Глава 1

Мама, папа, знакомьтесь, это STAN

Варкалось. Хливкие шорьки
Пырялись по наве,
И хрюкотали зелюки,
Как мюмзики в мове.

О, бойся Бармаглота, сын!
Он так свирлеп и дик!
А в глу́ши ры́мит исполин -
Злопасный Брандашмыг...

Льюис Кэрролл

1.1 Знакомство

Мы уже несколько раз говорили о том, что байесовский подход начал интенсивно развиваться в течение последних 30 лет. Прежде всего, это связано с развитием компьютеров, которые позволили таскать за собой разные сложные распределения. В предыдущей главе мы с вами на простых примерах убедились, что расчёты в байесовском мире делать не очень просто. Если взять в качестве априорного распределения что-то сложное, можно нарваться на проблемы.

Из-за этого древние байесовцы поставили перед собой вполне разумную цель: брать в качестве априорных распределений такие, чтобы после домножения на правдоподобие апостериорное распределение принадлежало тому же классу распределений, но с другими параметрами. Эти новые параметры зависели бы от параметров априорного распределения, и наши алгоритмы бы просто осуществляли пересчёт априорных параметров в апостериорные по готовым формулам[1].

Это очень сильно связывало руки. Прошло время, и миру явился избранный[2]. Алгоритмы Монте-Карло по схеме Марковской цепи (MCMC) позволили разорвать оковы. Анализ моделей с априорными распределениями на любой вкус стал явью.

Для генерации апостериорного распределения MCMC используются марковские цепи. По идее, при таком подходе от исследователя требуется сформулировать модель, априорное мнение о параметрах модели и самостоятельно сконструировать цепь, сходящуюся к апостериорному распределению параметров. Однако оказалось, что конструирование цепи в большинстве ситуаций можно доверить компьютеру. Это ещё сильнее упростило жизнь.

В этой главе мы не будем разбираться в том, что именно происходит внутри чёрного ящика MCMC. Этому будут посвящены более поздние главы. Здесь мы на нескольких примерах посмотрим на то, как этот чёрный ящик можно применять в связке с R.

Использовать для этого мы будем отличный, свободный, бесплатный, уютный вероятностный язык программирования STAN. STAN — это не пакет R, а отдельный язык описания байесовских моделей. Исследователь описывает модель и априорное распределение параметров, а STAN генерирует код C++, в котором запрограммирована требуемая марковская цепь.

Нам для работы потребуется установить пакет rstan, который содержит и свежую версию программы STAN. В силу того, что здесь взаимодействует много программ (R, STAN, компилятор C++), установку следует выполнять аккуратно, строго по инструкции. Как только инструкция будет выполнена, а STAN установлен, мы сможем сварить что-нибудь эдакое с его помощью. Остановитесь. Установите STAN на свой компьютер и только после этого продолжите чтение.

[1] На Википедии можно найти огромную таблицу с сопряжёнными распределениями

[2] Слышишь гул мотора? Навуходоносор влетает в Зион.

Заждались? Книжка по статистике — не книжка по статистике, если в ней нет задачки про подкидывание монетки. Кстати, все коды, которые используются в этой и последующих главах, можно найти в репозитории на github.

1.2 Задача про монетку

Бесплатный совет: перед тем как с головой погрузится в пучину байесовского моделирования, тщательно продумайте то, как именно устроены данные. Каждый раз, когда вы сталкиваетесь с какой-то задачкой, попробуйте мысленно задать себе вопрос: «А как бы я сгенерировался на месте этого датасета?».

В случае с монеткой всё довольно просто. Например, если бы один из авторов был бы монеткой, он бы выпадал орлом либо решкой с вероятностью p. Скорее всего, p отличалось бы от 0.5. Автор хотел бы быть хитрой монеткой. На математическом языке задачка про монетку выглядит следующим образом:

Вероятность выпадения решки: p

Априорно: $p \sim U[0; 1]$

Данные: результаты прошлых бросков $y_1, \dots, y_n$

Апостериорно: $p \mid y \sim ?$

Если очень хочется, то можно представлять себе, что это задачка про бабушку и карасей из предыдущей главы. Смысл от этого не поменяется. Для того, чтобы замоделировать ситуацию, создадим два файла.

Для создания первого файла подойдёт любой текстовый редактор вроде блокнота. Создадим пустой текстовый документ. В нём будет содержаться описание нашей модели на вероятностном языке STAN. Этот файл нужно сохранить с расширением **.stan**. Например, **coin.stan**. Для удобства этот файл можно создать в RStudio. Главное — проконтролировать расширение.

Второй файл будет создан в RStudio. В нём мы будем писать на R команды для компиляции программы, передачи ей данных и анализа полученных результатов. В становском файле содержится четыре блока:

- Блок **data** описывает какие внешние данные нам нужно передать STAN. В нашем случае у нас есть N — количество наблюдений (целое число) и y[N] — вектор длины N с целочисленными наблюдениями.

- Блок **parameters** описывает какие у нас в модели есть параметры. Параметры могут быть ограничены или неограничены. У нас всего один параметр p — вероятность выпадения орла. Она лежит между нулём и единицей.

- В блоке **model** нужно задать описание модели. Под описанием модели подразумевается распределение параметров и то как они взаимосвязаны между собой и наблюдениями. В нашем случае вероятность имеет равномерное распределение на отрезке $[0; 1]$, а каждое наблюдение порождается из распределения Бернулли с этой вероятностью. Чтобы показать как порождается каждое наблюдение, придётся написать цикл. Обратите внимание, что внутри этого цикла часть наблюдений можно породить по одному закону, часть по другому.

- Блок **generated quantities** описывает правила генерации апостериорных величин.

```
// Тут лежат данные
data{
    int N;      // число наблюдений (целочисленный формат)
    int y[N];  // отдельные наблюдения (вектор длины N)
}
// Тут лежат параметры
parameters{
    real<lower=0, upper=1> p;   // доля золотых монет в шляпе
}
model{
    // априорно:
    p ~ uniform(0, 1);
    // модель: как наблюдения связаны с параметром
    for (n in 1:N) {
       y[n] ~ bernoulli(p);
```

```
    }
}
generated quantities {
    int y_new;       // какая монетка следующая?
    // тильда не означает генерации случайных чисел, это лишь
     ↪ наше мнение
    y_new = bernoulli_rng(p);     // генерация будущего игрека
}
// Чтобы было коректно последняя строка традиционно пустая
```

Это всё. Модель описывается на максимально человеческом языке. На самом деле есть и другие разные блоки, но мы пока что ограничимся этими. Справа сверху в R-Studio можно найти кнопочку **Check**. Если тыкнуть на неё, RStudio проверит корректность синтаксиса. Если ошибок не найдено, можно переходить к следующему этапу. Второй файл будем писать уже на языке R.

```
library(rstan)       # Вызываем дух преподобного Томаса!
library(bayesplot)   # За компанию призываем пакет для картинок

# Подгружаем из .stan-файлика описание модели.
model <- stan_model(file = "coin.stan")

# Шаманские заклинания для ускорения STAN:
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

# Загружаем данные. В данном случае придумываем данные...
y <- c(1, 1, 1, 1, 0, 1, 1)
N = length(y)

 # В списке переменные надо называть точно также как и в STAN
df <- list(N = N, y = y)

# Получаем выборку из апостериорных распределений:
fit <- sampling(model, data = df, iter = 1000, chains = 4)
```

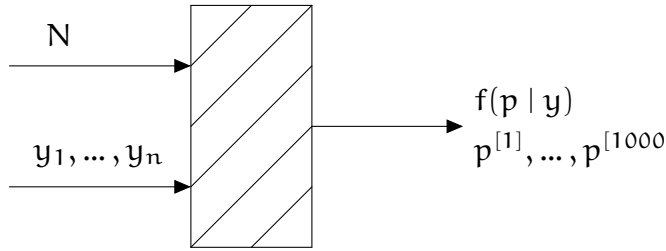
Предварительно мы установили пакет **rstan**, который связал R со STAN. Именно через него мы будем вызывать дух преподобного Томаса. При работе не забывайте положить все файлики в одну и ту же папочку и перед запуском кода выбрать её в качестве рабочей директории. Сделать это можно, нажав **«Session/Set Working Directory/Source file location»**.

