1 Вычисление параметров при заданных скорости кодирования, ошибке и мощности алфавита кодера

Программа запускается:

```
python3 task2.py json_files/single_coin.json 0.9 0.3 13 STAT
```

На вход подаётся:

- $R = 0.9$ – скорость кодирования;
- $\delta = 0.3$ – вероятность ошибки;
- $q = 13$ – мощность алфавита кодера;
- $STAT$ (NO_STAT) – флаг, что процесс – стационарный (не стационарный).

Если процесс – эргодический и не стационарный (рассматриваются только марковские процессы), то на вход подаётся также распределение финальных вероятностей.

1.1 Вычисление параметров ε и n

Пусть $X = \{x_1, \dots, x_n\}$. Тогда $Pr(|\frac{1}{n}\sum\limits_{i=1}^{n} I(x^{(i)}) - H(X)| \geq \varepsilon) \leq \frac{D}{n*\varepsilon^2}$

$D = D(I(X)) = \frac{1}{n}\sum\limits_{i=1}^{n} I(x^{(i)}) = \frac{1}{n}I(x) = MI^2(x) - H^2(x)$ – дисперсия количества информации вероятностного ансабля.

Задание параметров R, δ и q приводит к следующим ограничениям:

1. $\frac{D}{n\varepsilon^2} \leq \delta$
2. $|T_n| \leq 2^{n(H(X)+\varepsilon)} \leq 2^q$
3. $H(X) < \frac{log(|T_n|)}{n} \leq \frac{q}{n} \leq R$

Из этих ограничений вычисляем ограничения на ε: $\begin{cases} \varepsilon \geq \sqrt{\frac{D}{n\delta}} \\ \varepsilon \leq \frac{q}{n} - H(X) \end{cases}$

В таком случае положим $\varepsilon = \sqrt{\frac{D}{n\delta}}$. Тогда $\sqrt{\frac{D}{n\delta}} \leq \frac{q}{n} - H$.
И должны выполняться следующие условия на q:

$$\begin{cases} q \geq nH + \sqrt{\frac{nD}{\delta}} \\ q \leq nR \end{cases}$$

Получаем, что оптимальный код получается при:

$$\begin{cases} n \geq \frac{1}{(R-H)^2}\frac{D}{\delta} \\ q \leq nR \end{cases}$$

Программа работает по алгоритму 1

2 Эксперименты

2.1 Эксперименты со стационарными процессами

2.1.1 Пример дискретного стационарного источника без памяти "single_coin.json"

Программа запускается:

```
python3 task2.py json_files/single_coin.json 0.9 10 0.3 STAT out_files/code_for_single_
```

То есть:

- $R = 0.9$
- $q = 10$

Algorithm 1: Получение высоко вероятностного множества

```
Input   : R, δ, q, H
Output: HighProbSet
if R ≤ H then
    return Code does not exist: R < entropy
n_min ← ⌊ 1/(R−H)^2 · D/δ ⌋ + 1
if q < n_min H then
    return Code does not exist: q < n_min H
is_find_code ← False
i ← 0
while (not(is_find_code)) do
    n ← n_min + i
    if q ≤ nR then
        ε ← D/(nδ)
        words_number ← 2^n
        foreach j ∈ {0, ..., words_number − 1} do
            curr_mess ← convert_to_binary(j)
            mess_info ← CalculateAmountInfo(curr_mess)
            k ← 0
            if (mess_info ∈ (H − ε, H + ε)) then
                HighProbSet.append(mess_info)
                k ← k + 1
            j ← j + 1
        if (k ≤ 2^q) then
            is_find_code ← True
    i ← i + 1
return HighProbSet
```

- $\delta = 0.3$

Пример означает, что времени источник выбирает переключатель "switch_0"и реализует модель выбора монеты "монета_1 а потом выдаёт символ в соответствии с распределением для "монеты_1": Pr(монета_1 = 0) = 0.9 и Pr(монета_1 = 1) = 0.1.

