

Глава 1

Байесовский вывод

§ 1.1. Введение

Байесовский вывод, которому посвящена эта глава, — центральное понятие искусственного интеллекта. Байесовские принципы

Сведения из теории вероятностей

ОПРЕДЕЛЕНИЕ 1.1. Вероятностное пространство — это тройка (Ω, \mathcal{F}, p) , где

- Ω — множество, элементы которого называются элементарными событиями, исходами или точками;
- \mathcal{F} — сигма-алгебра подмножеств Ω , называемых (случайными) событиями;
- P — вероятностная мера или вероятность, т.е. такая σ -аддитивная конечная мера, что $p(\Omega) = 1$.

ОПРЕДЕЛЕНИЕ 1.2. Случайная величина $X: \Omega \rightarrow \mathbb{R}$ индуцирует распределение случайной величины X на борелевских $P^X(B) = P(X \in B)$.

Рассмотрим основные операции над вероятностями:

- Совместная вероятность $p(x, y)$, $x \in \Omega_X$, $y \in \Omega_Y$ определяется на прямом произведении $\Omega_X \times \Omega_Y$.
- Маргинализация из совместной вероятности:

$$p(x = a) = \sum_{y \in \Omega_Y} p(x = a, y).$$

- Условная вероятность:

$$p(x = a | y = b) = \frac{p(x = a, y = b)}{p(y = b)}, \text{ если } p(y = b) \neq 0.$$

ЗАМЕЧАНИЕ. В дальнейшем будем считать, что все вероятности условные, если не оговорено обратное.

Рассмотрим основные правила операций с вероятностями:

- Правило произведения:

$$p(x, y | H) = p(x | y, H) p(y | H) = p(y | x, H) p(x | H).$$

— Правило суммирования:

$$p(x|H) = \sum_y p(x, y|H) = \sum_y p(x|y, H)p(y|H).$$

ОПРЕДЕЛЕНИЕ 1.3. Независимость *определим следующим образом: две случайные переменные X и Y независимы, если*

$$p(x, y) = p(x)p(y).$$

ТЕОРЕМА 1.1 (Теорема Байеса).

$$p(y|x, H) = \frac{p(x|y, H)p(y|H)}{p(x|H)} = \frac{p(x|y, H)p(y|H)}{\sum_{y'} p(x|y', H)p(y'|H)}.$$

П Р И М Е Р 1.1. О болезнях и вероятностях.

Приведём классический пример из классической области применения статистики — медицины:

- Пусть некий тест на какую-нибудь болезнь имеет вероятность успеха 95% (т.е. 5% — вероятность как позитивной, так и негативной ошибки).
- Всего болезнь имеется у 1% респондентов (отложим на время то, что они разного возраста и профессий).
- Пусть некий человек получил позитивный результат теста (тест говорит, что он болен).

С какой вероятностью он действительно болен?

Обозначим через t результат теста, через d — наличие болезни.

$$p(d = 1) = p(d = 1|t = 1)p(t = 1) + p(d = 1|t = 0)p(t = 0).$$

Используем теорему Байеса:

$$\begin{aligned} p(d = 1|t = 1) &= \frac{p(t = 1|d = 1)p(d = 1)}{p(d = 1|t = 1)p(t = 1) + p(d = 1|t = 0)p(t = 0)} = \\ &= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.05 \times 0.99} = 0.16. \end{aligned}$$

Таким образом, данный тест выдаёт правильное решение относительно состояния пациента всего лишь в 16 процентах случаев.

Подобные задачи составляют суть вероятностного вывода (probabilistic inference). Поскольку они обычно основаны на теореме Байеса, вывод часто называют байесовским (Bayesian inference).

§ 1.1.1. Частотные и байесовские вероятности. Понятие *вероятность* может быть истолковано двояко: как частота и как степень доверия:

- В классической теории вероятностей, происходящей из физики, вероятность рассматривается как предел отношения количества определённого результата эксперимента к общему количеству экспериментов. Классическим примером тут служит бросание монетки.
- В байесовском подходе вероятности выступают уже как *степени доверия* (degrees of belief).

Мы можем рассуждать о том, «насколько вероятно» то, что В. В. Путин назначит преемником С. В. Иванова или то, что «Одиссею» написала женщина.

О «количестве экспериментов» говорить бессмысленно — эксперимент ровно один.

§ 1.1.2. Прямые и обратные задачи теории вероятностей. В теории вероятностей существуют как прямые, так и обратные задачи. Для начала рассмотрим несколько примеров:

П р и м е р 1.2. Прямая задача 1. _____

- Прямая задача: в урне лежат 10 шаров, из них 3 чёрных. Какова вероятность выбрать чёрный шар?

П р и м е р 1.3. Прямая задача 2. _____

- В урне лежат 10 шаров с номерами от 1 до 10. Какова вероятность, что номера трёх последовательно выбранных шаров дадут в сумме 12?

П р и м е р 1.4. Обратная задача. _____

- Перед нами две урны, в каждой по 10 шаров, но в одной 3 чёрных, а в другой — 6. Кто-то взял из какой-то урны шар, и он оказался чёрным. Какова вероятность, что он брал шар из первой урны?

Таким образом, прямые задачи теории вероятностей описывают некий вероятностный процесс или модель и просят подсчитать ту или иную вероятность (т.е. фактически по модели предсказать поведение). Обратные задачи, в свою очередь, содержат *скрытые переменные* (в примере — номер урны, из которой брали шар). В них часто требуется по известному поведению построить вероятностную модель.

Задачи машинного обучения обычно являются задачами второй категории (обратными задачами).

Для рассмотрения дальнейших задач нам потребуется несколько определений.

ОПРЕДЕЛЕНИЕ 1.4. *Запишем теорему Байеса:*

$$p(x|y, H) = \frac{p(x)p(y|x, H)}{p(y|H)}.$$

Здесь

- $p(x)$ — априорная вероятность (*prior probability*),
- $(y|x, H)$ — правдоподобие (*likelihood*),
- $p(x|y)$ — апостериорная вероятность (*aposterior probability*),
- $(y|H)$ — вероятность данных (*evidence*).

ОПРЕДЕЛЕНИЕ 1.5. *Принцип правдоподобия (likelihood principle): если дана модель данных d при условии параметров θ , дана функция правдоподобия $p(d|\theta)$ и мы пронаблюдали некоторый набор данных d_1 , в дальнейшем вывод и предсказания должны зависеть только от $p(d_1|\theta)$. Иными словами, нам нужно только знать, как вероятность данных конкретных тестовых примеров d зависит от гипотезы.*

§ 1.2. Задача о нечестной монетке

Рассмотрим следующую задачу: дана нечестная монетка, она подброшена N раз, имеется последовательность результатов падения монетки. Надо определить её «нечестность» и предсказать, чем она выпадет в следующий раз.

§ 1.2.1. Вывод скрытых параметров.

ЗАМЕЧАНИЕ. Если у нас есть вероятность p_h того, что монетка выпадет решкой (вероятность орла $p_t = 1 - p_h$), то вероятность того, что выпадет последовательность s , которая содержит n_h решек и n_t орлов, равна

$$p(s|p_h, H_1) = p_h^{n_h} (1 - p_h)^{n_t}.$$

Сделаем следующее предположение:

ПРЕДЛОЖЕНИЕ 1.1. *Будем считать, что монетка выпадает равномерно, т.е. у нас нет априорного знания p_h .*

Теперь применим теорему Байеса и вычислим скрытые параметры:

$$p(p_h|s, H_1) = \frac{p(s|p_h, H_1)p(p_h|H_1)}{p(s|H_1)}$$

Здесь $p(p_h)$ следует понимать как непрерывную случайную величину, сосредоточенную на интервале $[0, 1]$, коей она и является. Наше предположение о равномерном распределении в данном случае значит, что априорная вероятность $p(p_h|H) = 1$, $p_h \in [0, 1]$ (т.е. априори мы не знаем, насколько нечестна монетка, и предполагаем это равновероятным). А $p(s|p_h, H_1)$ мы уже знаем.

Итого получается:

$$p(p_h|s, H_1) = \frac{p_h^{n_h}(1 - p_h)^{n_t}}{p(s|H_1)}.$$

Осталось подсчитать $p(s|H_1)$; её нужно маргинализировать из функции правдоподобия:

$$p(s|H_1) = \int_0^1 p_h^{n_h}(1 - p_h)^{n_t} dp_h = \frac{\Gamma(n_h + 1)\Gamma(n_t + 1)}{\Gamma(n_h + n_t + 2)} = \frac{n_h!n_t!}{(n_h + n_t + 1)!}.$$

А теперь предскажем следующий исход. Для этого необходимо вычислить $p(\text{heads}|s)$, т.е. вероятность выпадения орла при данной последовательности s :

$$\begin{aligned} p(\text{heads}|s) &= \int_0^1 p(\text{heads}|p_h)p(p_h|s, H_1)dp_h = \int_0^1 \frac{p_h^{n_h+1}(1 - p_h)^{n_t}}{p(s)} dp_h = \\ &= \frac{(n_h + 1)!n_t!}{(n_h + n_t + 2)!} \cdot \frac{(n_h + n_t + 1)!}{n_h!n_t!} = \frac{n_h + 1}{n_h + n_t + 2}. \end{aligned}$$

Это называется *правило Лапласа*.

§ 1.2.2. Сравнение моделей. Мы использовали предположение H_1 о равномерном априорном распределении p_h . Это было нашей моделью ситуации. Можно использовать другие модели; например, можно априори предположить, что монетка падает решкой с вероятностью $1/3$, т.е. $p_h = 1/3$. Обозначим через H_0

Как сравнить, какая из моделей H_i лучше описывает данные?

Для сравнения применим теорему Байеса:

$$p(H_1|s) = \frac{p(s|H_1)p(H_1)}{p(s)}, \quad p(H_0|s) = \frac{p(s|H_0)p(H_0)}{p(s)}.$$

Нужно как-то оценить априорные вероятности $p(H_0)$ и $p(H_1)$; можно просто положить их равными $1/2$.

Теперь можно составить отношение:

$$\frac{p(H_1|s)}{p(H_0|s)} = \frac{p(s|H_1)p(H_1)}{p(s|H_0)p(H_0)} = \frac{n_h!n_t!}{(n_h + n_t + 1)!} \cdot \frac{1}{(1/3)^{n_h}(2/3)^{n_t}}.$$

Это отношение можно использовать для сравнения моделей.

§ 1.3. Гипотезы максимального правдоподобия

§ 1.3.1. Определения.

ОПРЕДЕЛЕНИЕ 1.6. Гипотеза максимального правдоподобия (*maximum likelihood hypothesis*) — это набор параметров θ , который максимизирует функцию правдоподобия

$$p(D|\theta, H),$$

где D — имеющаяся информация (тестовые примеры).

Иными словами, гипотеза максимального правдоподобия — гипотеза, которая лучше всего описывает имеющиеся данные D в текущих предположениях H .

§ 1.3.2. Гауссианы. Начнём с простейшего гауссиана. Его распределение содержит два параметра:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Функция правдоподобия данных x_1, \dots, x_n :

$$p(x_1, \dots, x_n | \mu, \sigma) = \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\sum_{i=1}^n \frac{(x_i - \mu)^2}{2\sigma^2}}.$$

Заметим, что функция эта зависит от двух параметров, а не от n :

$$p(x_1, \dots, x_n | \mu, \sigma) = \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{S + n(\bar{x} - \mu)^2}{2\sigma^2}},$$

где

$$\bar{x} = \frac{x_1 + \dots + x_n}{n}, \quad S = \sum_{i=1}^n (\bar{x} - x_n)^2.$$

ОПРЕДЕЛЕНИЕ 1.7. Параметры \bar{x} и S называются достаточными статистиками (*sufficient statistics*).

Какие параметры лучше всего описывают данные?

Перейдём к логарифму:

$$\ln p(x_1, \dots, x_n | \mu, \sigma) = -n \ln(\sigma\sqrt{2\pi}) - \frac{S + n(\bar{x} - \mu)^2}{2\sigma^2}.$$

Для того, чтобы выяснить, при каких параметрах функция правдоподобия максимизируется, необходимо взять частные производные и приравнять нулю.