Шаманские заклинания для ускорения STAN активируют режим записи скомпилированных программ на жёсткий диск. Это бывает полезно, если не хотите каждый раз компилировать программу заново для разных данных в одной модели[3].

Все байесовские чудеса происходят в 19-ой строчке кода. Конечно же STAN не занимается символьным вычислением апостериорного распределения. Компьютеры очень плохо работают с символьными вычислениями. На самом деле апостериорное распределение описывается с помощью большой выборки из него. STAN за кадром переписывает код нашей модели на C++, компилирует его и запускает несколько марковских цепей (в нашем случае четыре), порождающих апостериорную выборку с помощью специальных алгоритмов, которые мы разберём позже. Используя итоговую выборку, мы можем отвечать на интересующие нас вопросы. Четыре марковские цепи порождают четыре выборки для того, чтобы можно было проверить сошёлся ли алгоритм.

На блок-схеме ниже можно ещё раз увидеть, что происходит. Мы подаём в чёрный ящик число наблюдений и все наши измерения, а на выходе получаем выборку из апостериорного распределения для нашего параметра.

[3] Ликвидация безграмотности! Поясним что такое компиляция кода. Компьютер может понимать только очень ограниченный набор машинных команд, которые просто суммируют и сравнивают числа. Сложные приложения требуют миллиардов таких команд. Поэтому мы обычно пишем их на специально разработанных языках программирования. Компилятор это программа для перевода команд, которые мы написали на языке программирования в машинные команды, которые понятны процессору. Некоторые языки программирования выполняются без прямой компиляции в машинный код (R, Python). Команды в таком случае в режиме реального времени выполняет специальная программа — интерпретатор. Такой код работает намного медленнее, чем скомпилированный. Однако это обычно очень удобно. В больших проектах компиляция занимает много времени.



Результат работы нашего кода лежит в переменной fit. Попробуйте изучить внутри Rstudio её структуру. Если вызвать в консоли её содержимое, можно увидеть протокол с разными характеристиками, вычисленными по апостериорной выборке из 1000 наблюдений: среднее, стандартное отклонение, медиану, различные квантили. Можно даже увидеть прогнозное значение для нового игрека. Для него, кстати говоря, в ходе генерации было порождено своё распределение, на которое тоже можно взглянуть.

```
Inference for Stan model: 44492d184a933e08c1ee12cf9113fb17.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2000.

       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
p      0.78    0.00 0.14  0.46  0.69  0.80  0.88  0.97  1197    1
y_new  0.78    0.01 0.42  0.00  1.00  1.00  1.00  1.00  2939    1
lp__  -5.35    0.02 0.82 -7.60 -5.56 -5.04 -4.83 -4.77  1195    1
```

Чтобы извлечь апостериорные распределения, можно воспользоваться командой

```
1 p_aposterior = extract(fit)$p
2 y_aposterior = extract(fit)$y_new
```

Построить апостериорные гистограммы можно либо по-честному вбив `hist(p_aposterior)`, либо командами

```
fit_array = as.array(fit)
mcmc_hist(fit_array)
```

Намного удобнее визуализировать полученные результаты в интерактивном режиме с помощью строк

```
library("shinystan")
launch_shinystan(fit)
```

R заботливо перекинет вас на веб-страничку с кучей красивых рисунков и интерактивных визуализаций, отражающих то, как именно происходил процесс сэмплирования. На одной из них будет нарисован процесс сходимости наших четырёх марковских цепей.

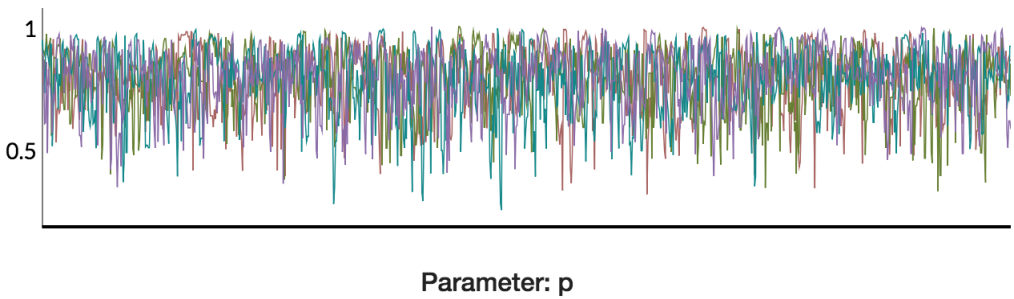
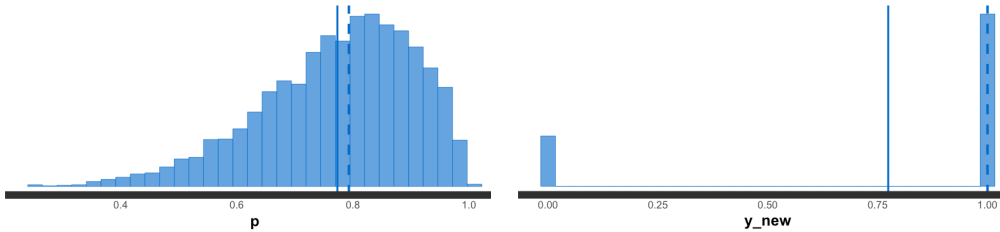


Рис. 1.1. Те самые четыре марковские цепи

С помощью этой диаграммы можно отслеживать пошло ли при генерации что-то не так.

Вставить пример расходящейся цепи

Среди огромной череды картинок нас прежде всего будут интересовать апостериорные распределения параметра p и прогноза y_{new}. Пунктирной линией на картинке обозначена медиана, непрерывной среднее.



Как и ожидалось, равномерное априорное распределение трансформировалось в горбатое апостериорное. Априорное распределение довольно легко поменять, отредактировав становский файл, а после заново скомпилировав его. Например, попробуйте заменить равномерное априорное распределение на бэта-распределение.

На той же веб-страничке во вкладке **More/Glossary** можно найти описание некоторых метрик, которые расчитываются при сэмплировании. Во вкладке **Estimate** можно найти готовый генератор латеховских таблиц с результатами сэмплирования. Давайте немного полюбуемся на табличку, а после пробежимся по основным определениям. Когда мы будем разбираться с устройством MCMC, мы ещё раз поговорим об этих параметрах и способах их расчёта.

Parameter	Rhat	n_eff	mean	sd	2.5%	50%	97.5%
p	1.0	1779	0.8	0.1	0.5	0.8	1.0
y_new	1.0	3351	0.8	0.4	0.0	1.0	1.0

- Мы генерируем апостериорное распределение с помощью марковской цепи. Результат первых итераций нельзя рассматривать как выборку из апостериорного распределения, так как к тому моменту алгоритм ещё не сошёлся. Параметр n_eff — это оценка числа наблюдений, которые мы можем использовать как выборку из апостериорного распределения. Если это число оказалось слишком маленьким, попробуйте увеличить число итераций.

- Выборку из апостерирорного распределения мы получим только если марковская цепь сошлась. Промониторить сходимость можно несколькими способами. Как уже говорилось выше, можно посмотреть на карти-

ночки. А можно посмотреть на метрику Rhat. По дефолту STAN генерирует четыре марковских цепочки. Статистика Rhat смотрит на отношение средней межвыборочной дисперсии (посчитали четыре дисперсии и усреднили) к дисперсии, рассчитанной по объединённой большой выборке. Если это отношение близко к единице, цепочки находятся в равновесии. Если они не сходятся к общему распределению, статистика Rhat будет больше единицы.

