ФАКУЛЬТЕТ
БИОИНЖЕНЕРИИ И
БИОИНФОРМАТИКИ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ

БИОИНФОРМАТИКА

МИРОНОВ
АНДРЕЙ АЛЕКСАНДРОВИЧ

ФББ МГУ

Оглавление

Лекция 1

Вероятность и случайные величины

Является ли последовательность cuttagcattggatua случайной? А число 2,83631273679? Если после броска монеты выпадает орел, то это случайное событие?

Все это - это неслучайные вещи. Случайная последовательность, число и другие величины - это вещи, которые не случились. Если на монетке уже выпал орел, то делая ставки, мы поставим 0 на выпадение решки, поскольку орел уже выпал. Если мы уже знаем ответ, то вероятность теряет смысл. Она есть только до события.

Какова вероятность, что случайно выбранное число, например, интервале [0, 1], будет случайным? Число иррациональных чисел счетно, их можно пронумеровать. Их мало, хотя кажется, что наоборот. Поэтому случайно выбранное число скорее всего будет рациональным, хотя это противоречит математике.

Задача:

Пусть вам дают два конверта, в каждом из которых лежат деньги. В одном из конвертов денег в два раза больше, чем в другом. Нужно угадать, в каком из конвертов больше денег.

Пусть в первом конверте x денег, а во втором - либо $2x$, либо $0{,}5x$. Математическое ожидание количества денег во втором конверте следующее: $2x + 0{,}5 \cdot 0{,}5x = 1{,}25x$

Задача:

Некая фармацевтическая кампания придумала лекарство от простуды и провела клинические испытания отдельно для мужчин и для женщин.

	М		Ж	
	Лечились	Не лечились	Лечились	Не лечились
Выздоровели	50	4	10	80
Всего	90	12	12	120
	$\frac{50}{90}$	$\frac{4}{12}$	$\frac{10}{12}$	$\frac{80}{120}$
	$\frac{5}{9}$	$\frac{1}{3}$	$\frac{5}{6}$	$\frac{2}{3}$

Поскольку $\frac{5}{9} > \frac{1}{3}$ (мужчины) и $\frac{5}{6} > \frac{2}{3}$ (женщины), можно считать, что лекарство работает.

Эта кампания отправила документы в FDA, который проверил лекарство по отношению ко всем людям в целом.

Получились следующие результаты:

	Люди	
	Лечились	Не лечились
Выздоровели	60	84
Всего	102	132
	$\frac{60}{102}$	$\frac{84}{132}$
	$\frac{20}{34}$	$\frac{21}{33}$

Поскольку $\frac{20}{34} < \frac{21}{33}$, то в данном случае лекарство не работает!

Великий Лаплас говорил, что "вероятность - это здравый смысл и немного математики". Математическая теория вероятностей правильная и применима ко многим задачам (например, на ней основаны статистические тесты), но ей не достает здравого смысла.

Теория вероятностей важна для азартных игр, а также для страхового дела. Что общего между этими сферами? Во-первых, случайность, во-вторых, - деньги.

Вероятность с точки зрения здравого смысла - это та ставка, которую мы готовы поставить. Классическая вероятность называется *частотной*. В ее основе лежит представление о том, что величина вероятности равна отношению числа успехов к числу испытаний (в пределе, когда число испытаний стремиться к бесконечности).

Рассмотрим простой пример:

На высоте около 100 метров натянут крепкий канат. Мы хотим оценить, что человек сможет пройти по этому канату с одного берега на другой. Сколько мы сделаем испытаний? Явно не бесконечное число. Именно поэтому ситуация с проведением бесконечного числа испытаний не является жизненной. Ее можно приближенно применить к таким вещам, как подкидывание монеты или игральных костей.

Таким образом, частотное представление о вероятности, хоть и является теоретически верным, но часто противоречит здравому смыслу.

Допустим, необходимо застраховать дом от падения метеорита. Страховая кампания начинает задумываться о том, как оценить вероятность того, что на наш дом упадет метеорит. В истории было очень малое количество событий, когда на дом падал метеорит. А домов очень много. Также есть зоны, где они падают чаще.

Теорема Байеса

Говоря о деньгах, отметим, что вероятность - это свойство субъекта, а не объекта. Объективная вероятность существует только в одной области - в квантовой механике

(распад радиоактивных частиц), там она не зависит от субъекта. Мы будем делать из субъективной вероятности объективную.

Рассмотрим простой пример:

Мы бросили монету 3 раза ($N = 3$), при этом 3 раза выпал орел ($n = 3$). Какова вероятность того, что в следующем испытании выпадет орел?

Монета пришла из некого пространства монет, о котором у нас есть некие представления. Мы вынимаем из него монету. У нас есть вероятность того, что произвольно выбранная монета будет иметь некий перекос:

$P(Po)$ – перекос в сторону орла, именно к этой категории относится монета из задачи (это вероятность, что вероятность выпадения орла будет равна Po)

D – наши наблюдения

$P(Po|D)$ - вероятность того, что будет некая определенная вероятность выпадения орла при условии наблюдения. Основываясь на наших наблюдениях, мы хотим узнать свойство монеты - вероятность выпадения орла Po.

По *теореме Байеса*:

$$P(Po|D) = \frac{P(D|Po) \cdot P(Po)}{P(D)} \tag{1}$$

Из формулы (1) мы знаем $P(D|Po)$. Это не является вероятностью, поскольку наблюдение D уже случилось, а вероятность того, что уже случилось равна 1. Однако, это не совсем так. Несмотря на то, что наблюдение случилось, если бы мы проводили серию таких экспериментов, эта величина не была бы равна 1. Поэтому данная величина называется *правдоподобием*. Это вероятность наблюдения, когда мы сделали вид, что наблюдения не было, или какая была бы вероятность наблюдения при условии, что наблюдения еще не было.

Вероятность - это та оценка, которую мы делаем до наблюдения. Она отличается от правдоподобия тем, что при ней мы не представляем, что наблюдения не было. Следует различать тонкую грань между этими двумя понятиями. Правдоподобие - это вероятность *увидеть* событие.

Если бы у нас была вероятность Po и исход из условия задачи, то наше правдоподобие было бы равно следующему:

$$P(Po|D) = \frac{P(D|Po) \cdot P(Po)}{P(D)} = \frac{Po^3 \cdot P(Po)}{P(D)}$$

- априорное распределение вероятности

Монеты могут быть правильными или неправильными, от этого будет зависеть их распределение вероятности. Это наше представление о том, как устроены монеты.

Po^3 нам известна, $P(Po)$ мы задаем самостоятельно, а $P(D)$ легко посчитать. $P(Po|D)$ - полное пространство событий (даже при наших зажатых условиях), поэтому:

$$\int_{Po} P(Po|D)dPo = 1 = \int_{Po} \frac{Po^3 \cdot P(Po)}{P(D)} dPo = \frac{1}{P(b)} \int_{Po} Po^3 P(Po)dPo \qquad (2)$$

$P(D)$ не зависит от Po.

Поскольку из формулы (2) $\int_{Po} Po^3 P(Po)dPo = 1$, мы получаем следующее:

$$\frac{Po^3 \cdot P(Po)}{P(D)} = \frac{Po^3 \cdot P(Po)}{\int_{Po} Po^3 P(Po)dPo}$$

Все сказанное выше называется *байесовой* статистикой, а также вероятностью или оцениванием.

δ-функция

В качестве отступления скажем, что всю "правильную"математику придумали физики. В том числе, они придумали δ-функцию. Затем возникла большая теория обобщенных функций.

Что такое δ-функция?

$$\delta(x) = \begin{cases} 0, & x \neq 0 \\ \int_{-\infty}^{+\infty} \delta(x)dx = 1 \end{cases} \qquad (3)$$

Функцию (3) можно легко представить. Возьмем формулу плотности нормального распределения:

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

Эта функция имеет следующий вид:

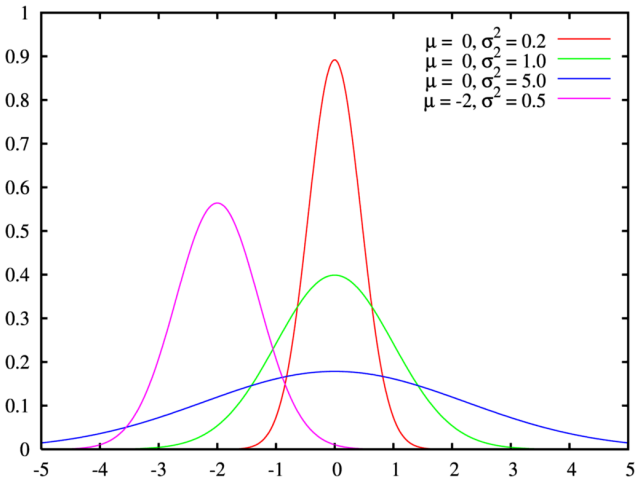

Рис. 1.1 График функции плотности нормального распределения вероятностей

В некотором смысле можно сказать следующее:

$$\delta(x) = \lim_{\sigma \to 0} n(x, \sigma)$$

Это и является функцией плотности нормального распределения, поэтому:

$$n(x,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Таким образом, график δ-функции узкий и высокий, ее интеграл равен 1. Она является примером обобщенной функции. По сути, любая интегрируемая с квадратом функция является обобщенной.

$$\int_{-\infty}^{+\infty} \delta(x-a) f(x) dx = f(a)$$

$f(x)$– любая непрерывная функция

Априорное и постериорное распределение вероятностей

$$p(\theta|D) = \frac{P(D|\theta)P(\theta)}{\int_\theta P(D|\theta)P(\theta)d\theta}$$

В частном случае θ - априорная вероятность выпадения орла $P(Po) = P(\theta) = \delta(\theta - \frac{1}{2})$, если все монеты одинаковые и правильные.

Получим математически, что если все монеты правильные, то и наша монета прямая:

$$P(\theta|D) = \frac{\theta^3 \delta(\theta - 1/2)}{\int_0^1 \theta^3 \delta(\theta - 1/2) d\theta} \tag{4}$$

delta-функция везде равна 0, кроме $1/2$. Поэтому произведение в числителе формулы (4) "живет"только тогда, когда $\theta = 1/2$.

Следовательно:

$$P(\theta|D) = \frac{1/2^3 \delta(\theta - 1/2)}{1/2^3} = \delta(\theta - 1/2)$$

Сколько бы монет мы не бросали, ответ всегда будет $1/2$ (вероятность выпадения орла).

Если бросить монету 100 раз и 100 раз выпадет орел, то можно сделать вывод, что либо с монетой что-то не так, либо нам очень не везет. То есть мы делаем вывод либо о субъекте, либо об объекте.

Если мы заранее уверены, что все монеты правильные, сколько бы опытов мы ни делали, ответ будет неизменным.

Если мы заранее не знаем, как устроены монеты (правильные они все или нет), то мы получим следующее:

$$P(\theta|D) = \frac{\theta^3 \cdot 1}{\int_0^1 \theta^3 \cdot 1 d\theta} = \frac{\theta^3}{1/4} = 4\theta^3$$

По результатам эксперимента мы получаем, что вероятность того, что монета кривая больше, чем того, что она правильная. Этот вывод получился из получившейся функции распределения вероятностей (см. рис. 1.2). Таким образом, мы меняем свою точку зрения, поскольку до эксперимента мы считали, что все монеты устроены одинаково, а после - что, скорее всего, наша монета перекошенная. Мы поменяли представление не о мире монет в целом, а о той конкретной монете, которую мы подкидывали.

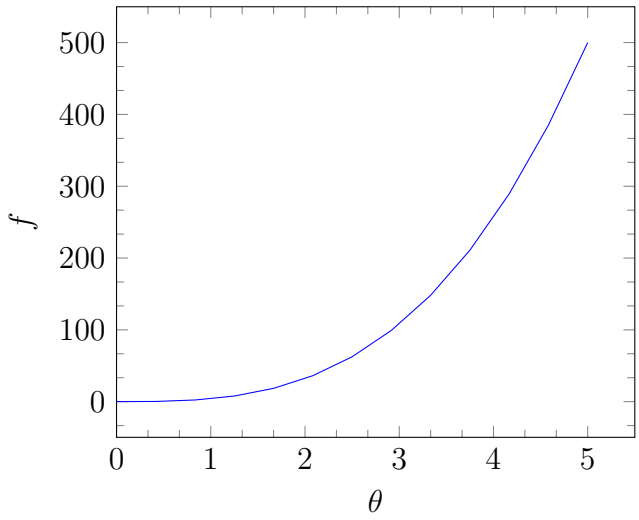

Рис. 1.2 График функции плотности апостериорного распределения вероятностей для исследуемой монеты

Апостериорное распределение вероятностей - то распределение вероятностей, которое получается по результатам опыта.

Априорное распределение - это то, что можно сказать о мнете, которую еще ни разу не подбрасывали.

Мир монет, вообще говоря, имеет распределение, похожее на нормальное, с пиком в 1/2. Оно не является нормальным, поскольку значение θ лежит в интервале от [0,1], а у нормального распределения (стандартного) такого ограничения нет.

Функция данного распределения:

$$P(\theta) = \theta^\alpha (1-\theta)^\alpha Z$$

Z – нормирующий множитель, необходимый для того, чтобы интеграл был равен 1.

Если $\alpha = 1$, то в 0 данная функция ведет себя линейно. Также она себя ведет в 1. Соответственно, график данной функции представляет собой параболу с ветвями, направленными вниз.

В зависимости от значения α мы получим разный вид графиков. Картина будет схожа с графиком, изображенным на рис. 1.1. Если $\alpha = 2$, то получится график зеленого цвета (квадратичное поведение параболы). Если $\alpha = 5$, то получим красный график. Это называется двухпараметрическое β-распределение.

β-распределение покрывает широкий спектр вполне понятных распределений.

Как будет выглядеть следующее распределение?

$$\frac{\theta^3\theta^\alpha(1-\theta^\alpha)Z}{Z\int_\theta \theta^3\theta^\alpha(1-\theta)^\alpha d\theta} = \frac{\theta^{3+\alpha}(1-\theta)^\alpha}{\int_\theta \theta^3\theta^\alpha(1-\theta)^\alpha d\theta}$$

Поведение функции справа определяется α, оно не отличается от рассмотренного выше случая. А поведение слева поменялось из-за изменения показателя степени. Именно поэтому график "прижимается"к одной из сторон - см. рис. 1.1 (фиолетовый график).

Таким образом, наш эксперимент поменял наши представления о том, как устроена наша конкретная монета.

Помимо двух вышеперечисленных представлений о мире монет можно выделить так называемое "кривое"представление. Назовем его "представлением шизофреника".

$$\frac{\theta^3\theta^\alpha(1-\theta^\beta)Z}{Z\int_\theta \theta^3\theta^\alpha(1-\theta)^\alpha d\theta} = \frac{\theta^{3+\alpha}(1-\theta)^\beta}{\int_\theta \theta^3\theta^\alpha(1-\theta)^\alpha d\theta}$$

График, как и в предыдущем случае, будет прижиматься к стороне.

Априорное распределение вероятностей - не математическое понятие. Однако, оно имеет отношение к биоинформатике и к жизни.

Допустим, у нас есть всего 5 экспериментов. Нам нужно сделать вывод по результатам этого небольшого (далеко не бесконечного) количества экспериментов. Нужно написать праер, исходя из предыдущего опыта, например, о том, как устроена в принципе экспрессия генов. По результатам экспериментов мы делаем вывод о том, что они дают в понимании данного вопроса.

Таким образом, байесово оценивание важно, когда у нас имеется малое количество наблюдений, что очень актуально в реальной жизни.

Если наблюдений много, то при любом распределении, кроме *delta*-функции, мы получим очень узкое распределение.

Величины, получаемые из байесова оценивания

По результатам байесова оценивания необходимо делать выводы, "делать ставки". То есть вместо распределения нам нужно получить некое число.

Из распределения можно получить *математическое ожидание* (среднее значение):

$$E(\theta) = \int_\theta \theta P(\theta|D) d\theta$$

Таким образом, по результатам экспериментов мы получаем математическое ожидание выпадения орла (*E-оценка*).

Возьмем простейший случай:

$$P(\theta) = 1$$

Тогда:

$$\int_\theta \theta P(\theta|D) d\theta = \frac{\int_\theta \theta^3 \cdot \theta \ \cdot 1 \cdot d\theta}{\int_\theta \theta^3 \cdot 1 \cdot d\theta} = \frac{1/5}{1/4} = \frac{4}{5}$$

Таким образом, можно делать ставки на следующие броски, исходя из определенной вероятности выпадения орла.

В результате наших опытов мы получаем определенный график распределения вероятностей. Мы можем поставить либо на получившееся математическое ожидание, либо на максимум. Во втором случае это называется *MAP-оценкой* – maximum aposteriorior probability: $MAP(\theta) = \text{org max } P(\theta|D)$.

Эту оценку легко посчитать. В нашем случае при плоском распределении она будет довольно сильно перекошена. Максимум находится в 1.

И математическое ожидание, и MAP-оценка - хорошие оценки, невозможно сказать, какая из них лучше. Однако, считать MAP-оценку проще.

Тренировочная задача

Представим, что мы бросили монету 3 раза и 2 из них выпал орел, то есть $N = 3, n = 2$.

Математически запишем, что в первый раз выпал орел, во второй раз - решка, в третий - снова орел.

$$P(\theta|D) = \theta^2(1-\theta)^1 C_3^2 = \theta^2(1-\theta) \cdot 3$$

C – число сочетаний

$$P(\theta) = 1$$

$$P(D|\theta) = \frac{P(\theta|D)P(\theta)}{P(D)} = \frac{\theta^2(1-\theta)}{\int_\theta \theta^2(1-\theta)d\theta} \tag{5}$$

Решим интеграл из знаменателя формулы (5):

$$\int_\theta (\theta^2 - \theta^3)d\theta = \frac{\theta^3}{3} - \frac{\theta^4}{4} \tag{6}$$

В постановке формулы (6) от 0 до 1 получаем следующий результат:

$$\frac{1}{3} - \frac{1}{4} = \frac{1}{12}$$

Таким образом, подставляем получившееся значение интеграла в формулу (5):

$$\frac{\theta^2(1-\theta)}{\int_\theta \theta^2(1-\theta)d\theta} = 12\theta^2(1-\theta)$$

Лекция 2

Что такое вероятность?

То, что вероятность - есть деньги, проявляется в еще одном примере - в *бирже предсказаний*. Участникам биржи по цене одного доллара продали определенное количество акций на выборах каждого кандидата. В процессе предвыборной кампании эти акции колебались в цене (это зависело от высказываний кандидатов и многих других факторов). По окончании выборов каждый получал деньги в соответствии с текущей ценой акций.

Оказалось, что биржа предсказаний дает фантастически точный ответ по процентам голосов. Участников было немного - всего около 50 человек. Соотношение голосов в точности соответствовало соотношению котировок акций на момент выборов. Однако, если перед выборами сделать соцопрос, то мы поучим +-10%.

Почему соцопрос дает гораздо менее точные предсказания, чем биржа предсказаний?

Ответ в том, что соцопрос не сопряжен с риском для человека, а в бирже предсказаний люди играют на свои деньги. После проведения эксперимента с биржей предсказаний они были запрещены законодательно, поскольку они будут оказывать слишком сильное влияние на процесс выборов.

Тем не менее, можно сделать биржи предсказаний не только на выборы, но и на другие события. Условно, можно сделать биржу предсказаний на то, случиться ли в ближайший месяц землетрясение в Японии.

Таким образом, биржа предсказаний еще раз демонстрирует нам, что вероятность - это деньги, хоть и не только. Вероятность и вся байесова наука также прописана в мозге, по крайней мере, у млекопитающих.

На собачьих собаках стоят специальные бумы для собак. В самый первый раз собака идет на него с большим страхом, поскольку она оценивает математическое ожидание потери. С третьего раза у собаки существенно меняется априорное представление, происходит сдвиг априорного распределения.

Человек хочет пройти по доске, ширина которой примерно равна ширине плеч. Пройти по такой доске не составит ему сложности. Однако, если эта доска находится между двумя домами, то человеку становится страшно, поскольку он внутренне

оценивает математическое ожидание потерь в случае, если произойдет неудача. Вероятность неблагоприятного исхода небольшая, но потери слишком большие. Именно поэтому в нашем мозгу постоянно происходит байесово переоценивание.

Самые аварийно опасные водители - со стажем от 3 до 5 лет, поскольку если стаж водителя - до 3х лет, то он едет очень осторожно и всего боится. После 3х лет уже начинает действовать своеобразный отбор. У опытных водителей со стажем от 5ти лет априорная оценка вероятности аварии очень хорошо соответствует реальному представлению о том, что происходит.

Таким образом, постоянно идет переоценивание вероятностей и возможных потерь.

MAP- и E-оценка для монеты

На предыдущей лекции мы получили формулу апостериорной вероятности:

$$P(\theta|D) = 12\theta^2(1-\theta)$$

Как получить MAP-оценку?

Продифференцируем:

$$\frac{dP}{d\theta} = 0$$

$$12(2\theta - 3\theta^2) = 0$$

$$2\theta = 3\theta^2$$

$$\theta = \frac{2}{3}, \theta \neq 0$$

MAP-оценка - это то самое значение θ, при котором вероятность достигает максимума. В данном случае она совпала с тривиальной оценкой (отношение числа успехов к числу испытаний), что, однако, бывает не во всех случаях (см. Лекцию 1).

Получим теперь E-оценку:

$$\int_\theta 12\theta^3(1-\theta)d\theta = 12\int_0^1(\theta^3 - \theta^4)d\theta = 12(\frac{1}{4} - \frac{1}{3}) = \frac{12}{20} = \frac{3}{5}$$

Таким образом, MAP $= \frac{2}{3}$, E $= \frac{3}{5}$

Байесово оценивание для игральной кости

Перейдет к более сложному объекту теории вероятностей - игральной кости.

В случае игральной кости θ - это вероятность выпадения определенной грани. У нас есть $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$.

$$\sum \theta_i = 1, \theta_i \geq 0$$

У нас есть игральная кость, и мы хотим по результатам некоторого количества бросков сделать выводы о том, как устроена данная игральная кость.

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Вообще говоря, θ - это вектор.

На каком пространстве живет эта вероятность?

Допустим, у нас 3 измерения. Наша сумма равна 1. Эта некая гиперплоскость, которая проходит через единицы. Мы получаем правильный треугольник.

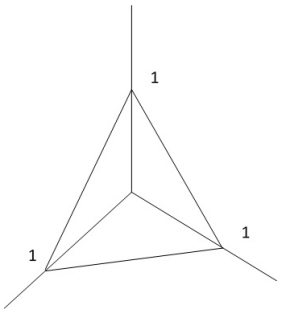

Рис. 2.1 Трехмерное вероятностное пространство

Пространство может также быть двумерным, как в случае с орлом и решкой на монете.

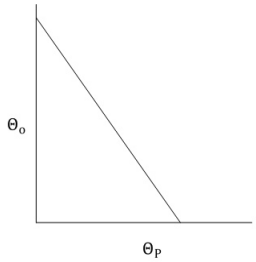

Рис. 2.2 Двумерное вероятностное пространство

Как будет выглядеть четырехмерное пространство? Это будет тетраэдр. В n-мерном пространстве будет фигура, называемая *симплексом* - нетривиальный многогранник с минимальным числом вершин.

В трехмерном пространстве сечение двумерное, а минимальный нетривиальный многогранник в двумерном пространстве - это треугольник.

Соответственно, в четырехмерном пространстве сечение - тетраэдр.

Таким образом, распределение лежит на соответствующем пространстве. Какое можно придумать распределение, живущее на плоскости треугольника?

Для двумерного пространства подходит β-распределение:

$$Z\theta^{\alpha}(1-\theta)^{\beta}$$

Иначе можно написать его так:

$$Z\theta_o^{\alpha}\theta_p^{\beta}\delta(\sum\theta-1) \tag{7}$$

В формуле (7) в выражении в скобках говорится о том, что сумма обязана быть равной 1.

Эту картину можно обобщить. В ряде случаев в качестве априорного распределения принято и удобно использовать следующее распределение:

$$P = Z\prod\theta_i^{\alpha_i}\delta(\sum\theta_i-1) \tag{8}$$

Функция (8) называется распределением Дирихле.

Представим, что мы бросили кость N раз, и у нас появились некоторые исходы. θ_i - параметр свойства кости, который мы хотим оценить по результатам испытаний. n_i - результаты наших испытаний.

Вероятность вероятности выпадения той или иной стороны игральной кости при условии наших наблюдений равна следующему:

$$P(\theta|D) = \frac{\frac{N!}{\prod n_i!}\prod\theta_i^{n_i}Z\prod\theta_i^{\alpha^i}}{\int\frac{N!}{\prod n_i!}\prod\theta_i^{n_i}Z\prod\theta_i^{\alpha_i}d\theta} = \frac{\prod\theta_i^{n_i+\alpha_i}}{\int\sum\theta_i=1,\theta_i\geq 0\prod\theta_i^{n_i+\alpha_i}d\theta} \tag{9}$$

Функция (9) содержит в первой части числителя *мультиномиальным* распределение. Это обобщение биномиального распределения, правдоподобие.

Сделаем MAP-оценку, посчитав максимум от функции (9). Поскольку интеграл в знаменателе - константа, то нам важен максимум только от функции в числителе.

Для подсчета максимума нам нужно посчитать все частные производные и приравнять их к 0. Однако, у нас есть ограничение, что $\sum\theta_i-1=0$. Это неопределенные множители Лагранжа.

$$F(\theta) = \frac{\prod\theta^{n_i+\alpha_i}}{P(D)}$$

Функция Лагранжа выглядит следующим образом:

$$\phi(\theta) = F(\theta) - \lambda(\sum \theta_k - 1)$$

λ – неопределенный множитель

Как посчитать частные производные?

$$f(x) = x^n;\ \ f' = nx^{n-1} = \frac{f(x)}{x}$$

$$\frac{\partial \phi}{\partial \theta_i} = \frac{(n_i + \alpha_i)\dfrac{F(\theta)}{\theta_i}}{P(D)} - \lambda = 0$$

$$\frac{\partial \prod\limits_k \theta_k^{n_k+\alpha_k}}{\partial \theta_i} = \prod_{k\neq i} \theta_k^{n_k+\alpha_k} \cdot (n_i+\alpha_i)\theta_i^{n_i+\alpha_i-1} = \prod_{k\neq i} \theta_k^{n_k+\alpha_k} \cdot \theta_i^{n_i+\alpha_i} \frac{1}{\theta_i}(n_i+\alpha_i) = \prod_k \theta_k^{n_k+\alpha_k} \frac{n_i+\alpha_i}{\theta_i}$$

Производные по всем $k \neq i$ равны 0. Если $k = i$, то они равны просто $1 \cdot \lambda$.