Возьмём производную по μ :

$$\frac{\partial \ln p}{\partial \mu} = -\frac{n}{\sigma^2}(\mu - \bar{x}).$$

То есть в гипотезе максимального правдоподобия $\mu = \bar{x}$, независимо от S . Теперь нужно найти σ из гипотезы максимального правдоподобия. Для этого мы продифференцируем по $\ln \sigma$ — полезный приём на будущее. Кстати, $\frac{dx^n}{d(\ln x)} = nx^n$.

$$\frac{\partial \ln p}{\partial \ln \sigma} = -n + \frac{\sum_{i=1}^n (x_i - \mu)}{\sigma^2}.$$

Следовательно, в гипотезе максимального правдоподобия

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)}{n}}.$$

Теперь то же самое для нескольких гауссианов сразу. Даны несколько точек x_1, \dots, x_n , но они принадлежат смеси гауссианов с разными μ_k (пусть σ будет одна и та же). Тогда распределение будет

$$p(x_1, \dots, x_n | \{\mu_k\}, \sigma) = \prod_{i=1}^n p(x_i | \{\mu_k\}, \sigma).$$

§ 1.4. Интересные распределения

Есть некоторое количество распределений, которые появляются в разных задачах и являются наиболее полезными на практике. Перечислим основные из них.

§ 1.4.1. Дискретные распределения.

- *Биномиальное распределение* возникает, когда мы подбрасываем нечестную монетку (вероятность решки q) n раз и хотим найти вероятность появления r решек.

$$p(r|q, n) = \binom{n}{r} q^r (1 - q)^{n-r}.$$

- *Распределение Пуассона* возникает, когда мы хотим подсчитать количество событий за фиксированный интервал, если нам дана средняя интенсивность этих событий. Если ожидается в среднем λ событий за этот интервал, то вероятность того, что произойдут ровно r событий, равна

$$p(r|\lambda) = e^{-\lambda} \frac{\lambda^r}{r!}.$$

- *Экспоненциальное распределение* обычно возникает в ответах на вопрос «сколько надо ждать события». Например, вероятность того, что выпадения орла на нечестной монетке придётся ждать ровно r шагов, равна

$$p(r|q) = q^r (1 - q) = (1 - q) e^{-\lambda r}, \quad \lambda = \ln \frac{1}{1 - q}.$$

- *Гипергеометрическое распределение* обычно возникает в ситуациях выбора без замещения. Если есть урна, в ней n шаров, из них k чёрных, то вероятность вынуть ровно i чёрных равна

$$p(i|n, k) = \frac{\binom{k}{i} \binom{n-k}{n-i}}{\binom{n}{k}}.$$

§ 1.4.2. Распределения на вещественных числах.

- *Распределение Стьюдента* применяется, когда нужно искать доверительные интервалы на параметры нормального распределения. Величина $T = \frac{\bar{x}_n - \mu}{S_n / \sqrt{n}}$ распределена по закону

$$f(t) = \frac{\Gamma(n/2)}{\sqrt{\pi(n-1)}\Gamma(\frac{n-1}{2})} \left(1 + \frac{t^2}{n-1}\right)^{-n/2}.$$

Если мы выберем число A так, чтобы $p(-A < T < A) = 1 - \alpha$, то

$$\left[\bar{x}_n - A \frac{S_n}{\sqrt{n}}, \bar{x}_n + A \frac{S_n}{\sqrt{n}} \right]$$

будет доверительным интервалом для μ с вероятностью ошибки α .

- *Экспоненциальное распределение* используется для оценки времени ожидания в пуассоновском процессе. Если мы ждём события, которое происходит в среднем каждые $1/\lambda$ единиц времени (с интенсивностью λ), то распределение времени ожидания

$$p(x|s) = \lambda e^{-\lambda x}.$$

- *Гамма-распределение* является аналогом нормального распределения, но на полуоси $[0, +\infty)$. Плотность распределения

$$p(x|k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \Gamma(k)}, \quad x > 0.$$

§ 1.4.3. Другие распределения.

- *Бета-распределение* определяется над вероятностями, т.е. на интервале $[0, 1]$.

$$p(x|\alpha, B) = \frac{1}{B(\alpha, B)} x^{\alpha-1} (1-x)^{B-1}, \quad B(\alpha, B) = \frac{\Gamma(\alpha)\Gamma(B)}{\Gamma(\alpha+B)}.$$

$B(i, j)$ — это распределение i -й по величине случайной величины из $i + j - 1$ случайных величин, распределённых равномерно на $[0, 1]$.

- *Распределение Дирихле* является обобщением бета-распределения на многомерный случай. На k -мерном симплексе $\{x \mid \sum_{i=1}^k x_i = 1\}$ плотность распределения Дирихле с параметром $\alpha = (\alpha_1, \dots, \alpha_k)$ равна

$$p(x|\alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^k x_i^{\alpha_i-1}, \quad B(\alpha) = \frac{\prod_{i=1}^k \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^k \alpha_i\right)}.$$

Глава 2

Байесовский вывод полным перебором

§ 2.1. Введение

Пусть нам, как обычно, нужно понять, какая гипотеза лучше других описывает имеющиеся данные. Предлагается алгоритм: перечислить все гипотезы и сравнить их правдоподобия (likelihoods). Это точный, но из-за своей сложности мало применяемый, метод. Мы сначала рассмотрим, как этот метод работает в дискретном булевском случае, а затем в непрерывном случае нормального распределения.

§ 2.2. Полный перебор на байесовских сетях

Байесовскую теорию вероятностей иногда называют "усиленным здравым смыслом". Когда будете изучать следующие вопросы, пожалуйста, подумайте, как бы на них ответили вы. В конце этой части вы увидите как байесовский метод подтверждает вашу интуицию.

Для того, чтобы решить байесовскую сеть полным перебором, нужно представить её в виде большого произведения всех вероятностей, которые в ней участвуют, а затем *маргинализовать* по вероятностям, которые нам известны. Рассмотрим ситуацию, на основе которой, разберемся с байесовским выводом полным перебором поподробнее.

Пусть Вася летом живет в пригороде, на даче, и каждый день ему приходится проезжать 60 километров до работы. Однажды, когда он приехал на работу, ему позвонил его сосед и сказал, что в его (Васином) доме сработала сигнализация. Какая вероятность того, что в Васин дом проникли грабители?

Когда расстроенный и встревоженный Вася уже возвращается домой, по радио передают, что в том же районе, где находится его дом, произошло незначительное землетрясение. Вася с облегчением вздыхает, так как знает, что землетрясение часто вызывает срабатывание сигнализации. Какая вероятность того, что грабители проникли в Васин дом?

Для того, чтобы ответить на поставленные вопросы построим модель данной ситуации. Введем переменные: b (в доме Васи произошла кража), a (сработала сигнализация), p (Васе позвонил сосед и рассказал о сигнализации), e (микроразземлетрясение)

случилось рядом с домом Васи), r (Вася услышал по радио, что произошло землетрясение). Вероятность всех этих событий может быть представлена в следующем виде:

$$p(b, e, a, p, r) = p(b)p(e)p(a|b, e)p(p|a)p(r|e).$$

На рисунке представлена несложная сеть (даже без циклов), описывающая затруднительное положение Васи.

Зададим априорные вероятности. Вероятность кражи:

$$p(b = 1) = \beta, \quad p(b = 0) = 1 - \beta,$$

где $\beta = 0.001$, то есть кражи происходят раз в три года.

Вероятность землетрясения:

$$p(e = 1) = \epsilon, \quad p(e = 0) = 1 - \epsilon,$$

где $\epsilon = 0.001$, то есть в этом районе землетрясения происходят раз в три года; мы так же предполагаем, что землетрясение и кража, независимые события, то есть $p(e, b) = p(b)p(e)$, это кажется разумным, по крайней мере, если не принимать во внимание грабителей, которые грабят сразу после землетрясений.

Для простоты предположим, что сосед никогда не ошибается и не обманывает, поэтому $p(p = 1|a = 0) = 0$; и радио надежно, то есть $p(r = 1|e = 0) = 0$. Таким образом нам не нужно задавать вероятности $p(p = 1|a = 1)$ и $p(r = 1|e = 1)$ для того, чтобы ответить на поставленные вопросы, так как из входных значений $p = 1$ и $r = 1$ сразу следует, что $a = 1$ и $e = 1$ соответственно.

Мы предполагаем, что сигнализация сработает, если произойдет хотя бы одно из трех событий: (а) грабитель проникнет в дом и активирует сигнализацию (давайте предположим, что сигнализация надежна с вероятностью $\alpha_b = 0.99$, то есть 99% грабителей инициируют сигнализацию); (b) землетрясения вызывает сигнализацию с вероятностью $\alpha_e = 0.01$; или (с) какие-то другие причины с инициировали сигнализацию и $\alpha_f = 0.001$, то есть раз в три года сигнализация срабатывает ложно. (Этот тип зависимости a от b , e и f называется *Noisy-OR*.) В результате получаем следующую формулу:

$$\text{alarm} = \text{burglar} \vee \text{earthquake} \vee \text{false_alarm},$$

но не точно логически, а с некоторыми вероятностями. Вероятность a от b и e тогда равна:

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= \alpha_f, \\ p(a = 1|b = 1, e = 0) &= 1 - (1 - \alpha_f)(1 - \alpha_b), \\ p(a = 1|b = 0, e = 1) &= 1 - (1 - \alpha_f)(1 - \alpha_e), \\ p(a = 1|b = 1, e = 1) &= 1 - (1 - \alpha_f)(1 - \alpha_b)(1 - \alpha_e). \end{aligned}$$

И в числах:

$$\begin{aligned} p(a = 1|b = 0, e = 0) &= 0.001, \\ p(a = 1|b = 1, e = 0) &= 0.99001, \\ p(a = 1|b = 0, e = 1) &= 0.01099, \\ p(a = 1|b = 1, e = 1) &= 0.9901099. \end{aligned}$$

Теперь давайте проводить маргинализацию. В первой ситуации мы знаем, что нам позвонили, и хотим выяснить распределение вероятностей визита грабителя и землетрясения, т.е. найти $p(b, e|p = 1)$. Используем теорему Байеса:

$$p(b, e|p = 1) = \frac{p(p = 1|b, e)p(b)p(e)}{p(p = 1)}$$

и маргинализуем $p(p = 1|b, e)$ и $p(p = 1)$ из нашей сети:

$$p(b, e|p = 1) = \frac{\sum_a p(p = 1|a)p(a|b, e)p(b)p(e)}{\sum_{a,b,e} p(p = 1|a)p(a|b, e)p(b)p(e)}.$$

В итоге получается

$$\begin{aligned} p(b = 0, e = 0|p = 1) &= 0.4993, \\ p(b = 1, e = 0|p = 1) &= 0.4947, \\ p(b = 0, e = 1|p = 1) &= 0.0055, \\ p(b = 1, e = 1|p = 1) &= 0.0005. \end{aligned}$$

То есть вероятность реального грабителя — около 50%. А если пересчитать с учётом события $r = 1$, то получится

$$\begin{aligned} p(b = 0|p = 1, r = 1) &= 0.92, \\ p(b = 1|p = 1, r = 1) &= 0.08. \end{aligned}$$

Вот поэтому Вася и может успокоиться.

§ 2.3. Полный перебор в непрерывном случае

Пусть мы хотим найти скрытые параметры, например, нормального распределения методом полного перебора. Так как параметры непрерывные, всех возможных вариантов не переберёшь, поэтому приходится делать пространство дискретным и перебирать параметры с каким-то шагом.

Рисунок показывает перечисление сотни гипотез, которые размещены в таблице десять на десять и покрывают десять различных μ и σ . Каждая гипотеза представлена графиком плотности распределения. Мы исследуем вывод μ и σ для заданного набора точек x_n , $n = 1, \dots, N$, случайной независимой выборки из целевого распределения.

При выводе этих параметров, мы должны как-то задать априорное распределение. Это дает нам возможность включить в вывод какие-то первоначальные знания о

μ и σ (полученные, например, из независимых экспериментов или обоснованные теоретически). Если же у нас отсутствует опыт в рассматриваемой области, приходится брать такое априорное распределение, которое покрывает все возможные, например, равномерное на интервале, который мы дискретизируем. Позже мы рассмотрим более разумные априорные распределения.

Итак, рассмотрим нормальное распределение:

$$p(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

У него два параметра, по которым можно перебирать. То есть алгоритм состоит из простого перебора параметров μ и σ и вычисления функции правдоподобия $p(\{x_i\}|\mu, \sigma)$.

Более сложный случай — когда распределение представляет собой смесь гауссианов, которые берутся с весами α_1 и α_2 , $\alpha_1 + \alpha_2 = 1$:

$$p(x|\mu_1, \sigma_1, \alpha_1, \mu_2, \sigma_2, \alpha_2) = \frac{\alpha_1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + \frac{\alpha_2}{\sigma_2\sqrt{2\pi}} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}.$$

Тут уже пять параметров.

§ 2.4. Маргинализация интегрированием

Вспомним, что маргинализация — это суммирование по некоторым переменным так, чтобы их изгнать из произведения. Маргинализация — основа байесовского вывода, главный (и самый вычислительно сложный) инструмент. Когда мы делали вывод грубой силой, мы проводили маргинализацию, суммируя по всем возможным значениям, а для непрерывных переменных рассматривали все возможные значения с некоторым шагом. Но ведь можно просто взять определённый интеграл.