- Ещё одной метрикой мониторинга сходимости является se_mean. По всем марковским цепям считаются апостериорные средние. После для них вычисляется средняя квадратическая ошибка. Если хотя бы одна цепь не сошлась, ошибка оказывается высокой. При увеличении числа итераций, при условии сходимости цепей, эта метрика сходится к нулю.

Написать про lp. Это последнее чёрное пятно в протоколе.

Подведём итоги. STAN позволяет довольно просто, с помощью человеческого языка оценивать разные модели. На выходе мы получаем целое распределение! Вычленив его из нашей модели, мы можем отвечать на разные, интересующие нас вопросы. Например, вот так можно узнать какова вероятность, что p больше 0.5

```
p_aposterior = extract(fit)$p
sum(p_aposterior > 0.5)/length(p_aposterior)
```

STAN взаимодействует со всем, что движется: R, python, julia, matlab, stata. У него довольно подробная инструкция, логарифм числа страниц в ней чуть больше шестёрки,

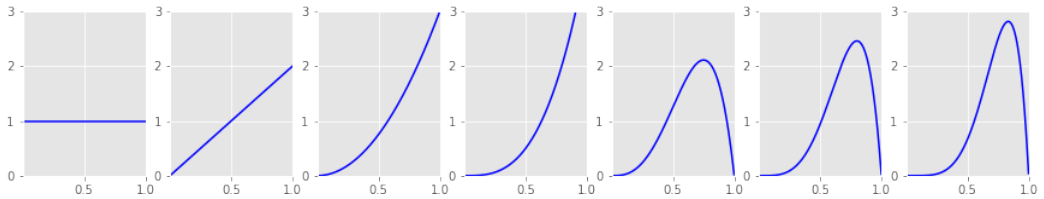
$$\ln(pages) \approx 6.36.$$

Даже официальных примеров куча,

$$\ln(examples) \approx 6.04.$$

Подгрузить примеры в RStudio можно командой `md <- stan_demo()`. В конце главы читатель с широким кругозором может найти целый пул источников информации по STAN.

Давайте напоследок посмотрим на то как меняется апостериорная плотность, когда к нам поступает всё больше и больше новых данных. Представим себе, что монетка трижды выпала орлом, потом один раз решкой, а потом снова два раза орлом.



В самом начале мы ничего не знаем о монетке и считаем, что p равномерно распределена на отрезке от нуля до единицы. При поступлении информации о первом орле, наши представления меняются, и вероятностная масса перетекает направо. Следущие два орла укрепляют нашу уверенность в том, что монета неправильная. Первая решка сбивает с нас спесь и распределение наконец становится горбатым. Следущие два орла сдвигают моду распределения вправо и делают его более вытянутым.

При каждом пересчёте параметры α и β нашего распределения растут, разброс для случайной величины p становится всё меньше, а купол становится всё острее. Попробуем перейти от монетки к паре более интересных задачек.

1.3 Наивный байесовский классификатор

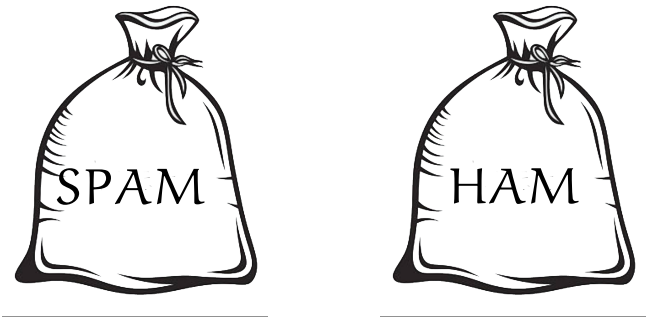
Представим себе следущую ситуацию: мы проснулись утром, заварили себе чаёк, взяли пару печенек и открыли почту, а там куча писем. Десяток про виагру, десяток про айфон в кредит и ещё двадцаток про онлайн-казино. Получить такое не очень приятно. Сразу хочется закрыть почту и завести себе сову или голубя.

Любому почтовому сервису хотелось бы, чтобы пользователь пореже получал спам и не хотел заводить себе почтовое животное. Для этого был приду-

ман наивный байесовский алгоритм. Машина считает сколько раз слово виагра (и другие слова) встречается в нормальных письмах, а сколько раз в спаме. На основе этих частот машина по формуле байеса находит вероятность, что письмо является плохим и ежели что, отправляет его в отдельную папку со спамом.

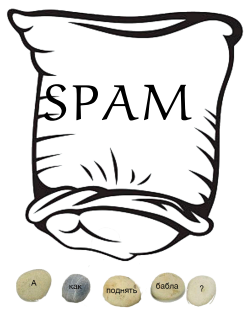
Классификатор писем на спам и не спам называется байесовским, так как он использует формулу Байеса. Наивным он называется, потому что перед применением мы делаем предположение о том, что вероятности попадания отдельных слов в письмо, не зависят друг от друга. На самом деле это неправда, и это предположение довольно наивно. Отсюда и название — наивный байес.

В самом начале главы мы сказали, что перед тем, как описывать математическую модель, полезным бывает представить в голове как именно устроено исследуемое нами явление с вероятностной точки зрения. Отбросим все предрассудки! Мир создал он — процесс порождения данных, и в случае спама он представляет собой пару мешков.



В каждом из мешков лежат камешки. На каждом камешке написано некоторое слово. Камешки с одинаковыми словами повторяются какое-то количество раз. В разных мешках разные наборы камешков. Конечно же, слово «виагра» есть и в мешке со спамом, и в мешке с нормальными сообщениями, но в мешке со спамом оно встречается гораздо чаще. Природа, перед тем как отправить письмо на наш почтовый ящик, генерирует его, доставая камешки из одного из мешков. Сначала она подкидывает неправильную монетку. В зависимости от того, какой стороной она выпала, природа определяется с

мешком. Пусть ей посчастливилось выбрать мешок со спамом. Природа начинает тянуть из этого мешка камешки и выстраивать спамное сообщение.



После сообщение отправляется к нам на почту. Вероятность увидеть конкретное письмо у себя на почте можно найти по формуле полной вероятности:

$$P(\mathrm{text}) = P(\mathrm{text} \mid \mathrm{spam}) \cdot P(\mathrm{spam}) + P(\mathrm{text} \mid \mathrm{ham}) \cdot P(\mathrm{ham}).$$

Мы можем по размеченной выборке оценить вероятности $P(\mathrm{text} \mid \mathrm{spam})$ и $P(\mathrm{text} \mid \mathrm{ham})$. У нас есть какое-то письмо text, оно состоит из слов $w_1, \ldots, w_n$ и, возможно, является спамом. Давайте посчитаем как часто конкретное слово встречается в спаме n_s и поделим это число на общую частоту слова n_w, получая при этом вероятность того, что слово всплывёт в спамном сообщении

$$P(w \mid \mathrm{spam}) = \frac{n_s}{n_w} \qquad P(w \mid \mathrm{ham}) = \frac{n_h}{n_w}$$

Перемножив вероятности встретить каждое из слов сообщения в спаме, мы получим вероятность появления этого текста, при условии что оно само является спамом

$$P(\mathrm{text}|\mathrm{spam}) = P(w_1 \mid \mathrm{spam}) \cdot \ldots \cdot P(w_n \mid \mathrm{spam}).$$

Именно здесь делается наивное предположение о независимости слов в сообщении. Это неправда. К сожалению, совместную вероятность $P(w_1 \ldots w_n \mid$

$spam$) оценить очень сложно. Для этого нам пришлось бы собрать громадный набор текстов, в который каждая из существующих последовательностей из слов вошла бы по несколько раз. Причём чем больше, тем лучше. Такая задача выглядит нерешаемой. Из-за этого приходится делать наивное предположение, что вероятности вхождения слов в текст — независимые события.

