Байесовское декодирование

Сергей Николенко

Академический Университет, весенний семестр 2011

Outline

Суть

- Что такое коды, исправляющие ошибки (error–correcting codes)?
- Это коды, которые умеют даже по неправильному кодовому слову достаточно часто выдавать правильное сообщение.
- Формально — какая задача стоит перед декодером?

Задача декодирования

- *Задача полного декодирования*: по сигналу понять, какое кодовое слово передавалось.
- *Задача побитового декодирования*: для каждого передаваемого бита t_n понять, насколько вероятно, что это был 0 или 1.
- Мы сейчас решим её в байесовской постановке.

Задача декодирования

- Обозначим: t — кодовое слово, y — полученный сигнал.

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

- Если канал не имеет памяти, то $p(y|t) = \prod_{i=1}^{n} p(y_i|t_i)$.
- Априорное распределение $p(t)$ считаем равномерным (почему?).
- Знаменатель $p(y) = \sum_t p(y|t)p(t)$.

Решение задачи декодирования

- Полное решение — список всех кодовых слов и их вероятностей при условии данного сигнала.
- Но на самом деле нам столько не нужно, нам нужно просто найти самое вероятное кодовое слово.
- *Задача декодирования с максимальным правдоподобием* (MAP codeword decoding problem) — задача поиска наиболее вероятного кодового слова t при условии данного сигнала y. При равномерном априорном распределении эта задача сводится к максимизации правдоподобия $p(y|t)$.

Решение задачи побитового декодирования

- 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 только что научились решать эту задачу полным перебором. Но в байесовских сетях мы уже умели делать это поумнее.
- Так будем и сейчас, но сначала придётся вспомнить о том, что же такое кодирование.

Идея кодов, исправляющих ошибку

- Есть канал, который может портить передаваемые сигналы.
- Нужно научиться передавать сообщения так, чтобы получатель мог расшифровать их, даже если случится где-то ошибка.
- Как это сделать?

Повторяющий код

- Простейший код — повторять каждый бит три раза; повторяющий код (repetition code).
- Тогда, даже если одна ошибка в тройке идущих подряд битов попадётся, мы всё равно раскодируем правильно, взяв большинство ответов.
- Защищает от однократной ошибки, не защищает от двух ошибок в одной тройке битов.

Упражнение. Докажите, что решение большинством голосов в этом случае реализует максимальную апостериорную гипотезу, если вероятность ошибки в одном бите меньше 1/2.

Недостатки повторяющего кода

- Главный недостаток — слишком большой overhead; неэффективно.
- При n повторениях вероятность ошибки равна

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

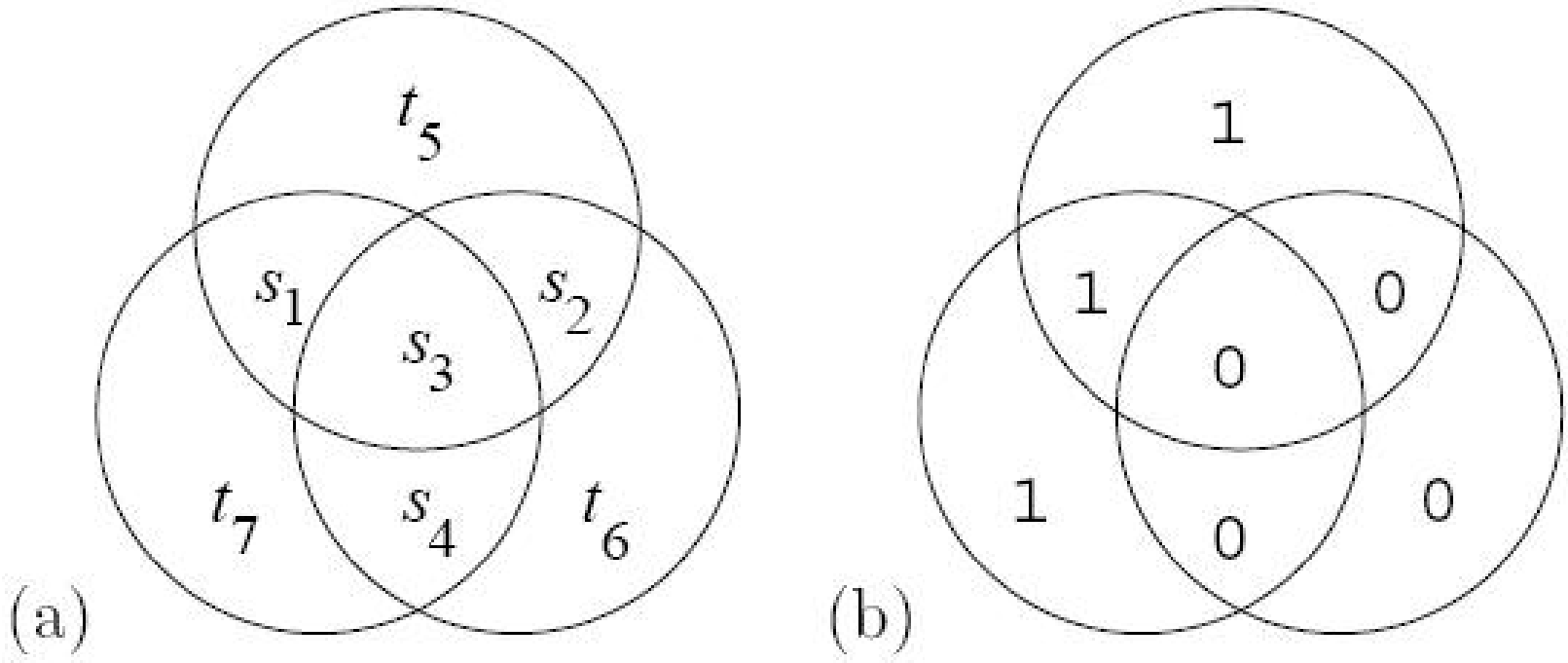
где α — вероятность ошибки в одном бите.

