

Оглавление

Предисловие.....	14
Введение	15
Благодарности.....	17
Об авторе.....	18
От издательства	19
Глава 1. Философия байесовского вывода.....	20
1.1. Введение	20
1.1.1. Байесовское мышление.....	20
1.1.2. Байесовский вывод на практике.....	23
1.1.3. Корректны ли фреквентистские методы?	24
1.1.4. О проблеме больших данных	24
1.2. Понятийный аппарат байесовского подхода.....	25
1.2.1. Пример: подбрасывание монетки (куда же без него)	25
1.2.2. Пример: библиотекарь или фермер?	27
1.3. Распределения вероятностей.....	29
1.3.1. Дискретный случай.....	30

1.3.2. Непрерывный случай	32
1.3.3. Но что такое λ ?.....	33
1.4. Использование компьютеров для автоматического байесовского вывода	34
1.4.1. Пример: вывод поведения на основе данных по обмену текстовыми сообщениями	34
1.4.2. Наш первый инструмент: РуМС	36
1.4.3. Толкование результатов	40
1.4.4. Какую пользу могут принести выборки из апостериорного распределения?	41
1.5. Выводы.....	43
1.6. Приложение.....	43
1.6.1. Статистическое определение фактического различия двух параметров λ	43
1.6.2. Обобщаем на случай двух точек ветвления.....	45
1.7. Упражнения	47
1.7.1. Ответы.....	47
1.8. Библиография.....	49
Глава 2. Еще немного о РуМС.....	50
2.1. Введение	50
2.1.1. Связи «предок — потомок»	50
2.1.2. Переменные РуМС.....	51
2.1.3. Учет наблюдений в модели	55
2.1.4. И наконец...	56
2.2. Подходы к моделированию	57
2.2.1. Та же история, но с другой концовкой	58
2.2.2. Пример: байесовское А/В-тестирование	62
2.2.3. Простой случай.....	62

2.2.4.	А и В вместе.....	65
2.2.5.	Пример: алгоритм обнаружения мошенничества.....	70
2.2.6.	Биномиальное распределение.....	70
2.2.7.	Пример: мошенничество среди студентов.....	71
2.2.8.	Альтернативная модель РумС.....	75
2.2.9.	Еще несколько хитростей РумС.....	77
2.2.10.	Пример: катастрофа космического челнока «Челленджер».....	77
2.2.11.	Нормальное распределение.....	81
2.2.12.	Что произошло в день катастрофы «Челленджера».....	87
2.3.	Адекватна ли наша модель?.....	87
2.3.1.	Разделительные графики.....	90
2.4.	Выводы.....	94
2.5.	Приложение.....	94
2.6.	Упражнения.....	95
2.6.1.	Ответы.....	95
2.7.	Библиография.....	96
Глава 3.	Открываем «черный ящик» МСМС.....	97
3.1.	Байесовский ландшафт.....	97
3.1.1.	Изучаем ландшафт с помощью МСМС.....	103
3.1.2.	Алгоритмы для МСМС.....	104
3.1.3.	Другие приближенные методы поиска апостериорных распределений.....	105
3.1.4.	Пример: кластеризация без учителя с использованием смеси распределений.....	105
3.1.5.	Не смешивайте апостериорные выборки.....	115
3.1.6.	Использование MAP для улучшения сходимости.....	118

3.2. Диагностика проблем со сходимостью	120
3.2.1. Автокорреляция	120
3.2.2. Прореживание	123
3.2.3. Функция <code>plt.rcParams</code>	124
3.3. Полезные советы по поводу MCMC.....	126
3.3.1. Интеллектуальный выбор начальных значений	127
3.3.2. Априорные распределения.....	127
3.3.3. Народная теорема статистических расчетов.....	127
3.4. Выводы.....	128
3.5. Библиография.....	128
Глава 4. Величайшая из несформулированных теорем	129
4.1. Введение	129
4.2. Закон больших чисел.....	129
4.2.1. Интуиция	129
4.2.2. Пример: сходимость пуассоновских случайных переменных.....	130
4.2.3. Как вычислить $\text{Var}(Z)$	134
4.2.4. Математические ожидания и вероятности.....	134
4.2.5. Какое отношение все это имеет к байесовской статистике	135
4.3. Некорректная работа при малых числах	135
4.3.1. Пример: агрегированные географические данные.....	135
4.3.2. Пример: конкурс Kaggle (перепись населения США).....	138
4.3.3. Пример: сортировка комментариев на Reddit	139
4.3.4. Сортировка	144
4.3.5. Но для режима реального времени это слишком медленно!	146
4.3.6. Расширение на системы оценки с присвоением звезд	151
4.4. Выводы.....	151

