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 ($0 \le X \le N$). Чем больше p (в пределах от 0 до 1), тем больше событий, вероятно, произойдет. Математическое ожидание биномиального распределения равно N_p. 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],
            edgecolor=colors[i],
            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");
```

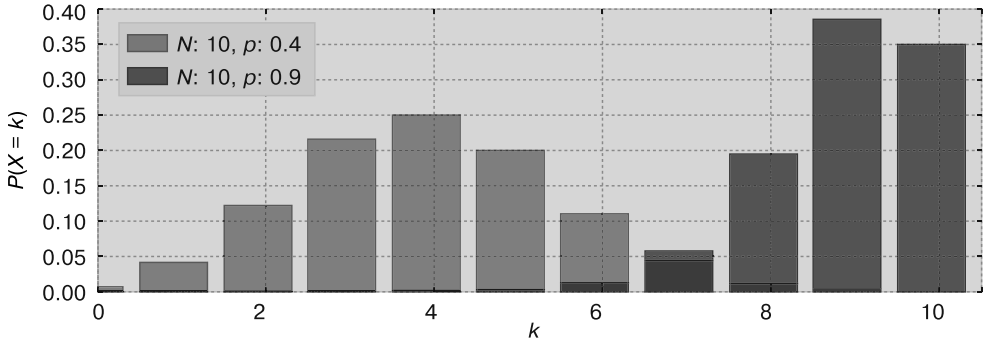

Рис. 2.8. Распределения вероятностей биномиальных случайных переменных

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... X_N с одинаковым параметром p, то $Z = X_1 + X_2 + ... + X_N \sim \text{Binomial}(N, p)$.

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 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  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 $p =$ `observed_proportion`, а `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 а-ля «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=u"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.

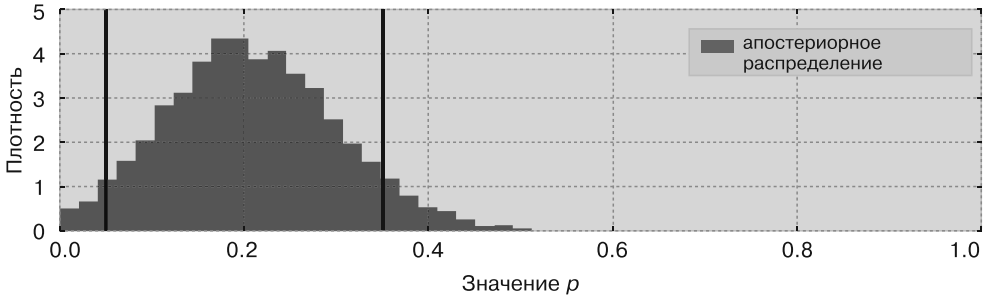

Рис. 2.9. Апостериорное распределение параметра p

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

2.2.8. Альтернативная модель PyMC

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

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

Следовательно, если нам известно p, то известна и вероятность получения от студента положительного ответа. Создадим в PyMC 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.