Мы научились по выборке оценивать $P(text|spam)$. Когда мы говорим о процедуре порождения текста, мы говорим о прямой задаче. Нас больше интересует обратная задача, состоящая в понимании того, при каких условиях родился текст. Перейти от прямой задачи к обратной помогает формула Байеса. Перевернём игру.

$$P(spam \mid text) \propto P(text \mid spam) \cdot P(spam) \propto$$
$$\propto P(w_1 \mid spam) \cdot ... \cdot P(w_n \mid spam) \cdot P(spam)$$

Как оценить вероятности $P(w_i \mid spam)$ мы обсудили выше. По сути их произведение это ни что иное как функция правдоподобия. Вероятность $P(spam)$ представляет собой априорную вероятность того, что сообщение перед нами является спамом. Если вы ничего не знаете о спаме, можно взять в качестве априорного распределения равномерное. При нём вероятность спама будет равна 0.5. Недавние исследования показали, что на сегодняшний день вероятность того, что сообщение, попавшее на почту является спамом, составляет около 0.8. В принципе, априорно можно быть очень скептическим и проповедовать презумпцию спама, оценивая $P(spam)$ очень высоко[4].

После того, как мы научим классификатор распознавать спам, мы сможем для каждого нового сообщения получить апостериорные вероятности $P(spam \mid text)$ и $P(ham \mid text)$. В качестве прогноза мы выберем тот класс, вероятность появления для которого больше. Это правило можно записать следующим незатейливым образом

$$\hat{y} = \arg\max_y P(text \mid y) \cdot P(y).$$

Переводится как: выбирай тот y для которого апостериорная вероятность максимальна.

[4]Найти эту не очень радостную статистику можно тут: http://certmag.com/more-than-90-percent-of-e-mails-in-third-quarter-were-spam/

Обучая классификатор (под обучением подразумевается оценка вероятностей для каждого слова и применение формулы Байеса) мы работаем с текстами писем. Вполне возможно, что в одном из новых писем мы встретим какое-то новое слово, которого не оказалось в нашей выборке. Вероятность его появления в спаме будет равна нулю. Итоговая вероятность занулится. Для того, чтобы такого не произошло, нужно будет либо выбросить все неизвестные слова из письма, либо сгладить вероятности, добавив в них дополнительное слагаемое, которое помешает им занулиться

$$P(w \mid spam) = \frac{n_s + a}{n_w + a + b} \qquad P(w \mid ham) = \frac{n_s + b}{n_w + a + b}.$$

В качестве a и b обычно берут какие-нибудь маленькие числа. Для сглаживания нужно ввести два числа, чтобы вероятности в сумме по-прежнему давали единицу.

Попробуем оценить два фильтра: без предубеждений (с априорной вероятностью спама 0.5) и с предубеждениями (с априорной вероятностью спама 0.8) и посмотреть какой из них лучше справляется со своими обязанностями.

В наших руках есть табличка. По строкам в ней отложены письма, по столбцам слова. В первом столбце стоит вердикт было ли сообщение спамом. Во втором столбце стоит 1, если слово make встретилось в письме и 0, если не встретилось, в третьем столбце та же информация приведена по слову adress и так далее (вместо факта вхождения слова в письмо можно было бы собирать количество слов в письме, суть от этого не меняется, но качество работы классификатора улучшается).

	is_spam	make	address	all	num3d	our	over	remove	internet	order	...
0	0	0	0	1	0	0	0	0	0	0	...
1	1	0	0	0	0	0	1	1	1	0	...
2	1	1	1	1	0	1	1	0	1	1	...
3	0	0	0	0	0	0	0	0	0	0	...
4	1	0	0	1	0	1	0	1	1	0	...
5	1	1	1	1	0	1	1	1	0	1	...

Давайте попробуем на этих данных оценить байесовский классификатор. Следущие два абзаца для прошаренного читателя будут очевидными. Разо-

бьём всю выборку на две части: обучающую и тестовую. На обучающей выборку мы будем искать для каждого слова условные частоты. На тестовой выборке мы будем проверять насколько точно работает наш классификатор. Проверять качество модели на обучающей выборке — нечестно! Из-за того, что мы учили частоты на этой выборке, качество модели на ней будет завышено. Она не встретит ничего непредсказуемого. Нам же хотелось бы видеть как модель работает в боевых условиях. Именно поэтому мы откладываем тестовую выборку.

Вообще сама по себе ситуация, когда модель вылизывает обучающую выборку, называется переобучением. Тестовая выборка должна помочь нам такую ситуацию отловить. Иногда выборку дробят не на две, а на три части: обучающую, валидационную и тестовую. При таком дроблении на обучающей выборке тренируют модель, на валидационной подбирают гиперпараметры (пример таких гиперпараметров вы найдёте через раздел), а на тестовой проверяют качество модели.

Дальше мы будем вбивать модель в STAN, оценивать её и получать прогнозы. Чтобы лучше понимать что происходит, попробуйте самостоятельно закодить наивного байеса в R без участия STAN. Это задание мы вынесли в конец главы в отдельное упражнение. К нему есть решение. Имеет смысл попробовать сделать его уже сейчас. Обратите внимание, что для удобства выборка уже разбита на обучающую и тестовую.

Сделали? Тогда врываемся в STAN. Создаём файлик **naive_bayes.stan** и начинаем его понемногу заполнять.

Возникли проблемы с тем, чтобы нормально задать связь между y и X. Пока без алгоритма на стане. Плак плак плак...

Качество угадывания оказалось равно 75%. Если каждое сообщение обзывать спамом, точность прогноза будет 42%. Если каждое сообщение называть хорошим, точность прогноза составит 58%. Поздравляю! Наш байесовский классификатор работает существенно лучше константного алгоритма. Можно побороться за качество его работы и расширить выборку из слов и писем. При больших выборках она окажется порядка 99%.

В реальном мире довольно часто оказывается, что интуитивные простые модели работают лучше сложных и навороченных. Перед тем как оценивать какую-нибудь забубенистую нейросеть, попробуйте сначала оценить несколь-

ко простых моделей. Возможно, они будут работать хорошо, и их качества вам уже будет хватать. В крайнем случае их можно будет использовать как стартовую точку.

Первая известная программа, фильтрующая почту с помощью байесовского классификатора появилась в 1996 году. Примерно года эдак до 2010 все почты оборудовали байесовским классификатором и этого хватало. Со временем спамеры стали хитрее и научились обходить фильтр Байеса, просто вставляя в конец письма много слов, которые часто фигурируют в хороших письмах. Такой метод обхода получил ироничное название отравление Байеса, а фильтровать спам стали другими алгоритмами.

В современных спамовых фильтрах стараются учитывать совместные появления слов. По большому корпусу текстов смотрят какие слова чаще всего встречаются вместе и добавляют такие устойчивые словосочетания, называемые **биграммами** в модель. Иногда смотрят ещё и триграммы. Также тексты перед построением модели обычно предобрабатывают. Например, выбрасывают нейтральные слова, которые встречаются во всех текстах подряд и не несут никакой информации о его природе. Обычно такими словами являются союзы, предлоги и тп. Их называют **стоп-словами**. Ещё стараются привести разные формы слова к одной форме. Так слово «люди» пытаются привратить в слово «человек», а слово «предкризисный» в слово «кризис». Такая процедура называется **нормализацией**. Когда мы будем говорить про вероятностные модели, связанные с порождением текстов, мы подробнее поговорим про все эти методы предобработки.

1.4 Лирическое отступление

Давайте ликвидируем ещё немного безграмотности. Обычно, когда ты спрашиваешь у человека про то, с помощью какой случайной величины можно замоделировать счётчик (число лайков, число людей в очереди, число букв «п» на странице книги), человек отвечает, что для этого идеально подходит распределение Пуассона, которое рождается из распределения Бернулли[5].

Если спросить того же человека о том, с помощью какого распределения можно смоделировать время, которое нужно прождать до появления события

[5]Ну либо говорит «Эээээээ».