Таким образом,

$$\frac{(n_i + \alpha_i)}{\theta_i} \frac{F(\theta)}{P(D)} = \lambda$$

$$\theta_i = \frac{n_i + \alpha_i}{\lambda} \frac{F(\theta)}{P(D)}$$

Откуда нам посчитать λ?

У нас есть условие $\sum \theta_i - 1 = 0$ и свободная переменная - одно уравнение и одна неизвестная. Делаем подстановку.

$$\sum \theta_i = \sum (n_k + \alpha_k)\lambda \frac{F(\theta)}{P(D)} = 1$$

$$\lambda = \sum (n_k + \alpha_k) \frac{F(\theta)}{P(D)}$$

Таким образом,

$$\theta_i = \frac{n_i + \alpha_i}{\lambda} \frac{F(\theta)}{P(D)} = \frac{n_i + \alpha_i}{\sum n_k + \alpha_k}$$

Распределение Дирихле:

$$P(\theta) = Z \prod \theta_i^{\alpha_i}$$

В распределении Дирихле α_i - параметр распределения Дирихле.

Если у нас есть такое априорное распределение и такой набор наблюдений, то байесова оценка $\theta_i = \dfrac{n_i + \alpha_i}{\sum(n_k + \alpha_k)}$

Как бы мы делали тривиальную частотную оценку?

$$\theta_i = \frac{n_i}{\sum n_k}$$

Чем отличается байесов анализ от частотного? Тем, что мы добавили α_i, которые живут так же, как и n_i. То есть мы сделали вид, что у нас было сделано еще некоторое количество исыптаний, которых мы на самом деле не делали, но которые должны нам дать определенные исходы. Поэтому параметры α_i, они же параметры распределения Дирихле, называются *псевдокаунтами (псевдоотсчетами)*.

Таким образом, у нас есть результаты опыта и наша фантазия, которую мы добавляем к опытам.

На самом деле, это является поправкой на малое число испытаний, поскольку байесово оценивание стремится к частотному, если испытаний очень много. Однако, в жизни у нас обычно бывает очень мало испытаний.

E-оценка устроена следующим образом:

$$\theta_i = \frac{n_i + \alpha_i + 1}{\sum(n_k + \alpha_k + 1)}$$

MAP-оценка и E-оценка, в принципе, одинаково устроены.

Байесово оценивание очень широко применяется, в том числе, в биоинформатике и многих других прикладных вещах. Сейчас даже существует наука, основанная на байесовых нейронных сетях.

Байесова оценка параметров распределения

Представим себе, что у нас есть возможность делать одноклеточное секвенирование. Когда мы секвенируем одну клетку и смотрим уровень экспрессии генов, то у нас количество ридов очень небольшое.

Если мы знаем, что уровень экспрессии гена, например, λ, то как (по какому закону) распределено число ридов, которое мы получим? Оно распределено по Пуассону:

$$P(n) = \frac{\lambda^n}{n!} e^{-\lambda}$$

Мы смотрим на один и тот же ген в разных клетках. В первом опыте мы считаем, что клетки примерно в одинаковом состоянии. У нас есть некое наблюдение. В первой клетке у нас было 2 рида, попавших на этот определенный ген, во второй клетке - 0, в третьей - 5 ($n_1 = 2, n_2 = 0, n_3 = 5$).

Мы хотим узнать, имея всего 3 клетки, какой в клетке на самом деле уровень

экспрессии генов? В каждой из 3х клеток секвенирование показало разный уровень экспрессии некого фиксированного гена. Вопрос в том, чему равно λ?

Мы определяем параметр распределения Пуассона. Мы из некоторых общих соображений, глядя, как устроены все возможные гены, рисуем априорное распределение λ. Есть малоэкспрессируемые гены, есть сильноэкспрессируемые гены и др. Данное распределение будет похоже на γ-распределение, которое устроено следующим образом:

$$P(\theta = \lambda) = f(\lambda) = Z\lambda^k e^{\frac{-\lambda}{r}} \tag{10}$$

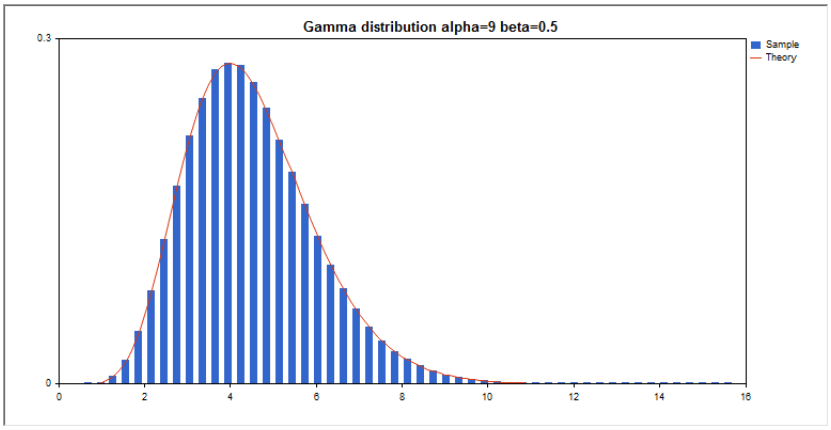

Рис. 2.3 График функции плотности нормального распределения вероятностей

Данное распределение является двухпараметрическим (параметры k и r). k отвечает за поведение вначале, r - в конце.

γ-распределение очень широко и часто применяется в биоинформатике.

Это распределение заведомо никогда не бывает отрицательным. Носитель распределения - отрезок $[0, +\infty)$.

Глядя на массу других генов и экспериментов, мы можем прийти к выводу, что уровень экспрессии генов распределен примерно по γ-распределению. Это есть априорное распределение.

Лайклехуд (условная вероятность) нашего наблюдения:

$$P(\lambda|D) = \frac{\frac{\lambda^2}{2!}e^{-\lambda}e^{-\lambda}\frac{\lambda^5}{5!}e^{-\lambda}\frac{1}{3!}Z\lambda^k e^{-\frac{\lambda}{r}}}{P(D)}$$

$$\frac{\lambda^2}{2!}e^{-\lambda}e^{-\lambda}\frac{\lambda^5}{5!}e^{-\lambda}\frac{1}{3!} = P(D|\theta)$$

$\frac{1}{3!}$ – число опытов

Мы получаем следующее:

$$P(\lambda|D) = \frac{e^{-\lambda(3+r)}\lambda^{7+k}}{P(D)2!3!5!\frac{1}{Z}}$$

$P(D)2!3!5!\frac{1}{Z} = Z'$ – нормировочный множитель
Таким образом,

$$P(\lambda|D) = Z'e^{-\lambda(3+r)}\lambda^{7+k}$$

Нормировочный множитель обеспечивает нам равенство единице интеграла по всем возможным λ из основной формулы.

Иначе это можно записать так:

$$P(\lambda|D) = \frac{\lambda^{7+k}e^{-\lambda(3+r)}}{\int_0^\infty \lambda^{7+k}e^{-\lambda(3+r)}d\lambda}$$

Нам нужно, чтобы этот интеграл был равен 1, поэтому мы на него и делим. Новый нормировочный множитель вобрал в себя и все факториалы, и ту Z, которая была нормировочным множителем в изначальном распределении. Интеграл от знаменателя есть интеграл от числителя.

У нас есть априорное представление, что k у всех генов равно, например, 2, а также, что $r \simeq 1/3$. Эти априорные параметры распределения мы берем из анализа всех генов, которые у нас есть в разных других экспериментах.

Таким образом, априорное распределение отражает наше представление о том, как устроен мир генов.

Мы видим, что байесова наука может применяться в разных местах, и не только для оценки вероятностей, но и для оценки параметров распределения, ведь λ - это не вероятность, а интенсивность пуассоновского процесса, которую мы оцениваем по Байесу.

Нормировка r идет из размеров библиотеки. Поэтому мы можем при правильно нормированных библиотеках получить формулу (10).

Лекция 3

Тренировочная задача

Есть одноклеточное секвенирование и данные для определенного гена: 0, 6 и 15 ридов.

Априорное распределение уровня экспрессии генов - γ:

$$f = zx^{k-1}e^{-\frac{x}{\theta}}$$

Уровень экспрессии генов - это параметр интенсивности λ распределения Пуассона (число ридов распределено по Пуассону).

$K = 2$, нужно проварьировать (посмотреть, что будет, если будет иметь другие значения - провести мини-исследование)

$\theta = 3$, проварьировать

Необходимо получить (построить) постериорное распределение, посмотреть, как параметры априорного распределения будут влиять на апостериорное распределение. Также необходимо сделать MAP- и E-оценки.

Вспомним формулу апостериорного распределения:

$$P(\lambda|D) = \frac{P(D|\lambda)P(\lambda)}{\int_\lambda P(D|\lambda)P(\lambda)d\lambda}$$

Это - γ-распределение.

Необходимо численно посчитать интеграл в апостериорном распределении. Также можно посмотреть, чему равен этот интеграл от нормировочного множителя, а также посмотреть, как делать MAP- и E-оценки без подсчетов. Также нужно посмотреть, где находится максимум функции (при каком значении λ он реализуется). Также можно численно посчитать математическое ожидание.

Данное задание можно делать в Exel, Area или на языке Python.

Рекомендуемая литература - Дурбин Эдди Крог Митчесон - "Анализ биологических последовательностей. Вероятностные модели белков и нуклеиновых кислот"

Вероятностные модели

Есть окружность со вписанным в нее правильным треугольником. Какова вероятность того, что случайная хорда окажется короче длины стороны данного треугольника?

Первый вариант ответа:

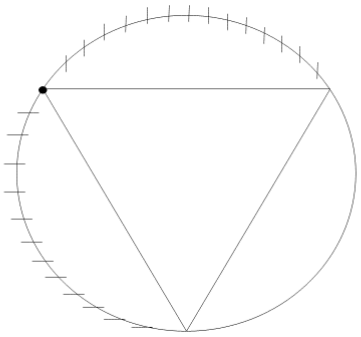

Рис. 3.1 Первый вариант решения

Если мы возьмем первую точку хорды на одной из вершин треугольника, то если вторая попадет в заштрихованную область, то хорда будет короче, чем каждая из сторон треугольника. Соответственно, вероятность $P_1 = \frac{2}{3}$.

Также возможен второй вариант ответа:

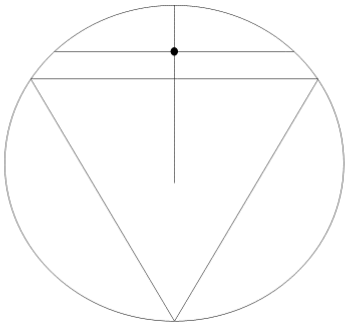

Рис. 3.2 Второй вариант решения

Проводим радиус окружности. Любая точка на радиусе этой окружности определяет хорду (сколько точек на этом радиусе, столько и хорд). Через эту точку мы можем провести перпендикуляр, и, таким образом, определить хорду. Впишем правильный треугольник. Мы видим, что в половине случаев сторона будет короче, чем хорда, а в половине - длиннее. То есть вероятность $P_2 = \frac{1}{2}$.

Третий вариант ответа:

Любая точка, кроме центра, внутри окружности, определяет хорду. Все точки, лежащие во внутренней половине круга, дают длинную хорду. А все точки, лежащие во внешней половине круга, дают короткую хорду.

Площадь всего круга $S_0 = \pi R^2$. Площадь внутреннего круга $S_1 = \pi(\frac{r}{2})^2 = \frac{1}{4}\pi r^2$. Площадь внешнего круга $S_2 = \frac{3}{4}\pi r^2$. Соответственно, третий ответ: $P_3 = \frac{3}{4}$.

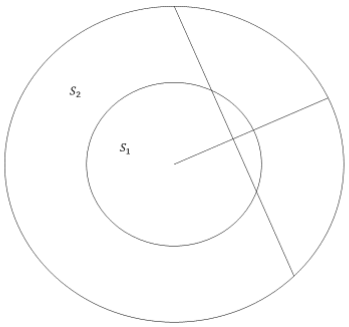

Рис. 3.3 Третий вариант решения

Чему в итоге равна вероятность? Все три пути решения кажутся правильными. Суть в том, что во всех трех случаях использовались разные модели случайной хорды. Таким образом, случайность возможна только в смысле некоторой вероятностной модели. Один и тот же объект мы можем включать в разные вероятностные модели и получать разные результаты.

Случайные модели последовательностей

Бернуллевская модель порождения последовательностей:

Каждую букву, например, последовательности нуклеотидов, определяем с помощью датчика случайных чисел.

В генбанке есть огромное число последовательностей (миллиарды). Общее число последовательностей в генбанке - N. Возьмем случайное число, равномерно распределенное от 0 до N. Мы получаем номер, по которому из генбанка мы можем получить определенную последовательность. Эта также случайная модель порождения последовательностей.

Последовательность перестает быть случайной после того, как она определилась.

Другая случайная модель порождения последовательностей - вытащить модель из генбанка и промутировать ее по определенному закону.

Что такое случайная последовательность? Это последовательность, которой еще нет, но для каждой ее реализации есть определенная вероятность.

Еще один способ порождения случайных последовательностей - в ближайшей луже черпаем воду, берем два произвольно взятых праймера, делаем ПЦР и проводим секвенирование. Таким образом, получаем случайную последовательность.

Когда мы говорим о случайной последовательности или величине, у нас должен быть способ порождения этой последовательности.

В случае модели с генбанком в нем будет представлено больше всего человеческих последовательностей. Будет перекос в определенные стороны.

В луже же мы, вероятнее всего, увидим последовательность бактериофага или бактерии.

Таким образом, в разных моделях вероятность получения разных последовательностей разная.

Еще один способ: мы зачерпнули воду из лужи, сделали обратную транскрипцию и прочитали получившуюся РНК. Там будут перепредставлены рРНК.

Разные способы порождения случайных последовательностей дают разные перекосы.

Отметим, что во всех перечисленных способах, кроме первого, тяжело оценить вероятность. Для этого потребовалось бы слишком большое количество экспериментов. При этом случайных выбор из банка довольно часто применяется.

Бернуллиевская модель порождения последовательностей

Итак, у нас есть первый *математический* способ порождения последовательности: есть алфавит, в котором каждой букве соответствует определенная вероятность:

$A \to P_\alpha$

$\sum P_\alpha = 1$

P_α - вероятность буквы α.

Способ порождения последовательности заключается в том, что мы генерируем случайную величину. Поскольку сумма равна 1, мы разбиваем отрезок [0,1] на P_a, P_b, P_c, P_d, P_e. Сумма этих отрезков дает 1.

Далее возьмем случайную величину, равномерно распределенную от 0 до 1. В зависимости от того, куда попадет эта случайная величина, мы будем генерировать определенную букву. Это *бернуллиевский* способ порождения последовательности.

Если последовательность уже родилась, то мы можем посчитать правдоподобие ее появления. Напомним, что *правдоподобие* - это вероятность того, что случилось при условии, что мы не знаем о том, что это случилось.

Любая вероятность живет на вероятностном пространстве. А модель - это способ определения вероятностного пространства. Таким образом, все происходит в рамках модели. Любая случайная последовательность должна быть определена. Для одной и той же последовательности правдоподобие будет разное.

Мы порождаем последовательность, для которой определяем правдоподобие. Допустим, для последовательности "аааааааааа"правдоподобие, учитывая, что все буквы в ней равновероятны, равно $0{,}25^L$.

Теперь возьмем все возможные последовательности и просуммируем их вероятности. Мы должны получить 1. Однако, мы получим бесконечность. Число последовательностей длины L равно $N(L) = 4^L$. Вероятность последовательности длины L равна $0{,}25^L$. Однако, $4^L \cdot 0{,}25^L = 1$. Если мы возьмем последовательность с большей длиной, то мы получим еще 1. Сумма вероятностей всех последовательностей длины L равна 1. Такой же величине равна сумма вероятностей последовательности любой другой длины.

Вообще говоря, описанное выше не является вероятностной моделью порождения последовательности, поскольку сумма вероятностей (не только длины L, а всех длин) бесконечна.

Бернуллиевская последовательность, кроме вектора P_α содержит в себе функцию распределения длин $f(L)$, то есть вероятность длины.

$$\sum f(L) = 1$$

Сумма равна 1 по определению, поскольку это функция распределения.

$$\sum f(L) \cdot 4^L \cdot \frac{1}{4}^L = 1$$

Определение бернуллиевской последовательности обязательно содержит в себе длину. Она может быть фиксированной. В этом случае функция распределения устроена очень просто. Например, если она равна 20, то она везде равна 0, а не 20 равна 1. Таким образом, если длина задана, то все работает. Длина должна быть задана функцией распределения длин.

Для того, чтобы задать бернуллиевскую последовательность, у нас должна быть функция распределения от длин $f(L)$ и набор вероятностей.

Как происходит генерация последовательности?

Сначала мы *семплируем* - из распределения пропорционально вероятностям вытаскиваем некое значение длины. Мы фиксируем длину, а потом методом Монте-Карло генерируем ровно L символов и получаем бернуллиевскую последовательность.

Задача генерации последовательности - это *прямая* задача. С практической точки зрения она является очень полезной.

Рассмотрим *обратную* задачу:

У нас есть набор последовательностей X_i, и мы верим, что все последовательности в нем бернуллиевские (гипотеза). Нам нужно найти параметры этого набора.

Каждая последовательность набора имеет свою длину. Мы подбираем параметры функции распределения длин, смотря на то распределение длин, которое нам предоставлено. Столь же просто мы находим $P_\alpha = \frac{n_\alpha}{\sum L_i}$

Если нам представлено мало последовательностей, то $P_\alpha = \dfrac{n_\alpha + \psi_\alpha}{\sum L_i + \sum \psi_\alpha}$

Мы применяем Байесов подход, говорим, что у нас есть априорное распределение Дирехле на вероятностях. Далее делаем MAP-оценку и получаем формулу выше.

ψ_α - маленькая константа порядка единиц, а n_α - константа порядка тысяч. Таким образом, количество букв не играет роли.

Мы получаем стандартную оценку.

Таким образом, у нас есть 2 задачи:

1 - сгенерировать последовательность

2 - по набору оценить параметр

Это две разные встречные задачи. Если нам известно много последовательностей, то наша задача - определить параметры. Если у нас известны параметры, то мы можем породить любое количество последовательностей. Далее на них мы можем делать некие численные эксперименты про то, как что устроено.

Бернуллиевская модель - наиболее простая и часто используемая, но она является наименее правильной.

Порождение марковской последовательности

Великий русский математик А.А. Марков заметил, что после буквы Е не встречается мягкий знак. Поэтому была введена другая модель порождения текстов, когда вероятность порождения буквы зависит от предыдущей буквы $P(S_i|S_{i-1})$.

В информатике тоже встречается такая задача: CpG - редкая встречаемость буквы G после буквы C в геноме человека. Поэтому в порождении последовательностей геномов многих эукариот CpG избегается, поскольку это мутагенный сайт метилирования цитозина, который спонтанно может перейти в урацил, а потом стать тимином. Есть метилтрансферазы, которые зачем-то метилируют нуклеотиды, что может быть опасно, хоть и не всегда.

Таким образом, при порождении последовательности мы должны смотреть на предыдущую букву.

$$\sum_\alpha P(\alpha|\beta) = 1$$

После буквы β может пойти любая другая буква. Но сумма вероятностей этих букв равна 1. Поэтому в четырехбуквенном алфавите - 16 пар и 4 связи.

Итак, у нас есть следующий набор параметров: буква α при условии, что предыдущая буква - β. В нуклеотидном контексте таких вероятностей 16. Сколько из них независимых? Вообще говоря, у нас сумма по всем α при фиксированном β должна быть равна 1. Из них 4 зависимых (на 16 чисел - 4 связи). Итого, 12 параметров - независимые.

Также нам необходимо учитывать вероятность первой буквы, поскольку она не имеет предыдущей буквы. Из 4 параметров 3 независимые (1 связь).

15 независимых параметров определяют марковскую модель I порядка (12 + 3).

Если нам дано много последовательностей, то как мы можем определить параметры? Мы должны просто посмотреть на появление первой буквы.

$$P(\alpha|\beta) = \frac{P(\alpha,\beta)}{P(\beta)}$$

$P(\alpha\beta)$ - частоты встречаемости пар букв друг за другом (β - предыдущая, α - следующая)

Первую букву мы будем генерировать, исходя их распределения $P(\alpha)$, а каждую следующую - исходя из распределения $P(\alpha|\beta)$.

Все мы знаем о существовании кодонов, в которых каждая буква генетического кода зависит не от одной, а сразу от двух предыдущих нуклеотидных букв. Это также является марковской моделью II порядка.

Мы можем это обобщить:

$$P(S,|S_{i-1},S_{i-2})$$

Когда следующая буква зависит от двух предыдущих букв, у нас должно быть $P(\alpha)$ - для первой буквы, $P(\alpha|\beta)$ - для второй буквы, $P(\alpha,|\beta,\gamma)$ - для всех остальных букв.

В $P(\alpha,|\beta,\gamma)$ - 64 параметра, 16 связей (64-16)

$P(\alpha|\beta)$ - 16 параметров и 4 связи (16-4)

$P(\alpha)$ - 4 параметра и 1 связь (4-1)

Просуммировав, мы получим 63 независимых параметра.

Заметим, что бернуллиевская модель - это марковская модель 0 порядка.

$$N = |A|^{\text{пор}+1} - 1$$

N – число независимых параметров

$|A|$ – размер алфавита

В любой модели должно обязательно присутствовать распределение длин $f(L)$.

Байесов информационный критерий (BIC)

Допустим, нам предъявлен набор последовательностей x_i. Для этого набора последовательностей мы можем построить модель любого порядка. Какая их них будет лучше описывать набор предъявленных нам последовательностей (например, промоторных областей или интронов)?

Разбиваем выборку $W = x_i$ на две: x_k - обучающая (training) и x_n - тестирующая (test). $Tr \cup Tst = W$.

При увеличении порядка модели (строго говоря, ее сложности), мы столкнемся с проблемой переобучения.

Для оценки качества обучения модели применяют так называемых байесов информационный критерий(BIC) или информационный критерий Акаике (AIC).

BIC выглядит так:

На нашей выборке мы обучаем параметры. Далее мы считаем правдоподобие этого набора x_i относительно модели:

$$L = \prod_i (f(|x_i|) \cdot \prod (k) P(x_i^k)) \tag{11}$$

Формула (11) - произведение по всем последовательностям.

Нам нужно понять, насколько правдоподобно наше наблюдение при условии тех параметров, которые мы обучили.

В обучении нам был дан набор последовательностей. Мы определяем параметры, глядя на этот набор. Затем мы смотрим, насколько этот набор соответствует тем параметрам, которые мы из него получили.

Если у нас есть две модели, то мы можем посчитать лайклехуд относительно каждой из них (правдоподобие, произведение вероятностей, очень маленькая величина): $L(x_i|M_1)$ и $L(x_i|M_2)$. Отношение $\frac{L(x_i|M_1)}{L(x_i|M_2)}$ говорит нам о том, какая модель лучше описывает наши предъявленные последовательности.

У нас есть борьба между двумя вещами: с одной стороны, размером выборки, на которой мы обучаемся, а с другой стороны - сложностью модели.

Если выборка очень большая, то мы можем пытаться применить достаточно сложные модели. Если выборка маленькая, то мы должны применять простые модели.

Посчитаем логарифм правдоподобия:

$$L = \log(\prod_i (f(|x_i|) \cdot \prod_k P(x_i^k)))$$

$$BIC = -L(x_i) + a ln K$$

a – число параметров или степеней свободы

K – размер выборки – то, по какому числу наблюдений мы вычисляем наши параметры

Мы можем для разных моделей посчитать BIC. Та модель, для которой BIC самый маленький, - самая сбалансированная относительно числа наблюдений. Если нам задан размер выборки, то параметров должно быть поменьше, а правдоподобие

побольше (в благоприятном случае).

Таким образом, BIC позволяет нам сравнивать модели друг с другом.

BIC и AIK (устроена похожим образом) очень широко применяются в машинном обучении. Однако, напрямую для нейросетей они не применяются из-за большого количества параметров.

Таким образом, размер выборки определяет, какая модель нам лучше подходит.

Лекция 4

По предыдущей лекции можно сказать, что бывают разные вероятностные модели порождения последовательностей. Также можно сказать, что модели порождения последовательностей зависят от параметров, которые можно вычислить (обучить). Третий вывод заключается в том, что модели можно сравнивать между собой. Например, для сравнения можно использовать Байесов информационный критерий.

Поиск гомологии (сходства)

Есть биологическая задача, которая называется поиск гомологии. Два биологических объекта называются *гомологичными*, если они имеют общее происхождение.

В мире последовательностей даны последовательности и нужно понять, гомологичны они или нет, то есть ли у них общий предок.

Вообще говоря, утверждать, что последовательности гомологичны, как правило, мы не можем. Мы можем только предположить это.

Несмотря на это, в некоторых случаях мы даже видим общего предка. Это касается, например, микроэволюции. Есть изоляты различных вирусов, в том числе, испанки, ретровирусы гепатита С и ВИЧ. Как правило, они взяты их одного пациента и имеют общего предка. Иногда этот предок даже виден. Тем не менее, такое бывает очень редко.

Как правило, мы не можем говорить о точной гомологии. Поэтому вместо поиска гомологии обычно занимаются поиском сходства.

Не стоит говорить о проценте гомологии, вместо этого лучше сказать о проценте сходства, который и понимают под словосочетанием "процент гомологии". Если последовательности гомологичны на 50%, то это не означает, что они с вероятностью 50% имеют общего предка. Это значит, что у них одинаковы 50% аминокислотных остатков.

Таким образом, гомология - это абстракция, а поиск сходства - это конкретная задача. Мы предполагаем, что поиск гомологии можно подменить поиском сходства, что не всегда является правдой, поскольку сходство может уже потеряться, а гомология - сохраниться.

Отсюда возникает биологическая задача выравнивания, которая заключается в написании одной последовательности под другую так, чтобы, например, гомологичные аминокислотные остатки стояли друг под другом. Заметим, что поиск сходства - это не всегда выравнивание. Можно искать сходства, не выстраивая выравнивания.

Однако, тут мы возвращаемся к вопросу о гомологии, про которую мы ничего не можем утверждать.