Когда мы хотим выполнять точную маргинализацию, зачастую полезно использовать так называемые *сопряжённые априорные распределения* (conjugate priors), потому что с ними вычисления упрощаются.

Для параметра μ нормального распределения сопряжённое априорное распределение — это тоже нормальное распределение с параметрами μ_0 , σ_μ . А для σ , точнее, для $\beta = 1/\sigma^2$, естественным априорным распределением будет гамма-распределение:

$$p(\beta|k_\beta, \theta_\beta) = \beta^{\theta_\beta-1} \frac{e^{-\beta/k_\beta}}{k_\beta^{\theta_\beta} \Gamma(\theta_\beta)}, \quad \beta > 0.$$

Но мы будем пользоваться только их вырожденными случаями, чтобы необходимые математические выводы выглядели предельно просто.

Для μ будем рассматривать $p(\mu) = \text{const}$; это совершенно неправильное распределение. Для σ будем рассматривать предел при $k_\beta\theta_\beta = 1$, $\theta_\beta \rightarrow 0$, т.е. плоское распределение $\ln \sigma$ (он же $\ln \beta$). Это и не распределения вовсе (не интегрируются); так и называются — *improper priors*.

Пусть есть данные $D = \{x_i\}_{i=1}^n$. Тогда оценка среднего

$$\bar{x} = \sum_{i=1}^n x_i / n,$$

а две возможные оценки дисперсии —

$$\sigma_n = \sqrt{\frac{\sum_{i=1}^n (\bar{x} - x_n)^2}{n}}, \quad \sigma_{n-1} = \sqrt{\frac{\sum_{i=1}^n (\bar{x} - x_n)^2}{n-1}}.$$

σ_n — оценка максимального правдоподобия, но смещенная: ожидание σ_n при условии σ не равно σ ; σ_{n-1} — несмещенная оценка.

Мы уже доказывали, что (\bar{x}, σ_n) — гипотеза максимального правдоподобия. Теперь давайте попробуем найти апостериорное распределение μ при данном σ :

$$p(\mu | \{x_i\}_{i=1}^n, \sigma) = \frac{p(\{x_i\}_{i=1}^n | \mu, \sigma) p(\mu)}{p(\{x_i\}_{i=1}^n | \sigma)} = C e^{-n(\mu - \bar{x})^2 / (2\sigma^2)}.$$

То есть получили нормальное распределение на μ с параметрами $(\bar{x}, \sigma^2/n)$.

Теперь решим задачу маргинализации. Пусть мы хотим найти наиболее вероятную σ при имеющихся данных. Это значит, что нам придётся маргинализировать μ из данных, когда мы будем подсчитывать

$$p(\sigma | \{x_i\}_{i=1}^n) = \frac{p(\{x_i\}_{i=1}^n | \sigma) p(\sigma)}{p(\{x_i\}_{i=1}^n)}.$$

Здесь мы, как и раньше, предполагаем, что $p(\mu) = 1/\sigma_\mu = \text{const}$. Тогда

$$p(\{x_i\}_{i=1}^n | \sigma) = \int p(\{x_i\}_{i=1}^n | \sigma, \mu) p(\mu) d\mu = \frac{1}{\sigma_\mu} \int \frac{1}{\sigma\sqrt{2\pi}} e^{-\sum_{i=1}^n \frac{(x_i - \mu)^2}{2\sigma^2}} d\mu,$$

и

$$\ln p(\{x_i\}_{i=1}^n | \sigma) = -n \ln(\sqrt{2\pi}\sigma) - \frac{\sum_{i=1}^n (\bar{x} - x_n)^2}{2\sigma^2} + \ln \frac{\sqrt{2\pi}\sigma / \sqrt{n}}{\sigma_\mu}.$$

За счёт последнего члена (так называемого *фактора Оккама* — мы эти факторы ещё обсудим) максимум и сдвигается с σ_n на σ_{n-1} .

Мы вычислили один простой интеграл, которым маргинализовали одну из переменных. В этом суть точной маргинализации с непрерывными переменными: нужно проводить точное интегрирование. Интеграл был такой простой, потому что мы предполагали очень простые (неправильные) априорные распределения.

Байесовский вывод и теория кодирования

§ 3.1. Постановка задачи

Принципы байесовского вывода оказываются настолько универсальны, что применяются в самых различных областях. В этом разделе мы увидим, как байесовские методы оказываются полезными для области, формально не относящейся к искусственному интеллекту: теории кодирования.

Дадим сначала общую постановку задачи кодирования и определение кодов, исправляющих ошибки. Пусть два устройства соединены некоторым каналом связи, и по этому каналу требуется передать некоторые цифровые данные. Мы будем предполагать, что канал способен передавать информацию только в виде битов (например, по проводнику идёт ток двух различных устойчивых уровней напряжения, и переходы между этими уровнями означают нули и единицы). Поэтому передаваемые данные нужно закодировать — преобразовать в последовательность битов. Мы будем предполагать, что это кодирование уже произведено, т.е. данные уже имеются в виде последовательности битов.

Если канал является абсолютно помехоустойчивым (гарантирует, что будет получено в точности то, что было отправлено), то больше ничего делать не нужно. В противном случае необходим механизм, позволяющий выявлять и исправлять ошибки, внесённые вследствие передачи данных по каналу связи.

Эту задачу позволяют решать так называемые *коды, исправляющие ошибки* (error-correcting codes). Неформально их можно определить как «коды, которые могут даже по неправильному кодовому слову достаточно часто выдавать правильное сообщение».

Дадим более формальную постановку задачи. Обозначим через \mathcal{M} множество всех сообщений, а через \mathcal{C} — множество всех кодовых слов (в нашем основном случае эти множества совпадают и являются множествами всех битовых строк $\{0, 1\}^*$). Пусть необходимо передать некоторое сообщение m . Для этого передающее устройство применяет к нему некоторую функцию кодирования $E : \mathcal{M} \rightarrow \mathcal{C}$ и передает по каналу результат ее применения $t = E(m)$. Обозначим то, что получило принимающее устройство, как y (возможно, $y \neq t$ из-за ошибок при передаче данных). Тогда

это устройство применяет к y декодирующую функцию $D : \mathcal{C} \rightarrow \mathcal{M}$. Если при определённых ограничениях на «отличие» y от t $D(y) = m$, то такой код (совокупность функций кодирования и декодирования) называется кодом, исправляющим ошибки.

Описанная выше задача — по искажённому $E(m)$ найти m — называется *задачей полного декодирования*. Кроме неё, имеет смысл также задача *побитового декодирования* — для каждого передаваемого бита t_n понять, насколько вероятно, что это был ноль или единица. Хотя это кажется частным случаем задачи полного декодирования, позже мы увидим, что результаты побитового декодирования могут и не совпадать с результатом полного.

Обозначим через t кодовое слово, через y — полученный сигнал. Тогда, по формуле Байеса,

$$p(t|y) = \frac{p(y|t)p(t)}{p(y)}.$$

Если канал не имеет памяти (биты, передававшиеся ранее, не оказывают влияния на последующие биты), то

$$p(y|t) = \prod_{i=1}^n p(y_i|t_i).$$

Априорное распределение $p(t)$ будем считать равномерным — т.е. будем считать, что априори частота появления нулей и единиц в сообщениях примерно одинакова. Казалось бы, это предположение весьма далеко от истины: например, если закодирован текст на русском языке, то код буквы «О» будет встречаться гораздо чаще кода буквы «Щ». Однако методы эффективного кодирования, такие, например, как код Хаффмана, позволяют кодировать более часто встречающиеся символы алфавита более короткими кодами, и средняя частота единиц и нулей будет оставаться примерно одинаковой.

П р и м е р 3.1. Кодирование по Хаффману.

Предположим, что в алфавите четыре символа $\alpha, \beta, \gamma, \delta$, которые встречаются с частотами $1/2, 1/5, 1/5, 1/10$ соответственно. Тогда естественное кодирование $\alpha \rightarrow 00, \beta \rightarrow 10, \gamma \rightarrow 01, \delta \rightarrow 11$ приведёт к тому, что ожидание количества нулей в кодовом слове будет составлять $2 \times \frac{1}{2} + 1 \times \frac{1}{5} + 1 \times \frac{1}{5} = 1.4$, и баланс явно нарушен. Однако это, разумеется, далеко не оптимальное кодирование. Применим кодирование по Хаффману. Сначала объединим γ и δ с кодами 0 и 1 соответственно; затем объединим полученный узел с β , а потом новый узел — с α . Читая метки графа в обратном направлении, получим кодирование

$$\begin{aligned} \alpha &\rightarrow 0, & \beta &\rightarrow 10, \\ \gamma &\rightarrow 110, & \delta &\rightarrow 111. \end{aligned}$$

При таком (близком к оптимальному) кодировании вероятность получить ноль в той или иной позиции кодового слова всё ещё больше $1/2$, но уже гораздо ближе к ней.

Но вернёмся к формуле Байеса. Знаменатель дроби в этой формуле можно преобразовать следующим образом

$$p(y) = \sum_t p(y|t)p(t).$$

Полным решением задачи декодирования является список всех кодовых слов и их вероятностей при условии получения данного сигнала. Однако такое полное решение обычно не требуется, а для практических целей необходимо просто найти наиболее вероятное кодовое слово. *Задача декодирования с максимальным правдоподобием* (MAP codeword decoding problem) — это задача поиска наиболее вероятного кодового слова t при условии данного сигнала y . При равномерном априорном распределении эта задача сводится к максимизации правдоподобия $p(y|t)$.

Задачу побитового декодирования можно решить маргинализацией:

$$p(t_i = 1|y) = \sum_{t: t_i=1} p(t|y) = \sum_t [t_i = 1]p(t|y),$$

$$p(t_i = 0|y) = \sum_{t: t_i=0} p(t|y) = \sum_t [t_i = 0]p(t|y).$$

Мы уже рассказывали, как решать задачу маргинализации полным перебором. Ниже мы рассмотрим более эффективный метод, но сначала придётся изучить кое-что из теории кодирования.

§ 3.2. Линейные коды

Простейшим кодом, исправляющим ошибки, является так называемый *повторяющий код* (repetition code). Он заключается в том, чтобы при кодировании повторять несколько раз каждый передаваемый бит, а при декодировании выбирать большинство из полученных битов. Например, если повторять каждый бит три раза, такой код позволит исправлять ошибку в одном бите.

Давайте проанализируем повторяющий код при помощи байесовских методов. Мы докажем, что решение большинством голосов в случае повторяющего кода реализует максимальную апостериорную гипотезу, если вероятность ошибки в одном бите меньше $1/2$. Пусть $p < 1/2$ — вероятность ошибки. Рассмотрим все варианты того, какой сигнал был получен. Мы уже видели, что для максимизации апостериорной вероятности достаточно максимизировать правдоподобие $p(y|t)$.

Предположим, что мы приняли сигнал $t = 000$. В этом случае правило большинства даст ответ $y = 000$, $p(y|t) = (1 - p)^3$, а для варианта $y = 111$, $p(y|t) = p^3$. Первое число больше второго, так как $1 - p > p$. Это и означает, что мы выбрали максимальную апостериорную гипотезу.

Если мы приняли сигнал $t = 001$, $t = 010$ или $t = 100$, то правило большинства даст ответ $y = 000$, $p(y|t) = (1-p)^2p$, а для варианта $y = 111$ $p(y|t) = p^2(1-p)$. Опять же, первое число больше второго, так как $1-p > p$. Аналогично, сигнал $t = 011$, $t = 110$ или $t = 101$ приведёт к ответу $y = 111$, имеющему правдоподобие $p(y|t) = (1-p)^2p$ против правдоподобия $p(000|t) = p^2(1-p)$, а в случае сигнала $t = 111$ $p(111|t) = (1-p)^3$, а $p(000|t) = p^3$.

Эти рассуждения легко обобщаются на случай n бит: $p^k(1-p)^{n-k} > p^{n-k}(1-p)^k$ тогда и только тогда, когда $k > \frac{n}{2}$. Но более интересным результатом будет собственно вероятность ошибки. При повторении n раз вероятность ошибки при декодировании составляет

$$\sum_{i=(n+1)/2}^n \binom{n}{i} p^i (1-p)^{n-i}.$$

Эта функция растёт достаточно быстро. Для практических целей было бы неплохо иметь возможность переписывать гигабайтные файлы так, чтобы терялся не более чем один из тысячи, то есть сделать вероятность ошибки порядка 10^{-15} . Для этого, при $p = ???$, необходимо брать n порядка 60. Передавать по каналу в 60 раз больше информации, чем в результате получится — это всё-таки чересчур. Попробуем рассмотреть более эффективные варианты.