Линейные коды

- Обобщим. Пусть у нас блок размера k переходит в блок размера n при кодировании ($n > k$, разумеется).
- Предположим, что первые k бит кодового слова — это просто само сообщение, а оставшиеся $n - k$ — линейные функции от битов сообщения (parity checks).
- Такие коды называются *линейными*.

Пример

- Широко известен код Хэмминга $(7, 4)$ (на 4 бита сообщения 7 битов сигнала).
- Линейные функции — сумма битов сообщения, попадающих в соответствующий круг.



Пример

- Главное свойство этого кода — то, что кодовые слова отличаются друг от друга как минимум в трёх битах. То есть мы достигли того же эффекта, что и с троекратными повторениями, но битов стало больше не в 3, а в $\frac{7}{4}$ раза.
- На самом деле не вполне того же самого; в чём разница?
- Как декодировать?

Линейный код в общем виде

- Кодовое слово получается в виде $t = G^t s$, где G — матрица, называющаяся *генератором* кода.
- Например, для $(7, 4)$–кода Хэмминга

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

- В нашем определении у G всегда будет сверху единичная подматрица.

Синдромы

- Для декодирования используются так называемые *синдромы*. Синдром — это разница между реальным сигналом и сигналом, вычисленным на основании полученных битов сообщения.
- Если $t = G^t s$, и $G = \begin{bmatrix} I_k \\ P \end{bmatrix}$, то синдром $z = Hr$, где $H = [-P\ I_{n-k}]$. Например, для $(7, 4)$–кода Хэмминга

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

- Для всякого валидного кодового слова t $Ht = 0$. Докажите!

Постановка задачи декодирования

- Получаемый вектор r — это сумма кодового слова и шума:

$$r = G^t s + n.$$

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

$$Hn = z.$$

- Её-то мы и будем решать.

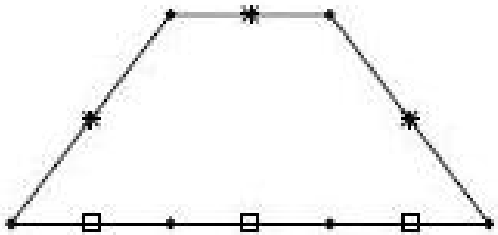
Outline

Решётки

- Поставим коду в соответствие его решётку (trellis). Решётка — это такой граф, вершины которого разделены на несколько групп (*времён*, times), причём каждое ребро соединяет вершину из времени i с вершиной времени $i-1$ или $i+1$. Крайнее левое и крайнее правое времена имеют по одной вершине.
- Такая решётка определяет код: кодовое слово соответствует пути из левого в правое время.
- Если код линейный, решётка тоже линейная; отныне у нас все решётки линейные.

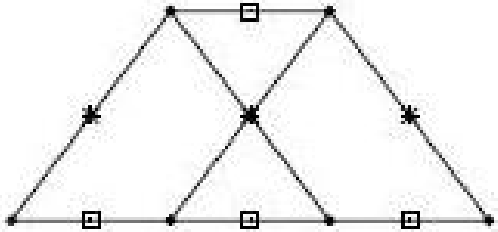
Примеры

(a)