После сопоставления пространственных структур (структурное выравнивание) возникает гипотеза о том, что если эти структуры похожи, то, наверное, они являются гомологичными, что не всегда верно. Иногда в результате конвергенции совершенно негомологиченые белки приходят к общей структуре, по крайней мере, по архитектуре.

Итак, более точно гипотеза заключается в том, что при сопоставлении пространственных структур белков гомологичные остатки занимают одинаковые позиции.

Если у нас уже есть структурное выравнивание, то зачем строить выравнивание последовательностей?

Это нужно, поскольку структур на несколько порядков меньше, чем последовательностей. Последовательностей же, в свою очередь, очень много. Получить последовательность не стоит ничего - это быстро и дешево. Именно поэтому сравнивать последовательности удобно и полезно.

Золотой стандарт

Задача заключается в том, чтобы придумать способ построения выравнивания между последовательностями, который будет воспроизводить структурное выравнивание. Далее мы верим, что если у нас есть хорошее соответствие выравнивания последовательностей структурному выравниванию, то и в других случаях оно будет правильным. Отсюда возникает концепция *золотого стандарта*. Это те выравнивания, которые были получены из структурных выравниваний.

Далее мы придумываем какой-либо алгоритм, который строит выравнивания, и проверяем, как он работает на золотом стандарте. Если он работает нормально, то мы предполагаем, что во всех остальных случаях этот алгоритм будет работать так же хорошо.

Это можно считать почти верным, но не совсем. Структурные выравнивания делаются для белков, структура которых предсказана. Но есть значительная часть белков, которые неструктурированы. По самым экстремальным оценкам чуть ли не половина белков человека неструктурированы (unfolded proteins). Если белки не структурированы, то они не кристаллизуются или кристаллизуются плохо.

Таким образом, структурные выравнивания строятся хорошо для тех белков, которые хорошо кристаллизуются. Мы знаем, что хорошо кристаллизуются глобуляр-

ные белки. С трансмембранными же белками возникают проблемы. Поэтому золотой стандарт сильно перекошен в сторону глобулярных белков. Таким образом, использование структурных выравниваний в качестве золотого стандарта для выравнивания трансмембранных белков, скорее всего, неправомерно.

У нас есть основания предполагать, что эволюция трансмембранных белков отличалась от эволюции глобулярных белков. В трансмембранных белках по-другому устроены замены. Например, алифатические аминокислотные остатки заменяются на ароматические крайне редко, в глобулярных же белках - гораздо чаще. Однако, это утверждение о характере замен следует из выравнивания. Поэтому здесь возникает замкнутый круг.

Тем не менее, общая логика следующая:

Мы должны построить такой способ построения выравнивания, который хорошо воспроизводит золотой стандарт.

Сравнение последовательностей без выравнивания

Берем последовательность S_1 и строим архив $S_1 \longrightarrow zipS_1$.

Берем последовательность S_2 и строим архив $S_2 \longrightarrow zipS_2$.

Построение архива предполагает схлопывание повторов.

Если длина архива от конкатенации двух последовательностей $|zip(S_1 + S_2)| \sim |zipS_1| + |zipS_2|$, то последовательности S_1 и S_2 не похожи.

Если $|zip(S_1 + S_2)| \ll |zipS_1| + |zipS_2|$, то последовательности похожи.

Таким образом, мы сравниваем последовательности, а выравнивание не строим, что подтверждает идею о том, что сравнение не обязательно предполагает построение выравнивания.

Выравнивание

Если у нас есть последовательности S_1 и S_2, то сколькими способами можно их написать друг под другом, вставив разные делеции?

Пример:

$$aaa--aaa$$

$$b-bbbb-b$$

Выше мы видим выравнивание последовательностей, которое не является оптимальным. Сколькими способами можно расставить делеции между этими последовательностями?

0 - буква пришла из первой последовательности, 1 - буква пришла из второй последовательности. Смпотрим сверху-вниз по колонкам. Если мы видим пропуск, не пишем ни 0, ни 1.

Получаем следующее кодирование:

$$010011101001$$

Глядя на получившуюся последовательность 0 и 1, мы можем очень просто вернуться к нашим последовательностям.

Сколько последовательностей 0 и 1, столько и выравниваний. Заметим, что есть вырожденные случаи.

Что мы знаем про эту последовательность 0 и 1? Мы знаем, что число 0 - это число букв в первой последовательности ($n_0 = n$), а число 1 - это число букв во второй последовательности ($n_1 = m$).

При таких ограничениях сколькими способами мы можем составить эту последовательность 0 и 1, состоящую из $n + m$ букв?

$$C_{m+n}^{n} = \frac{(n+m)!}{n!m!}$$

Пусть эти последовательности имеют одинаковую длину, то есть $n = m$.

Тогда:

$$C_{m+n}^{n} = \frac{(2n)!}{(n!)^2}$$

По формуле Кирлинга:

$$n! \approx \sqrt{2\pi n} n^n e^{-n}$$

Делаем подстановку:

$$(2n)! = \sqrt{4\pi n}(2n)^{2n} e^{-2n} = \sqrt{4\pi n} 2^{2n} n^{2n} e^{-2n}$$

$$(n!)^2 = 2\pi n n^{2n} e^{-2n}$$

Итого получаем:

$$C_{m+n}^{n} = \frac{1}{\sqrt{2\pi n}} 2^{2n}$$

Таким образом, этих выравниваний очень много. Поэтому простой перебор будет малоэффективен. Мы можем использовать разные способы получения выравниваний.

Алгоритм Нилдмана-Вунша

Каким-нибудь алгоритмом (например, генетическим) расставляем эти 0 и 1 и получаем выравнивания. Для того, чтобы оценить качество выравнивания, нам надо посчитать score без выравнивания.

У нас есть каким-то образом построенное выравнивание. И для любого такого выравнивания мы должны посчитать, насколько оно хорошо (W - вес). Чем больше вес, тем лучше выравнивание.

Чтобы оптимизировать вес выравнивания, можно, например, применить генетический алгоритм оптимизации.

Есть замечательный алгоритм динамического программирования Нидлмана-Вунша, который устроен следующим образом:

Строится матрица, каждой ячейке которой записан вес оптимального выравнивания для префиксов последовательностей S_1 и S_2.

Мы можем сделать делецию в каждой из последовательностей, либо пройти по диагонали.

$$W_{ij} = max \begin{cases} 2\delta(S_i^1 S_j^2) - 1 + W_{i-1,j-1} \\ W_{i-1,j} - 1 \\ W_{i,j-1} - 1 \end{cases}$$

δ в данном случае - символ Кронекера. Если S_i^1 и S_j^2 совпадают, то он равен 1, если не совпадает - 0.

Получается, что мы уходим в плюс от предыдущего веса, если буквы S_i^1 и S_j^2 совпадают и $\delta = 1$, а если буквы не совпадают, и $\delta = 0$, то мы уходим в минус от предыдущего веса.

Вес оптимального выравнивания - это вес оптимального пути по получившейся матрице (графу).

Однако, если мы напишем программу, основанную на работе алгоритма в таком виде, то мы получим ошибку, поскольку мы не сможем выставить значения в первом столбце и в первой колонке, так как мы ссылаемся на предыдущие столбцы и колонки, которых нет.

Видоизменяем алгоритм:

$$W_{ij} = max \begin{cases} 2\delta(S_i^1 S_j^2) - 1 + W_{i-1,j-1}; i > 0, j > 0 \\ W_{i-1,j} - 1, i > 0 \\ W_{i,j-1} - 1, j > 0 \end{cases}$$

В таком виде алгоритм будет работать.

Алгоритм Нидлмана-Вунша в простейшей форме заключается в поиске оптималь-

ного пути в графе из точки, находящейся за пределами ($W_{-1,-1} = 0$) до точки, находящейся в правом нижнем углу. Оптимальный путь имеет наибольший вес. При поиске такого пути мы используем динамическое программирование.

Чтобы получить само выравнивание, мы пишем следующее:

$$\pi_{ij} = arg\ max \begin{cases} 2\delta(S_i^1 S_j^2) - 1 + W_{i-1,j-1} \\ W_{i-1,j} - 1 \\ W_{i,j-1} - 1 \end{cases}$$

$\pi_{i,j} = \{k,l\}$ - это пара двух индексов, обозначающая некую предыдущую вершину, из которой мы пришли к вершине с индексами i,j.

Так устроен стандартный поиск оптимального пути в графе с помощью динамического программирования.

Матрица замены аминокислотных остатков

Если применить такой поиск к золотому стандарту, ничего не получится. Если мы заменяем изолейцин на лейцин, то такая замена должна иметь меньший штраф, чем, например, замена триптофана на глицин. То есть в приведенным выше алгоритме мы не разделяем несовпадения по весу и не смотрим на степень схожести аминокислотных остатков.

Именно поэтому необходима матрица замены аминокислотных остатков $M(\alpha, \beta)$. Эта матрица определяет, насколько похожи друг на друга аминокислоты. На самом деле, эта матрица должна отвечать частотам зафиксировавшихся в эволюции замены аминокислотных остатков.

Замена может произойти в виде мутации: мы мутировали кодон, и аминокислота стала совсем другой, и это может привести к смерти организма. Мы же должны учитывать число замен, которые зафиксировались в эволюции. Это говорит о том, что они несмертельные и слабовредные, а иногда даже полезные.

Однако, пока что мы останавливаемся лишь на простейшей концепции.

Общая рекурсия Нилдмана-Вунша

У нас есть параметры match (совпадение, премия), mism (несовпадение, штраф) и del (делеция). Можно задать любое значение этих параметров.

У нас значение параметров следующее:

$match = 1$

$mism = 1$

$del = 1$

Как поменяется выравнивание, если эти умножить эти параметры на 3?

Вес выравнивания, безусловно, увеличится в 3 раза. Однако, сама структура выравнивания не поменяется. То есть эти три числа можно умножать на любое положительное число и получать одно и то же выравнивание.

Пусть match = 1 всегда, поскольку совпадения для нас очень важны.

Общая рекурсия Нидлмана-Вунша выглядит так:

$$W_{ij} = max \begin{cases} W_{i-1,j-1} + M(S_i^1 S_j^2) \\ W_{i-1,j} - del \\ W_{i,j-1} - del \end{cases}$$

Выравнивания, которые дает рекурсия Нилдмана-Вунша, получаются плохими. Одна из проблем данного алгоритма заключается в концах последовательностей.

Аминокислотные последовательности появляются в результате трансляции нуклеотидных последовательностей. Старт-кодон обычно - метионин (ATG). Метионин может быть как стартовым остатком, так и внутренним и концевым. Те аминокислотные последовательности, с которыми мы, в основном, работаем, нарисованы "кривым пальцем".

На самом деле, концы белков эволюционируют довольно интенсивно и быстро. Не понятно, где правильно заканчивать и начинать последовательности.

Из каких данных мы можем узнать, где белок начинается на самом деле? Мы можем выяснить это с помощью *N-концевого секвенирования*. То есть мы можем отсеквенировать белок (примерно 10 акт) с N-конца. Однако, даже это не будет правильным началом белка (транслируемой области). Причина в том, что мы определяем N-конец зрелого белка. При этом любой нормальный белок имеет вначале различные сигнальные пептиды и другие последовательности, которые отрезаются. Таким образом, мы не можем узнать правильное начало гена. Мы получаем лишь правильное начало зрелого белка. Начало зрелого белка обычно не имеет первым остатком метионин, поскольку он часто отрезается, даже если сигнальный пептид остается.

Вообще говоря, мы будем иметь некую перекошенную информацию. Начало белка нельзя определить из данных, с которыми мы работаем.

Тем не менее, задача поиска оптимального выравнивания остается. Мы продолжаем "штрафовать"за несовпадения того, чего на самом деле нет. В последовательности могут включаться нетранслируемые последовательности, за которые тоже идет штраф.

В наше время все же существует один экспериментальный метод, который позволяет узнать, где на самом деле находится начало белка (первый кодон). Этот метод называется *рибосомным профайлингом*.

В этом методе мы фиксируем рибосомы, "съедаем"всю РНК, потом отмываем рибосомы и читаем то, что было покрыто рибосомой. Там, где рибосома была, там

белок. Там, где рибосомы ни разу не было, белка нет. Этот метод позволяет нам выяснить, где на самом деле начинается трансляция.

Однако, в большинстве случаев, мы не знаем ни начала, ни конца белка. Стопкодоны также довольно быстро эволюционируют и могут переехать со временем в другие места.

Задача локального выравнивания (алгоритм Смита-Ватермана)

У нас есть две оттранслированные аминокислотные последовательности. Мы хотим узнать "сущность"белка, то последовательность аминокислот в его середине. На концы белков мы не обращаем внимания. Возникает задача *локального выравнивания.*

На языке графов эта задача формулируется очень просто:

У нас есть та же матрица похожести с теми же переходами. На этой матрице есть множество различных путей. Мы хотим найти путь наибольшего веса. При этом нам не важно где он начинается и где кончается. Таким образом, мы ищем самый тяжелый фрагмент пути на этом графе.

Этот фрагмент мы можем найти, модифицировав граф. В этот граф мы добавляем ребра из начала во все точки. Вес этих ребер будет равен 0. Из всех точек этого графа мы будем переходить в конец также с нулевыми ребрами.

Самый тяжелый путь будет выглядеть следующим образом: мы пробегаем по нулевому ребру, затем - по самому тяжелому фрагменту, затем из конца этого пути "прыгаем"в конец матрицы.

Нам нужно научиться считать вес этого тяжелого (локально оптимального) пути.

Он характерен тем, что мы можем "прыгнуть"в любую точку этого пути из нуля, а также можем из любой точки выйти в конец. Среди всех путей надо найти путь, который неизвестно где начинается и заканчивается.

В любую точку на пути мы можем перейти сверху, слева и по диагонали. Также мы можем перейти в эту точку по нулевому ребру из начала. Таким образом, к рекурсии добавляем 0, который соответствует нулевым ребрам:

$$W_{ij} = max \begin{cases} W_{i-1,j-1} + M(S_i^1 S_j^2) \\ W_{i-1,j} - del \\ W_{i,j-1} - del \\ 0 \end{cases}$$

Мы получаем алгоритм *Смита-Ватермана.*

Закончить мы должны там, где будет самый большой вес:

$$end = arg\ max\ W_{i,j}$$

То есть мы получаем самый большой вес и прыгаем из него в конец. Правильный путь заканчивается так, где значение матрицы W самое большое.

Поиск наибольшей общей подпоследовательности

Теперь рассмотрим следующий предельный случай:

$match = 1$

$mism = 0$

$del = 0$

Будет ли задача глобального выравнивания отличаться от задачи локального выравнивания при данных значениях параметров? На самом деле, нет. Нам не важно, проходим ли мы по нулевым ребрам или по несовпадениям, поскольку штрафа за них нет. То есть мы не штрафуем за концевые делеции.

Это называется задачей *наибольшей общей подпоследовательности.*

У нас есть выравнивание, устроенное следующим образом:

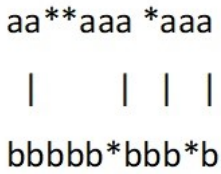

Рис. 4.1 Выравнивание для поиска наибольшей общей подпоследовательности

Совпадения показаны вертикальными линиями. Остальное не совпадает. Совпадающие буквы являются некой подпоследовательностью как из первой последовательности, так и во второй. Обозначим совпадающие буквы как $a_1 - a_4$ и $b_1 - b_4$.

Задача оптимального выравнивания заключается в том, чтобы набрать как можно больше вертикальных линий, которые обозначают совпадающие буквы, являющиеся членами общей подпоследовательности. То есть нам нужно найти такую подподследовательность, которая была бы общей (одинаковой) у двух входных последовательностей.

Зависимость веса выравнивания от длины последовательности

Допустим, мы применили алгоритм Нилдмана-Вунша и сравнили последовательности, получив некий вес. Допустим, вес получился равным 15. Это много или мало?

Нам необходимо иметь представление о том, как вес выравнивания отражает его "необычность".

На самом деле, мы будем решать другой вопрос: как вес выравнивания зависит от длины последовательности, если последовательность в некотором смысле (в смысле любой модели) случайная?

Сделаем наблюдение:

Пусть у нас есть последовательностb S^1 и S^2. Мы разделяем эти последовательности пополам и строим оптимальное выравнивание в каждой из половин. Мы посмотрим вес выравнивания, рассмотрев три варианта:

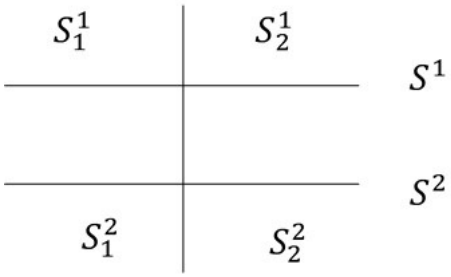

Рис. 4.2 Разделение последовательностей пополам

Нужно сравнить $W(S_1^1 S_1^2) + W(S_2^1 S_2^2)$ и $W(S^1, S^2)$. Какой вес оптимального выравнивания будет больше: тот, что получается при складывании выравниваний двух половинок или тот, что получается при оптимальном выравнивании всего целиком?

На самом деле, ответ следующий:

$$W(S_1^1 S_1^2) + W(S_2^1 S_2^2) \leq W(S^1, S^2)$$

Причина в том, что прием разделения последовательностей на две части запрещает нам выравнивания перекрестных частей (например, S_1^1 и S_2^2). Заметим, что делить последовательности мы можем в любой пропорции.

Если есть такое наблюдение, то математическое ожидание при условии, что это конкретные последовательности:

$$E(W_1 + W_2) \leq E(W)$$

Из этого свойства следует, что:

$$E(L) \geq CL$$

Таким образом, вес выравнивания не меньше, чем линейная функция от длины.

Заметим, что ничего не говорится о том, какая модель последовательностей используется, поскольку это утверждение верно для любой модели последовательности,

которую можно делить пополам. Например, если в какой-то модели все последовательности имеют длину 100, то мы не можем поделить их пополам в рамках этой модели, поскольку в этом случае поделенные последовательности будут относиться к другой модели.

С другой стороны, если последовательности одинаковые, то вес выравнивания пропорционален длине: $W \sim L$. Если последовательности одинаковые, то любое добавление увеличит вес выравнивания на 1.

Если последовательности неодинаковые, то вес выравнивания меньше, чем длина: $W < L$.

Поэтому для любых последовательностей верно следующее:

$$W(S_1 S_2) \leq L$$

Получается, что вес выравнивания заключен между двумя линейными функциями: $CL \leq W(S_1 S_2) \leq L$.

Отсюда мы можем сделать вывод, что при данных параметрах вес выравнивания линейно зависит от длины входных последовательностей все зависимости от используемой модели:

$$E(L) = C'L$$

Заметим, что константа C зависит от используемой модели.

Задача поиска наибольшего общего слова

Рассмотрим другой предельный случай локального выравнивания:

$match = 1$

$mism = \infty$

$del = \infty$

Это значит, что мы хотим найти подряд совпадающие буквы. Это называется задачей поиска наибольшего общего слова.

Здесь мы используем только локальное выравнивание, так как глобальное не имеет смысла из-за бесконечных штрафов за любое несовпадение.

Здесь важно, что используется бернуллиевская модель, хотя потом это ограничение будет снято.

Мы берем две последовательности и накладываем их друг на друга каким-то образом. Отмечаем совпадающие буквы (совпало- успех, не совпало - неуспех). Такое наложение последовательностей соответствует некой серии бернуллиевских испытаний.

Поскольку штраф за делецию равен бесконечности, никаких вставок и делеций нет.

В классической теории вероятностей есть задача о наибольшей серии успехов в бернуллиевских испытаниях.

$$E(L_{\text{серии}}) \sim \log_{1/P} n$$

$E(L_{\text{серии}})$ - длина серии успехов

P – вероятность успеха

n – число испытаний

То есть длина серии успехов логарифмически зависит от длины серии испытаний (по теории вероятностей).

Мы сдвигаем последовательности друг относительно друга и каждый раз ищем максимально длинную серию успехов. Оказывается, что:

$$E(W) \sim \log_{1/P} n \cdot m + C$$

Поэтому в нашем случае:

$$E(W) \sim logL$$

Теорема Ватермана

Таким образом, есть один крайний случай, когда вес выравнивания пропорционален длине. Также есть другой крайний случай, когда вес выравнивания пропорционален логарифму длины.

У нас есть пространство и два параметра: del и mism.

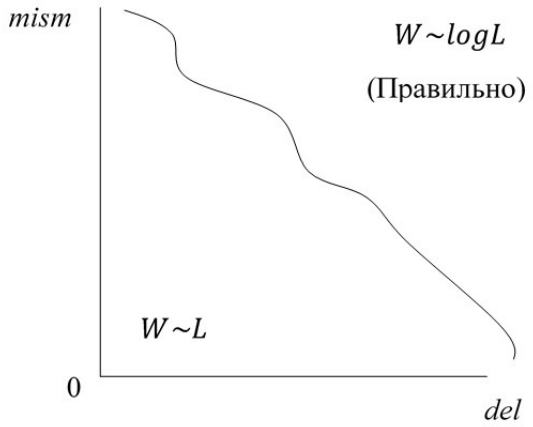

Рис. 4.3 Теорема Ватермана

Есть некое общее утверждение (теорема Ватермана, доказательство не приводится), которое заключается в том, что все пространство параметров разбивается на две области, в одной из которой мы видим логарифмическое поведение веса выравнивания, а в другой части - линейное поведение (см. рис. 4.3). Другого поведения (зависимости веса выравнивания от длины последовательности) не бывает.

Поведение на границе нам не важно, поскольку никто не знает, как она устроена.

Более того, если мы рассматриваем аминокислоты, то у нас есть матрица замен $M(\alpha, \beta) = M(\beta, \alpha)$, и тогда у нас 210 параметров. В этом случае нам понадобится очень большое пространство. И все равно это пространство параметров разбивается на две области, в которых вес выравнивания зависит от длины последовательностей либо логарифмически, либо линейно.

Практика показывает, что более-менее приличные и аккуратные выравнивания получаются, если параметры del и mism находятся в области логарифмической зависимости.

Выравнивания, находящиеся в области линейной зависимости, получаются неправильными.

Лекция 5

Наша биологическая задача состояла в том, чтобы научиться строить правильные выравнивания.

Мы приняли, что биологическое качество выравнивания и математическое качество выравнивания - это одно и то же.

Мы сказали, что математическое качество выравнивания определяется следующим образом:

$$W = \sum \mathrm{M}(x, y) - \Delta \tag{12}$$

Δ – штраф за делеции

Если нам дано какое-то выравнивание (не важно, какое), в котором есть какие-то сопоставления, то его вес определяется формулой выше.

Мы не можем сказать, почему это соответствует биологическому выравниванию, но мы это принимаем. В результате мы получаем математическую задачу, которая заключается в поиске максимального веса такой суммы.

Таким образом, мы сделали формализацию биологической задачи.

Устройство матрицы выравниваний

Как устроена матрица выравниваний?

Пусть у нас есть какое-то выравнивание двух последовательностей. У этого выравнивания есть некая модель, которую мы назовем A. Также у нас есть некая модель R, в которой последовательности не выровнены и не имеют никакого отношения друг к другу (см. рис. 5.1).

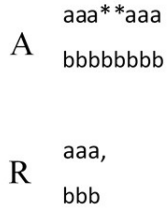

Рис. 5.1 Две модели, включающие последовательности

$$P(x,y|A) = P(x_1 \text{ o } y_1) \cdot P(x_2 \text{ o } y_2)...$$

Делеционные куски мы не считаем.

$P(x,y|A)$ - это правдоподобие, а не вероятность, поскольку x и y уже существуют. Это правдоподобие того, что у нас появилось две последовательности - x и y при условии, что они выровнены (зависимое порождение последовательностей).

Аналогично при условии, что последовательности независимы (независимое порождение последовательностей):

$$P(x,y|R) = P(x_1) \cdot P(y_1) \cdot P(x_2) \cdot P(y_2)...$$

Когда у нас есть два правдоподобия, мы можем написать *отношение правдоподобия*:

$$L = \frac{\prod P(x_i \text{ o } y_i)}{\prod P(x_i)P(y_i)} = \prod \frac{P(x_i \text{ o } y_i)}{P(x_i)P(y_i)}$$

Если взять логарифм от отношения правдоподобия, то мы увидим следующее:

$$\log L = \sum \log \frac{P(x_i \text{ o } y_i)}{P(x_i)P(y_i)} \tag{13}$$

Сраним формулы (12) и (13):

Обе эти формулы нам говорят, что если мы принимаем такую вероятностную модель, то $\log \frac{P(x_i \text{ o } y_i)}{P(x_i)P(y_i)}$ - *матрица сопоставления.*

Мы можем взять матрицу из головы. Но мы также можем сделать некую вероятностную модель, посчитать отношения правдоподобия и постулировать, что, например, матрица сопоставления аминокислотных остатков - это есть логарифм отношения того, что эти два остатка "знают"друг друга к тому, что они друг друга "не знают".

Поэтому матрица сопоставления аминокислотных остатков, как правило:

$$M(\alpha, \beta) = \log \frac{P(\alpha,\beta)}{P(\alpha)P(\beta)}$$

Посчитать то, что в знаменателе этой формулы просто:

Мы должны посмотреть частоты встречаемости аминокислотных остатков в мире, либо в том классе белков, про который нужно что-либо узнать. Например, если мы хотим узнать что-либо про трансмембранные белки, то вероятности будут устроены немного по-другому.

Вообще говоря, матрица может зависеть от того, какая у нас выборка. Также заметим, что сопоставление аминокислотных остатков может зависеть от позиции. Но поскольку нам нужно универсальная, которая не зависит от позиции, то мы говорим,

что важно, какой тип остатков мы используем. Потом мы увидим, что существуют позиционно зависимые матрицы сопоставления, которые мы будем применять. Пока что мы считаем, что матрицы сопоставления позиционно независима.

Есть несколько способов вычислить числитель. Для этого нам нужно много правильных выравниваний. В этой матрице нужно определить 210 чисел (мы предполагаем, что матрица симметричная).

Мы должны вспомнить, что вероятность в классической теории вероятностей - это частоты. Частота событий является оценкой вероятности. Чтобы сделать оценку вероятностей 210 чисел, нам нужно сделать очень много наблюдений. Желательно, чтобы выборка, по которой мы делаем оценку, была представительной (чтобы в ней не были перепредставлены какие-то значения).

Современная матрица устроена так (это было придумано в конце 80-х годов):

Мы можем взять стандартное выравнивание, которое мы имеем из пространственной структуры. Таких выравниваний не очень много. Таким способом мы вряд ли сможем получить матрицу.