Придадим нашему общему определению конкретное содержание. Будем считать, что кодирование происходит следующим образом: сообщение длиной в k бит отображается (функцией E) в кодовое слово длиной n бит ($n > k$, разумеется).

ОПРЕДЕЛЕНИЕ 3.1. Код называется линейным, если его функция кодирования $E: \{0, 1\}^k \rightarrow \{0, 1\}^n$ тождественна на первых k битах и линейна на оставшихся $n - k$ битах.

Иначе говоря, первые k бит кодового слова линейного кода — само сообщение, а оставшиеся $n - k$ — линейные функции от битов сообщения (parity checks).

Широко известным примером такого кода является так называемый код Хэмминга $(7, 4)$ (на 4 бита сообщения приходятся 7 битов сигнала). Пусть $s_1 s_2 s_3 s_4$ — исходное сообщение, а $t_1 t_2 t_3 t_4 t_5 t_6 t_7$ — соответствующее ему кодовое слово. Тогда:

$$t_1 = s_1, \quad t_2 = s_2, \quad t_3 = s_3, \quad t_4 = s_4,$$

$$t_5 = s_1 \oplus s_2 \oplus s_3, \quad t_6 = s_2 \oplus s_3 \oplus s_4, \quad t_7 = s_1 \oplus s_3 \oplus s_4.$$

Например, исходное сообщение 0101 будет закодировано как 0101101.

Очевидно, что для того, чтобы код мог исправлять ошибку в одном бите, необходимо, чтобы кодовые слова отличались друг от друга хотя бы в трёх битах (если есть два кодовых слова, отличающихся в двух битах, то при ошибке в одном из этих битов мы не будем знать, куда отнести сообщение).

Нетрудно видеть, что в коде Хэмминга (7, 4) кодовые слова отличаются друг от друга в трёх битах. На первый взгляд кажется, что достигнут такой же результат, как и в повторяющем коде. На самом деле это не совсем так, потому что вероятность двух ошибок в семи битах больше вероятности двух ошибок в трёх битах. Но мы в любом случае действуем в предположении малой вероятности ошибки, поэтому будем пренебрегать этой разницей.

Обобщим теперь всё вышесказанное. Так как каждый бит кодового слова является линейной функцией от битов сообщения, то процесс кодирования есть применение некоторого линейного оператора, который описывается матрицей G : $t = Gm$, где t — кодовое слово, а m — исходное сообщение. Матрица G называется *генератором* кода.

П р и м е р 3.2. Генератор (7, 4)-кода Хэмминга.

Для (7, 4)-кода Хэмминга

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}.$$

Таким образом, кодирование слова $m = 0101$ осуществляется следующим образом:

$$t = Gm = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

Отметим, что в нашем определении линейных кодов у G всегда будет сверху единичная подматрица.

Для декодирования используются так называемые *синдромы*. Синдром — это разница между реальным сигналом и сигналом, вычисленным на основании полученных битов сообщения. Если $t = Gs$, и $G = \begin{bmatrix} I_k \\ P \end{bmatrix}$, то синдром $z = Hr$, где $H = \begin{bmatrix} -P & I_{n-k} \end{bmatrix}$, где I_k и I_{n-k} — единичные матрицы размера $k \times k$ и $(n - k) \times (n - k)$ соответственно.

П р и м е р 3.3. Применение синдромов для (7,4)-кода Хэмминга. _____

Для (7,4)-кода Хэмминга

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Например, если передавалось сообщение $t = 0101101$, и оно было передано без ошибок ($r = t$), то

$$z = Hr = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Если же при передаче была внесена ошибка, и было принято сообщение $r = 0101100$, то синдром равен:

$$z = Hr = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Пусть t — корректное кодовое слово, не содержащее ошибок. Тогда $t = Gs$ для некоторого сообщения s . Тогда

$$Ht = HGs = \begin{bmatrix} -P & I_{n-k} \end{bmatrix} \begin{bmatrix} I_k \\ P \end{bmatrix} s$$

Нетрудно видеть, что в таком случае $Ht = 0$.

Пусть теперь t содержит одну ошибку, в бите с номером...

§ 3.3. Коды и решётки

В этом разделе мы введём специальную конструкцию, которая позволит нам в дальнейшем описать эффективные алгоритмы решения задачи декодирования.

Итак, что же мы на самом деле имеем, когда начинаем декодирование? На вход алгоритму декодирования поступает вектор y , который представляет собой сумму кодового слова и шума:

$$y = Gs + n.$$

Задача декодирования синдрома — это задача поиска *наиболее вероятного* вектора шума n , удовлетворяющего уравнению

$$Hn = z.$$

Как мы уже говорили в § 3.1, мы будем искать кодовое слово с максимальным правдоподобием $p(y|t)$.

Поставим коду в соответствие его *решётку* (trellis). Решётка — это ориентированный граф, вершины которого разделены на несколько групп (их иногда называют *временами*, times), причём каждая дуга соединяет вершину из времени $i-1$ с вершиной из времени i . Крайнее левое и крайнее правое времена имеют по одной вершине. Каждая дуга решётки помечена одним из символов из алфавита, над которым мы рассматриваем кодовые слова.

Такая решётка определяет код: каждое кодовое слово соответствует пути из крайнего левого в крайнее правое время, а развилки в решётке происходят там, где происходит выбор между разными кодовыми словами.

Решётка, соответствующая линейному коду, также называется *линейной*. Далее мы ограничимся линейными решётками. На рис. ?? приведена линейная решётка, соответствующая повторяющему коду.

Решётку можно рассматривать как модель вероятностного процесса, которым получается кодовое слово. Каждое разветвление решается при помощи случайных битов (на самом деле — битов сообщения). Задача теперь превращается в такую: дана последовательность меток на рёбрах с шумом, а найти нужно наиболее вероятную исходную последовательность или наиболее вероятное значение того или иного бита.

Воспользуемся тем, что $p(y|t) = \prod_{i=1}^n p(y_i|t_i)$. Значит, $\log p(y|t) = \sum_{i=1}^n \log p(y_i|t_i)$, причём указанную сумму необходимо максимизировать. Поменяв знак, будем минимизировать сумму величин $-\log p(y_i|t_i)$. Для этого просто снабдим ребра весами $-\log p(y_i|t_i)$ и будем искать путь с минимальным весом.

Поскольку решётка является ациклическим графом, для поиска кратчайшего пути применимо динамическое программирование. В следующем параграфе мы опишем схемы алгоритмов (фактически, алгоритмов динамического программирования), которые решают задачу декодирования.

§ 3.4. Алгоритм min–sum и алгоритм sum–product

Для поиска минимума величины $\sum_{i=1}^n -\log p(y_i|t_i)$ мы будем использовать обычную схему динамического программирования. Нужно найти путь минимальной длины в графе.

Будем использовать передачу сообщений между узлами. Каждый узел, как только он получит сообщение от всех своих предков, может передать сообщение о своей

Рис. 3.1. Решётка повторяющего кода.

цене всем своим потомкам, с учетом нового задействованного ребра. Когда этот процесс закончится, все узлы будут знать свою минимальную стоимость, в том числе и целевой узел.

Отметим, что это на самом деле распределённый алгоритм — в узлах могут стоять независимые машины, причём довольно слабые.

Min-Sum(i):

1. Дождаться получения сообщений от всех узлов, входящих в $pa(i)$.
2. Пусть $m = \min_{j \in pa(i)} p_{ij}$.
3. Для каждого узла $j \in ch(i)$ передать сообщение $p_{ij} = m + w_{ij}$.

Рис. 3.2. Min-sum алгоритм.

Приведём более формальное описание этого алгоритма. Обозначим через $pa(i)$ множество узлов решётки, из которых идут дуги в узел i , через $ch(i)$ — множество узлов, в которые идут дуги из i -ого, через p_{ij} — сообщение, которое i -ый узел передал j -ому, а через w_{ij} — вес дуги из i -ого узла в j -ый. На рис. 3.2 приведён алгоритм работы одного узла.

Результатом работы алгоритма будет значение m в узле, соответствующем последнему временному слою. Чтобы найти сам кратчайший путь, нужно восстановить его из значения в последнем узле, т.е. для каждого узла на пути найти тот из его родителей, который приводит к этому минимальному значению.

П р и м е р 3.4. Применение min-sum алгоритма для декодирования «parity code».

Рассмотрим решетку (рис. ??), соответствующую так называемому «parity code». Этот код используется для кодирования слова из двух битов с помощью слова из трёх битов. При этом, если исходное слово есть s_1s_2 , то ему соответствует кодовое слово $t_1t_2t_3$, определяемое соотношениями $t_1 = s_1$, $t_2 = s_2$, $t_3 = s_1 \oplus s_2$.

Пусть по каналу связи было получено слово 010. Пусть используемый канал связи совершает ошибку в первом бите с вероятностью 0.05, во втором — с вероятностью 0.1, в третьем — с вероятностью 0.15. Тогда имеет место распределение вероятностей $p(y_i|t_i)$, изображённое на рис. ??.

Перейдя к логарифмам вероятностей, взятых со знаком минус, получаем взвешенный ориентированный граф (рис. ??).

Опишем теперь работу алгоритма.

— На первом шаге узел 0 посылает два сообщения:

$$p_{01} = 2.995, \quad p_{02} = 0.051.$$

— На втором шаге узлы 1 и 2 посылают сообщения:

$$p_{13} = 5.297, \quad p_{14} = 3.1, \quad p_{23} = 0.156, \quad p_{24} = 2.353.$$

— На третьем шаге узел 3 посылает сообщение

$$p_{35} = \min(p_{13}, p_{23}) + 1.897 = 0.156 + 1.897 = 2.053,$$

а узел 4 посылает сообщение

$$p_{45} = \min(p_{14}, p_{24}) + 0.162 = 2.353 + 0.162 = 2.515.$$

Рис. 3.3. Решётка, соответствующая «parity code».

Таким образом, вес кратчайшего пути равен 2.053. Восстанавливая этот путь, получаем $0 \rightarrow 2 \rightarrow 3 \rightarrow 5$, что соответствует переданному кодовому слову 011.

Алгоритм min-sum помогает быстро и эффективно решить задачу полного декодирования. А для решения задачи побитового декодирования применяется немного




Рис. 3.4. Решётка с вероятностями.

другой алгоритм, называемый *алгоритм sum-product*. Как видно из названия, его основное отличие от min-sum состоит в том, что операция взятия минимума заменяется на суммирование, а операция суммирования — на умножение. Кроме этого, в качестве весов на ребрах выступают не логарифмы вероятностей, взятые с противоположным знаком, а сами вероятности.

Рис. 3.5. Взвешенный граф.

Пусть необходимо определить вероятность того, что i -ый бит кодового слова равен 1. Для этого рассмотрим все рёбра, соединяющее i -ое время с $(i + 1)$ -м, метка на которых равна 1. Пусть $\{(u_{1,1}, v_{1,1}), (u_{1,2}, v_{1,2}), \dots, (u_{1,k}, v_{1,k})\}$ — множество этих рёбер. Тогда вероятность того, что i -ый бит кодового слова равен 1 равна вероятности

Sum-Product-Forward(i):

1. Дождаться получения сообщений от всех узлов, входящих в $pa(i)$.
2. Если i — узел, соответствующий нулевому времени, то $\alpha_i = 1$,
иначе $\alpha_i = \sum_{j \in pa(i)} p_{ij}$.
3. Для каждого узла $j \in ch(i)$ передать сообщение $p_{ij} = \alpha_i \cdot w_{ij}$.

Рис. 3.6. Прямой проход sum-product алгоритма.

Sum-Product-Backward(i):

1. Дождаться получения сообщений от всех узлов, входящих в $children(i)$.
2. Если i — узел, соответствующий конечному времени, то $\beta_i = 1$,
иначе $\beta_i = \sum_{j \in children(i)} p_{ij}$.
3. Для каждого узла $j \in parents(i)$ передать сообщение $p_{ij} = \beta_i \cdot w_{ij}$.

Рис. 3.7. Обратный проход sum-product алгоритма.

того, что случайный путь из первого времени в последнее проходит через одно из перечисленных ребер.

§ 3.4.1. Прямой проход sum-product алгоритма. Сообщение на первом шаге $\alpha_0 = 1$, сообщение на очередном шаге $\alpha_i = \sum_{j \in pa(i)} w_{ij} \alpha_j$. В итоге прямого прохода сообщение, полученное каждым из узлов i с временной координатой l — это совместная вероятность того, что путь кодового слова прошел через этот узел, а первые l символов кодового слова были y_1, \dots, y_l .

Приведём более формальное описание алгоритма.