(врямя до нового лайка, до нового человека в очереди, до новой поломки станка и тп), то он скажет, что для этого идеально подходит экспоненциальное распределение.

Выходит, что распределение Пуассона и экспоненциальное распределение как-то связаны. Если того же самого человека спросить как, он почти наверное не ответит. Давайте попробуем ликвидировать этот небольшой пробел в образовании.

Любую Пуассоновскую случайную величину можно увязать со временем. Например, можно рассматривать число лайков, которые сорвал пост за промежуток времени t. Тогда Пуассоновская случайная величина превращается в **простейший поток событий**. Вероятность того, что за промежуток t наступит k событий будет считаться по ɸормуле

$$P_t(k) = \frac{e^{-\lambda t} \cdot (\lambda t)^k}{k!}.$$

Простейший поток событий обладает несколькими интересными свойствами:

1. **Стационарность**. Появление k событий на каком-то промежутке зависит только от числа k и от длины этого промежутка. Точка начального отсчёта не имеет значения.

2. **Ординарность**. Наступление более одного события этого потока за бесконечно малый промежуток времени является практически невозможным.

3. **Отсутствия после действия**. Вероятность появления k событий на любом промежутке времени не зависит от того сколько событий произошло до этого.

Все эти свойства можно довольно легко доказать. Невооруженным взглядом видно, что вероятность зависит только от k и t, что характеризует стационарность потока, в ɸормуле не используется никакой инɸормации о том сколько событий наступило до этого, что характеризует свойство отсутствия

после действия. Немного поколдовав можно убедиться в ординарности. Можно даже отталкиваясь от этих свойств получить формулу Пуассона. Мы не будем делать это[6] .

Перейдём к самому интересному. Пусть T это промежуток времени между событиями. Давайте попробуем найти функцию распределения для этой случайной величины.

Будем делать ровно также, как мы делали это на семинарах по теории вероятностей ... через вероятность. Нас интересует

$$F_T(t) = P(T \leqslant t).$$

Вероятность $P(T \leqslant t)$ можно прочитать, как вероятность возникновения хотябы одного события в течение первых t минут. Вероятность того, что за t минут не возникнет ни одного события можно записать как

$$1 - F(t) = P(T > t).$$

Это логично, если время ожидания T перевалило за t, значит ни одного события ещё не наступило. На языке распределения пуассона

$$P(T > t) = P_0(t) = e^{-\lambda t}.$$

Значит

$$F_T(t) = 1 - e^{-\lambda t}, \quad t \geqslant 0.$$

Получилась функция распределения экспоненциальной случайной величины. Прочувствовали связь? Давайте немного углубим её. Для Распределения Пуассона математическое ожидание составляет λ. Этот параметр интерпретируется как интенсивность потока событий. Чем он больше, тем больше событий за единицу времени может произойти.

Для экспоненциального распределения математическое ожидание составляет $\frac{1}{\lambda}$. Чувствуете? Чем больше интенсивность потока событий, тем меньше в среднем времени проходит между ними. Теперь вы больше знаете о том,

[6]Можно посмотреть как это делается в книге Б.В. Гнеденко «Курс Теории вероятностей» на страницах 294-300.

как взаимосвязаны между собой Пуассоновская и экспоненциальная случайные величины, и мы можем закончить с лирическим отступлением и перейти к следующему примеру в STAN.

1.5 О Пуассоне и структурных сдвигах

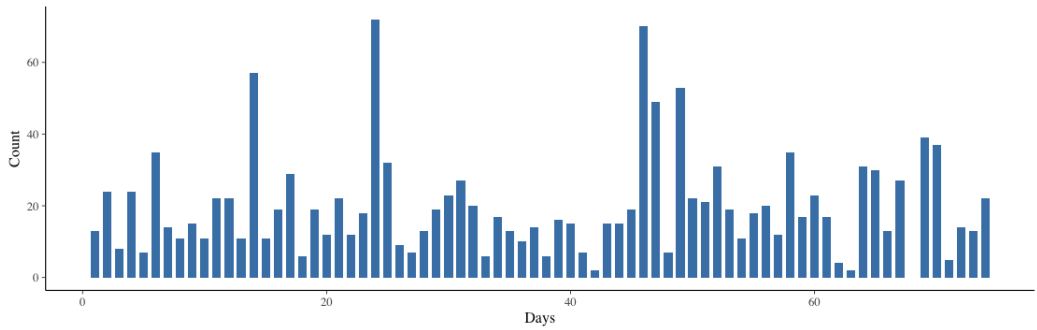
Думали, что мы уже закончили бороться со спамом? Нет, всё только начинается. Давайте проанализируем более интересный пример. Он касается скорости, с которой пользователь отправляет и получает сообщения.

Все мы пользуемся социальными сетями. Все мы пишем сообщения в некотором режиме. Иногда происходит ужасное, в наш аккаунт вламывается спамер и начинает всем рассылать непристойности. С помощью байесовских методов можно попытаться найти тот момент времени, когда в поведении пользователя произошло неожиданное изменение. Такие изменения в поведении принято называть **структурными сдвигами**.

Сразу же стоит оговориться, что структурный сдвиг байесовские методы то найдут, но не факт, что он происходит именно из-за взлома аккаунта. Возможно, всплеск в сообщениях произошёл по какой-то иной причине. Мы можем лишь идентифицировать его наличие в данных. Поэтому сразу же предостережём читателя, сказав, что для полноценной борьбы со злоумышленниками придётся строить более сложные модели. Например, с помощью байесовских методов можно понимать в какой момент произошёл структурный сдвиг, а после с помощью накрученного поверх классификатора (любого от логистической регрессии до нейросетки, можно даже байесовского, хватало бы только данных для обучения) можно было бы определять был ли этот структурный сдвиг злокачественным.

Давайте посмотрим на следущие данные:[7]

[7]Этот пример позаимствован из книги «Байесовские методы для хакеров». Ознакомиться с оригиналом можно по ссылке : Там вас ждёт решение части этой же задачки на питоне.



По оси x отложены дни, по оси y количество сообщений, отправленное пользователем в каждый из дней. Моделировать такие штуки можно распределением Пуассона. Параметр λ отражает то, насколько интенсивно отправляются сообщения. Большие значения λ присваивают большую вероятность более высоким значениям случайной величины.

Пусть в некоторый момент времени в поведении пользователя происходят изменения, он начинает писать больше сообщений, чем обычно. Момент, когда произошёл такой структурный сдвиг обозначим буквой τ. После наступления этого момента случайная величина стала стабильно принимать большие значения, значит интенсивность потока сообщений возросла и λ увеличилась. Таким образом до момента τ случайная величина имела распределение Пуассона с одной интенсивностью, после момента τ интенсивность подскочила.

$$X_t \sim \begin{cases} \text{Poiss}(\lambda_1), \text{ если } t < \tau \\ \text{Poiss}(\lambda_2), \text{ если } t \geqslant \tau \end{cases}$$

Если при этом никакого скачка в данных не было, то на выходе мы получим, что значения λ_1 и λ_2 будут очень близки, а их апостериорные распределения окажутся очень похожи.

Итак, у нас есть два параметра λ. Мы хотели бы оценить их используя процедуру байесовского вывода. Для этого нам нужно выбрать для λ априорные распределения. Что мы знаем об этих параметрах? Они могут принимать положительные значения. Скорее всего, их значения сосредоточены в какой-то конкретной области. Можно было бы попробовать использовать в качестве априорного распределения экспоненциальное $\lambda \sim \text{Exp}(\alpha)$.