4.5. Приложение.....	152
4.5.1. Дифференцирование формулы сортировки комментариев	152
4.6. Упражнения	152
4.6.1. Ответы.....	154
4.7. Библиография.....	154
Глава 5. Что лучше: потерять руку или ногу?	155
5.1. Введение	155
5.2. Функции потерь	155
5.2.1. Функции потерь на практике.....	158
5.2.2. Пример: оптимизация для раунда «Витрина» в викторине «Справедливая цена»	159
5.3. Машинное обучение с помощью байесовских методов	167
5.3.1. Пример: предсказание финансовых показателей	168
5.3.2. Пример: конкурс Kaggle по поиску темной материи.....	173
5.3.3. Данные	174
5.3.4. Априорные распределения.....	176
5.3.5. Обучение и РумС-реализация	177
5.4. Выводы.....	185
5.5. Библиография.....	185
Глава 6. Расставляем приоритеты	186
6.1. Введение	186
6.2. Субъективные и объективные априорные распределения	186
6.2.1. Объективные априорные распределения	186
6.2.2. Субъективные априорные распределения	187
6.2.3. Выбираем, выбираем...	188
6.2.4. Эмпирическая байесовская оценка.....	190

6.3. Некоторые полезные априорные распределения	191
6.3.1. Гамма-распределение	191
6.3.2. Распределение Уишарта.....	192
6.3.3. Бета-распределение.....	194
6.4. Пример: байесовские многорукие бандиты	195
6.4.1. Приложения.....	196
6.4.2. Предлагаемое решение	196
6.4.3. Мера качества.....	201
6.4.4. Обобщения алгоритма	205
6.5. Сбор информации для априорных распределений у специалистов по предметной области.....	208
6.5.1. Метод рулетки испытаний.....	209
6.5.2. Пример: биржевая прибыль	210
6.5.3. Советы от профи по поводу распределения Уишарта.....	219
6.6. Сопряженные априорные распределения.....	220
6.7. Априорное распределение Джеффриса.....	221
6.8. Влияние априорных распределений при изменении N	223
6.9. Выводы.....	225
6.10. Приложение.....	226
6.10.1. Байесовская точка зрения на линейную регрессию со штрафом	226
6.10.2. Выбор вырожденного априорного распределения	228
6.11. Библиография.....	230
Глава 7. A/B-тестирование	231
7.1. Введение	231
7.2. Краткое резюме вышеприведенного A/B-тестирования конверсий	231

7.3. Добавляем линейную функцию потерь	234
7.3.1. Анализ ожидаемой выручки	234
7.3.2. Обобщение на случай А/В-эксперимента.....	238
7.4. Выходим за рамки конверсий: тест Стьюдента.....	240
7.4.1. Схема теста Стьюдента	241
7.5. Оценка показателя роста	245
7.5.1. Создание точечных оценок	248
7.6. Выводы.....	249
7.7. Библиография.....	250
Глоссарий.....	251

2.2.5. Пример: алгоритм обнаружения мошенничества

В данных социологических исследований есть еще один интересный пласт. Люди не всегда правдивы в своих ответах, что еще более усложняет получение выводов. Например, если просто спросить испытуемых: «Мошенничали ли вы когда-нибудь на экзамене?» — то полученные ответы наверняка будут не вполне честны. Наверняка можно сказать лишь то, что фактическая частота будет меньше наблюдаемой (в предположении, что респонденты лгут *только* о том, что *не мошенничали*; мне сложно представить себе респондента, который бы утверждал, что мошенничал, если на самом деле вел себя честно).

Чтобы показать изящный способ обхода этой проблемы нечестности и продемонстрировать байесовское моделирование, я сначала познакомлю вас с биномиальным распределением.