§ 3.4.2. Обратный проход sum-product алгоритма. Другой набор сообщений в это время отправляется справа налево. Сообщение на первом шаге $\beta_0 = 1$, сообщение на очередном шаге $\beta_j = \sum_{i \in ch(j)} w_{ij} \beta_i$. Эти сообщения пропорциональны *условным* вероятностям того, что, при условии что путь кодового слова прошел через вершину i , последующие символы были равны y_{l+1}, \dots, y_n .

Приведем более формальное описание алгоритма.

§ 3.4.3. Завершающий шаг алгоритма. Как говорилось ранее, чтобы найти вероятность того, что n -й бит был равен 0 или 1, нужно найти две суммы. Пусть i пробегает узлы времени n , j — узлы времени $n - 1$, t_{ij} — значение t_n , стоящее на

ребре решетки от узла j к узлу i . Тогда

$$r_n^{(0)} = \sum_{i,j \in \text{parents}(i), t_{ij}=0} \alpha_j w_{ij} \beta_i, \quad r_n^{(1)} = \sum_{i,j \in \text{parents}(i), t_{ij}=1} \alpha_j w_{ij} \beta_i.$$

Эти числа после нормализации (деления на $r_n^{(0)} + r_n^{(1)}$) суммы и дадут искомые вероятности.

Приведем пример работы Sum-product алгоритма.

П р и м е р 3.5. Применение sum-product алгоритма для поиска наиболее вероятного символа.

Рассмотрим снова решетку (рис. ??), соответствующую так называемому «parity code».

Пусть по каналу связи было получено слово 010. Пусть используемый канал связи совершает ошибку в первом бите с вероятностью 0.05, во втором — с вероятностью 0.1, в третьем — с вероятностью 0.15. Тогда имеет место следующее распределение вероятностей $p(y_i | t_i)$ (рис. ??).

Теперь задача состоит не в восстановлении наиболее вероятного переданного слова, а в поиске наиболее вероятного значения, например, второго символа. Опустим детальное описание прямого и обратного прохода, так как они аналогичны min-sum алгоритму.

В результате прямого и обратного прохода алгоритма были получены следующие значения α_i и β_i : $\alpha_0 = 1$, $\alpha_1 = 0.05$, $\alpha_2 = 0.95$, $\alpha_3 = 0.86$, $\alpha_4 = 0.14$, $\alpha_5 = 0.248$, $\beta_0 = 0.248$, $\beta_1 = 0.78$, $\beta_2 = 0.22$, $\beta_3 = 0.15$, $\beta_4 = 0.85$, $\beta_5 = 1$,

Найдем теперь $r_2^0 = \alpha_1 \cdot \beta_3 \cdot w_{13} + \alpha_2 \cdot \beta_4 \cdot w_{24} = 0.05 \cdot 0.15 \cdot 0.1 + 0.95 \cdot 0.85 \cdot 0.1 = 0.0815$ и $r_2^1 = \alpha_1 \cdot \beta_4 \cdot w_{14} + \alpha_2 \cdot \beta_3 \cdot w_{23} = 0.05 \cdot 0.85 \cdot 0.9 + 0.95 \cdot 0.15 \cdot 0.9 = 0.1665$.

После нормализации получаем: $r_2^0 = 0.3286$, $r_2^1 = 0.6714$. Таким образом, наиболее вероятным вторым переданным символом является 1.

§ 3.4.4. Некоторые замечания. Sum-product алгоритм, также как и min-sum алгоритм, является распределенным. Кроме этого, он, на самом деле, является просто частным случаем общего алгоритма вывода в байесовских сетях.

Глава 4

Скрытые марковские модели

§ 4.1. Марковские цепи и процессы

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 предыдущего; 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 переходов между состояниями $p(x' = i | x = j)$.

Мы будем рассматривать только дискретные задачи. С помощью марковской модели можно моделировать марковский процесс, при этом подразумевается, что мы можем наблюдать какие-то функции от марковского процесса.

Скрытые марковские модели предполагают, что мы не можем обычно получить сами состояния, т.е. мы не знаем, сколько этих состояний и какие между ними связи, — это всё неизвестные параметры модели.

Нам известны лишь наблюдаемые величины $y(t)$, которые зависят от скрытых переменных $x(t)$ (см. рис. 4.1). Наша задача заключается в том, чтобы определить скрытые параметры процесса. Иными словами, по имеющимся данным $y(t)$ необходимо понять, каковы наиболее вероятные $x(t)$ и наиболее вероятная модель этого марковского процесса.

Главное свойство — следующее состояние зависит только от предыдущего и не зависит от истории. Формально это значит, что вероятность $x(t)$ при условии всех остальных установленных значений равна вероятности $x(t)$ при условии только предыдущего значения:

$$p(x(t) = x_j | x(t-1) = x_{j_{t-1}}, \dots, x(1) = x_{j_1}) = p(x(t) = x_j | x(t-1) = x_{j_{t-1}}).$$

Более того, вероятности $a_{ij} = p(x(t) = x_j | x(t-1) = x_i)$ ещё и от времени t не зависят. Эти вероятности составляют матрицу перехода $A = (a_{ij})$.

Для вероятностей перехода a_{ij} выполняются следующие естественные свойства:

- $a_{ij} \geq 0$;
- $\sum_j a_{ij} = 1$ (если мы начнём в состоянии i , то куда-нибудь мы точно попадём).

§ 4.2. Постановка задачи

Прямая задача заключается в определении того, с какой вероятностью выпадет та или иная последовательность событий $Q = q_{i_1} \dots q_{i_k}$. Если Q — это последовательность состояний марковской цепи, то отыскание её вероятности не составляет никакого труда:

$$p(Q|\text{модель}) = p(q_{i_1})p(q_{i_2}|q_{i_1}) \dots p(q_{i_k}|q_{i_{k-1}}).$$

Но реальные задачи гораздо сложнее. Сложность в том, что никто нам не скажет, что модель должна быть именно такой. Мы обычно наблюдаем не $x(t)$, т.е. реальные состояния модели, а $y(t)$, т.е. некоторую функцию от них (данные).

Давайте приведём пример.

П р и м е р 4.1. Подбрасывание монетки.

Предположим, что кто-то бросает монетку и сообщает нам результаты — последовательность орлов и решек. Но при этом подбрасывание монетки происходит где-то там, в другой комнате, поэтому мы только знаем, что есть определённая последовательность битов.

Если он бросает одну монетку, то модель будет выглядеть следующим образом (см. рис. 4.2а). Будет два состояния: 0 (выпал орёл) и 1 (выпала решка). С вероятностью $1-p$ выпадает орёл, с вероятностью p — решка. Это одна модель.

Но мы же можем подумать, что у него две монетки! Он кидает монетку. Когда у него выпадает решка, он переходит на другую монетку. Когда у него выпадает орёл на другой монетке, он возвращается к подбрасыванию первой монетки. В данном случае уже две вероятности: p и q . Модель уже совершенно другая (см. рис. 4.2б). В ней по-прежнему два состояния, но параметров больше. А если три монетки?..

Наша задача в том числе и в том, чтобы понять, какая из этих моделей лучше соответствует известным данным.

Рассмотрим формулировки трёх основных задач.

1. Найти вероятность последовательности наблюдений в данной модели. Например, для модели с одной монеткой (рис. 4.2а) найти вероятность последовательности ОРРОО (О — орёл, Р — решка).
2. Найти «оптимальную» последовательность состояний при условии данной модели и данной последовательности наблюдений.
3. Найти наиболее правдоподобную модель (параметры модели). Например, найти параметры p и q для модели на рис. 4.2б.

§ 4.3. Обозначения в скрытых марковских моделях

Введём обозначения для состояний и наблюдаемых:

- $X = \{x_1, \dots, x_n\}$ — множество состояний;
- $V = \{v_1, \dots, v_m\}$ — алфавит, из которого мы выбираем наблюдаемые y (множество значений y);
- q_t — состояние во время t , y_t — наблюдаемая во время t .

Данные будем обозначать через $D = d_1 \dots d_T$ (последовательность наблюдаемых, d_i принимают значения из V).

Для вероятностей и распределений будем использовать следующие обозначения:

- $a_{ij} = p(q_{t+1} = x_j | q_t = x_i)$ — вероятность перехода из i в j ;
- $b_j(k) = p(v_k | x_j)$ — вероятность получить данные v_k в состоянии j ;
- начальное распределение $\pi = \{\pi_j\}$, $\pi_j = p(q_1 = x_j)$.

Модель $\lambda = (A, B, \pi)$ состоит из матрицы перехода A , матрицы наблюдаемых B и начального распределения π .

На рис. § 6.4 показано, как работает скрытая марковская модель (Hidden Markov Model, HMM). Таким алгоритмом можно выбрать случайную последовательность наблюдаемых.

§ 4.4. Задачи формально

Теперь можно формализовать постановку задач.

- Первая задача: по данной модели $\lambda = (A, B, \pi)$ и последовательности D найти $p(D|\lambda)$. Фактически, это нужно для того, чтобы оценить, насколько хорошо модель подходит к данным.
- Вторая задача: по данной модели λ и последовательности D найти «оптимальную» последовательность состояний $Q = q_1 \dots q_T$. Как и раньше, будет два решения: «побитовое» и общее.
- Третья задача: оптимизировать параметры модели $\lambda = (A, B, \pi)$ так, чтобы максимизировать $p(D|\lambda)$ при данном D (найти модель максимального правдоподобия). Эта задача — главная, в ней и заключается обучение скрытых марковских моделей.

Перейдём к рассмотрению решений этих задач.

§ 4.5. Первая задача

Формально, первая задача выглядит так. Требуется найти вероятность данных D при условии модели λ . Кроме явных переменных D у нас есть ещё скрытые переменные Q . Давайте сделаем их явными. Для этого нам придётся по ним просуммировать

$$p(D|\lambda) = \sum_Q p(D|Q, \lambda)p(Q|\lambda).$$

Вероятность $p(Q|\lambda)$ последовательности $Q = q_1 \dots q_T$ при условии λ — это вероятность того, что мы выберем первый элемент, а потом будем переходить от первого ко второму и т.д. Чтобы подсчитать вероятность D при условии последовательности Q , мы на каждом шаге вычисляем вероятности получения соответствующих данных. Каждая последовательность — это отдельное событие. Они не пересекаются, поэтому мы просто берём сумму по всем этим событиям:

$$p(D|\lambda) = \sum_Q p(D|Q, \lambda) p(Q|\lambda) = \sum_{q_1, \dots, q_T} b_{q_1}(d_1) \dots b_{q_T}(d_T) \pi_{q_1} a_{q_1 q_2} \dots a_{q_{T-1} q_T}.$$

Решение напоминает нам задачу маргинализации. Есть «большая» функция, которая представляется в виде «большого» произведения, и требуется её просуммировать по части переменных. Мы воспользуемся так называемой forward-backward procedure, по сути — вычислением на решётке, как в декодировании. Будем последовательно вычислять промежуточные величины вида

$$\alpha_t(i) = p(d_1 \dots d_t, q_t = x_i | \lambda),$$

т.е. искомые вероятности, но ещё с учётом текущего состояния. Величина $\alpha_t(i)$ показывает вероятность всех имеющихся данных вплоть до t и того, что мы пришли в состояние x_i на шаге t в данной модели.

Тогда искомая вероятность $p(D|\lambda)$ представляется в виде

$$p(D|\lambda) = \sum_{i=1}^n \alpha_T(i).$$

Вспомогательные величины $\alpha_t(i)$ можно вычислять с помощью динамического программирования (см. рис. § 5.4). Сначала инициализируем $\alpha_1(i)$ произведением начального распределения π_i и вероятности получить d_1 . Затем выполняем шаг индукции. Для вычисления $\alpha_{t+1}(j)$ необходимо подсчитать сумму по всем предыдущим состояниям вероятностей достичь этого состояния и перейти в состояние j , после чего умножить эту сумму на вероятность получить в j -м состоянии новое данное d_{t+1} . После того как дойдём до шага T , просуммируем и найдём величину $p(D|\lambda)$.

Фактически, это только прямой проход, обратный нам здесь не понадобился. Обратный проход вычислял бы условные вероятности $\beta_t(i) = p(d_{t+1} \dots d_T | q_t = x_i, \lambda)$ — вероятности получения цепочки $d_{t+1} \dots d_T$, начиная из состояния i на шаге t . Их можно вычислить, проинициализировав $\beta_T(i) = 1$, а затем по индукции:

$$\beta_t(i) = \sum_{j=1}^n a_{ij} b_j(d_{t+1}) \beta_{t+1}(j).$$

Это нам пригодится чуть позже, при решении второй и третьей задачи.

§ 4.6. Вторая задача

Как мы уже упоминали, возможны два варианта.