Однако есть одна проблема. У экспоненциального распределения также есть параметр. Не очень понятно откуда его брать. Он находится как-бы за рамками нашей основной модели. Такие параметры называют **гиперпараметрами**. Можно поступить с ним несколькими способами:

- Придумать, опираясь на свои знания о структурных сдвигах, социальных сетях и сообщениях. Не забудьте быть максимально честным с собой.
- Ввести в модель ещё один уровень иерархии и сказать, что мы практически ничего про α не знаем, то есть $\alpha \sim U[0, +\infty]$.
- На данные смотреть в байесовском подходе до процедуры вывода нельзя. Но давайте поглядим. Оценим α по выборке, как $\frac{1}{\bar{x}}$, вспомнив о том как именно взаимосвязаны между собой экспоненциальное распределение и распределение Пуассона и будем оценивать оценку гиперпараметра в качестве априорной. Такой способ комбинирования частотной статистики с байесовской более-менее честный, однако не подойдёт для идеалиста.
- Можно подсмотреть в данные чуть хитрее. Если наша цель спрогнозировать, то можно раздробить выборку на обучающую и валидационную, а после по решётке перебрать все разумные значения гиперпараметра. В итоге выбрать тот, с которым прогнозы самые крутые. Возможно, это поможет выжать из модели максимум.

Мы попробуем применить второй и третий способы. Модель почти собрана. Осталось обсудить лишь структурный сдвиг τ. Это тоже параметр. Мы не знаем когда он произошёл, поэтому скажем, что $\tau \sim U[0; 70]$. Итак, наша модель:

$$
\begin{aligned}
X_t &\sim \begin{cases} \text{Poiss}(\lambda_1), \text{ если } t < \tau \\ \text{Poiss}(\lambda_2), \text{ если } t \geqslant \tau; \end{cases} \\
\lambda_1 &\sim \text{Exp}(\alpha); \\
\lambda_2 &\sim \text{Exp}(\alpha); \\
\tau &\sim \text{U}[0; 70]; \\
\alpha &= \frac{1}{\bar{x}}.
\end{aligned}
$$

С математической точки зрения у нас получилась редкостная монстрятина. Давайте перебьём её в STAN и посмотрим как он с ней разберётся. Перед этим найдём по выборке, что $\frac{1}{\bar{y}} = 0.05$, для того чтобы использовать это знание при моделировании λ.

```
data{
    int N;    // число наблюдений
    int y[N];  // в веторе y лежит сколько сообщений отправили в
     ↪ какой день
}
parameters{
    real <lower=1, upper=70> tau;  // момент сдвига
    real <lower=0> lambda1;           // интенсивность до сдвига
    real <lower=0> lambda2;          // интенсивность после
     ↪ сдвига
}
model{
    // априорно:
    tau ~ uniform(0, 70);
    lambda1 ~ exponential(0.05);
    lambda2 ~ exponential(0.05);
    // модель: как наблюдения связаны с параметром
    for (n in 1:N) {
        if(n < tau){
            y[n] ~ poisson(lambda1);
        }else{
```

```
            y[n] ~ poisson(lambda2);
        }
    }
}
// Чтобы было коректно в последняя строка традиционно пустая
```

Оборачиваем всё это дело в R ровно тем же самым кодом, что и в предыдущих случаях:

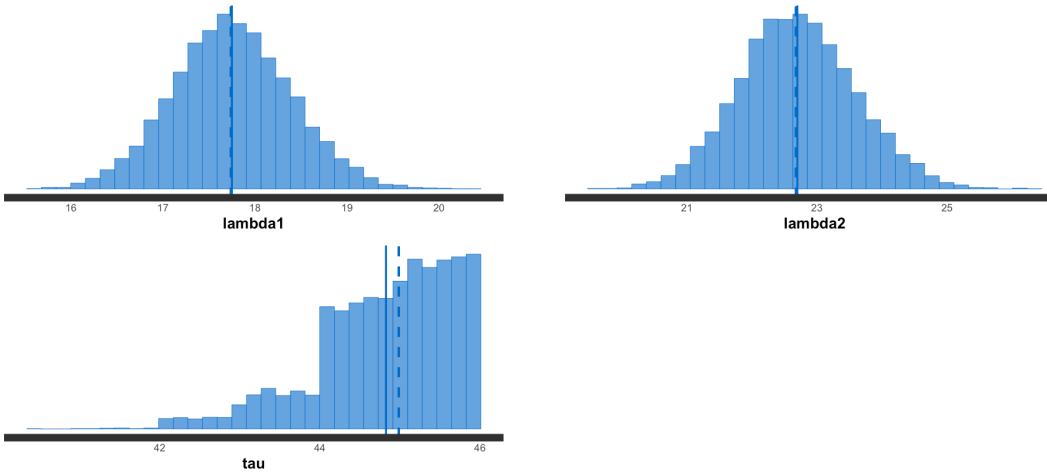
```
model <- stan_model(file = "poisson_shift.stan")
dff <- list(N = length(y), y = y)

# Получаем выборку из апостериорных распределений:
fit <- sampling(model, data = dff,iter = 5000)
fit   # Посмотрим на наш fit
```

Итоговый протокол оценивания модели будет выглядеть как-то так:

Parameter	Rhat	n_eff	mean	sd	2.5%	50%	97.5%
tau	1.0	1486	44.8	0.9	42.7	45.0	46.0
lambda1	1.0	2095	17.7	0.6	16.5	17.7	19.0
lambda2	1.0	1945	22.7	0.9	21.0	22.7	24.5

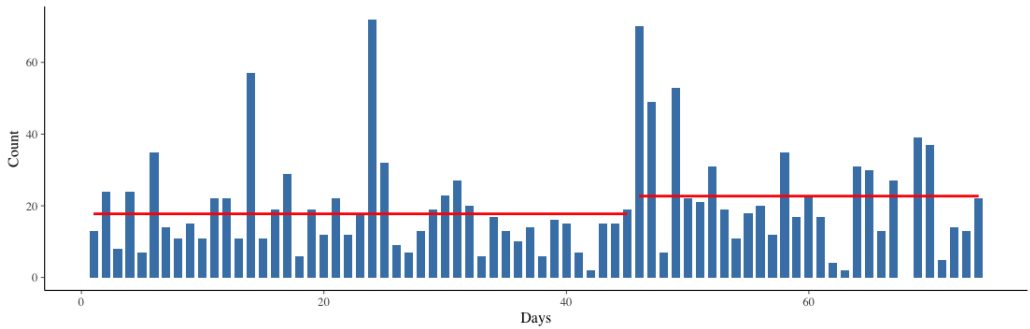
Напомним, что байесовский вывод возвращает нам целое распределение. В протоколе можно найти только точечные характеристики этих распределений. Давайте посмотрим как выглядят апостериорные гистограммы:



Сразу же мы видим неопределённость в наших оценках. Чем шире распределение, тем меньше наша уверенность в апостериорном результате. Заметим, что распределения для λ_1 довольно сильно отличается от распределения для λ_2. Гистограммы практически не пересекаются. Это говорит о высокой вероятности того, что произошёл структурный сдвиг.

Обратите внимание, что апостериорные распределение для λ вообще не похожи на экспоненциальные. Может возникнуть иллюзия, что они имеют нормальную форму. На самом деле это не так. Обычно апостериорные распределения не имеют какого-то описания через привычные нам станадртные распределения, если конечно это не случай сопряжённых распределений. Если бы мы попытались разрешить сформулированную выше модель аналитически, мы бы упёрлись в интегралы. Использование MCMC позволило избежать этого.

Распределение для τ сконцентрировалось в диапазоне от 42 до 46. Именно в этот период произошёл структурный сдвиг. В качестве точечной оценки подходит 45 день. Отметим ещё раз, что если бы никаких изменений не было или если бы изменение было постепенным с течением времени, то распределение для τ оказалось бы более неоднородным и захватило бы гораздо больший временной диапазон. Мы же видим, что только 4 дня могут рассматриваться в качестве потенциальных точек структурного изменения. Давайте нанесём среднее значение случайной величины X до сдвига и после сдвига на нашу картинку с динамикой сообщений.