Сопоставлений пространственных структур у нас немного, и если есть надежное сопоставление пространственных структур, то эти белки, как правило, довольно схожи друг с другом. Получается перекос, что не есть хорошо.

В 1989 году была написана некая программа, которая строила выравнивания. Однако, тут возникают противоречия: чтобы построить выравнивание, нам нужна матрица, а чтобы получить матрицу, нам нужно построить выравнивание.

Было построено много-много локальных выравниваний без делеций по следующему принципу: совпадение - хорошо, несовпадение - плохо. При этом выравнивания должны быть не парными, а множественными:

$$\begin{cases} AFGGIK \\ AFAGLR \\ CYGGLK \\ AFGGVK \\ AFGGIK \end{cases}$$

Мы строим выравнивания, глядя на тот банк данных, который у нас есть в данный момент.

В 1990 году с помощью некой программы была построена база данных множественных безделеционных блоков $BLOCKS$. В основе этой программы лежит построение множественного выравнивания без учета матрицы.

Всего в базе данных BLOCKS 200-300 тыс. колонок выравниваний. Толщина блока, как правило, порядка 30.

Далее мы берем одну колонку, например, последнюю, и считаем количество пар

(сверху-вниз):

KR - 1

KK - 3

RK - 2

Поскольку KR и RK - это одно и то же, то:

KK - 3

KR - 3

Соответственно, мы можем получить такие пары по всей базе данных. Мы получаем число замен:

$$\frac{n(\alpha,\beta)}{\sum\limits_{\alpha\beta} n(\alpha,\beta)} = f(\alpha\beta)$$

$n(\alpha,\beta)$ - число раз, которое встретилась определенная пара

$\sum\limits_{\alpha\beta} n(\alpha,\beta)$ - общее количество пар

Таким образом, мы получили некую матрицу. Хорошая ли это матрица и имеет ли она отношение к реальным выравниваниям?

Эта матрица не очень хорошая, поскольку мы имеем перепредставленность каких-то последовательностей. Если мы, например, возьмем современный банк данных, то там будет перепредставленность последовательностей человека. Если же мы возьмем банк данных тех времен, то будет перепредставленность глобинов, поскольку в то время ими много занимались.

Таким образом, выборка, которая ложится в основу матрицы, исходно перекошена.

Матрица BLOSUM

Очень много последовательностей одинаковые или почти одинаковые. Поэтому нам стоит ввести некий порог на процент идентичности ($\%ID$).

В первых двух последовательностях всего 3 замены, в 1 и 3 - тоже 3, во 2 и 4 - 1. Мы зададим процент замен и выкинем те последовательности, которые имеют больше замен, чем задано этим процентом. Мы требуем, чтобы в блоке были представлены последовательности, которые имеют не больше заданного числа замен (заданного $\%ID$).

Таким образом, мы выкидываем пятую последовательность, которая идентична с первой. Также мы выкидываем четвертую последовательность, так как она слишком похожа на вторую.

Каждая последовательность - это точка в графе. Между последовательностями - проценты идентичности. В этом графе мы должны выкинуть определенные реб-

ра, которые нам не подходят. Ребро есть - последовательности похожи, ребра нет - последовательности непохожи.

Пусть у нас есть задача: приведенном графе найти клику. Клика - это когда все достаточно сильно друг на друга не похожи.

Поскольку задача поиска клики не дает однозначного решения, поскольку у нас может быть несколько клик в графе. Именно поэтому такое "прореживание"графа неоднозначно. Мы делаем ее, основываясь на определенной последовательности. Вокруг этой последовательности мы собираем клику. В каждом блоке мы выбираем одну из возможных.

Такое выбрасывание похожих последовательностей избавляет нас от перекошенности базы данных.

При разном $\%ID$ мы будем получать разные матрицы. Это - матрицы $BLOSUM$.

Если матрица называется BLOSUM 90, то $\%ID = 90$. При построении такой матрицы было сделано очень слабое просеивание блоков, и в ней могут встречаться последовательности, схожие на 90%. Матрица с самым маленьким $\%ID$ - BLOSUM 30. Меньше не бывает, так как теряется возможность построить выравнивание.

Самая распространенная матрица - BLOSUM 62. У нас есть эталонное выравнивание. Мы можем построить эталонные выравнивания с разными матрицами BLOSUM разного уровня и посмотреть, при каком уровне у нас будет наилучшее восстановление.

Почему 62, а не 60? Есть одно выравнивание, которое при BLOSUM 62 имеет одно дополнительное сопоставление, и становится немного лучше.

Это один способ построения матриц сопоставления, который основан на том, что у нас есть много последовательностей, которые позволили нам построить базу данных BLOCKS.

Матрицы PAM

До изобретения матрицы BLOSUM, когда в банке данных было всего несколько сотен последовательностей, а матрицы строить было нужно, Дейхоф первый в мире атлас аминокислотных последовательностей. Появился алгоритм Нидлмана-Вунша, возникла потребность в матрице сопоставлений. При всем этом последовательности в столь маленьком банке имели довольно сильную перепреставленность.

Представим эволюционный процесс:

Есть две последовательности, между которыми мы, возможно, можем построить выравнивание, а также есть их предок.

Если последовательности сильно друг на друга не похожи, то построение выравнивания зависит от матрицы, алгоритма, способа оптимизации. Если же последовательности друг на друга очень похожи, то как бы мы ни строили, выравнивание

будет одним и тем же.

Допустим, если у нас $\%ID = 90$, то построить неправильное выравнивание практически невозможно.

Пусть и в первой, и во второй последовательности по сравнению с предковой произошло по 1% замен (количество замен на 100 аминокислот). Тогда между двумя последовательностями будет примерно 2% замен, если точнее, то 0-2%.

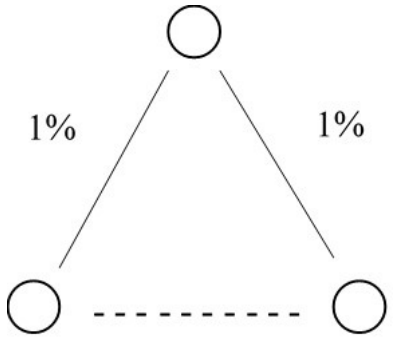

Рис. 5.2 Вычисление матрицы PAM1

Задание: сделать оценку процента замен между двумя последовательностями с помощью теории вероятностей. Заметим, что если процент замен с предком мал, то будет происходить суммирование, а если велик, то суммирования не будет.

Эволюционное расстояние, которое называется PAM, такое, чтобы был 1% замен. PAM - point accepted mutation.

Из того, что у нас есть, мы построим выравнивания и отберем те, у которых процент идентичности будет равен около 2. Мы получим некую матрицу замен:

$$P_1(\alpha|\beta)$$

Необходимо понимать, что замены, которые мы видим - это не скорость мутаций. Это частота зафиксированных мутаций.

Допустим, у нас был кодон TAT в середине белка. Произошла одиночная замена в третьей позиции, и он превратился в кодон TAG. Таким образом, в середине белка появился стоп-кодон. Данная мутация, скорее всего, не будет зафиксирована эволюцией, поскольку такой обрезанный белок вряд ли будет функционален.

Используя довольно небольшой набор данных, мы простым способом получили $P_1(\alpha|\beta)$. Как получить из этого матрицу? Используем формулу Байеса:

$$P(\alpha, \beta) = P(\alpha|\beta) \cdot P(\beta)$$

Мы получаем матрицу PAM1. Однако, выравнивание при таком малом уровне замен, скорее всего, в любом случае будет нормальном.

$$PAM1 = \log \frac{P(\alpha|\beta)P(\beta)}{P(\alpha)P(\beta)}$$

Мы получили матрицу PAM1.

Вообще говоря, мы пренебрегли случаем, когда оба предка имеют одинаковые замены, поскольку это очень маловероятно. Такие редкие замены - это очень важная вещь. Именно поэтому довольно часто при анализе эволюции рассматривают не человека против дрозофилы, а человека против шимпанзе или макаки. Это наиболее часто используемый анализ, поскольку в нем нет двойных замен (замен, попавших в один сайт, симметричных).

В нашем случае получившаяся матрица PAM1 никого не будет интересовать поскольку выравнивание при таком малом уровне замен, скорее всего, в любом случае будет нормальном. Как нам получить матрицу PAM2?

Во-первых, мы можем взять эволюционно более далекие последовательности и построить между ними выравнивание:

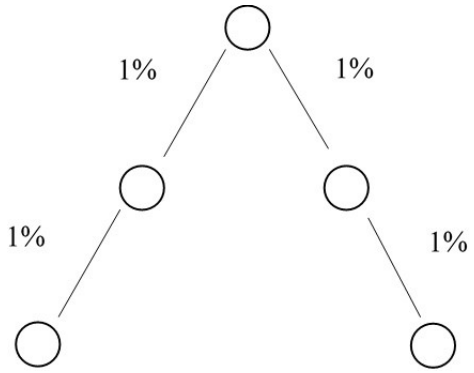

Рис. 5.3 Вычисление матрицы PAM2

Однако, такой вариант не подойдет, например, для построения матрицы PAM100 (эволюционное расстояние, при котором произошло 100 замен на 100 букв) и других высокоуровневых матриц. Там мы увидим, что замены в одном сайте происходят несколько раз.

Мы можем написать формулу для условной вероятности:

$$P_2(\alpha|\beta) = \sum_{\gamma} P_1(\alpha|\gamma)P_1(\gamma|\beta) = P_1^2(\alpha|\beta)$$

У нас что-то произошло за PAM1, и альфа поменялась на гамма, и что-то произошло за PAM2, и гамма превратилась в бета.

Мы имеем стандартный марковский процесс. У нас есть частота событий на маленьком шаге, по которой мы считаем частоту событий на более длинном шаге. Для этого мы возводим матрицу в определенную степень.

Как устроена матрица PAM3?

$$P_3(\alpha|\beta) = P_1 \cdot P_2 = \sum_\gamma P_1(\alpha|\gamma)P_2(\gamma|\beta) = P_1^3(\alpha|\beta)$$

Мы получаем следующее утверждение:

$$P_N(\alpha|\beta) = P_1^N(\alpha|\beta)$$

Таким образом мы можем посчитать эволюционные расстояния любой длины.

На PAM1 мы никогда не увидим, например, замены глицина на тирозин (G → Y). Сможем ли мы увидеть это на других матрицах? Откуда такая замена может взяться в PAM высокой степени? Там она может произойти через другие аминокислоты (G → A → V → Y). Таких путей замен аминокислот много. Мы же суммируем все эти пути. Таким образом, возведение в степень позволяет нам увидеть даже те события, которые мы не видели в PAM1.

Однако, в этом способе есть трудность: как возвести матрицу в большую степень?

Умножение матриц - это сумма произведений столбцов на строки. Делать это напрямую сложно и может привести к накоплению большого количества ошибок.

Любую матрицу можно представить в следующем виде:

$$A = UBU^{-1}$$

B - диагональная матрица

$$A^2 = UBU^{-1}UBU^{-1} = UB^2U^{-1}$$

Что такое квадрат диагональной матрицы? Это матрица, в которой все элементы возведены в квадрат. Поэтому:

$$A^N = UB^NU^{-1}$$

Матрица, например, PAM100 считается следующим образом:

$$PAM100 = \log \frac{P_1^N(\alpha|\beta)}{P(\alpha)}$$

Мы возводим условную вероятность в N-ную степень.

Пусть у нас есть эволюционное расстояние в 100 PAM и 250 PAM. Сколько в среднем будет замен? Сколько букв останутся прежними?

Во-первых, есть вероятность того, что произойдет обратная замена. Возьмем две независимые аминокислотные последовательности и наложим друг на друга. Сколько у нас будет одинаковых букв? Ответ - каждая 20-я.

Таким образом, на PAM100 будет порядка 40% одинаковых букв.

Диагональные элементы даже на PAM250, будут ненулевыми.

На достаточно больших PAM у нас возможна реверсия, при которой буква менялась несколько раз, после чего вернулась в исходную позицию.

Поскольку у нас много путей возвращения в исходную позицию, то эта вероятность достаточно значима.

Второй вариант заключается в том, что в некоторых сайтах у нас не происходит замен по случайным причинам. Когда мы строим матрицы, мы считаем, что поток замен не зависит от позиции, хотя, на самом деле, это не так: в белке есть активные центры, в которых нельзя менять аминокислоты, поскольку тогда у нас потеряется функциональная активность белка. Например, мы не можем заменить гистидин в каталитической триаде или серин в сериновой протеазе. Однако, эта наука не предполагает таких запретов и считают, что все позиции в последовательности ведут себя примерно одинаково.

С одной стороны, мы можем учесть это с помощью построения позиционно-зависимой матрицы, в которой каждая позиция будет иметь свою частоту замен. С другой стороны, поскольку у нас нет пространственной структуры, мы, не построив выравнивание, не сможем понять, какая аминокислота правильная, какая важная или неважная. Только когда мы построим выравнивания и увидим, что аминокислота не меняется, мы можем сделать вывод, что она важная.

Таким образом, логика анализа последовательностей обратна биологической логике.

Вес делеций

addcontentslinetocsectionВес делеций

Следует сказать, что все нормальные матрицы (BLOSUM, PAM) устроены так, что вес выравнивания находится в логарифмической зоне (см. Лекцию 4).

Все элементы матрицы из-за логарифма бывают как положительными, так и отрицательными. Допустим, мы прибавили ко всем элементам матрицы 8. Все элементы станут положительными, и мы попадем в линейную область.

Такие вероятностные матрицы, в которых есть в числителе логарифм вероятности, приводят нас в логарифмическую зону, что есть хорошо.

Допустим, мы взяли "умную"матрицу и начали строить локальное выравнивание. Отметим, что выбор матрицы должен зависеть от степени сходства последовательностей. Если последовательности похожи, то нужно применять более жесткую матрицу, например, BLOSUM90. Если же последовательности не похожи, то стоит взять матрицу BLOSUM30-40.

Чтобы узнать, похожи последовательности или нет, мы должны их выровнять. А с помощью какой матрицы мы будем их выравнивать? Получается порочный круг.

У нас есть стандартная матрица BLOSUM62, с помощью которой мы можем сде-

лать выравнивание. Мы получим процент идентичности, и если увидим, что он высок, то посчитаем с помощью другой матрицы. То есть мы с помощью какой-то разумной матрицы строим выравнивание, оцениваем уровень сходства, и к этому уровню сходства подбираем другую матрицу.

В итоге мы получаем "ерунду". Ее характер устроен очень просто. Выравнивание выглядит следующим образом:

aaa-aa-aaa

b-bbbbbb-b

Рис. 5.4 Неблочное выравнивание

Мы имеем выравнивание с одиночными редкими вставками.

В нашем понимании хорошее выравнивание должно выглядеть так:

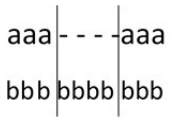

Рис. 5.5 Блочное выравнивание

Матрица должна иметь выраженный блочный характер. На рис. 5.4 мы не видим никаких блоков. Мы же, проведя все операции, описанные выше, получим то, что изображено на рис. 5.4.

Как мы можем прийти к выравниванию, изображенному на рис. 5.5?

Изначально было устроено так:

$$\Delta = d \cdot L \tag{14}$$

d - некий параметр

В этом случае выравнивание на рис. 5.5 не имеет преимущества перед выравниванием на рис. 5.3, поскольку все делеции живут независимо друг от друга. Эти делеции имеют одну цену вне зависимости от того, где они находятся.

Наверное, это неправильно. Но как сделать правильно?

Посмотрим, как устроены делеции в нормальных выравниваниях: коротких будет немного, а длинных - много. Это будет выглядеть примерно так:

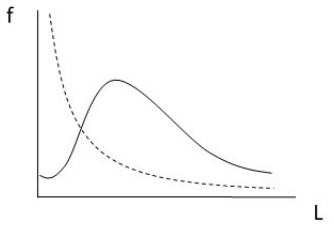

Рис. 5.6 Распределение частот делеций по длинам

Пунктирной кривой на рис. 5.5 показано распределение в соответствии с формулой (14).

Чтобы получить распределение частот, показанное сплошной кривой, нам нужно сказать, что вес делеции зависит от длины гэпа (подряд идущих делеций) примерно так (сплошная кривая):

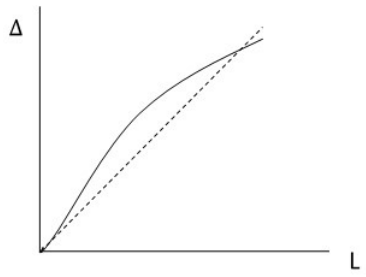

Рис. 5.7 Зависимость веса делеции от длины гэпа

Однако, эта зависимость выглядит примерно как пунктирная линия на рис. 5.7.

Сплошной график на рис. 5.7 похож на логарифмическую зависимость от длины.

Допустим, мы построили правильную зависимость штрафа от длины гэпа. Как будет выглядеть при этом алгоритм?

У нас будет граф, который будет искать оптимальный путь при условии, что Δ зависит от длины так, как показано на рис. 5.7 сплошной линией. Он будет выглядеть следующим образом:

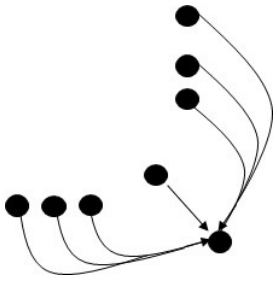

Рис. 5.8 Граф поиска оптимального выравнивания

Либо у нас совпадение, либо у нас был гэп определенной длины (1, 2, 3,...). Мы можем разными способами прийти в одну точку. Граф становится более сложным. Для этого графа можно написать следующую рекурсию:

$$W_{ij} = max \begin{cases} W_{i-1,j-1} + M(x_i y_j) \\ max_k W_{i-k,j} - \Delta(k) \\ max_k W_{i,j-k} - \Delta(k) \end{cases}$$

Этот алгоритм имеет кубическую сложность:

$$T \sim O(N^3)$$

Такая сложность нас не устраивает.

Тогда мы выведем следующую зависимость для веса гэпа:

$$\Delta = d_o + Ld_e$$

o - open

e - extention

Соответственно, график на рис. 5.6 изменится. В этом случае мы можем построить эффективный алгоритм.

Лекция 6

Выравнивание Смита-Ватермана с аффинными штрафами за делецию

Заведем в каждой ячейке три точки (вершины) вместо одной. Они будут называться следующим образом: "Мы"(M), "X "Y".

Вершина M соответствует проходу по формуле (12) (см. Лекцию 5).

Где-то у нас рождается делеция, и где-то она продолжается. Это соответствует точке Y. Y - это делеции, которые идут вдоль последовательности Y. Также они могут идти вдоль последовательности X.

У нас была делеция, которая в какой-то момент закончилась. Из точки Y мы переходим в точку M. Кончается делеция либо по Y, либо по X. Также делеция может начаться (по Y или по X).

Таким образом, у нас либо идут сопоставления, либо у нас рождается делеция. Тогда у нас будет рекурсия:

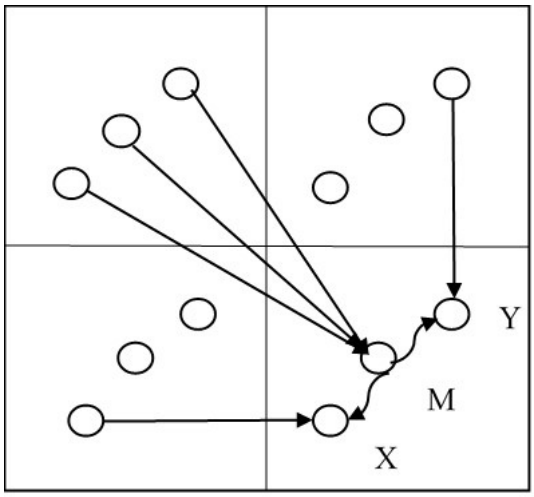

Рис. 6.1 Выравнивание с аффинными штрафами за делецию

$$M_{ij} = max \begin{cases} 0 \\ M_{i-1,j-1} + M(X_i Y_j) \\ X_{i-1,j-1} + M(X_i Y_j) \\ Y_{i-1,j-1} + M(X_i Y_j) \end{cases} \tag{15}$$

$$\pi_{ij}^{M}$$

$$Y_{ij} = max \begin{cases} Y_{i,j-1} - d_e \\ M_{ij} - d_o \end{cases} \tag{16}$$

$$\pi_{ij}^{Y}$$

$$X_{ij} = max \begin{cases} X_{i-1,j} - d_e \\ M_{ij} - d_o \end{cases} \tag{17}$$

$$\pi_{ij}^{X}$$

Мы не понимаем, началась делеция или нет и рассматриваем все случаи. Если делеция началась, что максимум будет у Y, а у M будет другое. Мы никак не можем понять, началась делеция или нет, но мы можем понять, что в Y нам дешевле прийти из M, чем из Y в другой ячейке. Это значит, что в этом месте делеция может начаться, но не значит, что она точно произойдет. Мы заполняем три матрицы: M, Y, X.

Таким образом, мы ищем оптимальный путь в усложненном графе. В этом графе втрое больше вершин. До этого у нас в каждую клетку входило 3 ребра, а сейчас 5, и еще 2 внутриклеточных (итого - 7 ребер).

Мы запоминаем, откуда мы пришли в точки M, X, Y. Когда мы дойдем до конца, мы начнем обратную раскрутку.

Во-первых, это, как правило, применяется в варианте Смита-Ватермана к поиску наилучшего пути в графе (не оптимального пути из начала в конец, а в принципе самого тяжелого пути). Этим обусловлен 0 в (18). В (16) и (17) 0 писать не надо, поскольку M_{ij} всегда больше, чем Y_{ij} и X_{ij}.

Может ли наилучшее выравнивание закончится делецией? Наверное, нет.

На данный момент, выравнивание с аффинными штрафами за делецию является самым лучшим выравниванием.

Параметры выравнивания обычно следующие:

$d_o = 10$

$d_e = 1$

Таким образом, мы сильно штрафуем за открытие, но слабо штрафуем за удлинение.

На самом деле, здесь возможно несколько вариантов. Один из них заключается в следующем: мы можем построить граф, искать наилучший путь, а затем искать все пути, у которых M_{ij} дает некий локальный максимум. После этого мы видим альтернативные пути. Можно сделать довольно хитрую модификацию алгоритма, которая позволит нам находить не один, а несколько оптимальных путей. В некотором смысле эти пути будут локально максимальными. Это задача решается довольно легко.

Есть еще одно обобщение для данного алгоритма. Если мы посмотрим, например, на выравнивание пространственных структур белков, то мы увидим, что у нас некоторые части структур более-менее соответствуют друг другу, но при этом есть некоторые места (петли), которые сильно отличаются друг от друга. Эти петли эволюционируют иначе, чем вся остальная последовательность.

Если строить выравнивание двух таких последовательностей, то мы "насильно"заставим эти участки соответствовать друг другу, несмотря на то, что они не имеют ничего общего между собой. Эти участки могут быть не только петлями, но и линкерами между доменами. Они не обязаны быть выравнены.

Как сделать так, чтобы эти петли не сравнивались друг с другом? Когда мы сравниваем последовательности, мы пишем матрицу сопоставления, по которой буквы эволюционно заменяются друг на друга. Нам бы хотелось, чтобы штрафы за такие участки были меньше.

В этом случае, мы добавляем еще одну вершину и соответствующие ребра:

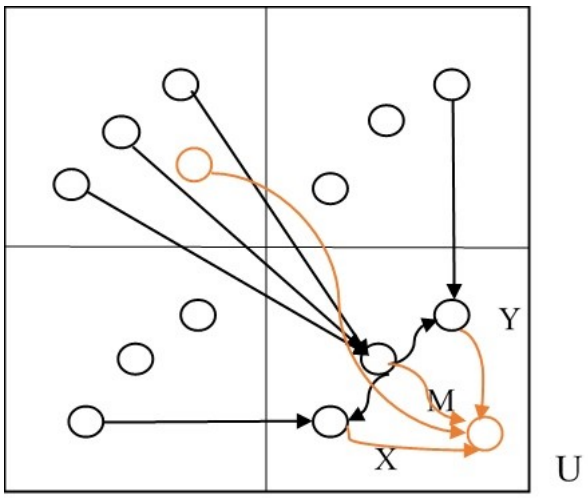

Рис. 6.2 Выравнивание с аффинными штрафами за делецию (дополнение)

Оранжевые вершины и ребра соответствуют тому состоянию, при котором мы не должны ничего выравнивать. Из этого состояния мы можем перейти в состояние, при котором мы будем выравнивать.

Обозначим оранжевую вершину как U.

Либо мы выравниваем, либо мы заканчиваем выравнивать и переходим в состояние U, после чего мы не выравниваем, и после этого опять начинаем выравнивать. Также если мы не выравниваем, то мы можем начать делецию. Мы можем написать следующее:

$$U_{ij} = max \begin{cases} U_{i-1,j-1} - u_e \\ M_{ij} - u_o \\ X_{i-1,j} \\ Y_{i,j-1} \end{cases}$$

Мы либо начинаем (строка 2), либо продолжаем (строка 1) невыравненный участок, либо прыгаем в невыравненный участок из делеции (строки 3,4).

Также у нас добавился еще один переход в M, тогда мы добавляем следующее (см. последнюю строку):

$$M_{ij} = max \begin{cases} 0 \\ M_{i-1,j-1} + M(X_iY_j) \\ X_{i-1,j-1} + M(X_iY_j) \\ Y_{i-1,j-1} + M(X_iY_j) \\ U_{i-1,j-1} + M(X_iY_j) \end{cases} \tag{18}$$

Вот этот алгоритм работает лучше всего. Ни в одном стандартном пакете такого дополнения не реализовано.

Пусть у нас есть алгоритм Смита-Ватермана с аффинными штрафами. Был сделан специальный процессор, который решал задачу по этому конкретному алгоритму. Этот процессор работал очень быстро и эффективно. Однако, у него было проблема: он был дорогой, поскольку не использовался массово, а разработка стоила много.

В начале 90-х был сделан специальный процессор для искусственных нейронных сетей. Однако, потребителей также оказалось мало, особенно в то время.

Сейчас было бы интересно сделать и это, и алгоритм Смита-Ватермана с аффинными штрафами не на специальном процессоре, а на граф-процессоре. Мы бы получили большую скорость решения задач.

Проблема быстрого поиска по банку данных

Алгоритм Смита-Ватермана со всеми дополнениями имеет квадратичную сложность:

$$T \sim O(N^2)$$

Это достаточно долго. В 90-х годах возникла проблема быстрого поиска по банку данных. Было найдено несколько разных подхода, но прижился только один.

