Task 1. Estimating source entropy

© Daria Yakovleva

2016/10/11

Variant 25
Source file: xargs.1
File size: 4227 bytes = 33816 bits
Count of symbols = 4227
Compressed file size (using RAR): 1 823 bytes = 14584 bits
Weight of one symbol = 3,45 bits
Task:
Input data is string s. Length of s is $len(s)$
Length of block is n
$X = \{s[i..i+n], i \in 1..len(s) - n\}$
$p(x) = \frac{|\{y \in X | x = y\}|}{|X|}$

Entropy of block
$H_n(X) = -\sum_{x \in X} p(x) \cdot \log_2(p(x))$
Entropy per letter
$H(x) = \frac{H_n(X)}{n}$

- **Results**

block length = n	entropy per letter	expected size, bits
1	4.89835	20705.3
2	4.04639	17104.1
3	3.21412	13586.1
4	2.59235	10957.8

- **Conclusion**

  Entropy limits the maximum possible lossless compression.

  Consequently we found out that archive program (RAR) uses interesting coding techniques and minimum length of block is 3.

  But we can say that RAR uses not global optimal data compression algorithms.

- **Source code**

```
string inputData = "";
map<string, int> counts;
vector<pair<string, double>> probability;
int main() {
freopen("xargs.1", "r", stdin);
string cur = "";
while (getline(cin, cur)) {
inputData += cur;
}
for (int n = 1; n <= 4; n++) {
int len = inputData.size() - n;
counts.clear();
probability.clear();
for (int i = 0; i < len; i++) {
cur = "";
for (int j = 0; j < n; j++) {
cur += inputData[i + j];
}
counts[cur]++;
}
for (pair<string, int> elem: counts) {
probability.push_back(make_pair(elem.first, (double)elem.second / len));
}
double entropy = 0.0;
int uniqueN = probability.size();
for (int i = 0; i < uniqueN; i++) {
double curProb = probability[i].second;
entropy -= curProb * log2(curProb);
}
entropy /= n;
cout << "n = " << n << ", entropy = " << entropy << " expected size = " << entropy * len
```