1. Решать «побитово», отвечая на вопрос, каково наиболее вероятное состояние во время j .
2. Решать задачу нахождения наиболее вероятной последовательности состояний.

Эти варианты решения задачи принципиально различаются. Если взять наиболее вероятные состояния во время j и составить из них последовательность, то можем получить некорректную последовательность (содержит переход, вероятность которого равна 0).

Приведём побитовое решение задачи. Для этого нам потребуются вспомогательные переменные

$$\gamma_t(i) = p(q_t = x_i | D, \lambda).$$

$\gamma_t(i)$ — это вероятность того, что мы придём на t -м шаге в i -е состояние при условии имеющихся данных и заданной модели. Наша задача — найти

$$q_t = \operatorname{argmax}_{1 \leq i \leq n} \gamma_t(i), \quad 1 \leq t \leq T.$$

Для нахождения значения q_t , которое максимизирует $\gamma_t(i)$, применяется вариант min-sum алгоритма.

Выражаем $\gamma_t(i)$ через α и β :

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{p(D|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^n \alpha_t(i)\beta_t(i)}.$$

На знаменатель можно не обращать внимания, поскольку нам нужен argmax .

Чтобы ответить на вопрос о наиболее вероятной последовательности, мы будем использовать уже знакомый алгоритм Витерби, то есть, по сути, то же самое динамическое программирование.

Теперь нам понадобятся новые вспомогательные переменные

$$\delta_t(i) = \max_{q_1, \dots, q_{t-1}} p(q_1 q_2 \dots q_t = x_i, d_1 d_2 \dots d_t | \lambda).$$

$\delta_t(i)$ — максимальная вероятность достичь состояния x_i на шаге t среди всех путей с заданными наблюдаемыми.

По индукции получаем

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij} \right] b_j(d_{t+1}).$$

Кроме того, потребуется запоминать аргументы, а не только значения; для этого будет массив $\psi_t(j)$.

На рис. § 4.6 приведён алгоритм решения задачи относительно последовательности. Сначала проинициализируем $\delta_1(i)$ начальными вероятностями, а $\psi_1(i)$ — пустым массивом. Затем по индукции вычисляем значения $\delta_t(j)$ и аргументы $\psi_t(j)$, на которых достигаются максимумы. Когда мы дойдём до шага T , подсчитаем p^* (оно нам не понадобится) и q_T^* . После этого, возвращаясь обратно по массиву ψ , будем считать аргументы.

§ 4.7. Третья задача

Задача состоит в нахождении параметров модели a_{ij} , $b_j(k)$, π_i при условии имеющихся данных.

Отметим, что аналитически найти глобальный максимум $p(D|\lambda)$ у нас никак не получится. Зато мы рассмотрим итеративную процедуру (по сути — градиентный подъём), которая приведёт к локальному максимуму этой функции. У функции может быть несколько локальных максимумов, поэтому на практике этот алгоритм следует запускать несколько раз, чтобы он попадал в разные элементы пространства и мог бы привести к разным максимумам. Алгоритм, который мы будем рассматривать, называется алгоритмом Баума–Велха (Baum–Welch algorithm). Он является на самом деле частным случаем алгоритма ЕМ. При наличии некоторой модели с некоторыми параметрами согласно алгоритму ЕМ (Estimation–Maximization algorithm) мы сначала считаем ожидание данных при условии этой модели, потом поправляем модель при условии имеющихся данных. В ходе такого итеративного процесса мы приходим к наилучшей модели.

Теперь нашими вспомогательными переменными будут вероятности того, что мы во время t в состоянии x_i , а во время $t + 1$ — в состоянии x_j :

$$\xi_t(i, j) = p(q_t = x_i, q_{t+1} = x_j | D, \lambda).$$

Выражение для $\xi_t(i, j)$ можно переписать через уже знакомые переменные:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}{p(D|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{ij} b_j(d_{t+1}) \beta_{t+1}(j)}.$$

Отметим также, что $\gamma_t(i) = \sum_j \xi_t(i, j)$.

$\sum_t \gamma_t(i)$ — это ожидаемое количество переходов из состояния x_i , а $\sum_t \xi_t(i, j)$ — из x_i в x_j . Теперь на шаге M мы будем переоценивать вероятности:

$$\begin{aligned} \bar{\pi}_i &= \text{ожидаемая частота появления } x_i \text{ на шаге } 1 = \gamma_1(i), \\ \bar{a}_{ij} &= \frac{\text{количество переходов из } x_i \text{ в } x_j}{\text{количество переходов из } x_i} = \frac{\sum_t \xi_t(i, j)}{\sum_t \gamma_t(i)}, \\ \bar{b}_i(k) &= \frac{\text{количество появлений в } x_i \text{ и наблюдений } v_k}{\text{количество появлений в } x_i} = \frac{\sum_{t: d_t = v_k} \gamma_t(i)}{\sum_t \gamma_t(i)}. \end{aligned}$$

ЕМ-алгоритм приведёт к цели: начинаем с какой-то модели $\lambda = (A, B, \pi)$, вычисляем вспомогательные переменные, подсчитываем $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$, снова пересчитываем параметры и т.д.

§ 4.8. Обоснование алгоритма Баума–Велха

Расстояние Кульбака–Лейблера (Kullback–Leibler distance, divergence) — это информационно-теоретическая мера того, насколько далеки распределения друг от друга:

$$D_{KL}(p_1, p_2) = \sum_x p_1(x) \log \frac{p_1(x)}{p_2(x)}.$$

Известно, что это расстояние всегда неотрицательно, равно нулю тогда и только тогда, когда p_1 и p_2 совпадают в каждой точке: $p_1 \equiv p_2$.

Определим распределения p_1 и p_2 на множествах последовательных состояний для моделей λ и λ' следующим образом:

$$p_1(Q) = \frac{p(Q, D|\lambda)}{p(D|\lambda)}, \quad p_2(Q) = \frac{p(Q, D|\lambda')}{p(D|\lambda')}.$$

Тогда расстояние Кульбака–Лейблера для распределений p_1 и p_2 :

$$\begin{aligned} 0 \leq D_{LK}(\lambda, \lambda') &= \sum_Q p_1(Q) \log \frac{p_1(Q)}{p_2(Q)} = \sum_Q \frac{p(Q, D|\lambda)}{p(D|\lambda)} \log \frac{p(Q, D|\lambda)p(D|\lambda')}{p(Q, D|\lambda')p(D|\lambda)} = \\ &= \log \frac{p(D|\lambda')}{p(D|\lambda)} + \sum_Q \frac{p(Q, D|\lambda)}{p(D|\lambda)} \log \frac{p(Q, D|\lambda)}{p(Q, D|\lambda')}. \end{aligned}$$

Введём вспомогательную функцию

$$\mathcal{Q}(\lambda, \lambda') = \sum_Q p(Q|D, \lambda) \log p(Q|D, \lambda').$$

Тогда из неравенства следует, что

$$\frac{\mathcal{Q}(\lambda, \lambda') - \mathcal{Q}(\lambda, \lambda)}{p(D|\lambda)} \leq \log \frac{p(D|\lambda')}{p(D|\lambda)}.$$

Если $\mathcal{Q}(\lambda, \lambda') > \mathcal{Q}(\lambda, \lambda)$, то $0 \leq \log \frac{p(D|\lambda')}{p(D|\lambda)}$, следовательно $p(D|\lambda') > p(D|\lambda)$. Значит, если мы максимизируем $\mathcal{Q}(\lambda, \lambda')$ по λ' , мы тем самым будем двигаться в нужную сторону.

Итак, требуется максимизировать $\mathcal{Q}(\lambda, \lambda')$ по λ' . Перепишем:

$$\begin{aligned} \mathcal{Q}(\lambda, \lambda') &= \sum_Q p(Q|D, \lambda) \log p(Q|D, \lambda') = \sum_Q p(Q|D, \lambda) \log \pi_{q_1} \prod_t a_{q_{t-1} q_t} b_{q_t}(d_t) = \\ &= \sum_Q p(Q|D, \lambda) \log \pi_{q_1} + \sum_Q p(Q|D, \lambda) \sum_t \log a_{q_{t-1} q_t} + \sum_Q p(Q|D, \lambda) \sum_t \log b_{q_t}(d_t). \end{aligned}$$

Последнее выражение легко продифференцировать по a_{ij} , $b_i(k)$ и π_i , добавить соответствующие множители Лагранжа и решить. Покажем, что получится именно пересчёт по алгоритму Баума–Велха.

Поскольку параметры π_i , a_{ij} и $b_i(k)$, по которым выполняется оптимизация, независимо входят в каждое из трёх слагаемых выражения для $Q(\lambda, \lambda')$, можно оптимизировать каждое из этих слагаемых по-отдельности.

Рассмотрим первое слагаемое:

$$\sum_Q p(Q|D, \lambda) \log \pi_{q_1} = \sum_{i=1}^n p(q_1 = x_i | D, \lambda) \log \pi_i.$$

Добавляя множитель Лагранжа μ и используя ограничение $\sum_i \pi_i = 1$, запишем выражение для производной и приравняем его к нулю:

$$\frac{\partial}{\partial \pi_i} \left(\sum_{i=1}^n p(q_1 = x_i | D, \lambda) \log \pi_i + \mu \left(\sum_{i=1}^n \pi_i - 1 \right) \right) = 0.$$

Продифференцировав это выражение по π_i , просуммировав по i с использованием равенства $\sum_i \pi_i = 1$ для отыскания μ и решив относительно π_i , получаем

$$\pi_i = \frac{p(q_1 = x_i | D, \lambda)}{p(D | \lambda)}.$$

Перепишем второе слагаемое суммы $Q(\lambda, \lambda')$:

$$\sum_Q p(Q|D, \lambda) \sum_t \log a_{q_{t-1} q_t} = \sum_{i=1}^n \sum_{j=1}^n \sum_t p(q_t = x_i, q_{t+1} = x_j | D, \lambda) \log a_{ij}.$$

Вводя множитель Лагранжа и применяя ограничение $\sum_{j=1}^n a_{ij} = 1$, получаем

$$a_{ij} = \frac{\sum_t p(q_t = x_i, q_{t+1} = x_j | D, \lambda)}{\sum_t p(q_t = x_i | D, \lambda)} = \frac{\sum_t \xi_t(i, j)}{\sum_t \gamma_t(i)}.$$

Наконец, третье слагаемое суммы $Q(\lambda, \lambda')$ запишем в виде

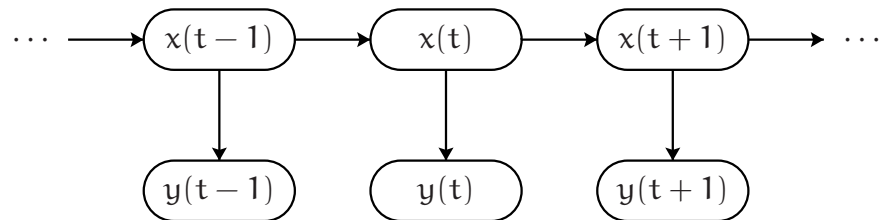
$$\sum_Q p(Q|D, \lambda) \sum_t \log b_{q_t}(d_t) = \sum_{i=1}^n \sum_t p(q_t = x_i | D, \lambda) \log b_i(d_t).$$

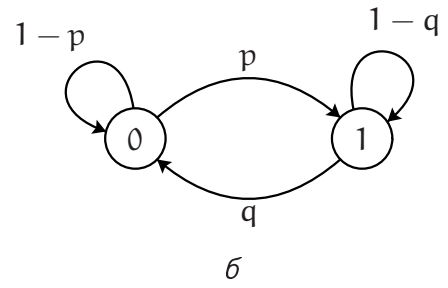
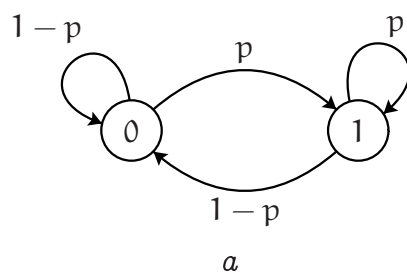
Снова применяем метод множителей Лагранжа с ограничением $\sum_j b_i(j) = 1$. Заметим, что только наблюдаемые d_t , равные v_k , вносят вклад в результирующее значение для вероятности:

$$b_i(k) = \frac{\sum_{t: d_t = v_k} p(q_t = x_i | D, \lambda)}{\sum_t p(q_t = x_i | D, \lambda)} = \frac{\sum_{t: d_t = v_k} \gamma_t(i)}{\sum_t \gamma_t(i)}.$$

§ 4.9. Упражнения и задачи

1. Реализовать систему обучения скрытых марковских моделей, которая умела бы решать все три вышеописанные задачи.