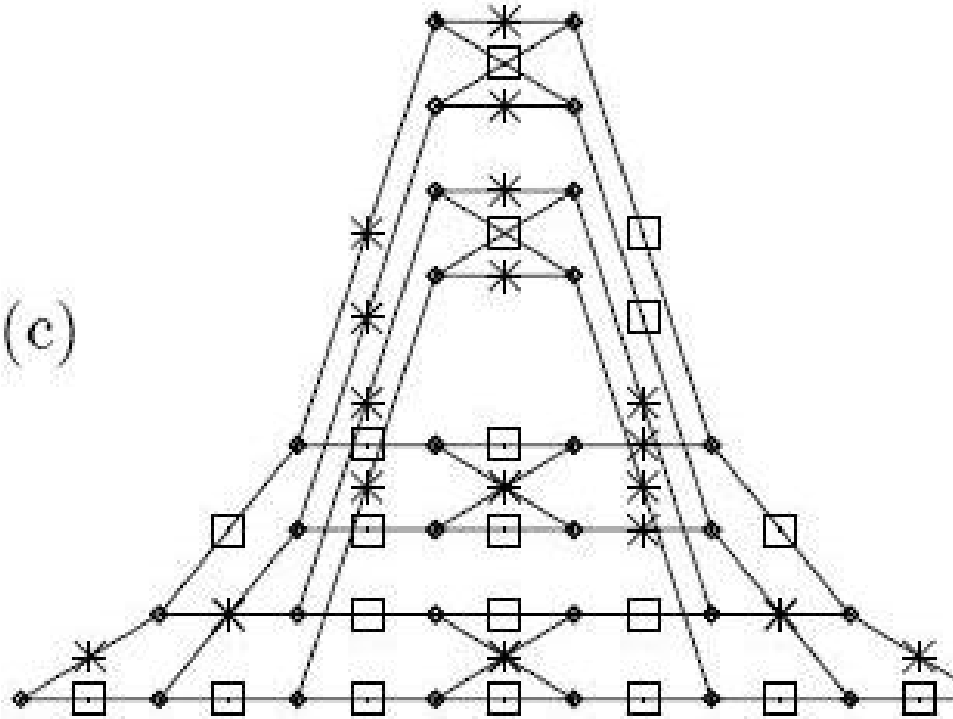
Repetition code R_3

(b)



Simple parity code P_3

(c)



$(7, 4)$ Hamming code

Решётки и кодовые слова

- Решётку можно рассматривать как модель вероятностного процесса, которым получается кодовое слово.
- Каждое разветвление решается случайным образом (по текущему биту 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.

Декодирование как поиск MAP

- Обозначим y — то, что мы получили, t — то, что хотели передать. Тогда

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

- Будем предполагать, что априорное распределение $p(t)$ равномерно (кодовые слова появляются с равной вероятностью).
- Этого, кстати, всегда можно достичь при помощи разумного кодирования.

Веса и правдоподобие

- Так как ошибки независимы, $p(y|t) = \prod_{i=1}^{n} p(y_i|t_i)$. Значит, $\log p(y|t) = \sum_{i=1}^{n} \log p(y_i|t_i)$.
- И нам нужно максимизировать эту сумму. Поменяв знак, будем минимизировать сумму величин $-\log p(y_i|t_i)$.
- Для этого просто снабдим рёбра весами $-\log p(y_i|t_i)$ и будем искать путь с минимальным весом. Как?

Min-sum algorithm

- Будем использовать передачу сообщений между узлами.
- Каждый узел, как только он получит сообщение от всех своих предков, может передать сообщение о своей цене всем своим потомкам, с учётом нового задействованного ребра.
- Когда этот процесс закончится, все узлы будут знать свою минимальную стоимость, в том числе и целевой узел.
- Это и есть min-sum algorithm, или алгоритм Витерби (Viterbi).

Побитовое декодирование

- Это мы решали задачу декодирования вообще, т.е. поиска кодового слова.
- Теперь давайте решим задачу декодирования побитового, т.е. поиска наиболее вероятного данного бита.
- Для этого придётся от min-sum перейти к min-product; т.е. считать правдоподобия $p(y_n|t_n)$, а не их логарифмы. И вместо поиска минимума будем суммировать.
- Но теперь понадобится два прохода. Рассмотрим их подробнее.

Прямой проход

- Сообщение на первом шаге $\alpha_0 = 1$, сообщение на очередном шаге
$$\alpha_i = \sum_{j \in \mathrm{pa}(i)} w_{ij} \alpha_j .$$
- В итоге первого прохода сообщение, полученное каждым из узлов i с временной координатой l — это совместная вероятность того, что путь кодового слова прошёл через этот узел, а первые l символов кодового слова были $y_1, \ldots, y_l$ (с точностью до константы).
- Это — *прямой проход* алгоритма (forward pass).

Обратный проход

- Другой набор сообщений в это время отправляется справа налево.
- Сообщение на первом шаге $\beta_0 = 1$, сообщение на очередном шаге
$$\beta_j = \sum_{i \in \mathrm{ch}(j)} w_{ij} \beta_i.$$
- А эти сообщения пропорциональны *условным* вероятностям того, что, при условии что путь кодового слова прошёл через вершину i, последующие символы были равны $y_{l+1}, \ldots, y_n$.

Последний шаг

- Итого, чтобы найти вероятность того, что n–й бит был равен 0 или 1, нужно подсчитать две суммы. Пусть i пробегает узлы времени n, j — узлы времени $n-1$, t_{ij} — значение t_n, стоящее на ребре решётки от узла j к узлу i. Тогда

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

- Эти суммы и дадут искомые вероятности; нужно только их нормализировать (разделить на $r_n^{(0)} + r_n^{(1)}$).
- Заметим, что у нас получился тот же алгоритм, что и в скрытых марковских моделях (для второй задачи).

Thank you!

Спасибо за внимание!