Подход был такой:

$X \to arc(X)$

$Y \to arc(Y)$

$X + Y \to arc(X + Y)$

Если $|arc(X+Y)| \sim |arc(X)|+|arc(Y)|$, то они заведомо непохожи. Если же сумма намного больше, значит там есть что-то похожее.

Мы получаем линейную сложность, причем мы не строим никакого выравнивания. Мы не знаем, что конкретно похоже.

Также была другая идея - сравнение частотных словарей. Мы берем последовательность, считаем, какие комбинации букв встречаются часто, сравниваем два вектора, и получаем некое число, которое характеризует, насколько последовательности похожи.

Однако, сработала идея Дэвида Липмана - директора NCBI. По разным историческим причинам молекулярная биология зародилась у физиков-ядерщиков. Изначально институты молекулярной генетики были подразделениями радиобиологического отдела Курчатовского института. В России молекулярная биология и генетика функционировали под крышей физиков. В Америке также были радиобиологические проблемы и приборы. Главный прибор молекулярной биологии - электрофорез. Однако, в те времена его не было. Но у физиков была центрифуга, которой они разделяли изотопы. Вся начальная молекулярная биология делалась на центрифугах. Ученые на них могли ловить одиночные замены в геноме *e. coli*. Также ее геном был с довольно большой точностью откартирован.

Дэвид Липман - полковник американской армии, поскольку генбанк изначально делался в Лос-Аламосской национальной лаборатории в рамках атомного проекта. Только в конце 80-х он выделился в отдельное государственное подразделение. До сих пор министерство энергетики дает очень большие деньги на биологию.

В начале нулевых более трети населения генбанка - это русские.

Алгоритм FASTA

Дэвид Липман примерно в 1988 году совместно с Уилбором придумал следующую схему, которую назвал хешированием.

Сначала для банка данных строится таблица. Составляем тройки аминокислот и смотрим, в каких местах они встречаются (последовательность, позиция). По всем возможным тройкам мы строим эти ссылки, указывающие на места, где они встречаются.

Далее представим, что у нас есть последовательность запроса. Мы берем последовательность запроса, допустим, MGILVKAFG...

Далее мы берем первую тройку аминокислотных остатков - MGI - и смотрим по хеш-таблице, где она встречается. Мы мгновенно находим нужные строки, в которых она встречается.

Затем мы берем тройку GIL и также по хеш-таблице смотрим, где она встречается в банке. Таким же образом мы ищем тройку ILV.

Отмечая найденные тройки в генбанке, мы получаем диагонали, которые по-русски называют *затравками* (seeds). Иногда они могут пересекаться. Каждая такая затравка имеет параметр - номер диагонали $(i - j)$.

i - вертикаль (позиция в одной последовательности - в запросе)

j - горизонталь (позиция в другой последовательности - в банке)

Таким образом, мы можем отсортировать затравки в каждой последовательности по номеру диагонали, что очень быстро. Мы увидим, что на некоторых диагоналях, оказывается, живет много затравок. Соответственно, мощность диагонали определяется следующим образом: $W(diag)$ =число затравок.

Важно, что такая сортировка массива происходит быстро. Просмотрим массив сверху вниз, мы легко сможем такие участки массива, где много троек с одинаковой диагональю.

Мы отбираем те диагонали, на которых много затравок (например, больше трех). Такие диагонали мы считаем перспективными.

Далее мы применяем алгоритм Смита-Ватермана. Однако, мы делаем его, не просматривая весь массив (все диагонали), а просматривая лишь узкую полоску. Нас интересует только ограниченное число путей на графе. Полоска этих путей занимает очень небольшую часть пространства. Мы получаем вес затравок, находящихся на перспективных диагоналях.

На одной последовательности может быть несколько перспективных диагоналей.

Время работа алгоритма Смита-Ватермана будет равно длине последовательности n, помноженной на ширину полосы L:

$$T \sim \mathrm{O}(nL)$$

Таким образом, сложность алгоритма получается гораздо меньшей, чем квадратичная, особенно, если L мала.

Таким образом, в узкой полосе мы получили score.

Далее мы случайным образом отбираем много пар последовательностей и строим между ними весь описанный выше алгоритм, получая score.

Score = 0, если нет ни одной диагонали. Если диагонали есть, то он положителен.

В результате мы получаем распределение, в котором ясно видно, что много пар имеют маленький score и мало - большой.

Далее мы делаем преобразование score в z-score:

У нас есть распределения для score. Мы приведем их к стандартному виду, а именно вместо score напишем следующее:

$$\frac{score - M(score)}{\sigma(score)}$$

Зная распределение, мы можем посчитать мат. ожидание и $\sigma(score)$.

Если наш score был произвольно нормально распределен, то мы переходим к стандартному нормальному распределению.

В результате у нас получается распределение, которое тоже как-то устроено.

Нас интересует хвост, в котором score очень большой. Поскольку мы сравнивали случайные последовательности из банка, z-score не попадет в область хвоста, поэтому мы не можем посчитать p-value.

Тем не менее, мы условно называем получившееся распределение нормальным, хоть оно и не является таковым, и получаем p-value.

Чтобы получить p-value, равный 10^{-6}, нужно провести не менее 10 млн. опытов. Если нам нужен p-value, равный 10^{-12}, то мы вообще вряд ли когда-нибудь получим его в случайных испытаниях. В этом случае мы делаем некоторое ограниченное количество испытаний, определяем M и σ, далее из каких-то соображений мы делаем предположение о том, как устроено распределение. Затем мы делаем экстраполяцию на хвост распределения. В работа Уилбора-Липмана считалось, что z-score асимптотически нормален.

Хеширование в этой науке называется look-up table. Хеш-таблица строится всего один раз, поскольку является свойством банка.

Важно то, что распределение было получено из модели, в которой мы случайно "дергали"последовательности из банка. Поэтому p-value получается очень хорошим.

Описанный выше алгоритм носит название FASTA. Сейчас им пользуются редко, но пользуются. Он обладает некоторыми свойствами, отличающимися от BLAST.

В первую очередь, FASTA работает медленнее (примерно раз в 10). Во-вторых, он работает лучше (дает лучшие результаты).

У FASTA есть один серьезный минус. Если 4 буквы совпадают, то две затравки статистически не очень хорошие. Мы получаем лишние просмотры, которые не следовало делать.

Алгоритм BLAST

Этот алгоритм разработали Липман и Альтшуль.

Мы также строим хеш-таблицу для всего банка, которая устроена, как и в алгоритме FASTA.

Далее мы находим диагональ и затравку. Эту затравку мы пробуем расширить вправо и влево, не допуская делеций. Мы с помощью этой затравки вытягиваем из банка одну из последовательностей:

Q: INQ*MGI*LNKY G

GB1: LND*MGI*VMRK W

На концах у нас большой score. При продвижении вдоль последовательностей от затравки score сначала растет, а потом падает. Мы расширяем затравку в одну сторону до тех пор, пока не получим отрицательный вес. После этого мы находим максимум. Аналогично мы поступаем с другой стороной.

Эта простая линейная операция. Мы получаем некую пару позиций, а также хороший фрагмент последовательностей. У него есть score по матрице.

Далее решается некая вероятностная задача. Она звучит так:

Какова вероятность увидеть такой score по случайным причинам? HSP - high scoring pair - пары расширенных затравок. Сколько таких пар мы ожидаем увидеть по случайным причинам во всем банке?

Здесь мы принимаем бернуллиевскую модель генерации последовательностей. Из score мы получаем ожидаемое количество пар, которые мы ожидаем увидеть в банке. Отсюда мы получаем e-value.

e-value является неким экспоненциальным распределением максимума:

$$e - value \sim Ke^{-\lambda Score}$$

λ - наименьший корень некоторого уравнения, которое получается из матрицы

K - некоторая константа

Значения K и λ могут быть оценены.

e-value - это ожидаемое количество HSP в банке данных. Далее мы ставим порог на e-value, в соответствии с которым мы отбрасываем значения, которые меньше данного порога.

Таким образом, основная идея BLAST заключается в том, что мы берем seed, начинаем расширять его вправо и влево. Затем мы считаем e-value и выбрасываем лишние значения. Затем мы опять берем seed, считаем e-value и вновь смотрим, подходит ли оно.

Если последовательность совпала случайно, то, скорее всего, она довольно быстро оборвется, так как пойдут совершенно не похожие буквы. Если же совпадающие буквы пошли достаточно далеко, то мы тратим время на то, чтобы пройти далеко. Таким образом, мы тратим время только на интересных нам местах. В генбанке на одну и ту же затравку может попасть несколько совпадающих последовательностей. Мы расширяем любое совпадение.

Вторая версия BLAST

Формула Альтшуля:

$$e - value \sim NmKe^{-\lambda S} \tag{19}$$

N - длина банка

m - длина последовательности

Поскольку со временем банк менялся, e-value увеличивался.

На какое e-value имеет смысл смотреть? Ответ - на e-value 10^{-6}. На e-value $10^{-}3$ смотреть уже не стоит. Если значение e-value маленькое, то оно примерно равно p-value.

Мы знаем, что p-value, равный 1-5% - это очень хорошо. Значит, при p-value, равном 10^{-2} мы должны получать отличные результаты. Почему же нам подходит $10^{-}6$, если он является запредельной статистической значимостью?

Формула (19) была выведена из предположения бернуллиевских последовательностей. Она работает правильно, если у нас все последовательности - бернуллиевские с правильными частотами. Если бы это было так, то смотреть на e-value, равный $10^{-}2$, нормально. Однако, наши последовательности не являются бернуллиевскими. Поэтому расхождение между теоретическим и реальным e-value обусловлено тем, что банк устроен совсем не так, как заложено в формуле (19). Эта формула дает сильно завышенную оценку.

Используя BLAST, можно заметить, что есть некие провалы, которые никак не могут нас удовлетворять. Допустим, у нас есть затравки:

QMG

Q I G

Эти затравки примерно одинаковые. Однако, поскольку буквы разные, простое хеширование говорит нам, что зацепки за такой seed не будет. Поэтому было сделано следующее:

У каждой затравки есть некоторое количество затравок, которые похожи на нее. Такие затравки называются T-соседями. Это затравки, у которых:

$$Score(Seed1, Seed2) \geq T$$

T - некий порог

Когда мы строим хеш-таблицу, напротив затравок приписываем их T-соседей. Таким образом, хеш-таблица получается намного больше, затравок становиться больше. Ситуация усложняется.

Эффективность алгоритма заключалась в том, что затравок было мало. Однако, несмотря на снижение эффективности работы при увеличении количества затравок, повышается ее точность.

При этом мы будем смотреть только на те затравки, которые образуют перспективную диагональ (вспоминаем FASTA), и только те затравки мы будем расширять.

Расширение затравок мы будем делать не просто линейно, вдоль диагонали, а в узкой полосе. То есть допускаются маленькие инделы. Мы также ищем максимум и получаем высокоскорную (от англ. score) пару, считаем e-value и т.д.

У современной версии BLAST есть много дополнений. Мы рассмотрели только общую суть.

Лекция 7

Задача поиска сходства в нуклеотидных последовательностях

Выравнивания, рассмотренные в предыдущих лекциях, нужны для поиска сходства. Поиск сходства, на самом деле, нужен не только для поиска гомологии. Он нужен также для массы других задач.

Допустим, мы сделали секвенирование генома. Нам нужно положить прочтения, которые мы получили, на этот геном. Это также является задачей поиска сходства, хоть она и отличается от задачи поиска гомологии, поскольку чтение, как правило, без ошибок или с редкими ошибками. В этой задаче в модель не вкладывается никакой эволюционный процесс. Вообще говоря, задача выглядит так:

Дан текст (T) и паттерны (P, около 10 млн.). Нужно прогнать паттерны по тексту. Текст один и тот же, а паттерны разные.

Что нам говорит компьютерная наука о том, что надо сделать для того, чтобы искать это? Надо сделать преподготовку текста (например, сделать индексную таблицу), а затем класть на него паттерны.

Какие способы преподготовки текста мы знаем? Это суффиксное дерево и суффиксный массив. Суффиксное дерево довольно громоздкое и занимает много памяти. Суффиксный массив также громоздкий, но он проще и лучше, чем дерево.

Чтобы собрать все суффиксы в суффиксном дереве, нам нужно идти по дереву до конца, либо делать некие дополнительные структуры данных, которые позволят из суффиксного дерева сделать список.

В суффиксном массиве мы бинарным поиском находим одно вхождение. Еще куча суффиксов будет иметь это первое слово. Они будут лежать рядом. То есть все суффиксы, имеющее общее начало, будут жить рядом. Таким образом, мы собираем все довольно быстро. Именно поэтому суффиксный массив лучше, чем дерево.

Поиск в суффиксном массиве

Поиск в суффиксном массиве - это:

$$|P| \log |T|$$

Слово "Миссисипи"длинное, его долго писать. Поэтому во всех лекциях и других материалах, посвященных данной теме, используется более короткое слово. Важно, чтобы в слове были какие-либо повторы. Все любят использовать слово "BANANA".

Построим суффиксный массив:

B A N A N A $
1 2 3 4 5 6 7

В начале суффиксного массива будет пустой символ $, который обычно считается самым маленьким при всех сравнениях. Он ставится в начало.

1 7 $
2 6 A$
3 4 ANA$
4 2 ANANA$
5 1 BANANA$
6 5 NA$
7 3 NANA$

Допишем этот массив до конца (зациклим):

1 7 $BANANA
2 6 A$BANAN
3 4 ANA$BAN
4 2 ANANA$B
5 1 BANANA$
6 5 NA$BANA
7 3 NANA$BA

Ничего не изменилось. Мы не храним это в памяти. Такая запись нужна лишь для понимания.

Выделим последний столбец:

1 7 $BANAN |A
2 6 A$BANA |N
3 4 ANA$BA |N
4 2 ANANA$ |B
5 1 BANANA |$
6 5 NA$BAN |A
7 3 NANA$B |A

Последний столбец в таком зацикленном суффиксном массиве называется преобразованием Барроуза-Уилера (BWT, bowtie). Все картировщики так или иначе используют это преобразование.

Как использовать это преобразование?

В результате преобразования у нас получилась строка ANNB$AA. Если мы знаем только эту преобразованную строку, то мы можем восстановить исходную строку.

Буква в последнем столбике - предыдущая от буквы в первом столбике. Отсортируем буквы в последнем столбике:

$
A
A
A
B
N
N

Затем мы прикладываем неотсортированный столбик к отсортированному и получаем следующее:

A $
NA
NA
BA
$ B
AN
AN

Отсортируем это снова и приложим неотсортированный столбик:

A $ B
N A $
N AN
B AN
$ BA
A NA
A NA

Мы снова будем сортировать и прикладывать, и в конце концов получим исходный зацикленный массив. Однако, такое восстановление занимает много времени. Его можно сделать гораздо быстрее:

Для начала пронумеруем буквы в слове:

$B_1A_1N_1A_2N_2A_3\$_1$

1 2 3 4 5 6 7

Далее переносим эту нумерацию на массив:

1 $\quad \$_1BANANA_3$

2 $\quad A_3\$BANAN_2$

3 $\quad A_2NA\$BAN_1$

4	$A_1NANA\$B_1$
5	$B_1ANANA\$_1$
6	$N_2A\$BANA_2$
7	$N_1ANA\$BA_1$

Мы видим, что и в первом, и в последнем столбиках сверху-вниз нумерация A идет в порядке 3→2→1. N и в первом, и в последнем столбике идут в порядке 2→1.

Таким образом, порядок следования букв каждого типа и в первом, и в последнем столбике одинаков. Это логично, поскольку буквы в первом столбике - это буквы, которые идут после тех, что стоят в последнем столбике.

Функция LastFirst

Введем функцию LastFirst (LF):

$$LF(\text{номера строки})$$

Для каждой строки эта функция будет равна номеру той строки, в которой последняя буква в этой строке является первой.

Посчитаем LF каждой строки:

		LF
1	$\$_1BANANA_3$	2
2	$A_3\$BANAN_2$	6
3	$A_2NA\$BAN_1$	7
4	$A_1NANA\$B_1$	5
5	$B_1ANANA\$_1$	1
6	$N_2A\$BANA_2$	3
7	$N_1ANA\$BA_1$	4

Заметим, что все получившиеся числа разные. Если у нас есть значения LF, то мы можем быстро восстановить исходное слово.

Мы знаем последний символ, и мы с него начинаем ($). Далее мы идем по LF и смотрим, какой символ был перед ним. Соответственно, мы получим наше слово $B_1A_1N_1A_2N_2A_3\$_1$. Таким образом, нам нужен только номер строки и последняя и первая буквы.

Массивы C(a) и occurence

Нам бы не хотелось обращаться к первому столбцу вообще. Построение LF довольно долгое как раз из-за этого. Мы сделаем следующее:

У нас есть наблюдение, что номера букв идут в одном и том же порядке и в первом, и в последнем столбце. Мы введем следующие массивы:

Первый массив будет размером с алфавит. В алфавите у нас 4 буквы. Запомним, где начинается блок с $ в первом столбце (1 строка), а также где начинаются A (2 строка), B (5 строка) и N (6 строка). На всякий случай, мы вычтем 1 (чтобы удобно было все адаптировать для языка программирования, в котором нумерация начинается с нуля). Таким образом, массив C(a) будет устроен так:

$ 0

A 1

B 4

N 5

Если мы вычтем из числа в какой-либо строке массива предыдущее число, то получим количество букв в предыдущей строке. Если мы вычтем из всей длины слова (7) число в последней строке (5), то получим количество букв N (2).

Таким образом, с помощью данного массива мы легко можем посчитать, сколько каких букв у нас есть в слове.

Следующий массив - occurence. Мы вводим этот массив по количеству букв, хотя для построения LF occurence можно не писать.

В таблице ниже записано, сколько каких букв встретилось в позиции i и выше в массиве BWT.

$\mathrm{Oc}(\alpha, i)$			
$	A	B	N
0	1	0	0
0	1	0	1
0	1	0	2
0	1	1	2
1	1	1	2
1	2	1	2
1	3	1	2

Зная, сколько определенных букв в позиции i и выше, мы можем сказать, в какой позиции первого столбика находится буква с определенным индексом из последнего столбика. Для этого нужно спуститься от первой строки на количество строк, равное количеству букв в индексе i и выше.

Напишем псевдокод:

T=$, i = 1

% Мы прыгаем по LF до тех пор, пока не вернемся к $ и не зациклим слово. Мы начинаем и заканчиваем $.

% Первая буква живет в первой строке, поскольку она была перед $.

a = BWT[i]
T = aT
% В первом круге мы получаем $ A. Затем мы должны пройти дальше по occurence.
i = C(a) + Oc(a, i)
while a $\neq$ $

Таким образом, мы будем шаг за шагом восстанавливать наше слово. Заметим, что из occurence мы можем легко получить LF.

Поиск паттерна

Алгоритм выше имеет линейное время. Такую идею можно использовать и для поиска. Однако, при этом нужно помнить, что геном человека занимает много памяти, как и комплементарная цепь. Все это мы еще должны умножить на 4, в соответствии с 4мя основаниями в генетическом коде. Таким образом, плохо то, что массив occurence большой.

Тем не менее, попробуем поискать паттерн. Что значит, найти, например, паттерн ANA? Это значит в суффиксном массиве найти следующий участок:

3 4 $A_2NA\$BAN_1$
4 2 $A_1NANA\$B_1$

Этот участок начинается везде, где начинаются буквы A. Далее мы напечатаем все значения суффиксного массива для этого участка. Таким образом, надо найти фрагмент суффиксного массива, в котором суффикс начинается с нашего паттерна.

Стандартный поиск имеет логарифмическую сложность:

$$T \sim |P| \cdot \log |T|$$

Поскольку наш текст очень большой, логарифмическая сложность нам не подходит.

Мы начнем с последней буквы паттерна - A: она живет от 1 до 3. Нам нужно узнать, где живут предыдущие буквы.

Нахождение паттерна означает нахождение нижней и верхней его границы в массиве.

Формула для нахождения нижней границы:

$$L(aW) = C(a) + Oc(a, L(W) - 1) + 1 = 6$$

Таким образом, нам нужно откартировать все буквы A, чтобы получить все фрагменты NA.

Формула для нахождения верхней границы:

$$U(aw) = C(a) + aC(a, U(w)) = 7$$

Все это опирается на утверждение о том, что порядок следования букв в BWT такой же, как и порядок следования букв в первом столбце.

Нижняя граница для A находится в 1, верхняя - 3 (мы можем получить границы, просто посмотрев на массив C(a) и не забыв прибавить единицу, которую мы вычитали при его составлении), следовательно, получаем:

$$A \quad 2...4$$

Нижняя граница:
$L = 5 + Oc(N, 1) = 6$
Верхняя граница:
$5 + Oc(N, 5) = 7$
Таким образом, мы знаем, что NA живет в интервале от 6 до 7:

$$NA \quad 6...7$$

Нижняя граница для ANA:

$$L(ANA) = 1 + Oc(A, 6 - 1) + 1 = 3$$

$$U(ANA) = 1 + aC(A, 7) = 4$$

Таким образом:

$$ANA \quad 3...4$$

Таким образом, за линейное время мы нашли паттерн ANA.

Сокращение массивов

Большие объемы требуемой памяти остаются большой проблемой в поиске паттерна. Массив occurence весит очень много:

$$\text{4 байта (int)} \cdot \text{4 (размер алфавита)} \cdot \text{6 Гб (геном человека)} = \text{48 Гб}$$

Нам также нужно выделять память для суффиксного массива и строки Бороуза-Уилера:

$$48 \text{ Гб} + 4 \cdot 6 \text{ Гб} + 6 \text{ Гб}$$

Нужно сделать так, чтобы массив occurence хранился в памяти не целиком, а только каждая четвертая его строка. С помощью массива Бороуза-Уилера мы легко сможем восстановить остальные строки. В оригинальной статье хранится каждая 128 строка (позиция).

Каждую букву в BWT мы можем помнить в двух битах. Это сокращает требуемый объем памяти в 4 раза.

64 бита - длина слова "LONG"в языке C. Как мы можем посчитать occurence, проходя вниз по массиву, который лежит в двух однобитовых массивах? Нам нужно считать количество букв, например, A.

Если мы укладываем массив в два однобитовых массива, то каждый разряд такого парного массива - это одна буква. Кодируем следующим образом:

$A/T \to 0$

$A/G \to 0$

В других случаях $\to 0$.

Теперь посчитаем количество букв A в следующих массивах:

10110011011

00101010110

Для этого мы логически умножаем первую последовательность 0 и 1 на вторую последовательность и берем отрицание. Там, где 0, будет 1, а где 1 - 0. Таким образом, с помощью одной логической операции мы из этих последовательностей можем получить битовую маску:

01000100000

Далее нам нужно количество букв A, которое дошло до 4го символа. Для этого нам нужна логическая маска. Ее нужно умножить на следующую маску:

1111000000

Мы получаем следующую битовую маску:

010000

Нам нужно посчитать количество 1 в маске. Мы можем считать их с помощью сдвига (64 раза). Однако, можно сделать это проще:

LONGLONG - это 8 байт. У нас есть число, которое состоит из 8 байт. Разнообразие байта - 256. Мы можем заранее заготовить небольшую таблицу размером 256 байт:

00 0
01 1
02 1
03 2
.
.
.

FF 8

Каждый фрагмент в маске - номер в этой табличке. По таблице мы смотрим, сколько байтов занимает этот фрагмент. Количество ненулевых байт в маске можно посчитать с помощью восьми обращений к таблице. Нет необходимости поштучно считать биты. Мы получаем определенное число, используя простые логические операции. Это легко делается на языке C.

Таким образом, у нас есть возможность сократить массив occurence в 128 раз.

Нам остается сократить большой суффиксный массив. Его также можно сжать, однако в данном курсе этот аспект рассмотрен не будет.

Лекция 8

Нулевые гипотезы и p-value

Мы видим много p-values в различных работах. Люди любят применять их, поскольку их более-менее легко считать. Вообще говоря, p-value - это очень хорошая вещь, несмотря на критику.

Представим себе, что мы получили некий результат. Мы исходим из того, что этот результат случайный и что смысла в полученных данных нет. По-научному это называется нулевой гипотезой (H_0). Эта гипотеза глубока и позволяет посчитать определенные вероятности.

Давайте подумаем о том, какая будет вероятность нашего или еще более "кривого"наблюдения при условии того, что все плохо, что означало бы, что нулевая гипотеза верна. Что значит более кривое наблюдение? Это мы понимаем тогда, когда понимаем задачу, над которой думаем. Скажем, мы видим, что у нас есть определенное количество больных и здоровых людей или носителей и не носителей гена, и мы их всех посчитали. У нас есть определенная мера ассоциации болезни с геном, и мы говорим, что более "кривые это те, у которых мера ассоциации еще выше, чем то, что мы видим. То есть мы сразу "запихали"все, что мы видим или могли бы увидеть в хвост функции распределения. Вес этого хвоста мы и называем p-value. Это вполне разумная вероятность.

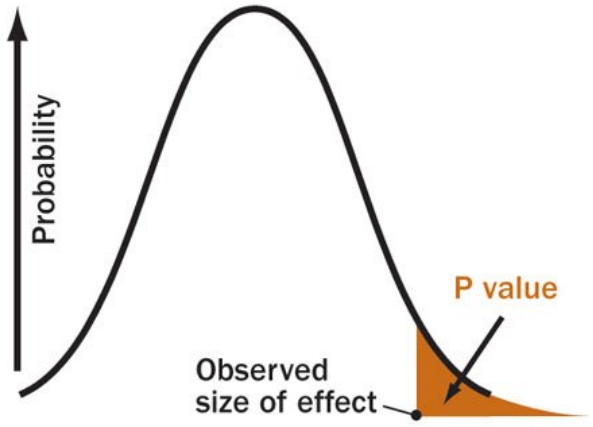

Рис. 8.1 P-value

Формула для вычисления p-value:

$$p - value = \int_{\xi_{obs}}^{\infty} p(\xi)d\xi = 1 - cdf(\xi_{obs})$$

Понять, какие результаты являются более или менее странными, тяжело. Наше наблюдение образует многомерный параллелепипед. Если выйти за его пределы, то наблюдение является более странным. Также можно рисовать сферу. Многомерные p-value определять тяжело.

Можно считать p-value по каждой из координат, а далее объединять их методом Фишера. На самом деле, таких методов несколько.