Результат эксперимента:

- $n_{min} = n = 17$;
- $entropy = 0.4689955935892812$;
- $info\_disp = 0.9043582063292139$;
- $epsilon = 0.421099915098453$;
- Количество кодовых слов = 834;
- Вероятность выдать слово из высоко вероятностного множества = 0.9173593774439441;

2.1.2 Пример дискретного стационарного источника без памяти "p_single_coin.json"

Программа запускается:

```
python3 task2.py json_files/p_single_coin.json 0.9 13 0.3 STAT out_files/code_for_p_si
```

То есть:

- $R = 0.9$
- $q = 13$
- $\delta = 0.3$

Пример означает, что источник выбирает переключатель "switch_0"и реализует модель выбора монеты "монета_1"с вероятностью 0.9 и "монета_2"с вероятностью 0.1. А потом выдаёт символ в соответствии с распределением для "монеты_1":

- Pr(монета_1 = 0) = 0.9 и
- Pr(монета_1 = 1) = 0.1,

и для "монеты_2":

- Pr(монета_2 = 0) = 0.8 и
- Pr(монета_2 = 1) = 0.2.

Тогда

- $Pr(x = 0) = 0.9 * 0.9 + 0.1 * 0.8 = 0.89$ и
- $Pr(x = 1) = 0.9 * 0.1 + 0.1 * 0.2 = 0.11$.

Результат эксперимента:

- $n_{min} = n = 19$;
- $entropy = 0.49991595816452783$;
- $info\_disp = 0.890701701397556$;
- $epsilon = 0.39530172828554155$;
- Количество кодовых слов = 5036;
- Вероятность выдать слово из высоко вероятностного множества = 0.949844223765575;

2.2 Эксперименты с эргодическими не стационарными процессами

2.2.1 Пример дискретного стационарного источника без памяти "simple_markov.json"

Программа запускается:

```
python3 task2.py json_files/simple_markov.json 0.9 12 0.3 NO_STAT out_files/code_for_si
```

То есть:

- $R = 0.9$
- $q = 12$
- $\delta = 0.3$
- $Pr(x = 0) = 0.8952380952380953$
- $Pr(x = 1) = 0.10476190476190472$

Марковская цепь, моделирующая источник, изображена на Рис. 1.

- В состоянии s_0:
  - $Pr(x = 0) = 0.85$
  - $Pr(x = 1) = 0.15$
- В состоянии s_1:
  - $Pr(x = 0) = 0.9$
  - $Pr(x = 1) = 0.1$

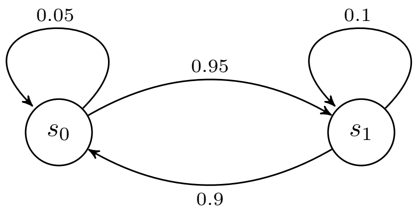

Рис. 1: Цепь Маркова

Тогда, чтобы посчитать финальные вероятности Π_0, Π_1, нужно решить систему уравнений:

$$\begin{cases} \Pi_0 = \frac{10}{100}\Pi_0 + \frac{5}{100}\Pi_1 \\ \Pi_0 = \frac{95}{100}\Pi_0 + \frac{90}{100}\Pi_1 \\ \Pi_0 + \Pi_1 = 1 \end{cases}$$

Решая систему, получаем, что:

$$\begin{cases} \Pi_0 = \frac{2}{21} \\ \Pi_1 = \frac{19}{21} \end{cases}$$

Таким образом, у цепи Маркова существуют финальные вероятности, и, следовательно, источник – эргодический.

Тогда

- $Pr(x=0) = \frac{2}{21} * \frac{85}{100} + \frac{19}{21} * \frac{90}{100} = 0.8952380952380953$ и
- $Pr(x=1) = \frac{2}{21} * \frac{15}{100} + \frac{19}{21} * \frac{10}{100} = 0.10476190476190472$.

Результат эксперимента:

- $n_{min} = n = 18$;
- $entropy = 0.4839112332593779$;
- $info\_disp = 0.8984778448388356$;
- $epsilon = 0.4079029125677652$;
- Количество кодовых слов = 4048;
- Вероятность выдать слово из высоко вероятностного множества = 0.9663007491370017;