- $TD(0)$ смотрит на один шаг вперёд. Можно сделать алгоритм, учитывающий много шагов, $TD(\lambda)$:

$$V(u) := V(u) + \alpha(r + \gamma V(s') - V(s))\epsilon(u),$$

применяемый к каждому состоянию u на основе его *eligibility* $\epsilon(u)$.

$$\epsilon(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k}[s = s_k].$$

- Eligibility — то, насколько часто это состояние посещалось в недавнем прошлом. Если $\lambda = 0$, $TD(\lambda)$ — это $TD(0)$. Eligibility можно пересчитывать в реальном времени:

$$\epsilon(s) := \gamma\lambda\epsilon(s) + [s = \text{текущему состоянию}].$$

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