Двусторонний p-value

Также странные наблюдения могут быть с обеих сторон (два хвоста). Допустим, у нас есть наблюдение 5. Конечно, наблюдение 6 будет более "странным чем 5, а что насчет -6?

Если, в силу нашей задачи, 5 означает, что у нас есть стандартные ситуации от -5 до 5, то все, что лежит вне этих границ, является нестандартной ситуацией. Например, у нас есть ген, который ассоциирован с болезнью, но мы не знаем заранее, в какую сторону. При этом нам все равно, в какую сторону направлена эта ассоциация, нам важен только факт ее наличия. В этом случае мы считаем сумму двух хвостов. Таким образом, формула для подсчета p-value видоизменяется.

Обычно люди говорят про односторонний (one-sided) p-value, поскольку мы, как правило, заранее знаем, в какую сторону направлена наша ассоциация. В каком-то смысле это разумная байесовская вероятность. Однако, вопрос в том, что это за вероятность.

P-values и Байесовская парадигма

Байесовская парадигма устроена следующим образом: у нас есть некоторые знания о мире, которые мы называем прайором (prior). Затем происходит нечто такое, отчего наши знания изменяются, такие знания мы называем пассириором (passirior). При этом вероятности того, что, собственно, произошло, мы называем лайклехудом (likelyhood). Это переходные вероятности, которые перемножаются с прайором, и получается пассириоры:

$$p(null) \sim P(event|null) \cdot prior(null) \tag{20}$$

В этом интересно то, что мы когда все произошло, то наши новые знания о мире - прайор для последующих наблюдений. Так мы можем делать целые цепочки.

В данном случае мы считаем вероятности мерой человеческого доверия к какому-либо сообщению. В этом смысле p-value ни чуть не лучше, и не хуже. У нас есть нулевая гипотеза, гласящая, что все плохо, а теперь появились некоторые наблюдения, о которых мы говорили выше.

Вероятность нулевой гипотезы изменится от этого наблюдения. Почему мы так уверенно говорим, что, если у нас p-value равен 10^-10, то мы отвергнем нулевую гипотезу? Поскольку в формуле (20) множитель оказывается равным 10^{-10}. Понятно, что произведение становится более маленьким, и мы меньше доверяем нулевой гипотезе.

С другой стороны, мы должны четко понимать, что лайклехуды обладают множеством странных свойств. Если мы аккуратно рассмотрим все возможные гипотезы в мире и сложим их, то в сумме они дадут 1. Лайклехуд же не обязан вести себя так, поскольку это - условная вероятность события в предположениях. Мы спрашиваем, не какова вероятность нулевой гипотезы, а какова вероятность того, что наши данные соответствуют этой гипотезе. Понятно, что лайклехуд должен суммироваться на все возможные события, а не на все возможные предположения (гипотезы). Поэтому, на самом деле, непонятно с чем сравнивать 10^-8, приписанное к нулевой гипотезе. Возможно, к альтернативной гипотезе должно быть приписано 10^{-11}. Аналоги p-value для разных гипотез не должны в сумме давать 1. Поэтому с ними непросто.

Таким образом, лайклехуд (термин из байесовской теории) - это вероятность получить некое наблюдение при условии некоторой априорной гипотезы (чего-то, что мы проверяем). P-value - это частный случай лайклехуда для нулевой гипотезы ("все плохо"). Байесовская теория работает для любых априорных гипотезах, а вся теория p-value построена на том, что наши данные бессмысленны и ассоциации нет, и все возможные корреляции получены случайным образом. При этом и само событие, которое рассматривает p-value, не одно, а целый хвост (события от нашего и еще страннее).

Лайклехуд всегда устроен так: у нас есть одна гипотеза - то, что у Байеса справа от условия, а также есть одно наблюдение, и мы считаем вероятность этого наблюдения. Вообще говоря, люди часто этим и ограничиваются, не понимая, что нужно приписать прайор к теории. Мы здесь делаем две вещи: во-первых, мы рассматриваем именно нулевую гипотезу, во-вторых, наш евент - это хвост, который в этом смысле нормирован, поскольку одно событие, вероятность которого мы считаем по Байесу, может значит все, что угодно, а мы считаем целый хвост. Понятно, что хвост в знаменателе всегда содержит единицу, отражающую распределение.

Под графиком на рис. 8.1 - плотность вероятности, а p-value - это отношение площади под хвостом к площади под основным графиком, а также это - просто площадь хвостика, поскольку площадь под всем графиком равна 1 (распределение вероятностей). В этом смысле они сильно лучше нормированы, чем лайклехуды в

общем случае.

То, что p-value - это вероятность хвоста, является и силой, и слабостью. С одной стороны, мы сразу же получаем нормировку только за то, что это хвост. Если хвост будет очень плохим и большим, то мы получим максимум единицу (максимальное значение p-value).

С другой стороны, из-за того, что мы работаем с хвостами, мы меньше знаем о самом нашем событии. Однако, когда у нас все переменные континуальны, нам трудно говорить о вероятности конкретного события. Например, что такое число 5? Чем оно отличается от $5 + 10^{-6}$? Для ответа на этот вопрос нужно считать какие-то плотности, делать определенные нарезки. Тут же мы вместо нарезок просто отсекаем хвост от графика. При этом мы знаем, сколько весит этот хвост.

Конечно, у нас всегда есть нулевая гипотеза о том, что все бессмысленно, а также альтернативная гипотеза (сигнал, ассоциация, дифференциальная экспрессия и т.д.), которую мы пытаемся обнаружить в этих данных. Вообще говоря, для обнаружения этой связи, заложенной в альтернативной теории, в большинстве разумных систем анализа данных мы должны сформулировать, что же именно мы ищем (какую конкретную ассоциацию). Когда же мы работаем с p-value, у нас выходит "поблажка": нам ничего не нужно знать про то, что мы ищем, мы лишь подтверждаем факт наличия самой ассоциации. Иногда это хорошо, а иногда плохо, но, в любом случае, это спасает нас от построения модели сигнала, которая часто гораздо сложнее, чем нулевая модель. Нулевая модель заключается в том, что все распределено равномерно, все выпало случайно и др. Когда нам нужно думать о модели сигнала, то обычно все становится сложнее. В модели сигнала мы, так же, как и в нулевой гипотезе, будем рассматривать какой-то случайный процесс, хоть и при условии наличия ассоциации. Однако, написать его так же легко и изящно, как случайный процесс при условии, что ассоциации нет, труднее. Во-первых, этот процесс будет параметрический, и у нас будет определенная сила ассоциации, и нам нужно будет разложить аллели в соответствии с этой силой ассоциации. Также возникает вопрос о том, как сравнивать модели с разным числом параметров. Таким образом, хоть с этим и можно работать, но ситуация становится гораздо более сложной.

Нулевая модель отсутствия сигнала, как правило, описывается проще, поэтому p-value так популярны. Они дают возможность быстро оценить, есть ли смысл в сигнале. Допустим, если p-value = 0,5, то сигнала точно нет, и можно его не искать. При небольших значениях p-value нужно проводить более глубокий анализ.

Порог p-value

Несмотря на обширную критику p-value, заключающуюся в том, что это лишь отвлечение от модели сигнала, мы посмотрим на то огромное количество выпущенной ли-

тературы в мире, использующей p-value и попробуем понять, какой разумный порог нам надо поставить для него. Количество такой литературы лавинообразно растет.

Такая обширная и трудная работа действительно была проведена. С помощью какой-то случайной модели был проанализирован поток литературы с разными порогами p-value и оценено количество ложноположительных результатов. Исследователи, проводившие эту работу, пришли к выводу, что в эру, когда все публикуются, нужно ставить границу 0,005, а не традиционную 0,05.

Порог p-value - это, по сути, вероятность ложно положительного результата.

Разумное употребление p-value

Люди также показали, почему p-value нельзя использовать бездумно и всегда. Они положили лосося в томограф и показывали ему картинки разной эмоциональности. Томограф - это объемная штука, которая измеряет активность движения воды внутри массы. Она смотрит резонанс водорода или фосфора, и если регистрируется поток воды, то мы интерпретируем это как нервную активность. При этом все поле зрения томографа нарезано на множество кусочков. Оказалось, что в лососе не просто происходила существенная нервная активность, но и существенно скоррелированная с эмоциональным фоном показанных лососю картинок. Однако, этот лосось был замороженный! Этого факта могли не заметить невнимательные читатели.

Таким образом, p-value - это вероятность, что означает, что при наличии достаточного количества наблюдений ее возможно пробить. Если наша вероятность 10^{-6}, и мы делаем миллион наблюдений, то хотя бы одно из них будет совершенно случайным и никаким образом не отражающем корреляцию.

Американское статистическое общество утверждает следующее:

Во-первых, нужно думать, прежде чем употреблять p-value, иначе мы можем получить ложно положительный результат.

Во-вторых, p-value ничего не говорит о силе эффекта. Он может быть очень достоверным, но при этом очень слабым. Допустим, какой-то аллель гена меняет восприимчивость к болезни на 0,0001%. Однако, если мы проведем исследование на популяции земного шара, то мы получим очень хороший p-value. Мы увидим хорошую и очень достоверную ассоциацию. Однако, эта ассоциация будет бесполезна, поскольку эффект очень слабый. Также p-value не говорит о том, что эффект полезный. Он говорит о том, что здесь что-то неслучайно.

Как правило, p-value говорит не о связи двух величин, а о том, что есть какая-то третья величина, например, образ жизни, которая влияет на связь этих двух величин.

Вычисление p-value

Во-первых, есть параметрические тесты (например, тест Стьюдента). Большинство людей используют именно их, поскольку в предположении любого параметрического теста лежит предположение о неком нулевом распределении. В самом простом варианте это - распределение Гаусса. Как только мы ввели предположение о гауссиане, у нас возникли параметры (среднее значение, разброс), и далее мы, исходя из этого предположения, можем написать какие-то вероятности для наших наблюдений. С одной стороны, это жульничество, поскольку мы не доказываем гауссовость и т.д. С другой стороны, часто ни на что, кроме этого, не хватает статистики. В общем случае, мы все равно предполагаем, что у нас риды падают не совсем случайно, а в результате какого-то процесса. Как только возникает процесс, возникает какой-то параметрический тест, более или менее сложный.

Во-вторых, это непараметрические тесты. Самый классический из них - тест Вилкоксона (Манна-Уитни). Мы понимаем, что какие-то распределения у нас есть, но мы не пытаемся использовать знания об этих распределениях. Мы предполагаем, что медианы этих распределений совпадают или что-то в этом роде. Мы делаем попарные сравнения двух выборок. Эта группа тестов избегает предположения распределения.

Также у нас есть комбинаторика (тест Фишера). В точном тесте Фишера мы хотим измерить ассоциацию двух факторов.

Вообще, сэр Фишер присутствовал на неком званом обеде, на котором одна из дам сказала ему, что она может на вкус отличить, добавляли ли молоко в чай или чай в молоко. Это задача про ассоциацию (или про распознавание). Нужно поставить опыт, где в 5 кружек мы наливаем чай в молоко, а в 5 других - молоко в чай и дать все попробовать даме. У нас есть таблица событий, в которой записаны два фактора - что действительно добавлено первым и что говорит дама. Нулевая гипотеза заключается в том, что на самом деле, дама не может ничего различать. В таком случае все события будут равномерно раскиданы по этой таблице. Для решения этой задачи и подсчета вероятности сэр Фишер придумал бессмертный тест Фишера, который в наше время используется практически всеми исследователями. Следует отметить, что по этому тесту выяснилось, что дама действительно может различать такие тонкости в чае с молоком.

Таким образом, Фишер превратил ассоциацию в некий комбинаторный фактор. Мы не понимаем, как считать ассоциации, но понимаем, как считать комбинаторные факторы. Он сделал некий параметр, растущий с ассоциацией, который является простым параметром населенности одного из уголков матрицы. Он сказал, что самые кривые матрицы - те, у которых диагонали тяжелые (см. рис. 8.2). Чем выше ассоциация, тем тяжелее диагональ. В этот момент загадочное слово "ассоциация"превращается в нечто, что можно комбинаторно посчитать.

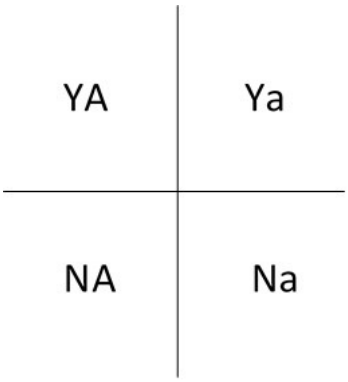

Рис. 8.2 Матрица Фишера

Еще один способ считать p-value - это пермутации. Этот способ используется чаще всего, поскольку посчитать разумную модель мы можем не всегда, а перемешать имеющиеся у нас данные - всегда. В тех же самых случаях с ассоциациями у нас есть какая-то структура популяции, и вообще все данные имеют какие-то скрытые структуры. Именно из-за этих скрытых структур все посчитанные наивно (в лоб) модели, как правило, дают странные p-value. Эти структуры не имеют отношение к вопросу, который мы хотим задать (мы не спрашиваем, не случайна ли наша структура популяции, потому что она, в любом случае, неслучайна).

Мы можем попробовать перемешивать метки "больной"и "здоровый аккуратно сохраняя все остальное - возраста, профессии и все остальные детали истории болезни. Также мы можем сохранить и историю болезни, и метки, а перемешать аллель гена, который нас интересует. Однако, надо помнить, что перемешивать нужно с гаплотипами, поскольку соседние гены всегда ассоциированы. Таким образом, нужно всегда понимать, что мы перемешиваем, а что оставляем на месте.

Так или иначе, если мы правильно сформулируем нашу задачу, то мы всегда можем перемешать так, чтобы получить воображаемый пермутированный мир, в котором все люди устроены так же, однако, именно тот вопрос, который мы рассматриваем, точно не имеет смысла, потому что касающиеся его данные в этом модельном мире перемешаны.

Дальше мы делаем это много раз и получаем какое-то распределение. Чтобы получить распределение, нам нужно сформулировать какую-то статистику. Это может быть и ассоциация, и Фишер, и результат любого параметрического критерия. Нам нужна одна конкретная цифра на перемешанный мир. Много перемешанных миров дают нам распределение этой цифры, после чего мы берем мир, в котором мы все это считали и смотрим, где он находится в этом распределении. Мы получаем то, о чем мы говорили вначале, но кривая распределения получается путем перемешивания миров.

На самом деле, именно так получается большинство разумных результатов. Интересно то, что, вообще говоря, как бы мы ни считали p-value, если мы считаем их

правильно (используем правильную модель), то при верной нулевой гипотезе они должны быть нормально распределены. Это следует просто из определения p-value. Вероятность того, что правильно посчитанный p-value будет меньше какой-то величины, это и есть p-value этой величины. Мы строим его так, что он является вероятностью самого себя (вероятность того, что мы попадем в хвост p-value).

Из этого следует некоторое количество забавных методов, а также неплохой способ тестирования наших моделей. Мы можем накидать пермутаций и посмотреть, что p-value нормально распределены. Если они не окажутся равномерно распределены, это означает, что у нас либо ошибка в программе, либо мы неправильно делаем преобразования, либо у нас отсутствует понимание задачи. Таким образом, это хороший способ тестирования.

Преобразование функций плотности

Также есть общий факт про распределения, который полезно помнить при работе с p-value. Это факт о том, как преобразовывать функции плотности:

У нас есть случайная величина ξ, и $\rho(\xi))$ - это вероятность случайной величины. Пусть у нас есть какая-то функция случайной величины $g(\xi)$, которая также является случайной величиной, которая как-то распределена.

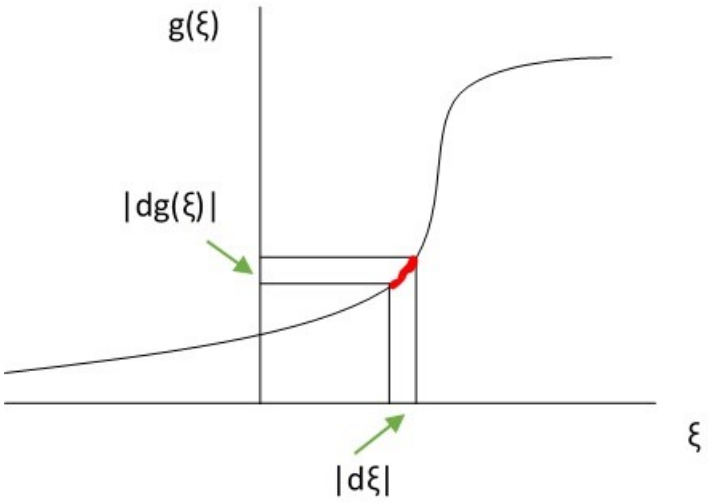

Рис. 8.3 График функции случайной величины $g(\xi)$

Вопрос следующий: если мы знаем плотность $\rho(\xi)$, то как будет выглядеть плотность $\rho(g(\xi))$? Мы берем $\rho(\xi)$ и делим на производную:

$$\rho(g(\xi)) = |dg(\xi)| = \rho(\xi)|d(\xi)|$$

$$\rho(g(\xi)) = \frac{\rho(\xi)}{g'(\xi)} \tag{21}$$

С помощью этой формулы мы получаем нужную нам плотность.

Интуитивно это очень понятно: на отрезок на рис. 8.3, выделенный красным цветом, случайно падали какие-то события. Однако, на самом деле, те события, которые мы видим на этом отрезке ($|dg(\xi)|$) - это те же события, которые мы видим через математическое преобразование (21). Сами события не появляются и не исчезают. Просто мы можем смотреть на них через ξ, а можем - через $g(\xi)$.

На обеих осях отмечаем 15 событий. Размер отрезка $g(\xi)$, на который упали эти 15 событий, деленный на размер отрезка на оси ξ - это производная (по определению). Получается, что плотность этих событий - это плотность событий на отрезке $|d(\xi)|$, деленная на производную.

Далее мы можем вывести следующую формулу через производную (вывод не приведен):

$$\rho(p - value(\xi)) = 1$$

Важная математическая фраза:

Из любого приличного распределения можно сделать любое другое приличное распределение, если есть приличная функция перехода.

Множественные гипотезы. Поправка Бонферрони

Пусть у нас есть 10 тыс. генов. Мы смотрим на какую-то дифференциальную экспрессию и видим, что один из генов дифференциально экспрессируется, исходя из нашей модели дифференциальной экспрессии, а вернее - в модели ее отсутствия, поскольку мы вычисляем p-value. Мы вычисляем p-value для этого гена и получаем значение 10^{-3}. Это хороший p-value, в связи с чем мы можем сделать вывод, что ген участвует в процессе. Однако, проблема в том, что этот ген не один, а 10 тыс. Мы выбирали из этих генов лучший, то есть хотя бы один работающий. Это значит, что у нас есть событие (эксперимент) с вероятностью 10^-3, которое мы повторим 10 тыс. раз, в результате чего мы можем получить и ложно положительный результат, хоть его вероятность и очень мала.

Если мы с самого начала задаемся одним вопросом, то мы получаем ответ на него в виде p-value. Если мы задавались многими вопросами и нам было все равно, на какой из них мы получим ответ "да то p-value соответствующего ответа "да"надо юстировать, поскольку иначе мы будем всегда что-нибудь получать просто в силу того, что при большом количестве наблюдений маловероятное событие рано или поздно произойдет.

Самое простое, что можно сделать - это ввести *поправку Бонферрони*:

Допустим, мы получили p-value 10^{-8} при 1000 проведенных тестах. Если мы умножим $10^{-8} \cdot 1000 = 10^{-5}$, то мы получим поправку Бонферрони:

$$p - value_{corr} = P(any\ p - value(\xi) \neq p) = Np$$

Это самое страшное и жестокое, что можно сделать с p-value. Однако, эта поправка имеет смысл. Более правильно считать ее нужно так:

$$p - value_{corr} = 1 - (1 - p)^N \approx Np\ if\ Np \ll 1 \tag{22}$$

p - малая вероятность события

$(1 - p)^N$ - вероятность того, что событие не произойдет N наблюдений

$1 - (1 - p)^N$ - вероятность того, что событие произойдет хотя бы раз за N наблюдений

Однако, нас интересуют вероятности, сильно далекие от единицы. Когда вероятности далеки от единицы, то все раскладывается в ряд, и мы получаем примерно Np, то есть менее точный способ введения поправки Бонферрони. Таким образом, можно спокойно использовать упрощенную формулу.

Представим, что мы получили данные для нашего гена с помощью PCR. А затем мы размножили наш эксперимент 10 тыс. раз. Таким образом, вместо нашего эксперимента с результатом 10^{-3} мы получили матрицу с 10 тыс. абсолютно одинаковых экспериментов, произведенных случайным образом. У всех этих экспериментов $p - value = 10^{-3}$, что есть плохо.

Нужно ли нам вводить поправку Бонферрони? Из формализма Бонферрони следует, что вводить ее нужно. Однако, здравый смысл подсказывает, что это не требуется, поскольку наш эксперимент один и тот же.

Бонферрони предполагает, что ряд p-values, полученный в разных экспериментах, получен независимо. Мы считаем эти p-value независимыми событиями. Поэтому мы можем спокойно применять формулу (22).

Как только независимость исчезает из наших предположений, мы понимаем, что Бонферрони слишком сильно увеличивает наш p-value.

Если случилось так, что наш эксперимент прошел проверку Бонферрони, то это отлично. Однако, иногда он не работает, но у нас все прекрасно получается по разным причинам.

Одна из причин - это Np, которое получается равным 10 или около того.

Главная же проблема заключается не в независимости. Эксперимент со случайным размножением мы будем обходить чем-то вроде пермутаций. Мы представим себе, что много человек делают такие эксперименты, и мы сравниваем результаты этих людей, поскольку корреляция экспериментов, получившихся от размножения

данных внутри каждого эксперимента одна и та же. Именно для этого нужна пермутация.

Бонферрони - это Famile-Wise Error Rate. Это условно означает, что мы хотим "семью без урода". Когда мы рассматривали задачу, мы выбирали самый лучший и самый маловероятный ген из всех. Мы можем по-разному разделить след из p-value. Мы хотим, чтобы за нашей отсечкой (в данном случае самой простой отсечкой одного самого лучшего) не было ни одного "урода"с верной нулевой гипотезой.

На самом деле, у нас есть ряд экспериментов, в результате которых мы получаем ряд p-value. Мы для удобство понимания нумеруем их по убыванию p-value:

$$0 \leq p_1 \leq p_2 \leq p_3 \leq p_4 \leq ... \leq p_N$$

Далее мы задаемся вопросом о том, как нам стоит разделять этот ряд, чтобы получать осмысленные с нашей точки зрения результаты.

Самый простой способ - взять самое маленькое p-value. Какова вероятность, что эта часть будет хорошей, и нулевая гипотеза будет неверной?

Скорректированный поправкой Бонферрони p-value - это хорошая оценка, но есть и другие способы, которые являются более щадящими.

Метод Холма-Бонферрони

Метод Холма-Бонферрони похож на Уилера. Его идея состоит в том, что если у нас хорошие p-value в ряду, что мы отрезаем не с краю, а посередине, то мы боимся шума уже не от всего набора данных (N), а только от того, что мы оставили как неотвергнутые нулевые гипотезы (справа). Тогда поправку нужно вводить не на N, а на $N - i + 1$, где i - это то место, где мы ставим отсечку.

Холм-Бонферрони доказывается через неизвестный параметр (число настоящих нулей).

Пусть α - это наша вероятность получить "урода"левее отсечки, то есть настоящего нуля. Однако, это и есть тот же самый p-value. Однако теперь это p-value не одного эксперимента, а целой группы.

Во всех остальных случаях, кроме α мы можем доказать, что при таком алгоритме поиска первого i мы не сможем избавиться от α. Поэтому мы можем получать вполне приличные значения, лучшие, чем Бонферрони. Мы экономим много значений по сравнению с Бонферрони.

Экономить значения мы можем разными способами. В данном случае мы делаем это, исходя из того, что если у нас уже получаются более-менее приличные результаты, то мы можем отсекать гораздо правее, и тогда проблема с нулями будет меньше, поскольку и нулей будет меньше.

Холлм-Бонферрони часто используется и дает лучшие результаты, чем Бонферрони. Более того, он так же отвечает на вопрос о том, как бы не получить ни одной неотвергнутой нулевой гипотезы (ошибки первого рода).

Метод пермутаций Вестфолла-Янга

Тут мы возвращается к нашему эксперименту с размножением данных. Мы берем максимум, самое лучшее p-value из каждого перемешивания. Далее мы считаем статистику не по множественным экспериментам, а по лучшим.

Таким образом, все наши взаимосвязи остаются в горизонтальных рядах. Сигнал перемешан, и мы сравниваем его только среди лучших. Это очень медленно, но часто дает гораздо более щадящие поправки, чем Бонферрони. С помощью этого метода мы боремся с независимостью.

С этой вероятностью у нас нет непойманных нулей. Однако, поскольку мы используем только лучшие значения из этих переменных, у нас все происходит медленнее. Однако, иногда это стоит того.

Задачи распознавания

Набор каким-то образом ранжированных экспериментов мы хотим разделить на две группы: "есть сигнал"и "нет сигнала".

Мы временно отходим от p-value, поскольку для него нужно знать только модель при условии шума. Мы же знаем также какое-то распределение при условии сигнала.

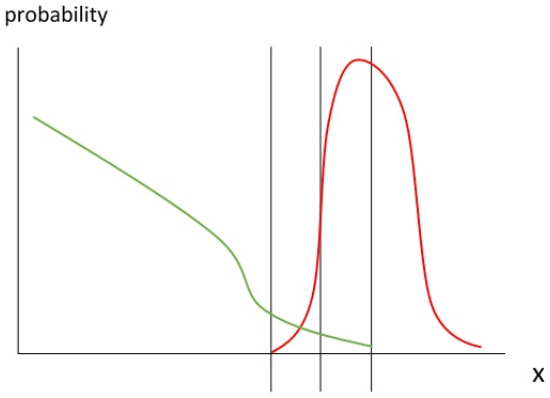

Рис. 8.4 Задача распознавания (зеленое - шум, красное - сигнал)

Параметр x характеризует каждый эксперимент. Где нам стоит поставить отсечку на x, чтобы получить хороший результат?

А что такое хороший результат?

Если нам нужно, чтобы как можно меньше шумовых, то есть "не конфеток"попало правее отсечки, тогда нам стоит сделать порог как можно правее, несмотря на то, что мы потеряем много красных "конфеток". Наша цель - просто уменьшить ошибку первого рода, и потери нас не волнуют.