Сдвиг получился не очень большим. В голове рождается мысль, что он связан со статистической погрешностью. Нужна какая-то величина, которая отражала бы нашу степень уверенности в наличии этого сдвига. И у нас такая величина есть. Это $P(\lambda_2 > \lambda_1)$. Чтобы сразу же получить её, давайте немного видоизменим модель. Будем говорить, что

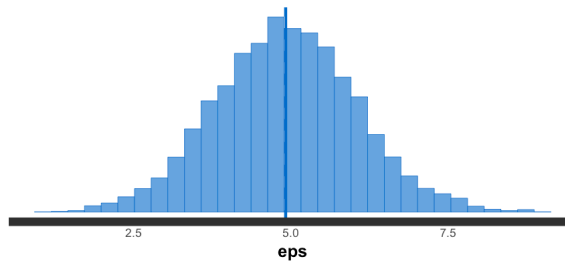
$$X_t \sim \begin{cases} \text{Poiss}(\lambda), \text{ если } t < \tau, \\ \text{Poiss}(\lambda + \varepsilon), \text{ если } t \geqslant \tau. \end{cases}$$

Тогда мы сразу же после генерации сможем посмотреть по распределению случайной величины ε насколько велика вероятность структурного сдвига. Для этого немного подкорректируем наш код в становском файлике:

```
parameters{
    real <lower=1, upper=70> tau;  // момент сдвига
    real <lower=0> lambda;
    real  eps;
}
model{
    // априорно:
    tau ~ uniform(0, 70);
    lambda ~ exponential(0.05);
    eps ~ exponential(0.05);
    // модель: как наблюдения связаны с параметром
    for (n in 1:N) {
        if(n < tau){
```

```
            y[n] ~ poisson(lambda);
        }else{
            y[n] ~ poisson(lambda + eps);
        }
    }
}
```

Теперь наш структурный сдвиг заложен в ε. Апостериорное распределение этого параметра будет выглядеть следущим образом:



Вероятностная масса сосредоточена существенно правее нуля. Апостериорно у нас есть все основания верить в него. Вероятность того, что ε больше нуля можно посчитать, вынув из модели апотсериорное распределение. В этом может помочь следущий зверь:

```
sum(fit@sim$samples[[1]]$eps > 0)/fit@sim$iter
```

Спойлер: $P(\varepsilon > 0) = 1$. Теперь давайте попробуем проделать всё ровно то же самое, но заменим подглядывание в данные для вычисления α на дополнительный уровень в иерархии.

$$
\begin{aligned}
X_t &\sim \begin{cases} \text{Poiss}(\lambda_1), \text{ если } t < \tau \\ \text{Poiss}(\lambda_2), \text{ если } t \geqslant \tau; \end{cases} \\
\lambda_1 &\sim \text{Exp}(\alpha_1); \\
\lambda_2 &\sim \text{Exp}(\alpha_2); \\
\alpha_1 &\sim \text{U}[0; +\infty]; \\
\alpha_2 &\sim \text{U}[0; +\infty]; \\
\tau &\sim \text{U}[0; 70].
\end{aligned}
$$

Замоделировать α_i внутри становского кода можно следующим образом:

```
alpha1 ~ uniform(0,1000);
```

На выходе мы получим ровно такие же распределения для интересующих нас параметров. Распределения для α_1 и α_2 будут подозрительно похожи на экспоненциальные. В этом любопытный читатель может самостоятельно убедиться, перебив соответствующий код в STAN.

Пришло время раскрыть правду. На самом деле 45 день соответствует рождеству. Автор задачи в этот день уехал в другой город к родным, на время расставшись со своей девушкой. Расставание пришлось компенсировать перепиской. Именно из-за этого интенсивность потока сообщений возросла. Мы на этом не прощаемся со структурными сдвигами. Позже мы ещё раз вернёмся к ним, но на этот раз в контексте макроэкономических проблем.

Если эксперименты с вк закончатся успешно, заменть задачу на новую.

1.5.1 Почиташки про STAN

- По ссылкеможно скачать pdf-ку с документацией.
- В документации можно найти большое количество интересных примеров моделирования. Некоторые довольно нетривиальные.

1.6 Ещё задачи

Упражнение 1. Немного простых задачек на генерации. Если вы не знаете, какая команда порождает необходимое вам распределение, никто не запрещает подглядеть это в документации.

1. Вспомним задачку про Машу и медведей из прошлой главы. Попробуйте смоделировать её в SNTAN.
2. Попробуйте вбить в STAN задачку про Сашу, Наташу и ресторан ПушкинЪ;

3. Вбейте в STAN задачку про подтягунов. Постарайтесь не запутаться в ней;

4. Напишите в STAN код для другой своей любимой задачи из третьей главы.

По нижеследующим ссылкам можно найти решения всех трёх задачек.

Сделать решения

1. Задачка про Машу.

2. Задачка про ресторан.

3. Задача о подтягиваниях.

Упражнение 2. Придумать упражнение на двумерное нормальное распределение!

Упражнение 3. Помните, что сюжет книги нелинеен? Так вот. В первой главе вы решали задачку на метод максимального правдоподобия. Про наркотики среди студентов. Пришло время решить её ещё разок, но уже используя байесовский подход. Напомню условия задачи.

Каждый из группы второкурсников, пришедших на пару, подкинул монетку два раза и никому не рассказывал, что у него выпало. В зависимости от того, что выпало на монетках, а также от того пробовал он наркотики или нет, он сказал одну из двух фраз.

	выпали два орла	другая комбинация
пробовал наркотики	ололо	юпи
не пробовал наркотики	юпи	ололо

Таким образом, если человек ололо, либо он курил травку и выпали два орла, либо он не курил травку и выпала другая комбинация. Всегда можно отмазаться. Получается, что при такой форме опроса нет смысла врать, и мы получим из него намного больше информации, чем если бы спрашивали у человека напрямик. «Ололо» сказали 10 человек, «юпи» сказали 4 человека.

1. Вспомните оценку максимального правдоподобия для доли пробовавших наркотики.
2. С помощью STAN найдите апостериорное распределение доли пробовавших наркотики.
3. Постройте для доли 80% байесовский интервал. Чем этот интервал отличается от доверительного?
4. Исследователь забыл правильную ли он монетку дал ребятам. Пусть q - вероятность орла. Оценить одновременно долю наркоманов и q.

Предположим, что мы вообще ничего не знаем о второкурсниках. В качестве априорного мнения о доле наркоманов возьмём равномерное распределение.

$$p \sim U[0; 1]$$

На первом этапе все студенты подбрасывают монетку. На ней выпадает одна из заданных комбинаций. На втором этапе природа подбрасывает свою монетку и либо подсаживает студента на наркотики, либо оставляет его в покое[8].

Упражнение 4. Самостоятельно на своём любимом языке программирования реализуйте наивного байеса. Используйте для оценивания модели данные из главы. Сравните результаты оценивания с тем, что мы выше получили в STAN.

Скорее всего отважный читатель ринулся со всей своей прытью писать два вложенных друг в друга цикла. Один по словам бегает, другой вхождения считает. Это конечно мило, но можно проще. Мы будем делать это разными векторными командами! R — векторный язык программирования. Если какая-то команда вам сходу непонятна, попробуйте применить её отдельно от всего.

```
df_train <- read.csv("spam_train.csv")    # подгрузили
  ↪ тренировочную выборку
df_test <- read.csv("spam_test.csv")      # подгрузили тестовую
  ↪ выборку
```

[8]Реальный мир очень суров.

```
3
4  n_col = ncol(df_train)   # число столбцов
5  n_row = nrow(df_train)   # число наблюдений
6  p_spam = 0.5             # априорная вероятность спама
7
8  # сделали срез по всем сообщениям со спамом и нашли сумму по
    ↪  столбцам
9  words_spam_freq <- colSums(df_train[df_train[,1] == 1,])
10
11 # избавились от столбца с ответами и поделили на число спамных
    ↪  строк
12 words_spam_freq <- words_spam_freq[2:n_col]/sum(df_train[,1] ==
    ↪  1)
13
14 # сделали то же самое с неспамными словами
15 words_ham_freq <- colSums(df_train[df_train[,1] == 0,])
16 words_ham_freq <- words_ham_freq[2:n_col]/sum(df_train[,1] == 0)
```

Сейчас в листе $\mathtt{words\_spam\_freq}$ лежат $P(w_i \mid spam)$. Лист это структура, к которой мы можем обращаться по именам столбцов через квадратные скобки. Кусок кода выше отвечает за обучение модели. Мы оценили для каждого слова вероятность встретиться в спамном сообщении. При желании можно завернуть этот код в функцию и как-нибудь красиво назвать.