2.2.6. Биномиальное распределение

Биномиальное распределение — одно из самых популярных распределений, в основном из-за его простоты и удобства. В отличие от других встречавшихся в этой книге распределений, у биномиального два параметра: N — положительное целочисленное значение, отражающее количество испытаний или количество экземпляров потенциальных событий, и p — вероятность появления события в отдельном испытании. Аналогично пуассоновскому распределению оно является дискретным, но, в отличие от пуассоновского, в нем задаются вероятности только для чисел от 0 до N (в пуассоновском — от 0 до бесконечности). Функция распределения масс выглядит следующим образом:

$$P(X = k) = \binom{N}{k} p^k (1-p)^{N-k}.$$

Если X — биномиальная случайная переменная с параметрами N и p , что обозначается как $X \sim \text{Bin}(N, p)$, то X равна числу событий, произошедших при N испытаниях ($0 \leq X \leq N$). Чем больше p (в пределах от 0 до 1), тем больше событий, вероятно, произойдет. Математическое ожидание биномиального распределения равно Np . Построим распределение вероятностей при различных значениях параметров (рис. 2.8).

```
figsize(12.5, 4)
```

```
import scipy.stats as stats
binomial = stats.binom
parameters = [(10, .4), (10, .9)]
colors = ["#348ABD", "#A60628"]
```

```
for i in range(2):
```



```

N, p = parameters[i]
_x = np.arange(N + 1)
plt.bar(_x - 0.5, binomial.pmf(_x, N, p), color=colors[i],
        edgewidth=3,
        alpha=0.6,
        label="$N$: %d, $p$: %.1f" % (N, p),
        linewidth=3)

plt.legend(loc="upper left")
plt.xlim(0, 10.5)
plt.xlabel("$k$")
plt.ylabel("$P(X = k)$")
plt.title(u"Распределения вероятностей биномиальных случайных переменных");

```

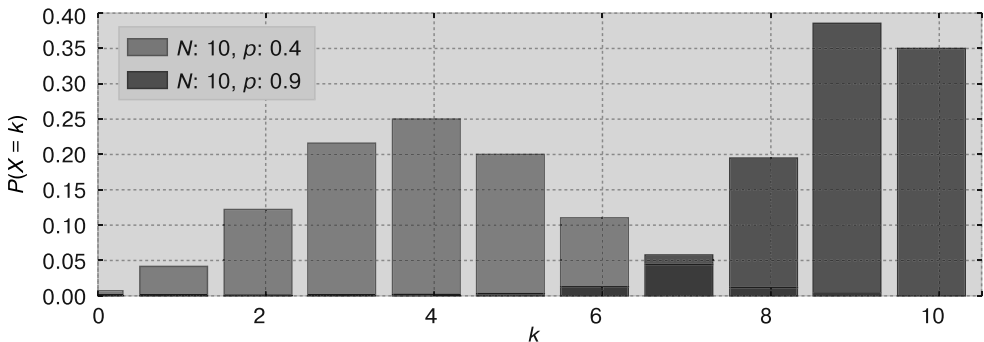


Рис. 2.8. Распределения вероятностей биномиальных случайных переменных

Частный случай с $N = 1$ соответствует распределению Бернулли. Бернуллиевы и биномиальные случайные переменные связывает и еще кое-что. Если даны бернуллиевы случайные переменные X_1, X_2, \dots, X_N с одинаковым параметром p , то $Z = X_1 + X_2 + \dots + X_N \sim \text{Binomial}(N, p)$.

2.2.7. Пример: мошенничество среди студентов

Воспользуемся биномиальным распределением для определения процента студентов, мошенничавших во время экзамена. Пусть N — общее число сдававших экзамен студентов. После экзамена всех студентов опрашивают (причем результаты опроса не влекут за собой никаких последствий), и мы получаем целое число X ответов «Да, я мошенничал». Далее можно найти апостериорное распределение p при заданных N , некотором априорном распределении p и данных наблюдений X .

Это совершенно абсурдная модель. Никакой студент даже при полной безнаказанности не сознается в мошенничестве. Нам нужен более совершенный алгоритм опроса студентов относительно мошенничества. Желательно, чтобы он поощрял

честность в ответах при полной конфиденциальности. Ниже приведен алгоритм, который восхитил меня своей неординарностью и эффективностью [3]:

«В процессе опроса студентов каждый студент подбрасывает (скрытно от опрашивающего) монету. Если выпадает орел, то студент отвечает честно. В противном случае (если выпадает решка) студент (скрытно) подбрасывает монету опять и отвечает: “Да, я мошенничал” — если выпадает орел, и “Нет, я не мошенничал” — если выпадает решка. При этом опрашивающий не знает, был ли ответ “Да” результатом чистосердечного признания, или орлом, выпавшим при втором подбрасывании монеты. Таким образом, конфиденциальность сохраняется, а исследователи получают правдивые ответы».