HMMEmulation(λ):

1. Выберем начальное состояние x_1 по распределению π .
2. Для всех t от 1 до T :
 - а) Выберем наблюдаемую d_t по распределению $b_j(k) = p(v_k|x_j)$.
 - б) Выберем следующее состояние по распределению $a_{ij} = p(q_{t+1} = x_j|q_t = x_i)$.

Рис. 4.3. Алгоритм эмуляции скрытой марковской модели.

DynamicHMMObservationSequenceProbability(λ, D):

1. Инициализировать $\alpha_1(i) = \pi_i b_i(d_1)$.
2. Шаг индукции:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^n \alpha_t(i) a_{ij} \right] b_j(d_{t+1}).$$

3. После того как дойдём до шага T , подсчитаем искомую вероятность:

$$p(D|\lambda) = \sum_{i=1}^n \alpha_T(i).$$

Рис. 4.4. Алгоритм нахождения вероятности последовательности наблюдений в данной модели.

DynamicHMMOptimalStateSequence(λ, D):

1. Инициализировать $\delta_1(i) = \pi_i b_i(d_1)$, $\psi_1(i) = []$.
2. Шаг индукции:

$$\delta_t(j) = \max_{1 \leq i \leq n} [\delta_{t-1}(i) a_{ij}] b_j(d_t),$$

$$\psi_t(j) = \operatorname{argmax}_{1 \leq i \leq n} [\delta_{t-1}(i) a_{ij}].$$

3. После шага T вычислить:

$$p^* = \max_{1 \leq i \leq n} \delta_T(i), \quad q_T^* = \operatorname{argmax}_{1 \leq i \leq n} \delta_T(i).$$

4. Вычислить последовательность: $q_t^* = \psi_{t+1}(q_{t+1}^*)$.

Рис. 4.5. Алгоритм нахождения наиболее вероятной последовательности состояний.

Глава 5

Байесовская кластеризация динамических процессов

§ 5.1. Постановка задачи

Пусть есть несколько динамических процессов, имеющиеся данные о которых представимы в виде набора точек (t, s) , где t – момент времени ($t = 0..T$, где T – момент времени, до которого наблюдался процесс), а s – одно из возможных состояний процесса (t и s принимают конечные дискретные наборы значений). На рис. 1 и 2 приведены примеры графиков подобных процессов. Исследуемая задача состоит в разбиении некоторого числа имеющихся наблюдений на непересекающиеся кластеры. В качестве примера таких процессов можно привести электрокардиограммы сердца. Если на вход алгоритму кластеризации подать набор кардиограмм, то он должен разбить их на кластеры, соответствующие различным патологиям сердца. В рассматриваемом алгоритме данные о процессах преобразуются в цепи Маркова, «похожие» цепи объединяются до получения разбиения на кластеры с наибольшей апостериорной вероятностью.

§ 5.2. Марковские цепи

Формально цепь Маркова можно определить как совокупность $\langle X, P, p_0 \rangle$, где $X = 1..s$ – набор состояний, P – матрица размерности $s \times s$ вероятностей переходов (p_{ij} определяет вероятность перехода из состояния i в состояние j), p_0 – вектор размерности s , задающий распределение вероятности первого состояния. Таким образом, множества $\langle X, P, p_0 \rangle$ задают процесс, который в каждый момент времени t может находиться в одном из s состояний из множества X . Состояние процесса в момент времени $t + 1$ определяется случайным испытанием, при этом вероятность того, что процесс находясь в момент времени t в состоянии x_t , перейдет в состояние j , не зависит от предыдущей истории процесса (x_0, \dots, x_{t-1}) :

$$p(x_{t+1} = j | (x_0, \dots, x_t)) = p(x_{t+1} = j | x_t) = p_{x_t j}$$

Таким образом, вероятность того, каки будет следующее состояния полностью определяется текущим состоянием и задается матрицей P .

Рассматриваемый алгоритм исходит из предположения, что кластеризуемый набор данных $S = \{S_i\}$ ($S_i = (x_0^i, \dots, x_T^i)$ будем называть наблюдением) порожден несколькими марковскими процессами.

Пусть есть наблюдение (x_0, \dots, x_T) . В предположении, что наблюдался марковский процесс, построим аппроксимацию матрицы P этого процесса. Определим n_{ij} как общее число переходов из состояния i в состояние j , а n_i – как общее число посещений состояния i в имеющемся наблюдении. Это апостериорные данные об исследуемом марковском процессе, то есть данные, полученные опытным путем. Кроме апостериорных данных в рассматриваемую модель так же вводятся априорные данные, которые можно рассматривать как информацию об исследуемом марковском процессе, полученную из «общих соображений». Эти априорные знания задают числа α_{ij} и α_i (называемые гипер-параметрами), которые можно интерпретировать как данные, полученные из проведенных ранее наблюдений, в которых число переходов из состояния i в состояние j было пропорционально α_{ij} и число посещений состояния i было пропорционально α_i .

Байесовская оценка \hat{p}_{ij} элемента p_{ij} матрицы исследуемого марковского процесса определяется формулой:

$$\hat{p}_{ij} = \frac{\alpha_{ij} + n_{ij}}{\alpha_i + n_i},$$

которую можно переписать как

$$\hat{p}_{ij} = \frac{\alpha_{ij}}{\alpha_i} \frac{\alpha_i}{\alpha_i + n_i} + \frac{n_{ij}}{n_i} \frac{n_i}{\alpha_i + n_i}$$

То есть \hat{p}_{ij} – взвешенное среднее двух оценок – основанной только на апостериорных данных и основанной только на априорных данных.

В таблице 1 приведены значения частот и апостериорных оценок матрицы переходов для процесса, график которого изображен на рис. 1. Значения гипер-параметров $\alpha_{ij} = 0.2$.

N =		1	2	3	4	5	=> \hat{P} =		1	2	3	4	5
	1	3	12	3	0	3		1	0.15	0.55	0.15	0.01	0.15
	2	11	1	2	2	0		2	0.66	0.07	0.13	0.13	0.01
	3	6	0	0	0	0		3	0.78	0.03	0.03	0.03	0.15
	4	0	0	2	0	0		4	0.07	0.07	0.73	0.07	0.07
	5	0	4	0	0	0		5	0.04	0.84	0.04	0.04	0.84

§ 5.3. Модель с максимальной апостериорной вероятностью

Рассматриваемый алгоритм подразумевает, что наблюдения, объединенные в один кластер, порождены одним и тем же марковским процессом, матрица переходов которого аппроксимируется способом, аналогичным описанному выше:

$$\hat{p}_{kij} = \frac{\alpha_{kij} + n_{kij}}{\alpha_{ki} + n_{ki}},$$

где n_{kij} – общее число переходов из состояния i в состояние j во всех наблюдениях в k -ом кластере, $n_{ki} = \sum_{j=1}^s n_{kij}$ – общее число посещений состояния i в k -ом кластере, α_{kij} и α_{ki} – гипер-параметры для k -ого кластера. В рассматриваемом алгоритме, как правило, используется следующий подход: изначально всем α_{kij} присваивается значение $\alpha/(ms^2)$ (m – общее число наблюдений), а при объединении двух кластеров гипер-параметры для нового кластера получаются сложением гипер-параметров исходных.

Рассматриваемый алгоритм, получая на вход набор наблюдений $S = \{S_i\}$ ($i = 1..m$), ищет такое разбиение C этих наблюдений на кластеры, которому соответствует модель M_c , обладающая максимальной апостериорной вероятностью. Под моделью здесь подразумевается предположение о том, что данные, принадлежащие каждому концертному кластеру, порождены марковским процессом, соответствующим этому кластеру.

Таким образом, задача сводится к поиску разбиения C , максимизирующего величину $p(M_c|S)$. Применим к этой величине формулу Байеса:

$$p(M_c|S) = \frac{p(M_c)p(S|M_c)}{p(S)},$$

где $p(S)$ – маргинальная вероятность получения набора данных S , $p(M_c)$ – априорная вероятность справедливости модели M_c , $p(S|M_c)$ – вероятность получения набора данных S при испытаниях, при условии что модель M_c справедлива (то есть данные генерируются марковскими процессами, соответствующими этой модели). Поскольку сравнение моделей происходит на одном и том же наборе данных S и все модели априори эквивалентны, то задача сводится к максимизации величины $p(S|M_c)$. Пусть модель C содержит s кластеров, а m_k – число наблюдений, принадлежащих кластеру C_k . Можно показать, что вероятность $p(S|M_c)$ представляется в виде:

$$p(S|M_c) = f(S, C)f(S, X_{t-1}, X_t, C).$$

Величину $f(S, C)$ можно определить как вероятность получения разбиения C данных S при распределении m наблюдений по c кластерам:

$$f(S, C) = \frac{\Gamma(\alpha)}{\Gamma(\alpha + m)} \prod_{k=1}^c \frac{\Gamma(\alpha_k + m_k)}{\Gamma(\alpha_k)}$$

Эта формула станет понятнее, если предположить отсутствие априорных данных: тогда рассматриваемая величина есть ничто иное, как $\frac{m_1! \dots m_c!}{m!}$, то есть вероятность получить разбиение при случайном распределении m наблюдений по c кластерам, так что в k -ый кластер попадает m_k наблюдений.

$f(S, X_{t-1}, X_t, C)$ - величина, определяющая достоверность данных S при условии, что кластеров c и каждое наблюдение отнесено к одному и только одному из кластеров:

$$f(S, X_{t-1}, X_t, C) = \prod_{k=1}^c \prod_{i=1}^s \frac{\Gamma(\alpha_{ki})}{\Gamma(\alpha_{ki} + n_{ki})} \prod_{j=1}^s \frac{\Gamma(\alpha_{kij} + n_{kij})}{\Gamma(\alpha_{kij})}$$

Если отбросить априорные данные, то данная величина – произведение по всем кластерам величины $\frac{n_{ki1}! \dots n_{kis}!}{n_{ki}!}$, то есть вероятности получения данных, идентичных данным S при случайном размещении n_{ki} переходов из состояния i в k -ом кластере по другим состояниям (в которые осуществляется переход), при условии что в j -ое состояние ведет ровно n_{kij} переходов.

§ 5.4. Эвристический поиск модели с максимальной апостериорной вероятностью

Наивный алгоритм поиска модели с максимальной апостериорной вероятностью – полный перебор всех возможных разбиений на кластеры, что требует экспоненциального времени по отношению к числу наблюдений во входных данных. В рассматриваемом алгоритме используется эвристика, позволяющая не перебирать все возможные разбиения.

Пусть P_1 и P_2 – матрицы вероятностей переходов двух марковских процессов. *Kullback-Liebler* расстояние между двумя строками этих матриц определяется как:

$$d(p_{1i}, p_{2i}) = \sum_{j=1}^n p_{1ij} \log \frac{p_{1ij}}{p_{2ij}}.$$

Симметризуя определенное таким способом расстояние получим: $D(p_{1i}, p_{2i}) = (d(p_{1i}, p_{2i}) + d(p_{2i}, p_{1i}))/2$. Расстояние между двумя марковскими процессами (а значит, и между двумя кластерами) определим как $D(P_1, P_2) = \sum_{i=1}^s D(p_{1i}, p_{2i})$. Заметим, что, в принципе, в рассматриваемом алгоритме можно использовать и любую другую меру расстояния между марковскими процессами.

```

procedure BCD(S)
01  MCs:= $\emptyset$ ; Distances:= $\emptyset$ ;
02  while  $i \leq |S|$  do MCs:=MCs $\cup\{S_i\}$ 
03  while  $i \leq |MCs|$  do
04    while  $j \leq |MCs|$  do
05      if  $i \neq j$  then Distances:=Distances $\cup\{\text{DISTANCE}(MC_{s_i}, MC_{s_j})\}$ ;
06    end;
07  end;
08  Distances:=SORT(Distances);
09   $P_{\text{new}} = \text{ML}(MCs)$ ; Best:=MCs;
10  while  $P_{\text{new}} > P_{\text{old}}$  do
11     $P_{\text{old}} := P_{\text{new}}$ ;
12    while  $i < |Distances|$  do
13      MC:=MERGE(Distances $_i$ );
14      Current:=MCs $\setminus\{Distances_i\} \cup \{MC\}$ 
15      if  $\text{ML}(\text{Current}) < P_{\text{new}}$  then
16        Best:=Current;  $P_{\text{new}} := \text{ML}(\text{Current})$ ;
17        while  $j \leq |Distances|$  do
18          if  $Distances_i \cap Distances_j \neq \emptyset$  then
19            Distances:=Distances $\setminus Distances_i$ ;
20          end
21          while  $j \leq |Best|$  do
22            Distances:=Distances $\cup \text{DISTANCE}(MC, Best_j)$ ;
23          end
24          Distances:=SORT(Distances);
25          break
26        endif
27      end
28    end
29    return Best;
end

```

Рис. 5.1. Алгоритм нахождения вероятности последовательности наблюдений в данной модели.