Второй частью кода будет применение полученных частот к тестовой выборке. Для первого наблюдения из тестовой выборки мы можем подсчитать спамность следующим образом:

сделать нормальные формулы!

```
1  names(df_test[1,])[1]   # выдаст имя первой колонки, то есть
    ↪  слово,  которому соотвествуют частоты в столбце
2
3  # выдаст P(w_i | spam) для этого слова
4  words_freq[names(df_test[1,,2:n_col])[1]]
5
6  # найдём P(text | spam)*P(spam)
```

```
prod(words_freq[names(df_test[,2:n_col])[df_test[1,2:n_col] > 0]])*p_spam

# найдём P(text | ham)*P(ham)
prod(words_ham_freq[names(df_test[,2:n_col])[df_test[1,2:n_col] > 0]])*(1 -p_spam)
```

Обратите внимание, что мы при поиске апостериорной вероятности, пренебрегли константой. Для того, чтобы сделать прогноз нам остаётся сравнить две, полученные на выходе вероятности между собой и определиться с тем, что из себя представляет сообщение.

```
# Сделаем прогнозы по всей тестовой выборке
forecasts = c()

for(i in 1: nrow(df_test)){
    spam = prod(words_freq[names(df_test[,2:n_col])[df_test[i,2:n_col] > 0]])*p_spam
    ham = prod(words_ham_freq[names(df_test[,2:n_col])[df_test[i,2:n_col] > 0]])*(1 -p_spam)
    if(spam > ham){
        forecasts = append(forecasts,1)
    }else{
        forecasts = append(forecasts,0)
    }
}

# Посмотрим как часто мы угадываем, где в письмах есть спам
sum(df_test[,1] == forecasts)/length(forecasts)
```

Качество угадывания оказалось равно 75%. Если каждое сообщение обзывать спамом, точность прогноза будет 42%. Если каждое сообщение называть хорошим, точность прогноза составит 58%. Поздравляю! Наш байесовский классификатор работает существенно лучше наивного алгоритма.

Разобрались с наивным байесом? Контрольный вопрос! Предположим, что у нас есть k классов. Наблюдения распределены по ним неравномерно. Апостериорная вероятность выпадения каждого класса составляет $P(y \mid x) = \frac{1}{k}$. Какой класс в таком случае будет выдавать байесовсий классификатор для произвольного x?

Внимание, ответ: наибольший.

Упражнение 5. 28 января 1986 года двадцать пятый полёт программы космических шаттлов США закончился катастрофой[9]. Один из ракетных ускорителей шаттла «Челленджер» взорвался вскоре после взлёта, в результате чего погибли все семь членов экипажа. Президентская комиссия по авиационным происшествиям пришла к выводу, что взрыв был вызван отказом уплотнительного кольца в полевом стыке ракеты-носителя. Кольцо отказала в силу неправильной конструкции, которая сделала уплотнительное кольцо слишком чувствительным к ряду факторов. В том числе к температуре наружного воздуха.

Из предыдущих 24 рейсов были доступны данные о сбоях 23 уплотнительных колец (в течение одного из рейсов кольцо потеряли в море). Эти данные обсуждались вечером накануне трагедии. К сожалению, важными считались только данные, соответствующие 7 рейсам, на которых произошло повреждение. Они не показывали очевидной тенденции. Температура оценивается в градусах по Фарингейту. Данные представлены в таблице.

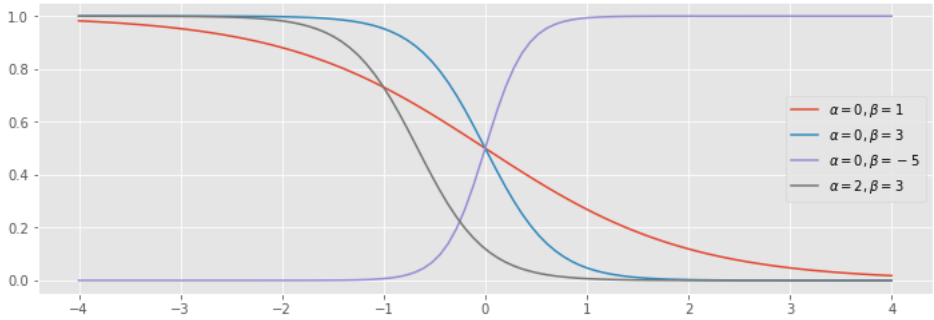
скрин таблички с данными

Итак, вероятность катастрофы зависит от температуры. Обычно, в моделях, где необходимо предсказать вероятность бинарного события, используется сигмоида:

$$P(t) = \frac{1}{1 + e^{\alpha + \beta t}}.$$

Параметр β отвечает за то насколько резко сигмоида перетекает из нулевого состояния в первое. Параметр α отвечает за сдвиг.

[9] Ещё одна задача, взятая из «Байесовских методов для хакеров».



Поначалу вероятность возникновения дефекта довольно низка. По мере изменения температуры она увеличивается. Сначала постепенно, по достижении некоторого критического уровня в вероятности происходит резкий подскок и мы по S-образной кривой переходим в новое состояние[10]. Наша задача оценить как именно происходит этот переход в новое состояние. С помощью STAN:

1. Найдите апостериорное распределение для параметров α и β. Априорно предположите, что $\alpha \sim N(0, 0.001)$, $\beta \sim N(0, 0.001)$. То есть априорно вероятность катастрофы очень слабо зависит от температуры.

2. Какова вероятность того, что α меньше нуля? Какова вероятность, что β меньше нуля?

3. Представим, что вечером накануне катастрофы вы попали на совещание. Прогноз погоды сообщает, что при завтрашнем запуске температура составит 30 градусов по Фарингейту. Постройте апостериорное распределение вероятности катастрофы, при этом условии. Постройте для этой вероятности HPD.

4. Вопрос про качество модели.

Добавить про качество модели в текст про структурные сдвиги

[10]Такая ситуация характерна для многих задач, связанных с классификацией. Например, если мы исследуем зависимость наличия у семьи автомобиля от стоимости дохода, мы понимаем, что поначалу вероятность покупки машины растёт медленно. Как только доход достигает критического уровня, происходит резкий рост вероятности.

Вбиваем в STAN нашу модель.

1

Отметить, что у апостериорных распределений два горба
Пояснить за логистическую регрессию

Упражнение 6. В группе мемы про машинное обучение для взрослых мужиков постят мемы про машинное обучение для взрослых мужиков. Взрослые мужики смотрят мемы про машинное обучение, лайкают их, комментируют и репостят. В итоге рождается табличка со статистикой.

Предположим, что просмотры, лайки, число комментариев и репосты имеют распределение Пуассона, $X_i \sim \mathtt{Poiss}(\lambda)$. Также будем предполагать, что число лайков (и других показателей) на текущем посте не зависит от числа лайков на других постах.

1. ZERO INFLATED MODEL для репостов и лайков
2. HPD для них
3. в какой из ситуаций $\mathtt{p}$ нулевые и тп
4. для репостов и для лайков геометрические распределения до первой неудачи, какие из моделей лучше

Упражнение 7. Что-нибудь про шары и гипергеометрическое распределение

Упражнение 8. байесовская оптимизация самостоятельно в R какой-нибудь функции

Упражнение 9.

Задачка на карму, просто подумать