Я называю его конфиденциальным алгоритмом (для краткости: конф-алгоритмом). Можно спорить, что опрашивающие все равно получают ложные данные, поскольку часть ответов «Да» не признания, а результат случайного процесса. Иными словами, исследователи теряют примерно половину исходного набора данных, поскольку половина ответов случайны. С другой стороны, они получили хорошо моделируемый систематический процесс генерации данных. Более того, отсутствует необходимость учитывать (вероятно, немного наивно) возможность неискренних ответов. А далее можно с помощью PyMC обработать эту модель с «помехами» и найти апостериорное распределение фактической частоты лжецов.

Пусть на предмет мошенничества опрашивается 100 студентов, и нам нужно найти p — процент мошенничавших. Смоделировать это в PyMC можно несколькими способами. Я покажу сначала наиболее очевидный способ, а позднее — упрощенную версию. Выводы в обеих версиях получаются одинаковые. В нашей модели генерации данных мы производим выборку p — фактической доли мошенников — из априорного распределения. Поскольку никакой информации относительно переменной p у нас нет, мы выберем для нее априорное распределение $\text{Uniform}(0, 1)$.

```
import pymc as pm
```

```
N = 100
```

```
p = pm.Uniform("freq_cheating", 0, 1)
```

Опять же, с учетом нашей модели генерации данных используем для 100 студентов бернуллиевы случайные переменные: 1 означает, что студент смошенничал, а 0 — что нет.

```
true_answers = pm.Bernoulli("truths", p, size=N)
```

Если продолжить выполнение этого алгоритма, то следующим шагом будет первое подбрасывание монеты студентами. Опять же, его можно промоделировать путем выборки 100 бернуллиевых случайных переменных с $p = 1 / 2$, где 1 обозначает орел, а 0 — решку.

```
first_coin_flips = pm.Bernoulli("first_flips", 0.5, size=N)
print first_coin_flips.value
```

[Output]:

```
[False False True True True False True False True True True True
False False False True True True True False True False True False
True True False False True True False True True True True False False
False False False True False True True False False False True True
True False False True False True True False False False False True
False True True False False False False True False True False False
True True False True True False True True False True False False
True True False True True False True True False True False True
True True True True]
```

Хотя *не все* студенты подбрасывают монету второй раз, смоделировать вероятную реализацию второго подбрасывания монеты все равно можно:

```
second_coin_flips = pm.Bernoulli("second_flips", 0.5, size=N)
```

С помощью этих переменных можно вернуть вероятную реализацию *наблюдаемой доли* ответов «Да». Для этого мы воспользуемся детерминистической переменной PyMC:

```
@pm.deterministic
def observed_proportion(t_a=true_answers,
                        fc=first_coin_flips,
                        sc=second_coin_flips):
    observed = fc*t_a + (1-fc)*sc
    return observed.sum() / float(N)
```

Строка `fc*t_a + (1-fc)*sc` содержит самую суть конф-алгоритма. Элементы в этом массиве равны 1 *тогда и только тогда, когда*: 1) в результате первого подбрасывания монеты выпал орел и студент жульничал на экзамене; 2) в результате первого подбрасывания монеты выпала решка, а второго — орел; в противном случае они равны 0. Наконец, в последней строке вычисляется сумма вектора и делится на `float(N)`, в результате чего возвращается процентное соотношение.

```
observed_proportion.value
```

[Output]:

```
0.26000000000000001
```

Далее нам нужен набор данных. После проведения опросов с подбрасываниями монет исследователи получили 35 ответов «Да». Относительно нашей ситуации выходит: если бы на самом деле жульничавших не было, можно было бы ожидать в среднем, что 1/4 всех ответов будут «Да» (равная 1/2 вероятность выпадения решки в первый раз и равная 1/2 вероятность выпадения орла во второй), так что

таких ответов при отсутствии жульничавших было бы около 25. С другой стороны, если бы *все студенты жульничали*, можно было бы ожидать, что около 3/4 всех ответов будут положительными.