Однако, в противовес p-value, у нас возникает проблема ошибок второго рода, при которых мы выкидываем "конфетки". У нас хорошие измерения, но мы загоняем все в такие жесткие рамки, что в результате мы потеряем много конфеток.

Таким образом, нам нужно идти на компромисс. Этот компромисс во всей теории распознавания - главный момент принятия решения. Самое главное, от чего мы отталкиваемся - это смысл задачи.

Для этого есть следующая таблица (см. рис. 8.5):

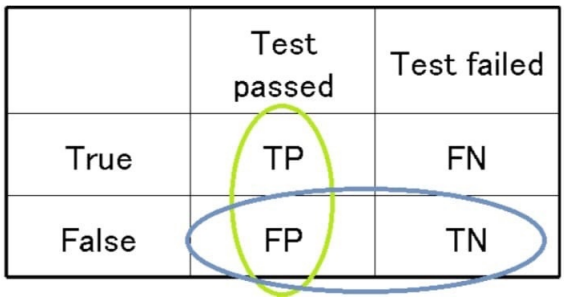

Рис. 8.5 Таблица для принятия решения в задачи распознавания

В таблице на рис. 8.5 есть полезная величина, называемая False Discovery Rate (FDR). На самом деле, это мат. ожидание ложно положительных значений (FP) среди всех значений, которые прошли порог. Все модели, связанные с p-value, касаются только ложных результатов.

$$FDR=E(FP/(FP+TP))$$

Контроль частоты ошибок, метод Бенджамини-Хучберга

FDR - это вероятность принять ложное за истинное при том, что истинное в принципе существует. Для этого есть много разной статистики, но главное, что это порождает способ разделения последовательности измерений.

Начало у нас такое же, как и при Бонферрони. Однако, тогда мы говорили, что вероятность, которую мы хотим достичь - это вероятность того, что у нас нет ни одного ложного значения слева от нашей отсечки. Это и есть FWER.

Отсечка 0,01 означает, что с вероятностью 0,99 слева от отсечки не будет ни одного ложного значения. Теперь же мы говорим, что пусть будет 5% ложных значений, которые, однако, будут отобраны дешево. Это и есть FDR.

Для этого есть метод Бенджамини-Хучберга. Это история о том, на какой p-value из ряда получившихся экспериментов, например, генов, нам поставить. Также это о

том, как нам разделить правых и левых так, чтобы при этом мы не нашли больше ложных значений, чем α ил FDR в том, что мы отобрали как истинное. Например, это не больше 5% шумовых генов или 5% взорванных двигателей.

Это аналог Бонферрони, но для другого вопроса. Бонферрони отвечал на вопрос, как сделать так, чтобы наше единственное отобранное значение не было ложным, а Бенжамини-Хучберг на вопрос, как сделать так, чтобы среди отобранных было не больше определенного процента ложных значений.

Хотя здесь на самом деле в оценке есть независимость, поскольку иначе с распределениями все будет неочевидно. И именно поэтому Вестфолл-Янг дублирует Бонферрони, а при Бенджамини-Хучберге его нет.

Очень редко делает так, что перемешиваниями считают FDR. Обычно делают либо какую-то оценку FDR, либо делают перемешивания.

Стори и Тибширани

Эта история состоит из двух, которые обсуждались ранее: про распределение p-value и про FDR.

У нас есть некий поток наблюдений из разных экспериментов. Мы понимаем, что в нем есть как сигнал, так и шум. Допустим, у нас есть правильная модель p-value, которая адекватна этому шуму. То есть шум в ней все же даст равномерное распределение. Заметим, что нас интересует не шум, а сигнал. Мы хотим понять, куда нам нужно ставить отсечку, чтобы получить в экспериментах, попавших в левую часть, шума не больше, чем определенный процент.

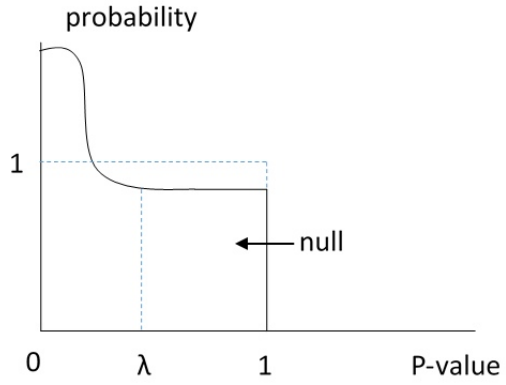

Рис. 8.6 Распределение p-values

Для этого нам нужно понять, как устроено распределение сигнала. Однако, мы помним, что p-value шума распределены равномерно. Таким образом, шум - это плоская часть графика на рис. 8.6, а сигнал - это горб, который торчит над ним.

Когда мы по какому-то параметру отрежем на 8.6 λ, то мы уже знаем, какие соотношение всего к шуму у нас будет, поскольку шум - это кусочек равномерного распределения, который попадает в эту λ.

Таким образом, распределение p-values на рис. 8.6 - это смесь двух распределений - неизвестного нам распределения от реального сигнала, который там есть, а также равномерного распределения от шума. Их очень легко различить на графике даже просто геометрически.

Когда мы разделили распределения, мы знаем, сколько шума у у нас есть в любом отрезке, как бы мы его ни построили. Мы точно знаем процент шума, и поэтому при любом способе выбирания экспериментов из этого распределения мы можем оценивать, какой процент из них пришел из шума.

По хвосту справа мы аппроксимируем общую долю шума в этой смеси. Когда мы знаем эту долю шума, то мы можем в любом участке сигнала мы можем сказать, сколько у нас процентов шума. Все это благодаря аппроксимации шума из правой части (из левой части мы не можем понять, сколько шума).

Пороговое значение

У нас есть какие-то события, которые мы как-то ранжировали по их невероятности. Мы знаем фоновую модель, но опять же, не знаем модель сигнала. Мы пытаемся понять, где нам сделать отсечку так, чтобы разрез соответствовал самому невероятному событию. Самый невероятный разрез - это то, что имеет физический смысл.

$$S_1 \geq S_2 \geq S_3 \geq ... \geq S_N; p(S_i) = P(S < S_i) \text{ для шумовых } S$$

$$p(S_1) \leq p(S_2) \leq p(S_3) \leq ... \leq p(S_N)$$

Мы по очереди пробуем разные вероятности отсечки и каждый раз говорим, что отсечка означает, что у нас был бернуллиевский процесс, в котором с этой вероятностью успеха у нас налево запрыгнуло два или меньше события. При этом испытаний было N. Это обычная бернуллиевская формула вероятности успеха:

$$i = argmin(\sum_{k=1}^{i}(|_k^N)p(S_i)^k(1-p(S_i))^{N-k})$$

Мы можем использовать эту бернуллиевскую формулу для каждой из вероятностей.

Таким образом, мы считаем отсечкой то место, которое никак не могло получиться случайно. Это очень разумная интуиция.

Лекция 9

Марковские и бернуллиевские модели последовательностей обладают одним существенным недостатком - они однородные.

Представим, что у нас есть геном. В нем есть гены, межгенные участки, промоторы и т.д. Статистические свойства последовательностей, находящихся в разных местах, разные. Поэтому однородные модели для моделирования реальных биологических последовательностей практически неприемлемы.

Скрытые марковские модели (HMM)

Рассмотрим следующую модельную ситуацию:

Главным объектом исследования теории вероятностей является монета. Представим себе, что у кого-то есть две монеты: прямая и кривая. Этот кто-то подбрасывает монету много раз, время от времени подменяет прямую монету на кривую, и наоборот. Нам выдаются только наблюдения:

$$00011011011110111011111001011 \tag{23}$$

Мы видим, что монета, скорее всего, была перекошена в сторону единицы.

Цель моделирования может быть как генеративной, так и предсказательной. Скрытая марковская модель может порождать последовательность.

У нас есть некое начало (B, см. рис. 9.1). В какой-то момент этот некто выбирает монету, допустим, прямую и делает бросок (синяя стрелка на рис. 9.1), после чего вновь выбирает прямую монету и делает бросок определенное количество раз (красная стрелка на рис. 9.1). Затем он выбирает кривую монету и также бросает ее определенное количество раз.

Заметим, что могло быть и не так. Он мог изначально выбрать кривую монету, и после ее подбрасываний перейти к прямой. Когда человеку надоест подбрасывать монеты, он перейдет в состояние конца (E на рис. 9.1)

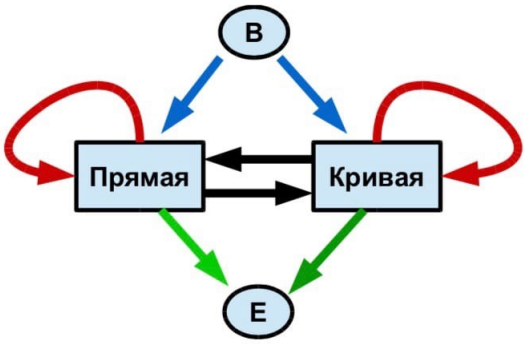

Рис. 9.1 Конечный автомат

Схема на рис. 9.1 называется *конечным автоматом*, который порождает последовательность (23). Она похожа на конечные автоматы, рассмотренные в курсе алгоритмов. Есть довольно простой способ сведения одного автомата к другому.

Для полного определения модели нам необходимо определить свойства монет (вероятности исхода одного броска) и вероятности переходов. В первый момент времени человек равновероятно выбирает любую из двух монет (вероятности - 0,5). С вероятностью 0,8 он будет подбрасывать изначально выбранную монету. С вероятностью 0,1 он переходит на другую монету. Также с вероятностью 0,1 он заканчивает процесс подбрасывания и выдает нам последовательность результатов.

Про сами монеты нам известно следующее:

Прямая монета с вероятностью 0,5 выдает как орел, так и решку. Кривая монета выдает орла с вероятностью 0,2 и решку вероятностью 0,8. Таким образом, дополненная схема скрытой марковской модели будет выглядеть следующим образом:

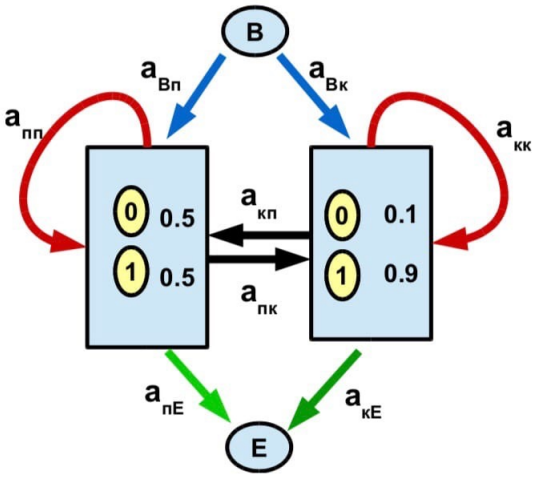

Рис. 9.2 Скрытая марковская модель

Такая модель называется марковской, поскольку следующий исход зависит от того, в каком мы были состоянии. Следующее состояние зависит от того, какое было предыдущее состояние.

Конечный автомат - это частный случай. В первую очередь, у нас есть алфавит (A). В данном случае наш алфавит состоит из двух букв: 0 и 1. Если бы мы говорили про аминокислотные последовательности, то наш алфавит состоял бы из 20 букв, про нуклеотидные последовательности - из 4 букв.

Также мы имеем набор состояний (Q), как и в определении конечного автомата. У нас есть два выделенных состояния: $B \in Q$ и $E \in Q$.

Также мы имеем вероятности перехода из состояния k в состояние l и вероятности порождения буквы α, если мы находимся в состоянии k.

Итого:

$$\{A, Q, B \in Q, E \in Q, a_{kl}, e_k(\alpha)\}$$

В данном случае:

$a_{B\to\text{П}} = 0{,}5$

$a_{\text{П}\to\text{К}} = 0{,}1$

Также у нас есть эмиссионные вероятности - вероятности порождения буквы, если мы находимся в соответствующем состоянии. Например,:

$e_{\text{п}}(1) = 0{,}5$

$e_{\text{к}}(1) = 0{,}8$

Таким образом, скрытой марковской моделью называется совокупность алфавита, конечного множества состояний, двух выделенных состояний (начальное и конечное), матрицы переходных вероятностей и матрицы эмиссионных вероятностей.

Заметим, что в состояниях B и E эмиссионных вероятностей нет, поскольку B и E букв не порождают.

На матрицы a_{kl} и $e_k(\alpha)$ наложены ограничения. Мы должны обязательно куда-то перейти из состояния. Например, из состояния "прямое"мы должны перейти в состояние "прямое". Если мы остаемся на том же месте, это частный случай перехода куда-нибудь. Например, :

$a_{\text{пп}} = 0{,}8$

То, что мы обязательно должны куда-либо перейти, означает, что мы имеем полное пространство событий, поэтому сумма должна быть равна 1:

$$\sum_e a_{kl} = 1$$

В каждом состоянии мы обязательно должны породить букву, поэтому сумма всех вероятностей порождения буквы также равна 1:

$$\sum_\alpha e_k(\alpha) = 1$$

Разметка последовательностей

Как генеративная модель (модель порождения последовательностей) данная модель нам особо не интересна.

Если нам дана данная модель (модель определена, M) и дано наблюдение (последовательность, x), мы хотим определить состояние в каждой позиции i. То есть основываясь на модели и последовательности, мы хотим предсказать, какие участки этой последовательности порождены прямым состоянием, а какие - кривым. Это нам более интересно.

Говоря о биологических примерах, мы можем иметь геном, и, построив, соответствующую модель, предсказывать, где же в нем находятся гены.

Другой пример: у нас есть трансмембранные белки, которых, кстати, очень много. Вообще говоря, у кишечной палочки 35% белков трансмембранные, у людей - около 50%. В трансмембранном белке мы хотим определить, какие сегменты проходят через мембрану, а какие - торчат наружу. Таким образом, мы хотим сделать разметку последовательности на сегменты, обладающие разными свойствами.

Приведенные выше примеры являются вполне реальными и полезными биологическими задачами. Однако, мы вернемся к монете.

Для нашей модели построим последовательность того, как все могло происходить:

Из состояния В мы могли перейти в состояние П и породить 0. Также мы могли из В перейти в состояние К, также породив 0. Это возможные варианты нашей судьбы. Далее мы могли остаться в состоянии П (порождаем 0), а могли перейти в состояние К (порождаем 0). Будучи в состоянии К, мы могли либо остаться в нем же (0), либо перейти в состояние П (0). Мы породили уже две буквы. Мы можем продолжить построение возможных путей порождения последовательности в этом же духе.

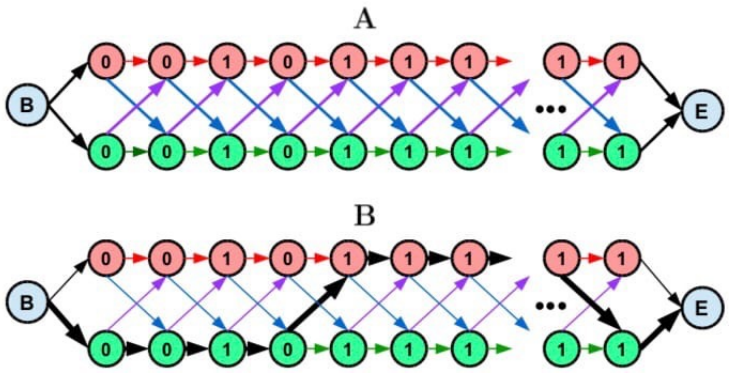

Рис. 9.3 Граф состояний

Любой путь, проходящий по этому графу, будет соответствовать некоторому способу генерации последовательности с помощью нашей модели.

Также на каждой вершине и ребре следует написать число, соответствующее эмиссионной вероятности.

Обратим внимание, что буквы в графе на рис. 9.3 и сверху, и снизу одинаковые, поскольку в каждой конкретной позиции мы порождаем одну и ту же букву. Они разные вдоль графа, поскольку мы не знаем, в каком состоянии мы были, когда мы породили букву. Таким образом, у нас может быть одна и та же буква, но разные состояния. Поэтому сверху и снизу мы должны ставить разные значения.

Алгоритм Витерби

Как на графе состояний выбрать путь, который будет хорошим? Нам хотелось бы пройти путь так, чтобы цифры, которые стоят над состояниями, были бы достаточно большими (такой путь будет иметь наибольшую вероятность). Выберем путь, показанный на рис. 9.3 снизу. Мы можем написать правдоподобие этого пути. Для этого нам нужно просто перемножить все вероятности:

$$L(\text{П}) = a_{\text{ВК}} \cdot e_{\text{к}}(0) \cdot a_{\text{кк}} \cdot e_{\text{к}}(0) \cdot a_{\text{КК}} \cdot e_{\text{к}}(1) \cdot a_{\text{КК}} \cdot e_{\text{к}}(0) \cdot a_{\text{КП}} \cdot e_{\text{П}}(1) \cdot a_{\text{ПП}} \cdot e_{\text{П}}(1) \cdot a_{\text{ПП}} \cdot e_{\text{П}}(1) ... a_{\text{ПЕ}}$$

Давайте найдем такой путь, который имеет максимальное правдоподобие. Это будет, наверняка, самый правдоподобный путь, который точно будет каким-то предсказанием (плохим или хорошим).

Нам дан путь, то есть множество состояний:

$$\text{П} = \{\pi(i)\}$$

π_i - это то состояние, в котором мы были в позиции i. Например, в нашем случае $\pi_3 = \text{К}$(кривое), а $\pi_5 = \text{П}$(прямое). Набор таких состояний - это путь.

Заметим, что $\pi_0 = \text{В}$ и $\pi_{L+1} = \text{Е}$

Если нам задан путь, так мы можем написать правдоподобие от пути как следующее произведение:

$$L(\text{П}) = \prod_i a_{\pi(i-1) \to \pi(i)} e_{\pi_i}(X_i) \tag{24}$$

Такое правдоподобие можно посчитать для любого пути. Это правдоподобие можно максимизировать.

Введем переменную S, которая соответствует состоянию, а также $v_k(i)$ (k - номер состояния). $v_k(i)$ - это наилучшее правдоподобие пути $B \to \pi_i$. Таким образом, например, в состояние S_2 мы можем прийти разными путями. Мы можем по формуле (24) посчитать правдоподобие каждого из путей, который из начала приводит в состояние S_2. Какой-то из этих путей - самый лучший, и он имеет свое правдоподобие.

Чему равно правдоподобие для самого начала?

$v_0(B) = 1$, поскольку мы не производим умножения, поскольку у нас нет никакого перехода.

Допустим на все наилучшие правдоподобия определены. Как найти наилучшее правдоподобие для S_k? Как мы можем попасть в это состояние?

Мы можем попасть в это состояние из всех других состояний (см. рис. 9.5):

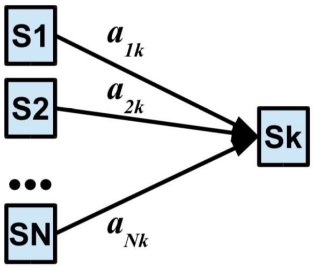

Рис. 9.4 Поиск максимального правдоподобия

Мы можем взять наилучшее правдоподобие, которое привело нас в S_2, умножить на переходную вероятность и на эмиссию, и среди всех возможных переходов выбрать максимум. То есть мы выберем следующее:

$$v_k(i) = \max_l v_l(i-1) \cdot a_l k \cdot e_k(x_i)$$

Эта рекурсия называется алгоритмом Витерби. Он относится к динамическому программированию. Однако, оно отличается от динамического программирования, рассмотренного в курсе алгоритмов. Это частный случай динамического программирования, в котором полукольцо, над которым мы его совершаем, определено на неотрицательных действительных числах. Операция сложения - есть операция максимума. Операция умножения - это и есть операция умножения.

Тем не менее, здесь мы не получаем путь. Мы нигде не видим оптимального пути. В результате этой процедуры, когда мы дойдем до конца, мы узнаем только правдоподобие, но не путь:

$$v_{\mathrm{E}} = L^*(x)$$

Мы можем написать следующее:

$$\pi_k(i) = argmax\ v_l(i-1) \cdot a_l k \cdot e_k(x_i)$$

Таким образом, пройдя вперед, мы запомним, откуда мы прошли, и, в конце концов, когда у нас будут разбросаны "хлебные крошки мы из состояния E сможем по этому пути вернуться в состояние B.

Таким образом, алгоритм Витерби находит одно наилучшее оптимальное декодирование скрытой марковской модели.

Данная модель называется скрытой, поскольку нам дана последовательность, но нам не сказано, где у нас какие состояния были. Модель марковская, поскольку мы переходим из состояния в состояние.

Наша задача называется задачей *декодирования* последовательности с помощью заданной заранее модели.

Однако, это не единственный алгоритм. Этот алгоритм однозначно говорит нам, какой путь является оптимальным. Однако, мы не знаем, насколько достоверна эта расшифровка. Правдоподобие нам ни о чем не говорит.

С помощью этого алгоритма мы не можем понять, какие из путей также являются хорошими и не сильно уступающими оптимальному.

Апостериорное декодирование. Алгоритм вперед-назад (FB)

Мы можем поставить себе другую задачу. Пусть нам дано все то же самое: модель M и некая последовательность x. Мы хотим найти вероятность того, что в позиции i мы прошли через состояние k. Таким образом, мы ищем условную вероятность $P(\pi_i = K|x)$.

Мы можем переписать условную вероятность следующим образом:

$$P(\pi_i = K|x) = \frac{P(\pi_i = K, x)}{P(x)}$$

Таким образом, мы получаем стандартную формулу Байеса (полная вероятность, деленная на маргинальную вероятность).

Итак, у нас в некоторой позиции i есть множество состояний. Мы возьмем состояние номер k. Каждый из путей, которыми мы можем прийти в это состояние, имеет свое правдоподобие. Из состояния k мы можем разными путями уйти в конец.

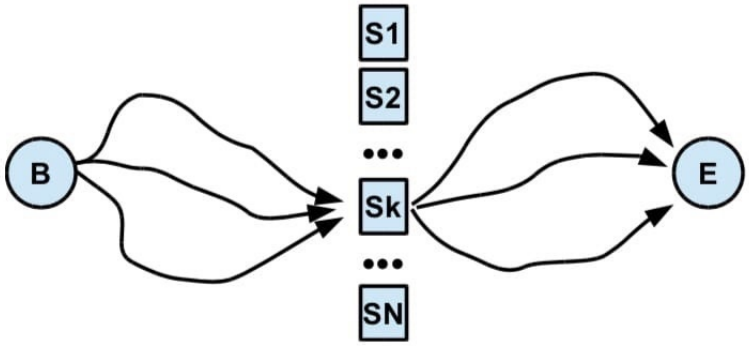

Рис. 9.5 Поиск вероятности того, что в позиции i мы прошли через состояние k

Если мы просуммируем все эти вероятности, то мы узнаем то, что нам нужно:

$$P(\pi_i = k, x) = P(x_1, x_2, ..., x_i, \pi_i = k) = \text{сумма по всем путям } L(B \to K)$$

Далее нам нужно из k уйти в конец:

$$P(x_{i+1}...x_e|\pi_i = k) = \text{сумма по всем путям } L(S_k(i) \to E)$$

Таким образом:

$$P(\pi_i = K|x) = \frac{P(\pi_i = K, x)}{P(x)} = \frac{P(X_1...X_i, \pi_i = k) \cdot P(x_{i+1}...x_e|\pi_i = k)}{P(x)}$$

Введем соответствующие обозначения:
$P(X_1...X_i, \pi_i = k) = f_k(i)$
$P(x_{i+1}...x_e|\pi_i = k) = b_k(i)$

$$f_k(i) = P(x_1...x_i, \pi_i = k) \to \sum \text{по всем путям } B \to S_k(i)$$

Путей из начала в начало (из B в B) всего один, и при этом мы никуда не двигаемся. Поэтому сумма по всем путям правдоподобий из B в B равняется 1:

$$f_b(0) = 1$$

У нас есть позиция i, в которой возможные состояния - $S_1...S_k...S_n$. Пусть мы находимся в позиции i и хотим понять, чему равняется сумма по всем путям, которые приведут нас в S_k.

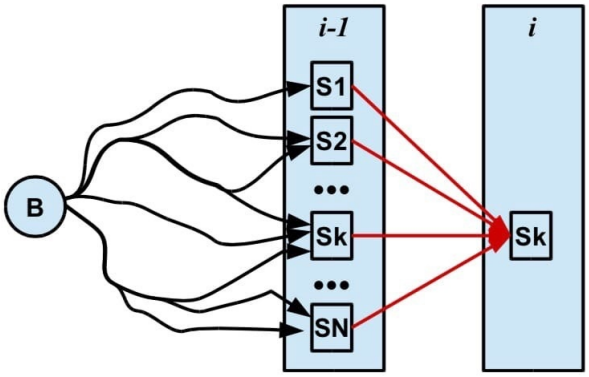

Рис. 9.6 Сумма по всем путям, от B до $S_k(i)$

Мы можем написать следующее:

$$f_k(i) = \sum_l f_l(i-1)a_{lk}e_k(x_i)$$

Таким образом, мы довольно легко вычисляем сумму по всем путям рекурсивно, если у нас есть начало и $f_b(0) = 1$.

Сумма по всем путям из B в S_1 (в данном случае у нас просто переход) - это переход, умноженный на эмиссию. Таким образом, мы решаем примерно такую же задачу, как тогда, когда мы подсчитывали число путей в графе.

В конце концов, идя слева-направо, мы получаем f_{E}. Это сумма по всем путям из начала в конец. Это есть просто вероятность последовательности $P(x)$.

$$b_k(i) = (x_{i+1}...x_L|\pi_i = k)$$

Иначе, $b_k(i)$ - это вероятность того, что была порождена последовательность, начиная со следующей буквы и до самого конца при условии, что мы стартовали с позиции k.

Таким образом, если мы находимся в позиции i, и у нас есть состояния $S_1...S_k...S_N$, также у нас есть позиция $i+1$ и E, из которого мы можем прийти разными путями в разные состояния в позиции $i+1$.

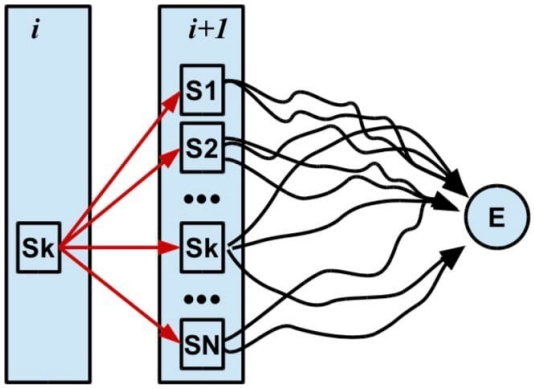

Рис. 9.7 Сумма по всем путям, от $S_k(i)$ до E

Чему равна сумма по всем путям $b_E(L+1)$? Нетрудно догадаться, что тут все то же самое, что и с B. Из E в E у нас ведет единственный путь, то есть $b_E(L+1) = 1$.