На вход алгоритму подается множество S из m наблюдений. На выходе алгоритма – некоторое разбиение B множества S . Приведенны

Алгоритм инициализируется (строки 01 и 02) разбиением на s кластеров, каждый из которых содержит ровно одно наблюдение из множества S . Переменная MCs обозначает текущее разбиение на кластеры. Также во время инициализации заполняется и затем сортируется массив $Distances$, хранящий расстояния между всеми парами кластеров. Процедура $SORT()$ сортирует этот массив по возрастанию относительно расстояний. Условие в цикле в строке 10 говорит о том, что алгоритм продолжает работу до тех пор, пока текущее разбиение на кластеры можно улучшить. Улучшение выполняется следующим образом: берется ближайшая пара кластеров (первый элемент массива $Distances$) и сливается в один кластер. Переменная $Current$ обозначает получившееся в результате слияния разбиение на кластеры. Если апостериорная вероятность (подсчитываемая при помощи функции $ML()$) у нового разбиения лучше, чем у старого, то старое разбиение замещается новым. При этом производится необходимое обновление массива $Distances$ (строки 18-23). Если новое разбиение не лучше старого, то происходит попытка слияния следующих двух кластеров из массива $Distances$ и т.д. Если ни одно из возможных слияний не улучшило ситуацию, алгоритм заканчивает работу.

Иерархический Байесовский Алгоритм Кластеризации

§ 6.1. Краткий Обзор

Классификация текстов, группировка текстов в несколько кластеров, нужна для увеличения эффективности поиска текстов и их категоризации. В данной статье представлен иерархический алгоритм который создает множество кластеров с максимальной Байесовской апостериорной вероятностью, вероятностью того, что данные тексты сгруппированны в кластеры. Алгоритм называется Иерархическим Байесовским Алгоритмом Кластеризации (Hierarchical Bayesian Clustering - *HBC*).

Его преимущества:

- HBC может построить оригинальные кластеры более аккуратно, чем другие не вероятностные алгоритмы
- Когда вероятностная категоризации текстов расширяется до основанной на кластерах, использование HBC дает лучшую эффективность и производительность, чем его не вероятностные аналоги

§ 6.2. Введение

Классификация текстов, группировка их в несколько кластеров используется для улучшения качества и эффективности категоризации текстов и их поиска. Для примера, чтобы извлечь тексты соответствующие запросу пользователя, простейшей стратегией будет найти все тексты в базе данных посчитав среднее соответствие каждого документа запросу. Так или иначе для больших баз данных данный поиск будет требовать значительных вычислений. Классификация текстов помогает уменьшить количество расчетов, произведя их только для каждого конкретного представителя кластера. Кластеризация также может использоваться для улучшения качества выборки, но это утверждение пока не проверялось. Модель которая включает в себя предварительное деление информации на кластеры обычно называется кластер ориентированной.

В данной работе представлен вероятностный алгоритм иерархической кластеризации и приведено его сравнение с другими алгоритмами кластеризации. Практически все предыдущие алгоритмы, такие как метод Варда, используют расчет расстояния между двумя объектами и объединение ближайших. Представленный здесь алгоритм,

создает множество кластеров которое имеет наибольшую Байесовскую апостериорную вероятность, вероятность того, что данные объекты классифицированы в кластеры. Максимизация данной задачи является более общей формой хорошо известной задачи наибольшей похожести, и алгоритм называется алгоритмом Иерархической Байесовской Кластеризации.

§ 6.3. Алгоритм Варда

Мы приведем алгоритм Варда, чтобы вы могли сравнить его с подходом приведенным в данной главе.

Согласно *алгоритму Варда* (*Ward's Criterion*) две пары сегментов текста объединяются на каждом шаге с целью минимизации приращения несоответствия кластеров.

Каждый кластер i представляется как L -размерный вектор $(x_{i_1}, x_{i_2}, \dots, x_{i_L})$, где каждый x_{i_k} это значение для слова $tf \cdot idf$ (*term frequency* - отношение числа вхождений слова в документ к длине документа, *Inverse Document Frequency* - это логарифм отношения числа всех документов к числу документов содержащих слово).

Если m_i - количество объектов в кластере i , возведенное в квадрат Евклидово расстояние между двумя сегментами i и j будет вычисляться как:

$$d_{ij}^2 = \sum_{k=1}^L (x_{ik} - x_{jk})^2 \quad (6.1)$$

Тогда когда два сегмента объединяются *приращение несоответствия* I_{ij} вычисляется как:

$$I_{ij} = \frac{m_i \cdot m_j}{m_i + m_j} \cdot d_{ij}^2 \quad (6.2)$$

Процесс объединения кластеров продолжается до тех пор, пока соединение любых двух кластеров не дестабилизирует весь массив существующих на данный момент кластеров, произведенных на предыдущих шагах. На каждом шаге, два кластера x_{ik} и x_{jk} выбираются так, чтобы приращение I_{ij} было минимально. При этом последнее представляется как процент изменения несоответствия на предыдущем шаге. Если данный процент достигает ранее установленного значения порога, это означает, что ближайшие два кластера значительно дальше друг от друга в сравнении с предыдущим шагом; таким образом объединение последних двух принесет дестабилизирующее изменение и не должно иметь место.

Так в кратце выглядит алгоритм Варда.

§ 6.4. Иерархическая Байесовская Кластеризация

Как и многие агломерирующие алгоритмы кластеризации, НВС содает кластер иерархически (древовидная схема) с низу в верх, объединяя два кластера за шаг. В начале (на нижнем уровне древовидной структуры), каждый объект информации принадлежит кластеру, единственным членом которого и является. Для каждой пары кластеров НВС считает вероятность объединения пары и выбирает для следующего объединения наилучшую (для которой вероятность объединения максимальна). Данные шаги объединения происходят $N-1$ раз для коллекции из N объектов информации. Последний шаг объединения дает один кластер состоящий из всех объектов информации.

ATTENTION: а сюда я хотел картинку pdf/diagram1.pdf

Формально, НВС выбирает пару кластеров для объединения с учетом максимальной апостериорной вероятности $P(C|D)$, где D - множество документов ($D = d_1, d_2, \dots, d_N$), а C - множество кластеров ($C = c_1, c_2, \dots, c_N$). Каждый кластер c_j является множеством документов и все кластеры взаимно непересекаются. На первом шаге - каждый кластер является множеством одного элемента; $c_i = d_i$ для всех i . $P(C|D)$ определяет вероятность того, что коллекция документов D будет определена в множество кластеров C . Максимизация $P(C|D)$ - это обобщение задачи наибольшего правдоподобия.

Чтобы изучить процесс более подробно - рассмотрим шаг объединения $k+1$, ($0 \leq k \leq N-1$). На шаге $k+1$, множество документов D было поделено на множество кластеров C_k . Каждый документ $d \in D$ принадлежит кластеру $c \in C_k$. Рассмотрим апостериорную вероятность на данном этапе:

$$P(C_k|D) = \prod_{c \in C_k} \prod_{d \in c} P(c|d) = \prod_{c \in C_k} \prod_{d \in c} \frac{P(d|c) \cdot P(c)}{P(D)} = \prod_{c \in C_k} \frac{P(c)^{|c|}}{P(D)} \cdot \prod_{c \in C_k} \prod_{d \in c} P(d|c) \quad (6.3)$$

Произведя два переобозначения:

$$\frac{PC(C_k)}{P(D)} \cdot \prod_{c \in C_k} SC(c) \quad (6.4)$$

Здесь $PC(C_k)$ является априорной вероятностью, что N случайных документов будут распределены в множество кластеров C_k . Ее можно определить следующим образом:

$$PC(C_k) = \prod_{c \in C_k} P(c)^{|c|} \quad (6.5)$$

$SC(c)$ является вероятностью того, что все документы в кластере c получаются из кластера и определяется следующим образом:

$$SC(c) = \prod_{d \in c} P(d|c) \quad (6.6)$$

Когда алгоритм объединяет два кластера $c_x, c_y \in C_k$, множество кластеров C_k обновляется следующим образом:

$$C_{k+1} = C_k - \{c_x, c_y\} + \{c_x \cup c_y\} \quad (6.7)$$

После объединения, апостериорная вероятность индуктивно пересчитывается следующим образом:

$$P(C_{k+1}|D) = \frac{PC(C_{k+1})}{PC(C_k)} \cdot \frac{SC(c_x \cup c_y)}{SC(c_x) \cdot SC(c_y)} \cdot P(C_k|D) \quad (6.8)$$

Хочется обратить внимание, что это обновление локально, и очень эффективно, потому что все что нужно пересчитать - вероятность для нового, созданного объединением кластера; то есть $SC(c_x \cup c_y)$. Что до множителя $\frac{PC(C_{k+1})}{PC(C_k)}$, мы используем хорошо известную оценку: априорная вероятность модели (в данном случае кластера) является убывающей функцией от размера модели. Для примера $P(c) \propto A^{-|c|}$ для некоторой константы $A > 1$. В соответствии с данной оценкой:

$$PC(C) = \prod_{c \in C} P(c)^{|c|} \propto \prod_{c \in C} A = A^{|C|} \quad (6.9)$$

Так как количество кластеров $|C|$ уменьшается на единицу на каждом шаге объединения, $\frac{PC(C_{k+1})}{PC(C_k)}$ уменьшается на постоянную величину A^{-1} независимо от объединяемой пары. Это означает, что мы можем опустить данный множитель для задачи максимизации. НВС рассчитывает обновленный $P(C_{k+1}|D)$ для каждого кандидата на объединение и создает того, который предлагает наибольшее значение $P(C_{k+1}|D)$.

§ 6.5. Пример вычисления $P(d|c)$

В заключение мы приведем пример расчета элементарной вероятности $P(d|c)$ того что кластер c порождает свой документ d . В зависимости от представления данных и доступной информации о данных могут быть различные способы расчета этой вероятности.

П р и м е р 6.1. Кластеризация текстов.

Допустим каждая единица данных d - документ, и представлен как множество термов (как правило только существительные используются как термы). Поскольку кластер c является множеством документов, он также представлен как множество термов, которые содержатся во всех документах в нем содержащихся. Положим теперь событие $T = t$ - "произвольно изъятый терм T из множества термов равен t ". Тогда можем вычислить $P(d|c)$ как:

HBCGeneralAlgorithm(D):

1. Входные данные: $D = \{d_1, d_2, \dots, d_N\}$: Множество входных документов;
2. Инициализация $C_0 = \{c_1, c_2, \dots, c_N\}$: Множество кластеров;
 $c_i = \{d_i\}$ for $1 \leq i \leq N$
 рассчитать $SC(c_i)$ for $1 \leq i \leq N$
 рассчитать $SC(c_i \cup c_j)$ for $1 \leq i < j \leq N$
3. Шаг цикла: for $k = 1$ to $N - 1$ do
 $(c_x, c_y) = \operatorname{argmax}_{c_x, c_y} \frac{SC(c_x \cup c_y)}{SC(c_x) \cdot SC(c_y)}$
 $C_k = C_{k-1} - c_x, c_y + c_x \cup c_y$
 вычислить $SC(c_x \cup c_z)$ для всех $c_z \in C_k$ где $z \neq x$
4. Функция: $SC(c)$ вернуть $\prod_{d \in c} P(d|c)$

Рис. 6.1. Алгоритм Иерархической Байесовской Кластеризации

$$P(d|c) = \sum_t P(d|c, T = t) \cdot P(T = t|c) \quad (6.10)$$

Если мы полагаем условную независимость между c и d , при $T = t$, we obtain

$$P(d|c) = \sum_t P(d|T = t) \cdot P(T = t|c) \quad (6.11)$$

Теперь используя теорему Байеса, перепишем последнее:

$$P(d|c) = P(d) \cdot \sum_t \frac{P(T = t|d) \cdot P(T = t|c)}{P(T = t)} \quad (6.12)$$

Так как $P(d)$ появляется в каждой оценке $P(C|D)$ только единожды, $P(d)$ можно опустить в целях максимизации. Остальные вероятности $P(T = t|d)$, $P(T = t|c)$ и $P(T = t)$ рассчитываются на основе имеющейся информации по следующему:

- $P(T = t|d)$: относительная частота термина t в документе d
- $P(T = t|c)$: относительная частота термина t в кластере c
- $P(T = t)$: относительная частота термина t во всем множестве данных

Хочется заметить, что есть основания полагать, что в задачах классификации текстов, *HBC* будет работать лучше чем алгоритм Варда, так как стратегия построения кластеров в *HBC* напрямую связана с задачами подобных приложений, против алгоритма Варда.