Мы наблюдаем биномиальную случайную переменную с $N = 100$ и $p = \text{observed_proportion}$, a value = 35:

```
X = 35
observations = pm.Binomial("obs", N, observed_proportion, observed=True,
                           value=X)
```

Далее мы добавляем все интересующие нас переменные в контейнер Model и запускаем для нее наш алгоритм MCMC «черный ящик».

```
model = pm.Model([p, true_answers, first_coin_flips,
                  second_coin_flips, observed_proportion, observations])
```

```
# Будет объяснено в главе 3
mcmc = pm.MCMC(model)
mcmc.sample(40000, 15000)
```

[Output]:

```
[-----100%-----] 40000 of 40000 complete in 18.7 sec
```

```
figsize(12.5, 3)
p_trace = mcmc.trace("freq_cheating")[:]
plt.hist(p_trace, histtype="stepfilled", normed=True,
         alpha=0.85, bins=30, label="апостериорное распределение",
         color="#348ABD")
plt.vlines([.05, .35], [0, 0], [5, 5], alpha=0.3)
plt.xlim(0, 1)
plt.xlabel(u"Значение $p$")
plt.ylabel(u"Плотность")
plt.title(u"Апостериорное распределение параметра $p$")
plt.legend();
```

Даже с учетом рис. 2.9 мы все равно не вполне уверены, какова фактическая частота жульничавших студентов, хотя и сузили ее до возможного диапазона между 0,05 и 0,35 (отмечен на рисунке сплошными вертикальными линиями). Это довольно неплохой результат, ведь *априори* мы понятия не имели, сколько студентов могло жульничать (поэтому и использовали равномерное априорное распределение). С другой стороны, не так уж этот результат и хорош: фактическое значение находится где-то в пределах окна шириной 0,3. Добились ли мы хоть чего-то или все еще не уверены в фактической частоте?

Могу поспорить, что да, мы действительно получили какую-то новую информацию. В соответствии с нашим апостериорным распределением невозможно, чтобы все студенты сдали экзамен честно, то есть согласно апостериорному распределению вероятность $p = 0$ очень низка. Мы начали с равномерного априорного распределе-

ния и считали все значения p равновероятными, однако сами данные исключили возможность $p = 0$, так что мы уверены: мошенники на экзамене были.

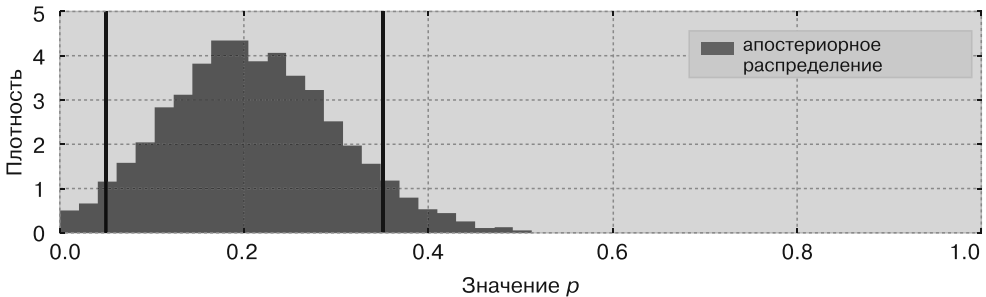


Рис. 2.9. Апостериорное распределение параметра p

Подобные алгоритмы подходят для сбора конфиденциальной информации и обеспечивают *разумную* степень уверенности в правдивости, хотя и «зашумленности» данных.

2.2.8. Альтернативная модель РуМС

При заданном значении p (которое мы, как всеведущие, знаем) можно найти вероятность получения от студента положительного ответа:

$$\begin{aligned} P(\text{«Да»}) &= P(\text{выпадение орла при первом подбрасывании монеты})P(\text{мошенник}) + \\ &+ P(\text{выпадение решки при первом подбрасывании})P(\text{выпадение орла} \\ &\text{при втором подбрасывании}) = \frac{1}{2}p + \frac{1}{2}\frac{1}{2} = \\ &= \frac{p}{2} + \frac{1}{4}. \end{aligned}$$

Следовательно, если нам известно p , то известна и вероятность получения от студента положительного ответа. Создадим в РуМС детерминистическую функцию для вычисления вероятности ответа «Да» по заданному значению p :

```
p = pm.Uniform("freq_cheating", 0, 1)
```

```
@pm.deterministic
def p_skewed(p=p):
    return 0.5*p + 0.25
```

Можно было написать `p_skewed = 0.5*p + 0.25`, вместо того чтобы создавать однострочную функцию, ведь элементарные операции сложения и скалярного умножения приводят к неявному созданию детерминистической переменной, но я хотел ради большей понятности написать подробный стереотипный код.