Рассмотрим некую позицию i. Из состояния k мы можем перейти во все другие состояния в позиции $i+1$. Поэтому:

$$b_k(i) = \sum_l a_{kl} \cdot e_l(x_{i+1}) b_l(i+1)$$

Если в рекурсии мы шли слева направо, то есть позиция i зависела от $i-1$, то здесь позиция i зависит от $i+1$. Поэтому рекурсия идет справа-налево. Соответственно,

$$b_B(0) = P(x)$$

Таким образом, с одной стороны, $P(x) = b_B(0)$.

С другой стороны, $P(x) = f_E(L+1)$.

Рекурсия направо и налево должны привести к одному и тому же значению.

Таким образом, мы можем посчитать в каждой позиции вероятность состояния, то есть

$$P_k(i) = P(\pi_i = K|X) = \frac{f_k(i) \cdot b_k(i)}{f_E(L+1)}$$

Такое декодирование гораздо более поучительное, чем алгоритм Витерби.

Итеративное предсказание

То, о чем будет сказано ниже, не имеет никакого математического обоснования. Однако, это вполне применимо.

$e_k(x_i)$ - это эмиссионная вероятность будучи в состоянии k породить символ, который у нас находится в позиции i.

Давайте в Витерби сделаем следующую замену:

$$e_k(x_i) \to P_k(i)$$

Эта замена означает, что если у нас было какое-то выдающееся состояние, обладающее большой вероятностью, то мы этой букве в соответствующей позиции дадим большую вероятность. Таким образом, вместо эмиссионной вероятности мы получаем позиционно зависимую.

Таким образом, вероятность эмиссии букв у нас зависит от их позиции.

В позиции i у нас только буква x_i. Остальных букв у нас в этой позиции нет. Сделав такую замену, про остальные буквы и эмиссионные вероятности мы можем забыть в этой позиции. Заметим, что в других позициях будут другие замены.

Такая штука позволяет делать хорошие предсказания, хоть и не имеет математического обоснования. Сделав замену, мы снова можем посчитать forward и backword, опираясь не на эмиссионные вероятности, а на замененные. Мы получим обновленное значение вероятности. Мы можем сделать такие итерации несколько раз. Подход такого типа называется expectation maximisation (EM, максимизация ожидания).

Модель сайта рестрикции CAGT

В качестве биологической задачи выберем по-прежнему простую задачу, которая может решаться гораздо более простыми способами, без HMM. Она звучит так:

У нас есть сайт рестрикции CATG. Наша задача состоит в том, чтобы построить скрытую марковскую модель для поиска таких сайтов рестрикции. Как стоится скрытая марковская модель?

У нас есть состояние начала (B, background), и мы друг за другом с вероятностью 0,25 порождаем разные буквы.

Таким образом, у нас есть буквы, потом сайт рестрикции, потом опять буквы.

Как мы опишем сайт? Сколько у него будет состояний? У него будет 4 состояния: C (эмиссионные вероятности - A - 0, C - 1, G - 0, T - 0). Если мы вошли в первое состояние (состояние первой позиции сайта), то следующая (вторая) позиция будет обязательно (вероятность равна 1) позиция A (1 0 0 0), далее обязательно идет третья позиция сайта - с вероятностью 1 мы переходим в состояние T (0 0 0 1). Далее мы обязательно переходим в четвертую позицию сайта (переходная вероятность строго равна 1) G (0 0 1 0). После того, как сайт закончился, мы обязательно должны выйти обратно в background. Затем мы должны выйти обратно в конец (E). Наглядно эту модель см. на рис. 9.8:

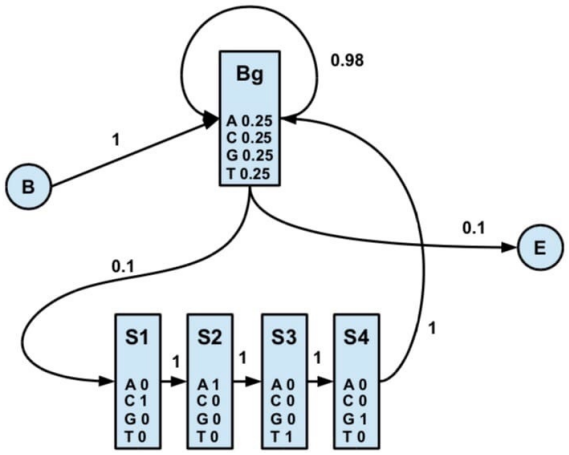

Рис. 9.8 Первый вариант модели поиска сайта CAGT

Таким образом, мы смоделировали сайт рестрикции CATG внутри некоторой последовательности.

Эта модель имеет следующие ограничения:

Эта модель не допускает наличия сайта в самом начале и в самом конце последовательности, поскольку из begin мы сразу входим в состояние background. Мы не входим в состояние сайта. По окончании сайта, мы обязательно входим в состояние background. Мы не можем сразу выйти в состояние end.

Также эта модель не позволяет ставить два сайта подряд. Например, ситуация CATGCATG не может быть порождена этой моделью при условии, что обе части являются сайтами рестрикции. Эта модель не видит двух сайтов, слепленных друг с другом.

Данная модель достаточно интересна и обладает замечательным свойством: она допускает существование сколь угодно большого количества сайтов в последовательности. Действительно, мы сначала идем по background, затем по сайту рестрикции,

затем мы обратно входим в background, из которого мы снова можем войти в сайт рестрикции.

Однако, эту модель надо исправить:

У нас есть сайт, состоящий из четырех состояний. Заметим, что состояний довольно много по сравнению с монетой.

Также у нас есть background, который устроен так, что мы можем оставаться в нем сколь угодно долго. Из него мы можем перейти сайт. Чтобы разрешить быть сайтам в самом начале, мы должны допустить переход из begin сразу в сайт.

Если мы хотим допустить существование двух подряд идущих сайтов, мы должны поставить переход из конца сайта в начало.

Чтобы допустить наличие сайта в конце последовательности, мы должны поставить переход из конца сайта сразу в end.

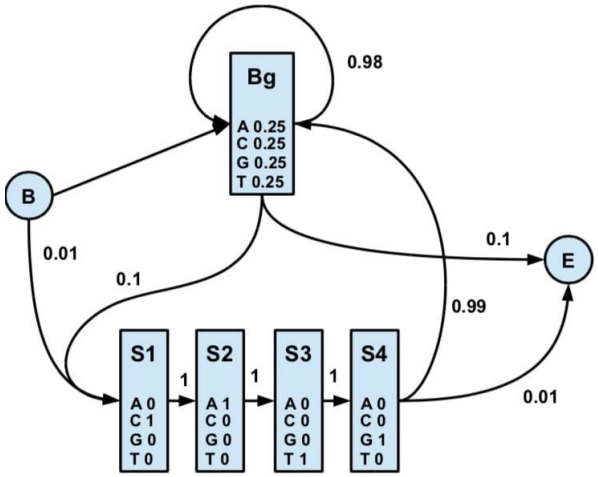

Рис. 9.9 Второй вариант модели поиска сайта CAGT

При взгляде на модель на рис. 9.9 становится понятно, что у нас есть фоновые состояния, состояния сайта с переходами, имеющими вероятность 1, а также другие вероятности.

Если вероятность перехода из begin в первую позицию сайта будет слишком большая, то при генерации последовательности у нас сайт часто будет появляться в начале последовательности. Обычно у нас вероятность такого перехода равна вероятности перехода из background в первую позицию сайта. Таким образом, не имеет значения, где начинается сайт.

Появление сайта после background и после самого сайта имеют одинаковую вероятность. Они также равны вероятности появления сайта в самом начале.

Вероятность выхода из background равна вероятности выхода из сайта:

$$a_{s \to E} = a_{Bg \to E}$$

Таким образом, мы можем поставить некоторое количество разумных ограничений, и из условия, что сумма исходящих вероятностей равна 1, мы можем определить все вероятности переходов.

Нарисуем граф состояний:

Сайт CATG может появиться и по случайным причинам, и как значимое событие. Какой путь выбрать? При каких условиях будет выполняться правильный проход через CATG?

Если мы в модели зададим очень маленькую вероятность (допустим, 10^-6) перехода из background в сайт, то будет ли сайт когда-нибудь узнан? Или наша модель всегда будет пропускать этот сайт и не будет его распознавать и будет считать, что catg появился по случайным причинам и не является сайтом?

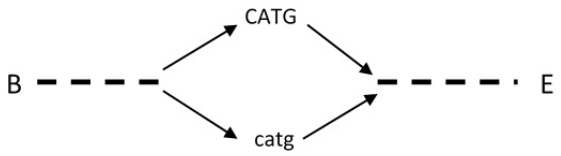

Рис. 9.10 Пути появления сайта рестрикции в последовательности

Сделаем следующую оценку:

$$L(B \to Bg \to SITE \to E) = L(B \to Bg) \cdot a_{Bg \to S} \cdot a_{S \to Bg} \cdot L(Bg \to E)$$

Такое правдоподобие будет иметь первый возможный путь, при котором появление сайта распознается и считается неслучайным событием (верхний путь на рис. 9.10).

$$L(B \to Bg \to Bg \to Bg \to E) = L(B \to Bg) \cdot a_{Bg \to Bg} \cdot e_{Bg}^4 \cdot a_{Bg \to Bg} \cdot L(Bg \to E)$$

Такое правдоподобие будет иметь второй возможный путь, при котором появление сайта не распознается и считается случайным событием, то есть частью background (нижний путь на рис. 9.10).

Мы можем получить отношение правдоподобия, поделив первый лайклехуд на второй:

$$\frac{L(SITE)}{L(Bg)} = \frac{a_{Bg \to S}}{a_{Bg \to Bg} \cdot e_{Bg}^4 a_{Bg \to Bg}}$$

Если мы, глядя на это отношение, видим, что вероятность в числителе меньше, чем в знаменателе, то мы никогда не найдем сайт. Если же знаменатель больше, чем числитель, то мы всегда будем находить сайт.

Таким образом, мы видим смысл переходных вероятностей. Они вносят некоторые дополнительные ограничения.

У сайтов рестрикции бывает звездная активность. Например, самая первая и самая известная рестриктаза, с которой работали биоинженеры, это рестриктаза Ecor1. Она узнает шестерку. Также она может узнавать четверку, хоть и слабее. Поэтому, вообще говоря, сайт распознавания Ecor1 является не совсем шестеркой.

По правде говоря, все сайты в некоторых условиях имеют определенные "предпочтения которые зависят от контекста и многого другого.

Лекция 10

Сайты связывания транскрипционных факторов

У нас есть вероятность появления букв в сайтах связывания в первой позиции (0.8, 0.1, 0, 0), а также во второй позиции (0.05, 0.05, 0.1, 0.8) и в третьей.

Таким образом, в первой позиции самая предпочитаемая буква - A, а также допустима буква C (другие буквы не могут появиться в той позиции). Во второй позиции самая предпочитаемая буква - T, буквы A и C очень редки, буква G встречается немного чаще.

Мы получили описание сайта связывания факторов транскрипции.

Наша модель состоит из background и сайта. Она выглядит, как и в случае с моделью сайта рестрикции (см. Лекцию 9).

Повторим то же самое упражнение, что и в случае с сайтом рестрикции, то есть напишем отношение правдоподобия:

$$\frac{L(SITE)}{L(Bg)} = \frac{a_{Bg\to S'}\prod e_k(x_i)\cdot 1}{a_{Bg\to Bg}\prod e_{Bg}(x_i)a_{BgBg}}$$

Заметим, что у нас сократились переходы из begin и переходы в end.

Теперь мы возьмем логарифм отношения правдоподобия:

$$\log L = \log\frac{a_{Bg\to S}}{a^2_{Bg\to Bg}} + \sum \log\frac{e_k(x_i)}{e_{Bg}(x_i)}$$

$$\log\frac{a_{Bg\to S}}{a^2_{Bg\to Bg}} = -A$$

$$\log L = -A + \sum \log\frac{e_k(x_i)}{e_{Bg}(x_i)}$$

Если мы в качестве x_i возьмем все возможные буквы, то мы получим *позиционную весовую матрицу* (PWM):

$$M_k(\alpha) = \log\frac{e_k(\alpha)}{e_{Bg}(\alpha)}$$

Когда мы подставляем в матрицу соответствующие буквы, то получаем значение PWM на сайте. $-A$ же является порогом.

Мы можем добавить следующее требование:

$$\log L = -A + \sum \log \frac{e_k(x_i)}{e_{Bg}(x_i)} > 0$$

По сути, это то же самое, что:

$$\sum M_k x_i > A$$

Таким образом, данная скрытая марковская модель, с одной стороны, приводит нас к позиционной весовой матрице.

С другой стороны, она дает нам некое представление о природе порога. Порог - это вероятность перехода из background в сайт, деленная на квадрат вероятности перехода из background в background.

Трансмембранные сегменты в белках

Как уже говорилось ранее, мембранных белков очень много. При этом мы знаем про них довольно мало. Нам бы хотелось знать, где находятся глобулярные участки, которые могут содержать какие-либо рецепторы, а также где находятся каналы, трансмембранные сегменты.

Поиск трансмембранных сегментов очень полезен в практическом плане.

Представим себе простейшую модель:

У нас есть начало (B), глобулярная (Globul) и трансмембранная (TM) часть. Они могут переходить друг в друга, а также оставаться в том же состоянии. Также возможен выход в конец (E). Эта схема (см. рис. 10.1) очень похожа на модель с монетой.

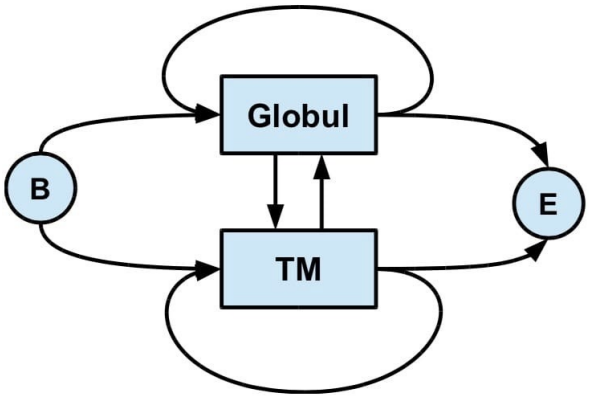

Рис. 10.1 Простейшая модель трансмембранных сегментов

В globul у нас эмиссионные вероятности, характерные для глобулярных белков, которые содержат гидрофильные (иногда гидрофобные) аминокислотные остатки.

Трансмембранные части обязательно содержат гидрофобные аминокислотные остатки. Более того, предпочтительными являются алифатические остатки, поскольку они имеют большее сродство к липидам.

Давайте посмотрим на эту модель не как на распознавательную, а (для проверки того, насколько эта модель хороша) как на генеративную, то есть на модель, которая порождает трансмембранные белки.

Мы вошли в ТМ сегмент, и с некоторой вероятностью мы в нем останемся (a_{TT}), а с некоторой - уйдем из него (a_{TG}). Какова вероятность породить трансмембранный сегмент длиной в 5 букв?

$$P(5) = a_T^5 \cdot a_{TG}$$

Выведем общую формулу:

$$P(L) = a_T^L \cdot a_{TG}$$

Это геометрическое распределение. Эта вероятность ведет себя следующим образом:

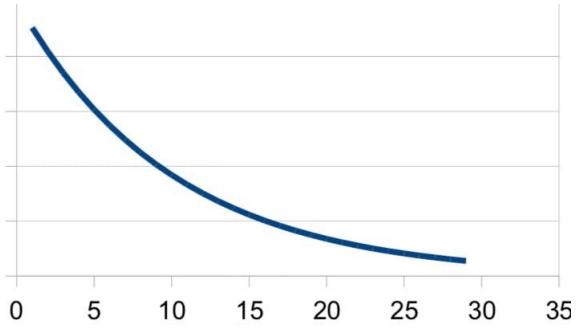

Рис. 10.2 Распределение длин трансмембранных сегментов при генерации с помощью модели на рис. 10.1

Таким образом, модель на рис. 10.1 будет порождать белки с убывающей вероятностью длины ТМ сегмента.

Трансмембранный сегмент, как правило, представляет из себя α-спираль, которая пронизывает мембрану, имеющую некую толщину.

Понятно, что трансмембранный сегмент длиной в 2 аминокислотных остатка невозможен. То есть модель, построенная из разумных соображений, не будет работать правильно. На самом деле картина распределения длин будет выглядеть примерно так:

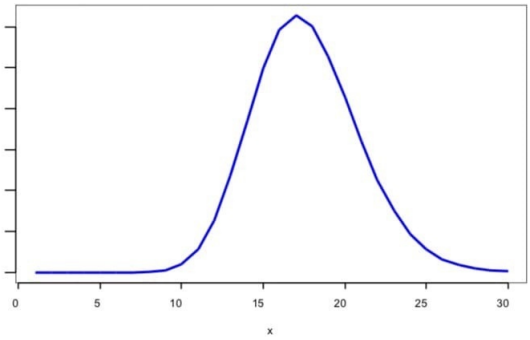

Рис. 10.3 Реально наблюдаемое распределение длин трансмембранных сегментов

Усложненная модель трансмембранных сегментов

Модель на рис. 10.1 не отвечает реальности, хотя, казалось бы, придумана она правильно. Как нам ее исправить?

Если бы длина трансмембранного сегмента распределена как на рис. 10.3, то мы знаем, как правильно нужно устроить модель.

Если длина трансмембранного сегмента равна 20 аминокислотных остатков, то мы имеем следующую картину:

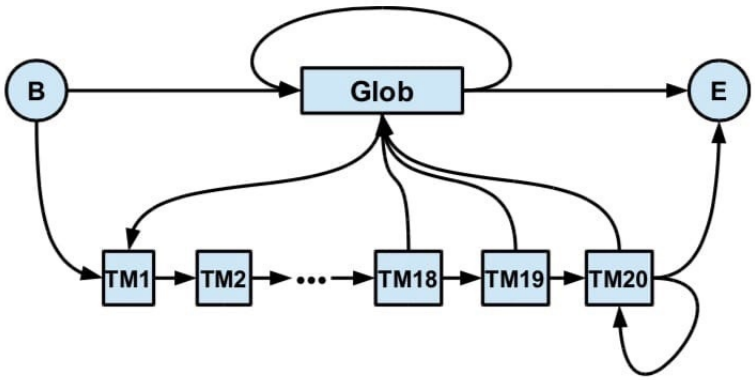

Рис. 10.4 Усложненная модель трансмембранных сегментов

Модель на рис. 10.4 порождает TM сегменты длиной от 1 до 20 остатков. Модель на рис. 10.1 порождает TM сегменты, которые распределены по убывающей экспоненте (рис. 10.2).

Модель на рис. 10.4 намного сложнее, чем на рис. 10.1, поскольку в ней гораздо больше состояний. Однако, такой подход позволяет нам сгенерировать модель с любым распределением длин.

В этой модели состояний, помимо стартового и конечного состояний, есть 20 других состояний только в трансмембранной в части.

Глобулярная часть белка тоже имеет свое распределение длин, о чем мы не должны забывать.

Ограничений на длину в трансмембранных частях нет.

Модель распознавания генов

Мы построим модель распознавания генов у прокариот, поскольку у эукариот мы должны учитывать возможность интронов, за счет чего модель будет слишком сложной и непонятной.

В модели распознавания генов сначала идет background, не являющийся геном. Ген начинается со старт-кодона, в котором 3 состояния. Мы описываем это как простой сайт.

Далее следует *молчащее состояние*, которое не генерирует никаких символов. Оно является "пересадочным пунктом". Его эмиссионные вероятности равны 1.

Далее в модели следует кодон, который устроен так же, как и сайт рестрикции и сайт связывания (три подряд идущие буквы). Он также содержит 3 состояния. И эмиссионные, и переходные вероятности здесь равны 1.

Итак, у нас есть 61 кодон и 183 состояния, что есть много.

Далее мы снова переходим в молчащее состояние. После прочтения первого кодона мы должны либо прочитать второй кодон, либо прочитать стоп-кодон. Далее снова следует молчащее состояние.

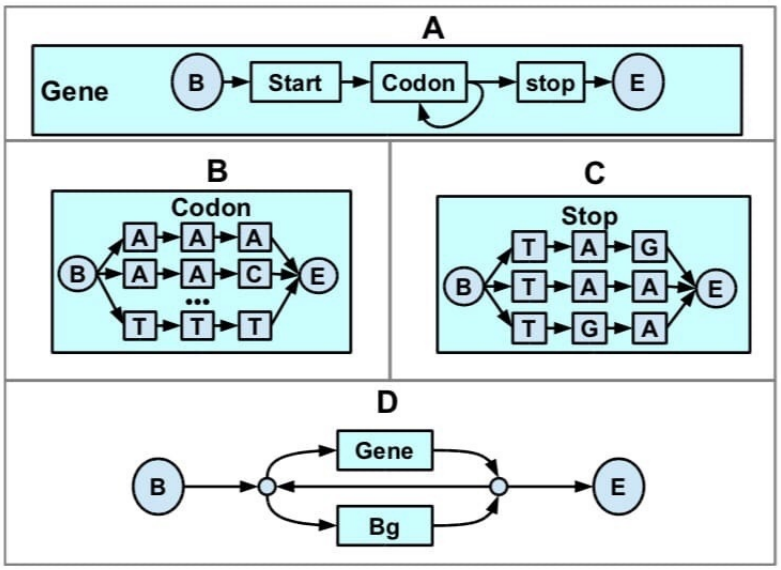

Рис. 10.5 Модель гена в геноме (A - композитная модель гена; B - модель кодона; C - модель стоп-кодона; D - композитная модель генома с генами)

В этой модели не хватает распределения длин белков. Если мы будем применять модель на рис. 10.5, то мы увидим следующее распределение длин белков такое же, как на рис. 10.2. Настоящее же распределение длин белков мы видим на рис. 10.3.

Мы должны несколько раз повторить блок с кодоном. Таким образом, мы получим очень много состояний.

Переходные вероятности у нас не равны друг другу, поскольку у нас есть некие предпочтения:

Во-первых, не все аминокислотные остатки встречаются с одинаковой частотой. Также пределах группы кодонов, которая кодирует один и тот же аминокислотный остаток, частоты кодонов разные. Поэтому мы используем так называемую *частоту использования кодонов*.

HMM-лего

Молчащие состояния позволяют нам собирать модели как лего, в связи с чем мы получаем простейшую модель сайта, background. Через молчащие состояния мы можем соединять эти части друг с другом. Таким образом, они являются некими коннекторами между подмоделями в больших моделях.

Например, у нас есть модель сайта Шайна-Дальгарно (SD), которая содержит в себе некие буквы с какими-то частотами.

Также у нас есть спейсеры, которые также являются некими моделями. Они внутри себя имеют небольшие предпочтения букв, хоть и очень слабые.

Модель старт-кодона тоже имеет определенное строение. Она включает следующие варианты кодонов (состояний): ATG, GTG, CTG. Переходные вероятности, соответствующие этим кодонам, различны, поскольку встречаемость этих кодонов разная (ATG встречается чаще, а CTG всречается гораздо реже).

Модель стоп-кодона мы видим на рис. 10.4 (C). Стоп-кодон TGA встречается чаще, чем все остальные. Эта модель довольно важна, поскольку она помогает нам отличать правильные белки, которые кончаются правильным стоп-кодоном, от неправильных.

На самом деле, в прокариотах довольно часто встречаются тандемные стоп-кодоны, поскольку стоп-кодоны очень часто прочитываются и проскакиваются. Именно поэтому в важных белках стоп-кодоны обычно идут подряд друг за другом.

Также у нас есть перекрывающиеся старт-стоп кодоны, например, TGATG. Возможна также ситуация, в которой старт-кодон начинается до стоп-кодона. В модель распознавания генов необходимо включать и такие кодоны. В этом случае у нас не происходит полного освобождения рибосомы, а происходит реинициация (Шайн-Дальгарно для этих случаев не нужен).

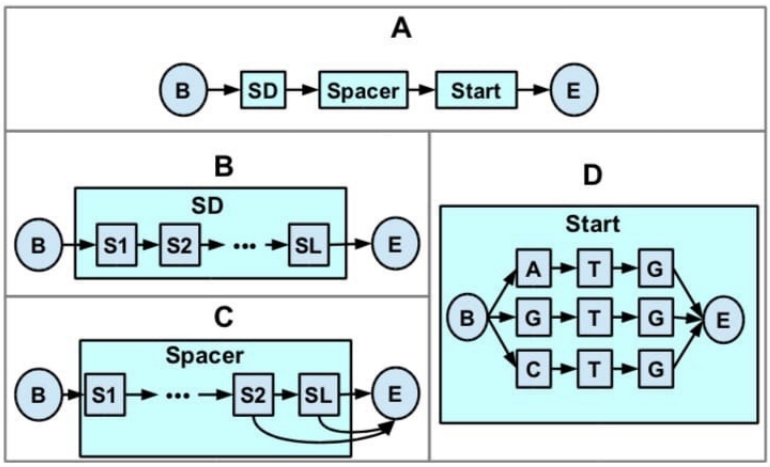

Рис. 10.6 Модель инициации трансляции (A - композитная модель; B - модель мотива Шайна-Дальгарно; C - модель спейсера; D - модель старт-кодона)

Проблема сайта Шайна-Дальгарно сама по себе довольно сложная. Он хорошо выражен у бацилл, *e. Coli* и у многих грамотрицательных бактерий. Однако, есть довольно много бактерий, у которых вообще нет сайта Шайна-Дальгарно. Поэтому никто не знает, как у них происходит инициация трансляции.

При изучении биологии нужно помнить, что многие вещи построены на модельных организмах по типу кишечной палочки. Это позволяет нам понять как общебиологические вещи, так и вещи, характерные только для данного организма (вида, типа, класса).

Например, Шайна-Дальгарно совсем нет у цианобактерий. У них не найдено никакого сайта, отличающего настоящий старт-кодон от ненастоящего.

У микобактерий, которые являются возбудителями туберкулеза, проказы и других болезней, также нет сайта Шайна-Дальгарно.

ФАКУЛЬТЕТ
БИОИНЖЕНЕРИИ И
БИОИНФОРМАТИКИ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